

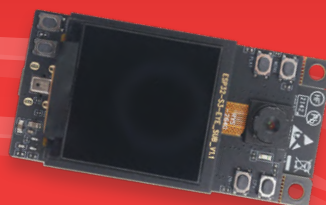
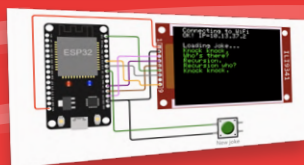
SPECIALE EDITIE

140
Pagina's

Gastredactie:



ESPRESSIF



Prototypes maken met Espressif ICs



ADF, IDF en andere SDKs

Try it with
ESP Launchpad

ESP32 en ChatGPT

Inzichten van
Espressif-
ontwikkelaarsAutomatisering
met Rainmaker
en MatterTest van de
ESP32-S3-BOX-3

In dit nummer

- › Een open-source spraakherkenningsserver...
- › Rust en embedded: een krachtig duo
- › Gezichtsherkenning met de ESP32-S3-EYE
- › Walkie-talkie met ESP-NOW
- › Ronde Elektor LED-kerstboom

en veel meer!

De ESP32-P4 ontketend
De volgende microcontroller-era

p. 59

Acoustic fingerprinting
met de ESP32
Songherkenning met de ESP32

p. 80

AIoT Chip-innovatie
Interview met Espressif CEO
Teo Sweeney

p. 35



BP 8 710966 101223



Design-In Expertise & Service

YOUR PARTNER FOR



ESPRESSIF

Ineltek

Leading Espressif franchise distribution partner

Experience

36 years of experience in the semiconductor industry

Latest
News

Ineltek is always up to date regarding Espressif's innovations – Contact us!

Innovation

Features are missing? We address your requests for next product generations

Application
Support

Dedicated and focused in-house support offering

Time
to
Market

Espressif & Ineltek's FAE team are in close & regular contact to speed up your designs

Information

We inform you on Espressif's products & wireless trends in trainings & webinars

Supply

Popular Espressif SoCs, modules and EVKs available from stock

CONTACT US FOR PARTS & SUPPORT



www.ineltek.com

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

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

Start your career with us!

Apply now at personal@ineltek.com



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



Follow us!

INELTEK GmbH

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



63^e jaargang nr. 02401
december 2023/januari 2024
ISSN 2590-0765

Elektor verschijnt acht keer per jaar en is een uitgave van

Elektor International Media B.V.
Postbus 11, 6114 ZG Susteren (Nederland)
Tel.: +31 (0)46 4389444

www.elektor.nl | www.elektormagazine.nl

Voor al uw vragen: service@elektor.nl

Lid worden: www.elektormagazine.nl/abo

Advertenties

Raoul Morreau
Tel. +31 (0)6 4403 9907
raoul.morreau@elektor.com
www.elektormagazine.nl/adverteren

Auteursrecht

© Elektor International Media B.V. - 2023

Niets uit deze uitgave mag verveelvoudigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm of op welke wijze dan ook, zonder voorafgaande schriftelijke toestemming van de uitgever. De auteursrechtelijke bescherming van Elektor strekt zich mede uit tot de illustraties met inbegrip van de printed circuits, evenals de ontwerpen daarvoor. In verband met artikel 30 van de Rijksoctrooiwet mogen de in Elektor opgenomen schakelingen slechts voor particuliere of wetenschappelijke doeleinden vervaardigd worden en niet in of voor een bedrijf. Het toepassen van de schakelingen geschiedt buiten de verantwoordelijkheid van de uitgever. De uitgever is niet verplicht ongevraagd ingezonden bijdragen, die hij niet voor publicatie aanvaardt, terug te zenden. Indien de uitgever een ingezonden bijdrage voor publicatie aanvaardt, is hij gerechtigd deze op zijn kosten te (doen) bewerken. De uitgever is tevens gerechtigd een bijdrage te (doen) vertalen en voor haar andere uitgaven en activiteiten te gebruiken tegen de daarvoor bij de uitgever gebruikelijke vergoeding.

Druk

Senefelder Misset, Mercuriusstraat 35,
7006 RK Doetinchem (Nederland)

Distributie

Betapress, Nederland – AMP, België



IoT-innovatie in galop



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

Teo Swee Ann
(*Oprichter/CEO, Espressif Systems*)

Na het doorslaande succes van zowel de SparkFun- (2021) als de Arduino-samenwerking (2022), is het een genoegen om Espressif aan boord te hebben als onze 2023 gastredactie. Bedankt, Espressif! Met baanbrekende oplossingen zoals de ESP8266 en ESP32 heeft het Espressif-team consequent geavanceerde tools geleverd aan 's werelds meest innovatieve professionele ingenieurs, serieuze makers en toekomstgerichte studenten. Met deze gasteditie van Elektor Mag en de daaruit voortvloeiende boeiende samenwerking met de getalenteerde ingenieurs van Espressif, stellen we diverse voorbeelden van de oplossingen en expertise van het bedrijf ter beschikking van Elektor-leden over de hele wereld.

Wat staat u te wachten bij het lezen van deze gasteditie van Elektor Mag? Op de bekende wijze van Elektor hebben we deze editie gevuld met een breed scala aan content, van projecten tot tutorials, over onderwerpen als songherkenning op een ESP32, embedded ontwikkeling met Rust, ESP32 en ChatGPT, gezichtsherkenning met de ESP32-S3-EYE, handige ingenieursinzichten en nog veel meer. We zijn ervan overtuigd dat het magazine samen met de gratis Bonuseditie die we delen op onze site en via onze nieuwsbrief met de community - maanden van innovatie en tientallen interessante nieuwe elektronica-projecten en -toepassingen zal genereren.

Of u nu ervaring hebt met de producten van Espressif of de die nog niet kent, zorg ervoor dat u al uw op Espressif gebaseerde projecten plaatst op elektormagazine.com/labs. We zijn benieuwd naar uw ontwerpen. Veel plezier met deze uitgave en succes met uw volgende project!

Welkom bij deze speciale gasteditie van Elektor Magazine - bekend om zijn inzet voor het promoten van innovatie, samenwerking en het delen van kennis - mede geschreven door Espressif Systems. We willen het geweldige Elektor-team en de redacteurs hartelijk bedanken voor hun fantastische werk en niet-aflatende steun bij het realiseren van deze publicatie. Het zijn jullie bijdragen die dit allemaal mogelijk hebben gemaakt.

We willen ook onze diepste waardering uitspreken voor de Espressif-gemeenschap en onze gewaardeerde partners, die ons niet alleen hebben gesteund, maar ook actief hebben bijgedragen aan deze editie. Uw vastberaden toewijding en waardevolle ideeën waren van groot belang bij het tot stand komen van deze publicatie.

Deze speciale editie is een bewijs van de gezamenlijke spirit en gedeelde passie voor open-source technologie die onze community en partners uitstralen. In deze editie duiken we diep in de principes die Espressif Systems naar de voorhoede van de IoT-industrie hebben gestuurd. Kijk uit eerste hand naar onze open-source initiatieven en hoe ze hebben bijgedragen aan de opbouw van een robuuste community van ontwikkelaars, die op hun beurt een cruciale rol hebben gespeeld in onze tocht van innovatie en 'empowerment'.

We kijken ook naar de toekomst en zien het transformatieve potentieel van generatieve AI-technologieën, zoals ChatGPT. We zullen zien hoe ze op het punt staan industrieën opnieuw vorm te geven - en hoe Espressif Systems van plan is ze in te zetten voor innovatie, met de inzichten van onze community en partners.

Ga met ons mee op ontdekkingsstocht naar technologie, innovatie en interactie. Gezamenlijk zullen we de verrassende inhoud van deze gasteditie verkennen, die ongetwijfeld uw enthousiasme voor de grenzeloze mogelijkheden van de hedendaagse 'connected' wereld zal aanwakkeren.

Blijf innoveren en delen!



Ons team

Internationaal hoofdredacteur: Jens Nickel | **Content Director:** C.J. Abate | **Internationale redactie:** Asma Adhimi, Roberto Armani, Eric Bogers, Jan Buiting, Stuart Cording, Rolf Gerstendorf (RG), Ton Giesberts, Hedwig Hennekens, Saad Imtiaz, Alina Neacsu, Dr. Thomas Scherer, Jean-Francois Simon, Clemens Valens, Brian Tristram Williams | **Vaste medewerkers:** David Ashton, Tam Hanna, Ilse Joostens, Prof. Dr. Martin Ossmann, Alfred Rosenkränzer | **Vormgeving & Layout:** Harmen Heida, Sylvia Sopamena, Patrick Wielders | **Directeur:** Erik Jansen | **Technische vragen:** redactie@elektor.com

ESP32 en ChatGPT

op weg naar een zelf-programmerend systeem...



AIoT chipinnovatie

een vraaggesprek met Teo Swee-Ann, CEO van Espressif



Rubrieken

3 Voorwoord

50 Must-haves voor je elektronica-werkplek

overwegingen en tips van Espressif-technici

106 Achter de schermen van Espressif

138 Tech the Future: waar gaat Smart Home-IoT naar toe?



Achtergrond & info

13 ESP-Launchpad tutorial

van nul tot flashen in een paar minuten

28 Van idee tot schakeling met de ESP32-S3

zo bouw je een prototype met Espressif-chips

35 AIoT chipinnovatie

een vraaggesprek met Teo Swee-Ann, CEO van Espressif

40 Simuleer ESP32 met Wokwi

de digitale tweeling van je project

46 Test van de ESP32-S3-BOX-3

een uitgebreid AIoT-ontwikkelingsplatform

53 Het ESP Rainmaker-verhaal

hoe we 'jouw' IoT-cloud bouwden

56 Bouw van de Elektor Cloc 2.0

een Elektor-product uitgepakt door Espressif

59 De ESP32-P4 ontketend

de volgende microcontroller-era

64 Rust + embedded

een krachtig ontwikkelduo

70 Wie zijn de onervaren embedded Rust-ontwikkelaars?

zo cultiveert Espressif embedded Rust voor de ESP32

74 De Espressif SoC-reeks

76 Een PLC met oplossingen van Espressif

met de mogelijkheden en functionaliteit van het ISOBUS-protocol

78 Het ESP32-S3 VGA-board

volg Bitluni bij dit productontwerp



100 Achter de schermen: vraaggesprek met Arduino over de Nano ESP32

110 Stroomlijnen van MCU-ontwikkeling met ESP-IDF Privilege Separation

132 Matter stuwt het slimme huis voorwaarts

ontsluit het potentieel van Smart Home IoT

Industry

89 Een eenvoudiger en handiger leven

een amateurproject op basis van de ESP8266-module van Espressif

90 IoT-apps bouwen zonder software-expertise

met het Blynk IoT-platform en Espressif-hardware

92 Bouw een slimme gebruikersinterface op ESP32

94 Snelle en gemakkelijke IoT-ontwikkeling met M5Stack

99 Een distributeur met meerwaarde voor IoT en meer

Knoopcel-schakelaar met ESP32-C2

batterij gaat tot 5 jaar mee



126



Projecten

- 6 Een E-Ink WiFi kleuren-fotolijst**
- 16 ESP32 en ChatGPT**
op weg naar een zelf-programmerend systeem...
- 24 Walkie-talkie met ESP-NOW**
niet echt WiFi, niet echt Bluetooth, maar...
- 80 Acoustic fingerprinting met de ESP32**
songherkenning met het open-source project Olaf
- 84 Ronde kerstboom 2023** 
een high-tech manier om de feestdagen te vieren
- 96 Prototype van een ESP32-gebaseerde energiemeter**
- 114 Een open-source spraakherkenningsserver**
...en de ESP BOX
- 119 Het denkende oog**
gezichtsherkenning en meer met de ESP32-S3-EYE
- 126 Knoopcel-schakelaar met ESP32-C2**
ontwerp en prestaties



Binnenkort

Elektor januari/februari 2024

Zoals u van ons gewend bent, hebben we weer een interessante mix van projecten, achtergrond- en andere informatie en tips & trucs van en voor elektronica-engineers en -makers voor u samengesteld. De focus ligt op voeding en energie.

Een greep uit de inhoud:

- > ESP32 energiemeter
- > Optimaliseren van balkoncentrales
- > Variabele lineaire voeding
- > Eenvoudige PV-vermogensregelaar
- > Groente bewaren in de slimme keuken
- > Programmeren voor PC, tablet en smartphone
- > Project in het kort: lader/ontlader

...en nog veel meer!

Aankondigingen onder voorbehoud.

Elektor januari/februari 2024 verschijnt omstreeks 10 januari 2024.

BONUS EDITIE

Wil je meer inhoud van Elektor en Espressif? In de komende weken zullen we een bonuseditie van Elektor Magazine publiceren – ook met Espressif als gastredacteur – die boordevol meer projecten, tutorials en achtergrondartikelen zit: een Dekatron in Espressif-stijl, een op ESP32 gebaseerde authenticatiedongle, een ESP-BOX-gebaseerd pratend ChatGPT-systeem, een interview met Home Assistant-oprichter Paulus Schoutsen en nog veel meer! Abonneer je op het Elektor E-Zine (elektormagazine.nl/ezone) en wij bezorgen de bonuseditie rechtstreeks in je inbox!

Een E-Ink WiFi kleuren-fotolijst

Jeroen Domburg (Espressif)

E-ink displays worden vaak in apparaten toegepast, maar dat zijn dan vooral zwart/wit-exemplaren; de weinige kleuren-displays zijn vrij duur en moeilijk in te passen in een DHZ-project. In dit artikel bespreken we het gebruik van een redelijk geprijsd zeven-kleuren-display dat een goede kleurechtheid biedt – dankzij dithering-technieken – en met de mogelijkheid om verbinding te maken met een WiFi-netwerk met behulp van een ESP32-C3-WROOM-02-module van Espressif.

Een tijdje geleden werd onze kleine geboren. Omdat haar (over)grootouders nogal verspreid over de globe wonen, zijn bezoeken in levenden lijve niet vaak mogelijk. Natuurlijk is videobellen tegenwoordig een fluitje van een cent zodat we wel contact kunnen houden, maar ik wilde dat iedereen haar ook kon zien als we niet samen online waren. Dus dacht ik: waarom maken we geen fotolijstje met een kleurendisplay en een WiFi-chip? Vooral omdat ze zo snel groeit, zou het leuk zijn om elke dag een nieuwe actuele foto te sturen.

Je hebt vast wel eens een E-Ink display gezien: zo niet in een e-boek-lezer, dan toch zeker in een supermarkt of andere winkel waar ze de oude papieren prijskaartjes hebben vervangen. Ze zijn geweldig voor het weergeven van statische afbeeldingen, omdat ze het meest recente beeld vasthouden zonder dat dit stroom kost. De meeste van deze schermpjes zijn zwart/wit, met soms rood of geel als extra kleuroptie. Leuk, maar een zwart/wit-foto doet geen recht aan de levendige kleuren van een blij baby die met haar kleurrijke speelgoed aan de slag is.

Wat er bestaat

Op het moment van schrijven (begin 2023) beginnen er eindelijk e-boek-lezers met kleurenschermen te verschijnen. De kwaliteit lijkt behoorlijk, met kleuren die die van een krant benaderen. Helaas zijn ze duur, en de displays zelf lijken nog niet beschikbaar te zijn als reserveonderdelen, dus er is geen kans op eenvoudig hergebruik in een doe-het-zelf project.

Er is echter één type E-Ink kleurendisplay beschikbaar voor gebruik in DHZ-projecten. Het lijkt dat deze meestal verkocht worden door Waveshare, een Chinees bedrijf dat zich richt op de fabrikantenmarkt; het meest voorkomende model is een 5,65-inch 640×448-unit met zeven kleuren (**figuur 1**). Met slechts zeven kleuren en een verversingstijd van ongeveer een minuut lijkt het bedoeld als de grote broer van de varianten met een prijskaartje, waarbij statische informatie wordt weergegeven met de kleuren die worden gebruikt om bijvoorbeeld egale achtergrondkleuren te hebben. Ze zijn zeker niet bedoeld om fotorealistische afbeeldingen weer te geven. Dat is natuurlijk jammer: een E-Ink-display kan heel goed een "afgedrukte foto" simuleren, omdat het geen achtergrondverlichting nodig heeft en volledig statisch is. Ik ben niet de enige die er zo over denkt, en heel wat mensen ([1], [2], [3]) hebben al geprobeerd om met dithering de weergave zo op te peppen dat afbeeldingen een beetje natuurgetrouw worden weergegeven. Hier beschrijf ik mijn poging om dit voor elkaar te krijgen.

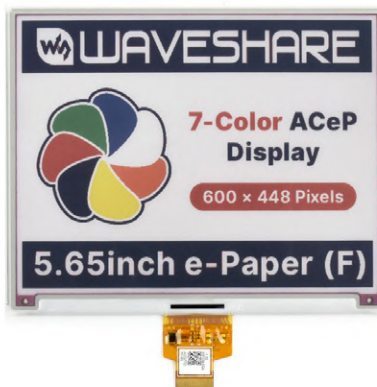


De hardware

In de eerste plaats moest ik de hardware bouwen. Ik begon mijn ontwerp met een aantal vereisten. Ik wilde dat het fotolijstje één keer per dag via WiFi verbinding zou maken met een server om nieuwe foto's op te halen. Een moest dan worden weergegeven op het E-Ink display. Als het lukte om nieuwe afbeeldingen te downloaden, moest een daarvan worden getoond, anders moest een eerdere afbeelding zichtbaar blijven. Daarna gaat de fotolijst weer 24 uur slapen. De keuze van een geschikte chip was niet moeilijk omdat ik toevallig voor Espressif werk, een bedrijf dat geweldige microcontrollers met WiFi-functionaliteit maakt die ook een deep-sleep modus hebben die behoorlijk goed werkt. Ik heb voor mijn project een ESP32C3 gekozen: die is lekker goedkoop en heeft alle functies die ik nodig heb.

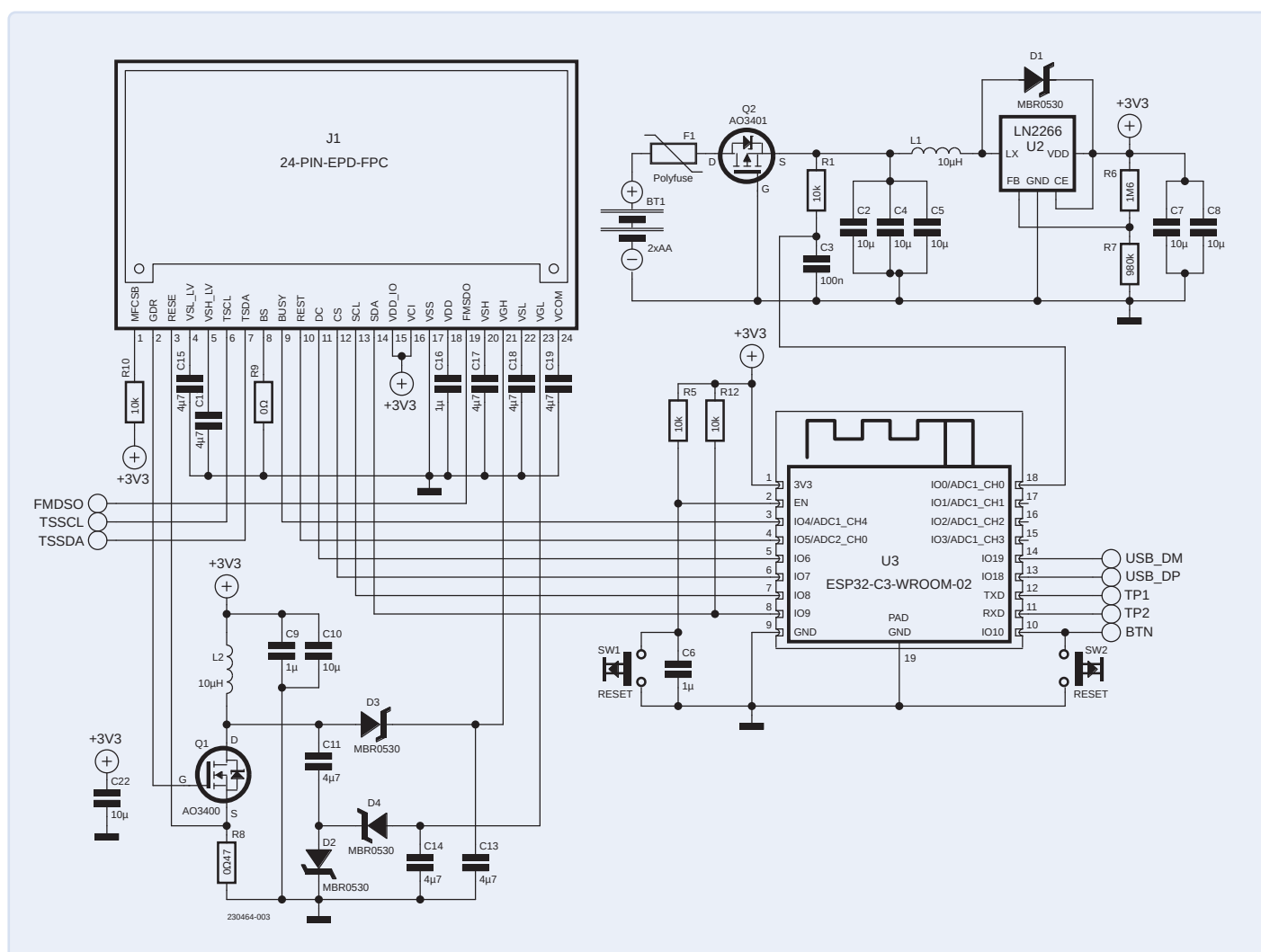
Verder had ik ook een aantal eisen voor de voeding. Ik wilde het display zoveel mogelijk op een normaal fotolijstje laten lijken (afgezien van het feit dat het 's

nachts van foto wisselt), dus een externe voeding was uit den boze; de voeding moest een aan boord zijn. Ik wilde het ook internationaal verzenden, dus het zou het beste zijn als het zonder batterij verzonden kon worden. Die eis sloot elke soort oplaadbare Li-ion cel uit. Zoals je kunt zien in het schema van **figuur 2**, heb ik besloten om 2 AA-batterijen te gebruiken; die zijn overal verkrijgbaar en zelfs de grootouders kunnen die moeiteloos vervangen als dat nodig is.

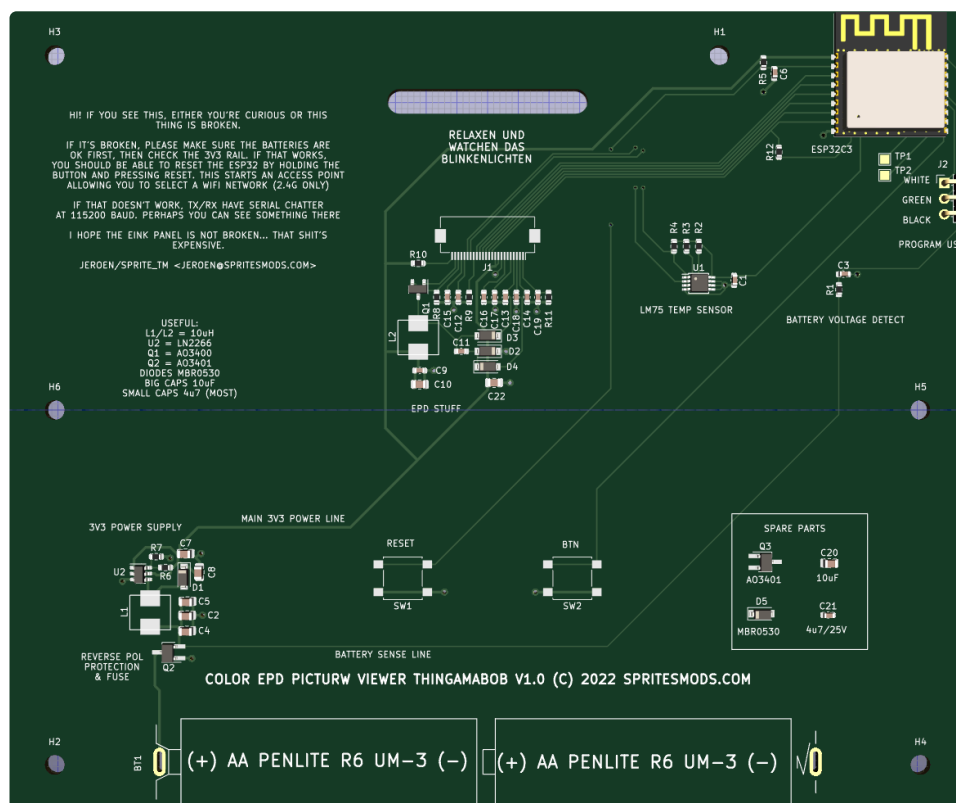


Figuur 1. Het 5,65-inch, zeven kleuren, 640x448-display van Waveshare (bron: Waveshare).

Figuur 2. Het volledige schema van dit project, inclusief de 3,3 V DC/DC-converter, de ESP32-C3-WROOM-02-module en de voeding voor het display.



Figuur 3.
Onderdelenopstelling
(componentzijde) van de
ruim bemeten print voor
het project.



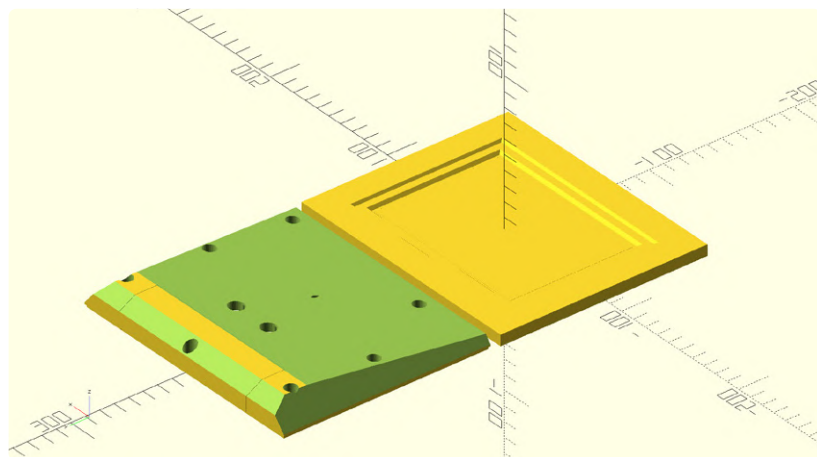
Omdat 2 AA-batterijen een totale spanning leveren die begint bij ongeveer 3,2 V en die afneemt tijdens de levensduur, had ik een boostschakeling nodig om die spanning op te voeren tot de 3,3 V die het E-Ink display en de ESP32C3 nodig hadden. Als ik naar de ontlaadgrafieken [4] kijk, zou het boostcircuit moeten werken met een ingangsspanning van ongeveer 2,0 V om bij alkaline-batterijen het onderste uit de kan te halen. Bovendien moet de ruststroom laag genoeg zijn, omdat de ESP32C3 zelfs in deep sleep-modus van stroom moet worden voorzien. Met dit 'boodschappenlijstje' ging ik op zoek naar een bruikbare boost-converter. Ik heb gekozen voor de Natlinear LN2266. Deze kleine chip wordt gemaakt door een Chinese fabrikant, zodat hij waarschijnlijk iets minder gevoelig voor de huidige siliciumcrisis zal zijn. Hij werkt vanaf 2 V, heeft een nullaststroom van 56 μA en kan de ongeveer 500 mA aan WiFi-opstartstroom leveren die de ESP32C3 nodig

heeft. Ik heb de schakeling zo ontworpen dat de LN2266 zijn stroom uit de twee batterijen haalt via een p-kanal MOSFET die zo is aangesloten de batterijen te beschermen als ze verkeerd worden geplaatst. De accuspanning gaat ook via een RC-filter naar een van de ADC-pinnen van de ESP32-C3, zodat deze een idee heeft van hoeveel stroom er nog over is. Aanvankelijk had ik ook een kleine polyfuse in het ontwerp opgenomen, maar die had voor de goede werking een te grote spanningsval nodig. Ik heb hem vervangen door een weerstand van nul ohm in mijn printen, en heb hem helemaal verwijderd uit de ontwerpbestanden.

De ESP32-C3 wordt geleverd in de vorm van een ESP32-C3-WROOM-02-module, rechts te zien in het schema van figuur 2. Deze module bevat de WiFi microprocessor plus 4 MB flash alsmede alle benodigde RF-componenten, inclusief een antenne in de vorm van een printspoor. Om hem te programmeren heb ik een interne header toegevoegd, waar ik een USB-kabel aan kan solderen. De interne USB-JTAG serieelconverter zorgt voor de rest. Ik heb een OTA firmware-update-functie toegevoegd aan de firmware (dus de lijst kan nieuwe firmware downloaden van mijn server); als ik dan units moet updaten die al gesloten en op locatie zijn, kan ik dat doen door de nieuwe firmware remote te pushen.

Dan komen we bij het E-Ink display. Het is verbonden met de print via een platte lintkabel en heeft een aantal dingen nodig om te werken, zoals te zien is linksonder in figuur 2: een MOSFET, een spoel en wat diodes en condensatoren om de benodigde spanningen te genereren, een paar ontkoppelcondensatoren en een weerstand of twee. Het E-Ink display heeft ook een aansluiting voor een externe LM75-temperatuursensor. Ik heb er 'voor het geval dat' een in het ontwerp opgenomen, maar de

Figuur 4. De behuizing,
ontworpen met
OpenSCAD.



huidige firmware gebruikt hem niet, omdat het display net zo goed werkt zonder die sensor.

Dit alles is gemonteerd op een zeer ruim bemeten print, waarvan de componentenopdruk (componentenzijde) te zien is in **figuur 3**. Aangezien het kale E-Ink display van glas is en ietwat kwetsbaar, heb ik de print zo ontworpen dat hij als een steun fungeert om de fotolijst als geheel minder breekbaar te maken. De print is iets groter dan het display, omdat de montagegaten, de paar through-hole componenten (zoals de batterijcontacten) en de WiFi-antenne niet onder het E-Ink display mogen verdwijnen.

Behuizing

Tot slot is er de behuizing, die ik heb ontworpen met OpenSCAD en die is afgebeeld in **figuur 4**. Met een paar kunstgrepen ziet die er niet zo zwaarlijvig uit – de AA-batterijen moeten er immers in passen, en ik wilde geen grote bult aan de onderkant, dus heb ik wat grote afschuiningen en een schuine achterzijde gebruikt. Die zie je niet als je de fotolijst van voren bekijkt, dus het geheel ziet er een stuk slanker uit. Het fotolijstje leek ook tamelijk fors in vergelijking met het zichtbare deel van het E-Ink display. Om dat te verhelpen, heb ik een kader toegevoegd (ook wel ‘passe-partout’ geheten). Omdat dit kader wit is en mooi contrasteert met de zwarte behuizing, breekt het het teveel aan zwart en ziet alles er proportioneel uit.

Aan de achterkant zit het batterijcompartiment. Dit wordt geopend door één schroef los te draaien (ik wilde niet vertrouwen op fragiele 3D-geprinte clips om het op zijn plaats te houden) en omdat de opdruk van de print de juiste stand van de batterijen toont, zou het vervangen van batterijen eenvoudig moeten zijn.

Software/firmware en WiFi-verbinding

Aangezien de ESP32C3 wordt gevoed door batterijen, heb ik geprobeerd om hem zo weinig mogelijk te laten doen. Hij moet verbinding maken met WiFi, met mijn server communiceren, kijken of er nieuwe foto's zijn die hij nog niet heeft gedownload en, als dat zo is die ophalen, uitzoeken wat de beste afbeelding is om te tonen (de nieuwste of de minst bekeken foto), die weergeven en afsluiten. Uiteraard zijn er nog andere dingen die gedaan moeten worden, zoals het afhandelen van fouten en de omgang met lege batterijen.

Om verbinding te maken met WiFi heeft het apparaat een SSID en een wachtwoord nodig. Ik wilde dit niet hard coderen, voor het geval het veranderd moest worden. Daarom wordt de firmware geleverd met een kopie van ESP32-WiFi-Manager [5], aangepast om enkele bugs op te lossen. Als je op een van de knoppen op de achterkant van het fotolijstje drukt terwijl je het lijstje reset met de andere knop, wordt er een access point gestart waarmee je verbinding kunt maken. Een embedded webserver



levert vervolgens een gebruikersinterface om een nieuwe SSID te kiezen en het wachtwoord ervoor in te stellen. Nadat de fotolijst verbinding heeft gemaakt met WiFi, probeert hij gegevens op te halen van een vast gecodeerde URL om te zien wat gedaan moet worden. De URL codeert ook enkele statusgegevens, zoals de MAC van het apparaat, de batterijspanning en de huidige firmware-ID. De server stuurt de ID van de meest recente firmware voor dat specifieke apparaat terug, evenals een index van de tien meest recent opgehaalde afbeeldingen. Er worden ook enkele voorkeuren verstuurd, zoals de tijdzone en het tijdstip waarop de fotolijst moet proberen wakker te worden en een update uit te voeren.

De fotolijst heeft eigenlijk plaats voor tien foto's in zijn flashgeheugen; hij vervangt de oudste door nieuw gedownloade afbeeldingen als de server foto's in de aanbieding heeft die nog niet in het lokale geheugen zitten. Op deze manier is er altijd een voorraad relatief recente foto's, wat handig is als om wat voor reden dan ook de verbinding wegvalt. Het is zelfs mogelijk om de fotolijst naar een andere locatie te verplaatsen: hoewel hij zonder WiFi-verbinding oude foto's zal hergebruiken, wordt er nog steeds elke dag iets anders weergegeven. Het grootste deel van de server-side software is niet zo ingewikkeld: er is een eenvoudige front-end webpagina gemaakt rond Cropper.js [6], die je kunt openen op je telefoon of PC. Hiermee kun je een foto selecteren en het gedeelte dat je op het fotolijstje wilt weergeven uitsnijden. Er is een stukje JavaScript dat de afbeelding vervolgens bijsnijdt en opschaaft aan de clientzijde en de resulterende gegevens naar de server stuurt. De server neemt deze over, converteert ze naar ruwe data voor het E-Ink display en slaat ze op in een MariaDB-database. Wanneer een fotolijst verbinding maakt, slaat de server de door de lijst verzonden gegevens op, zodat ik een logboek van accuspanningen heb en ik kan zien of een firmware-update echt 'gelukt is'. De server controleert vervolgens de MariaDB-database voor de laatste tien afbeeldingen en andere informatie, zoals de laatste firmware-versie, en codeert dat in JSON en retourneert die. Dat is allemaal eigenlijk triviaal.



Figuur 5. De foto die voor testdoeleinden is gebruikt.



Figuur 6. Eerste conversiepoging, met 100% zwart en 100% wit.



Figuur 7. Zwart-wit conversie met error diffusion.



Figuur 8. De foto van figuur 7 met toevoeging van RGB-waarden (zie tekst).



Figuur 9. De voorbeeldfoto met de toepassing van de CIEDE2000-standaard.

Dithering

Alleen het omzetten van de RGB-afbeelding in de zeven vrij specifieke kleuren die het E-Ink display kan weergeven, is werkelijk gecompliceerd. Neem bijvoorbeeld de afbeelding van **figuur 5**.

Als we dit in zwart en wit willen omzetten, kunnen we gewoon de luminantie (lichtheid) voor elke pixel bepalen en, als deze dicht bij zwart ligt, de pixel zwart maken; als deze dicht bij wit ligt, maken we hem wit. Met andere woorden, we nemen de dichtstbijzijnde 'kleur' (waarbij we de 'kleuren' beperken tot 100% zwart en 100% wit) en veranderen de afbeelding in het resultaat van **figuur 6**. Dat lijkt in de verste verte niet op het origineel. Zelfs met zwart/wit kunnen we het beter doen door iets te gebruiken dat we *error diffusion* noemen. In feite nemen we, elke keer dat we een grijze pixel zwart of wit maken, het verschil tussen de luminantie van de pixel in de originele foto en de luminantie van de pixel die we daadwerkelijk op het E-Ink scherm tonen (de 'error' in 'error diffusion') en voegen dit gedeeltelijk toe aan de omliggende pixels (de 'diffusion'). Dit proces kan op verschillende manieren worden uitgevoerd, waarvan Floyd-Steinberg de meest voorkomende is, en het geeft een vrij goed zwart/wit

geditherede afbeelding, zoals geïllustreerd in **figuur 7**. We kunnen deze methode ook gebruiken voor ons display met zeven kleuren. Het probleem is dat de definitie van 'dichtstbijzijnde kleur' een stuk ingewikkelder wordt, net als de definitie van 'toevoegen'. Zelfs de definitie van 'kleur' wordt lastig, omdat een E-Ink display geen achtergrondverlichting heeft en de waargenomen kleuren dus verschillen afhankelijk van de lichtbron: als je het verlicht met een kaars, zie je andere kleuren dan wanneer je het display in fel zonlicht bekijkt.

Om de kleuren te krijgen, nam ik een lamp met regelbare kleurtemperatuur, stelde die in op 4800 K (de gemiddelde kleurtemperatuur waarop ik denk dat het scherm zal worden bekeken), gaf de zeven kleuren weer als egale rechthoeken op het display en nam er een foto van. Ik importeerde deze in mijn computer en paste de kleuren handmatig aan totdat de kleuren op de monitor zo dicht mogelijk in de buurt kwamen van de E-Ink kleuren. Vervolgens nam ik de gemiddelde RGB-waarden van de zeven kleuren en voerde die in mijn programma in.

Voor deze zeven kleuren hebben we een manier nodig om twee kleuren te vergelijken, en omdat we echte kleuren gebruiken en niet alleen zwart en wit, kunnen we er niet

langer mee weggelaten om gewoon de helderheid te gebruiken. Een snelle manier om twee kleuren te vergelijken is door de (geïntegreerde) RGB-ruimte te zien als een driedimensionale ruimte en de rechte lijnige afstand te gebruiken als maat voor hoe dicht twee kleuren bij elkaar liggen. Met dit model kunnen kleuren worden toegevoegd door simpelweg de RGB-waarden bij elkaar op te tellen. Als we de Floyd-Steinberg dithering aanpassen om deze methode voor het kiezen van de dichtstbijzijnde kleur toe te passen, krijgen we een redelijk acceptabele afbeelding (**figuur 8**).

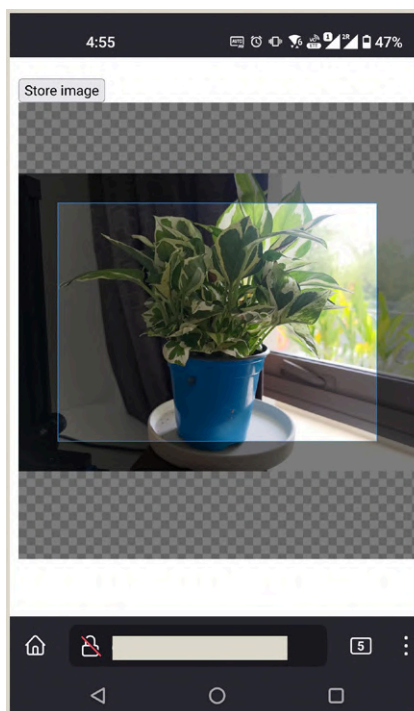
Dat is eigenlijk niet zo slecht! Er zijn echter een paar vreemde kleurafwijkingen. De meest in het oog springende is dat de bloempot niet dezelfde tint blauw heeft: dat komt omdat het E-Ink display simpelweg niet over de juiste kleuren beschikt om die tint te reproduceren, en geen algoritme kan dat compenseren. Maar er is meer vreemds: merkwaardige kleurbanden, bijvoorbeeld in de schaduw onder de linkerarm van de aap, en de buik van de aap is meer oranje dan in de originele foto.

Een andere aanpak: kleurruimte

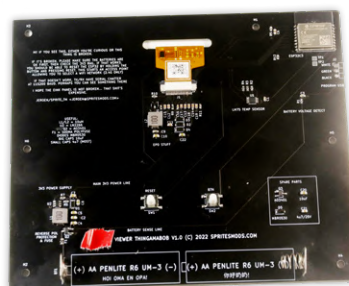
Het probleem met het berekenen van kleurverschillen in RGB-ruimte is dat je ogen eigenlijk niet in een lineaire RGB-ruimte werken. Het verschil tussen twee kleuren is eigenlijk veel moeilijker te definiëren en er zijn hiervoor verschillende benaderingen bedacht. Een van de eerste was om de RGB-kleuren te converteren naar de CIELAB-kleurruimte en het verschil te nemen. Deze aanpak wordt veel aanbevolen op internet, maar het blijkt dat het niet echt goede resultaten geeft voor kleuren die niet volledig verzadigd zijn. Voor mij bleek de beste aanpak het gebruik van de CIEDE2000-standaard [7]. Dit is een van de meer recente modellen, en hoewel het niet triviaal is om te berekenen, geeft het wel de beste resultaten, zoals te zien **figuur 9**. Het is maar goed dat ik al besloten had om dit aan de server-kant te doen, zodat ik de batterijen niet leeg hoefde te trekken met dit zware rekenwerk. Merk op dat de kleurband in de schaduwen weg is en dat de buik van de aap een meer accurate tint rood heeft. Er zijn nog steeds fouten in de afbeelding, zoals de kleur van de bloempot, maar zoals ik al eerder zei, komt dit omdat de E-Ink gewoon niet de kleuren bezit om die specifieke tint goed weer te geven.

Omwillen van de snelheid is alle logica geïmplementeerd in een eenvoudig C-programma. Nadat de afbeelding is bijgesneden in JavaScript op de client (met Cropper.js), wordt deze naar de server gestuurd, waar het PHP-script dit C-script aanroept om de afbeelding om te zetten in E-Ink pixels, die worden opgeslagen in een MariaDB-tabel voor de fotolijstjes om op te halen wanneer ze verbinding maken.

De webpagina is ook geschikt voor mobiel gebruik, dus als ik of iemand anders met toegang tot de pagina een bijzonder mooie foto neemt, kunnen we deze meteen bijknippen en in de wachtrij zetten voor distributie naar alle geïnstalleerde fotolijstjes (**figuur 10**).



Figuur 10. Quick-crop functie voor mobiel gebruik.



Figuur 11. De compleet geassembleerde print.



Figuur 12. De twee delen van de behuizing voor dit project.

Resultaat

Als de print gereed is en de 3D-geprinte onderdelen klaarliggen, is het tijd om alles in elkaar te zetten (**figuur 11**). Alle elektronica is op de achterkant van de print gemonteerd. De print is eigenlijk ontworpen met het oog op reparbaarheid: als het stuk gaat en ik ben niet in de buurt om het te repareren, ken ik misschien wel een paar mensen in de buurt die goed zijn met elektronica en er even naar kunnen kijken. Dat betekent dat er wat debug-instructies op de achterkant staan en zelfs wat reserveonderdelen voor het geval er iets kapot gaat. Verder is de print niet al te dicht bestukt: de afmetingen worden voornamelijk bepaald door het E-Ink display aan de andere kant. Gezien de beschikbare ruimte had ik grotere componenten kunnen gebruiken, maar ik heb veel rollen met 0603-onderdelen op voorraad, dus daar heb ik het bij gehouden voor het standaardgrut; met behulp van een goede soldeerbout en een binoculaire microscoop kan ik dat allemaal prima met de hand solderen.

Figuur 13. Het batterijvak met zijn eigen klepje.



Figuur 14. De uiteindelijke fotolijst naast het origineel.



De behuizing is te zien in **figuur 12** compleet met wit passe-partout. Die laatste heeft een uitsparing voor het E-Ink display: zelfs als de lijm die het op de print houdt op de een of andere manier zou losraken, wordt het hierdoor op zijn plaats gehouden. De behuizing wordt gesloten met schroeven die in messing bussen worden geschroefd. Die laatste worden op hun plaats gehouden door ze te verhitten met een soldeerbout (met een oude verroeste soldeerstift erin – ik wil geen goede stiften opofferen) en ze op hun plaats te smelten.



Gerelateerde producten

- > **Waveshare 5.65" ACeP 7-Color E-Paper E-Ink Display Module (600x448)**
www.elektor.nl/19847
- > **ESP32-DevKitC-32E**
www.elektor.nl/20518
- > **Dogan Ibrahim, The Complete ESP32 Projects Guide, Elektor 2019**
www.elektor.nl/18860

Alles zit aan elkaar met een stel M3-schroeven. Het batterijvak heeft een eigen klepje (figuur 13), dus je hoeft niet alles uit elkaar te halen om de batterijen te vervangen. De batterijen gaan volgens mijn berekeningen meer dan een jaar mee. Merk ook op dat er twee knoppen op de achterkant zitten. De ene is direct verbonden met de reset van de ESP32 en de andere met een GPIO voor algemeen gebruik. Je kunt de resetknop gebruiken om het apparaat een handmatige verbodings- en verversingscyclus te laten uitvoeren. Als je de andere knop ingedrukt houdt terwijl de fotolijst wordt gereset, wordt een toegangspunt gestart waarmee je verbinding kunt maken met de fotolijst en de WiFi-verbindingsgegevens opnieuw kunt configureren.

Tot slot kun je in **figuur 14** het eindresultaat bewonderen. Deze foto haalt niet de eerste prijs qua natuurgetrouwheid, maar het feit dat we de lijst elke dag een nieuwe foto kunnen sturen van waar ook tere wereld, maakt dat meer dan goed.

Zoals gebruikelijk voor mij is dit project open source: met een 3D-printer, toegang tot een server en wat handigheid kun je je eigen versie van dit fotolijstje maken. Alle bronnen, print-layouts enzovoort zijn te vinden op GitHub [8].

230464-03

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via jeroen@spritesmods.com of naar de redactie van Elektor via redactie@elektor.com.

Over de auteur

Jeroen Domburg is Senior Software en Technical Marketing Manager bij Espressif Systems. Met meer dan 20 jaar embedded ervaring is hij betrokken bij zowel het software- als het hardware-ontwerpproces van de SoC's van Espressif. In zijn vrije tijd knutselt hij ook graag met elektronica om dingen te maken die praktisch bruikbaar zijn, maar ook minder nuttige zaken.

WEBLINKS

- [1] GitHub E-Paper zeven-kleuren-fotolijst: <https://github.com/robertmoro/7ColorEPaperPhotoFrame>
- [2] Raspberry Pi E-Paper lijst:
https://reddit.com/r/raspberry_pi/comments/10dbhnj/i_built_a_raspberry_pi_epaper_frame_that_shows_me
- [3] E-Paper fotoproject op YouTube: <https://youtu.be/YawP9RjPcJA>
- [4] Ontlaadgrafieken: <https://powerstream.com/AA-tests.htm>
- [5] ESP32-WiFi-Manager: <https://github.com/tonyp7/esp32-wifi-manager>
- [6] Cropper.js: <https://fengyuanchen.github.io/cropperjs>
- [7] CIEDE2000-standaard: https://en.wikipedia.org/wiki/Color_difference#CIEDE2000
- [8] GitHub-repository: https://github.com/Spritetm/picframe_colepd



ESP-Launchpad tutorial

van nul tot flashen in een paar minuten

Dhaval Gujar, Espressif

Ontwikkel je met Espressif-chips? ESP-Launchpad is een webgebaseerde tool die het evalueren en testen van firmware vereenvoudigt. Laten we het eens van dichterbij bekijken.

Dit artikel introduceert ESP-Launchpad, een webgebaseerde tool dat het evalueren en testen van firmware voor Espressif-chips tot een fluitje van een cent maakt. Het vereenvoudigt het opzetten van de ontwikkelomgeving, maakt gebruik van webbrowsers voor het flashen van firmware en biedt voordelen voor zowel ontwikkelaars als niet-ontwikkelaars.

Waarom ESP-Launchpad?

Wanneer je een open-source project bouwt, wil je een gemakkelijke manier hebben voor gebruikers om het te evalueren. Voor embedded ontwikkelaars betekent komt dat neer op het instellen van de development host en programmeertools, het klonen van het project, compileren en vervolgens het programmeren van de hardware.

Door variaties in de development host en zijn software-omgeving kan er daarbij een heleboel misgaan. Zelfs als alles goed werkt, is het voor gebruikers die geen ontwikkelaar zijn maar gewoon het project willen uitproberen, te veel werk.

Als je ontwikkelt met een van de Espressif-chips, heb je nu een betere manier om je project eenvoudig te evalueren. ESP-Launchpad vereenvoudigt dit proces, waardoor het evalueren en testen van firmware ontworpen voor het ESP-platform snel en probleemloos verloopt.

"Op uw plaatsen, klaar, Launchpad!"

ESP-Launchpad is een webgebaseerde ervaring die gebruik maakt van de nieuwste browsermogelijkheden om op een eenvoudige manier

vooraf gebouwde firmware op ESP-apparaten te flashen. Dit omzeilt de noodzaak voor de traditionele speciale software-installaties voor het instellen van de development host, en maakt gebruik van de toegankelijkheid van webplatforms. Het benut de seriële webinterface die wordt ondersteund door de browsers Chrome, Edge en Opera om vanuit de browser te communiceren met het development board om het te programmeren en toegang tot de seriële console te bieden. Laten we eens kijken hoe je gemakkelijk je firmware kunt lanceren met ESP-Launchpad.

Vorbereiden van de firmware

Zodra de firmware is voltooid, is de volgende stap om deze te bouwen voor alle targets die je wilt ondersteunen en ze te converteren naar

enkele binaries. Wees gerust, *esptool.py* (het one-stop-shop Python-hulpprogramma van Espressif voor alles wat met flashen te maken heeft) biedt een handige *merge_bin* optie die je kan helpen om op eenvoudige wijze zo'n binary te maken. Je kunt hier meer over vinden op onze documentatiepagina [1]. Deze enkele binaries moeten worden overgebracht naar een publiek toegankelijke URL die we later zullen gebruiken.

Opmerking: deze tutorial gaat niet verder in hoe je dat moet doen.

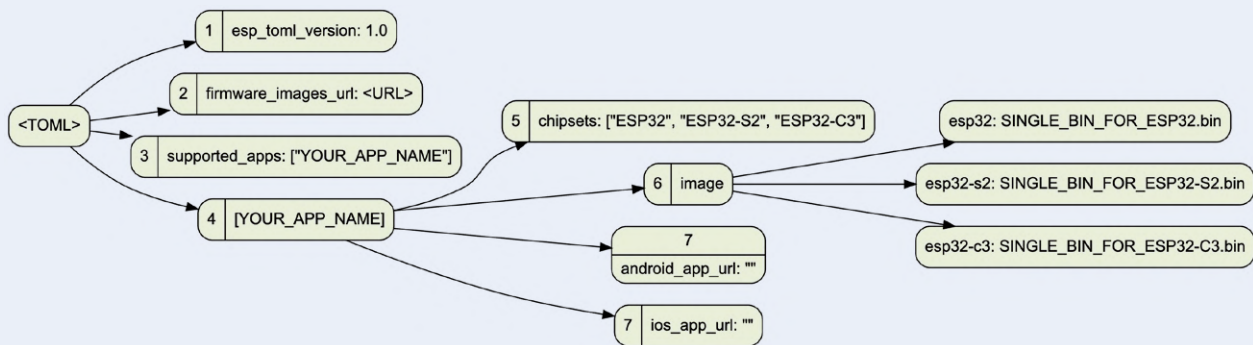
Leuk om te weten: het geheime ingrediënt dat ESP-Launchpad aandrijft is in feite een JavaScript-port van *esptool* met de naam *esptool-js*, die onder de motorkap gebruik maakt van de WebSerial API.

Begrip begint bij de basis: de kracht van TOML

Voordat we in de praktijk duiken, maken we ons eerst vertrouwd met de TOML-structuur van ESP-Launchpad, een centraal aspect van ESP-Launchpad. De TOML-configuratiebestanden van ESP-Launchpad dienen als eenvoudige configuratie-blauwdruk die je firmware-componenten, ondersteunde hardware en aanvullende telefoon-apps in detail beschrijven.



ESP-Launchpad is een webgebaseerde ervaring die gebruik maakt van de nieuwste browsermogelijkheden om op een eenvoudige manier vooraf gebouwde firmware op ESP-apparaten te flashen.



Figuur 1. Diagram van de TOML-structuur.

Hieronder een voorbeeld:

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

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

Werp een blik op **figuur 1** om dit beter te begrijpen.

- `esp_toml_version` specificeert de versie van het TOML-schema.
- `firmware_images_url` is een publiek toegankelijke URL van de bestandsserver waar je firmware-binaries beschikbaar zijn om te downloaden. Pro tip: wij hosten onze TOML-bestanden en binaries beide op GitHub, met behulp van GitHub Pages.
- `supported_apps` is een array van de lijst met apps die je ondersteunt en waarvoor de binaire bestanden beschikbaar zijn. Je kunt meerdere apps hebben en de ESP-Launchpad UI zal deze apps tonen in de apps-dropdown. De drie waarden die we zojuist hebben bekeken, waren de key value-paren op het hoogste niveau.
- `[ " YOUR_APP_NAME" ]` – dit is een tabel die in feite een verzameling is van key value-paren. Deze bevat:
- `.....chipsets` – een matrix met een lijst van ESP-chipsets waarvoor je vooraf gebouwde firmware zult leveren. Let op de schrijfwijze hier, met uitsluitend hoofdletters, `ESP32-C3`.
- `.....image.<chip-naam>` – dit zijn sleutels die de naam van het binaire image associëren met elk van de ondersteunde targets. Dit wordt toegevoegd aan de `firmware_images_url` om de uiteindelijke URL voor de binary te verkrijgen. Let nogmaals op de schrijfwijze hier, met kleine letters, `esp32-c3`.

- `.....ios_app_url` en `android_app_url` zijn optionele maar speciale key value-paren onder dezelfde app-tabel, waarmee je links kunt weergeven in de vorm van QR-codes die gebruikers eenvoudig kunnen scannen nadat je app succesvol is geflashed.

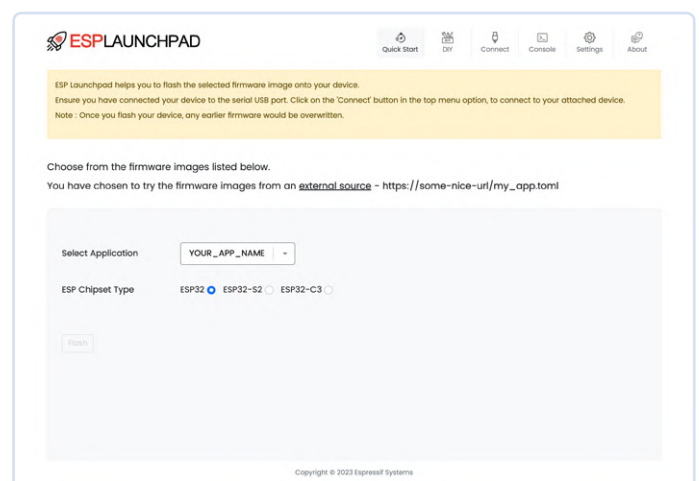
Laten we eens kijken hoe ESP-Launchpad eruit ziet (**figuur 2**) als we de voorbeeld-TOML geven die we zojuist hebben gemaakt, met de volgende denkbeeldige URL:

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

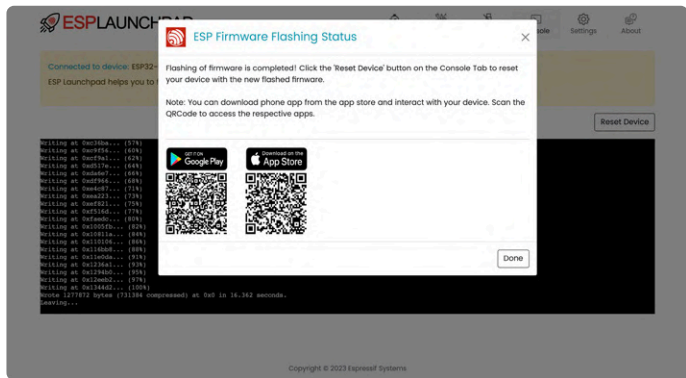
Oké, dat ziet er goed uit! En nu?

Slechts drie eenvoudige stappen voor de gebruiker!

- Aansluiten: sluit het ESP-apparaat aan op de seriële USB-poort van je computer.
- Verbinden: gebruik het tool-menu om verbinding te maken met het apparaat.
- Kiezen en flashen: selecteer de vooraf gebouwde firmware en flash deze.



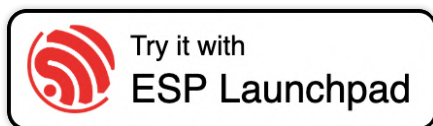
Figuur 2. ESP-Launchpad met voorbeeld-configuratie geladen.



Figuur 3. Het laatste scherm van ESP-Launchpad.

Na het flashen wordt de gebruiker begroet met een console die het apparaatlog toont en een popup met de QR-codes van de telefoon-apps, als je die hebt toegevoegd aan de TOML (figuur 3).

Gefeliciteerd, je hebt zojuist je app gepubliceerd met ESP-Launchpad! Pro-tip: we hebben een badge gemaakt (figuur 4) die je kunt toevoegen aan de *README* of webpagina van je project en die een hyperlink toevoegt naar de flash config-URL van je app! Je kunt meer informatie vinden in de *README.md* van het project [2].



Figuur 4: Voeg deze badge toe!

ESP-Launchpad: waar firmware en community elkaar ontmoeten

Uniek aan ESP-Launchpad is de mogelijkheid om gebruikers hun firmware-apps te laten delen zodat anderen ze kunnen proberen. Door te verwijzen naar een eenvoudig configuratiebestand kunnen ontwikkelaars bepalen waar de vooraf gebouwde binaire bestanden van hun

WEBLINKS

- [1] Basiscommando's – binaries combineren voor flashen met merge_bin: <https://tinyurl.com/esp32mergebinaries>
- [2] De GitHub-repository van dit project: <https://github.com/espressif/esp-launchpad>



firmware vandaan komen, welke hardware wordt ondersteund en zelfs linken naar aanvullende telefoon-apps. Deze holistische benadering zorgt ervoor dat de firmware niet alleen wordt gedeeld als een binair bestand, maar als de ervaring die de ontwikkelaar voor ogen heeft.

Tot slot

Het ware potentieel van elke tool wordt pas gerealiseerd wanneer het in handen is van de gebruikers. We raden je daarom aan om ESP-Launchpad te verkennen, te integreren in je workflow en je ervaringen te delen. Je inzichten en bijdragen zijn van onschatbare waarde bij het verfijnen van ons aanbod. Laten we samen de open-source evaluatie-ervaring naar een hoger plan tillen. [◀](#)

230596-03

Vragen of opmerkingen?

Stuur een e-mail naar de auteur via dhaval.gujar@espressif.com of naar de redactie van Elektor via redactie@elektor.com.

Over de auteur

Dhaval Gujar is engineer bij Espressif Systems, met een focus op embedded systemen. Zijn passie is de dynamische combinatie van technologieën zoals cloud, IoT en meer. Gefascineerd door het zich steeds ontwikkelende technologielandschap verdiept Dhaval zich met enthousiasme in moderne technologische veranderingen.



Gerelateerde producten

- ESP32-DevKitC-32E
www.elektor.nl/20518
- LILYGO T-Display-S3 ESP32-S3 Development Board (met Headers)
www.elektor.nl/20299



Het ESP-NOW-protocol voor flexibele communicatie en besturing



Voor sommige projecten wenst u misschien rechtstreekse communicatie tussen apparaten zonder een netwerkinfrastructuur te gebruiken. Het ESP-NOW protocol ondersteunt een bereik van meer dan 220 meter in een open omgeving. ESP-NOW biedt één-op-één en één-op-veel apparaatcommunicatie en -besturing. Deze SDK biedt voorbeelden van hoe het ESP-NOW-protocol kan worden gebruikt voor OTA, device provisioning en -besturing. Deze SDK bevat ook een implementatie voor een knoopcel-schakelaar en die apparaten kan besturen via het ESP-NOW-protocol.

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



ESP32 en ChatGPT



op weg naar een zelf-programmerend systeem...



Figuur 1- De Arduino Nano ESP32-boards met het Arduino Framework (links) en MicroPython (rechts).

Saad Imtiaz (Elektor Lab)

We hebben de kracht van twee Arduino Nano ESP32-microcontrollers, draadloze communicatie en de ChatGPT API gecombineerd om een slim, interactief communicatiesysteem te maken. Eén van de boards is verbonden met ChatGPT. De antwoorden en de code die dit populaire AI-tool levert worden draadloos overgedragen naar het andere Nano-board. Zou dit gecombineerde systeem zichzelf kunnen programmeren met behulp van ChatGPT? Volg de stap-voor-stap instructies en leer hoe de boards communiceren en hoe de ChatGPT API werkt.

Espressif's ESP32-microcontroller is door zijn veelzijdigheid en robuustheid populair geworden voor toepassing in embedded systemen en het Internet of Things (IoT). Met zijn geïntegreerde WiFi-interface en rekenkracht opent hij een wereld van mogelijkheden voor het bouwen van slimme connected systemen. In dit artikel onderzoeken we een demonstratieproject dat de ChatGPT-API, een AI-taalmodel gemaakt door OpenAI, gebruikt om twee Arduino Nano ESP32-boards naar nieuwe hoogten te tillen.

Het doel van dit project is een naadloos communicatiesysteem tussen de Nano ESP32's te maken. De ene draait met de Arduino IDE en de andere met MicroPython. Door gebruik te maken van de ChatGPT-API, kunnen deze kaarten gezellige interactieve gesprekken aangaan. Stel je eens voor wat er allemaal mogelijk zou zijn als deze microcontrollers zelf op zoek gaan naar codefragmenten om tegemoet te komen aan jouw queries.

In dit artikel gaan we in op de technische aspecten van het opzetten van de hardware, het configureren van de software en het opbouwen van de communicatie tussen de Nano ESP32's. We onderzoeken de voordelen en beperkingen van zowel de Arduino IDE als het MicroPython-framework voor verschillende toepassingen. Ook geven we praktische codefragmenten, uitleg en inzichten om de interne werking van dit project te begrijpen.

Na het lezen van dit artikel zul je begrijpen hoe je een eigen Nano ESP32-communicatiesysteem kunt bouwen om de kracht van AI en IoT te combineren. Laten we ons samen in het avontuur van het bouwen van een bevattelijke, interactieve omgeving met Nano ESP32 en ChatGPT storten!

De hardware

Een aantal componenten is essentieel voor dit project. Om te beginnen kijken we naar de hardware-vereisten.

1. Arduino Nano ESP32-modules (x2)

Het hart van ons project wordt gevormd door de ESP32-microcontroller. Voor de communicatie boards zijn gemakkelijk verkrijgbaar en bieden allerlei mogelijkheden, zoals WiFi, veel rekenkracht en GPIO-pinnen voor het aansluiten van externe componenten.

2. USB-C-kabel (x2)

De USB-C-kabels dienen voor het voeden en programmeren van de Nano ESP32-boards. Deze kabels vormen de verbinding tussen de boards en onze computer voor de overdracht van programma's en data.

3. WiFi-netwerk

Een stabiel WiFi-netwerk is essentieel voor de communicatie tussen de twee boards en de ChatGPT-API. Zorg voor een betrouwbaar WiFi-netwerk met internettoegang. Hiermee kan de eerste Nano ESP32, (we noemen hem "Nano ESP32-1") communiceren met de ChatGPT-API.

Nu we de noodzakelijke hardwarecomponenten hebben, komt de software en het programmeren van het project aan de beurt.

Software en programmeren

Voor het communicatiesysteem en het programmeren hebben we de volgende softwaretools nodig:

1. Arduino IDE

De Arduino Integrated Development Environment (IDE) is de meest gebruikte IDE voor het programmeren van Arduino-microcontrollers. De nieuwe IDE 2.0 heeft een wat gebruikersvriendelijker interface, een vereenvoudigde programmeertaal en een grote bibliotheek van voorgebakken functies die het gemakkelijker maakt om met de Nano ESP32-boards te werken.

2. MicroPython-firmware

MicroPython is een lichtgewicht versie van de Python-programmeertaal die is geoptimaliseerd voor microcontrollers. Deze levert een meer flexibele en interactieve programmeerervaring dan de Arduino IDE. Om de tweede Nano ESP32 ("ESP32-2") te programmeren met MicroPython, moeten we de MicroPython-firmware erop installeren. In **figuur 1** zie je de beide boards die met elkaar verbonden zijn. Het linker board draait het Arduino-framework en het rechter draait MicroPython.

3. Toegang tot de ChatGPT-API

Om de Nano ESP32-1 met de ChatGPT-API te verbinden, heb je toegang tot de API nodig. OpenAI biedt verschillende manieren om toegang te krijgen tot de ChatGPT-API met API-keys of tokens. Om een API-key te maken, ga je naar [1]. Maak daar een Open AI-account aan, ga dan naar [2] en klik op *Create New Secret Key*. Je kunt ook op het profielpictogram rechts bovenaan de pagina klikken en daar *View API Keys* kiezen in het menu. Als je een API-key hebt, bewaar die dan op een veilige plaats, want je kunt hem niet nog een keer kopiëren.

4. Programmeren

We beginnen met de Nano ESP32-1 op de Arduino IDE. Voeg de voor onze code noodzakelijke bibliotheken toe. *WiFi.h* is nodig om de Nano ESP32 te verbinden met het WiFi-netwerk. *HTTPClient.h* wordt gebruikt voor het zenden van onze data (de ChatGPT-respons) naar de tweede Nano ESP32. En *ArduinoJson.h* is de bibliotheek om JSON te gebruiken op vrijwel elk board. Daarmee doen we de JSON-serialisatie en -deserialisatie, die we gebruiken voor de communicatie met de ChatGPT-API. De code voor de twee boards is ook beschikbaar op GitHub [3].

```
// Load Wi-Fi library
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
// Replace with your network credentials
const char* ssid = "WIFI_SSID";
const char* password = "WIFI_PWD";
//chatgpt api key
const char* apiKey =
    "sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
```

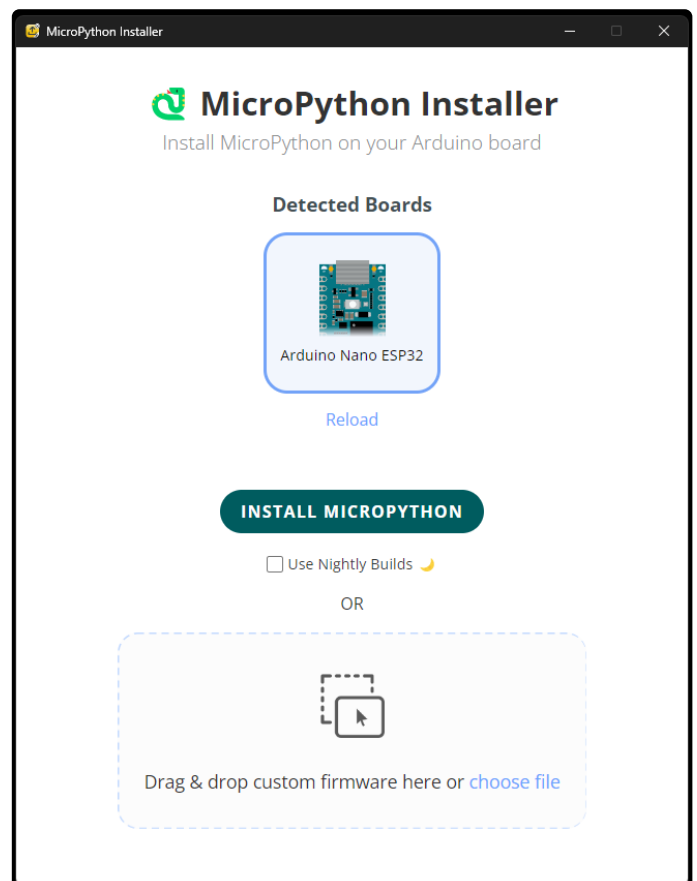
Nu voegen we het IP-adres van de tweede Nano ESP32 toe (dit IP-adres wordt getoond in de seriële monitor in de IDE als de tweede Nano ESP32 wordt verbonden met het WiFi-netwerk).

```
const char* serverIP = "192.168.1.82";
// IP address of the second Nano ESP32
const uint16_t serverPort = 80;
// Port number on the second Nano ESP32
```

Er is een speciale functie geschreven voor het verbinden en communiceren met de ChatGPT-API (**listing 1**).

Nu gaan we naar de code van de tweede Nano ESP32. Voordat we beginnen met programmeren, bespreken we eerst hoe we MicroPython uitvoeren op de Nano ESP32: we gebruiken *Arduino Lab for MicroPython*. Eerst gaan we naar [4] en downloaden het Arduino MicroPython-firmwaretool om de Arduino Nano ESP32 te flashen (**figuur 2**). Sluit de Arduino Nano ESP32 gewoon aan op de computer. Als het board is gedetecteerd, klik je op *Install MicroPython* en na enkele seconden is het board geflasht met de nieuwste firmware en kun je MicroPython gaan gebruiken op de Nano ESP32.

Nu downloaden we de nieuwste versie van Arduino Lab for MicroPython IDE van de officiële website op [5]. Na het uitpakken kun je de software starten door *Arduino Lab voor Micropython.exe* te draaien. Op de tweede Nano ESP32 is de code even kort als eenvoudig. Eerst importeren we de bibliotheken. We gebruiken de bibliotheken *network* en *socket* om verbinding te maken met het WiFi-netwerk en onze serverpoort te maken voor het ontvangen van de code of antwoorden van de Nano ESP32-1. Verder zijn de bibliotheken *machine* en *time* nodig om de GPIO's en de timer-functie voor de Nano ESP32.



Figuur 2. Het MicroPython Firmware-tool van Arduino Labs.



Listing 1. Arduino-code om requests naar ChatGPT te sturen.

```
String sendChatGPTRequest(String prompt) {
  String received_response= "";
  String apiUrl = "https://api.openai.com/v1/completions";
  String payload = "{\"prompt\": \"" + prompt +
    "\", \"max_tokens\":100, \"model\": \"text-davinci-003\"}";

  HTTPClient http;
  http.begin(apiUrl);
  http.addHeader("Content-Type", "application/json");
  http.addHeader("Authorization", "Bearer " + String(apiKey));

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

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

```
import network
import socket
import machine
import time
```

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

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

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

Dan wordt een socket-server aangemaakt voor het ontvangen van data van Nano ESP32-1 en gaat de MCU in een while-lus waar hij wacht op binnenkomende antwoorden. Als code of een antwoord wordt ontvangen, voert de Nano ESP32 de code uit en dan gaat hij weer wachten op verdere instructies (zie **listing 2**).

In het kader **Arduino IDE, MicroPython – en andere** bespreken we

de voor- en nadelen van de beide frameworks en om te begrijpen waarom het ene beter geschikt zou kunnen zijn dan het andere voor bepaalde toepassingen.

Communicatie tussen de Nano ESP32-boards

Laten we ons nu in de technische details storten, hoe de twee Nano ESP32-boards in dit project via WiFi met elkaar praten. We geven hier een gedetailleerde uitleg van het communicatieproces.

Nano ESP32-1 (Arduino IDE framework)

Nano ESP32-1, die op het Arduino IDE framework draait, maakt een WiFi-verbinding om te communiceren met externe services en apparaten. Hij maakt verbinding met een specifiek WiFi-netwerk met de gegeven parameters. Als de verbinding is gemaakt, wacht de Nano ESP32-1 op invoer van de gebruiker via de seriële monitor van de Arduino IDE.

Als de gebruiker *GPT* invoert en op Enter drukt, vraagt de Nano ESP32-1 hem om een boodschap in te tikken. Deze boodschap wordt als prompt van de Nano ESP32-1 naar de ChatGPT-API gestuurd. Dat gaat via de opgebouwde WiFi-verbinding, waarmee de Nano ESP32-1 kan communiceren met de externe API. In **figuur 3** zie je de verbindingen en communicatie tussen de twee boards en de ChatGPT-API.

De ChatGPT-API, een interface naar het AI-taalmodel, ontvangt de prompt van Nano ESP32-1. Met behulp van geavanceerde natuurlijke taalverwerkingstechnieken en machine learning-algoritmen analyseert de API de prompt om die te 'begrijpen' en een gevat antwoord of een codefragment te genereren. Het ChatGPT-model in de API

gebruikt zijn enorme hoeveelheid trainingsdata en taalherkenningsmogelijkheden om een relevante en nuttige uitvoer te genereren.

Nano ESP32-1 ontvangt het gegenereerde codefragment van de ChatGPT-API en slaat dat op. Dit codefragment is de AI-gegenereerde respons op de prompt van de gebruiker. Nano ESP32-1 geeft dan het ontvangen codefragment weer op de seriële monitor, zodat gebruikers de door de AI gegenereerde instructies kunnen beoordelen. In **figuur 4** zie je de seriële uitvoer van de beide boards, waarbij de Nano ESP32-1 de code naar Nano ESP32-2 stuurt na het ontvangen van de respons van ChatGPT.

Nano ESP32-2 (MicroPython framework)

De Nano ESP32-2 wordt aangestuurd met MicroPython. Ook de Nano ESP32-2 maakt verbinding met het WiFi-netwerk met behulp van de gegeven parameters. Zo kan hij draadloos instructies ontvangen van de Nano ESP32-1. Nano ESP32-2 zet een socket-verbinding op en luistert op een specifieke poort (typisch poort 80). Een socket is een software-eindpunt dat communicatie tussen twee apparaten over een netwerk mogelijk maakt. Door op een specifieke poort te luisteren, is Nano ESP32-2 klaar om binnenkomende data van Nano ESP32-1 te ontvangen. Als Nano ESP32-1 de gegenereerde code wil verzenden, maakt hij een verbinding met Nano ESP32-2 via de socketverbinding. Dan wordt het codefragment naar Nano ESP32-2 verzonden, waar het wordt ontvangen en verwerkt. Nano ESP32-2 verwerkt de ontvangen code door de instructies die erin staan uit te voeren. Deze instructies leggen specifieke taken vast die Nano ESP32-2 moet uitvoeren. Dat kan bijvoorbeeld de opdracht zijn om een LED voor een bepaalde tijd te laten knipperen, of een andere door de AI gegenereerde respons.

Na het uitvoeren van de instructies stuurt Nano ESP32-2 een antwoord terug naar Nano ESP32-1 via de opgebouwde socketverbinding. Dit geldt als een bevestiging dat de ontvangen code succesvol is uitgevoerd. Zo weet Nano ESP32-1 dat de gevraagde taak is uitgevoerd.

Met behulp van dit communicatieproces kunnen de twee Nano ESP32-boards effectief berichten uitwisselen en samenwerken. Nano ESP32-1 praat met de ChatGPT-API om gebruik te maken van de AI-mogelijkheden, terwijl Nano ESP32-2 functioneert als de ontvanger en uitvoerder van de door AI gegenereerde instructies.

Beperkingen van ChatGPT bij het programmeren

ChatGPT is een heel goed AI-taalmodel, maar het heeft wel beperkingen als er functionele code gegenereerd moet worden, vooral bij ingewikkelde programmeerscenario's. Bij het testen bleek dat ChatGPT in slechts 60% van de gevallen in staat was om functionele code voor een eenvoudige knipper-sketch te maken. Dat wijst erop dat het model niet altijd in staat is om nauwkeurige, werkende code te genereren.



Listing 2. Code ontvangen en uitvoeren op het tweede board (Python).

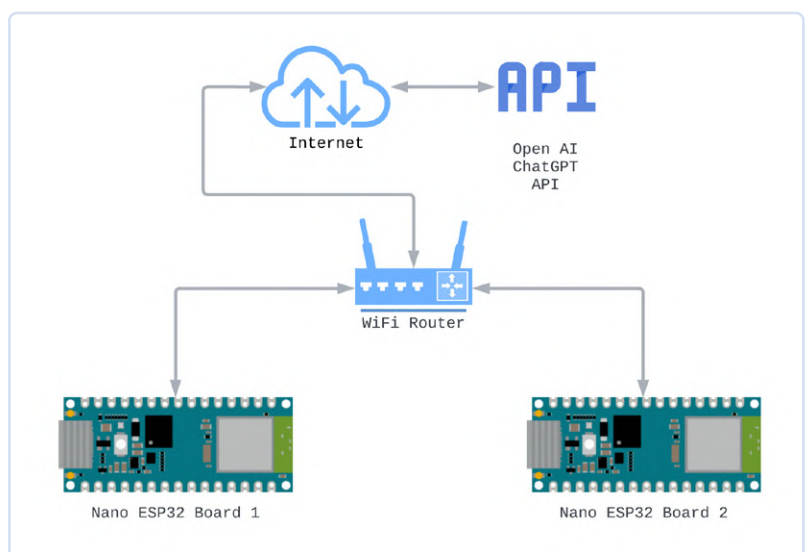
```
# Create a socket server
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('', 80))
server.listen(1)

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

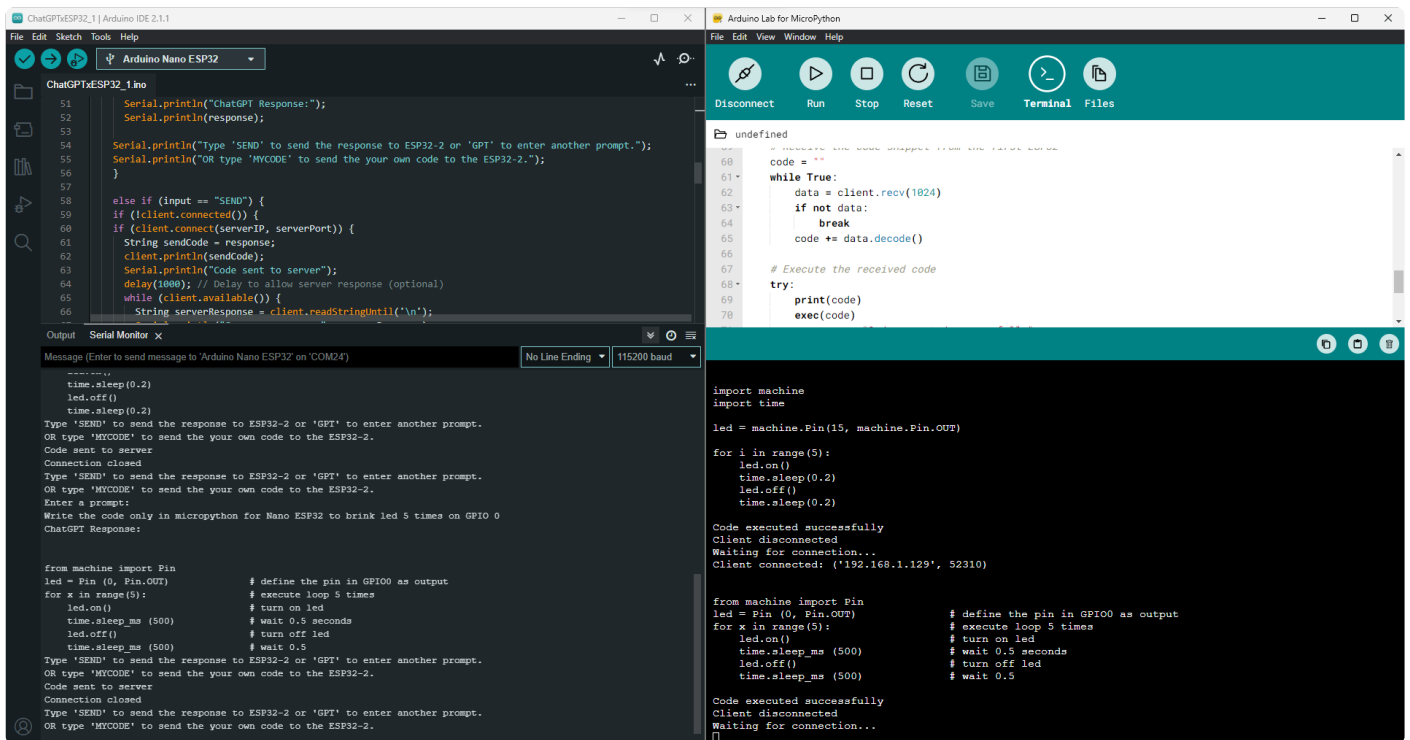
# Receive the code snippet from the first ESP32
code = ""
while True:
    data = client.recv(1024)
    if not data:
        break
    code += data.decode()

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

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



Figuur 3. Communicatie tussen de twee Arduino Nano ESP32-boards en de ChatGPT-API.



