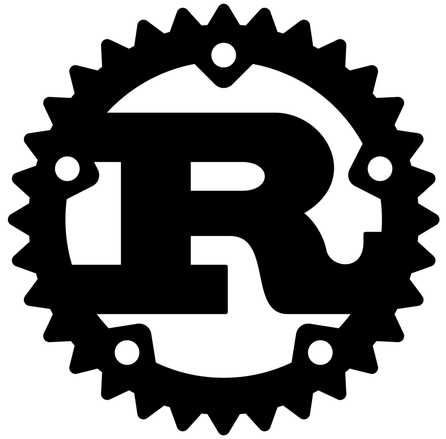
Embedded Systems with Rust

AeroRust Workshop Day



Omar Hiari - June 2023

About Me

Associate Professor of Computer Engineering at GJU

Formerly in Industry (~10 years) at companies like Intel, Magna, and Continental Automotive Systems

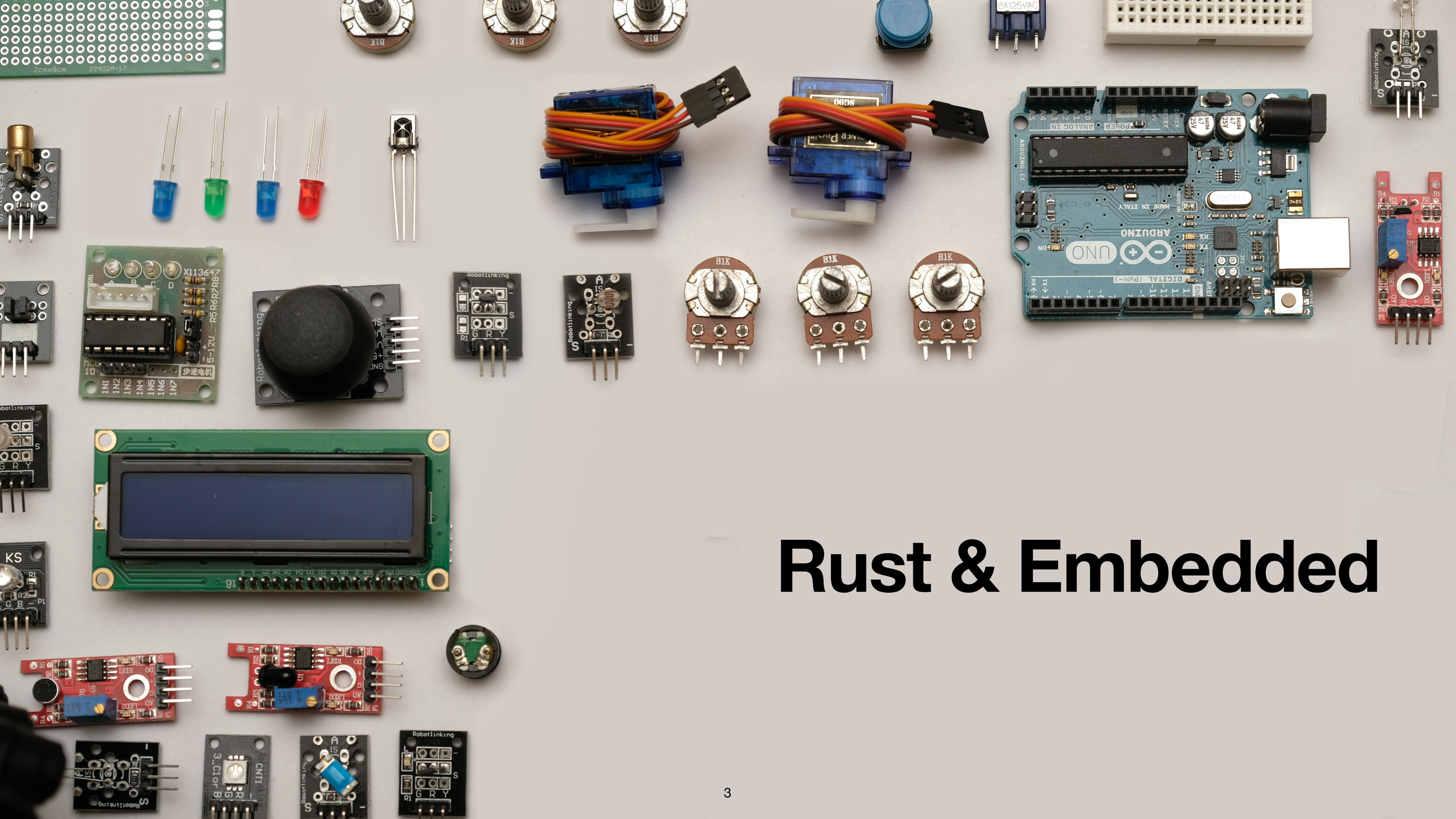
Most academic interests and professional experience in or around Embedded Systems

