LOW POWER & LOW FRUSTRATION

A tour of embedded Rust

Dion Dokter

WHO AM I?

- Dion Dokter
  - 25
  - Working at Tweede golf
  - Embedded Rust since 2019

  - @Geoxion on twitter
  - `Stackdump` on github

TOOLCHAIN

How to get things running

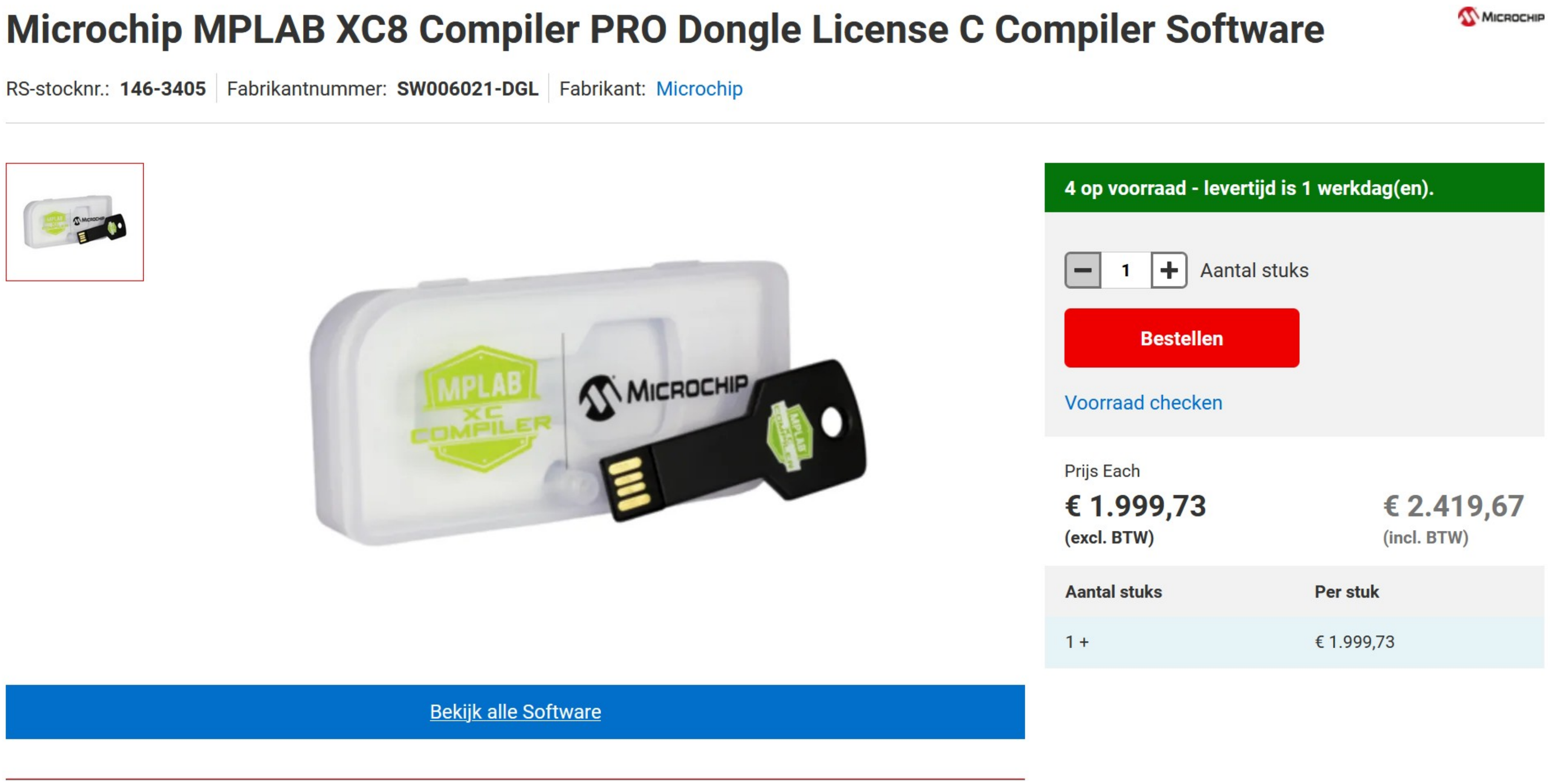
INSTALL THE RUST COMPILER

We can use all the normal tools!

