Embedded Rust Ecosystem

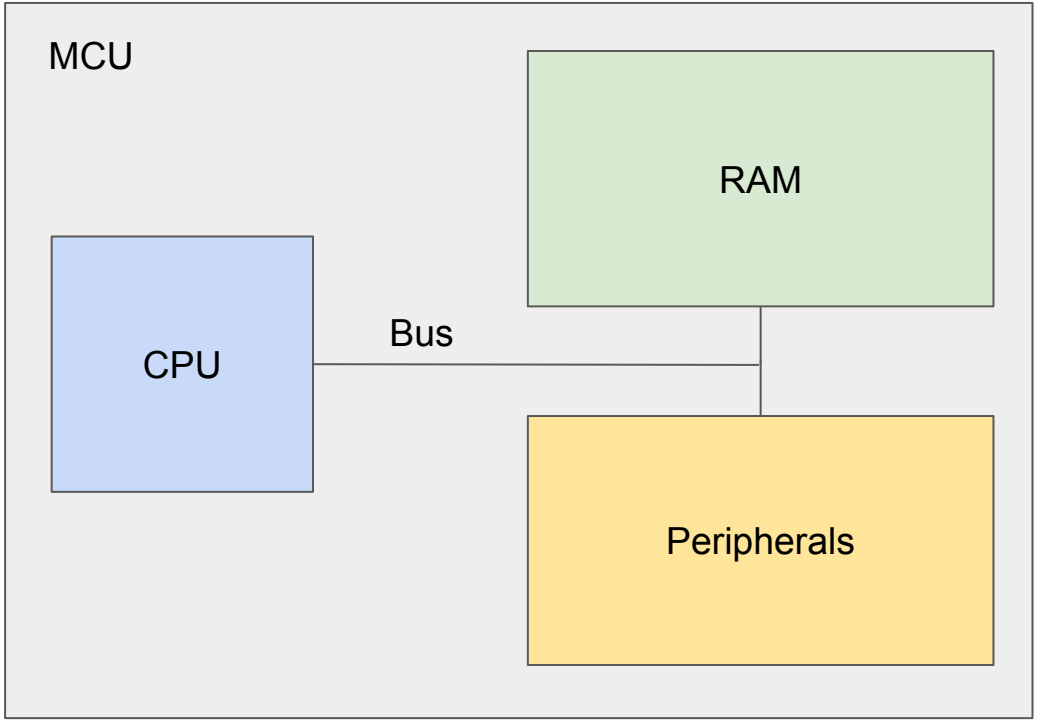
Who am I?

Wiktor Więcław

Currently working as Rust Software Engineer at Codilime

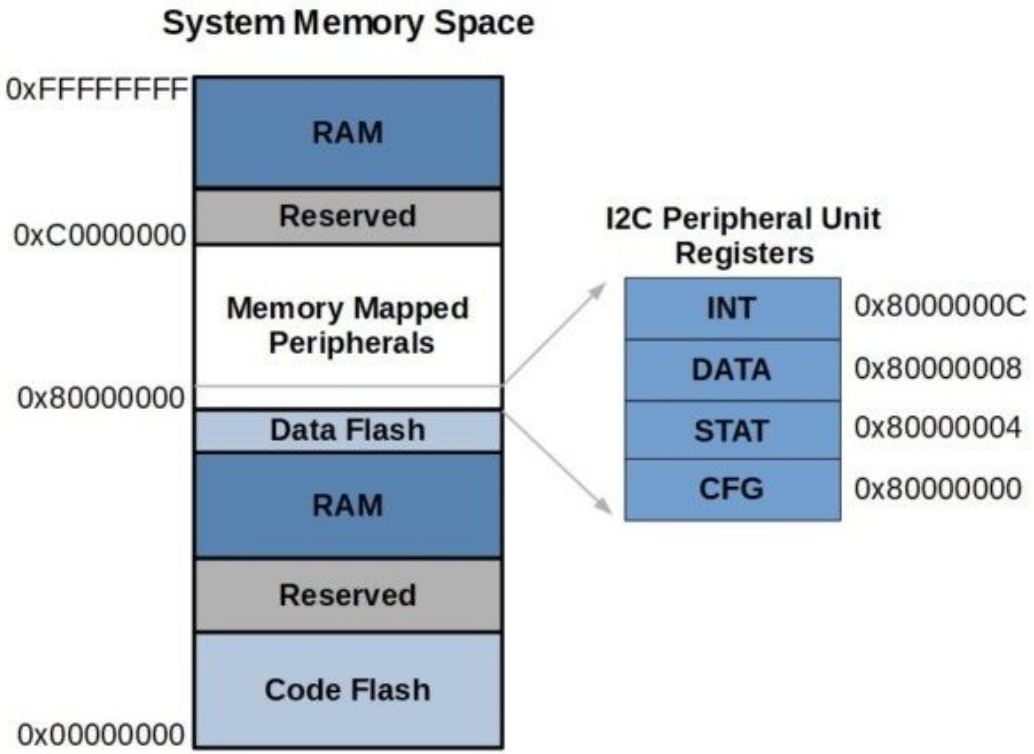
I’m a coordinator at COSMO PK Science Club

