OnMCU

Embedded Rust on RP2350: From Zero to Blink (and Beyond)

30-minute hands-on mini-workshop

Agenda

Learning Outcomes

1. Getting Rust onto a chip is dead simple.
2. Embedded Rust feels like real Rust.
3. You can move forward without losing your mind.

Agenda

Learning Outcomes

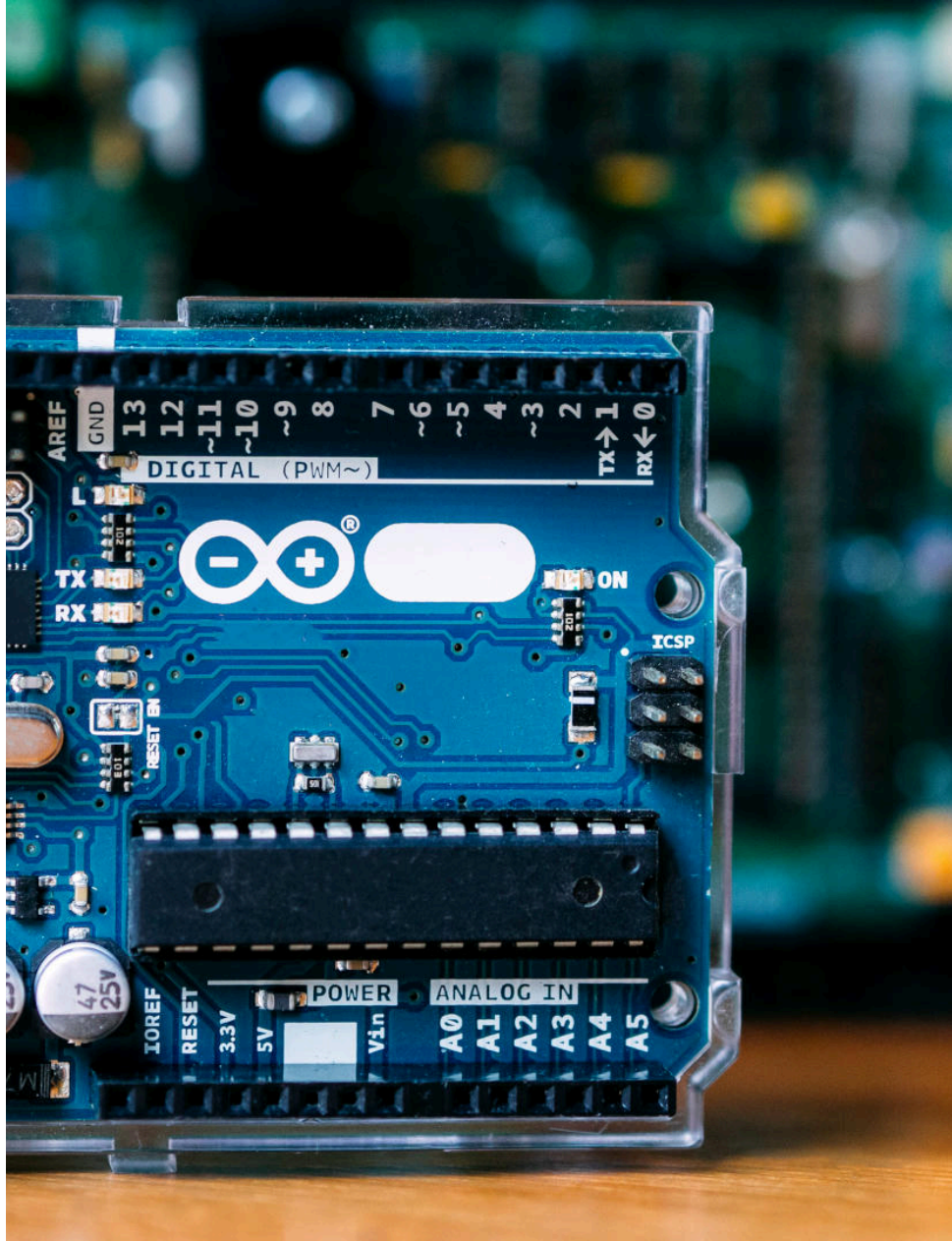
1. Getting Rust onto a chip is dead simple.
2. Embedded Rust feels like real Rust.
3. You can move forward without losing your mind.

Slides available at yrust.de/rust-rp2350

Who this is for

Prerequisites and assumed knowledge

- You are **comfortable using *Rust***
- You have a *Rust* **toolchain** set up on your machine
- We assume **no prior experience** with embedded development



Learning Outcome 1

Flashing Rust is easier than brewing coffee ☕

How to flash: What we'll learn

- What **toolchain** you need (rustup + target + picotool/UF2)
- **Compile and flash** an RP2350 in only 3 steps (no 2-hour setup nightmare)
- Convert `.elf → .uf2`, drag-and-drop and you are running **Rust on the bare metal**
- **Demo:** blinking an LED

Setting up your Rust toolchain

1. Make sure Rust is installed via rustup.rs

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

2. Make sure the toolchain is up-to-date

   ```
   rustup update
   ```

3. Install the `thumbv8m.main-none-eabihf` target

   ```
   rustup target add thumbv8m.main-none-eabihf
   ```

RP2350 MCU Flashing Methods

Method 1: Online Conversion

Prerequisites: None

1. Go to elf2uf2.yrust.de[1]
2. Upload and convert your `.elf` file to `.uf2`
3. Put RP2350 into boot mode:
   - Hold **BOOTSEL** button
   - Power on
   - Release **BOOTSEL**
4. Drag & drop the `.uf2` file to the mounted drive
5. Device will automatically reset and run