- Rustup
- Rustc
- Cargo
- Rust Analyzer

Go to rust-lang.org and follow the instructions.

INSTALL THE RUST COMPILER



Microchip MPLAB XC8 Compiler PRO Dongle License C Compiler Software

RS-stocknr.: **146-3405** | Fabrikantnummer: **SW006021-DGL** | Fabrikant: Microchip

4 op voorraad - levertijd is 1 werkdag(en).

1 Aantal stuks

Bestellen

Voorraad checken

Prijs Each

€ 1.999,73 (excl. BTW)

€ 2.419,67 (incl. BTW)

Aantal stuks	Per stuk
1 +	€ 1.999,73

Bekijk alle Software

EMBEDDED TOOLS

- Normal compiler
  - Everything can be downloaded through rustup
- Probe-rs
  - Probe-run
  - Cargo-flash
- Knurling
  - Defmt
  - Flip-link

ECOSYSTEM

From low to high level

PERIPHERAL ACCESS CRATES

C

```c
#include "samd21e17l.h"

// Raw
bool is_8_cycles = ((WDT->CONFIG.reg & WDT_CONFIG_PER_Msk) << WDT_CONFIG_PER_Pos) == WDT_CONFIG_PER_8_val;
WDT->CONFIG.reg = (WDT->CONFIG.reg & ~WDT_CONFIG_PER_Msk) | WDT_CONFIG_PER_16;

// Bitfield
bool is_8_cycles = WDT->CONFIG.bit.PER == WDT_CONFIG_PER_8_val;
WDT->CONFIG.bit.PER = WDT_CONFIG_PER_16;
```

Rust

```rust
// Take ownership of the peripherals
let dp = atsamd21e::Peripherals::take().unwrap();

let is_8_cycles = dp.WDT.CONFIG.read().per().is_8();
dp.WDT.CONFIG.modify(|_, w| w.per()._8());
```