Quick refresher



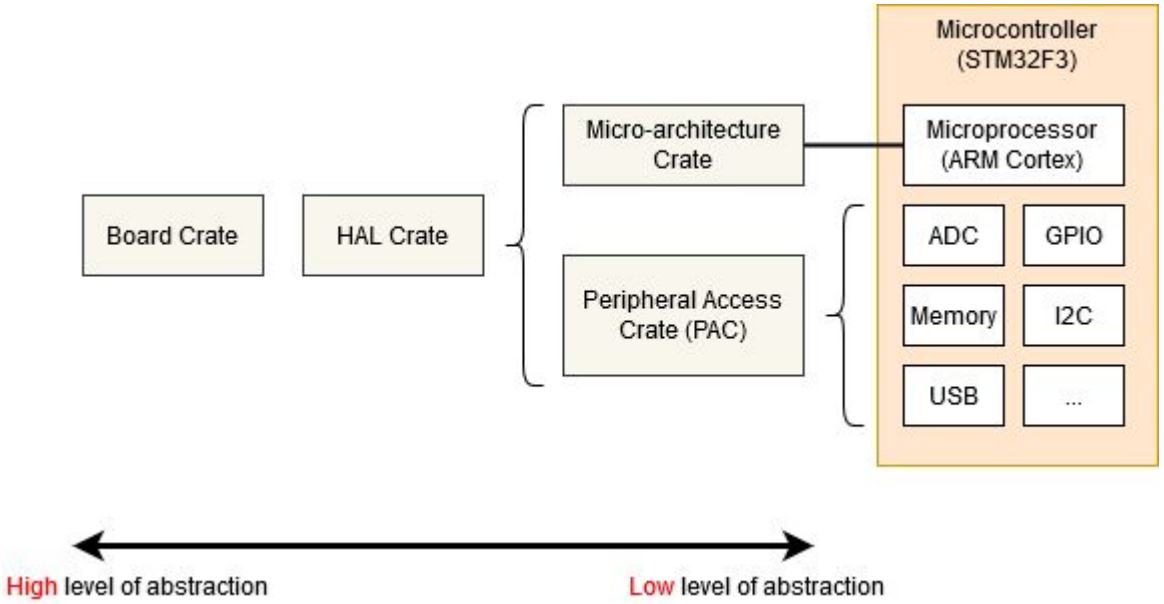
source: https://open4tech.com/peripheral-access-layer-for-microcontrollers/

Memory mapped IO



source: https://open4tech.com/peripheral-access-layer-for-microcontrollers/

Ecosystem overview



source: https://docs.rust-embedded.org/book/start/registers.html

cortex-m and cortex-m-rt - Micro-architecture crates

cortex-m - Low level access to Cortex-M processors
cortex-m-rt - Startup code and minimal runtime for Cortex-M microcontrollers

Source:
https://docs.rs/cortex-m/0.7.7/cortex_m/index
https://docs.rs/cortex-m-rt/0.7.3/cortex_m_rt/index.html

```rust
#![no_main]
#![no_std]

extern crate cortex_m_rt as rt;
extern crate panic_halt;

use rt::entry;

// the program entry point
#[entry]
fn main() -> ! {
    loop {}
}
```

source: https://github.com/rust-embedded/cortex-m/blob/1746a63ca16b68514ea23dcca1543aed00165452/cortex-m-rt/examples/minimal.rs

```rust
let peripherals = Peripherals::take().unwrap();
let mut systick = peripherals.SYST;
systick.set_clock_source(syst::SystClkSource::Core);
systick.set_reload(1_000);
systick.clear_current();
systick.enable_counter();
while !systick.has_wrapped() {
    // Loop
}
```

