Embedded Rust

Raphael Nestler (@rnestler)

2018-12-11

Rust Zürichsee Meetup

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

Hi! I'm Raphael (@rnestler).

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

Hi! I'm Raphael (@rnestler).

I live in Rapperswil

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

Hi! I'm Raphael (@rnestler).

I live in Rapperswil

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

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

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`).

Outline

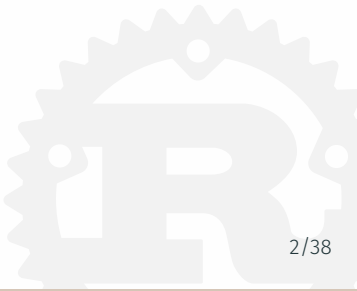
1. Embedded Programming
2. State of Embedded in 2018
3. Getting started – STM32F3 Discovery
4. RTFM
5. Future

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]



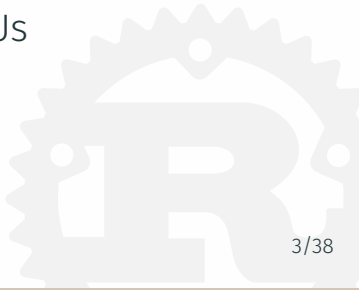
[1] https://barrgroup.com/Embedded-Systems/Glossary-E#embedded_system

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

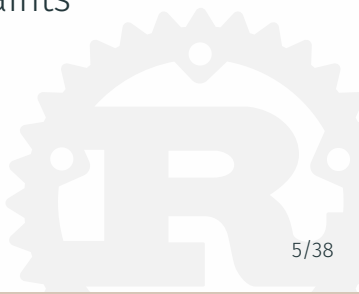


Why do they say it's hard?

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

Why could Rust be awesome for it?

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



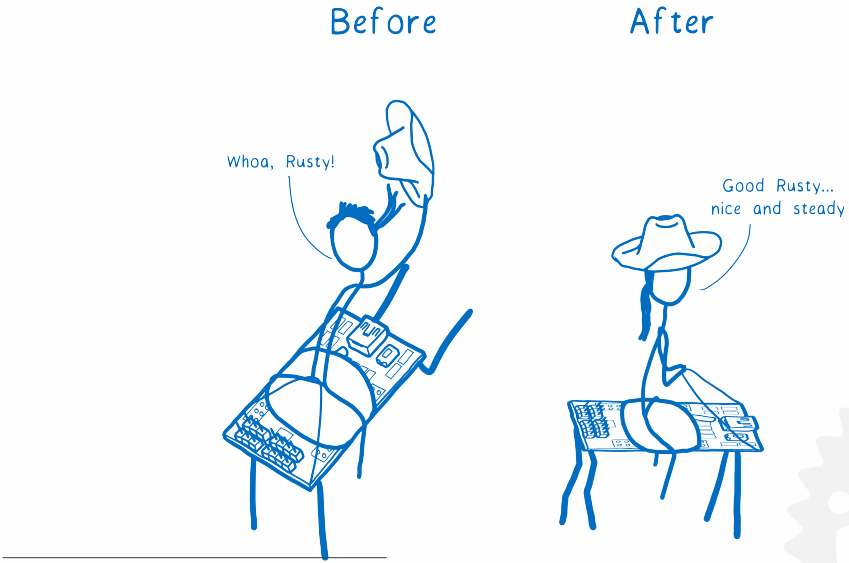
State of Embedded in 2018

You can use stable!

- Previously we needed nightly for `no_std` binaries
- Rust 1.6 `no_std / libcore` gets introduced[2]
- Rust 1.27 cortex-m embedded libraries possible on stable[3]
- Rust 1.30 embedded binaries possible on stable[4]
- I recommend to use Rust 1.31 2018 edition (Released recently)

[2] https://blog.rust-lang.org/2016/01/21/Rust-1.6.html
[3] https://twitter.com/japaricious/status/995633889858277376
[4] https://blog.rust-lang.org/2018/10/25/Rust-1.30.0.html

You can use stable![5]



[5] https://hacks.mozilla.org/2018/12/rust-2018-is-here/

No more lib core cross compiling

- Previously we needed to use xargo[6] to cross compile lib core
- Now Cortex-M targets are supported by rustup / cargo (thumbvxx-none-eabi)

[6] https://github.com/japaric/xargo

Collaborative Effort

- Rust Embedded Working Group
- Started in the beginning of 2018[7]
- Works on
  - Documentation (Blog posts, The Embedded Rust Book, ...)
  - Tooling (LLVM, rustc, ...)
  - Standartization in the ecosystem (embedded-hal, ...)

