Rust on ESP32

Mathy Vanvoorden


I'M SORRY,
DO I KNOW YOU?
NETFLIX

Rust

A language empowering everyone
to build reliable and efficient software.

Why Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

Packages, crates and modules

- Modules
  - Simplified: single source file
  - Referenced with use statements (think #include, import)
- Crate
  - Simplified: group of modules
  - Either binary or library
- Package
  - Group of crates
  - Added as dependency in Cargo.toml

Cargo.toml

```toml
[package]
name = "workshop"
version = "0.1.0"
authors = ["Mathy Vanvoorden <mathy@vanvoorden.be>"]
edition = "2021"
resolver = "2"
rust-version = "1.77"

[[bin]]
name = "workshop"
harness = false # do not use the built in cargo test harness -> resolve rust-analyzer errors

[profile.release]
opt-level = "s"

[profile.dev]
debug = true    # Symbols are nice and they don't increase the size on Flash
opt-level = "z"
```

Cargo.toml

```toml
[features]
default = ["std", "embassy", "esp-idf-svc/native"]

pio = ["esp-idf-svc/pio"]
std = ["alloc", "esp-idf-svc/binstart", "esp-idf-svc/std"]
alloc = ["esp-idf-svc/alloc"]
nightly = ["esp-idf-svc/nightly"]
experimental = ["esp-idf-svc/experimental"]
embassy = ["esp-idf-svc/embassy-sync", "esp-idf-svc/critical-section", "esp-idf-svc/embassy-time-driver"]

[dependencies]
log = { version = "0.4", default-features = false }
esp-idf-svc = { version = "0.49", default-features = false }

[build-dependencies]
embuild = "0.32.0"
```

Macro’s

- Macro’s end with a bang !
- Think #define but then more advanced
  - Metaprogramming, in Rust!
- Example uses
  - Avoid code repetition
  - Create your own DSL
  - Variadic “functions”
- Example
  - println!(“The value is {}.”, value);

Ownership

- Core principle of Rust programming
  - Each value has an owner
  - There can only be one owner at the same time
  - When the owner goes out of scope, the value is dropped
- Ownership can be transferred
  - Temporary (= Borrowing, using references)
  - Permanently (compare to std::move)
- Main reason of memory safety at compile time in Rust

Error handling as a core language feature

```rust
use std::fs::File;
use std::io::ErrorKind;

fn main() {
    let greeting_file_result = File::open("hello.txt");

    let greeting_file = match greeting_file_result {
        Ok(file) => file,
        Err(error) => match error.kind() {
            ErrorKind::NotFound => match File::create("hello.txt") {
                Ok(fc) => fc,
                Err(e) => panic!("Problem creating the file: {e:?}"),
            },
            other_error => {
                panic!("Problem opening the file: {other_error:?}");
            }
        },
    };
}
```

Error handling as a core language feature

```rust
use std::fs::File;

fn main() {
    let greeting_file = File::open("hello.txt").unwrap();
}
```

```rust
use std::fs::File;

fn main() {
    let greeting_file = File::open("hello.txt")
        .expect("hello.txt should be included in this project");
}
```

```rust
use std::fs::File;
use std::io::{self, Read};

fn read_username_from_file() -> Result<String, io::Error> {
    let mut username_file = File::open("hello.txt")?;
    let mut username = String::new();
    username_file.read_to_string(&mut username)?;
    Ok(username)
}
```

Rust on ESP32

- esp-idf-sys
  - Implements std Rust
  - Uses ESP-IDF internally
  - Almost everything in IDF has a compatible Rust interface
  - If not: just write a Rust interface to the C function!
- Core Library
  - Only no_std, so no std!
  - Interacts directly with the hardware
  - Also no IDF!
  - Very minimal, but also, very minimal! (< 150kB images are possible)

Rust 💘 async

- Language-native feature
- Syntax is supported by the language
- Free choice of executor (execution engine)
  - Community provided crates
  - Tokio very popular on desktop
  - On ESP32: Embassy

```
async fn get_two_sites_async() {
    // Create two different "futures" which, when run to completion,
    // will asynchronously download the webpages.
    let future_one = download_async("https://www.foo.com");
    let future_two = download_async("https://www.bar.com");

    // Run both futures to completion at the same time.
    join!(future_one, future_two);
}
```

Enough talk, start programming!

https://github.com/MathyV/fri3d-rust-workshop

Tips

- cargo generate
- global .embuild
- compile in release
- sdkconfig.defaults
  - debug output
- If you move ESP32 projects on disk, completely remove target directory

Questions?

Contact


mathy@vanvoorden.be


mathyvanvoorden


https://github.com/MathyV