Eine Einführung in Bare Metal Programmierung mit Rust

Rafael Bachmann
Embedded Software Developer

ESRLabs

Embedded Automotive (embedded Linux)

Netzwerkapplikationen (u.a. mit Rust)

We are hiring!

esr_

Part of **Accenture**

Bare Metal Embedded in 2 Minuten

- Direkte Interaktion mit der Umwelt
  - Analoge + Digitale Signale, Kommunikation
- Programmierung ohne Betriebssystem
  - Oft ohne Dynamischem Speicher
- Relativ kleine Speicher- und Leistungsressourcen
- “Embedded”: Computer ist Implementierungsdetail
  - Fahrkartenautomat? Türklinke? Gaspedal?

Bare Metal Embedded in 2 Minuten

- Nebenläufigkeit durch Interrupts
- Singleton Pattern: “Es gibt WIRKLICH nur 3 Serial Ports”
- Herausforderung: Zuordnung von Ressourcen
  - Timer 7 -> ADC Channel 3 -> DMA Peripheral to Memory
- Softwaremodule können Ressourcen **besitzen**
  - Regelungsmodul (z.b. ADC, Timer/PWM)
  - Sensortreiber (GPIOB12, UART1)

Rust in 3 Minuten

- Moderne Systemprogrammiersprache
- Typsystem erzwingt korrekten Umgang mit gemeinsamen Ressourcen
- Compiler + Tooling: freundlich, ergonomisch, einfach erweiterbar
- Vielfältige Bibliotheken durch

  Abhängigkeitsmanagement und Modulsystem

Rust in 3 Minuten – Ziele und Tradeoffs

- Binaries, Performance wie C
- Typsystem angelehnt an Haskell
- Ergonomie wie Python
- Tooling wie… Javascript/Node.js
- “Einzigartige” Lernkurve
- Compile-times wie C++ (oder schlimmer)

www.rust-lang.org

Rust in 3 Minuten – Garantien

- Keine SEGFAULTS
  - Panic: Strukturierte Dekonstruktion
- Kein Undefiniertes Verhalten
- Keine Data Races
- Zero-Cost Abstractions

www.rust-lang.org

Rust + Bare Metal = ❤️ ?

Welche Rolle spielen die Stärken von Rust für Bare Metal?

- Geringer Speicherverbrauch, kein Runtime oder Heap erforderlich
- Sichere Fehlerbehandlung ohne Heap oder Exceptions
- Eingebaute “Safety”
- Tooling (Toolchain Management, Testing, Flashing)
- Architekturspezifische Optimisation durch LLVM
- Aber: Manche Targets nicht in LLVM supported

Bibliotheken und Ökosystem

Grundlagen: **P**eripheral **A**bstraction **C**rates

- Pendant zu traditionellen C-Headern (register/offsets/...)
- API erzwingt Read/Write/ReadWrite/... Register
- Generiert aus Vendorspezifischen SVD files (ARM-Standard)

https://www.keil.com/pack/doc/CMSIS/SVD/html/index.html

https://github.com/rust-embedded/svd2rust

Grundlagen: **P**eripheral **A**bstraction **C**rates

```
I2C1.icr.reset();

I2C1.timingr.write(|w| w.bits(0x0000020B));
I2C1.cr2.modify(|_, w| w.autoend().set_bit());
I2C1.oar1.modify(|_, w| w.oa1en().clear_bit());
I2C1.oar2.modify(|_, w| w.oa2en().clear_bit());
I2C1.cr1.modify(|_, w| w.nostretch().clear_bit());
I2C1.cr1.modify(|_, w| w.pe().clear_bit());
```

Closures werden garantiert wegoptimiert (da kein Heap verfügbar)

Family- und Board Support Crates

Freundlichere APIs (auf Basis von PAC) für ADC, Timer, I2C, etc.

Chip-spezifisch: DMA, extension traits für besondere Peripherals

Ganze Familien (e.g. STM32F4) durch Feature Flags abgedeckt

github.com/stm32-rs/stm32f4xx-hal

github.com/stm32-rs/stm32f3xx-hal

github.com/nrf-rs/nrf-hal

github.com/rp-rs/rp-hal/tree/main/rp2040-hal

hal-implementation-crates

Family- und Board Support Crates

```
let sda_pin = pins.gpio18.into_mode::<I2C>();
let scl_pin = pins.gpio19.into_mode::<I2C>();
// let not_an_scl_pin = pins.gpio20.into_mode::<I2C>(); // fails

// Create the I²C struct, using the two pre-configured pins.
// Fails to compile if the pins are in the wrong mode,
// or if this I²C peripheral isn't available on these pins
let mut i2c = i2c1(pac.I2C1, sda_pin, scl_pin, 400.kHz());

// Write three bytes to the I²C device with 7-bit address 0x2C
i2c.write(0x2c, &[1, 2, 3]).unwrap();
```

This example from rp-hal on GitHub

Gemeinsam Genutzte Abstraktionen

Einige Abstrakte Schnittstellen zu SPI, I2C, ADC, Timers, Serial, ...

Warum?

- Wiederverwendbarkeit, Lernbarkeit, Portabilität
- Plattform-agnostische Treiber

Family- und Board Support Crates implementieren diese Schnittstellen!
Aber: Momentan noch Blocking API.

