Rust on Espressif chips

Juraj Sadel

Why Rust?

- It's fast, compiling down to machine code just like C
  - Memory is deallocated (dropped) as it goes out of scope, no garbage collection is required.
- Eliminates a whole class of memory and synchronization bugs at compile time
- Package management with `cargo` which is similar to the esp-idf component manager, but supports the entire language.

Advantages compared with C/C++

- Rust provides:
  - Memory safety at compile time through ownership and borrowing system
  - Zero-cost abstractions and safe concurrency and multi-threading (**async**/**await**)
  - Package manager and build system `cargo` automates pretty much everything from build, test, and publish process to creating a new project and managing its dependencies

Disadvantages compared with C/C++

ESPRESSIF

- Steep learning curve because of unique features such as ownership and borrowing
- Compilation time due to advanced type system and borrow checker
- Low-level control can be limited due to safety features
- Tooling & Community size

Why Rust at Espressif?

- 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?

- 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 is `async`?

- form of cooperative multitasking
- "tasks" manually yield processing time when they are blocked
- `async` -> block or function which will be asynchronous
- `await` -> yield point *within* an `async` block
- `Future` -> value that migh not be ready yet

```
pub async 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.

Embassy

- Lightweight embedded async framework
- `no_std` support
- Cooperativne multitasking (scheduling, task yields control explicitely)
- Built in support for time operations (delay, timeouts)
- Industry standard

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 we do?

- Rust compiler fork for Xtensa targets
- Tooling (espup, espflash)
- HAL for RISC-V and Xtensa targets
- WiFi and BLE (IEEE 802.15.4)
- Adding support for new chips

What's next?

- Continue upstreaming efforts, both LLVM & Rust (RISCV targets are already upstream!)
- Provide Rust bindings for existing components in esp-idf
- Improve bare metal support, add WiFi support
- Add support for upcoming chips
- Check out milestones

Learning resources - Links

- std training
- no_std 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
- matrix room