hello_blinky.rs

Introduction to Embedded Rust

Dawid Królak 22.11.2023

Dawid Królak

- Expert engineer @ mobica a cognizant company
- Part time Rust developer
- Passionate about space exploration industry

https://twitter.com/Taavicjusz

https://github.com/taavit

https://mastodon.social/@taavit

Agenda

- Why Rust?
- How to start?
- Few words on embedded-hal
- Demo #1
- Let’s build a blinky driver
- Demo #2
- Bonus demo: Orientation tracker

What is Rust

- System programming language
- Single ownership for variables
- Type safety
- Lifetime for references
- Separation of safe and unsafe code
- Unified ecosystem (cargo)
    - Testing methods
    - Dependency management
    - Built in linter and formatter

Installing rust

https://www.rust-lang.org/tools/install

Using rustup (Recommended)

It looks like you're running macOS, Linux, or another Unix-like OS. To download Rustup and install Rust, run the following in your terminal, then follow the on-screen instructions. See "Other Installation Methods" if you are on Windows.

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Installing target

```
dawid@atarynka:~$ rustup target add thumbv6m-none-eabi
dawid@atarynka:~$ cargo install elf2uf2-rs
```

Blinky

```rust
/// Regular desktop demo
use core::time::Duration;

fn main() {
	let mut blink = false;

	loop {
		println!("{}", if blink {"[*]"} else {"[ ]"});
		blink = !blink;
		std::thread::sleep(Duration::from_millis(250));
	}
}
```

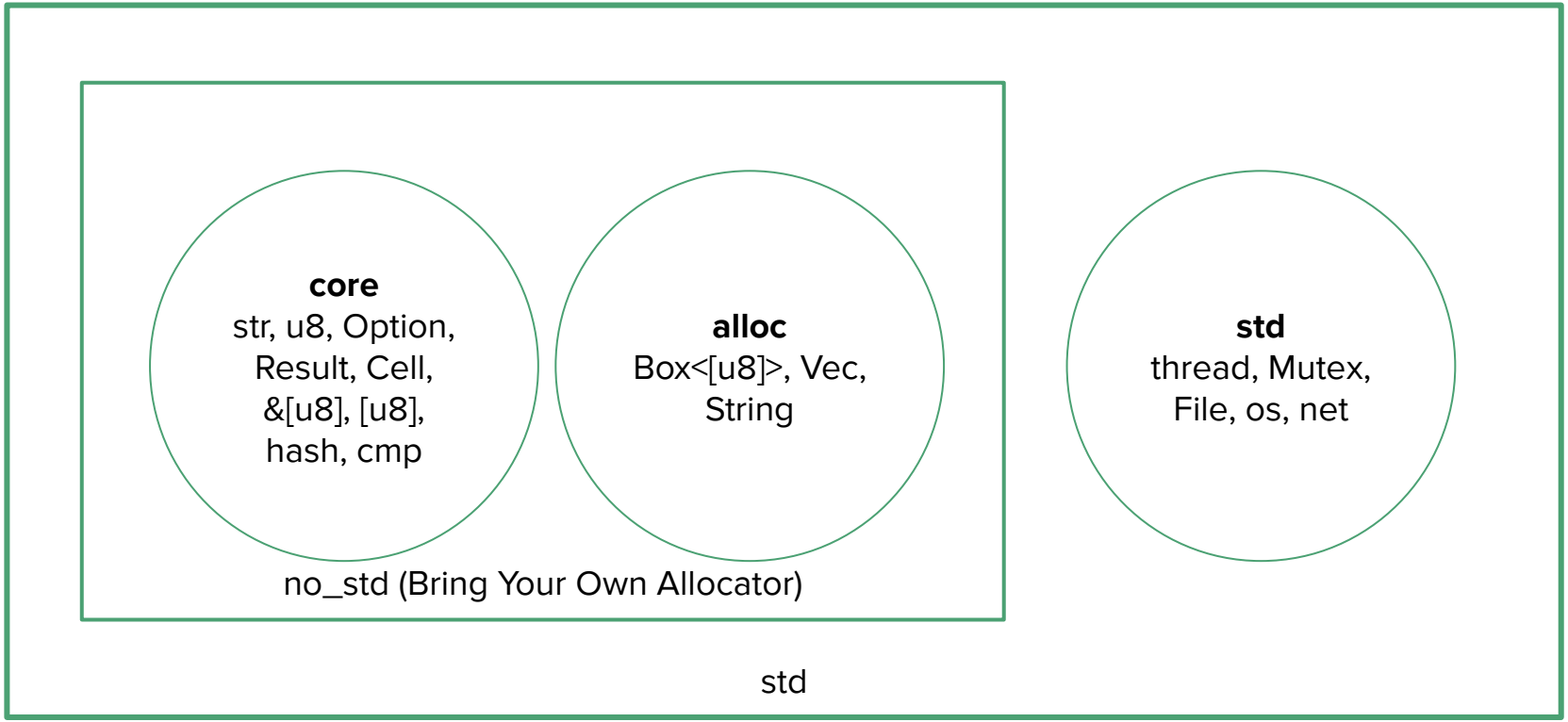
rp-pico example

```rust
#![no_std]
#![no_main]
use rp_pico::entry;
use rp_pico::hal::prelude::*;
use rp_pico::hal::pac;
use rp_pico::hal;
use panic_halt as _;
#[entry]
fn main() -> ! {
    let mut pac = pac::Peripherals::take().unwrap();
    let core = pac::CorePeripherals::take().unwrap();
    let mut watchdog = hal::Watchdog::new(pac.WATCHDOG);
    let clocks = hal::clocks::init_clocks_and_plls(
        rp_pico::XOSC_CRYSTAL_FREQ,
        pac.XOSC,
        pac.CLOCKS,
        pac.PLL_SYS,
        pac.PLL_USB,
        &mut pac.RESETS,
        &mut watchdog,
    )
    .ok()
    .unwrap();
```