OVERVIEW

- PAC SAMD21E
- PAC nRF9160
- PAC nRF52840
- PAC STM32H743
- PAC STM32H753
- PAC STM32L476
- PAC STM32L496

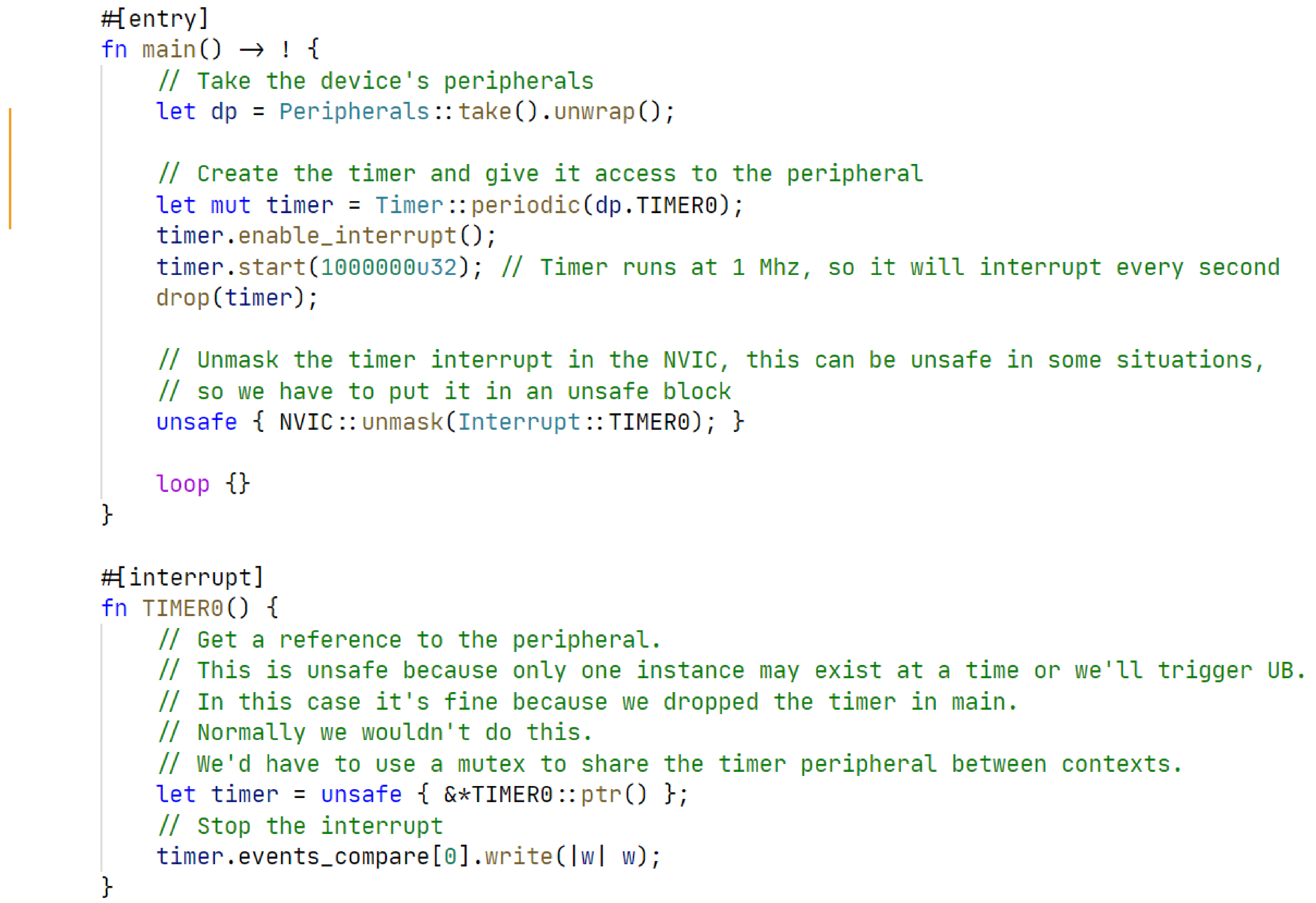
HARDWARE ABSTRACTION LAYERS

Many open source HALs

- Vendor HAL for RiscV ESP chips
- Also async support (embassy)

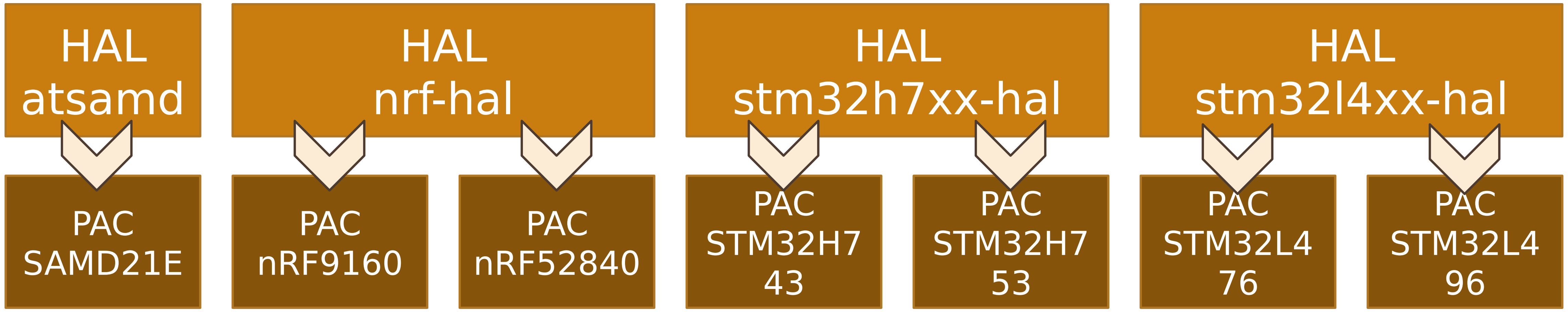
Implementation of high level chip features

