RIOT-rs

Re-Imagining RIOT in Rust

RIOT Summit – September 2024

Kaspar Schleiser

Agenda

1. Why?
2. How?
3. RIOT-rs Status
4. Community Aspects
5. Conclusions/Outlook

RIOT as we know it

Awesome developer experience
combining

- easy to get started
- lots of functionality available & integrated
- wide hardware support
- applications usually pretty portable

What’s the problem?

Inherent limits of C programming

- API design, abstraction, safety…
- Dealing with the toolchain mess
- Reliability issues

Bottlenecks in RIOT(-C)

- Peoplepower for system maintenance
- Peoplepower for CI

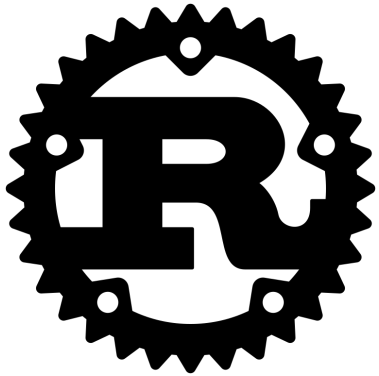
Enter Rust

The “new” kid on the block, challenging C…

… with a different trade-off combining:

- Built-in memory safety;
- High-level ergonomics;
- Low-level control;

With modern tooling …



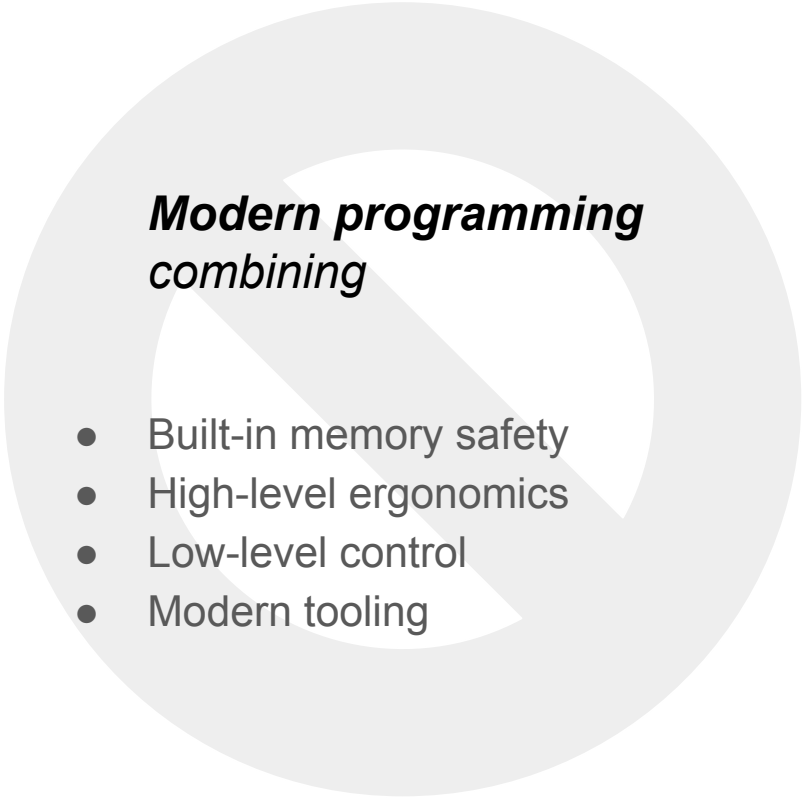
Recent Rust rant: see this post on Google Open Source Blog

Awesome developer experience
combining

- easy to get started
- lots of functionality available & integrated
- wide hardware support
- applications usually pretty portable

Modern programming
combining

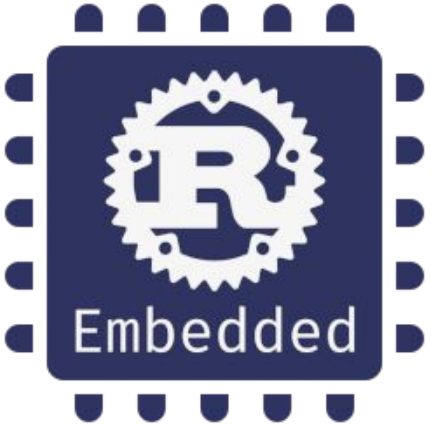
- Built-in memory safety
- High-level ergonomics
- Low-level control
- Modern tooling



Embedded Rust

➔ There’s even a very lively Embedded Rust open source community!