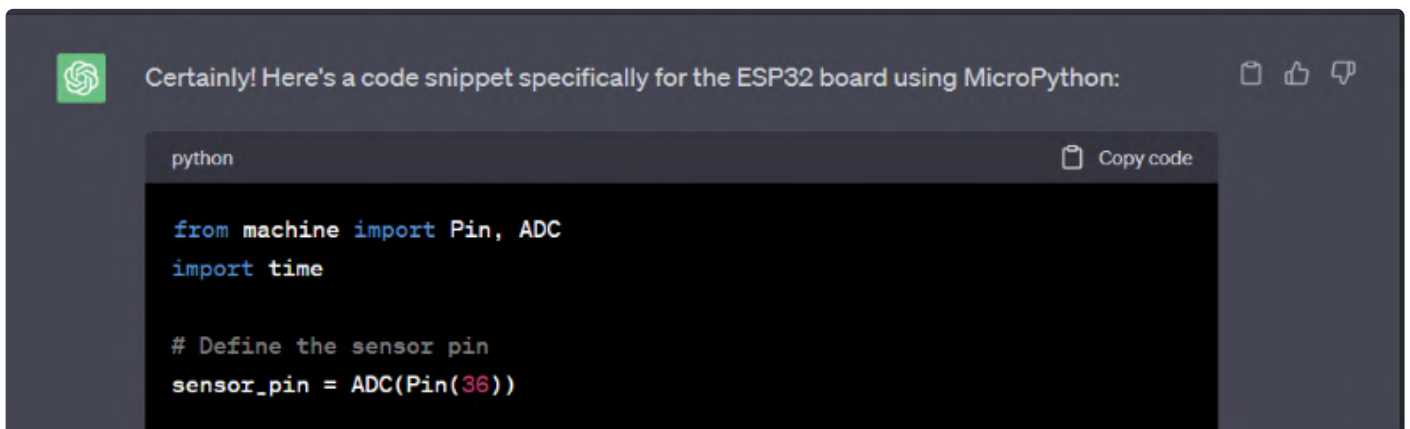
Figuur 4: Communicatie tussen de twee Nano ESP32's, met Arduino IDE (links) en met Arduino Lab for MicroPython (rechts).

Wanneer we ChatGPT om code vragen, moeten we nog met iets anders rekening houden: soms bevat de respons extra tekst voor en na de eigenlijke code, die problemen kan geven als we die één-op-één naar de tweede Nano ESP32 sturen. Die tekst heeft waarschijnlijk niet de juiste syntax en is dus niet uitvoerbaar. In **figuur 5** zie je de respons van ChatGPT toen hem gevraagd werd om code te maken voor het uitlezen van de sensorwaarde op pin 36 van de ESP32.

Als oplossing voor dit probleem geeft ons systeem gebruikers de mogelijkheid om de code zelf in te voeren en ChatGPT alleen te gebruiken als een referentie of gids bij het maken van de code. Dit geeft de gebruiker meer controle over de gegenereerde code en lost de layout- en syntaxproblemen op die ontstaan als we 100% zouden vertrouwen op de uitvoer van ChatGPT.

We moeten ons wel realiseren dat ChatGPT nuttige inzichten en suggesties kan leveren, maar dat er nog ruimte voor verbetering is bij het programmeren. Ontwikkelaars moeten voorzichtig zijn en niet puur vertrouwen op ChatGPT's antwoorden voor belangrijke of ingewikkelde programmeertaken. Gebruik van ChatGPT als referentie of voor code-optimalisatie kan het ontwikkelproces verbeteren, maar moet worden aangevuld met menselijke deskundigheid en nauwkeurige beoordeling van de code.

Als AI-taalmodellen zich blijven ontwikkelen, mogen we verwachten dat ze steeds beter worden in het genereren van betrouwbare code. Die vooruitgang zal nieuwe uitzichten openen voor het gebruik van AI bij het programmeren en nog meer samenwerking tussen mensen en machines mogelijk maken. Om optimaal te profiteren van ChatGPT,



Figuur 5. Respons van ChatGPT na een verzoek om een codefragment te schrijven.

Toepassingsscenario's

Het project van het integreren van Nano ESP32-kaarten met de ChatGPT-API opent een wereld van interessante toepassingen en use cases. Hierbij enkele opmerkelijke voorbeelden:

- **Smart Home Automation:** door gebruik te maken van de ChatGPT API kunnen gebruikers communiceren met hun smart home-systemen in natuurlijke taal. Ze kunnen de verlichting bedienen, de temperatuurinstellingen aanpassen of zelfs om aanbevelingen vragen over energiebesparende praktijken.
- **Snelle prototypes en ontwikkeling:** de combinatie van Nano ESP32-boards en door AI gegenereerde codefragmenten maakt snelle prototyping van IoT-projecten mogelijk. Gebruikers kunnen ideeën snel testen en itereren dankzij direct ontvangen codesuggesties voor verschillende functionaliteiten.
- **Educatief hulpmiddel:** het project biedt een waardevol educatief hulpmiddel voor wie leert te programmeren. Studenten kunnen deelnemen aan interactieve codeersessies met de AI, begeleiding krijgen en nieuwe programmeertechnieken leren.
- **Persoonlijke assistent en informatiebron:** de chat GPT-integratie kan worden gebruikt om een persoonlijke assistent-applicatie te ontwikkelen. Gebruikers kunnen vragen stellen, aanbevelingen zoeken of informatie verkrijgen over verschillende onderwerpen, allemaal mogelijk gemaakt door AI-gestuurde antwoorden.
- **Creatieve code-inspiratie:** het project dient als een bron van creatieve code-inspiratie. Het kan unieke en innovatieve ideeën genereren voor animaties, visualisaties of interactieve projecten, waardoor de creativiteit van ontwikkelaars wordt gestimuleerd.

moeten we het zien als een gereedschap en niet blindelings vertrouwen op zijn uitvoer. De combinatie van menselijke deskundigheid met de AI-gegenereerde antwoorden kan leiden tot nauwkeuriger en betrouwbaarder resultaten. Als we rekening houden met deze beperkingen, kunnen we ChatGPT effectief gebruiken en de potentiële risico's verminderen.

Alternatieve benaderingen

Naast de besproken aanpak in dit project zijn er alternatieve manieren en benaderingen om de Nano ESP32 te laten communiceren en te integreren met AI. We noemen hier een aantal verschillende manieren om dit project te implementeren op het Nano ESP32-platform:

1. MQTT-Protocol

Het MQTT-protocol (Message Queuing Telemetry Transport) is een lichtgewicht en efficiënt protocol voor het uitwisselen van berichten dat veel wordt gebruikt in IoT-toepassingen. In plaats van WiFi- en socket-verbindingen kunnen Nano ESP32-boards met elkaar communiceren met behulp van het MQTT-protocol. MQTT-brokers kunnen het uitwisselen van berichten tussen de boards vergemakkelijken voor naadloze communicatie en AI-integratie.

2. Direct Web Socket-verbinding

Een andere benadering is het opzetten van een directe Web Socket-verbinding tussen de Nano ESP32's. Web Socket is een communicatieprotocol dat tweerichting-communicatie over één enkele TCP-verbinding mogelijk maakt. Door Web Socket te implementeren op de Nano ESP32-boards, kunnen ze een permanente verbinding in twee richtingen opbouwen, waarmee real time-communicatie en AI-integratie mogelijk wordt.

3. ESP-NOW-protocol

ESP-NOW is een communicatieprotocol dat specifiek is ontworpen voor low power-apparaten, waarmee snelle en

betrouwbare dataoverdracht tussen Nano ESP32-boards mogelijk is. Dit protocol kan worden gebruikt om directe communicatie tussen de boards op te zetten zonder gebruik te maken van WiFi-verbindingen. Door de AI-mogelijkheden te integreren op één kaart en gebruik te maken van ESP-NOW voor de communicatie, kunnen real time-antwoorden en code efficiënt worden uitgewisseld.

4. On-Device AI-integratie

Het is ook mogelijk om AI-modellen rechtstreeks op Nano ESP32-kaarten te draaien, in plaats van te vertrouwen op externe API's. Zo kunnen bijvoorbeeld met TensorFlow Lite AI-modellen lokaal op microcontrollers worden ingezet en uitgevoerd. Door een model te trainen of te tunen voor specifieke taken, zoals het genereren van code, kunnen de Nano ESP32-boards zelf AI-gestuurde antwoorden geven zonder externe verbindingen.

Deze alternatieve benaderingen hebben verschillende voor- en nadelen afhankelijk van de specifieke projectvereisten en -beperkingen. We moeten factoren als energieverbruik, latency, complexiteit en schaalbaarheid in overweging nemen om de beste aanpak te kiezen. Door deze alternatieven te onderzoeken, kunnen ontwikkelaars de mogelijkheden van Nano ESP32's uitbreiden, AI integreren op verschillende niveaus en het communicatiesysteem aanpassen aan hun behoeften.

Het project van Nano ESP32-communicatie en AI-integratie kan op meerdere manieren worden aangepakt. Door alternatieven zoals MQTT, Web Socket, ESP-NOW, of On-Device AI-integratie te overwegen, kunnen ontwikkelaars het project aanpassen aan hun unieke eisen en profiteren van alle kracht van het Nano ESP32-platform. Dus laat je creativiteit de vrije loop, experimenteer met verschillende benaderingen en bouw innovatieve IoT-systemen die AI en Nano ESP32 op nieuwe manieren combineren.

Arduino IDE, MicroPython – en andere

Arduino IDE framework

De Arduino IDE is populair bij beginners en hobbyisten vanwege zijn eenvoud en bedieningsgemak. Hij biedt een beginnersvriendelijke programmeeromgeving met een vereenvoudigde versie van de programmeertaal C++. Hier enkele voordelen en nadelen van het gebruik van de Arduino IDE met de Nano ESP32:

Voordelen

- Gemakkelijk te leren: de Arduino IDE heeft geen steile leercurve, en is dus toegankelijk voor wie begint met programmeren of niet bekend is met microcontrollers.
- Grote community and ondersteunende bibliotheken: Arduino heeft een grote community van hobbyisten, uitgebreide online-hulpmiddelen en een grote verzameling van bibliotheken en voorbeelden, die het ontwikkelen vergemakkelijken.
- Snelle prototypes: de Arduino IDE maakt snelle prototypes mogelijk door zijn vereenvoudigde syntax en bibliotheek-ondersteuning, zodat snel resultaten zichtbaar worden.

Nadelen

- Beperkte taalmogelijkheden: de Arduino IDE heeft een vereenvoudigde versie van C++, waardoor er minder geavanceerde programmeermogelijkheden zijn dan in andere talen.
- Geheugen en prestaties: de Arduino IDE is misschien niet zo efficiënt qua geheugengebruik en prestaties vergeleken met andere frameworks.
- Minder flexibiliteit: de Arduino IDE legt bepaalde conventies en beperkingen op, die de mogelijkheden van de Nano ESP32 beperken.

MicroPython framework

MicroPython is een lichtgewicht versie van de Python-programmeertaal die is geoptimaliseerd voor microcontrollers. Het biedt een flexibele, interactieve programmeerervaring. Hier enkele voordelen en nadelen van het gebruik van MicroPython met de Nano ESP32:

Voordelen

- De taal Python: met MicroPython kunnen ontwikkelaars de krachtige taal Python gebruiken, waardoor gemakkelijker ingewikkelde code en algoritmen te schrijven zijn.
- Interactieve REPL: MicroPython heeft een Read-Eval-Print Loop-omgeving (REPL), waarmee ontwikkelaars rechtstreeks op het Nano ESP32-board de code interactief kunnen testen en ermee experimenteren.
- Efficiënt geheugengebruik: MicroPython is ontworpen om zuinig met geheugen om te gaan, wat het geschikt maakt voor apparaten met beperkte resources, zoals de Nano ESP32.

Nadelen

- Leercurve: MicroPython kan lastig zijn voor beginners die nog niet bekend zijn met de taal Python.
- Beperkte bibliotheek-ondersteuning: er komen steeds meer bibliotheken voor MicroPython beschikbaar, maar het aanbod is toch een stuk kleiner dan bij het Arduino-ecosysteem.
- Prestatie-compromis: MicroPython biedt veel flexibiliteit, maar het is een geïnterpreteerde taal. Daardoor worden programma's wat langzamer uitgevoerd dan bij gecompileerde talen zoals C++.

Conclusie: een boeiend snijvlak van AI en IoT

De integratie van Nano ESP32-boards met de ChatGPT-API vormt een fascinerend snijvlak tussen AI, IoT en programmeren. Door Nano ESP32-boards onderling te laten communiceren en te koppelen met een AI-taalmodel, ontsluiten we een wereld van mogelijkheden. Van slimme domotica tot rapid prototyping en educatieve gereedschappen, de toepassingen zijn eindeloos (zie het kader **Toepassingsscenario's**). We moeten wel steeds de beperkingen van ChatGPT in gedachten houden en voorzichtig zijn met het gebruik van AI-gegenereerde antwoorden, zeker bij het programmeren en coderen. De combinatie van menselijke deskundigheid en beoordelingsvermogen met AI-gegenereerde inhoud geeft de beste resultaten.

Met dit project kunnen gebruikers hun creativiteit benutten, nieuwe uitzichten in de IoT-ontwikkeling ontdekken en hun coderingsvaardigheden vergroten. Dus pak je Nano ESP32's, duik in de wereld van AI-gestuurde interacties en benut de mogelijkheden van deze interessante integratie! 

230485-03 (vertaling: Evelien Snel)

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de redactie van Elektor via redactie@elektor.com.

Over de auteur

Saad Imtiaz is een ontwikkelaar met ervaring in embedded systemen, mechatronische systemen en productontwikkeling. Hij heeft samengewerkt met meer dan 200 bedrijven, van startups tot globale ondernemingen, bij prototyping en ontwikkeling. Saad heeft ook gewerkt in de luchtvaartindustrie en heeft een technische startup geleid. Hij kwam in 2023 bij Elektor aan boord en stimuleert de projectontwikkeling in zowel software als hardware.

Andere frameworks

Eén van de opmerkelijke voordelen van de Nano ESP32 is zijn flexibiliteit: hij werkt met verschillende frameworks – naast Arduino IDE en MicroPython zijn dat onder meer PlatformIO, ESP-IDF, JavaScript (Node.js) en Lua. De Nano ESP32 is compatibel met verschillende andere frameworks, wat bijdraagt aan de veelzijdigheid. Door die flexibiliteit kunnen ontwikkelaars de ontwikkelomgeving kiezen die het best past bij hun projecteisen en hun kennis van programmeertalen.

Dit zijn een paar andere frameworks die je met de Nano ESP32 kunt gebruiken:

Mongoose OS: Mongoose OS is een open source-besturingssysteem voor microcontrollers, zoals de Nano ESP32. Het is een platform-onafhankelijke omgeving met ingebouwde cloud-verbinding en ondersteunt verschillende programmeertalen zoals JavaScript, C, en C++. Mongoose OS vereenvoudigt het ontwikkelproces met een gebruikersvriendelijke commandoregel-interface, rijke bibliotheken en mogelijkheden voor beheer op afstand.

Zephyr RTOS: Zephyr RTOS is een real time-besturingssysteem ontworpen voor systemen met beperkte resources, zoals de Nano ESP32. Het heeft een schaalbare, modulaire architectuur die het geschikt maakt voor projecten die multitasking, real time-verwerking en geavanceerd apparaatmanagement nodig hebben. Zephyr's ruime hardware-ondersteuning en uitgebreide verzameling van drivers en bibliotheken helpen ontwikkelaars om met gemak ingewikkelde IoT-toepassingen te bouwen.

ESPHome: ESPHome is ontworpen voor de ESP32 en de ESP8266. Het is een veelzijdig IoT-framework met een modulaire architectuur, vergelijkbaar met Zephyr RTOS. Het biedt een brede hardware-ondersteuning, veel drivers en bibliotheken voor vereenvoudigde ontwikkeling voor systemen met beperkte resources. Dit platform ondersteunt real time-verwerking, multitasking en geavanceerd apparaatbeheer, wat het een uitstekende keus maakt voor domotica en IoT-projecten.

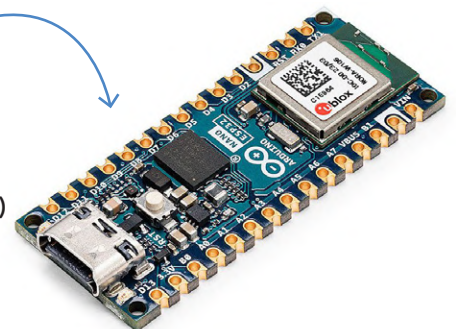
De keuze van het juiste framework wordt bepaald door de projecteisen, bekendheid met de programmeertaal, beschikbare bibliotheken, ondersteuning door een community en het gewenste niveau van besturing van de hardware en performance. Elk framework heeft zijn eigen voordelen en compromissen, zodat ontwikkelaars het meest geschikte kunnen kiezen voor hun specifieke toepassing.

De compatibiliteit van de Nano ESP32 met verschillende frameworks geeft ontwikkelaars de vrijheid om verschillende programmeertalen, ecosystemen en ontwikkelmethoden te onderzoeken. Met die flexibiliteit kunnen ze innovatieve IoT-oplossingen maken met bestaande tools en bibliotheken en de mogelijkheden van de Nano ESP32 efficiënt benutten.



Gerelateerde producten

- > **Arduino Nano ESP32**
www.elektor.nl/20562
- > **Arduino Nano ESP32 with Headers**
www.elektor.nl/20529
- > **Dogan Ibrahim and Ahmet Ibrahim, The Official ESP32 Book (E-Book) (Elektor 2017)**
www.elektor.nl/18330
- > **Günter Spinner, MicroPython for Microcontrollers (Elektor 2021)**
www.elektor.nl/19736



WEBLINKS

- [1] Open AI: <https://platform.openai.com>
- [2] Open AI - API Keys: <https://platform.openai.com/account/api-keys>
- [3] GitHub-repository van dit project: <https://github.com/elektor-labs/ChatGPTxESP32>
- [4] Arduino Labs - MicroPython Installer: <https://labs.arduino.cc/en/labs/micropython-installer>
- [5] Arduino Labs - MicroPython IDE: <https://labs.arduino.cc/en/labs/micropython>

Walkie-talkie met ESP-NOW

niet echt WiFi, niet echt Bluetooth, maar...

Clemens Valens (Elektor)

Heeft je draadloze project zowel snelle reactietijden als een groot bereik nodig? WiFi en Bluetooth zijn ongeschikt voor dergelijke toepassingen. Misschien is ESP-NOW een goed alternatief? Verbindingen worden vrijwel onmiddellijk tot stand gebracht en een bereik van enkele honderden meters is haalbaar. In dit artikel proberen we het uit in een eenvoudige walkie-talkie of draadloze intercomtoepassing.

De ESP32 van Espressif wordt vaak ingezet vanwege zijn WiFi- en Bluetooth-mogelijkheden, waar hij werkelijk topprestaties levert. WiFi en Bluetooth zijn prima protocollen voor allerlei draadloze toepassingen, maar ze hebben hun beperkingen.

Een ongemak van WiFi is de tijd die nodig is om een verbinding tot stand te brengen. Bovendien maakt WiFi directe communicatie (peer-to-peer, **figuur 1**) tussen apparaten niet mogelijk. Er is altijd een router bij betrokken. Daarom is WiFi niet echt geschikt voor eenvoudige afstandsbedieningen met een geringe latentie om een garagedeur te

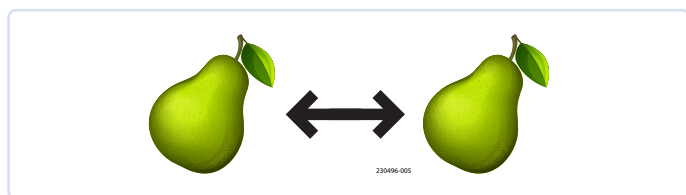
openen of een licht aan en uit te doen. Dergelijke taken vereisen een onmiddellijke reactie. Om dit te omzeilen, zijn WiFi-toepassingen vaak de hele tijd ingeschakeld en verbonden. Hierdoor verbruiken ze veel energie, zelfs als ze niet actief zijn.

Bluetooth daarentegen maakt een snelle verbinding en peer-to-peer communicatie mogelijk en is uitstekend geschikt voor afstandsbedieningen met een geringe latentie. Bluetooth is echter bedoeld voor de korte afstand met communicerende apparaten die maximaal tien meter van elkaar verwijderd zijn. Bluetooth voor lange afstanden bestaat wel, maar is nog niet algemeen beschikbaar.

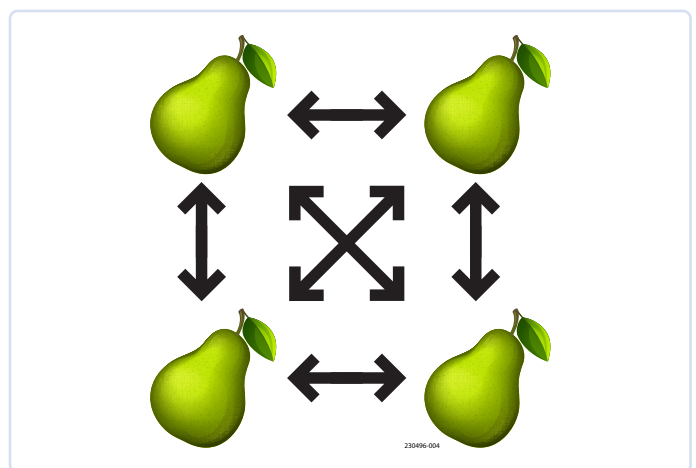
De oplossing: ESP-NOW

Het draadloze protocol ESP-NOW [1] van Espressif is een oplossing voor situaties die zowel snelle reactietijden als een groot bereik vereisen, terwijl het dezelfde frequentieband gebruikt als WiFi en Bluetooth. Het protocol combineert de voordelen van WiFi en Bluetooth. ESP-NOW is gericht op domotica en smart home. Omdat het geschikt is voor one-to-many en many-to-many topologieën (**figuur 2**), heeft het geen router, gateway of, erger nog, een cloud nodig.

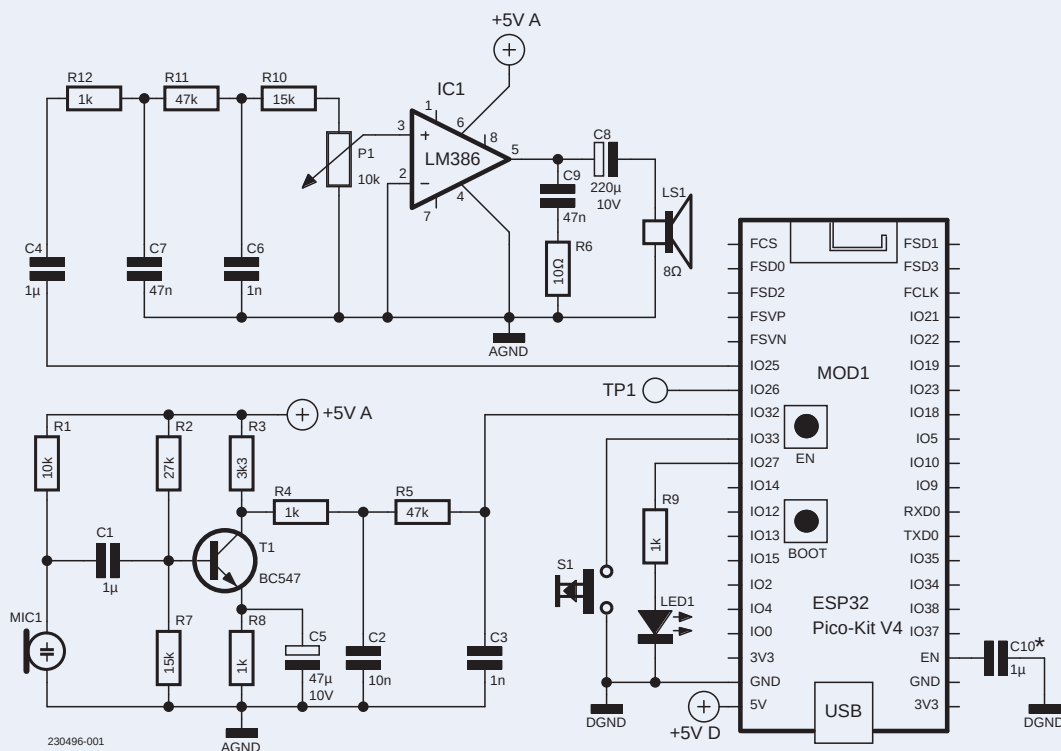
ESP-NOW implementeert geen ingewikkelde verbindingprotocollen of communicatieprotocollen op hoog niveau. Adressering is gebaseerd



Figuur 1. Peer-to-peer communicatie zoals hier getoond is niet mogelijk met WiFi; er is dan altijd een router nodig tussen de twee nodes.



Figuur 2. ESP-NOW ondersteunt many-to-many netwerken. In een dergelijk netwerk kan elk knooppunt direct met de andere knooppunten praten zonder dat er een router nodig is.



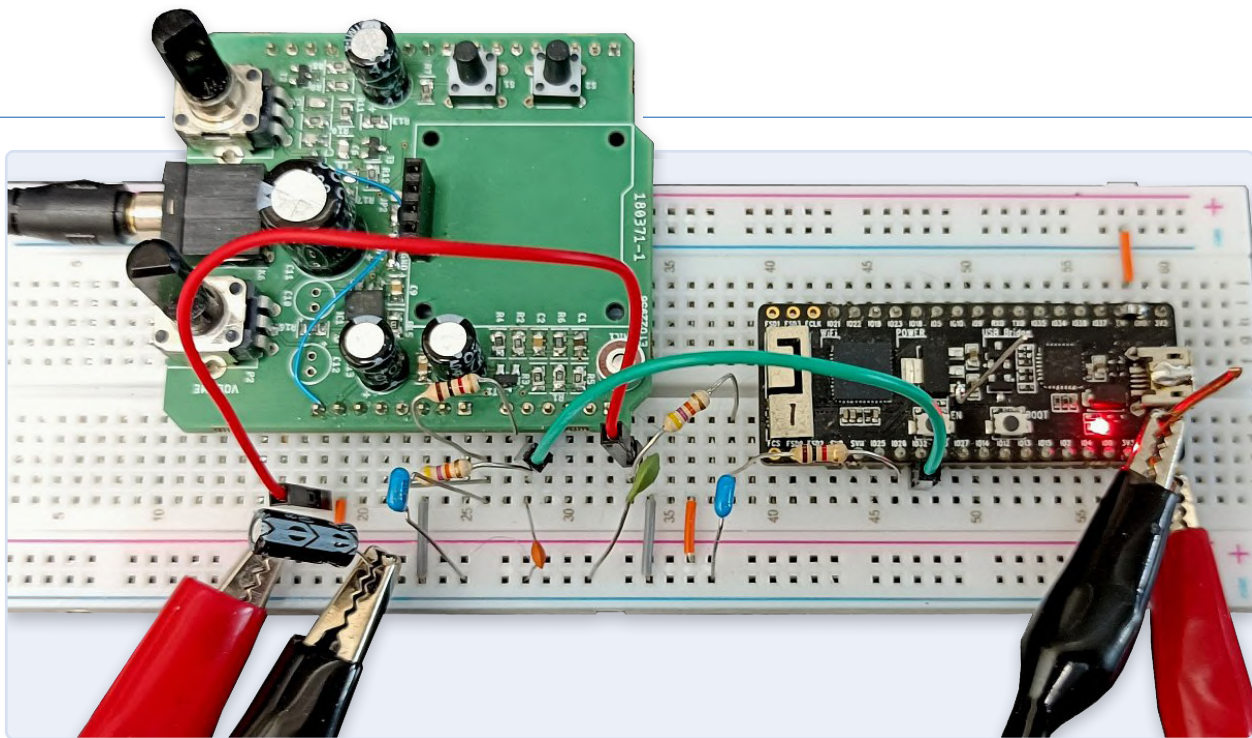
op het Ethernet MAC-adres van het knooppunt en er is pairing nodig om ze met elkaar te laten praten. Ook is het niet gegarandeerd dat gegevenspakketten in de juiste volgorde aankomen. Voor eenvoudige toepassingen met afstandsbediening is dit alles geen probleem. De gegevenssnelheid van ESP-NOW is standaard 1 Mbit/s (instelbaar) en een datapakket kan een payload van maximaal 250 bytes hebben. Samen met de header en checksum bytes enzovoort resulteert dit in een maximale pakketgrootte van 255 bytes.

Mijn doel was om een portofoonachtig apparaat of intercom te maken op basis van ESP-NOW. Een snelle blik op de specificaties van de ESP32 laat zien dat deze alles in zich heeft wat hiervoor nodig is: een analoog/digitaal-omzetter (ADC), een digitaal/analoog-omzetter (DAC), veel rekenkracht, en natuurlijk al het radiospul. In de praktijk ziet het er echter iets minder rooskleurig uit.

Een portofoon kan echter weggkomen met deze getallen als de audio-bandbreedte wordt beperkt tot de standaard telefoniebandbreedte van 3,5 kHz. Een bemonsteringsfrequentie van 8 kHz resulteert in een datasnelheid van $(8.000 / 250) \times 255 \times 8 = 65.280$ bit/s (vergeet niet dat de maximale payload 250 bytes bedraagt). Dit blijft ver onder de standaardsnelheid van 1 Mbit/s. Met deze specificaties krijgen we geen natuurgetrouwe audio, maar dat is ook niet ons doel. Verstaanbaarheid is belangrijker.

Om het eenvoudig te houden, gebruikte ik een één-transistor condensatormicrofoon-voorversterker met begrensde bandbreedte als audio-ingang en voegde ik een klassieke versterker op basis van een LM386 als audio-uitgang. Het schema is getekend in **figuur 3**. De ingangsbreedte wordt aan de onderkant beperkt door C1 en C5, die iets te klein gedimensioneerd zijn. Laagdoorlaatfilters R4/C2 en R5/C3 zorgen voor de begrenzing aan de bovenkant. Vergelijkbare laagdoorlaatfilters zijn aan de uitgang van de DAC opgenomen. Het signaal aan de 'hete' kant van P1 mag niet groter zijn dan 400 mV_{PP} .

Merk op hoe de voeding is opgesplitst in een analoog en een digitaal deel. De reden hiervoor is niet om te voorkomen dat snelle digitale schakelruis op de audio-ingang wordt ingekoppeld, maar om een klinkend geluid aan de uitgang te voorkomen. Blijkbaar produceert een taak die op de ESP32 wordt uitgevoerd periodieke stroompieken die hoorbaar kunnen worden als de schakeling niet zorgvuldig is bedraad. De beste manier die ik vond om dit te vermijden is door twee aparte voedingen te gebruiken (**figuur 4**). De ESP32-module moet worden behandeld als een component die een eigen voeding nodig heeft (zoals de LM386), en niet als een module die ook de rest van de schakeling van stroom kan voorzien; in deze toepassing gaat dat niet.



Figuur 4. Een proof-of-concept gebouwd op een breadboard met een ESP32-PICO-KIT en een licht aangepast Elektor Snore Shield [2] voor de ingangs- en uitgangsversterkers. Let op de twee paar krokodillenklemmen die zorgen voor de afzonderlijke analoge en digitale voedingen.

Houd er rekening mee dat de LM386 een voedingsspanning bereik heeft van 4...12 V.

C10 is optioneel en is alleen nodig in enkele zeldzame gevallen van vroege ESP32-modules die niet goed opstarten als ze niet zijn aangesloten op een computer (of iets dergelijks). Toevallig heb ik een paar van deze 'oude' modules en daarom heb ik C10 in mijn ontwerp opgenomen.

De software

Ik heb het programma voor de portofoon gebaseerd op het *ESPNow_Basic_Master* voorbeeld dat wordt geleverd met het Arduino ESP32 boards-package van Espressif. Nadat ik het had aangepast aan mijn behoeften, heb ik er audio-sampling en -afspelen aan toegevoegd. Er zijn een paar dingen die je misschien wilt weten over het programma. Het bemonsteren en afspelen van audio wordt geregeld door een timer-interrupt die op 8 kHz draait. Voor de bemonstering zet de interrupt-service-routine (ISR) van de bemonsteringstimer alleen een vlag om aan te geven dat er een nieuw sample moet worden opgehaald. De `loop()`-functie kijkt naar deze vlag en onderneemt de noodzakelijke acties. De reden hiervoor is dat de ADC niet in een ISR gelezen moet worden als de ADC-API van Espressif wordt gebruikt. De functie `adc1_get_raw()` die hier wordt gebruikt, roept allerlei andere functies aan die dingen kunnen doen waarover je geen controle hebt. Omdat de ESP32-software in een multitasking-omgeving draait, is het belangrijk om de veiligheid van threads te garanderen. Wanneer je Arduino gebruikt voor ESP32-programmering, wordt veel hiervan al voor je afgehandeld, maar als je van plan bent om mijn programma te porten naar de ESP-IDF, moet je misschien wat voorzichtiger zijn.

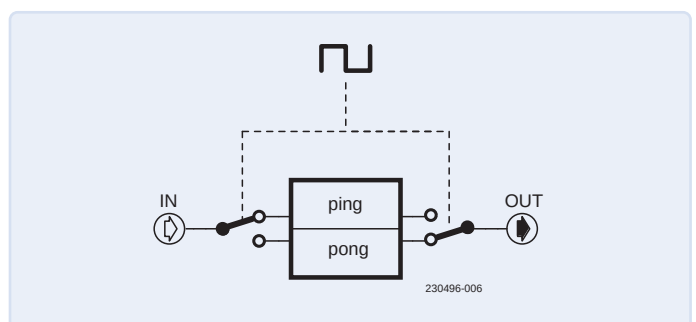
Het afspelen van audio is eenvoudig omdat de sample rate timer-ISR gewoon een sample naar de DAC schrijft als er een beschikbaar is. Zo niet, dan wordt de DAC-uitgang vastgezet op de helft van de ESP32-voedingsspanning (1,65 V). Het enige waar je hier op moet letten is dat er een zogenaamde pingpong-buffer wordt gebruikt voor het stroomlijnen van digitale audio-ontvangst (figuur 5). Zo'n buffer bestaat uit twee afzonderlijke buffers, waarvan de ene wordt gevuld terwijl de andere wordt gelezen. Hierdoor is overlapping mogelijk.

In theorie zou dit niet mogen gebeuren omdat zender en ontvanger dezelfde sample rate en timinglogica gebruiken, maar in werkelijkheid gebeurt dit wel vanwege timingtoleranties. Een pingpong- of dubbele buffer helpt om vervelende klikken tijdens het afspelen te voorkomen. Merk op dat het niet in de juiste volgorde ontvangen van datapakketjes niet wordt behandeld.

Pairing

De portofoon-firmware is een master/slave-systeem. De master werkt in WiFi-station-modus (STA), terwijl een slave in access-point-modus (AP) werkt. De master maakt onmiddellijk verbinding met een slave als hij er een detecteert en kan meteen beginnen met het verzenden van gegevens. Wanneer de master echter verbinding maakt met de slave, wordt hierdoor de slave niet ook met de master verbonden. De slave kan geen gegevens naar de master sturen en tweewegbedrijf is niet mogelijk (dat is me tenminsten niet gelukt; als je het beter weet, laat het me dan weten).

Een manier om de slaaf verbinding met de master te laten maken, is door de callback voor gegevensontvangst te gebruiken. Wanneer data wordt ontvangen, wordt het adres van de afzender samen met de data doorgegeven aan deze functie. Daarom kan de slave,



Figuur 5. Dubbele of pingpong-buffering helpt om onderbrekingen in een gegevensstroom te vermijden.



zodra hij iets ontvangt, verbinding maken met de afzender van de data. Hiervoor heb ik dezelfde functies en procedure gebruikt als die de master gebruikt om verbinding te maken met de slave. Er is echter een subtiliteit die niet erg goed (of helemaal niet) gedocumenteerd is: de slave moet zijn WiFi-interfaceveld instellen op `ESP_IF_WIFI_AP`, anders werkt het niet. Dit veld staat standaard op `ESP_IF_WIFI_STA` zoals de master nodig heeft, dus het programma (of de programmeur) hoeft het niet expliciet in te stellen. Als gevolg hiervan verschijnt het veld nergens in de voorbeeldprogramma's, zodat de gebruiker niet weet dat het bestaat.

Push-to-talk

Als ESP-NOW continu streamt, wordt de MCU behoorlijk heet. In de portofoon-toepassing is er geen reden om continu te streamen, en daarom heb ik een push-to-talk-knop (S1, ook bekend als PTT-knop) toegevoegd. Houd deze knop ingedrukt terwijl je praat. Als de zender gekoppeld is met de ontvanger, gaat de LED branden. Aan de kant van de ontvanger gaat de LED ook branden om aan te geven dat er een oproep binnenkomt. Om audiofeedback te voorkomen, wordt de audio-uitgang aan de kant van de zender gedempt als de PTT-knop wordt ingedrukt. Hoewel de communicatie in principe full-duplex is, moeten de twee peers dus niet proberen tegelijkertijd te praten. Dit is een mooie gelegenheid om 'roger' en 'over' in je zinnen te verwerken.

Eén universeel programma

Het programma bestaat uit één Arduino .ino-bestand ('sketch'). Naast het Espressif ESP32 boards package zijn er geen andere bibliotheken nodig. De portofoon heeft een master- en een slave-apparaat nodig. Om het programma te compileren voor een master, moet je regel 12 uitcommentariëren, dat is de regel waarin `NODE_TYPE_SLAVE` staat. Voor een slave moet deze macro worden gedefinieerd. Je kunt ook enkele andere instellingen aanpassen als je dat wilt. Het is ook mogelijk om te compileren zonder ondersteuning voor audio-invoer (`AUDIO_SOURCE`) en/of -uitvoer (`AUDIO_SINK`). Dit is praktisch voor debugging of voor een toepassing die alleen eenrichtingscommunicatie nodig heeft. De broncode kan worden gedownload van [3].

Betere geluidskwaliteit?

Het zou niet al te ingewikkeld moeten zijn om audiogegevens van hoge kwaliteit over ESP-NOW te streamen als je, in plaats van de eenvoudige microfoonversterker en de ingebouwde ADC en DAC van de ESP32 te gebruiken, overschakelt op I²S. Dit maakt de schakeling en het programma iets complexer, maar zou het – in theorie althans – mogelijk maken om 16bit-audiogegevens te streamen met een bemonsteringsfrequentie van 48 kHz. Er moet echter wel goed worden omgegaan met de mogelijke ontvangst van pakketten die niet in de juiste volgorde zijn. Maar is Bluetooth daar niet voor ontworpen?

Bereiktest

Om te zien of ESP-NOW communicatie over lange afstanden mogelijk maakt, heb ik een eenvoudig programma geschreven dat één keer per seconde een 'ping' naar de slave stuurt. De slave was niets meer dan een ESP32-PICO-KIT met een LED aangesloten op GPIO27, gevoed door een USB-powerbank. Telkens als er een ping wordt ontvangen, licht de LED kort (100 ms) op.

Met de zender buiten geplaatst op 1 m boven de grond, haalde ik een line-of-sight (LOS) communicatieafstand van ongeveer 150 m. Op deze afstand werd de ontvangst onderbroken en moest de slave hoog gehouden worden (ongeveer 2 m boven de grond). Dit kan waarschijnlijk worden verbeterd door de twee peers zorgvuldig te positioneren (en te ontwerpen). 

230496-03

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via clemens.valens@elektor.com of naar de redactie van Elektor via redactie@elektor.com.

Over de auteur

Na een carrière in de maritieme en industriële elektronica begon Clemens Valens in 2008 bij Elektor als hoofdredacteur van Elektor Frankrijk. Hij heeft sindsdien verschillende functies bekleed en is onlangs overgestapt naar de afdeling productontwikkeling. Zijn belangrijkste interesses zijn signaalverwerking en het genereren van geluid.



Gerelateerde producten

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



WEBLINKS

- [1] Meer over ESP-NOW: <https://espressif.com/en/solutions/low-power-solutions/esp-now>
- [2] Het Elektor Snore Shield: <http://www.elektormagazine.nl/magazine/elektor-72/42288>
- [3] Downloads voor dit artikel: <http://www.elektormagazine.nl/230496-03>

Van idee tot schakeling met de ESP32-S3

zo bouw je een prototype met Espressif-chips

Liu Jing Hui, Espressif

Veel producten en apparaten – vooral unikaten en kleine aantallen – waarin Espressif-chips verwerkt zijn, worden in eerste instantie op development boards gebouwd. Daarna migreren ontwikkelaars hun ontwerp naar gebruik van een module of een kale chip als het in grotere aantallen geproduceerd moeten worden of als het product er gewoon beter uit moet zien. Er is een aantal zaken waarop je bij die migratie moet letten. In dit artikel behandelen we de meest voorkomende valkuilen.

We nemen de ESP32-S3 als voorbeeld voor het ontwerpen met Espressif-chips. Deze chip bevat een geïntegreerde energiezuinige, 2,4 GHz WiFi + Bluetooth LE (5) System-on-Chip (SoC) oplossing. Hij heeft twee snelle CPU-kernen, extern uitbreidbaar RAM-geheugen en een veelheid aan interfaces die een breed scala aan toepassingen ondersteunen. Belangrijk: de SoC heeft ook geïntegreerde kalibratie-hardware waarmee een kleine afwijking in de radioconfiguratie gecompenseerd kan worden. Dit maakt het relatief eenvoudig om je ontwerp werkend te krijgen en maakt gespecialiseerde testapparatuur grotendeels overbodig. De ESP32-S3 is een ideale keuze voor een breed scala aan toepassingen op het gebied van kunstmatige intelligentie (AI) en het AI of Things (AIoT). De ESP32-S3 is als kale chip verkrijgbaar, maar ook als module (zoals de ESP32-S3-WROOM-1) die de ESP32-S3 samen met het benodigde kristal, flash-geheugen, hoogfrequent-schakeling en naar keuze een printantenne of een aansluiting voor een externe antenne. We gaan in dit artikel in op het implementeren van de kale chip en van de module.

Als je deze chip in een ontwerp wilt toepassen, zijn er vier belangrijke zaken waar je op moet letten:

- › ontwerp van de schakeling en tekenen van het schema;
- › print-layout;
- › RF- en kristal-afregeling;
- › firmware-download en foutzoeken.

Ontwerp van de schakeling en tekenen van het schema

Om te beginnen is het raadzaam om de documentatie van de te gebruiken chip of module goed te bestuderen. We hebben diverse documen-

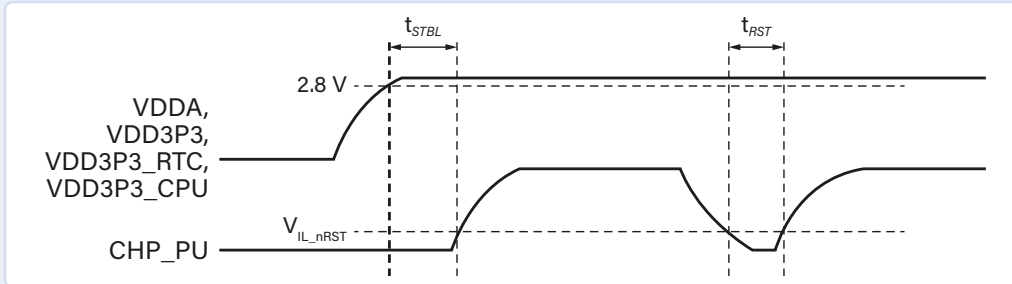
ten online staan op [1]. Onder andere de datasheet, de Hardware Design Guidelines en de Module Reference Designs zijn beschikbaar en laten zien wat nodig is om de chip optimaal te laten presteren.

Voor wie met de chip aan de slag gaat tonen we een basisconfiguratie voor de ESP32-S3 chip met alle benodigde onderdelen in **figuur 1**. Let op dat in tegenstelling tot chips zonder geïntegreerde radio een goede ont koppeling hier van wezenlijk belang is. We specificeren het aantal en de waarden van de ont koppelcondensatoren aan elke voedingspin; het is van belang deze te gebruiken en geen condensatoren weg te laten. Met name aan pin 2 en 3 moet het CLC-filternetwerk (C8, L1 en C9 in het schema) gebruikt worden. Op de meeste Espressif-chips zijn specifieke voedingspinnen toegewezen aan het RF-gedeelte, waar altijd een CLC-of CCL-filter gebruikt moet worden om harmonischen te onderdrukken.

Een ander punt van aandacht met betrekking tot de voeding van de ESP32-S3 is dat die tenminste 500 mA moet kunnen leveren. Dit geldt in principe voor alle Espressif-chips die tot nu toe zijn uitgebracht. (De enige uitzondering is de ESP32-H2, die tevreden is met 350 mA.) Een voeding die onvoldoende stroom levert zal vaak leiden tot brownout reset- en andere vervelende verschijnselen, die meestal zullen optreden bij het initialiseren van WiFi-verbindingen.

De ESP32-S3 heeft een kristal nodig als bron van het kloksignaal voor het hele systeem. Gebruik een spoel in serie met de XTAL_P-aansluiting, ook hier weer om harmonischen te onderdrukken. De waarden van ont koppelcondensatoren C1 en C4 zijn afhankelijk van het kristal, maar ook van de parasitaire impedantie van printsporen en soldeer-eilanden. Adafruit heeft een goede handleiding hoe de beginwaarden van deze onderdelen bepaald kunnen worden [2] maar voor de optimale waarde en certificatie zul je moeten meten en afregelen als

Timing-parameters voor power-up en reset



Visualisatie van power-up en reset timing (bron: [5]).

Parameter	Beschrijving	Min (µs)
t_{STBL}	Tijd om de 3,3V-rails stabiel te laten worden voor CHIP_PU hoog mag worden om de chip te activeren	50
t_{RST}	Tijd dat de spanning op CHIP_PU lager dan VIL_nRST moet zijn om de chip te resetten	50

Maar waarschijnlijk wil je nog verbindingen maken met andere chips, sensoren of actoren. Welke GPIO-pinnen kun je daarvoor gebruiken? Het goede nieuws is dat alle Espressif-chips een zogenaamde GPIO-matrix hebben. Deze matrix maakt het eenvoudig om zaken aan te sluiten op de ESP32-S3 omdat hij toestaat om vrijwel elke GPIO-pin met vrijwel elk signaal van de interne periferie (zoals I²C, SPI, SDIO enzovoort) te koppelen. Uiteraard zijn er nog wel een paar periferie-aansluitingen die vaste GPIO-pinnen gebruiken zoals de ADC (analoog/digitaal-converter). Raadpleeg de datasheet van de chip voor meer informatie. Algemeen geldt: lees de technische documentatie eerst – het is beter om bekende informatie nog eens te lezen dan een print opnieuw te moeten ontwerpen vanwege een fout die voorkomen had kunnen worden.

De print-layout

Als je een chip gebruikt, zou in dit stadium een perfect uitgewerkt schema beschikbaar moeten zijn, klaar om een layout te maken. Bij WiFi- en Bluetooth-apparaten is de print-layout essentieel voor goede

RF-prestaties. Zelfs als de Espressif-chip alleen als microcontroller gebruikt wordt, zonder toevoeging van de antenne of gebruikmaking van de radio, zullen de onderstaande richtlijnen resulteren in een stabielere en betrouwbaardere geheel. Alle details zijn te vinden in de Hardware Design Guidelines; dit artikel beschrijft alleen de belangrijke punten in algemene termen.

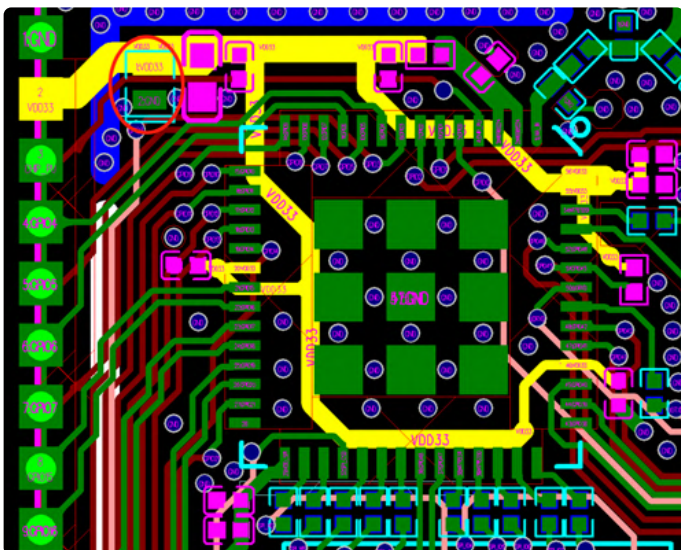
Als eerste de verschillende lagen van de print. Aanbevolen worden vier of meer lagen omdat we daarmee een groot en ononderbroken aardvlak kunnen creëren als referentiemassa voor de chip. Als je werkelijk wilt beperken tot twee lagen, zorg dan in elk geval voor goede aardvlakken!

Een tweede belangrijk punt is het routeren van de voedingssporen. We adviseren deze in een ster-configuratie op een van de binnenlagen te routeren. De ontkoppelcondensatoren en CLC-filters moeten daarbij zo dicht mogelijk bij de betreffende voedingsaansluitingen geplaatst worden (**figuur 2**).

Als derde punt moet aandacht worden besteed aan de RF-sporen. Informeer de printproducent dat een geverifieerde 50Ω-impedantie noodzakelijk is voor alle hoogfrequent-sporen. Dit begint al bij de print-layout, waar je verondersteld wordt de juiste spoorbreedte en de afstand tot het aardvlak te berekenen op basis van de opbouw en volgorde van de lagen. De producent kan hier eventueel nog iets aanpassen. Als SMD-onderdelen in het 0201-formaat gebruikt worden, moet je een 'stub' in het ontwerp van het RF-aanpasnetwerk opnemen, dicht bij de chip.

Vervolgens het kristal. De serie-zelfinductie moet zo dicht mogelijk bij de ESP32-S3 chip geplaatst worden en de twee lastcondensatoren moeten aan weerszijden van het kristal komen (**figuur 3**). Een ruim bemeten aardvlak en een fors aantal via's zijn belangrijk rond het kristal. Een laatste punt van aandacht zijn de flash- en UART-sporen, maar ook de sporen naar de andere GPIO-aansluitingen. Deze sporen kunnen hoogfrequente harmonischen genereren en daarom adviseren we deze zo veel mogelijk op de binnenlagen te routeren en ze zoveel mogelijk met massavlakken of -sporen te omgeven.

Bij gebruik van een module is het van belang die zo te plaatsen dat optimale RF-prestaties worden verkregen. De volgende richtlijnen zijn van toepassing op modules met een printantenne; bij gebruik van een externe antenne is er meer vrijheid bij de plaatsing van de module op de print.



Figuur 2. Elke ontkoppelcondensator moet zo dicht mogelijk bij de betreffende voedingspin geplaatst worden.

Zoals te zien in **figuur 4** adviseren we de modules op een hoek van de dragerprint te plaatsen. Om precies te zijn is de beste plaats zodanig dat het voedingspunt van de antenne zo dicht mogelijk bij de rand komt. (Let op dat niet alle modules het voedingspunt aan dezelfde kant hebben, raadpleeg de datasheet voordat je met de print-layout begint.) Verder adviseren we om de antenne in zijn geheel buiten de print te positioneren. Als dat onmogelijk is, zorg dan in elk geval voor 15 mm vrije ruimte rond de antenne: dat betekent dat in dat gebied geen sporen, kopervlakken, via's of componenten geplaatst mogen worden (**figuur 5**).

De meeste aansluitingen van onze Wroom-modules zijn goed te solderen met een standaard soldeerbout, maar er is een aardvlak aan de onderzijde dat niet toegankelijk is voor handmatig solderen. Als je de mogelijkheid hebt (bijvoorbeeld een reflow-oven of een hetelucht rework-station), adviseren we deze aansluiting te verbinden met een onderliggend aardvlak omdat dat helpt bij het afvoeren van warmte en daarnaast een goede aarding oplevert voor de radio, maar in de meeste omstandigheden zal de module ook zonder deze verbinding functioneren.

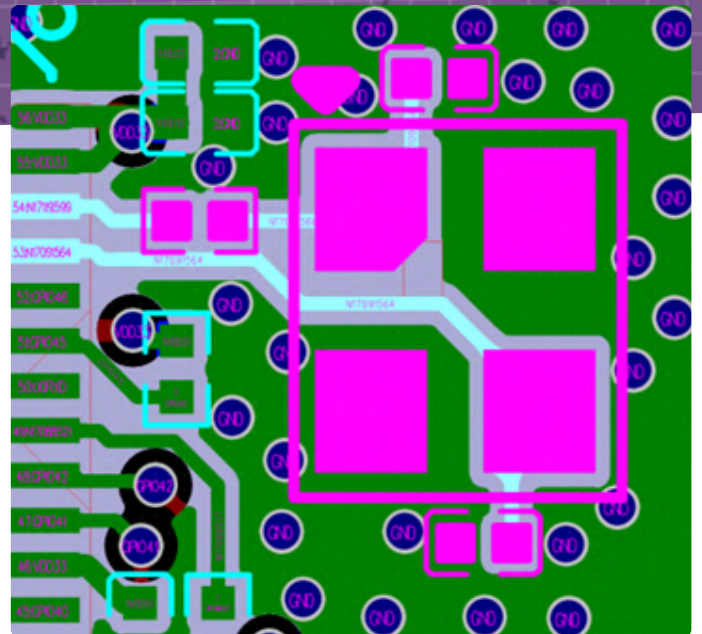
RF- en kristal-afregeling

Bij gebruik van een chip: er vanuit gaande dat alle bovenstaande punten zijn opgevolgd en een functionerende print beschikbaar is, zijn voor optimale betrouwbaarheid en om aan de EMC-richtlijnen te voldoen helaas professionele instrumenten nodig voor het afregelen van de RF-aanpassing en de lastcondensatoren bij het kristal. Als je toegang hebt tot dergelijke apparatuur (of als goedkopere producten zoals MicroVNA goed genoeg worden voor deze metingen) leggen we de procedure hieronder uit.

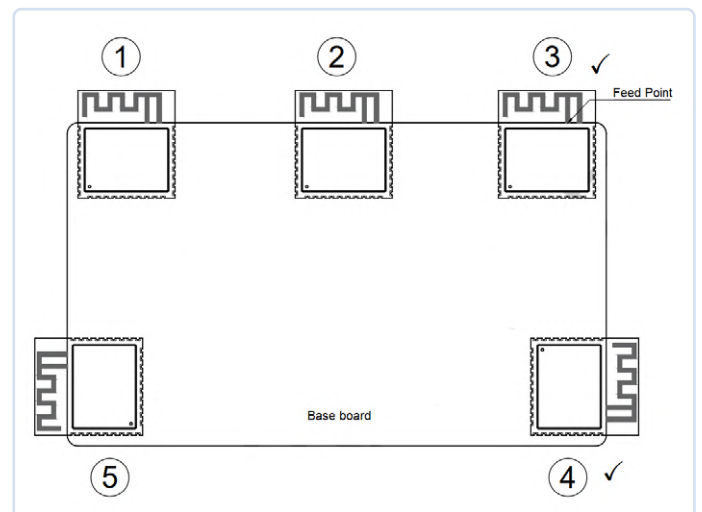
Je hebt de lastcondensatoren waarschijnlijk gekozen op basis van de geschatte parasitaire capaciteit van de printsporen. Nu is het tijd om de waarde van deze onderdelen exact te bepalen: omdat de 2,4 GHz WiFi-frequentie wordt afgeleid van dit kristal zorgt optimalisatie voor een groter bereik en minder harmonischen. Voor deze afregeling gebruiken we het ESP RF Test Tool , dat je kunt downloaden van de website [3].

1. Selecteer TX tone mode in het Certificatie en Test Tool.
2. Bekijk het 2,4GHz-sigitaal met een radiocommunicatie- of spectrum-analyzer en demoduleer het signaal om de frequentie-offset te bepalen.
3. Regel de frequentie-offset af binnen ± 10 ppm (aanbevolen) door de lastcondensatoren aan te passen.
 - Als de frequentie-offset positief is, wil dat zeggen dat de totale lastcapaciteit te klein is, en moet de waarde van de externe lastcondensatoren worden vergroot.
 - Een negatieve offset betekent een te hoge lastcapaciteit en de waarde van de externe lastcondensatoren moet worden verkleind.
 - De externe lastcondensatoren hebben normaal gesproken gelijke waarden, maar in bijzondere gevallen kunnen deze waarden enigszins verschillen.

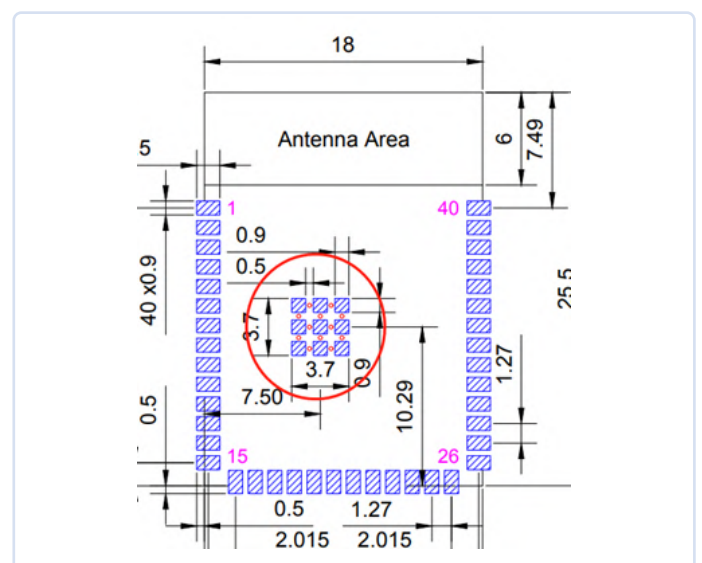
Nu de lastcapaciteit is afgeregeld, gaan we door naar het aanpasnetwerk van de antenne. We definiëren de kant bij de chip als Port 1 en de kant bij de antenne als Port 2. S11 beschrijft het deel van het ingangssignaal dat gereflecteerd wordt door Port 1, terwijl S21 het signaalverlies tussen Port 1 en Port 2 beschrijft. Bij de ESP32-S3 serie IC's geldt dat



Figuur 3. De beide lastcondensatoren moeten aan weerszijden van het kristal geplaatst worden..



Figuur 4. De juiste plaatsing van de module op een basisprint (bron: [6]).



Figuur 5. De voor de ESP32-S3-WROOM-1 aanbevolen footprint (bron: [6]).

als S11 kleiner dan of gelijk is aan -10 dB en S21 is kleiner dan of gelijk aan -35 dB als een signaal op 4,8 GHz en op 7,2 GHz aangelegd word, het aanpassingsnetwerk aan de eisen voldoet.

Verbind beide kanten van het aanpascircuit met de netwerkanalyzer (Zie het kader **Aanpassingsnetwerk**), en bepaal de reflectieparameter S11 en de transmissieparameter S21. Wijzig de waarden van de onderdelen in de schakeling net zo lang tot zowel S11 als S21 voldoen aan de eis. Als het print strikt volgens de PCB design-richtlijnen is ontworpen en ook de antenne voldoet aan de eisen, kunnen de waarden in het tekstkader aangehouden worden om de schakeling af te stemmen. Maar wat als je een ESP32-S3 chip in je schakeling wil gebruiken maar geen toegang hebt tot de geavanceerde apparatuur om dit volgens het boekje te doen? Als je alle regels van dit artikel hebt gevolgd, zou je weg kunnen komen met de initieel aanbevolen waarden voor de lastcondensatoren en het weglaten van het CLC-netwerk. (Als je dat netwerk in de print-layout hebt opgenomen, kun je C11/C12 gewoon weglaten en voor L2 een 0Ω -weerstand kiezen. Dan kan je eventueel in een later stadium alsnog het CLC-netwerk toevoegen zonder het printontwerp te moeten aanpassen.) In onze ervaring heeft het weglaten van het CLC-netwerk geen ernstige gevolgen voor het bereik; het belangrijkste doel is om de EMV-waarde (Error Magnitude Vector) te reduceren, wat nodig is om de schakeling te kunnen certificeren. Als je een module gebruikt, is het bovenstaande allemaal niet nodig: bij zo'n module zijn alle onderdelen afgestemd op optimale prestaties.

Firmware-download en foutzoeken

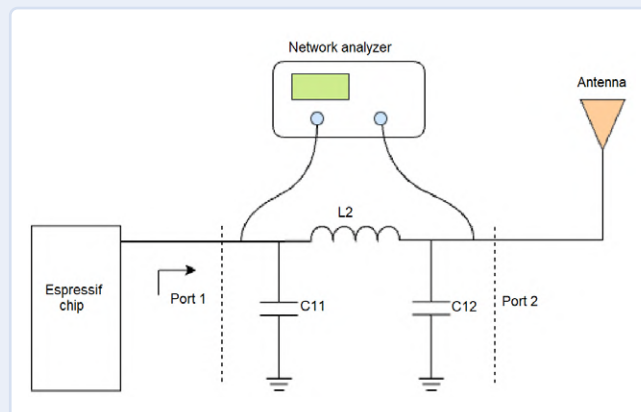
Of je nu een module of een chip gebruikt: voordat de ESP32-S3 ook maar iets zal doen, moet firmware in het flash-geheugen worden geladen (dat wordt *programmeren* genoemd, of *downloaden van de firmware*). Bij alle Espressif-chips is de procedure hetzelfde: reset de chip in de download-modus – dan downloaden – nogmaals resetten en de chip opstarten in *SPI Boot*-modus.

Espressif-chips kunnen op twee manieren van firmware worden voorzien: alle Espressif-chips kunnen firmware via UART laden, de nieuwere chips hebben daarnaast een USB-interface waardoor firmware direct via een USB-verbinding vanuit een computer geladen kan worden. Hieronder worden de betreffende stappen voor de ESP32-S3 beschreven.

Downloaden via UART:

1. Voordat het downloaden kan beginnen, moet de chip in de *Download Boot*-modus worden gebracht door pin GPIO0 (standaard hoog via een pull-up) laag te maken en GPIO46 (standaard laag door een pull-down) open te laten of laag te maken. Configureer pin GPIO45 conform de strapping pin-tabel, of laat hem open als je een module gebruikt.
2. Verbind de chip of module met de voedingsspanning en controleer dat deze in *UART Download*-modus bevindt via de seriële poort UART0. Als het log "waiting for download" aangeeft, verkeert de chip of module in de *Download Boot*-modus.
3. Download de firmware naar het flash-geheugen via de UART. Hiervoor kunt je elke willekeurige optie gebruiken die je ontwikkelomgeving (Arduino, ESP-IDF, VSCode ...) in de aanbieding heeft, of je kunt het standalone Flash Download Tool gebruiken.
4. Nadat de firmware is geladen, maak je pin IO0 hoog (of laat hem zweven) zodat de chip of module in *SPI Boot*-modus kan opstarten.
5. Sluit de voedingsspanning weer aan en de chip zal de nieuwe firmware lezen en uitvoeren tijdens het initialiseren.

Aanpassingsnetwerk



Bron [7]

Nummer in schema	Aanbevolen waarde	Onderdeelnummer
C11	1,2...1,8 pF	GRM0335C1H1RXBA01D
L2	2,4...3,0 nH	LQP03TN2NXB02D
C12	1,8...1,2 pF	GRM0335C1H1RXBA01D

De meeste development boards kunnen zo worden geconfigureerd dat de download-modus van de ESP32-chip softwarematig in- en uitgeschakeld kan worden. Bestudeer hiervoor bijvoorbeeld het schema van de ESP32-S3-Devkit-C-1 als je dit zelf wilt implementeren [4].

Downloaden via USB:

1. Als de chip al werkende firmware heeft, kan het flashing-tool in de meeste gevallen de chip in de *USB download*-modus kunnen zetten via de USB aansluiting. In die gevallen kun je doorgaan naar de volgende stap. Zo niet, moet de chip in de *Download Boot*-modus gebracht worden door pin GPIO0 (standaard hoog door een pull-up) laag te maken en GPIO46 (standaard laag door een pull-down) laag te maken of te laten zweven. Configureer pin GPIO45 volgens de strapping pin-tabel, of laat hem zweven bij gebruik van een module..
2. Sluit de chip of module op de voedingsspanning aan en controleer of deze in *UART Download*-modus verkeert via de USB seriële poort. Als het log "waiting for download" aangeeft staat de chip of module in de *Download Boot*-modus.
3. Download de firmware in flash via de UART. Hiervoor kun je elke willekeurige optie gebruiken die je ontwikkelomgeving (Arduino, ESP-IDF, VSCode ...) te bieden heeft, of je kunt het standalone Flash Download Tool gebruiken.
4. Als de firmware is geladen en je automatisch in de USB download-modus komt, ben je klaar. Zo niet maak je IO0 hoog of laat hem zweven om ervoor te zorgen dat de chip of module in *SPI Boot*-modus komt. Voer een reset uit of schakel de voeding uit en weer in in en de chip zal de nieuwe firmware lezen en uitvoeren bij het opstarten.

Als het downloaden van firmware via de UART mislukt, controleer dan het UART0-log in een seriële terminal. Het ESP32-S3 startup-bericht zal de huidige waarde van de strapping pins melden waardoor u kunt nagaan of hier iets is misgegaan. De waarde achter **boot:** laat de strapping pin-waarde hexadecimaal zien: bit 2 = GPIO46, bit 3 = GPIO0, bit 4 = GPIO45, bit 5 = GPIO3:

ets Jun 8 2016 00:22:57

```
rst:0x1 (POWERON_RESET),boot:0x3 (DOWNLOAD_BOOT(UART0/
UART1/SDIO_REI_REO_V2))
```

Als het USB-apparaat niet herkend wordt, of als de USB-verbinding niet stabiel is, probeer dan eerst in download-modus te starten en probeer daarna de firmware te downloaden. Vergeet ook niet dat op de computer slechts één programma tegelijk een seriële (USB-) poort open mag hebben: het downloaden van de firmware terwijl bijvoorbeeld een serieel monitorprogramma dezelfde poort open heeft, zal niet werken.

Meestal lukt het prima om de chip via USB in de download-modus te zetten, maar het kan om verschillende redenen mislukken:

- USB PHY is uitgeschakeld door het programma;
- de USB poort is afwijkend geconfigureerd, bijvoorbeeld als USB host of als standaard USB-apparaat;
- de USB-GPIO's zijn afwijkend geconfigureerd, bijvoorbeeld als uitgang voor LEDC, SPI of als generieke GPIO.

In deze gevallen is het noodzakelijk om de download-modus handmatig te forceren. Daarom adviseren we om tijdens de ontwikkeling van de firmware altijd een methode achter de hand te houden (knoppen, jumpers of jumperdraden) om de strapping pinnen indien nodig te kunnen instellen om in download-modus op te starten.

En nog iets anders: soms lukt het niet om een board goed te laten opstarten (dat wil zeggen: het programma in het flash-geheugen te laten uitvoeren) zonder een handmatige reset. Dit kan voorkomen als een condensator is verbonden met een strapping pin (GPIO0 op de ESP32-S3). Zo'n condensator wordt soms gebruikt om een drukknop dendervrij te maken, maar als de voeding wordt ingeschakeld, stijgt de spanning op GPIO0 zo langzaam dat de ESP32-S3 deze als laag interpreteert bij het initialiseren.

De magie van modules

Zoals je in dit artikel kon lezen, is het heel goed mogelijk iets te maken dat 'het gewoon doet', maar het zodanig ontwerpen van een print met een ESP32-S3 dat deze optimaal functioneert en gecertificeerd kan worden is echt niet triviaal en vraagt diepgaande kennis en de juiste

Over de auteur

Liu Jinghui is na haar afstuderen als elektronica-ingenieur bij Espressif gaan werken en is betrokken geweest bij het oplossen van hardware-problemen. Ze was vijf jaar de vraagbaak bij Espressif met betrekking tot het gebruik van de hardware en heeft een uitstekend beeld van de producteigenschappen en de problemen die zich kunnen voordoen als deze producten in de praktijk worden ingezet.

tools. Daarom is het aan te raden om, als de ruimte ervoor beschikbaar is, een module te gebruiken; alle lastige 'RF-magie' is dan al door Espressif gedaan. Dat maakt de integratie van de chip in je ontwerp een stuk eenvoudiger.

Voor wie dit allemaal nog steeds te ingewikkeld vindt, heeft Espressif zoals al opgemerkt een breed scala aan development boards in de aanbieding, van eenvoudige versies waar alle GPIO-pinnen toegankelijk zijn via headers zoals de ESP32-S3-DevKitC-1, tot en met spraakgestuurde AI-alleskunnners met alles aan boord zoals de ESP32-S3-Box-3 en ESP32-S3-Eye met camera. Meestal zijn de schema's en print-layouts van deze systemen beschikbaar; dus zelfs als je ze niet in uw product wilt toepassen vormen ze een uitstekend uitgangspunt voor je eigen ontwerp. ◀

230563-03 (vertaling: Adrie Kooijman)

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via liujinghui@espressif.com of naar de redactie van Elektor via redactie@elektor.com.



Gerelateerde Producten

- **ESP32-S3-WROOM-1**
www.elektor.nl/20696

WEBLINKS

- [1] Technische documenten Espressif:
https://espressif.com/en/support/documents/technical-documents?keys=&field_type_tid%5B%5D=842
- [2] Adafruit, "Choosing the Right Crystal and Caps for your Design," 2012:
<https://blog.adafruit.com/2012/01/24/choosing-the-right-crystal-and-caps-for-your-design>
- [3] ESP RF Test Tool: <https://espressif.com/en/support/download/other-tools>
- [4] Schema ESP32-S3-DevKitC-1: https://dl.espressif.com/dl/schematics/SCH_ESP32-S3-DevKitC-1_V1.1_20221130.pdf
- [5] Datasheet ESP32 Series: https://espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
- [6] Hardware-ontwerprichtlijnen ESP32-H2 Series:
https://espressif.com/sites/default/files/documentation/esp32-h2_hardware_design_guidelines_en.pdf
- [7] Datasheet ESP32-S3-WROOM-1:
https://espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf

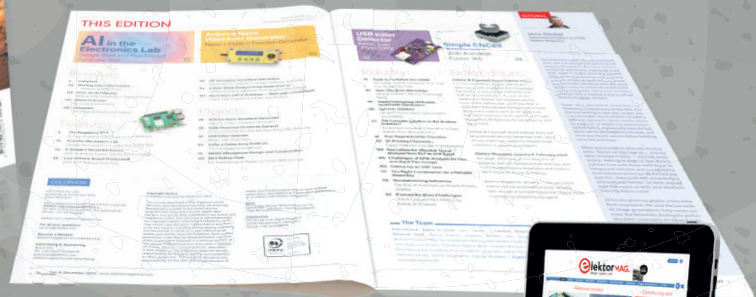
20% korting

op het eerste jaar van
uw lidmaatschap

Word lid van de Elektor Community



Neem **nu** een
lidmaatschap!



- ✓ Een compleet web-archief t/m 1980!
- ✓ 8x Elektor Magazine (Print)
- ✓ 8x digitaal (PDF)
- ✓ 10% korting in onze webshop, en exclusieve aanbiedingen
- ✓ Toegang tot meer dan 5000 Gerberfiles
- ✓ Gratis bezorging binnen Nederland en België



www.elektormagazine.nl/gold-member

Gebruik kortingscode

ESPRESSIF20



ESPRESSIF



elektor

AIoT Chip-innovatie



een vraaggesprek met Teo Swee-Ann, CEO van Espressif

Wat is ervoor nodig om de markt open te breken met een innovatieve elektronische toepassing? Teo Swee-Ann, CEO van Espressif, vertelt over zijn ervaringen. We horen meer over zijn carrière en hoe hij ertoe kwam om populaire chipsets, modules en AIoT-oplossingen te ontwerpen en te implementeren.

Elektor: Wanneer raakte je voor het eerst geïnteresseerd in elektronica? Werd je geïnspireerd door een familielid of een leraar? Of misschien werd je nieuwsgierig door reverse engineering en het knutselen aan elektronische producten?

Teo Swee-Ann: Mijn interesse in elektronica begon toen ik nog heel jong was. Toen ik opgroeide, was ik altijd gefascineerd door gadgets en machines. We deden de gebruikelijke dingen, zoals radio's en TV's uit elkaar halen om te proberen te begrijpen hoe ze werkten. Het ging toen niet zozeer om reverse engineering, maar meer om pure nieuwsgierigheid om achter de coulissen te kijken. Zoals gewoonlijk gaf het uit elkaar halen van dingen geen antwoord op vragen, maar maakte het ons nog nieuwsgieriger.

Niemand in mijn directe familie hield zich bezig met elektronica, dus mijn inspiratie kwam niet direct van een familielid. Maar ik heb wel aanleg voor software en wiskunde. Mijn universitaire scripties gingen over computationele elektromagnetica. Ik heb een paar technieken uitgevonden die handig waren om berekeningen zeer efficiënt uit te voeren – Greense functies voor gelaagde media. Het bleek dat dit plus een techniek met de naam 'partial

elements equivalent circuit' nuttig kon zijn voor de bepaling van on-chip zelfinducties, in het bijzonder om rekening te houden met substraatverliezen. Ik denk dat ik zo een baan in de chipindustrie heb gevonden. Toen ik de kans kreeg om aan het ontwerp van een schakeling te werken, pakte ik meteen een paar klassieke IC-ontwerpboeken van Paul Gray, Philip Allen, Ken Martin en leerde ze zo ongeveer uit mijn hoofd. Daarna was er geen weg meer terug.

Elektor: Waren techniek en ondernemerschap altijd al iets voor jou? Droomde je al van het oprichten en leiden van een bedrijf toen je voor het eerst naar de universiteit ging?

Teo Swee-Ann: Ik heb niet nagedacht over de zakelijke kant van de dingen. Ik ben meer een idealist; ik geloof dat kennis de wereld zal veranderen – niet het bedrijfsleven. Maar mijn latere ervaring leerde me natuurlijk dat je soms meer moet doen, je moet leren aanpassen en dat je ook een zakelijk platform voor de technologie moet creëren.

Elektor: Vertel ons over je technische interesses. Waar heb je je op geconcentreerd tijdens je master-opleiding aan de National University of Singapore?

Teo Swee-Ann: Als student had ik een voorliefde voor wiskunde en software en schuwde ik alles waar hardware bij kwam kijken. Dus mijn master-scriptie ging over de kenmerken van passieve microgolf-componenten in gelaagde media met behulp van EM-berekeningen. Een gelaagd medium kan iets zijn als een print of een IC, dat laag voor laag wordt gefabriceerd en waarvan het materiaal van laag tot laag verschilt. Het doel was om de hoogfrequente eigenschappen van passieve microgolf-componenten zoals spoelen in dergelijke media te berekenen.

We werden toen geconfronteerd met veel problemen: de moeilijkheid om een enkele Greense functie te maken om zowel het nabije als het verre veld te beschrijven, hoe de polen van de functies op te lossen en hoe de Sommerfeld-transformaties van de functies van

het frequentiedomein naar het ruimtelijke domein uit te voeren. Mijn werk werd dus opgesplitst in twee delen.

Het eerste deel was om te laten zien hoe we een enkele functie konden maken om zowel het nabije als het verre veld te beschrijven en om een efficiënte berekeningsmethode te hebben voor het berekenen van hun Sommerfeld-transformaties (zoiets als de Laplace-transformatie, maar dan met een Bessel- of Hankel-kern). We weten dat deze functies singulariteiten en vertakkingen hebben. Het was dus alsof we ons op een wiskundig mijnenveld met enkele afbakeningen begaven.

De hoofdzaak was om ten eerste de singulariteiten van deze functies op te lossen en ten tweede om 'synthetische polen' te creëren die gesloten oplossingen hebben in de latere berekening, om de convergentie van de berekening te helpen versnellen (een probleem dat ontstond door het insluiten van de polen voor het oplossen van singulariteiten). Het overgebleven kleine deel van de functies waar geen rekening mee is gehouden zal dan berekenbaar zijn en snel convergeren.

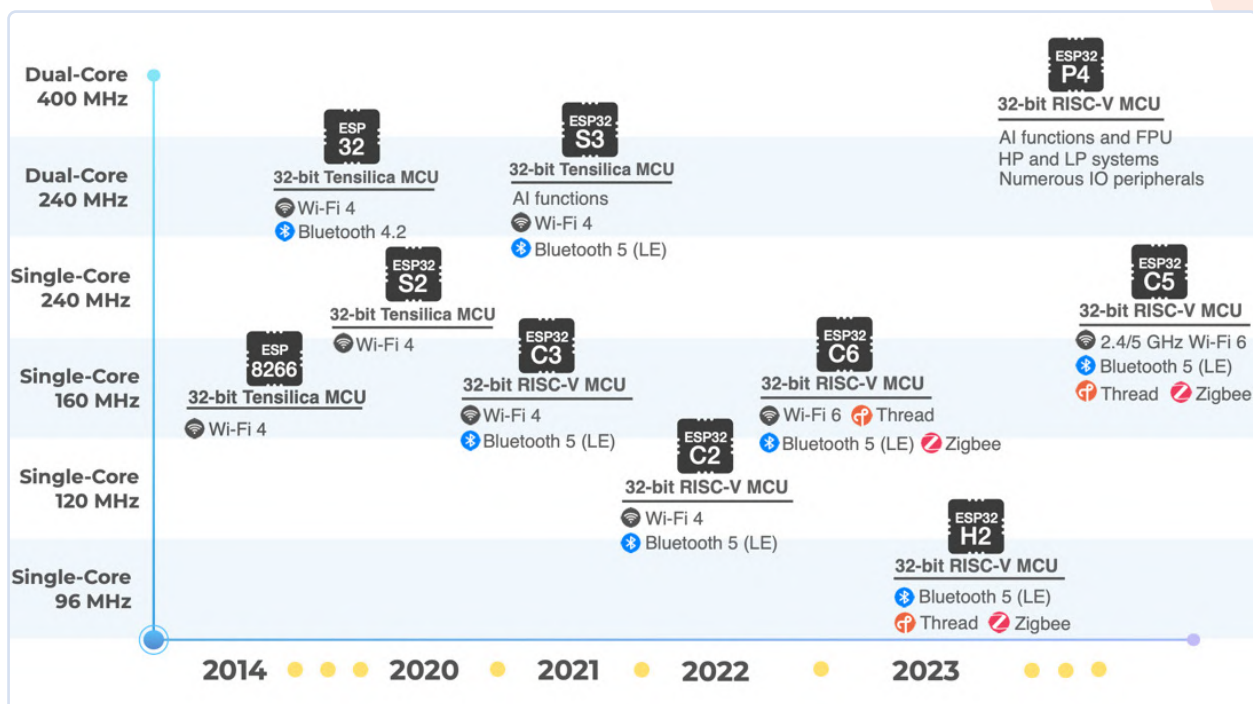
Tot slot ging het laatste probleem over hoe we de singulariteiten in het complexe domein konden oplossen. Ik creëerde een alternatieve techniek, die ik 'Cauchy Tomografie' noemde, om deze singulariteiten op te lossen, door gebruik te maken van het Cauchy-residu theorema en hun locaties terug te vinden via de 'Generalized Pencil-of-Function Technique' (GPOF). Ik gebruik deze techniek vandaag de dag nog steeds af en toe bij het ontwerpen van schakelingen om de overdrachtsfunctie van een systeem te berekenen op basis van de transiëntrespons. Het blijkt dat de transiëntrespons een soort tijddomein integratie-uitkomst is van de frequentierespons van het systeem. Door de twee domeinen om te rekenen, kun je als het ware zien hoe de systeempolen een bepaald soort transiëntrespons zouden creëren en vice versa.

Elektor: Je reactie op de uitnodiging van Elektor om de volgende gastredacteur te worden was een duidelijk en onmiddellijk "ja". Wanneer heb je Elektor leren kennen? Heeft Elektor jou en Espressif op een of andere manier beïnvloed?

