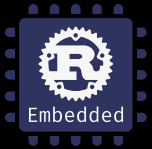
Embedded Rust, by example of RIOT-OS applications

Christian M. Amsüss <ca@etonomy.org>

2018-11-27



etonomy

RIOT

Embedded Devices

- 10kB – 1MB ROM
- 1kB – 100kB RAM
- Typical hardware: ARM Cortex-M3, eg. STM32

Embedded Devices

- allocation: static or on stack
- Someone needs to initialize the RAM
- CPU specific linker scripts
- Software shipped via hardware debugger or using a bootloader

Embedded Devices: On-board peripherals

- Clock(s)
- UARTs (eg. console)
- GPIO pins (eg. LEDs, buttons)
- SPI (eg. to access SD cards)

need to be set up, need drivers / file systems

Why Rust?

Why not?

Why Rust: My personal selection

- Fearless development
- Reusing code throughout the infrastructure

no_std

Options 1a: Bare metal

- `cortex-m-rt`
- peripherals wrapped from `svd2rust` (eg. `stm32f30x`)
- device drivers (eg. `stm32f30x-hal`)
- board support crate (eg. `f3`)

See `f3` crate for examples

Options 1b: RTFM

"Real-Time For The Masses"

- `cortex-m-rtfm`
- peripherals
- device drivers
- board support crate

More descriptive knowledge, fewer mutexes

Options 2: Full Rust operating system

Tock

- Operating system written in Rust
- Trusted (cooperative) and untrusted (preemtive) processes
- Network stack is WIP
- Limited hardware support

Options 3: RIOT-OS

- Operating system written in C
- Trusted processes (cooperative or preemtive)
- Mature network stack
- Large community
- Good hardware support

Which to pick?

Does it matter?

Abstractions

embedded-hal

embeded-hal traits

- GPIO
- SPI
- ADC
- I2C
- UART
- delays
- ...

covers usage, not initialization

embedded-hal driver example: ENC28J60

```
impl<E, SPI, NCS, INT, RESET> Enc28j60<...>
where
    SPI: spi::Transfer<u8, Error=E> + ...,
    NCS: OutputPin,
    INT: IntPin + InputPin,
    RESET: ResetPin,
{
  ...
}
```

Traits in general

Emulating a different network stack

```
impl<'a> jnet::Resize for
    &'a mut riot_sys::Pktsnip<Writable>
{
    fn truncate(&mut self, len: u16) {
        self.realloc_data(len as usize).unwrap();
    }
}
```

Translation at build time; no runtime overhead if concepts align

RIOT Operating System

https://riot-os.org

Recap: RIOT-OS

- Operating system written in C
- Trusted processes (cooperative or preemtive)
- Mature network stack
- Large community
- Good hardware support

`riot-sys`

bindgen

many unsafe functions and raw pointers

`riot-wrappers`

safe wrappers

Mutext, RwLock: like in `std::sync`

embedded-hal implementations

Examples

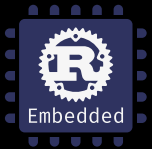
Recap

- What are embedded devices?
- Bare metal development is possible
- Choice of operating systems
- embedded-hal & co (Embedded Rust WG)
- RIOT Operating System
- riot-sys and riot-wrappers
- Go try it!

Questions?

Thanks for your attention

Slides and more links on
`http://christian.amsuess.com/presentations/2018/embedded-rust-riot/`



etonomy

RIOT