[7] https://rust-embedded.github.io/blog/2018-03-15-newsletter-1/

More Resources

- The Rust Embedded Book[8]
- The Embedded Bookshelf[9]
- The Discovery book[10]
- Awesome Embedded Rust[11]

[8] https://docs.rust-embedded.org/book/
[9] https://docs.rust-embedded.org/
[10] https://docs.rust-embedded.org/discovery/index.html
[11] https://github.com/rust-embedded/awesome-embedded-rust

Getting started – STM32F3 Discovery

Prerequisites

- ☐Target support by compiler
- ☐Libcore compiled for the target
- ☐A Peripheral Access Crate (PAC)
- ☐Runtime to setup micro controller

Prerequisites

- ☑Target support by compiler
- ☑Libcore compiled for the target
- ☐A Peripheral Access Crate (PAC)
- ☐Runtime to setup micro controller

svd2rust

- Every Cortex-M µC vendor must provide an SVD (System View Descriptions) file
- SVD is an XML standard to describe peripheral registers
- svd2rust[12]: Generate Rust register maps (structs) from SVD files
- Done for our µC family[13]

[12] https://github.com/japaric/svd2rust
[13] https://github.com/japaric/stm32f30x

Prerequisites

- ☑Target support by compiler
- ☑Libcore compiled for the target
- ☑A Peripheral Access Crate (PAC)
- ☐Runtime to setup micro controller

cortex-m, cortex-m-rt

- The cortex-m[14] crate gives access to common low level features of all Cortex-M devices.
- The cortex-m-rt[15] implements a runtime for Cortex-M µCs

[14] `https://github.com/rust-embedded/cortex-m`
[15] `https://github.com/rust-embedded/cortex-m-rt`

Prerequisites

- ☑Target support by compiler
- ☑Libcore compiled for the target
- ☑A Peripheral Access Crate (PAC)
- ☑Runtime to setup micro controller

Install Toolchain[16]

```
$ rustup override set 1.31.0
$ rustc --version
rustc 1.31.0 (abe02cefd 2018-12-04)
$ rustup target add thumbv7em-none-eabihf
$ rustup component add llvm-tools-preview
$ cargo install cargo-generate
$ cargo install cargo-binutils
```

[16] https://rust-embedded.github.io/book/intro/tooling.html

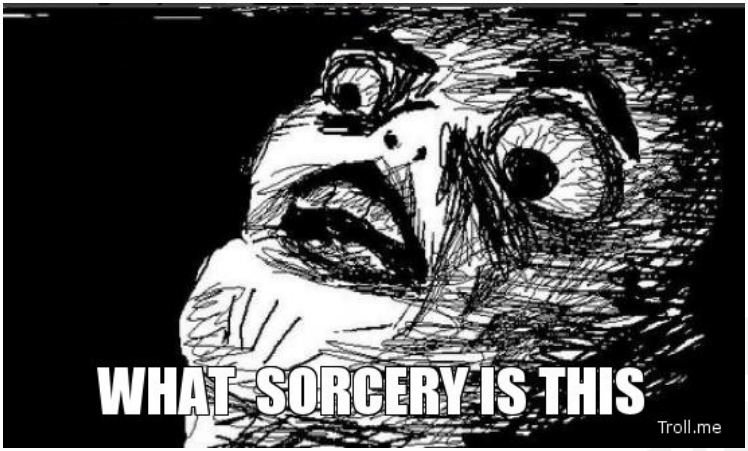
cortex-m-quickstart

```
$ cargo generate --git
↪  https://github.com/rust-embedded/cortex-m-quickstart --name
↪  hello-discovery
$ cd hello-discovery
$ vim .cargo/config
# Cortex-M4 -> thumbv7em
# (https://en.wikipedia.org/wiki/ARM_Cortex-M)
+[build]
+target = "thumbv7em-none-eabihf"
$ vim memory.x # from datasheet
CCRAM : ORIGIN = 0x10000000, LENGTH = 8K
FLASH : ORIGIN = 0x08000000, LENGTH = 256K
RAM : ORIGIN = 0x20000000, LENGTH = 40K
```

Building / Running

```
$ cargo build --example hello
# separate terminal
$ openocd openocd.cfg
# Optional if the above fails
$ lsusb|grep ST-LINK
Bus 001 Device 004: ID 0483:374b STMicroelectronics ST-LINK/V2.1
$ sudo chgrp input /dev/bus/usb/001/004
$ arm-none-eabi-gdb -x openocd.gdb -q
↪  target/thumbv7em-none-eabihf/debug/examples/hello
...
(gdb) continue
# Other terminal
Hello, world!
semihosting: *** application exited ***
```

Semihosting? Sorcery!