Teo Swee-Ann: In 2015 maakten we voor het eerst kennis met Elektor. Wat opviel aan het tijdschrift was zijn kenmerkende vermogen om opkomende trends in de industrie te herkennen en te belichten. Ik geloof zelfs dat Elektor een van de pioniers was in het noemen van de Espressif-chips, te beginnen met de ESP8266, waardoor het een belangrijke rol heeft gespeeld voor de populariteit van deze chips. We raadplegen Elektor regelmatig als inspiratiebron, altijd bereid om de veranderende behoeften en interesses van het lezerspubliek te peilen. Bovendien vervult Elektor een rol van onschatbare waarde als kenniscentrum voor mensen die de elektronica-sector willen betreden. Het onderstreept een gedeelde verplichting die wij in de branche voelen: het verspreiden van onze kennis en er zo voor zorgen dat de elektronica-sector nieuw talent blijft aantrekken en stimuleren.

Elektor: Wat zijn de voordelen en mogelijke nadelen van je technische achtergrond in je huidige rol als CEO van Espressif?

Teo Swee-Ann: Ik ondervind er weinig nadelen van. Ik vind het heerlijk om in de frontlinie te zitten en ik werk nog steeds aan schakelingen. Door samen met het team te werken, weet ik beter met welke problemen we te maken hebben en hoe moeilijk het kan zijn om bepaalde features te bouwen. Dit stelt me in staat om middelen beter toe te wijzen, belangrijke problemen te identificeren, met het team te communiceren en een consensus te bereiken.





Een van onze eerste beslissingen was dat een IoT-chip idealiter 600 DMIPS zou moeten hebben. Dit prestatieniveau biedt een balans tussen de prestaties, die veel andere MCU's overtreffen, en kosteneffectiviteit, dankzij de dual-core architectuur en het efficiënte cache-systeem.

Elektor: Wat zijn je interesses naast techniek en elektronica?

Teo Swee-Ann: Ik speel vaak muziek terwijl ik nadenk over technische problemen. Het zet de hersenen op de een of andere manier open. Vroeger speelde ik veel klassieke gitaar en onlangs ben ik cello gaan spelen. Ik blader nu beetje bij beetje door de cellosuites van Bach en werk aan het verbeteren van mijn intonatie.

Elektor: Je hebt Espressif in 2008 opgericht. Je eerste product was de ESP8086, die in 2013 uitkwam. Was dat product het belangrijkste aandachtspunt van het bedrijf in die vijf jaar, of bood je ook andere diensten aan?

Teo Swee-Ann: De eerste vijf jaar was het een heel klein bedrijf. Ik werkte aan software om een taal te maken om schakelingen te kunnen beschrijven. Vandaar de naam "Espressif", wat betekende dat we onze ideeën over schakelingen konden formuleren. Het verkopen van software is echter moeilijk en we veranderden naar een consultancybedrijf dat analoge IP's bouwde voor onze klanten. Uiteindelijk raakten we geïnspireerd door het concept van een naderend technologisch breekpunt en besloten we dat we IoT-chips moesten bouwen, die zouden fungeren als het zenuwstelsel voor zo'n intelligent systeem.

Elektor: In 2014 lanceerde Espressif zijn eerste IoT SoC, de ESP8266EX. Voor welke technische uitdagingen was die oplossing bedoeld?

Teo Swee-Ann: Rond die tijd waren er ideeën om een goedkope WiFi-chip te maken. We besloten dat we deze uitdaging aan konden gaan door alle externe RF-componenten in de chip te integreren. In het verleden zag je modules die vol zaten met meer dan 50 tot 100 componenten. We hebben dit ontwerp gerevolutioneerd door de balun, antenneswitch, PA en LNA rechtstreeks in de chip op te nemen. De grootste uitdaging was het beheersen van het piekvermogen van 27 dBm van de eindversterker. Zonder een voldoende robuuste schakelaar liep de LNA het risico beschadigd te raken. We veranderden ook de manier waarop eindversterkers intern werden aangestuurd. In plaats van spoelen gebruikten we baluns, wat de stabiliteit van het systeem verbeterde. In het verleden laadden de meeste WiFi-chips de software-images op een nogal statische manier in het geheugen. Wij gebruikten een cache-ontwerp zodat de software in plaats daarvan continu uit het flash-geheugen kon worden gelezen. Het zou veel langzamer zijn, maar het werkte. We stopten ook veel van de logica van het systeem in de software in plaats van in de hardware. Dit hielp bij de ontwikkeling. Deze reeks van innovaties culmineerde in een zeer compact ontwerp. Ons eindproduct kostte een kwart tot een derde van wat

onze concurrenten boden. Vandaag de dag zijn we er trots op te kunnen zeggen dat veel van de technieken die we hebben ontwikkeld nu standaardpraktijken zijn in de industrie en aanzienlijk bijdragen aan het terugdringen van de kosten van IoT-chips.

Elektor: 2016 was een belangrijk jaar voor Espressif. Vertel ons eens over de ontwikkeling van de ESP32 en het uitbrengen ervan.

Teo Swee-Ann: Vóór de introductie van de ESP32 was onze aanpak wat spontaan en niet zo gestructureerd. De ESP32 betekende een belangrijke verandering in onze manier van denken, waardoor we marketing systematischer gingen benaderen. Een van onze eerste beslissingen was dat een IoT-chip idealiter 600 DMIPS zou moeten hebben. Dit prestatieniveau biedt een balans tussen de prestaties, die veel andere MCU's overtreffen, en kosteneffectiviteit, dankzij de dual-core architectuur en het efficiënte cache-systeem. De weg naar de ESP32 liep via talloze discussies, overwegingen en revisies. Tijdens deze fase stelden we formeel onze teamstructuur vast, verfijnden we ons werkproces en stemden we onze werkprocessen af op het ontwerp en de specificaties van het product. Dit was een verschil met onze vorige, minder formele, 'huis-tuin-en-keuken' aanpak. Tegelijk met de ontwikkeling van de ESP32 lanceerde ons softwareteam ESP-IDF. Beslissingen, zoals die met betrekking tot ons open-source beleid, werden in deze periode vastgelegd en vormden een solide basis voor toekomstige ontwikkelingen. Samengevat onderstreepte de evolutie van de ESP32 het toenemende belang van software, in plaats van alleen te focussen op hardware. Deze overgang werd verder verrijkt door de actieve betrokkenheid en waardevolle feedback van onze toegewijde community van ontwikkelaars.

Elektor: Vertel ons eens over de Matter-compatibele oplossingen van Espressif. Matter lijkt een prioriteit te zijn in 2023 zal dat in 2024 blijven, klopt dat?

Teo Swee-Ann: Tot nu toe heeft de smart home-industrie haar top nog niet bereikt en is het voor consumenten nog steeds moeilijk om de producten effectief te gebruiken. De gebruiksscenario's zijn beperkt en de ervaring die ze bieden is gefragmenteerd. Matter probeert hier verandering in te brengen door de apparaten dezelfde taal te laten spreken, en het is erg hoopgevend om te zien dat de meeste grote spelers samen deze problemen willen oplossen. Het belangrijkste is dat we positieve reacties van klanten zien. Daarom is het een logische prioriteit voor ons. Espressif's benadering van het bouwen van Matter-oplossingen blijft niet beperkt tot hardware. Hoewel we verschillende chips hebben



die Matter via Thread- en WiFi-protocollen ondersteunen, gaan we verder door specifieke problemen van klanten bij de adoptie van Matter te begrijpen en te proberen daar een oplossing voor te creëren. Onze ESP ZeroCode-modules, Certificate Provisioning Service en Certification Assistance Service zijn goede voorbeelden van deze aanpak.

Elektor: Wat kwam er kijken bij het maken van ESP RainMaker?

Teo Swee-Ann: ESP RainMaker is ook ontworpen om de pijnpunten van klanten aan te pakken. Voorheen zagen we dat klanten het wiel opnieuw uitvonden als het ging om het bouwen van een IoT-cloudplatform of dat ze producten bouwden op basis van cloudplatforms van derden met minimale flexibiliteit en controle over de gegevens. Aan de andere kant hield de serverloze cloudarchitectuur zoveel beloften in voor het bouwen van zo'n cloud zonder zich zorgen te hoeven maken over het beheer van de infrastructuur. Voor fabrikanten van apparaten is het echter nog steeds moeilijk om zo'n platform vanaf nul op te bouwen. Daarom hebben we ESP RainMaker gemaakt, met de visie om een cloudplatform voor klanten te bouwen dat hen volledige controle en aanpasbaarheid geeft om hun eigen IoT-cloud te creëren met aanzienlijk minder investeringen in engineering.

Elektor: AIoT-systemen en -producten verzenden veel gegevens. Dit leidt natuurlijk tot zorgen over gegevensbescherming en privacy. Denkt Espressif dat de markt voor AIoT-producten zal opbloeien zodra er overeenstemming is bereikt over internationale normen voor gegevensbescherming en privacy?

Teo Swee-Ann: In het huidige AIoT-landschap zijn dataprivacy en regelgevingskwesaties zeker van belang, en we bieden onze klanten meerdere hulpmiddelen om daar verlichting te bieden. Allereerst moedigen we edge AI aan door er ondersteuning voor op te nemen in onze hardware. De ESP32-S3 en komende ESP32-P4 hebben bijvoorbeeld AI-instructies voor snelle verwerking van deep learning-modellen, zodat apparaten gegevens lokaal kunnen verwerken in plaats van ze naar servers te moeten sturen. Voor de gegevens die op servers worden opgeslagen, bieden we klanten tools die dit doen op een manier die voldoet aan de regelgeving – zo biedt Espressif's ESP RainMaker cloud-implementatie bijvoorbeeld alle functies die nodig zijn om GDPR-compliance te bereiken wanneer klanten hun eigen IoT-cloud bouwen op basis van ESP RainMaker (zoals ook door TÜV Rheinland is beoordeeld).

Beveiliging speelt hier ook een rol: kwaadwillenden die een systeem binnendringen en er met alle gegevens vandoor gaan is tegenwoordig een veel voorkomend verschijnsel. We hebben ons verplicht om veilige hardware- en softwarecomponenten te leveren die voldoen aan het Amerikaanse Cybersecurity Labeling-programma en andere internationale initiatieven, zoals het Cybersecurity Labeling Scheme van Singapore. Apparaten die gecertificeerd zijn onder programma's als deze kunnen klanten een gerust gevoel geven als het gaat om hun beveiliging.

Een meer urgente uitdaging die we hebben vastgesteld is het gebrek aan ontwikkelaars van embedded software. Daarnaast maakt de

versnippering van de markt de situatie nog gecompliceerder, waardoor de complexiteit toeneemt.

Bij Espressif pakken we deze uitdagingen actief aan. We hebben veel oplossingen geïntroduceerd. We hebben bijvoorbeeld ESP ZeroCode voor het ontwikkelen van Matter-applicaties, ESP SkaiNet voor lokale AI-spraakherkenning, ESP RainMaker voor het maken van je eigen cloudplatform, ESP ADF voor het maken van je eigen WiFi-audiosysteem enzovoort.

Ons doel is om onze klanten uit te rusten met de tools die ze nodig hebben om oplossingen te creëren zonder vanaf nul te beginnen, om efficiëntie en snelheid in de ontwikkeling te garanderen. Voor een belangrijke positieve verandering in het AIoT-landschap geloven we dat het verminderen van fragmentatie cruciaal is. Het is noodzakelijk dat verschillende platforms samenwerken en een uniforme standaard bevorderen. Hoewel de Matter-standaard veelbelovend is in deze richting, moeten het volledige potentieel en de impact op de lange termijn nog worden vastgesteld.

Elektor: Heeft het wereldwijde chipstekort het gebruik van op Espressif gebaseerde technologie in commerciële producten gestimuleerd? Zo ja, hoe kunnen deze producten helpen om het merk Espressif verder te versterken?

Teo Swee-Ann: Tijdens de uitdagende periode van het wereldwijde chipstekort heeft Espressif een strategische aanpak geïmplementeerd door de verkoop van chips aan onze klanten te rantsoeneren. Deze beslissing werd door veel van onze klanten gewaardeerd. Ten eerste zorgde het ervoor dat ze een constante aanvoer van chips bleven ontvangen, ook al waren de hoeveelheden beperkter. Ten tweede voorkwam deze strategie dat onze klanten hun voorraden overhaast zouden aanvullen als reactie op het tekort. En, wat belangrijk is, we bleven ons gedurende deze periode inzetten om onze prijzen niet te verhogen, zodat de volatiliteit van de markt geen negatieve invloed had op onze prijsstructuur. Ik denk dat onze stabiliteit en bescheiden groei in 2023 de effectiviteit en de vooruitziende blik van dit beleid weerspiegelen.

Elektor: In een interview met Elektor een paar jaar geleden zei je dat AI de volgende belangrijke ontwikkeling zou zijn, en dat is uitgekomen. Wat denk je dat de volgende grote ontwikkeling in de elektronica zal zijn?

Teo Swee-Ann: Op het gebied van elektronica zijn er twee verschillende trajecten waar we rekening mee moeten houden. Ten eerste is er het traject dat leidt naar een meer immersief ecosysteem met virtuele of gemengde realiteit. Stel je een toekomst voor waarin we in de komende decennia implantaten kunnen hebben die niet alleen de traditionele zintuigen zoals zien en horen aanvullen, maar ook nieuwe zintuiglijke ervaringen of cognitieve paden introduceren die de natuurlijke evolutie nog niet eerder heeft voortgebracht. Dit zou betekenen dat onze emoties, cognitieve capaciteiten en zintuiglijke ervaringen aanzienlijk versterkt of gemoduleerd zouden worden door AI. Het realiseren van deze visie vereist aanzienlijke technologische innovatie, met name in het bereiken van hogere verwerkingscapaciteiten bij lager stroomverbruik en lagere spanning, met de meest compacte vormfactoren. Aan de andere kant legt de tweede richting de nadruk op de



Als we naar de toekomst kijken, blijft ons belangrijkste doel het creëren van baanbrekende oplossingen die prioriteit geven aan efficiënt energiegebruik en kleinere vormfactoren.



Over Teo Swee-Ann

Teo Swee-Ann is de oprichter en CEO van het beursgenoteerde halfgeleiderbedrijf Espressif Systems uit Shanghai. Hij heeft een master in elektrotechniek van de National University of Singapore. Teo werkte bij de Amerikaanse chipmakers Transilica en Marvell Technology Group en het Chinese Montage Technology voordat hij in 2008 zijn eigen bedrijf oprichtte.

toepassing van AI in het ontwerpaspect van elektronica. Ter vergelijking: als AI in staat is om autonoom voertuigen te besturen, dan heeft het zeker de potentie om geavanceerde schakelingen te ontwerpen. Deze verschuiving pakt een opmerkelijke uitdaging in ons vakgebied aan, namelijk de neiging om ontwerpen te ingewikkeld te maken. De integratie van AI in het ontwerpproces biedt de mogelijkheid om deze ontwerpen te verfijnen, ze gestroomlijnder en effectiever te maken en zo de realisatie van het eerste traject te versnellen.

Elektor: Wat is jouw visie voor Espressif? Waar zie je het bedrijf over, laten we zeggen, vijf jaar?

Teo Swee-Ann: Bij Espressif positioneren we onszelf in de eerste plaats als toonaangevend AIoT-chipproducent. Dat gezegd hebbende, reiken onze mogelijkheden veel verder dan alleen

hardware. We hebben een sterke positie in softwareontwikkeling, met ons eigen softwaresysteem ESP-IDF en uitgebreide frameworks voor oplossingen voor cloudintegratie, edge-AI, mesh-netwerken, audio- en cameratoepassingen en nog veel meer. Wat ons echt onderscheidt is onze bloeiende community en ecosysteem, waar professionals en amateurs samenkomen om baanbrekende ideeën te delen en te ontwikkelen. Als we naar de toekomst kijken, blijft ons belangrijkste doel het creëren van baanbrekende oplossingen die prioriteit geven aan efficiënt energiegebruik en kleinere vormfactoren. In essentie hopen we dat Espressif een onderscheidend ethos en een vooruitstrevende houding ten opzichte van engineering, innovatie en het stimuleren van een samenwerkende community vertegenwoordigt. ◀

230614-03 (vertaling: Hans Adams)

WEBLINK

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



Aan de slag met ESP-DL de Deep Learning SDK van Espressif

ESP32-S3 wordt geleverd met AI-versnelling die u kunt gebruiken voor verschillende Machine Learning-toepassingen. ESP-DL is de Deep Learning SDK van Espressif, waarmee u uw voordeel kunt doen met de AI-versnellings-instructies om geoptimaliseerde ML-modellen te bouwen. Deze SDK biedt voorbeelden van gezichtsherkenning van mensen en katten. ESP-DL ondersteunt nu ook de TVM-compiler waarmee u TensorFlow-, PyTorch- of ONNX-modellen kunt gebruiken. Espressif biedt ook een port van de TensorFlow Lite Micro SDK van Google met ESP32-S3 versnellingsondersteuning, waarmee u standaard TensorFlow Lite-modellen kunt gebruiken op de ESP32-S3.

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

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



Simuleer ESP32 met Wokwi

de digitale tweeling van je project

Uri Shaked, Wokwi

Wokwi is een simulator voor embedded systemen en IoT-apparaten. Het biedt een online-omgeving waar je ESP32-projecten kunt opzetten, debuggen en delen zonder dat je tastbare hardware nodig hebt. De simulator draait gewoon in een webbrowser en kan alle chips uit de ESP32-familie simuleren: de klassieke ESP32, maar ook de S2-, S3-, C3-, C6- en H2-varianten. En ook andere microcontrollerfamilies kunnen worden gesimuleerd.

Met Wokwi kun je een digitale tweeling maken van je project: je kunt verschillende invoer- en uitvoerapparaten simuleren [1], zoals LC-displays, sensoren, motoren, LED's, drukknoppen, luidsprekers en potentiometers, en je kunt zelfs eigen simulatiemodellen

maken voor nieuwe apparaten. Met de simulator kun je sneller itereren en aan je firmware werken; zelfs als de hardware niet beschikbaar is, kun je gebruik maken van krachtige debugtools, je projecten delen en gemakkelijk samenwerken met andere ontwikkelaars.

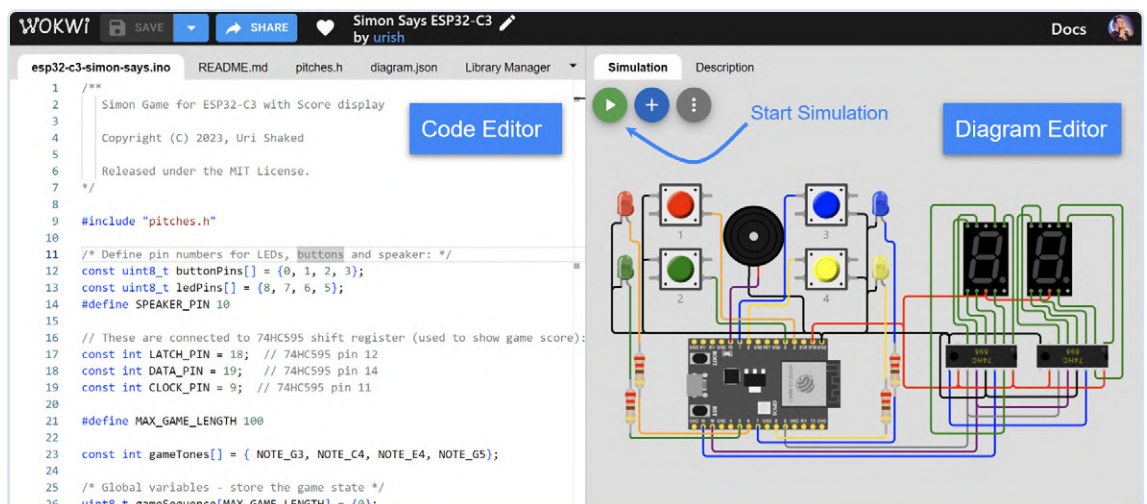
Start je Wokwi-avontuur

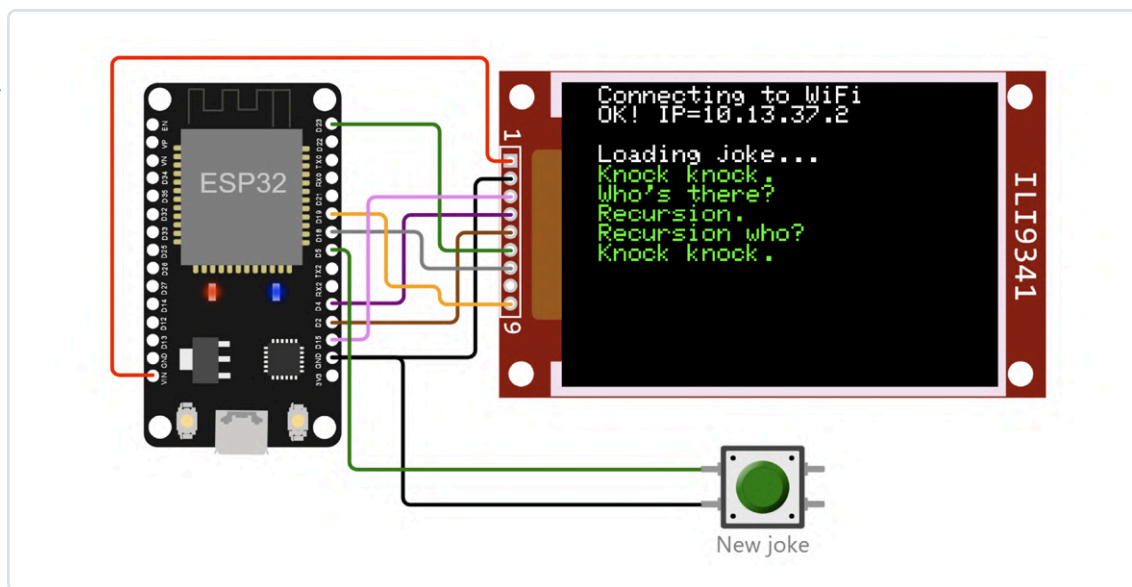
Je kunt je favoriete programmeertaal gebruiken met Wokwi. Als je een beginner bent, is MicroPython of Arduino Core waarschijnlijk een goede keus. Voor professionals en geavanceerde gebruikers zijn er de ESP-IDF, Rust en embedded RTOS zoals Zephyr of NuttX.

Het is een goed idee om enkele bestaande voorbeelden te bekijken om ideeën te krijgen van wat je zoal met Wokwi kunt bouwen:

- › MicroPython [2]
- › ESP32 met Arduino Core [3]
- › Rust [4]
- › DeviceScript (TypeScript for ESP32) [5]

Figuur 1. De gebruikersinterface van Wokwi met "Simon Says"





Figuur 2. WiFi-simulatie: client [8].

Open een voorbeeld en druk op de groene play-knop om de simulatie te starten en met het project aan de slag te gaan. Sommige projecten hebben ook een [README](#)-bestand met meer informatie over het project en hoe ermee te werken. Om een nieuw project aan te maken, ga je naar [6]. Kies dan een microcontroller en een programmeertaal en klik op *Start*.

De gebruikersinterface van Wokwi

De webversie van Wokwi bestaat uit twee delen: In het linker paneel kun je de broncode bewerken (de *Code Editor*), en rechts teken je het schema van het project door verschillende apparaten/componenten toe te voegen en ze met de microcontroller te verbinden (de *Diagram Editor*), zie **figuur 1**.

De Diagram Editor

Voeg elementen toe aan het schema door op de blauwe + knop te klikken en het gewenste onderdeel uit de lijst te kiezen. Sleep de onderdelen naar de gewenste positie. Om een draad te tekenen, klikt je op de pin waar de draad begint en dan op de pin waar de draad naartoe moet gaan. Als je de draad een specifieke route wilt geven, klik je na het aanklikken van de begin-pin op de plaats waar je hem wilt hebben.

Voor een net resultaat kun je het raster inschakelen door op *G* te drukken. Dat lijnt alle componenten en draden uit op een raster van 0,1" (2,54 mm) – dat is de standaard pinafstand van een header. Enkele nuttige sneltoetsen: *D* dupliceert het geselecteerde onderdeel, *R* roteert het, de cijfers 0 t/m 9 veranderen de kleur van een draad, LED of drukknop en als je Shift indrukt kun je met de muis je meerdere componenten tegelijk selecteren. Zie voor meer sneltoetsen de handleiding van de Diagram Editor [7].

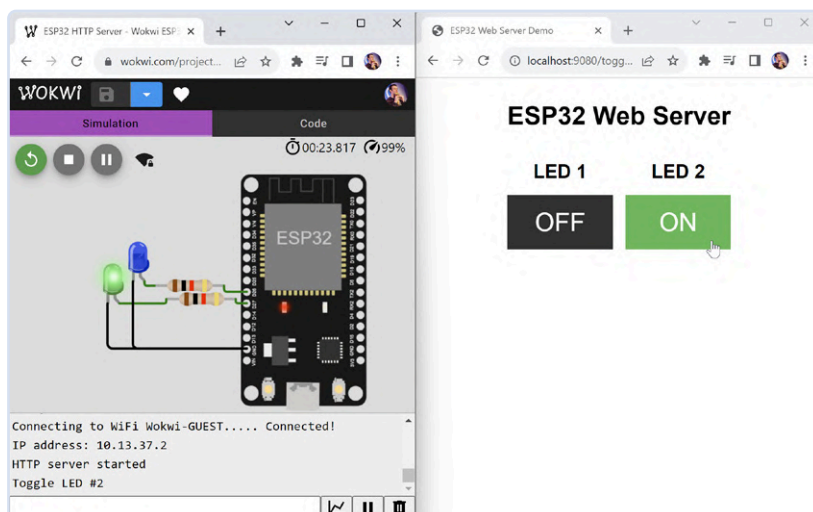
Figuur 3. WiFi-simulatie: webserver [9].

De Library Manager

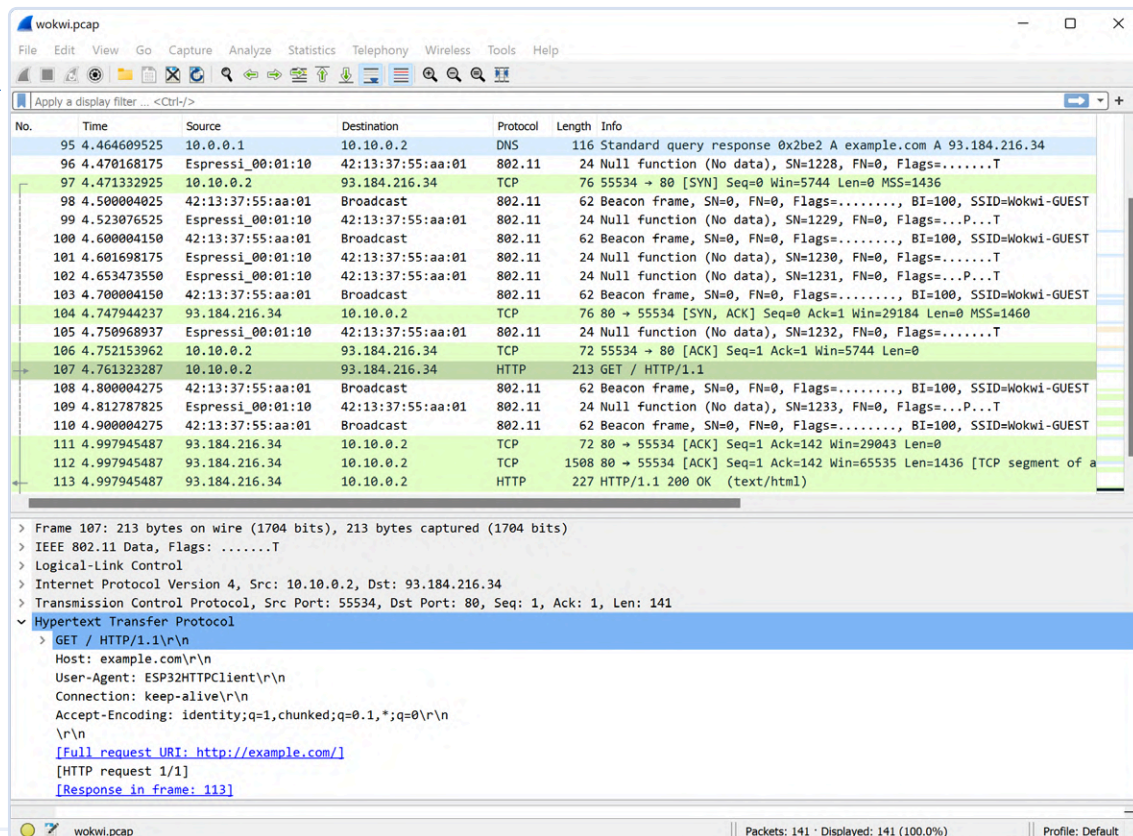
Gebruik de Library Manager om snel standaard Arduino-bibliotheken toe te voegen aan een project. Betalende gebruikers kunnen ook eigen Arduinobibliotheken uploaden en meenemen in hun project. Voor andere programmeertalen kun je bibliotheken meenemen door de project-configuratiebestanden aan te passen (bijvoorbeeld *Cargo.toml* voor Rust), of door de broncode van de bibliotheek rechtstreeks op te nemen in je project (bijvoorbeeld voor MicroPython).

WiFi-simulatie

De ESP32 heeft ingebouwde WiFi-functionaliteit, daarom is hij populair voor IoT-projecten. Wokwi simuleert de WiFi-functionaliteit van de chip. De simulator (**figuur 2**) maakt een virtueel access point met de naam *Wokwi-GUEST* aan. Het is een open access point (dus zonder wachtwoord). Het access point is verbonden met het internet, zodat je de gesimuleerde projecten met http-servers, MQTT-brokers en andere cloudd-services zoals Firebase, ThingsSpeak, Blynk en Azure IoT kunnen verbinden (**figuur 3**).



Figuur 4. Bekijk het Wi-Fi-dataverkeer.



Betalende gebruikers kunnen ook een speciale IoT-Gateway [10] draaien op hun computer, waarmee ze verbinding kunnen maken met lokale servers vanuit de simulatie, en hun computer kunnen verbinden met een http-server (of een ander soort server) die draait in de simulator. Onder de motorkap simuleert Wokwi een complete netwerk-stack: dat gaat van de laagste 802.11 MAC-laag, via de IP- en TCP/UDP-laag, helemaal omhoog tot aan de protocollen zoals DNS, HTTP, MQTT, CoAP enzovoort. We kunnen de ruwe netwerkdata [11] bekijken in een netwerkprotocol-analyzer zoals Wireshark (figuur 4).

Pin Function Debugger

De ESP32 biedt een flexibele pinconfiguratie: de software kan kiezen welke periferie is verbonden met elk van de pinnen door de IO MUX, GPIO-Matrix en de low-power/RTC periferie te configureren. De pinconfiguratie softwarematig vastleggen kan heel handig zijn, maar het maakt het wel ingewikkelder om uit te vinden wat de werkelijke pinconfiguratie van de firmware is. Gelukkig kunnen we in Wokwi gewoon de simulatie pauzeren en direct de actuele functie en toestand van elke pin zien (figuur 5).

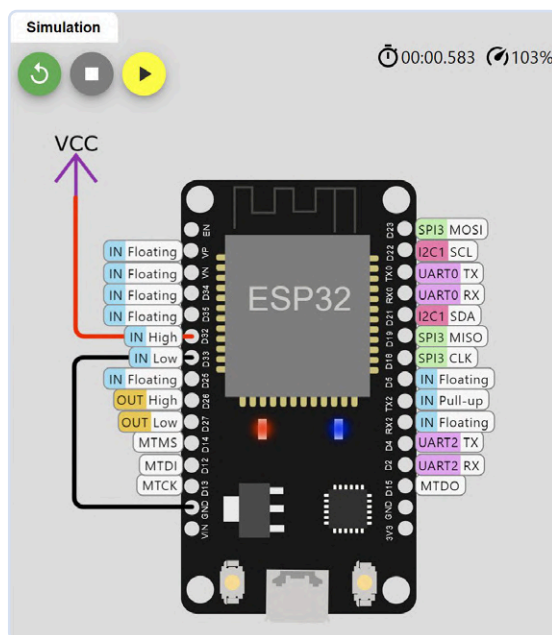
Integratie met Visual Studio Code

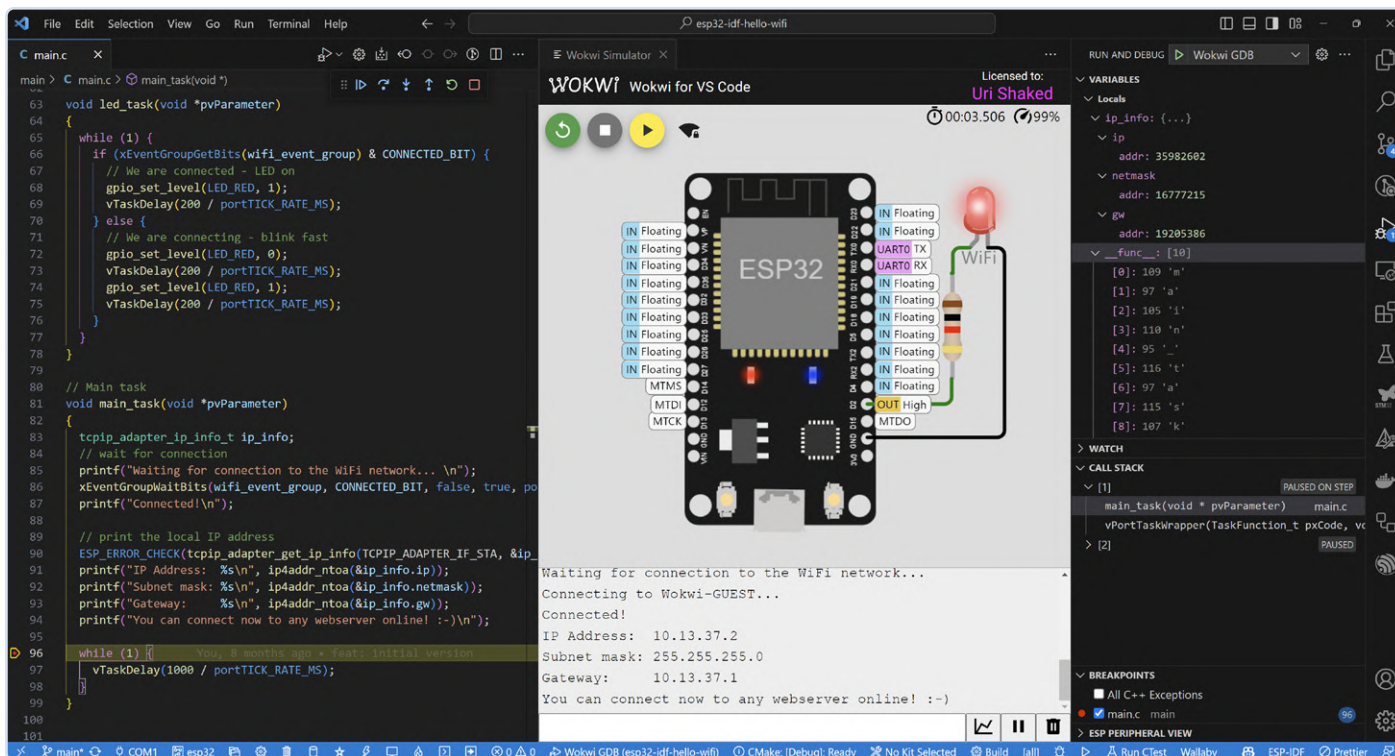
Het gebruik van Wokwi in de webbrowser is heel leerzaam en snel, en ook het delen van projecten is gemakkelijk. Maar voor ingewikkelde projecten is het toch aan te raden om Wokwi te gebruiken in combinatie met de gewone IDE en development tools. Wokwi integreert naadloos met Visual Studio Code. Je hoeft alleen de Wokwi for VS Code-extensie te installeren [12] en een eenvoudig configuratiebestand aan te maken [13] dat Wokwi vertelt waar je gecompileerde firmware staat.

We kunnen Wokwi ook integreren met de ingebouwde debugger van VS Code door een paar regels aan de configuratie toe te voegen [14]. In tegenstelling tot echte hardware heeft Wokwi een onbeperkt aantal breakpoints, zodat je efficiënt kunt debuggen (figuur 6).

Voor IoT-projecten kunt je TCP port-forwarding gebruiken [15]. Daarmee kun je je computer verbinden met een webserver (of een ander soort TCP-server) die draait in de gesimuleerde ESP32-chip.

Figuur 5. Simulatie gepauzeerd: van elke pin is de toestand te zien.





▲

Integratie met Espressif IDE

Liefhebbers van de Espressif IDE kunnen ook een start-configuratie maken om een project in de simulator te draaien. De (UART-)uitvoer van het programma gaat dan naar het console van de IDE. Zie *How to Use Wokwi Simulator with Espressif-IDE* [16] voor alle details.

Eigen IC's

Met Custom Chips kunnen we de functionaliteit van Wokwi uitbreiden met nieuwe onderdelen. We kunnen sensoren, displays, geheugens, testinstrumenten (figuur 7) en zelfs eigen hardware simuleren. Om een eigen IC te maken, leggen we de pinning ervan vast in een JSON-file en schrijven code om de logica van de chip te implementeren. De code wordt meestal geschreven in C, en de API lijkt op gebruikelijke hardware-abstractielagen (HAL) voor embedded chips, dus firmware-ontwikkelaars zullen daar mee uit de voeten kunnen. Er is ook experimentele ondersteuning voor andere talen zoals Rust, AssemblyScript en Verilog. Zie voor meer informatie en voorbeelden de Chips API-documentatie [17].

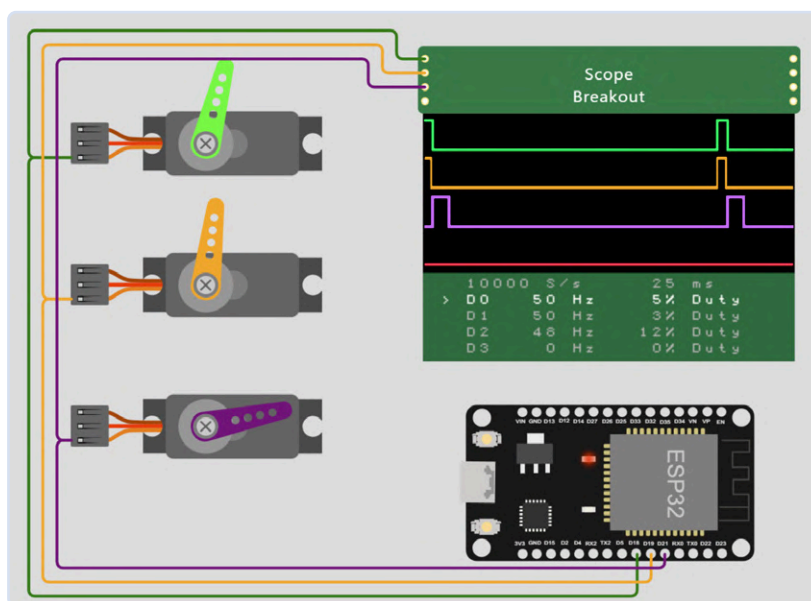
Wokwi voor CI

Simulatie kan veel tijd besparen bij het maken van prototypes, maar het kan ook helpen bij het foutzoeken tijdens de ontwikkeling. Continuous Integration (CI) is een moderne aanpak bij het ontwikkelen van software: bouw en test het project na elke aanpassing in de code. Maar testen met echte hardware is

tijdrovend en moeilijk te automatiseren. Het wordt zelfs nog lastiger als je volledige systeemintegratie wenst. Denk maar eens aan geautomatiseerde WiFi-tests of het vastleggen van het beeld op een LC-display. Dat is heel lastig stabiel en betrouwbaar te realiseren.

Wokwi voor CI lost dat allemaal op. Als je GitHub Actions gebruikt, kun je met `wokwi-ci-action` [18] simulatie integreren in je bestaande workflow met slechts een paar regels code. Voor andere CI-omgevingen biedt `wokwi-cli` [19] een eenvoudige commandoregel-interface voor het uitvoeren en testen van firmware in simulatie. De CI Simulation Server is beschikbaar als een beheerde clouddienst, en kan ook op de werkplek worden ingezet.

Figuur 6. Debuggen van een ESP-IDF-project met VS Code en Wokwi.



Figuur 7. Een digitale 'scoop' die een gebruiker heeft gemaakt met behulp van de Custom Chips API.


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

build-and-test

succeeded 1 minute ago in 1m 15s

Search logs

> Build PlatformIO Project50s

> Test with Wokwi5s

1 ▶ Run wokwi/wokwi-ci-action@v1

14 ▶ Run wget -q -O /usr/local/bin/wokwi-cli https://github.com/wokwi/wokwi-cli/releases/latest/download/wokwi-cli-linuxstatic-x64

25 ▶ Run wokwi-cli --timeout "10000" --expect-text "" \

38 Wokwi CLI v0.6.4 (8fa2d7b7b70f)

39 Connected to Wokwi Simulation API 1.0.0-20230804-gf0f4bfc7

40 Starting simulation...

41 ets Jul 29 2019 12:21:46

42

43 rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)

44 configsip: 0, SPIWP:0xee

45 clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00

46 mode:DIO, clock div:2

47 load:0x3fff0030,len:1156

48 load:0x40078000,len:11456

49 ho 0 tail 12 room 4

50 load:0x40080400,len:2972

51 entry 0x400805dc

52 Pushbutton Counter

53 [Pushbutton counter test] Expected text matched: "Pushbutton Counter"

54 [Pushbutton counter test] delay 100ms

55 Button pressed 1 times

56 [Pushbutton counter test] delay 200ms

57 [Pushbutton counter test] delay 100ms

58 Button pressed 2 times

59 [Pushbutton counter test] delay 200ms

60 Button pressed 3 times

61 [Pushbutton counter test] Expected text matched: "Button pressed 3 times"

62 [Pushbutton counter test] Scenario completed successfully

▲
Figuur 8. De code voor het automatiseringsscenario [20] (links) met de corresponderende GitHub Action-uitvoer (rechts).

Automatiseringsscenario's

Met automatiseringsscenario's (figuur 8) kun je ingewikkelde tests en verificaties uitvoeren op firmware. Die zijn gemakkelijk te schrijven: elk scenario is een YAML-bestand met een reeks commando's zoals het indrukken van een knop, instellen van een temperatuursensorwaarde of zoeken naar een string in de seriële uitvoer.

Beperkingen

Wokwi biedt veel mogelijkheden, maar heeft ook enkele beperkingen. De belangrijkste focus van de simulator is het draaien van embedded code. De simulator is vooral digitaal en de ondersteuning voor analoge simulatie is beperkt. Wokwi zal je niet waarschuwen als je een stroombeperzingsweerstand vergeet; er komt ook geen magische rook vrij (maar wie weet wat de toekomst nog brengt...). [↩](#)

vertaling: Evelien Snel — 230523-03

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de redactie van Elektor via redactie@elektor.com.



Over de auteur

Uri Shaked is al heel lang maker. Op dit moment werkt hij aan Wokwi, een online IoT- en embedded system simulatieplatform, en aan Tiny Tapeout, dat het fabriceren van eigen ASIC's betaalbaar en toegankelijk maakt.

Gerelateerde producten

- ▶ ESP32-C3-DevKitM-1
www.elektor.nl/20324
- ▶ Koen Vervloesem, Getting Started with ESPHome (Elektor 2021)
www.elektor.nl/19738

WEBLINKS

- [1] Wokwi Supported Hardware: <https://docs.wokwi.com/getting-started/supported-hardware>
- [2] Wokwi Online Embedded Python Simulator: <https://wokwi.com/micropython>
- [3] Wokwi Online ESP32 Simulator: <https://wokwi.com/esp32>
- [4] Wokwi Online Embedded Rust Simulator: <https://wokwi.com/rust>
- [5] Wokwi Online DeviceScript Simulator: <https://wokwi.com/devicescript>
- [6] Nieuw Arduino Uno Project startpagina: <https://wokwi.com/projects/new>
- [7] Diagram Editor sneltoetsen: <https://docs.wokwi.com/guides/diagram-editor#keyboard-shortcuts>
- [8] _esp32-jokes-api_ W-Fi-simulatievoorbeeld: <https://wokwi.com/projects/342032431249883731>
- [9] ESP32 HTTP-server voorbeeld: <https://wokwi.com/projects/320964045035274834>
- [10] ESP32 Wi-Fi Network: de Private Gateway-applicatie: <https://docs.wokwi.com/guides/esp32-wifi#the-private-gateway>
- [11] ESP32 Wi-Fi Network: WiFi-dataverkeer bekijken met Wireshark: <https://tinyurl.com/wsviewtraffic>
- [12] Wokwi Simulator — Visual Studio Code Extension: <https://tinyurl.com/wokwivsext>
- [13] Projectconfiguratie (_wokwi.toml_): <https://docs.wokwi.com/vscode/project-config>
- [14] Wokwi — code debuggen: <https://docs.wokwi.com/vscode/debugging>
- [15] Projectconfiguratie — IoT Gateway (Port Forwarding): <https://docs.wokwi.com/vscode/project-config#iot-gateway-esp32-wifi>
- [16] Gebruik van de Wokwi Simulator met Espressif-IDE: <https://tinyurl.com/wokwiespide>
- [17] Aan de slag met de Wokwi Custom Chips C API: <https://docs.wokwi.com/chips-api/getting-started>
- [18] _wokwi-ci-action_: <https://github.com/wokwi/wokwi-ci-action>
- [19] _wowkwi-cli_: <https://github.com/wokwi/wokwi-cli>
- [20] _platform-io-esp32-counter-ci_ — Automation Scenario Code: <https://tinyurl.com/pushcnttst>

Revolutioneer uw producten met WiFi Motion™,
nu beschikbaar op Espressif Wi-Fi chips.



Ontdek de toekomst van bewegingsdetectie.

Neem nu contact met ons op via www.cognitivesystems.com

COGNITIVE ∞

ESP32-S3-BOX-3



Figuur 1. De ESP32-S3-BOX-3 kit uitgepakt. Niet afgebeeld zijn de USB-kabel en de kleine RGB LED-module met de vier jumperdraden. De nectarine was niet inbegrepen.

Test van de ESP32-S3-BOX-3

een uitgebreid AIoT-ontwikkelpplatform

Clemens Valens (Elektor)

De ESP32-S3-BOX-3 van Espressif is een platform voor het ontwikkelen van toepassingen zoals persoonlijke assistenten, slimme speakers en andere spraakgestuurde apparaten. Laten we die eens aan te tand voelen.

De ESP32-S3-BOX-3 [1] kit is gebaseerd op een ESP32-S3 en wordt op de markt gebracht als "je volgende AIoT-ontwikkeltol". AIoT staat voor *Artificial Intelligence of Things* en is nauw verwant aan, maar niet te verwarren met IoT, wat Internet of Things betekent (zoals je vast al wist). Het heeft de vorm van een kleine console met een aanraakscherm. Als toepassingen voor de kit worden AI-chatbots, Matter (een verenigend protocol voor domotica), robotcontrollers en slimme sensorapparaten gesuggereerd.

Wat zit er in de doos

De ESP32-S3-BOX-3 (figuur 1) wordt geleverd als een van die dozen die langzaam opengaan (om het 'wow-effect' te versterken, zoals iemand die bij Apple heeft gewerkt me vertelde). Bij het openen zie je een displaymodule (55×50 mm) plus een soort standaard, en een korte (ongeveer 30 cm) USB-C kabel. Maar dat is niet alles. Onder de standaard zit zich een tweede, iets kleinere standaard verborgen. Onder de displaymodule vind je een nog kleinere standaard en zelfs een veel kleinere vierde die je op een breadboard kunt aansluiten. Samen met de USB-kabel is een zakje met een kleine RGB LED-module en vier jumperdraden verpakt.

Displaymodule

De displaymodule is verrassend zwaar (60 g) voor zo'n klein ding (55×50×12 mm BHD). Deze module heeft alle kracht in zich: een ESP32-S3-WROOM-1-N16R16V-module met 2,4GHz-WiFi (802.11b/g/n) en Bluetooth 5 (LE). Het display zelf is een 2,4" TFT-display met 300×240 pixels TFT-scherm met capacitieve aanraakfunctionaliteit. Mocht je je afvragen: er zit geen batterij in (figuur 2).

Aan de voorkant zitten, naast het display, twee kleine openingen (voor de microfoons) en een rode cirkel (de return-knop). Er zijn twee drukknoppen (Boot en Rst) en een USB-C connector aan de linkerkant en aan de rechterkant iets dat op een microSD-kaartsleuf lijkt maar in werkelijkheid een luidspreker is. Aan de bovenkant zit een mute-knop en twee LED's; aan de onderkant steekt een PCIe-connector uit. Er zit niets op de achterkant van de module.

Standards en houders

De PCIe-connector past in een socket op een van de standards of beugels. Zoals eerder vermeld zijn er vier, elk met een andere functie (**figuur 3**). De bovenste is de DOCK. Deze heeft twee Pmod-uitbreidingsaansluitingen (2x6 female header) aan de achterkant, samen met een USB-C-aansluiting voor voeding (in en uit). Een USB-A hostaansluiting is beschikbaar aan de linkerkant. Een label toont de namen van de signalen die toegankelijk zijn op de Pmod-connectoren.

Sensorhouder

De grootste standaard heet SENSOR. Het heeft een temperatuur- en vochtigheidssensor met een kleine aan/uit-schakelaar aan de achterkant, een microSD-kaartsleuf (geen luidspreker deze keer) aan de linkerkant en een USB-C power-in aansluiting aan de rechterkant. Aan de voorkant zitten een IR-sensor en LED, en een echt coole functie als je het mij vraagt: een radar. Deze standaard heeft ruimte voor een accu van het type 18650.

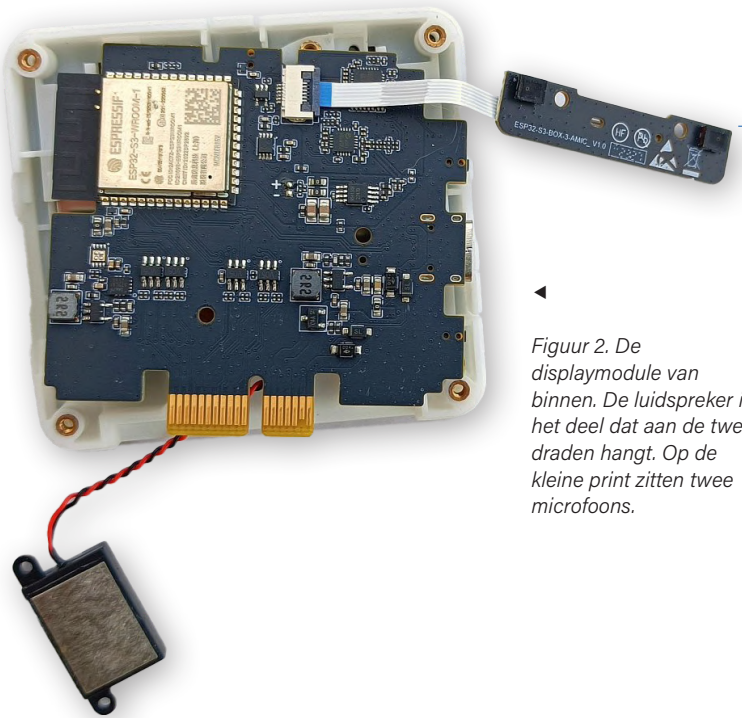
De derde standaard wordt BRACKET genoemd, en deze kan ergens anders aan worden bevestigd dankzij de twee schroeven aan de achterkant. Deze standaard heeft ook twee Pmod-headers en een USB-C aansluiting. Er zijn hier geen labels, dus we nemen aan dat de Pmod-headers op dezelfde manier zijn bedraad als op de DOCK. De kleinste standaard, BREAD, is een eenvoudig 24-polig PCIe/breadboard break-out board. De pinnen die in het breadboard gaan zijn handig gelabeld.

Eerste keer inschakelen

Wanneer je de displaymodule voor de eerste keer inschakelt, start hij op in wat later de help- of hint-modus blijkt te zijn. In deze modus krijg je een paar schermen met tips te zien die enige basisfunctionaliteit van het apparaat uitleggen. Daarna gaat het naar de normale modus. In deze modus kun je door zes functies bladeren: Sensor Monitor, Device Control, Network, Media Player, Help en About Us.

Spraakbesturing

Volgens de helpschermen kun je sommige dingen met je stem activeren door "Hi ee es pie" (ESP letter voor letter uitgesproken) te zeggen. Omdat er niets gebeurde toen ik dit probeerde, heb ik de gebruikershandleiding gedownload [2]. Hiervoor staat een QR-code op de onderkant



Figuur 2. De displaymodule van binnen. De luidspreker is het deel dat aan de twee draden hangt. Op de kleine print zitten twee microfoons.

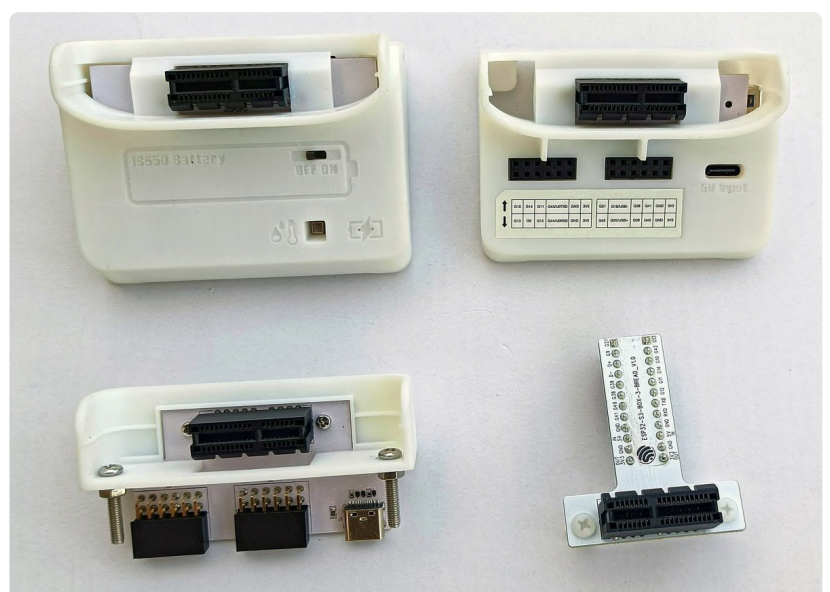
van de doos. Volgens de handleiding werkt stembesturing in elk scherm (tenzij de mute-LED oplicht). Met deze kennis gewapend probeerde ik het opnieuw en lukte het een paar keer.

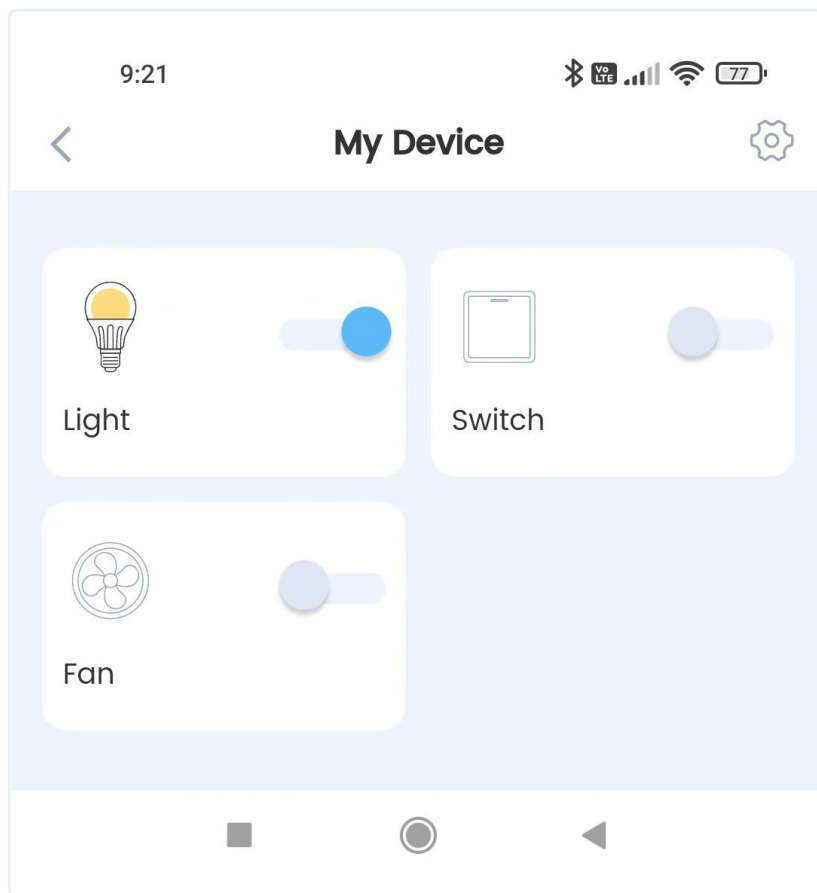
Als het lukt, hoor je een ping-geluid en verschijnt er een microfoon op het scherm. Je hebt nu zes seconden om een commando uit te spreken. Na zes seconden inactiviteit zegt het apparaat: "Timeout, see you next time" en keert terug naar de modus waarin het stond voordat het werd geactiveerd. Ik vond het erg moeilijk om het apparaat te wekken, maar als dat eenmaal gelukt is, herkende het mijn commando's zonder problemen. Opmerkelijk.

Radar

Ik ontdekte een interessante functie toen ik met een verse kop koffie terugkwam. Het display had zich uitgeschakeld terwijl ik weg was en ging plotseling weer aan. Dat moet het werk zijn van de ingebouwde radar. Als je een tijdje niet beweegt, schakelt het display ook uit.

Figuur 3. Vier standards/houders maken allerlei toepassingen mogelijk.





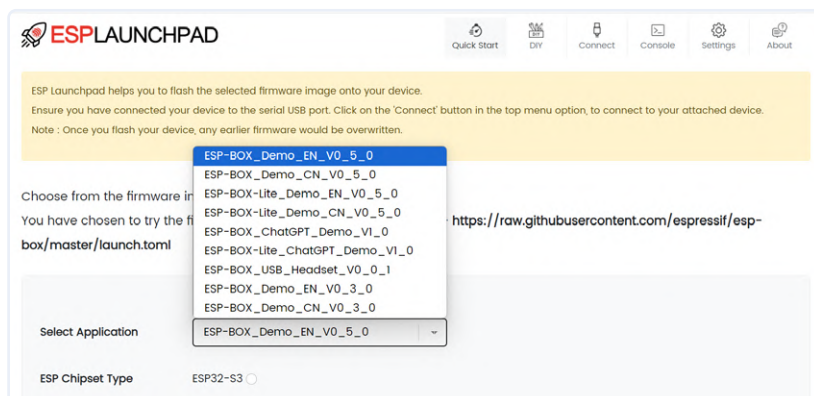
▲
 Figuur 4. De ESP BOX app zorgt voor een verbinding tussen de kit en de Rainmaker cloudservice van Espressif.

De doos in de cloud

Na wat gerommeld te hebben met de ingebouwde 'apps' (schakel het display uit en weer in voor het geval je het in ingeschakelde toestand op de sensorstandaard had aangesloten; anders werkt het niet), ging ik verder met de ESP-BOX app. Je kunt deze verkrijgen via de netwerkpagina door op de knop *To Install APP* te tikken en de QR-code te scannen.

Met de app kun je de kit (en andere apparaten) via een WiFi-netwerk verbinden met de Espressif Rainmaker cloudservice. Nadat je het apparaat hebt toegevoegd aan de app, krijg je toegang tot dezelfde bedieningselementen als op de Device Control-pagina (Light, Switch en Fan maar niet Air, **figuur 4**). Op je telefoon kun je tussen deze functies omschakelen en het resultaat wordt weergegeven op het display van de ESP32-S3-BOX-3. Dit lijkt echter slechts eenrichtingscommunicatie te zijn. Als je een knop op het display aanklikt, wordt de app niet bijgewerkt.

▼
 Figuur 5. ESP Launchpad is een online-tool voor firmwareprogrammering en een seriële terminal.



Als je in de app op het pictogram van de knop tikt in plaats van op de schuifschakelaar, wordt er een instellingenpagina geopend. Hier kun je opgeven welke GPIO-pinnen worden aangestuurd door de knop en wat de spraakcommando's doen. Je kunt de spraakopdrachten ook opnieuw definiëren door een nieuwe opdrachtzin in te typen. In de gebruikershandleiding staan enkele tips voor goede commando's. Helaas kon ik geen nieuwe commando's uitproberen omdat ik het apparaat niet wakker kreeg. Ik heb nergens een document gevonden waarin wordt uitgelegd hoe je je eigen applicaties kunt maken voor gebruik met de ESP-BOX app.

Seriële poort

Je moet weten dat je kunt controleren wat er in het apparaat gebeurt als je het display aansluit op een computer en een seriële terminal opent. Deze functie bleek erg waardevol te zijn voor de volgende paragrafen.

ESP Launchpad

Op de onderkant van de doos van de ESP32-S3-BOX-3 staan drie QR-codes. Eén daarvan is voor ESP Launchpad [3]. Deze app werkt alleen in Chrome (in ieder geval onder Windows) en is bedoeld om de kit te herprogrammeren met andere firmware. Het lijkt erop dat je hier zelfs je eigen firmware kunt publiceren, zodat anderen die ook kunnen proberen (**figuur 5**).

Eerst kon hij om de een of andere reden geen verbinding maken met mijn apparaat, maar na een herstart kreeg ik een lijst met demoprogramma's voorgeschoteld.

ChatGPT-demo

Uit deze lijst selecteerde ik (natuurlijk) ESP-BOX_ChatGPT_Demo_V1_0, uploadde het gretig naar het display en... niets. Het display bleef zwart. Ik probeerde toen de eerstvolgend demo, ESP-BOX-Lite_ChatGPT_Demo_V1_0, maar kreeg een soortgelijk resultaat. Na nog een paar voorbeelden geprobeerd te hebben, begon het me ineens te dagen (bedankt seriële poort!): deze bestanden zijn niet voor de ESP32-S3-BOX-3, ze zijn bedoeld voor eerdere versies van de kit, de ESP32-S3-BOX (zonder '-3') en de ESP32-S3-BOX-Lite. Hopelijk is dit probleem opgelost tegen de tijd dat je dit leest.

Eigen toepassingen bouwen

Op dit punt bevond ik me op GitHub, in de repository die alle drie de versies van de ESP32-S3-BOX ondersteunt [4]. Waarom hebben ze geen QR-code voor deze pagina in het groot op de doos geprint? Het wordt ook niet vermeld in de gebruikershandleiding. Deze repository heeft veel informatie over de kit en ook de broncode van de voorbeeldprogramma's.

ESP-IDF V5.1 of hoger nodig

Er zijn geen voorgecompileerde eenvoudig uit te proberen exe-bestanden, tenminste ik heb ze niet gevonden, dus je moet die zelf bouwen. Hiervoor moet je eerst de

repository klonen. Je hebt ook ESP-IDF V5.1 of hoger nodig.

Het compileren en uploaden naar het apparaat duurde even, maar toen dat eenmaal klaar was, was ik eindelijk terug bij het begin van deze bespreking, toen het display nog werkte. Tot mijn verbazing leek de spraakbesturing beter te reageren en was het gemakkelijker om het apparaat te wekken. De softwareversie was nog steeds V1.1.1, maar mijn ESP-IDF-versie was 'v5.11-dirty' geworden (was 'v5.2-dev-2164-g3bafd5fff7-dirty').

ChatGPT-demo, tweede poging

Nu ik een ontwikkelomgeving had geïnstalleerd voor de ESP32-S3-BOX-3, besloot ik om te proberen de ChatGPT-demo te bouwen volgens de meegeleverde instructies. Dat lukte prima.

Na het compileren en uploaden van het programma naar de display startte de demo normaal op en gaf hij instructies over hoe hij geconfigureerd moest worden [5]. Toen hij klaar was, maakte hij verbinding met mijn WiFi-netwerk en kon ik ermee gaan praten. De chatbot antwoordde echter nooit op mijn vragen, maar toonde in plaats daarvan het bericht "API Key is not valid". In de seriële terminal vond ik deze regel:

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

Het blijkt dat je krediet nodig hebt op je ChatGPT-account om API-sleutels te kunnen gebruiken. Mijn tegoed was verlopen. Je krijgt wat gratis als je voor de eerste keer een account aanmaakt (telefoonnummer nodig). Omdat ik geen (betaald) account wilde aanmaken, ben ik hier gestopt.

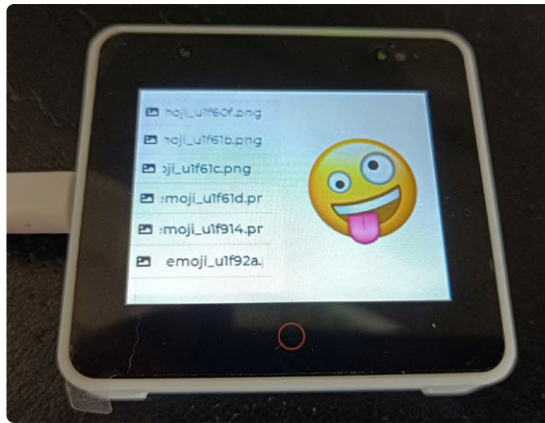
Image Display-voorbeeld

Tot slot besloot ik een laatste voorbeeld te proberen, het Image Display. De beschrijving zegt niet wat het doet, dus dat zou een leuke verrassing kunnen zijn. Dit voorbeeld is een beetje kleiner dan de anderen en daarom gaat het compileren en uploaden sneller. Het resultaat? Een lijst met zes smileys waaruit je er een kunt kiezen om weer te geven (figuur 6).

Complex en veel te bieden

Deze review heeft me iets meer tijd gekost dan ik had verwacht toen ik begon. De reden hiervoor is de complexiteit van het product. De ESP32-S3-BOX-3 heeft zeker veel te bieden, maar om er iets nuttigs uit te halen, moet je tijd en moeite investeren.

De plek om te beginnen is de GitHub-repository en niet de QR-codes op de achterkant van de doos. Op GitHub vind je niet alleen de gebruikershandleiding, maar ook andere nuttige documenten en voorbeeldprogramma's. Het had me een hoop tijd bespaard als ik dit meteen had geweten.



Figuur 6. Kunstmatige intelligentie der dingen (AIoT), nog een lange weg te gaan?

Hoewel er vrij veel informatie bestaat, voelt het product nog steeds een beetje onvolledig aan. Een tutorial over hoe om te gaan met de ESP-BOX app of hoe je een spraakgestuurde toepassing van de grond af kunt opbouwen, zou op prijs worden gesteld. Er staat niet veel in over AIoT, maar daar gaat dit product toch over? Wat de hardware betreft, kan ik alleen maar zeggen dat het echt heel mooi gedaan is. De kwaliteit van het geheel is goed, de beugels passen perfect, de knoppen werken, het display ziet er goed uit, de aanraakfunctionaliteit is soepel en alles zit verpakt in een hoogwaardige doos. Het scherm, met of zonder beugel, is een coole gadget op een bureau of in een slaapkamer of woonkamer. ◀

230555-03

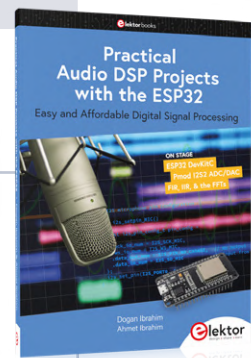
Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via clemens.valens@elektor.com of naar de redactie van Elektor via redactie@elektor.com.



Gerelateerde producten

- > ESP32-S3-BOX-3
www.elektor.nl/20627
- > Arduino Nano ESP32
www.elektor.nl/20562
- > Praktical Audio DSP Projects with the ESP32
www.elektor.nl/20558



WEBLINKS

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

Essentiële tools voor je werkplek

overwegingen en tips van Espressif-technici

De elektronica-werkplek: dat is waar de magie plaatsvindt. Ben je benieuwd hoe andere creatieve engineers hun werkplek hebben ingericht? Ben je op zoek naar tips over het inrichten en organiseren van je eigen werkplek? Een paar technici van Espressif gunnen je een kijkje in de keuken.



Omar Chebib

locatie: Shanghai, China

Organiseer en noteer

Mijn werkplek is mijn bureau thuis. Het is misschien niet groot, maar ik probeer het altijd opgeruimd en georganiseerd te houden. Ik bewaar er alleen de benodigde gereedschappen, apparaten en onderdelen die ik nodig heb voor waar ik op dat moment mee bezig ben. Door alles consequent op de juiste plek te leggen, kan ik efficiënt werken. Dit vereist natuurlijk ook een goed georganiseerde plank met waar pure elektronica, through-hole en SMD-componenten, embedded tools, een logic analyzer, wat ESP32-chips of zelfs I²C-apparaten een plaatsje hebben. Zo kon ik aan verschillende projecten werken met logische poorten, een Z80 8-bit processor, FPGA's, through-hole en SMD-soldeerwerk (inclusief de 'angstaanjagende' BGA-behuizing). Tip: bewaar wat de hardware betreft alleen die gereedschap-

pen, apparaten en kabels die je nodig hebt voor je huidige project op je werkbank, pak alleen nieuwe gereedschappen als je die nodig hebt en ruim na afloop alles op. Wat de software betreft: bewaar altijd de verschillende versies van je werk, al is het alleen maar lokaal. Het is nooit te laat om met versiebeheer van een lopend project te beginnen. Meer in het algemeen: als je wisselt tussen sterk verschillende projecten, maak dan een notitie waarin staat wat je al gedaan hebt, waar je gebleven bent en wat de volgende stap wordt. Dit helpt enorm om de draad weer op te pakken. Huidige projecten: in mijn vrije tijd heb ik vanaf nul een Zeal 8-bit computer ontworpen – een Z80-gebaseerde computer – inclusief printontwerp en software. In mijn werktijd implementeer ik een ESP32-C3-emulatie op QEMU.



Essentiële tools

- > Logic analyzer
- > Multimeter
- > Oscilloscoop
- > Elektronische microscoop
- > Soldeerstation + vloeimiddel
- > Een goede dampafzuiger!
- > Linux-computer

Verlanglijstje

- > Reflowplaat voor solderen
- > Betere microscoop
- > Betere stoel



Jeroen Domburg
locatie: Singapore

Georganiseerde rommel

Mijn werkbank is doorgaans 'chronologisch geordend.' Anders uitgedrukt: een grote berg spullen met de meest recent gebruikte zaken bovenop en de oudste onderop. Het wordt gereorganiseerd als ik geen ruimte meer heb om te werken, als ik vergeet wat onderop ligt of als er spullen op de stapel liggen die ik weg moet gooien. Als jij ook zo iemand bent, hoeft je je niet te schamen voor een 'rommelige werkplek': tenzij je die met iemand anders deelt, mag je werkplek eruitzien zoals jij dat wilt! Het lijkt erop dat je het beste functioneert door je geheugen te gebruiken om je gereedschap met zo min mogelijk moeite op te bergen en terug te vinden, waardoor je waarschijnlijk efficiënter bent dan de 'altijd-opgeruimd-typen.' Langdurig bewaren is een ander verhaal. Ik hanteer het motto 'als ik niet weet dat ik het heb, kan ik het net zo goed niet hebben,' maar ik wil niet alles wat ik heb actief moeten onthouden. Daarom bewaar ik al mijn onderdelen, oude projecten enzovoort in genummerde dozen, met een (zeer goed opgeslagen!) bestand op mijn PC waarin ik kan opzoeken wat waar ligt. Dat maakt het makkelijk om een onderdeel of tool te vinden als ik het nodig heb. Over gereedschap gespro-

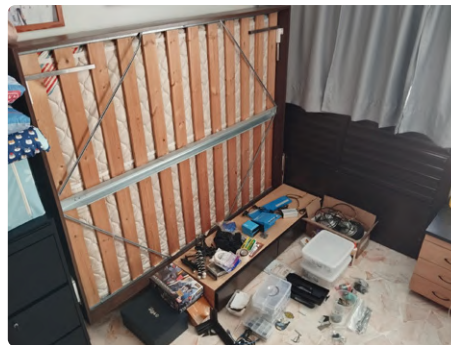
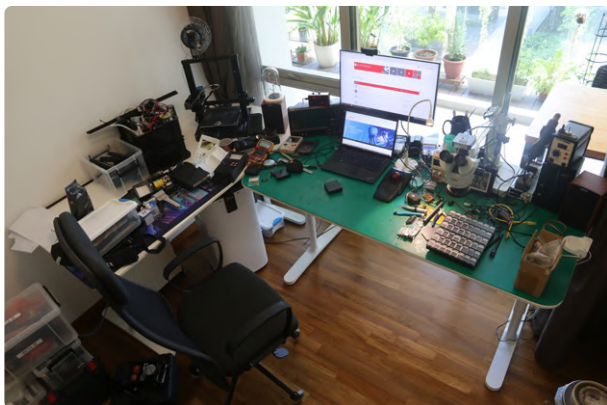
ken: als je van plan bent om SMD's te gaan solderen, koop dan een goede microscoop; als je een stereo-exemplaar kunt krijgen, nog beter! Ik vind dat je bij gebruik van een microscoop een vastere hand hebt, waardoor je zonder problemen kleinere componenten kunt solderen. Natuurlijk helpt het ook om een goede soldeerbout te hebben, en goede soldeerbouten worden steeds goedkoper. Zelfs als de Weller WTCP van je vader destijds het beste van het beste was, zal een moderne, goedkope Pinecil of TS-100 deze doen verbleken, en dat voor een fractie van de prijs.

Essentiële tools

- Computer. Je kunt geen chips programmeren zonder.
- Modern soldeerstation. Mijn huidige favoriet is een Aixun T3A met een T245-handvat.
- Soldeerdamp-afzuiger.
- Optische binoculaire microscoop
- Oscilloscoop
- Logic analyzer
- Labvoeding

Verlanglijstje

- Betere labvoeding
- Ruimte om 'vieze' dingen te doen (zoals 3D-harsprinten, schilderen enzovoort)
- Meer vrije tijd om met elektronica te 'spelen'



Kai Jie Tan
locatie: Singapore



Begin gewoon

Een professionele, volledig ingerichte werkplek met alle apparatuur op je werkbank en een gaatjesboard vol gereedschap binnen handbereik – dat is het doel waarnaar we streven, maar om te beginnen heb je dat niet nodig. Jaren geleden begon ik met niet meer dan een gereedschapskist en een elektronicaset die iedereen verplicht moest kopen op mijn ingenieursopleiding. Die stonden altijd in een kast in mijn kamer vanwege ruimtegebrek. Als jongere in Singapore, die in een slaapkamer in het appartement van zijn ouders woonde, is ruimte altijd een probleem. De huizenprijzen zijn hoog en er zijn lange wachttijden voor betaalbare woningen, dus de meeste jongeren verlaten het ouderlijk huis pas als ze getrouwd zijn. Omdat ik bij elk project dat ik bouw snel ruimte tekort kom, vooral als het grootschaliger wordt (denk aan cosplay-rekwisieten en e-scooters), heb ik overal gereedschap rondslingeren. Een voorbeeld hiervan is mijn 3D-printer die bovenop een vitrinekast staat, twee meter boven de grond. Op het hoogtepunt van de Covid-19 restricties had ik geen

toegang tot een Fab Lab meer en moest ik alles thuis doen. Dit leidde ertoe dat ik een opklapbed bouwde met hardware van AliExpress. Dat was een openbaring, want zo kon ik mijn werk veilig onder mijn bed bewaren – snel toegankelijk zonder veel gesleep in vergelijking met alles in gereedschapskisten te moeten bewaren. Het is de bedoeling dat dit project een tweede fase krijgt, met planken op vier stangen waarbij die planken in ingeklapte toestand als poten onder het bed fungeren. Daar moet ik nog een nachtje over slapen (...).

Snel naar het heden: mijn werkplek lijkt in de verste verte niet op zo'n professionele Fab Lab-setup waarvan ik alleen maar kan dromen. Ik heb gewoon veel meer gereedschapskisten en dozen met elektronica dan vroeger; maar ik had zelden het gevoel dat dit me belemmerde om iets te bouwen wat ik wilde. Mijn advies: naarmate je meer projecten ontwikkelt, zul je meer gereedschap verzamelen en al doende ook versleten gereedschap vervangen. Je hoeft echt niet te wachten op een volledig ingerichte werkruimte om te beginnen. Begin gewoon.

Essentiële tools

- Fluke 115 multimeter
- Proskit 7-in-1 draadstripper
- 3D-printer (het is erg fijn om er een te hebben)

Verlanglijstje

- Soldeerdamp-afzuiger
- LuchtfILTER
- Behuizing voor 3D-printer
- Een garage om gereedschap goed op te bergen

**Pedro Minatel**

locatie: Braga, Portugal

Een plek om je terug te trekken

Ik noem mijn elektronica-werkplek mijn heiligdom – een plek waar ik dingen kan creëren en uitvinden, ook al zal het meeste daarvan altijd werk in uitvoering blijven. Ik streef ernaar om het meeste van mijn gereedschap georganiseerd te houden, maar soms is een beetje rommel onvermijdelijk. Ik ben trots op mijn gereedschap; sommige tools zijn ouder dan ik, zoals mijn schuifmaat uit de jaren '50. Mijn favoriete en een van de belangrijkste tools op mijn elektronische werkbank is mijn

Rohde & Schwarz RTB2002-oscilloscoop. Tip: organiseer je componenten! Ik gebruik het liefst goedkope bakjes en opbergdoosjes (zoals voor pillen). Voor ESD-gevoelige componenten is het echter essentieel om ze in antistatische zakjes op te bergen. Ik gebruik een etiketteermachine om er zeker van te zijn dat al mijn onderdelen correct zijn opgeborgen. Huidig project: ik werk weer aan een development board, dit keer gebaseerd op de ESP32-C6.

Essentiële tools

- Oscilloscoop (met logic analyzer)
- Betrouwbare multimeter
- Goed soldeerstation voor SMD-onderdelen
- Goed basisgereedschap
- 3D-printer

Verlanglijstje

- Spectrumanalyzer met VNA (bijvoorbeeld een SVA1032X)
- Tafelmodel pick&place-machine
- Reflow-oven
- Digitale voeding (60 V/10 A)

**Jakob Hasse**

locatie: China, Shanghai

Mijn werkplek is een Mahjong-tafel

Mijn werkplek wordt voornamelijk gebruikt voor allerhande soorten soldeerwerk, het reflow-solderen van printen en voor diverse reparaties van elektronische apparaten. Ik heb een T12-achtig soldeerstation dat heel goed werkt dankzij het verwarmingselement direct in de punt. Soldeerdampen zijn echt niet gezond, daarom heb ik ook voor ongeveer \$ 150 een dampafzuiger gekocht op Taobao, een Chinees online-platform dat eigendom is van Alibaba – een goede investering in mijn gezondheid. Voor het reflow-solderen van printen heb ik een kookplaat die zo goedkoop was dat ik me de prijs niet kan herinneren. Maar als het gaat om meetapparatuur en 'domme' metalen gereedschappen, koop ik kwaliteit. Geloof me, het leven is te kort voor slecht gereedschap!

Omdat ik momenteel in Shanghai woon en ruimte hier een luxe is, meet mijn werkplek slechts 80×80 cm. Het is een tafeltje dat ik op Taobao heb besteld. Het tafelblad is gemaakt van bamboe, een heel hard en duurzaam materiaal. Het blijkt echter dat de coating oplost als

die in contact komt met IPA. Het volgende dat ik ga kopen is dus zeker een echte siliconen mat om mijn werkblad niet alleen te beschermen tegen solderen, maar ook tegen isopropyl-alcohol. Er is één voordeel van de kleine 'vormfactor' van mijn werkruimte: als ik bezoek heb, maak ik het tafelblad leeg en breng ik het tafeltje naar de woonkamer. De afmetingen zijn zo perfect om Mahjong te spelen.