Method 2: Bootloader and picotool

Prerequisites: picotool installed

1. Put RP2350 into boot mode:
   - Hold **BOOTSEL** button
   - Power on
   - Release **BOOTSEL**
2. Flash directly with picotool:

```
picotool load -u -v -x -t elf your_file.elf
```

3. Device will automatically reset and run

Method 3: Debug Probe

Prerequisites: Debugger + probe-rs installed

1. Connect debugger or second Pico to target RP2350
2. Flash and run with probe-rs:

```
probe-rs run --chip RP235x --protocol swd
your_file.elf
```

3. Device will be programmed and starts running automatically

Additional Notes

To erase the flash, you can use `picotool erase` or drag & drop the flash nuke utility: flash_nuke.uf2 (direct download, 96 kB)

[1]Our online tool elf2uf2.yrust.de simply executes `picotool uf2 convert your_file.elf your_file.uf2` - it's equivalent to having picotool installed locally but runs in the browser.

The Hello World of Firmware: Blinking an LED

```rust
// Set up GPIO (General Purpose Input/Output) pins
let sio = Sio::new(dp.SIO); // Single-cycle I/O block
let pins = hal::gpio::Pins::new(dp.IO_BANK0, dp.PADS_BANK0, sio.gpio_bank0, &mut dp.RESETS);

// Configure GPIO pin 25 as an output pin (this is where the onboard LED is connected)
let mut led = pins.gpio25.into_push_pull_output();

// Main program loop - runs forever
loop {
    led.set_high().ok(); // Turn the LED on (ignore any errors with .ok())
    delay.delay_ms(500); // Wait for 500 milliseconds (half a second)
    led.set_low().ok();  // Turn the LED off
    delay.delay_ms(500); // Wait for another 500 milliseconds
    // Loop continues, creating a blinking pattern: on 500ms, off 500ms, repeat
}
```

Repository available at github.com/systemscape/pico2-blink.

We have pre-built UF2-images in the repo so you can flash even if building fails.

Takeaway: ***You can put Rust on real hardware faster than you expect.***

Learning Outcome 2

Embedded Rust feels like Rust — not like C. 🤮

Rust stays idiomatic: What we'll learn

- What is a HAL and why does `embedded-hal` make Rust so **special**
- The borrow checker, traits and async patterns – they all **just work.**
- **Example:** Drivers and GPIOs

Embedded-HAL

```rust
// Embedded-HAL defines SpiDevice
pub trait SpiDevice {
    fn read(&mut self, buf: &mut [u8]);
    fn write(&mut self, buf &[u8]);
}

// MCU HAL creates SpiDevice
impl SpiDevice for McuSpi {
    fn read(&mut self, buf: &mut [u8]) {
        // HAL abstracts away register access
        pac::spi_control_register::write(0x01);
    }
}

// Driver consumes SpiDevice
impl<T: SpiDevice> Driver<T> {
    pub fn new(spi_device: T) -> Self { ...}
    pub fn foo(&mut self) { self.spi_device.read(); self.spi_device.write(); }
}
```

Embedded-HAL

- App logic can use **different drivers**
- App logic can use **different HALs**
- Only **requirement:** everyone implements `embedded-hal` traits

→ Using a **different HAL** lets you use a **different chip** (even different CPU architecture!)

Generic Async Code? In `#[no_std]`?

```rust
#[embassy_executor::main]
async fn main(spawner: Spawner) {
    let p = embassy_rp::init(Default::default());
    let led = Output::new(p.PIN_25, Level::Low);
    spawner.spawn(blink(led)).unwrap(); // Returns immediately
}
// One of many embassy tasks
#[embassy_executor::task]
async fn blink(led: Output<'static>) {
    generic_blink(led, embassy_time::Delay {}).await;
}
// `led` and `delay` must implement embedded-hal traits
async fn generic_blink(mut led: impl OutputPin, mut delay: impl DelayNs) {
    loop {
        led.set_high().unwrap();
        delay.delay_ms(500).await; // No busy waiting!
        led.set_low().unwrap();
        delay.delay_ms(500).await;
    }
}
```

Takeaway:

You don’t need to “unlearn” Rust to do embedded.

Learning Outcome 3

Scale without pain: async & tooling to keep you sane 📈

You can move forward without losing your mind: What we'll learn

- Async and `embassy` make concurrency predictable and power-friendly.
- Tooling (`cargo`, `probe-rs`, `defmt`) reduces trial-and-error time.
- Going forward: Further learning resources
- **OnMCU** for CI/CD and real project workflows

Takeaway:

Developing embedded systems has never been easier and more fun!

Contact

OnMCU

Phone: see website

E-Mail: see website

www.onmcu.com

Disclaimer

This presentation is intended solely for the designated recipient(s) and is provided for informational purposes only. While every effort has been made to ensure the accuracy of the information, no guarantees or warranties are made, express or implied. The presenter accepts no liability for any decisions or actions taken based on the content herein.

OnMCU

Embedded Rust on RP2350: From Zero to Blink (and Beyond)

30-minute hands-on mini-workshop

www.onmcu.com