



Martino Facchin



Alessandro Ranellucci

Dazu gehören natürlich WLAN, BLE, Dual-Core-Processing und all die anderen Funktionen, die der ESP32 mitbringt. Wir haben uns entschieden, den Arduino Nano ESP32 nicht mit zusätzlichen Funktionen auszustatten, wie wir es bei anderen Boards getan haben. Dieses Board ist als Baustein konzipiert. Im Gegensatz zum Arduino BLE ist es kein Gerät, das man sofort für umfangreiche Anwendungen einsetzen kann. Sie müssen zusätzliche Komponenten auf dem Board anbringen, zum Beispiel Sensoren oder Module. Der Nano ESP32 ist mit einer RGB-LED ausgestattet, aber darüber hinaus hat der Maker die Freiheit, ihn mit anderen Komponenten zu ergänzen und als gut kalibrierte Basis für seine Projekte zu verwenden.

Elektor: Hatten Sie Schwierigkeiten, den ESP32-Chip dem Arduino-Nano-Formfaktor einzupassen, und wenn ja, wie haben Sie diese Probleme gelöst?

Alessandro Ranellucci: In der Tat bestand die erste Herausforderung darin, eine Partnerschaft mit Espressif einzugehen. Sie entwickeln seit vielen Jahren einen hervorragenden Arduino-Kern für die ESP32-Boards, der tief in das Arduino-Framework eingebettet ist, aber auch ihre eigenen technischen Vorlieben berücksichtigt. Die Herausforderung bestand darin, herauszufinden, wie wir unsere Bemühungen kombinieren können, um ein kohärenteres und verbessertes Nutzungserlebnis zu schaffen. So kam es zu dieser Zusammenarbeit. Ich denke, Martino kann auch andere Herausforderungen nennen, zum Beispiel die Pin-Nummerierung.

Martino Facchin: Aus Sicht der Hardware gab es definitiv keine Probleme. Natürlich ist das einfach im Vergleich zu anderen Boards, die wir in den letzten Jahren hergestellt haben. Vom Software-Standpunkt aus gesehen ist das eine ganz andere Sache, weil die Pin-Nummerierung sehr an die ESP-Welt gebunden war. Sie haben die Pins genau so nummeriert, wie sie auf dem CPU-Gehäuse sind. Auf der Unterseite der Platine sehen Sie vielleicht Aufkleber von anderen Herstellern mit Nummern wie 31, dann daneben eine 4, dann eine 55 und so weiter. Das funktioniert nicht für den Nano - nicht die Lösung, die wir gesucht haben. Also haben wir einen Weg entwickelt, um logische Pin-Nummern in interne Pin-Nummern

zu übersetzen. Wir haben dies in den ESP32-Core der Community aufgenommen. Jetzt kann jedes Board mit einem ESP32, egal von welchem Hersteller, die gleiche Logik verwenden, wenn sie unsere Philosophie übernehmen wollen, und das ist ein direkter Beitrag zu einem Kern, den wir nicht pflegen. Es war schwierig, weil die Veröffentlichungszeit dieser Funktion perfekt mit der Veröffentlichungszeit des Boards übereinstimmen musste. Letztendlich ist es aber gelungen.

Es war eine Herausforderung, denn es war das erste Mal, dass wir etwas veröffentlicht haben, das nicht vollständig von uns kontrolliert wurde, und das war schwierig, aber wir haben es geschafft.

Alessandro Ranellucci: Wir haben aus dem gesamten Iterationsprozess wichtige Erkenntnisse gewonnen, denn, wie Martino bereits erwähnte, kann sich nun jeder Hersteller für die Verwendung logischer PINs entscheiden. Es ist benutzerfreundlicher, die Pins fortlaufend zu nummerieren, als die Pin-Nummerierung des Controllers zu verwenden, zum Beispiel PA1 und PD1.

Elektor: Haben die Entwicklung der ESP32-Software, das Ökosystem und die Unterstützung durch die Community Ihre Entscheidung für den Einsatz des ESP32 im neuen Arduino Nano beeinflusst?

Martino Facchin: Nein. Wir hätten uns auch ohne die Unterstützung der Community dafür entschieden, sie zu nutzen. Da die Community umfangreiche Arbeit leistet und unsere Projekte florieren, profitieren wir natürlich sehr von ihren Beiträgen, insbesondere beim Bibliotheksverwalter. Einige Bibliotheken waren bereits mit unserem Board kompatibel, was von Vorteil war. Andere waren exklusiv für den ESP32, was ein Nachteil war, aber jetzt können wir auch diese nutzen. Diese Kompatibilität war ein Faktor bei unserer Entscheidung, aber wir hätten uns so oder so für den ESP32 entschieden.

Alessandro Ranellucci: Bei der Software hatten wir die Möglichkeit, einen neuen Kern zu entwickeln, so wie wir es bei anderen Produkten wie dem RP2040 getan haben, wo wir unseren eigenen Kern entwickelt haben, um die volle Kontrolle über die Software zu haben. Aber Espressif hat im Laufe der Jahre hervorragende Arbeit

“

Die Pin-Nummerierung war eine Herausforderung, weil die Nummern sehr an die ESP32-Welt gebunden waren. Aber wir haben eine Lösung gefunden - logische Pin-Nummern.

Martino Facchin

geleistet und verfügt über eine solide Community. Deshalb haben wir mit ihnen zusammengearbeitet - eine Entscheidung, die durch die Stärke des Ökosystems beeinflusst wurde.

Wir sind bestrebt, technologisch neutral zu bleiben und uns nicht ausschließlich auf die Mikroprozessoren eines bestimmten Herstellers zu beschränken. Unser Ziel ist es, eine vielseitige und interoperable Arduino-Plattform anzubieten, denn das ist es, was die Benutzer von Arduino erwarten und wahrnehmen. Aus diesem Grund experimentieren und forschen wir ständig für die Entwicklung von neuen Produkten.

Martino Facchin: Personen, die von anderen Feldern zu Arduino wechseln, wie BSP oder integrierten Umgebungen, die zeitaufwändige Einrichtungs- und Konfigurationsprozesse enthalten, sagen oft, dass Arduino einfach funktioniert. Das ist, glaube ich, unser größter Mehrwert - dass User auf dem einfachsten und schnellsten Weg mit der Benutzung von Arduino anfangen können. Und wie Alessandro erwähnte, sind wir unabhängig. Vor sechs Jahren konnte ich diese Behauptung nicht aufstellen, weil Atmel im Wesentlichen unser Hauptsponsor war, aber jetzt sind wir völlig unparteiisch.

Elektor: Können Sie uns etwas über die Verbesserungen oder Änderungen sagen, die Sie an der ESP32-Plattform vorgenommen haben, um sie auf die Ziele des Arduino Nano ESP32 zuzuschneiden?

Martino Facchin: Wir haben den Upload-Prozess, der normalerweise für ESP32 verwendet wird, modifiziert. Bisher mussten die Benutzer auf das native USB-Modul wechseln, um das ESP-Tool zu bedienen - ein Prozess, der nicht nahtlos in unsere IDE integriert ist. Jetzt wird beim Hochladen eines Sketches eine Standardmethode verwendet. Der Sketch wird über ein Update der Gerätefirmware (Device Firmware Update, DFU) hochgeladen und durch ein Update über kabellose Verbindung (Over-the-Air, OTA) auf die zweite Partition übertragen. Der Bootloader wird dann angewiesen, einen Neustart von dieser zweiten Partition aus zu versuchen. Ist dies erfolgreich, wird der neue Sketch geladen, andernfalls bleibt der ursprüngliche Sketch erhalten. Diese Implementierung stellt eine erhebliche Verbesserung dar, da Espressif bereits verschiedene Anwendungsfälle berücksichtigt hatte. Wir haben ihren Ansatz angepasst, um ein schnelleres, zuverlässigeres System mit einem Wiederherstellungsmodus zu entwickeln.

Wir haben Double-Tap-Unterstützung (man kann den Wiederherstellungsmodus durch doppeltes Drücken der Reset-Taste aufrufen). Das war auf dem ESP-Board nicht verfügbar. Dies sind einige Änderungen, die von der Community wegen des Aufwands gut angenommen wurden. Auch wenn der Community-Kern von Espressif unterstützt wird, handelt es sich doch um eine Gemeinschaftsarbeit, also waren alle sehr glücklich darüber.

Alessandro Ranellucci: Es gibt noch zwei weitere Beiträge, die wir für das Ökosystem im Allgemeinen geleistet haben, also nicht

speziell für den Kern. Ein Teil der Arbeit, die wir geleistet haben, war das Debugging, und wir haben auch MicroPython-Unterstützung für ESP32 hinzugefügt.

Elektor: Was das Debugging betrifft, gibt es auf dieser Platine einen Zugang zum ESP über Lötpunkte oder ist es eine andere Platine?

Martino Facchin: Nein, Sie können dieses Board direkt über USB debuggen - Espressif war so freundlich, dies zu ermöglichen. Über USB gibt es eine serielle Schnittstelle, CDC, das übliche Zeugs, und auch eine JTAG-Schnittstelle. Sie können mit der JTAG-Schnittstelle interagieren, indem Sie einfach den normalen USB neben der normalen Kommunikation verwenden. Es kann nicht schaden, einen externen Debugger anzuschließen, aber man muss es nicht. Er ist sogar schon in die IDE integriert. Man muss nur das Board anschließen, die Option *Debug Mode (Hardware CDC)* aus dem Menü auswählen und den großen *Debug*-Button drücken. Unter der Haube laufen einige Dinge, die mit OpenOCD und GDB zu tun haben, aber das ist auch transparent für den Nutzer.

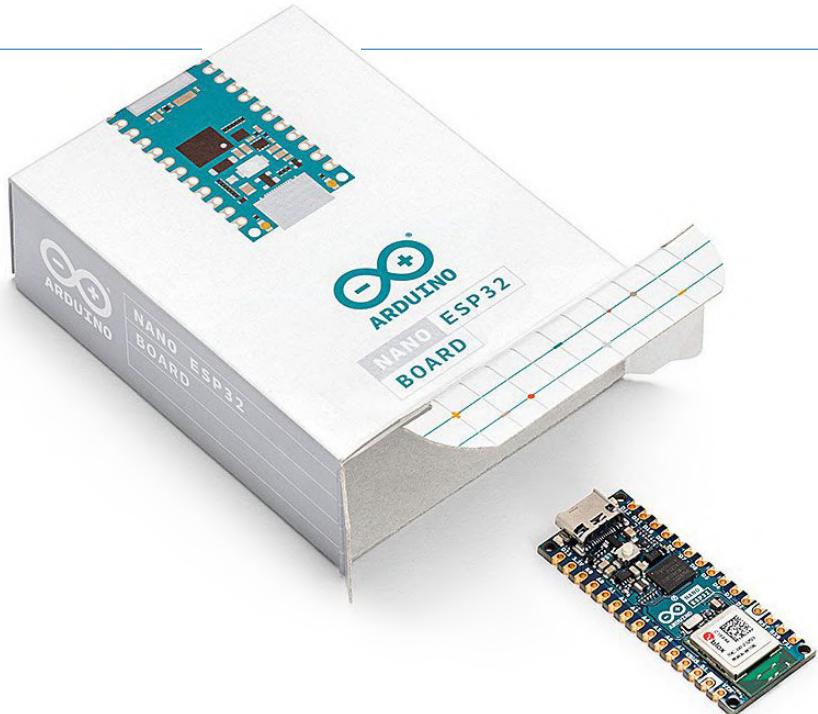
Alessandro Ranellucci: Unser Beitrag bestand darin, hierfür ein Benutzererlebnis zu entwickeln. Mit Arduino ist es viel einfacher, als professionelle, vollwertige Debugging-Tools zu verwenden, zu testen, zu dokumentieren, das Bewusstsein dafür zu schärfen, es an die Nutzer weiterzugeben und dann auch mit Espressif zu interagieren.

Elektor: Die Feinheiten der Umsetzung bringen mich auf einige Fragen aus dem Internet. Es gab Verwirrung über die Anschlüsse, wenn man das Board einsteckte - es gab zwei Anschlüsse und die Leute wussten nicht, welchen sie benutzen sollten.

Martino Facchin: Ja, es ist nicht einfach zu erklären, weil unsere grundsätzliche Vorgehensweise immer war, den Leuten zu sagen, dass sie mit den Grundlagen beginnen sollen. Zum Beispiel machen Sie das so und so, dann gibt es einen seriellen Port und so weiter, und dann gehen Sie weiter zu einem fortgeschrittenen Thema. Fortgeschrittene Themen müssen sehr gut erklärt werden. Aus diesem Grund haben wir eine Dokumentation darüber, wie man Debugging richtig durchführt. Wir lassen die Funktion nicht einfach stehen und überlassen es den Leuten, zu experimentieren. Außerdem lesen die Leute normalerweise keine Dokumentation, also tun wir unser Bestes, um diesen Konflikt zu vermeiden, und stellen alles zur Verfügung. Die Dokumentation sollte trotzdem gelesen werden, und alles wird darin sehr gut erklärt.

Alessandro Ranellucci: Und die neueste Version der Arduino-IDE, die vor einem Monat veröffentlicht wurde, hat viele Verbesserungen bei der Auswahl von Ports und Boards; die Verwirrung über mehrere Ports oder Ports, die sich ändern, nachdem man Operationen durchführt, und so weiter wird nun freundlicher behandelt.

Martino Facchin: Außerdem haben wir dieses DFU-Menü hinzugefügt, so dass wir jetzt eine steckbare Erkennung haben. Die steckbare Erkennung ist ein sehr interessantes Konzept, weil man Dinge



über die serielle Schnittstelle oder über WLAN via MDNS erkennen kann. Paul Stoffregen fügte der Version 1 der Arduino-IDE die Erkennung über HID hinzu, weil sein Bootloader HID-basiert war, aber dies war immer ein externer Patch für den Arduino-Installer. Wir haben beschlossen, das allgemein verfügbar zu machen. Also portierte er sein Erkennungsprogramm auf die Arduino-IDE Version 2. Dann hatten wir ein weiteres Problem mit dem Minima – wenn man in den Bootloader-Modus ging, wurde keine serielle Schnittstelle angezeigt. Es war verwirrend, ihn nicht im Menü zu sehen. Daher beschlossen wir, unserem Erkenner zum DFU-Port hinzuzufügen. ESP32 hat ein Firmwareupdate während der Laufzeit, aufgrund der Art und Weise, wie Code hochgeladen wird. Gelegentlich kann es auf Ihrem Computer erscheinen, aber, wie ich bereits in der Dokumentation erwähnt habe, wird alles erklärt. Dieser Port ist für das Hochladen gedacht, und Sie können eine der beiden Varianten wählen. Es ändert nichts an der Tatsache, dass er Uploads akzeptiert.

Elektor: Gab es bei der Umstellung auf den ESP32-Chip für das neue Nano-Modell irgendwelche Kompatibilitätsänderungen in Bezug auf die Bibliotheken des bestehenden Codes?

Alessandro Ranellucci: In der Tat - jedes Mal, wenn wir eine neue Architektur in die Familie aufnehmen. Es gibt die grundlegenden offiziellen Bibliotheken, die portiert werden müssen, vor allem wenn sie stark optimiert sind, so dass sie mit Low-Level-Code laufen. In diesem Fall mussten wir einige grundlegende Bibliotheken erweitern. Aber das Ökosystem der Bibliotheken für den ESP32 war viel ausgereifter.

Elektor: Können Sie Sicherheitsmerkmale oder Überlegungen nennen, die Sie dazu veranlasst haben, den ESP32 für den Arduino Nano ESP32 zu wählen, insbesondere für IoT-Anwendungen?

Alessandro Ranellucci: Ich würde nicht sagen, dass wir den ESP32 in erster Linie aus Sicherheitsgründen gewählt haben. Natürlich hat der ESP32 dort einige Möglichkeiten, die wir im Moment zum Teil auch nutzen.

Martino Facchin: Wirklich nur zum Teil – wir verwenden keine Verschlüsselung oder das, was man „Secure Boot“ nennt, weil es dem Benutzer das Leben extrem schwer macht. Man muss jede Binärdatei, die man erzeugt, signieren, und dann muss man sie für jedes Board ändern, sonst hat es keinen Zweck. Wir implementieren es also nicht, aber wir lassen die Möglichkeit natürlich offen. Vom Standpunkt eines Integrators aus betrachtet, hat dieses Produkt alle Möglichkeiten, um es wirklich sicher zu machen. Aber das ist nicht der Grund, warum wir es ausgewählt haben.

Alessandro Ranellucci: Normalerweise hatten wir bei all unseren Boards ein separates Hardware-Sicherheitselement. Das ist also die Methode, die wir für alle unsere Produkte gewählt haben. In diesem Fall haben wir es nicht implementiert, weil der Haupt-Mikrocontroller theoretisch über diese Fähigkeiten verfügt. Für den Moment haben wir uns jedoch entschieden, den Chip so zu verwenden, wie er ist, um das Ganze einfach zu halten. Es gibt natürlich keinen Grund, warum es hier keine weiteren Entwicklungen geben könnte.

Martino Facchin: Das Gleiche geschah zum Beispiel mit dem Portenta H7, als wir den sicheren Bootloader und die MCO-Boot-Infrastruktur freigaben, um sichere OTA und sichere Updates zu ermöglichen; dies war eine Option und nicht etwas, das wir direkt ab Werk anbieten. Ich bin mir ziemlich sicher, dass wir irgendwann auch eine Art von Dokumentation über ein Menü anbieten werden.

Elektor: Gibt es Pläne, den Espressif-Chip in weiteren Arduino-Boards zu verwenden, zum Beispiel in der PRO-Serie?

Alessandro Ranellucci: Für die PRO-Linie kann ich diese Frage nicht beantworten, aber im Allgemeinen: ja.

Martino Facchin: Wir lieben das Format von u-blox, weil es wirklich gut zu den Nanos passt. Es ist sehr klein. Wir haben die Module von u-blox in fast allen Nano-Boards verwendet, entweder als Zusatzchip für WLAN oder, beim Nano BLE, als Haupt-Mikrocontroller. Wir sind also sehr zufrieden mit ihnen als Hersteller. Gleichzeitig hatten wir auf dem UNO R4 natürlich auch das normale Espressif-Modul. Und ja, haben noch andere Dinge in der Planung.

Elektor: Können Sie näher darauf eingehen, wie Sie die Zusammenarbeit und den Kontakt mit dem ESP32-Entwicklungsteam gestaltet haben, um eine nahtlose Integration in das Arduino-Ökosystem zu gewährleisten?

Alessandro Ranellucci: Also, es gibt ein gutes Entwicklungsteam bei Espressif. Sie leisten großartige Arbeit, indem sie die Community in den Entwicklungsprozess einbeziehen. Wir haben uns also persönlich mit ihnen getroffen und auch immer wieder telefoniert. Wir hatten auch einen gemeinsamen Kommunikationskanal auf Slack oder anderen Kommunikationsplattformen, so dass wir einen direkten Weg hatten, uns täglich gegenseitig auf den neuesten Stand zu bringen, uns gegenseitig über Probleme zu informieren und aufzuzeigen, bevor wir unsere Arbeit auf GitHub veröffentlichten. Es war eine sehr enge Zusammenarbeit. Wir können also viele Leute von Espressif namentlich erwähnen, die uns bei dieser Zusammenarbeit geholfen haben.

Alessandro Ranellucci: Auf höherer Ebene hat uns Ivan Grokhov (VP of Software Platforms, Espressif) geholfen, die Kommunikation zu ermöglichen, und Pedro Minatel (Developer Advocate) hat uns sehr geholfen, wenn es darum ging, mit genau der richtigen Person in Kontakt zu treten.

Elektor: Können Sie sich vorstellen, dass sich die Partnerschaft zwischen Arduino und Espressif weiterentwickelt, und welche Möglichkeiten sehen Sie für weitere Innovationen und Zusammenarbeit?

Alessandro Ranellucci: Ich würde sagen, dass wir uns über die Hardware-Produkte hinaus in beiden Teams sehr einig sind, dass wir bei der API-Standardisierung eng zusammenarbeiten müssen. Die API bietet jetzt mehr Interoperabilität zwischen der Arduino- und der ESP32-Welt. In diesem Bereich würden wir die Zusammenarbeit gerne fortsetzen.

Martino Facchin: Außerdem hat sich Espressif in viele verschiedene Bereiche verzweigt, wie zum Beispiel Rust. Sie haben viele Entwickler, die an Rust arbeiten, was für uns keine Priorität ist, aber die Arbeit, die sie dort machen, könnte uns auf einige sehr interessante Ideen bringen. Dann gibt es noch das Zephyr-Betriebssystem, bei dem wir beide Partner sind. Im technischen Lenkungsausschuss des Boards können wir Dinge entscheiden, und Espressif kann das auch. Die gesamte Entwicklung, die wir beide betreiben, zielt darauf ab, künftige Werkzeuge für künftige Generationen besser zu machen. Letztendlich arbeiten wir auf dasselbe Ziel hin.

Elektor: Wenn Sie den ESP32 mit anderen Chips vergleichen, könnten Sie die spezifischen Benchmarks oder Leistungskennzahlen erläutern, die Sie zu dem Schluss gebracht haben, dass er die ideale Wahl für den Arduino Nano ESP32 ist?

Martino Facchin: Der ESP32-S3 ist ein guter Chip. Sein Strombedarf ist nicht unglaublich niedrig, aber er ist auch nicht schlecht. Ultra-Low-Power-Fähigkeiten sind vorhanden, wenn man es wirklich weit treiben will, aber wir werden unseren Nutzern nicht sagen, dass sie alles in den Tiefschlaf versetzen und den Ultra-Low-Power nutzen sollen, auch wenn es schön ist, dass es so etwas gibt. Ich würde sagen, dass die Leistung nicht der Hauptgrund

für die Existenz der Nano-Familie ist. Vielmehr ist sie einfach zu benutzen. Wir hatten die Gelegenheit, einen Vergleich mit dem Portenta H7 durchzuführen, zum Beispiel in Bezug auf die Fähigkeiten im Bereich Machine Learning. Der Portenta H7 ist etwa siebenmal schneller als der ESP32, selbst bei vergleichbaren Taktraten. Der ESP32 hat die Hälfte der Taktrate des Portenta H7. Der Nano existiert aus Gründen der Benutzerfreundlichkeit, der Umgebung, der Verfügbarkeit von Bibliotheken, der Bauform, und so weiter.

Elektor: Welche Rolle spielt das Echtzeit-Betriebssystem des ESP32 bei der Entwicklung des ESP32 und seiner Multitasking-Fähigkeit?

Alessandro Ranellucci: Ich würde sagen, dass dies ein wachsender Bedarf ist - wie man mehrere Kerne nutzt, Multitasking und so weiter. Dieser Bedarf wird derzeit von den Herstellern nicht so stark nachgefragt, aber es wird notwendig sein, ein Ökosystem verfügbar zu machen.

Martino Facchin: Vor etwas mehr als einem Jahr haben wir versucht, eine Standardisierung zu erreichen, mit etwas, das Arduino-Threads genannt wird. Wenn Sie sich GitHub ansehen, haben wir versucht, die Idee voranzutreiben, Threads hinter verschiedenen Registerkarten in Ihrer IDE zu verstecken. Man hat also eine Registerkarte mit setup() und loop(), wie üblich. Dann gibt es eine weitere Registerkarte mit einem weiteren setup() und loop(), und das repräsentiert den zweiten Thread. Dann gibt es noch eine dritte, eine vierte und so weiter, wie viel auch immer erforderlich ist. Es gibt immer noch das Problem der Variablenynchronisation, bei dem Sie die übliche RTOS-Beschränkung haben, sodass Sie unterschiedliche Variablennamen zwischen zwei Threads verwenden müssen. Wir haben es auf einer höheren Ebene gelöst. Wir haben diese Komplexität also mit Mbed versteckt, aber es war nicht wirklich eine reine Mbed-Sache. Wir wollten dies auf mehrere Betriebssysteme portieren.