Blog 👉 **https://apollolabsblog.hashnode.dev/**

Twitter 👉 **@apollolabsbin**

Repo 👉 **https://github.com/apollolabsdev**

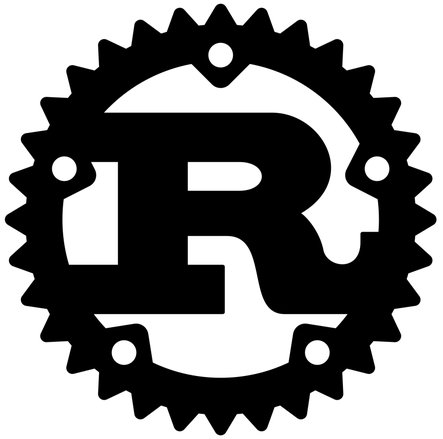
Newsletter 👇



Rust & Embedded

Why Rust?

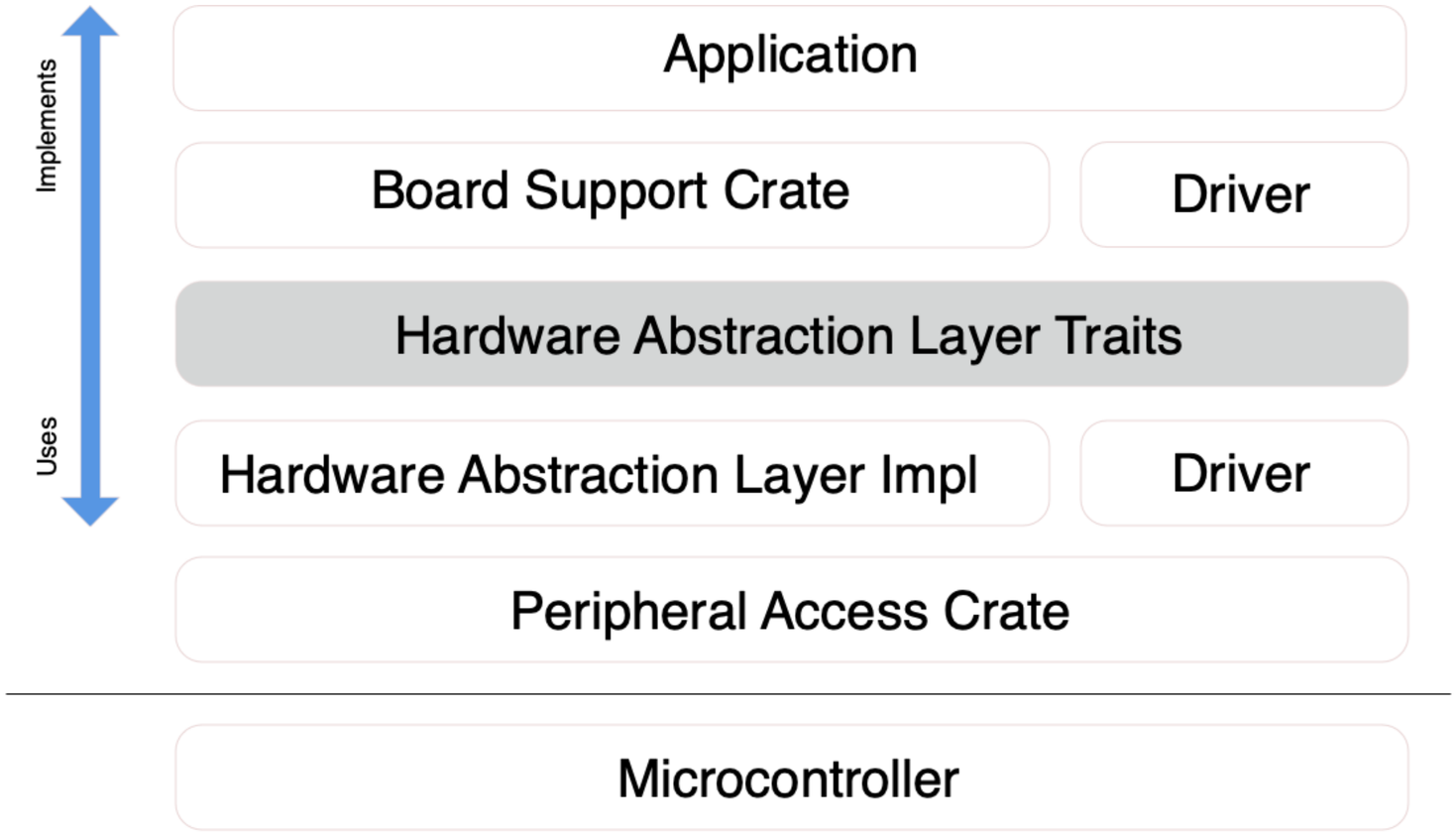
- **Memory Safe** 🧠
  - No garbage collection
  - Borrow Checker for static memory checks
- **Fast** 🏎️
  - No Underlying Runtime
- **Easy Concurrency** 🤹
  - Compiler Ensures No Data Race Conditions
- **Zero Cost Abstractions** 💰
  - Abstractions Don’t Generate Extra Code