source: https://docs.rust-embedded.org/book/start/registers.html

tm4c123x - Peripheral Access Crate (PAC)

```rust
let cp = cortex_m::Peripherals::take().unwrap();
let p = tm4c123x::Peripherals::take().unwrap();

let pwm = p.PWM0;
pwm.ctl.write(|w| w.globalsync0().clear_bit());
// Mode = 1 => Count up/down mode
pwm._2_ctl.write(|w| w.enable().set_bit().mode().set_bit());
pwm._2_gena.write(|w| w.actcmpau().zero().actcmpad().one());
// 528 cycles (264 up and down) = 4 loops per video line (2112 cycles)
pwm._2_load.write(|w| unsafe { w.load().bits(263) });
pwm._2_cmpa.write(|w| unsafe { w.compa().bits(64) });
pwm.enable.write(|w| w.pwm4en().set_bit());
```

source: https://docs.rust-embedded.org/book/start/registers.html

stm32f4xx_hal - Hardware Abstraction Layer (HAL)

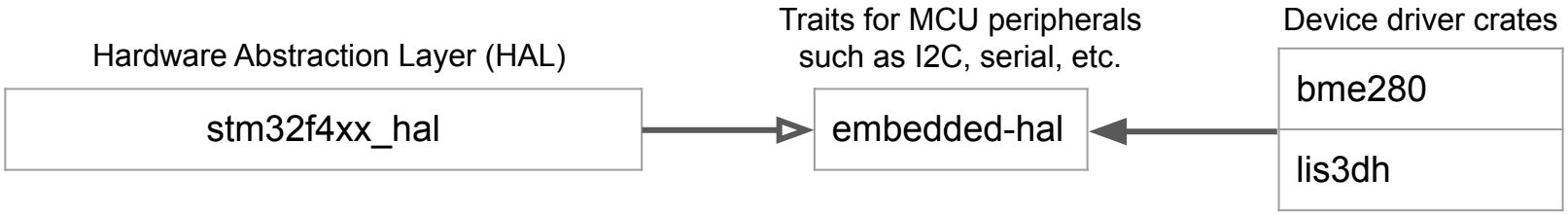
stm32f4xx-hal contains a multi device hardware abstraction on top of the peripheral access API for the STMicro STM32F4 series microcontrollers.

source: https://github.com/stm32-rs/stm32f4xx-hal/tree/a7406b69e46254a529fc7a4360ba1c9efd27ca50

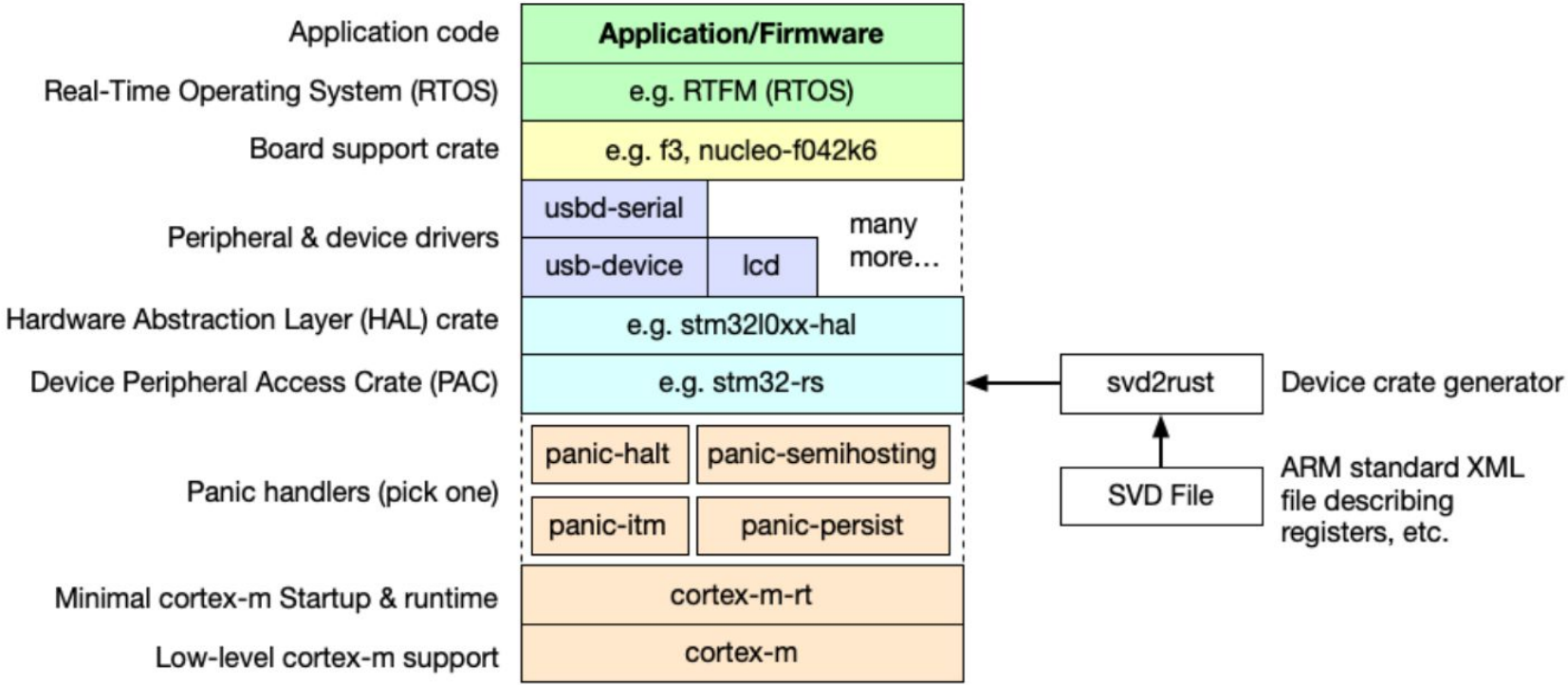
```rust
fn init_board(device: pac::Peripherals) -> Board {
    let gpioa = device.GPIOA.split();
    let gpiob = device.GPIOB.split();
    let gpioc = device.GPIOC.split();

    let led = gpioc.pc13.into_push_pull_output();
    let buzzer = gpioa.pa8.into_push_pull_output();

    let i2c1 = {
        let scl1 = gpiob
            .pb8
            .into_alternate()
            .internal_pull_up(false)
            .set_open_drain();
        let sda1 = gpiob
            .pb9
            .into_alternate()
            .internal_pull_up(false)
            .set_open_drain();
        let mode = i2c::Mode::Fast {
            frequency: 400000.Hz(),
            duty_cycle: i2c::DutyCycle::Ratio2to1,
        };
        device.I2C1.i2c((scl1, sda1), mode, &clocks)
    };
```

