A tale of Rust, the ESP32 and IoT

It can’t be that hard...
Who am I?

Jens Reimann
• Principal Software Engineer
• Red Hat
  • Middleware, Messaging, IoT
• Programming languages
  • 90s: Basic, Pascal, C
  • 00s: C, C++, Java
  • 10s: Java, Go, Rust

@ctron
https://dentrassi.de
Telemetry data for Eclipse Hono
Goal of this talk?

Get you excited about Rust!
Why Rust?

Do we really need another programming language?
Rust

"A language empowering everyone to build reliable and efficient software."

https://www.rust-lang.org/
Rust

“Rust is a language for systems programming.”
Jim Blandy & Jason Orendorff, *Programming Rust*

“Systems programming is for:

...  
• Code that runs in very cheap devices, or devices that must be extremely reliable  
...  
”
Rust is ...

“A safe, concurrent language with the performance of C and C++”

Jim Blandy & Jason Orendorff, Programming Rust

“...eliminate many classes of bugs at compile-time.”

https://www.rust-lang.org/
History of languages

http://rigaux.org/language-study/diagram.html
History of Rust

• Started 2006 by Mozilla employee Graydon Hoare
• Announced 2010 by the Mozilla Foundation
• Self-compiled 2011
• Getting things right, before moving on
• A community to grow the language, not only use it
Fixing stuff at compile time

• Have a compiler which understands your code
• Have language rules which prevent bugs
• Eliminate “undefined behavior”
• Reduce “unexpected behavior”
What is undefined behavior?

**Undefined behavior:** behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements

§3.4.3, Cx11

**Undefined behavior:** Renders the entire program meaningless if certain rules of the language are violated.

cppreference.com
Go FAQ  
Q: „Why are map operations not defined to be atomic? “  
A: „...This was not an easy decision, however, since it means uncontrolled map access can crash the program....“

Java „CME“ - HashSet  
“Fail-fast iterators throw ConcurrentModificationException on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: the fail-fast behavior of iterators should be used only to detect bugs.”
A helping compiler...

Code

```rust
fn main() {
    let mut s1 = "Foo";
    let mut x = &mut s1;
    assert_eq!(*x, "Foo");

    {  
        let mut s2 = "Bar"; 
        x = &mut s2; 
    }

    assert_eq!(*x, "Bar");
}
```

Output

```text
Compiling playground v0.0.1 (/playground)
error[E0597]: `s2` does not live long enough
 --> src/main.rs:8:13
  |
8 |         x = &mut s2;
  |         ^^^^^^^ borrowed value does not live long enough
9 |     }
  | `s2` dropped here while still borrowed
10 | assert_eq!(*x, "Bar");
  | ---------------------- borrow later used here

= note: this error originates in a macro outside of the current crate (in Nightly builds, run with -Z external-macro-backtrace for more info)

error: aborting due to previous error

For more information about this error, try `rustc --explain E0597`.
error: Could not compile `playground`.

To learn more, run the command again with --verbose.

https://play.rust-lang.org/?version=stable&mode=debug&edition=2018&gist=cc5f9c6bbb709ac1c0e292b3042a4b40
The cost of a bug over time...
...with Rust
Dependencies

• “Crates” are the “JAR files of Rust” ... but contain code, not binaries!
• Then “crates.io” is “Maven Central for Rust”
• “cargo” manages dependencies, and orchestrates the build and test
The problem with “std”

• “std” provides all kinds of functionality
  • Files, Streams, ...
  • Network, Sockets, ...
  • ...
• But also requires a POSIX-like operating system
• So, what about embedded systems? Like the ESP32?
#![no_std], “core” & “alloc”

- You can disable the usage of “std” and switch to “core” instead
- If you can provide an allocator, you can also use “alloc” for dynamic memory allocations (like String, Vec, ...)
- Some crates support this by using “features”, which enable/disable features of the crate at compile time
  - e.g. “serde” with “serde-json-core”

https://crates.io/keywords/nostd
Unsafe Superpowers

• How?
  • “unsafe {}” block
  • “unsafe” keyword

• Why?
  • Call other “unsafe” methods
  • Dereference raw pointers
  • ...
ESP-IDF & Rust

My App

Rust
- serde / JSON
- core
- alloc
- "timer"
- "http"

C – ESP IDF
- malloc
- TLS
- HTTP
- ...
- FreeRTOS
- TCP/IP
- WiFi
End-to-end example