Great Fit for Mission Critical & Safety Applications!

Embedded Rust

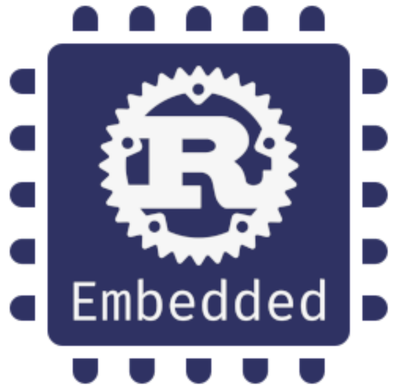
Abstractions

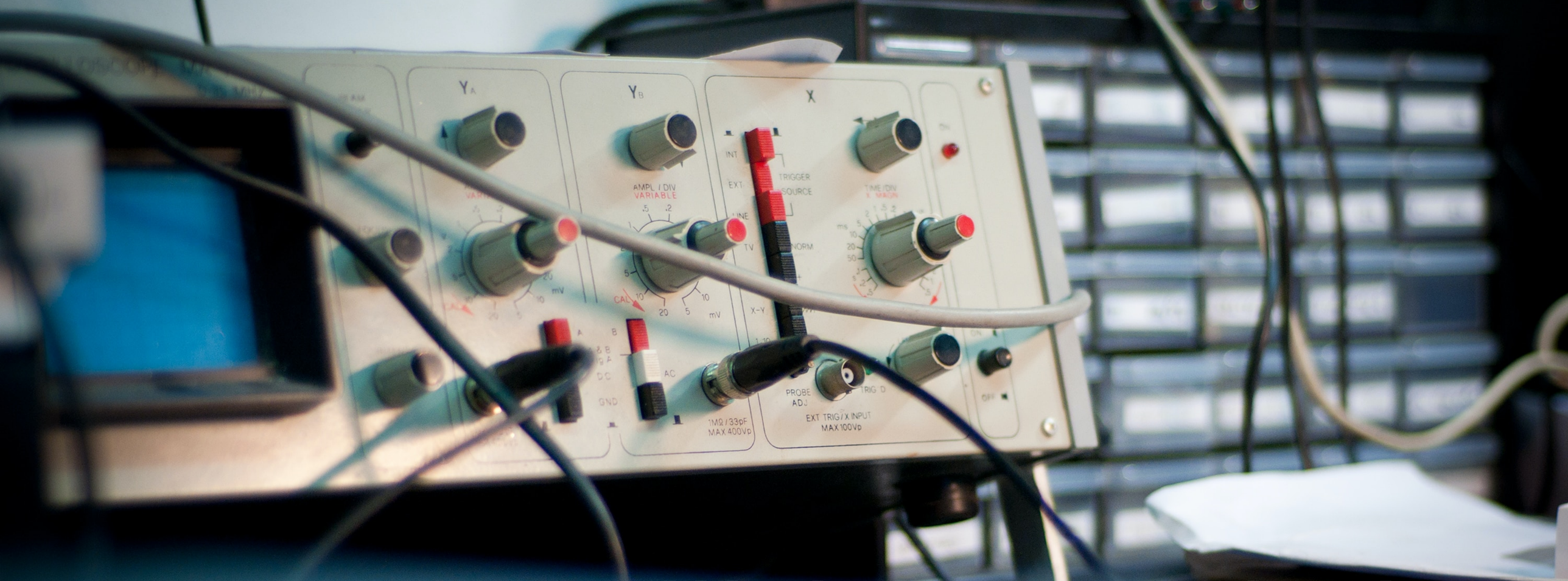


Embedded Rust

Important Ecosystem Frameworks and Crates

- **RTIC** 
  - Real-Time Interrupt-driven Concurrency
- **Embassy**