Built on top of PACs



```rust
#[entry]
fn main() -> ! {
    // Take the device's peripherals
    let dp = Peripherals::take().unwrap();

    // Create the timer and give it access to the peripheral
    let mut timer = Timer::periodic(dp.TIMER0);
    timer.enable_interrupt();
    timer.start(1000000u32); // Timer runs at 1 Mhz, so it will interrupt every second
    drop(timer);

    // Unmask the timer interrupt in the NVIC, this can be unsafe in some situations,
    // so we have to put it in an unsafe block
    unsafe { NVIC::unmask(Interrupt::TIMER0); }

    loop {}
}

#[interrupt]
fn TIMER0() {
    // Get a reference to the peripheral.
    // This is unsafe because only one instance may exist at a time or we'll trigger UB.
    // In this case it's fine because we dropped the timer in main.
    // Normally we wouldn't do this.
    // We'd have to use a mutex to share the timer peripheral between contexts.
    let timer = unsafe { &*TIMER0::ptr() };
    // Stop the interrupt
    timer.events_compare[0].write(|w| w);
}
```

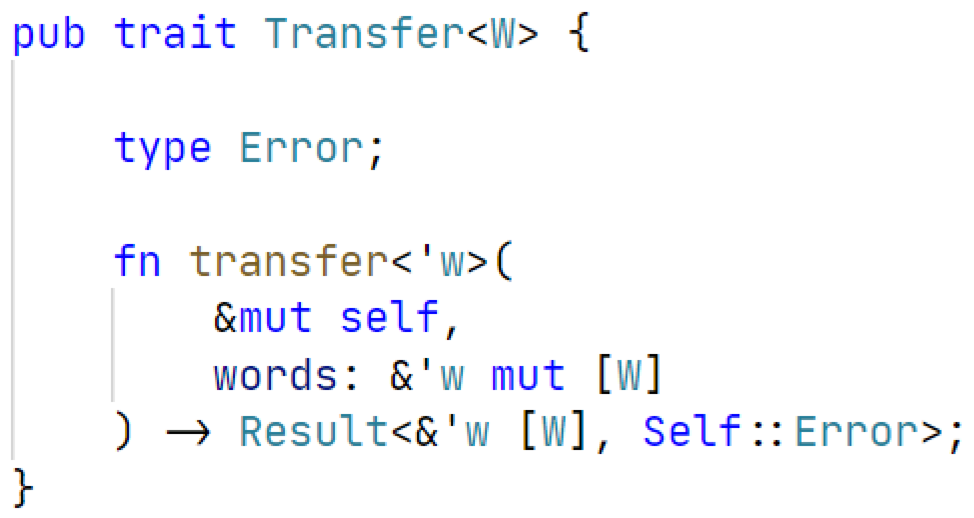
OVERVIEW


HAL atsamd
HAL nrf-hal
HAL stm32h7xx-hal
HAL stm32l4xx-hal
PAC SAMD21E
PAC nRF9160
PAC nRF52840
PAC STM32H743
PAC STM32H753
PAC STM32L476
PAC STM32L496

COOPERATION

`Embedded HAL` is the glue of the entire ecosystem.