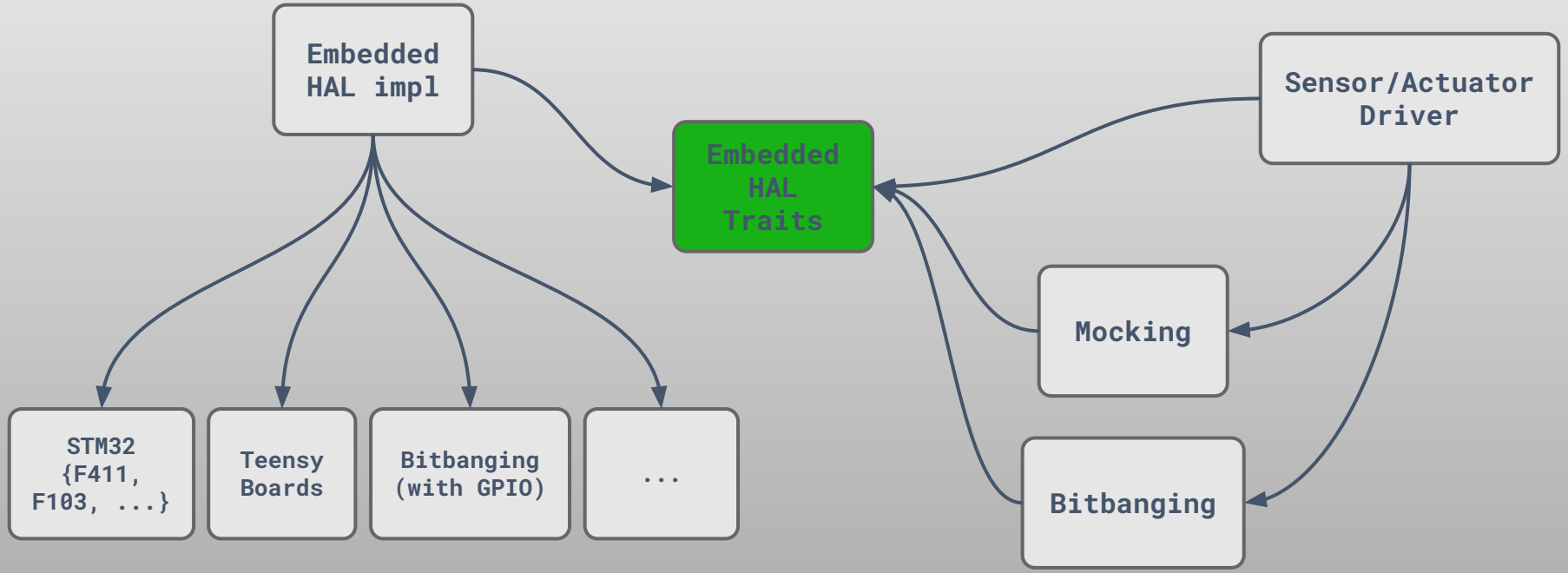
github.com/rust-embedded/embedded-hal

Portable Treiber

```
use bitbang_hal::i2c::*;

let timer_i2cbb1 = Timer::tim2(dp.TIM2, 200.khz(), &mut rcc);

// Configure I2C with 100kHz rate
let i2cbb1 = I2cBB::new(i2cbb1_scl, i2cbb1_sda, timer_i2cbb1);

let mut sdp8xx1 = Sdp8xx::new(i2cbb1, 0x25, delay.clone());
```

Bitbanging Code

Embedded HAL Vogelperspektive



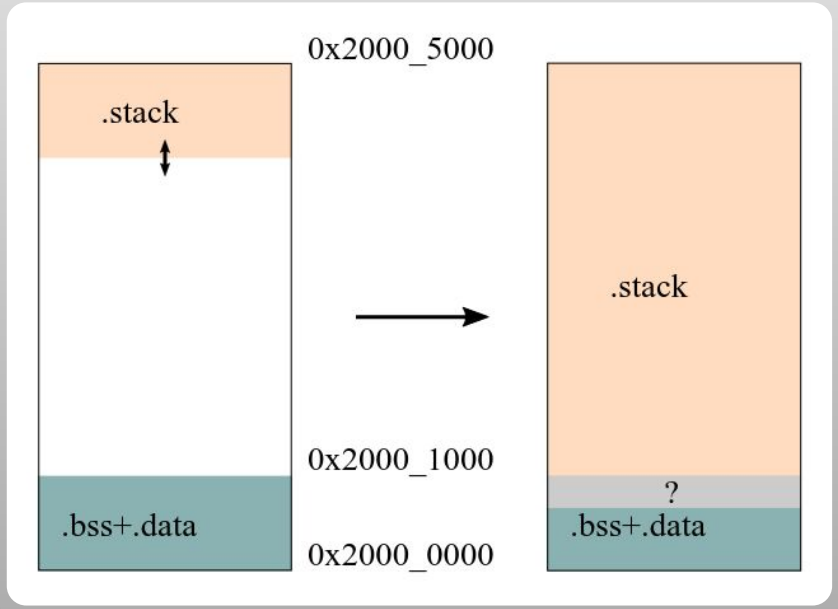
Tooling

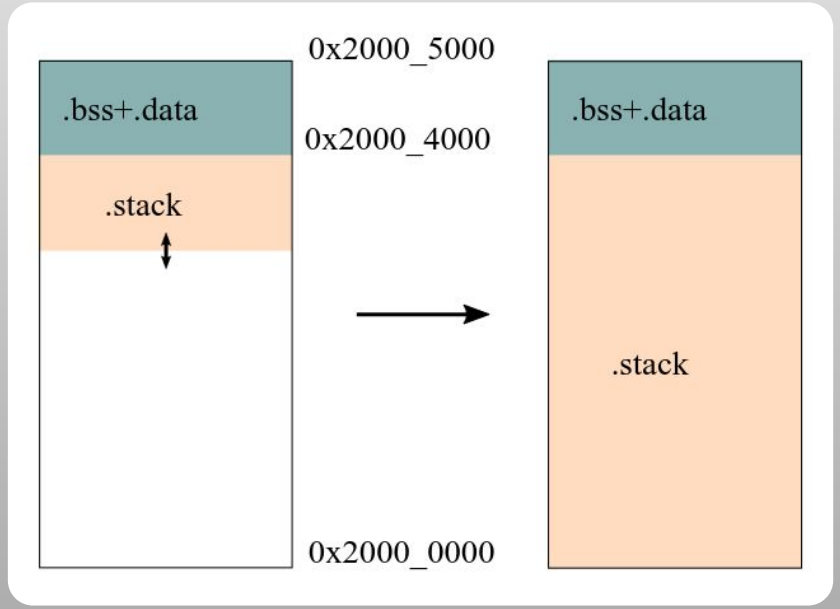
Rust Embedded Tooling: flip-link

knurling-rs

ADC++

[flip-link](): vertausche .stack and .data+.bss: trigger **HardFault** on stack overflow





Rust Embedded Tooling: defmt

knurling-rs

ADC++

- defmt: Deferred Formatting erst auf dem Host + “komprimierte” Strings
- Nur Rohdaten werden transferiert, keine formatierten Strings
- Nicht **“temperature is {}”**, sondern **ID**, die auf dem Host bekannt ist
- Full-featured logging (levels, timestamps, panic/assert print, …)