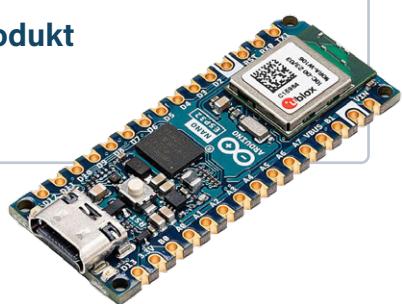
Auch wenn es gelungen ist, hat die Maker-Community überhaupt nicht auf diese Sache reagiert. Es ist also wahrscheinlich nichts, was von den Makern wirklich benötigt wird. Threading ist nett, aber verursacht auch Schwierigkeiten. Und gleichzeitig kann man natürlich FreeRTOS benutzen; man kann tun, was immer man will, wenn man geschickt genug ist, aber wir drängen nicht darauf - Benutzerfreundlichkeit ist das, worum es uns immer gegangen ist. ▶

Übersetzung von Matze Schrumpf -- 230524-02



Passendes Produkt

› **Arduino Nano ESP32**
www.elektor.de/20562



What Arduino Cloud is

Develop from anywhere

- + NO CODE**
With ready-to-use templates
- + LOW-CODE**
Automatically generated sketches
- + FULL ARDUINO EXPERIENCE**
Either offline with the UDE2 or online with the Cloud Editor
- + STORE YOUR SKETCHES ONLINE**
Use your code in your favourite Arduino development environment

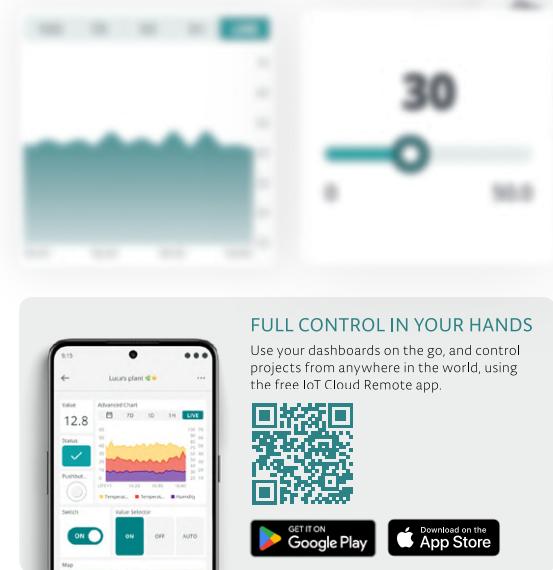
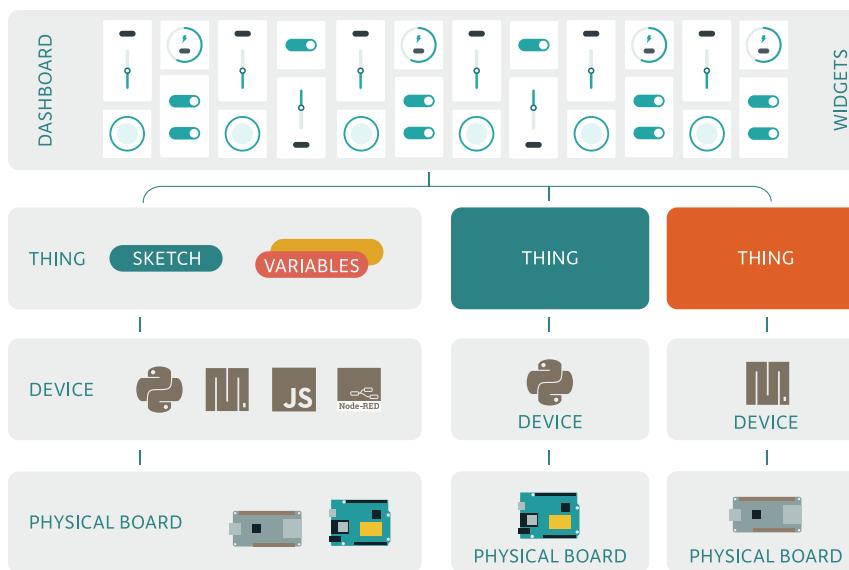
Program/Deploy

- + CABLE**
Traditional USB programming
- + OVER-THE-AIR (OTA) UPDATES**
Deploy your firmware wirelessly to your devices
- + MASS SCALE & AUTOMATION**
With the Arduino Cloud CLI

Monitor & Control

- + CUSTOM DASHBOARDS**
Using drag and drop widget
- + INSIGHTFUL WIDGETS**
Interact with the devices and get real-time and historical data with dozens of widgets
- + MOBILE APP**
Visualise your data in real-time from your phone with the IoT remote app

How does it work?



Compatible hardware

WITHIN ARDUINO DEVELOPMENT ENVIRONMENTS



ARDUINO



ESP32/ESP8266
+70% Of Arduino Cloud active users use ESP-based boards.

OUTSIDE ARDUINO DEVELOPMENT ENVIRONMENTS



Use your favourite programming environment and language to connect your devices to the Cloud.

TRIGGER ACTIONS ON THIRD PARTY PLATFORMS

Connect your Arduino Cloud devices to external platforms such as IFTTT, Zapier and Google Services using webhooks and unlock endless possibilities.

Seamlessly integrate your IoT devices with over 2 000 apps, enabling tasks like receiving phone notifications, automating social media updates, streamlining data logging to external files, creating calendar events, or sending e-mail alerts.

Third party platform integration

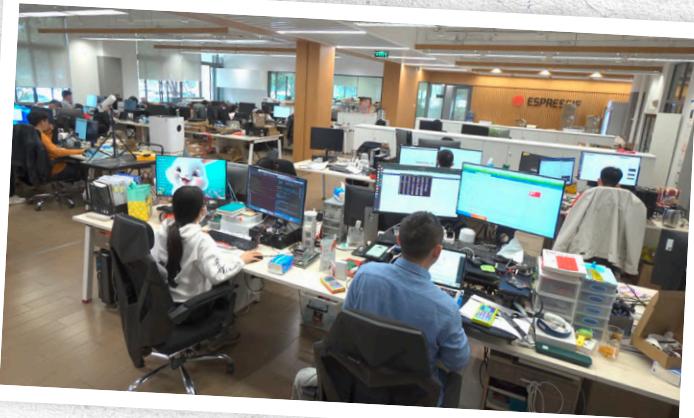


IoT-Chip-Innovationen

Einblicke von und in



ESPRESSIF



Die globale Elektor-Community besteht aus Elektroingenieuren, Makern und Studenten, die sich für ein breites Spektrum von Themen rund um die Elektronik interessieren. Einige dieser Community-Mitglieder haben kürzlich einige anregende Fragen für die Ingenieure und Führungskräfte von Espressif formuliert.

Teo Swee Ann gründete Espressif im Jahr 2008. Was hat ihn dazu bewogen? Was war im Jahr 2008 seine Vorstellung für Espressif, und wie hat sich sein Zukunftsbild für das Unternehmen seither verändert?
— Margriet Debeij (Niederlande)



Bei der Gründung von Espressif war es Teos ursprüngliches Ziel, ein Produkt zur Automatisierung von Analogtechnik zu entwickeln. Das Unternehmen entdeckte jedoch seine wahre Berufung in der Entwicklung proprietärer Internet-of-Things-Chips (IoT). Teo entwickelte ein klares Bild und eine Strategie, bei der die Entwicklung innovativer, erschwinglicher und energieeffizienter IoT-Lösungen im Vordergrund stand. Teos unerschütterlicher Fokus auf Innovation hat maßgeblich zum Erfolg von Espressif Systems beigetragen. Er verstand von Anfang an den Wert der Zusammenarbeit mit der Entwicklergemeinschaft und unterstützte aktiv deren Wachstum. Durch die Förderung eines Ökosystems der Zusammenarbeit und des Wissensaustauschs sorgte er dafür, dass Espressif an der Spitze des technologischen Fortschritts blieb. Mit der Weiterentwicklung der Technologie führte Teo Espressif Systems zu neuen Technologien wie künstlicher Intelligenz, maschinellem Lernen und Edge Computing und sorgte dafür, dass das Unternehmen an der Spitze der Innovation blieb.

Wie lässt sich ESP-NOW mit ZigBee oder mit Matter vergleichen?

— Clemens Valens (Frankreich)

Sowohl ESP-NOW als auch ZigBee können zum Aufbau von IoT-Geräten mit geringem Strombedarf verwendet werden. Die Kopplung von ESP-NOW mit einem WLAN eröffnet jedoch eine breite Palette von Anwendungsmöglichkeiten, die für einzigartige Lösungen genutzt werden können. ESP-NOW kann überdies eine größere Reichweite und eine höhere Bandbreite bieten, vor allem beim Einsatz im Freien. ESP-NOW funktioniert auch ohne eine WLAN-Infrastruktur und ist ein proprietäres Protokoll von Espressif, während Matter und ZigBee von der *Connectivity Standards Alliance* definierte Standards sind. Matter ist ein Protokoll in der Anwendungsschicht, das Thread und Wi-Fi als Protokolle für die drahtlose Kommunikation unterstützt. Sowohl ZigBee- als auch ESP-NOW-Geräte können mit Hilfe einer Matter-Bridge Teil des Matter-Netzwerks werden. Wir haben Lösungen für die ESP-NOW-Matter-Bridge und die Zigbee-Matter-Bridge.

Espressif arbeitet mit Arduino zusammen, woraus das Arduino Nano ESP32 Board entstanden ist. Ist eine Zusammenarbeit mit dem Raspberry-Pi-System und die Entwicklung eines gemeinsamen Produkts denkbar? — Ferdinand te Walvaart (Niederlande)

Wir schätzen unsere Zusammenarbeit mit Arduino. Sie umfasst die offizielle Unterstützung für die Arduino-IDE (Integrated Development Environment) für den ESP-IDF über verschiedene Chipsätze von Espressif. Darüber hinaus haben wir den Arduino UNO R4 Wi-Fi und den Arduino Nano ESP32 angekündigt, die beide das ESP32-S3-Modul enthalten. Wir glauben, dass wir alle, Arduino, Raspberry Pi und Espressif, eine gemeinsame Community und eine Open-Source-orientierte Denkweise teilen, was die Zusammenarbeit erleichtert. Wir sind der Meinung, dass sich Raspberry Pi derzeit hauptsächlich auf das MPU-Segment konzentriert und wir in diesem Segment kein Produkt anbieten können. Mit der Einführung der Pico-Serie besteht jedoch die Möglichkeit, in Zukunft zusammenzuarbeiten.

Für die Entwicklung und Verwaltung von KI/IoT-Technologien sind besondere Fähigkeiten erforderlich. Begrenzt der Mangel an qualifizierten Fachkräften in allen technischen Märkten das Wachstum der Produkte und des Umsatzes von Espressif? — Ruud Schaay (Niederlande)

Der Mangel an qualifizierten Fachkräften im Bereich KI/IoT stellt in der Tat eine Herausforderung dar. Unsere Produkte richten sich an Entwickler und bieten benutzerfreundliche Plattformen, die durch umfangreiche Ressourcen unterstützt werden. Dies ermöglicht es Enthusiasten mit unterschiedlichsten Hintergründen, effektiv mit KI/IoT-Technologien zu arbeiten. Über unsere globalen Teams und Online-Communities arbeiten wir auch aktiv mit Bildungseinrichtungen auf der ganzen Welt zusammen und bieten Workshops und Ressourcen an, um die Qualifikationslücke zu schließen. Dieser Ansatz unterstützt nicht nur das Wachstum von Espressif-Produkten, sondern trägt auch zu einer größeren Technologieszene bei. Unser Fokus auf Zugänglichkeit und Bildung stärkt aufstrebende Fachkräfte und fördert eine Kultur der Innovation und Kompetenzentwicklung. Auch wenn der Mangel an Talenten ein Problem darstellt, sind wir bestrebt, Menschen mit den notwendigen Werkzeugen und Kenntnissen auszustatten, um in der KI/IoT-Arena erfolgreich zu sein und sowohl Espressif als auch die Branche voranzubringen.

Was sind die Herausforderungen beim Marketing für Ingenieure im Gegensatz zu einem allgemeineren Zielpublikum?

— Erik Jansen (Niederlande)

Ingenieure beschäftigen sich sehr intensiv mit den technischen Details, daher müssen wir den Spagat zwischen Genauigkeit und Verständlichkeit schaffen. Unser Ziel ist es, ihnen zu zeigen, wie unsere Produkte genau zu ihren spezifischen Bedürfnissen und Herausforderungen passen. Sie sind sehr forschungsorientiert, daher sorgen wir dafür, dass sie alle technischen Informationen erhalten, die sie benötigen, um fundierte Entscheidungen treffen zu können. Wir wissen auch, dass sie viel Zeit in technischen Foren und Plattformen verbringen, also wollen wir dort sein, wo sie sind. Unser Marketing muss in diesen Bereichen wie ein „Leuchtturm“ wirken. Ingenieure legen Wert auf logische Entscheidungen, daher konzentrieren sich unsere Botschaften darauf, wie unsere Produkte ihren Projekten Wert, Nutzen und technische Innovation bringen. Es geht darum, ihre Sprache zu sprechen und ihnen zu zeigen, dass es uns darum geht, ihre Entwicklung reibungsloser und spannender zu gestalten.



Wie viele Ingenieure arbeiten bei Espressif? Erzählen Sie uns ein wenig über die Unternehmenskultur. Wie sieht das Betriebsklima für Ingenieure aus, die bei Espressif arbeiten? — C. J. Abate (USA)

Espressif ist im wahrsten Sinne des Wortes ein technologieorientiertes Unternehmen. Mehr als 75 % unserer Belegschaft sind Ingenieure, insgesamt rund 450 Mitarbeiter in den verschiedenen Entwicklungsabteilungen. Unsere Mitarbeiter haben vielfältige Werdegänge und gehören 30 verschiedenen Nationalitäten an. Sie sind in unseren Büros in fünf verschiedenen Ländern tätig: China, Indien, Tschechien, Singapur und Brasilien. Wir sind stolz auf unsere Unternehmenskultur, die unsere Mitarbeiter dazu ermutigt, innovative Ideen zu entwickeln, kalkulierte Risiken einzugehen und unsere Produkte und Dienstleistungen kontinuierlich zu verbessern. Jeder ist ansprechbar, und es wird eine Einstellung zu Innovation und Zusammenarbeit vermittelt. Die Mitarbeiter werden ermutigt, das zu tun, woran sie glauben und wofür sie sich begeistern.

Dies trägt dazu bei, einzigartige und bahnbrechende Lösungen zu schaffen.





Espressif scheint eines der stärksten RISC-V-Angebote für MCUs zu haben. Werden alle zukünftigen Geräte RISC-V verwenden? — Stuart Cording (Deutschland)

Wir sorgen dafür, dass unsere Hardware weithin zugänglich ist und unsere Software in der Open-Source-Community verfügbar ist. Dies ist eine Kernphilosophie von Espressif. Die Aufnahme von RISC-V-Prozessoren in unser MCU-Angebot war eine nahe liegende Entwicklung, und wir sind stolz auf das, was wir erreicht haben: einen problemlosen Übergang für Kunden zwischen den verschiedenen Produkten unter demselben Entwicklungsrahmen. Wir sind bestrebt, RISC-V weiter in unser Portfolio aufzunehmen und werden in Kürze unser erstes Dual-Core RISC-V-Produkt, den ESP32-P4, auf den Markt bringen. Die Einführung von RISC-V bietet uns Flexibilität bei der Implementierung und erweitert unser reichhaltiges Portfolio an geistigem Eigentum, was für die Bereitstellung fortschrittlicher und erschwinglicher Lösungen für unsere Kunden entscheidend ist.

Während sich der Markt mit 5G in Richtung deterministischer Netze mit hoher Rechengeschwindigkeit und Echtzeitreaktion bewegt, wird der Markt gleichzeitig sehr aufmerksam hinsichtlich des Energiebedarfs. Wie geht Espressif mit dieser scheinbar „unmöglichen“ Gleichung um? — Roberto Armani (Italien)

Es stimmt, dass Energieeinsparung und die Verringerung des Kohlendioxidausstoßes immer wichtige Ziele für unsere Kunden und deren Endverbraucher darstellen. Wir sind uns dessen bewusst und verbessern unsere Produkte und Lösungen kontinuierlich, um diesem wachsenden Bedarf gerecht zu werden. So haben wir beispielsweise unseren Ansatz für den leichten Schlafmodus überarbeitet, der es der Anwendung bei neueren Produkten wie dem ESP32-C6 und dem ESP32-H2 ermöglicht, die meisten Peripheriegeräte abzuschalten, was zu einer Reduzierung der Stromaufnahme um bis zu 80 % führen kann. Wir bieten auch Produktlösungen an, die den wachsenden Anforderungen an einen niedrigen Stromverbrauch gerecht werden, zum Beispiel der auf dem ESP32-C2 basierende ESP-NOW-Schalter, der 10.000 Tastendrücke mit einer einzigen Knopfzelle ermöglicht. Wir werden weiterhin innovativ sein und uns darauf konzentrieren, den Gesamtstromverbrauch unserer Produkte zu senken.

Espressif-Lösungen werden in Tausenden von Elektronikdesigns eingesetzt, von professionellen Anwendungen bis hin zu DIY-Maker-Projekten. Ihr Entwicklungsteam hat sicher einige Lieblingsprojekte, die Sie in der Community entwickelt haben. Können Sie uns drei der spannendsten oder innovativsten Espressif-basierten Projekte nennen, die durch die Community entwickelt wurden? — C. J. Abate (USA)

In unserer Community und in der Industrie stoßen wir immer wieder auf zahlreiche fesselnde Projekte. Diese Ausgabe enthält eine Auswahl dieser bemerkenswerten Projekte, die sowohl Hardware als auch Firmware umfassen. Darunter befinden sich ESP32-basierte Evaluierungsboards, die nicht größer als Knopfzellenbatterien sind, sowie geniale Projekte, bei denen Linux auf einem ESP32-S3 läuft. Wir haben auch Projekte kennengelernt, die Spielkonsole auf ESP-Produkten simulieren und synchronisierte Digitaluhren mit mehreren ESP-Boards realisieren. Andere haben sich mit der Entwicklung von VGA-Karten mit Espressif-Produkten oder der Portierung von Soundtreibern auf den ESP32 beschäftigt. Es wäre unfair, sich auf nur drei herausragende Projekte zu beschränken.

Wie viele Boards beziehungsweise Module haben Sie bisher verkauft? — Muhammed Söküt (Deutschland)

Unser erstes IoT-SoC-Produkt brachten wir 2014 mit der Veröffentlichung des beliebten ESP8266 auf den Markt. Es handelte sich um ein bahnbrechendes IoT-Produkt, das drahtlose Konnektivität und den Mikrocontroller auf demselben Chip vereinte. Der ESP8266 und das Flaggschiff ESP32, das 2016 auf den Markt kam, revolutionierten die Branche und führten bis 2017 zu einem Gesamtabsatz von 100 Millionen Einheiten. Unsere Kunden sind begeistert und wir haben uns als Unternehmen mit neuen und innovativen Produkten und Lösungen weiterentwickelt. Allein im Jahr 2021 haben wir mehr als 200 Millionen Produkte ausgeliefert. Wir haben kürzlich bekannt gegeben, dass wir mehr als eine Milliarde Geräte ausgeliefert haben.

Der Erfolg von Espressif begann mit dem erschwinglichen und einfach zu bedienenden ESP8266, der hauptsächlich für den Aufbau von WLAN-Verbindungen verwendet wurde. Heute werden die Chips und SoCs von Espressif zwar oft für die WLAN-Verbindung eingesetzt, aber andere Controller auf derselben Platine beschäftigen sich mit der Hauptanwendung. Der ESP32-P4 ist da anders, da er eine eigenständige Hochleistungs-CPU ist. Wird Espressif noch weiter in diese Richtung gehen? Werden wir zum Beispiel einen ESP64 sehen? — Jens Nickel (Deutschland)

Wir haben den ESP32-P4 entwickelt, nachdem wir gesehen haben, dass viele Designs den ESP32 ohne Konnektivität verwenden, und wir glaubten, dass wir eine leistungsfähigere und dennoch optimierte Lösung für diesen Bedarf anbieten können. Unsere Optimierungen der RISC-V-Implementierung haben es uns ermöglicht, eine leistungsstarke Multi-Core-MCU mit einer KI-Erweiterung zu entwickeln. Wir haben die Peripherie kontinuierlich verbessert. Damit sind wir gut gerüstet, um schnell neue System-on-a-Chip-Definitionen zu erstellen. Es ist jedoch vielleicht noch zu früh, um zu sagen, welche spezifischen „MCU-only“-Angebote wir in Zukunft haben werden.



Was sind Ihre Pläne für zukünftige Mikrocontroller (zum Beispiel Nachfolger des ESP32)? Können wir Dinge wie integriertes USB, 5-GHz-Wi-Fi oder Bluetooth 5.x erwarten? — Dr. Thomas Scherer (Deutschland)

Seit der Markteinführung von ESP32 im Jahr 2016 haben wir eine Reihe von Produkten auf den Markt gebracht: die ESP32-C-Serie, die ESP32-S-Serie und in letzter Zeit die ESP32-H- und die ESP32-P-Serie. Diese Produkte zielen darauf ab, die vielfältigen Anforderungen verschiedener Anwendungen und Branchen zu erfüllen. Die meisten dieser neu eingeführten Produkte unterstützen Bluetooth Low Energy 5, der ESP32-S2/S3 und der ESP32-C6 unterstützen USB OTG. Der ESP32-H2, der Anfang des Jahres in die Massenproduktion gegangen ist, unterstützt IEEE802.15.4-Konnektivität, wodurch er in Zigbee- und Thread-Anwendungen eingesetzt werden kann. Kürzlich haben wir auch den ESP32-C5 angekündigt, der Dual-Band (2,4 GHz und 5 GHz) Wi-Fi 6 unterstützen und in Kürze erhältlich sein wird. Der kommende ESP32-P4 wird einige fortschrittliche Mensch-Maschine-Schnittstellen (HMI) und Medienperipheriegeräte wie MIPI-DSI und MIPI-CSI mit integriertem Bildsignalprozessor (ISP), H.264-Encoder und so weiter unterstützen und eine Fülle neuer und vielfältiger Anwendungen ermöglichen.

Die Produkte von Espressif werden häufig in Produkten wie Haushaltsgeräten, Leuchten, intelligenten Lautsprechern, Unterhaltungselektronik und Zahlungsterminals eingesetzt. Gibt es Pläne von Espressif, die Einsatzmöglichkeiten seiner Produkte über die aktuellen Einsatzgebiete hinaus zu erweitern?
— Alina Neacșu (Deutschland)

Espressif hat sich zum Ziel gesetzt, den Zugang zu IoT-Technologien zu demokratisieren und mit innovativen, entwicklerzentrierten und erschwinglichen drahtlosen Konnektivitätslösungen zu segmentieren. Die Chips der ESP32-Serie werden weiter entwickelt und bieten bessere Konnektivität, höhere Rechenleistung, stärkere Sicherheit, eine zunehmend verbesserte Peripherie und einen geringeren Stromverbrauch. Darüber hinaus hat sich Espressif zu einem Anbieter von Komplettlösungen entwickelt, der die Probleme seiner Kunden identifiziert und diese mit Lösungen, die über die typischen Hardware- und Softwareentwicklungskits hinausgehen, effektiv angeht. Module wie ESP RainMaker, ESP Insights und ESP ZeroCode sind gute Beispiele dafür. Diese Lösungen und Produkte bedeuten aber nicht, dass wir uns auf ein bestimmtes Segment oder eine bestimmte Branche beschränken.



Viele Espressif-Produkte enthalten eine Art von Transmitter, häufig für die ISM-Bänder und mit integrierter Antenne. Wie stellt Espressif sicher, dass HF-Leistung, Bandbreite und Störaussendungen innerhalb der Spezifikationen und gesetzlichen Grenzwerte liegen? — Jan Buiting (Niederlande)

Der Industriesektor verlangt nach zuverlässigen und robusten Mikrocontrollern. Wie will Espressif seine Fortschritte und Fähigkeiten nutzen, um der Konkurrenz voraus zu sein, wenn es darum geht, die spezifischen Anforderungen des Industriemarktes zu erfüllen?
— Saad Imtiaz (Pakistan)

Der Industriemarkt stellt einige spezifische Anforderungen, die im Verbrauchersegment nicht üblich sind. Dazu gehören strengere Betriebsbedingungen wie höhere Temperaturen, größere Zuverlässigkeit (niedrige Ausfallraten) und Langlebigkeit, eine Unterstützung des Produkts über viele Jahre hinweg. Unsere Module und SoCs vertragen Temperaturen von bis zu 105°C, was sie für den Einsatz in den meisten industriellen Anwendungen geeignet macht. Außerdem verwenden unsere Produkte bewährte Halbleiterprozesse und werden umfangreichen Zuverlässigkeitstests unterzogen, um die geringstmöglichen Ausfallraten für unsere Kunden zu gewährleisten.

Bei der Entwicklung unserer Produkte berücksichtigen wir die Richtlinien und Spezifikationen für die verschiedenen Kommunikationsprotokolle und -medien, die von den zuständigen Behörden und Verbänden festgelegt wurden, und halten uns streng daran. Unsere Produkte und Module werden von externen Labors geprüft und zertifiziert, um sicherzustellen, dass sie den Spezifikationen und Grenzwerten der verschiedenen geografischen Regionen entsprechen. Diese Zertifizierungen sind auf unserer Website zu finden. Sie können von unseren Kunden genutzt werden, um die Zertifizierungsprozesse ihrer Produkte zu beschleunigen.

Übersetzung von Holger Neumann – (230604-02)

Rationelle MCU-Programmierung dank ESP Privilege Separation

Von Harshal Patil, Espressif

In diesem Artikel wird die Privilegentrennung in Mikrocontroller-Anwendungen durch ESP-IDF von Espressif Systems vorgestellt. Dabei wird die Firmware in einen geschützten Kern und eine Benutzeranwendung aufgeteilt, was die Entwicklung vereinfacht. Dieser Beitrag erleichtert den Einstieg und macht die MCU-Entwicklung effizienter.

Normalerweise werden Anwendungen für Mikrocontroller als monolithische Firmware entwickelt. In modernen Betriebssystemen gibt es jedoch zwei Betriebsmodi, den Kernel- und den Benutzermodus. Im Kernelmodus hat das Programm direkten und uneingeschränkten Zugang zu den Systemressourcen, während das Anwendungsprogramm im Benutzermodus davon abgeschirmt ist. Für Zugriffe auf Ressourcen müssen Systemaufrufe erfolgen.