embedded-hal

Serves as a foundation for building an ecosystem of platform-agnostic drivers



Detailed overview



source: http://essay.utwente.nl/89253/1/balciunas_MA_EEMCS.pdf

Real-time Interrupt-based Concurrency (RTIC)

The hardware accelerated Rust RTOS

A concurrency framework for building real-time systems.

```rust
#[app(device = stm32f3xx_hal::pac, peripherals = true, dispatchers = [SPI1])]
mod app {
    use super::*;

    #[shared]
    struct Shared {}

    #[local]
    struct Local {
        led: PA5<Output<PushPull>>,
        state: bool,
    }

    #[init]
    fn init(cx: init::Context) -> (Shared, Local) {
    }

    #[task(local = [led, state])]
    async fn blink(cx: blink::Context) {
        loop {
            rprintln!("blink");
            if *cx.local.state {
                cx.local.led.set_high().unwrap();
                *cx.local.state = false;
            } else {
                cx.local.led.set_low().unwrap();
                *cx.local.state = true;
            }
            Systick::delay(1000.millis()).await;
        }
    }
}
```

Embassy

Embassy is the next-generation framework for embedded applications.

```rust
// Declare async tasks
#[embassy_executor::task]
async fn blink(pin: AnyPin) {
    let mut led = Output::new(pin, Level::Low, OutputDrive::Standard);

    loop {
        // Timekeeping is globally available, no need to mess with hardware timers.
        led.set_high();
        Timer::after(Duration::from_millis(150)).await;
        led.set_low();
        Timer::after(Duration::from_millis(150)).await;
    }
}

// Main is itself an async task as well.
#[embassy_executor::main]
async fn main(spawner: Spawner) {
    let p = embassy_nrf::init(Default::default());

    // Spawned tasks run in the background, concurrently.
    spawner.spawn(blink(p.P0_13.degrade())).unwrap();

    let mut button = Input::new(p.P0_11, Pull::Up);
    loop {
        // Asynchronously wait for GPIO events, allowing other tasks
        // to run, or the core to sleep.
        button.wait_for_low().await;
        info!("Button pressed!");
        button.wait_for_high().await;
        info!("Button released!");
    }
}
```

Where to start?

Get started!

The Discovery book

Learn embedded development from the ground up—using Rust!

READ

The Embedded Rust book

Already familiar with Embedded development? Jump in with Rust and start reaping the benefits.

READ

The Embedonomicon

Look under the hood of foundational embedded libraries.

READ

MORE DOCUMENTATION

source: https://www.rust-lang.org/what/embedded

Thanks