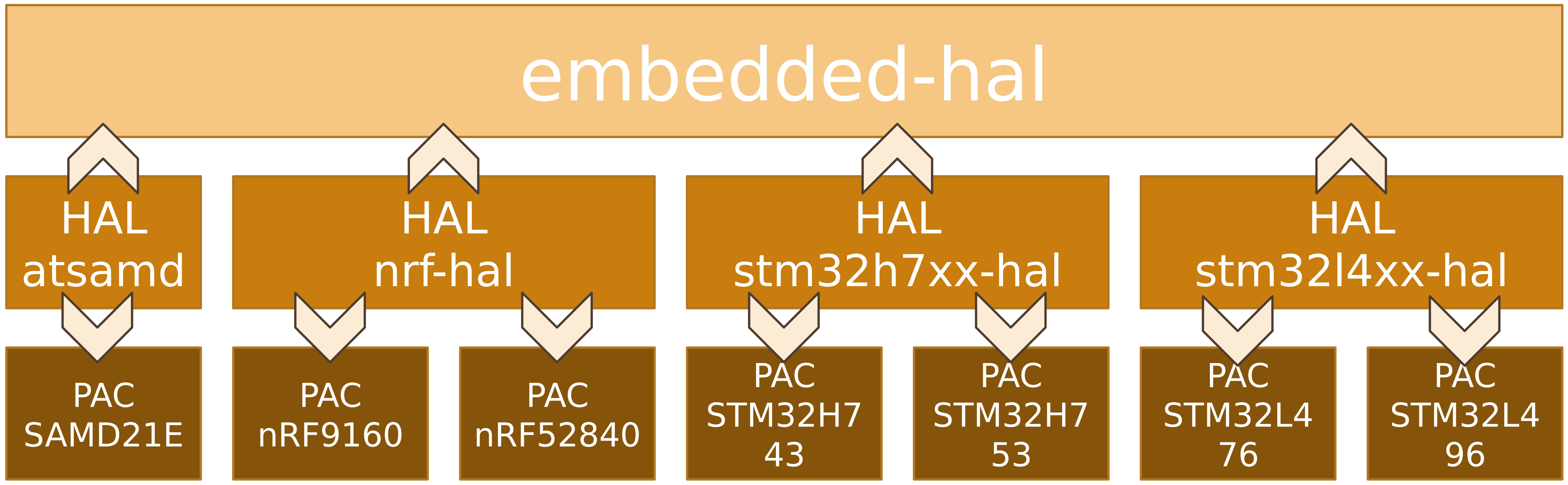
- Contains abstractions for many common operations
- SPI example trait:



```
pub trait Transfer<W> {

    type Error;

    fn transfer<'w>(
        &mut self,
        words: &'w mut [W]
    ) → Result<&'w [W], Self::Error>;
}
```

FYI: SPI (Serial Peripheral Interface) is a common communication protocol to talk with other devices

OVERVIEW


embedded-hal
HAL
atsamd
HAL
nrf-hal
HAL
stm32h7xx-hal
HAL
stm32l4xx-hal
PAC
SAMD21E
PAC
nRF9160
PAC
nRF52840
PAC
STM32H743
PAC
STM32H753
PAC
STM32L476
PAC
STM32L496