  - Enables use of async (Will be used in this workshop)
- **Embedded-hal**
  - Enables platform-agnostic driver implementations





Workshop Introduction

Your Mission

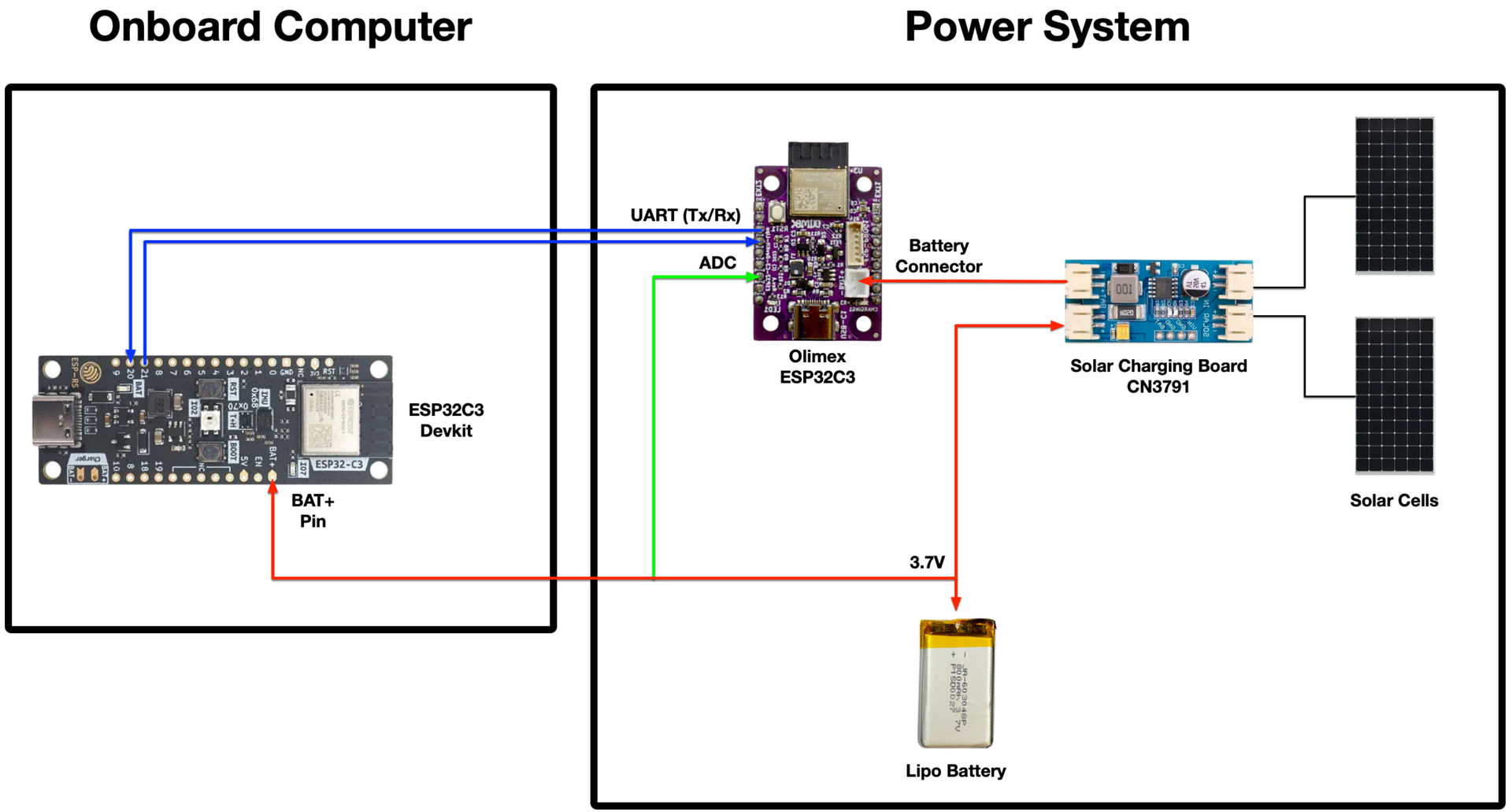
You are going to create a framework for a small pseudo nano satellite that will be launched into space.

