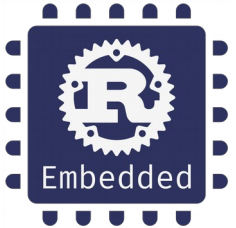
Embedded Rust Workshop

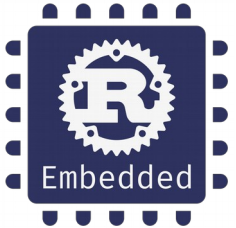


Hanno Braun
https://braun-embedded.com/

Matthias Endler
https://endler.dev

Please download the following package:
https://braun-embedded.com/barcelona-2019-11-09.zip

Today's Agenda



0. Introduction
1. Install prerequisites
2. Blink an LED
3. Go below the surface
4. Bonus round!

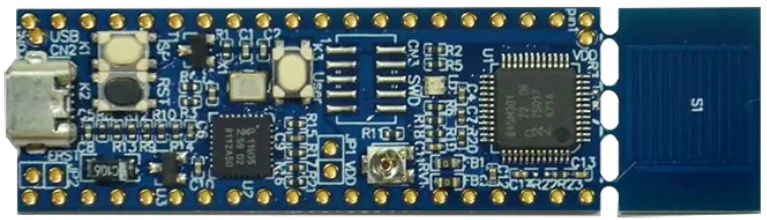
What is an embedded system?

My favorite definition: **A computer that you don’t sit in front of.**

That covers a wide range of systems, but today we’ll focus on the lower end: Microcontrollers with RAM and flash space measured in kilobytes.

The Hardware

LPC845-BRK Development Board

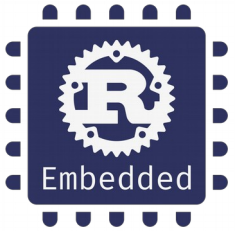


NXP LPC845 MCU
(ARM Cortex-M0+)

30 MHz clock frequency
64 kB Flash
16 kB RAM

Kindly sponsored by NXP!

Why Rust for Embedded?

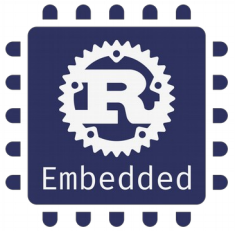


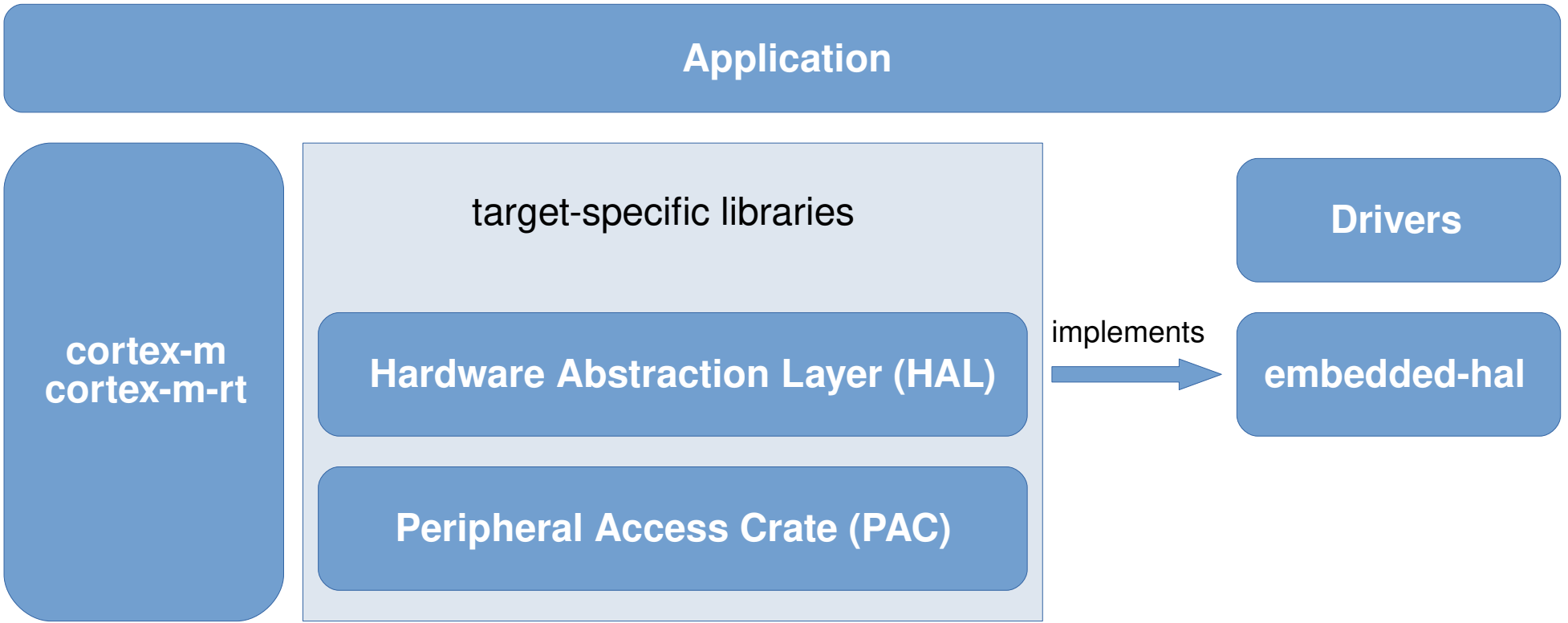
Reliability: Common errors are prevented and rules can be enforced at compile-time.