Essentiële tools

- Soldeerstation (T12-kloon)
- Weinan-soldeerdampafzuiger
- Vectech-kookplaat
- iFixit reparatieset
- Maisheng MS-605D 300W-labvoeding
- Isopropyl-alcohol
- Gossen-Metrawatt Metraline DM62 multimeter
- LUX-schroevendraaiers voor de gangbare schroeftypes
- Knippex multifunctionele tang

Verlanglijstje

- Echte siliconenmat
- Oscilloscoop
- Hetelucht-reworkstation
- Reservepincet



Toon ons je werkplek!

Wil je dat de redactie van Elektor jouw elektronica-werkplek publiceert op elektormagazine.com of in de gedrukte *Elektor*-uitgave? Gebruik onze online Workspace Submission Form om meer te vertellen over de plek waar je innoveert, ontwerpt, debugt en programmeert! www.elektormagazine.com/workspaces

230579-03



Het ESP RainMaker-verhaal

hoe we “jouw” IoT-cloud bouwden

Amey Inamdar, Espressif

Met de cloud verbonden apparaten zijn een belangrijk onderdeel van het IoT. Echter, met cloudproviders die een volledige oplossing bieden, verliezen de fabrikanten van die apparaten controle over de data. Het bouwen van een eigen cloudoplossing vereist daarentegen een aanzienlijke inspanning. Bij Espressif zagen we deze problemen en besloten we een oplossing te bieden die het beste van beide werelden biedt.

De 'I' in IoT staat voor internet. Het internet - en daarmee ook de cloud - is een integraal onderdeel van communicerende objecten om de voordelen van connectiviteit te benutten. Apparaten die de cloud gebruiken om gegevens te verzenden en opdrachten te ontvangen, bieden niet alleen toegevoegde waarde aan eindgebruikers, maar ook aan hun fabrikanten. Het hebben van een eigen IoT-cloud brengt echter beveiligings- en privacyoverwegingen met zich mee voor de consument en technische uitdagingen voor de eigenaar. Het was een logische keuze voor veel fabrikanten om in zee te gaan

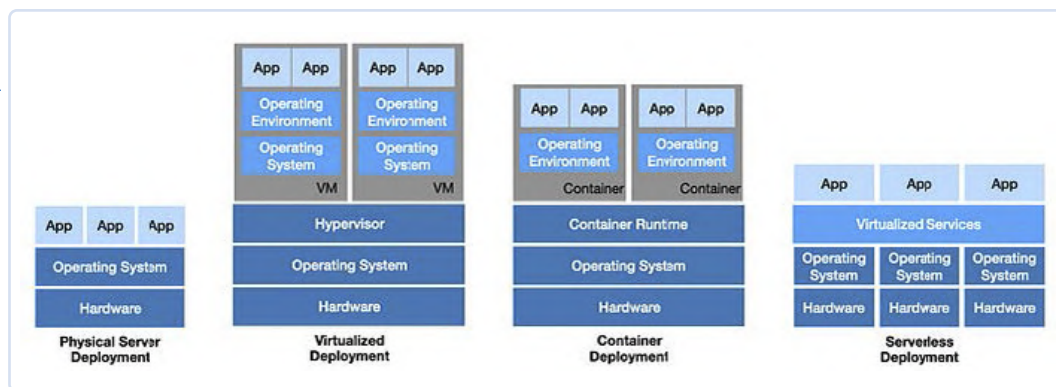
met leveranciers van volledige IoT-oplossingen die cloud-, embedded- en mobiele applicaties boden. Ze kwamen er echter al snel achter dat deze oplossing onvoldoende differentiatiemogelijkheden bood en niet de vereiste controle over de data gaf. Sommige fabrikanten die de mogelijkheid hadden besloten hun eigen cloud te bouwen en te onderhouden, maar dit vereist een aanzienlijke inspanning.

Bij Espressif zagen we deze problemen en besloten we een oplossing te bieden die het beste van beide werelden biedt. Aan de ene kant wilden we een cloud met alles erop en eraan, en aan de andere kant wilden we onze klanten volledig eigendom, controle en aanpasbaarheid bieden, zonder dat ze vanaf niets hoefden te beginnen. Dit was de aanzet tot ESP RainMaker.

Kies de juiste architectuur

De eerste keuze die we moesten maken was welke architectuur we wilden gebruiken (**figuur 1**).

Je hebt misschien wel eens horen zeggen dat de cloud gewoon de computer van iemand anders is. Dit is inderdaad zo. Maar waar het om gaat, is hoe die computer wordt gebruikt om je applicatie te hosten. Met de opkomst van virtualisatietechnologieën worden krachtige servers gebruikt om meerdere applicaties volledig geïsoleerd van elkaar te hosten op dezelfde hardware. Met virtuele machines (denk aan VMWare) kun je meerdere instanties van besturingssystemen op dezelfde hardware draaien, terwijl je met containertechnologie (Kubernetes, Docker) parallelle besturingsomgevingen op dezelfde hardware kunt hebben.



Figuur 1. Evolutie van cloudarchitecturen.

Dit klinkt allemaal heel mooi, maar je moet wel nadenken over welke software je moet draaien en onderhouden en hoe je moet schalen met variërende belastingen (dit is de taak van zogenaamde DevOps-engineers). Toen we besloten onze klanten hun eigen IoT-cloud te geven, wilden we ze niet belasten met deze taken. Dat is waar de relatief nieuwe techniek van de cloud-zonder-server uitkomst bood met mogelijkheden voor het bouwen van een eenvoudig te onderhouden IoT-cloud.

Een architectuur zonder servers maakt servers niet overbodig. De abstracties worden gewoon op een hoger niveau aangeboden. Je krijgt bijvoorbeeld een MQTT-broker zonder dat je je zorgen hoeft te maken over welke MQTT-software hij gebruikt, hoeveel verbindingen hij aankan of op welk platform hij draait. Amazon Web Services (AWS) levert zulke diensten, waardoor we onze droom IoT-cloud konden implementeren. Van de vele beschikbare diensten biedt AWS IoT Core een MQTT-broker, DynamoDB biedt een NoSQL database, S3 biedt opslag en, belangrijk, Lambda biedt een "function-as-a-service", waarbij je algoritmes kunt implementeren zonder je zorgen te maken over waar deze moeten worden uitgevoerd.

Met de op AWS managed services gebaseerde implementatie lijkt ESP RainMaker op een combinatie van het configureren van verschillende services en hun interactie, en het gebruik van Lambda-functies. Belangrijk is dat het geheel past in elk AWS-account. Dit geeft ons de mogelijkheid om een IoT-cloud te maken die klanten in hun eigen AWS-account kunnen implementeren, zodat ze volledige controle hebben over de data en het systeem.

Schaalbaarheid

Een IoT-cloud heeft vaak veel objecten en gebruikers (via mobiele apps of andere clients zoals spraakassistenten) die verbinding maken met de cloud en berichten uitwisselen. Afhankelijk van de branche varieert dit aantal. Voor fabrikanten van consumentenapparatuur kunnen de aantallen in de miljoenen lopen. Daarom is het belangrijk om ervoor te zorgen dat de cloud kan worden opgeschaald naar grote aantallen.

Met AWS managed services mag je aannemen dat je je geen zorgen hoeft te maken over de schaalbaarheid van de cloud. Hoewel dit groten-deels waar is, is het ook zo dat een service bepaalde limieten kan hebben. Het maximale aantal Lambda-functies dat gelijktijdig kan worden uitgevoerd in een AWS-account is bijvoorbeeld beperkt. Dit vereist een zorgvuldig ontwerp om ervoor te zorgen dat het systeem zoveel mogelijk berichten goed kan afhandelen dankzij een adequaat wachtrijsysteem. Het vereist ook dat objecten en mobiele apps op een goede manier met problemen kunnen omgaan.

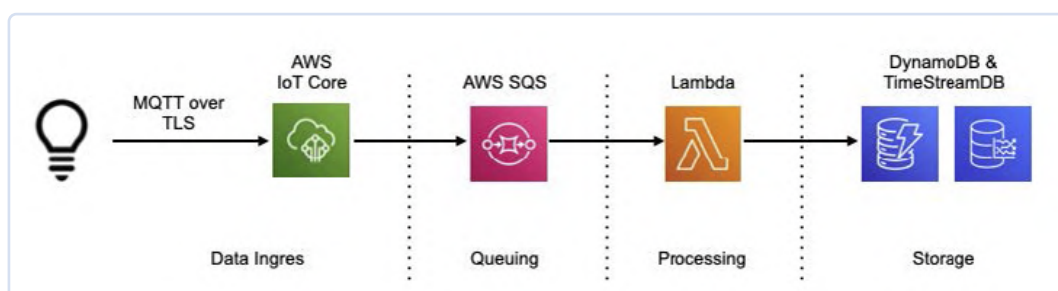
ESP RainMaker kan momenteel meer dan 3,5 miljoen objecten en gebruikers aan die verbinding maken en communiceren via de cloud (figuur 2).

Operationele kosten

In een traditionele cloudarchitectuur betaal je meestal voor computing, opslag en geheugen voor je virtuele machines. In de serverloze architectuur verandert de eenheid van facturering. Je moet betalen voor de daadwerkelijk gebruikte resources, zoals het aantal verbonden objecten en de totale verbindingstijd, het aantal MQTT-berichten, het aantal keer dat de database gelezen/geschreven is, de hoeveelheid gegevens die opgeslagen is in S3 Storage en de resources die verbruikt zijn door uitgevoerde Lambda-functies. Hoewel dit een pay-as-you-go prijs is, kan het veel meer kosten als je architectuur en implementatie niet geoptimaliseerd zijn.

De operationele kosten voor de klanten waren een van de belangrijkste criteria bij het ontwerp van ESP RainMaker. Het doel was gemakkelijk te bepalen. We wilden een cloud backend ontwikkelen dat ook zou werken voor een slimme gloeilamp – waarschijnlijk het goedkoopste IoT-object. Het heeft veel moeite gekost om de AWS-services zo te kiezen dat ESP RainMaker dit doel kon bereiken. Verder is het flink getuned om het aantal database lees/schrijfbewerkingen te optimaliseren; de implementatietaal moest zo weinig mogelijk resources gebruiken, enzovoort.

ESP RainMaker heeft momenteel meerdere klanten die daadwerkelijk goedkope IoT-lampen verkopen die voldoen aan de operationele kostenvereisten.



Figuur 2. Een voorbeeld van verwerking van objectberichten in ESP Rainmaker.

Figuur 3. ESP RainMaker componenten.



Gegevensbeveiliging en privacy

Het gebruik van AWS managed services hielp ook bij het waarborgen van databeveiliging. Ze bieden standaardmethoden voor authenticatie, autorisatie, versleutelde data-opslag, firewalls enzovoort. De AWS IoT Core MQTT-broker biedt bijvoorbeeld objectverificatie op basis van X.509-certificaten, waardoor een apparaat de cloud kan verifiëren en de cloud een apparaat. IAM-beleidsregels zorgen voor fijnmazige autorisatiecontrole. Web Application Firewall (WAF) biedt bescherming tegen DoS- en DDoS-aanvallen op de RESTful service-eindpunten.

ESP RainMaker gebruikt al deze beveiligingsmethoden op de voorgeschreven manier en zorgt voor de juiste authenticatie en autorisatie met behulp van standaardmethoden; er is geen onnodige privilege-escalatie. We hebben ook samengewerkt met een test- en certificeringsbedrijf om ervoor te zorgen dat ESP RainMaker voldoet aan de vereisten voor dataprivacy. Hierdoor is het voor klanten eenvoudig om te voldoen aan GDPR-compliance wanneer ze ESP RainMaker inzetten in hun eigen account.

De cloud alleen is niet genoeg

Hoewel de cloud een belangrijk onderdeel is van de complete IoT-oplossing, is het niet genoeg. Gezien vanuit het verbonden IoT-object is het, naast het hebben van de cloud, net zo belangrijk om over goede firmware te beschikken. Ook zijn er apps nodig voor de configuratie en de bediening. En dan is er nog de integratie met spraakassistenten, die de meest natuurlijke manier zijn geworden voor gebruikers om te communiceren met verbonden apparaten in het slimme huis (**figuur 3**). Bij Espressif hebben we al deze componenten gemaakt. De ESP RainMaker SDK is volledig open source en eenvoudig te integreren met behulp van de vele voorbeelden. De iOS en Android voorbeeldapps zijn ook volledig open source. ESP RainMaker ondersteunt ook Alexa en Google Assistant skills.

De cloud moet ook gebruikt kunnen worden voor het verbeteren van een product wegens voortschrijdend technisch- en zakelijk inzicht. Hiervoor hebben we ESP-Insights gebouwd die remote device logs, crash analytics, metrics monitoring en, niet te vergeten, uitgebreide dataqueries mogelijk maakt. Dit is allemaal beschikbaar via het ESP RainMaker webdashboard, dat ook OTA-updates, objectgroepering, rolgebaseerde toegangscontrole en bedrijfsinzichten biedt.

Vorbereiden op de toekomst

Het Matter protocol is bedoeld voor de broodnodige standaardisatie voor smart home apparaten. Matter is een lokaal netwerkprotocol dat zich niet speciaal met de cloud bemoeit. Toch speelt de cloud een rol bij objectbeheer en de toegang op afstand tot de objecten. ESP RainMaker ondersteunt Matter zodat je je eigen Matter ecosysteem kunt creëren. Een combinatie van ESP RainMaker cloud backend en telefoon apps implementeren Matter, inclusief de complete PKI-infrastructuur die hiervoor nodig is. De databases voor object, gebruiker en ACL zijn veilig gesynchroniseerd over meerdere apps. Als je een

smart home controller hebt die verbinding maakt met ESP RainMaker, kunnen je mobiele apps niet alleen jouw apparatuur op afstand bedienen, maar ook alle andere Matter objecten.

We hebben ook ondersteuning aangekondigd voor de integratie van Mesh-Lite netwerktopologie met ESP RainMaker. Mesh-Lite is een speciaal WiFi mesh netwerk voor IoT-apparaten om een groter bereik en dode hoeken in huis te dekken. Op de ESP32 familie gebaseerde IoT-objecten vormen toegangspunten om verbinding te maken met andere objecten en vormen zo een mesh-topologie. Met ESP RainMaker kunnen gebruikers eenvoudig een mesh-netwerk opzetten en de nodes erin hangen.

Aan de slag

We hebben de ESP RainMaker-cloud gebouwd met overwegingen die elke fabrikant zou maken bij het opzetten van een eigen IoT-cloudplatform. Het belangrijkste is dat Espressif dit aanbiedt als een volledig rebrandable en aanpasbaar platform. Voor ontwikkelaars bieden we ook een door Espressif beheerde ESP RainMaker instance, samen met apps (beschikbaar in de diverse app stores). Je kunt het webdashboard gebruiken voor objectbeheer en OTA firmware upgrades uitvoeren. Hiermee kun je niet alleen de functionaliteit evalueren, maar ook je projecten doen voor persoonlijk gebruik.

Om aan de slag te gaan, kun je een van de Espressif-ontwikkelboards gebruiken en de stappen in de Get Started guide volgen [1].

230621-03



Over de auteur

Amey Indamar is hoofd technische marketing bij Espressif. Hij heeft 20 jaar ervaring op het gebied van embedded systemen en communicerende objecten, met functies in engineering, productmanagement en technische marketing. Hij heeft met veel klanten samengewerkt om succesvolle communicerende apparaten te bouwen op basis van WiFi- en Bluetooth-connectiviteit.

WEBLINK

[1] Aan de slag met ESP Rainmaker:
<https://rainmaker.espressif.com/docs/get-started>

Bouw van de **ELEKTOR CLOC 2.0** een Elektor-product uitgepakt door Espressif

Jeroen Domburg (Espressif)

De Elektor Cloc 2.0 is een flexibele wekker gebaseerd op een ESP32-Pico-Kit van Espressif. Hij wordt geleverd als een kit met onderdelen die je zelf in elkaar moet zetten. In dit artikel zetten medewerkers van Espressif er een in elkaar en vertellen ze wat ze ervan vinden.

Hoe word jij 's morgens wakker? Laat je je door je smartphone wekken, heb je nog een ouderwetse wekkerradio naast je bed, word je gewekt door een kraaiende haan of laat je gewoon de gordijnen open, zodat de vroege zonnestralen je gezicht strelen? Misschien heb je het gevoel dat al deze methoden een paar nadelen hebben. Met je telefoon kun je niet echt met een half oog zien hoe lang je nog kunt slapen, de wekkerradio heeft waarschijnlijk maar één alarm dat je ook in het weekend of op vakantie 's morgens wekt als je het niet uitzet, en hoewel de haan en de zonnestralen goedkoop en over het algemeen betrouwbaar zijn, is het vrij moeilijk om een andere wektijd in te

stellen. Over het algemeen is geen van de oplossingen 'werkelijk' flexibel.

Maar als je die flexibiliteit op prijs stelt, is de beste optie natuurlijk om je eigen wekker te bouwen. Als je er zelf een ontwerpt, kun je elke gewenste functionaliteit inbouwen. Alarmtijden per week, WiFi-connectiviteit, leuke wektoontjes, de kamertemperatuur verhogen, de koffiezetter inschakelen – alles wat je hart begeert.

Cloc 2.0 brengt soelaas

Maar wat als je niet de mogelijkheden, de tijd of de energie hebt om het hele traject van architectuur, hardware- en softwareontwerp, productie enzovoort te doorlopen? Elektor kan je helpen met de Cloc 2.0. Dit ontwerp heeft een dubbel 7-segment LED-display die de momentele tijd en de alarmtijd tonen, een WiFi-verbinding zodat hij altijd de juiste tijd weergeeft en veel configureerbare opties mogelijk maakt. Verder heeft hij een IR-afstandsbediening waarmee je bijvoorbeeld een geluidsinstallatie kunt inschakelen als het alarm afgaat. Het beste van alles is dat het is gebaseerd op een ESP32-Pico-Kit en dat de software open-source is, wat betekent dat het eenvoudig is om een functie toe te voegen als je er een mist. Elektor was zo aardig om ons een kit op te sturen, zodat we hem konden beoordelen. Het volledige project van Cloc 2.0 is gepubliceerd op de Elektor Labs-website [1].

De kit

We kregen de Cloc 2.0 als kit, en dus ontvingen we een berg onderdelen (**figuur 1**). Het meest opvallend was de rode Hammond-behuizing, met het logo van Elektor Labs erop gedrukt. Behalve dat het er goed uitziet, is de combinatie van een rode kast en een rood LED-display een goed idee, omdat het contrast van de LED's daardoor verbetert, zodat de klok beter af te lezen is. De rest van de componenten zat in meerdere zakjes, gelabeld met de waarden van de componenten die erin zaten. Interessant detail: elk zakje bevat een groep duidelijk verschillende componenten. Geen enkel zakje bevatte bijvoorbeeld meer dan twee

Figuur 1. De kit wordt geleverd in slim gelabelde zakjes, plus de Hammond-behuizing.



weerstandswaarden, en je hoeft dus niet naar kleurringen te staren om erachter te komen welk onderdeel welke waarde heeft. Dit is een slim idee dat plastic bespaart en bovendien het vinden van het juiste onderdeel makkelijker maakt; hulde aan het Elektor-team voor het bedenken van deze methode! Opmerkelijk is dat de kit geen knoppen heeft die toegankelijk zijn vanaf de buitenkant van de behuizing; de achterliggende gedachte is dat je misschien zelf nog ergens een paar mooie hebt liggen.

Waar is de documentatie?

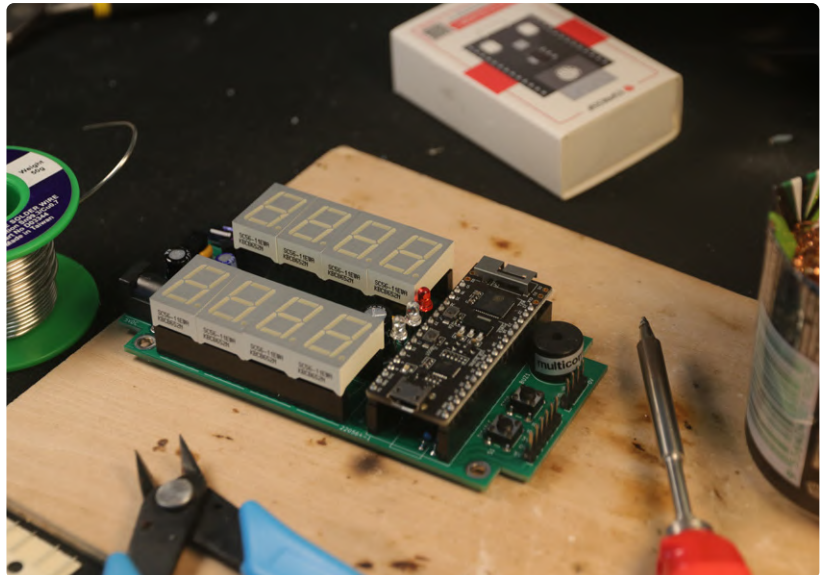
Het vinden van de documentatie om de kit te bouwen had makkelijker kunnen zijn. De Elektor-site heeft pagina's verspreid over drie domeinen (elektorlabs.com, elektor.com en elektormagazine.com) met verschillende downloads en tekst op elke pagina. Dit is een beetje verwarrend en het zou handig zijn geweest als er op zijn minst een stuk papier was waarop stond waar je alle benodigde downloads en documentatie van de kitt kon vinden. Toen we eenmaal alles hadden gevonden, bleek het echter behoorlijk compleet te zijn: het Elektor-artikel over het ontwerp is inbegrepen, net als alle downloads en zelfs een video [2] over hoe je alles het beste in elkaar kunt zetten.

Met de documentatie in de hand is het volbouwen van de print een fluitje van een cent (**figuur 2**). Alle componenten zijn 'through hole', dus solderen is eenvoudig en de video suggereert een volgorde voor het solderen van de componenten die voor ons heel goed werkte.

Eén ding om op te merken is dat de headers voor de 7-segment LED-displays van hetzelfde type zijn als die voor de ESP32-Pico-Kit. Dat is een probleempje omdat de pinnen van de LED-displays dunner zijn en niet altijd goed contact maken. Dit is vergeeflijk omdat er geen headers voor dunnere pinnen lijken te bestaan met dezelfde hoogte als deze; bovendien is dit eenvoudig te verhelpen door de pinnen van de displays een beetje te buigen.

De ESP32-Pico-Kit programmeren

Nadat het board in elkaar is gezet, moet de ESP32-Pico-Kit geprogrammeerd worden. Dit kostte een paar stappen om de juiste bibliotheken en het ondersteuningspakket voor het board geïnstalleerd te krijgen, maar was over het algemeen ook niet al te ingewikkeld. Dit proces had wat eenvoudiger kunnen zijn door bijvoorbeeld ESP-Launchpad [3] te gebruiken, maar de huidige opzet heeft het voordeel dat het de sketch vanaf de broncode bouwt: dit betekent dat als je iets vindt dat je wilt veranderen in de code, je alles al hebt wat nodig is om de firmware opnieuw te bouwen en te flashen.



Mechanische constructie

Wat betreft de plaatsing van de print in de behuizing: dat kan je op verschillende manieren doen. Elektor geeft een boormal en in recente kits wat extra hardware, zodat je het op de 'standaard' manier kunt bouwen. Het is echter zeker mogelijk dat je dingen wilt of zelfs moet aanpassen om ze geïnstalleerd te krijgen op de manier die jij wilt. Afhankelijk van de vorm en grootte van de knoppen die je wilt gebruiken, kun je hier bijvoorbeeld je eigen plan maken.

De voeding

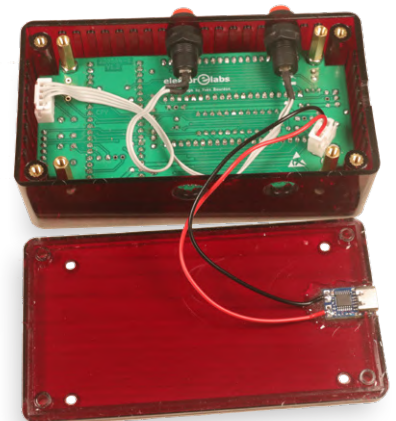
In onze 'Big Box of Power Supplies' konden we geen 5V-voeding met passende connector vinden, hoewel we zeker weten dat we er tientallen moeten hebben. Wat we wel vonden was een kleine print met een USB-C connector die we konden aanpassen om gewoon 5 V te leveren bij de vereiste 500 mA of meer. Dit heeft als bijkomend voordeel dat onze Cloc geen separate voeding nodig heeft; elke USB-C voeding met kabel is voldoende. Uiteraard hebben we de boormal voor de behuizing aangepast: in plaats van een gat te boren voor de aansluiting, hebben we een stukje plastic weggesneden aan de achterkant van de behuizing waardoor de USB-C connector toegankelijk wordt. Het USB-printje kon daarna aan de achterkant worden vastgemaakt, met epoxy zodat het op zijn plaats blijft.

Voorzichtig boren

In dat verband een wijze raad: als je gaten boort en alles installeert, houd dan rekening met de volgorde ervan en de juiste stand in de behuizing. Omdat wij hier bij Espressif professionals zijn, zijn we ons hier uiteraard terdege van bewust en daarom kunnen we je verzekeren dat de... drainageopeningen... in de bodem van onze behuizing uiteraard van meet af aan deel waren van onze oorspronkelijke plannen (**figuur 3**).

▲
Figuur 2. Omdat alleen through-hole onderdelen worden gebruikt, was het solderen een fluitje van een cent.

▼
Figuur 3. De achterkant van de behuizing biedt voldoende ruimte voor knoppen en kabels, maar je kunt er ook een uitbreidingsprintje monteren.





Figuur 4. Onze Cloc, gevoed via een USB-C adapter.

Uiteindelijk hebben we alles netjes in elkaar gekregen. We hadden nog een paar mooie rode knoppen die goed bij de behuizing pasten. In plaats van de meegeleverde headers hebben we een paar JST-connectoren gebruikt om ze op de print aan te sluiten. We hebben de USB-voedingsprint ook met een JST-connector aangesloten: zo kunnen we de hoofdprint volledig uit de behuizing halen voor eventuele reparaties.

Cloc-configuratie

Toen alles in elkaar zat, was het configureren van de Cloc eenvoudig. Zet hem aan, maak er verbinding mee en ga naar het aangegeven IP-adres, waarna je hem kunt verbinden met het lokale WiFi-netwerk. Vanaf dat moment zal de Cloc elke keer dat hij opstart het IP-adres laten zien dat hij heeft, wat een goed idee is omdat je niet hoeft te rommelen met netwerkscanners of naar de routerconfiguratie moet kijken om toegang te krijgen tot de webinterface. Die is zelf eenvoudig genoeg om nauwelijks een handleiding nodig te hebben, maar wel erg compleet: we hadden de basisfuncties zoals tijdzone en zomertijd in een paar seconden ingesteld.

Al met al vinden we de Cloc (**figuur 4**) heel geslaagd. Het LED-display is goed leesbaar en omdat het rood is, zal het je nachtzicht niet verstoren als je er in het donker naar kijkt. Hij kan vrij worden geconfigureerd via de webinterface en als je hem wilt aanpassen om je te wekken zoals jij dat wilt, is daar ruimte voor: zowel in metaforische zin, gezien de lage instapdrempel die het gebruik van de Arduino-omgeving met zich meebrengt, als in fysieke zin, omdat de Hammond-behuizing waarin hij is gemonteerd ruimte biedt voor bijvoorbeeld een extra print achter de Cloc-print zelf.

Suggesties voor Cloc 3.0

Er zijn enkele verbeteringen die we graag zouden zien voor een volgende versie ('Cloc 3.0'), zoals het gebruik van een USB-connector voor de voeding (tip: als in het ontwerp bijvoorbeeld een ESP32-S3 zou worden gebruikt, zouden de GPIO-USB-pinnen die deze levert ook op de USB-connector kunnen worden aangesloten, zodat je de Cloc kunt programmeren en debuggen zonder de behuizing te openen). Daarnaast kan de ESP32 misschien een echte luidspreker aansturen voor een iets minder onaangenaam wekgeluid dan de huidige buzzer. We beseffen dat de IR-zender bijvoorbeeld een radio kan inschakelen, maar we denken dat steeds minder mensen tegenwoordig zoiets in hun slaapkamer hebben staan. Maar aan de andere kant, als dat dingen zijn die je werkelijk nodig hebt, is er niets dat je tegenhoudt om die te integreren in de huidige versie van de Cloc, gezien de openheid van het ontwerp en de software. ◀

230561-03 (vertaling: Jelle Aarnoudse)

Vragen of opmerkingen?

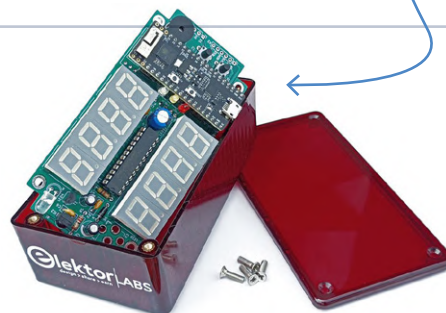
Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de redactie van Elektor via redactie@elektor.com.



Gerelateerde producten

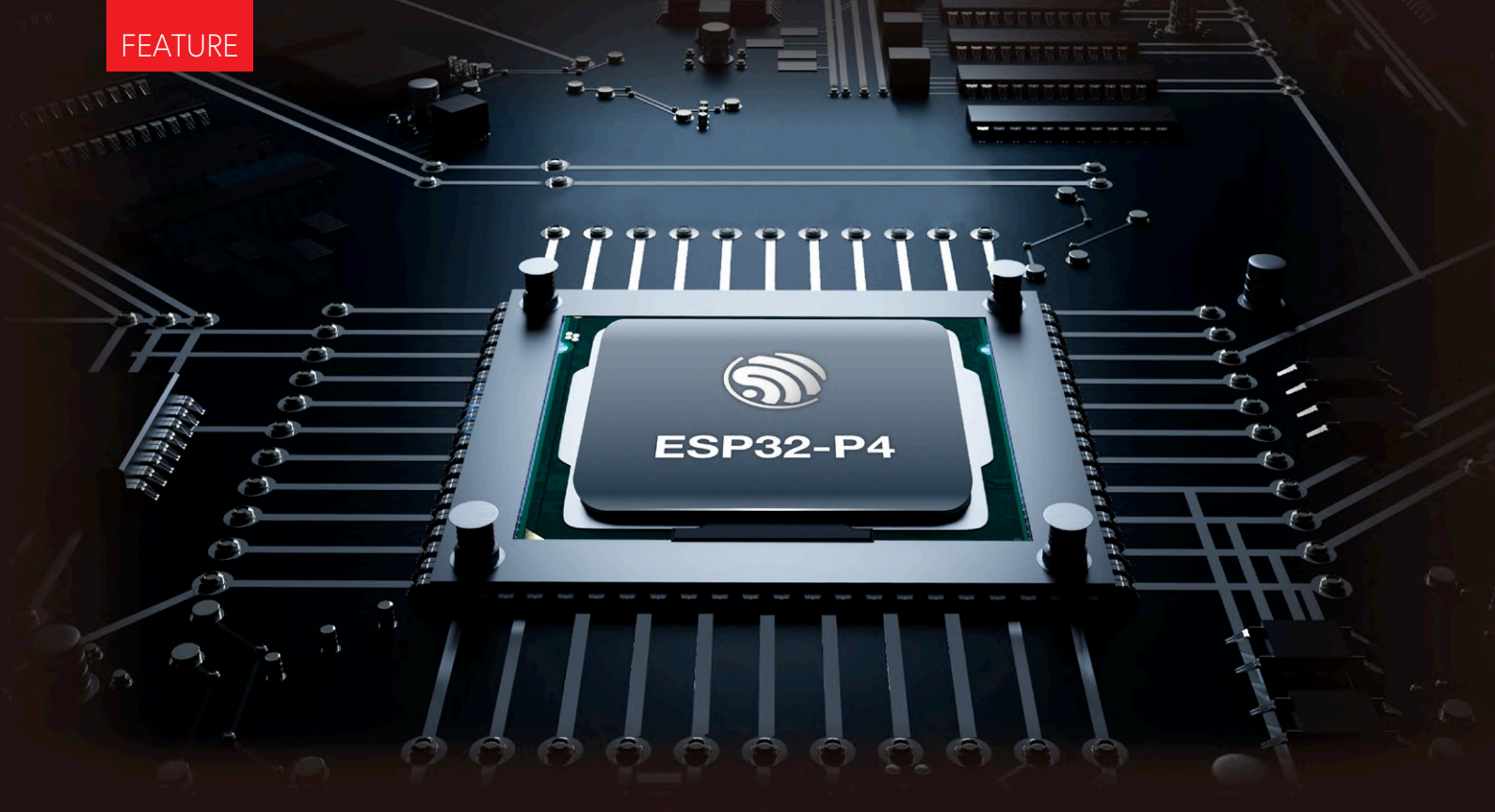
➤ **Elektor Cloc 2.0 Kit**
www.elektor.nl/20438

➤ **ESP32-PICO-Kit V4**
www.elektor.nl/18423



WEBLINKS

- [1] Cloc 2.0 Project op de Elektor Labs-website: <https://elektormagazine.com/labs/cloc-le-reveil-20>
- [2] DHZ wekkerklok – Elektor Cloc 2.0 bouwbeschrijving: <https://youtu.be/9VSLdFz6jyI>
- [3] ESP-Launchpad: <https://espressif.github.io/esp-launchpad>



De **ESP32-P4** ontketend

de volgende microcontroller-era

Anant Raj Gupta, Espressif

Welkom in de volgende microcontroller-era, waarin betaalbare beveiliging, geavanceerde prestaties en ongeëvenaarde connectiviteit samenkomen. De krachtige microcontroller belooft de wereld van embedded systemen een nieuwe vorm te geven en deuren te openen naar een nieuwe wereld van mogelijkheden voor ontwikkelaars, technici en de hele IoT-gemeenschap.

Het hart van deze technologische sprong wordt gevormd door de ESP32-P4, een SoC (system-on-chip) die zorgvuldig is ontworpen voor hoge prestaties en is voorzien van topklasse beveiligingsfuncties. Deze krachtige microcontroller ondersteunt een uitgebreid intern geheugen, heeft indrukwekkende beeld- en spraakverwerkingsmogelijkheden en integreert snelle periferie. Dit alles (**figuur 1**) biedt hem de capaciteit om te voldoen aan de hoge eisen van de volgende generatie van embedded toepassingen.

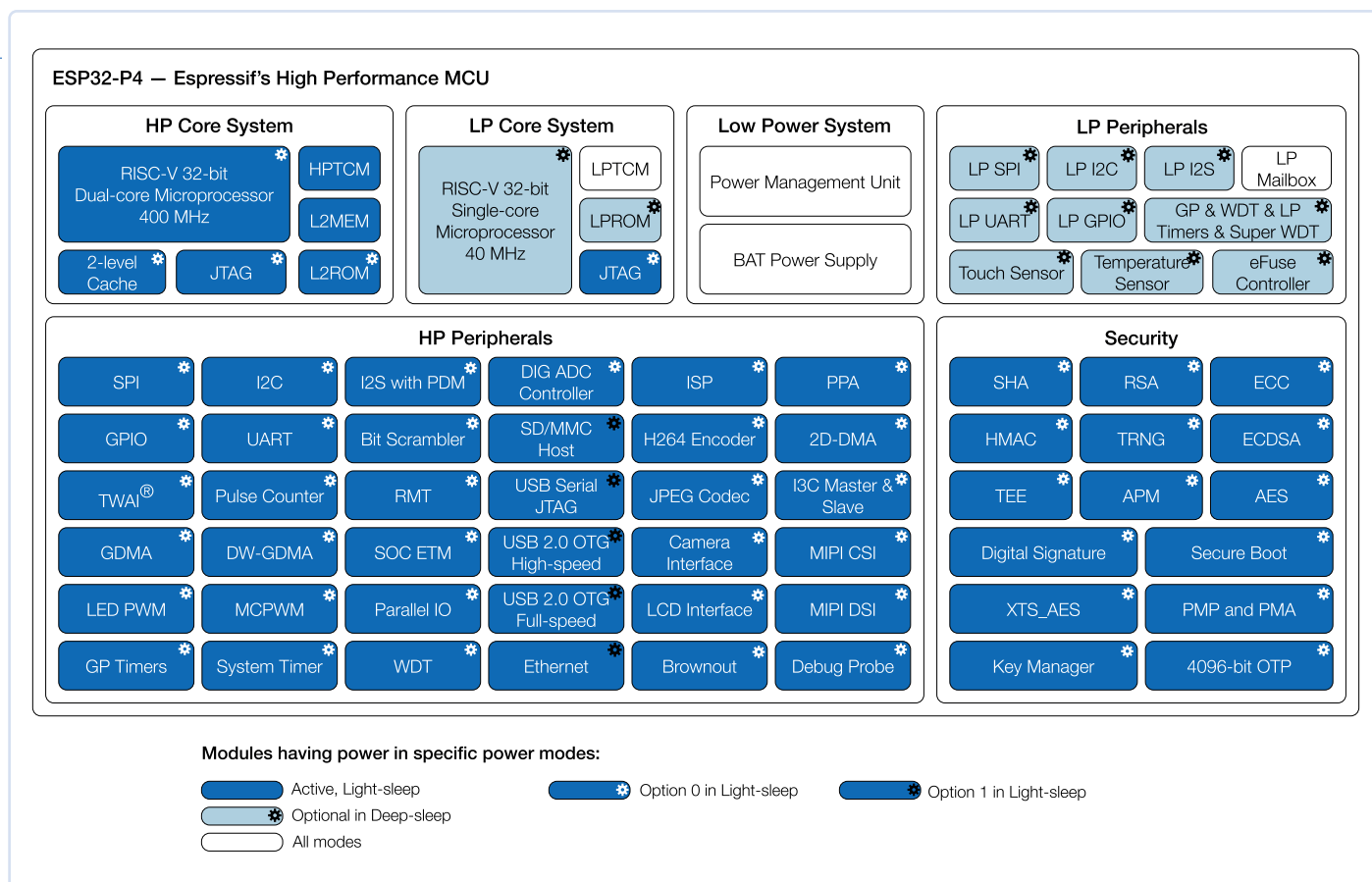
Zo rekent de ESP32-P4

Krachtige CPU- en geheugen-subsysteem

Onder de motorkap van de ESP32-P4 zit een dual-core RISC-V CPU met een kloksnelheid tot 400 MHz. Deze krachtpatser is verder uitgerust met single-precision floating-point units (FPU's) en AI-uitbreidingen, waardoor veel rekenmogelijkheden voorhanden zijn. Dit wordt aangevuld door de integratie van een LP-Core, die tot 40 MHz haalt. Deze groot/klein-architectuur is cruciaal voor het ondersteunen van ultra-low-power toepassingen die af en toe een portie rekenkracht vereisen en tegelijkertijd energie besparen. Waar het uitzonderlijke prestaties betreft, staat de ESP32-P4 zijn mannetje, zoals **figuur 2** illustreert met een vergelijking ten opzichte van andere Espressif-producten zoals de ESP32 en de ESP32-S3.

Geheugenarchitectuur voor ongeëvenaarde efficiëntie

Geheugentoeegang is een kritieke factor bij het leveren van 'naadloze' prestaties. Het geheugensysteem van de ESP32-P4 is meesterlijk efficiënt. Met 768 KB SRAM on-chip, dat kan worden geconfigureerd



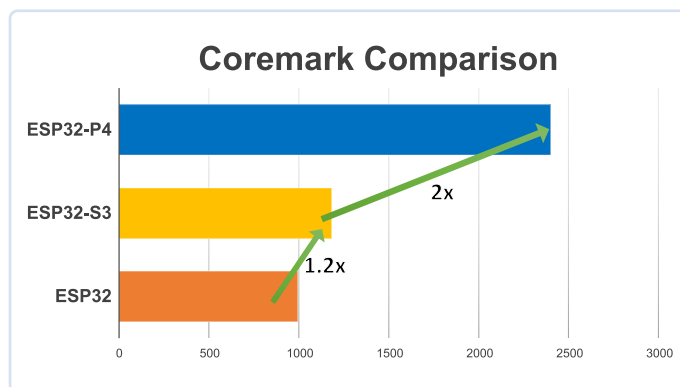
Figuur 1. Blokschema van de ESP32-P4.

als cache wanneer extern PSRAM beschikbaar is, en 8 KB wachttijdloos TCM RAM als snelle databuffer zorgt deze SoC ervoor dat geheugentoegangs-latentie en -capaciteit nooit knelpunten zijn voor je toepassingen.

Bovendien is het externe geheugen van de ESP32-P4 direct toegankelijk, met maar liefst 64 MB aaneengesloten ruimte voor flash- en PSRAM-toegang via de cache. Het interne geheugen van 768 KB kan worden geconfigureerd als L2MEM, gepartitioneerd voor instructies en gegevens, en wordt aangevuld met een speciale 16 KB grote L1-instructiecache en een 64 KB grote L1-datecache. Een extra 8 KB groot direct met de HP-kern gekoppeld geheugen zorgt voor single-cycle toegang tot opgeslagen gegevens. De PSRAM-SPI biedt de mogelijkheid van zestien-regelig DDR-lezen en -schrijven, terwijl ook de ondersteuning voor operatie bij een kloksnelheid van 250 MHz wordt verbeterd, wat een maximale gegevensdoorvoer van 1 GB/s oplevert. Deze meertraps geheugenarchitectuur is ontworpen om geheugentoegang voor toepassingen vrijwel transparant te maken, wat zeer gunstig is voor gebruiksscenario's die hoge prestaties vereisen.

Maatgesneden verwerking voor AI- en DSP-taken

De ESP32-P4 heeft een 32-bits RISC-V dual-core processor die standaard RV32IMAFZC-extensies ondersteunt. Wat hem echter werkelijk anders maakt, is zijn aangepaste, uitgebreide instructieset. Deze is ontworpen om het aantal instructies in *loop bodies* te verminderen, waardoor de prestaties aanzienlijk verbeteren. Daarnaast bevat de SoC aangepaste AI- en DSP-extensies, die de efficiëntie van specifieke AI- en DSP-algoritmen optimaliseren.

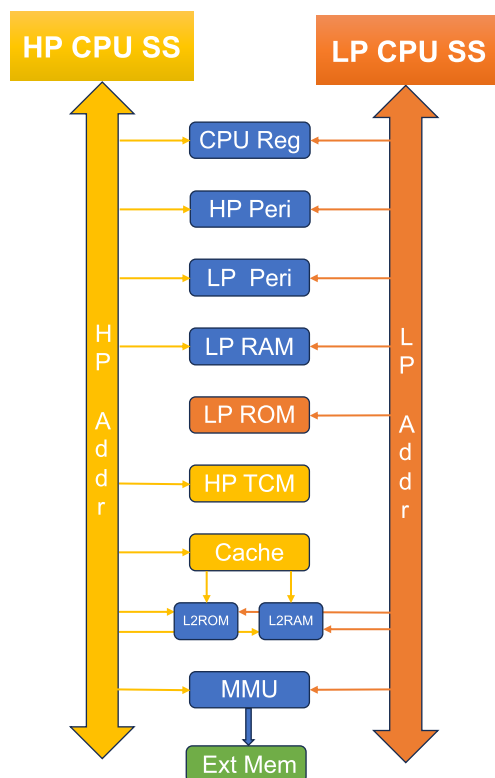


Figuur 2. Coremark-vergelijking.

Deze aangepaste vectorinstructies maken de ESP32-P4 geschikt voor taken zoals neurale netwerken en deep learning, waardoor snellere en efficiënte berekeningen mogelijk worden.

Efficiënte geheugentoegang voor elke core

Zoals te zien in figuur 3, zorgt de architectuur van de geheugentoegang van de ESP32-P4 ervoor dat alle periferie en geheugen toegankelijk zijn vanuit zowel de HP- als de LP-cores. Dankzij deze opzet kan de LP-core de meeste functies afhandelen, terwijl de HP-core alleen wordt ingeschakeld als er 'zware' berekeningen nodig zijn. Bovendien maakt de periferie van de LP-core kritische functies mogelijk, zelfs in de bedrijfsmodi met het laagste energieverbruik. Met deze eigenschappen positioneert de ESP32-P4



Figuur 3. Toegang tot de adresruimte van de ESP32-P4.

zich als de ideale keuze voor toepassingen die geavanceerde reken-capaciteiten vereisen met behoud van een laag energieverbruik gedurende langere perioden.

Gewapend voor de toekomst: beveiliging van topklasse

Secure boot

Beveiliging is geen bijzaak; het is de hoeksteen van de ontwikkeling van de ESP32-P4. Die is zorgvuldig ontworpen om betaalbare beveiligingsoplossingen voor iedereen te bieden. Secure boot, een cruciale beveiligingsfunctie, beveiligt het apparaat tegen het uitvoeren van

niet-geautoriseerde, niet-ondertekende code. Elk stukje software wordt grondig geverifieerd, inclusief de second-stage bootloader en elke binaire toepassing. Zoals te zien in **figuur 4** blijft de first-stage bootloader, als ROM-code, onveranderlijk en hoeft deze niet ondertekend te worden. De ESP32-P4 biedt flexibiliteit door zowel RSA-PSS als ECDSA-gebaseerde secure boot verificatieschema's te ondersteunen, waarbij ECDSA vergelijkbare beveiliging biedt als RSA, maar met kortere sleutellengtes.

Flash-encryptie

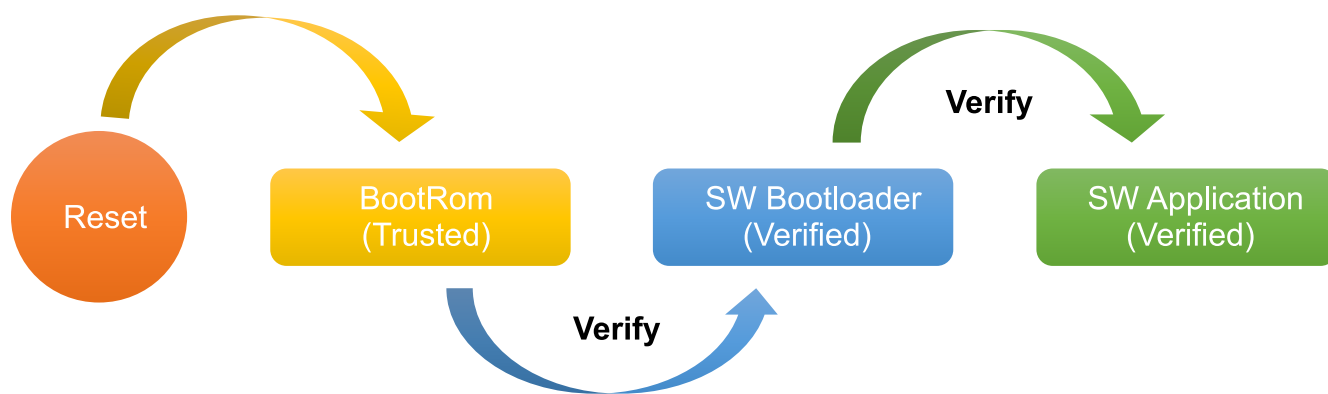
Flash-encryptie is een game-changer bij het beschermen van de inhoud van het off-chip flash-geheugen van de ESP32-P4. Als deze functie is ingeschakeld, wordt de firmware aanvankelijk als platte tekst geflasht en vervolgens versleuteld tijdens de eerste maal opstarten. Dit betekent dat fysieke pogingen om de flash uit te lezen nutteloos zullen zijn en geen bruikbare gegevens opleveren. Wanneer flash-encryptie is geactiveerd, worden elke memory-mapped leestoeegang tot flash transparant en veilig gedecodeerd tijdens runtime. De ESP32-P4 gebruikt de XTS-AES block cipher-modus met een robuuste 256-bit sleutelgrootte voor flash-encryptie, waardoor gegevens optimaal worden beschermd.

Cryptografische hardwareversnelling

De ESP32-P4 is uitgerust met een uitgebreide reeks cryptografische hardwareversnellers, klaar om een reeks standaardalgoritmen aan te pakken die worden gebruikt in netwerk- en beveiligingstoepassingen. Deze versnellers bieden ondersteuning voor AES-128/256, SHA, RSA, ECC en HMAC. Dit vergroot het vermogen van de ESP32-P4 om gegevenstransmissie, -opslag en -authenticatie te beveiligen, waardoor de P4 bij uitstek geschikt is voor veiligheidsgevoelige toepassingen.

Bescherming van de private key

Het beschermen van private keys is van het grootste belang; de ESP32-P4 maakt gebruik van robuuste mechanismen om de veiligheid ervan te garanderen. De SoC genereert private keys on-chip, ontoegankelijk voor software of fysieke aanvallen in tekstvorm. Er worden hardwareversnelde digitale handtekeningen geproduceerd,



Figuur 4. Beveiligd opstartproces.

waardoor toepassingen ondertekeningen kunnen uitvoeren met de versleutelde private key van het apparaat zonder deze in platte tekst bloot te leggen. De ESP32-P4 maakt gebruik van een key manager op basis van functies die niet fysiek gekloond kunnen worden (PUF) en die uniek zijn voor elke chip om de unieke hardware-sleutel (HUK) te genereren. Deze HUK dient als *root of trust* (RoT) voor de chip en wordt automatisch gegenereerd bij elke inschakelcyclus en vervluchtigt wanneer de chip wordt uitgeschakeld. De hardware-gestuurde bescherming van de private key van de ESP32-P4 zorgt voor een ongeëvenaarde beveiliging van gevoelige activiteiten.

Toegangscontrole

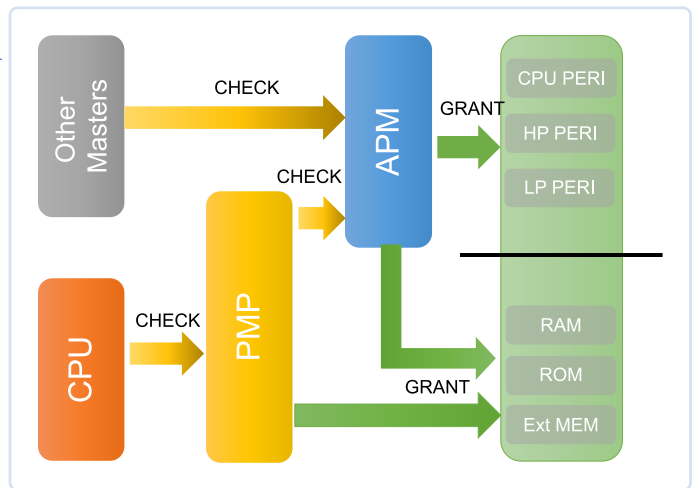
De ESP32-P4 tilt toegangscontrole naar een nieuw niveau met hardwarematige toegangsbeveiliging, waardoor beheer van toegangsrechten en scheiding van rechten mogelijk wordt (**figuur 5**). Dit systeem bestaat uit twee componenten: fysieke geheugenbeveiliging (PMP) en beheer van toegangsrechten (APM). PMP beheert CPU-toegang tot alle adresruimtes, terwijl APM de toegang tot ROM en SRAM voor zijn rekening neemt. PMP moet toestemming geven voor toegang van de CPU tot ROM en HP-SRAM, en pas daarna komt APM in actie voor andere adresruimtes. Als PMP geen toestemming geeft, worden APM-controles niet uitgevoerd. Deze gelaagde aanpak zorgt voor robuuste toegangscontrole, waardoor de ESP32-P4 een veilige keuze is voor een breed scala aan toepassingen.

Uitgebreide periferie en mens/machine-interface (HMI)

Verbeterde visuele en touch-ervaring

De ESP32-P4 verbetert de HMI-ervaring met ondersteuning voor MIPI-CSI (Camera Serial Interface) en MIPI-DSI (Display Serial Interface). Geïntegreerd met beeldsignaalverwerking (ISP) op de MIPI-CSI interface, kan deze SoC de grote invoerformaten aan, zoals RAW8, RAW10 en RAW12, met ondersteuning van resoluties tot full HD (1980×1080) bij maximaal 30 frames per seconde (fps). Deze mogelijkheden maken hem ideaal voor toepassingen zoals IP-camera's (**figuur 6**) en video-deurbellen die een high-res camera-input vereisen.

Aan de beeldschermzijde ondersteunt de MIPI-DSI interface van de ESP32-P4 twee lanes met 1,5 Gbps, wat neerkomt op beeldschermmondersteuning voor 720p HD bij 60 fps of 1080p full HD bij 30 fps. Door de integratie van capacitieve touch-input en spraakherkenningsmogelijkheden is de ESP32-P4 perfect voor elke HMI-gebaseerde toepassing, van interactieve bedieningspanelen tot laadregelaars.



Figuur 5. Toegangscontrole op de ESP32-P4.

Mediaverwerking

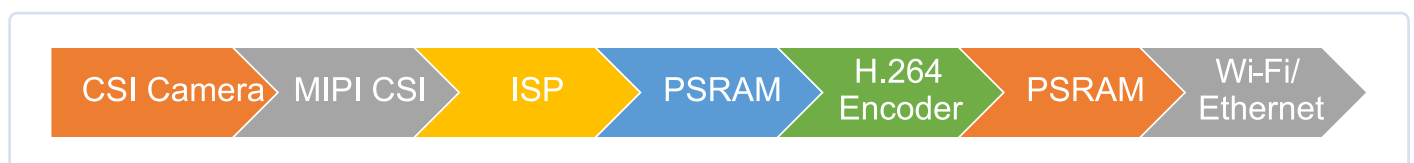
De ESP32-P4 stopt niet bij camera's en displays, maar is ook klaar voor media-codering en -compressie. Met ingebouwde ondersteuning voor H.264 en andere coderingsformaten maakt de ESP32-P4 videostreaming en -verwerking mogelijk. Deze functies kunnen worden gebruikt om budgetvriendelijke IP camera-oplossingen te maken, waarbij de veelzijdige HMI-periferie wordt benut die eerder is genoemd. De SoC heeft ook een geïntegreerde pixel processing-versneller (PPA) in hardware, geschikt voor GUI-ontwikkeling.

Uitgebreide ondersteuning voor GPIO en periferie

Met een opmerkelijke 55 programmeerbare GPIO's zet de ESP32-P4 een nieuwe standaard voor Espressif-producten. Hij biedt ondersteuning voor diverse veelgebruikte periferie, zoals SPI, I²S, I²C, LED PWM, MCPWM, RMT, ADC, DAC, UART en TWAITM. De ESP32-P4 ondersteunt USB OTG 2.0 HS, Ethernet en SDIO Host 3.0 voor snelle connectiviteit.

Naadloze draadloze connectiviteit

Voor toepassingen die draadloze connectiviteit vereisen, koppelt de ESP32-P4 moeiteloos met producten uit de ESP32-C/S/H-serie als draadloze companion chip. Dit kan worden gerealiseerd via SPI/SDIO/UART met behulp van ESP-Hosted of ESP-AT oplossingen. Bij gebruik van de ESP-Hosted oplossing blijft de applicatieontwikkeling consistent met het werken aan een SoC met geïntegreerde draadloze connectiviteit. Daarnaast kan de ESP32-P4 fungeren als host-MCU voor andere connectiviteitsoplossingen, waaronder ACK en AWS IoT ExpressLink. Ontwikkelaars kunnen vertrouwen op het volwassen IoT-ontwikkelingsraamwerk (ESP-IDF) van Espressif voor ondersteuning, dankbaar gebruikmakend van hun bekendheid met een platform dat al bij miljoenen connected apparaten onder de motorkap zit.



Figuur 6. Voorbeeld van de gegevensstroom bij een IP-camera.

Met vertrouwen in de toekomst dankzij ESP32-P4

De ESP32-P4 vertegenwoordigt een nieuwe era in microcontrollers en levert prestaties, veiligheid en connectiviteit, waardoor ontwikkelaars hun meest gedurfde ideeën tot leven kunnen wekken. De ESP32-P4 is geworteld als een veelzijdig, veilig en krachtig fundament.

De toekomst van IoT is rooskleurig en de ESP32-P4 is klaar om bij deze evolutie voorop te lopen, door snellere en efficiëntere verwerking mogelijk te maken. Edge computing brengt intelligentie dicht bij de bron van gegevens. Kunstmatige intelligentie en machine learning zullen IoT-apparaten in staat stellen om intelligente, autonome beslissingen te nemen. De ESP32-P4 is klaar voor deze veranderingen, gewapend met verwerkingskracht, robuuste beveiliging en een dynamische ontwikkelaars-community.

De ESP32-P4 vertegenwoordigt een grote sprong in de wereld van microcontrollers. Het is niet zomaar een chip; het is een uitnodiging tot innovatie. Of je nu een doorgewinterd ontwikkelaar bent of een gepassioneerd nieuwkomer, met de ESP32-P4 kun je je ideeën werkelijkheid laten worden. Waag de sprong, verken het potentieel van de ESP32-P4 en neem deel aan de transformatieve reis die deze microcontroller vertegenwoordigt.

Samen kunnen we de toekomst van het IoT vormgeven en een meer connected, efficiënte en veilige wereld creëren. 

230615-03

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Neem contact op met Espressif via de QR-code of stuur een e-mail naar de redactie van Elektor via redactie@elektor.com.



Over de auteur

Anant Gupta is een Technical Marketing Manager met meer dan 15 jaar ervaring in systeemarchitectuur, IP-architectuur en SoC-ontwerp. Hij is gepassioneerd over het stimuleren van innovatie en het oplossen van problemen van klanten bij Espressif Systems.



Gerelateerde producten

> **Espressif ESP32-reeks**
www.elektor.com/espressif



De meest opwindende ESP-IDF componenten

Het is niet alleen *Node.js* dat *npm* heeft, of *Python* dat *pip* heeft. ESP-IDF heeft ook een component manager, waar zowel door Espressif geleverde componenten als componenten van derden in uw project kunnen worden geïmporteerd.

Componenten bestaan uit een softwarebibliotheek samen met de voorbeelden en documentatie die u helpen om ze in uw project te gebruiken. U vindt hier een verscheidenheid aan softwarebibliotheken voor uw project, variërend van diverse drivers voor randapparatuur, middleware (bijvoorbeeld LVGL porting) tot verschillende board support packages voor verschillende boards.

Deze lijst groeit voortdurend omdat zowel componenten van Espressif als van derden worden

toegevoegd. Als open-source ontwikkelaar kunt u ook uw eigen componenten maken en ze hier registreren zodat andere ontwikkelaars ze gemakkelijk kunnen vinden.

<https://components.espressif.com>

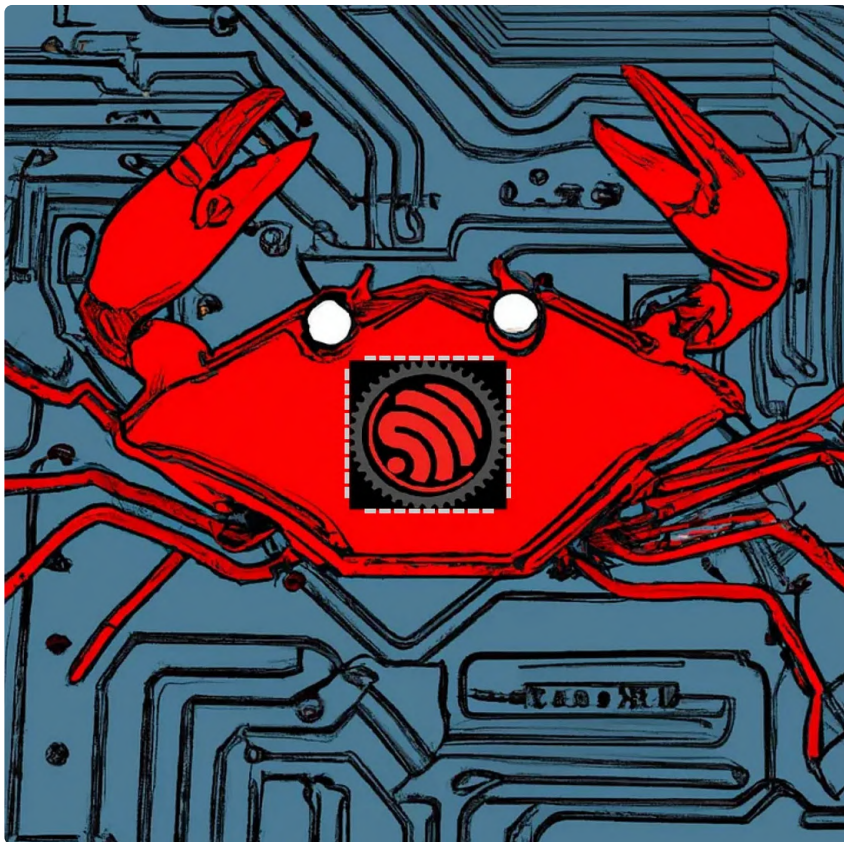


Rust + *Embedded*

een krachtig ontwikkelduo

Juraj Sadel (Espressif)

Met zijn focus op geheugen- en threadveiligheid is Rust een populaire taal voor het maken van betrouwbare en veilige software. Maar is Rust een slimme oplossing voor embedded toepassingen? Dit artikel laat zien dat Rust veel voordelen biedt ten opzichte van traditionele embedded ontwikkeltalen zoals C en C++, waaronder geheugenveiligheid, ondersteuning voor gelijktijdigheid en betere prestaties.



Bron: gegenereerd door DALL-E.

Rust is de populairste nieuwe taal ter wereld geworden, met elk jaar meer geïnteresseerden. De focus ligt op geheugen- en threadveiligheid met als doel betrouwbare en veilige software te maken. De ondersteuning van Rust voor gelijktijdigheid (concurrency) en parallel programmeren is vooral relevant voor embedded ontwikkeling, waar efficiënt gebruik van resources cruciaal is.

Het begin van Rust

Het oorspronkelijke idee voor een programmeertaal Rust is bij toeval ontstaan. In 2006 keerde Graydon Hoare terug naar zijn appartement in Vancouver, waar de lift weer eens niet werkte door een softwarebug. Hoare woonde op de 21ste verdieping en terwijl hij de trap opliep, begon hij te denken: "Wij computeraars kunnen niet eens liftsoftware maken die niet crasht!". Dit bracht hem ertoe een nieuwe programmeertaal te ontwerpen. Hij hoopte dat het mogelijk zou zijn om kleine, snelle code te schrijven zonder geheugenbugs [1]. Als je geïnteresseerd bent in een meer gedetailleerde en technische geschiedenis van Rust, bezoek dan [2] en [3].

Bijna 18 jaar later is Rust de populairste nieuwe taal ter wereld geworden, met elk jaar meer geïnteresseerden. In het eerste kwartaal van 2020 waren er ongeveer 600.000 Rust-ontwikkelaars en in het eerste kwartaal van 2022 steeg het aantal naar 2,2 miljoen [4]. Techgiganten zoals Mozilla, Dropbox, Cloudflare, Discord, Facebook (Meta), Microsoft en anderen gebruiken Rust in hun codebase. In de afgelopen zes jaar bleef Rust de meest 'geliefde' programmeertaal [5].

Embedded ontwikkeling

Embedded ontwikkeling is niet zo populair als web- of desktopontwikkeling, en dit zijn een paar redenen waarom:



- › Hardwarebeperkingen: de embedded systemen hebben waarschijnlijk beperkte hardware-resources, zoals prestaties en geheugen. Dit kan het moeilijker maken om software voor deze systemen te ontwikkelen.
- › Beperkte en/of nichemarkt: de embedded markt is beperkter dan die voor web- en desktoptoepassingen, en dit kan het financieel minder interessant maken voor ontwikkelaars die gespecialiseerd zijn in embedded programmeren.
- › Gespecialiseerde low-level kennis: gedegen kennis van de concrete hardware en low-level programmeertalen is noodzakelijk bij embedded ontwikkeling.
- › Langere ontwikkelcycli: het ontwikkelen van software voor embedded systemen kan langer duren dan het ontwikkelen van software voor web- of desktopapplicaties, omdat de code moet worden getest en geoptimaliseerd voor de specifieke hardware-vereisten.
- › Low-level programmeertalen: deze talen, zoals assembler of C, bieden weinig abstractie voor de ontwikkelaar en geven directe toegang tot hardware-resources en geheugen, wat zal leiden tot geheugenbugs.

Dit zijn maar een paar redenen waarom en hoe embedded ontwikkeling uniek is en niet zo bekend en lucratief voor jonge programmeurs als webontwikkeling. Als je gewend bent aan de meest gebruikte en moderne programmeertalen zoals Python, JavaScript of C# waar je niet elke proces-sorcyclus en elke kilobyte in het geheugen hoeft te tellen, dan is het een ingrijpende verandering om met embedded ontwikkeling te beginnen. Het kan erg ontmoedigend zijn om in de embedded wereld te stappen, niet alleen voor beginners, maar ook voor ervaren web-/desktop-/mobiele ontwikkelaars. Daarom is het erg interessant en noodzakelijk om een moderne programmeertaal te hebben voor embedded ontwikkeling.

Waarom Rust?

Rust is een moderne en relatief jonge taal met een focus op geheugen- en thread-veiligheid, met als doel betrouwbare en veilige software te maken. Daarnaast is de ondersteuning van Rust voor gelijktijdig-

De ondersteuning van Rust voor concurrency en parallelisme is vooral relevant voor embedded ontwikkeling, waar efficiënt gebruik van resources cruciaal is.

heid en parallel programmeren bijzonder relevant voor embedded ontwikkeling, waar efficiënt gebruik van resources cruciaal is. De groeiende populariteit en het ecosysteem van Rust maken het een aantrekkelijke optie voor ontwikkelaars, vooral voor degenen die op zoek zijn naar een moderne taal die zowel efficiënt als veilig is. Dit zijn de belangrijkste redenen waarom Rust een steeds populairdere keuze wordt, niet alleen voor embedded ontwikkeling, maar vooral voor projecten waarbij veiligheid, beveiliging en betrouwbaarheid hoog in het vaandel staan.

Voordelen (vergeleken met C/C++)

Laten we eens kijken naar de belangrijkste voordelen van Rust.

- › Geheugenveiligheid: Rust biedt sterke garanties voor geheugenveiligheid via het eigendoms- en leensysteem, wat erg nuttig is bij het voorkomen van veel voorkomende geheugengerelateerde bugs zoals null-pointer verwijzingen of buffer overflows. Met andere woorden, Rust garandeert geheugenveiligheid tijdens het compileren door middel van het ownership en borrowing systeem. Dit is vooral belangrijk bij embedded ontwikkeling, waar geheugen-/resourcebeperkingen het moeilijker kunnen maken om zulke bugs te detecteren en op te lossen.
- › Concurrency (gelijktijdigheid): Rust biedt uitstekende ondersteuning voor zero-cost abstracties en veilige concurrency en multithreading, met een ingebouwde `async/await` syntaxis en een krachtig type-systeem dat veel voorkomende concurrency-bugs zoals data races voorkomt. Dit kan het makkelijker maken om veilige en efficiënte parallelle code te schrijven,

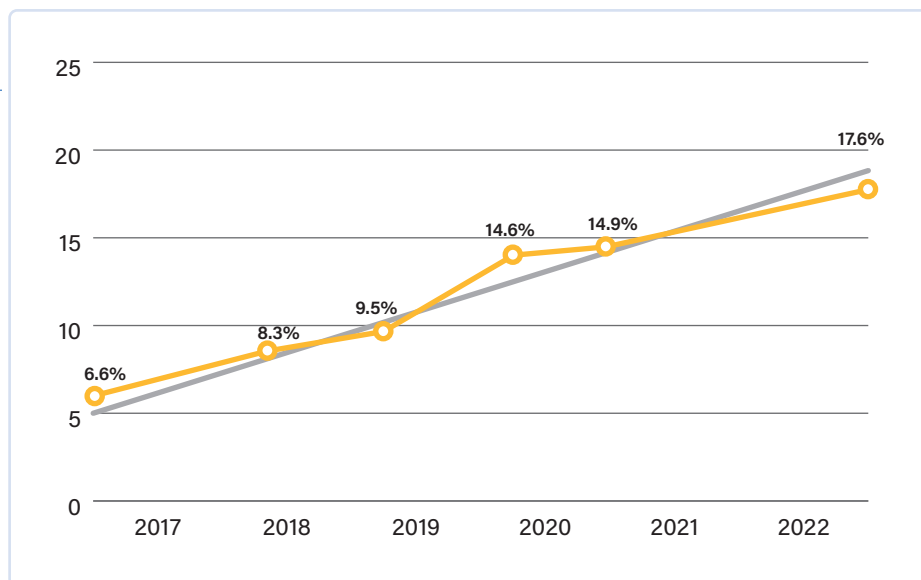
niet alleen in embedded systemen.

- › Prestaties: Rust is ontworpen voor hoge prestaties en kan op dat gebied wedijveren met C en C++, terwijl het nog steeds de veiligheid van het geheugen garandeert en ondersteuning voor gelijktijdigheid biedt.
- › Leesbaarheid: de syntaxis van Rust is ontworpen om leesbaarder en minder foutgevoelig te zijn dan C en C++, met functies zoals patroonherkenning, type-inferentie en functionele programmeerconstructies. Dit kan het makkelijker maken om code te schrijven en te onderhouden, vooral voor grotere en complexere projecten.
- › Groeiend ecosysteem: Rust heeft een groeiend ecosysteem van bibliotheken (crates), tools en resources voor (niet alleen) embedded ontwikkeling, wat het makkelijker kan maken om met Rust aan de slag te gaan en de nodige ondersteuning en resources voor een bepaald project te vinden.
- › Packet manager en build system: de Rust-distributie bevat een officiële tool genaamd Cargo, die wordt gebruikt om het build-, test- en publicatieproces te automatiseren, samen met het maken van een nieuw project en het beheren van de afhankelijkheden.

Nadelen (vergeleken met C/C++)

Aan de andere kant is Rust geen perfecte taal en heeft het ook enkele nadelen ten opzichte van andere programmeertalen (niet alleen ten opzichte van C en C++).

- › Leercurve: Rust heeft een steilere leercurve dan veel andere programmeertalen, waaronder C. De unieke functies, zoals het al genoemde ownership en borrowing, kunnen enige tijd kosten om te begrijpen en eraan te wennen en vormen daarom een grotere uitdaging om met Rust aan de slag te gaan.
- › Compilatietijd: het geavanceerde type-systeem en de borrow checker van Rust kunnen resulteren in langere compilatietijden in vergelijking met andere talen, vooral bij grote projecten.
- › Tooling: hoewel het ecosysteem van Rust snel groeit, heeft het misschien nog niet hetzelfde niveau van tool-ondersteuning als meer gevestigde



Figuur 1. Het groeiende percentage ontwikkelaars dat met Rust wil werken (bron: Yalantis [4]).

programmeertalen. C en C++ bestaan bijvoorbeeld al tientallen jaren en hebben een enorme codebase. Dit kan het moeilijker maken om de juiste tools voor een bepaald project te vinden en te gebruiken.

- Gebrek aan low-level controle: de veiligheidsfuncties van Rust kunnen soms de low-level controle beperken tot C en C++. Dit kan het uitdagender maken om bepaalde low-level optimalisaties uit te voeren of direct met hardware te communiceren, maar het is mogelijk.
- Omvang van de community: Rust is nog steeds een relatief nieuwe programmeertaal vergeleken met meer gevestigde talen als C en C++, wat betekent dat het een kleinere community van ontwikkelaars en bijdragers heeft, en minder resources, bibliotheken en tools.

Over het algemeen biedt Rust veel voordelen ten opzichte van traditionele embedded ontwikkeltalen zoals C en C++, waaronder geheugenveiligheid, ondersteuning voor concurrency, prestaties, leesbaarheid van de code en een groeiend ecosysteem. Hierdoor wordt Rust een steeds populairdere keuze voor embedded ontwikkeling, vooral voor projecten waar veiligheid, beveiliging en betrouwbaarheid de hoogste prioriteit hebben. De nadelen van Rust in vergelijking met C en C++ hebben te maken met het feit dat Rust een relatief nieuwe taal is, en met de unieke mogelijkheden. Veel ontwikkelaars vinden echter dat de voordelen van Rust het een aantrekkelijke keuze maken voor bepaalde projecten.

Hoe kan Rust draaien?

Er zijn verschillende manieren om op Rust gebaseerde firmware te draaien, afhankelijk van de omgeving en de vereisten van de toepassing. De op Rust gebaseerde firmware kan meestal in twee modi gebruikt worden: *hosted-environment* of *bare-metal*. Laten we daar eens naar kijken.

Wat is hosted-environment?

In Rust komt de *hosted-environment* dicht in de buurt van een normale PC-omgeving [6], wat betekent dat je de beschikking krijgt over een besturingssysteem. Met het besturingssysteem is het mogelijk om de Rust-standaardbibliotheek (*std*) te bouwen. [7] De *std* verwijst naar de standaardbibliotheek, die kan worden gezien als een verzameling modules en typen die bij elke installatie van Rust worden meegeleverd. De *std* biedt een set van meerdere functionaliteiten voor het bouwen van Rust-programma's, waaronder datastructuren, netwerken, mutexen en andere synchronisatieprimitiveën, invoer/uitvoer en meer.

Met de *hosted-environment* benadering kun je de functionaliteit van het op C gebaseerde development framework ESP-IDF [8] gebruiken, omdat het een *newlib*-omgeving [9] biedt die 'krachtig' genoeg is om de Rust-standaardbibliotheek bovenop te bouwen. Met andere woorden, met de *hosted-environment* (soms gewoon *std* genoemd) benadering, gebruiken we de ESP-IDF als een besturingssysteem en bouwen we de Rust-applicatie daar bovenop. Op deze manier kunnen we gebruik maken van alle standaardbibliotheekfuncties die hierboven zijn genoemd en kunnen we ook al C-functionaliteit implementeren vanuit

de ESP-IDF API.

In **listing 1** kun je zien hoe een blinky-voorbeeld [10] dat draait bovenop ESP-IDF (FreeRTOS) eruit kan zien. Meer voorbeelden zijn te vinden in *esp-idf-hal* [11].

Waarom je hosted environment zou gebruiken

- Uitgebreide functionaliteit: als je embedded systeem veel functionaliteit vereist, zoals ondersteuning voor netwerkprotocollen, file I/O of complexe datastructuren, dan zul je waarschijnlijk de hosted-environment aanpak willen gebruiken omdat *std*-bibliotheken een breed scala aan functionaliteiten bieden die gebruikt kunnen worden om relatief snel en efficiënt complexe applicaties te bouwen.
- Overdraagbaarheid: de *std-crate* biedt een gestandaardiseerde set API's die gebruikt kunnen worden op verschillende platformen en architecturen, waardoor het gemakkelijker wordt om code te schrijven die overdraagbaar en herbruikbaar is.
- Snelle ontwikkeling: de *std-crate* biedt een rijke set aan functionaliteit die gebruikt kan worden om snel en efficiënt applicaties te bouwen, zonder zorgen over low-level details.

Wat is bare-metal?

Bare-metal betekent dat we geen besturingssysteem hebben om mee te werken. Wanneer een Rust-programma gecompileerd is met het `no_std` attribuut, betekent dit dat het programma geen toegang heeft tot bepaalde mogelijkheden. Dit betekent niet noodzakelijkerwijs dat je geen netwerken of complexe datastructuren kunt gebruiken met `no_std`. Je kunt zonder `std` alles doen wat je met `std` kunt doen, maar het is complexer en uitdagender. `no_std` programma's vertrouwen op een set kernfuncties van de taal die beschikbaar zijn in alle Rust-omgevingen, bijvoorbeeld datatypes, controlestructuren of low-level geheugenbeheer. Deze aanpak is handig voor embedded programmeren waar geheugengebruik vaak beperkt is en low-level controle over hardware nodig is.

In **listing 2** zie je een mogelijk bare-metal blinky-voorbeeld [12] (dus zonder besturingssysteem). Meer voorbeelden zijn te vinden in *esp-hal* [13].

Waarom je bare-metal zou gebruiken

- Kleine geheugen-footprint: als je embedded systeem beperkte resources heeft en een kleine geheugen-footprint moet hebben, zul je waarschijnlijk *bare-metal* willen gebruiken, omdat *std*-functies het uiteindelijke binaire bestand veel groter maken en meer compilatietijd nodig hebben.
- Directe controle over de hardware: als je embedded systeem meer directe controle over de hardware vereist, zoals low-level apparaatdrivers of toegang tot gespecialiseerde hardwarefuncties, dan zul je waarschijnlijk *bare-metal* willen gebruiken omdat *std* abstracties toevoegt die het moeilijker kunnen maken om direct met de hardware te communiceren.
- Real-time beperkingen of tijdkritische toepassingen: als je embedded systeem real-time prestaties of reacties met geringe latentie vereist; *std* kan onvoorspelbare vertragingen en overhead

introduceren die de real-time prestaties kunnen beïnvloeden.

- Speciale eisen: *bare-metal* maakt meer maatwerk en fijnmazige controle over het gedrag van een applicatie mogelijk, wat nuttig kan zijn in gespecialiseerde of niet-standaard omgevingen.

Moet je overstappen van C naar Rust?

Als je een nieuw project of een taak begint waarbij geheugenveiligheid of concurrency vereist is, kan het het overwegen waard zijn om van C naar Rust over te stappen. Als je project echter al goed is opgezet en functioneel is in C, wegen de voordelen van een overstap naar Rust misschien niet op tegen de kosten van het herschrijven en opnieuw testen van je hele codebase. In dat geval kun je overwegen om de huidige C-codebase te behouden en te beginnen met het schrijven en toevoegen van nieuwe functies, modules en functionaliteit in Rust – het is relatief eenvoudig om C-functies aan te roepen vanuit Rust-code. Het is ook mogelijk om ESP-IDF componenten in Rust te schrijven

[14]. De uiteindelijke beslissing om van C naar Rust over te stappen moet gebaseerd zijn op een zorgvuldige evaluatie van je specifieke behoeften en de afwegingen die daarbij komen kijken. ◀

230569-03 (vertaling: Jelle Aarnoudse)

Over de auteur

Juraj Sadel is een embedded software ontwikkelaar met een passie voor Rust en toegewijd aan het verbeteren van embedded systemen. Hij is ook een actief lid van het Rust-team, draagt expertise bij aan de community en is enthousiast over de geavanceerde technologie van Espressif.

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen hebt naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via juraj.sadel@espressif.com of naar de redactie van Elektor via redactie@elektor.com.



