Embedded Rust

Raphael Nestler (@rnestler), Noah Hüsser (@Yatekii)

2020-06-18

Rust Zürichsee Meetup

println!("{:?}", rnestler)

Hi! I'm Raphael (@rnestler).

println!("{:?}", rnestler)

Hi! I'm Raphael (@rnestler).

I live in Rapperswil

println!("{:?}", rnestler)

Hi! I'm Raphael (@rnestler).

I live in Rapperswil

I work at Sensirion (`https://sensirion.com`).

println!("{:?}", rnestler)

Hi! I'm Raphael (@rnestler).

I live in Rapperswil

I work at Sensirion (`https://sensirion.com`).

I'm a founding member of Coredump hackerspace (`https://coredump.ch`).

println!("{:?}", yatekii)

Hi! I'm Noah (@Yatekii).

println!("{:?}", yatekii)

Hi! I'm Noah (@Yatekii).

I live in Lenzburg

println!("{:?}", yatekii)

Hi! I'm Noah (@Yatekii).

I live in Lenzburg

I work at Technokrat (`https://technokrat.ch/`).

println!("{:?}", yatekii)

Hi! I'm Noah (@Yatekii).

I live in Lenzburg

I work at Technokrat (`https://technokrat.ch/`).

I started the probe-rs project (`https://probe-rs`).

Outline

1. Embedded Programming
2. State of Embedded in 2020
3. probe-rs
4. probe_rs.await?;

Embedded Programming

What is an *Embedded System*?

> *A combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function.*

Michael Barr. "Embedded Systems Glossary"[1]

[1] https://barrgroup.com/Embedded-Systems/Glossary-E#embedded_system

What is embedded programming?

- Dedicated, not general purpose, µC system
- Baremetal
- Low-Level

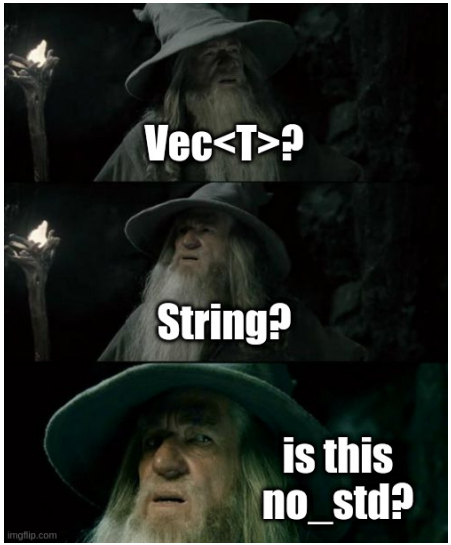
What is embedded programming?

- Dedicated, not general purpose, µC system
- Baremetal
- Low-Level
- For this talk: Bare metal on Cortex-M MCUs

Why do they say it's hard?

- Harsh environment (No OS which protects you)
- Resource constrained (Remember dedicated?)
- Non-standard, Non-OSS toolchain
- Hard realtime requirements
- ...

What does it mean for Rust?



Why is Rust awesome for it?

- Zero cost abstractions!
- Provides safety at compiler level, not OS
- Expressive type system to encode constraints

State of Embedded in 2020

The compiler

- `const fn`
- `core::future::Future`
- `alloc` crate is stable
- `MaybeUninit<T>`: Fixes undefined behavior

The compiler

`async` / `await` of course

The `.await` is over, `async fn`s are here

→ We'll get to it

State of Embedded in 2020

`const fn`

Constant Fun!

- Stabilized at the end of 2018
- Tons of functions got const during 2019
- More functions from libcore
- More features (destructuring, bindings, assignment, ...)

Constant Fun Example

An example from the lin-bus crate[2]

```
/// Calculate the PID from an ID.
/// P0 = ID0 ^ ID1 ^ ID2 ^ ID4
/// P1 = ¬(ID1 ^ ID3 ^ ID4 ^ ID5)
const fn from_id_const(id: u8) -> PID {
    // count parity bits and check if they are even / odd
    let p0 = (id & 0b1_0111).count_ones() as u8 & 0b1;
    let p1 = ((id & 0b11_1010).count_ones() as u8 + 1) & 0b1;
    PID(id | (p0 << 6u8) | (p1 << 7u8))
}
```

[2] https://github.com/Sensirion/lin-bus-rs/blob/e89739e326daf091803373419a4d60d0888422fc/src/frame.rs

Less Constant Fun Example

Saw the problem?