- tons of drivers, libs developed and maintained
- operating systems (Tock OS, Hubris),
- frameworks (Embassy, RTIC…)



So why not Embedded Rust as-is?

Bare-metal? Well… no.

Operating systems? Capable, but:

- Tock OS: Cortex-M only (until recently), MPU-dependency, rather “big” last time we checked, no async Rust
- Hubris: purpose-built for Oxide’s in-house needs, Cortex-M+MPU only, no async Rust

Framework? Interesting middle-ground.

- For instance: Embassy, RTIC
- That’s what we looked at in more details

Embassy? Case-study

- “Embassy is the next-generation framework for embedded applications. Write safe, correct and energy-efficient embedded code faster, using the Rust programming language, its async facilities, and the Embassy libraries.”

- long feature list: timers, real-time, low-power, networking, bluetooth, LoRa, USB, bootloader & DFU

- reasonable hardware support: nrf, rp2040, stm32, esp32

Embassy? Case-study

Why not (as is):

- more of a collection of building blocks
- high quality code, but quite low level
- lacking on portability and more complex examples

Awesome developer experience
combining

- easy to get started
- lots of functionality available & integrated
- wide hardware support
- applications usually pretty portable

Modern programming
combining

- Built-in memory safety
- High-level ergonomics
- Low-level control
- Modern tooling

We want it all!

Awesome developer experience
combining

- easy to get started
- lots of functionality available & integrated
- wide hardware support
- applications usually pretty portable

Modern programming
combining

- Built-in memory safety
- High-level ergonomics
- Low-level control
- Modern tooling

Bring memory safety to C ? No.
So Rust is the way. Let’s go!

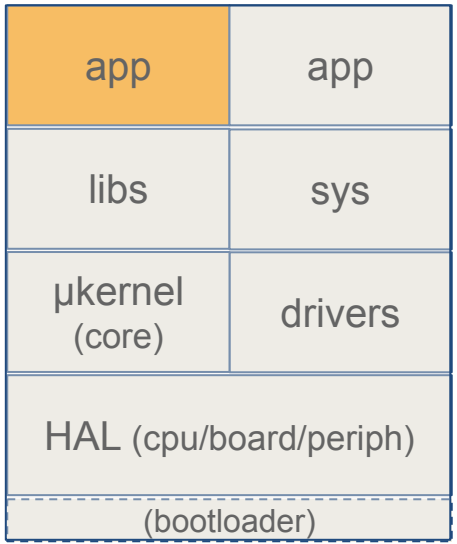
Agenda

1. Why?
2. How?
3. RIOT-rs Status
4. Community Aspects
5. Conclusions/Outlook

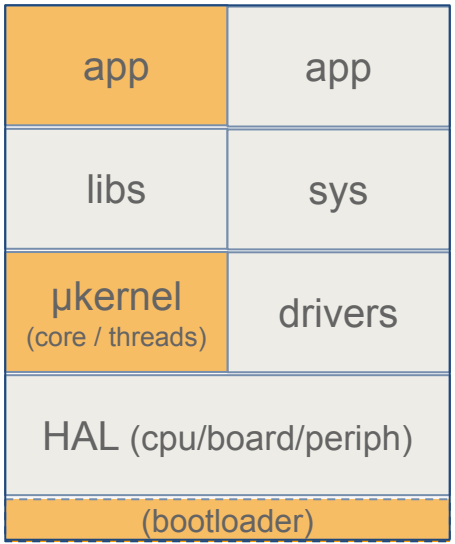
Road to Rust

2018 2019 2020 2021 2022 2023 2024

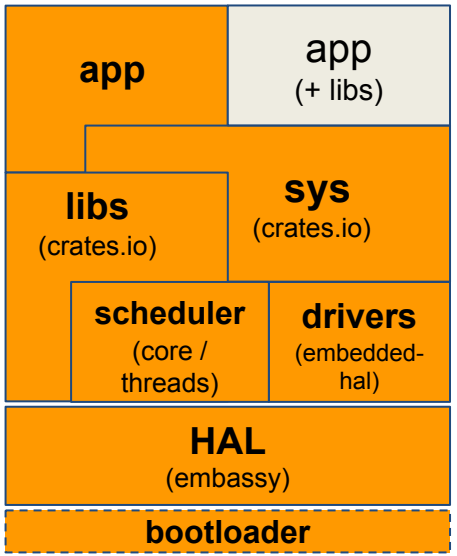
RIOT + Rust wrappers

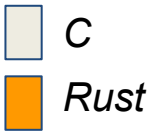


RIOT-fp prototypes



RIOT-rs





RIOT-rs Abstraction Example (1)