Framework	`.text`	relative size	`.rodata`	relative	`.text`+`.rodata`	relative
`core::fmt`	10348	1.0	3840	1.0	14188	1.0
`defmt`	1272	0.1229	360	0.0938	1632	0.1150

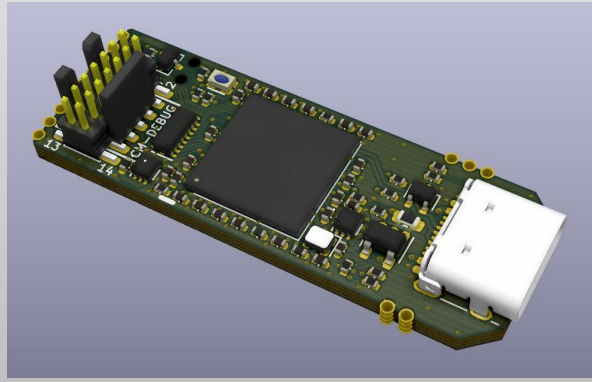
Rust Embedded Tooling: probe-rs

knurling-rs

ADC++

probe-rs: Rust Toolset, Interaktion mit MCUs über debug probes

- Arm + Risc-V supported (SWD + JTAG)
- Flash, debug, inspect core, dump memory, stacktrace
- Microsoft DAP support: Editor-unabhängiges Debugging



https://github.com/probe-rs/hs-probe

probe.rs

Rust Embedded Tooling: probe-rs

knurling-rs

ADC++

```rust
use probe_rs::Probe;

// Get a list of all available debug probes.
let probes = Probe::list_all();

// Use the first probe found.
let probe = probes[0].open()?;

// Attach to a chip.
let session = probe.attach("nrf52")?;

// Select a core.
let core = session.core(0)?;

// Halt the attached core.
core.halt()?;
```

Rust Embedded Tooling: cargo-bloat

```
> cargo bloat --release
Analyzing target/thumbv7em-none-eabi/release/client

File  .text    Size          Crate Name
0.1%  10.4%  2.1KiB      rtic_core <(T1,T2,T3,T4,T5,T6,T7)
0.0%   4.5%    962B         shared shared::setup_radio_with_payload_len
0.0%   4.4%    932B      rtic_core <(T1,T2,T3,T4,T5,T6)
0.0%   3.4%    720B            std core::fmt::Formatter::pad
0.0%   2.2%    476B  stm32wlxx_hal stm32wlxx_hal::rtc::Rtc::set_date_time
0.0%   2.0%    428B      defmt_rtt <Logger as defmt::traits::Logger>::write
0.0%   2.0%    426B      rtic_core <(T1,T2,T3,T4,T5,T6)
0.0%   1.8%    388B            std core::fmt::write
0.0%   1.7%    352B  stm32wlxx_hal stm32wlxx_hal::rcc::sysclk
0.0%   1.7%    350B  stm32wlxx_hal <Status as defmt::traits::Format>::format
```

Rust Embedded Tooling: Tarpaulin

Einfache Line Coverage Analysis

```
May 16 23:31:18.162  INFO cargo_tarpaulin::report: Coverage Results:
|| Tested/Total Lines:
|| src/command.rs: 12/14
|| src/lib.rs: 95/125
|| src/product_info.rs: 31/34
|| src/sample.rs: 28/31
|| src/test.rs: 162/163
||
89.37% coverage, 328/367 lines covered
```

crates.io/crates/cargo-tarpaulin

Rust Embedded Tooling: Tarpaulin

github.com/barafael/cd74hc4067/blob/main/coverage.pdf

```
impl<P, E> CD74HC4067<P, E, EnabledState>
where
    P: OutputPin,
    P: OutputPin,
    P: OutputPin,
    P: OutputPin,
    E: OutputPin,
{
    /// Disable the mux display by pulling `pin_enable` high
    pub fn disable(mut self) -> Result<CD74HC4067<P, E, DisabledState>, Error<P, E>> {
        self.pin_enable.set_high().map_err(Error::EnablePinError)?;
        Ok(CD74HC4067 {
            pin_0: self.pin_0,
            pin_1: self.pin_1,
            pin_2: self.pin_2,
            pin_3: self.pin_3,
            pin_enable: self.pin_enable,
            state: PhantomData::<DisabledState>,
        })
    }
}
```

Rust Embedded Tooling: Proptest + Mocking

```
#[test]
fn fuzz(mut bytes in vec(0..255u8, 9)) {
    ...
    let expectations = [
        Transaction::write(...),
        ...
    ];
    let sdp = Sdp8xx::new(I2cMock::new(&expectations), 0x10, DelayMock);
    let mut sampling = sdp.start_sampling_differential_pressure(true).unwrap();
    let _result = sampling.read_continuous_sample();
    let sdp = sampling.stop_sampling().unwrap();
    sdp.release().done();
}
```

https://crates.io/crates/proptest

Beispiel auf GitHub

Beispielprojekt:
LoRa Module Driver

LoRa Transmitter/Receiver

LoRa: Stromsparende Funktechnik mit großer Reichweite

Ebyte E32 Module: bieten vereinfachte Schnittstelle zu SemTech Radios

Angeblich 8Km Reichweite mit E32-433T30D, bei 433MHz (ISM-Band)

Transceiver: Sender + Empfänger

Produkt auf ebyte.com

LoRa Transmitter/Receiver

github.com/barafael/ebyte-e32-rs

`#[no_std]` Treiber für Ebyte E32 LoRa Module

- Embedded-Hal: Serial Peripheral + ein paar GPIOs
- Mocking mit embedded-hal-mock
- Property-Based Testing mit proptest
- Mutation Testing mit cargo-mutants
- Konfiguration als Datenstruktur dargestellt

LoRa Transmitter/Receiver: Parameter

```
#[derive(Debug                                                    )]


pub enum BaudRate {
    Bps1200,
    Bps2400,
    Bps4800,

    Bps9600,
    Bps19200,
    Bps38400,
    Bps57600,
    Bps115200,
}
```