Hello World using semi hosting[17]

Kind of “system calls” into debugger

- Breakpoint instruction with a special tag. → Debugger gets notified.
- Two registers indicate which procedure call and points to a structure with arguments.
- The debugger reads the memory to retrieve the arguments and passes these on to the host’s procedure.
- The target’s CPU is unhalted and execution continues.

[17] `https://rust-embedded.github.io/book/start/semihosting.html`

Hello World explained

```
...
use cortex_m_semihosting::hprintln;
...
#[entry]
fn main() -> ! {
  hprintln!("Hello, world!").unwrap();
  loop {}
}
```

RTFM

RTFM

Real Time For the Masses

What is RTFM?

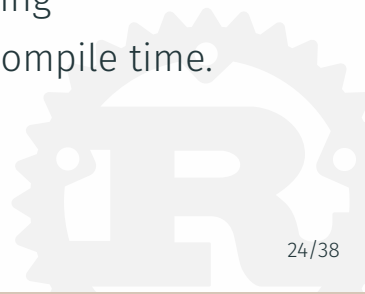
> *Framework based on the RTFM language created by the Embedded Systems group at Luleå University of Technology, led by Prof. Per Lindgren.*

The cortex-m-rtfm book [18]

[18] https://japaric.github.io/cortex-m-rtfm/book/preface.html

Features of RTFM

- Tasks triggered asynchronously or by the application
- Message passing between tasks
- Timer queue (schedule in the future, periodic)
- Priorization of tasks
- Efficient and data race free memory sharing
- **Deadlock free execution** guaranteed at compile time.
- Uses the hardware for scheduling



Minimal Example

Follow along: `https://github.com/rnestler/hello-rtfm-rs/tree/minimal-example`

```
#![no_std]
#![no_main]
extern crate panic_semihosting; // logs messages to the host stderr; requires a debugger

use cortex_m_semihosting::hprintln;
use rtfm::app;

#[app(device = stm32f30x)]
const APP: () = {
    #[init]
    fn init() {
        hprintln!("init").unwrap();
    }
    #[idle]
    fn idle() -> ! {
        hprintln!("idle").unwrap();
        loop {}
    }
};
```

Minimal Example

Follow along: `https://github.com/rnestler/hello-rtfm-rs/tree/minimal-example`

Output:

```
init
idle
```

Switching contexts

Follow along: `https://github.com/rnestler/hello-rtfm-rs/tree/first-demo`

```
#[app(device = stm32f30x)]
const APP: () = {
    #[init]
    fn init() {
        rtfm::pend(Interrupt::SPI1);
        hprintln!("init").unwrap();
    }
    #[idle]
    fn idle() -> ! {
        hprintln!("idle").unwrap();
        rtfm::pend(Interrupt::SPI1);
        hprintln!("idle 2").unwrap();
        loop {}
    }
    #[interrupt]
    fn SPI1() {
        static mut TIMES: u32 = 0;
        *TIMES += 1; // Safe access to local `static mut` variable
        hprintln!("SPI1 called {} time{}", *TIMES, if *TIMES > 1 { "s" } else { "" }).unwrap();
    }
};
```