Der wichtigste Aspekt dieser Trennung ist, die Entwicklung der Anwendung zu vereinfachen, ohne sich um die zugrundeliegenden Änderungen im System kümmern zu müssen, genau wie bei der Entwicklung von Anwendungen für Desktops oder Smartphones: Das zugrunde liegende Betriebssystem kümmert sich um die kritischen Funktionen, und die eigentliche Anwendung nutzt die vom Betriebssystem bereitgestellten Schnittstellen.

ESP-Privilege Separation

Traditionell wird jede ESP-IDF-Anwendung für SoCs von Espressif als eine einzige monolithische Firmware ohne jegliche Trennung zwischen den „Core“-Komponenten (Betriebssystem, Netzwerk und so weiter) und der „Anwendungs“- oder „Business“-Logik entwickelt. Im *ESP Privilege Separation Framework* wird das Firmware-Image in zwei separate und unabhängige Binärdateien aufgeteilt: in einen geschützten Bereich und die Benutzeranwendung.

Erste Schritte

Das erste Projekt Anfang ist „Blink“. Eine blinkende LED ist quasi das „Hello World“ für die Welt der Mikrocontroller und demonstriert die einfachste physikalische Ausgabe mit einer MCU. Los geht's!

Schritt 0: Hardwareanforderungen des Projekts

- Ein ESP32-C3 oder ESP32-S3-basiertes Entwicklungsboard mit integrierter LED. Einige Beispiele brauchbarer Devkits sind das ESP32-C3-DevKitM-1 oder das ESP32-S3-DevKitC-1. Aufgrund der Praktikabilität fällt die Entscheidung auf das ESP32-C3-DevKitM-1.
- Ein USB-2.0-Kabel (Standard-A auf Micro-B)
- Ein PC mit Windows, Linux oder MacOS

Schritt 1: Download des ESP Privilege Separation Projekts

- Klonen Sie das ESP Privilege Separation Projekt-Repository:
- ```
git clone https://github.com/espressif/esp-privilege-separation.git
```

### Schritt 2: Einrichtung der ESP-IDF-Umgebung

- Klonen Sie das ESP IDF Projekt-Repository:
- ```
git clone -b v4.4.3 --recursive https://github.com/espressif/esp-idf.git
```

➤ Einrichtung der Tools:

```
cd esp-idf  
./install.sh
```

➤ Einrichtung der Umgebungsvariablen:

```
source ./export.sh
```

➤ Anwendung des Patches für die ESP Privilege Separation auf ESP-IDF:

```
git apply -v /idf-patches/  
privilege-separation_support_v4.4.3.patch
```

Schritt 3: Ausführen des Blink-Beispiels.

- Erstellung des Beispiels:

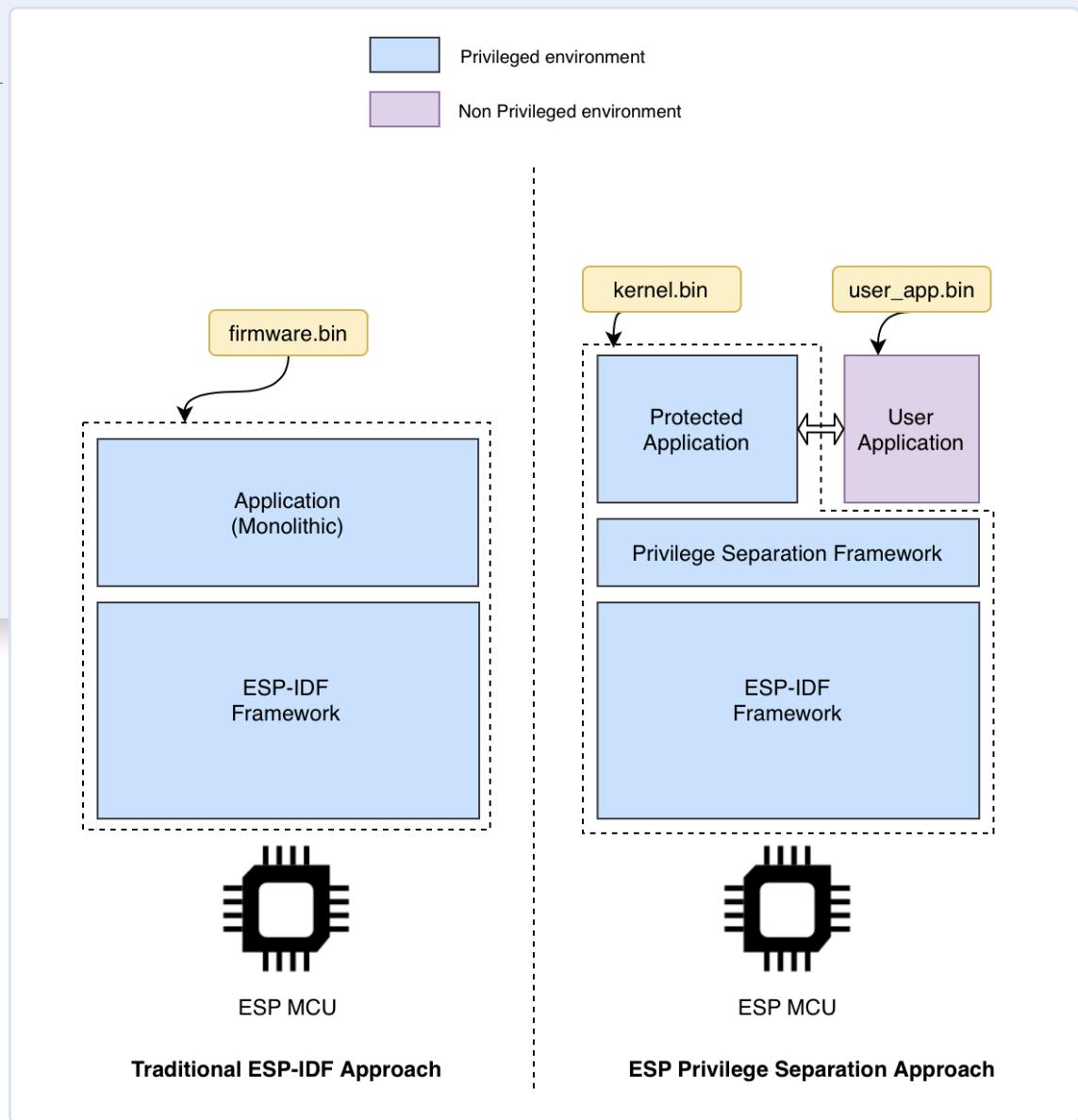


Bild 1: Vergleich eines herkömmliches ESP-IDF aus einer einzigen monolithischen Firmware mit dem ESP Privilege Separation Framework, bei dem das Firmware-Image aufgeteilt ist. (Quelle: Espressif)

```
cd /examples/blink
idf.py set-target esp32c3
idf.py build
```

➤ Flashen der Firmware und Ausführung des Beispiels:

```
idf.py flash monitor
```

Die Implementierung

Jetzt haben wir eine blinkende LED vor uns, doch inwieweit unterscheidet sich dieses Projekt von anderen Blinkprojekten? Die Antwort ist: Privilegientrennung. Es folgt eine Beschreibung der Implementierung:

- Das Beispiel ist in zwei Anwendungen aufgeteilt: die geschützte und die Benutzeranwendung.
- Die geschützte Anwendung sucht eine Benutzerpartition in der Partitionstabelle und lädt die Anwendung in die benutzerdefinierten Abschnitte.
- Anschließend konfiguriert sie die Speicherabschnitte für die Region mit niedrigeren Rechten ([WORLD1](#)) und gewährt den vom Entwickler konfigurierten Zugriff.
- Zum Schluss wird ein separater Task mit niedrigen Rechten mit

dem Benutzereinstiegspunkt erzeugt.

- Sobald die Benutzeranwendung geladen ist, wird ein Task gestartet, der die an GPIO8 (beim ESP32-C3-DevKitM-1) angeschlossene LED zum Blinken bringt.

Integration mit ESP RainMaker

Jetzt gibt es ein blinkendes LED-Devkit, aber es ist noch nicht klar, wie man den Status der LED per Befehl umschalten kann. Um das Devkit als echtes „verbundenes“ Gerät zu qualifizieren und die LED nach Bedarf zu steuern, muss es in ESP RainMaker integriert werden. ESP RainMaker ist eine einfache IoT-Cloud-Software, mit der Benutzer maßgeschneiderte IoT-Lösungen mit einem Minimum an Code und einem Maximum an Sicherheit entwickeln, erstellen und einsetzen können. Den Anfang macht die Erstellung des im Projekt ESP Privilege Separation enthaltenen Beispiels [rmaker_switch](#).

Schritt 0: Voraussetzungen

- ESP-IDF für ESP Privilege Separation wurde schon vor der Erstellung eines Beispiels eingerichtet.
- Die Mobil-Anwendung ESP RainMaker (Android/iOS).
- Ein WLAN.

Schritt 1: Einrichtung von ESP RainMaker

- Klonen des ESP RainMaker-Projekt-Repository:

```
cd /examples/rmaker_switch  
idf.py set-target esp32c3  
idf.py build
```

Schritt 2: Flashen der Firmware

- Flashen der Firmware und Ausführung des Beispiels:

```
idf.py flash monitor
```

Schritt 3: Bereitstellung des Geräts

- Sobald das Beispiel ausgeführt wird, wird ein QR-Code im Terminal angezeigt. Der QR-Code kann für die WLAN-Einrichtung verwendet werden.
- Anmeldung bei der Mobil-Anwendung ESP RainMaker und Klick auf *Add Device*. Dadurch wird die Kamera zum Scannen des QR-Codes aktiviert.
- Befolgung des Ablaufs für *Provision*, damit sich Ihr Gerät mit Ihrem WLAN verbinden kann.

Schritt 4: Steuerung des LED-Schalters

- Schließlich zeigt sich auf dem Startbildschirm der Mobil-App ein hinzugefügtes *Switch*-Gerät.
- Man kann nun das Schaltersymbol betätigen, um die LED zu steuern.

Zwecks nahtloser Integration von ESP Rainmaker wurde die Komponente `rmaker_syscall` in das Blink-Beispiel eingefügt, welche die Systemaufrufe für das ESP-Rainmaker-Framework bereitstellt. Beim Hochfahren der Benutzeranwendung initialisiert sie den ESP-Rainmaker-Service in der geschützten Anwendung und erstellt ein ESP-Rainmaker-Switch-Gerät. Da die ESP-Rainmaker-Komponente in der geschützten Anwendung steckt, sind für alle öffentlichen APIs von ihr bereitgestellte Systemaufrufe erforderlich. Die einfachen Erweiterungsfunktionen der ESP-Privilege Separation ermöglichen das Hinzufügen von anwendungsspezifischen, benutzerdefinierten Systemaufrufen sowie die Ausführung von Lese- und Schreibaufrufen für die im Gerätekontext gespeicherten Daten.

OTA-Firmware-Updates mit ESP-Privilege Separation

Over-the-Air beziehungsweise OTA-Firmware-Updates sind wichtige Funktionen für jedes angeschlossene Gerät. Sie ermöglichen die Bereitstellung neuer Funktionen und die Behebung von Fehlern per Fernaktualisierung. Bei der ESP-Privilege Separation gibt es mit `protected_app` und `user_app` zwei Anwendungen. Das Framework die Möglichkeit erlaubt die unabhängige Aktualisierung beider Binärdateien. Die geschützte Anwendung genießt höhere Privilegien und kann über die normale OTA-Schnittstelle des ESP-IDF aktualisiert werden, wobei auch die OTA-Aktualisierung der weniger privilegierten Benutzeranwendung möglich ist, da die ESP Privilege Separation dafür den Systemaufruf

`usr_esp_ota_user_app()` für die Benutzeranwendung bereitstellt.

Nachfolgend das OTA-Beispiel für die Benutzeranwendung des Projekts der ESP-Privilege Separation:

Schritt 0: Voraussetzungen

- ESP-IDF für ESP Privilege Separation wurde bereits vor der Erstellung des Beispiels eingerichtet.
- Eine öffentliche URL des gehosteten Images der neuen Benutzeranwendung.

Schritt 1: Ausführung des Beispiels `rmaker_switch`

- Erstellen des Beispiels:

```
cd /examples/esp_user_ota  
idf.py set-target esp32c3  
idf.py build
```

- Flashen der Firmware und Ausführung des Beispiels:

```
idf.py flash monitor
```

Schritt 2: Initiierung des OTA-Updates

Wenn die Benutzer-App die OTA-URL mit dem Befehl `user-ota` über die Konsole akzeptiert, wird die OTA-URL eingegeben. Dadurch wird ein OTA-Update initiiert, und die neue Benutzeranwendung startet, sobald das Update abgeschlossen ist. Jetzt ist das verbundene Gerät mit den wichtigsten Funktionen bereit.

Nun wird der QR-Code gescannt, um das Beispiel für die ESP Privilege Separation zu sehen, das eine reale Anwendung für ein OTA-Update einer Benutzeranwendung mit ESP Rainmaker und ESP Privilege Separation demonstriert.



Der Erfolg eines jeden Produktes liegt bekanntlich in den Händen seiner Nutzer - und das sind in diesem Fall Sie! Wir von Espressif sind sehr daran interessiert, Ihre Meinung zu hören. Darum bitten wir Sie um Ihre Erfahrungen und Vorschläge sowie um Bewertungen. Ihr Feedback hilft, unsere Produkte weiter zu verbessern, und ermöglicht somit auch, Ihre Bedürfnisse besser zu erfüllen. Besuchen Sie unsere Website [1], schreiben Sie uns eine E-Mail, vernetzen Sie sich mit uns in den sozialen Medien, um uns Ihre Erkenntnisse mitzuteilen. Sie können auch einen neuen Eintrag im GitHub-Repository [2] des Projekts machen. Wenn Sie eine spezielle Anforderung haben, von der Sie glauben, dass sie gut in dieses Framework passt, oder wenn es Sie reizt, solche Probleme zu lösen, würden wir uns sehr gerne mit Ihnen über eine Zusammenarbeit unterhalten. ↗

Übersetzung von Dr. Thomas Scherer -- 230598-02



Sie haben Fragen oder Kommentare?

Gerne können Sie sich an den Autor unter harshal.patil@espressif.com oder die Elektor-Redaktion unter der E-Mail-Adresse redaktion@elektor.de wenden.

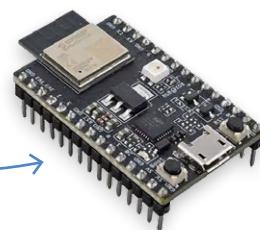
Über den Autor

Harshal Patil arbeitet als Software-Ingenieur bei Espressif Systems. Er beschäftigt sich mit der Lösung realer Probleme durch die Entwicklung sicherer, vernetzter Systeme. Er begeistert sich für Technik wie für Sport und verbringt seine Freizeit am liebsten mit der Erkundung neuer, innovativer Gadgets oder dem Fußballspielen.



Passende Produkte

- › **ESP32-C3-DevKitM-1**
www.elektor.de/20324
- › **ESP32-S3-DevKitC-1**
www.elektor.de/20697



WEBLINKS

- [1] Espressif: <https://espressif.com>
- [2] Projekt-Repository auf GitHub: <https://github.com/espressif/esp-privilege-separation/issues>



ESP-IDF

Das Espressif IoT Development Framework

ESP-IDF ist das offizielle Entwicklungs-SDK von Espressif, das alle SoCs der Serien ESP32, ESP32-S, ESP32-C und ESP32-H unterstützt. ESP-IDF ist der beste Ausgangspunkt, um alles zu entwickeln, was kein spezielles SDK auf ESP-IDF benötigt. ESP-IDF enthält den FreeRTOS-Kernel, Gerätetreiber, Flash-Storage-Stacks, Wi-Fi-, Bluetooth-, BLE-, Thread- und Zigbee-Protokollstacks, einen TCP/IP-Stack, TLS, Protokolle auf Anwendungsebene (HTTP, MQTT, CoAP) und viele andere Softwarekomponenten und Tools. ESP-IDF bietet auch häufig verwendete High-Level-Funktionen wie OTA und Netzwerkbereitstellung. Zusätzlich zur Befehlszeilenbedienung unterstützt es auch die Integration von Eclipse und VSCode-IDE. Es stehen viele Beispielanwendungen zur Verfügung,

die Ihnen einen schnellen Einstieg ermöglichen. ESP-IDF hat einen klar definierten Support- und Wartungszeitraum. Für die Entwicklung neuer Projekte wird die neueste stabile Version empfohlen.

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



Ein Open-Source

Spracherkennungsserver

...und die ESP-BOX

Von Kristian Kielhofner (USA)

Viele von uns mögen Alexa, Echo und ähnliche Geräte, aber es gab schon immer Bedenken bezüglich des Datenschutzes und der Datennutzung. Die Willow-Plattform ist eine freie und quelloffene Alternative. Neben dem Willow Inference Server wird eine hochwertige Sprachschnittstelle benötigt, die das Audiosignal erfasst und veredelt. Die ESP-BOX von Espressif bietet nicht nur Mikrofone und Audioverarbeitungsleistung zu einem attraktiven Preis, sondern wird auch von einem leistungsstarken Software-Ökosystem begleitet.

Seit der Einführung der Alexa-Plattform vor acht Jahren hat Amazon über 500 Millionen Echo-Geräte verkauft. Allerdings gab es immer wieder Bedenken und Kontroversen in Bezug auf den Datenschutz, die Datennutzung und die zunehmend lästigen Versuche von Amazon, Echo-Nutzer weiter zu zur Kasse zu bitten. Willow [1] ist eine Plattform, die eine kostenlose und quelloffene, herstellerfreundliche und zu Alexa konkurrenzfähige Sprachbenutzerschnittstelle bietet, ohne Abstriche bei der Qualität zu machen oder das Budget zu sprengen.

Raspberry Pi

Eine hochwertige Sprachbenutzerschnittstelle muss in der physischen Welt existieren und mit ihr interagieren. Viele Jahre lang hat

das Open-Source-Ökosystem versucht, mit Hilfe des Raspberry Pi, verschiedener Mikrofone und so weiter Sprachbenutzerschnittstellen zu entwickeln (**Bild 1**). Dieser Ansatz ist mit erheblichen Problemen verbunden:

➤ **Preis.** Wenn man einen Raspberry Pi, ein Touch-LCD, ein hochwertiges Mikrofon-Array, einen Lautsprecher, ein geeignetes Gehäuse und noch einiges mehr benötigt, sind die Kosten mindestens dreimal so hoch wie bei einem Echo-Gerät, das für 50 \$ oder weniger erhältlich ist. Die Einzelteile müssen beschafft, sorgfältig zusammengebaut, in einem

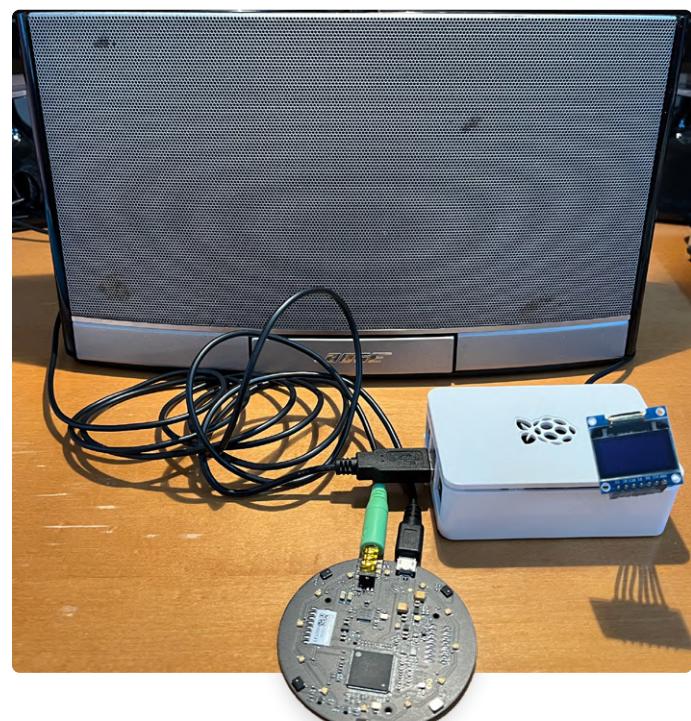


Bild 1. Aus früheren Versuchen des Autors mit dem Raspberry Pi.

eigens bearbeiteten Gehäuse untergebracht und nicht zuletzt die Software entwickelt werden. Wiederholen Sie dies für mehrere Geräte in Ihrer Umgebung, und der Zeit- und Kostenaufwand steigt beträchtlich.

- **Verfügbarkeit.** In letzter Zeit hat sich die Situation zwar verbessert, aber beim Raspberry Pi gab es gravierende Probleme mit der Lieferkette. In den letzten Jahren war die Beschaffung dieser Komponente nicht nur unmöglich, sondern aufgrund des Weiterverkaufs auch unnötig teuer.
- **Verwaltung.** Raspberry-Pi-basierte Lösungen erfordern oft eine vollständige Linux-Distribution mit Unterstützung für Hardwarekomponenten und einer verwirrenden Vielzahl von Software, die für eine Sprachbenutzeroberfläche erforderlich ist. Die Verwaltung eines halben Dutzend Linux-Computern kann für viele Benutzer eine Herausforderung darstellen.
- **Qualität.** Amazon (und andere) haben erhebliche Ressourcen in die Entwicklung einer Sprachbenutzerschnittstelle investiert, die in akustisch schwierigen Umgebungen mit einer Vielzahl von menschlichen Sprechern funktioniert. Es ist nicht ganz so einfach, ein Mikrofon und einen Lautsprecher adäquat an einen Pi anzuschließen.
- **Ökosystem.** Sobald man eine genaue Transkription eines Sprachbefehls erhalten hat, muss man damit etwas anfangen können und dem Benutzer eine Rückmeldung geben.
- **Leistung.** Mit den am meisten optimierten Implementierungen zur Spracherkennung kann ein Raspberry Pi 4 Echtzeit-Spracherkennung nur auf dem niedrigsten Qualitätslevel durchführen. Die Genauigkeit lässt zu wünschen übrig und „Echtzeit“ ist einfach nicht schnell genug - vor allem, wenn man bedenkt, dass ein Fehler in der Transkription dazu führt, dass man den Befehl wiederholen muss, wodurch der Komfortaspekt einer Sprachbenutzerschnittstelle verloren geht.

Tabelle 1 zeigt, dass ein Raspberry Pi etwas schneller ist als die Echtzeit-Spracherkennung mit dem niedrigsten verfügbaren Spracherkennungsmodell.

Interessanterweise haben die Spracherkennungsmodelle bei längeren Sprachsegmenten eine höhere Echtzeit-Multiple-Leistung, doch leider sind Sprachbefehle für Sprachassistenten sehr kurz, was zu erheblichen Leistungseinbußen führt. Dieser Benchmark begünstigt die Echtzeit-Spracherkennungsleistung eines Raspberry Pi. Wir alle kennen und lieben den Raspberry Pi - er ist fantastisch für eine breite Palette von Anwendungen und Anwendungsfällen, doch leider gehört ein Sprachassistent mit Spracherkennung und Sprachsynthese nicht dazu.

Wie wir im Screenshot eines Beispiels (**Bild 2**) sehen können, benötigte die Willow-Spracherkennung mit dem Willow Inference Server 222 Millisekunden vom Ende der Sprache bis zur Bestätigung der Aktion durch das Hausautomatisierungssystem Home Assistant. Das sind nur etwa 25 % der Zeit, die der Raspberry Pi 4 allein für die Spracherkennung benötigt - und das mit einem Spracherkennungsmodell, das wesentlich genauer ist als das „winzige“ Modell, das auf dem Raspberry Pi kaum in Echtzeit läuft.

Tabelle 1: Spracherkennungsleistung des Raspberry Pi 4.

Geräte	Modell	Beam-Größe	Sprachdauer (ms)	Inferenz-Zeit (ms)	Echtzeitfaktor
Pi	tiny	1	3.840	3.333	1,15x
Pi	base	1	3.840	6.207	0,62x
Pi	medium	1	3.840	50.807	0,08x
Pi	large-v2	1	3.840	91.036	0,04x

```
I (11:05:38.037) WILLOW/AUDIO: AUDIO_REC_WAKEUP_START
I (11:05:38.091) WILLOW/WAS: received text data on WebSocket: {
  "wake_start": {
    "hostname": "willow-7cdfa1e1aa84",
    "hw_type": "ESP32-S3-BOX",
    "mac_addr": [124, 223, 161, 225, 170, 132]
  }
}
I (11:05:38.207) WILLOW/AUDIO: AUDIO_REC_VAD_START
I (11:05:38.285) WILLOW/AUDIO: WIS HTTP client starting stream, waiting for end of s
I (11:05:38.287) WILLOW/AUDIO: Using WIS URL 'http://wis:20001/api/willow?model=smal
I (11:05:39.177) WILLOW/AUDIO: AUDIO_REC_VAD_END
I (11:05:39.178) WILLOW/AUDIO: AUDIO_REC_WAKEUP_END
I (11:05:39.217) WILLOW/AUDIO: WIS HTTP client HTTP_STREAM_POST_REQUEST, write end o
I (11:05:39.230) WILLOW/WAS: received text data on WebSocket: {
  "wake_end": {
    "hostname": "willow-7cdfa1e1aa84",
    "hw_type": "ESP32-S3-BOX",
    "mac_addr": [124, 223, 161, 225, 170, 132]
  }
}
I (11:05:39.270) WILLOW/AUDIO: WIS HTTP client HTTP_STREAM_FINISH_REQUEST
I (11:05:39.271) WILLOW/AUDIO: WIS HTTP Response = {"infer_time":47.108999999999995,
I (11:05:39.283) WILLOW/HASS: sending command to Home Assistant via WebSocket: {
  "end_stage": "intent",
  "id": 1693411539,
  "input": [
    {
      "text": "turn off dining room."
    },
    "start_stage": "intent",
    "type": "assist_pipeline/run"
  ]
}
I (11:05:39.399) WILLOW/HASS: home assistant response_type: action_done
I (11:05:39.401) WILLOW/HASS: received run-end event on WebSocket: {
  "id": 1693411539,
  "type": "event",
  "event": [
    {
      "type": "run-end",
      "data": null,
      "timestamp": "2023-08-30T16:05:39.426786+00:00"
    }
  ]
}
I (11:05:39.416) WILLOW/AUDIO: Using WIS TTS URL 'http://wis:20001/api/tts?format=W
```

Bild 2. Willow Spracherkennungsleistung.