LoRa Transmitter/Receiver: Parameter

```rust
#[derive(Debug, Copy, Clone, PartialEq, Eq, SmartDefault)]
#[cfg_attr(test, derive(proptest_derive::Arbitrary))]
#[cfg_attr(feature = "arg_enum", derive(clap::ArgEnum))]
pub enum BaudRate {
    Bps1200,
    Bps2400,
    Bps4800,
    #[default]
    Bps9600,
    Bps19200,
    Bps38400,
    Bps57600,
    Bps115200,
}
```

This code on GitHub

LoRa Transmitter/Receiver: Parameter

```
#[derive(Debug                                    )]

pub struct Parameters {
    pub address: u16,
    pub channel: u8,

    pub uart_rate: BaudRate,
    ...
}
```

LoRa Transmitter/Receiver: Parameter

```
#[derive(Debug, Clone, PartialEq, Eq, TypedBuilder)]
#[cfg_attr(test, derive(proptest_derive::Arbitrary))]
pub struct Parameters {
    pub address: u16,
    pub channel: u8,
    #[builder(default)]
    pub uart_rate: BaudRate,
    ...
}
```

This code on GitHub

LoRa Transmitter/Receiver: I/O

```rust
pub fn model_data(&mut self) -> Result<ModelData, Error> {
    Program::set_pins(&mut self.aux, &mut self.m0, &mut self.m1);
    let result = self.read_model_data();
    Normal::set_pins(&mut self.aux, &mut self.m0, &mut self.m1);
    result
}
```

This code on GitHub

LoRa Transmitter/Receiver: I/O

```rust
fn read_model_data(&mut self) -> Result<ModelData, Error> {
    block!(self.serial.write(0xC3)).map_err(|_| Error::SerialWrite)?;

    let save = block!(self.serial.read()).map_err(|_| Error::SerialRead)?;
    let model = block!(self.serial.read()).map_err(|_| Error::SerialRead)?;
    ...

    if save == 0xC3 {
        Ok(ModelData {
            model,
            version,
            features,
        })
    } else {
        Err(Error::ReadModelData)
    }
}
```

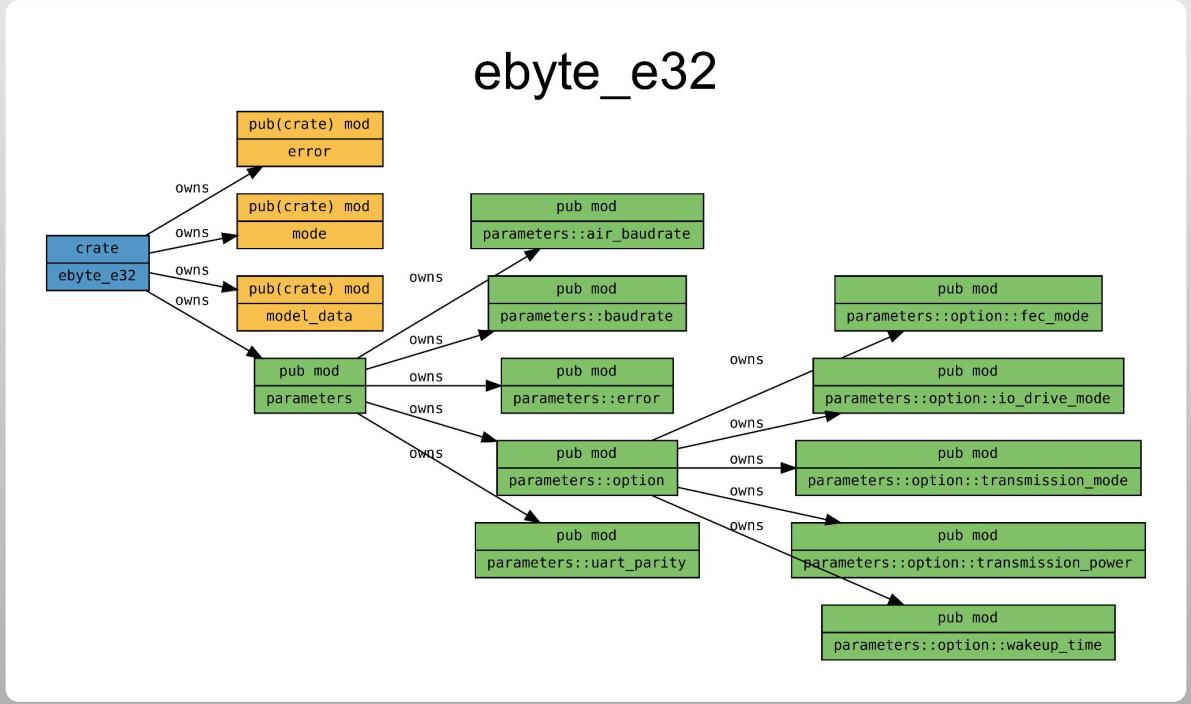
This code on GitHub

LoRa Transmitter/Receiver: Usage

```
let ebyte = Ebyte::new(serial, aux, m0, m1, delay).unwrap();
let mut params = ebyte.read_parameters().unwrap();

params.air_rate = AirBaudRate::Bps300;

ebyte
    .set_parameters(&params, Persistence::Temporary)
    .unwrap();

loop {
    delay_tim5.delay_ms(5000u32);
    rprintln!("Sending it!");
    ebyte.write_buffer(b"it").unwrap();
}
```