Example: USB stack

- Starting point:
    - Embassy: **15** versions of `usb_serial.rs`, **4** versions of `usb_ethernet.rs`, **each** with
        - MCU specific clock & USB peripheral setup
        - USB stack setup
        - USB class setup (serial or ethernet)
        - in case of ethernet, network stack setup
        - serial or TCP echo logic

RIOT-rs Abstraction Example (1)

RIOT-rs:

- cleanly separate MCU specific clock & USB peripheral setup
- provide shared USB stack setup
- there's **one** usb_serial example
- there's **no** usb_ethernet example
- USB classes, ethernet & network stack as ready-to-use modules

-> much increased reusability and portability

RIOT-rs Abstraction Example (2)

Example: Peripheral APIs, “embedded-hal”

- Starting point
  - “embedded-hal” is the Rust API for GPIO/I2C/SPI/…
  - many (e.g., sensor) drivers available
  - but, **initialization not part of it**
- RIOT-rs
  - we analyze embassy-nrf, embassy-rp, embassy-stm32, esp-rs APIs
  - we abstract peripheral initialization
  - we provide a unified API

  -> this enables portable peripheral-using applications

RIOT-rs High-Level Features (1)

Example: Random numbers

- Starting points:
  - Embassy: per-MCU RNG peripheral examples (**~8**)
  - Ecosystem: multiple PRNG algos, a random trait

- RIOT-rs:
  - abstracts RNG peripherals
  - chooses a usable PRNG
  - provides a high level API providing fast RNG and CSRNG

-> “random” just available as module

RIOT-rs High-Level Features (2)

Example: CoAP stack

- Starting point:
  - Embassy stops at TCP/UDP layer 4

- RIOT-rs:
  - Develops & integrates CoAP/EDHOC/OSCORE on embedded_nal by @chrysn

-> secure CoAP server can just be enabled

RIOT-rs Build System

- Starting point (Embassy):
  - mostly pure Cargo with board specific settings in Cargo.toml, .cargo/config.toml, …

- RIOT-rs:
  - wraps Cargo in laze
  - Cargo board specific settings get generated on-the-fly

    → this provides the equivalent of `make BOARD=foo …`

RIOT-rs in a nutshell

The general approach is to use Embassy as starting point, and:

- increase portability
- reduce boilerplate
- provide higher-level “turn-key” features
- (add other OS facilities)

Agenda

1. Why?
2. How?
3. RIOT-rs Status
4. Community Aspects
5. Conclusions/Outlook

RIOT-rs features (Summer 2023)

System: - async runtime - preemptive scheduler	Network stack:	Tooling:
Peripherals:	Integration: - non-portable “hello-world”	Supported MCUs / boards: - NRF52840DK

We started with basically nothing but an idea.

RIOT-rs features (Summer 2024)

System: - async runtime - preemptive scheduler - random PRNG/CSRNG	Network stack: - Ethernet / WiFi - IPv4 / IPv6 - ICMP/UDP/TCP/DHCPv4 - CoAP/OSCORE/EDHOC - HTTP server	Tooling: - unified debug logging - defmt
Peripherals: - GPIO - I2C - SPI - USB (ethernet, serial, HID)	Integration: - portable blinky - portable net examples - portable usb examples	Supported MCUs / boards: - NRF5x (some -DKs) - RP2040 (Rpi Pico (W)) - STM32 (some Nucleos) - RISC-V ESP32 (esp32c6)

Obviously we’re not done yet

Missing features include:

- IPv6 auto configuration
- IEEE80215.4/6lowpan
- power management
- software updates
- much more

But, feature-wise, we’re getting there.

Boilerplate?