ESP-Box

Dann entdeckte ich die Entwicklungsplattform ESP-BOX von Espressif (**Bild 3**). Wie wahrscheinlich viele von Ihnen verwenden ich seit zehn Jahren Espressif-Bausteine für eine Vielzahl von Anwendungen und weiß, wie robust, funktionsreich, kostengünstig und tatsächlich verfügbar die Hardware ist. Aus früheren Projekten weiß ich auch, dass Espressif eine Menge hochwertiger Unterstützungssoftware und Dokumentationen für ihre Hard- und Software bereitstellt.



Bild 3. Die ESP32-S3-BOX-3 verfügt über ein 2,4-Zoll-Display, zwei Mikrofone und einen Lautsprecher.

Die ESP BOX von Espressif ist eine Entwicklungsplattform, die speziell für diesen Anwendungsfall entwickelt wurde. Für etwa 50 \$ erhalten Sie bei Ihrem bevorzugten DIY-Elektronikhändler oder Distributor:

- Ein ESP32-S3 mit 16 MB Flash und 16 MB SPI-Hochgeschwindigkeits-RAM
- Ein 2,4"-Display mit kapazitivem Touchscreen
- Zwei Mikrofone (sehr wichtig - mehr dazu später)
- Lautsprecher für die Audioausgabe
- Hardware-Taste zum Stummschalten
- Viele Erweiterungsmöglichkeiten mit den neuen ESP32-S3-BOX-3-Baugruppen und Komponenten

All dies fertig in einem ästhetisch ansprechenden, akustisch optimierten Gehäuse. Theoretisch könnte ich mit der ESP-BOX und 50 \$ ein Gerät aus der Schachtel nehmen, flashen und in meiner Küche, meinem Schlafzimmer, meinem Büro oder wo es mir beliebt aufstellen.

Software

Espressif stellt nicht nur eine nahezu perfekte Hardware her, sondern setzt sich seit langem für Open Source ein. Ich war begeistert zu sehen, dass sie die ESP-BOX mit einer Vielzahl von freien und Open-Source-Bibliotheken unterstützen:

- ESP-IDF als grundlegendes SDK
- ESP-ADF für Audioanwendungen
- ESP-SR für die Spracherkennung
- ESP-DSP für hochoptimierte Signalverarbeitungsroutinen (FFT, Vektormathematik, et cetera)
- Eine LVGL-Komponente zur Ansteuerung von LCD-Displays (mit Touch)

Mit der ESP-BOX und diesen Bibliotheken habe ich innerhalb einer Woche einen sehr groben Entwurf entwickelt, der Sprache erfass-

sen, an meine Spracherkennungsimplementierung senden und das Transkript einer Plattform zur Verfügung stellen kann, um entsprechende Maßnahmen zu ergreifen.

Audio

Wie ich aus meinen früheren Fehlversuchen gelernt habe, beginnt eine Sprachschnittstelle mit dem Aufwachwort. Sie sollten in der Lage sein, eine Sprachschnittstelle mit einem bestimmten Wort anzusprechen, damit sie aufwacht und mit der Sprachaufzeichnung beginnt. Das Aufwachwort ist das Äquivalent zu einem Einschaltknopf – die Schnittstelle sollte sich nicht zufällig einschalten, und wenn Sie den Knopf drücken, sollte sie zuverlässig funktionieren. Wenn Sie sich bei Anwendungen mit Sprachschnittstelle mehrmals wiederholen müssen, damit sich das Gerät einschaltet, kommen Sie schnell auf den Gedanken, dass es schneller, einfacher und weitaus weniger frustrierend ist, einfach das Handy aus der Tasche zu nehmen und das zu tun, was Sie dort tun wollen.

Glücklicherweise bietet das Framework *ESP Speech Recognition* (SR) nicht nur eine Wake-Word-Engine, sondern auch mehrere Wake-Words. Ich habe festgestellt, dass die Weckimplementierung extrem zuverlässig ist - sie weckt konsistent und minimiert gleichzeitig falsche Weckaktivierungen. Dank ESP-SR hatte ich also einen zuverlässigen „Einschaltknopf“.

Dann die nächste Herausforderung – sauberes Audiosignal. Auch hier ist ESP-SR die Rettung. ESP-SR enthält das Espressif AFE (Audio Front End). Über das AFE allein könnte man viel schreiben, aber in einem Satz gesagt, bietet AFE eine Audioverarbeitungsschicht zwischen dem Mikrofoneingang und der Audioaufnahme, die Folgendes leistet:

- **Akustische Echokompensation** (Acoustic Echo Cancellation, AEC). Eine der vielen Herausforderungen bei der Fernfeld-Spracherkennung sind die akustischen Eigenschaften der physischen Umgebung. Entfernungen, harte reflektierende Oberflächen, verschlungene Audiowege (Ecken, Objekte im Weg) und so weiter können viele Echos aus der Umgebung einfangen. Die AEC-Implementierung in ESP-SR eliminiert einen Großteil dieser Echos und beseitigt außerdem das Echo in bidirektionalen Szenarien, zum Beispiel bei einer Freisprechanwendung.

- **Blindquellen-Trennung** (Blind Source Separation, BSS). In lauten Umgebungen ist es wichtig, Nicht-Sprachgeräusche zu eliminieren, die zu einer schlechten Qualität der Spracherkennung beitragen können. Die BSS-Implementierung in ESP-SR, die mehrere Mikrofone verwendet, kann die Audioerfassung im Wesentlichen auf die Richtung des eingehenden Tons „fokussieren“, um die erfassten Hintergrundgeräusche deutlich zu reduzieren.

- **Rauschunterdrückung** (Noise Suppression, NS). In Fällen, in denen nur ein Mikrofon vorhanden ist (oder nur ein Mikrofon aktiv ist), kann die Rauschunterdrückung die Erfassung nicht-menschlicher Töne erheblich reduzieren. Bei Anwendungen in kundenspezifischer Hardware können einzelne Mikrofone die Materialkosten und die Komplexität des Designs erheblich reduzieren.

› **Sprachaktivitätserkennung** (Voice Activity Detection, VAD). Ein zuverlässiger Weckruf ist nur ein Teil des Puzzles. Genau so, wie das Gerät aufwacht und mit der Audioerfassung beginnt, muss die Audioerfassung enden, wenn die Person zu Ende gesprochen hat. VAD ist in der Lage, Anfang und Ende der Sprache zu erkennen, so dass der Benutzer die Aufnahme nicht manuell beenden muss.

Willow Inference Server

Nun, da das Gerät aufwacht, saubere Sprache erfasst und am Ende der Sprache anhält, muss es die Information auch irgendwo hinschicken. Glücklicherweise bietet Espressif sein *Audio Development Framework* (ADF) für diese Aufgabe an. In meinem schnellen und schmutzigen Proof of Concept habe ich das HTTP-Stream-Pipeline-Beispiel von ESP-ADF verwendet, um die von AFE erfassten Audiodaten stückchenweise über HTTP POST an einen HTTP-Endpunkt zu senden. Ich konnte die Server-Implementierung meiner Spracherkennung schnell anpassen, um eingehende Audio-Frames von der ESP-BOX zu puffern, auf einen Endmarker zu warten (dank VAD) und diesen Puffer sofort an das zugrunde liegende Spracherkennungsmodell weiterzuleiten. Wenn der Spracherkennungsserver das Sprachtranskript zurückschickt, wird es als HTTP-JSON-Antwort an die ESP-BOX zur weiteren Ausführung übergeben. Diese Implementierung des Spracherkennungs-Inferenzservers wird als *Willow Inference Server* (WIS) bezeichnet.

Die ESP-BOX mit Willow ist in der Lage, das Sprachtranskript zu nehmen und es an einen vom Benutzer konfigurierten Home Assistant, OpenHAB oder einen benutzerdefinierten HTTP-REST-Endpunkt zu senden. Willow zeigt das Spracherkennungs-Transkript und die Antwort des Befehlsendpunkts auf dem Display an. Abhängig von der Benutzerkonfiguration spielt es auch einen Erfolgs-/Fehlerton ab oder verwendet Text-to-Speech vom Willow Inference Server, um den Ausgabetext zu sprechen, der aus dem Sprachbefehl resultiert.

Heute bin ich mit der ESP-BOX, Willow und dem Willow Inference Server in der Lage, ein Weckwort zu sagen, einen Befehl zu erfassen und ihn an den Home Assistant zu senden - mit einer Latenzzeit vom Sprachende bis zum Abschluss der Aktion von deutlich unter 500 ms (**Bild 4**), je nach WIS-Hardware und Konfiguration.

Multinet: Keine zusätzlichen Server oder Hardware

Die Magie von ESP-SR ist jedoch noch nicht vorbei. Zusätzlich zu den mitgelieferten Wake-Word-Modellen enthält ESP-SR ein geräteinternes Spracherkennungsmodell namens MultiNet für eine vollständig geräteinterne Sprachbefehlserkennung. ESP-SR bietet die Möglichkeit, bis zu 400 vordefinierte Sprachbefehle vollständig auf dem Gerät (ohne Willow Inference Server) zu erkennen. Willow unterstützt MultiNet und kann in Verbindung mit dem Home Assistant sogar die Namen der konfigurierten Einheiten abrufen, um diese Grammatik automatisch zu generieren (**Bild 5**) - ohne zusätzliche Komponenten und mit einer Leistung und Genauigkeit, die mit der des Willow Inference Servers für diese vordefinierten Befehle vergleichbar ist.

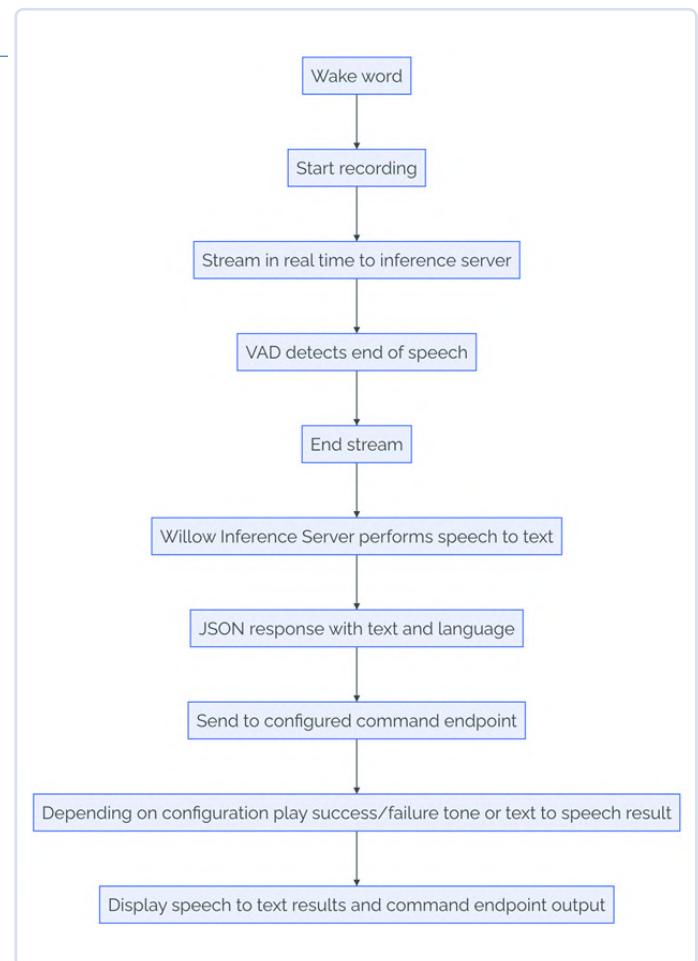


Bild 4. Ablauf des Willow-Inferenzservers.

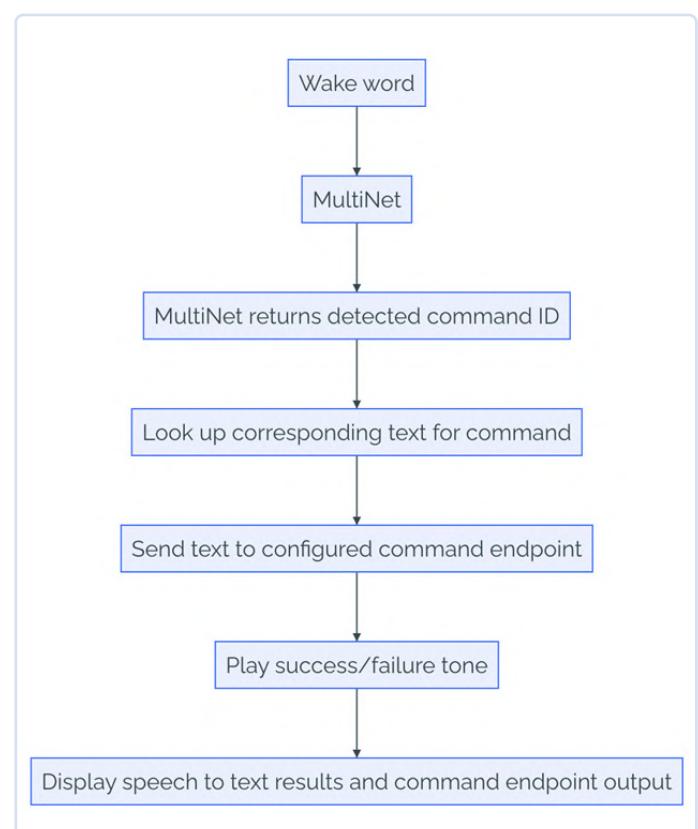


Bild 5. Ablauf des MultiNet-Modus.



Stets hersteller- und hackerfreundlich

Indem die ESP-BOX mit Willow Alexa-Geräte in wenigen Minuten ersetzen kann, bleibt Espressif seinen Maker-Wurzeln treu. Die ESP-BOX unterstützt eine breite Palette [2] von Komponenten mit verschiedenen Sensoren, GPIO, USB-Host-Schnittstelle und mehr über die Erweiterungsschnittstelle. Seriell und JTAG sind natürlich auch über den USB-C-Port am Hauptgerät verfügbar. Willow ist in C mit ESP-IDF geschrieben, aber die ESP-BOX unterstützt auch Plattformen wie Arduino, PlatformIO und CircuitPython. Mit Willow und der ESP-BOX haben wir jetzt den „Heiligen Gral“ der Open-Source-Sprachsteuerung in der Hand: eine Alexa-ähnliche Erfahrung zu einem ähnlichen Preis mit mehr Flexibilität, Kontrolle und vor allem vollständiger Privatsphäre, die auch die Open-Source-Software und Maker-Hardware-Schnittstellen bietet, die wir alle genießen! ↪

Übersetzung von Raphael Schermann – 230564-03

Über den Autor

Kristian Kielhofner ist der Gründer von Willow - einem Open-Source-Projekt zur Entwicklung eines lokalen, selbst gehosteten Sprachassistenten, der mit Amazon Echo und Google Home konkurriert. Seit er als Kind mit dem Apple Ile gespielt hat, ist Kristian Kielhofner Technologie-Enthusiast und hat mehrere Open-Source-Projekte und Startups in den Bereichen Sprache und Maschinelles Lernen ins Leben gerufen. Kristian verbringt jetzt seine Zeit mit der Arbeit an Willow, anderen Open-Source-Projekten und der Beratung von Technologieunternehmen in der Frühphase.



Passendes Produkt

› **ESP32-S3-BOX-3**

www.elektor.de/20627

WEBLINKS

[1] Willow-Plattform: <https://heywillow.io>

[2] ESP32-S3-BOX-3:

<https://www.espressif.com/en/news/ESP32-S3-BOX-3>



ESP-IoT-Solution

Gerätetreiber, Code-Frameworks und mehr für Ihre IoT-Systeme!

ESP-IoT-Solution ist ein Repository, in dem Sie viele Sensor-, Display-, Audio-, Eingabe- und Aktuator-Treiber-Implementierungen für Espressif-SoCs finden. Darüber hinaus bietet es auch einige häufig benötigte Frameworks auf höherer Ebene, zum Beispiel einen Drucktastentreiber, der lange und kurze Betätigungen erkennen kann. Es enthält auch spezifische Anwendungsbeispiele für Stromsparmodi, Speicherung und Sicherheit.

Wenn Sie BLE-basierte OTA-Updates direkt vom Handy auf das Gerät durchführen möchten, sollten Sie sich hier umsehen. Viele dieser Funktionalitäten sind als einzelne IDF-Komponenten verfügbar, die Sie direkt in Ihr Projekt importieren können.



<https://github.com/espressif/esp-iot-solution>



Das mitdenkende Auge

Gesichtserkennung und mehr mit ESP32-S3-EYE

Von Tam Hanna (Ungarn)

Mit dem ESP32-S3-EYE führt Espressif ein Board ins Feld, das speziell zur Evaluierung von Bilderkennung und anderen AI-Verfahren gedacht ist. Dazu kommt ein Ökosystem aus Software und vielen Beispielen - für das herstellereigene ESP-WHO-Framework, aber für den „manuellen Betrieb“ mit TensorFlow. Legen wir los!

Die Weiterentwicklung von Algorithmen der künstlichen Intelligenz erfolgte im Laufe der letzten Jahre in geradezu atemberaubender Geschwindigkeit. Es handelt sich inzwischen um hochleistungsfähige Systeme, die (das trifft insbesondere auf in der Cloud gehostete KI zu) beeindruckende und an Menschen erinnernde Leistungen erreichen. Moore'sches Gesetz und die immer höheren Ansprüche der Endan-

wender sorgen dafür, dass Mikrocontroller-Hersteller „AI-optimierte“ Chips ins Rennen schicken. Im Fall Espressif handelt es sich dabei um den ESP32 in der Variante S3, der mit den *Vector Instructions* eine Art AI-Beschleunigerengine beziehungsweise einen AI-spezifischen Befehlssatz mitbringt.

Im Laufe der letzten Monate entstand um diesen ESP32-S3 herum ein reiches Ökosystem, dass dem Entwickler die einfache Realisierung verschiedener Aufgaben der künstlichen Intelligenz ermöglicht. In diesem Artikel wollen wir mit dieser Technologie experimentieren.

Stressfreier Start

Espressif ist sich der Bedürfnisse der Entwicklerschaft gut bewusst: Seit dem vor einigen Jahren erschienenen und auf Audio-Aufgaben spezialisierten LyraT legt das Unternehmen immer wieder Evaluationsplatinen auf, die auf „Special Purpose-Anwender“ optimiert sind. Der Autor dieser Zeilen wird in den folgenden Schritten auf den ESP32-S3-Eye setzen - **Bild 1** bietet einen Überblick der enthaltenen Komponenten.

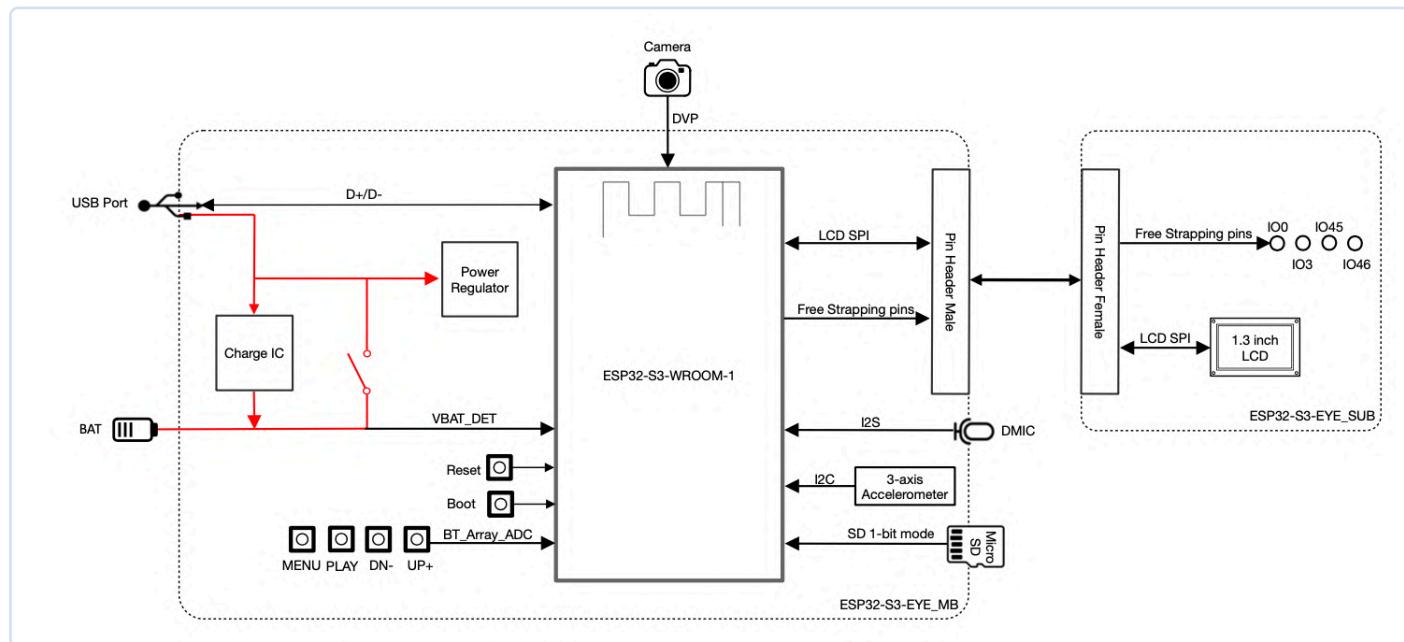


Bild 1. Klarer sehen mit Blockdiagramm! (Quelle: [10])

Für die Nutzung des mit rund 42 Euro (siehe [1]) nicht allzu teuren Evaluationsboards spricht, dass es sowohl die Kamera als auch ein kleines Display mitbringt - insbesondere die Verfügbarkeit eines Farbbildschirms erweist sich als sehr hilfreich, weil man die Ergebnisse des ML-Prozesses direkt und ohne Umweg über einen PC visualisiert bekommt.

Auf der Vorderseite des Boards findet sich außerdem noch ein digitales Mikrofon, so dass man verschiedenen Spracherkennungsengines evaluieren kann.

Angemerkt sei, dass die durchgeführten Experimente naturgemäß auch mit anderer Hardware durchgeführt werden können. Außerdem findet sich der Schaltplan des ESP32-S3-EYE unter [2] - es ist empfehlenswert, ihn als Basis für eigene Schaltungsdesigns heranzuziehen. Die von Espressif verwendeten Komponenten erweisen sich dabei im Allgemeinen als problemlos erhältlich.

Erhältlichkeit am Beispiel des ES8388.

Die Beschaffung der von Espressif bevorzugten Halbleiter setzt mitunter Umdenken voraus. Im Fall des Audio-Codecs ES8388 stellte der Autor im Rahmen einer Distributorsuche fest, dass kein westlicher Versender lieferfähig war. Eine Anfrage direkt beim Hersteller lieferte dann mit Newtech Component Ltd. aber einen Distributor, der sogar gleich 20 kostenlose Samples schickte. Zur Entgegennahme dieser Services ist es empfehlenswert, in China einen Freight Forwarder zu haben - der Autor setzt gerne auf die Dienstleistungen von TipTrans.

Im Interesse einfacherer Inbetriebnahme stattet Espressif die Platine außerdem mit einer Basis-Firmware aus. Falls Ihr ESP32-S3-EYE im jungfräulichen Zustand vorliegt, müssen Sie ihn lediglich über den Micro-USB-Port mit Energie versorgen und einige Sekunden warten. Lohn der Mühen ist das Starten einer Gesichtserkennungs-Applikation, die sich wie in **Bild 2** gezeigt verhält.

Angemerkt sei an dieser Stelle noch, dass AI-beziehungsweise ML-Algorithmen im Bereich ihrer Eingangs-Daten eine beeindruckende Resilienz an den Tag legen. Das in Bild 2 gezeigte Foto entstand mit angebrachter Schutzfolie auf der Kamera. Die dadurch einhergehende Kontrast-Reduktion (und das unrasierte Gesicht des Autors) reichten nicht aus, um den AI-Gesichtserkennungsprozess aus dem Konzept zu bringen.

Wiederherstellung des Beispielprojekts.

Möchten Sie Ihr ESP32-S3-EYE aus irgendeinem Grund in den „Auslieferungszustand“ zurückversetzen, so finden Sie unter der URL https://github.com/espressif/esp-who/tree/master/default_bin schlüsselfertige.bin-Dateien. Diese lassen sich wie jede andere Binärdatei auf das Board brennen.