The satellite will:

- Monitor Satellite Battery Levels
- Manage Satellite Power Consumption
- Collect GNSS Data
- Collect Battery Temperature Data

Your mission is to use Rust to complete the different parts of the code.

The Satellite System



Task 1
Environment Setup

Setting Up the Environment

Dependencies 🦯

1. Install Rust ®️

- http://rustup.rs

2. Install Target Toolchain ⛓️

- Install the nightly toolchain with the rust-src component:

```
rustup toolchain install nightly --component rust-src
```

3. Set The Target for ESP32-C3 🎯

```
rustup target add riscv32imc-unknown-none-elf
```

Setting Up the Environment

Tooling 🧰

1.Install VSCode 👨‍💻

Recommended Extensions:

- **Rust Analyzer**
- **Even Better TOML**
- **Crates**

2.Install esp-flash 📸

```
cargo-binstall espflash@2.0.0-rc.3
```

Creating Projects

Generating Project From Standard Template

1. Install Cargo Generate

```
cargo install cargo-generate
```

2. Generate no-std Template from Repo

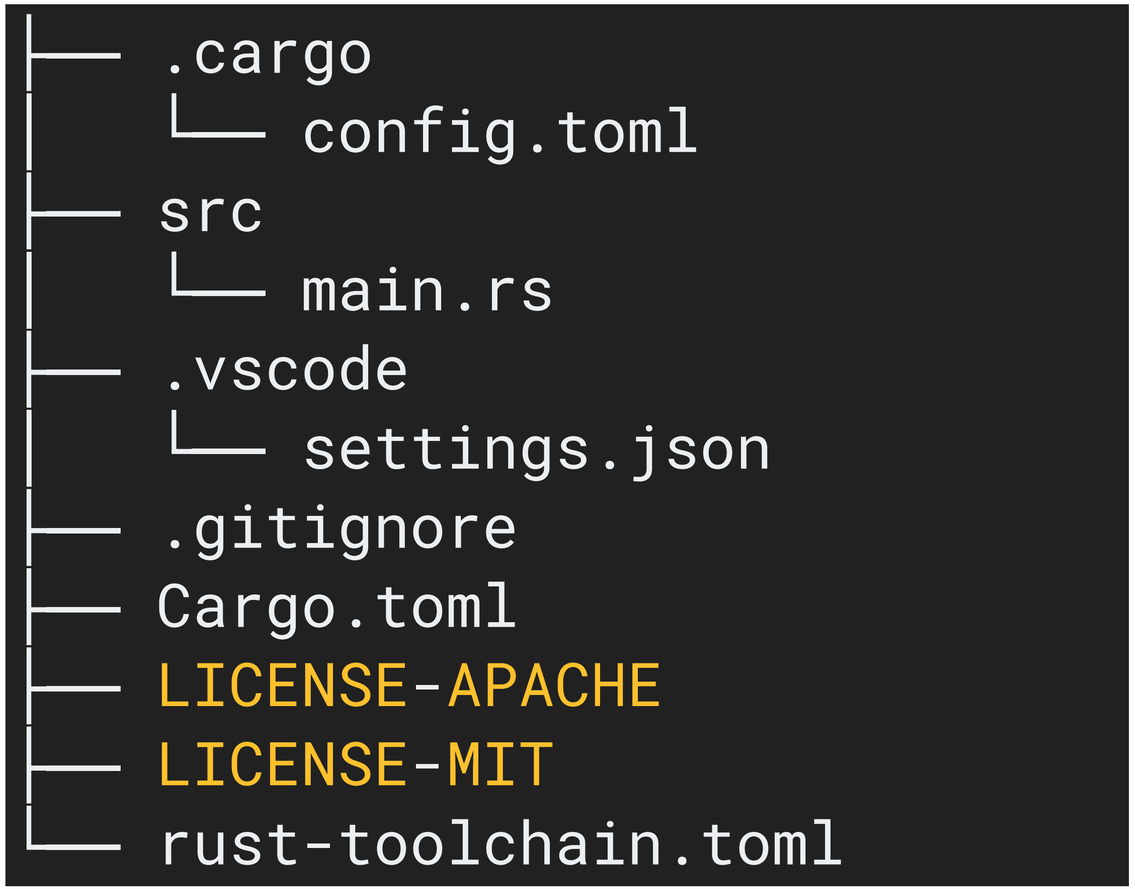
```
cargo generate --git https://github.com/esp-rs/esp-template
```

3. Connect ESP Board then Run and Flash

```
cargo run
```

Inspecting the Basic Template

Directory Structure 📦



⬅️ Cargo config and build options

⬅️ Main source file

⬅️ Settings for VSCode (optional)

⬅️ Git ignore files definition

⬅️ Project Dependencies

⬅️ Applies Licenses

↙️

⬅️ Defines which Rust Toolchain to use

The `main.rs`

Part 1️⃣