Listing 1. Blinky-voorbeeld met hosted environment en std.

```
// Import peripherals we will use in the example
use esp_idf_hal::delay::FreeRtos;
use esp_idf_hal::gpio::*;
use esp_idf_hal::peripherals::Peripherals;

// Start of our main function i.e entry point of our example
fn main() -> anyhow::Result<()> {
    // Apply some required ESP-IDF patches
    esp_idf_sys::link_patches();

    // Initialize all required peripherals
    let peripherals = Peripherals::take().unwrap();

    // Create led object as GPIO4 output pin
    let mut led = PinDriver::output(peripherals.pins.gpio4)?;

    // Infinite loop where we are constantly turning ON and OFF the LED every 500ms
    loop {
        led.set_high()?;
        // we are sleeping here to make sure the watchdog isn't triggered
        FreeRtos::delay_ms(1000);

        led.set_low()?;
        FreeRtos::delay_ms(1000);
    }
}
```




Listing 2. Bare-metal blinky-voorbeeld.

```
#![no_std]
#![no_main]

// Import peripherals we will use in the example
use esp32c3_hal::{
    clock::ClockControl,
    gpio::IO,
    peripherals::Peripherals,
    prelude::*,
    timer::TimerGroup,
    Delay,
    Rtc,
};
use esp_backtrace as _;

// Set a starting point for program execution
// Because this is `no_std` program, we do not have a main function
#[entry]
fn main() -> ! {
    // Initialize all required peripherals
    let peripherals = Peripherals::take();
    let mut system = peripherals.SYSTEM.split();
    let clocks = ClockControl::boot_defaults(system.clock_control).freeze();

    // Disable the watchdog timers. For the ESP32-C3, this includes the Super WDT,
    // the RTC WDT, and the TIMG WDTs.
    let mut rtc = Rtc::new(peripherals.RTC_CNTL);
    let timer_group0 = TimerGroup::new(
        peripherals.TIMG0,
        &clocks,
        &mut system.peripheral_clock_control,
    );
    let mut wdt0 = timer_group0.wdt;
    let timer_group1 = TimerGroup::new(
        peripherals.TIMG1,
        &clocks,
        &mut system.peripheral_clock_control,
    );
    let mut wdt1 = timer_group1.wdt;

    rtc.swd.disable();
    rtc.rwdt.disable();
    wdt0.disable();
    wdt1.disable();

    // Set GPIO4 as an output, and set its state high initially.
    let io = IO::new(peripherals.GPIO, peripherals.IO_MUX);
    // Create led object as GPIO4 output pin
    let mut led = io.pins.gpio5.into_push_pull_output();

    // Turn on LED
    led.set_high().unwrap();

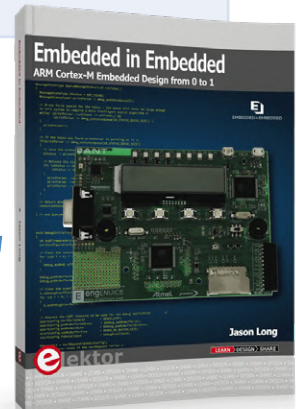
    // Initialize the Delay peripheral, and use it to toggle the LED state in a
    // loop.
    let mut delay = Delay::new(&clocks);
```

```
// Infinite loop where we are constantly turning ON and OFF the LED every 500ms
loop {
    led.toggle().unwrap();
    delay.delay_ms(500u32);
}
```



Gerelateerde producten

- **J. Long, Embedded in Embedded (Elektor 2018)**
Book: www.elektor.nl/18876
E-book: www.elektor.nl/18877
- **A. He and L. He, Embedded Operating System (Elektor 2020)**
Book: www.elektor.nl/19228
E-book: www.elektor.nl/19214



WEBLINKS

- [1] MIT Technology Review, "How Rust went from a side project to the world's most-loved programming language," 2023: <https://technologyreview.com/2023/02/14/1067869/rust-worlds-fastest-growing-programming-language>
- [2] Rust Blog, "Announcing Rust 1.0," 2015: <https://blog.rust-lang.org/2015/05/15/Rust-1.0.html>
- [3] Rust Blog, "4 years of Rust," 2019: <https://blog.rust-lang.org/2019/05/15/4-Years-Of-Rust.html>
- [4] Yalantis, "The state of the Rust market in 2023" : <https://yalantis.com/blog/rust-market-overview>
- [5] Stack Overflow, "Stack Overflow Developer Survey," 2021: <https://insights.stackoverflow.com/survey/2021#most-loved-dreaded-and-wanted>
- [6] The Embedded Rust Book: <https://docs.rust-embedded.org/book/intro/no-std.html#hosted-environments>
- [7] The Rust Standard Library (std): <https://doc.rust-lang.org/std>
- [8] ESP-IDF: <https://github.com/espressif/esp-idf>
- [9] Newlib-bibliotheek: <https://sourceware.org/newlib>
- [10] Blinky-voorbeeld bovenop ESP-IDF: <https://github.com/esp-rs/esp-idf-hal/blob/master/examples/blinky.rs>
- [11] ESP-IDF-HAL: <https://github.com/esp-rs/esp-idf-hal/tree/master/examples>
- [12] Bare-metal blinky-voorbeeld: <https://github.com/esp-rs/esp-hal/blob/main/esp32c3-hal/examples/blinky.rs>
- [13] ESP-HAL: <https://github.com/esp-rs/esp-hal/tree/main>
- [14] ESP-IDF componenten in Rust: <https://github.com/espressif/rust-esp32-example>



ESP-Matter

de Matter-SDK van Espressif



Espressif biedt een gebruiksvriendelijke API bovenop de open-source "connectedhomeip" SDK om u te helpen eenvoudig Matter-compatibele projecten te bouwen. Deze SDK ondersteunt alle SoC's van Espressif, zoals ESP32, ESP32-C3, ESP32-C2, ESP32-S3, ESP32-H2 en ESP32-C6. Hij ondersteunt zowel het Matter-protocol via WiFi als via Thread. Naast standaard Matter-voorbeelden die verschillende soorten devices ondersteunen, biedt hij ook implementaties van Matter ZigBee bridge, Matter BLE Mesh bridge en Matter ESP-Now bridge.

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





Wie zijn de onvervaarde embedded Rust-ontwikkelaars?

zo cultiveert Espressif embedded Rust voor de ESP32

Inleiding en vragen door Stuart Cording (Elektor)

Het is net iets meer dan tien jaar oud, maar de programmeertaal Rust heeft al de zeventiende plaats bereikt in de TIOBE Programming Community-index [1]. De taal is in die tijd 194 plaatsen opgeschoven en staat nu naast R, Ruby, Delphi, Scratch en MATLAB.

En niet alleen dat: Linus Torvalds, de vader van Linux, heeft voorstellen geaccepteerd om in Rust geschreven code te gebruiken in de kernel – iets wat C++ al jaren probeert en wat niet lukt. Het heeft ook een groeiende schare fans onder ontwikkelaars van embedded systemen, iets waar Espressif op in wil springen.

C is al tientallen jaren de standaardtaal voor embedded systemen; assembler is iets voor degenen die met de hand optimaliseren of real-time kernels ontwikkelen. C werd in de jaren '70 ontwikkeld door Dennis Ritchie bij Bell Labs en is nauw verbonden met het ontstaan van het besturingssysteem Unix. Unix was geschreven in assembler voor de PDP-7 [2] maar moest herschreven worden om het over te zetten naar de PDP-11 [3] (PDP's waren kleinere, universele computers die in de jaren '60 werden verkocht als alternatief voor mainframes). C maakte het mogelijk om code te ontwikkelen, zoals Unix, die processoronafhankelijk was en overdraagbaar naar veel verschillende architecturen.

Embedded systemen werden aanvankelijk geprogrammeerd in assembler, maar naarmate de code complexer werd en ontwikkelaars op zoek gingen naar meer hergebruik, werden ook zij aangetrokken door C. Extra functies die hun leven eenvoudiger maakten, waren onder andere ondersteuning voor low-level

bitmanipulatie, moeiteloze definitie van variabelen, het gemak waarmee algoritmen konden worden gecodeerd en de eenvoud van geheugenbewerkingen. Dit laatste punt zorgt er echter voor dat de meeste applicaties vastlopen. Hoewel pointers programmeurs in staat stellen om (bijna) elke geheugenlocatie op elk moment te benaderen, iets wat erg krachtig en efficiënt is in een embedded systeem, kunnen ze ook erg gevaarlijk zijn. Pointers kunnen naar geheugen wijzen dat lang niet meer is toegewezen, of gemanipuleerd worden om een functie aan te roepen die resulteert in een inbreuk op de beveiliging. Microsoft schrijft zelfs ongeveer 70% van de problemen met C/C++ software toe aan bugs die het geheugen corrumperen [4].

Rust pakt dit probleem aan door geheugenveiligheid te forceren. Bovendien worden, in tegenstelling tot C/C++ code, veel programmeerproblemen gedetecteerd tijdens het compileren in plaats van tijdens runtime. En dankzij de overeenkomsten in syntaxis en prestaties zullen C en C++ ontwikkelaars zich snel thuis voelen, zowel op hun PC als op embedded systemen. Hoe serieus nemen fabrikanten van microcontrollers Rust daarom als taal voor embedded systemen? Om daar meer over te weten te komen, sprak Elektor met Scott Mabin, een jonge ontwikkelaar die, vrij van de ketenen van het verleden, besloot zich meer en meer in de wereld van Rust te verdiepen. Als fervent gebruiker van de ESP32 heeft hij belangrijk bijgedragen aan embedded Rust, waardoor hij werd aangenomen bij Espressif.

Stuart Cording: Embedded systemen worden traditioneel geprogrammeerd in C. Maar je besloot om dat te negeren en meteen in Rust te duiken. Waarom?

Scott Mabin: Ik had Rust gebruikt in mijn afstudeerproject aan de universiteit. Ik bouwde een smartwatch – niet slim naar hedendaagse maatstaven, maar het deed meer dan de tijd tonen. Een deel van dat project onderzocht of Rust C kon vervangen voor firmware-ontwikkeling en waar de afwegingen lagen.

En toen werd ik er verliefd op. Ik zat daar en vroeg me af waarom niet iedereen Rust gebruikte. Het bood zoveel dat, als het eenmaal gecompileerd was, je er gerust op kon zijn dat een hele categorie van geheugen- en race-problemen geëlimineerd was. Natuurlijk begreep ik dat sommige projecten legacy-code hebben en dat je niet zonder goede reden een nieuwe taal wilt leren en implementeren. Toch leek het me vreemd dat niet meer mensen Rust gingen gebruiken.

Stuart Cording: Rust wordt al door de community ondersteund op de STM32 en sommige Scandinavische apparaten. Waarom heb je toen voor Espressif gekozen?

Scott Mabin: Thuis had ik een heleboel ESP32's liggen en ik vroeg me af of ik die dingen ook in Rust kon programmeren. Nou, daarvoor moest ik behoorlijk diep in de materie duiken. Het grootste probleem was dat de core, Xtensa, die chips zoals de ESP8266 aandrijft, alleen ondersteund werd door de GCC-compiler maar niet door LLVM. Ik keek naar *mrustc* [5], een tool die Rust omzet naar C en vervolgens kan compileren, maar dit cross-compilatie proces kon me niet overtuigen. Gelukkig bracht Espressif een fork uit voor de LLVM toolchain om Xtensa te ondersteunen en hoewel het in eerste instantie moeilijk te gebruiken was, betekende het wel dat je native Rust kon compileren voor ESP32-chips. Daarmee schreef ik mijn eerste *blinky LED*-code in Rust; nou ja, ik zeg 'in Rust' maar het schreef eigenlijk alleen maar naar registers en was niet erg 'Rustig' qua structuur. Het was echter een vroeg succes, dat ik documenteerde in mijn eerste blogpost [6] waarin ik mijn ontdekkingsreis met Rust op de ESP32 vastlegde.

Stuart Cording: Dus je sleutelde aan Rust op de ESP32. Hoe ontwikkelde zich je verhouding met Espressif?

Scott Mabin: Ik begon de *esp-rs* groep op GitHub in mijn vrije tijd om Rust-projecten voor ESP32 te verzamelen. Samen met andere leden van de community hebben we ondersteuning voor periferie geïmplementeerd, zoals SPI en andere. Helaas was WiFi ons nog steeds niet gelukt. Toen, na een jaar van gemeenschapswerk en bloggen over onze vooruitgang, vroeg Espressif of ze me konden inhuren om Rust-ondersteuning voor hun ecosysteem te garanderen. Het team van Espressif was in onze richting altijd erg behulpzaam geweest, door te reageren op ondersteuningsverzoeken op hun forums of Reddit. Maar toen ze contact met me opnamen, werd het me duidelijk dat ze al een tijdje geïnteresseerd waren in Rust.

Stuart Cording: Zijn er specifieke toepassingsgebieden die de interesse in Rust aanwakkeren, of is het een beetje glazen-bol-kijken van de kant van Espressif? En hoe is de mate van betrokkenheid?

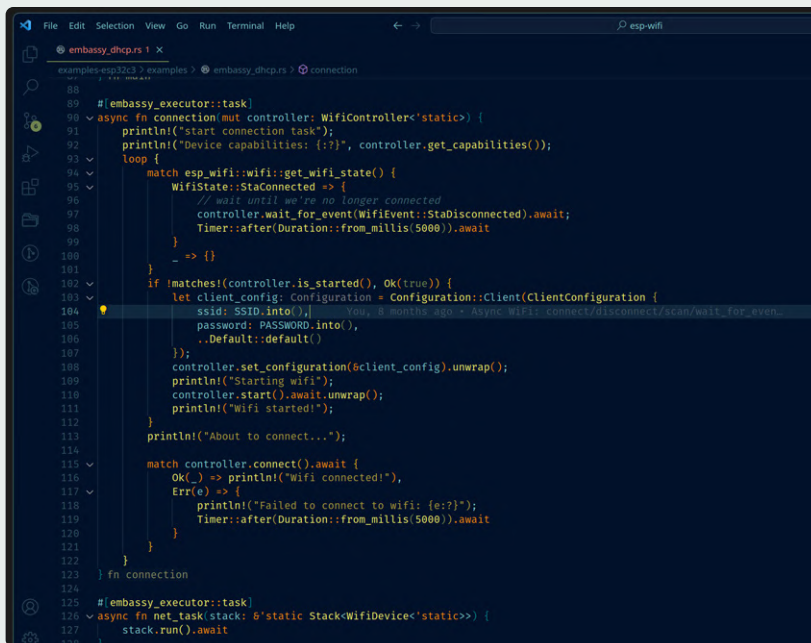
Scott Mabin: Er is zeker interesse van klanten. Klanten gebruiken Rust voornamelijk in IoT-projecten of toepassingen die met het internet verbonden zijn. De meeste projecten zijn vrij eenvoudig, zoals dataloggers. Daarnaast zijn er complexere toepassingen zoals een full-body VR-tracker en een open-hardware nachthemelsensor. Er is ook een zekere mate van toekomstbestendigheid door Espressif, ter voorbereiding op wanneer Rust een belangrijkere rol gaat spelen in het leven van ontwikkelaars van embedded systemen. Tien of elf van ons dragen bij aan het Rust Enablement Team van Espressif, waarvan ongeveer de helft voltijds. Bovendien heeft Rust embedded een geweldige community die ook bijdraagt.

Stuart Cording: Je hebt wat embedded C-ontwikkeling gedaan en gebruikt Rust nu dagelijks op microcontrollers. Waar moeten traditionele C-programmeurs rekening mee houden als ze overstappen op Rust?

Scott Mabin: Als je uit een pure C-omgeving komt en geen ervaring hebt met C++, is het behoorlijk moeilijk. De meest radicale verandering is dat Rust nogal zwaar op types berust. Dan is er nog de resource-kant. Kleine microcontrollers met een paar KB SRAM en flash zullen het moeilijk krijgen vanwege geheugenbeperkingen. In zulke gevallen zul je uiteindelijk C-achtig Rust moeten schrijven om het geheugengebruik te beperken, waardoor je een deel van de voordelen van deze nieuwe taal verliest. Je krijgt nog steeds de voordelen op het gebied van geheugenseveiligheid van Rust, dus het is nog steeds beter dan C. In een appels met appels vergelijking presteren C- en Rust-applicaties ongeveer gelijk. In sommige gevallen is Rust beter, maar de binary is meestal groter.

Stuart Cording: Terwijl leveranciers van microcontrollers veel ondersteuning bieden met drivers voor periferie, is versiebeheer van al die code voor de ontwikkelaar als integrator vaak een puinhoop. Zullen Rust-crates een reden zijn om van programmeertaal te veranderen?

Scott Mabin: Ja, ik denk dat crates een groot voordeel bieden – een grote bonus die verder gaat dan de taal zelf. Crates bieden een packaging-ecosysteem of, op zijn minst, een methode om afhankelijkheden binnen te halen zonder ze handmatig te definiëren in je build systeem. De *embedded_hal* [7] crate biedt een op eigenschappen gebaseerde interface voor veelgebruikte periferie, zoals I²C. Als je dan een I²C-temperatuursensordriver neemt, werkt het gewoon bovenop. Als je dan je microcontroller verandert of een nieuwe temperatuursensor gebruikt, werkt het nog steeds, wat erg prettig is. Maar dat is nog maar het begin. Crates zoals *heapless* [8] maken het mogelijk om statisch gealloceerde datastructuren te maken zoals wachtrijen, vectoren, hash tables, hash sets en strings.



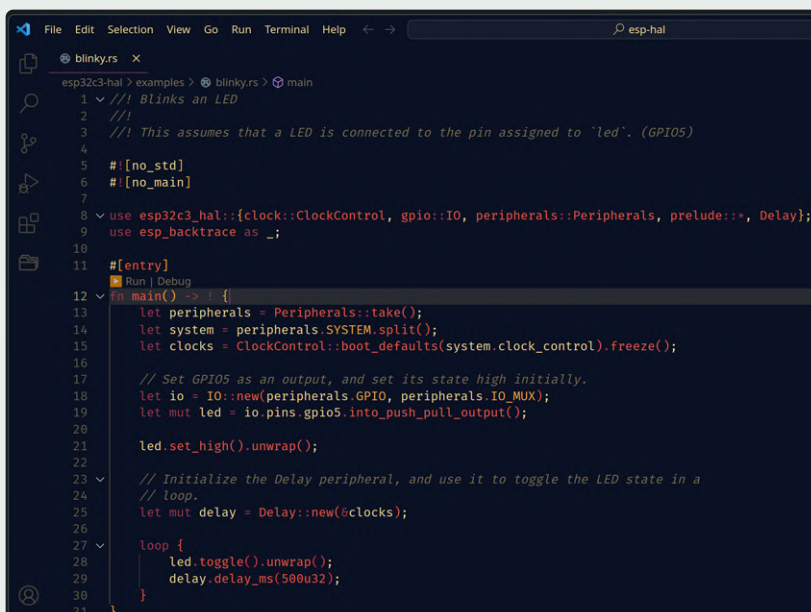
Figuur 1: Een voorbeeld van een WiFi-toepassing voor de ESP32, geschreven in Rust.

In elk ander C-project waar ik aan heb gewerkt, zit iemand een wachtrij te schrijven en spendeert vele uren aan het perfectioneren van een standaard datastructuur. Die tijd kan nu besteed worden aan applicatieontwikkeling.

Stuart Cording: Net als C biedt Rust een standaardbibliotheek. Embedded ontwikkelaars verafschuwen de standaardbibliotheek, dus kunnen ze die van Rust ook afschaffen?

Scott Mabin: Historisch gezien denk ik dat we het moeilijk hadden om de gebruikssituaties te beschrijven waarin een professionele embedded programmeur een standaardbibliotheek zou moeten gebruiken [9]. Ze kijken één keer en zeggen dat het teveel overhead is, en daar ben ik het helemaal mee eens. Maar de Rust-standaardbibliotheek is echt nuttig, op een aantal manieren. Ten eerste, als je Rust kent maar

Figuur 2. Scott Mabin raadt VS Code aan voor Rust-ontwikkeling. Hier is de klassieke blinky LED-code te zien.



niet embedded, voelt het vertrouwd aan. Je krijgt alle threads, netwerken en andere dingen die je gewend bent. Ten tweede, als je ESP32's programmeert met het ESP-IDF [10] (Espressif IoT Development Framework), kun je Rust standaardbibliotheek-projecten maken en aan de slag gaan. Als je het vergelijkt met de Python-standaardbibliotheek, is het eigenlijk vrij slank en, hoewel er overhead is, is het niet zo erg als het klinkt. Het maakt een paar meer aannames dan de corebibliotheek (die je minimaal nodig hebt), zoals de beschikbaarheid van threads en netwerken.

Veel mensen ontwikkelen embedded Rust zonder de standaardbibliotheek (`no_std` [11]) en als je weet wat je doet, ga dan zeker voor de `no_std` aanpak als dat zinvol is voor het project. Je zult echter de stukken die je wilt hebben moeten kiezen en aan elkaar plakken. Dus, bijvoorbeeld, de WiFi-crate bevat een voorbeeld van communicatie met een HTTP server of het opzetten van een access point en het draaien van een server (**figuur 1**). Je kunt alles doen in `no_std` wat je kunt doen met `std` – het is alleen meer werk. Met `std` is het een 'inclusief batterijen' soort van omgeving.

Stuart Cording: Is een deel van het probleem dat het gewoon tijd is om te accepteren dat we microcontrollers met grote geheugens nodig hebben?

Scott Mabin: Ik denk dat, hoe groot microcontrollers ook worden, er altijd iemand zal zijn die een miljoen stuks wil maken en de kleinste en goedkoopste mogelijke oplossing nodig heeft. Er zal altijd een use case zijn voor assembler en C – en misschien zelfs Rust – in een specifieke context voor die kleine, compacte toepassing. Maar ik denk wel dat er een algemene trend is naar grotere microcontrollers en dat de ervaring van de ontwikkelaar meer prioriteit krijgt dan de kosten van de hardware. Maar dat is slechts mijn observatie in mijn korte tijd in de industrie, hoewel het er wel op lijkt.

Stuart Cording: Het is nog vroeg voor Rust. Hoe gaat Espressif trainingen verzorgen om ontwikkelaars op snelheid te krijgen?

Scott Mabin: We zijn een samenwerking aangegaan met Ferrous Systems [12], een Rust-consultancy, om een cursus aan te bieden die we open source hebben gemaakt. Er is trainingmateriaal voor de standaardbibliotheek-benadering en we zijn ook bijna klaar met het toevoegen van de niet-standaard bibliotheekbenadering.

Stuart Cording: En hoe zit het met ontwikkelomgevingen en debugging tools?

Scott Mabin: Ik bewonder de esthetiek van omgevings zoals vim, Neovim of Emacs, maar omdat ik geen ervaring heb met deze tools, zitten ze alleen maar in de weg. Dus gebruik ik VS Code (**figuur 2**) – dat doet het werk voor mij. Met de nieuwere ESP's (C3 en hoger)

hebben we een module die de USB Serial JTAG heet. Het is een uiterst klein, ingebouwd stukje periferie dat een seriële poort en een JTAG-apparaat biedt (**figuur 3**). Om het te gebruiken sluit je het gewoon aan op de USB-poort van je PC en het wordt automatisch gedetecteerd door OpenOCD of *probe-rs*, een Rust debugging-toolkit met een vergelijkbaar doel als OpenOCD.



Figuur 3. De ESP32-C3 integreert een USB-gebaseerde debugmodule met een seriële interface, zodat er geen externe probe-hardware nodig is.

Stuart Cording: Het ondersteunen van een nieuwe taal op een processor is een enorme taak. Waar staan jullie op dit moment en wat is er nog te doen?

Scott Mabin: We hebben bijna alles voor de standaardbibliotheek, maar we hebben niet alles voor ESP-IDF. ESP-IDF is groot, bestaat al jaren en is geschreven in C, dus gebruiken we *bindgen* om bindingen met Rust te maken. In principe moeten we kijken naar de interfaces die nog niet geïmplementeerd zijn en daar een Rust API voor maken. We moeten per geval beslissen wat prioriteit krijgt. Voor *no_std* hebben we het over programmeren in puur Rust, dus we moeten code schrijven voor alle bestaande hardware. We hebben WiFi-, Bluetooth- en Thread-ondersteuning op alle chips. Dus op een schaal van 0 tot 100, zou ik zeggen dat we ongeveer 65 tot 70 procent van de weg zijn. Helaas is het een beetje een bewegend doel, omdat we voortdurend nieuwe apparaten moeten ondersteunen als die uitkomen. We hebben besproken om sommige elementen van ESP-IDF in Rust te maken waar dat zinvol is. Rust is bijvoorbeeld erg geschikt voor het doorgeven van bytestreams. Dus als er een nieuwe component nodig is, dan maken we die misschien in Rust en bieden we een C API. We zullen moeten zien of de voordelen opwegen tegen de inspanningen.

Stuart Cording: Tot slot, waar kun je als embedded ontwikkelaar terecht voor meer informatie over Rust?

Scott Mabin: Ik hang meestal rond in het ESP Rust matrix-kanaal [13] en verschillende andere Rust embedded-kanalen. Verder Google, soms Reddit; het lezen van goede Rust-code kan me helpen dingen te begrijpen en te leren die ik nog niet wist. ◀

230616-03

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de redactie van Elektor via redactie@elektor.com.



Over de auteur

Scott Mabin is een embedded software engineer. Nadat hij op de universiteit de mogelijkheden van embedded Rust had verkend, besloot hij er in zijn vrije tijd mee te experimenteren op de ESP32. Deze inspanningen leidden ertoe dat Espressif hem vroeg om zich bij hun team te voegen om zich daar te richten op het verbeteren van de ondersteuning voor Rust op hun draadloze MCU's en AIoT-oplossingen.

WEBLINKS

- [1] TIOBE Programming Community index: <https://tinyurl.com/tioberust>
- [2] PDP-7 [Wikipedia]: <https://tinyurl.com/pdp7wikipedia>
- [3] PDP-11 [Wikipedia]: <https://tinyurl.com/pdp11wikipedia>
- [4] MSRC Team, "A proactive approach to more secure code," Microsoft, July 2019: <https://tinyurl.com/msrcsecurecode>
- [5] mrustc Project: <https://tinyurl.com/mrustcproject>
- [6] S. Mabin, "Rust on the ESP32," September 2019: <https://tinyurl.com/rustesp32>
- [7] embedded_hal Crate documentatie: <https://tinyurl.com/embeddedhalcrate>
- [8] heapless Crate documentatie: <https://tinyurl.com/heaplesscrate>
- [9] Het Rust op de ESP boek, "Using the Standard Library (std)": <https://tinyurl.com/rustbookstd>
- [10] ESP-IDF Programming Guide, "Get started": <https://tinyurl.com/espidfgetstarted>
- [11] Het Rust op de ESP boek, "Using the Core Library (no_std)": <https://tinyurl.com/rustbooknostd>
- [12] Training, "Embedded Rust on Espressif": <https://tinyurl.com/rustonespressif>
- [13] Rust gebruikersgroep op Matrix: <https://tinyurl.com/rustmatrixug>

De SoC-serie van Espressif

Dual-Core
400 MHz

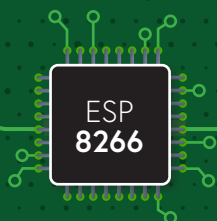
Dual-Core
240 MHz

Single-Core
240 MHz

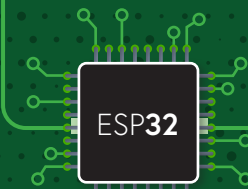
Single-Core
160 MHz

Single-Core
120 MHz

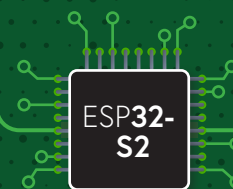
Single-Core
96 MHz



32-bit
Tensilica
MCU
Wi-Fi 4



32-bit Xtensa MCU
Wi-Fi 4
Bluetooth 4.2



32-bit Xtensa MCU
Wi-Fi 4



ESPRESSIF



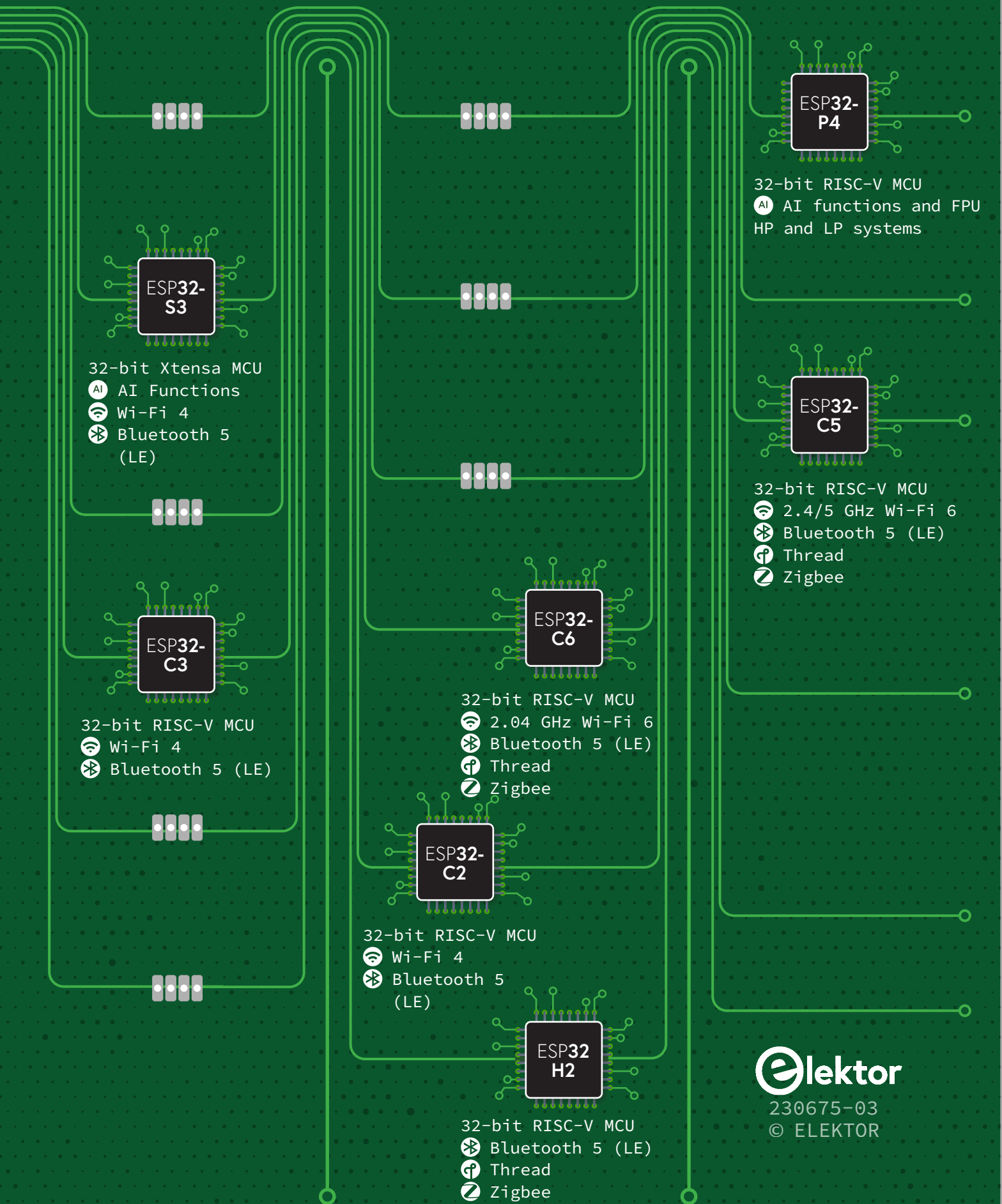
2014

2016

2020



[elektormagazine.nl/posters](https://www.elektormagazine.nl/posters)





Een PLC met oplossingen van Espressif

met de mogelijkheden en functionaliteit van het ISOBUS-protocol

Franz Höpfinger, HR Agrartechnik

HR Agrartechnik GmbH had een goedkoop, maar krachtig PLC-systeem nodig met ISOBUS (ISO 11783) mogelijkheden. De combinatie van een ESP32 van Espressif met de ESP-IDF, samen met het Eclipse 4diac™ framework, leverde het logiBUS®-project op.

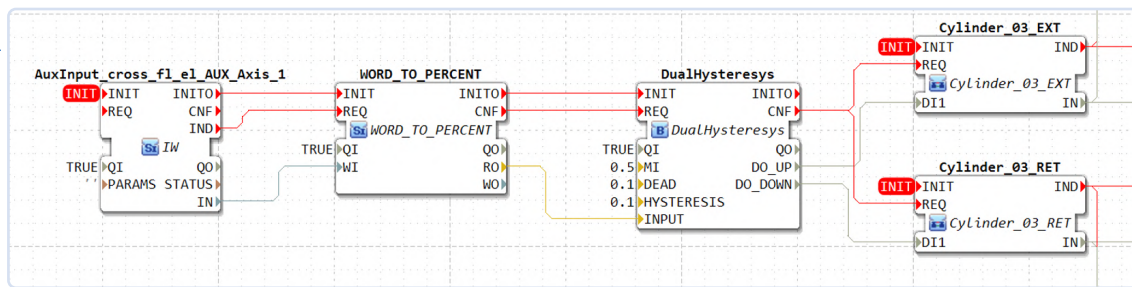
De uitdaging

Moderne landbouwmachines vereisen flexibele besturingssystemen. De aantallen zijn meestal klein en de machines gaan lang mee, dus het achteraf installeren



▲ *Figuur 1. ISOBUS- (ISO 11783-)gebruikersinterface op een John Deere-monitor. Deze bestuurt een slurry tanker.*

van besturingssystemen op bestaande machines in het veld is heel gewoon. In **figuur 1** zie je de gebruikersinterface van de besturingsmonitor na de retrofit. Het is niet makkelijk om C of C++ en het beheer van bibliotheken en systemen te leren. Debuggen met JTAG is niet flexibel en vereist extra tools en toegang tot de microcontroller. Aan de andere kant zijn er veel programmeurs beschikbaar om PLC's te programmeren en is grafisch programmeren eenvoudiger te leren. ISOBUS, ook bekend als ISO 11783, is een communicatieprotocol dat is ontwikkeld voor landbouwmachines om de uitwisseling van gegevens tussen verschillende soorten landbouwmachines en -werktuigen te vergemakkelijken. 'ISOBUS' bestaat uit ISO (International Standardization Organisation) en bus, wat een communicatiesysteem tussen meerdere apparaten is.



Figuur 2. Deze broncode-listing laat zien hoe programmeren in IEC 61499/Eclipse 4diac™ werkt. Je schrijft geen regels code, maar gebruikt in plaats daarvan functieblokken.

In die context wilden we een PLC-systeem ontwerpen dat bestond uit een runtime-systeem op de controller zelf en een IDE op de PC, dat een flexibel, gebruiksvriendelijk, snel automatiseringssysteem zou bieden met modelgebaseerde, low-code grafische programmering en krachtige bibliotheken. Een strikte scheiding tussen een herbruikbare, flexibele runtime-omgeving en de applicatie stond bovenaan in ons verlanglijstje. Het tweede punt was de mogelijkheid om variabelen en functieblokken werkelijk online te observeren. Andere belangrijke punten waren de kwaliteit van de ondersteuning door de leverancier van de software-stack en het gebruik van resources. We wilden een duur en volledig embedded LINUX-systeem vermijden, voornamelijk vanwege de opstarttijd. Agrarische besturingen worden waarschijnlijk tientallen keren per dag in- en uitgeschakeld.

We bekeken verschillende oplossingen, zowel closed als open-source, en kozen uiteindelijk voor Eclipse 4diac™ [1]. “Waarom niet het OpenPLC-project?” zullen sommige Elektor-lezers zich wellicht afvragen. Het antwoord ligt voor de hand: aan de twee belangrijkste eisen wordt dan niet voldaan. OpenPLC biedt geen strikte scheiding tussen applicatie en runtime, noch een online watch-functionaliteit.

De oplossing

Voor de logiBUS®-oplossing [2] combineerden we de softwarekant (zie hieronder) met onze basistoepassing:

- ESP-IDF development framework (momenteel versie 5.1.1)
- open-source IEC 61499 runtime: Eclipse 4diac™ FORTE PLC
- CCI ISOBUS-stuurprogramma, een ISO 11783-protocolstack

Dit resulteert in een PLC-runtime-systeem dat onafhankelijk is van de beoogde toepassing. Het programmeren gebeurt vanuit de Eclipse 4diac™ IDE. Het flashen van de applicatie kan via een TCP/IP-verbinding, dus via WiFi of Ethernet. De oplossingen bieden echte PLC-functies zoals online watch en online change.

Aan de hardwarekant is een ESP32 met PSRAM een goede keuze omdat het model veel RAM gebruikt, dus voor grotere modellen is een gebrek aan PSRAM een probleem. Voor de ISOBUS-verbinding gebruiken we de ESP32 TWAI-periferie als CAN-buscontroller, samen met een externe Infineon TLE9251VSJ CAN-transceiver. Sommige modellen van de hardware, vooral die voor het onderwijs, zijn open source [6].

Met het logiBUS®-project dat nog steeds groeit en wordt uitgebreid, hebben we al een dozijn echte klantprojecten gerealiseerd; in **figuur 2** zie je een voorbeeld van een broncode-listing voor een van hen.

Het systeem lijkt ook geschikt voor andere industrieën, zoals gebouwautomatisering [5]. Daarom hebben we een open-source versie van logiBUS® gemaakt zonder de ISOBUS-stack, maar met dezelfde functionaliteit en dezelfde kracht.

Een paar demotoepassingen, zoals Bale Counter [4] en Slurry Tanker [5], zijn ook open source gemaakt. We hopen een community op te bouwen rond het onderwerp van open source PLC-systemen [6].

230610-03

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de redactie van Elektor via redactie@elektor.com.

WEBLINKS

- [1] Eclipse 4diac™: <https://eclipse.dev/4diac>
- [2] logiBUS® homepage: <https://www.logibus.tech>
- [3] logiBUS® open-source derivaten voor bijvoorbeeld gebouwautomatisering (niet ISOBUS): <https://gitlab.com/meisterschulen-am-ostbahnhof-munchen>
- [4] Bale Counter: https://github.com/Meisterschulen-am-Ostbahnhof-Munchen/4diac_EasyExampleCounter
- [5] Slurry Tanker: <https://github.com/Meisterschulen-am-Ostbahnhof-Munchen/4diac-SlurryTanker-sample>
- [6] Resources en Wiki: <https://github.com/Meisterschulen-am-Ostbahnhof-Munchen>



Import

Edit

Export

I_Made_a_VGA_Card_



Het ESP32-S3 VGA-board

volg Bitluni tijdens het ontwerpproces

Samengesteld door Elektor en Espressif

YouTuber Bitluni ontwierp een ESP32-S3 VGA-board en bereikte daarmee een opmerkelijke resolutie en kleurechtheid. Hij maakte gebruik van de LCD-periferie van de Espressif ESP32-S3 die in de plaats is gekomen van de I2S van de vorige versie. Laten we zijn stappen eens reconstrueren. Zo leer je veel over het proces om een nieuw product tot leven te brengen.

Jaren geleden maakte de bekende Duitse YouTuber Bitluni een VGA-bibliotheek en een paar VGA-boards voor de ESP32 (**figuur 1**). Veel van zijn toegewijde volgers hielden van het ontwerp, maar hij geeft toe dat hij geen fabrikant is en dat hij er maar een beperkt aantal verkocht. Sindsdien is er veel veranderd! De ESP32 API is een paar keer bijgewerkt en de ESP32-S3 is op de markt gekomen (**figuur 2**). Omdat veel van zijn volgers hem vroegen om een nieuwe versie van de bibliotheek en het board te maken, omdat deze niet compatibel waren met de S3, besloot Bitluni het te proberen. Zo werd de ESP32-S3 VGA geboren (**figuur 3**).

Toen Bitluni besloot om het nieuwe board (**figuur 4**) te ontwerpen, wilde hij dat het toegankelijker zou zijn. De eerdere boards moesten gesoldeerd worden en hadden een extra dev board nodig. Gelukkig had hij genoeg ervaring opgedaan om er nu een te ontwerpen met alles erop en eraan (**figuur 5**). Plug&play zonder veel solderen! Eén ding dat hij wilde was dat het breadboard-vriendelijk zou zijn, zodat als je er headers op soldeerde, je het in een breadboard zou kunnen prikken en twee extra rijen zou hebben om bij de pinnen te komen (**figuur 6**). De VGA-connector was vrij groot en de antenne had wat vrije ruimte nodig, dus deze opzet leek redelijk.

Daarna volgde het ontwerp (**figuur 7**), de printen werden besteld en de resultaten waren grandioos (**figuur 8**). Bitluni ging ervan uit dat de VGA-connectoren die hij had compatibel zouden zijn met de footprint van de printjes, maar al snel ontdekte hij dat de pinning niet overeenkwam (**figuur 9**). Gelukkig had hij een paar smallere connectoren bij de hand. De pinnen daarvan pasten net en de print stak een beetje uit (**figuur 10**). Omdat er daar geen sporen liepen, besloot hij er gewoon een paar millimeter af te frezen en het probleem was opgelost (**figuur 11...14**). Toen alles gemonteerd was, werd het tijd voor wat code (**figuur 15**).

Tot dan had Bitluni nog niet in het technische handboek gekeken, maar toen hij dat wel deed, realiseerde hij zich waarom zijn oude bibliotheek niet werkte op de S3. Espressif had de parallelle I2S-modus geschrapt en nieuwe periferie voor camera's en LCD's gemaakt. Dus volgde er het nodige studeren (**figuur 16**), denken en testen volgde (**figuur 17**). En daarna nog meer studeren, nog meer denken (**figuur 18**), nog meer testen, en uiteindelijk nog wat desolderen en herconfigureren, wat werd gevolgd door nog meer herconfigureren, nog meer solderen en nog meer testen (**figuur 19**). Hij had het er maar druk mee! Gelukkig werkte het na al die moeite. Niet alleen 640×480, maar ook 800×600 (**figuur 20**).

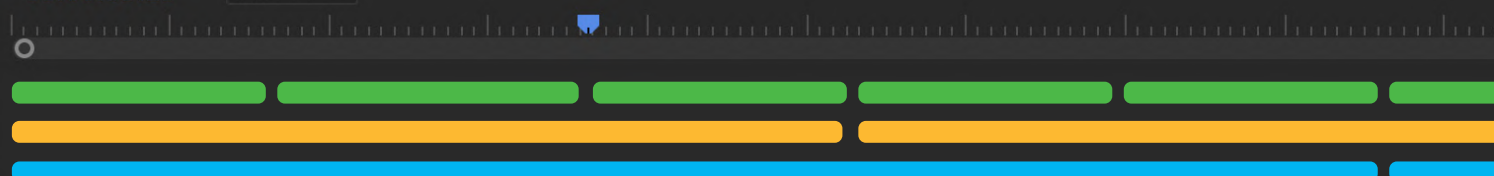
Na wat testen op verschillende apparaten ontdekte Bitluni wat synchronisatieproblemen. Drie van zijn andere schermen leken de synchronisatie aan het begin van elk frame aan te passen. Dat bracht de ontwikkeling tot stilstand, legt hij uit: toen hij inzoomde met zijn 'scoop, ontdekte hij een kleine vertraging aan het begin van elk frame (**figuur 21**). Dus ging hij weer studeren en nadenken en activeerde hij de 'Hacker Mode' (**figuur 22**).

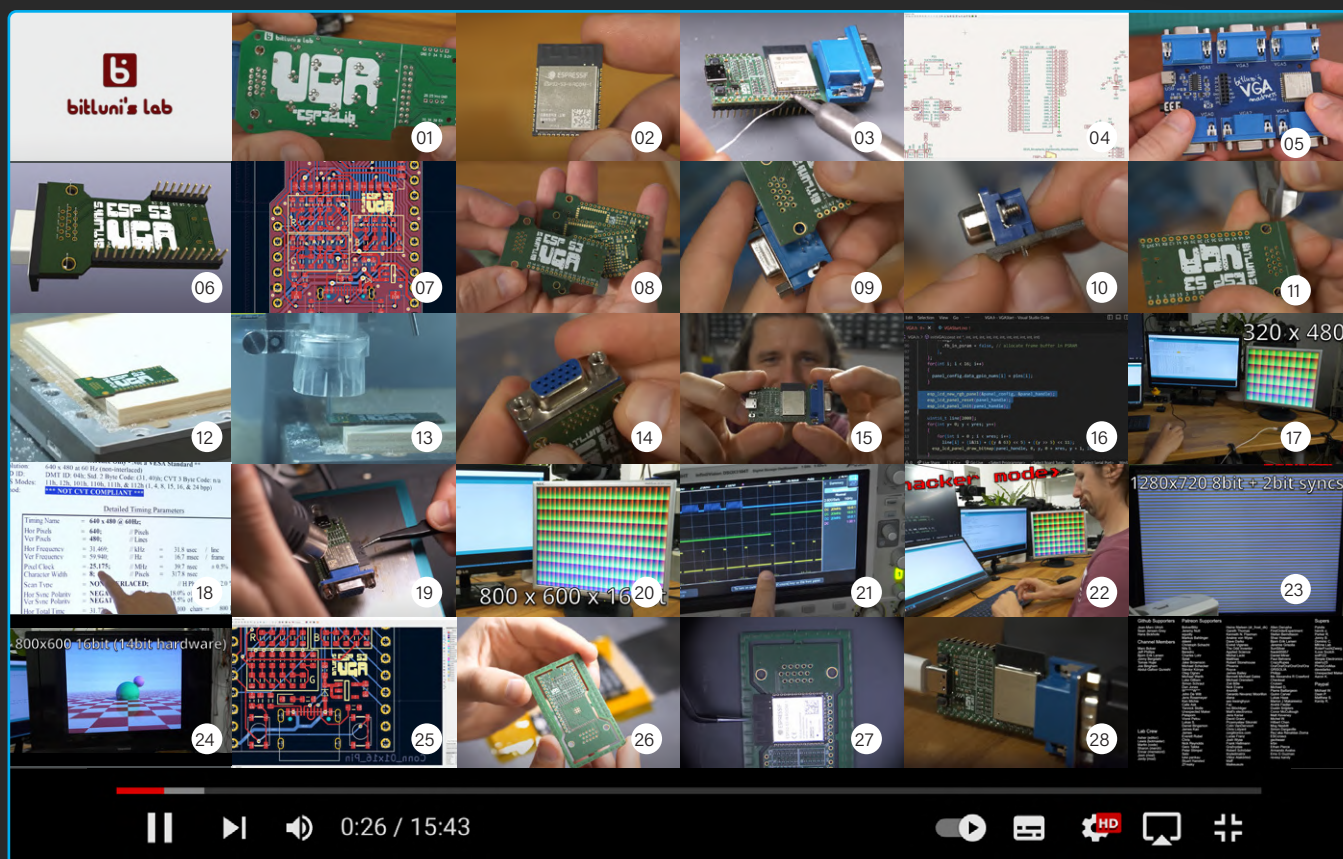
Daarna ging het snel. "Het lukte me eindelijk om een zuiver, continu signaal te krijgen," legt Bitluni uit. "Ik kon eindelijk een succes melden op de live-stream." Hij was zelfs in staat om er 1280×720p bij 8 bit uit te persen (**figuur 23**). Je ziet daar geen ruis, maar superkleine vierkantjes! Een dag later kon hij weer glimlachen (**figuur 24**). Vervolgens voegde hij extra bits toe, reorganiseerde de boel (**figuur 25**) en bestelde nieuwe printen. Een week later: **figuur 26**, **figuur 27** en **figuur 28**. Succes! ◀

230529-03

00:00:03:15

Fit





bitluni ✓

231K subscribers



Bekijk: "Ik heb een VGA-board gemaakt dat me versteld deed staan"



Gerelateerde producten

- Espressif ESP32-S3-EYE
www.elektor.nl/20626
- D. Ibrahim, *The Complete ESP32 Projects Guide* (Elektor, 2019)
www.elektor.nl/18860

Scan de QR-code...



...en bekijk de video over dit artikel in AR.

of



bekijk het direct via YouTube
<https://youtu.be/muuhgrige5Q>



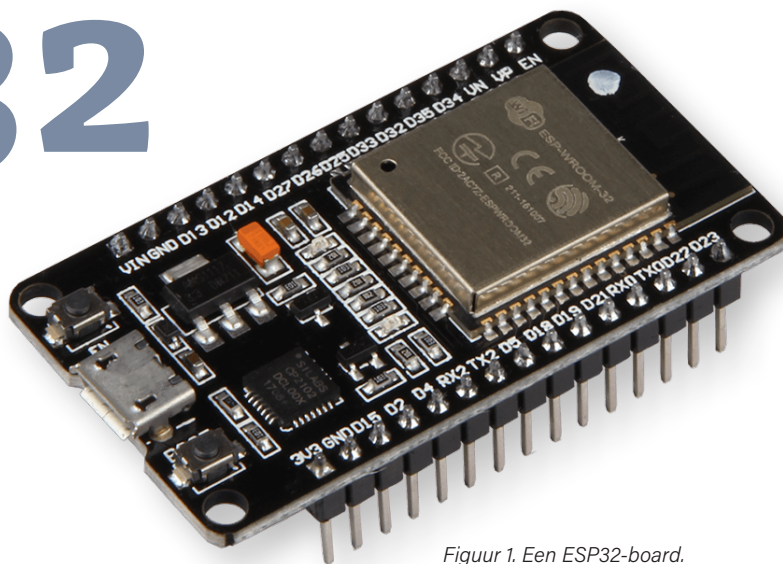
1/2



00:00:18:14

Acoustic fingerprinting met de ESP32

songherkenning met het
open-source project Olaf



Figuur 1. Een ESP32-board.

Joren Six, Universiteit Gent (België)

Een paar jaar geleden kreeg computerwetenschapper Joren Six de opdracht om een draagbaar computerplatform – dat wil zeggen, een microcontroller – een liedje te laten herkennen. Zijn experimenten met een ESP32 leidden tot een volwaardig multiplatform open-source project dat hij Olaf noemde, wat staat voor “Overly Lightweight Acoustic Fingerprinting”. Olaf is een bibliotheek voor muziekherkenning die eenvoudig te gebruiken is op embedded platforms met weinig geheugen en rekenkracht. Volg hem bij de ontwikkeling!

Ergens rond 2019 was ik computerwetenschapper aan de Universiteit Gent in België. Ik kreeg de opdracht om audio-herkenningstechnologie te ontwikkelen voor een elektronisch kostuum. Het concept bestond erin om de lampjes in het kostuum zich te laten synchroniseren met een specifiek liedje. Slechts één bepaald liedje zou de lampjes mogen activeren; alle andere muziek moest genegeerd worden. Normaal gesproken worden muziekherkenning en synchronisatie bereikt met behulp van acoustic fingerprinting-technieken. De uitdaging was dat dit op een betaalbare batterijgevoede microcontroller met weinig rekenkracht en geheugen moest werken. Destijds bouwde ik met goed gevolg een eerste prototype, maar het idee bleef in mijn hoofd rondspoken. Toen de vierde verjaardag van mijn dochtertje naderde, besloot ik het

prototype om te bouwen tot een extravagant verjaardagscadeau: een ‘Elsa’-jurk die reageert op het liedje “Let It Go” van de soundtrack van *Frozen* [1]. Voortbordurend op het eerste prototype bestelde ik een RGB-LED-strip, een robuuste Li-Ion-accu, een I²S digitale microfoon en natuurlijk een Elsa-jurk.

Ik had een ESP32-microcontroller (**figuur 1**) bij de hand en gebruikte deze als de centrale component van het systeem. Hij ondersteunt I²S, bevat een floating-point unit (FPU), is eenvoudig te combineren met LED-strips en heeft voldoende RAM-geheugen. De FPU vereenvoudigt het gebruik van dezelfde code op zowel conventionele computers als embedded apparaten, omdat fixed-point wiskunde niet meer nodig is.

Bouw van de ESP32-versie

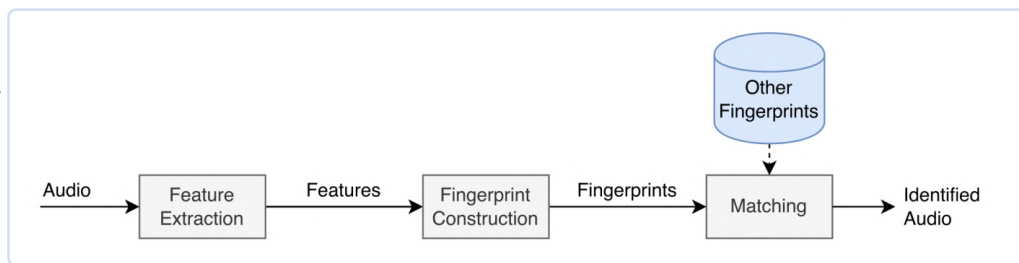
De eerste versie van het project maakte gebruik van een ESP32 Thing van Sparkfun – gekozen vanwege de geïntegreerde accuvoeding – in combinatie met een MEMS-microfoon, om precies te zijn een INMP441. Deze combinatie werd aangevuld met een LED-strip met individueel adresseerbare LED's, een generiek type dat gemakkelijk te vinden is op eBay of AliExpress.

Hardwarematig gezien is het een eenvoudige klus. Het is een kwestie van de I/O-pinnen van de microfoonmodule (SDA, SCK, WS) aan te sluiten op geschikte GPIO's op de ESP32, zoals GPIO 32, 33 en 25. En natuurlijk moet de microfoonmodule ook gevoed worden.

De bedrading van de LED's (**figuur 2**) is afhankelijk van het type LED-strip dat wordt gebruikt. Voor WS2812B-gebaseerde strips zijn drie draden (voeding, aarde, data) nodig. Ik heb de *FastLED*-bibliotheek gebruikt in de ESP32-code, dus als je het project wilt kopiëren, zorg er dan voor dat de LED-strip die je gebruikt compatibel is met *FastLED*. Na het solderen van de componenten en (met hulp van mijn partner) het innaaien van de LED-strip, was het project klaar. In de video die op [1] is te vinden, kun je het resultaat van onze inspanningen bekijken.



Figuur 2. RGB LED-strip.



Figuur 3. Blokschema van geluidsherkenning.

De video laat aanvankelijk een liedje horen dat niet wordt herkend en de lampjes niet activeert. Vervolgens wordt 'Let It Go' gespeeld en correct herkend. Als het liedje pauzeert, blijven de lampjes even aan voordat ze uitgaan, om onderbrekingen bij de herkenning op te vangen. Tot slot wordt het nummer hervat en opnieuw correct herkend. Je kunt op [2] meer lezen over de laatste updates van het project. Natuurlijk gebeurt, zoals meestal het geval bij microcontrollerprojecten, alle tovenarij in de code.

Onder de motorkap

Hoe kunnen we computers gebruiken om songs of muziek te herkennen? Dat kan op een aantal manieren. Een veelgebruikte benadering is herkenning op basis van spectrale pieken. Dit is de technologie die gebruikt wordt door Olaf en bekende muziekherkenningsdiensten zoals Shazam.

Het concept hierachter is vrij eenvoudig: de app neemt de audio die op je telefoon is opgenomen en zet deze om in een formaat dat een computer gemakkelijk kan vergelijken met andere nummers (figuur 3). In de geluidsleer wordt dit proces aangeduid als audio fingerprinting, waarbij een liedje wordt gecomprimeerd tot een unieke handtekening die herkenbaar blijft, zelfs te midden van veel achtergrondgeluid. Deze handtekeningen zijn gebaseerd op spectrogrammen, die de frequentie-inhoud van het nummer in de loop van de tijd visueel weergeven. Spectrogrammen geven ook het vermogensniveau van het signaal weer en tonen de waargenomen luidheid in de vorm van kleurvariaties.

Als Shazam echter rechtstreeks spectrogrammen als deze zou vergelijken, zou het extreem lang duren voordat er een resultaat verschijnt. Een belangrijke doorbraak was dat het algoritme zich concentreert op de pieken in het spectrogram, omdat deze precies die informatie

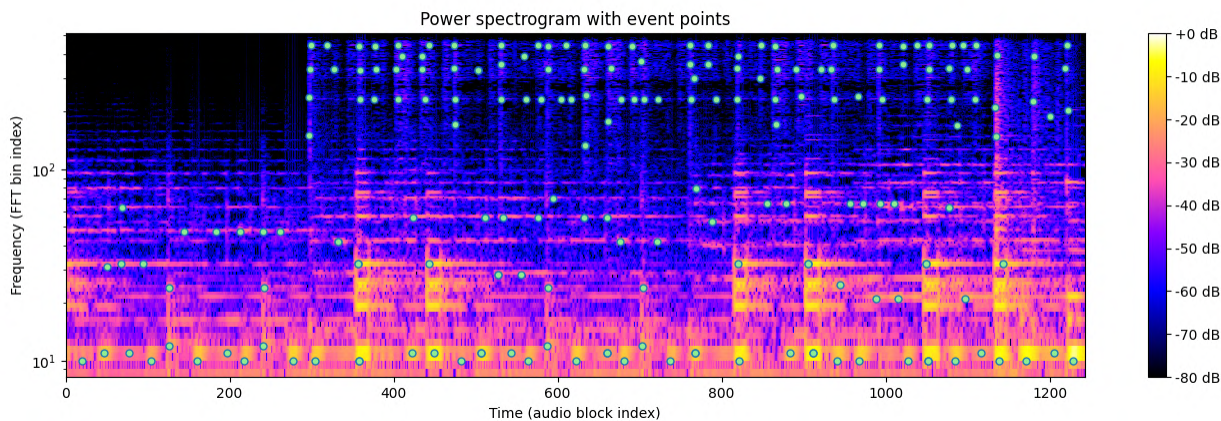
bevatten die ook door het menselijk brein wordt verwerkt.

Olaf neemt dus een geluidsmonster op en berekent vervolgens een spectrogram. Vervolgens wordt het spectrogram vereenvoudigd tot een spreidingsdiagram van de frequentiepieken. Een voorbeeld van zo'n spectrogram met gemarkeerde pieken is te zien in figuur 4. Nu moeten deze spreidingsplots, die in wezen de meest prominente signalen bij verschillende frequenties in de loop van de tijd weergeven, worden vergeleken met een database van een groot aantal bekende songs. Of, in het geval van Elsa's kostuum, met één specifieke song. In plaats van te zoeken naar een overeenkomst in een database voor al deze punten in de exacte volgorde, is een slimme techniek om naast elkaar liggende pieken te verbinden om zo een groot aantal paren te creëren. Vervolgens zoekt het algoritme naar corresponderende paren binnen een gestructureerde database die een willekeurig aantal nummers bevat. Als er genoeg overeenkomsten worden gevonden en deze in de loop van de tijd correct op elkaar aansluiten, kan Olaf (of Shazam) het nummer identificeren.

De ontwikkeling van het project

De initiële code voor het prototype uit 2019 was niet bijzonder goed georganiseerd. Tijdens mijn tweede poging voerde ik echter aanzienlijke verbeteringen door, waardoor ik een niveau bereikte dat ik de code kon delen op GitHub. Op dat moment kreeg het project de naam *Olaf – Overly Lightweight Acoustic Fingerprinting* [3].

De code onderging verschillende iteraties en breidde zich uit voorbij het oorspronkelijke doel, en ontwikkelde zich tot een veelzijdig, algemeen toepasbaar acoustic fingerprinting-systeem met talloze potentiële toepassingen. De indrukwekkende prestaties van Olaf kunnen onder andere worden toegeschreven aan het resource-efficiënte ontwerp en het gebruik van de PFFFT-bibliotheek.



Figuur 4. Een spectrogram laat zien hoe welke frequenties voorhanden zijn in muziek op een bepaald moment. De groene stippen zijn geëxtraheerd door Olaf en tonen pieken in het spectrogram.

Dit acroniem zegt sommige lezers misschien niet veel, vooral als net als bij Bob Pease je favoriete programmeertaal soldeert in! PFFFT [4] staat voor *Pretty Fast FFT*; het is een bibliotheek die is ontworpen als een veel lichter alternatief voor het bekende *FFTW*. Het is inderdaad compact en werkt lekker snel, waardoor het ideaal is voor de ESP32. Op embedded apparaten worden referentie-fingerprints opgeslagen in het geheugen, waardoor er geen database nodig is. Op traditionele computers worden de fingerprints opgeslagen in een krachtig databasesysteem – LMDB. Hoewel het buiten het bestek van dit artikel valt om de voordelen ervan te bespreken, moedig ik lezers aan om dit bij interesse verder te onderzoeken.

Olaf is niet langer alleen een coole gadget gebaseerd op ESP32. Het is veranderd in een volwassen applicatie/bibliotheek, ontworpen voor landmark-based acoustic fingerprinting. Olaf kan op een efficiënte manier fingerprints extraheren uit een audiostream, waarna deze kunnen worden opgeslagen in een database of overeenkomsten tussen geëxtraheerde en opgeslagen fingerprints kunnen worden geïdentificeerd.

Er bleken weinig lichtgewicht bibliotheken voor acoustic fingerprinting te bestaan die eenvoudig te implementeren zijn op embedded platformen, die vaak ernstige beperkingen hebben wat betreft geheugen en rekenkracht.

Olaf is geschreven in C, en ik heb geprobeerd om de code zo overdraagbaar mogelijk te houden. Hij richt zich voornamelijk op 32-bit ARM chips zoals bepaalde Teensy-modellen, specifieke Arduino-boards en de ESP32. Andere moderne embedded platformen met vergelijkbare specificaties kunnen ook compatibel zijn. Ongetwijfeld zal Olaf, dat open-source is, de bouw van innovatieve projecten stimuleren die muziek/geluidsherkenning en IoT-apparaten combineren!

Een belangrijk gevolg van deze inspanningen voor overdraagbaarheid is dat Olaf ook op PC's draait, en bovendien snel. De code kan gecompileerd worden om lokaal op je PC te draaien, maar er is een ander alternatief: Olaf kan ook in een webbrowser draaien! Zie het kader **Uitvoering in een webbrowser**.

De broncode, met een werkend voorbeeld voor ESP32 plus kleine debug-tools die ik heb geschreven, is te vinden in mijn GitHub-repository. Er is ook een PlatformIO-projectbestand ter referentie [6].

Bouw het zelf met een ESP32

Voor audioverwerking had het Olaf-project een beetje RAM nodig, wat vaak een probleem is bij veel microcontrollers. Met een key-value store op een PC had Olaf ongeveer 512 KB RAM nodig, terwijl de ESP32-versie aanzienlijk minder nodig had – ongeveer 200 KB. Ik werk slechts sporadisch met microcontrollers, dus een gebruiksvriendelijke omgeving is belangrijk: de beperkte tijd die ik met embedded apparaten doorbreng, moet niet verloren gaan met het configureren en onderhouden van een ontwikkelomgeving. De ESP32-IDE met PlatformIO of de Arduino IDE bleken aan die eisen te voldoen. Computationeel was er meer headroom en veel microcontrollers werkten (Teensy 3+, RP2040, elk ding met een Cortex-M). Een FPU zou kunnen helpen om het stroomverbruik laag te houden, vooral relevant als je met batterijen werkt.

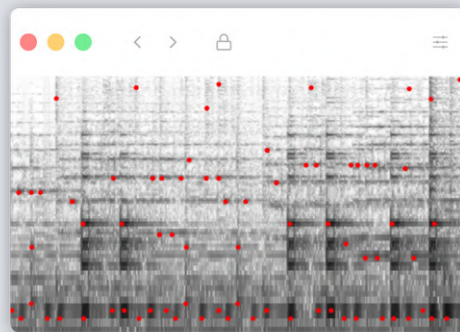
Ik heb ook een paar M5StickC's van M5Stack gebruikt, omdat ze een eenvoudig te gebruiken package vormen en een geïntegreerde microfoon hebben: ik ben meer van de softwarekant dan van het 'hardcore hacken' van hardware, dus dat was een handige keuze. Ik heb ook een

Uitvoering in een webbrowser

Heb je ooit van WebAssembly [5] (of WASM)? WASM is een overdraagbaar binair codeformaat, samen met een bijbehorend tekstformaat, ontworpen voor uitvoerbare programma's. Het belangrijkste doel is om de ontwikkeling van krachtige applicaties binnen webpagina's te vergemakkelijken.

Het blijkt dat er manieren zijn om C-code te compileren naar WASM, zoals de Emscripten-compiler. Volgens de website stelt Emscripten je in staat om C en C++ code op het web te draaien op bijna native snelheid, helemaal zonder de noodzaak van plugins. Het combineren van de Web Audio API met de WASM-versie van Olaf opent mogelijkheden voor webgebaseerde akoustische fingerprinting-toepassingen.

Op mijn blog kun je met Olaf experimenteren in je browser en mijn laatste berichten erover lezen. Precies dezelfde code die op de ESP32 draait, zoals eerder gedemonstreerd, draait nu in je browser. Dit betekent dat Olaf actief luistert om het liedje "Let It Go" van de Frozen-soundtrack te herkennen. Voor je gemak kun je links op je scherm het liedje afspelen vanuit de YouTube-video, en rechts kun je Olaf starten, zodat het binnenkomende audio kan analyseren. Olaf berekent de snelle fouriertransformatie (FFT) en visualiseert deze met behulp van Pixi.js. Na een paar seconden zouden de rode vingerafdrukken (zie screenshot) groen moeten worden, wat een succesvolle match aangeeft. Als je het nummer stopt, worden de vingerafdrukken uiteindelijk weer rood. Net als bij de eerder genoemde videodemonstratie duurt de overgang van een match naar geen match een paar seconden om rekening te houden met hiaten in de herkenning.



paar domotica-apparaten gebouwd (meer informatie op mijn website) en een paar andere projecten, zoals het regelen van de ventilatie in huis of het controleren van het niveau van de regenwatertank.

Tijdens het voorbereiden van dit artikel heb ik code voor een ESP32-demo en aanvullende documentatie toegevoegd aan mijn GitHub-repository [7]. Nu werkt de laatste versie van Olaf betrouwbaar op ESP32 met een goed verkrijgbare MEMS-microfoon. Op die helppagina vind je ook instructies over het gebruik van I²S-microfoons zoals de INMP441. Er is een demoprogramma dat audio van de microfoon via WiFi naar een computer stuurt, zodat je naar de microfoon kunt luisteren. Zo weet je dat de microfoon naar behoren werkt en dat de audiosamples correct worden geïnterpreteerd. Het valideert de I²S-instellingen, inclusief bufferomvang, sample rates, audioformaten en stereo of mono instelling.

Er is nog een ander codevoorbeeld dat laat zien hoe je binnenkomende audio van een INMP441 microfoon kunt matchen met fingerprints die zijn opgeslagen in een headerbestand. In tegenstelling tot de PC versie gebruikt de embedded versie van Olaf geen key-value opslag, maar een lijst van hashes opgeslagen in het `src/olaf_fp_ref_mem.h` header-bestand, wat dient als de fingerprint-index.

Standaard is `src/olaf_fp_ref_mem.h` opgenomen in de ESP32-code. Om dit header-bestand te testen en te debuggen, kun je de mem-versie van Olaf opvragen op je computer: `bin/olaf query olaf_audio_your_audio_file.raw "arandomidentifier"`. De ESP32-versie is in essentie identiek aan de mem-versie, met als enige verschil dat de audio in de ESP32-versie afkomstig is van een MEMS-microfoon en niet uit een bestand.

Als je er zeker van bent dat de mem-versie van Olaf werkt zoals bedoeld en de INMP441-microfoon is getest, kan de ESP32-versie worden overgebracht naar de ESP32-hardware met behulp van de Arduino IDE.

Hoe gaat het verder?

Ik hoop dat dit artikel je op ideeën heeft gebracht voor projecten met akoestische herkenning. Het is altijd spannend om te zien hoe je, door te beginnen met een klein, speels project en het steeds verder te verfijnen, indrukwekkende resultaten kunt bereiken. In dit geval ging het project van een eerste prototype dat niet helemaal aan mijn verwachtingen

voldeed naar een tweede prototype op basis van een ESP32, dat verschillende verbeteringen opleverde en ongetwijfeld veel plezier voor mijn dochter (figuur 5).



Figuur 5. Het kostuum met ESP32-lichteffecten.

Na verloop van tijd en door iteratieve verfijningen ontwikkelde het project zich tot een uitgebreide, overdraagbare, open-source, cross-platform, high-performance applicatie en bibliotheek. Ik heb twee academische artikelen over het onderwerp geschreven om ervoor te zorgen dat dit werk erkend en gewaardeerd kan worden binnen de onderzoeksgemeenschap. De vraag aan de lezers is nu: welke spannende projecten heb jij in gedachten om je ESP32 voor in te zetten? 

230578-03

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via joren.six@ugent.be of naar de redactie van Elektor via redactie@elektor.com.



Over de auteur

Joren Six is een computerwetenschapper die onderzoek doet op het gebied van muziekinformatica, Music Information Retrieval en computationele etnomusicologie. Hij behaalde een PhD aan de Universiteit Gent, België, en is momenteel betrokken bij verschillende projecten waarin IT en akoestiek worden gecombineerd.



Gerelateerde producten

- > **JOY-iT NodeMCU ESP32 Development Board**
www.elektor.nl/19973
- > **ESP32-DevKitC-32E**
www.elektor.nl/20518
- > **ESP32-C3-DevKitM-1**
www.elektor.nl/20324

WEBLINKS

- [1] Eerste prototype van het project: https://0110.be/posts/Olaf_-_Acoustic_fingerprinting_on_the_ESP32_and_in_the_Browser
- [2] Laatste updates van het project: <https://0110.be/search?q=olaf>
- [3] De project-repository: <https://github.com/JorenSix/Olaf>
- [4] Pretty Fast FFT-bibliotheek: <https://bitbucket.org/jpommier/pfft/src/master/>
- [5] WebAssembly bij Wikipedia: <https://en.wikipedia.org/wiki/WebAssembly>
- [6] Downloads bij dit artikel: <https://0110.be/files/attachments/475/ESP32-Olaf.zip>
- [7] ESP32 Olaf-project op GitHub: <https://github.com/JorenSix/Olaf/tree/master/ESP32>



Ronde kerstboom 2023

een high-tech manier
om de feestdagen te vieren

Ton Giesberts (Elektor)

De WS2812D-F8 is een high-tech, digitaal programmeerbare 8 mm RGB-LED die individueel kan worden geadresseerd in serieschakelingen van maximaal 255 stuks. In dit driedimensionale ronde kerstboomproject kunnen 36 van deze LED's extern of door een on-board Arduino Nano ESP32 aangestuurd worden. De lichteffecten zijn verbluffend – deze stralende Elektor-kit mag met de feestdagen niet in je woonkamer ontbreken!

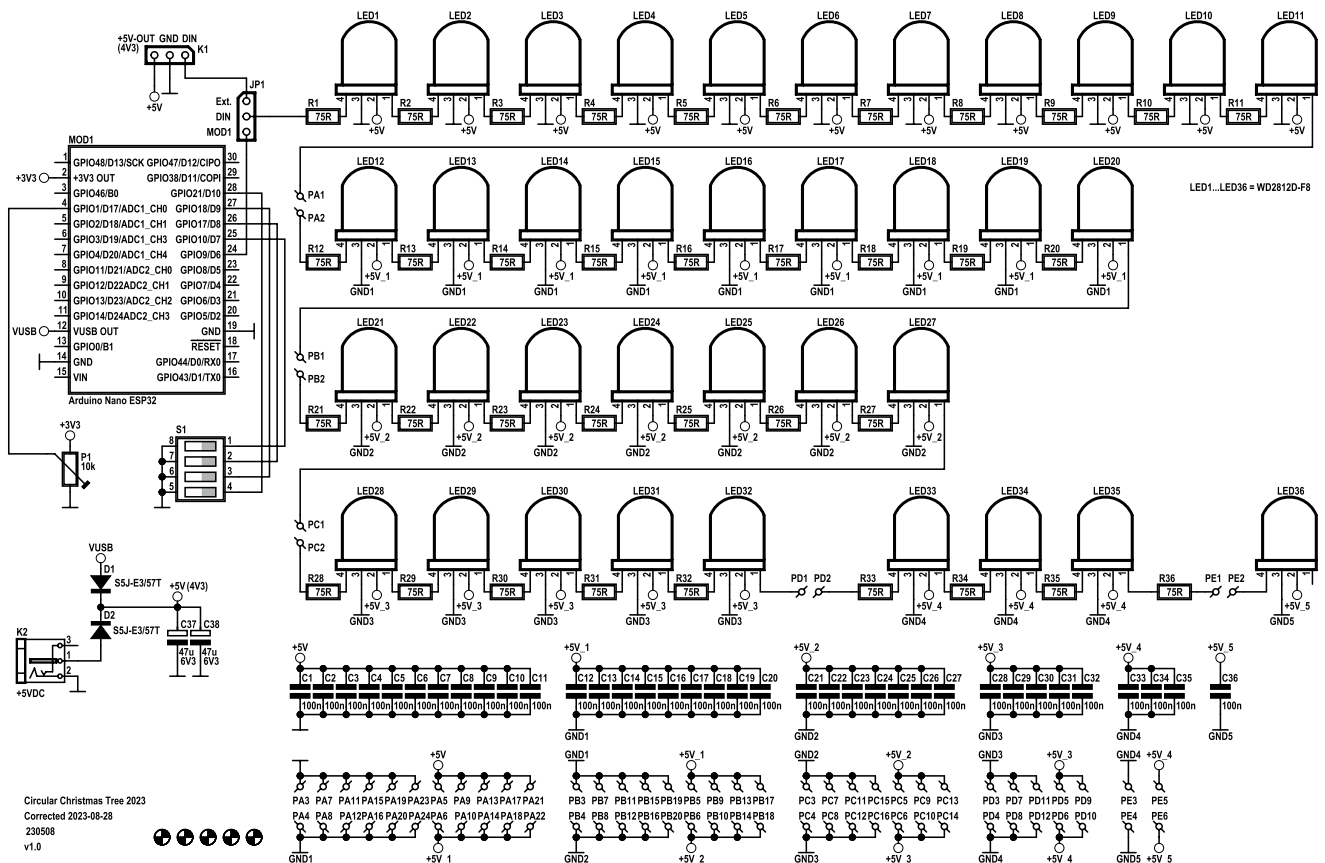
Bij deze kerstboom draait alles om de vorm van de constructie en het gebruik van 8 mm digitale RGB-LED's. Hij lijkt meer op een echte boom dan de meeste platte printen met een kerstboomachtige omtrek. In plaats van een gewoon 2D-ontwerp is dit een echt 3D-ontwerp, en het is gebaseerd op de twee eerdere versies [1] en [2]. Dit nieuwe ontwerp wordt gerealiseerd door de vijf ringvormige, concentrische secties van de print die bij de kit wordt geleverd, te scheiden en ze boven elkaar te monteren met enkele stijve draden die de print van elkaar scheiden, waardoor dit project dat typische naaldboom-uitelijk krijgt.

Het project in [1] gebruikte een eenvoudige toevalsgenerator met twee standaard logische IC's en kleine witte SMD-LED's. [2] was gebaseerd op een microcontroller, met de LED's aangesloten in een 6x6 matrix. Deze nieuwste versie gebruikt 36 stuks 8 mm digitale RGB-LED's voor through-hole montage van het type WS2812D-F8. Dit maakt de schakeling eenvoudig. In plaats van een matrix zijn de in- en uitgangen van de digitale LED's in serie geschakeld, waarbij elke LED is aangesloten op een 5V-voeding. De keten van LED's kan worden aangesproken als een NeoPixel LED-strip. Er is een groot aantal programma's op het web te vinden, in vele talen, die in detail uitleggen hoe je deze componenten kunt aansturen.

Connector K1 is verbonden met de interne +5 V (4,3 V), massa en de ingang van de eerste LED (via jumper JP1). Op deze manier kunnen verschillende externe microprocessors, microcontrollers of modules worden gebruikt om de LED's van onze ronde kerstboom aan te sturen. Om de boom onafhankelijk van externe elektronica te maken, kan een Arduino Nano ESP32-module [3] op de basisprint worden gemonteerd.

Het schema

De schakeling bestaat in wezen uit 36 digitale RGB-LED's van het type WS2812D-F8, waarvan de ingangs- (D_{in}) en uitgangspinnen (D_{out}) in serie zijn geschakeld; elke LED wordt gevoed met een 4,3V-voedingspanning (**figuur 1**). De werkelijke spanning over de LED's is lager vanwege de spanningsval die wordt veroorzaakt door de beveiligingsdiodes D1 en D2. De LED's kunnen worden aangestuurd door een externe microprocessor, microcontroller, module of optionele on-board Arduino Nano ESP32-module (MOD1).

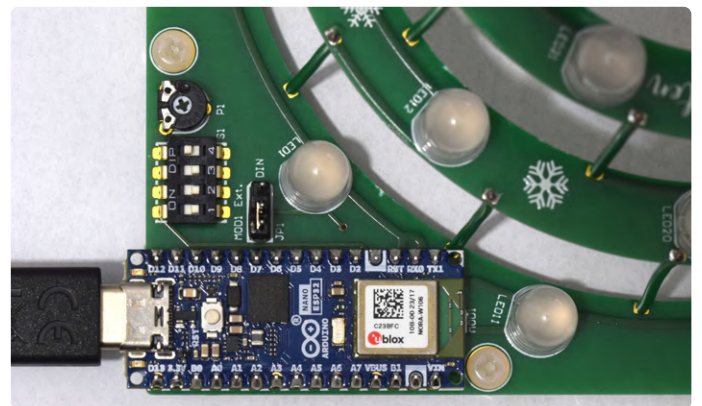


Figuur 1. Schema van de ronde kerstboom.

Jumper JP1, zichtbaar in **figuur 2** vlakbij het Arduino Nano-board, selecteert het stuursignaal voor LED1 (die de eerste ontvangende component is van de 36 in serie geschakelde LED's). Het gebruik van een Arduino Nano ESP32 maakt de boom veelzijdiger. Via software is het mogelijk om de verlichtingspatronen van de boom op afstand te wijzigen met behulp van de ingebouwde WiFi en/of Bluetooth LE. Alle LED-ingangen hebben een weerstand van 75 Ω in serie (R1...R36, volgens de voorschriften in de datasheet van de WS2812D-F8 [4]). Elke LED is ontkoppeld door een condensator van 100 nF (C1...C36). De voeding voor de LED's op de basisprint is extra ontkoppeld door de 47 μ F-taantalcondensatoren C37 en C38.

Het zijn allemaal SMD-componenten die aan de onderkant van de printen worden gemonteerd, zodat ze uit het zicht blijven. De LED's kunnen worden gevoed uit een 5VDC-adaptor (die bij voorkeur minstens 1,5 A kan leveren) via DC-voedingsconnector K2, of via de VBUS-pin op het Arduino Nano-board (indien aanwezig) in het MOD1 PCB-gebied. Diodes D1 en D2 (5 A, 50 VDC, S5J-E3/57T, SMD-behuizing, SMC-formaat), ook gemonteerd aan de onderkant van de print, voorkomen schade als de twee voedingen tegelijkertijd worden aangesloten. Het is sowieso mogelijk om een voeding aan te sluiten op zowel MOD1 als K2, maar het is niet noodzakelijk. De module zelf wordt gevoed via de USB-C connector, en de maximale stroom van de VBUS-pin is voldoende (ervan uitgaande dat de aangesloten AC-adaptor krachtig genoeg is, natuurlijk).

De diodes zijn bewust geen Schottky-types. De spanningsval over de diodes is hoger en we moeten ervoor zorgen dat logische 3,3V-signalen ruim binnen de WS2812D-F8 specificatie vallen. Om deze reden



Figuur 2. Detail van MOD1, S1 en P1 op de basisprint.

is het, als de LED-keten door een externe schakeling wordt aangestuurd, altijd de beste optie om de schakeling te voeden via de interne voedingsbron op K1. De feitelijke spanning op K1 zal lager zijn dan de 5 V van K2 – ongeveer 4,3 V (of nog minder, bij hoge stroom/helderheid-instelling) – vanwege de spanningsval over D2.

Om de printen met elkaar te verbinden, zorgen 39 draadsegmenten voor het doorgeven van de +5 V en massa van de ene print naar de print erboven (**figuur 3**). In het schema zijn dit de aansluitingen PA1–PA2 tot en met PE5–PE6. Eén extra draad verbindt de D_{out}-pin van elke LED op een lageregelegen print met de eerste LED op de print erboven.



Figuur 3. De onderste draad is gemarkeerd en de bovenste is gestript. De overblijvende isolatie moet 3 cm lang zijn.

Als een Arduino Nano ESP32 wordt gebruikt om de LED's aan te sturen, kunnen DIP-schakelaar S1 en de kleine instelpot P1, beide te zien in figuur 2, ook op de print worden gemonteerd. Software kan deze componenten uitlezen om verschillende patronen te selecteren of de helderheid aan te passen. Zonder de onboard-module hebben deze componenten echter geen functie.

Voeding

Als voeding voor de LED's dient een +5 V AC/DC-adapter met een voedingsplug die wordt aangesloten op K2. De kit die je in onze shop kunt kopen, bevat alleen de onderdelen voor de print, inclusief de kleine instelpot en de DIP-schakelaars. De AC/DC-adapter en optionele Arduino Nano ESP32-module zijn niet inbegrepen en moeten apart worden aangeschaft.

Deze module heeft een USB-C connector die ook de LED's voedt via zijn VBUS-pin. Zoals al opgemerkt voorkomen twee diodes (D1, D2) dat de beide voedingen rechtstreeks met elkaar worden verbonden. VBUS is intern verbonden met de 5 V van de USB-C connector via een Schottky-diode van het type PMEG6020AELR (Nexperia). Deze diode is gespecificeerd voor een nominale stroom van 2 A. Op dit moment is de maximale stroom van de VBUS-pin niet gespecificeerd in de datasheet [5]. De stroom voor de WS2812D-F8 is 12 mA, volgens een overzicht van deze familie op de website van de fabrikant.