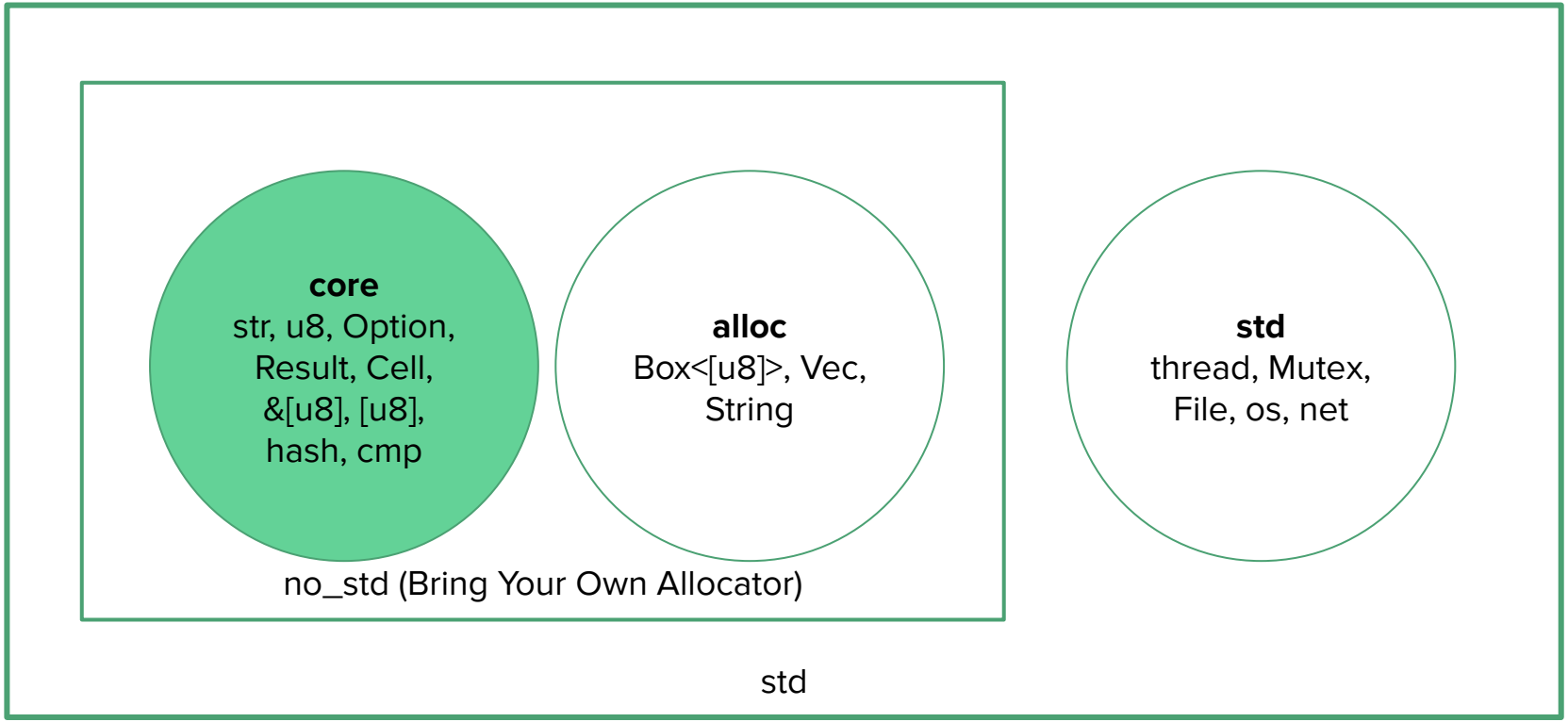
```rust
    let mut delay = cortex_m::delay::Delay::new(
        core.SYST,
        clocks.system_clock.freq().to_Hz()
    );

    let sio = hal::Sio::new(pac.SIO);
    let pins = rp_pico::Pins::new(
        pac.IO_BANK0,
        pac.PADS_BANK0,
        sio.gpio_bank0,
        &mut pac.RESETS,
    );
    let mut led_pin = pins.led.into_push_pull_output();
    loop {
        led_pin.set_high().unwrap();
        delay.delay_ms(500);
        led_pin.set_low().unwrap();
        delay.delay_ms(500);
    }
}
```