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

⬅️ Tell Compiler we're not using Standard Library

⬅️ Tell Compiler we're not using standard main function (we'll define our own)

```
use esp32c3_hal::{
clock::ClockControl,
pac::Peripherals,
prelude::*,
timer::TimerGroup,
Rtc,
};
use esp_backtrace as _;
```

} Type imports from the ESP32C3 HAL

⬅️ Define `panic` behavior

The `main.rs`

Part 2️⃣

Entry attribute to define custom entry point (`main`) from the riscv-rt crate

```
#[riscv_rt::entry]
fn main() -> ! {}
```

Take all the peripherals from the PAC to pass them to the HAL drivers

```
let peripherals = Peripherals::take().unwrap();
```

Promote System peripheral to HAL driver and configure then freeze clocks

```
let system = peripherals.SYSTEM.split();
let clocks =
ClockControl::boot_defaults(system.clock_control).freeze();
```

The `main.rs`

Part 3️⃣

Code necessary to disable ESP watchdogs

```
// Disable the RTC and TIMG watchdog timers
let mut rtc = Rtc::new(peripherals.RTC_CNTL);
let timer_group0 = TimerGroup::new(peripherals.TIMG0, &clocks);
let mut wdt0 = timer_group0.wdt;
let timer_group1 = TimerGroup::new(peripherals.TIMG1, &clocks);
let mut wdt1 = timer_group1.wdt;

rtc.swd.disable();
rtc.rwdt.disable();
wdt0.disable();
wdt1.disable();
```

The Application Loop

```
loop{}
```

Hello World Exercise 🏋️

Adding Dependencies

Either in `Cargo.toml` under `[dependencies]`

```
esp-println = { version = "0.3.1", features = ["esp32c3"] }
```