De datasheet vermeldt echter 15 mA bij de specificaties, en veel parameters zijn gemeten bij $I_F = 20$ mA. In ons prototype was de maximale stroom bij maximale instelling (3×255 decimaal per LED) net iets minder dan 11 mA per kleur – ver van 20 mA verwijderd. Deze verschillen met de specificaties zijn een beetje verwarrend. Blijkbaar is een meer realistische waarde voor de stroom ongeveer 12 mA. Ervan uitgaande dat dit de juiste waarde is, bedraagt de maximale stroom bij maximale LED-helderheid:

$$36 \text{ (LED's)} \times 3 \text{ (kleuren)} \times 12 \text{ mA} = 1,3 \text{ A.}$$

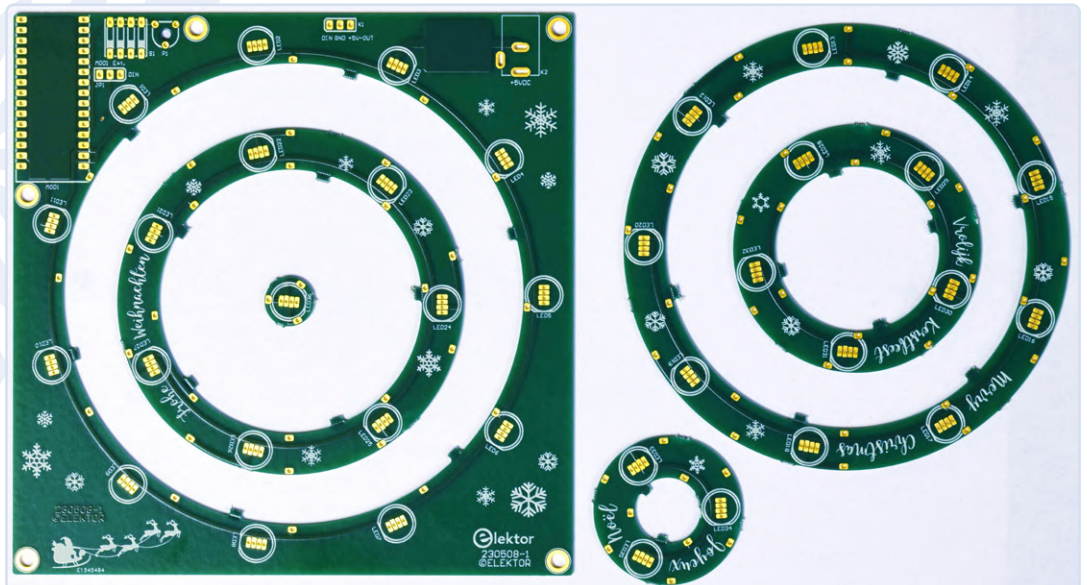
In ons kerstboom-prototype bedroeg de totale gemeten maximale stroom 1156 mA. De VBUS-pin op MOD1 zou geen probleem moeten hebben met deze stroom. Maar in het algemeen raden we je sterk aan om de LED's nooit bij de maximale stroom te gebruiken, omdat dit de levensduur van de LED's aanzienlijk verkort! Het helderheidsverschil tussen bijvoorbeeld een derde en de maximale nominale stroom is niet groot. Houd hier rekening mee bij het schrijven van de software. Als je alle LED's instelt om wit licht te geven, voor zeer heldere verlichting in een goed verlichte kamer, heb je een totale stroom van ongeveer 200 mA nodig. Dit zal in de meeste gevallen meer dan genoeg zijn en je kunt elke standaard USB type-A poort gebruiken om de module van stroom te voorzien.

De print

Het zal je niet verbazen dat de print van 136×136 mm uit de basisprint bestaat plus de vijf cirkelvormige printen die daaruit losgebroken kunnen worden.

De figuren bij de onderdelenlijst tonen de print zoals die wordt geleverd, terwijl **figuur 4** alle deelprinten toont na het doorsnijden van de verbindingstabs. Bedenk dat het enige kracht kost om ze te breken. Een montagehandleiding, beschikbaar op Elektor Labs [6], geeft alle details over het bouwen van de boom. Het koper van de 39 draadsegmenten is 0,8 mm dik. Deze draden helpen ook om de boom vorm te geven en daarom hebben we groene geïsoleerde draad gekozen. Maar als je een naakte vertinde draad mooier vindt, kun je natuurlijk alle isolatie verwijderen.

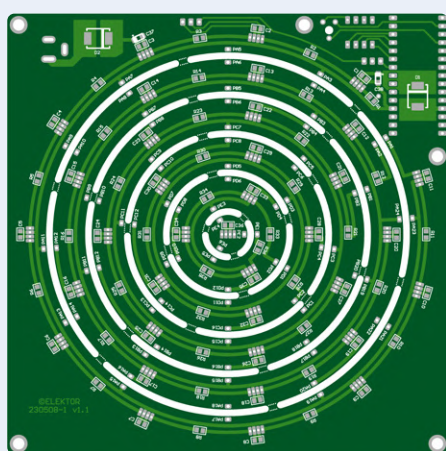
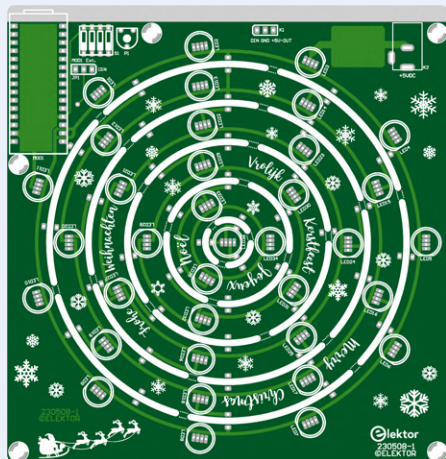
Let goed op dat alle printplaten evenwijdig worden gemonteerd, met een onderlinge afstand van 3 cm. Het grote aantal draden maakt de constructie vrij stijf, zoals je kunt zien in **figuur 5**. Dat toont het volttooide prototype zonder de extra module. Om de spanningsval van onder naar boven minimaal te houden, zijn alle draden, behalve de signaaldraad, om en om verbonden met +5 V en massa. Pas bij het strippen van alle isolatie op voor kortsluiting tussen de naakte draden!



Figuur 4. Alle printplaten van elkaar gescheiden. De breekstabs die de 6 printen bij elkaar hielden, kunnen nu verwijderd worden met een draadtang.



Onderdelenlijst



Weerstanden:

R1...R36 = 75 Ω , 0,125 W, 5%, SMD 0805

P1 = instelpotmeter liggend 10 k Ω , 0,1 W, 20 %, 6 mm rond
(Piher PT6KV-103A2020)

Condensatoren:

C1...C36 = 100 n, 50 V, 5%, X7R, SMD 0805

C37, C38 = 47u, 6V3, 10%, tantaal, case A (1206)

Halfgeleiders:

D1, D2 = S5J-E3/57T, SMD-behuizing SMC

LED1-LED36 = WS2812D-F8, 8 mm, THT

MOD1 (optioneel) = Arduino Nano ESP32 met headers

Diversen:

K1, JP1 = pinheader, 3x1, verticaal, raster 2,54 mm

Jumper voor JP1, steek 2,54 mm

K2 = MJ-179PH (Multicomp Pro), DC-voedingsconnector, 4 A,
pindiameter 1,95 mm

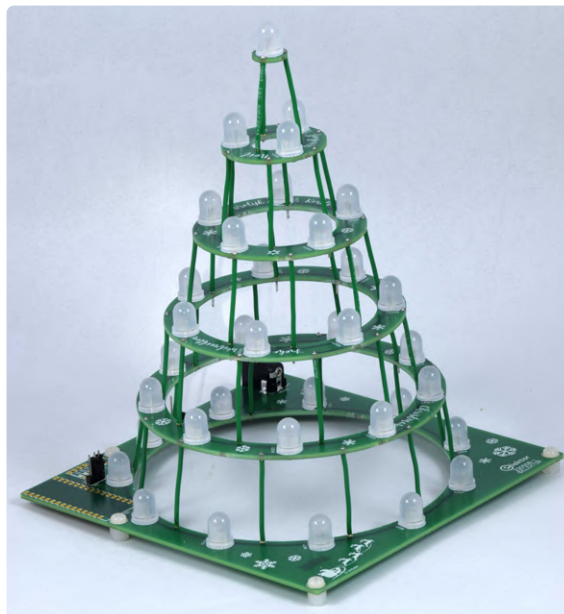
S1 = DIP-schakelaar, 4-voudig

PA1...PE6 = 2 m draad, 0,81 mm massief, 0,52 mm²/20AWG, groene
isolatie (Alpha Wire 3053/1 GR005)

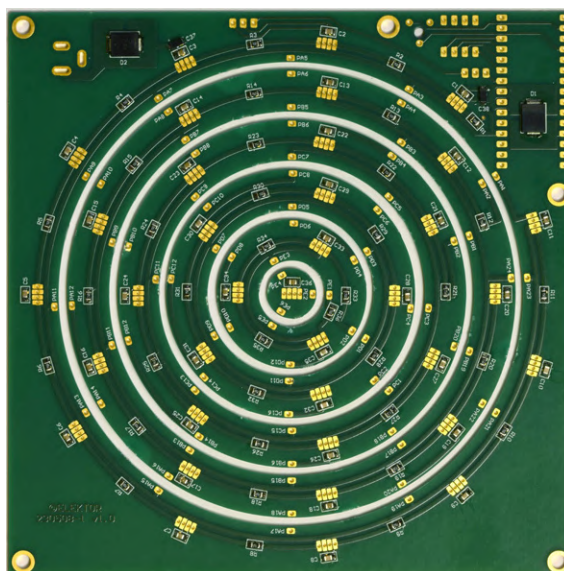
H1...H5 = nylon afstandsbus, female/female, M3, 5 mm

H1...H5 = nylon boutje, M3, 5 mm

Print 230508-1



Figuur 5. Het voltooide prototype, zonder de optionele Arduino-module.



Figuur 6. Alle SMD's zijn aan de onderkant van de printen gesoldeerd: R1...R36, C1...C38, D1 en D2.

De sporen voor de voeding zijn 1 mm breed. Alle SMD-componenten worden aan de onderkant van de printen gemonteerd, zoals te zien is in **figuur 6**. Zorg ervoor dat je de printen van elkaar scheidt voordat je begint te solderen. Behalve de LED's zitten er geen andere componenten aan de bovenkant van de cirkelvormige printen; de aansluitdraden van die LED's moeten worden doorgeknipt zoals te zien in **figuur 7**. De through-hole componenten JP1, K1, K2, P1, S1 en MOD1 worden aan de bovenzijde van de vierkante basisprint geplaatst. Aangezien in dit project SMD-componenten worden gebruikt, is enige ervaring met het solderen daarvan aanbevolen.

Met een soldeerbout met een fijne punt en dunne soldeerdraad zijn de 0805-weerstanden en -condensatoren niet zo moeilijk te solderen. We raden je aan om heel dun soldeer te gebruiken met een doorsnede van 0,35 mm, zodat er niet te veel tin wordt gebruikt voor de soldeerverbindingen. De boom staat op vijf witte nylon afstandsbusen van 5 mm die zijn vastgezet met 5 witte nylon schroeven. Deze vallen minder op.



Figuur 7. De aansluitdraden van de RGB-LED's van het type WS2812D-F8 zijn net boven de verbreding afgeknipt.

De optionele module MOD1 wordt in een hoek van de print geplaatst om de basis zo klein mogelijk te houden. P1 en S1 worden naast de module gemonteerd.

Optionele Arduino Nano ESP32-module

Om deze kerstboom onafhankelijk van externe schakelingen te laten werken, kan de Arduino

Nano ESP32-module (MOD1) op de basisprint worden gemonteerd, samen met instelpot P1 en DIP-schakelaars S1. Zet jumper JP1 in de stand MOD1. De twee extra componenten kunnen worden gebruikt om de helderheid/snelheid en verschillende patronen/modi in te stellen en/of te selecteren.

De module kan worden geprogrammeerd met de nieuwste versie van de Arduino IDE en de programmeertaal C. Als je op zoek bent naar een MicroPython-project, dan is deze kerstboom ook een goede keuze. Een speciale IDE met de naam *Arduino Lab for MicroPython* [7] kan worden gedownload van GitHub. Ook is een MicroPython-installer [8] nodig; die is ook beschikbaar op GitHub. Het is allemaal goed gedocumenteerd op de Arduino-website en geeft beginners goede mogelijkheden om het geheel aan de praat te krijgen [9].

Elektor heeft basissoftware ontwikkeld die kan worden gedownload van de Elektor Labs-webpagina van dit project [6], waar je ook wat details over de functionaliteit vindt.

Tijdens de ontwikkeling van de software zou je het project moeten kunnen voeden via de USB-connector van een PC, maar een AC/DC-adaptor met een UCB-C connector is nodig om de boom van stroom te voorzien als hij klaar is. De maximaal beschikbare stroom hangt natuurlijk af van de specifieke poort. Via een USB-C/USB-A adapter of adapterkabel kan de Arduino Nano ook worden aangesloten op een 'legacy' USB-poort als de helderheid beperkt blijft. De stroom is dan beperkt tot 500 mA, iets om in gedachten te houden bij het testen van de software! Of sluit ook een voeding aan op K2 en maak de spanning iets hoger dan 5 V, zodat de LED's van stroom worden voorzien. **Figuur 8** laat niet echt de volle, heldere kleur van de LED's zien, maar geeft toch een goede indruk van wat er met verschillende instellingen bereikt kan worden. ◀

230665-03



Over de auteur

Ton Giesberts begon na zijn studie bij Elektuur (nu Elektor), toen daar gezocht werd naar iemand met affiniteit voor audio. In de loop der jaren heeft hij voornamelijk aan audioprojecten gewerkt. Analoog ontwerpen heeft altijd zijn voorkeur gehad. Projecten op andere gebieden van de elektronica horen er natuurlijk ook bij. Een van Ton's motto's is: "Als je iets beter wilt hebben, doe het dan zelf". Bij een print voor een audioproject met vervormingscijfers in de orde van grootte van 0,001% is een goede layout bijvoorbeeld cruciaal!

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de redactie van Elektor via redactie@elektor.com.



Gerelateerde producten

- > **Circular Christmas Tree Kit**
www.elektor.nl/20672
- > **Arduino Nano ESP32 with Headers**
www.elektor.nl/20529



Figuur 8. Testopstelling van de kerstboom.

WEBLINKS

- [1] Kerstboom V1: <https://elektormagazine.com/labs/130478-xmas-tree-2014>
- [2] Kerstboom V2: <https://elektormagazine.com/labs/circular-christmas-tree-150453>
- [3] Documentatie van de Arduino Nano ESP32: <https://docs.arduino.cc/hardware/nano-esp32>
- [4] WS2812D-F8 datasheet [PDF]: https://soldered.com/productdata/2021/03/Soldered_WS2812D-F8_datasheet.pdf
- [5] Arduino Nano ESP32 datasheet [PDF]: <https://docs.arduino.cc/resources/datasheets/ABX00083-datasheet.pdf>
- [6] Dit project op Elektor Labs: <https://elektormagazine.com/labs/circular-christmas-tree-2023-230508>
- [7] Arduino Lab for Windows [Zip]: <https://tinyurl.com/arduino4win>
- [8] Arduino MicroPython Installer: <https://github.com/arduino/lab-micropython-installer/releases/tag/v1.2.1>
- [9] MicroPython course MicroPython 101 by Arduino: <https://docs.arduino.cc/micropython-course/>

Een eenvoudiger en handiger leven



een amateurproject op basis van de ESP8266-module van Espressif

Een bijdrage van Transfer Multisort Elektronik Sp. z.o.o.

Elk jaar wordt het concept van het smart home steeds populairder, en komen er meer oplossingen beschikbaar die ons helpen om onze leefruimte efficiënter te beheren. Bepaalde producten die op de markt verschijnen, zijn compatibel met oudere apparaten, waardoor we de bestaande apparatuur samen met de nieuwste technologische ontwikkelingen kunnen gebruiken. Het op afstand bedienen van huishoudelijke apparaten en het automatiseren van verschillende processen helpt de energie-efficiëntie te verbeteren, het milieu te beschermen, ons comfort te verhogen en geld te besparen. Dit project, een Smart ESP8266 afstandsbediening die is ontwikkeld voor een wedstrijd van TechMasterEvent, combineert al deze voordelen.

Espressif is een producent van goed gewaardeerde SoC geïntegreerde schakelingen en draadloze communicatiemodules, veel waarvan verkrijgbaar zijn bij TME. Dankzij hun compacte afmetingen en gering energieverbruik kunnen de producten van Espressif met succes worden toegepast in zowel consumenten- als industriële elektronica.

Hieronder lees je over een apparaat gebaseerd op de ESP8266-module. Het is een amateurproject gemaakt door een deelnemer aan een TechMasterEvent-wedstrijd. Er werd gevraagd om een elektronica-project te ontwerpen dat het leven gemakkelijker moest maken.

Je kunt de ESP8266-module en verschillende andere onderdelen die nuttig kunnen zijn bij

het bouwen van je IoT-projecten (single-board computers, communicatie- en geheugenmodules, displays en nog veel meer) vinden op [1].

ESP8266, IR-LED and IR-ontvanger

Smart SP8266 afstandsbediening is een project dat als doel heeft om het bedienen van je apparaten in huis eenvoudig te maken. Door het gebruik van een ESP8266, IR-LED en IR-ontvanger elimineert dit project de noodzaak van meerdere afstandsbedieningen voor verschillende apparaten zoals airconditioners of televisies. Het project maakt verbinding met een telefoon-app, zodat gebruikers

eenvoudig commando's naar hun apparaten kunnen sturen en zelfs de signalen van hun huidige afstandsbedieningen kunnen opslaan voor toekomstig gebruik.

Behalve het gemak en het gebruiksgemak is de Smart ESP8266 afstandsbediening ook een prima oplossing voor oudere apparaten die mogelijk niet compatibel zijn met traditionele smart home-technologie. Met de mogelijkheid om signalen van traditionele afstandsbedieningen in te lezen en op te slaan, kun je met de Smart ESP8266 afstandsbediening oudere apparaten bedienen die misschien geen verbinding kunnen maken met internet of andere smart home-systemen. Dit maakt het een kostenbesparend alternatief voor het upgraden van je apparaten of de aanschaf van dure smart home-apparaten.

Naast de hardwarecomponenten heeft de Smart ESP8266 afstandsbediening ook software nodig om te functioneren. Je vindt deze op [2].

De Smart ESP8266 afstandsbediening biedt een aantal voordelen, zoals gemak en eenvoudige bediening, kosteneffectiviteit en flexibiliteit. Er hoeven geen dure smart home-apparaten aangeschaft of oudere apparaten geactualiseerd te worden, waardoor het een kosteneffectief alternatief is. Het project is bovendien flexibel genoeg om eenvoudig te worden aangepast om te werken met verschillende apparaten en verschillende IR-protocollen, waardoor het een veelzijdige oplossing is voor het bedienen van verschillende apparaten met één enkele app. ◀

230656-03

WEBLINKS

[1] TME-shop: <https://tme.eu>

[2] Broncode van dit project: <https://techmasterevent.com/project/how-to-make-old-devices-smarter-with-a-esp8266>

IoT-apps bouwen zonder software-expertise

met het Blynk IoT-platform en Espressif-hardware

Een bijdrage van Blynk Inc.

Wat als u een mobiele app kon ontwikkelen zonder een regel code te schrijven, deze kon branden en binnen een maand uitbrengen in app-stores? Productieklare IoT-software lanceren zonder software-engineers in dienst te nemen? Met Blynk IoT duurt dat geen jaren maar is dat binnen een maand mogelijk!

Blynk app-builder voor iOS en Android

Hiermee kunt u snel prototypes en volledig functionele apps maken zonder dat u moet kunnen programmeren. In de constructor-modus kunt u kiezen uit meer dan 50 aanpasbare UI-elementen zoals knoppen, schuifregelaars, grafieken, kaarten, meters enzovoort, en deze naar het canvas slepen om een aangepaste UI voor uw connected product te creëren.

Web dashboard-builder

Het heeft een vergelijkbare architectuur voor het maken van historische en realtime datavisualisaties en voor het besturen en bewaken van apparaten met behulp van voorgedefinieerde UI-elementen.

De firmware-bibliotheek van Blynk ondersteunt:

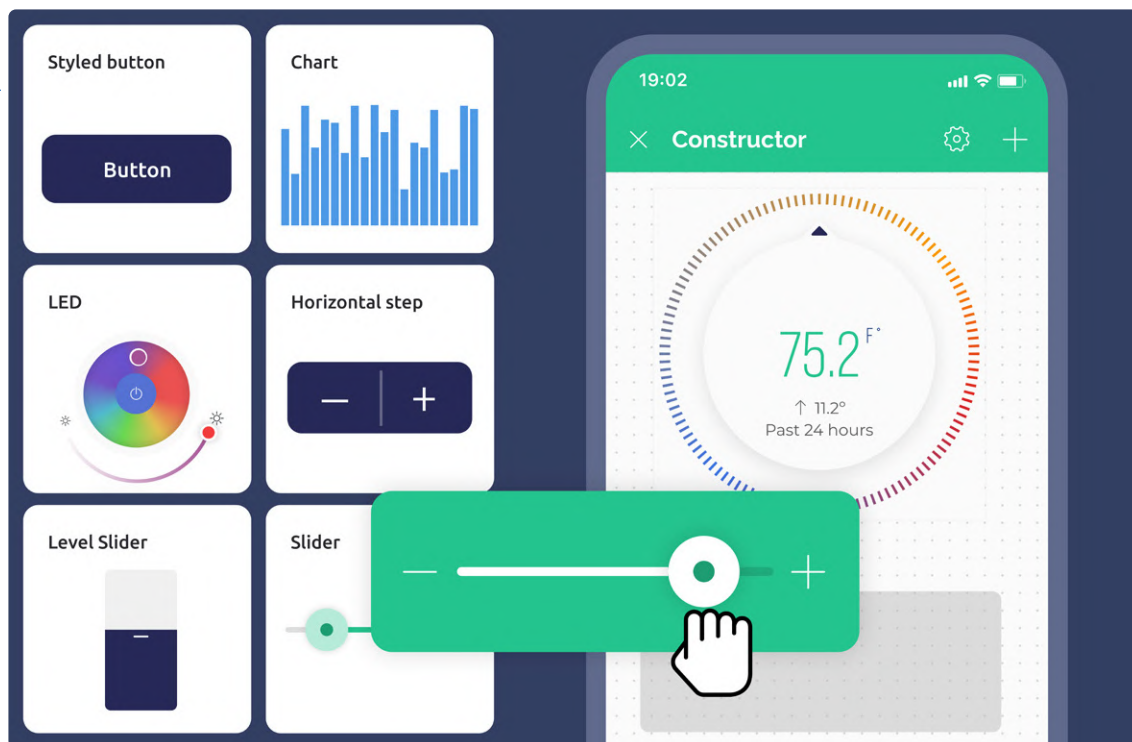
- ESP32
- ESP32-S2
- ESP32-S3
- ESP32-C3
- ESP8266
- en andere

Wat omvat het Blynk IoT-platform?

Blynk [1] is een low-code IoT-softwareplatform met cloud, firmware-bibliotheken, een no-code native mobiele app-builder en een webconsole voor beheer. U krijgt functionaliteit zoals WiFi-apparaatprovisieoning, datavisualisatie, automatiseringen, meldingen, OTA-updates en een robuust gebruikers- en apparaat-beheersysteem direct uit de doos [1].



Figuur 1. No-Code interfaces gemaakt met Blynk.



Figuur 2. Blynk app-builder: slepen&neerzetten.

Geavanceerd gebruikersbeheersysteem

Dit helpt alles gestructureerd te houden, zelfs op ondernemingsschaal. U kunt een organisatiestructuur met meerdere niveaus creëren en apparaten en gebruikersrollen, -rechten, -wachtwoorden en nog veel meer beheren.

Ingebouwd device life cycle-beheer

De functionaliteit voldoet aan alle behoeften met betrekking tot tokenbeheer, WiFi-provisioning met dynamische tokengeneratie, het toevoegen van apparaten en het toewijzen ervan aan gebruikers. Het biedt betrouwbare en veilige OTA-firmware-updates die worden beheerd in een eenvoudige interface.

Automatiseringsscenario's zonder code

Kan worden ingesteld op basis van datum, tijd van de dag, gebruikersacties of apparaatstatus. U kunt gebruikers via pushmeldingen, in-apps, e-mails of sms op de hoogte brengen van belangrijke hardwaregebeurtenissen.

Uw ESP met Blynk-verbinden - hoe moeilijk is de integratie?

Afhankelijk van uw hardware-setup kunt u twee wegen volgen. U kunt kiezen voor *Blynk.Edgent* [3] bij een

single-MCU-configuratie of *Blynk.NCP* [4][5] bij een dual-MCU-configuratie. Beide mogelijkheden bieden alle ingebouwde Blynk IoT-functies en een beveiligde verbinding met Blynk Cloud.

Ze vereisen minimale inspanning, met door Blynk geleverde codevoorbeelden. Bij de dual-MCU-setup gebruikt u een kant-en-klaar binair bestand voor de NCP en een lichtgewicht bibliotheek voor de primaire MCU die communiceert met de NCP via de UART-interface.

De weg van apparaatconfiguratie naar volledige IoT-infrastructuur hoeft niet meer dan enkele weken te duren [6].

230659-03

30% korting op het Blynk PRO-plan tijdens het eerste jaar!

Promotiecode: **ELEKTOR**

Geldig tot 31 januari 2024 [2]

WEBLINKS

[1] Officiële website: <https://bit.ly/blynk-io>

[2] Blynk.Console - maak uw gratis account aan: <https://bit.ly/web-cloud>

[3] Blynk.Edgent documentatie: <https://bit.ly/docs-edgent>

[4] Blynk.NCP documentatie: <https://bit.ly/docs-ncp>

[5] Wat is Blynk.NCP: <https://bit.ly/info-ncp>

[6] Kant-en-klaar weerstationproject om mee te spelen: <https://bit.ly/blueprint-weather>

Bouw een slimme gebruikersinterface op ESP32

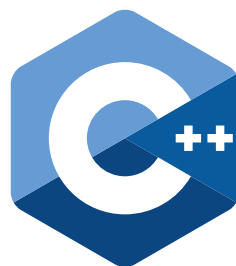
Een bijdrage van Slint

Smartphones hebben de gebruikerservaring van aanraakbediening opnieuw gedefinieerd.

Het ontwikkelen van een moderne slimme gebruikersinterface vereist het gebruik van moderne grafische bibliotheken en tools.

In dit artikel delen we tips en bespreken we Slint, een toolkit om interactieve gebruikersinterfaces te maken die de verwachtingen van de gebruiker overtreffen.

Slint is een gloednieuwe toolkit voor het bouwen van native grafische interfaces in C++, Rust en JavaScript, met uitgebreide cross-platform ondersteuning, inclusief bare metal, RTOS-systemen en embedded Linux. Op GitHub heeft Slint meer dan 10.000 sterren verzameld..

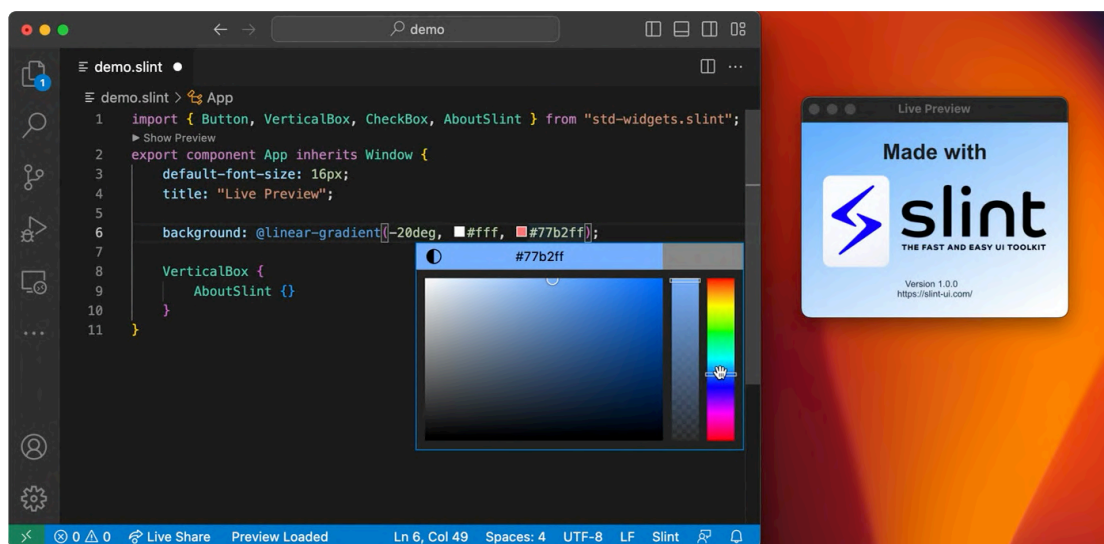


Figuur 1. De logo's van C++ en Rust.

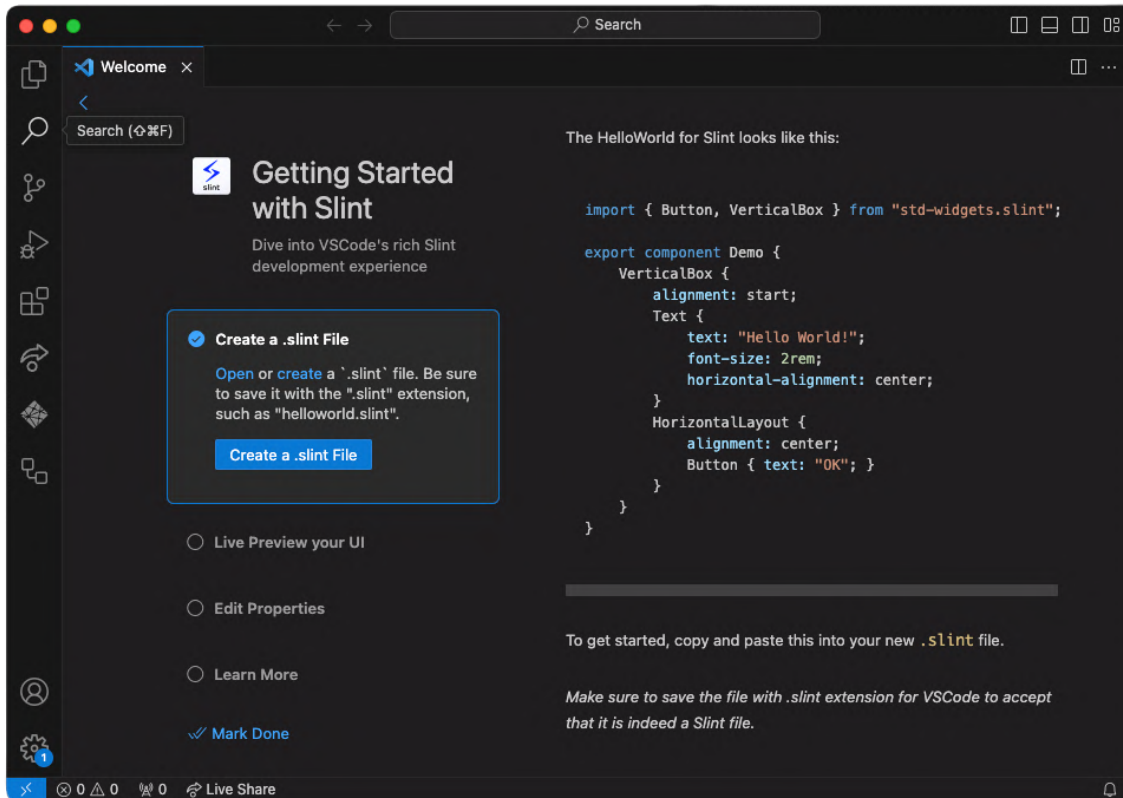
Kies een programmeertaal - C/C++ of Rust

Bij embedded programmeren was C/C++ lange tijd de favoriete programmeertaal. Maar Rust, bekend om zijn geheugenveiligheid en prestaties, wordt steeds populairder onder embedded ontwikkelaars.

Slint is de enige toolkit die native API's voor zowel C++ als Rust ondersteunt (figuur 1). Dit geeft ontwikkelaars keuzevrijheid om de logica van het bedrijf in een van beide talen te schrijven. Bovendien biedt het een praktische oplossing voor iedereen die de overstap wil maken van hun code in C/C++ naar Rust.



Figuur 2. Snel itereren met de live preview van Slint.



Figuur 3. Aan de slag met Slint.

Scheid de gebruikersinterface van de bedrijfslogica

Gebruikelijke ontwerppatronen zoals MVC en MVVM bevorderen de scheiding van de bedrijfslogica van de gebruikersinterface om de efficiëntie en codekwaliteit te verbeteren.

In Slint wordt de gebruikersinterface gedefinieerd met een aan HTML/CSS verwante taal, wat een strikte scheiding tussen presentatie en bedrijfslogica bevordert. Door middel van Live Preview is het in Slint mogelijk met snelle iteraties de gebruikersinterface te voltooien (figuur 2).

Geniet van een goede ontwikkelervaring (DX)

De complexiteit in de softwareontwikkeling vereist een goede DX: ontwikkelaars bouwen met vertrouwen, realiseren grotere impact en hebben meer voldoening van hun werk als ze werken met goede ontwikkelprogramma's.

U kunt met uw favoriete IDE blijven werken. Kies tussen Slints VS code-extensie (figuur 3) en de generieke taalserver: profiteer van code completion, syntaxmarkering, diagnostiek, live preview en meer. Daarnaast biedt Slint een ESP-IDF-component, wat de integratie ervan met het Espressif IoT Development Framework (IDF) vereenvoudigt.

Lever een uitzonderlijke gebruikerservaring

UI-prestaties zijn cruciaal voor een uitzonderlijke gebruikerservaring. Slint biedt flexibiliteit in hardware-ontwerp door de mogelijkheden voor line-by-line en framebuffer-rendering op het ESP32-platform, wat zorgt voor een veelzijdiger benadering van apparaatontwikkeling (figuur 4).

Als u met Slint op de ESP32 wilt beginnen, bezoek dan de website van Slint [1].

230670-03



Figuur 4. Een Slint-demo op ESP32.

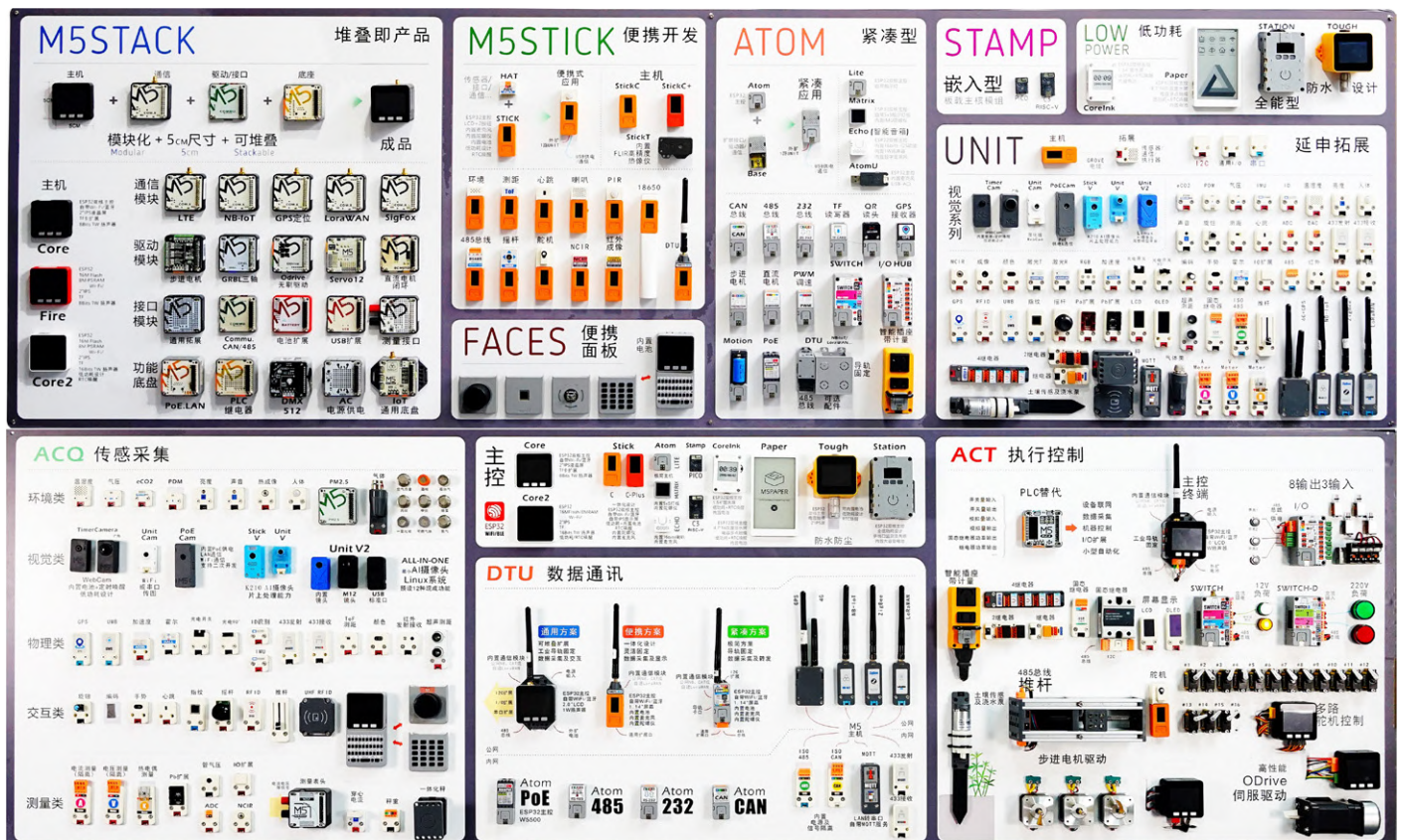
WEBLINK

[1] Slint op de ESP32: <https://slint.dev/esp32>

Snelle en gemakkelijke IoT-ontwikkeling met M5Stack

Een bijdrage van M5Stack

Als gerenommeerd modulair IoT-ontwikkelplatform op basis van de ESP32 bouwt M5STACK honderden controllers, sensoren, actuatoren en communicatiemodules in modulaire vorm, die via standaard-interfaces kunnen worden aangesloten. Door modules met verschillende functionaliteiten te stapelen kunnen gebruikers productverificatie en -ontwikkeling versnellen.



Figuur 1. De M5Stack ecosysteem-familie.



Figuur 2. M5Dial is geschikt voor het Smart Home.



*Als je een liefhebber van
de ESP32 bent, dan is
M5Stack een must-have!*

De M5Stack-modules (**figuur 1**) [1] kunnen worden aangesloten en gebruikt met de UIFlow low-code grafische programmeer-IDE, wat zorgt voor de beste ervaring bij het prototypen van IoT-projecten, zowel voor beginnende hobbyisten als voor professionele ontwikkelaars.

Met 'stapelbare' hardware-modules en een gebruiksvriendelijk grafisch programmeerplatform biedt M5Stack zijn klanten in Industrial IoT, Home Automation, Smart Retail, Smart Agriculture en STEM-onderwijs een efficiënte en betrouwbare Quick&Easy IoT-ontwikkelervaring.

Nieuw: de M5Dial

De onlangs gelanceerde M5Dial [2] is een product dat bij uitstek geschikt is voor het Smart Home. Als veelzijdig embedded development board integreert de M5Dial verschillende functionaliteiten en sensoren die nodig zijn voor Smart Home-besturing (**figuur 2**).

De hoofdcontroller van de M5Dial is de M5StampS3, een microcontroller gebaseerd op de ESP32-S3 chip die bekend staat om zijn hoge prestaties en laag stroomverbruik. Hij ondersteunt WiFi- en Bluetooth communicatie, evenals meerdere interfaces zoals SPI, I²C, UART, ADC en meer. De M5StampS3 wordt ook geleverd met 8 MB on-board flash, wat de gebruikers voldoende geheugenruimte biedt. De M5Dial is voorzien van een 1,28 inch rond TFT touch-display, een draai-encoder, een RFID-detectiemodule, een RTC-schakeling, een zoemer, fysieke knoppen, en andere functionaliteiten waarmee gebruikers eenvoudig verschillende projecten kunnen implementeren.

Het opvallendste kenmerk van de M5Dial is de draai-encoder die nauwkeurig de positie en draairichting van de knop registreert, waardoor gebruikers een verbeterde interactieve ervaring krijgen. Gebruikers kunnen instellingen zoals volume, helderheid en menu's aanpassen, of huishoudelijke apparaten zoals verlichting, airconditioning, gordijnen enzovoort bedienen met behulp van deze draaiknop. Het ingebouwde display van het apparaat kan ook verschillende interactieve kleuren en effecten weergeven. Met zijn compacte formaat van 45 × 45 × 32,2 mm en geringe gewicht van 46,6 g is de M5Dial eenvoudig te implementeren. Of hij nu wordt gebruikt om huishoudelijke apparaten in Smart Home te bedienen of om systemen in de industriële automatisering te bewaken en te besturen, de M5Dial kan eenvoudig worden geïntegreerd en biedt dan dan slimme bediening en interactieve functionaliteit. ◀

230662-03

WEBLINKS

[1] The Innovator of Modular IoT Development Platform | M5Stack: <https://m5stack.com>

[2] ESP32-S3 Smart Rotary Knob w/ 1.28" Round Touch Screen: <https://shop.m5stack.com/products/m5stack-dial-esp32-s3-smart-rotary-knob-w-1-28-round-touch-screen>

Prototype van een ESP32-gebaseerde energiemeter



Figuur 1. Zo zou onze enkelfasige energiemeter eruit kunnen zien.

Saad Imtiaz (Elektor)

Dit artikel beschrijft de ontwikkeling van een energiemeter met behulp van een Espressif ESP32, waarbij de nadruk ligt op het observeren van het energieverbruik in real-time en op de veiligheid. Het belicht de eerste stappen, voorwaarden en overwegingen die plaatsvinden wanneer je aan een embedded project begint.

Naarmate het project vordert, zullen de resultaten worden gedeeld in de komende uitgaven van Elektor.

Op het gebied van techniek kan het combineren van de juiste technologieën tot aanzienlijke vooruitgang leiden. Dit project heeft als doel een energiemeter te ontwikkelen met behulp van de Espressif ESP32 microcontroller en Microchip's energiemeter-IC ATM90E32AS. In dit artikel wordt het beginstadium van dit project kort beschreven, van componentkeuze tot prototyping. Het doel is eenvoudig: het maken van een betrouwbaar systeem voor nauwkeurige energiemeting vanuit de groepenkast van je huis of werkplaats. Met deze meter kunnen gebruikers het stroomverbruik in realtime volgen en een inzicht krijgen dat kan leiden tot een efficiënter energiegebruik.

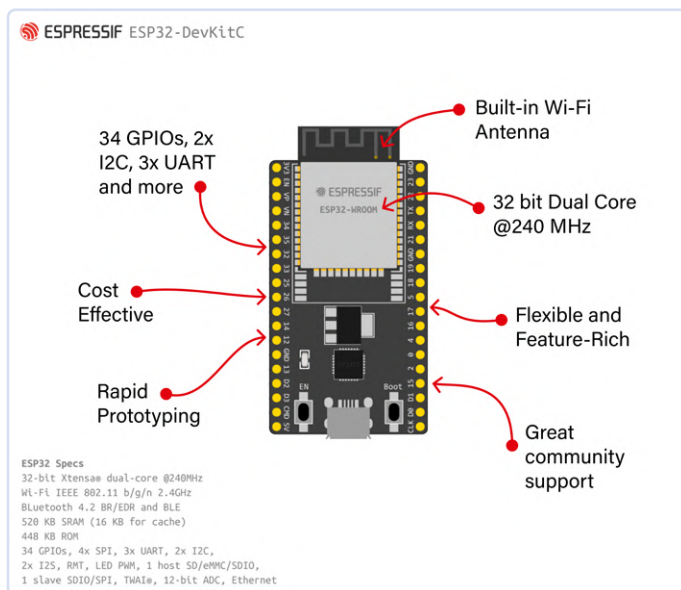
Ontwerp en voorwaarden

Het project heeft duidelijke doelen en ontwerpvoorwaarden: real-time monitoren van enkelfasig vermogen met behulp van drie stroomtransformatoren (CT's), het betaalbaar houden en het gebruiksvriendelijk maken. De keuze voor de ESP32 en ATM90E32AS IC-componenten werd bepaald door deze doelstellingen, die zowel een goede prijs-kwaliteitverhouding als betrouwbare prestaties bieden. Een ander doel was om de afmetingen kleiner dan 100×80×30 mm (L×B×H) te houden, zodat het in een groepenkast past. Om de gebruikerservaring te verbeteren, is er ook een mobiele interface voor monitoring op afstand en een OLED-scherm met knoppen voor directe bediening. Het ontwerp maakt ook toekomstige software-updates mogelijk, zodat de consument verzekerd is van langdurig gebruik. In **figuur 1** is een gerenderde weergave van het huidige prototype van de behuizing te zien.

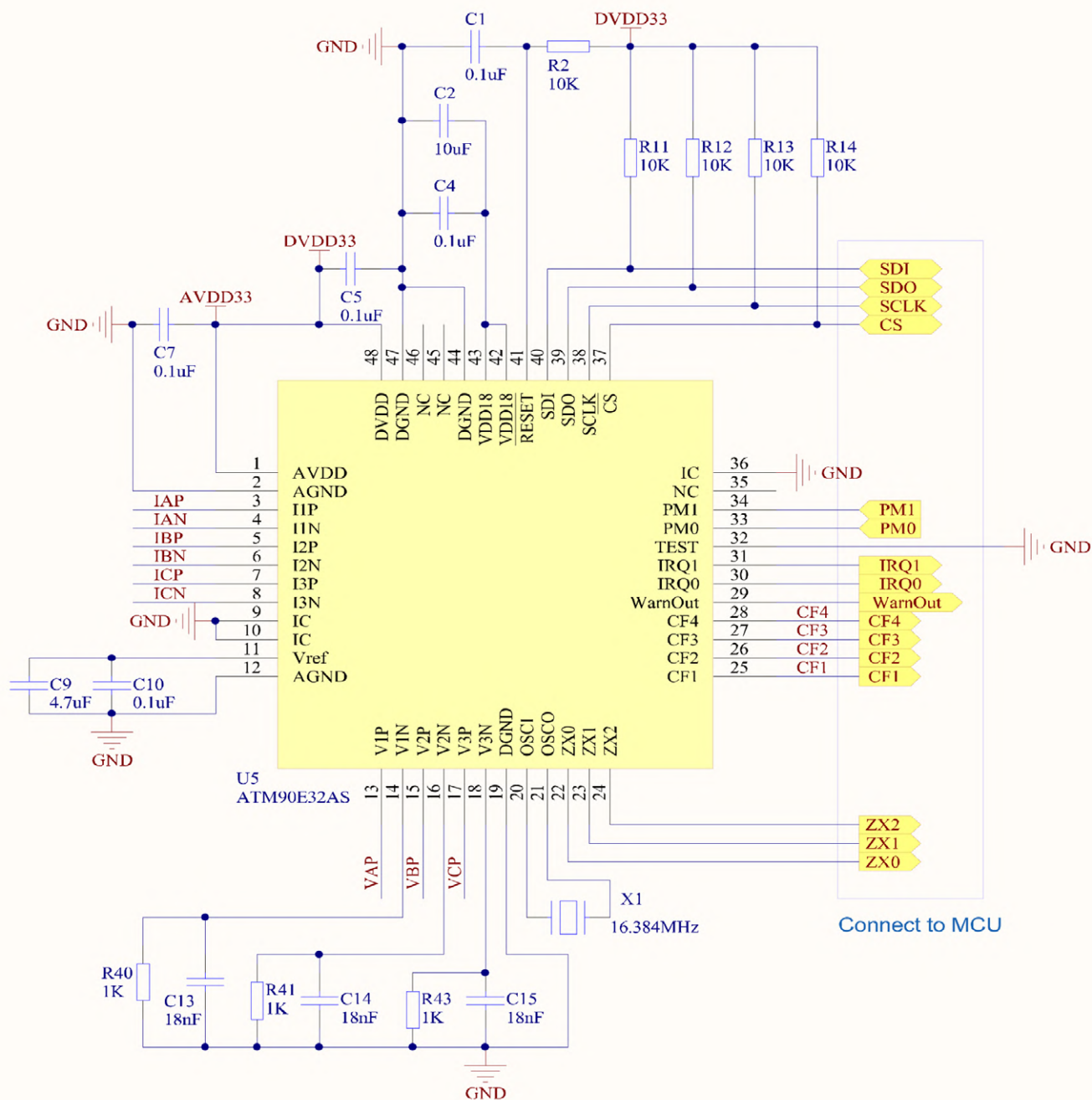
Keuze microcontroller

De keuze voor de ESP32-microcontroller was gebaseerd op een gedetailleerde evaluatie van zijn functies. De chip blinkt uit op verschillende gebieden die cruciaal zijn voor het succes van dit project. Ten

eerste het gemak waarmee hij in verschillende ontwerpen kan worden geïntegreerd; dat biedt flexibiliteit tijdens de ontwikkelingsfase. Ten tweede maakt de kosteneffectiviteit hem een aantrekkelijke keuze voor een prototype dat prestaties en kosten in evenwicht brengt. Ten derde biedt de compatibiliteit met een breed scala aan sensoren en IC's aanzienlijke voordelen. Tot slot vergroot de uitgebreide ondersteuning van de ESP32 chip door de community zijn geschiktheid voor dit project. **Figuur 2** laat de belangrijkste kenmerken en voordelen van de ESP32-D0WD-V3 zien, waardoor deze voor dit project is geselecteerd.



Figuur 2. Belangrijkste kenmerken en voordelen van de ESP32.



Figuur 3. De energiemeter is gebaseerd op een application note van Atmel [2]. Hier zie je de schakelingen rond het energiemeter-IC.

Integratie van het energiemeter-IC

Het ATM90E32AS-IC van Microchip werd geïntegreerd volgens een application note van de fabrikant; het document diende als leidraad om ervoor te zorgen dat het energiemeter-IC naadloos communiceerde met de ESP32-microcontroller. Deze fase was echter niet helemaal vrij van hindernissen. De aanschaf van de juiste componenten binnen de budgetbeperkingen vereiste een nauwgezette planning, omdat de beschikbaarheid beperkt was. In **figuur 3** wordt de application note van Atmel (nu Microchip) getoond.

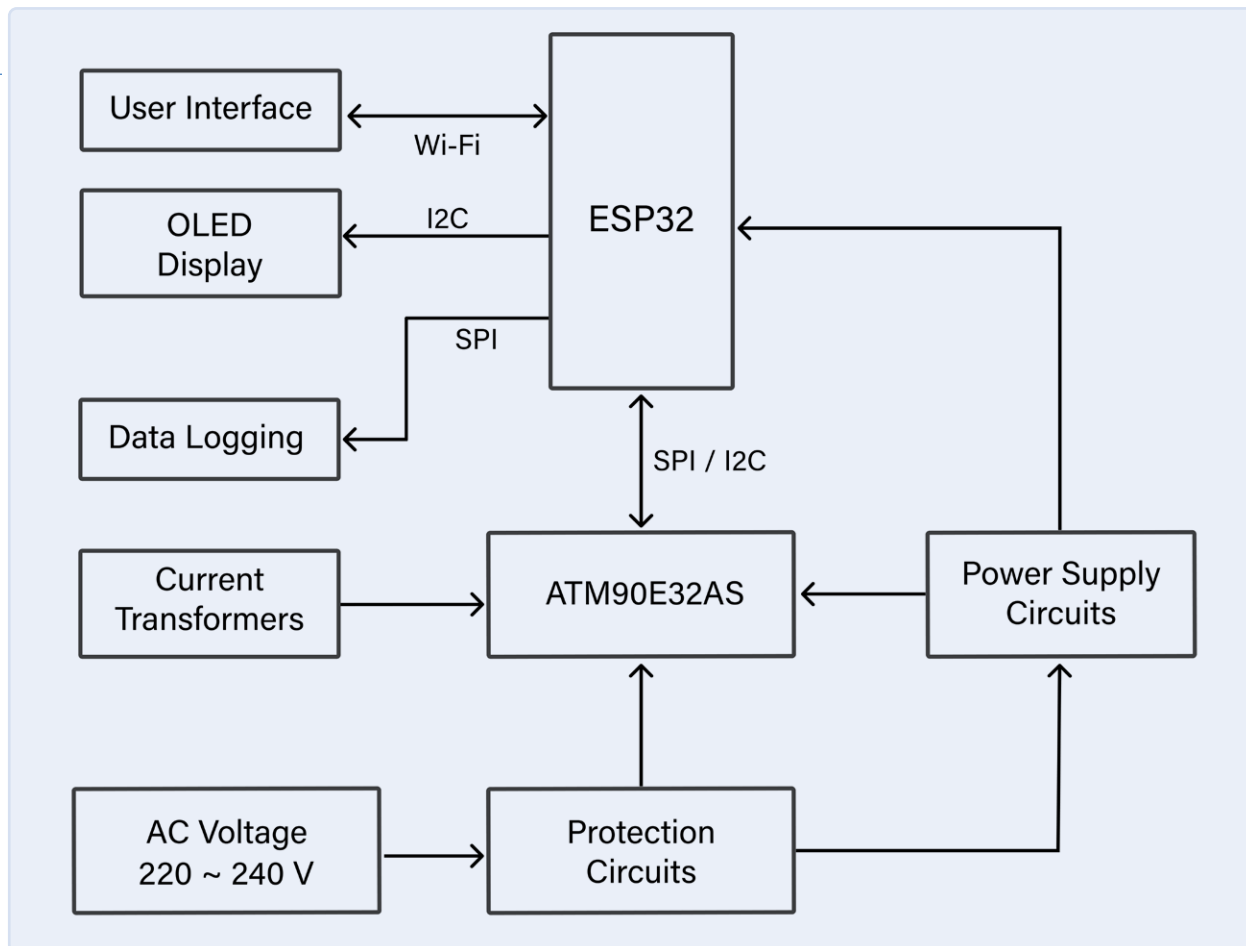
Ontwerpfase en elektrische veiligheidsnormen

De ontwerpfase is inderdaad een cruciaal onderdeel van het ontwerp-proces, vooral wanneer veiligheid een belangrijk aandachtspunt is. Bij een apparaat dat ontworpen is om te werken met de lichtnet-wissel-

spanning moet er nauwgezet op worden toegezien dat het voldoet aan de vastgestelde veiligheidsnormen. In **figuur 4** wordt het blokschema van het project getoond.

Om de veiligheid te garanderen werden verschillende speciale elektrische componenten in het ontwerp opgenomen. Metaaloxide-varistoren (MOV's) werden gebruikt om het circuit te beschermen tegen kortstondige spanningspieken. Verder werden er zekeringen ingebouwd als essentiële beveiliging tegen te hoge stromen.

Behalve op de keuze van componenten richtte het ontwerp zich ook op printlayout-overwegingen die zouden voldoen aan de veiligheidsnormen. Er werden voldoende kruip- en onderlinge afstanden aangehouden tussen de geleidende elementen op de printplaat om overslag te voorkomen. De spoorbreedte voor AC-spanningslijnen werd zorgvuldig berekend om de verwachte stromen aan te kunnen, volgens de



Figuur 4. Blokschema van ons energiemeterproject.

IPC-2221 normen [1]. Dit was cruciaal om de thermische prestaties van de print onder maximale belasting te kunnen waarborgen. Om de integriteit van de aarding te garanderen, werd een degelijk aardvlak gebruikt. Er werd speciale aandacht besteed aan het ontwerp van differentiële paren voor een goede signaalintegriteit, waarbij ervoor werd gezorgd dat de routing een nauwkeurige geometrie volgde om elektromagnetische interferentie tot een minimum te beperken.

Fabricage: we kozen voor JLC PCB

Na het bestuderen van verschillende bedrijven voor PCB-assemblage werd JLC PCB geselecteerd. De belangrijkste reden voor deze keuze was de balans tussen kosteneffectiviteit en betrouwbaarheid die zij bieden. Deze beslissing was belangrijk om het project binnen het budget te houden zonder afbreuk te doen aan de kwaliteit van de geassembleerde print. Op dit moment wordt de laatste hand gelegd aan het prototype-schema en de printlayout, en deze zullen binnenkort worden aangeboden voor productie.

Terugblik en vooruitzicht

Achteraf gezien toont dit project wat er bereikt kan worden als zorgvuldige planning en goede techniek hand in hand gaan. De hindernissen die we tegenkwamen hebben ons geholpen ons ontwerp te verbeteren. Nu we overgaan van het maken van een prototype naar de mogelijke massaproductie ervan, verwachten we dat het echt een verschil zal maken in de manier waarop mensen omgaan met energie. Dit project zal in de komende edities van dit tijdschrift uitgebreid aan bod komen – we zijn nog steeds bezig met het prototype, dit te testen en de software te ontwikkelen waarmee het zal werken. Er komt nog meer, dus blijf op de hoogte van updates over dit project. We zullen de voltooiing en

laatste updates delen in het januari/februari 2024 nummer van Elektor, met de focus op voeding en energie.

vertaling: Hans Adams — 230646-03

Vragen of opmerkingen?

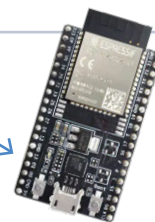
Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de redactie van Elektor via redactie@elektor.com.



Gerelateerde producten

> **ESP32-DevKitC-32E** www.elektor.nl/20518

> **ESP32-C3-DevKitM-1** www.elektor.nl/20324



WEBLINKS

- [1] IPC-2221A standaarden:
[https://www-eng.lbl.gov/~shuman/NEXT/CURRENT_DESIGN/TP/MATERIALS/IPC-2221A\(L\).pdf](https://www-eng.lbl.gov/~shuman/NEXT/CURRENT_DESIGN/TP/MATERIALS/IPC-2221A(L).pdf)
- [2] ATM90E32AS application note:
<https://ww1.microchip.com/downloads/en/Appnotes/Atmel-46103-SE-M90E32AS-ApplicationNote.pdf>

Een distributeur met meerwaarde voor IoT en meer

Source: Adobe Stock

Een bijdrage van Stelium Technology

Stelium Technology is een innovatief bedrijf dat gespecialiseerd is in elektronische oplossingen. Het bedrijf onderscheidt zich door zijn technische expertise en passie voor innovatie. Stelium Technology levert van zijn partner Espressif essentiële elektronica-componenten voor draadloze connectiviteit en het IoT, zoals de ESP32-C5, ESP32-C6, ESP32-P4, ESP32-S6 en nog talloze andere producten.

Stelium Technology [1], opgericht in 2018, omschrijft zichzelf als een distributeur met meerwaarde van elektronische oplossingen. Mens/machine-interfaces, display- en aanraakoplossingen, connectiviteit en IoT, allemaal expertisegebieden die Stelium goed beheerst en waarmee het bedrijf naam heeft gemaakt op de elektronicamarkt.

Stelium Technology staat bekend om zijn strategische partnerschappen met toonaangevende elektronicabedrijven, waardoor het zijn positie op het gebied van IoT en connectiviteit kan versterken. Stelium Technology is al lange tijd een belangrijke distributeur van Espressif, een partnerschap dat in de loop der jaren is geconsolideerd. Als officiële distributeur voor Frankrijk en Italië is Stelium Technology de one-stop-shop voor Espressif-oplossingen.

Stelium Technology is perfect uitgerust om volledige ondersteuning te bieden voor het gehele assortiment Espressif-producten, zowel op het gebied van hardware als embedded software. Het team beschikt over grondige technische expertise met betrekking tot alle aspecten van connecti-

viteit, van hardwareontwerp tot software-programmering. Dat betekent dat Stelium Technology klanten volledige ondersteuning kan bieden, zodat ze verzekerd kunnen zijn van robuuste en hoogwaardige connectiviteitsoplossingen.

Dit sterke partnerschap garandeert onze klanten bevoorrechte toegang tot de beste connectiviteitsoplossingen op de markt, zoals de nieuwste Espressif-generaties: ESP32-C5, ESP32-C6, ESP32-P4, ESP32-S6. Stelium Technology levert via zijn partner Espressif essentiële elektronische componenten voor draadloze connectiviteit en IoT in diverse markten en sectoren en draagt zo bij aan de constante evolutie van de technologie.

IoT-oplossingen en meer

In de eerste plaats worden de WiFi- en Bluetooth-microcontrollers van Espressif veel gebruikt op het gebied van het Internet of Things (IoT). Deze componenten zijn van vitaal belang voor smart home-toepassingen zoals connected thermostaten, beveiligingscamera's en verlichtingregelaars. Ze



spelen een leidende rol bij interoperabiliteit in connected home-producten met compatibele Matter-oplossingen (via WiFi in het bijzonder). De oplossingen van Espressif worden ook gebruikt in de industriële sector voor bewaking op afstand, gegevensverzameling en machinebesturing. Daarnaast zijn de producten van Espressif aanwezig in de gezondheidszorg, waar ze het hart vormen van connected medische apparaten en fitness-trackers.

Als wereldwijde elektronicapartner kan Stelium oplossingen bieden waarin de producten van Espressif voor touchscreen-bediening zijn geïntegreerd. Dankzij zijn expertise is Stelium Technology in staat om geïntegreerde oplossingen te ontwerpen waarin de producten van Espressif voor de aanraakfunctionaliteit zorgen, en het bedrijf kan bogen op een aantal succesverhalen, met name voor displayformaten tot 7 inch. Ons vermogen om een wereldwijde oplossing te bieden wordt versterkt door specifieke technische ondersteuning op al deze gebieden. ◀

230661-03

Vragen?

Stelium staat klaar om klanten te helpen met eventuele vragen. Neem contact op met remi.krief@stelium-technology.com voor vragen over Espressif-oplossingen.

WEBLINK

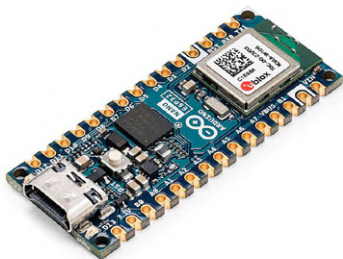
[1] Stelium Technology: <https://stelium-technology.com/en>

Achter de schermen: vraaggesprek met Arduino over de Nano ESP32

Alessandro Ranellucci en Martino Facchin over de samenwerking met Espressif

Vragen van Saad Imtiaz en Clemens Valens (Elektor)

In de zomer van 2023 lanceerde Arduino de Nano ESP32. Gebaseerd op de ESP32-S3 van Espressif, heeft het nieuwe board 2,4 GHz, 802.11 b/g/n WiFi- en Bluetooth 5 (LE) connectiviteit met groot bereik in een Nano-vormfactor. De Nano ESP32 is niet het eerste Arduino-board met een processor van Espressif, maar deze keer is het de hoofdrolspeler in plaats van slechts een draadloze communicatiemodule die een andere MCU ondersteunt. Elektor vroeg Alessandro Ranellucci (Head, Arduino Makers Division) en Martino Facchin (Hardware & Firmware Manager, Arduino) naar deze samenwerking tussen Espressif en Arduino.



Elektor: Kun je uitleggen waarom je de ESP32 hebt gekozen voor de nieuwe Arduino Nano-serie in plaats van een andere microcontroller?

Alessandro Ranellucci: De Arduino Nano-serie vertegenwoordigt een groep boards die is ontworpen om consistentie te bieden in termen van vorm, pinning en technische kernfuncties. Deze consistentie is iets wat onze gemeenschap van makers waardeert omdat het naadloze uitwisselbaarheid binnen de Nano-serie mogelijk maakt. We hadden echter nog geen board op de markt gebracht op basis van de populaire ESP32-architectuur. We vonden het nodig om binnen de Nano-serie een krachtige optie te bieden die gebruikmaakt van de ESP32, vandaar deze beslissing.

Martino Facchin: Ja, in de afgelopen vier jaar hebben we de Nano-serie uitgebreid met microcontrollers van verschillende chipfabrikanten waar we nog niet eerder mee hadden gewerkt. We begonnen met het integreren van microcontrollers van Nordic, gevolgd door Raspberry Pi. In wezen is het onze proeftuin om zowel nieuwe als gevestigde architecturen uit te proberen, terwijl we unieke waarde toevoegen die elders niet beschikbaar is. De kale ESP32-chip had bijvoorbeeld geen USB. In de C3 is er wel USB, maar die heeft een vast vendor- en product-ID, zodat je niet echt onderscheid kunt maken tussen individuele boards. De S3 is de eerste die deze mogelijkheid biedt, dus die hebben we gebruikt in plaats van iets anders.

Elektor: Welke unieke technische voordelen heeft de ESP32 ten opzichte van andere opties?

Martino Facchin: Ja, inderdaad, we hadden de mogelijkheid om rechtstreeks samen te werken met u-blox om een gespecialiseerde chip te verwerven die niet beschikbaar is voor algemene aankoop, met PSRAM geïntegreerd in de chip zelf. We bieden een chip met PSRAM en de grootst mogelijke hoeveelheid extern flashgeheugen. Deze details zijn de technische diepgang, zou je kunnen zeggen, op maat gemaakt voor de liefhebbers en *geeks*. De functies van het board stuwden de ESP32 naar het hoogste potentieel dat momenteel beschikbaar is. Dan bevat het natuurlijk WiFi, BLE, dual-core processing en alle andere functionaliteiten die bij de ESP32 horen. We hebben ervoor gekozen om geen extra functies toe te voegen aan de Arduino Nano ESP32 zoals we met andere board hebben gedaan. Dit board is ontworpen als bouwsteen. In tegenstelling tot



Martino Facchin



Alessandro Ranellucci

de Arduino BLE is het geen apparaat dat je direct uit de doos kunt gebruiken voor substantiële toepassingen. Je moet extra componenten, zoals sensoren of modules, aan het bord koppelen. Het is uitgerust met een RGB-LED, maar verder heeft de maker de vrijheid om het uit te breiden met andere componenten en het te gebruiken als een goed gekalibreerde basis voor eigen projecten.

Elektor: Hebben jullie problemen ondervonden met het inpassen van de ESP32-chip in de Arduino Nano-vormfactor en zo ja, hoe hebben jullie deze problemen aangepakt?

Alessandro Ranellucci: De eerste uitdaging bestond inderdaad uit het aangaan van een partnerschap met Espressif. Ze maken al jaren een uitstekende Arduino-core voor de ESP32-boards die diep in het Arduino-framework is ingebed, maar die ook aan hun eigen technische voorkeuren voldoet. De uitdaging was om uit te vinden hoe we onze inspanningen konden combineren om een meer samenhangende en verbeterde gebruikerservaring te creëren. Daarom hebben we deze samenwerking opgezet. Ik denk dat Martino ook andere uitdagingen kan noemen, zoals bijvoorbeeld de pinnummering.

Martino Facchin: Zeker niet veel, hardwarematig waren er geen problemen. Dat is natuurlijk makkelijk in vergelijking met andere boards die we de afgelopen jaren hebben gemaakt. Softwarematig was dat een heel anders, omdat de pinnummering erg gebonden was aan de ESP-wereld. Ze hebben de pinnen precies zo genummerd als op de CPU-socket. Aan de onderkant van het board zie je misschien labels van andere fabrikanten met nummers als 31, daarnaast een 4, en dan een 55, enzovoort. Dat werkt niet voor de Nano – niet de oplossing die we zochten. Dus ontwikkelden we een manier om logische pinnummers te vertalen naar interne pinnum-

mers. We hebben dit laten goedkeuren in de ESP32 core-community. Op dit moment kan elk board met een ESP32, van elke andere fabrikant, dezelfde logica gebruiken als ze onze filosofie willen overnemen, en dit is een directe bijdrage aan één core die we niet onderhouden. Het was moeilijk omdat de releasetijd van deze functie perfect moest overeenkomen met de releasetijd van het board. Maar uiteindelijk is het gelukt.

Het was een uitdaging omdat het de eerste keer was dat we iets uitbrachten dat niet volledig door ons werd gecontroleerd, en dat was moeilijk, maar we hebben het gedaan.

Alessandro Ranellucci: We hebben belangrijke inzichten verkregen uit het hele iteratieproces omdat, zoals Martino al zei, elke fabrikant er nu voor kan kiezen om logische pinnen te gebruiken. Het is gebruiksvriendelijker om pinnen opeenvolgend te nummeren dan om de pinnummering van de controller te gebruiken, zoals PA1 en PD1.

Elektor: Heeft de ontwikkeling van de ESP32-software, het ecosysteem en de ondersteuning van de community invloed gehad op de beslissing om het in de nieuwe Arduino Nano te gebruiken?

Martino Facchin: Nee. We zouden ervoor gekozen hebben om het te gebruiken, zelfs als er geen steun van de community was. Zeker nu de community veel werk verzet en onze projecten bloeien, hebben we veel profijt van hun bijdragen, vooral voor de bibliotheekmanager. Sommige bibliotheken waren al compatibel met ons board, wat voordelig was. Andere waren exclusief voor de ESP32, wat een nadeel was, maar die kunnen we nu ook gebruiken. Deze compatibiliteit was een factor in onze beslissing, maar we zouden hoe dan ook verder zijn gegaan met de ESP32.



Pinnummering was een uitdaging omdat de nummers nauw waren gekoppeld aan de ESP32-wereld. Maar we vonden een oplossing – logische pinnummers.

Alessandro Ranellucci: Wat de software betreft, hadden we de optie om een nieuwe core te maken, net zoals we bij andere producten hebben gedaan, zoals de RP2040 waar we onze eigen core hebben ontwikkeld om volledige controle over de software te hebben. Maar Espressif heeft door de jaren heen uitstekend werk geleverd en ze hebben een robuuste community. Daarom hebben we ervoor gekozen om met hen samen te werken – een beslissing die werd beïnvloed door de kracht van het ecosysteem.

We streven ernaar om neutraal te blijven met betrekking tot technologie, en exclusief gebruik van slechts één lijn van microprocessoren van één fabrikant te vermijden. Ons doel is om een veelzijdig en interoperabel Arduino-platform aan te bieden, want dat is hoe gebruikers Arduino zien en wat ze verwachten. Daarom experimenteren en onderzoeken we voortdurend nieuwe producten om te ontwikkelen.

Martino Facchin: Mensen die overstappen naar Arduino vanuit andere achtergronden, zoals BSP of geïntegreerde omgevingen die tijdrovende installatie- en configuratieprocessen met zich meebrengen, zeggen vaak dat Arduino 'gewoon werkt'. Dit is, denk ik, onze grootste waarde – dat een gebruiker kan beginnen Arduino te gebruiken op de eenvoudigste en snelst denkbare manier. En zoals Alessandro al zei, we zijn agnostisch. Zes jaar geleden kon ik dit niet beweren omdat Atmel in wezen onze hoofdsponsor was, maar nu zijn we werkelijk onpartijdig.

Elektor: Kun je iets vertellen over verbeteringen of veranderingen die je hebt aangebracht in het ESP32-platform om het beter te laten aansluiten op de doelstellingen van de Arduino Nano ESP32?

Martino Facchin: We hebben het uploadproces dat normaal gesproken wordt gebruikt voor ESP32 gewijzigd, waarbij gebruikers van oudsher moesten overschakelen naar de native USB-module om de ESP-tool te bedienen – een proces dat niet naadloos is geïntegreerd met onze IDE. Nu wordt bij het uploaden van een sketch een off-the-shelf (OTS) methode gebruikt. De sketch wordt overgebracht via Device Firmware Update (DFU) en naar de tweede partitie gestuurd via een over-the-air (OTA) update. De bootloader krijgt dan de opdracht om te proberen opnieuw op te starten vanaf deze tweede partitie. Als dat lukt, wordt de nieuwe sketch geladen; zo niet, dan blijft de originele sketch behouden. Deze implementatie is een aanzienlijke verbetering, omdat Espressif al rekening had gehouden met verschillende gebruikssituaties. We hebben hun aanpak aangepast om een sneller, betrouwbaarder systeem met een herstelmodus te ontwikkelen.

We hebben ondersteuning voor dubbeltikken (je kunt de herstelmodus starten door twee keer op de resetknop te drukken), wat niet mogelijk was op het ESP-board. Dit zijn enkele aanpassingen die

allemaal goed werden geaccepteerd door de community vanwege de inspanning. Ook al wordt de community core ondersteund door Espressif, het is een inspanning van de community, dus iedereen was hier blij mee.

Alessandro Ranellucci: Er zijn nog twee bijdragen die we hebben geleverd aan het ecosysteem in het algemeen, dus niet specifiek aan de core. Een deel van het werk dat we hebben gedaan was debuggen, en we hebben ook MicroPython-ondersteuning voor ESP32 toegevoegd.

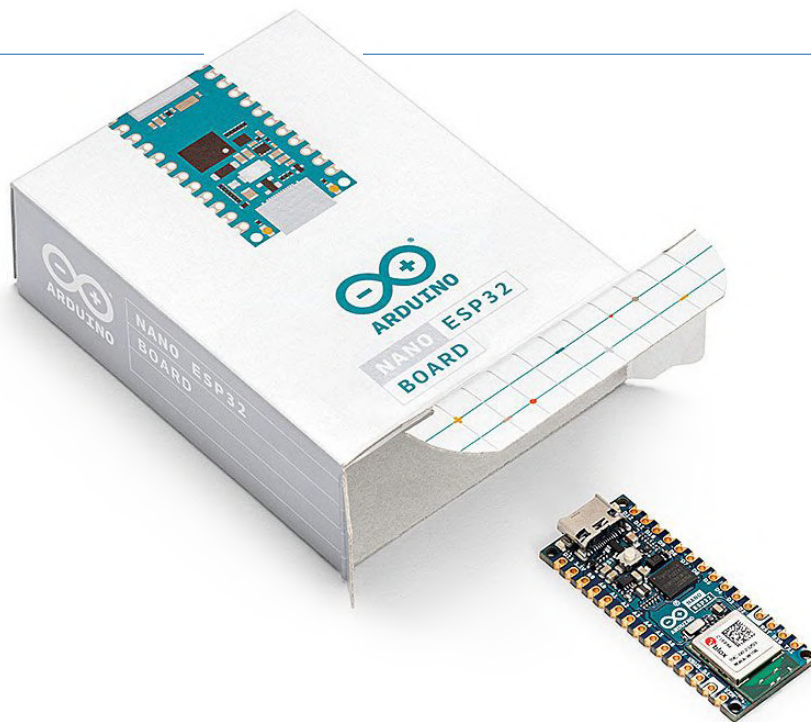
Elektor: Wat betreft debuggen, biedt dit board enige toegang is tot de ESP via soldeereilandjes, of is het een ander board?

Martino Facchin: Nee, je kunt dit bord direct debuggen via USB – Espressif was zo vriendelijk om hiervoor te zorgen. Via USB heb je een seriële poort, CDC, de gebruikelijke dingen en ook een JTAG-interface. Je kunt communiceren met de JTAG-interface door gewoon de normale USB te gebruiken naast de normale communicatie. Het kan geen kwaad om een externe debugger aan te sluiten, maar dat hoeft niet. In feite is het al geïntegreerd in de IDE. Je moet gewoon het board aansluiten, 'Debug mode (Hardware CDC)' selecteren in het menu en op de grote Debug-knop drukken. Onder de motorkap zal er een heleboel gebeuren met OpenOCD en GDB, maar dit is transparant, gezien door de ogen van de gebruiker.

Alessandro Ranellucci: Onze bijdrage was het ontwerpen van een gebruikerservaring hiervoor. Met Arduino is het veel eenvoudiger dan het gebruik van professionele volwaardige debug-tools, het testen, te documenteren, het bewustzijn vergroten, het verspreiden onder gebruikers en dan ook nog de interactie met Espressif.

Elektor: De details van de implementatie hiervan doen me denken aan online-vragen. Er was verwarring over de poorten wanneer je het board aansloot – er waren twee poorten en mensen wisten niet welke poort ze moesten gebruiken.

Martino Facchin: Ja, het is niet gemakkelijk om uit te leggen omdat de belangrijkste manier waarop we mensen altijd hebben verteld om dingen te doen is om te beginnen bij de basis, bijvoorbeeld: je doet het op deze manier en er is één seriële poort enzovoort, enzovoort, en dan ga je verder met een geavanceerd onderwerp. Geavanceerde onderwerpen moeten heel goed uitgelegd worden. Daarom hebben we documentatie over hoe debuggen op de juiste manier in zijn werk gaat. We laten de functie daar niet gewoon staan en om mensen dan te laten experimenteren. Bovendien lezen mensen meestal geen documentatie, dus we doen ons best om dit probleem te vermijden en alles beschikbaar te maken. De documentatie moet nog steeds gelezen worden, maar alles wordt er heel goed in uitgelegd.



Alessandro Ranellucci: En de meest recente versie van de Arduino IDE, die een maand geleden uitkwam, heeft veel verbeteringen op het gebied van poort- en boardselectie; de verwarring over meerdere poorten of poorten die veranderen nadat je bewerkingen hebt uitgevoerd enzovoort, wordt op een vriendelijkere manier afgehandeld.

Martino Facchin: We hebben ook dit DFU-menu toegevoegd, dus we hebben nu *pluggable discovery*. Dit is een heel interessant concept omdat je dingen kunt ontdekken via de seriële poort of via WiFi en MDNS. Paul Stoffregen voegde discovery via HID toe aan versie 1 van Arduino IDE omdat zijn bootloader HID-gebaseerd was, maar dit was altijd een externe patch voor het Arduino-installatieprogramma. We besloten om dit voor iedereen beschikbaar te maken. Dus portte hij zijn discoverer naar Arduino IDE versie 2. Toen hadden we een ander probleem met de Minima – wanneer je in bootloader-modus ging, werd er geen seriële poort zichtbaar. Het was verwarrend om die niet in het menu te zien. Daarom besloten we onze discoverer toe te voegen aan de DFU-poort. ESP32 heeft een DFU tijdens runtime, door de manier waarop je code uploadt. Af en toe kan het verschijnen op je computer, maar, zoals ik al eerder zei in de documentatie, alles wordt uitgelegd. Deze poort is bedoeld voor het uploaden en je kunt beide selecteren. Het verandert niets aan het feit dat het uploads accepteert.

Elektor: Waren er compatibiliteitswijzigingen op het gebied van bibliotheken van bestaande code bij de overgang naar de ESP32-chip voor het nieuwe Nano-model?

Alessandro Ranellucci: Inderdaad – telkens als we een nieuwe architectuur aan de familie toevoegen. Er zijn de officiële basisbibliotheken die geport moeten worden, vooral als ze sterk geoptimaliseerd zijn, zodat ze draaien met low-level code. In dit geval moesten we enkele basisbibliotheken uitbreiden. Maar het ecosysteem van bibliotheken voor de ESP32 was veel volwassenere.

Elektor: Kun je beveiligingsfuncties of overwegingen noemen die ertoe hebben geleid de ESP32 te kiezen voor de Arduino Nano ESP32, met name voor IoT-toepassingen?

Alessandro Ranellucci: Ik zou niet zeggen dat we speciaal voor de ESP32 hebben gekozen uit veiligheidsoverwegingen. Natuurlijk heeft de ESP32 een aantal mogelijkheden die we op dit moment gedeeltelijk gebruiken.

Martino Facchin: Echt gedeeltelijk – we gebruiken geen encryptie of wat ze ‘secure boot’ noemen omdat dat het leven van de gebruiker extreem moeilijk maakt. Je moet elke binary die je maakt ondertekenen, en dan moet je die voor elk board veranderen, anders is het zinloos. We implementeren het dus niet, maar we staan het natuurlijk wel toe. Vanuit het oogpunt van een integrator heeft dit product alle mogelijkheden om het echt veilig te maken. Maar dat is niet de reden waarom we het hebben gekozen.

Alessandro Ranellucci: Gewoonlijk hadden we op al onze boards een afzonderlijk hardware-beveiligingselement. Dat is de methode die we voor al onze producten hebben gekozen. In dit geval hebben we het niet geïmplementeerd omdat de hoofd-microcontroller in theorie deze mogelijkheden heeft. Voor nu hebben we echter besloten om deze chip te gebruiken zoals hij is, om het eenvoudig te houden. Niets weerhoudt ons natuurlijk van verdere ontwikkeling.