#![no_std]


core
str, u8, Option,
Result, Cell,
&[u8], [u8],
hash, cmp
alloc
Box<[u8]>, Vec,
String
std
thread, Mutex,
File, os, net
no_std (Bring Your Own Allocator)
std

#![no_std]


core
str, u8, Option,
Result, Cell,
&[u8], [u8],
hash, cmp
alloc
Box<[u8]>, Vec,
String
std
thread, Mutex,
File, os, net
no_std (Bring Your Own Allocator)
std

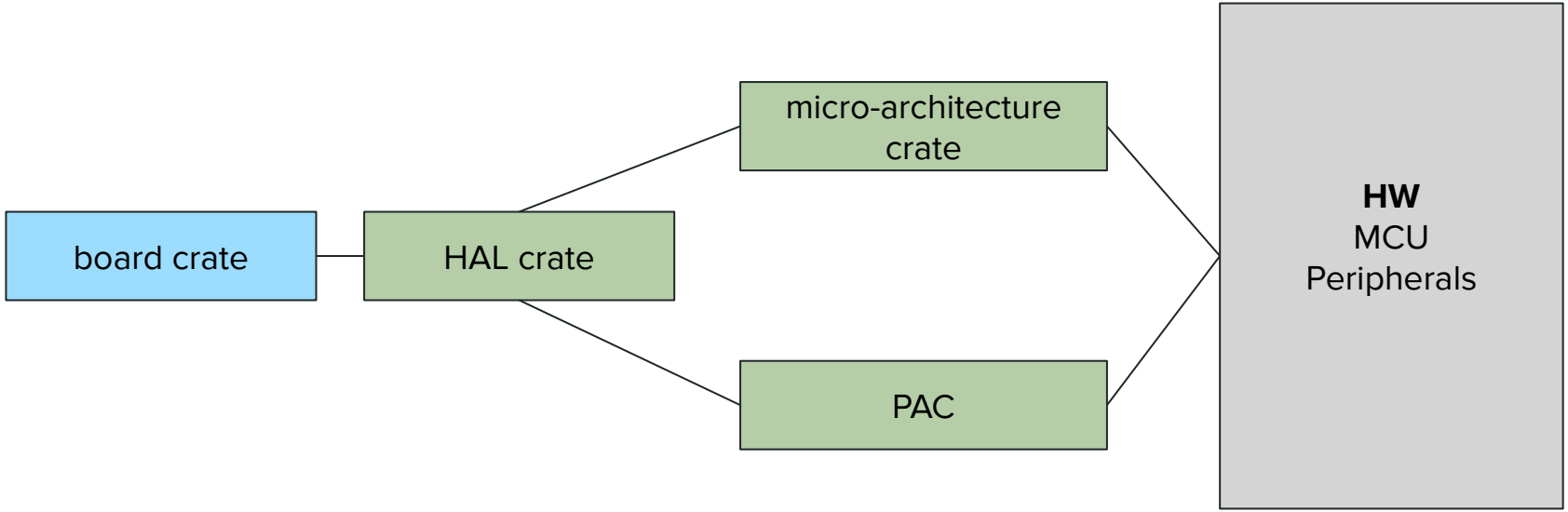
rp-pico example

```rust
#![no_std]
#![no_main]
use rp_pico::entry;
use rp_pico::hal::prelude::*;
use rp_pico::hal::pac;
use rp_pico::hal;
use panic_halt as _;
#[entry]
fn main() -> ! {
    let mut pac = pac::Peripherals::take().unwrap();
    let core = pac::CorePeripherals::take().unwrap();
    let mut watchdog = hal::Watchdog::new(pac.WATCHDOG);
    let clocks = hal::clocks::init_clocks_and_plls(
        rp_pico::XOSC_CRYSTAL_FREQ,
        pac.XOSC,
        pac.CLOCKS,
        pac.PLL_SYS,
        pac.PLL_USB,
        &mut pac.RESETS,
        &mut watchdog,
    )
    .ok()
    .unwrap();
```