DEVICE DRIVERS

Traits + generics

- Reuse traits from embedded-hal
- Efficient
- Convenient

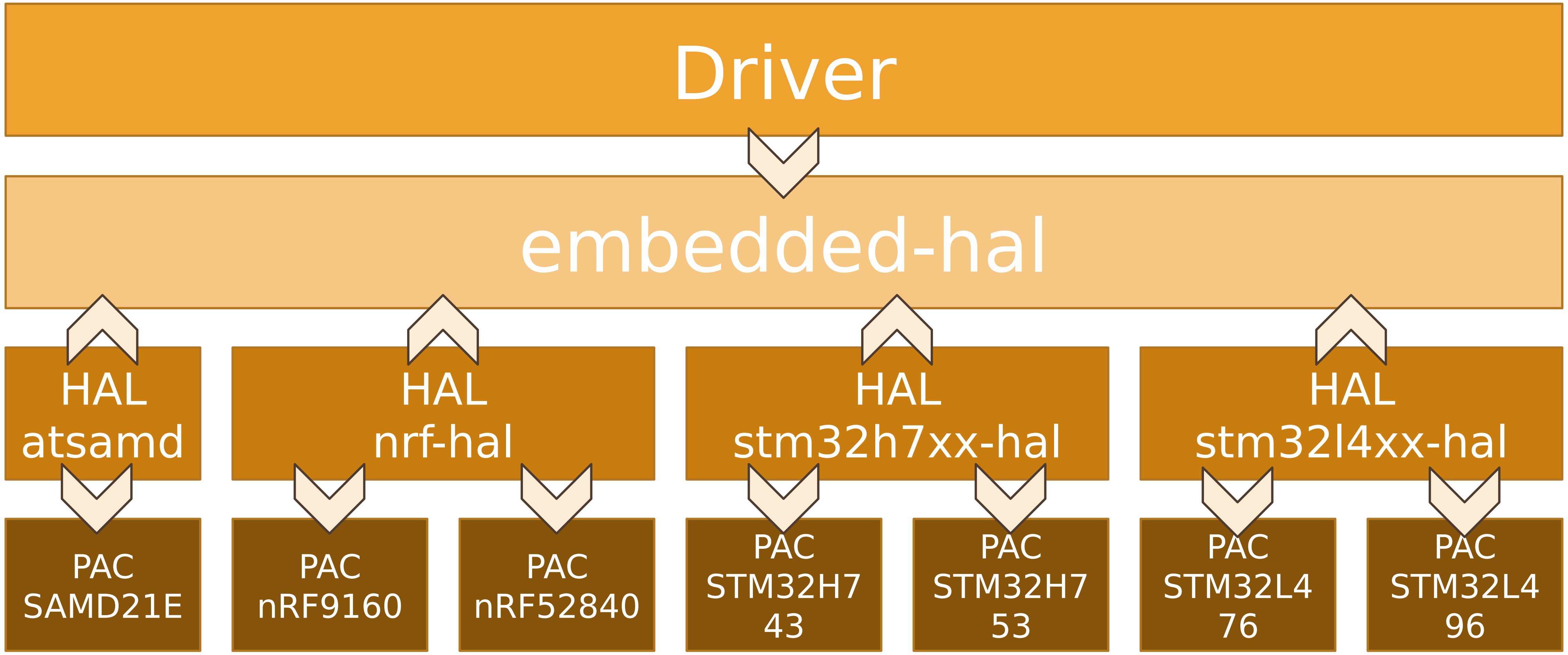
In C this is frustrating

- No standard interfaces
- No abstractions
  - Function pointers?
  - Extern functions?
  - Fork & implement functions?