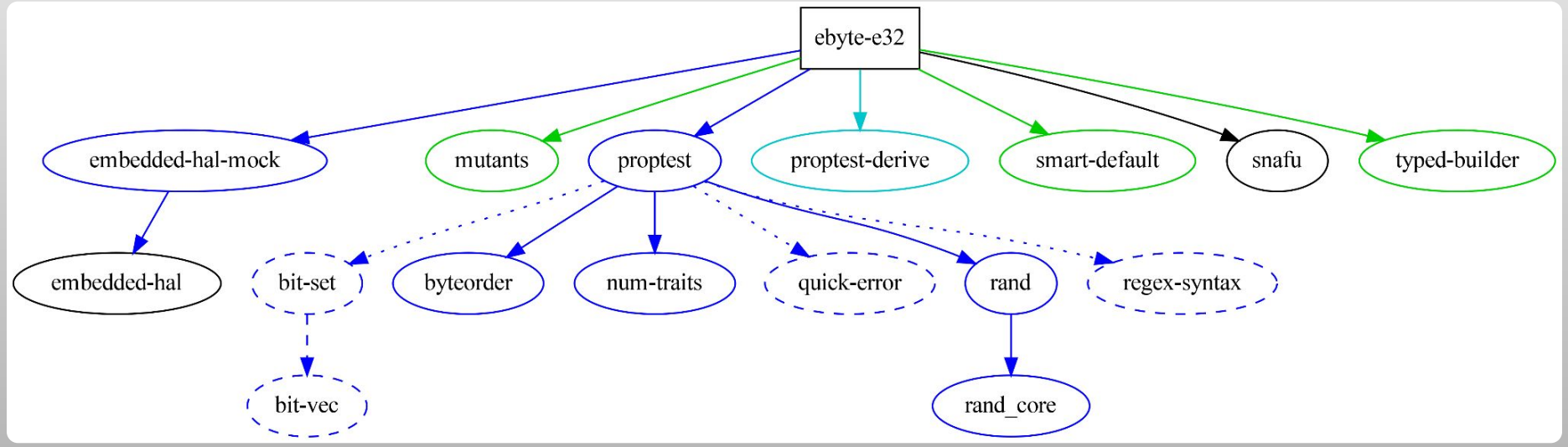
This code on GitHub

LoRa Transmitter/Receiver Module Structure




```
cargo modules generate graph > mods.dot
```

LoRa Transmitter/Receiver Dependencies




```
cargo depgraph > deps.dot
```

Raspberry Pi LoRa Transmitter/Receiver (CLI/GUI)

So viele Konfigurationen. Wie am besten testen? CLI und GUI.

- Deklarative CLI Definition via `clap`
- Generierte GUI mit `klask` (nutzt `clap`)
- Cross-Compilation für Raspberry Pi mit `cross` (Docker):

```
cross build --target armv7-unknown-linux-musleabihf
```

Raspberry Pi LoRa Transmitter/Receiver (CLI)

```
#[derive(Debug                                          )]

pub struct App {
    /// Module Address (16 Bit).

    pub address: u16,

    /// UART Baudrate.

    pub uart_rate: BaudRate,
    ...
}
```

Raspberry Pi LoRa Transmitter/Receiver (CLI)

```rust
#[derive(Debug, Clone, PartialEq, Eq, clap::Parser)]
#[clap(author, version, about, long_about = None)]
pub struct App {
    /// Module Address (16 Bit).
    #[clap(short, long, required = true)]
    pub address: u16,

    /// UART Baudrate.
    #[clap(arg_enum, long, required = false, ignore_case(true))]
    pub uart_rate: BaudRate,
    ...
}
```

This code on GitHub

Raspberry Pi LoRa Transmitter/Receiver (CLI)

Fields

`address: u16`
Module Address (16 Bit).

`channel: u8`
Channel (8 Bit).

`persistence: Persistence`
Whether settings should be saved persistently on the module.

`uart_parity: Parity`
UART Parity.

`uart_rate: BaudRate`
UART Baudrate.

`air_rate: AirBaudRate`
Air Baudrate.

`transmission_mode: TransmissionMode`
Transmission Mode.

`io_drive_mode: IoDriveMode`
IO drive Mode for AUX pin.

`wakeup_time: WakeupTime`
Wireless Wakeup Time.

`fec: ForwardErrorCorrectionMode`
Forward Error Correction Mode.

`transmission_power: TransmissionPower`
Transmission Power.

Enum ebyte_e32::parameters::uart_parity::Parity

source · [-]

```
pub enum Parity {
    None,
    Odd,
    Even,
}
```

Raspberry Pi LoRa Transmitter/Receiver (CLI)

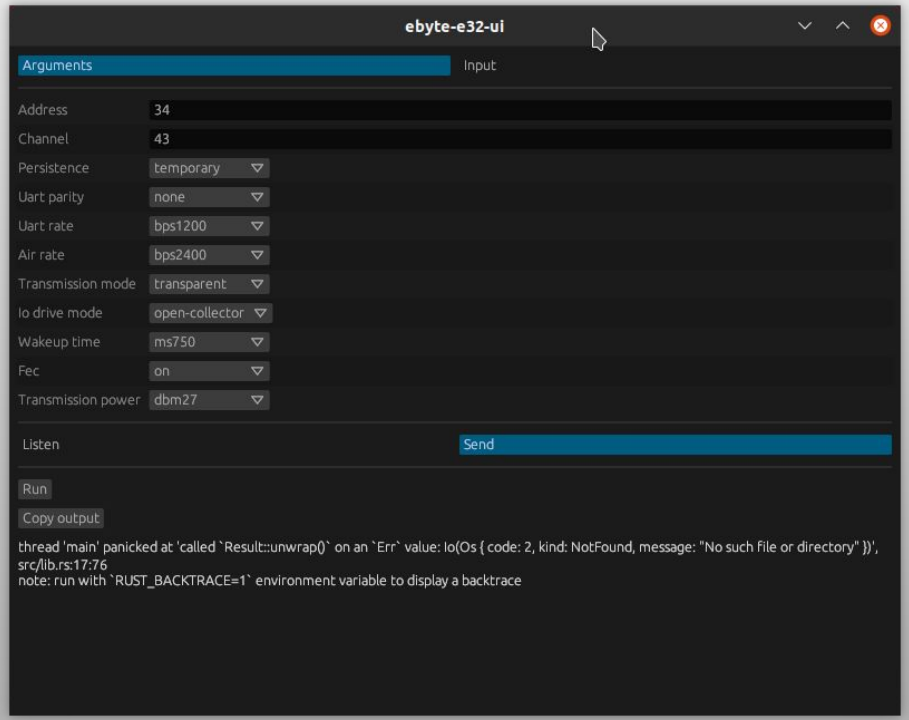
```
ebyte-e32-cli 0.1.0

USAGE:
    ebyte-e32-cli [OPTIONS] --address <ADDRESS> --channel <CHANNEL> <SUBCOMMAND>

OPTIONS:
    -a, --address <ADDRESS>
            Module Address (16 Bit)

        --air-rate <AIR_RATE>
            Air Baudrate [default: bps2400] [possible values: bps300, bps1200, bps2400, bps4800,
            bps9600, bps19200]

    -c, --channel <CHANNEL>
            Channel (8 Bit)
```