Or using Cargo

```
cargo add esp-println --features "esp32c3"
```

Hello World Exercise 🏋️

Adding Code and Running

In the `main` `loop` add the following line

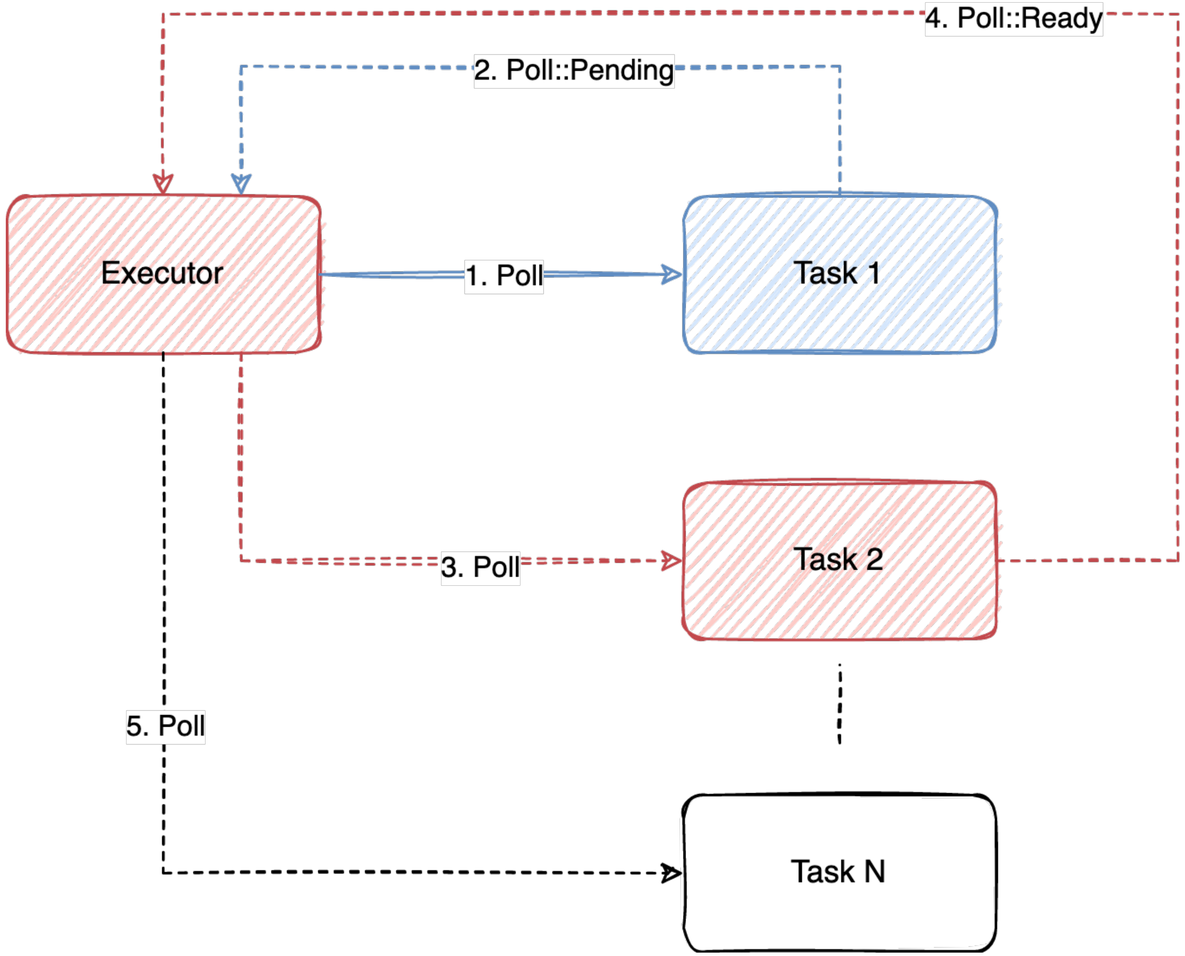
```
esp_println::println!("Hello World");
```