Less Constant Fun Example

Saw the problem?

```
pub fn from_id(id: u8) -> PID {
    assert!(id < 64, "ID must be less than 64");
    PID::from_id_const(id)
}
```

Less Constant Fun Example

Saw the problem?

```
pub fn from_id(id: u8) -> PID {
    assert!(id < 64, "ID must be less than 64");
    PID::from_id_const(id)
}
```

Workaround

```
// check that id < 64. Compile error during const evaluation
// and panic on debug build
let _: u8 = id + 192;
```

Constant Future

What is missing?

- Panicking in constants[3]
- Const generics[4]

[3] https://github.com/rust-lang/rust/issues/51999

[4] https://github.com/rust-lang/rust/issues/44580

State of Embedded in 2020

Other News

New Platform Support

- AVR basic support got merged[5] [6]
- New ARM targets: thumbv7neon-linux-{androideabi,gnueabihf}, armv{6/7}-unknown-freebsd-gnueabihf, aarch64-unknown-none-*
- New RISC-V targets: riscv64imac-unknown-none-elf, riscv64gc-unknown-none-elf, riscv32i-unknown-none-elf
- Sony PSP support[7]

[5] https://github.com/rust-lang/rust/pull/69478

[6] https://github.com/rust-lang/rust/issues/44052

[7] https://github.com/rust-lang/rust/pull/72062

RTFM → RTIC

- RTFM got renamed → RTIC[8]
- Examples: https://github.com/coredump-ch/nixie-counter/tree/master/firmware/

[8] https://rtic.rs/0.5/book/en/migration_rtic.html

State of Embedded in 2020

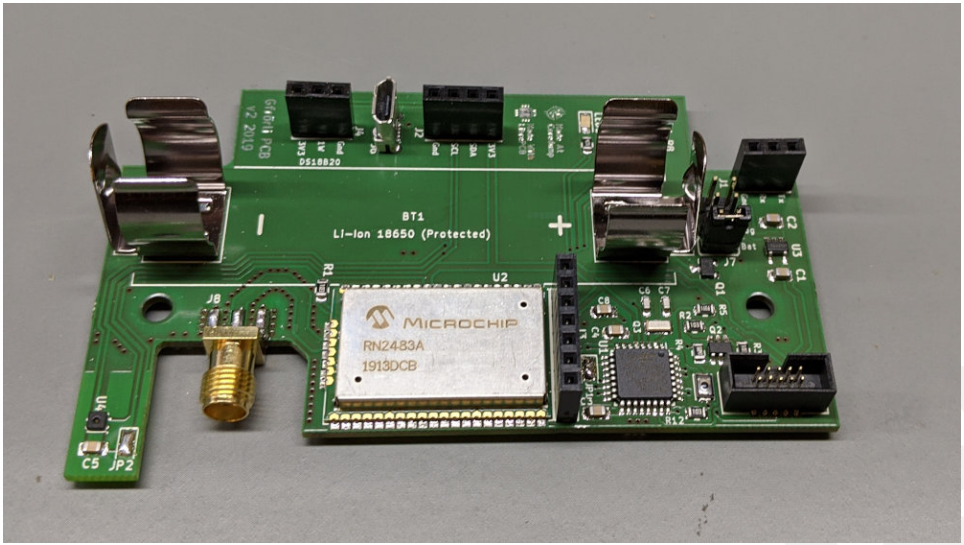
Embedded HAL

What is the Embedded HAL?