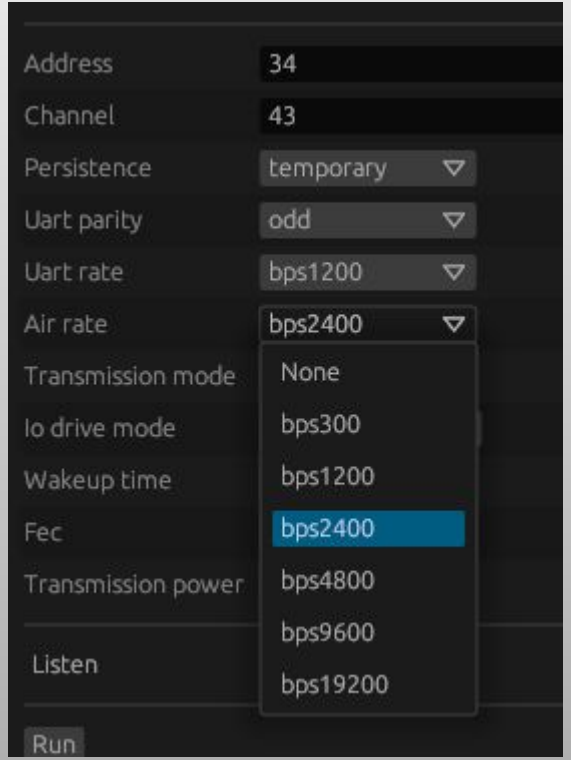
Raspberry Pi LoRa Transmitter/Receiver (GUI)

```rust
fn main() {
    klask::run_derived::<App, _>(Settings::default(), process);
}
```

Funktion `process`: on GitHub

Raspberry Pi LoRa Transmitter/Receiver (GUI)





Raspberry Pi LoRa Transmitter/Receiver

```
~/ebyte-e32-ui main *3 ?2 cargo audit
    Fetching advisory database from `https://github.com/RustSec/advisory-db.git`
      Loaded 416 security advisories (from /home/rafael/.cargo/advisory-db)
    Updating crates.io index
    Scanning Cargo.lock for vulnerabilities (228 crate dependencies)
Crate:         xcb
Version:       0.10.1
Title:         Multiple soundness issues
Date:          2021-02-04
ID:            RUSTSEC-2021-0019
URL:           https://rustsec.org/advisories/RUSTSEC-2021-0019
Solution:      Upgrade to >=1.0
Dependency tree:
xcb 0.10.1
└── x11-clipboard 0.5.3
    └── copypasta 0.7.1
        └── egui-winit 0.16.0
            ├── egui_glium 0.16.0
            │   └── eframe 0.16.0
            │       └── klask 1.0.0
            │           └── ebyte-e32-ui 0.1.0
            └── eframe 0.16.0

error: 1 vulnerability found!
```

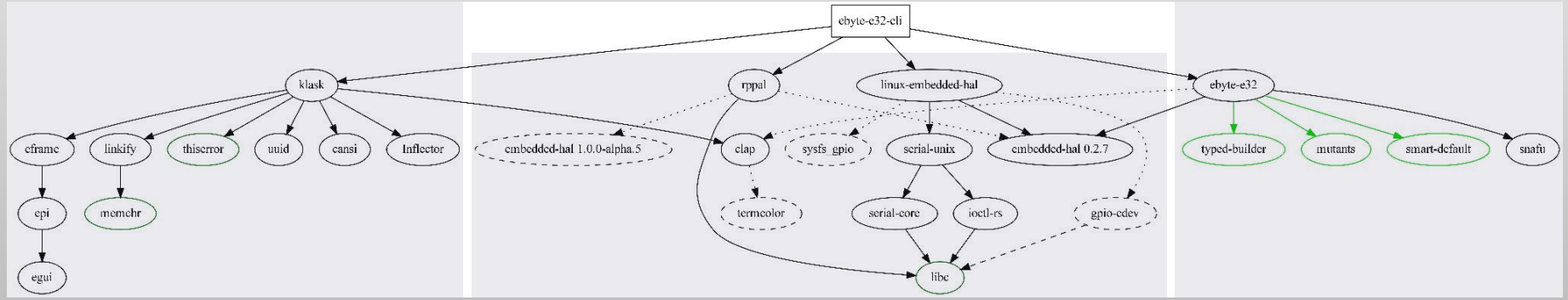
cargo audit

Raspberry Pi LoRa Transmitter/Receiver (CLI/GUI)

GUI

HW Support

Driver



github.com/barafael/ebyte-e32-rs

github.com/barafael/ebyte-e32-ui

RTIC:
Real-Time
Interrupt-driven
Concurrency

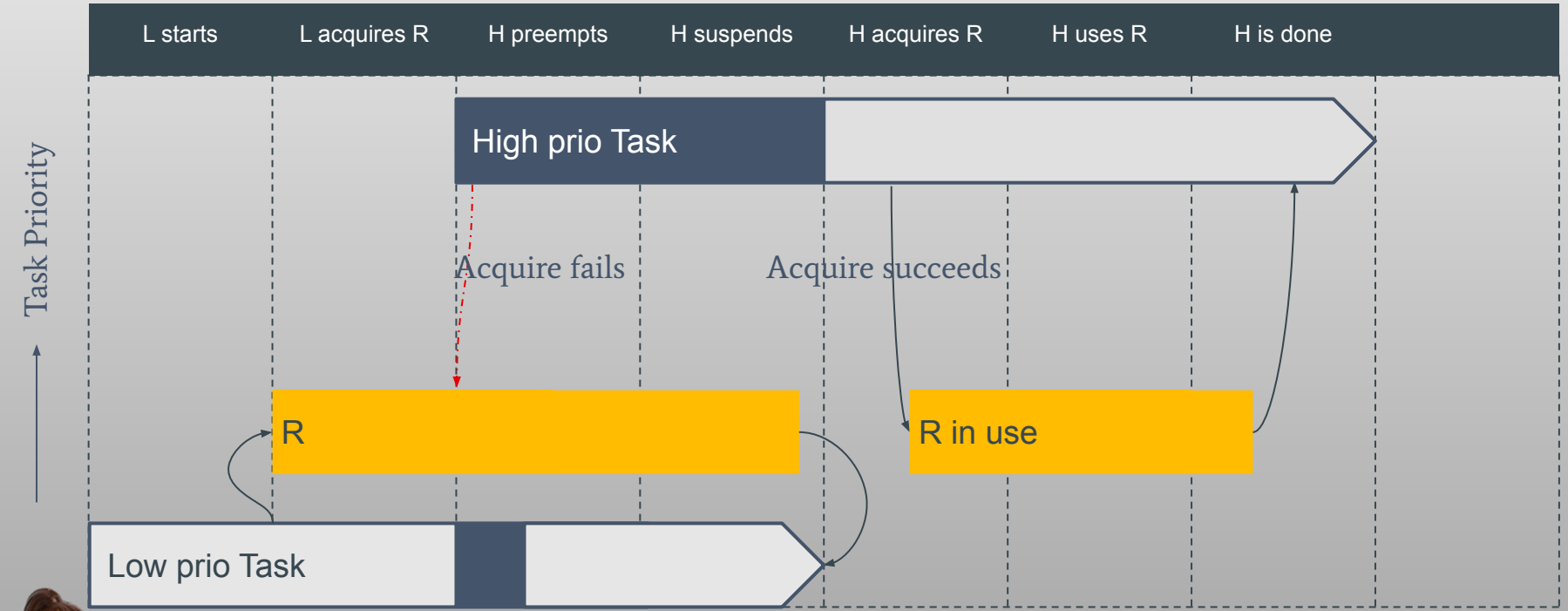
RTIC: Real-Time Interrupt-Driven Concurrency