Erste Experimente

Die zugegebenermaßen fast unbegrenzte Verfügbarkeit von Risikokapital hat zu einer wahren Schwemme von AI-Frameworks geführt. Aus

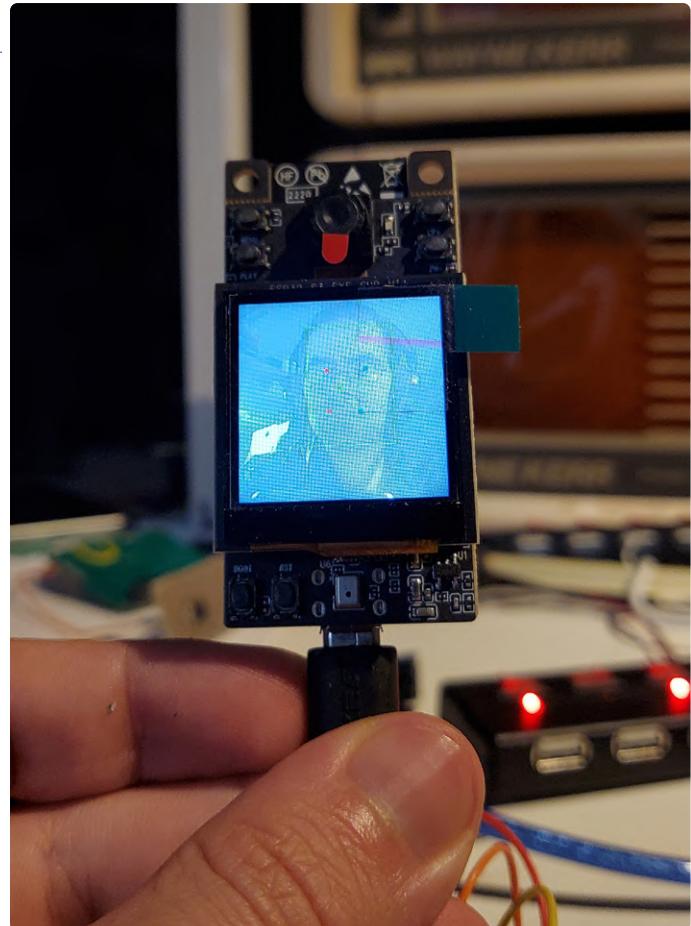


Bild 2. Die Kamera funktioniert sogar im schummerigen Labor des Autors.

der Logik folgt, dass derartige Unternehmen nicht nur PCs, sondern auch Mikrocontroller in die Liste der unterstützten Ziele aufnehmen. Ein kompletter Marktüberblick würde den Rahmen dieses Artikels sprengen, weshalb wir hier im ersten Schritt mit dem von Espressif verwalteten *ESP-WHO* arbeiten wollen. Dabei handelt es sich um ein für manche Bewegungsbilderkennungs-Arten „optimiertes“ Framework, dessen struktureller Aufbau in **Bild 3** gezeigt ist.

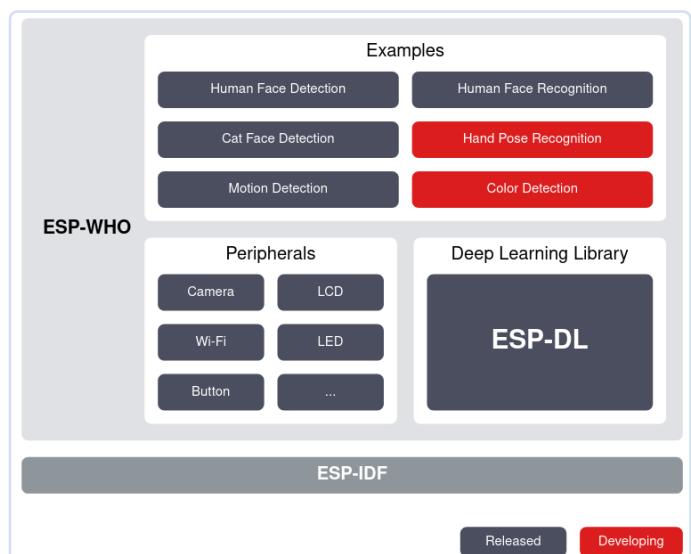


Bild 3. ESP-WHO nutzt diverse andere Teile des Espressif-Ökosystems (Quelle: [11]).

```
tamhan@TAMHAN18:~$ ls -l | grep "esp"
drwxrwxr-x 3 tamhan tamhan 4096 júl 13 2021 lcd
drwxr-xr-x 8 tamhan tamhan 4096 aug 7 04:30 esp4
drwxrwxr-x 3 tamhan tamhan 4096 máj 21 10:23 esp5
drwxrwxr-x 2 tamhan tamhan 4096 máj 12 04:41 esp_backups
drwxrwxr-x 4 tamhan tamhan 4096 jan 16 2023 esp_rust
-rw-rw-r-- 1 tamhan tamhan 589 jan 15 2023 export-esp.sh
drwxrwxr-x 3 tamhan tamhan 4096 aug 7 12:36 [REDACTED]
tamhan@TAMHAN18:~$
```

Bild 4. Verschiedene Versionen der *ESP-IDF* können im Allgemeinen auf einer Workstation koexistieren.

In besonderer Weise hervorzuheben ist die unter [3] zur Verfügung stehende *ESP Deep Learning Library*, kurz als *ESP-DL* bezeichnet. Im Prinzip handelt es sich dabei um eine optimierte Bibliothek, die verschiedene Verfahren der AI zur Verfügung stellt und bei Verwendung einer Beschleunigerengine eine wesentliche Einsparung im Bereich der Latenz erreicht.

Interessant ist, dass die schlüsselfertige WHO-Komponente *ESP-IDF* in Version 4.4 voraussetzt und noch nicht auf Version 5.0 umgestellt ist. Da der Autor für Consulting-Projekte ebenfalls noch 4.4 verwendet, hat er einfach, wie in Bild 4 gezeigt, mehrere Varianten der ESP32-Arbeitsumgebung am Rechner.

Die in den folgenden Schritten zum Einsatz kommende Version identifiziert sich folgendermaßen:

```
tamhan@TAMHAN18:~/esp4/esp-idf$ idf.py --version
ESP-IDF v4.4.4-dirty
```

Im nächsten Schritt sind wir für das Herunterladen des *ESP-WHO*-Codes bereit. Dies erfolgt durch Eingabe des folgenden `git`-Befehls in die Kommandozeile:

```
tamhan@TAMHAN18:~/esp4$ git clone --recursive https://github.com/espressif/esp-who.git
...

```

Wundern Sie sich während der Abarbeitung des Kommandos nicht, wenn der Schritt **Compressing objects**: einige Zeit in Anspruch nimmt - die Repositorien sind manchmal überlastet. Nach getaner Arbeit ist es noch empfehlenswert, nach folgendem Schema eine Initialisierung der Submodule durchzuführen:

```
tamhan@TAMHAN18:~/esp4$ cd esp-who/
tamhan@TAMHAN18:~/esp4/esp-who$ git submodule update
--recursive --init
```

In den meisten Fällen wird der Befehl `git submodule update --recursive --init` übrigens keine Inhalte am Bildschirm ausgeben - das informiert Sie darüber, dass unser vorliegendes Image „komplett“ ist.

Im Strom der Beispiele

ESP-WHO wird von Espressif mit einem Beispielkomplement ausgeliefert, das verschiedene Aufgaben illustriert. Interessant ist die in Bild 5 gezeigte Aufteilung der Beispielprojekte.

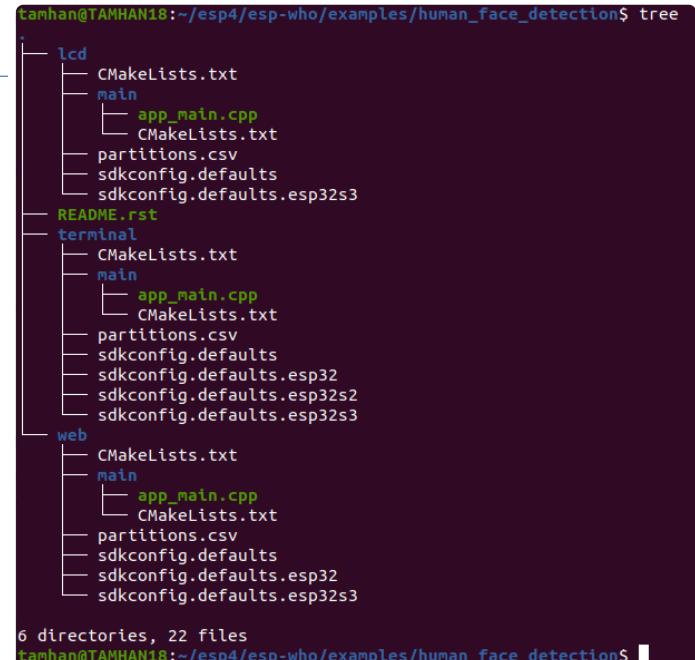


Bild 5. Variantenreichtum im Hause Espressif.

Wir wollen in den folgenden Schritten die LCD-Variante des Beispiels `~/esp4/esp-who/examples/human_face_detection` verwenden, die die errechneten Ergebnisse direkt auf einem mit dem ESP32 verbundenen Display zur Anzeige bringt. Die Terminal-Varianten nutzen stattdessen den `idf.py` Monitor zur Kommunikation, während *Web* einen Webserver aufspannt.

Im ersten Schritt ist es erforderlich, nach folgendem Schema den Typ des anzusprechenden ESP32-Controllers festzulegen. Wer wie der Autor eine Platine vom Typ ESP32-S3-EYE verwendet, muss hier den String `esp32s3` übergeben:

```
tamhan@TAMHAN18:~/esp4/esp-who/examples/human_face_detection/lcd$ idf.py set-target esp32s3
```

Der eigentliche, in der Datei `main/app_main.cpp` befindliche Quellcode des Beispiels, ist dann beeindruckend einfach - zur „Würdigung“ dieser Meisterleistung wird der Autor ihn hier im ersten Schritt als Ganzes abdrucken, um erst danach zu kommentieren (**Listing 1**).

Fleisch der *ESP-WHO*-Engine ist die Nutzung der in FreeRTOS implementierten und unter [4] im Detail beschriebenen *Queue*. Die beiden `static`-Deklarationen stellen eine Eingangs- und eine Ausgangs-Queue zur Verfügung, über die später der Datenfluss durch die Erkennungs-Pipeline bewerkstelligt wird.

Der Rest des vom Entwickler beizusteuernden Codes beschränkt sich dann darauf, durch Aufrufe der drei Methoden eine Pipeline zu errichten. Analogien zum Pipeline-Modell des Audio-Frameworks *ESP-ADF* [5] sind hier deutlich zu sehen.

Zur weiteren Inbetriebnahme müssen Sie im ersten Schritt `idf.py menuconfig` eingeben, um die von anderen ESP32-Projekten (und vom Linuxkernel) bekannte *menuconfig*-Konfigurationsumgebung zu laden.

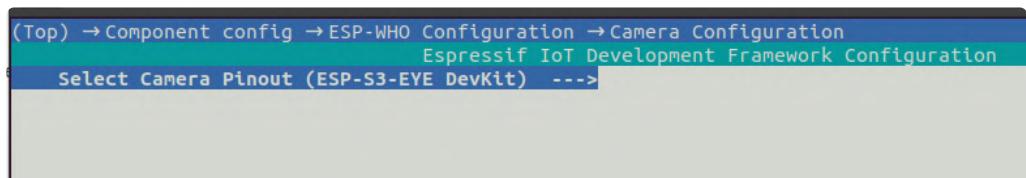


Bild 6. Wer ein hauseigenes Kamera-Modul benutzt, muss hier umkonfigurieren!



Listing 1. Queues als Basis.

```
#include "who_camera.h"
#include "who_human_face_detection.hpp"
#include "who_lcd.h"

static QueueHandle_t xQueueAIframe = NULL;
static QueueHandle_t xQueueLCDFrame = NULL;

extern "C" void app_main()
{
    xQueueAIframe = xQueueCreate(2, sizeof(camera_fb_t *));
    xQueueLCDFrame = xQueueCreate(2, sizeof(camera_fb_t *));

    register_camera(PIXFORMAT_RGB565, FRAMESIZE_240X240, 2, xQueueAIframe);
    register_human_face_detection(xQueueAIframe, NULL, NULL, xQueueLCDFrame, false);
    register_lcd(xQueueLCDFrame, NULL, true);
}
```

Wechseln Sie danach in die Rubrik *Component config ESP-WHO Configuration*. Im Feld *Camera Configuration* sollten Sie darauf achten, dass wie in **Bild 6** gezeigt die Kamera unseres Evaluationsboards vorkonfiguriert ist.

Speichern Sie die in *menuconfig* angelegte Buildkonfiguration danach, und geben Sie wie gewohnt durch Eingabe von *idf.py build* den Befehl zu einem Kompilationsprozess. Die erstmalige Erzeugung eines Images wird wie immer etwas mehr Zeit in Anspruch nehmen, weil das Framework gute 1200 Codedateien kompilieren muss.

Bevor die Auslieferung durch Eingabe von *idf.py flash* samt Übergabe des Ports erfolgen kann, müssen Sie das Board in den Bootloader-Modus versetzen. Hierzu stehen in der Nähe des USB-Connectors zwei Tasten zur Verfügung: Halten Sie BOOT gedrückt, während Sie RST touchieren. Danach können Sie das Programm wie gewohnt auf den ESP32 flashen. Die erfolgreiche Anmeldung des Bootloadermodus erkennen Sie in DMESG übrigens anhand der in **Bild 7** schematisch gezeigten Statusmeldung.

Nach dem abermaligen Drücken der RST-Taste startet eine neue Variante der aus Bild 2 bekannten Gesichtserkennung.

Kurzanalyse des Code Behind

Die Nutzung von *ESP-WHO* ermöglicht „AI-versen“ Entwicklern die aufwandslose Erzeugung von Applikationen der künstlichen Intelligenz. Da Espressif unter [6] einige der Komponenten im Quellcode zur Verfügung stellt, wollen wir oberflächlich einen Blick auf die Gesichtserkennungsroutine werfen.

Sie nutzt einen vorgefertigten Detektor vom Typ *HumanFaceDetect-MSR01*, der im Rahmen der Parametrisierung einige Gewichte eingeschrieben bekommt:

```
static void task_process_handler(void *arg) {
    camera_fb_t *frame = NULL;
    HumanFaceDetectMSR01 detector
        (0.3F, 0.3F, 10, 0.3F);
```

Die eigentliche Arbeit erledigt das ML-Modell dann in einer Endlosschleife, die nach folgendem Schema die einzelnen zu bearbeitenden Frames aus der weiter oben angelegten Queue entnimmt:

```
while (true) {
    bool is_detected = false;
    if (xQueueReceive(xQueueFrameI,
                      &frame, portMAX_DELAY)) {
        std::list &detect_results =
            detector.infer((uint16_t *)frame->buf,
                           {(int)frame->height, (int)frame->width, 3});
```

Aufrufe der Methode *detector.infer* sorgen dann dafür, dass der eigentliche Interferenz-Prozess zur Ausführung gelangt. So das zurückgegebene Objekt „Ergebnisse“ aufweist, ruft das Programm zwei Methoden für die Ausgabe aus:

```
if (detect_results.size() > 0) {
    draw_detection_result((uint16_t *)frame->buf,
                          frame->height, frame->width, detect_results);
    print_detection_result(detect_results);
    is_detected = true;
}
```

```
[ 8792.171038] usb 1-1.5: new full-speed USB device number 6 using ehci-pci
[ 8792.301169] usb 1-1.5: New USB device found, idVendor=303a, idProduct=1001, bcdDevice= 1.01
[ 8792.301175] usb 1-1.5: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 8792.301178] usb 1-1.5: Product: USB JTAG/serial debug unit
[ 8792.301181] usb 1-1.5: Manufacturer: Espressif
[ 8792.301183] usb 1-1.5: SerialNumber: 34:85:18:8C:50:D8
[ 8792.301642] cdc_acm 1-1.5:1.0: ttyACM0: USB ACM device
tanhan@TAMHAN18:~$
```

Bild 7 Diese Statusmeldung zeigt den erfolgreichen Verbindungsauflauf an.

Performance Comparison

A quick summary of ESP-NN optimisations, measured on various chipsets:

Target	TFLite Micro Example	without ESP-NN	with ESP-NN	CPU Freq
ESP32-S3	Person Detection	2300ms	54ms	240MHz
ESP32	Person Detection	4084ms	380ms	240MHz
ESP32-C3	Person Detection	3355ms	426ms	160MHz

Bild 8. Wer den AI-Beschleuniger aktiviert, rechnet wesentlich schneller (Quelle: [9]).

Der Rest von *ESP-WHO* besteht im Allgemeinen aus recycelten Komponenten, der Kamera-Zugriff erfolgt beispielsweise unter der unter [7] bereitstehenden *ESP-Cam*. Aus diesem Grund wollen wir unsere Besprechungen von *ESP-WHO* an dieser Stelle „pausieren“ – die bereitgestellten Beispiele sind sprechend.

Eine Ebene tiefer mit TensorFlow

Wer mehr Kontrolle über das Verhalten seines Gesamtsystems haben möchte, ist mitunter versucht, komplett von Hand zu kodieren. Diese Vorgehensweise ist schon deshalb wenig sinnvoll, weil man in diesem Fall alles manuell realisieren darf und das Finden von Maschine-Learning- beziehungsweise Data-Science-Experten, die mit dem vorliegenden System zurecht kommen, in komplizierte Arbeit ausartet. Googles unter [8] zur Verfügung stehende *TensorFlow*-Bibliothek hat sich als Quasistandard etabliert und steht seit längerer Zeit auch als für verschiedene Mikrocontroller optimierte Varianten zur Verfügung. Zu beachten ist bei seiner Nutzung vor allem, dass auf GitHub zwei Repositorien zur Verfügung stehen: Dies ist eine „historische Last“, weil Google die Verteilung von generischen und Vendor-spezifischen Code in der *TensorFlow*-Bibliothek im Rahmen der Entwicklung des Frameworks einmal verschoben hat. Sei dem wie es sei, steht die von uns benötigte Variante der Bibliothek unter [9] zur Verfügung.

Da auch *TensorFlow* im Hintergrund auf verschiedene Komponenten des ESP-IDF-Frameworks zurückgreift, muss das Deployment aus GitHub auch hier unter Übergabe des Parameters `--recursive` erfolgen. Er weist das Kommandozeilenwerkzeug auf die Notwendigkeit hin, verknüpfte Coderepositorien ebenfalls bereitzustellen:

```
tamhan@TAMHAN18:~/esp4$ git clone --recursive https://github.com/espressif/tflite-micro-esp-examples.git
```

Für einen ersten Rauchtest bietet sich dann das Beispiel `~/esp4/tflite-micro-esp-examples/examples/hello_world` an. Es nutzt ein ML-Modell, das auf die „Vorhersage“ der Werte der Sinusfunktion optimiert ist - „Sinn“ dieser normalerweise durch CORDIC und Co. wesentlich vernünftiger lösbarer Aufgabe ist die Bereitstellung eines Modells, das mit minimalen Ansprüchen an die Eingabedaten zur Ausführung gebracht werden kann.

Im nächsten Schritt erfolgt abermals die Eingabe von `idf.py set-target esp32s3`, um das Projektskelett auf den ESP32-S3 zu optimieren.

Je nach dem auf der Workstation vorliegenden Versionsstand der ESP-IDF kommt es bei der Parametrisierung des aus dem Repository heruntergeladenen Codes mitunter zu Fehlermeldungen (*Invalid manifest ..*).

Sie deuten auf teilweise veraltete Komponenten in der Kompilations-Toolchain hin. Zur Behebung reicht es aus, in der Kommandozeile folgendes Programm zur Ausführung zu bringen:

```
/home/tamhan/.espressif/python_env/idf4.4_py3.8_env/bin/python -m pip install --upgrade idf-component-manager
```

Öffnen Sie im nächsten Schritt durch Eingabe des folgenden Befehls ein *Nautilus*-Fenster, das auf den Stammordner des Projektskeletts verweist. Das Löschen des *build*-Ordners muss von Hand erfolgen, um eine abermalige Errichtung der Kompilations-Unterstützungsdateien durch den Befehl `set-target` freizuschalten:

```
tamhan@TAMHAN18:~/esp4/tflite-micro-esp-examples/examples/hello_world$ nautilus .
tamhan@TAMHAN18:~/esp4/tflite-micro-esp-examples/examples/hello_world$ idf.py set-target esp32s3
tamhan@TAMHAN18:~/esp4/tflite-micro-esp-examples/examples/hello_world$ idf.py menuconfig
```

Interessant ist in *Menuconfig* vor allem der Eintrag *ESP-NN*, der die Konfiguration des Verhaltens der *DL*-Bibliothek ermöglicht. Die Auswahl der Option *Optimized versions* in der Rubrik *Optimization for nn functions* ist in höchstem Maße ratsam, weil sie den Beschleuniger aktiviert. Er ermöglicht, wie in Bild 8 gezeigt, erhebliche Steigerungen der Performance.

```
I (292) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
x_value: 0.000000, y_value: 0.000000
x_value: 0.314159, y_value: 0.372770
x_value: 0.628319, y_value: 0.559154
x_value: 0.942478, y_value: 0.838731
x_value: 1.256637, y_value: 0.965812
x_value: 1.570796, y_value: 1.042060
x_value: 1.884956, y_value: 0.957340
x_value: 2.199115, y_value: 0.821787
x_value: 2.513274, y_value: 0.533738
x_value: 2.827433, y_value: 0.237217
x_value: 3.141593, y_value: 0.008472
x_value: 3.455752, y_value: -0.304993
x_value: 3.769912, y_value: -0.533738
x_value: 4.084070, y_value: -0.779427
x_value: 4.398230, y_value: -0.965812
x_value: 4.712389, y_value: -1.109837
```

Bild 9. Sinus, mal per neuronalem Netzwerk.

Die eigentliche Kompilation und Ausführung erfolgt dann wie erwartet. In Bild 9 sehen Sie die Ausgabe der errechneten Ergebnisse.

Analyse des TensorFlow-Codes

Wundern Sie sich übrigens nicht dabei, wenn am Display des ESP-EYE chaotische Werte erscheinen - das verbaute LCD hat einen eigenen Framebuffer-Controller, der bei Nicht-mehr-Aktualisierung einfach den zuletzt in ihm abgelegten Wert weiter zur Anzeige bringt.

Wer die Datei *main.cc* hoffnungsfroh in einen Texteditor seiner Wahl öffnet, findet folgendes Snippet:

```
#include "main_functions.h"

extern "C" void app_main(void) {
    setup();
    while (true) {
        loop();
    }
}
```

Der erste Eindruck täuscht in diesem Bereich nicht – TensorFlow richtet sich eng an der Arduino-Umgebung aus. Die eigentliche Implementierung der Methoden `setup` und `loop` findet sich in der Datei *main_functions.cc*.

Besonders interessant ist naturgemäß die Methode `loop`, die für die eigentliche Ausführung der Payloads verantwortlich zeichnet. Ihre erste Amtshandlung ist das Erzeugen von Eingabedaten, mit denen wir unseren Sinus-Predictor füttern wollen:

```
void loop() {
    float position =
        static_cast<inference_count> /
        static_cast<kInferencesPerCycle>;
    float x = position * kXrange;
```

Im nächsten Schritt werden die Informationen quantifiziert, um sie für das neuronale Netzwerk bekommlich zu machen. Dabei handelt es sich um eine im ML-Bereich durchaus populäre Vorgehensweise: Wer alle Eingangswerte in den Wertebereich von 0...1 normalisiert, kann die entstehende algorithmische Komplexität im System besser unter Kontrolle halten.

```
int8_t x_quantized =
    x / input->params.scale +
    input->params.zero_point;
input->data.int8[0] = x_quantized;
```

Die eigentliche Berechnung und Quantifizierung erfolgt dann in folgendem Block:

```
TfLiteStatus invoke_status =
    interpreter->Invoke();
if (invoke_status != kTfLiteOk) {
    MicroPrintf("Invoke failed on x: %f\n",
    static_cast<x>);
    return;
```

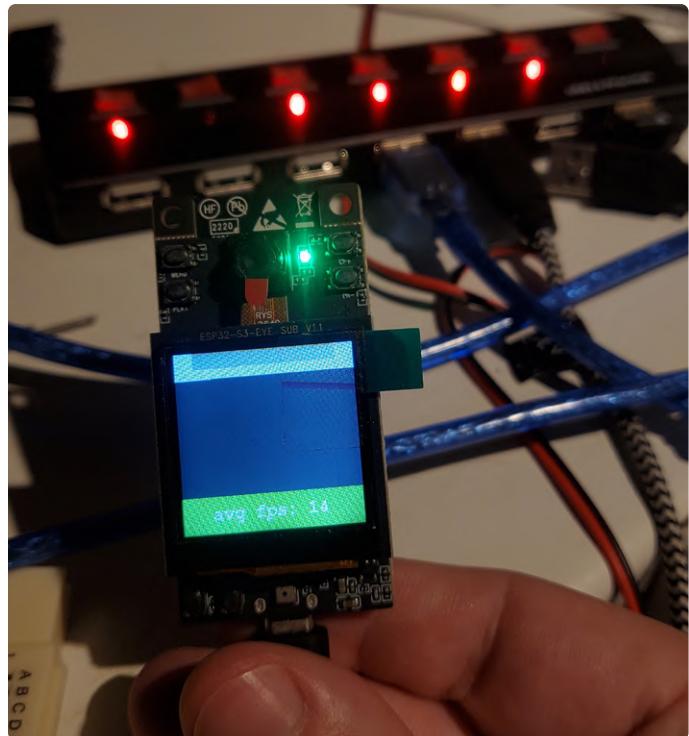


Bild 10. Die grüne Fußzeile zeigt eine erfolgreiche Erkennung an.

```
}
```

$$\text{int8_t } y_{\text{quantized}} = \text{output}->\text{data}.int8[0];$$

$$\text{float } y = (y_{\text{quantized}} -$$

$$\text{output}->\text{params}.zero_point) *$$

$$\text{output}->\text{params}.scale;$$

Zu guter Letzt ist Housekeeping erforderlich - interessant ist, dass die für die eigentliche Ausgabe der Daten zuständige Methode `HandleOutput` laut der TensorFlow-Micro-Dokumentation vom Entwickler zu stellen ist:

```
HandleOutput(x, y);
inference_count += 1;
if (inference_count >= kInferencesPerCycle)
    inference_count = 0;
}
```

Experimente mit höherwertigen Modulen

Wer den von Google ausgelieferten Ordner *~/esp4/tflite-micro-esp-examples/examples* sorgfältig analysiert, stellt fest, dass mit *micro_speech* auch ein Spracherkennungsbeispiel und mit *person_detection* sogar ein Gesichtserkennungsbeispiel zur Verfügung steht.