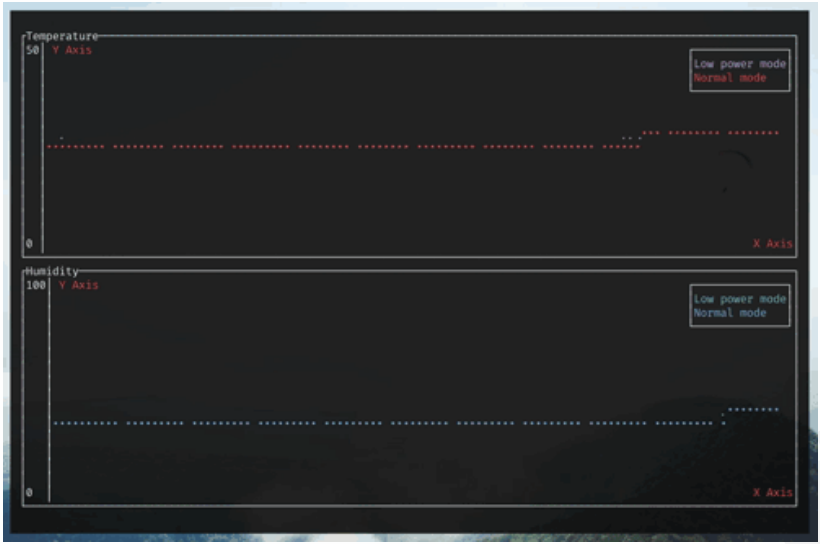
- Unified interface to the same HW functionality which has different interfaces
- Higher level crates (drivers) are write once use always
- Nice examples: shtc3 driver[9] developed and debugged on a Raspberry Pi and used with an STM32

[9] `https://github.com/dbrgn/shtcx-rs`

SHTC3 Driver as Example



SHTC3 Driver as Example



Embedded HAL — Challanges / Future

- It's hard to define traits which work everywhere
- How to define async traits?
- Ecosystem needs to move as a whole on breaking changes
- 1.0 Release planed[10]

[10] https://github.com/rust-embedded/embedded-hal/issues/177

State of Embedded in 2020

async on embedded

async — Some Background

- async/await landed in stable Rust 1.39[11]
- In February 2020 Ferrous Systems started blogging about bringing it to embedded[12] [13]
- Since Rust 1.44 usable on `no_std` [14]

[11] https://blog.rust-lang.org/2019/11/07/Async-await-stable.html
[12] https://ferrous-systems.com/blog/embedded-async-await/
[13] https://ferrous-systems.com/blog/async-on-embedded/
[14] https://ferrous-systems.com/blog/stable-async-on-embedded/

async — What is it Even?

- Mostly syntactic suger for a state machine
- Useful for concurrent waiting

async — State Machine

```
async fn foo() -> u8 {
    1
}
async fn bar(a: u8) -> u8 {
    a + 2
}
async fn f() -> u8 {
    let a = foo().await;
    bar(a).await
}
```

async — State Machine

```rust
async fn foo() -> u8 {
    1
}
async fn bar(a: u8) -> u8 {
    a + 2
}
async fn f() -> u8 {
    let a = foo().await;
    bar(a).await
}
```

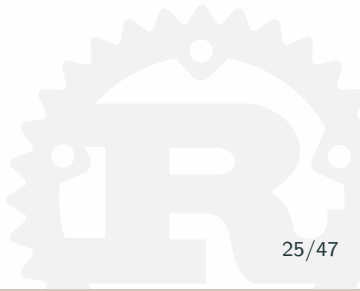
```rust
enum Future {
    Ready(u8),
    Pending,
}

enum State {
    Foo,
    Bar(u8),
    Done(u8),
}
```

async — State Machine

```
fn f_desugared(state: State) -> State {
    match state {
        State::Foo => match foo_() {
            Future::Ready(a) => State::Bar(a),
            Future::Pending => State::Foo,
        },
        State::Bar(a) => match bar_(a) {
            Future::Ready(b) => State::Done(b),
            Future::Pending => State::Bar(a),
        }
        State::Done(b) => State::Done(b)
    }
}
```

async — Why on Embedded?