https://youtu.be/z9z74VpqO9A

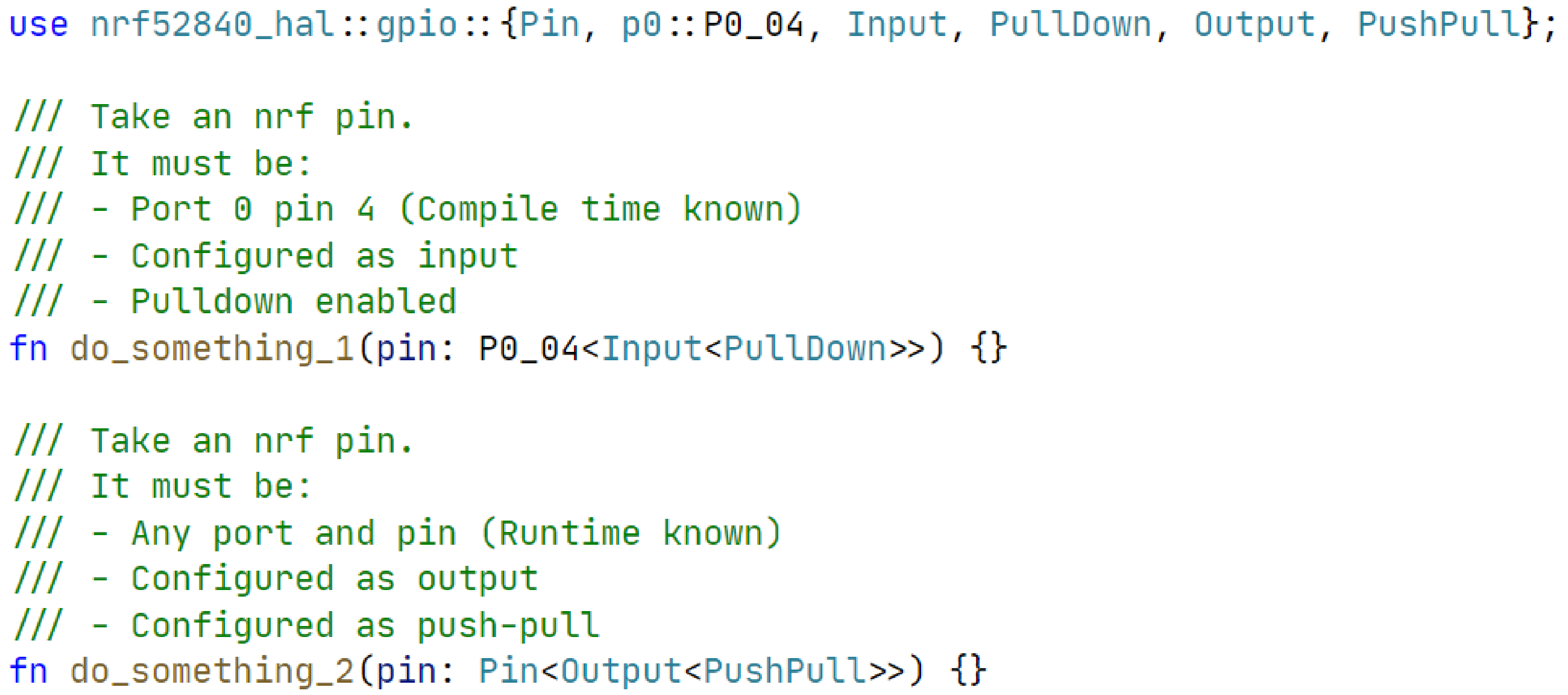
```rust
use embedded_hal::blocking::spi;
use embedded_hal::digital::v2::OutputPin;

pub struct Device<SPI, CS>
where
    SPI: spi::Transfer<u8>,
    CS: OutputPin,
{
    bus: SPI,
    chipselect: CS,
}

impl<SPI, CS> Device<SPI, CS>
where
    SPI: spi::Transfer<u8>,
    CS: OutputPin,
{
    pub fn new(bus: SPI, chipselect: CS) -> Self {
        Self { bus, chipselect }
    }

    pub fn example(&mut self) -> u8 {
        self.chipselect.set_low().ok();
        self.bus.transfer(&mut [0xDE]).ok();
        let result = self.bus.transfer(&mut [0xAD]).ok().unwrap()[0];
        self.chipselect.set_high().ok();

        result
    }
}
```

OVERVIEW


Driver
embedded-hal
HAL atsamd
HAL nrf-hal
HAL stm32h7xx-hal
HAL stm32l4xx-hal
PAC SAMD21E
PAC nRF9160
PAC nRF52840
PAC STM32H743
PAC STM32H753
PAC STM32L476
PAC STM32L496

TYPESTATE

State encoded in the type of the variable



```rust
use nrf52840_hal::gpio::{Pin, p0::P0_04, Input, PullDown, Output, PushPull};

/// Take an nrf pin.
/// It must be:
/// - Port 0 pin 4 (Compile time known)
/// - Configured as input
/// - Pulldown enabled
fn do_something_1(pin: P0_04<Input<PullDown>>) {}

/// Take an nrf pin.
/// It must be:
/// - Any port and pin (Runtime known)
/// - Configured as output
/// - Configured as push-pull
fn do_something_2(pin: Pin<Output<PushPull>>) {}
```