Die „Inbetriebnahme“ beider Codes erfolgt dabei durch den von oben bekannten Dreikampf aus Parametrisierung, Überprüfung der *menuconfig*-Einstellungen und Kompilation sowie Auslieferung des Maschinencodes auf den ESP32.

Zur Nutzung dieser fortgeschrittenen Beispiele ist zu beachten, dass diese normalerweise nur auf Kommandozeilenebene arbeiten. Wer dem Personen-Detektor die Fähigkeit zur Bildschirmausgabe hinzufügen möchte, muss die Datei *esp_main.h* nach folgendem Schema adaptieren:

```
// Enable this to do inference on embedded images
#define CLI_ONLY_INFERENCE 1
#define DISPLAY_SUPPORT 1
```

Nach einer Rekomplilation erscheint dann der in **Bild 10** gezeigte Bildschirm. Die Behebung der Fehl-Darstellung wäre Thema eines weiteren Artikels - das „Grün-werden“ der Fußzeile informiert darüber, dass die Erkennung erfolgreich verlief.

Zwei Wege

Unsere hier durchgeföhrten Experimente zeigen, dass die ESP32-S3-Plattform auf verschiedene Arten zur Ausführung von Objekt- und Personenerkennungspayloads befähigt ist. Während die Nutzung von ESP-WHO das schnelle Erreichen beeindruckender Ergebnisse erlaubt, ermöglicht die Integration in das TensorFlow-Ökosystem die Nutzung fortgeschrittenster Verfahren aus dem Bereich ML. ↵

230556-02

Sie haben Fragen oder Kommentare?

Gerne können Sie sich an den Autor unter der E-Mail-Adresse tamhan@tamoggemon.com oder an die Elektor-Redaktion unter der E-Mail-Adresse redaktion@elektor.de wenden.



Passendes Produkt

› **ESP32-S3-EYE**

www.elektor.de/20626

WEBLINKS

- [1] ESP32-S3-EYE Distributorensuche: <https://www.oemsecrets.com/compare/ESP32-S3-EYE>
- [2] Schaltplan des ESP32-S3-EYE:
https://github.com/espressif/esp-who/blob/master/docs/en/get-started/ESP32-S3-EYE_Getting_Started_Guide.md#52-schematic
- [3] ESP Deep Learning Library: <https://github.com/espressif/esp-dl>
- [4] FreeRTOS Queues: <https://www.freertos.org/a00018.html>
- [5] Tam Hanna, „Audio mit dem ESP32“, Elektor 1-2/2023: <https://www.elektormagazine.de/magazine/elektor-289/61384>
- [6] AI-Beispiele ESP-WHO: <https://github.com/espressif/esp-who/tree/master/components/modules/ai>
- [7] ESP32 Kamera-Zugriff : <https://github.com/espressif/esp32-camera>
- [8] TensorFlow: <https://www.tensorflow.org/>
- [9] TensorFlow Lite Micro: <https://github.com/espressif/tflite-micro-esp-examples>
- [10] ESP32-S3-EYE Block-Diagramm:
https://github.com/espressif/esp-who/blob/master/docs/en/get-started/ESP32-S3-EYE_Getting_Started_Guide.md
- [11] ESP-WHO auf GitHub: <https://github.com/espressif/esp-who>



Arduino-ESP32

Die ganze Unterstützung, die Sie für ESP32, ESP32-S2, ESP32-S3 und ESP32-C3 brauchen



In diesem Repository ist die Arduino-Unterstützung für ESP32, ESP32-S2, ESP32-S3 und ESP32-C3 Chips verfügbar. Sie können die „normale“ Arduino-IDE oder die PlatformIO-IDE für die Entwicklung von Arduino-basierten Anwendungen auf diesen Chips verwenden. Dieses Repository enthält viele nützliche Bibliotheken, darunter häufig verwendete Gerätetreiber, Protokollunterstützung für Wi-Fi, BLE und ESP-NOW sowie High-Level-Funktionen wie ESP-Insights und ESP-RainMaker.

All diese Bibliotheken enthalten Beispiele, mit denen die Funktionalität demonstriert werden kann. Es werden auch viele Entwicklungsboards unterstützt, die auf den oben genannten Chips basieren.

<https://github.com/espressif/arduino-esp32>



Knopfzellen-Sender mit ESP32-C2

Entwurf und Leistungsbewertung

Von Li Junru und Zhang Wei, Espressif

Dieser ESP32-C2-basierte Knopfzellen-Sender ist eine Lösung für den Smart-Home-Markt. Er bietet direkte Kommunikation mit ESP-Geräten, stabile bidirektionale Kommunikation, kompakte Größe und eine Batterielebensdauer von bis zu fünf Jahren. Dieser Artikel beschreibt die Software- und Hardware-Implementierung und zeigt die Möglichkeiten für moderne Smart-Home- und IoT-Anwendungen auf.

Mit der rasanten Entwicklung des Smart-Home-Marktes steigt die Nachfrage nach Funkschaltern mit geringer Stromaufnahme und hoher Effizienz. In diesem Artikel wird eine innovative Lösung vorgestellt: ein ESP32-C2-basierter Knopfzellen-Sender (**Bild 1**). Er zielt darauf ab, Herausforderungen wie verzögerte Reaktion zu bewältigen und die Notwendigkeit zusätzlicher Gateways zu eliminieren, wie dies bei anderen drahtlosen Schalterlösungen, die auf Technologien wie Bluetooth LE und ZigBee basieren, häufig der Fall ist.

Der Knopfzellen-Sender mit ESP32-C2 bietet mehrere Vorteile gegenüber anderen drahtlosen Schalteralternativen:

- Kommuniziert direkt mit zum Beispiel intelligenten Leuchten oder Wandschaltern, die mit ESP-Chips ausgestattet sind.

Dadurch wird ein zusätzliches Gateway überflüssig.

- Sorgt für eine stabile bidirektionale Kommunikation und garantiert eine geringe Fehlerquote bei der Übertragung von Wi-Fi-Paketen.
- Durch den Betrieb mit Knopfzellen werden die Größe des Geräts minimiert und vielseitige Formfaktoren wie Klebeschalter, Mehrtastenschalter, Touch-Schalter oder Drehschalter unterstützt.
- Der ESP32-C2 schaltet sich vollständig ab, wenn er nicht benutzt wird. Dadurch kann eine einzelne CR2032-Knopfzelle bis zu fünf Jahre halten (bei zehn Betätigungen pro Tag)!

In diesem Artikel gehen wir auf die umfassenden Implementierungsdetails des ESP-Knopf-

zellen-Senders ein und zeigen, wie hervorragend er die Anforderungen der modernen Smart-Home-Technologie erfüllt.

Knopfzellen wie die häufig verwendete CR2032 sind eine weit verbreitete Spannungsquelle für IoT-Geräte. Knopfzellen sind für ihr kompaktes Gehäuse und ihr geringes Gewicht bekannt und eignen sich daher gut für kleine elektronische Geräte, ohne dass diese an Masse oder Gewicht über Gebühr zunehmen. Dank ihrer hohen Energiedichte können sie mehr Energie in einem kleineren Volumen speichern, was zu einer längeren Nutzungsdauer führt. Darüber hinaus weisen Knopfzellen eine niedrige Selbstentladungsrate auf, was bedeutet, dass sie ihre Ladung auch bei längerem Nichtgebrauch beibehalten. Mit ihrer stabilen Ausgangsspannung während der Entladung spielen Knopfzellen eine wichtige Rolle beim Betrieb von drahtlosen Geräten. Da sie jedoch aufgrund



Bild 1. Der ESP32-C2-basierte Knopfzellen-Sender im Gehäuse.



Bild 2. Das ESP-NOW-Modell fasst die oberen fünf Schichten des OSI-Modells in einer Schicht zusammen.

ihrer Bauweise in der Regel eine geringere Stromabgabe aufweisen, sind sie für Geräte mit hoher Leistung ungeeignet.

In den verschiedensten Anwendungsbereichen werden IoT-Geräte mit Wi-Fi-Technologie in großem Umfang eingesetzt, da sie von der großen Verfügbarkeit von WLANs profitieren. Durch den Wegfall herkömmlicher Kabelverbindungen zwischen Geräten ermöglicht die WiFi-Technologie eine flexiblere und bequemere Anwendung und Verwaltung von IoT-Geräten. Darüber hinaus unterstützt die WiFi-Technologie gleichzeitige Verbindungen zu mehreren Geräten, was sie für IoT-Szenarien mit zahlreichen vernetzten Geräten besonders wertvoll macht. Der Markt bietet eine Fülle von Geräten und Komponenten, die WiFi unterstützen, wodurch die Entwicklungs- und Produktionskosten von IoT-Geräten erheblich gesenkt werden. Dennoch ist es wichtig zu bedenken, dass WiFi-Geräte im Vergleich zu anderen drahtlosen Technologien mit geringer Stromaufnahme relativ viel Energie verbrauchen. Daher ist WiFi für bestimmte IoT-Geräte, die auf Batteriebetrieb angewiesen sind, möglicherweise ungeeignet. Der ESP32-C2 weist zum Beispiel einen maximalen Sendestrom von 370 mA und einen maximalen Empfangsstrom von 65 mA während des HF-Betriebs auf.

Der Energiebedarf von WiFi-Geräten stellt zwei große Herausforderungen dar, wenn sie in Bereichen mit geringer Stromaufnahme eingesetzt werden. Einerseits macht der hohe Betriebsstrom im Empfangsmodus es schwierig, den Dauerbetrieb im Empfangszustand aufrechtzuerhalten. Andererseits kann der kurzfristig hohe Strom während der Übertragung von Paketen die Spannungsstabilität der Chip-Stromversorgung beeinträchtigen, was zu einem Reset des Chips führen könnte. In diesem Artikel stellen wir einen auf dem ESP32-C2 basierenden Knopfzellen-Sender vor, der diese Herausforderungen durch eine ausgewogene Kombination aus Software und Hardware meistert, was sich in einer beeindruckenden Batterielebensdauer niederschlägt.

Der Artikel gliedert sich in drei Abschnitte. Im ersten Abschnitt befassen wir uns mit der Software-Implementierung. Wir geben einen detaillierten Einblick in die Methodik des Programms und die jeweiligen Ergebnisse. Im letzten Abschnitt werden die tatsächliche Leistung und die Paketlatenz der vorgestellten Lösung bewertet, was dem Leser eine umfassende Beurteilung ermöglicht.

Software-Design

Auswahl des Protokolls: Die erste wichtige Entscheidung war die Wahl eines geeigneten Protokolls. Üblicherweise arbeitet WiFi in einem Verbindungsmodus, bei dem die Geräte eine ständige Verbindung zum Netzwerk aufrechterhalten, solange sie nicht absichtlich getrennt werden oder sich außerhalb der Reichweite befinden. Diese ständige Verbindung ermöglicht es den Geräten, jederzeit für die Kommunikation zur Verfügung zu stehen, ohne dass sie sich bei jeder Verwendung neu verbinden müssen. Die Aufrechterhaltung dieser Verbindung verbraucht jedoch viel Energie.

Um diese Aufgabe zu bewältigen, haben wir uns für eine schlankere Kommunikationsme-

thode entschieden. Softwareseitig haben wir dies erreicht, indem wir die Nachrichten in der Sicherungs-Schicht (Data Link Layer) übertragen. So können die empfangenden Geräte die Informationen leicht abrufen und weiterleiten. Außerdem haben wir eine vorbestimmte Empfangszeit implementiert, um die Stromaufnahme so weit wie möglich zu minimieren. Zu diesem Zweck erwies sich das von Espressif Systems entwickelte ESP-NOW-Protokoll als ideal. ESP-NOW ist ein drahtloses Kommunikationsprotokoll, das auf der Sicherungs-Schicht (Data Link Layer) basiert. Es vereinfacht die oberen fünf Schichten des OSI-Modells auf eine Schicht (**Bild 2**).

Die Geräteidentifizierung erfolgt über eine eindeutige Kennung (MAC-Adresse). Dadurch müssen die Daten nicht mehr nacheinander komplexe Schichten wie die Vermittlungsschicht (Network Layer), die Transportschicht (Transport Layer), die Sitzungsschicht (Session Layer), die Darstellungsschicht (Presentation Layer) und die Anwendungsschicht (Application Layer) durchlaufen. Außerdem entfällt die Notwendigkeit, auf jeder Schicht Header hinzuzufügen und zu entfernen, was die durch Netzüberlastung bedingte Verzögerung und die durch Paketverluste

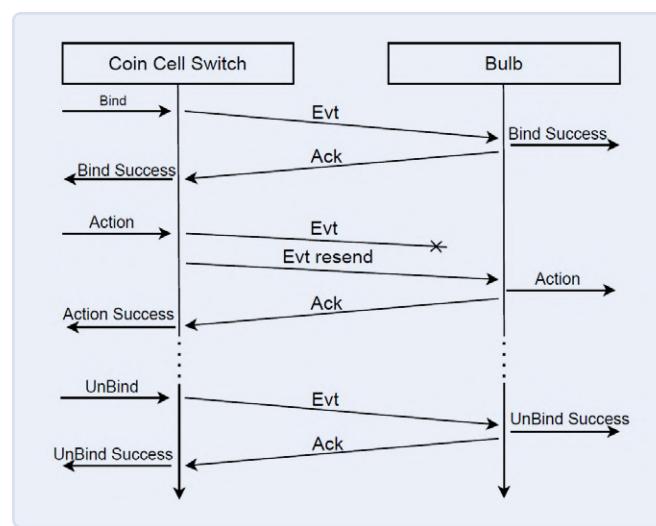


Bild 3. Die Implementierung des Bestätigungs-Mechanismus in der Anwendungs-Schicht

Tabelle 1:
Energiebedarf vor der Optimierung der Initialisierung.

Phase	Dauer (ms)	Mittlere Leistung (mW)	Energiebedarf (mJ)
Boot	364,7	58	21,16
Wi-Fi Init	115,2	69,8	8,03
Wi-Fi Start	56,6	303,9	17,2

verursachten Verzögerungen erheblich reduziert. Dies führt zudem zu einer höheren Reaktionsgeschwindigkeit. Darüber hinaus kann ESP-NOW mit Wi-Fi und Bluetooth LE parallel betrieben werden und es unterstützt mehrere SoC-Reihen von Espressif, die über WLAN-Funktionalität verfügen.

Workflow: Um eine zuverlässige Kommunikation zu gewährleisten, haben wir einen Bestätigungs-Mechanismus in der Anwendungsschicht implementiert (**Bild 3**). Der Kommunikationsprozess beginnt mit der Initialisierung der Kommunikation durch den Knopfzellen-Sender. Nach dem Empfang einer Nachricht antwortet das empfangende Gerät mit einem ACK, das den erfolgreichen Empfang der Nachricht anzeigen. Die Kommunikation ist damit abgeschlossen. Wird die Bestätigung nicht innerhalb einer bestimmten Zeitspanne empfangen, versucht der Schalter, die Nachricht erneut zu übermitteln, und zwar mehrmals. Um ein schlankes Protokoll aufrechtzuerhalten, werden alle Pakete unter Verwendung des ESP-NOW-Protokolls auf der Sicherungsschicht übertragen.

Die Kopplung und Trennung von Geräten wird durch eine bidirektionale MAC-Adressen-Überprüfung bewerkstelligt. Dadurch wird die Sicherheit und Einfachheit des Protokolls gewährleistet.

Kanalumschaltung: Die Kanäle für ESP-NOW im Empfangs- und Sendebetrieb folgen dem Kanal des aktuell verbundenen Access Point (AP) des Geräts. Der Knopfzellen-Sender kann nur dann mit dem kontrollierten Gerät kommunizieren, wenn sich beide auf demselben Kanal befinden. Wenn das kontrollierte Gerät bereits mit dem angegebenen WLAN-AP verbunden ist, folgt sein Betriebskanal den Änderungen im Kanal des AP. Der Knopfzellen-Sender muss den aktuellen Betriebskanal des kontrollierten Endgeräts ermitteln. In einer Umgebung, in der die Netzwerkbedingungen relativ stabil sind, wechselt der Kanal des APs nicht häufig. Also sendet das Gerät zuerst auf dem Kanal, auf dem es das letzte Mal erfolgreich Daten gesendet hat. Falls mehrere Sendeversuche fehlgeschlagen und festgestellt wird, dass die

Tabelle 2:
Energiebedarf nach der Optimierung der Initialisierung.

Phase	Dauer (ms)	Mittlere Leistung (mW)	Energiebedarf (mJ)
Boot	37,52	53,14	2,0
Wi-Fi Init	6,55	72,62	0,48
Wi-Fi Start	19,12	164,0	3,13

empfangende Seite zu einem neuen Kanal für den Betrieb gewechselt hat, werden alle verbleibenden Wi-Fi-Kanäle durchlaufen.

Strom-Modulation während der Paketübertragung: Bei der Übertragung von Wi-Fi-Paketen kann der kurzzeitige Strom 300 mA übersteigen, was weit über den maximalen Strom hinausgeht, den Knopfzellen liefern können. Um dieses Problem zu lösen, haben wir eine intermittierende Paketübertragungs-Strategie eingeführt. Nach jeder Paketübertragung wird ein festgelegtes Intervall eingeführt, in dem der Chip in einen leichten Sleep-Zustand übergeht. In diesem Zustand zieht der Chip nur einige zehn Mikroampere, und die Entladung der Knopfzelle dient hauptsächlich dazu, die Kondensatoren vor der nächsten Paketübertragung zu laden. Nehmen wir an, der durchschnittliche Strom während der Paketübertragung ist I_0 mit einer Dauer von t_0 und der durchschnittliche Ruhestrom ist I_1 mit einer Dauer von t_1 . Der durchschnittliche Gesamtstrom während der Paketübertragung kann dann folgendermaßen berechnet werden:

$$I = \frac{I_0 t_0 + I_1 t_1}{t_0 + t_1}$$

Durch sorgfältige Modulation des Timings von I_0 und I_1 können wir sicherstellen, dass der durchschnittliche Strom geringer als der sichere Ausgangstrom der Knopfzelle ist. Diese Strategie trägt zu einem effizienten und energiesparenden Betrieb des Knopfzellen-Senders bei.



Der ESP32-C2-Knopfzellen-Sender bietet eine bequeme und vielseitige Möglichkeit, intelligente Geräte zu steuern.

Optimierung des Initialisierungsprozesses:

Der Initialisierungsprozess eines Wi-Fi-Chips durchläuft mehrere Phasen, vom Einschalten bis zum Abschluss der Signalübertragung. Wir führten eine gründliche Analyse jeder Phase und ihrer jeweiligen Dauer durch. Normalerweise umfasst der Chip-Startvorgang die Prozesse Booten, Wi-Fi-Initialisierung und Wi-Fi-Start. Davon nimmt die Boot-Initialisierung die längste Zeit in Anspruch, und der Beginn des Wi-Fi-Betriebs verursacht die höchste kurzfristige Stromaufnahme. Um die Stromaufnahme zu reduzieren, haben wir die folgenden Optimierungen vorgenommen:

- Deaktivierung nicht notwendiger Protokollierung: Wir haben unnötige Protokollierungsausgaben abgeschaltet, um den Stromverbrauch während des Bootvorgangs zu minimieren.
- Flash-Überprüfung: Wir haben die Flash-Überprüfung deaktiviert, da sie für unsere Zwecke nicht notwendig war.
- Wi-Fi-Kalibrierungsinformationen: Um eine häufige Wi-Fi-Kalibrierung zu vermeiden, haben wir die Wi-Fi-Kalibrierungsinformationen im nichtflüchtigen Speicher (Non-Volatile Storage, NVS) untergebracht.

Werfen Sie nun einen Blick auf **Tabelle 1** und **Tabelle 2**: Allein durch diese Optimierungen konnten wir die Gesamtenergieaufnahme (Boot + Wi-Fi Init + Wi-Fi Start) während des Initialisierungsprozesses von 46,39 mJ auf 5,61 mJ senken. Außerdem verringerte sich die Initialisierungszeit von 536,5 ms auf 63,19 ms. Ausführlichere Informationen zur Konfiguration finden Sie in der *ESP-NOW coin_cell_demo* [1].

Schaltungsentwurf

Der ESP32-C2 benötigt eine Betriebsspannung von 3,3 V, die höher ist als die von der Knopfzelle gelieferte Spannung. Daher muss eine Boost-Schaltung die Spannung erhöhen. Die Stabilität der Stromversorgung hat einen direkten Einfluss auf die Paketübertragungsleistung und die Gesamtstabilität des Geräts. Eine gut durchdachte Spannungsregler-Schal-

tung kann die HF-Leistung und die Batterielebensdauer des Geräts verbessern. Beim Entwurf der Schaltung müssen sowohl die Produktionskosten als auch die erforderliche Leistungsfähigkeit berücksichtigt werden. Die Boost-Schaltung sollte sorgfältig entworfen werden, um eine effiziente Leistungsumwandlung mit minimalem Leistungsverlust zu gewährleisten. Natürlich soll sie den ESP32-C2 stabil und zuverlässig mit Strom versorgen, sodass er sowohl im aktiven als auch im Sleep-Modus optimal funktioniert. Das Schaltungsdesign muss außerdem Faktoren wie Stromaufnahme, Wärmeabgabe und Effizienz berücksichtigen, um das richtige Gleichgewicht zwischen Leistung und Stromaufnahme zu finden.

Das wichtigste Ziel des Schaltungsentwurfs ist es, eine robuste und kostengünstige Lösung zu finden, die den Strombedarf des ESP32-C2 abdeckt und gleichzeitig die Leistung und Batterielebensdauer des Produkts optimiert. Durch die Zusammenarbeit mit erfahrenen Hardware-Ingenieuren und die Verwendung geeigneter Bauteile und Techniken kann ein erfolgreiches Schaltungsdesign erreicht werden. Dieses erfüllt die spezifischen Anforderungen des batteriebetriebenen WLAN-Schalters.

Gesamtentwurf

Auswahl des Spannungswandlers: Die Knopfzelle ist eine Spannungsquelle, deren Innenwiderstand beim Entladen schnell ansteigt. Anfänglich beträgt der Innenwiderstand etwa $10\ \Omega$, aber er kann bis zu Hunderten von Ohm ansteigen, wenn sie sich dem Ende ihres Entladezyklus nähert. Der ESP32-C2 als Ausgabegerät benötigt eine Stromquelle, die einen Spitzen-Ausgangsstrom von 500 mA oder mehr liefern kann. Die Restwelligkeit der Stromversorgung kann die HF-Übertragungsleistung erheblich beeinträchtigen. Bei der Messung der Restwelligkeit ist es wichtig, diese unter normalen Paketübertragungsbedingungen zu testen. Die Restwelligkeit der Stromversorgung kann je nach Betriebsmodus variieren. Eine höhere Paketübertragungsleistung kann zu einer größeren Restwelligkeit führen. Um die Auswirkungen des hohen Innenwiderstands der Knopfzelle abzuschwächen, sollte die minimale Eingangsspannung des Aufwärtswandlers so niedrig wie möglich sein und gleichzeitig ein hoher Wirkungsgrad beibehalten werden.