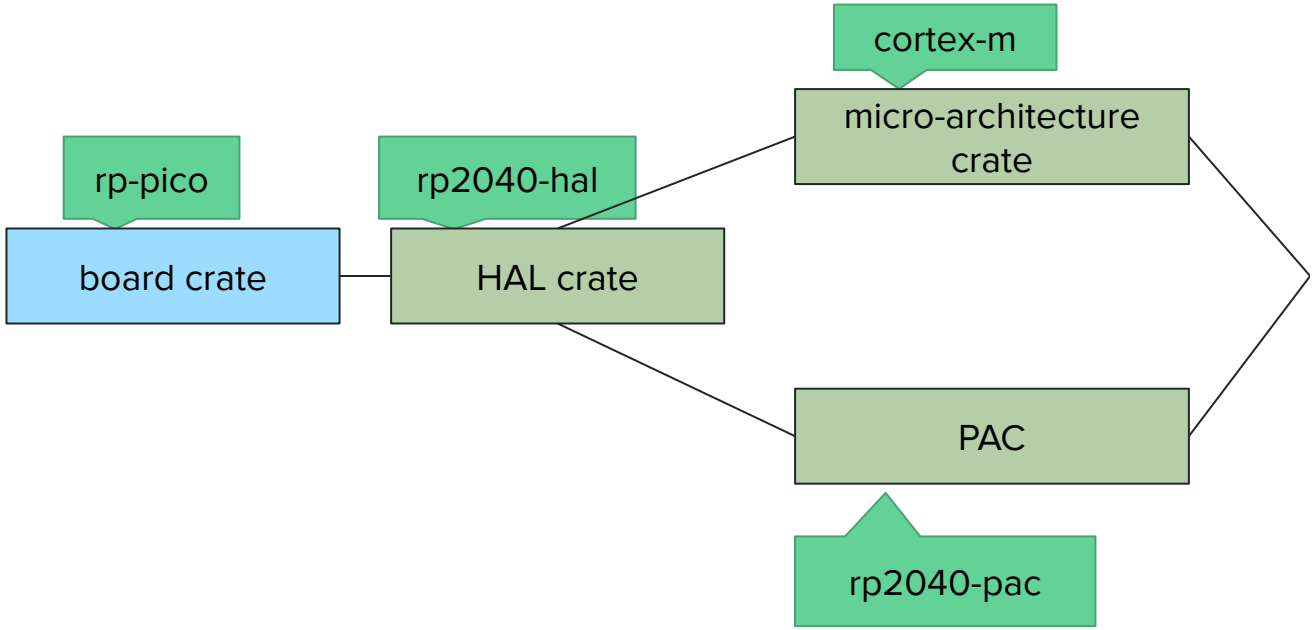
```rust
    let mut delay = cortex_m::delay::Delay::new(
        core.SYST,
        clocks.system_clock.freq().to_Hz()
    );

    let sio = hal::Sio::new(pac.SIO);
    let pins = rp_pico::Pins::new(
        pac.IO_BANK0,
        pac.PADS_BANK0,
        sio.gpio_bank0,
        &mut pac.RESETS,
    );
    let mut led_pin = pins.led.into_push_pull_output();
    loop {
        led_pin.set_high().unwrap();
        delay.delay_ms(500);
        led_pin.set_low().unwrap();
        delay.delay_ms(500);
    }
}
```

use rp_pico::entry;



use rp_pico::entry;



embedded-hal

embedded-hal

`embedded-hal`

> A Hardware Abstraction Layer (HAL) for embedded systems

This project is developed and maintained by the HAL team.