Switching contexts

Follow along: `https://github.com/rnestler/hello-rtfm-rs/tree/first-demo`

Output:

```
init
SPI1 called 1 time
idle
SPI1 called 2 times
idle 2
```

Sharing Resources

Follow along: `https://github.com/rnestler/hello-rtfm-rs/tree/shared-resources`

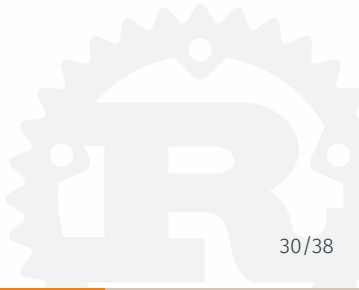
```
#[app(device = stm32f30x)]
const APP: () = {
    static mut SHARED: u32 = 0; // A resource
    #[init]
    fn init() {
        rtfm::pend(Interrupt::SPI1);
        rtfm::pend(Interrupt::SPI2);
        hprintln!("init").unwrap();
    }
    #[idle]
    fn idle() -> ! {
        hprintln!("idle").unwrap();
        // *resources.SHARED += 1; // doesn't compile
        loop {}
    }
    #[interrupt(resources = [SHARED])]
    fn SPI1() {
        *resources.SHARED += 1;
        hprintln!("SPI1: SHARED = {}", resources.SHARED).unwrap();
    }
    #[interrupt(resources = [SHARED])]
    fn SPI2() {
        *resources.SHARED += 1;
        hprintln!("SPI2: SHARED = {}", resources.SHARED).unwrap();
    }
};
```

Sharing Resources

Follow along: `https://github.com/rnestler/hello-rtfm-rs/tree/shared-resources`

Output:

```
init
SPI1: SHARED = 1
SPI2: SHARED = 2
idle
```



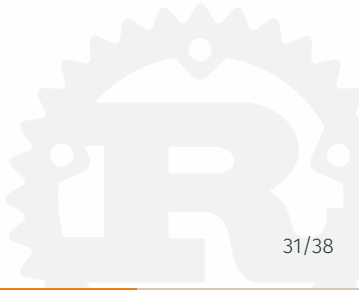
Late Resources

Follow along: `https://github.com/rnestler/hello-rtfm-rs/tree/late-resources`

Like static, but initialized in `init()`

```
...
const APP: () = {
    static mut LEDS: Leds = ();
    static mut DELAY: Delay = ();

    #[init]
    fn init() {
        ...
        LEDS = Leds::new(gpioe);
        DELAY = Delay::new(core.SYST, clocks);
    }
    #[idle(resources = [LEDS, DELAY])]
    fn idle() -> ! {
        let n = resources.LEDS.len();
        ...
    }
```



Late Resources

Demo Time!

Timer Queue

Follow along:
`https://github.com/rnestler/hello-rtfm-rs/tree/timer`

Needs a feature flag. Also delay won't be available.

```
[dependencies.cortex-m-rtfm]
git = "https://github.com/japaric/cortex-m-rtfm.git"
features= ["timer-queue"]
```

Timer Queue

Follow along:
`https://github.com/rnestler/hello-rtfm-rs/tree/timer`

```
...
#[init(schedule = [leds])]
fn init() {
    ...
    let now = Instant::now();
    schedule.leds(now + PERIOD.cycles()).unwrap();
    ...
}
#[task(resources = [LEDS], schedule = [leds])]
fn leds() {
    static mut curr: usize = 0;
    schedule.leds(scheduled + PERIOD.cycles()).unwrap();
    resources.LEDS[*curr].off();
    *curr += 1;
    if *curr > 7 {
        *curr = 0;
    }
    resources.LEDS[*curr].on();
}
```

Timer Queue

Demo Time!

Future

More targets

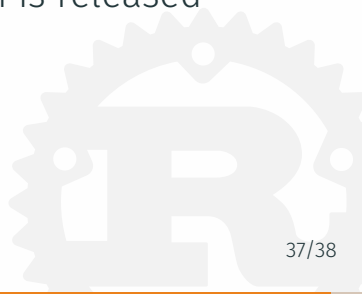
- Currently only Cortex-M are well supported
- Cortex-R
- MSP430
- RISCV

More targets

- Currently only Cortex-M are well supported
- Cortex-R
- MSP430
- RISCV
- AVR? There exits an LLVM and Rust compiler fork for it.
- ESP32? Currently people experiment compiling Rust to C and then to the ESP...

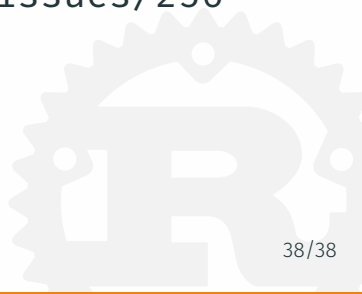
Ecosystem stabilization

- Currently still a lot of moving pieces (svd2rust, cortex-m crates, device crates, ...)
- Slowing down since the Rust 2018 edition is released



2019 Wishlist

- The Rust Embedded Working Group needs our input
- `https://github.com/rust-embedded/wg/issues/256`



Thank you!

https://coredump.ch

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