- Most embedded systems are event driven → Most code is *awaiting* events.
- Embedded systems may want to save power → Sleep whenever possible.
- We often need to wait for peripherals / IO



async — A blocking blinky

```rust
fn main() -> ! {
    let mut led = Led::new();
    let mut timer = Timer::new();

    loop {
        led.on();
        timer.delay_ms(1_000);
        led.off();
        timer.delay_ms(1_000);
    }
}
```

async — A async blinky

```
fn main() -> ! {
    let mut led = Led::new();
    let mut timer = Timer::new();

    // `block_on` runs the future (`async` block) to completion
    task::block_on(async {
        loop {
            led.on();
            timer.wait(Duration::from_secs(1)).await;
            // ^ suspends the task for one second
            led.off();
            timer.wait(Duration::from_secs(1)).await;
        }
    })
}
```

async — embrio-rs[17]

- Playground for async on `no_std`
- Provides a custom async desugaring via a proc macro
- Not actively developed anymore according to author[15]
- Works very nice on the NRF51
- Used in the Polymer mechanical keyboard firmware[16]

[15] https://www.reddit.com/r/rust/comments/f0ckiv/bringing_asyncawait_to_embedded_rust/fgwh6ij/

[16] https://josh.robsonchase.com/rest-of-the-keyboard/

[17] https://github.com/Nemo157/embrio-rs

async — The `Future`

- Reusable executors?
- Thread safety with interrupts?

async — The `Future`

- Reusable executors?
- Thread safety with interrupts?
- RTIC may provide a solution for that ☺

More Resources

- The Rust Embedded WG Blog[18]
- The Rust Embedded Book[19]
- The Discovery book[20]
- Awesome Embedded Rust[21]

[18] https://rust-embedded.github.io/blog/

[19] https://docs.rust-embedded.org/book/

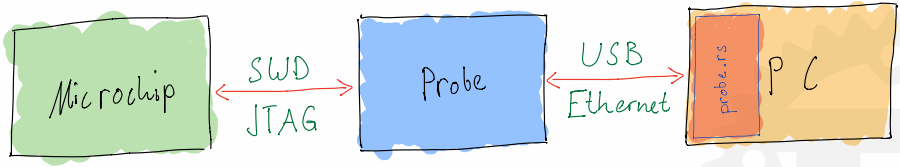
[20] https://docs.rust-embedded.org/discovery/index.html

[21] https://github.com/rust-embedded/awesome-embedded-rust

probe-rs

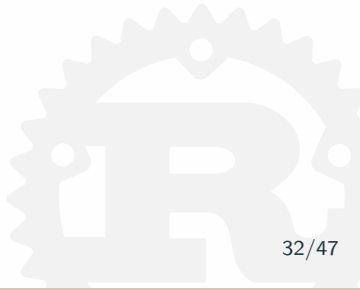
Embedded Development

- Software is written on a host instead of the target.
- The microchip has no direct feedback.
- A middleman is needed (soft- & hardware).
- probe-rs



probe-rs!

- Rust Library
- Easy to use
- Unified interface
- Developed on Github
- Talks to ARM and RISC-V cores (for now)
- Supports CMSIS-DAP, ST-Link, J-Link



Current State

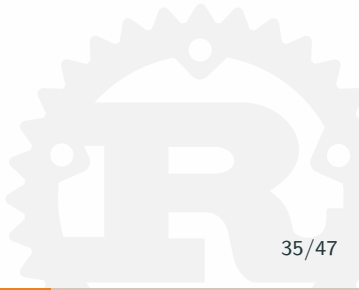
- Memory Access (read, write)
- CPU Manipulation (run, halt, step, reset, etc.)
- Breakpoints (set, reset)
- Flashing (ELF, BIN, HEX)

Memory Access

```rust
use probe_rs::Session;
use probe_rs::MemoryInterface;

let mut session = Session::auto_attach("nrf52")?;
let mut core = session.core(0)?;

// Read a single 32 bit word.
let word = core.read_word_32(0x2000_0000)?;
```

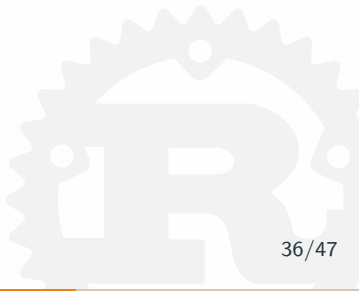
CPU Control

```rust
use probe_rs::Session;
use probe_rs::MemoryInterface;

let mut session = Session::auto_attach("nrf52")?;
let mut core = session.core(0)?;

// Reset the CPU.
core.reset()?;
```