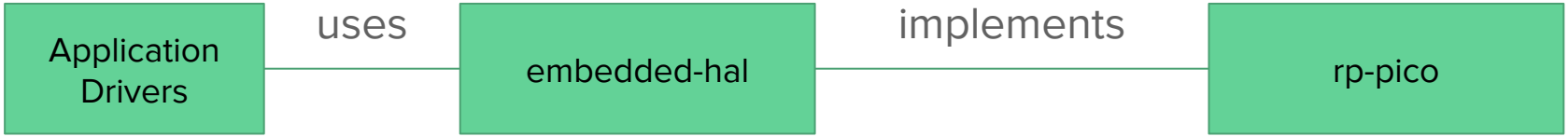
Scope

`embedded-hal` serves as a foundation for building an ecosystem of platform-agnostic drivers. (driver meaning library crates that let a target platform interface an external device like a digital sensor or a wireless transceiver).

The advantage of this system is that by writing the driver as a generic library on top of `embedded-hal` driver authors can support any number of target platforms (e.g. Cortex-M microcontrollers, AVR microcontrollers, embedded Linux, etc.).

The advantage for application developers is that by adopting `embedded-hal` they can unlock all these drivers for their platform.

embedded-hal



embedded-hal

Available versions:

- embedded-hal@0.2.7
- embedded-hal@1.0.0-rc
- embedded-hal-async@0.2.0-alpha
- embedded-hal-async@1.0.0-rc

All are valid and but not all drivers provides all implementations

embedded-hal drivers

Driver crates

Platform agnostic crates to interface external components. These crates use the `embedded-hal` interface to support all the devices and systems that implement the `embedded-hal` traits.

The list below contains drivers that have achieved the "released" status. In order to add a driver to this list, please ensure that your driver has a short blog post, article or sufficiently explanatory README showing an example of its use. Ideally this post would demonstrate using the device in a small project so that a Rust and/or embedded newcomer can also understand it. Otherwise please add it to the WIP section below.

1. AD983x - SPI - AD9833/AD9837 waveform generators / DDS - Intro blog post - crates.io v0.3.0
2. adafruit-alphanum4 - I2C - Driver for Adafruit 14-segment LED Alphanumeric Backpack based on the ht16k33 chip - crates.io v0.1.2
3. ADE791x - SPI - ADE7912/ADE7913 3-Channel, Isolated, Sigma-Delta ADC - github - crates.io v0.3.4
4. ADS1x1x - I2C - 12/16-bit ADCs like ADS1013, ADS1015, ADS1115, etc. - Intro blog post - crates.io v0.2.2
5. ADXL313 - SPI - 3-axis accelerometer - crates.io v0.2.4
6. ADXL343 - I2C - 3-axis accelerometer - crates.io v0.8.0
7. ADXL355 - SPI - 3-axis accelerometer - Intro blog post - crates.io v0.2.3
8. AFE4404 - I2C - Pulse oximeter - crates.io v0.2.4
9. AHT20 - I2C - Humidity and temperature sensor - github - crates.io v0.1.0
10. AHT20-driver - I2C - Humidity and temperature sensor - Intro blog post - github - crates.io v1.2.0
11. AnyLeaf - I2C - pH sensor module - github - crates.io v1.0.1
12. AT86RF212 - SPI - Low power IEEE 802.15.4-2011 ISM RF Transceiver - Intro blog post - crates.io v0.2.0
13. BlueNRG - SPI - driver for BlueNRG-MS Bluetooth module - Intro post crates.io v0.1.0
14. BMA400 - I2C/SPI - Bosch 12-bit 3-axis accelerometer - github crates.io v0.2.0
15. BNO055 - I2C - Bosch Sensortec BNO055 9-axis IMU driver - Intro post crates.io v0.3.3
16. CD74HC4067 - GPIO - 16-channel digital and analog multiplexer - Intro blog post - github - crates.io v0.2.0
17. dht-sensor - 1-Wire - DHT11/DHT22 temperature/humidity sensor driver - github - crates.io v0.2.1