RUNTIMES

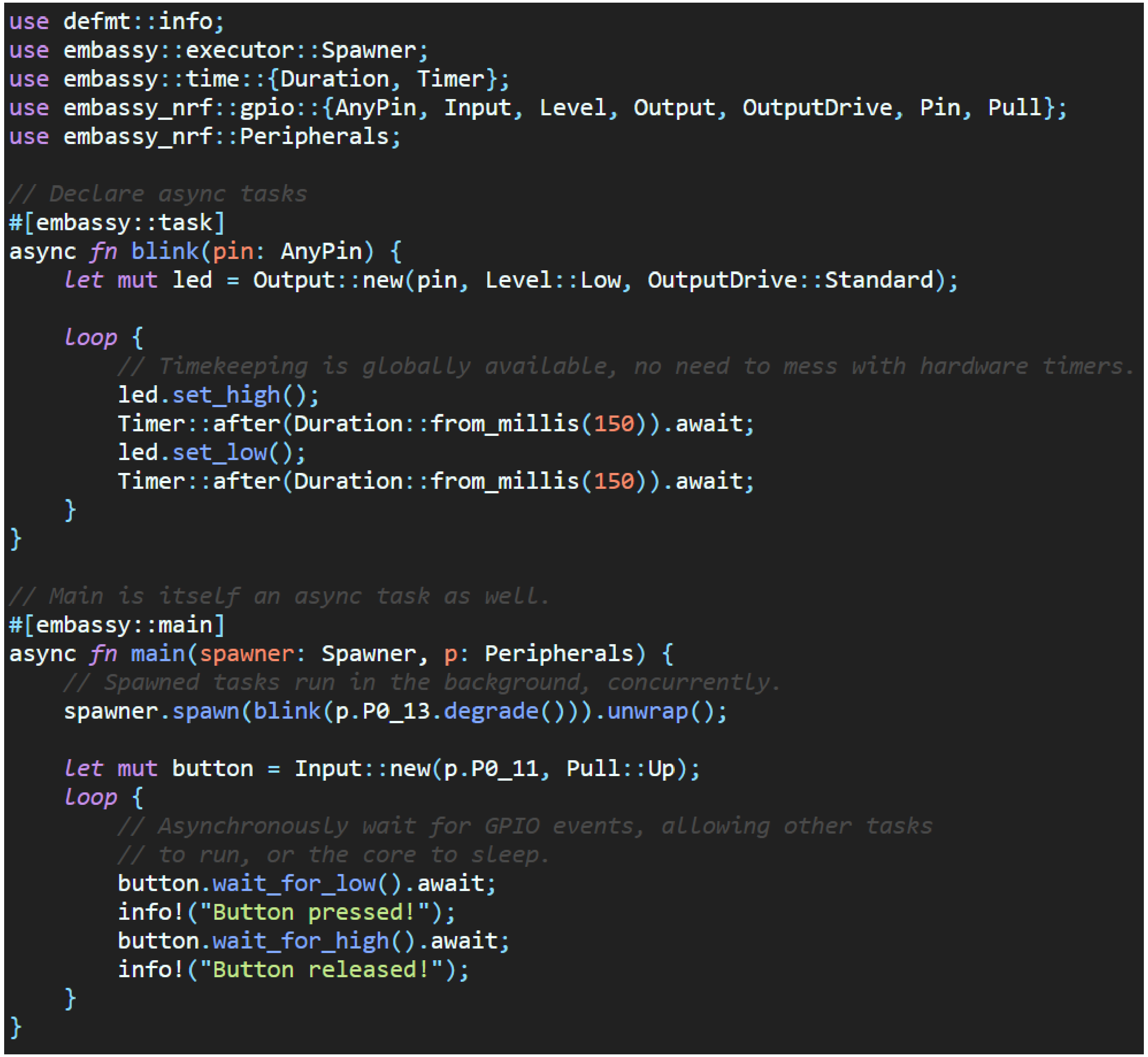
Bare metal
- Like Arduino

RTIC
- Interrupts on steroids

RTOS
- Many variants
- Not super popular in Rust

Async
- New and exciting

Other
- MnemOS
- Hubris



```rust
use defmt::info;
use embassy::executor::Spawner;
use embassy::time::{Duration, Timer};
use embassy_nrf::gpio::{AnyPin, Input, Level, Output, OutputDrive, Pin, Pull};
use embassy_nrf::Peripherals;

// Declare async tasks
#[embassy::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::main]
async fn main(spawner: Spawner, p: Peripherals) {
    // 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!");
    }
}
```

TRYOUT DEMO

Let’s look at a real project