```rust
//no_mangle
pub fn app_main() {
    log::log(Level::INFO, &TAG, format_args!("Hello World"));
    init_global_ca_store();
    let config = http::HttpClientConfig {
        url: MONO_HTTP_ADAPTER_URL,
        authentication_type: Some(http::AuthenticationType::BASIC),
        authentication_header: Some(MONO_AUTH_HEADER),
        method: http::Method::POST,
        ..Default::default()
    };
    let mut http = http::HttpClient::new(&config).expect("Failed to init HTTP client");
    let mut app = app::Application::new(WIFI_SSID, WIFI_PASSWORD, move || {
        let temp = temperature_sensor_read();
        log::log(Level::INFO, &TAG, format_args!("Ticked: {}", temp));
        if let Err(err) = publish_telemetry(&mut http, temp) {
            log::log(
                Level::ERROR,
                &TAG,
                format_args!("Failed to execute HTTP upload: {}", err),
            );
        }
    });
    app.run();
}
```

https://github.com/ctron/rust-esp32-hono
End-to-end example

```rust
#[derive(Serialize, Deserialize, Debug)]
pub struct TelemetryPayload {
    /// temp: f32,
}

fn publish_telemetry(client: &mut httpClient, temp: f32) -> Result<Response> {
    let payload = TelemetryPayload { temp: temp };
    client.send_json::<heapless::consts::U128, TelemetryPayload>(&payload)
}

http::HttpClient, temp: f32) -> Result<Response> {
    { temp: temp }
    const::U128
    with the
    :sencrypt

    temp: f32
    :cert.as

    redundant field names in struct initialization
    help: for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#redundant_field
    help: replace it with: 'temp'
    Change to 'temp'
```
IDE Integration

```rust
default_value() -> String {
    String::from("foo")
}

fn main() {
    let s = Some(String::from("foo"));
    let s = s.unwrap_or(default_value());
```
Eclipse Corrosion

Eclipse Corrosion: Rust edition and debug

Corrosion enables Rust application development in the Eclipse IDE.

!!! A standalone Eclipse IDE for Rust Developers is also available for download !!!

Corrosion provides a rich and smart Rust editor with: - Syntax highlighting (using TextMate grammar) and Error reporting, Hover, Content assist. Jump to references, Code Outline, Formatting... provided by the Rust Language Server

Corrosion also integrates various operations of the `cargo` command-line (New Project, Build, Run, Debug, Package) as typical Eclipse IDE wizards and workflows.

Corrosion contains a rich debugger for Rust applications, allowing to set breakpoints, jump in/over an instruction, view and edit structured variables...

Categories: Editor, IDE, Languages

Tags: rust, rustlang, cargo, fileExtension_rs, IDE, redox, corrosion
IntelliJ + Rust
One more thing...
How to compile for the Xtensa architecture?
Forked LLVM, forked rustc and a bunch of scripts

- Execute ~50 different commands
- Hope you have the right OS, in the right version, with the right packages
- Wait ~3½ hours
- Enjoy ... or try again
Containerized

- `docker run quay.io/cron/rust-esp`
  - `--ti -v $PWD:/home/project`
- Runs on Windows, Linux, (and should on Mac OS)
What lies ahead?

• Rust Runtime for AWS Lambda
  • https://aws.amazon.com/blogs/opensource/rust-runtime-for-aws-lambda/

• Rust Embedded Book
  • https://rust-embedded.github.io/book/

• Linux kernel experiments with Rust
  • https://lwn.net/Articles/797828/

• Microsoft
  • https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code/
  • “~70% of the vulnerabilities Microsoft assigns a CVE each year continue to be memory safety issues”
Questions?

... and answers!
A few links

• Rust
  • https://www.rust-lang.org/

• Rust Embedded Book
  • https://rust-embedded.github.io/book/

• Programming Rust
  • O'Reilly Media

• Eclipse Corrosion
  • https://marketplace.eclipse.org/content/corrosion-rust-edition-eclipse-ide

• Rust for ESP32
  • https://github.com/ctron/rust-esp-esp-container/

• Rust, ESP32, ESP-IDF, Hono
  • https://github.com/ctron/rust-esp32-hono

• LLVM for Xtensa
  • https://github.com/espressif/llvm-xtensa

• Rust fork for Xtensa
  • https://github.com/MabezDev/rust-xtensa
Thank you!