rp-pico example

```rust
#![no_std]
#![no_main]
use rp_pico::entry;
use rp_pico::hal::prelude::*;
use rp_pico::hal::pac;
use rp_pico::hal;
use panic_halt as _;
#[entry]
fn main() -> ! {
    let mut pac = pac::Peripherals::take().unwrap();
    let core = pac::CorePeripherals::take().unwrap();
    let mut watchdog = hal::Watchdog::new(pac.WATCHDOG);
    let clocks = hal::clocks::init_clocks_and_plls(
        rp_pico::XOSC_CRYSTAL_FREQ,
        pac.XOSC,
        pac.CLOCKS,
        pac.PLL_SYS,
        pac.PLL_USB,
        &mut pac.RESETS,
        &mut watchdog,
    )
    .ok()
    .unwrap();
```

```rust
    let mut delay = cortex_m::delay::Delay::new(
        core.SYST,
        clocks.system_clock.freq().to_Hz()
    );

    let sio = hal::Sio::new(pac.SIO);
    let pins = rp_pico::Pins::new(
        pac.IO_BANK0,
        pac.PADS_BANK0,
        sio.gpio_bank0,
        &mut pac.RESETS,
    );
    let mut led_pin = pins.led.into_push_pull_output();
    loop {
        led_pin.set_high().unwrap();
        delay.delay_ms(500);
        led_pin.set_low().unwrap();
        delay.delay_ms(500);
    }
}
```

Building

memory.x

```
MEMORY {
    BOOT2 : ORIGIN = 0x10000000, LENGTH = 0x100
    FLASH : ORIGIN = 0x10000100, LENGTH = 2048K - 0x100
    RAM   : ORIGIN = 0x20000000, LENGTH = 256K
}

EXTERN(BOOT2_FIRMWARE)

SECTIONS {
    /* ### Boot loader */
    .boot2 ORIGIN(BOOT2) :
    {
        KEEP(*(.boot2));
    } > BOOT2
} INSERT BEFORE .text;
```

config.toml

```
[build]
target = "thumbv6m-none-eabi"

[target.thumbv6m-none-eabi]
runner = "elf2uf2-rs -d"

rustflags = [
 "-C", "link-arg=-Tlink.x",
 "-C", "link-arg=--nmagic",
]
```

build.rs

```rust
use std::env;
use std::fs::File;
use std::io::Write;
use std::path::PathBuf;

fn main() {
   // Put `memory.x` in our output directory and ensure it's
   // on the linker search path.
   let out = &PathBuf::from(env::var_os("OUT_DIR").unwrap());
   File::create(out.join("memory.x"))
       .unwrap()
       .write_all(include_bytes!("memory.x"))
       .unwrap();
   println!("cargo:rustc-link-search={}", out.display());

   // By default, Cargo will re-run a build script whenever
   // any file in the project changes. By specifying `memory.x`
   // here, we ensure the build script is only re-run when
   // `memory.x` is changed.
   println!("cargo:rerun-if-changed=memory.x");
}
```

Building and flashing

```
dawid@atarynka:~/projects/hello_blinky/board$ cargo r --release
```

Demo

blinky-driver

Basic driver

```rust
use embedded_hal::digital::v2::OutputPin;
use embedded_hal::blocking::delay::DelayMs;

pub struct FinalBlinky<Pin: OutputPin, Delayer: DelayMs<u32>> {
   led: Pin,
   delayer: Delayer,
}

impl <Pin: OutputPin, Delayer: DelayMs<u32>> FinalBlinky<Pin, Delayer> {
   pub fn new(led: Pin, delayer: Delayer) -> Self {
       Self {
           led,
           delayer,
       }
   }
   pub fn blink_times(&mut self, count: usize) {
       for _ in 0..count {
           self.led.set_high().unwrap();
           self.delayer.delay_ms(125);
           self.led.set_low().unwrap();
           self.delayer.delay_ms(125);
       }
   }
}
```