Convenience: For example a rich type system and modern tools, like Cargo.

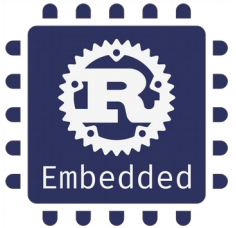
Efficiency: Despite all that, Rust has no runtime overhead compared to C/C++.

Rust's Embedded Ecosystem





Task 1: Install prerequisites



We need the following software:
1. Rust/rustup
2. Rust target for microcontroller
3. OpenOCD (more recent than the latest stable release)
4. arm-none-eabi-gdb

See installation instructions in the package you downloaded (***installation-instructions.md***) or https://braun-embedded.com/workshop/.

Verify that everything worked:
git clone https://github.com/lpc-rs/lpc8xx-hal.git
cd lpc8xx-hal; cargo run --release --features=845-rt --example gpio_delay

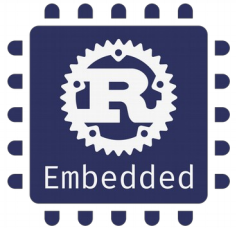
If everything worked correctly, the red LED should blink.

Sometimes the program doesn't start after being uploaded to the board. If you get a weird error message, reset the board (button marked **RST**), then it should work.

Intermission: A minimal application

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


extern crate panic_halt;


use cortex_m_rt::entry;


use lpc8xx_hal::Peripherals;


#[entry]
fn main() -> ! {
    let _p = Peripherals::take().unwrap();

    loop {}
}
```

Task 2: Blink an LED



Extend the minimal application to blink an LED.

To do this, you have to find out which pin is connected to an LED on the board, configure that pin correctly, then set it high and low in a loop (with some delays in between) to get it to blink.

You can make this easy or hard for yourself, depending on how much you want to learn. The easy way would be to copy from one of the GPIO examples in the *lpc8xx-hal* repository. The hard way would be to figure out how to do all of this from the documentation of the board, the microcontroller, and the HAL. I recommend mixing both approaches.

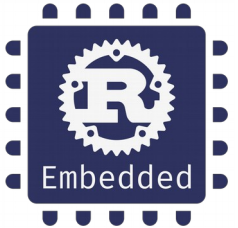
Some hints:

- You can find documentation for the hardware (board and microcontroller) in the *documentation* directory of the package you downloaded.
- Documentation for the HAL and other crates used is linked on https://crates.io/. Search for *lpc8xx-hal*, *lpc8xx-pac*, *cortex-m*, or any other library you might need.
- General Rust documentation: https://doc.rust-lang.org/book/ https://doc.rust-lang.org/book/ and https://doc.rust-lang.org/stable/rust-by-example/.

Bonus tasks:

- Blink multiple LEDs
- Experimentn with different blinking patterns
- Change blinking patterns over time
- If you want to learn more about what makes the application tick, check out all files and try to understand what they do.

Task 3: Go below the surface

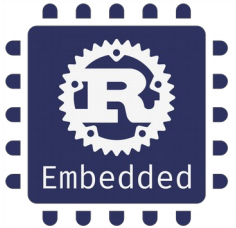


The HAL is designed to provide a safe and convenient interface to the hardware, because the hardware itself is often complicated and horrible. Let's find out more about that!

The HAL is built on top of the PAC (Peripheral Access Crate) and re-exports it as *lpc8xx-pac*. Rewrite your current program using only the PAC!

Bonus task:
Did you make any mistakes while rewriting your program, like forgetting to initialize something before trying to use it? Try to replicate these same mistakes using the HAL crate, to find some ways how the HAL protects you.

Task 4: Bonus Round!



Now that you've learned how to use the PAC to access any hardware features that aren't supported yet by the HAL, there's really no limit to what you can do. Here are some ideas.

1. Try to send text to the host PC via USB.
There are to pull requests trying to make this work for LPC845 in the *lpc8xx-hal* repository (https://github.com/lpc-rs/lpc8xx-hal). They don't work yet, but maybe you figure out why.

2. Write a game
Matthias wants to see how a game works when you only have one button and one pixel. Maybe figure it out and show him?

To write a game, you'll probably need support for some timer. Checkout the MRT or the SysTick in the LPC845 user manual.