Martino Facchin: Hetzelfde gebeurde bijvoorbeeld met de Portenta H7, toen we de beveiligde bootloader en de MCO boot-infrastructuur uitbrachten om beveiligde OTA en beveiligde updates te bieden; dit was een opt-in, en niet iets wat we direct uit de doos leveren. Ik ben er vrij zeker van dat we uiteindelijk ook wat documentatie via een menu zullen aanbieden.

Elektor: Zijn er plannen om de Espressif-chip in meer Arduino-boards te gebruiken, zoals de PRO-lijn?

Alessandro Ranellucci: Ik kan niet antwoorden voor de PRO-lijn, maar over het algemeen wel.

Martino Facchin: We houden van de u-blox vormfactor omdat die goed bij de Nano's past. Hij is erg klein. We hebben hun modules in bijna alle Nano-boards gebruikt, hetzij als een begeleidende chip voor WiFi of, in het geval van de Nano BLE, als de hoofd-microcontroller. We zijn dus heel tevreden met hen als fabrikant. Tegelijkertijd hadden we natuurlijk op de UNO R4 de normale Espressif-module. En ja, we zijn ook andere dingen van plan.

Elektor: Kun je dieper ingaan op de samenwerking of het contact dat je had met het ESP32-ontwikkelingsteam om een naadloze integratie in het Arduino-ecosysteem te garanderen?

Alessandro Ranellucci: Ze hebben een goed ontwikkelingsteam bij Espressif. Ze doen hun best om de community bij het ontwikkelproces te betrekken. We hadden dus persoonlijke ontmoetingen met hen en ook regelmatige gesprekken. We deelden ook een communicatiekanaal op Slack of andere communicatieplatforms, zodat we een directe, dagelijkse manier hadden om elkaar bij te praten, elkaar op de hoogte te houden van problemen en onze handen uit de mouwen te steken voordat we het op GitHub openbaar maakten. Het was een zeer nauwe samenwerking. We kunnen dus veel mensen van Espressif noemen die ons hebben geholpen bij deze samenwerking.

Alessandro Ranellucci: Op een hoger niveau, heeft Ivan Grokhotkov (VP van Software Platforms, Espressif) ons geholpen om alle communicatie te vergemakkelijken, en Pedro Minatel (Developer Advocate) was bijzonder behulpzaam om ons te helpen om met de juiste persoon te praten.

Elektor: Zie je de samenwerking tussen Arduino en Espressif in de toekomst evolueren en welke mogelijkheden zie je voor verdere innovatie en samenwerking?

Alessandro Ranellucci: Ik zou zeggen dat we het, voorbij de hardwareproducten, in de twee teams eens zijn over de noodzaak om nauw samen te werken aan API-standaardisatie. De API is nu meer interoperabel tussen de Arduino- en ESP32-werelden. Op dat gebied willen we graag blijven samenwerken.

Martino Facchin: Espressif heeft zich ook op veel verschillende onderwerpen gericht, zoals Rust. Ze hebben veel ontwikkelaars die aan Rust werken, wat geen prioriteit voor ons is, maar we kunnen een aantal zeer interessante ideeën halen uit het werk dat ze daar doen. Dan is er nog het besturingssysteem Zephyr, waarin we allebei partners zijn. In de technische stuurgroep kunnen we dingen beslissen en Espressif ook. Alle ontwikkelingen die we allebei doen, zijn erop gericht om toekomstige tools voor toekomstige generaties beter te maken. Uiteindelijk werken we naar hetzelfde doel.

Elektor: Als je de ESP32 vergelijkt met andere chips, kun je dan ingaan op de specifieke benchmarks of prestatiegegevens die je tot de conclusie hebben gebracht dat dit de ideale keuze was voor de Arduino Nano ESP32?

Martino Facchin: De ESP32-S3 is een goede chip. Het stroomverbruik is niet opmerkelijk laag, maar het is niet slecht. Ultra-low power mogelijkheden zijn er als je het echt wilt pushen, maar

we gaan onze gebruikers niet vertellen dat ze alles in deep sleep moeten zetten en ultra-low power moeten gebruiken, ook al is het leuk om te weten dat deze dingen bestaan. Ik zou zeggen dat prestaties niet de belangrijkste reden zijn waarom de Nano-familie bestaat. Het is eerder gebruiksgemak. We hadden bijvoorbeeld de kans om de Portenta H7 te benchmarken op machine learning-capaciteiten. De Portenta H7 is ongeveer zeven keer sneller dan de ESP32, zelfs bij vergelijkbare kloksnelheden. De ESP32 heeft de helft van de kloksnelheid van de Portenta H7. De Nano staat voor gebruiksgemak, omgeving, beschikbaarheid van bibliotheken, vormfactor enzovoort.

Elektor: In hoeverre speelt het realtime besturingssysteem van de ESP32 een rol in het ontwerp van de ESP32 en zijn potentieel voor multitasking?

Alessandro Ranellucci: Dat is, zou ik zeggen, een groeiende behoefte – hoe meerdere cores te gebruiken, multitasking enzovoort. Deze behoefte is momenteel niet zo groot bij makers, maar er zal een ecosysteem moeten worden aangeboden.

Martino Facchin: We hebben iets meer dan een jaar geleden een poging tot standaardisatie gedaan, met iets dat we *Arduino threads* noemden. Op GitHub kun je zien dat we geprobeerd hebben het idee te promoten om threads verborgen te houden achter verschillende tabbladen in je IDE. Dus, je hebt een tabblad met `setup()` en `loop()`, zoals gewoonlijk. Dan heb je een ander tabblad met nog een `setup()` en `loop()`, en dit vertegenwoordigt je tweede thread. Dan is er een derde, een vierde – zoveel als je nodig hebt. Er is nog steeds het probleem van het synchroniseren van variabelen, waar je de gebruikelijke RTOS-beperking hebt die vereist dat je verschillende variabelennamen gebruikt tussen twee threads. We hebben dat op een hoger niveau opgelost. We hebben deze complexiteit dus verborgen door Mbed te gebruiken, maar het was eigenlijk geen Mbed-only ding. We wilden dit overzetten naar meerdere besturingssystemen.

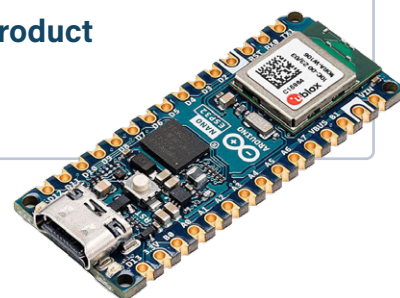
Als het gelukt is, heeft de maker-community hier helemaal niet op gereageerd. Het is dus waarschijnlijk niet iets wat echt nodig is voor makers. Threading is leuk, maar het maakt het je ook moeilijk. En tegelijkertijd kun je natuurlijk FreeRTOS gebruiken; je kunt doen wat je wilt als je er vaardig genoeg in bent, maar we dringen er niet op aan – gebruiksgemak is waar het ons altijd om is gegaan. ◀

230524-03



Gerelateerd product

➤ **Arduino Nano ESP32**
www.elektor.nl/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

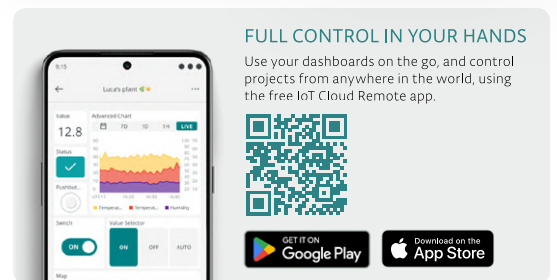
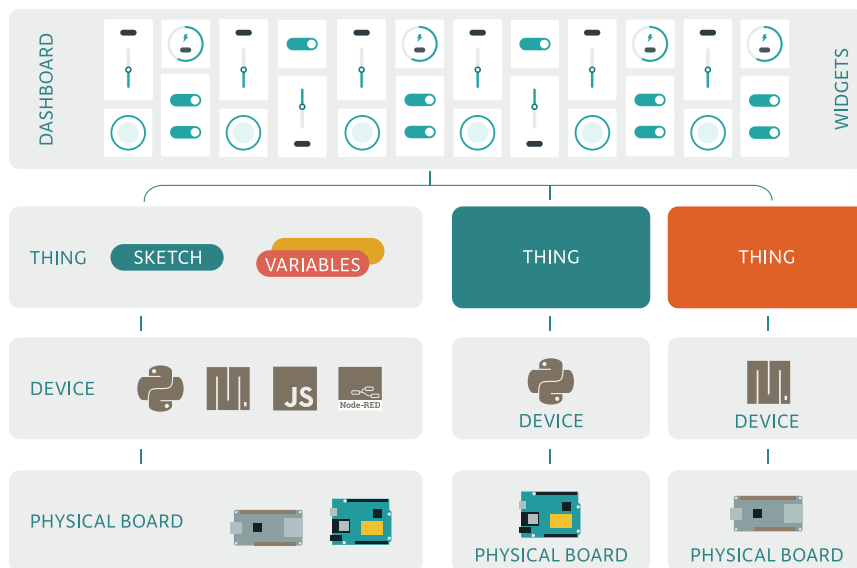
Program/Deploy

- + **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

Cloud Applications can be developed using the Arduino Cloud Editor or Arduino IDE 2.



ESP32/ESP8266

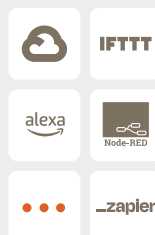
+70% Of Arduino Cloud active users use ESP-based boards.

OUTSIDE ARDUINO DEVELOPMENT ENVIRONMENTS



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

Third party platform integration



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.



Get 30% off
on the yearly Maker plan
with code ELEKTOR30*

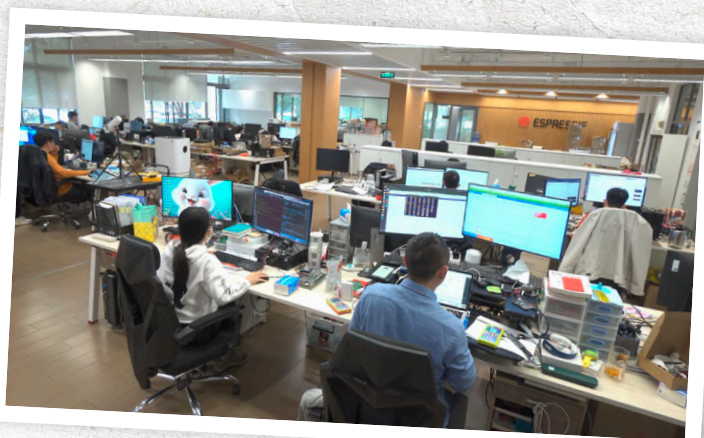
cloud.arduino.cc/elektor

IoT Chip Innovation

achter de schermen van



ESPRESSIF



De wereldwijde community van Elektor bestaat uit elektrotechnici, makers en studenten die geïnteresseerd zijn in een breed scala aan elektronica-gerelateerde onderwerpen. Enkele leden van deze community hebben onlangs een aantal prikkelende vragen opgesteld voor de engineers en het kader van Espressif.

Teo Swee Ann richtte Espressif in 2008 op. Wat bracht hem ertoe dit te doen? Wat was zijn visie voor Espressif in 2008 en hoe is zijn visie voor het bedrijf sindsdien veranderd? – Margriet Debeij (Nederland)

Bij de oprichting van Espressif was het oorspronkelijke doel van Teo om een product te maken dat analoog ontwerpen automatiseerde. Het bedrijf ontdekte echter zijn ware roeping in het ontwerpen van eigen internet-of-things (IoT) chips. Teo legde een duidelijke visie en strategie vast waarin de nadruk lag op het creëren van innovatieve, betaalbare en energiezuinige IoT-oplossingen. Teo's niet-aflatende nadruk op innovatie is van groot belang geweest voor het succes van Espressif Systems. Hij begreep ook de waarde van samenwerking met de ontwikkelaars-community en ondersteunde actief de groei van de developers' community van Espressif. Door een ecosysteem van samenwerking en kennisdeling te bevorderen, zorgde hij ervoor dat Espressif voorop bleef lopen op het gebied van technologische vooruitgang. Naarmate het technologielandschap zich ontwikkelde, leidde Teo Espressif Systems bij het adopteren van opkomende technologieën zoals kunstmatige intelligentie, machine learning en edge computing, om ervoor te zorgen dat het bedrijf voorop bleef lopen op het gebied van innovatie.



Hoe verhoudt ESP-NOW zich tot ZigBee of Matter? – Clemens Valens (Frankrijk)

Zowel ESP-NOW als ZigBee kunnen worden gebruikt om low-power IoT-apparaten te bouwen, maar de co-existentie van ESP-NOW met WiFi kan een breed scala aan mogelijkheden bieden en unieke toepassingen mogelijk maken. ESP-NOW biedt daarnaast een groter bereik en een grotere bandbreedte, vooral in outdoor-scenario's. ESP-NOW werkt zelfs zonder een WiFi-infrastructuur en is een eigen protocol van Espressif, terwijl Matter en ZigBee standaarden zijn die gedefinieerd zijn door de Connectivity Standards Alliance. Matter is een application layer-protocol dat Thread en WiFi ondersteunt als protocollen voor draadloze communicatie. Zowel ZigBee- als ESP-NOW-apparaten kunnen onderdeel worden van het Matter-netwerk met behulp van een Matter Bridge. We hebben oplossingen voor de ESP-NOW Matter-bridge en Zigbee Matter-bridge.

Espressif werkt samen met Arduino, en die samenwerking heeft geresulteerd in het Arduino Nano ESP32-board. Is het mogelijk dat jullie gaan samenwerken met het Raspberry Pi-ecosysteem en met een gezamenlijk product zullen komen?

– Ferdinand te Walvaart (Nederland)

We waarderen de samenwerking met Arduino. Deze omvat officiële ondersteuning voor de Arduino IDE (geïntegreerde ontwikkelomgeving) voor de ESP-IDF op verschillende chipsets van Espressif. We gaan nog een stap verder en hebben de Arduino UNO R4 WiFi en de Arduino Nano ESP32 aangekondigd, beide met de ESP32-S3 module. We geloven dat we allemaal, Arduino, Raspberry Pi en Espressif, een gemeenschappelijke community en een open source-gedreven mentaliteit delen, waardoor samenwerking iets natuurlijks is. We vinden dat Raspberry Pi zich op dit moment vooral richt op het MPU-segment en wij hebben geen product in dat segment. Maar met de lancering van de Pico-serie bestaat er misschien een mogelijkheid voor toekomstige samenwerking.

Er zijn speciale vaardigheden nodig voor het ontwikkelen en beheren van AI/IoT-technologieën. Er is een tekort aan gekwalificeerde professionals voor alle technische markten. Beperkt dit de groei van de producten en omzet van Espressif? – Ruud Schaay (Nederland)

Het tekort aan geschoolde professionals in AI/IoT vormt inderdaad een uitdaging. Onze producten zijn gericht op ontwikkelaars en bieden gebruiksvriendelijke platforms die worden ondersteund door uitgebreide hulpmiddelen. Hierdoor kunnen liefhebbers met verschillende achtergrond zich succesvol bezighouden met AI/IoT-technologieën. We werken ook actief samen met onderwijsinstellingen over de hele wereld via onze wereldwijde teams en online-communities, en bieden workshops en hulpmiddelen om ontbrekende vaardigheden bij te brengen. Deze aanpak ondersteunt niet alleen de groei van Espressif-producten, maar draagt ook bij aan de technologie in het algemeen. Onze focus op toegankelijkheid en onderwijs geeft aspirant-professionals meer mogelijkheden en stimuleert een cultuur van innovatie en het ontwikkelen van vaardigheden. Hoewel het tekort aan talent een punt van zorg is, zijn we vastbesloten om mensen uit te rusten met de tools en kennis die nodig zijn om te gedijen in de AI/IoT-arena, waardoor zowel Espressif als de industrie vooruit worden gestuwd.

Wat zijn de marketing-uitdagingen voor engineers in tegenstelling tot een meer algemeen publiek? – Erik Jansen (Nederland)

Ingenieurs vormen een groep die echt diep graaft in de technische details, dus we moeten de juiste balans vinden tussen nauwkeurigheid en begrijpelijkheid. Ons doel is om hen te tonen hoe onze producten passen bij hun specifieke behoeften en uitdagingen. Zij zijn werkelijk onderzoekers, dus we zorgen ervoor dat ze alle technische informatie krijgen die ze nodig hebben om weloverwogen beslissingen te kunnen nemen. We weten ook dat ze tijd doorbrengen op technische forums en platforms, en dus willen we daar ook bij zijn. Onze marketing moet als het ware een 'baken' zijn in die omgevingen. Engineers hechten waarde aan logische beslissingen, dus onze berichtgeving richt zich op hoe onze producten waarde, voordelen en technische innovatie toevoegen aan hun projecten. Het gaat erom hun taal te spreken en hen te laten zien hoe we hun engineering-inspanningen vlotter en spannender kunnen maken.

Hoeveel engineers werken er bij Espressif, en hoe is de bedrijfscultuur? Hoe is de werkomgeving voor engineers bij Espressif? – C. J. Abate (Verenigde Staten)

Espressif is in de waarste zin des woords een technologie-gedreven bedrijf. Meer dan 75% van ons personeelsbestand bestaat uit technische medewerkers; in totaal zijn er ongeveer 450 medewerkers in de verschillende R&D-afdelingen. Onze medewerkers hebben verschillende achtergronden; we tellen 30 verschillende nationaliteiten in onze kantoren in vijf verschillende landen, waaronder China, India, Tsjechië, Singapore en Brazilië. We zijn er trots op een bedrijfscultuur te hebben die werknemers aanmoedigt om innovatieve ideeën te onderzoeken, overwogen risico's te nemen en onze producten en diensten voortdurend te verbeteren. Iedereen is aanspreekbaar, er heerst een mentaliteit van innovatie en samenwerking en werknemers worden aangemoedigd om te doen waar ze in geloven en gepassioneerd over zijn. Dit helpt bij het creëren van gedifferentieerde en baanbrekende oplossingen.





Espressif lijkt een van de krachtigste RISC-V aanbiedingen voor MCU's te hebben. Zullen alle toekomstige IC's RISC-V gebruiken? – Stuart Cording (Duitsland)

We zorgen ervoor dat onze hardware breed toegankelijk is en dat onze software beschikbaar is in de open-source community. Dit is een kernfilosofie van Espressif. De introductie van RISC-V processoren in ons MCU-aanbod was een natuurlijke stap voorwaarts en we zijn trots op wat we hebben kunnen bereiken door klanten een probleemloze overgang te bieden tussen de verschillende producten binnen hetzelfde development framework. We zijn vastbesloten om RISC-V verder toe te passen in ons portfolio en zullen binnenkort ons eerste dual-core RISC-V product (ook bekend als ESP32-P4) op de markt brengen. De toepassing van RISC-V biedt ons flexibiliteit bij de implementatie en versterkt ons rijke portfolio van intellectueel eigendom, wat van cruciaal belang is om onze klanten meer geavanceerde en betaalbare oplossingen aan te kunnen bieden.

Terwijl 5G de markt in de richting van deterministische netwerken beweegt, met grote rekenprestaties en real-time respons, wordt de markt ook erg gevoelig voor energieverbruik. Hoe gaat Espressif om met deze schijnbaar onmogelijke combinatie? – Roberto Armani (Italië)

Het is waar dat energiebesparing en de ecologische footprint steeds belangrijker doelen worden voor onze klanten en hun eindgebruikers. We erkennen dit en blijven onze producten en oplossingen verbeteren om aan deze groeiende behoefte tegemoet te komen. We hebben bijvoorbeeld onze aanpak voor de light sleep-modus tegen het licht gehouden, waardoor de toepassing in de nieuwere producten zoals de ESP32-C6 en de ESP32-H2 de meeste periferie kan uitschakelen, wat tot 80% minder stroomverbruik kan opleveren. We bieden ook productoplossingen om tegemoet te komen aan de groeiende behoefte aan laag stroomverbruik. Zoals de ESP32-C2-gebaseerde ESP-NOW-schakelaar, die 10.000 maal kan worden ingedrukt op één enkele knoopcel. We blijven innoveren en ons richten op het vermindere van het totale stroomverbruik van onze producten.

Hoeveel boards/modules hebben jullie tot nu toe verkocht? – Muhammed Söküt (Duitsland)

We lanceerden ons eerste IoT SoC-product in 2014 in de vorm van de populaire ESP8266. Dit was een IoT-gamechanger-product dat draadloze connectiviteit en de microcontroller integreerde op dezelfde chip. De ESP8266 en het vlaggenschip ESP32, uitgebracht in 2016, brachten een revolutie teweeg in de industrie, wat leidde tot een cumulatieve verkoop van 100 miljoen stuks in 2017. Sindsdien blijven onze klanten ons trouw en hebben we ons ontwikkeld tot een bedrijf met nieuwe en innovatieve producten en oplossingen. Alleen al in 2021 hebben we meer dan 200 miljoen stuks verkocht. In totaal hebben we meer dan een miljard eenheden verscheept, zoals onlangs is bekendgemaakt.

Het succes van Espressif begon met de betaalbare en gebruiksvriendelijke ESP8266, die voornamelijk werd gebruikt om WiFi-verbindingen tot stand te brengen. Tegenwoordig worden de chips en SoC's van Espressif vaak gebruikt voor WiFi-verbindingen, in combinatie met andere controllers op hetzelfde board waarop de eigenlijke applicatie is ondergebracht. De ESP32-P4 valt vooral op omdat het een krachtige CPU op zichzelf is. Gaat Espressif nog verder in deze richting? Krijgen we bijvoorbeeld een ESP64? – Jens Nickel (Duitsland)

We hebben de ESP32-P4 gemaakt toen we zagen dat veel ontwerpers de ESP32 gebruiken zonder connectiviteit, en we ervan overtuigd waren dat we een krachtigere en geoptimaliseerde oplossing konden bieden om in deze behoefte te voorzien. Dankzij onze optimalisaties van de RISC-V-implementatie hebben we een krachtige multi-core MCU kunnen maken met een AI-uitbreiding (kunstmatige intelligentie). We hebben de periferie voortdurend verbeterd. Met dit alles zijn we goed uitgerust om snel nieuwe SoC-definities te maken. Maar het is misschien nog te vroeg om iets te zeggen over welke specifieke MCU-only aanbiedingen we in de toekomst zullen hebben.

De oplossingen van Espressif worden gebruikt in duizenden elektronicaontwerpen, van professionele toepassingen tot DHZ-makersprojecten. Jullie engineeringteam heeft vast een lijstje met favoriete projecten die je in de community hebt zien ontwikkelen. Kun je drie van de meest opwindende of innovatieve Espressif-gebaseerde projecten noemen die je via de community hebt zien ontstaan? – C. J. Abate (Verenigde Staten)

We komen voortdurend tal van boeiende projecten tegen binnen onze inventieve gemeenschap en de industrie. Dit nummer van Elektor bevat een selectie van deze opmerkelijke projecten, zowel hardware als firmware. Daaronder zijn ESP32-gebaseerde evaluatieboards, niet groter dan een knoopcel, maar ook ingenieuze pogingen om Linux op de ESP32-S3 te laten draaien. Daarnaast zijn we projecten tegengekomen die spelconsoles simuleren op ESP-producten en gesynchroniseerde digitale klokken maken met meerdere ESP-boards. Sommigen hebben zich zelfs verdiept in de bouw van VGA-kaarten met Espressif-producten of het porten van audiodrivers naar de ESP32. Het zou oneerlijk zijn om ons beperken tot slechts drie opvallende projecten.



Wat zijn jullie plannen voor toekomstige microcontrollers (bijvoorbeeld opvolgers van de ESP32)? Kunnen we dingen verwachten zoals geïntegreerde USB, 5 GHz WiFi of Bluetooth 5.x? – Dr. Thomas Scherer (Duitsland)

Sinds de lancering van ESP32 in 2016 hebben we een reeks producten uitgebracht in de ESP32-C-serie, de ESP32-S-serie en, recentelijk, de ESP32-H- en de ESP32-P-serie. Deze producten zijn gericht op de uiteenlopende behoeften van verschillende toepassingen en industrieën. De meeste van deze nieuw gelanceerde producten ondersteunen Bluetooth Low Energy 5. De ESP32-S2/S3 en ESP32-C6 ondersteunen USB OTG. De ESP32-H2, die eerder dit jaar in massaproductie ging, ondersteunt 802.15.4-connectiviteit, waardoor hij kan worden gebruikt in Zigbee- en Thread-connected toepassingen. We hebben onlangs ook de ESP32-C5 aangekondigd, die dual-band (2,4 GHz en 5 GHz) WiFi 6 zal ondersteunen en binnenkort beschikbaar zal komen. De komende ESP32-P4 zal een aantal geavanceerde human/machine-interface (HMI) en media-randapparaten ondersteunen, zoals MIPI-DSI en MIPI-CSI met geïntegreerde beeldsignaalprocessor (ISP), H.264-encoder, enzovoort, en zal een groot aantal nieuwe en diverse toepassingen mogelijk maken.

De producten van Espressif worden op grote schaal toegepast in producten zoals huishoudelijke apparaten, lampen, slimme luidsprekers, consumentenelektronica en betaalterminals. Zijn er plannen voor Espressif om het gebruik van zijn producten uit te breiden buiten de huidige toepassingen? – Alina Neacsu (Duitsland)

Bij Espressif zijn ons doel en onze focus het democratiseren van de toegang tot IoT-technologieën met innovatieve, op ontwikkelaars gerichte en betaalbare draadloze connectiviteitsoplossingen. De ESP32-chips serie zal zich blijven ontwikkelen en betere connectiviteit, meer rekenkracht, sterkere beveiliging, een steeds betere set periferie en een lager energieverbruik bieden. Daarnaast heeft Espressif zich ook ontwikkeld als leverancier van complete oplossingen, waarbij pijnpunten van klanten worden geïdentificeerd en effectief worden aangepakt met oplossingen die verder gaan dan de typische hardware en software development kits. ESP RainMaker, ESP Insights en ESP ZeroCode modules zijn hier goede voorbeelden van. Deze oplossingen en producten beperken ons niet tot een bepaald segment of een bepaalde branche.



Veel Espressif-producten bevatten een of andere draadloze zender, vaak voor de ISM-banden en met een ingebouwde antenne. Hoe zorgt Espressif ervoor dat RF-vermogen, bandbreedte en ongewenste emissies binnen de specificaties en wettelijke grenzen blijven? – Jan Buiting (Nederland)

Wanneer we onze producten ontwerpen, houden we rekening met en volgen we strikt de richtlijnen en specificaties voor de verschillende communicatieprotocollen en -media die zijn gedefinieerd door overheden en allianties. Onze producten en modules doorlopen laboratoria van derden voor regelgevende tests en certificeringen, wat garandeert dat ze voldoen aan de specificaties en limieten die voor verschillende geografische regio's zijn gedefinieerd. Je kunt deze certificeringen vinden op onze website. Ze kunnen door onze klanten worden gebruikt om het certificeringsproces van hun eigen producten te versnellen.

De industriële sector vraagt om betrouwbare en robuuste microcontrollers. Hoe denkt Espressif zijn kennis en capaciteiten te gebruiken om de concurrentie voor te blijven bij het inspelen op de specifieke behoeften van de industriële markt? – Saad Imtiaz (Pakistan)

De industriële markt heeft een aantal specifieke vereisten die niet gebruikelijk zijn in het consumentensegment. Deze kunnen worden onderverdeeld in ruwere bedrijfsomstandigheden, zoals hogere temperaturen, een grotere betrouwbaarheid (minder storingen) en een lange levensduur, waarbij het product meerdere jaren wordt ondersteund. Onze modules en SoC's werken bij temperaturen tot 105 °C, waardoor ze geschikt zijn voor gebruik in de meeste industriële toepassingen. Onze producten maken gebruik van uitgerijpte processen en ondergaan uitgebreide betrouwbaarheidstests om de laagst mogelijke storingspercentages voor onze klanten te garanderen.

Stroomlijnen van MCU-ontwikkeling met ESP-IDF Privilege Separation

Harshal Patil, Espressif

Dit artikel introduceert privilege separation in microcontroller-toepassingen met behulp van ESP-IDF van Espressif Systems. Hiermee wordt firmware in een beschermde kern en een gebruikerstoepassing gescheiden, wat de ontwikkeling vereenvoudigt. Het artikel behandelt stappen om aan de slag te gaan, waardoor MCU-ontwikkeling efficiënter verloopt.

Toepassingen op microcontrollers (MCU's) worden meestal ontwikkeld als monolithische firmware. Maar in een algemeen besturingssysteem zijn er twee werksmodi, de kernel- en de gebruikersmodus. In de kernelmodus heeft het programma directe en onbeperkte toegang tot systeembronnen, terwijl in de gebruikersmodus het applicatieprogramma geen directe toegang heeft tot systeembronnen. Om toegang te krijgen tot die systeembronnen, moet een *system call* gedaan worden. Het belangrijkste idee achter het realiseren van deze scheiding is om de ontwikkeling van de eindtoepassing gemakkelijker te maken zonder dat men zich zorgen hoeft te maken over de onderliggende veranderingen in het systeem, net zoals desktop- of smartphone-toepassingen worden ontwikkeld: het onderliggende besturingssysteem handelt de kritieke functionaliteit af en de eindtoepassing kan de interface gebruiken die door het besturingssysteem wordt aangeboden.

ESP Privilege Separation

Traditioneel wordt elke ESP-IDF-applicatie op een Espressif SoC gebouwd als één enkele monolithische firmware zonder enige scheiding tussen de 'kern'-componenten (besturingssysteem, netwerken enzovoort) en de 'applicatie'- of 'bedrijfs'-logica. In het ESP Privilege Separation framework splitsen we de firmware-image op in twee afzonderlijke en onafhankelijke binaries: de beveiligde en de gebruikerstoepassing.

Aan de slag

Laten we beginnen met het bouwen van ons eerste project, Blink. De knipper-LED is het "Hello World"-equivalent in embedded systeemwereld en demonstreert het eenvoudigste wat je met een MCU kunt doen om een fysieke uitvoer te zien. Aan de slag dus!

Stap 0: hardwarevereisten voor het project

- Een ESP32-C3 of ESP32-S3-gebaseerd development board met een on-board LED. Enkele devkits die gebruikt kunnen worden zijn de ESP32-C3-DevKitM-1 of de ESP32-S3-DevKitC-1. We kiezen de ESP32-C3-DevKitM-1 voor de volgende hands-on beschrijving.
- Een USB 2.0-kabel (standaard-A naar micro-B)
- Een computer met Windows, Linux of macOS

Stap 1: het ESP Privilege Separation-project ophalen

- Kloon de ESP Privilege Separation project-repository

```
git clone https://github.com/espressif/esp-privilege-separation.git
```

Stap 2: setup van de ESP-IDF-omgeving

- Kloon het ESP-IDF project-repository

```
git clone -b v4.4.3 --recursive https://github.com/espressif/esp-idf.git
```

- Setup van de tools

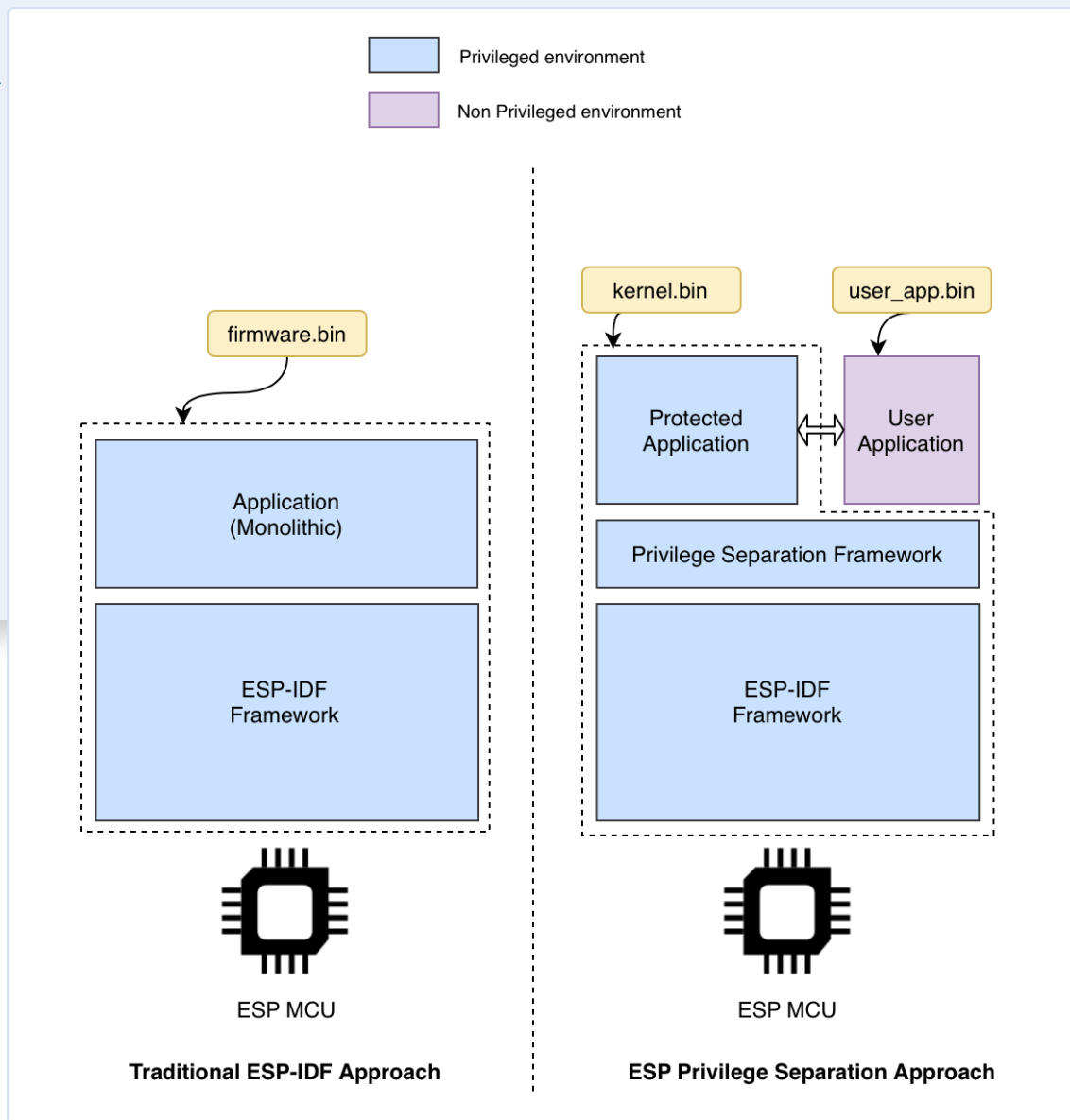
```
cd esp-idf
./install.sh
```

- Setup van de omgevingsvariabelen

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

- Toepassen van de ESP Privilege Separation-specifieke patch op ESP-IDF

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



Afbeelding 1. Traditionele ESP-IDF gebouwd als één enkele monolithisch blok firmware versus het ESP Privilege Separation-framework, waarbij het firmware-image is opgesplitst (bron: Espressif).

Stap 3: het Blink-voorbeeld uitvoeren

- Bouw van het voorbeeld

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

- Het voorbeeld flashen en uitvoeren

```
idf.py flash monitor
```

Een duik in de implementatie

Nu hebben we onze LED aan het knipperen, maar waarom verschilt dit van elk ander knipper-project? Antwoord: Privilege Separation. Laten we de implementatie proberen te begrijpen:

- Het voorbeeld is opgesplitst in twee applicaties: de beveiligde app en de gebruikersapp.
- De beveiligde app zoekt een gebruikerspartitie in de partitietabel en laadt de app in de door de gebruiker gedefinieerde secties.
- Vervolgens worden de geheugensecties voor de regio met lagere rechten (WORLD1) geconfigureerd en wordt toegang verleend zoals geconfigureerd door de ontwikkelaar.

- Tenslotte wordt er een aparte taak met lage privileges gestart met het toegangspunt voor de gebruiker.
- Zodra de gebruikersapp is geladen, wordt een taak gestart die de LED laat knipperen die is aangesloten op GPIO8 (ESP32-C3-DevKitM-1).

Integratie met ESP RainMaker

Nu hebben we onze LED-knipperende devkit, maar we zijn nog steeds niet in staat om de LED op ons commando te laten knipperen. Om het te kwalificeren als een echt 'connected' apparaat en de LED op commando te bedienen, moeten we het integreren met ESP RainMaker. ESP RainMaker is een lichtgewicht IoT Cloud-software, waarmee gebruikers op maat gemaakte IoT-oplossingen kunnen bouwen, ontwikkelen en implementeren met een minimum aan code en met maximale veiligheid. Laten we beginnen met het bouwen van het `rmaker_switch` voorbeeld in het ESP Privilege Separation-project.

Stap 0: vereisten

- ESP-IDF voor ESP Privilege Separation moet zijn ingesteld voordat er een voorbeeld wordt gebouwd
- ESP RainMaker Mobile Application (Android/iOS)
- WiFi-netwerk

Stap 1: setup van ESP RainMaker

- Kloon de ESP RainMaker project-repository

```
git clone --recursive https://github.com/espressif/esp-rainmaker
git checkout 00bcf4c0c30d96b8954660fb396ba313fb6c886f
export RMAKER_PATH=
```

Stap 2: het *rmaker_switch* voorbeeld uitvoeren

- Bouw van het voorbeeld

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

- Het voorbeeld flashen en uitvoeren

```
idf.py flash monitor
```

Stap 3: provisioneren van het apparaat

- Zodra het voorbeeld draait, verschijnt er een QR-code in de terminal. Deze QR-code kan gebruikt worden voor WiFi provisioning
- Meld je aan bij de mobiele applicatie ESP RainMaker en klik op *Add Device*. Dit opent de camera voor het scannen van QR-codes
- Volg de Provision-workflow zodat je apparaat verbinding kan maken met je WiFi-netwerk

Stap 4: de LED-schakelaar bedienen

- Uiteindelijk zie je dat een *Switch* is toegevoegd aan het start-scherm van de mobiele app
- Je kunt nu proberen het Switch-pictogram om te schakelen om zo de LED te bedienen

Maar hoe hebben we deze naadloze ESP RainMaker integratie bereikt? We introduceren een *rmaker_syscall* component in ons knipper-voorbeeld, dat systeemaanroepen voor het ESP Rainmaker framework blootlegt, en wanneer de gebruikersapplicatie opstart, initialiseert het de ESP Rainmaker-service in de beveiligde applicatie, en creëert het een ESP RainMaker switch-device. Omdat de ESP RainMaker-component in de beveiligde applicatie wordt geplaatst, zijn systeemaanroepen nodig voor alle openbare API's die door deze component worden blootgelegd. De eenvoudige uitbreidingsmogelijkheden van ESP Privilege Separation laten je toe om toepassings-specifieke aangepaste systeemaanroepen toe te voegen, en deze laag gebruikt de data opgeslagen in de apparaatcontext om user-space lees- en schrijf-callbacks uit te voeren.

OTA firmware-updates met ESP Privilege Separation

Over-the-air (OTA) firmware actualiseren is een van de belangrijkste functies voor elk connected apparaat. Het stelt ontwikkelaars in staat om nieuwe functies en bugfixes uit te brengen door de applicatie op afstand bij te werken. In ESP Privilege Separation zijn er twee applicaties – *protected_app* en *user_app* – waarvoor het framework de mogelijkheid biedt om beide binaries onafhankelijk van elkaar bij

te werken. De beschermde applicatie, die een hoger privilege heeft, kan worden bijgewerkt door gewoon de normale OTA-interface te volgen die wordt geleverd door ESP-IDF, terwijl de OTA-update van de lager geprivilegieerde gebruikersapplicatie mogelijk wordt gemaakt doordat ESP Privilege Separation een systeemaanroep voor de gebruikersapplicatie, *usr_esp_ota_user_app()*, beschikbaar stelt om het OTA-proces uit te voeren.

Laten we het OTA-voorbeeld van de gebruikersapp uit het ESP Privilege Separation-project eens uitproberen.

Stap 0: vereisten

- ESP-IDF voor ESP Privilege Separation moet zijn ingesteld voordat er een voorbeeld wordt gebouwd
- een openbare URL van de gehoste image van de nieuwe gebruikersapp

Stap 1: het *rmaker_switch* voorbeeld uitvoeren

- Bouw van het voorbeeld

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

- Het voorbeeld flashen en uitvoeren

```
idf.py flash monitor
```

Stap 2: de OTA-update starten

Voer de OTA-URL in omdat de gebruikersapp de OTA-URL accepteert met het commando *user-ota* via de console. Dit initieert een OTA en de nieuwe gebruikersapp start op zodra de OTA is voltooid. Nu hebben we ons connected apparaat met de belangrijkste functies klaar om te implementeren.

Je kunt de QR-code scannen om het ESP Privilege Separation-voorbeeld te bekijken. Dit voorbeeld demonstreert een echt praktijkgeval van een OTA-update van een gebruikersapp met behulp van ESP Rainmaker en ESP Privilege Separation.



Zoals gezegd wordt het succes van elk product door de gebruikers bepaald, en dat zijn jullie! We horen graag wat je ervan vindt, dus deel je ervaringen, suggesties en beoordelingen met ons. Je feedback voedt onze innovatie en helpt ons om ons product te verfijnen zodat het beter aan je behoeften voldoet. Bezoek onze website [1], stuur ons een e-mail, maak contact met ons op sociale media om je inzichten te delen of open een nieuw probleem in de GitHub-repository van het project [2]. Als je een specifieke wens hebt waarvan je denkt dat die goed in dit framework past, of als het oplossen van dergelijke problemen je enthousiast maakt, bespreken we een eventuele samenwerking graag met je! ➡

230598-03



Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via harshal.patil@espressif.com of naar de redactie van Elektor via redactie@elektor.com.

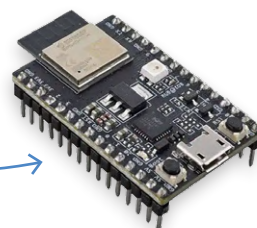
Over de auteur

Harshal Patil werkt sinds een jaar als software-engineer bij Espressif Systems. Hij is gepassioneerd over het oplossen van echte problemen door veilige, verbonden systemen te bouwen. Buiten zijn werk verkent hij graag nieuwe innovatieve gadgets en speelt hij voetbal.



Gerelateerde producten

- > **ESP32-C3-DevKitM-1**
www.elektor.nl/20324
- > **ESP32-S3-DevKitC-1**
www.elektor.nl/20697



WEBLINKS

- [1] Espressif: <https://espressif.com>
- [2] De project-repository: <https://github.com/espressif/esp-privilege-separation/issues>



ESP-IDF

Espressif IoT Development Framework

ESP-IDF is de officiële ontwikkel-SDK van Espressif die alle SoC's uit de ESP32-, ESP32-S-, ESP32-C- en ESP32-H-serie ondersteunt. Dit is uw beste startpunt voor het bouwen van alles dat geen specifieke SDK nodig heeft bovenop ESP-IDF. ESP-IDF bevat de FreeRTOS-kernel, device drivers, flash storage stacks, protocolstacks voor WiFi, Bluetooth, BLE, Thread en Zigbee, een TCP/IP-stack, TLS, protocollen op applicatieniveau (HTTP, MQTT, CoAP) en veel andere softwarecomponenten en tools.

Het biedt ook veelgebruikte high-level functionaliteit, zoals OTA en netwerkprovisioning. Naast commandoregel-ondersteuning, worden ook Eclipse en VSCode IDE-integraties ondersteund. Merk op dat u veel voorbeeldapplicaties vindt die u helpen

om snel aan de slag te gaan. ESP-IDF heeft een goed gedefinieerde ondersteunings- en onderhoudsperiode, en de laatste stabiele versie wordt aanbevolen voor de ontwikkeling van nieuwe projecten.

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



Een open-source

spraakherkenningsserver...

...en de ESP BOX

Kristian Kielhofner (Verenigde Staten)

Velen van ons zijn dol op Alexa Echo en soortgelijke apparaten, maar er zijn altijd zorgen geweest over privacy en datagebruik. Het Willow Platform is een gratis open-source alternatief. Naast de Willow Inference-server is er een hoogwaardige spraakgestuurde gebruikersinterface nodig die de audio opneemt en oppoetst. De ESP BOX van Espressif heeft niet alleen microfoons en audio processing voor een aantrekkelijke prijs, maar ook een krachtig software-ecosysteem.

Willow Platform is een gratis open-source alternatief. Naast de Willow Inference-server is er een hoogwaardige spraakgestuurde gebruikersinterface nodig die de audio opneemt en oppoetst. De ESP BOX van Espressif heeft niet alleen microfoons en audio processing voor een aantrekkelijke prijs, maar ook een krachtig software-ecosysteem.

Sinds de introductie van het Alexa-platform acht jaar geleden heeft Amazon meer dan 500 miljoen Echo-apparaten verkocht. Er bleven echter altijd bezorgdheid en controverses met betrekking tot privacy, gegevensgebruik en de alsmaar toenemende pogingen van Amazon om nog meer geld te verdienen aan Echo-gebruikers. Willow [1] is een alternatief platform dat een gratis, open source en maker-vriendelijke Alexa-spraakinterface biedt zonder dat dit ten koste gaat van de kwaliteit of de portemonnee.

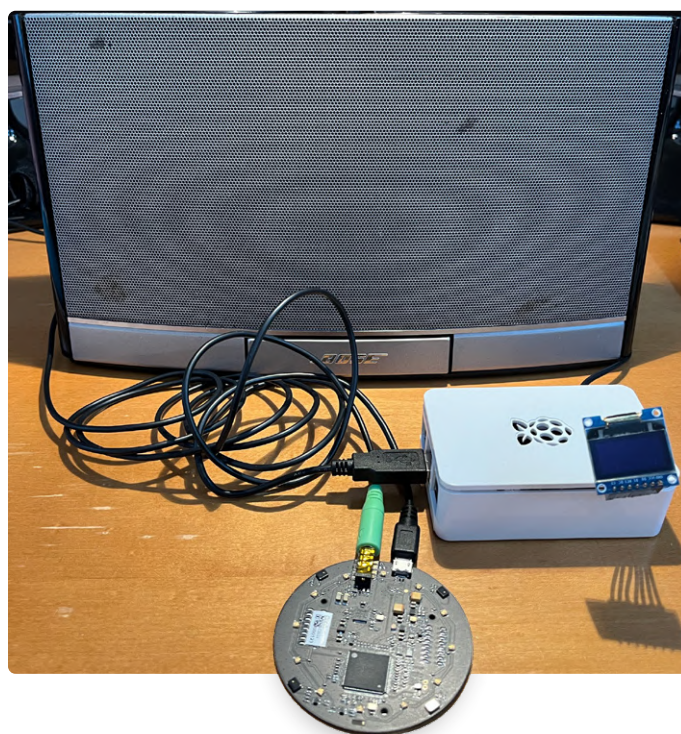
Hardware

Raspberry Pi

Een spraakinterface van hoge kwaliteit moet bestaan in en communiceren met de fysieke wereld. Jarenlang heeft het open-source

ecosysteem spraakinterfaces geprobeerd op basis van de Raspberry Pi, verschillende microfoons enzovoort (**figuur 1**). Deze aanpak gaat gepaard met enkele grote uitdagingen:

- **Prijs.** Tegen de tijd dat je een Raspberry Pi, aanraak-LCD, hoogwaardig microfoonarray, luidspreker, geschikte behuizing enzovoort bij elkaar hebt, kijk je al tegen een prijs aan die minstens driemaal zo hoog is als die van een Echo-apparaat (\$ 50 of minder). Bij deze aanpak moet je componenten



Figuur 1. Eerdere pogingen van de auteur met de Raspberry Pi.

zoeken, alles zorgvuldig opbouwen, een geschikte behuizing maken, de software ontwikkelen enzovoort. Doe dit voor meerdere apparaten in je omgeving en de benodigde tijd en kosten lopen aanzienlijk op.

- > **Verkrijgbaarheid.** Het is de laatste tijd beter geworden, maar alleen al met de Raspberry Pi waren er grote problemen met de leverbaarheid van deze componenten. De afgelopen jaren varieerde de verkrijgbaarheid van onmogelijk tot extreem duur vanwege secundaire doorverkoop.
- > **Beheer.** Oplossingen op basis van de Raspberry Pi vereisen vaak een volledige Linux-distributie inclusief ondersteuning voor hardware-componenten, alsmede een verbijsterende hoeveelheid software die nodig is voor een spraak-UI. Het beheren van een half dozijn Linux-computers kan voor veel gebruikers een uitdaging blijken.
- > **Kwaliteit.** Amazon (en anderen) hebben veel geïnvesteerd in het ontwerp van een spraakinterface die goed functioneert in akoestisch uitdagende omgevingen met een grote verscheidenheid aan menselijke sprekers. Het is meer dan alleen maar een microfoon en luidspreker op een Pi aansluiten.
- > **Ecosysteem.** Als je eenmaal een nauwkeurig transcript van een spraakcommando hebt, moet je er ook iets mee doen en de gebruiker feedback geven.
- > **Prestaties.** Met de meest optimale spraakherkennings-toepassingen kan een Raspberry Pi 4 bij benadering real-time spraakherkenning uitvoeren met het laagste kwaliteitsmodel. De nauwkeurigheid laat veel te wensen over en 'real time' is gewoon niet snel genoeg – vooral als je bedenkt dat je bij een fout in de transcriptie het commando moet herhalen, waardoor het gemak van een spraakinterface wegvalt.

In **tabel 1** kun je zien dat een Raspberry Pi iets sneller is dan realtime spraakherkenning bij gebruik van het model met de geringste kwaliteit.

Interessant is dat spraakherkenningsmodellen betere realtime-factoren hebben met langere spraakfragmenten. Helaas zijn de gesproken commando's voor spraakassistenten erg kort, waardoor de prestaties veel slechter worden. Deze benchmark is eigenlijk in het voordeel van de real-time spraakherkenningsprestaties van de Raspberry Pi.

We kennen de Raspberry Pi allemaal en houden ervan – het is een geweldig platform voor een breed scala aan toepassingen en gebruiksscenario's. Helaas hoort een assistent met spraakherkenning en -synthese daar niet bij.

Zoals het screenshot van **figuur 2** laat zien, duurde de spraakherkenning van Willow met de Willow Inference Server van het einde van de spraak naar de bevestiging van de door het Home Assistant-domoticasysteem voltooide actie 222 milliseconden. Dit is ruwweg 25% van de tijd die de Raspberry Pi 4 nodig heeft om alleen de spraakherkenning uit te voeren – en dat bij gebruik van een spraakherkenningsmodel dat aanzienlijk nauwkeuriger is dan het eenvoudige model dat nauwelijks realtime is op de Raspberry Pi. Net als vele anderen heb ik in de loop der jaren een paar pogingen gedaan om zelf een spraakinterface te bouwen, met verschillende

Table 1: Raspberry Pi 4 speech recognition performance.

Model	Beam Size	Spraakduur (ms)	Inferencetijd (ms)	Realtime-factor
tiny	1	3.840	3.333	1,15×
base	1	3.840	6.207	0,62×
medium	1	3.840	50.807	0,08×
large-v2	1	3.840	91.036	0,04×

```

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=sma
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
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

```

Figuur 2. De prestaties van Willow-spraakherkenning.

benaderingen. Helaas was het resultaat altijd hetzelfde – veel werk en kosten die een interessante demo kunnen opleveren, maar die in de echte wereld volledig onpraktisch en onbruikbaar zijn vanwege de hierboven genoemde (en andere) problemen.

ESP BOX

Toen ontdekte ik het ESP BOX-ontwikkelplatform van Espressif (**figuur 3**). Zoals zovelen gebruik ik al een jaar of tien Espressif-onderdelen voor verschillende toepassingen en ik weet hoe robuust, veelzijdig, kosteneffectief en feitelijk beschikbaar de hardware is.



Figuur 3. ESP32-S3-BOX-3 wordt geleverd met een 2,4-inch display, twee microfoons en een luidspreker.

Ik weet ook van eerdere projecten dat Espressif veel ondersteunende software en documentatie van hoge kwaliteit levert bij hun hard- en software.

De ESP BOX van Espressif is een ontwikkelplatform dat speciaal voor dit soort toepassingen is gemaakt. Voor ongeveer \$ 50 bij je favoriete DHZ-elektronicawinkel of -distributeur krijg je het volgende:

- › een ESP32-S3 met 16 MB flash en 16 MB high speed RAM (via SPI);
- › een 2,4"-display met capacitief aanraakscherm;
- › twee microfoons (erg belangrijk – hierover later meer);
- › luidspreker voor audio-uitvoer;
- › hardware mute-knop;
- › veel uitbreidingsmogelijkheden met de nieuwe ESP32-S3-BOX-3 printen en componenten.

En dat alles is klaar voor gebruik in een esthetisch aantrekkelijke, akoestisch geoptimaliseerde behuizing. Theoretisch zou ik met de ESP BOX en \$ 50 een apparaat kunnen hebben dat ik uit de doos kan halen, kan flashen en in mijn keuken, slaapkamer, kantoor of waar dan ook kan opstellen.

Software

Espressif levert niet alleen deze bijna perfecte hardware; het bedrijf is al lange tijd een voorstander van open source. Ik vond het geweldig om te zien dat ze de ESP BOX ondersteunen met een verscheidenheid aan vrije en open source-bibliotheken:

- › ESP IDF als standaard SDK;
- › ESP ADF voor audiotoeepassingen;
- › ESP SR voor spraakherkenning;
- › ESP DSP voor sterk geoptimaliseerde signaalverwerkingsroutines (FFT, vectorberekeningen enzovoort);

- › een LVGL-component om LC-displays aan te sturen (met aanraakfunctionaliteit).

Met de ESP BOX en deze bibliotheken heb ik binnen een week een zeer eenvoudig proefconcept ontwikkeld dat spraak kan vastleggen, naar mijn spraakherkenning-implementatie kan sturen en het transcript naar een platform kan sturen om actie te ondernemen.

Audio

Zoals ik had geleerd van mijn eerdere (mislukte) pogingen, begint een spraakinterface bij het wekwoord. Je moet een spraakinterface met een specifiek commando kunnen aanspreken, het zo doen ontwaken zodat het begint spraak vast te leggen. Het wekwoord is het equivalent van een aan/uit-knop – hij mag niet ‘zomaar’ inschakelen en net als bij een echte knop moet hij telkens betrouwbaar werken. Als je voor spraak-toepassingen jezelf meerdere keren moet herhalen om het apparaat te activeren, kom je al snel in een scenario terecht waarin het sneller, gemakkelijker en veel minder frustrerend is om gewoon je telefoon te pakken en te doen wat je wilde doen.

Gelukkig biedt het ESP spraakherkennings-framework (SR, Speech Recognition) niet alleen een engine voor wekwoorden, maar bevat het ook verschillende ingebouwde wekwoorden. Ik vond de ontwaak-implementatie extreem betrouwbaar – het ontwaakte consistent en minimaliseerde onterechte activering. Dankzij ESP SR had ik een betrouwbare gesproken “aan/uit-knop”.

Dan de volgende uitdaging - het verkrijgen van zuivere audio. Opnieuw kwam ESP-SR te hulp. ESP-SR bevat de Espressif AFE (audio front-end). Over deze AFE zouden we een boek kunnen volschrijven, maar in het kort biedt het een audioverwerkingslaag tussen de microfooningang en de audio-opname met de volgende functies:

Acoustic Echo Cancellation (AEC). Een van de vele uitdagingen bij spraakherkenning op grotere afstand zijn de akoestische eigenschappen van de fysieke omgeving. Afstand, harde reflecterende oppervlakken, ingewikkelde geluidsgolf-trajecten (hoeken, in de weg staande objecten) enzovoort kunnen veel echosignalen veroorzaken. De AEC-implementatie in ESP SR elimineert een groot deel van deze echo's, en verwijdert ook echo in scenario's met bidirectionele audio (zoals een speakerphone-toepassing).

Blind Source Separation (BSS). In luidruchtige omgevingen is het belangrijk om andere geluiden dan spraak, die kunnen bijdragen aan een slechte kwaliteit van de spraakherkenning, te elimineren. De ESP-SR BSS-implementatie, die gebruik maakt van meerdere microfoons, kan in wezen de audio-opname ‘richten’ op het binnenkomende geluid om zo de hoeveelheid achtergrondgeluid aanzienlijk te verminderen.

Noise Suppression (NS). In gevallen waar er slechts één microfoon is (of slechts één microfoon actief is) kan ruisonderdrukking de opname van niet door mensen gemaakte geluiden aanzienlijk reduceren. Voor toepassingen waarbij op maat gesneden hardware

wordt gebruikt, kan het gebruik van een enkele microfoon de materiaalkosten en ontwerpcomplexiteit aanzienlijk verlagen.

Voice Activity Detection (VAD). Een betrouwbaar wekwoord is slechts één stukje van de puzzel. Bij het ontwaken starten we met het opnemen van audio, en we moeten stoppen zodra de persoon ophoudt te spreken. VAD kan het begin en einde van spraak detecteren, zodat de gebruiker de opname niet handmatig hoeft te beëindigen.

Willow Inference Server

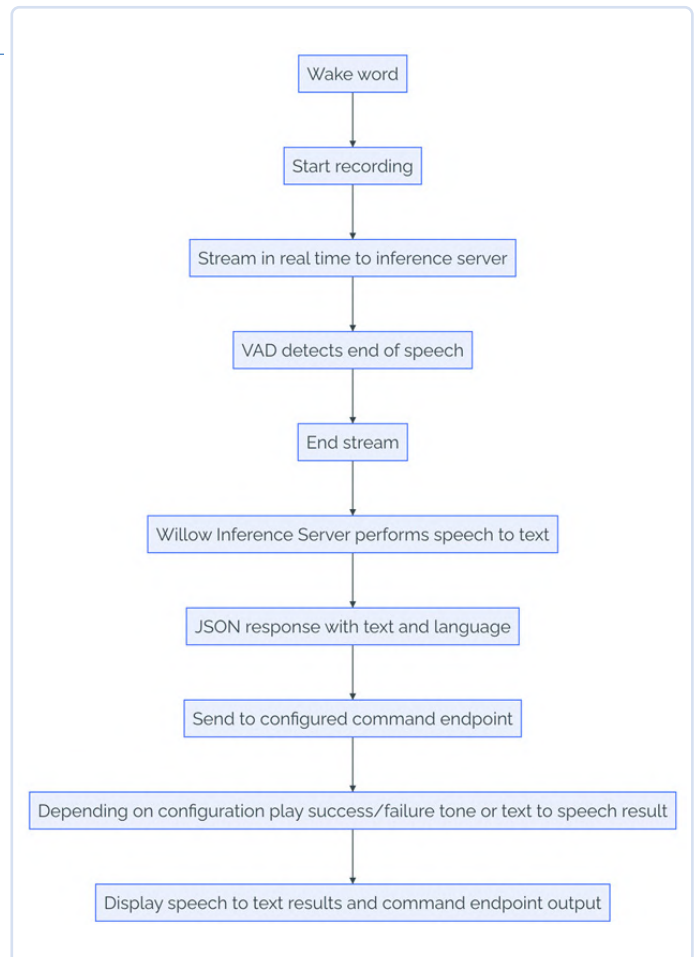
Nu we kunnen wekken, zuivere spraak kunnen opnemen en aan het einde van spraak kunnen stoppen, moeten we het ergens naartoe sturen. Gelukkig biedt Espressif het Audio Development Framework (ADF) voor deze taak aan. In mijn snelle en eenvoudige proefconcept gebruikte ik het ESP ADF *HTTP stream pipeline*-voorbeeld om AFE-bewerkte opgenomen audio in gedeelten via HTTP POST naar een HTTP-eindpunt te sturen. Ik kon mijn spraakherkenning inference server-implementatie snel aanpassen om binnenkomende audioframes van de ESP-BOX te bufferen, te wachten op een eindmarkering (dankzij VAD) en deze buffer onmiddellijk door te geven aan het onderliggende spraakherkenningsmodel. Wanneer de inference server het spraaktranscript retourneert, wordt dit als een HTTP JSON-antwoord aangeboden aan de ESP-BOX voor verdere uitvoering. Deze implementatie van een spraakherkennings-inference server werd bekend als de *Willow Inference Server (WIS)*.

De ESP-BOX met Willow kan het spraaktranscript ontvangen en doorsturen naar een door de gebruiker geconfigureerde Home Assistant, OpenHAB of aangepast HTTP REST-eindpunt. Willow toont het transcript en het antwoord van het eindpunt op het display. Afhankelijk van de gebruikersconfiguratie zal het ook een geslaagd/mislukt-toon afspelen of tekst-naar-spraak van de Willow Inference Server gebruiken om de uitgevoerde tekst van het spraakcommando uit te spreken.

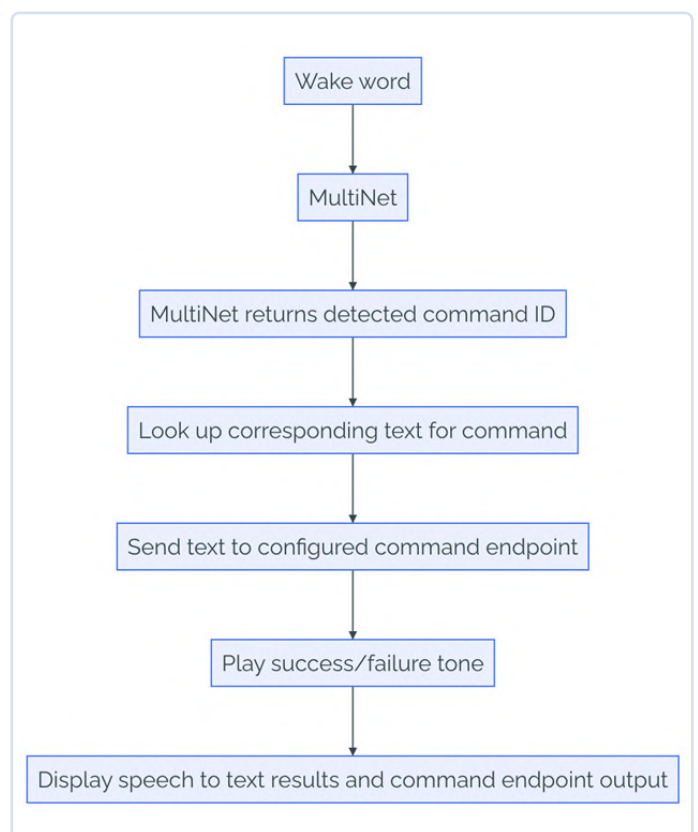
Nu kan ik met de ESP-BOX, Willow en de Willow Inference Server een wekwoord uitspreken, een commando opnemen en naar de Home Assistant sturen – met een latency van het einde van de spraak tot het voltooien van de actie van minder dan 500 ms (**figuur 4**), afhankelijk van de WIS-hardware en -configuratie.

Multinet: geen extra servers of hardware

De magie van ESP SR is echter nog niet voorbij. Naast de meegeleverde wekwoord-modellen bevat ESP SR een spraakherkenningsmodel genaamd MultiNet voor volledige on-device spraakherkenning. ESP SR biedt de mogelijkheid om tot 400 voorgedefinieerde spraakcommando's volledig op het apparaat te herkennen (zonder Willow Inference Server). Willow biedt ondersteuning voor MultiNet en wanneer het gebruikt wordt met Home Assistant kan het zelfs de namen van geconfigureerde entiteiten ophalen om automatisch deze grammatica te genereren (**figuur 5**) – zonder extra componenten en met prestaties en nauwkeurigheid die vergelijkbaar zijn met die van de Willow Inference Server voor deze voorgedefinieerde commando's.



Figuur 4. Willow Inferentie Server-flow.



Figuur 5. MultiNet modus-flow.

Nog steeds maker- en hacker-vriendelijk

Hoewel de ESP BOX met Willow Alexa-apparaten in enkele minuten kan vervangen, blijft hij trouw aan zijn Espressif-maker-afkomst. De ESP BOX ondersteunt een groot aantal componenten met diverse sensoren, GPIO, USB host-interface en meer via de uitbreidingsinterface [2]. Serieel en JTAG zijn natuurlijk beschikbaar via de USB C-poort op de hoofdunit.

Willow is geschreven in C met ESP IDF, maar de ESP BOX ondersteunt ook platforms als Arduino, PlatformIO en CircuitPython. Met Willow en de ESP BOX hebben we nu wat lang de 'heilige graal' van open-source spraakinterfaces is geweest: een Alexa-achtige ervaring voor een vergelijkbare prijs maar met meer flexibiliteit, controle en volledige privacy die ook de open-source software- en hardware-interfaces biedt waar we allemaal van genieten! 

230564-03 (vertaling: Hans Adams)

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via kris@tovera.com of naar de redactie van Elektor via redactie@elektor.com.



Over de auteur

Kristian Kielhofner is de oprichter van Willow – een open-source project om een lokale, zelf-gehoste spraakassistent te maken die een alternatief is voor Amazon Echo of Google Home. Sinds hij als kind met de Apple II speelde, is Kristian een technologie-enthousiast gebleven en heeft hij meerdere open-source projecten en startups in de spraak- en machine-learning-wereld opgericht. Kristian werkt nu aan Willow, andere open-source projecten en geeft advies aan beginnende technologiebedrijven.

WEBLINKS

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

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



Gerelateerde producten

> **ESP32-S3-BOX-3**
www.elektor.nl/20627



ESP-IoT-oplossing

device drivers, code-frameworks en meer voor uw IoT-systemen!

ESP-IoT-Solution is een repository waar u veel sensor-, display-, audio-, input- en actuator-driverimplementaties voor Espressif SoC's kunt vinden.

Daarnaast biedt het ook enkele higher-level frameworks die vaak nodig zijn, zoals een driver voor drukknoppen die lang en kort indrukken kan herkennen. Het bevat ook specifieke toepassingsvoorbeelden voor energiezuinige modi, opslag en beveiliging.

Als u directe telefoon-naar-apparaat BLE-gebaseerde OTA-upgrades wilt gebruiken, dan is dit de plek om te kijken. Veel van deze functionaliteit is beschikbaar als afzonderlijke IDF-component die u rechtstreeks in uw project kunt importeren.

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



ESPRESSIF

Het denkende oog

gezichtsherkenning en meer met de ESP32-S3-EYE

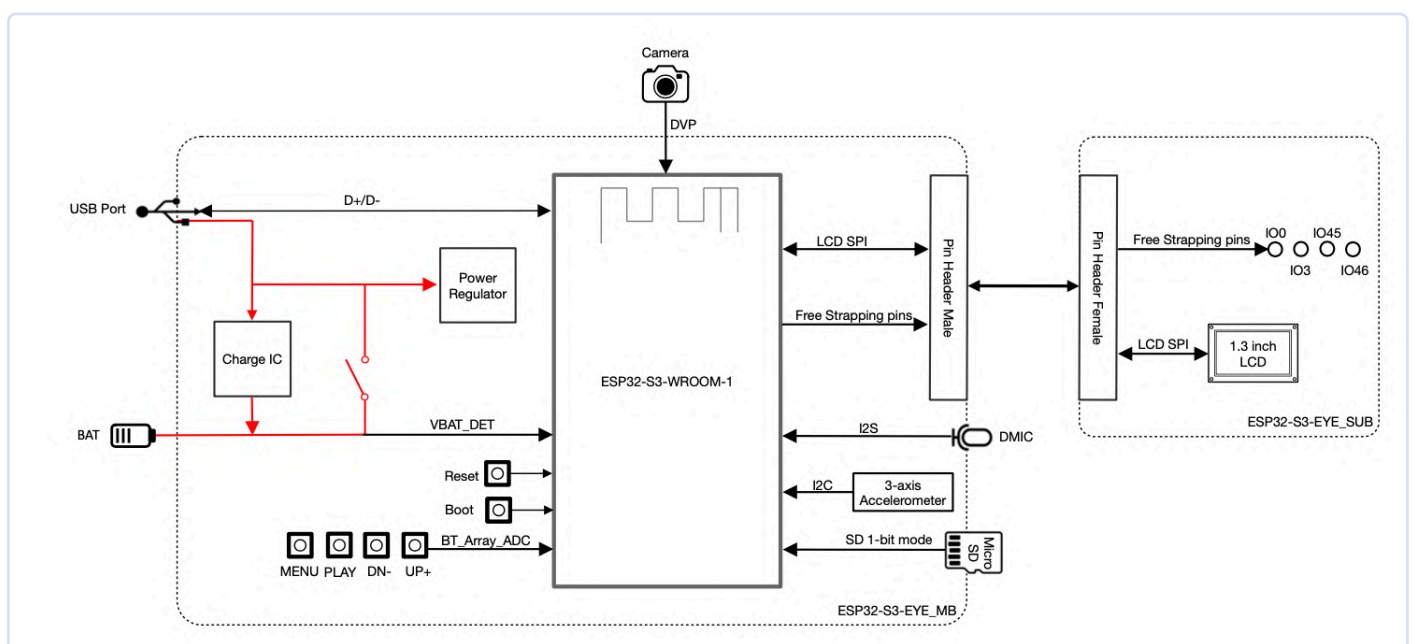
By Tam Hanna (Hongarije)

Het ESP32-S3-EYE-board van Espressif is een platform dat speciaal is ontworpen voor het evalueren van beeldherkennings- en andere AI-toepassingen. Het wordt geleverd met een software-ecosysteem en talloze voorbeelden, zowel voor het ESP-WHO framework voor gezichtsherkenning van de fabrikant als voor handmatige bediening met TensorFlow. Handen uit de mouwen dus!

Algoritmen voor kunstmatige intelligentie hebben de afgelopen jaren een enorme ontwikkeling doorgemaakt en zijn veel complexer geworden. Er zijn nu krachtige systemen die indrukwekkende prestaties leveren die lijken op die van mensen (vooral in het geval van AI die in de cloud wordt gehost). De wet van Moore en de steeds hogere eisen van eindgebruikers zorgen ervoor dat veel fabrikanten van microcontrollers 'AI-geoptimaliseerde' chips in hun assortiment opnemen. In het geval van Espressif is dit de ESP32 in de S3-variant, die met vector-instructies een soort AI-versneller of een specifieke AI-instructieset biedt. Rond deze ESP32-S3 is de afgelopen maanden een degelijk en actief ecosysteem van ontwikkelaars ontstaan, waardoor het voor gebruikers gemakkelijker is geworden om verschillende toepassingen voor kunstmatige intelligentie te implementeren. In dit artikel zullen we de mogelijkheden nader bekijken en met de technologie experimenteren.

Opstarten zonder stress

Espressif is zich terdege bewust van de behoeften van ontwikkelaars. Sinds de introductie van het ESP32-LyraT-board een paar jaar geleden, dat gericht was op audio-toepassingen, heeft het bedrijf consequent evaluatieboards geïntroduceerd die geoptimaliseerd zijn voor 'speciale' toepassingen.



Figuur 1. Het blokschema van de ESP32-S3-EYE (bron: [10]).

Ik zal de ESP32-S3-Eye gebruiken om alle voorbeelden uit te voeren die hieronder worden beschreven. **Figuur 1** toont een blokschema van alle componenten op de print.

Een belangrijke reden om dit evaluatieboard te gebruiken, dat in de VS ongeveer \$ 45 kost, is de aanwezigheid van zowel een camera als een klein LC-display op de printplaat. [1] Dit kleurenscherm bleek erg nuttig te zijn, omdat de resultaten van het machine learning-proces direct zichtbaar zijn, zonder dat er een PC aan te pas hoeft te komen. Op de voorzijde van de print, net onder het scherm, is een digitale microfoon gemonteerd. Deze kan gebruikt worden om verschillende spraakherkennings-engines te testen.

Het is de moeite waard om op te merken dat de experimenten van dit artikel ook uitgevoerd zouden kunnen worden op een ander hardware-platform. Het schema van het ESP32-S3-EYE-board vind je op [2], en ik beveel het aan als uitgangspunt voor aangepaste schakelingen. De door Espressif gebruikte componenten zijn over het algemeen gemakkelijk verkrijgbaar bij veel leveranciers.

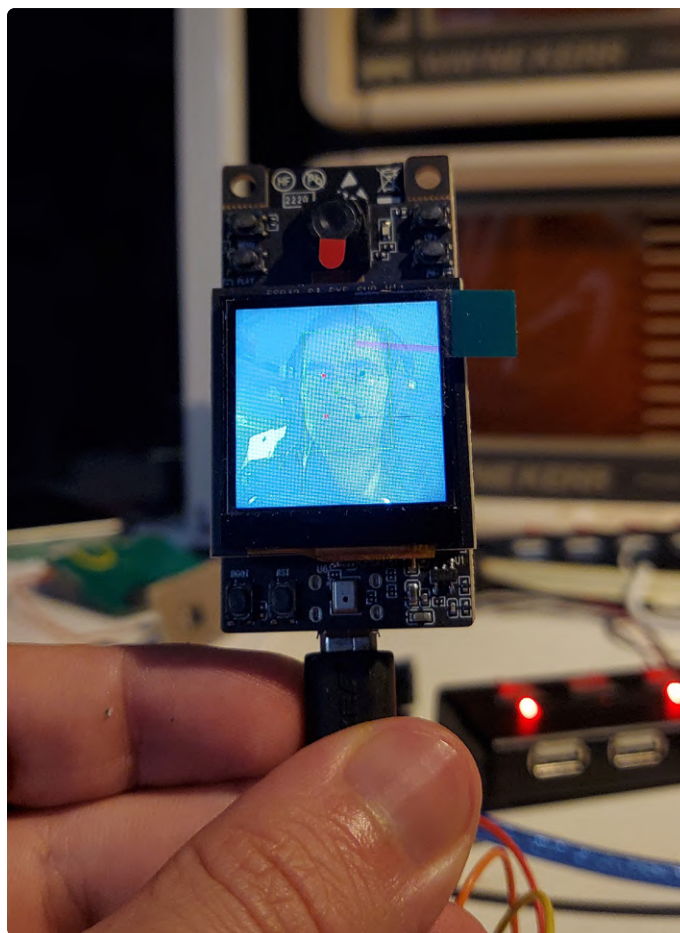
Beschikbaarheid van onderdelen en de ES8388

Het vinden van de halfgeleidercomponenten waar Espressif de voorkeur aan geeft, vereist soms een andere aanpak. In het geval van de ES8388-audiocodec kon ik geen van de traditionele distributeurs in de VS of Europa vinden die dit onderdeel op voorraad had. Een directe vraag aan de fabrikant wees me echter in de richting van Newtech Component Ltd., een distributeur die zo vriendelijk was om 20 monsters gratis op te sturen. Hiervoor is het raadzaam om gebruik te maken van een vracht- of pakketdienst die een adres kan leveren (in de buurt van de distributeur) waar de monsters naartoe kunnen worden gestuurd. De auteur gebruikt hiervoor vaak TipTrans.

Het board wordt geleverd met compleet geïnstalleerde standaard-firmware, waardoor eventuele 'hobbels' bij de installatieprocedure van het systeem vermeden worden. Bij het ESP32-S3-EYE-board met fabrieksinstellingen hoef je alleen maar een voeding via de micro-USB-poort aan te sluiten en een paar seconden te wachten. De gezichtsherkennings-applicatie wordt meteen gestart, zoals te zien in **figuur 2**. Het is de moeite waard om hier op te merken dat AI- en ML-algoritmen opmerkelijk robuust zijn met betrekking tot de kwaliteit van de opgenomen beelden. De foto in figuur 2 is gemaakt met de beschermende plasticfolie nog over de cameralens. Het verminderde contrast (en mijn ongeschoren gezicht) konden het AI-gezichtsherkenningsproces niet voor de gek te houden.

Back to the Future

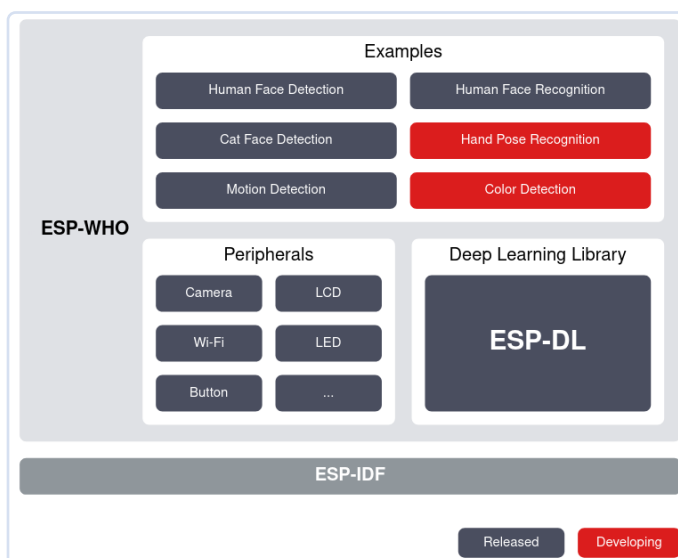
Misschien wil je je ESP32-S3-EYE op een bepaald moment terugzetten naar de 'fabrieksinstellingen'. Kant-en-klare .bin-bestanden zijn te vinden op de URL https://github.com/espressif/esp-who/tree/master/default_bin. Deze kunnen op dezelfde manier naar het board worden geflasht als elk ander binair bestand.



Figuur 2. De camera werkte, zelfs in mijn rommelige lab met de lenstape er nog op!

Het eerste experiment

Toegegeven, de bijna onbeperkte beschikbaarheid van durfkapitaal heeft geleid tot een stortvloed aan AI-frameworks. Het is logisch dat die bedrijven zich niet alleen op PC's richten, maar ook microcontrollers in hun portfolio van ondersteunde platforms opnemen.



Figuur 3. ESP-WHO maakt gebruik van verschillende andere onderdelen van het Espressif-ecosysteem (bron: [11]).

```
tamhan@TAMHAN18: ~
tamhan@TAMHAN18:~$ ls -l | grep "esp"
drwxrwxr-x 3 tamhan tamhan 4096 júl 13 2021 L
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 esprust
-rw-rw-r-- 1 tamhan tamhan 589 jan 15 2023 export-esp.sh
drwxrwxr-x 3 tamhan tamhan 4096 aug 7 12:36
```

Figuur 4. Verschillende versies van de ESP-IDF kunnen over het algemeen naast elkaar op een werkstation bestaan.

Een uitgebreid overzicht van de huidige marktsituatie zou buiten het bestek van dit artikel vallen, dus voor onze eerste stap werken we met het ontwikkelplatform *ESP-WHO* voor beeldverwerking, beheerd door Espressif. Dit is een framework dat 'geoptimaliseerd' is voor bepaalde soorten herkenning van bewegende beelden, en de structurele opzet wordt getoond in **figuur 3**.

De *ESP Deep Learning Library* (ESP-DL), beschikbaar op [3], is een belangrijke bron. Dit is in wezen een geoptimaliseerde bibliotheek die verschillende AI-technieken biedt en een aanzienlijke latency-reductie bereikt wanneer deze wordt gebruikt in combinatie met een hardware-versneller.

Een punt van aandacht is dat de WHO-component standaard alleen draait op versie 4.4 van de ESP-IDF en nog niet wordt ondersteund in versie 5.0. Voor mijn consultancywerk gebruik ik over het algemeen versie 4.4 voor projecten, maar ik heb meerdere varianten van de ESP32-ontwikkelomgeving op mijn computer geïnstalleerd, zoals je kunt zien in **figuur 4**.