Powerful enum

```rust
#[derive(Debug, Clone, Copy, PartialEq, Eq, Default)]
pub enum LedDuration {
    LONG,
    #[default]
    SHORT,
    CUSTOM(u32),
}

impl LedDuration {
    pub const SHORT_MS: u32 = 250;
    pub const LONG_MS: u32 = 750;

    pub fn new(ms: u32) -> Self {
        ms.into()
    }
}

impl From<u32> for LedDuration {
    fn from(ms: u32) -> Self {
        match ms {
            Self::SHORT_MS => LedDuration::SHORT,
            Self::LONG_MS => LedDuration::LONG,
            _ => LedDuration::CUSTOM(ms),
        }
    }
}
```

```rust
impl From<LedDuration> for u32 {
    fn from(value: LedDuration) -> Self {
        match value {
            LedDuration::SHORT => LedDuration::SHORT_MS,
            LedDuration::LONG => LedDuration::LONG_MS,
            LedDuration::CUSTOM(v) => v,
        }
    }
}

#[derive(Debug, Copy, Clone, PartialEq, Eq)]
pub enum LedSignal {
    BLINK(LedDuration),
    PAUSE(LedDuration),
}

impl From<bool> for LedSignal {
    fn from(value: bool) -> Self {
        if value {
            Self::BLINK(LedDuration::default())
        } else {
            Self::PAUSE(LedDuration::default())
        }
    }
}
```

Matching enums

```
pub fn blink_sequence(&mut self, sequence: &[LedSignal]) {
        for signal in sequence {
            match *signal {
                // Just for demo purposes, doesn't make sense as delay should be 0, not 250.
                LedSignal::BLINK(LedDuration::CUSTOM(0)) => {
                    self.delayer.delay_ms(250);
                }
                LedSignal::BLINK(duration) => {
                    self.led.set_high().unwrap();
                    self.delayer.delay_ms(duration.into());
                    self.led.set_low().unwrap();
                    self.delayer.delay_ms(125);
                }
                LedSignal::PAUSE(duration) => {
                    self.delayer.delay_ms(duration.into());
                }
            }
        }
    }
```

Driver usage

Global state

```
static DEVICE: Mutex<
   // Can be RefCell, Cell used to present ownership mechanism.
   Cell<
       Option<
           FinalBlinky<
               gpio::Pin<
                   gpio::bank0::Gpio25,
                   gpio::FunctionSioOutput,
                   PullDown
               >,
               Delay
           >
       >
   >
> = Mutex::new(Cell::new(None));
```

Step by step

- Static
  - global variable
  - mutating is unsafe
- Mutex
  - critical_section crate
  - Provides mutex and critical section for no_std (bare metal)
- Cell
  - core library
  - Provides zero cost internal mutability
- Option
  - core library
  - enumerator
  - Stores **Some**thing of our type or **None**
    - similar to null pointer

Device initialization

```
let delay = cortex_m::delay::Delay::new(core.SYST, clocks.system_clock.freq().to_Hz());
// Definition in which form we want to take ownership of pin,
// in case multiple function are assigned to it.
// As we are using BSP crate, led can be only defined as output here.
let led_pin = pins.led.into_push_pull_output();

let dev = FinalBlinky::new(led_pin, delay);
critical_section::with(
    |cs| DEVICE.borrow(cs).replace(Some(dev))
);
```

Device usage (ie. interrupt)

```rust
// Preparation for interrupts. Not included in this demo.
fn use_device() {
    critical_section::with(
        |cs| {
            let mut dev = DEVICE.borrow(cs).take().unwrap();
            dev.blink_times(5);
            DEVICE.borrow(cs).set(Some(dev));
        }
    );
}
```

Demo #2

Links

Links

- Repo with demo hello #1, #2: https://github.com/taavit/hello_blinky
- Repo with demo tracker #3: https://github.com/taavit/rp2040-logger-hal
- embedded-hal: https://github.com/rust-embedded/embedded-hal
- Rust based OS for automotive: https://oxidos.io/
- Critical safety systems in Rust (Compiler for ESA based LEON cpu): https://ferrous-systems.com/blog/rust-for-mission-critical-applications/
- Key ripper - rp2040 based DIY Keyboard: https://www.youtube.com/watch?v=x7LQevYn7d0
- Aerorust community: https://aerorust.org/

Ok(())