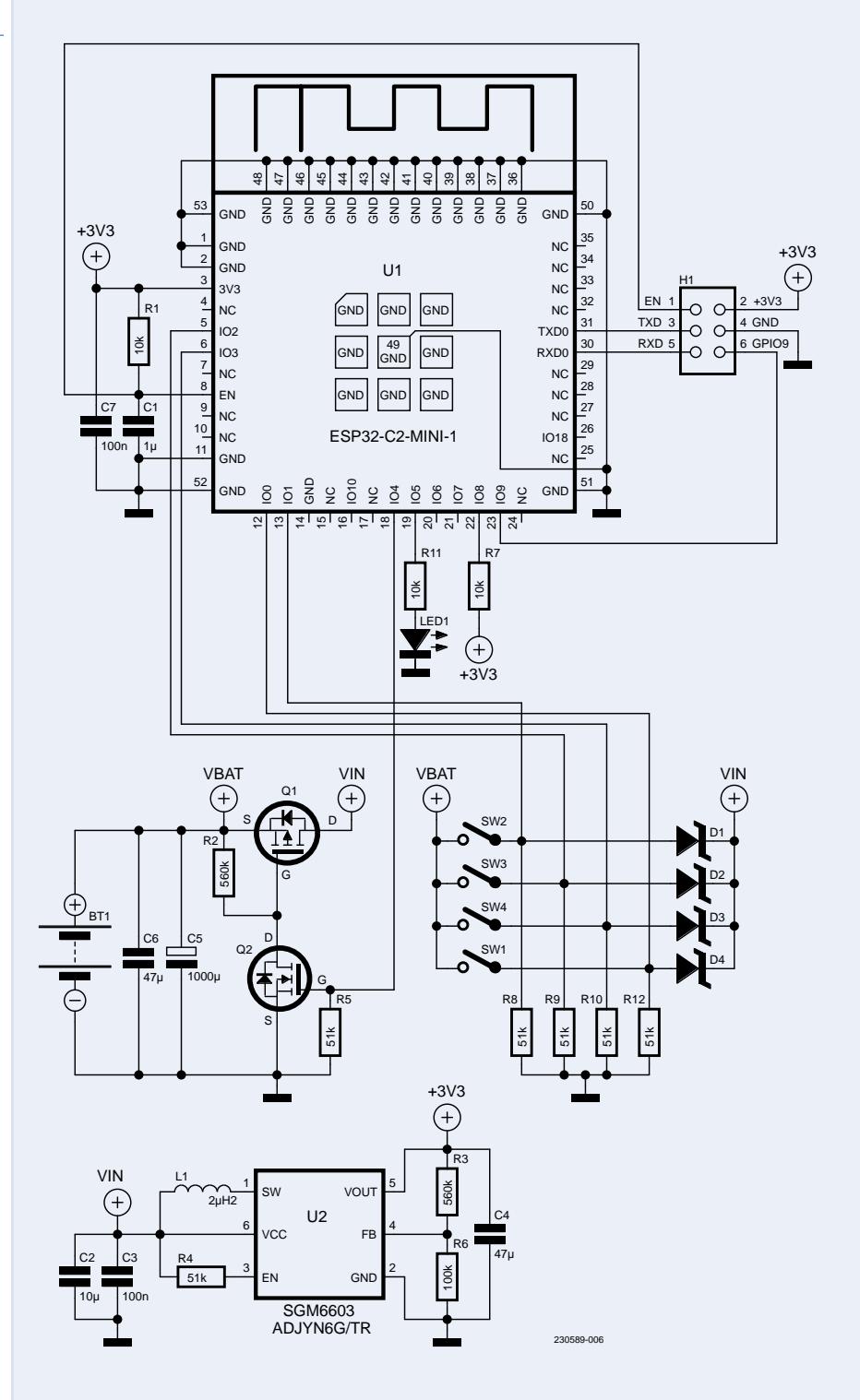


Bild 4. Der Schaltplan des Knopfzellen-Senders

Für dieses Projekt wurde der SGM6603 als Boost-Konverter ausgewählt, wie in Bild 4 zu sehen ist. Er bietet eine minimale Eingangsspannung von nur 0,9 V und einen maximalen Schaltstrom von 1,1 A. Damit ist er geeignet, die Spannung der Knopfzelle effizient zu erhöhen, um die Stromanforderungen des ESP32-C2 zu erfüllen.

Kondensator-Auswahl: Es gibt zwei Gruppen von Kondensatoren im Stromversorgungssystem: die Kondensatoren vor und nach dem Boost-Konverter. Die Kondensatoren nach dem Boost-Konverter sind in der Regel parallel zum Stromeingang des WLAN-Moduls geschaltet und haben die Aufgabe, die Ausgangsspannung zu stabilisieren und den



Bild 5. Oszilloskop-Screenshot von Spannungs- und Stromwerten während der Übertragung eines Standard-Datenpakets

Spannungsabfall während der Paketübertragung zu verringern. Größere Kondensatoren führen zu einer gleichmäßigeren Spannungsänderung in der Stromversorgung des Chips. Andererseits folgt die Spannung an diesen Kondensatoren der Stromversorgungsspannung des Chips. Wenn der WLAN-Chip mit Strom versorgt wird, werden die Kondensatoren aufgeladen, und wenn der Chip ausgeschaltet wird, ist diese Gruppe von Kondensatoren vollständig entladen. Daher kann die Verwendung übermäßig großer Kondensatoren sogar zu einer geringeren Systemeffizienz führen. Es ist wichtig, ein Gleichgewicht zwischen der Größe der Kondensatoren und der Systemeffizienz zu finden.

Die Kondensatoren vor dem Boost-Konverter sind parallel zur Batterie geschaltet. Ihre Hauptfunktion besteht darin, die momentane Stromaufnahme aus der Batterie zu reduzieren. Bei hohem Betriebsstrom fungieren die Kondensatoren als primäre Stromquelle, während bei niedrigem Betriebsstrom die Batterie die Hauptstromquelle ist und die Kondensatoren lädt. Wenn der Boost-Konverter nicht in Betrieb ist, wird der einzige Stromverbrauch in der Schaltung durch den Leckstrom des Kondensators verursacht. In Anbetracht des Bauvolumens und des Leckstroms sind Polymer- und Aluminium-Elektrolytkondensatoren die ideale Wahl. Ein 1000- μ F-Polymer-Elektrolytkondensator beispielsweise weist einen Leckstrom von etwa 1 μ A bei 3 V Spannung auf.

Gesteuerter Ein/Aus-Schalter: Bei Anwendungen, bei denen die Batterielebensdauer des Geräts in Jahren gemessen wird, wird

der Standby-Strom (Leckstrom) während des Ruhezustands zu einem kritischen Faktor, der die Gesamtlebensdauer der Batterie des Geräts beeinflusst. Um dies zu gewährleisten, enthält das hier vorgestellte Design einen gesteuerten Ein/Aus-Schalter, der aus zwei MOSFETs Q1 und Q2 besteht (siehe Schaltplan in Bild 4). Dieser Schalter ermöglicht es dem Chip, die Verbindung zur Batterie aktiv zu schließen oder zu öffnen und so das Stromversorgungsmodul und das HF-Modul effektiv von der Batterie zu trennen, wenn das Gerät nicht in Gebrauch ist. Um das Gerät einzuschalten, muss eine Taste gedrückt werden. Hierdurch wird Q1 aktiviert und der Chip mit Strom versorgt. Gleichzeitig wird auf dem Chip mittels ADC-Spannungsmessung ermittelt, welche Taste gedrückt wurde.

Der Einsatz dieses gesteuerten Ein/Aus-Schalters gewährleistet ein effizientes Energiemanagement, das unnötigen Energieverbrauch im Leerlauf verringert und die Batterielebensdauer verlängert. Durch die vollständige Abschaltung der Stromversorgung und der HF-Module bei Nichtgebrauch wird der Standby-Strom des Geräts minimiert, wodurch seine Langlebigkeit und Verwendbarkeit in verschiedenen Unterhaltungselektronik-Anwendungen optimiert wird. Zusätzlich zu den oben genannten Komponenten enthält die Schaltung auch das ESP32-C2-Minimal-System und eine LED-Anzeige.

Bewertung der Batterielelaufzeit

Nach der abschließenden Optimierung wurde ein typischer Paketübertragungsvorgang analysiert. Dabei wurden die eingangsseitige Spannung und der Strom des Aufwärtswand-

lers aufgezeichnet (Bild 5). Auf Grundlage der gemessenen Daten beträgt die gesamte Paketübertragungszeit für das Gerät 240 ms, und der durchschnittliche Strom während des Betriebs liegt bei 25,8 mA.

Um nun die Batterielebensdauer des Geräts unter Berücksichtigung der Unsicherheiten in Bezug auf die Effizienz des Spannungswandlers bei dynamischer Belastung genau zu beurteilen, wurde eine praktische Bewertung durchgeführt. Bei dieser Bewertung wurde die Gesamtanzahl der übertragenen Pakete des Geräts im angegebenen Arbeitsmodus getestet. Jeder Zyklus wurde von der Auslösung des Geräts (Tastendruck) bis zur erfolgreichen Übertragung eines Signals und dem Empfang einer Bestätigung gemessen, was die gesamte Betriebsdauer umfasst.

Bei diesen Tests wurde festgestellt, dass eine einzige CR2032-Knopfzelle etwa 65.000 Paketübertragungzyklen unterstützen kann. Außerdem wurde der Standby-Strom des Geräts mit nur 1 μ A gemessen. Geht man davon aus, dass das Gerät 10-mal pro Tag Pakete überträgt, beträgt die praktische Lebensdauer der CR2032-Knopfzelle etwa fünf Jahre.

Diese Messung zeigt die effiziente Energieverwaltung und den geringen Stromverbrauch des Geräts, wodurch es für den langfristigen Einsatz in Anwendungen der Unterhaltungselektronik geeignet ist. Mit dem optimierten Ein-/Aus-Schalter und dem fortschrittlichen Schaltungsdesign erreicht das Gerät eine beeindruckende Batterielelaufzeit, die einen zuverlässigen und langen Betrieb in verschiedenen IoT- und Smart-Home-Anwendungen gewährleistet.

Abschließende Betrachtungen

Der ESP32-C2 Knopfzellen-Sender bietet eine einfache und vielseitige Möglichkeit, intelligente Geräte zu steuern. Diese Lösung nutzt das flexible Energiemanagement und die Protokolle des ESP32-C2 und realisiert einen Knopfzellen-Sender, der auf dem Wi-Fi-Protokoll basiert und eine einfache Kommunikation mit anderen Geräten ermöglicht, die mit ESP-Chips ausgestattet sind.

Im Rahmen unserer Zukunftspläne wollen wir diese Technologie in die Matter-Standards integrieren, um eine flexible Steuerung mehrerer Geräte und die Zusammenarbeit mit konventionellen, netzgespeisten Geräten zu ermöglichen.

Wenn Sie das Thema interessiert, besuchen Sie bitte das GitHub-Repository von Espressif [2] für weitere Open-Source-Demos und Infor-

mationen zu ESP-IoT-Solution und ESP-NOW. Die hier vorgestellte Lösung eines Knopfzellen-Senders erhöht den Bedienkomfort und die Effizienz im Bereich des Smart Homes und des Internets der Dinge. Das optimierte Design und die effiziente Energieverwaltung sorgen für eine lange Batterielaufzeit und bieten den Nutzern eine zuverlässige und dauerhafte Steuerung intelligenter Geräte. ↗

Übersetzung von Holger Neumann -- 230589-02



Über die Autoren

Zhang Wei ist aktuell Senior Staff Application Engineer bei Espressif Systems. Als erfahrener Software-Ingenieur mit über zwei Jahrzehnten Erfahrung in den Bereichen eingebettete Systeme, drahtlose Netzwerke und IoT-Entwicklung findet er gerne einfache und saubere Lösungen für Probleme. Er hat einen Bachelor-Abschluss in Elektro- und Computertechnik und einen Master-Abschluss in Knowledge Engineering von der National University of Singapore und hat sein Fachwissen durch Tätigkeiten bei Technologieunternehmen wie STMicroelectronics, Greenwave Systems und dormakaba digital erweitert. Außerhalb der Arbeit spielt Zhang Wei Fußball und reist gern.



Li Junru ist AE-Ingenieur bei Espressif Shanghai. Eingebettete Systeme sind seine große Leidenschaft. Er will Enthusiasten dabei helfen, ihre Kreativität mit ESP32 zu entfesseln. Die inspirierende Botschaft von Li Junru lautet: „Lasst uns zusammenarbeiten und die Entwicklung eingebetteter Systeme gemeinsam spannend machen!“

Haben Sie Fragen oder Anmerkungen?

Wenn Sie technische Fragen oder Kommentare zu diesem Artikel haben, wenden Sie sich bitte an die Autoren (zhang.wei@espressif.com oder lijunru@espressif.com) oder an das Elektor-Redaktionsteam unter redaktion@elektor.de.



Passende Produkte

- **Espressif ESP32-Reihe**
www.elektor.de/espressif
- **Koen Vervloesem: Getting Started with ESPHome Buch,**
Paperback, englisch:
<https://www.elektor.de/19738>
E-Buch, PDF, englisch:
<https://www.elektor.de/getting-started-with-esphome-e-book>

WEBLINKS

- [1] ESP Now coin_cell_demo (GitHub): <https://tinyurl.com/espnnowcoincell>
- [2] Espressif GitHub-Repository: <https://github.com/orgs/espressif/repositories>



ESP-ADF

Espressif Audio Development Framework

Wenn Sie ein Gerät bauen, das Audio aufnehmen oder abspielen soll, ist das ESP-ADF (Audio Development Framework) das SDK, das Sie sich ansehen sollten. ESP-ADF bietet nicht nur grundlegende Audio-Pipelining-Unterstützung, sondern auch verschiedene Audio-Encoder, -Decoder, Audio-Container-Parser, Equalizer und Downmixer. Zusätzlich werden verschiedene High-Level-Protokolle wie DLNA, RTSP, RTCP und Bluetooth-A2DP-Wiedergabe unterstützt, zusammen mit den entsprechenden Beispielen. Es gibt eine Vielzahl von Entwicklungskits, die über Audio-Funktionen

verfügen. Schauen Sie sich die Boards der Reihe ESP32-S3-Korvo und ESP32-Lyra an, die Sie einfach für das Prototyping verwenden können.

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





Ein Meilenstein für das Smart Home

So schalten Sie das IoT-Potenzial von Smart Home frei

Von Kedar Sovani, Espressif

Verbraucher und Entwickler waren bislang oft frustriert über vernetzte Produkte. In den meisten Fällen handelt es sich um Insellösungen, für deren Konfiguration und Betrieb eigene Apps erforderlich sind, was weder ein einheitliches Nutzererlebnis noch eine vollständig interoperable Kommunikation zwischen den Geräten ermöglicht. Das Matter-Protokoll bietet eine sichere Methode, mit der vernetzte Geräte vom Endnutzer konfiguriert und bedient werden können. Seit der Einführung im Oktober 2022 sind bereits mehr als 1.000 Produkte von vielen verschiedenen Unternehmen Matter-zertifiziert.

Ende 2022 wurde der Smart-Home-Standard Matter eingeführt – ein Ergebnis jahrelanger gemeinsamer Entwicklung durch wichtige Akteure der Branche. In den letzten Monaten wurden zahlreiche Geräte auf den Markt gebracht, die mit diesem universellen, interoperablen Standard arbeiten. In diesem Artikel werfen wir einen Blick auf die Vorteile, die Matter bietet, wie weit es schon gediehen ist und wie die Zukunft aussehen wird.

Die Ära vor Matter

Als Nutzer und auch als Entwickler von vernetzten Produkten haben Sie vielleicht schon die Frustration erlebt, die mit solchen Produkten einherging. Diese waren oft durch Insellösungen

gekennzeichnet, für deren Konfiguration und Betrieb eigene Apps erforderlich waren, die weder ein einheitliches Benutzererlebnis noch eine vollständig interoperable Kommunikation mit anderen Geräten boten. Das einzige verbindende Merkmal war die integrierte Unterstützung der gängigen Sprachassistenten verschiedener Ökosysteme mit ihren jeweiligen proprietären Protokollen. Für die Entwickler/Hersteller bedeutete dies sehr hohe Gesamtkosten für die Implementierung eines guten Produktes, da die Entwickler für die Einhaltung von Vorschriften und Zertifizierungen verschiedener Organisationen sorgen mussten.

Matter kommt ins Spiel

Verschiedene Interessengruppen in der Smart-Home-Branche erkannten, dass diese Probleme den Wert und das Wachstum des Smart-Home-Marktes ausbremsen. Die daraus resultierenden branchenweiten gemeinsamen Bemühungen gipfelten in der Einführung des Matter-Standards Ende 2022.

Was genau hat Matter also zu bieten? Bei Matter handelt es sich um eine sichere Methode, mit der vernetzte Geräte vom Endbenutzer konfiguriert und bedient werden können. Seit der Einführung im Oktober 2022 wurden mehr als 1.000 Produkte für Matter zertifiziert, und die Connectivity Standards Alliance (CSA) kann auf mehr als 300 Unternehmen verweisen, die sich an den gemeinsamen Bemühungen zur Entwicklung und Einführung von Matter beteiligt haben. Das große Ausmaß der Bemühungen um die Einführung von Matter hat dazu geführt, dass zahlreiche vernetzte Geräte bereits von Hause aus Matter unterstützen.

Matter-fähige Geräte können über sogenannte Matter-Controller (etwa Smartphones, Lautsprecher oder Displays) konfiguriert und bedient werden. Mehrere Ökosysteme wie iOS, Android, Alexa und SmartThings haben die Unterstützung für Matter-Controller bereits in ihr Basisangebot integriert. Dies ermöglicht es Verbrauchern, Matter-fähige vernetzte Geräte zu konfigurieren und zu bedienen, ohne einen separaten Matter-Controller kaufen oder eine eigene Smartphone-App installieren zu müssen (**Bild 1**).



Sicherheit und Datenschutz

Bei vielen vernetzten Geräten, die wir heute nutzen, vermissen wir als Verbraucher Informationen über die Sicherheitspraktiken und -prinzipien, die bei der Entwicklung und dem anschließenden Management des vernetzten Produkts angewandt werden. Sicherheit und Datenschutz bildeten das Rückgrat bei der Entwicklung der Matter-Spezifikation. Da eine große Anzahl von Organisationen (mit hohem Marktvolumen und jahrelanger Erfahrung in der Smart-Home-Branche) an diesen Spezifikationen mitgearbeitet hat, wurde nichts unversucht gelassen, um den Anwendern diesbezüglich das Optimum zu bieten. Benutzer von Matter-Produkten können sich darauf verlassen, dass jeder Aspekt des Produktverhaltens, von der Erstkonfiguration bis hin zum späteren Betrieb und Management, unter Beachtung der höchsten Sicherheitsstandards und -praktiken entwickelt wurde.

Lokale Vernetzung

Die Architektur von Matter unterscheidet sich in einem Punkt grundlegend von der Architektur früherer vernetzter Geräte. Früher wurde jedes vernetzte Gerät in der Regel über das lokale Netzwerk von einem Gerät, zum Beispiel einem Smartphone gesteuert, das sich im selben Netzwerk befand. In allen anderen Szenarien waren die Geräte mit einer Cloud-Plattform verbunden, und eine App, die auf einem Smartphone lief, das sich nicht im selben Netzwerk befand, kommunizierte mit diesen Geräten über die Cloud. Die Integration von Sprachassistenten erfolgte ebenfalls über eine Cloud-zu-Cloud-Kommunikation, wie in **Bild 2** zu sehen ist.

Matter ist ein auf das lokale Netzwerk beschränktes Protokoll. Es ermöglicht die Erstkonfiguration, das Ausprobieren und den Betrieb von angeschlossenen Geräten im lokalen Netzwerk selbst. Die vernetzten Geräte selbst müssen nicht mit einem Cloud-Dienst kommunizieren, wie es bei früheren Geräten der Fall war. Die Integration von Sprachassistenten funktioniert daher direkt, indem die entsprechenden Matter-Befehle über das lokale Netzwerk gesendet werden (**Bild 3**). Es bleibt also den Matter-Controllern überlassen, wie sie die Fernsteuerung dieser Geräte realisieren. Dadurch kann der Endanwender frei entscheiden, welchen Matter-Geräten und -Ökosystemen er in Bezug auf Datenschutz und Sicherheit sein Vertrauen schenkt.

Datenübertragung

Matter bietet seine Funktionen über die Transportprotokolle WLAN/Wi-Fi und Thread (802.15.4) an, die im Smart Home allgegenwärtig sind. Jedes dieser Transportprotokolle hat seine eigenen Vorteile. Da WLAN-Netze allgegenwärtig sind, können entsprechende Matter-Geräte problemlos in bestehende Netzwerke integriert werden.

Geräte auf Basis von Thread (802.15.4) eignen sich besser für Anwendungen mit geringem Stromverbrauch wie Sensoren, bei denen nur sehr kleine Datenmengen übertragen werden müssen. Thread-basierte Matter-Geräte benötigen einen Thread-Border-Router, um Teil des Netzwerks zu sein. Die meisten der vorhandenen intelligenten Lautsprecher und ähnlichen Geräte unterstützen Thread beziehungsweise erhalten Software-Updates, um die Border-Router-Funktion zu unterstützen. Das macht die Integration von Matter-Geräten in das Heimnetzwerk sehr viel einfacher.

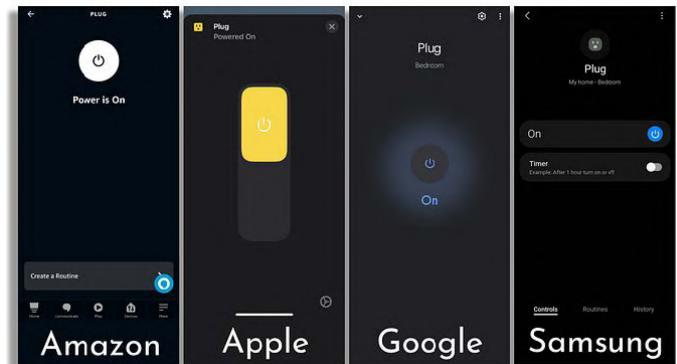


Bild 1. Eine Matter-fähige Steckdose in Apps verschiedener Ökosysteme

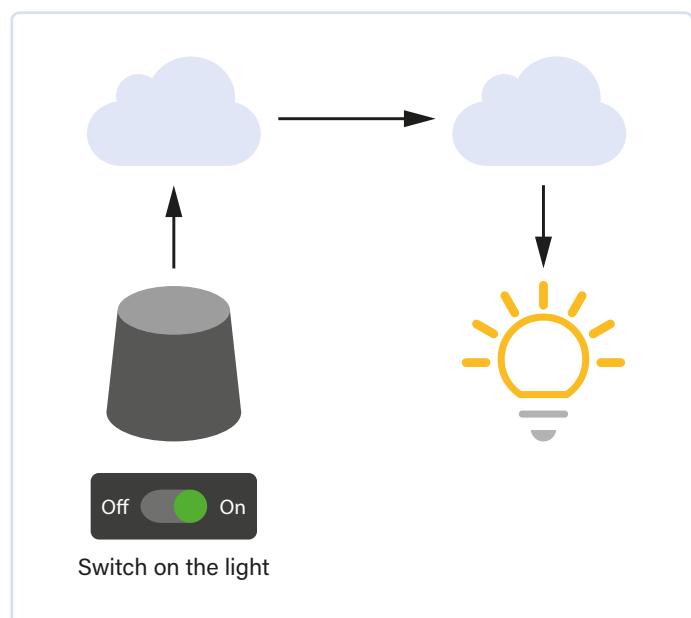


Bild 2. Kommunikationsbeispiel (ohne Matter)

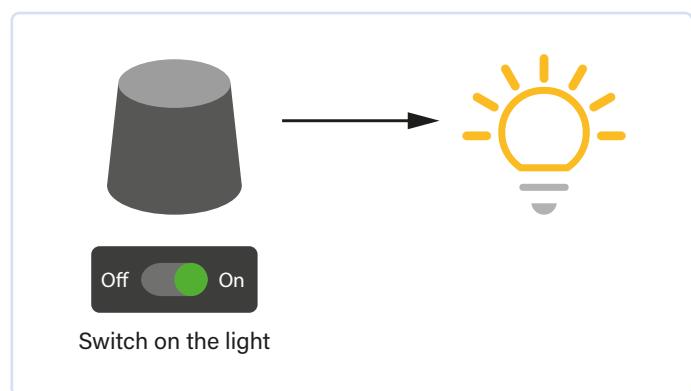


Bild 3. Kommunikationsbeispiel (mit Matter)

Inbetriebnahme

Alle vernetzten Geräte müssen zunächst über einen Bereitstellungsmechanismus in das Netzwerk des Benutzers eingebunden werden. Dies ist ein Prozess, der oft mit Fehlern, Fehlkonfigurationen und daraus resultierender Frustration des Benutzers einhergeht. Matter-Geräte werden über einen Prozess namens „Matter Commissioning“ in Betrieb genommen. Da Matter in den meisten Ökosystemen sofort unterstützt wird, ist dieser Prozess robust und ausfeinert und bietet jedem Benutzer einen reibungslosen Ablauf. Unter der Haube stellen die meisten Matter-Geräte diese Commissioning-Funktionalität über Bluetooth Low Energy (BLE) zur Verfügung. Matter-Controller übertragen die Anmeldedaten für das Zielnetzwerk (WLAN oder Thread) über eine sichere BLE-Verbindung.

Geräte können die Inbetriebnahme von Matter auch über Ethernet oder WLAN anbieten, wenn sie bereits ins Netzwerk des Benutzers eingebunden sind. Dieser Mechanismus ist entscheidend dafür, dass auch bestehende Geräte aufgerüstet werden können, um ihren Endbenutzern Matter-Unterstützung zu bieten.

