ESPRESSIF DevCon22

Rust on Espressif chips

Scott Mabin

What I'll cover today

ESPRESSIF DevCon22

- What Rust is
- Why we're adopting it at Espressif
- What you can do right now with Rust and Espressif chips
- What's the plan going forward

Background on Rust

- Rust is a systems programming language with the slogan "fast, reliable, productive: pick three."
- 1.0 release back in 2015
- 6-week release cycle
- Previously governed by Mozilla, but is now managed by an independent non-profit organization, the Rust Foundation.

Why Rust?

ESPRESSIF DevCon22

- It's fast, compiling down to machine code just like C
  - Memory is deallocated as it goes out of scope, no garbage collection is required.
- Eliminates a whole class of memory and synchronization bugs at compile time
  - In 2019 Microsoft announced that over 70% of CVEs in the last 12 years related to their system-level software (written in C or C++) were memory safety bugs.
- Package management with `cargo` which is similar to the esp-idf component manager, but supports the entire language.
- Imperative language, but with strong functional elements

Why Rust at Espressif?

ESPRESSIF DevCon22

- We see it as an emerging language in the embedded (and tooling) space
- We expect to be able to write new parts of esp-idf in Rust
- We expect to rewrite certain parts of esp-idf where Rust's safety guarantees can help
- Vast & diverse open source ecosystem
- Package management

Why Rust for embedded?

ESPRESSIF DevCon22

- Memory safety is even more important, most embedded systems do not have an MMU
- Native `async` support, more on this later
- Separation of core library & standard library
- Package management helps foster an open source ecosystem
  - Interface trait crates like `embedded-hal`
  - Non-allocating data structure crates like `heapless`
  - Thousands more, see awesome-embedded-rust

What's possible right now?

Three approaches:

- esp-idf project + Rust
- Rust project + esp-idf
- Bare metal Rust

esp-idf + Rust

Do you want to use Rust for some specific parts of your esp-idf application? No problem! It's possible to write esp-idf components in Rust that expose a C interface to the rest of your application.

Rust + esp-idf

Thanks to the newlib component of esp-idf, it's possible to build the Rust standard library on top of esp-idf. This gives you all of the benefits of Rust, with built-in support for threads, mutexes, networking and more!

On top of the standard library items, we can also leverage the peripheral drivers and other components in esp-idf by writing ergonomic Rust wrappers around the C interfaces.

Bare metal Rust

Bare metal (no OS) applications are less common, most embedded projects use some form of OS (usually an RTOS). This is the case for Espressif chips in C, but in Rust we have started building an ecosystem around bare metal support. `async` being natively supported by the language is a big driver behind this, as it allows for efficient and safe multitasking.

What is `async`?

Generally, `async` is a form of cooperative multitasking, where "tasks" manually yield processing time when they are blocked on something.

`async` in Rust adds two new keywords to the language, `async` & `await`, where `async` defines a block or function to be asynchronous and `await` defines yield points *within* an `async` block or function.

```
pub fn say_hello(uart0: &mut Serial<UART0>) {
  let message = "Hello World!";
  uart0.write_bytes(message).await; // yield point here!
  let message = "Goodbye";
  uart0.write_bytes(message).await; // another yield point here!
}
```

What `async` is not

A common misconception is that when developing an application it's either blocking *or* async. This is not the case, `async` and blocking can and should be used in conjunction with each other within an application.

Example: `async` in an RTOS

When you have threads, `async` may seem redundant, but it can actually reduce your applications memory usage.

Take an application that has one heavy computation task, and three, smaller I/O bound tasks. In a traditional thread-only based model, this could use up to four threads with varying stack spaces.

Combining the `async` and blocking approach could reduce this to just two threads, one thread for the heavy computation, and another which asynchronously multiplexes the three I/O bound tasks.

Chip support

For bare metal or esp-idf projects, there is one toolchain limitation that our LLVM fork needs to support the chip. For all our RISCV chips this is a given because the backend is already upstream in LLVM. For Xtensa we needed to add support to our fork. Fortunately, all our Xtensa-based chips, `esp32`, `esp8266`, `esp32s2` & `esp32s3` are already supported there.

Chip support - Standard library / esp-idf

The general rule of thumb is that if esp-idf supports the chip, so does the Rust standard library. The following table from the esp-idf repo shows chip support.

Chip	v4.1	v4.2	v4.3	v4.4	v5.0	
ESP32	supported	supported	supported	supported	supported	
ESP32-S2		supported	supported	supported	supported	
ESP32-C3			supported	supported	supported	
ESP32-S3				supported	supported	Announcement
ESP32-C2					supported	Announcement
ESP32-H2				preview	preview	Announcement

Chip support - bare metal

In general, it takes a little longer to support in bare metal, but once support has landed in esp-idf bare metal support should be available soon after. We track the support table in the esp-rs/book.

Chip	PAC	HAL
ESP32	✅	✅
ESP32-C2	*planned*	*planned*
ESP32-C3	✅	✅
ESP32-S2	✅	✅
ESP32-S3	✅	✅
ESP32-H2	*planned*	*planned*
ESP8266	✅	✅

What's next?

ESPRESSIF DevCon22

- Continue upstreaming efforts, both LLVM & Rust (RISCV esp-idf targets are already upstream!)
- Provide Rust bindings for existing components in esp-idf
- Improve bare metal support, add WiFi support to more than just the `esp32` & `esp32c3`
- Plenty more! Check out the project board!

Learning resources - Ferrous training

We have some free training available, created by Ferrous Systems. We contracted them to create a workshop book to teach running Rust on Espressif chips!

Learning resources - Links

- Ferrous training
- esp-rs/book
- esp-rs - Our github organization
- esp-template - bare metal project template
- esp-idf-template - Standard library or CMake project template
- mabez.dev/blog - The history of supporting Rust on Espressif chips