Q&A met Scott Mabin

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 efficient 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.

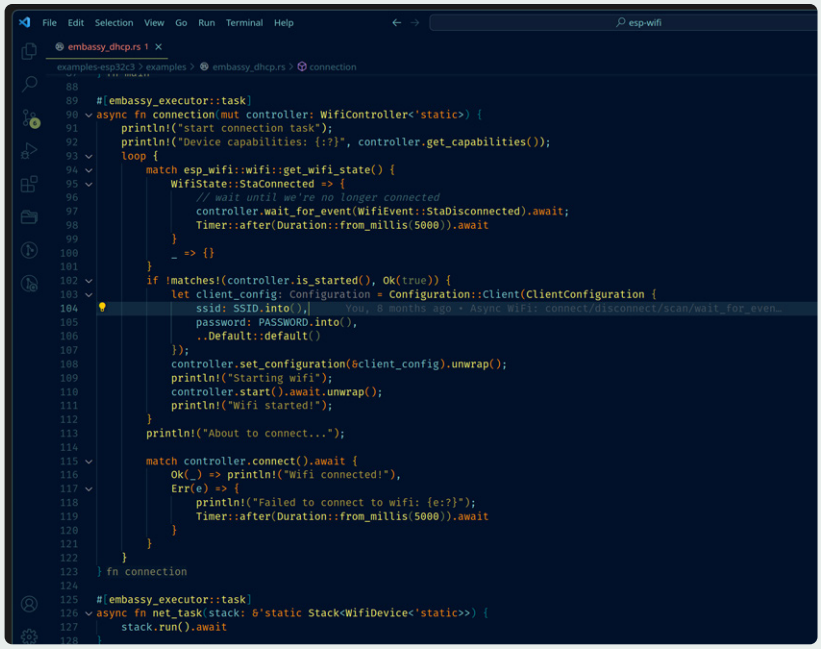
Stuart Cording: Dus je sleutelde aan Rust op de ESP32. Hoe ontwikkelde zich je verhouding met Espressif?
Scott Mabin: Ik begon de *esp-rs* groep op GitHub in mijn vrije tijd om Rust-projecten voor ESP32 te verzamelen. Samen met andere leden van de community hebben we ondersteuning voor periferie geïmplementeerd, 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 resourcekant. 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 geheugenveiligheid 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.

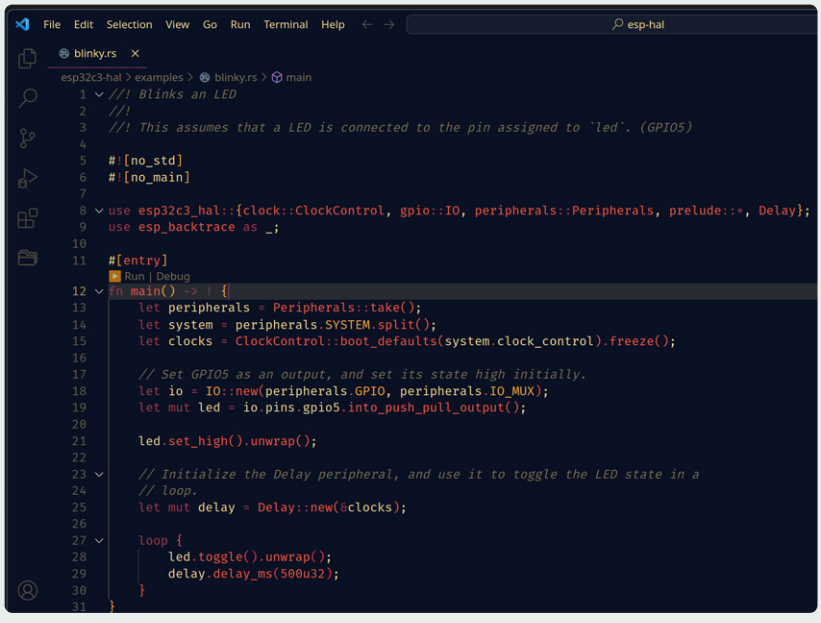


▲

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 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.



▼

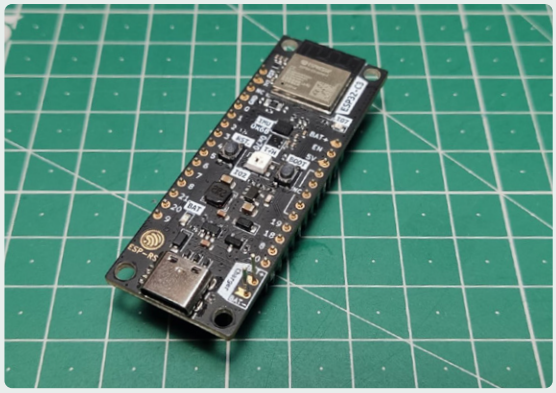
Figuur 2. Scott Mabin raadt VS Code aan voor Rust-ontwikkeling. Hier is de klassieke blinky LED-code te zien.

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 kleinst en goedkoopst 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 trainingsmateriaal 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 omgevingen 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