Authentifizierung

Die Geräteauthentifizierung (Device Attestation) ist eine wichtige Sicherheitsfunktion von Matter. Bei diesem Prozess, der während der Inbetriebnahme des Matter-Geräts stattfindet, stellt der Matter-Controller die Authentizität des Geräts fest. Jedes Matter-Gerät wird mit einem einzigartigen Satz von herstellerspezifischen Sicherheitszertifikaten programmiert. Die Geräteauthentifizierung stellt sicher, dass das Gerät wirklich von dem Hersteller stammt, für den es sich ausgibt. Im Fall einer Diskrepanz wird der Endbenutzer über das Problem bei der Geräteauthentifizierung gewarnt. Dies erschwert die Herstellung und den Einsatz gefälschter Matter-Produkte und schützt so die Benutzer und ihre Daten.

Kommunikation zwischen Geräten

Vor der Einführung von Matter wurde die Kommunikation zwischen Geräten größtenteils über die Cloud oder über Sprachassistenten abgewickelt. Dies erforderte, dass alle Geräte mit derselben Cloud kommunizierten oder Cloud-zu-Cloud-Integrationen bereitstellten, um Ökosysteme mit Mehrwert zu ermöglichen, die automatisch das Richtige taten.

Ein großer Vorteil von Matter-fähigen Geräten ist, dass die Kommunikation zwischen den Geräten vollständig innerhalb des lokalen Netzwerks stattfindet, was heißt, dass keine Cloud-Kommunikation erforderlich ist. Ein Matter-Gerät eines Herstellers kann so eingerichtet werden, dass es mit einem Matter-Gerät eines anderen Herstellers kommuniziert. So können Sie beispielsweise Automatisierungen einrichten, um eine Gruppe von Lampen (oder eine Klimaanlage) anhand des Zustands eines physischen Schalters oder eines Sensors zu schalten. Das funktioniert sogar dann, wenn die Geräte unterschiedliche Übertragungswege nutzen (etwa eines das WLAN und das andere Thread). Die Möglichkeit, diese Automatisierungen (in der Matter-Sprache „Bindings“ genannt) in jedem von Matter unterstützten Ökosystem und von jeder App aus einzustellen, macht es noch einfacher, die Vorteile des Smart Home wirklich zu nutzen (**Bild 4**).

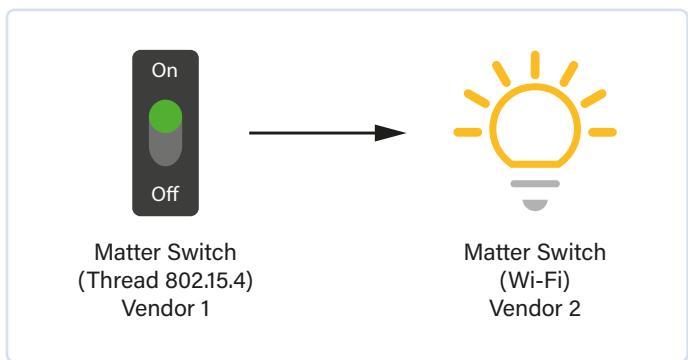


Bild 4. Matter-Binding: Kommunikation zwischen Geräten

Koexistenz der Ökosysteme

Ein weiteres Problem, mit dem die meisten Haushalte konfrontiert sind, ist die Heterogenität der Ökosysteme, die von den Bewohnern verwendet werden. Die meisten Benutzer haben ihre eigenen Vorlieben für Ökosysteme wie Android, iOS, Alexa, GVA, Smart-Things oder Home Assistant. Dies führt zu einer fragmentierten Benutzererfahrung, bei der einige vernetzte Geräte nur Teil eines einzigen Ökosystems sein können oder nur zweitklassige Funktionen für andere Ökosysteme bereitstellen können. Im Gegensatz dazu kann ein Matter-Gerät Teil mehrerer Ökosysteme gleichzeitig sein. Sie können ein Matter-Gerät zu bis zu fünf verschiedenen Ökosystemen hinzufügen. Alle Benutzer eines Ökosystems können gleichzeitig die gleichen Vorteile nutzen, die Matter-Geräte bieten. Selbst die Benachrichtigung über eine Änderung in einem Ökosystem wird von einem anderen Ökosystem empfangen.

Drahtlose Firmware-Updates

Firmware-Updates können drahtlos (Over the Air, OTA) aufgespielt werden und ermöglichen es nicht nur, ein Produkt mit den neuesten Sicherheits- oder Funktionskorrekturen zu versehen, sondern im Laufe der Zeit auch neue Funktionen zu aktivieren. Die Matter-Spezifikation sieht ein verteiltes Register (Ledger) vor, über das die Hersteller OTA-Firmware-Upgrade-Images für ihre Produkte hochladen und pflegen können. Der Ledger ist für alle Matter-Controller zugänglich, die die erforderlichen Firmware-Updates herunterladen und mit Zustimmung des Benutzers auf den Matter-Geräten installieren können.

Zertifizierung

Matter hat ein strenges Verfahren zur Gerätezertifizierung. Das Matter-Logo auf einem vernetzten Gerät zeigt an, dass das Gerät alle Tests durchlaufen und bestanden hat, die für eine Matter-Zertifizierung erforderlich sind. Die Matter-Zertifizierung garantiert, dass das Gerät mit anderen Matter-Geräten und -Controllern interoperabel ist und die von den Verbrauchern gewünschten Mindestanforderungen an Robustheit und Funktionalität erfüllt. Nutzer von Matter-Produkten müssen sich bei diesen grundlegenden Anforderungen nicht mehr auf das Wort des Herstellers verlassen. Alle Matter-Hersteller müssen sich an denselben Standard für Robustheit und Interoperabilität halten.



Neuerungen durch Matter 1.2

Vor kurzem wurde die Version 1.2 von Matter veröffentlicht. Die erste Version von Matter begann mit der Unterstützung für die gängigsten Geräte wie Lampen, Steckdosen, Stecker, Jalousien und Thermostate. Der neueste Matter-1.2-Standard unterstützt nun auch Haushaltsgeräte (Weiße Ware) wie Waschmaschinen, Kühlschränke, Klimaanlagen, Geschirrspüler und Roboterstaubsauger. Unterstützung für Rauch- und CO-Melder, Luftqualitäts-sensoren, Luftreiniger und Ventilatoren ist ebenfalls enthalten. Diese umfasst zusätzliche spezifische Befehle für diese Geräte, die über die typische Ein/Aus-Steuerung hinausgehen.

Die Kernspezifikation von Matter enthält auch Upgrades für eine flexiblere Zusammensetzung von Steuerungslösungen aus ihren Teilkomponenten. Dies ermöglicht eine genauere Modellierung einer Vielzahl von Lösungen. Die neu hinzugefügte Unterstützung für semantische Tags ermöglicht eine interoperable Beschreibung für die Standorte oder die semantischen Funktionen eines Geräts.

Und was ist mit den Entwicklern?

Der Matter-Standard steht öffentlich zum Download zur Verfügung (nachdem Sie ein Formular ausgefüllt haben). Das Matter-SDK (in C++) wird auf GitHub [1] gehostet, wobei die gesamte Entwicklung auf GitHub selbst stattfindet. Das SDK implementiert sowohl Funktionen auf der Firmware-Seite (Gerät) als auch auf der Controller-Seite. Experimentelle Implementierungen für JavaScript [2] und Rust [3] sind ebenfalls in Arbeit. Dadurch können Entwickler leicht auf die Spezifikationen zugreifen und Verbesserungen in das Repository einbringen, die für sie von Interesse sind.

Lösungen wie ESP-Launchpad [4] und ESP-ZeroCode [5] ermöglichen es Entwicklern, Matter auszuprobieren und ein Gefühl für die Benutzerfreundlichkeit zu bekommen, indem sie die Firmware auf eine Vielzahl von Hardware Development Kits flashen.

So können Entwickler ihre Vision eines Matter-Geräts auf einfache Weise realisieren und testen. Sobald die Geräte-Firmware und -Hardware verfügbar ist, können Entwickler diese in allen Matter-Ökosystemen testen, ohne bereits eine freigegebene Firmware oder Zertifikate zu benötigen. Das macht die Produktentwicklung mit Matter sehr viel einfacher.

Für Entwickler von Smartphone-Apps enthalten die neusten Versionen von iOS [6] und Android [7] APIs, mit denen diese Apps auf die Matter-APIs zugreifen können. App-Entwickler können dies nutzen, um umfangreichere Funktionalitäten als die vom Telefon-Betriebssystem mitgelieferten anzubieten.

Ich hoffe, dass diese Informationen Sie als Entwickler motivieren, sich mit Matter zu beschäftigen und mit der Entwicklung eigener Matter-Funktionen für Ihre Geräte zu beginnen. ↗

Übersetzt von Jörg Starkmuth -- 230617-02

Haben Sie Fragen oder Anmerkungen?

Wenn Sie technische Fragen oder Anmerkungen zu diesem Artikel haben, können Sie den Autor per E-Mail unter kedar.sovani@espresif.com oder die Elektor-Redaktion unter redaktion@elektor.de erreichen.



Über den Autor

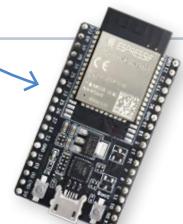
Kedar Sovani hat über 21 Jahre Erfahrung in den Bereichen Systemsoftware, Sicherheit und IoT. Seit sechs Jahren arbeitet er bei Espressif. Er ist Senior Director of Engineering mit dem Schwerpunkt IoT-Ökosysteme. Seine Arbeit hilft bei der Konzeption und dem Aufbau von Plattformen und Lösungen, die Entwickler dabei unterstützen, robustere und sicherere IoT-Produkte schneller zu realisieren. Als praktischer Entwickler beschäftigt er sich derzeit mit Matter und Rust.



Passende Produkte

› **ESP32-DevKitC-32E**
www.elektor.de/20518

› **ESP32-C3-DevKitM-1**
www.elektor.de/20324



WEBLINKS

- [1] Matter C++-SDK auf GitHub: <https://github.com/project-chip/connectedhomeip>
- [2] Implementation für JavaScript: <https://github.com/project-chip/matter.js>
- [3] Implementation für Rust: <https://github.com/project-chip/rs-matter>
- [4] ESP-Launchpad: <https://espressif.github.io/esp-launchpad>
- [5] ESP-ZeroCode: <https://zerocode.espressif.com>
- [6] Home-Mobile-SDK für iOS: <https://developer.apple.com/documentation/matter>
- [7] Home-Mobile-SDK für Android: <https://developers.home.google.com/matter/apis/home>

Holen Sie sich die neue ESPRESSIF Hardware!

Es gibt nichts, was uns mehr begeistert, als neue Hardware in die Hände zu bekommen, und so war diese Zusammenarbeit mit Espressif ein wahrer Genuss! Möchten Sie sich davon überzeugen? Elektor hat das Lager erweitert, um alle Produkte, die in dieser Ausgabe vorgestellt werden, unterbringen zu können!



ESP32-S3-Box-3

Die ESP32-S3-BOX-3 ist ein vollständig quelloffenes IoT-Entwicklungskit, das auf dem leistungsstarken ESP32-S3 AI SoC basiert und das Feld der traditionellen Entwicklungsbrettern revolutionieren soll. Die ESP32-S3-BOX-3 ist mit einer Vielzahl von Add-ons ausgestattet, die es Entwicklern ermöglichen, die Funktionalität des Kits einfach anzupassen und zu erweitern.

www.elektor.de/20627

ESP32-S3-Eye

Der ESP32-S3-EYE ist ein kleines KI-Entwicklungsboard. Es basiert auf dem ESP32-S3 SoC und ESP-WHO, dem KI-Entwicklungsframework von Espressif. Es verfügt über eine 2-Megapixel-Kamera, ein LCD-Display und ein Mikrofon, die für die Bilderkennung und Audioverarbeitung verwendet werden.

www.elektor.de/20626



ESP32-S3-DevKitC-1

Das ESP32-S3-DevKitC-1 ist ein Entwicklungsboard der Einstiegsklasse, das mit ESP32-S3-WROOM-1, ESP32-S3-WROOM-1U oder ESP32-S3-WROOM-2 plus einem universellen Wi-Fi + Bluetooth Low Energy MCU-Modul ausgestattet ist, das vollständige Wi-Fi- und Bluetooth LE-Funktionen integriert.

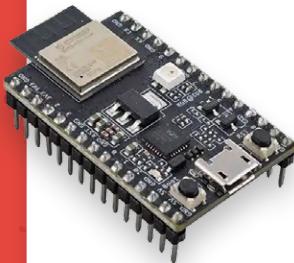
www.elektor.de/20697



ESP32-C3-DevKitM-1

ESP32-C3-DevKitM-1 ist ein Entwicklungsboard der Einstiegsklasse, das auf dem ESP32-C3-MINI-1 basiert, einem Modul, das aufgrund seiner geringen Größe benannt wurde. Dieses Board verfügt über vollständige Wi-Fi- und Bluetooth LE-Funktionen. Die meisten E/A-Pins des ESP32-C3-MINI-1-Moduls sind auf die Stiftleisten auf beiden Seiten des Boards aufgeteilt, um die Anbindung zu erleichtern. Entwickler können Peripheriegeräte entweder mit Jumper-Drähten anschließen oder ESP32-C3-DevKitM-1 auf einem Breadboard montieren.

www.elektor.de/20324



ESP32-Cam-CH340

Das ESP32-Cam-CH340-Entwicklungsboard kann in verschiedenen Internet-of-Things-Anwendungen eingesetzt werden, z. B. in intelligenten Heimgeräten, drahtloser Industrie-steuerung, drahtloser Überwachung, drahtloser QR-Identifikation, drahtlosen Ortungssystemen und anderen Internet-of-Things-Anwendungen.

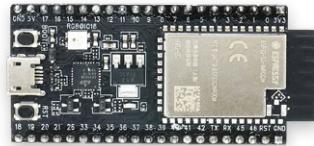
www.elektor.de/19333



Elektor Cloc 2.0 Kit

Cloc ist ein einfach zu bauender ESP32-basierter Wecker, der sich mit einem Zeitserver verbindet und Radio und TV steuert. Er hat ein doppeltes 7-Segment-Retro-Display mit variabler Helligkeit. Ein Display zeigt die aktuelle Zeit an, das andere die Alarmzeit.

www.elektor.de/20438



ESP32-S2-Saola-1M

ESP32-S2-Saola-1M ist ein kleines ESP32-S2-basiertes Entwicklungsboard. Die meisten E/A-Pins sind auf die Stifteleisten auf beiden Seiten aufgeteilt, um den Anschluss zu erleichtern. Entwickler können Peripheriegeräte entweder mit Jumperdrähten anschließen oder ESP32-S2-Saola-1M auf einem Breadboard montieren. ESP32-S2-Saola-1M ist mit dem ESP32-S2-WROOM-Modul ausgestattet, einem leistungsstarken, generischen Wi-Fi-MCU-Modul, das über eine Vielzahl von Peripheriegeräten verfügt.

www.elektor.de/19694



Arduino Nano ESP32

Das Arduino Nano ESP32 ist ein Nano-Formfaktor-Board, das auf dem ESP32-S3 (eingebettet im NORA-W106-10B von u-blox) basiert. Es ist das erste Arduino-Board, das vollständig auf einem ESP32 basiert. Es bietet Wi-Fi, Bluetooth LE, Debugging über natives USB in der Arduino-IDE sowie einen geringen Stromverbrauch.

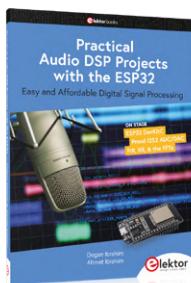
www.elektor.de/20562



MakePython ESP32 Development Kit

Dieses MakePython ESP32 Kit ist ein unverzichtbares Entwicklungskit für die ESP32 MicroPython-Programmierung. Neben dem MakePython ESP32-Entwicklungsboard enthält dieses Kit die grundlegenden elektronischen Komponenten und Module, die Sie benötigen, um mit dem Programmieren zu beginnen. Mit den 46 Projekten im beiliegenden Buch können Sie einfache Elektronik-Projekte mit MicroPython auf ESP32-Basis erstellen und eigene IoT-Projekte einrichten.

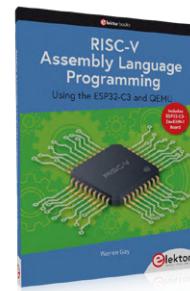
www.elektor.de/20452



Practical Audio DSP Projects with the ESP32

Das Ziel dieses Buches ist es, die Grundprinzipien der digitalen Signalverarbeitung (DSP) zu vermitteln und sie aus praktischer Sicht mit einem Minimum an Mathematik vorzustellen. Es werden nur die Grundlagen der Theorie zeitdiskreter Systeme vermittelt, die ausreichen, um DSP-Anwendungen in Echtzeit zu implementieren. Die praktischen Implementierungen werden in Echtzeit unter Verwendung des sehr beliebten ESP32 DevKitC Mikrocontroller-Entwicklungsboards beschrieben.

www.elektor.de/20558



RISC-V Assembly Language Programming using ESP32-C3 and QEMU (+ GRATIS ESP32 RISC-V Board)

Die Verfügbarkeit des Espressif ESP32-C3 Chips bietet eine Möglichkeit, praktische Erfahrungen mit RISC-V zu sammeln. Der quelloffene QEMU-Emulator bietet eine 64-Bit-Erfahrung mit RISC-V unter Linux. Dies sind nur zwei Möglichkeiten für Studenten und Enthusiasten gleichermaßen, RISC-V in diesem Buch zu erkunden. Die Projekte in diesem Buch sind auf das Nötigste reduziert, um die Assembler-Konzepte klar und einfach zu halten.

www.elektor.de/20296

MicroPython für Mikrocontroller

Leistungsfähige Controller wie der ESP32 der Firma Espressif Systems bieten eine hervorragende Performance sowie Wi-Fi- und Bluetooth-Funktionalität zu einem günstigen Preis. Mit diesen Eigenschaften wurde die Maker-Szene im Sturm erobert. Im Vergleich zu anderen Controllern weist der ESP32 einen deutlich größeren Flash und SRAM-Speicher, sowie eine wesentlich höhere CPU-Geschwindigkeit auf. Aufgrund dieser Leistungsmerkmale eignet sich der Chip nicht nur für klassische C-Anwendungen, sondern insbesondere auch für die Programmierung mit MicroPython. Dieses Buch führt in die Anwendung der modernen Ein-Chip-Systeme ein.

www.elektor.de/19412





Wohin steuert Smart Home IoT?

Von Amey Inamdar, Espressif

Während sich die Konnektivitätstechnologie weiterentwickelt hat und zugänglicher, sicherer und kostengünstiger geworden ist, steht das Smart Home für die Vision, die schon im letzten Jahrzehnt entworfen wurde, gerade erst am Anfang. Es gibt eine Vielzahl von Geräten, darunter intelligente Thermostate, Sicherheitssysteme, intelligente Geräte, intelligente Beleuchtung und Sprachassistenten. Was die Akzeptanz angeht, hat die Technologie noch einen langen Weg vor sich. In letzter Zeit gab es zwei wesentliche Fortschritte, die das Potenzial haben, die Akzeptanz zu beschleunigen, und die wir in diesem Artikel behandeln werden.

Stromverbrauch, Kosten, einfache Entwicklung und Konnektivitätsoptionen sind wichtige Faktoren bei der Entwicklung von Smart-Home-Geräten. Wenn wir uns umsehen, sind viele dieser Probleme bereits weitestgehend gelöst. Die Frage ist also, was eine breite Akzeptanz dieser Geräte behindert. Unter den zahlreichen Antworten auf diese Frage ist die wahrscheinlich wichtigste der wahrge nommene Wert von Smart-Home-Geräten - welche Anwendungsfälle sie erschließen. Sind diese Geräte intelligent, oder bieten sie nur die Möglichkeit der Fernsteuerung? Ein weiterer wichtiger Grund könnten Datenschutz- und Sicherheitsbedenken sein. Die Sprache ist zu einer natürlichen Schnittstelle für die Interaktion im intelligenten Haus geworden, aber sie hat den Preis eines wahrgenommenen Verlusts der Privatsphäre.

Hier bieten die beiden wichtigsten Fortschritte, die in letzter Zeit erzielt wurden, einen Hoffnungsschimmer. Wir sehen den großen Wandel, den generative KI und große Sprachmodelle in verschiedenen Bereichen bewirken. Große Sprachmodelle (Large Language Models, LLMs) haben das Potenzial, die Effizienz und Autonomie von Smart-Home-Geräten zu verbessern, indem sie die Entscheidungsfindung dezentralisieren und „in die Edge“ verlagern. Diese LLMs haben auch das Potenzial, zur Verbesserung des Datenschut-

zes beizutragen. Betrachtet man beispielsweise die derzeitige Sprachschnittstelle, so wird diese entweder durch Cloud-basierte Schlussfolgerungen oder eine durch feste Befehlen realisiert, die sich der Nutzer genau merken muss, damit er damit umgehen kann. Mit LLMs, die Offline-Modelle wie Whisper.ai bereitstellen, um verschiedene Sprachen zu verstehen und lokal zu inferieren, haben sie das Potenzial, natürliche Sprachschnittstellen für Smart Homes bereitzustellen, die vollständig offline funktionieren. Es gibt bereits einige Open-Source-Projekte, die in diese Richtung gehen.

Diese Fortschritte im Bereich der generativen KI werden durch winzige Mikrocontroller ergänzt, die immer besser in der Lage sind, ML-Modelle in den Bereichen der Edge auszuführen. Es ist heute nicht ungewöhnlich, dass Mikrocontroller über stromsparende KI-Beschleunigungsmodulen verfügen, die die Ausführung von ML-Modellen beschleunigen können. Dies ermöglicht Sensoren mit geringer Stromaufnahme, die nicht nur die Daten erfassen, sondern auch verarbeiten können, um aus den Daten eine Bedeutung zu extrahieren.

Doch allein die Verfügbarkeit dieser Intelligenz für IoT-Geräte im intelligenten Zuhause hilft nicht, wenn nicht alle Geräte die gleiche Sprache sprechen. Hier ist eine Standardisierung dringend erforderlich. Der Mangel an Standardisierung ist derzeit eines der Hauptprobleme bei der Masseneinführung von IoT-Geräten für das intelligente Zuhause. Die Tatsache, dass Geräte verschiedener Hersteller nicht miteinander kommunizieren können, schränkt die Anwendungsmöglichkeiten für den Verbraucher stark ein und macht es schwierig, Smart-Home-Geräte zu konfigurieren und zu betreiben. An dieser Stelle werden die jüngsten Standardisierungsbe mühungen wichtig. Ein Protokoll wie Matter kann, wenn es erfolgreich ist, den Geräten die Möglichkeit geben, dieselbe Sprache zu sprechen, und so ein verbraucherorientiertes



Smart Home mit besseren Anwendungsfällen und Benutzererfahrungen ermöglichen. Eine weitere indirekte Auswirkung der Standardisierung besteht darin, dass sie Entwicklern die Möglichkeit gibt, einen Mehrwert auf einer höheren Ebene zu schaffen, als sie die Gerätehersteller bieten.

KI und Standardisierung sind also beide wichtig, um die Hauptprobleme in Bezug auf den Wert und die Privatsphäre von Smart-Home-Geräten zu lösen. Dies muss durch Innovationen in den Bereichen Konnektivität, Sensorik und Datenverarbeitung unterstützt werden, um die Verbreitung kostengünstiger und sicherer Smart-Home-Geräte zu ermöglichen. Wir haben schon viele solcher Fortschritte erlebt. So ermöglicht der Standard Wi-Fi 6 die Entwicklung batteriebetriebener Geräte, ohne die Bandbreite zu beeinträchtigen. Das Aufkommen von mmWave- und UWB-Sensoren ermöglicht eine viel bessere Erkennung von Anwesenheit, ohne dass überall Kameras installiert werden müssen, die die Privatsphäre verletzen. Auch die Kennzeichnung der Cybersicherheit gewinnt an Bedeutung, und verschiedene Regionen und Länder arbeiten an einer klaren Methode zur Kennzeichnung von Smart-Home-Geräten mit einer Sicherheitseinstufung, die den Verbrauchern helfen kann, eine fundierte Entscheidung zu treffen.

Es gibt kein Patentrezept für die Verwirklichung eines flächendeckenden Smart Homes. Aber höchstwahrscheinlich bewegen wir uns jetzt in eine „bessere“ Richtung als zuvor. Es ist sicherlich eine interessante Zeit und ein interessantes Gebiet, um zu beobachten, wie die Dinge sich entwickeln und wie wir unseren Teil dazu beitragen, die Welt zu verbessern. 

230655-02

MACNICA

ATD EUROPE

Your official authorized distributor
in Europe for Espressif Systems

 **ESPRESSIF**



empowered
connectivity
everywhere

Macnica ATD Europe

+49 (0)89 899 143-11

sales.mae@macnica.com

www.macnica-atd-europe.com



High Performance MCU

With RISC-V Dual-Core Upto 400MHz

AI
Acceleration

High-Speed
MEMORY

Powerful
IMAGE & VOICE
Processing Capabilities

HMI Capabilities

- MIPI-CSI with ISP
- MIPI-DSI - 1080P
- Capacitive Touch
- H.264 Encoding - 1080P@30fps
- Pixel Processing Accelerator

Best-in-Class Security

- Cryptographic Accelerators
- Secure Boot, Flash Encryption
- Private Key protection
- Access Controls



Connectivity

- USB2.0 High Speed
- Ethernet
- SPI
- SDIO3.0
- UART
-
- I2C, I2S



IP Camera



Touch Panel



Video Door bell



Robotic Control



Industrial Robot



Learn More About
ESP SoCs

www.espressif.com