RIOT-c minimal project

```
❯ cat Makefile main.c
```

File: **Makefile**

```
APPLICATION = hello-world
BOARD ?= native
RIOTBASE ?= /home/kaspar/src/riot
include $(RIOTBASE)/Makefile.include
```

File: **main.c**

```c
#include <stdio.h>
int main(void)
{
    printf("Hello World!");
    return 0;
}
```

RIOT-rs minimal project

```
❯ cat Cargo.toml laze.yml src/main.rs
```

File: **Cargo.toml**

```
[package]
name = "hello-world"
edition = "2021"
[dependencies]
riot-rs = { path = "../../src/riot-rs", features = ["threading"] }
riot-rs-boards = { path = "../../src/riot-rs-boards" }
```

File: **laze.yml**

```
apps:
```

File: **src/main.rs**

```
#![no_main]
#![no_std]
#![feature(type_alias_impl_trait)]
#![feature(used_with_arg)]
use riot_rs::debug::log::*;
#[riot_rs::thread(autostart)]
fn main() {
    info!("hello world");
}
```

Reliability?

- not enough exposure to tell yet
- it “feels” very robust
- one anecdote: Upon first successful compilation & flashing after integrating Rpi Pico W WiFi + HTTP server setup RIOT-rs kept running for months (until the office was moved).

-> reliability wise, promising

Rust overhead?

Code-size reality check:

	RIOT	RIOT-rs
blinky	4436b	3860b
minimal networking	31872b	32527b

(on nrf52840dk using usb ethernet)

RIOT-rs: `examples/embassy-net-udp` with echo logic removed, compiled with `laze build -b nrf52840dk -d defmt -d debug-console`
RIOT-c: `examples/gnrc_minimal`, modified to use `usbus_cdc_ecm` instead of `netdev_default`, removed `core_panic` printf calls, built with `BOARD=nrf52840dk LTO=1 DEVELHELP=0 make clean all`

We can have it all!

Awesome developer experience
combining

- easy to get started
- lots of functionality available & integrated
- wide hardware support
- applications usually pretty portable

Modern programming
combining

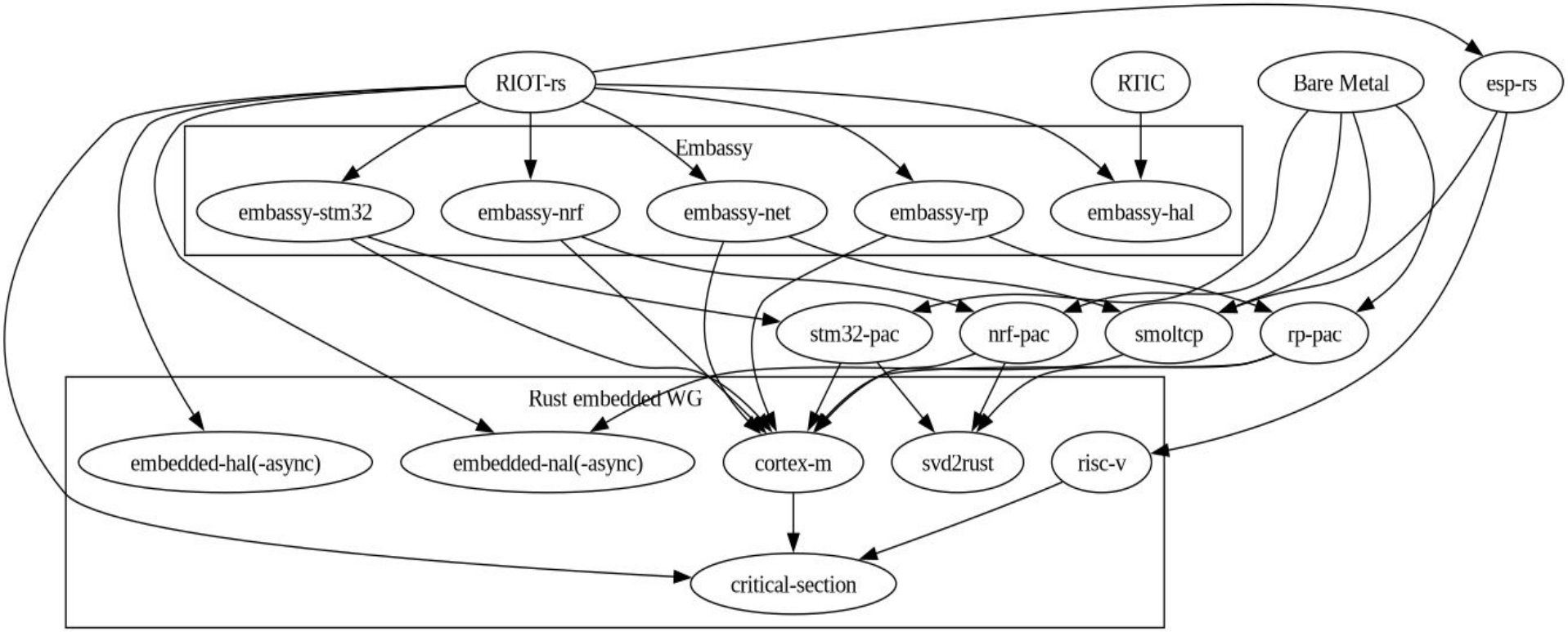
- Built-in memory safety
- High-level ergonomics
- Low-level control
- Modern tooling