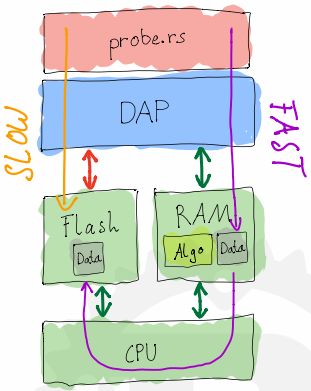
Breakpoint Manipulation

```rust
use probe_rs::Session;
use probe_rs::MemoryInterface;

let mut session = Session::auto_attach("nrf52")?;
let mut core = session.core(0)?;

// Set a breakpoint.
core.set_hw_breakpoint(0x1500_2000)?;
```



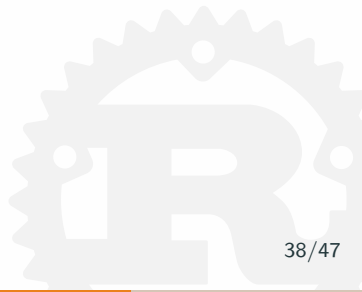
Flash Algorithms

- Used for fast flashing
- Used to circumvent slow flash access
- Small program loaded into RAM
- Copies data from RAM to Flash
- CMSIS-Packs
  - ARM standard
  - thousands of available targets



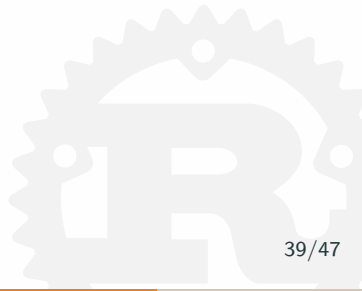
cargo flash

- slim cargo plugin
- for flashing targets
- to jumpstart in a project
- https://github.com/probe-rs/cargo-flash
- cargo install cargo-flash



cargo embed

- fat client embedded toolkit
- flashing
- RTT
- GDB
- more?
- highly configurable
- https://github.com/probe-rs/cargo-embed
- cargo install cargo-embed



probe_rs.await?;

VSCode

- VSCode plugin
- Microsoft DAP
- no GDB required
- Modern, extensive API (JSON)
- async
- working, with limitations
- https://github.com/probe-rs/vscode

ITM

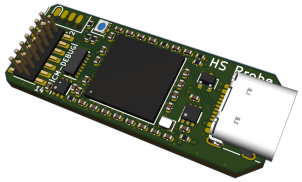
- powerful data streaming from the target
- ISR events
- memory access events
- custom binary data
- working, not on master yet
- https://github.com/probe-rs/probe-rs/pull/145

Custom Hooks

- use CMSIS-Pack hooks
- unlocking, special bytes, etc.
- uses custom WASM format
- poc, no PR yet

rs-probe

- open source probe
- pure rust firmware
- extremely fast (500 Mbit/s)
- uses standard ARM API
- can stream DAP, ITM and UART data
- https://github.com/korken89/hs-probe



Contribute

- https://probe.rs
- https://github.com/probe-rs/probe-rs
- #probe-rs:matrix.org on Matrix
- Questions & Bugreports very welcome
- PRs very welcome

Thank you!

https://coredump.ch

https://technokrat.ch/

Slides: https://github.com/rust-zurichsee/meetups/

Appendix

5. Appendix

Appendix

Alloc crate

alloc crate

- alloc allows you to use a subset of std (e.g. `Vec`, `Box`, `Arc`) in `#![no_std]` environments if the environment has access to heap memory allocation.[22]
- You need to define a `#[global_allocator]`[23]
- Check out `https://crates.io/crates/alloc-cortex-m`

[22] `https://github.com/rust-lang/rfcs/blob/master/text/2480-liballoc.md`

[23] `https://doc.rust-lang.org/stable/std/alloc/#the-global_allocator-attribute`

Appendix

svd2rust

svd2rust

- Every Cortex-M C vendor must provide an SVD (System View Descriptions) file
- SVD is an XML standard to describe peripheral registers
- svd2rust[24]: Generate Rust register maps (structs) from SVD files

[24] `https://github.com/rust-embedded/svd2rust`