De IDF-versie die in de volgende stappen wordt gebruikt, wordt als volgt geïdentificeerd:

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

We zijn nu klaar om de *ESP-WHO* code te downloaden. Dit wordt gedaan met dit **git**-commando in de commandoregel:

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

De stap **Compressing objects**: kan enige tijd in beslag nemen; repositories zijn soms vrij groot en traag om mee te werken. Nadat het werk voltooid is, is het aan te raden om de submodule op de volgende manier te initialiseren:

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

In de meeste gevallen zal het commando **git submodule update --recursive --init** geen informatie op het scherm tonen – en dat betekent dat het huidige image 'compleet' is.

```
tamhan@TAMHAN18:~/esp4/esp-who/examples/human_face_detection$ tree
.
├── lcd
│   ├── CMakeLists.txt
│   └── main
│       ├── app_main.cpp
│       ├── CMakeLists.txt
│       ├── partitions.csv
│       ├── sdkconfig.defaults
│       └── sdkconfig.defaults.esp32s3
├── README.rst
├── terminal
│   ├── CMakeLists.txt
│   └── main
│       ├── app_main.cpp
│       ├── CMakeLists.txt
│       ├── partitions.csv
│       ├── sdkconfig.defaults
│       ├── sdkconfig.defaults.esp32
│       ├── sdkconfig.defaults.esp32s2
│       └── sdkconfig.defaults.esp32s3
└── web
    ├── CMakeLists.txt
    └── main
        ├── app_main.cpp
        ├── CMakeLists.txt
        ├── partitions.csv
        ├── sdkconfig.defaults
        ├── sdkconfig.defaults.esp32
        └── sdkconfig.defaults.esp32s3

6 directories, 22 files
tamhan@TAMHAN18:~/esp4/esp-who/examples/human_face_detection$
```

Figuur 5. ESP-IDF interne varianten.

Een paar voorbeelden

De gezichtsherkenningssoftware *ESP-WHO* van Espressif biedt drie verschillende manieren voor de gebruikersinterface. Deze voorbeeldprojecten zijn te zien in **figuur 5**.

In de volgende stappen gebruiken we de LCD-variant van de voorbeelden `~/esp4/esp-who/examples/human_face_detection`. De berekende resultaten worden dan direct weergegeven op het kleine schermje dat op de ESP32 is gemonteerd. De terminalvariant gebruikt de *idf.py* monitor voor de communicatie, terwijl de *Web*-variant een webserver opzet.

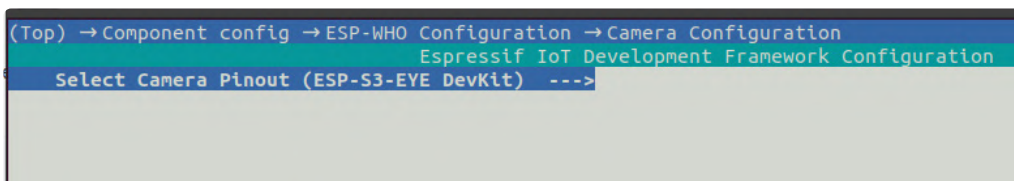
Eerst moet je het type ESP32-controller specificeren waarop je je wilt richten, met behulp van het volgende schema. Als je net als ik een ESP32-S3-EYE board gebruikt, wordt de string `esp32s3` doorgegeven met behulp van:

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

De eigenlijke broncode van het voorbeeld, die zich bevindt in het bestand `main/app_main.cpp`, is bijzonder eenvoudig. Om een beter inzicht te krijgen, zal ik het eerst in zijn geheel afdrucken voordat ik commentaar toevoeg (zie **listing 1**).

De kern van de *ESP-WHO* engine gebruikt het *Queue* kernel-object dat is geïmplementeerd in FreeRTOS, zoals beschreven in [4]. De twee statische definities leveren een *Input* en een *Output Queue*, die later worden gebruikt om de datastroom door het herkenningproces af te handelen.

De rest van de code die de ontwikkelaar heeft geschreven is voornamelijk gericht op het bouwen van een pijplijn via aanroepen van drie functies. Er zijn duidelijk overeenkomsten met het pijplijnmodel dat wordt gebruikt door het ESP-ADF audio framework [5].



Figuur 6. Als je een interne cameramodule gebruikt, moet je die hier opnieuw configureren!



Listing 1. In de wachtrij.

```
#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);
}
```

Om verder te gaan met de installatie, moet je in de eerste stap `idf.py menuconfig` openen om de bekende *menuconfig* configuratieomgeving te laden die in andere ESP32-projecten (en de Linux-kernel) wordt gebruikt. Navigeer daarna naar *Component config* → *ESP-WHO Configuration*. Zorg er in het veld *Camera Configuration* voor dat de camera van je evaluatieboard is voorgeconfigureerd zoals in **figuur 6**. Sla de zo gemaakte build-configuratie in *menuconfig* op en voer dan het gebruikelijke `idf.py build` commando uit om het compileren te starten. De aanvankelijke aanmaak van de image neemt wat meer tijd in beslag, omdat het framework ongeveer 1200 codebestanden moet compileren.

Voordat je het board flasht met `idf.py flash` en het poortnummer, moet je het board in bootloader-modus zetten. Gebruik de twee knoppen bij de USB-connector: houd BOOT ingedrukt terwijl je kort op RST drukt en laat dan beide knoppen los. Nu kun je zoals gebruikelijk het programma in de ESP32 flashen. Je kunt de toegang tot de bootloader-modus herkennen in *dmesg*, zoals te zien in **figuur 7**. Druk nu nogmaals op de RST-knop om de nieuwe variant van de bekende gezichtsherkenning (figuur 2) te starten.

Een snelle blik op de code

Het gebruik van *ESP-WHO* biedt een relatief eenvoudige introductie voor ontwikkelaars om zonder al te veel moeite te beginnen met het experimenteren met AI-toepassingen. Espressif levert ook de broncode voor sommige componenten [6], waardoor we de gezichtsherkenningsroutine kunnen bekijken.

Het gebruikt de al bestaande `HumanFaceDetectMSR01` detector, die enkele gewichten krijgt doorgegeven als onderdeel van de parametreer-procedure:

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

Het eigenlijke werk wordt gedaan door het ML-model in een eindeloze lus, die individuele frames ophaalt uit de wachtrij die eerder is aangemaakt:

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

Het aanroepen van de functie `detector.infer()` zorgt ervoor dat het eigenlijke inferentieproces wordt uitgevoerd. Als het geretourneerde *results*-object iets gedetecteerd heeft, roept het programma twee functies aan voor de uitvoer:

```
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
tamhan@TAMHAN18:~$
```

Figuur 7. Dit statusbericht geeft een succesvolle verbinding aan.

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

Figuur 8. Door de AI-versneller te activeren wordt de klus sneller geklaard (bron: [9]).

De rest van *ESP-WHO* bestaat grotendeels uit bestaande software-componenten zoals *ESP-Cam* [7], om de toegang tot de camera af te handelen. Ik zal niet verder ingaan op de *ESP-WHO* software omdat de gegeven voorbeelden voor zich spreken.

Dieper gaan met TensorFlow

Wie meer controle wil over het gedrag van zijn systeem, kan in de verleiding komen om helemaal vanaf nul te beginnen met coderen, maar dat is geen bijzonder praktische oplossing. Tenzij je er veel zorg aan besteedt, kan het systeem snel onbeheersbaar worden en leiden tot hogere onderhoudskosten gedurende de levensduur van de software. De *TensorFlow*-bibliotheek van Google, beschikbaar op [8], is een soort quasi-standaard geworden en is al enige tijd beschikbaar in geoptimaliseerde versies voor verschillende microcontrollers. Het is belangrijk om op te merken dat het op GitHub eigenlijk uit twee repositories bestaat: de reden voor deze eigenaardigheid is dat Google de distributie van algemene en leveranciersspecifieke code in de *TensorFlow*-bibliotheek halverwege de ontwikkeling van het framework heeft verschoven. De versie van de bibliotheek die we nodig hebben is te vinden op [9].

Omdat TensorFlow ook toegang heeft tot verschillende componenten van het ESP-IDF framework op de achtergrond, moet de implementatie vanaf GitHub ook plaatsvinden met de `--recursive` parameter die de opdrachtregel-tool wijst op de noodzaak om gerelateerde codelocaties aan te leveren en ervoor zorgt dat we de hoofdrepository en alle submodules krijgen:

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

Voor een eerste praktijktest kunnen we het voorbeeld `~/esp4/tflite-micro-esp-examples/examples/hello_world` gebruiken. Het gebruikt een ML-model dat is geoptimaliseerd door sinusfunctiewaarden te 'voorspellen', een alternatief voor het meer conventionele CORDIC-algoritme en een model dat wordt uitgevoerd met minimale invoergegevens. Voer in de volgende stap weer `idf.py set-target esp32s3` in om het projectmodel te optimaliseren voor het ESP32-S3 target. Afhankelijk van de versie van ESP-IDF die beschikbaar is op je werkstation of PC, kun je foutmeldingen tegenkomen tijdens het instellen van de parameters van de code die is gedownload uit de repository (*Invalid manifest...*). Deze fouten wijzen op gedeeltelijk verouderde componenten in de compilatie-toolchain. Om dit op te lossen kun je het volgende programma uitvoeren via de commandoregel:

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

Open vervolgens een *Nautilus*-venster dat naar de hoofdmap van het project-'skeleton' wijst door het volgende commando in te voeren. Het wissen van de *build*-map moet handmatig worden gedaan, zodat het `set-target` commando de ondersteunende bestanden voor de compilatie opnieuw kan aanmaken:

```
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
```

Let op de *ESP-NN* ingang in *Menuconfig*; deze staat het configureren van de DL bibliotheek toe. Het selecteren van de *Optimized* versions optie in de sectie *Optimization for neural network functions* is zeer aan te raden omdat dit de versneller activeert. De prestatiecijfers in **figuur 8** laten de aanzienlijke toename zien die kan worden verwacht als deze optie is ingeschakeld.

Het compileren en uitvoeren verloopt dan zoals verwacht. In **figuur 9** zie je de uitvoer van de berekende resultaten.

```
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
```

Figuur 9. Neurale netwerken werken ook met sinussen.

Analyse van de TensorFlow-code

Wees niet verbaasd als je chaotische beelden ziet op het ESP-EYE display – het ingebouwde LCD heeft zijn eigen framebuffer-controller, die gewoon het laatst opgeslagen beeld weergeeft totdat er een nieuwe update is.

Als je het bestand *main.cc* opent in een teksteditor naar keuze, vind je het volgende fragment:

```
#include "main_functions.h"
```

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

Eerste indrukken zijn hier niet misleidend. TensorFlow sluit nauw aan bij de Arduino-omgeving. De feitelijke implementatie van de *setup()* en *loop()* functies is te vinden in het *main_functions.cc* bestand. De *loop()*-methode is vooral interessant omdat deze verantwoordelijk is voor het uitvoeren van de payloads. De eerste taak is het genereren van invoergegevens die worden ingevoerd in de sinus-voorspeller:

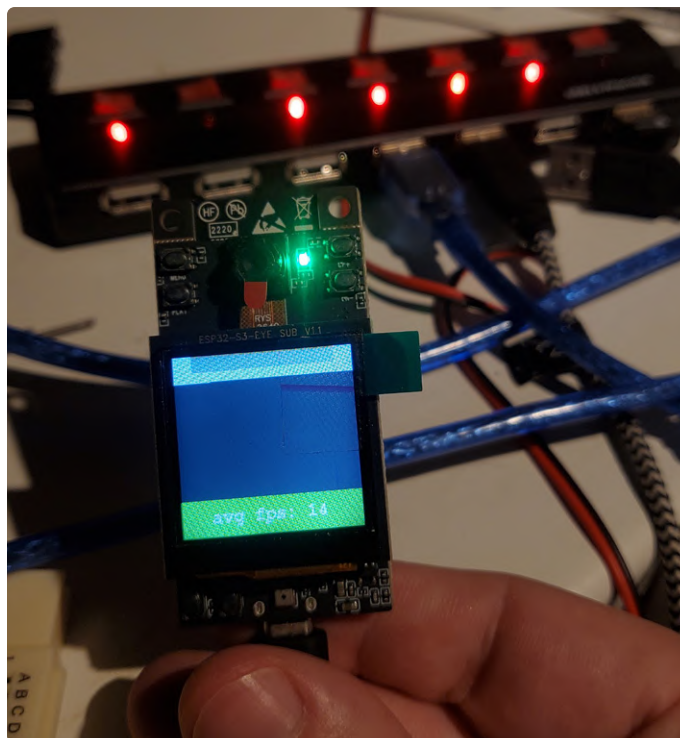
```
void loop() {  
    float position =  
        static_cast<float>(inference_count) /  
        static_cast<float>(kInferencesPerCycle);  
    float x = position * kXrange;
```

In de volgende stap wordt de informatie gekwantificeerd om deze beter 'verteerbaar' te maken voor het neurale netwerk. Dit is een populaire aanpak op het gebied van machine learning. Het normaliseren van alle invoerwaarden naar een bereik tussen 0 en 1 helpt om de complexiteit van het resulterende algoritme binnen de perken te houden:

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

De eigenlijke berekening en kwantificering vinden dan plaats in het volgende blok:

```
TfLiteStatus invoke_status =  
    interpreter->Invoke();  
if (invoke_status != kTfLiteOk) {  
    MicroPrintf("Invoke failed on x: %f\n",  
        static_cast<double>(x));  
    return;  
}  
int8_t y_quantized = output->data.int8[0];  
float y = (y_quantized -  
    output->params.zero_point) *  
    output->params.scale;
```



Figuur 10. De groene footer geeft een succesvolle herkenning aan.

Tot slot is er nog wat huishoudelijk werk nodig. Het is interessant om op te merken dat volgens de TensorFlow Micro-documentatie, de functie die verantwoordelijk is voor de daadwerkelijke gegevensuitvoer, *HandleOutput()*, moet worden geleverd door de ontwikkelaar:

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

Experimenten met meer geavanceerde modules

Als je de map met voorbeelden *~/esp4/tflite-micro-esp-examples/examples* van Google zorgvuldig analyseert, zul je zien dat er een spraakherkenningsvoorbeeld in zit met de naam *micro_speech* en zelfs een gezichtsherkenningsvoorbeeld met de naam *person_detection*. Om deze voorbeelden aan de praat te krijgen, moet je de bekende drietrap-procedure volgen van parameters instellen, *menuconfig*-instellingen controleren, compileren, gevolgd door het uploaden van de machinecode naar de ESP32.

Het is belangrijk op te merken dat deze geavanceerde voorbeelden meestal werken met behulp van commandoregel-communicatie. Als je schermuitvoer wilt toevoegen aan de personendetector, moet je het bestand *esp_main.h* als volgt aanpassen:

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

Na het hercompileren verschijnt het scherm van **figuur 10**. Het oplossen van het weergaveprobleem is het onderwerp van een ander artikel. Als de weergegeven voettekst groen wordt, betekent dit dat de herkenning geslaagd is.

Twee methodes

De experimenten die hier zijn uitgevoerd laten zien dat het ESP32-S3 platform in staat is om op verschillende manieren object- en gezichts-herkennings-payloads uit te voeren. Terwijl het ESP-WHO pad snelle en indrukwekkende resultaten oplevert, maakt de methode om het te integreren in het TensorFlow-ecosysteem het gebruik van geavanceerde technieken op het gebied van machinel learning mogelijk. 

230556-03 (vertaling: Hans Adams)



Gerelateerd product

> **ESP32-S3-EYE**
www.elektor.nl/20626



Vragen of opmerkingen?

Hebt u vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via tamhan@tamoggemon.com of naar de redactie van Elektor via redactie@elektor.com.

WEBLINKS

- [1] ESP32-S3-EYE distributeur zoeken: <https://oemsecrets.com/compare/ESP32-S3-EYE>
- [2] Schema van de ESP32-S3-EYE: <https://tinyurl.com/esp32eyeschematics>
- [3] ESP Deep Learning-bibliotheek: <https://github.com/espressif/esp-dl>
- [4] FreeRTOS Queues: <https://freertos.org/a00018.html>
- [5] T. Hanna, "Audio signalen en de ESP32", Elektor januari/februari 2023: <https://www.elektormagazine.nl/magazine/elektor-290/61475>
- [6] AI-voorbeeld ESP-WHO: <https://github.com/espressif/esp-who/tree/master/components/modules/ai>
- [7] ESP32 camerabesturing: <https://github.com/espressif/esp32-camera>
- [8] TensorFlow: <https://tensorflow.org>
- [9] TensorFlow Lite Micro: <https://github.com/espressif/tflite-micro-esp-examples>
- [10] ESP32-S3-EYE blokschema:
https://github.com/espressif/esp-who/blob/master/docs/en/get-started/ESP32-S3-EYE_Getting_Started_Guide.md
- [11] ESP-WHO op GitHub: <https://github.com/espressif/esp-who>

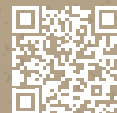


Arduino-ESP32

alle ondersteuning die u nodig hebt voor
ESP32, ESP32-S2, ESP32-S3 en ESP32-C3

Arduino-ondersteuning voor de ESP32, ESP32-S2, ESP32-S3 en ESP32-C3 IC's is in deze repository beschikbaar. U kunt de standaard Arduino IDE of PlatformIO IDE gebruiken voor Arduino-gebaseerde applicatie-ontwikkeling voor deze IC's. Deze repository bevat veel nuttige bibliotheken, waaronder veelgebruikte device drivers, ondersteuning voor WiFi-, BLE- en ESP-NOW-protocollen en high-level functionaliteit zoals ESP-Insights en ESP-RainMaker. Al deze bibliotheken bevatten voorbeelden die de functionaliteit kunnen demonstreren. Ook worden veel development boards ondersteund die op de genoemde IC's gebaseerd zijn.

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



Knoopcel-schakelaar met ESP32-C2

prestatiegericht ontwerpen

Li Junru en Zhang Wei, Espressif

De ESP32-C2-gebaseerde knoopcel-schakelaar is een toepassing voor de smart home-markt. Hij biedt directe communicatie met systemen die zijn uitgerust met ESP, stabiele bidirectionele communicatie, compacte afmetingen en een batterij-levensduur tot vijf jaar. Dit artikel behandelt de software- en hardware-implementatie en laat de voordelen zien voor moderne smart homes en IoT-toepassingen.



Figuur 1. De op de ESP32-C2 gebaseerde knoopcel-schakelaar in zijn behuizing.

Met de snelle ontwikkeling van de smart home-markt is er een toenemende vraag naar efficiënte draadloze schakelaars met een laag stroomverbruik. Dit artikel presenteert een baanbrekende toepassing, een ESP32-C2-gebaseerde knoopcel-schakelaar (**figuur 1**), gericht op het aanpakken van problemen zoals een vertraagde responsie en de noodzaak van extra gateways, waar andere draadloze schakeloplossingen op basis van technologieën als Bluetooth LE en ZigBee vaak mee te maken hebben.

De ESP32-C2-gebaseerde knoopcel-schakelaar heeft verschillende voordelen ten opzichte van andere draadloze schakelaar-varianten:

- rechtstreekse communicatie met slimme lampen of wandschakelaars die zijn uitgerust met ESP-chips, zodat er geen

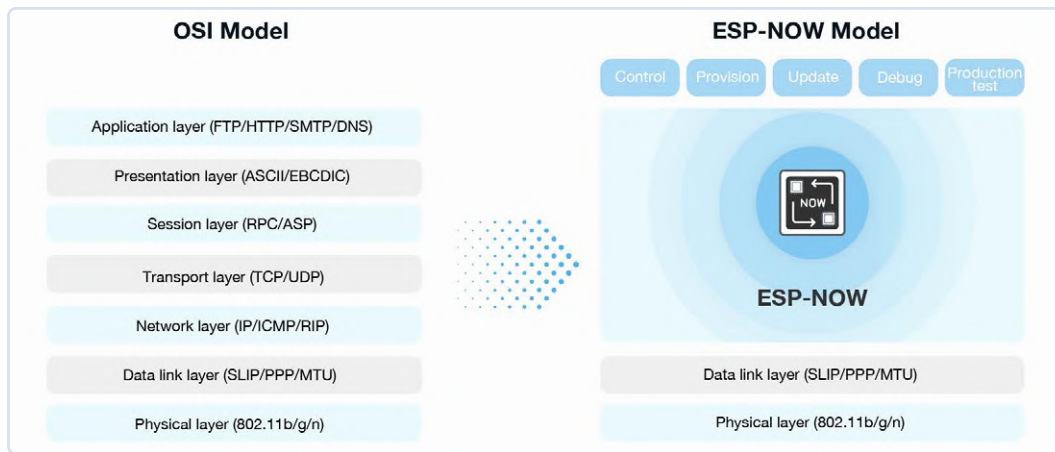
extra gateway nodig is;

- stabiele tweerichtingscommunicatie met een hoge betrouwbaarheid van de WiFi packet-transmissie;
- de schakelaar met knoopcel-voeding heeft minimale afmetingen en ondersteunt veelzijdige productvormen zoals zelfklevende schakelaars, schakelaars met meerdere toetsen, aanraakschakelaars en draaischakelaars;
- de ESP32-C2 is volledig uitgeschakeld als hij niet wordt gebruikt, waardoor een enkele CR2032-knoopcel tot 5 jaar meegaat (uitgaande van een gebruik van 10 keer per dag).

In dit artikel gaan we dieper in op de uitgebreide toepassingsmogelijkheden van de ESP knoopcel-schakelaar en laten we zien hoe

goed deze voldoet aan de eisen van moderne smart home-technologie.

Knoopcel-batterijen, zoals de veelgebruikte CR2032, zijn een populaire energiebron voor IoT-apparaten. Knoopcel-batterijen staan bekend om hun compacte ontwerp en lichte gewicht en zijn zeer geschikt voor kleine elektronische apparaten, zonder dat die veel groter of zwaarder worden. Door hun relatief hoge energiedichtheid kunnen ze meer energie opslaan in een kleiner volume, wat resulteert in een langere gebruiksduur. Bovendien vertonen knoopcellen een langzamere zelfontlading, wat betekent dat ze hun lading behouden, zelfs als ze langere tijd niet gebruikt worden. Knoopcel-batterijen leveren een relatief stabiele spanning tijdens het ontladen en spelen daarom een essentiële rol in de normale werking van apparaten.



Figuur 2. Het ESP-NOW model combineert de bovenste vijf lagen van een OSI-model tot één laag.

Door hun ontwerp leveren ze echter meestal een kleinere stroom, waardoor ze ongeschikt zijn voor apparaten met een groot vermogen. In de diverse toepassingsgebieden worden WiFi IoT-apparaten op grote schaal gebruikt, waarbij ze profiteren van de grootschalige dekking van WiFi-netwerken. Doordat traditionele bekabelde verbindingen tussen apparaten niet meer nodig zijn, maakt WiFi-technologie een flexibelere en handigere inzet en beheer van IoT-apparaten mogelijk. Bovendien ondersteunt WiFi-technologie gelijktijdige verbindingen met meerdere apparaten, waardoor het bijzonder waardevol is voor IoT-scenario's met veel onderling verbonden apparaten. De markt biedt een overvloed aan apparaten en componenten die WiFi ondersteunen, waardoor de ontwikkelings- en productiekosten van IoT-apparaten aanzienlijk worden verlaagd. Toch is het belangrijk om te bedenken dat WiFi-apparaten relatief meer energie verbruiken dan andere low-power draadloze technologieën. Daarom zijn bepaalde IoT-apparaten die afhankelijk zijn van batterijen soms minder geschikt voor WiFi. De ESP32-C2 heeft bijvoorbeeld een maximaal stroomverbruik van 370 mA bij het zenden en maximaal 65 mA bij ontvangst tijdens RF-bedrijf.

De verbruikskarakteristieken van WiFi-apparaten vormen twee grote uitdagingen voor low-power toepassingen. Aan de ene kant maakt de aanzienlijke stroom bij het ontvangen het moeilijk voor de chip om continu in ontvangstmodus te werken. Aan de andere kant kan de plotselinge hoge stroom tijdens het verzenden van packets de stabiliteit van de voedingsspanning voor de chip beïnvloeden, wat kan leiden tot een chip-reset. In dit artikel presenteren we een knooppcel-schakelaar gebaseerd op de ESP32-C2 die deze uitdagingen aanpakt door gebruik te maken van een weloverwogen combinatie van software en hardware, wat resulteert in een indrukwekkende levensduur van de batterij.

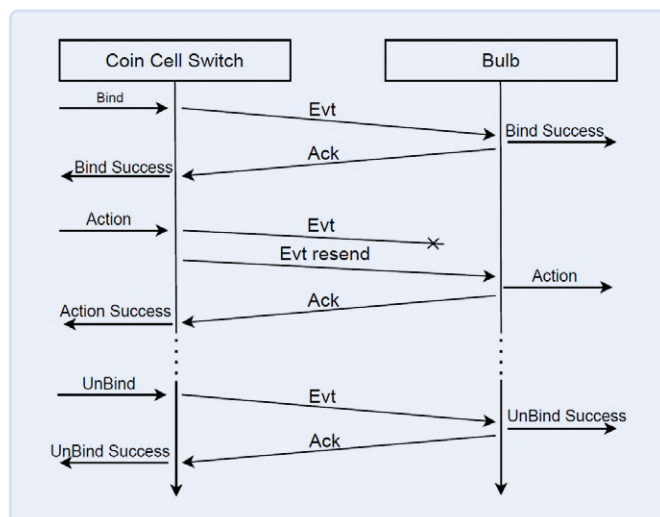
Het artikel is verdeeld in drie secties. Ten eerste gaan we in op de software-implementatie, waarbij we gedetailleerd inzicht geven in de methodologie van het programma en de resultaten die daaruit voortvloeien. Ten tweede onderzoeken we de hardware-implementatie, waarbij we richtlijnen geven voor de keuze van onderdelen. Tenslotte evalueren we in het experimentele gedeelte de werkelijke energieprestaties en de *packet latency* van het apparaat, zodat de lezer deze toepassing goed kan beoordelen.

Ontwerp van het programma

Keuze van de protocollaag: de eerste cruciale beslissing was het kiezen van de juiste protocollaag. Normaal gesproken werkt WiFi in een 'connected' modus, waarbij apparaten een constante verbinding met het netwerk onderhouden, tenzij ze opzettelijk worden losgekoppeld of buiten bereik zijn. Deze continue verbinding zorgt ervoor dat apparaten direct beschikbaar zijn voor communicatie zonder dat ze telkens opnieuw verbinding moeten maken. Het instandhouden van deze verbinding kost echter veel energie. Om deze uitdaging aan te gaan, hebben we

gekozen voor een lichtere communicatie-oplossing. In de software hebben we dit bereikt door berichten uit te zenden op de datalink-laag, zodat de ontvangende apparaten de informatie gemakkelijk kunnen ophalen en doorsturen. Daarnaast implementeerden we een vooraf ingestelde ontvangsttijd om het stroomverbruik zo laag mogelijk te houden. Voor dit doel bleek het ESP-NOW protocol, ontwikkeld door Espressif Systems, ideaal. ESP-NOW is een draadloos communicatie-protocol gebaseerd op de datalink-laag. Het vereenvoudigt de bovenste vijf lagen van het OSI-model tot één laag (figuur 2).

Apparaten worden geïdentificeerd door een uniek MAC-adres, waardoor het niet meer nodig is om gegevens achtereenvolgens door complexe lagen te sturen, zoals de netwerklaag, transportlaag, sessielaag, presentatielaag en toepassingslaag. Het elimineert ook de noodzaak om headers toe te voegen en te verwijderen op elke laag, waardoor vertragingen door netwerkcongestie en vertragingen door het verlies van packets sterk worden gereduceerd, wat resulteert in hogere reactiesnelheden. Bovendien kan ESP-NOW naast WiFi en Bluetooth LE bestaan en ondersteunt



Figuur 3. Het bevestigingsmechanisme geïmplementeerd in de applicatielaag.

Tabel 1. Energieverbruik vóór optimalisatie van het initialisatieproces.

Actie	Duur (ms)	Gemiddeld vermogen (mW)	Energieverbruik (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

Tabel 2: Energieverbruik na optimalisatie van het initialisatieproces.

Actie	Duur (ms)	Gemiddeld vermogen (mW)	Energieverbruik (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

het meerdere SoC-series van Espressif met WiFi-functionaliteit.

Workflow: om een betrouwbare communicatie te garanderen, hebben we een bevestigingsmechanisme geïmplementeerd in de applicatielaag (**figuur 3**). Het communicatieproces begint met de knoopcel-schakelaar die de communicatie tot stand brengt. Na ontvangst van een bericht antwoordt de ontvangende kant met een bevestiging (ACK), waarmee de succesvolle ontvangst van het bericht wordt aangegeven en de communicatie wordt afgesloten. Als de bevestiging niet binnen een bepaalde time out-periode wordt ontvangen, probeert de switch het bericht nog enkele keren opnieuw te verzenden. Om een gestroomlijnd protocol te behouden, worden alle packets verzonden met behulp van het ESP-NOW protocol in de datalink-laag. Het verbinden en loskoppelen van apparaten wordt bereikt door bidirectionele MAC-adresverificatie, waardoor de veiligheid en de eenvoud van het protocolontwerp worden gewaarborgd.

Kanaalwissel-strategie: de kanalen voor ontvangen en verzenden van het ESP-NOW protocol volgen het kanaal van het op dat moment verbonden AP (Access Point) van het apparaat. De knoopcel-schakelaar kan alleen communiceren met het aangestuurde apparaat als ze zich op hetzelfde kanaal bevinden. Als het aangestuurde apparaat al verbonden is met het gespecificeerde WiFi AP, volgt diens operationele kanaal veranderingen in het kanaal van het AP. De knoopcel-schakelaar moet het huidige operationele kanaal van het aangestuurde station bepalen. In een omgeving waar de omringende netwerksituatie relatief stabiel is, verandert het kanaal van het AP niet vaak.

Daarom zendt het apparaat eerst op het kanaal waar het de laatste keer met succes gegevens heeft verstuurd. Als meerdere verzendpogingen mislukken en er wordt vastgesteld dat het ontvangststation is overgeschakeld op een nieuw operationeel kanaal, dan worden alle resterende WiFi-kanalen doorlopen.

Stroommodulatie bij packet-transmissie:

tijdens het verzenden van WiFi-packets kan de momentane stroom meer dan 300 mA bedragen, veel meer dan de maximale stroom die knoopcel-batterijen kunnen leveren. Om dit probleem te ondervangen, hebben we een intermitterende packet-transmissie strategie toegepast. Telkens na het verzenden van een packet wordt een bepaald interval ingelast, waarin de chip in een light sleep-modus gaat. In deze toestand verbruikt de chip een stroom van slechts enkele tientallen microampère en wordt de ontlading van de knoopcelbatterij voornamelijk gebruikt om de condensatoren op te laden voor het verzenden van het volgende packet.

Laten we aannemen dat de gemiddelde stroom tijdens de pakketoverdracht I_0 is, met een duur van t_0 , en de gemiddelde stroom in de sleep-modus I_1 is, met een duur van t_1 . De totale gemiddelde stroom tijdens de packet-transmissie kan dan worden berekend volgens de formule:

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

Door de timing van I_0 en I_1 zorgvuldig te moduleren, kunnen we ervoor zorgen dat de gemiddelde stroom onder de veilige uitgangsstroom van de knoopcel blijft. Deze strategie helpt om een efficiëntere en energiebesparende werking van de knoopcel-schakelaar te bereiken.

rende werking van de knoopcel-schakelaar te bereiken.

Optimalisatie van het initialisatieproces: het initialisatieproces van een WiFi-chip bestaat uit verschillende fasen, van het opstarten tot de voltooiing van de signaaloverdracht. We voerden een grondige analyse uit van elke fase en de duur daarvan. Standaard omvat het opstarten van de chip de boot-, WiFi-initialisatie- en WiFi-startprocessen. Van deze processen neemt de boot-initialisatie de meeste tijd in beslag en resulteert het WiFi-startproces in een piek in het stroomverbruik gedurende korte tijd. Om het stroomverbruik te verminderen, hebben we de volgende optimalisaties doorgevoerd:

1. **Niet-essentiële logging uitschakelen:** we hebben onnodige uitvoer van log-data uitgeschakeld om het stroomverbruik tijdens het opstartproces te minimaliseren.
2. **Flash-verificatie:** we hebben flash-verificatie uitgeschakeld, omdat het niet essentieel was voor de werking van de applicatie.
3. **WiFi kalibratie-info:** om te voorkomen dat WiFi vaak gekalibreerd moet worden, hebben we de WiFi kalibratie-info opgeslagen in niet-vluchtig geheugen.

Werp eens een blik op **tabel 1** en **tabel 2**. Door het implementeren van deze optimalisaties is het gelukt om het totale energieverbruik (Boot + Wi-Fi Init + Wi-Fi Start) tijdens het initialisatieproces te verlagen van 46,39 mJ naar 5,61 mJ. Bovendien daalde de initialisatietijd van 536,5 ms naar 63,19 ms. Raadpleeg voor meer gedetailleerde configuratie-informatie de ESP-NOW coin_cell_demo [1].

Ontwerp van de schakeling

De ESP32-C2 heeft een werkspanning van 3,3 V nodig, wat hoger is dan de spanning die de knoopcel levert. Daarom moet er een boost-schakeling worden ontworpen om de spanning te verhogen. De stabiliteit van de voeding heeft een directe invloed op de prestaties van de packet-transmissie en de algehele stabiliteit van het apparaat. Een goed



De ESP32-C2 knoopcel-schakelaar biedt een handige en veelzijdige manier om slimme apparaten te bedienen.

ontworpen spanningsregelaar kan de RF-prestaties en de levensduur van de batterij van het apparaat verbeteren. Het is essentieel om bij het ontwerpen van de schakeling om rekening te houden met zowel de productiekosten als de gewenste prestaties.

De boost-converter moet zorgvuldig worden ontworpen om een efficiënte conversie met minimaal vermogensverlies te garanderen. Bovendien moet hij een stabiele en betrouwbare voeding leveren voor de ESP32-C2, zodat deze optimaal kan presteren in zowel de actieve als de slaapmodus. Daarnaast moet het ontwerp rekening houden met factoren zoals stroomverbruik, warmtedissipatie en rendement om de juiste balans te vinden tussen prestaties en stroomverbruik.

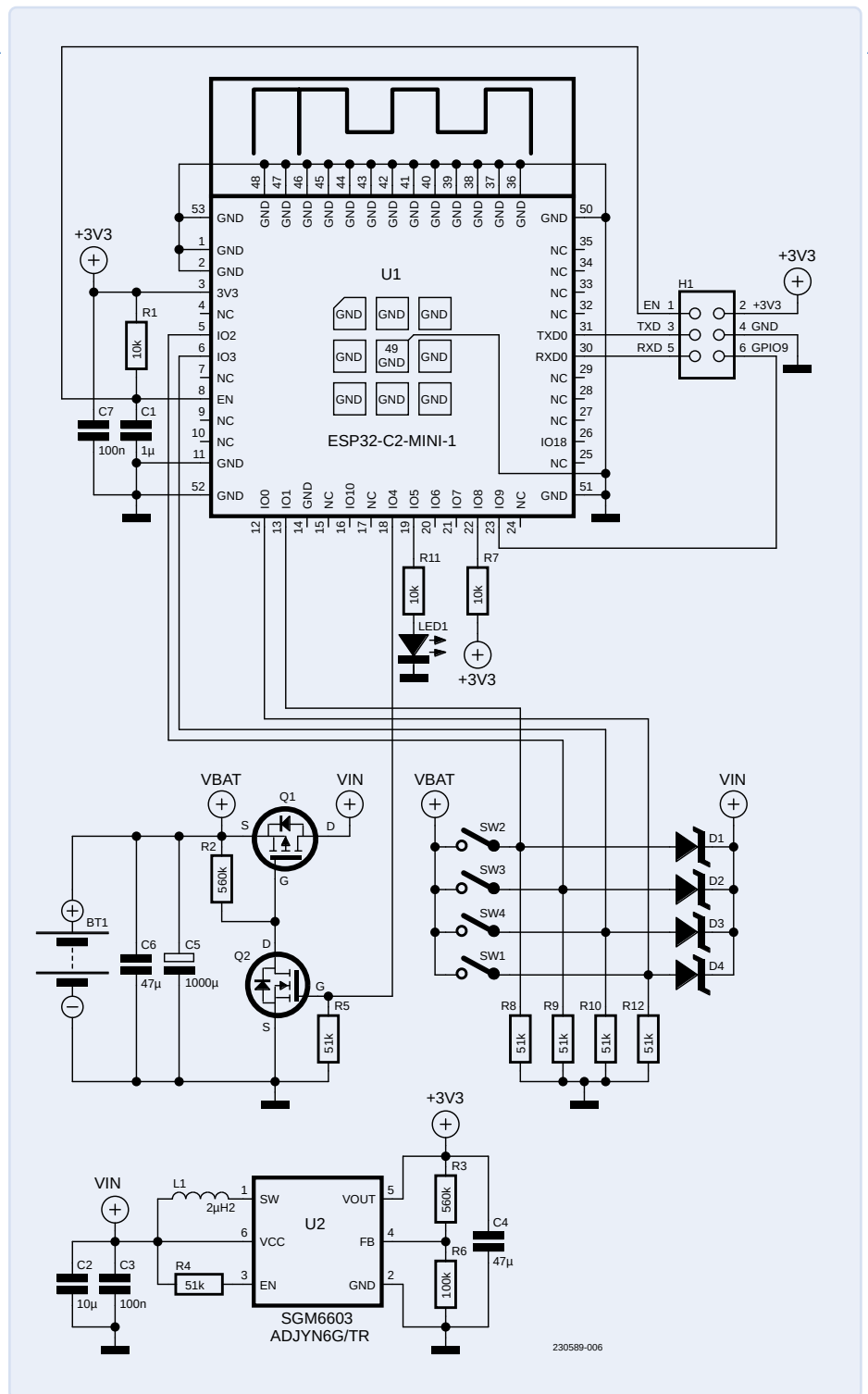
Het algemene doel van het ontwerp is om een robuuste en kosteneffectieve oplossing te realiseren die voldoet aan de voedingseisen van de ESP32-C2 en tegelijkertijd de prestaties van het apparaat en de levensduur van de batterij optimaliseert. Samenwerken met ervaren hardwaretechnici en gebruikmaken van de juiste componenten en technieken kan leiden tot een succesvol ontwerp van de schakeling dat voldoet aan de specifieke behoeften van de WiFi knoopcel-schakelaar.

De schakeling als geheel

Keuze van de converter: de knoopcel-batterij kan worden beschouwd als een spanningsbron met een snel toenemende inwendige weerstand tijdens het ontladen. Aanvankelijk is de inwendige weerstand ongeveer 10 Ω , maar deze kan oplopen tot honderden ohms naarmate hij het einde van de ontladcyclus wordt benaderd. De ESP32-C2 heeft als uitvoerapparaat een voeding nodig die een stroom van 500 mA of meer kan leveren. De voedingsrimpel kan de RF transmissie-prestaties sterk beïnvloeden. Bij het meten van de voedingsrimpel is het cruciaal om deze te testen onder normale packet-transmissie omstandigheden.

De voedingsrimpel kan afhankelijk van de veranderingen in de voedingsmodus variëren. Een groter packet-zendvermogen kan leiden tot grotere rimpel. Om de invloed van de hoge interne weerstand van de knoopcel-batterij te beperken, moet de minimale ingangsspanning van de boost-converter zo laag mogelijk zijn, met behoud van een hoog rendement.

Voor dit ontwerp is, zoals je kunt zien in het algemene schema van **figuur 4**, de SGM6603 gekozen als boost-chip. Deze heeft een



Figuur 4. Schema van de knoopcel-schakelaar.

minimale ingangsspanning van slechts 0,9 V en een maximale schakelstroom van 1,1 A, waardoor hij geschikt is om de spanning van de knoopcel-batterij efficiënt op te voeren om te voldoen aan de voedingseisen van de ESP32-C2.

Keuze van de condensatoren: er zijn twee sets condensatoren die met de voeding te maken hebben – de condensatoren voor en na de boost-converter. De condensatoren

na de boost converter zijn meestal parallel gekoppeld aan de voedingsingang van de WiFi-module en stabiliseren de uitgangsspanning en verminderen de spanningsval tijdens de packet-transmissie. Grotere condensatoren resulteren in een meer geleidelijke spanningsverandering van de voeding van de chip. Aan de andere kant volgt de spanning over deze condensatoren de voedingsspanning van de chip. Wanneer de WiFi-chip wordt



Figuur 5: Oscilloscoop-screenshot van spanning en stroom tijdens de verzending van een standaard datapakket.

gevoed, worden de condensatoren opgeladen en wanneer de chip wordt uitgeschakeld, wordt deze set condensatoren volledig ontladen. Daarom kan het gebruik van te grote condensatoren leiden tot verminderde systeemefficiëntie. Het is essentieel om een balans te vinden tussen de waarde van de condensator en de efficiëntie van het systeem. De condensatoren voor de boost-converter staan parallel aan de batterij. Hun belangrijkste functie is het verminderen van de piekstroom van de batterij. Tijdens een piekstroom-periode werken de condensatoren als de primaire stroombron, terwijl tijdens perioden van lage stroom de batterij de primaire stroombron wordt die de condensatoren oplaadt. Als de boost-converter niet in bedrijf is, wordt het enige stroomverbruik in de schakeling veroorzaakt door de lekstroom van de condensator. Met het oog op de omvang en lekstroom zijn solid-state elektrolytische condensatoren en aluminium elektrolytische condensatoren de ideale keuze. Zo heeft een elektrolytische condensator van 1000 μF meestal een lekstroom van ongeveer 1 μA bij een spanning van 3 V.

Ontwerp geregelde aan/uit-schakelaar: in toepassingen waar de operationele levensduur van het apparaat in jaren wordt gemeten, wordt de stand-by stroom (lekstroom) tijdens sleep modi een belangrijke factor die de totale levensduur van het apparaat beïnvloedt. Om dit probleem aan te pakken, bevat het huidige ontwerp een besturing van de aan/uit-schakelaar die bestaat uit de twee MOSFET's Q1 en

Q2 in het schema van **figuur 4**. Deze schakelaar zorgt ervoor dat de verbinding van de chip met de batterij gemaakt of verbroken kan worden, waardoor de voedingsmodule en RF-module effectief worden losgekoppeld van de batterij als het apparaat niet in gebruik is. Als het apparaat moet worden aangezet, wordt er op een knop gedrukt, waardoor de chip verbonden wordt met de voeding om te worden ingeschakeld. Tegelijkertijd gebruikt de chip ADC-spanningsbemonstering om vast te stellen welke knop is ingedrukt.

Het gebruik van deze geregelde aan/uit-schakelaar zorgt voor efficiënt energiebeheer, waardoor onnodig stroomverbruik tijdens inactieve perioden wordt verminderd en de algehele levensduur van de batterij van het apparaat wordt verlengd. Door de voeding en RF-modules volledig los te koppelen als ze niet worden gebruikt, wordt de stand-by stroom van het apparaat geminimaliseerd, waardoor de levensduur en bruikbaarheid in verschillende elektronische toepassingen voor consumenten worden geoptimaliseerd. Naast de bovengenoemde componenten bevat de schakeling ook een minimaal ESP32-C2 systeem en een LED-indicator.

Evaluatie batterij-levensduur

Na de laatste optimalisaties werd een typische packet-transmissie geanalyseerd en werden de spanning en stroom aan de ingangszijde van de boost-converter vastgelegd (**figuur 5**). Op basis van de verkregen informatie bedraagt de volledige packet-transmissietijd voor het

apparaat 240 ms en de gemiddelde stroom tijdens bedrijf 25,8 mA..

Om de levensduur van de batterij van het apparaat nauwkeurig te kunnen inschatten, rekening houdend met de onzekerheid in de efficiëntie van de boost-converter onder dynamische belasting, werd een praktische evaluatie uitgevoerd. Bij deze evaluatie werd het totale aantal verzonden packets van het apparaat getest in de gespecificeerde werkmodus. Elke cyclus werd gemeten vanaf het moment dat het apparaat werd geactiveerd (knop ingedrukt) tot de succesvolle transmissie van een signaal en de ontvangst van een bevestiging – dat omvat een complete activiteitscyclus.

Na het testen bleek dat een enkele CR2032 knoopcel-batterij ongeveer 65.000 transmissiecycli kon ondersteunen. Bovendien werd een stand-by stroom van het apparaat gemeten van slechts 1 μA . Ervan uitgaande dat het apparaat 10 keer per dag pakketten verzendt, bedraagt de praktische levensduur van de CR2032-knoopcel dus ongeveer 5 jaar. Deze evaluatie bewijst het efficiënt energiebeheer en het lage verbruik van het apparaat, waardoor het geschikt is voor langdurig gebruik in elektronische toepassingen voor consumenten. Met de geoptimaliseerde voedingsregeling en het geavanceerde ontwerp van de schakeling bereikt het apparaat een indrukwekkende batterijlevensduur, waardoor een betrouwbare en langdurige werking in diverse IoT- en smart home-toepassingen wordt gegarandeerd.

Slotopmerkingen

De ESP32-C2 knoopcel-schakelaar biedt een handige en veelzijdige manier om slimme apparaten te bedienen. Door gebruik te maken van het flexibele energiebeheer en de protocollen van de ESP32-C2, realiseert deze oplossing een knoopcel-schakelaar op basis van het WiFi-protocol, waardoor eenvoudige communicatie mogelijk is met andere apparaten die

zijn uitgerust met ESP-chips. Als onderdeel van onze toekomstige plannen willen we deze technologie integreren in Matter (voorheen bekend als Project CHIP) standaarden, waardoor flexibele besturing van meerdere apparaten en samenwerking met conventionele apparaten met voeding mogelijk wordt. Vergeet niet om de GitHub-repository [2] van Espressif te bekijken voor meer open-source

demo's en informatie over ESP-IoT-Solution en ESP-NOW.

De knoopcel-schakelaar applicatie verbetert het gemak en de efficiëntie op het gebied van smart homes en IoT. Dankzij het geoptimaliseerde ontwerp en efficiënte energiebeheer gaat de batterij lang mee, waardoor gebruikers slimme apparaten betrouwbaar en duurzaam kunnen bedienen. ◀

230589-03 (vertaling: Hans Adams)



Over de auteurs

Zhang Wei is momenteel een senior staff application engineer bij Espressif Systems. Hij is een doorgewinterde software-engineer met meer dan twintig jaar ervaring in embedded systemen, draadloze netwerken en IoT-ontwikkeling en houdt ervan om eenvoudige oplossingen te vinden voor complexe problemen. Hij heeft een bachelor-diploma in elektrotechniek en computertechniek en een master-diploma in kennistechniek van de National University of Singapore en heeft zijn expertise verrijkt via functies bij technische bedrijven als STMicroelectronics, Greenwave Systems en dormakaba digital. Buiten zijn werk houdt Zhang Wei van voetbal en reizen.



Li Junru werkt als AE engineer bij Espressif Shanghai. Embedded systemen zijn altijd zijn passie geweest. Hij is op een missie om enthousiastelingen te helpen hun creativiteit te ontplooiën met ESP32. Li Junru's inspirerende boodschap: "Laten we samenwerken en samen embedded systeemontwikkeling boeiend maken!"

WEBLINKS

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

Vragen of opmerkingen?

Hebt u vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via zhang.wei@espressif.com of lijunru@espressif.com of naar de redactie van Elektor via redactie@elektor.com.



Gerelateerde producten

- **Espressif ESP32 range**
www.elektor.nl/espressif
- **Getting Started with ESPHome**
www.elektor.nl/19738



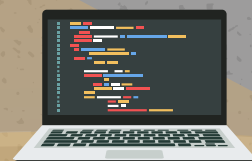
ESP-ADF

Espressif Audio Development Framework

Als u een apparaat bouwt waarmee u audio moet opnemen of afspelen, dan is ESP-ADF (Audio Development Framework) de SDK waar u moet kijken. ESP-ADF biedt niet alleen basisondersteuning voor audio-pipelining, maar ook verschillende audio-encoders, -decoders, audiocontainer-parsers, equalizers en downmixers.

Daarnaast worden verschillende high-level protocollen ondersteund, zoals DLNA, RTSP, RTCP en Bluetooth A2DP playback, inclusief de bijbehorende voorbeelden. Er zijn verschillende development kits die audio-ondersteuning bieden. Bekijk de boards uit de ESP32-S3-Korvo- en ESP32-Lyra-serie die prototyping gemakkelijker maken.

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





stuwt het slimme huis voorwaarts

ontsluit het potentieel van Smart Home IoT

Kedar Sovani, Espressif

Consumenten en ontwikkelaars raakten vaak gefrustreerd door connected producten – in de meeste gevallen merkgebonden apparaten die eigen apps nodig hebben voor configuratie en bediening en die noch een consistente gebruikerservaring noch onderlinge interoperabele communicatie ondersteunen. Het Matter-protocol biedt een veilige manier voor de eindgebruiker om connected apparaten te configureren, te lokaliseren en te bedienen. Sinds de lancering in oktober 2022 zijn meer dan duizend producten van een groot aantal verschillende bedrijven Matter-gecertificeerd.

Tegen het einde van 2022 werd de Matter smart home-standaard gelanceerd – het resultaat van jarenlange gezamenlijke ontwikkeling door belangrijke spelers in de industrie. In de afgelopen maanden hebben we gezien dat er talloze apparaten zijn gelanceerd die met deze universele interoperabele standaard werken. In dit artikel kijken we naar de voordelen die Matter biedt, hoe ver het inmiddels is gekomen en waar het toe kan leiden.

Het tijdperk vóór Matter

Als consument – maar ook als ontwikkelaar – van connected producten ken je wellicht de frustratie die met hiermee gepaard gaat. In de meeste gevallen betrof dat merkgebonden apparaten die eigen apps nodig hebben voor configuratie en bediening en die noch een consistente gebruikerservaring noch onderlinge interoperabele communicatie ondersteunden. Het enige gemeenschappelijke kenmerk was de spraaksturing van verschillende ecosystemen die ingebouwde ondersteuning hadden voor deze merkgebonden protocollen.

Aan de kant van de ontwikkelaars/fabrikanten van apparaten betekende dit dat de totale kosten om een geweldig product te implementeren veel hoger waren, omdat ontwikkelaars compliantie met en certificeringen van verschillende organisaties nodig hadden.

Matter verschijnt ten tonele

Verschiedende belanghebbenden in de smart home-industrie realiseerden zich dat deze problemen de waarde en groei van het slimme huis benadeelden. Deze industriebrede samenwerking culmineerde in de lancering van de Matter-standaard eind 2022.

Wat biedt Matter precies? Matter biedt een veilige manier waarop connected apparaten geconfigureerd, ontdekt en bediend kunnen worden door de eindgebruiker.



Sinds de lancering in oktober 2022 zijn meer dan 1000 producten Matter-gecertificeerd, terwijl de Connectivity Standards Alliance (CSA) kan bogen op 300+ bedrijven die nu deelnemen aan de gezamenlijke inspanning om Matter te bouwen en te implementeren. De grote schaal waarop dit gebeurt heeft ertoe geleid dat meerdere connected apparaten van huis uit Matter ondersteunen. Apparaten die geschikt zijn voor Matter kunnen worden geconfigureerd en bediend door 'dingen' die gewoonlijk Matter-controllers worden genoemd (telefoons, speakers, displays). Meerdere ecosystemen, zoals iOS, Android, Alexa en SmartThings, bezitten standaard al Matter-controllerondersteuning. Dit stelt consumenten in staat om te beginnen met het configureren en bedienen van Matter-enabled connected apparaten zonder dat ze een aparte Matter-controller hoeven aan te schaffen of een aparte telefoon-app hoeven te installeren (**figuur 1**).

Veiligheid en privacy

Voor veel connected apparaten die we vandaag de dag gebruiken, willen we als consument informatie over de beveiligingspraktijken en -principes die zijn toegepast tijdens de ontwikkeling, en vervolgens het beheer, van het betreffende product.

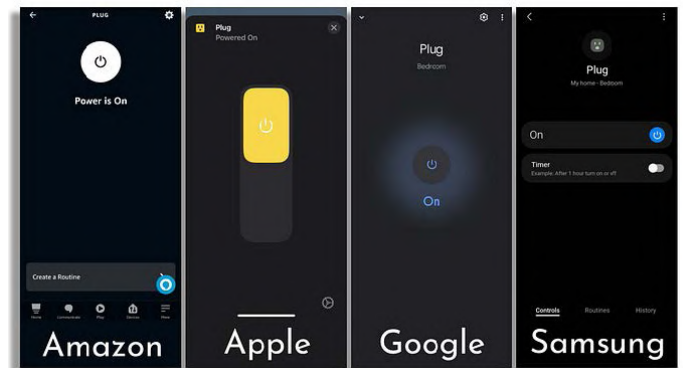
Beveiliging en privacy vormden de ruggengraat bij het opstellen van de Matter-specificatie. Omdat een groot aantal organisaties (met grote volumes en jarenlange ervaring in de smart home-industrie) hebben samengewerkt aan deze specificaties, hebben ze ervoor gezorgd dat er geen middel onbeproefd is gelaten om de gebruikers het beste van het beste te bieden. Gebruikers van Matter-producten kunnen er zeker van zijn dat elk aspect van het gedrag van het product, van de initiële configuratie tot de daaropvolgende werking en beheer, ontwikkeld is conform de beste beveiligingsstandaarden en -praktijken.

Lokaal netwerk

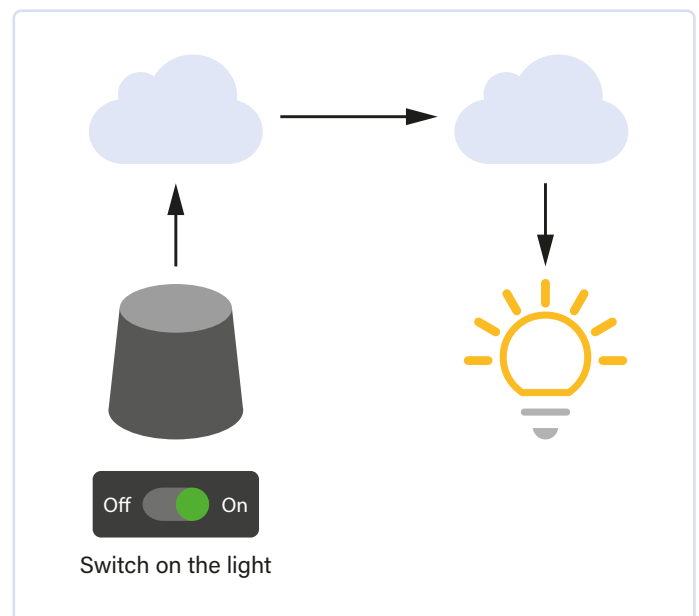
De architectuur van Matter verschilt op één fundamentele manier van die van de connected apparaten vóór Matter. Voorheen werd elk verbonden apparaat meestal bestuurd via het lokale netwerk door een apparaat, zoals een telefoon, op hetzelfde netwerk. Voor alle andere scenario's waren de apparaten verbonden met een cloud-platform en communiceerde een app op een telefoon die zich niet op hetzelfde netwerk bevond met deze apparaten via de cloud. De integratie met spraakassistenten vonden ook plaats via cloud-naar-cloud communicatie, zoals geschetst in **figuur 2**.

Matter is een protocol dat alleen op het lokale netwerk draait. Het zorgt voor de initiële configuratie, ontdekking en werking van aangesloten apparaten op het lokale netwerk zelf. De aangesloten apparaten zelf hoeven niet te communiceren met een cloud-service, zoals het geval was bij eerdere apparaten. Spraakassistent-integratie werkt daarom rechtstreeks door de juiste Matter-commando's over het lokale netwerk te sturen (**figuur 3**). Het wordt dus ook aan de Matter-controllers overgelaten om te beslissen hoe ze deze apparaten op afstand bedienen.

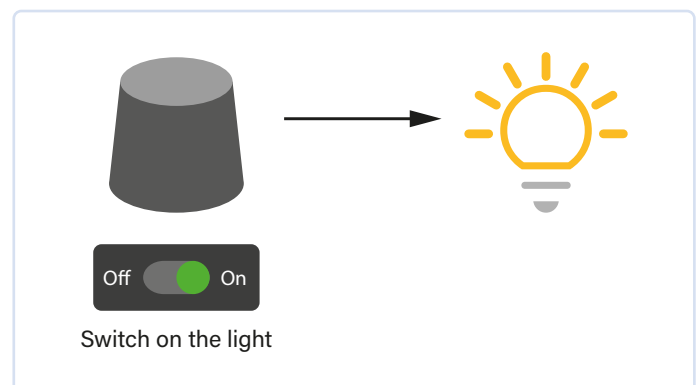
Dit biedt de eindgebruiker een keuze bij het selecteren van de Matter-apparaten en ecosystemen die hij vertrouwt met betrekking tot privacy en veiligheid.



Figuur 1. Een Matter Plug zoals die verschijnt in verschillende ecosysteem-apps.



Figuur 2. Voorbeeld van communicatie (niet-Matter).



Figuur 3. Voorbeeld van communicatie (Matter).

Overdracht

Matter biedt zijn functies via WiFi- en Thread- (802.15.4) overdracht-protocollen, die alomtegenwoordig zijn in het slimme huis. Elk van deze protocollen heeft zijn eigen voordelen.

Omdat het WiFi-netwerk alomtegenwoordig is, kunnen WiFi-apparaten van Matter gemakkelijk worden geïntegreerd in bestaande netwerken.

Apparaten op basis van Thread (802.15.4) zijn beter geschikt voor low-power toepassingen, zoals sensoren, waar de data duty cycle veel lager is. Thread-gebaseerde Matter-apparaten hebben wel een Thread border-router nodig om deel uit te maken van het netwerk. De meeste bestaande slimme luidsprekers/ecosysteem-apparaten ondersteunen Thread en upgraden hun software om de border-routerfunctie te ondersteunen. Dit maakt de integratie van Matter-apparaten in het thuisnetwerk veel eenvoudiger.

Inbedrijfstelling

Alle aangesloten apparaten moeten eerst op het netwerk van de gebruiker worden aangesloten met behulp van een of ander provisioning-mechanisme. Dit is een proces dat vaak gepaard gaat met fouten, verkeerde configuratie en frustratie bij de gebruiker.

Matter-apparaten worden aangesloten door middel van een proces dat Matter commissioning wordt genoemd. Gezien de standaard Matter-ondersteuning in de meeste ecosystemen, is dit proces robuust en verfijnd en biedt het een soepele ervaring voor elke gebruiker. Onder de motorkap stellen de meeste Matter-apparaten de inbedrijfstellings-functionaliteit beschikbaar via Bluetooth Low Energy (BLE). Matter-controllers zullen de referenties van het beoogde netwerk (WiFi of Thread) overdragen via een beveiligde BLE-link.

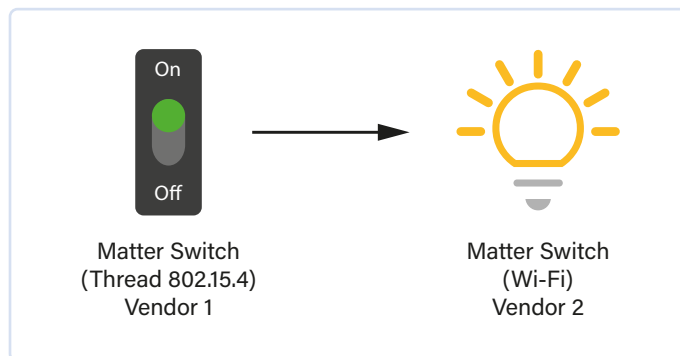
Apparaten kunnen ook Matter-inbedrijfstelling aanbieden via Ethernet of WiFi als ze al deel uitmaken van het netwerk van de gebruiker. Dit mechanisme is cruciaal om bestaande apparaten te kunnen upgraden om Matter-ondersteuning aan hun eindgebruikers te kunnen aanbieden.

Attest

Device Attestation is een belangrijke beveiligingsfunctie van Matter. Het is een proces dat plaatsvindt tijdens de ingebruikneming van het Matter-apparaat, waarbij de Matter-controller de authenticiteit van het apparaat vaststelt. Elk Matter-apparaat is geprogrammeerd met een unieke set van fabrikant-specifieke beveiligingscertificaten. Het Device Attestation-proces verzekert dat het apparaat echt afkomstig is van de fabrikant waarvan het beweert afkomstig te zijn. In het geval van een discrepantie wordt de eindgebruiker gewaarschuwd over het authenticatieprobleem van het apparaat. Dit maakt het moeilijk om vervalsingen van Matter-producten te maken en in gebruik te laten nemen, waardoor gebruikers en hun gegevens worden beschermd.

Apparaat-naar-apparaat communicatie

Vóór Matter werd de meeste communicatie tussen apparaten georkestreerd via de cloud of door spraakassistenten. Dit vereiste dat alle apparaten met dezelfde cloud communiceerden of een aantal cloud-naar-cloud-integraties bezaten om ecosystemen met



Figuur 4. Matter-binding: apparaat-naar-apparaat communicatie.

toegevoegde waarde te creëren die automatisch het juiste deden. Een groot voordeel van Matter-apparaten is dat de communicatie tussen apparaten volledig binnen het lokale netwerk plaatsvindt, dus er komt geen cloud-communicatie aan te pas. Een Matter-apparaat van de ene leverancier kan ingesteld worden om te communiceren met een Matter-apparaat van een andere leverancier. Je kunt bijvoorbeeld automatiseringen instellen om een groep lampen (of een airconditioner) te schakelen op basis van de status van een fysieke schakelaar of een sensor. Dit kan zelfs werken als de apparaten verschillende protocollen gebruiken (de ene kan WiFi gebruiken terwijl de andere Thread gebruikt). De mogelijkheid om deze automatiseringen (in Matter-taal 'bindingen' geheten) in te stellen vanuit elk Matter-ondersteund ecosysteem of app maakt het nog makkelijker om het slimme huis echt slim te laten zijn (figuur 4).

Coëxistentie van ecosystemen

Een ander probleem waar de meeste huizen mee te maken hebben, is de heterogeniteit van ecosystemen die door de bewoners van een huis worden gebruikt. De meeste gebruikers hebben hun eigen voorkeuren in ecosystemen, zoals Android, iOS, Alexa, GVA, SmartThings of Home Assistant. Dit leidt tot een gefragmenteerde gebruikerservaring, waarbij sommige connected apparaten slechts deel kunnen uitmaken van één ecosysteem of slechts tweederangs functies kunnen bieden in andere ecosystemen. Een Matter-apparaat kan daarentegen deel uitmaken van meerdere ecosystemen tegelijk. Je kunt een Matter-apparaat toevoegen aan maximaal vijf afzonderlijke ecosystemen. Alle ecosysteem-gebruikers kunnen tegelijkertijd genieten van dezelfde voordelen die Matter-apparaten bieden. Zelfs een melding van een verandering in het ene ecosysteem wordt ontvangen door het andere.

Over-the-Air (OTA) firmware-upgrade

Over-the-air firmware-upgrade stelt een product niet alleen in staat om de laatste beveiligings- of functionaliteitsfixes op te nemen, maar ook om na verloop van tijd nieuwere functies in te schakelen. De Matter-specificatie voorziet in een gedistribueerd grootboek waar fabrikanten OTA firmware-upgrade-images voor hun producten kunnen uploaden en onderhouden. Het gedistribueerde grootboek is toegankelijk voor alle Matter-controllers, die de benodigde firmware-upgrades kunnen downloaden en implementeren op de Matter-apparaten, met toestemming van hun gebruikers.



Certificering

Matter heeft een strikt certificeringsproces voor apparaten. De aanwezigheid van het Matter-embleem op een connected apparaat geeft aan dat het apparaat alle tests heeft ondergaan en doorstaan die nodig zijn om het apparaat Matter-gecertificeerd te krijgen. De Matter-certificering helpt garanderen dat het apparaat interoperabel is met andere Matter-apparaten en -controllers en voldoet aan een minimum aan robuustheid en functionaliteit die gewenst wordt door consumenten. Consumenten van Matter-producten hoeven niet langer te vertrouwen op het woord van de fabrikant voor deze basisvereisten. Alle Matter-fabrikanten worden gehouden aan dezelfde standaard van robuustheid en interoperabiliteit.

Wat is er nieuw aan Matter 1.2?

Onlangs werd Matter versie 1.2 uitgebracht. De eerste versie van Matter begon met ondersteuning voor de meest gangbare apparaten, zoals lampen, stopcontacten, stekkers, jaloezieën en thermostaten. De nieuwste Matter 1.2 standaard bevat ondersteuning voor witgoed, waaronder wasmachines, koelkasten, airconditioners, vaatwassers en robotstofzuigers. Ondersteuning voor rook- en koolmonoxidemelders, luchtkwaliteitssensoren, luchtreinigers en ventilatoren is ook inbegrepen. Deze ondersteuning omvat aanvullende specifieke commando's voor deze apparaten die verder gaan dan de typische aan/uit-bediening.

De kernspecificatie Matter bevat ook upgrades voor een flexibeler samenstelling van apparaten uit de subonderdelen. Dit maakt een nauwkeurigere modellering van een verscheidenheid aan apparaten mogelijk. De nieuw toegevoegde ondersteuning voor semantische tags zorgt voor een interoperabele manier om de locatie of semantische functies van een apparaat te beschrijven.

Hoe zit het met ontwikkelaars?

De Matter-standaard is publiek beschikbaar als download (na het invullen van een formulier). De Matter C++ SDK wordt gehost op GitHub [1], terwijl alle ontwikkeling op GitHub zelf gebeurt. De SDK implementeert zowel de functionaliteit aan firmware-kant (apparaat) als controller-kant. Experimentele implementaties voor JavaScript [2] en Rust [3] zijn ook in de maak. Hierdoor hebben ontwikkelaars eenvoudig toegang tot de specificaties en kunnen ze verbeteringen aanbrengen in de repository.

Met oplossingen zoals ESP-Launchpad [4] en ESP-ZeroCode [5] kunnen ontwikkelaars de gebruikerservaring van Matter eenvoudig uitproberen en ervaren door de firmware te flashen naar een van de vele hardware-ontwikkelkits.

Dit stelt ontwikkelaars in staat om Matter te bouwen en te ervaren voor het apparaat dat ze in gedachten hebben. Zodra apparaat-firmware en -hardware beschikbaar is, stellen alle Matter-ecosystemen ontwikkelaars in staat om deze apparaten te evalueren zonder productie-firmware of productiecertificaten nodig te hebben. Dit maakt iteraties bij de ontwikkeling van Matter-apparaten veel gemakkelijker.

Voor ontwikkelaars van telefoon-apps bevatten de nieuwste versies van iOS [6] en Android [7] API's waarmee deze apps toegang kunnen krijgen tot de Matter-API's. App-ontwikkelaars kunnen deze functie gebruiken om rijkere functionaliteit te bieden dan de basis-besturingssystemen voor telefoons bieden.

Ik hoop dat dit jullie, de ontwikkelaars, intrigeert om Matter-producten zelf te ervaren en aan de slag te gaan met Matter-productontwikkeling voor jullie apparaten. ◀

230617-03

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via kedar.sovani@espressif.com of naar de redactie van Elektor via redactie@elektor.com.

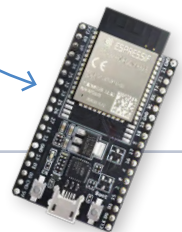
Over de auteur

Kedar Sovani heeft meer dan 21 jaar ervaring op het gebied van systeemsoftware, beveiliging en IoT. De afgelopen zes jaar heeft hij bij Espressif gewerkt. Hij is Senior Director of Engineering en richt zich op IoT-ecosystemen, waar zijn werk helpt bij het identificeren en bouwen van platforms en oplossingen die ontwikkelaars helpen sneller, robuuster en veiliger IoT-producten te bouwen. Hij is een hands-on ontwikkelaar en sleutelt momenteel aan Matter en Rust.



Gerelateerde producten

- ESP32-DevKitC-32E www.elektor.nl/20518
- ESP32-C3-DevKitM-1 www.elektor.nl/20324



WEBLINKS

- [1] Matter C++ SDK op GitHub: <https://github.com/project-chip/connectedhomeip>
- [2] Implementatie voor JavaScript: <https://github.com/project-chip/matter.js>
- [3] Implementatie voor 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 voor iOS: <https://developer.apple.com/documentation/matter>
- [7] Home Mobile SDK voor Android: <https://developers.home.google.com/matter/apis/home>

Bemachtig de nieuwe



ESPRESSIF hardware!

Er is niets dat ons meer enthousiast maakt dan nieuwe hardware in handen te krijgen, en deze samenwerking met Espressif was dan ook een waar genoegen! Hebben we je lekker gemaakt? Elektor heeft deshop gevuld met alle producten die in deze editie aan bod komen!



ESP32-S3-Box-3

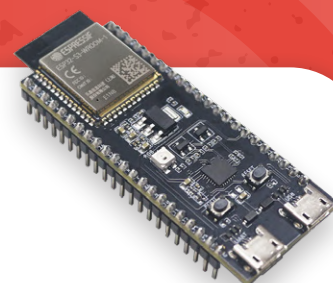
ESP32-S3-BOX-3 is een volledig open-source AIoT-ontwikkelkit gebaseerd op de krachtige ESP32-S3 AI SoC, en is ontworpen om een revolutie teweeg te brengen op het gebied van traditionele ontwikkelboards. ESP32-S3-BOX-3 wordt geleverd met een uitgebreide set add-ons, waarmee ontwikkelaars de functionaliteit van deze kit eenvoudig kunnen aanpassen en uitbreiden.

www.elektor.nl/20627

ESP32-S3-Eye

De ESP32-S3-EYE is een klein AI development board. Het is gebaseerd op de ESP32-S3 SoC en ESP-WHO, het AI development framework van Espressif. Het beschikt over een 2 megapixel camera, een LC-display en een microfoon, die worden gebruikt voor beeldherkenning en audioverwerking.

www.elektor.nl/20626



ESP32-S3-DevKitC-1

De ESP32-S3-DevKitC-1 is een instap-model development board uitgerust met de ESP32-S3-WROOM-1, ESP32-S3-WROOM-1U of ESP32-S3-WROOM-2, een universele WiFi + Bluetooth Low Energy MCU-module die complete WiFi en Bluetooth LE-functies integreert.

www.elektor.nl/20697



ESP32-C3-DevKitM-1

ESP32-C3-DevKitM-1 is een instapmodel development board gebaseerd op de ESP32-C3-MINI-1, een module die zijn naam dankt aan zijn kleine formaat. Dit board bevat volledige WiFi- en Bluetooth LE-functies. De meeste I/O-pinnen op de ESP32-C3-MINI-1 module zijn toegankelijk via de pinheaders aan beide zijden van dit board voor eenvoudig interfaceren. Ontwikkelaars kunnen randapparatuur aansluiten met jumperdraden of de ESP32-C3-DevKitM-1 op een breadboard monteren.

www.elektor.nl/20324



ESP32-Cam-CH340

Het ESP32-Cam-CH340 development board kan op grote schaal worden gebruikt in verschillende Internet of Things-toepassingen, zoals slimme huishoudelijke apparaten, industriële draadloze besturing, draadloze monitoring, QR draadloze identificatie, draadloze positioneringssystemen en andere Internet of Things-toepassingen.

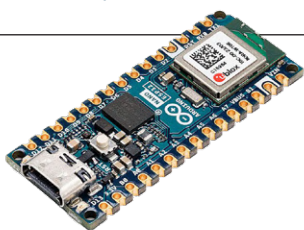
www.elektor.nl/19333



Elektor Cloc 2.0 Kit

Cloc is een eenvoudig te bouwen ESP32-gebaseerde wekker die verbinding maakt met een tijdserver en radio en TV aanstuurt. Hij heeft twee 7-segments retro-displays met variabele helderheid. Op het ene display staat de huidige tijd, op het andere de wektijd.

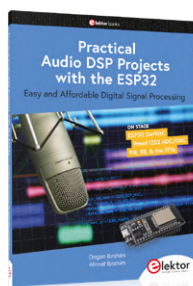
www.elektor.nl/20438



Arduino Nano ESP32

De Arduino Nano ESP32 is een board met nanovormfactor gebaseerd op de ESP32-S3 (embedded in de NORA-W106-10B van u-blox). Dit is het eerste Arduino-board dat volledig gebaseerd is op een ESP32 en beschikt over WiFi, Bluetooth LE, debuggen via native USB in de Arduino IDE en een laag stroomverbruik.

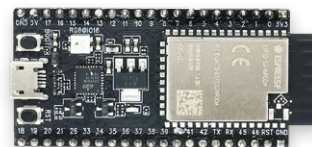
www.elektor.nl/20562



Practical Audio DSP Projects with the ESP32

Het doel van dit boek is om de basisprincipes van digitale signaalverwerking (DSP) te leren en het vanuit een praktisch oogpunt te introduceren met een minimum aan wiskunde. Alleen het basisniveau van discrete-tijdsysteemtheorie wordt gegeven, voldoende om DSP-toepassingen in realtime te implementeren. De praktische implementaties worden in realtime beschreven met behulp van het zeer populaire ESP32 DevKitC microcontroller-ontwikkelboard.

www.elektor.nl/20558



ESP32-S2-Saola-1M

ESP32-S2-Saola-1M is een klein ESP32-S2-gebaseerd ontwikkelboard. De meeste I/O-pinnen zijn toegankelijk via de pinheaders aan beide zijden voor eenvoudig interfacen. Ontwikkelaars kunnen randapparatuur aansluiten met jumperdraden of de ESP32-S2-Saola-1M op een breadboard monteren. ESP32-S2-Saola-1M is uitgerust met de ESP32-S2-WROOM module, een krachtige, generieke WiFi MCU-module met een uitgebreide set perifere schakelingen.

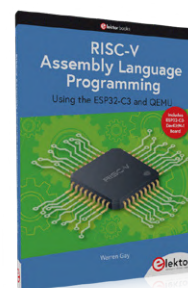
www.elektor.nl/19694



MakePython ESP32 Development Kit

De MakePython ESP32 Kit is een onmisbare ontwikkelkit voor het ESP32 MicroPython-programmeren. Samen met het MakePython ESP32 development board bevat de kit de elektronische basiscomponenten en modules die je nodig hebt om te beginnen met programmeren. Met de 46 projecten in het begeleidende boek kun je eenvoudige elektronische projecten met MicroPython op ESP32 aanpakken en je eigen IoT-projecten opzetten.

www.elektor.nl/20137



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

De beschikbaarheid van de Espressif ESP32-C3 chip biedt een manier om praktische ervaring op te doen met RISC-V. De open source QEMU-emulator voegt een 64-bits ervaring met RISC-V onder Linux toe. Dit zijn slechts twee manieren voor zowel studenten als liefhebbers om RISC-V in dit boek te verkennen. De projecten in dit boek zijn teruggebracht tot de essentie om de concepten van assembler duidelijk en eenvoudig te houden.

www.elektor.nl/20296

MicroPython for Microcontrollers

Krachtige controllers zoals de ESP32 bieden uitstekende prestaties en WiFi en Bluetooth functionaliteit tegen een betaalbare prijs. Met deze functies is de Maker-scene stormenderhand veroverd. Vergeleken met andere controllers heeft de ESP32 een aanzienlijk groter flash- en SRAM-geheugen en een veel hogere CPU-snelheid. Door deze eigenschappen is de chip niet alleen geschikt voor klassieke C-toepassingen, maar ook voor programmeren met MicroPython. Dit boek introduceert de toepassing van moderne een-chipsystemen.

www.elektor.nl/19736





Waarheen met Smarthome IoT?

Amey Inamdar, Espressif

Hoewel de technologie voor connectiviteit geëvolueerd is en toegankelijker, veiliger en kostenefficiënter geworden is, staat het slimme huis, zoals dat in het afgelopen decennium werd geschilderd, nog maar aan het begin. We zien een wildgroei aan apparaten, waaronder slimme thermostaten, beveiligingssystemen, slimme apparaten, intelligente verlichting en spraakassistenten. Toch heeft de technologie nog een lange weg te gaan op het gebied van acceptatie. Recentelijk zijn er twee belangrijke ontwikkelingen geweest die de acceptatie kunnen versnellen en die gaan we in dit artikel bespreken.

Stroomverbruik, kosten, ontwikkelingsgemak en connectiviteitsopties zijn belangrijke punten van overweging geweest bij het bouwen van elk smart home apparaat. Als we om ons heen kijken, zijn veel van deze problemen al voor een groot deel opgelost. De vraag is dan, wat houdt de brede acceptatie van deze apparaten tegen? Een van de vele antwoorden op deze vraag is waarschijnlijk de gevoelswaarde van smart home-apparaten en welke zaken ze ontsluiten. Zijn deze apparaten echt slim, of bieden ze alleen de mogelijkheid tot afstandsbediening? Een andere belangrijke reden kan te maken hebben met privacy en veiligheid. Spraak is een natuurlijke interface geworden voor interactie in het slimme huis, maar er hangt een prijskaartje aan het verlies van privacy.

Dat is precies waar de twee belangrijkste ontwikkelingen die onlangs hebben plaatsgevonden een sprankje hoop bieden. We zien de grote transformatie die generatieve AI en grote taalmodellen teweegbrengen op verschillende gebieden. Grote taalmodellen (LLM's) hebben het potentieel om de efficiëntie en autonomie van smart home-apparaten te verbeteren door de besluitvorming te decentraliseren en naar de 'edge' (rand) te

brengen. Deze LLM's hebben ook het potentieel om bij te dragen aan een betere privacy. Als we bijvoorbeeld kijken naar de bestaande spraakinterface, dan zien we dat deze wordt aangestuurd door cloudgebaseerde besluitvorming of door een vaste, op commando's gebaseerde spraakinterface waarbij de gebruiker de exacte commando's moet onthouden waarop deze kan werken. Maar met LLM's die offline modellen bieden zoals Whisper.ai om verschillende talen te begrijpen en lokaal te besluiten, hebben ze het potentieel om natuurlijke spraakinterfaces te bieden die volledig offline kunnen werken. We zien al een paar open-source projecten in deze richting gaan. Deze vooruitgang in generatieve AI wordt aangevuld door kleine microcontrollers die steeds beter in staat zijn om ML-modellen aan de 'edge' uit te voeren. Het is nu niet ongewoon om microcontrollers te zien met energiezuinige AI-acceleratie-engines die de uitvoering van ML-modellen kunnen versnellen. Dit maakt low-power sensoren mogelijk die niet alleen gegevens kunnen waarnemen, maar deze ook kunnen verwerken om er betekenis uit te halen.

Maar alleen het hebben van deze intelligentie voor smart home IoT-apparaten zal niet helpen als alle apparaten niet dezelfde taal spreken. Dat is precies waar standaardisatie hard nodig is. Een gebrek daar aan is nu een van de belangrijkste problemen voor de massale toepassing van smart home IoT-apparaten. Het feit dat apparaten van verschillende leveranciers niet met elkaar kunnen praten, beperkt de gebruiksmogelijkheden voor de consument en maakt het moeilijk om slimme thuisapparaten te configureren en te bedienen, en daar worden de recente standaardisatie-inspanningen belangrijk. Een protocol zoals Matter kan, als het succesvol is, de mogelijkheid bieden voor apparaten om



dezelfde taal te spreken, waardoor een meer consumentgerichte smart home met betere inzetbaarheid en gebruikerservaring mogelijk wordt. Nog een indirect effect van standaardisatie is dat het ontwikkelaars in staat stelt om waarde te creëren op een hoger niveau dan wat apparaatfabrikanten bieden.

Zowel AI als standaardisatie zijn dus van belang om de grootste problemen met de waarde en privacy van slimme thuisapparaten op te lossen. Dit moet worden ondersteund door innovatie op het gebied van connectiviteit, detectie en computerverwerking om de toename van kosteneffectieve en veilige smart home-apparaten mogelijk te maken. We zijn inmiddels getuige geweest van veel van dit soort ontwikkelingen. De Wi-Fi 6 standaard heeft bijvoorbeeld de mogelijkheid om batterijgevoede, verbonden apparaten te maken zonder afbreuk te doen aan de bandbreedte. De opkomst van mmgolf- en UWB-sensoren zorgt voor een veel betere bezettingsdetectie zonder dat er overal camera's nodig zijn die de privacy verstoren. Cybersecurity-labeling-programma's worden ook steeds populairder en verschillende regio's en landen komen met sterren-labels die de graad van beveiliging van smarthome-apparaten aangeven.

Er is geen wondermiddel om een universeel slim huis te realiseren. Maar waarschijnlijk gaan we nu in een betere richting dan voorheen. Het is beslist een interessante tijd en een boeiende omgeving om te zien hoe het zich ontwikkelt en om zelf onze rol te bepalen om de wereld beter te maken. ◀

230655-03

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

The Next Era of
Microcontrollers



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

www.espressif.com



Learn More About
ESP SoCs