Work to do but looking good!

Agenda

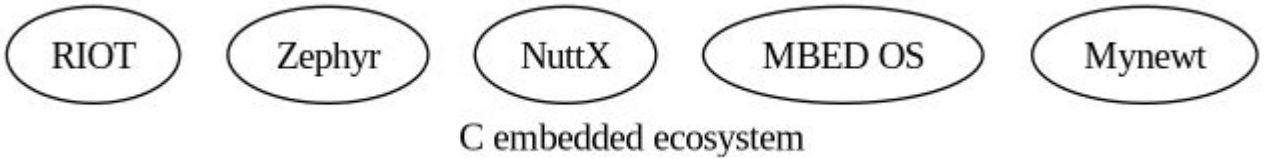
1. Why?
2. How?
3. RIOT-rs Status
4. Community Aspects
5. Conclusions/Outlook

We are not alone!



Rust embedded ecosystem (incomplete)

Compare to this:



Community Aspects

Rust embedded ecosystem is

- lively, growing & gaining traction
- distributed
- sharing lots of code, contributions flow in all directions
- Rust as a language is co-evolving!

Community aspects

Example:

1. RIOT-rs developed CoAP stack based on embedded-nal-async
2. smoltcp was missing some implementation
3. @chrysn did impl, PR’ed to smoltcp
4. RIOT-rs, embassy, RTIC, bare-metal can now all use it

… and much more in the other direction.

Agenda

1. Why?
2. How?
3. RIOT-rs Status
4. Community Aspects
5. Outlook & Conclusions

Outlook: around the corner

Around the corner:

- dual-core support
- finish peripheral API unification
- K/V config storage
- better integrate Cargo & laze
- HIL testing

Until next Summit:

- fix network stack feature holes (DHCPv6, 6LoWPAN …)
- secure software updates
- power management

Outlook: Code correctness

- Formal verification
  - Hax by Cryspen: regular Rust -> F* -> proofs (see https://hacspec.org/blog/posts/hax-v0-1/)
  - Hax integrated in RIOT-rs CI
  - bare minimum but big plans
  - RIOT-rs as guinea pig for Hax on embedded Rust

- Use in Safety critical systems
  - about to integrate Ferrocene Compiler (see https://ferrocene.dev/en/)
  - ISO26262 (ASIL D) and IEC 61508 (SIL 4) qualifications

  -> Potentially more reliability benefits

Conclusions / Perspectives with RIOT-rs

- **We can match the awesome sides of RIOT!**
  - Application portability, “batteries-included”, generally awesome DX
- **We can improve embedded Rust!**
  - Provide fully integrated system and distrib. (building on a decade of RIOT experience)
- **We can fix RIOT bottlenecks!**
  - Better share burden of HAL, periph/driver devel. & maintenance
  - Rationalize our broad, but uneven HW support
  - More modern tooling & ergonomics: ***increased productivity in the long-run!***
- **We can gain security guarantees**
  - Memory safety
  - Formal verification & qualified compilers

That’s all folks! Time for Q&A

RIOT-rs repository

FREIE UNIVERSITÄT

Inria informatics mathematics