- Framework für Event-driven Realtime Applikationen (aber kein RTOS)
- Run-to-completion tasks (einfach interrupt handlers)
- Preemptive Multitasking ohne Software Scheduler
  - ARM NVIC wird als Scheduler genutzt
- System verbietet deadlocks (statisch)
- System verbietet priority inversion (**Priority Ceiling Protocol**)
- Software-triggered tasks, timer queue, message passing

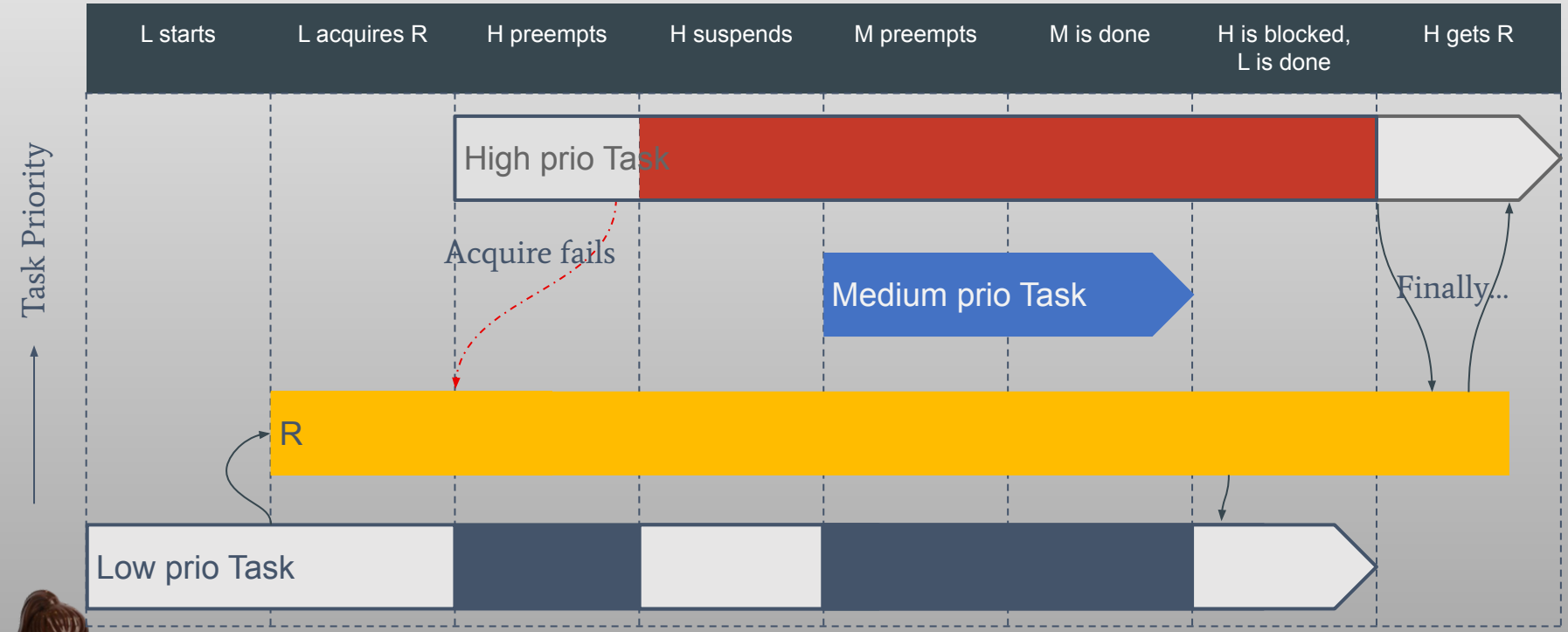
rtic.rs

minimal app

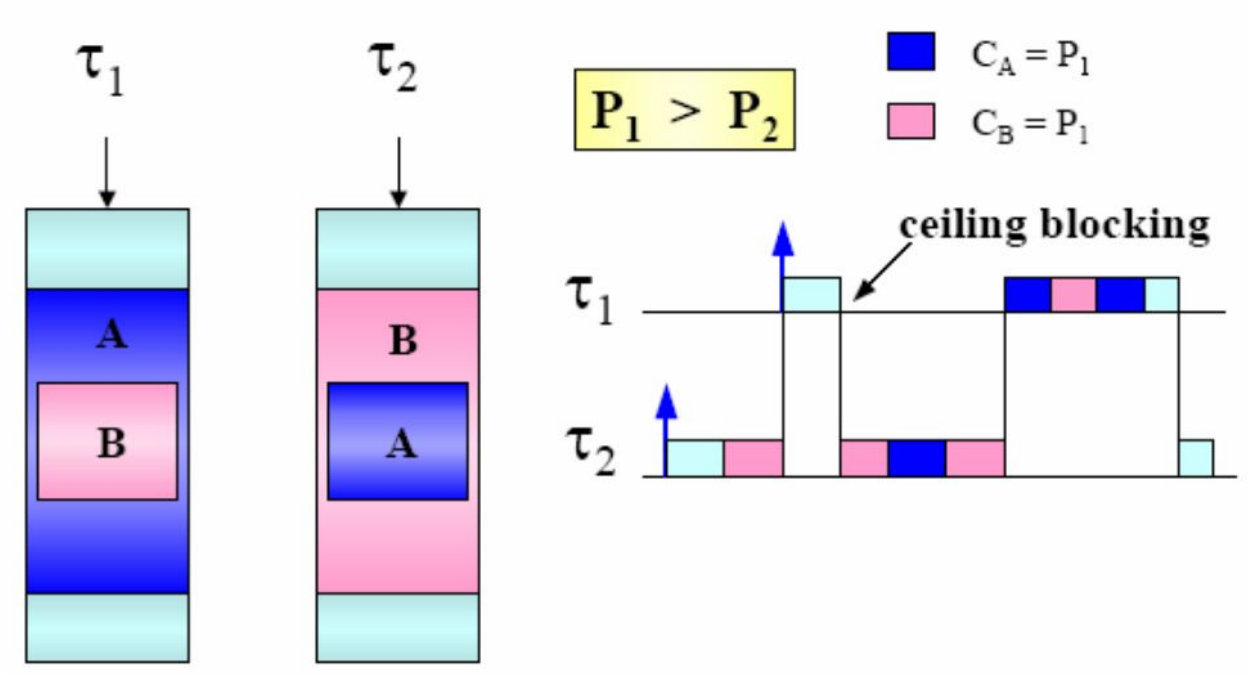
Preemptive Scheduling mit Prioritäten



Priority Inversion



Deadlock



J. Ras and A. M. K. Cheng, "An Evaluation of the Dynamic and Static Multiprocessor Priority Ceiling Protocol and the Multiprocessor Stack Resource Policy in an SMP System," 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium, 2009, pp. 13-22, doi: 10.1109/RTAS.2009.10.

RTIC: Real-time Interrupt-driven Concurrency

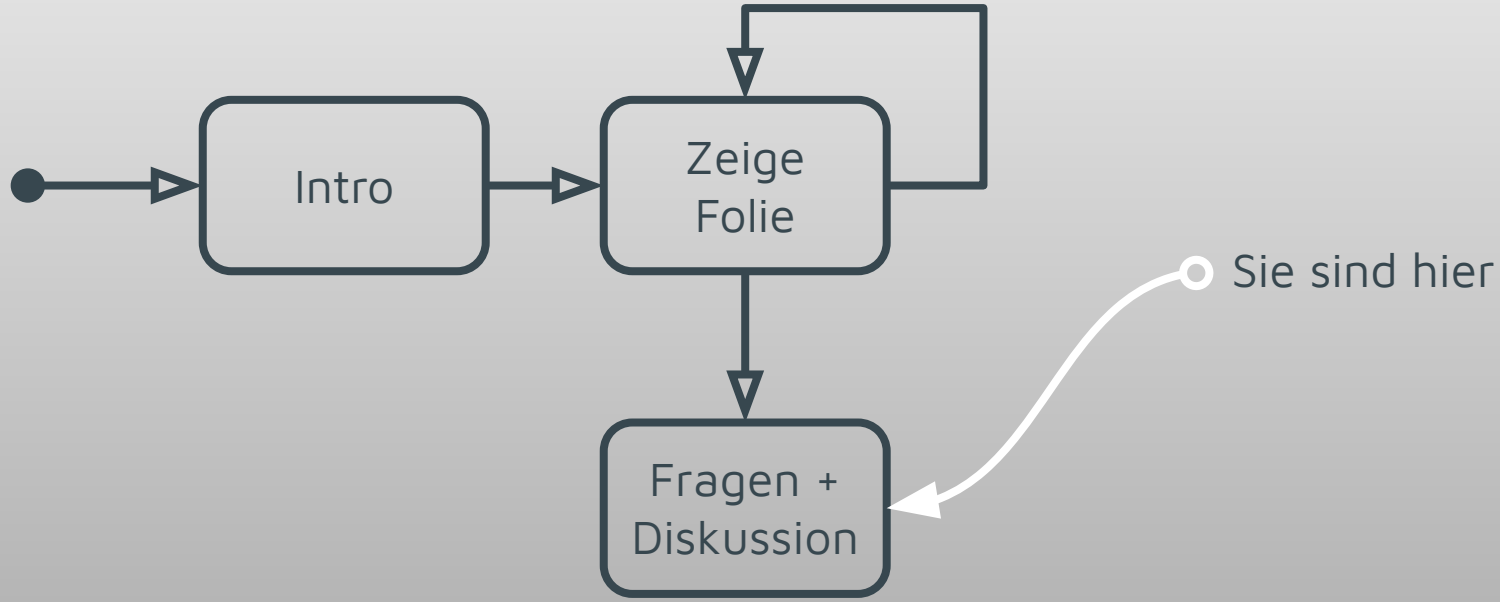
- Priority Ceiling Protocol
  - Task A sperrt Resource => A bekommt temporär höhere Priorität P
  - P so hoch gewählt, dass andere Tasks welche die geteilte Ressource nutzen nicht starten können (A kann nicht unterbrochen werden)
  - Ein einziger WRITE auf **BASEPRI** genügt

- PCP verhindert:
  - Priority inversion (medium prio task kann nicht unterbrechen)
  - Deadlock (higher priority task kann nicht unterbrechen)

RTIC Beispiel

```
// Im Setup:
pin.make_interrupt_source(&mut sys_cfg);
pin.enable_interrupt(&mut ctx.device.EXTI);
pin.trigger_on_edge(&mut ctx.device.EXTI, Edge::Falling);

blink::spawn().ok();

// Task:
#[task(binds = EXTI0, local = [pin])]
fn on_exti(ctx: on_exti::Context) {
    ctx.local.pin.clear_interrupt_pending_bit();
    rprintln!("incrementing");
    COUNTER.fetch_add(1, Ordering::SeqCst);
}
```

Viele Beispiele auf GitHub


Intro
Zeige Folie
Fragen + Diskussion
Sie sind hier


C++