Compile and flash using Cargo

```
cargo run
```

The Embassy Framework

The Executor 🔪



The Embassy Framework

Task Declaration

```
#[embassy_executor::task]
async fn blinker(mut led: Output<'static, P0_13>, interval: Duration) {
    loop {
        led.set_high();
        Timer::after(interval).await;
        led.set_low();
        Timer::after(interval).await;
    }
}
```

The Embassy Framework

The `main` Task

```
#[embassy_executor::main]
async fn main(spawner: Spawner) {
    let p = embassy_nrf::init(Default::default());

    let led = Output::new(p.P0_13, Level::Low, OutputDrive::Standard);
    unwrap!(spawner.spawn(blinker(led, Duration::from_millis(300))));
}
```

Workshop Project Files

Clone GitHub Repository

🔗 **GitHub Repo Link**

👇

https://github.com/LechevSpace/nanosat-workshop.git

The Cloned Project

Each Board has a Dedicated Directory

💻 Onboard Computer 👉 nanosat-workshop/onboard-computer

🔌 Power System 👉 nanosat-workshop/power-system

📝 Each directory has an application.rs in its source files 👉 this where you will be adding your code

📝 All the project templates use the embassy Framework

Task 2
Building the Power System - Part 1

Introduction

Requirements

- **The power system needs to monitor 🔍 the battery level 🔋 of the satellite system 🛰️**

  📏 Battery terminal is wired to ESP ADC pin gpio3

- **The power system needs to blink the LED light 🚨 in the power system 🔌**

  🚦 LED is wired to gpio8

Introduction

Tasks 👷

What You Need to Do:

1️⃣ In the `Application init` function configure the ADC and GPIO peripherals

2️⃣ Set up the `battery_measurement_adc` task to take an ADC measurement ever 300 ms

3️⃣ Set up `blink` task to blink LED every 500ms

Introduction

GPIO Configuration & Usage

Instantiate and create handle for IO

```
let io = IO::new(peripherals.GPIO, peripherals.IO_MUX);
```

Configure LED to an output

```
let mut led = io.pins.gpio7.into_push_pull_output();
```

Control LED

```
led.set_low().unwrap();
```

Additional methods are available in HAL documentation

Introduction

ADC Configuration - Part 1

Obtain a handle for ADC configuration

```
let mut adc_config = AdcConfig::new();
```

Configure ADC pin to analog pin

```
let adc_pin = io.pins.gpio3.into_analog();
```

Enable the analog pin

```
let mut adc_pin = adc_config.enable_pin(
            adc_pin,
            Attenuation::Attenuation11dB
        );
```

Introduction

ADC Configuration - Part 2

Promote ADC struct to HAL

```
let analog = peripherals.APB_SARADC.split();
```

Configure ADC

```
let mut adc = ADC::adc(
    &mut system.peripheral_clock_control,
    analog.adc1,
    adc_config,
)
.unwrap();
```

Introduction

ADC Usage

Obtaining an ADC Reading

```
let sample: u16 = adc.read(&mut adc_pin).unwrap();
```

Task 3
Building the Power System - Part 2

Introduction

Requirements

- **The power system needs to send the satellite system battery level to the on-board computer**

  📟 UART Tx/Rx pins are gpio21 (Tx) and gpio20 (Rx)

- **The power system is also supposed to provide a visual indicator**

  🔋 Voltage ok 👉 On-board LED blinking

  🪫 Voltage not ok (drops below threshold) 👉 On-board LED solid

Introduction

Tasks 👷

What You Need to Do:

1️⃣ In the `Application init` function configure the UART peripheral

2️⃣ Set up the `uart_comm` task to send battery percentage value every 1s

3️⃣ Rewrite the `blink` task to accommodate new blinking logic

Introduction

UART Configuration & Usage

Configure UART with defaults (8N1 @ 115200 Baud)

```
let mut uart0 = Uart::new(peripherals.UART0);
```

For custom settings there exists the `new_with_config` function

Use `write` method to send:

```
uart.write(u8Var).unwrap();
```

Additional methods are available in HAL documentation

Introduction

Sharing Data

Sharing among tasks has to happen safely without causing race conditions

Various types exist to share data, among them is the `AtomicU8`

`AtomicU8` is obtained from `core::sync::atomic`

Instantiation:

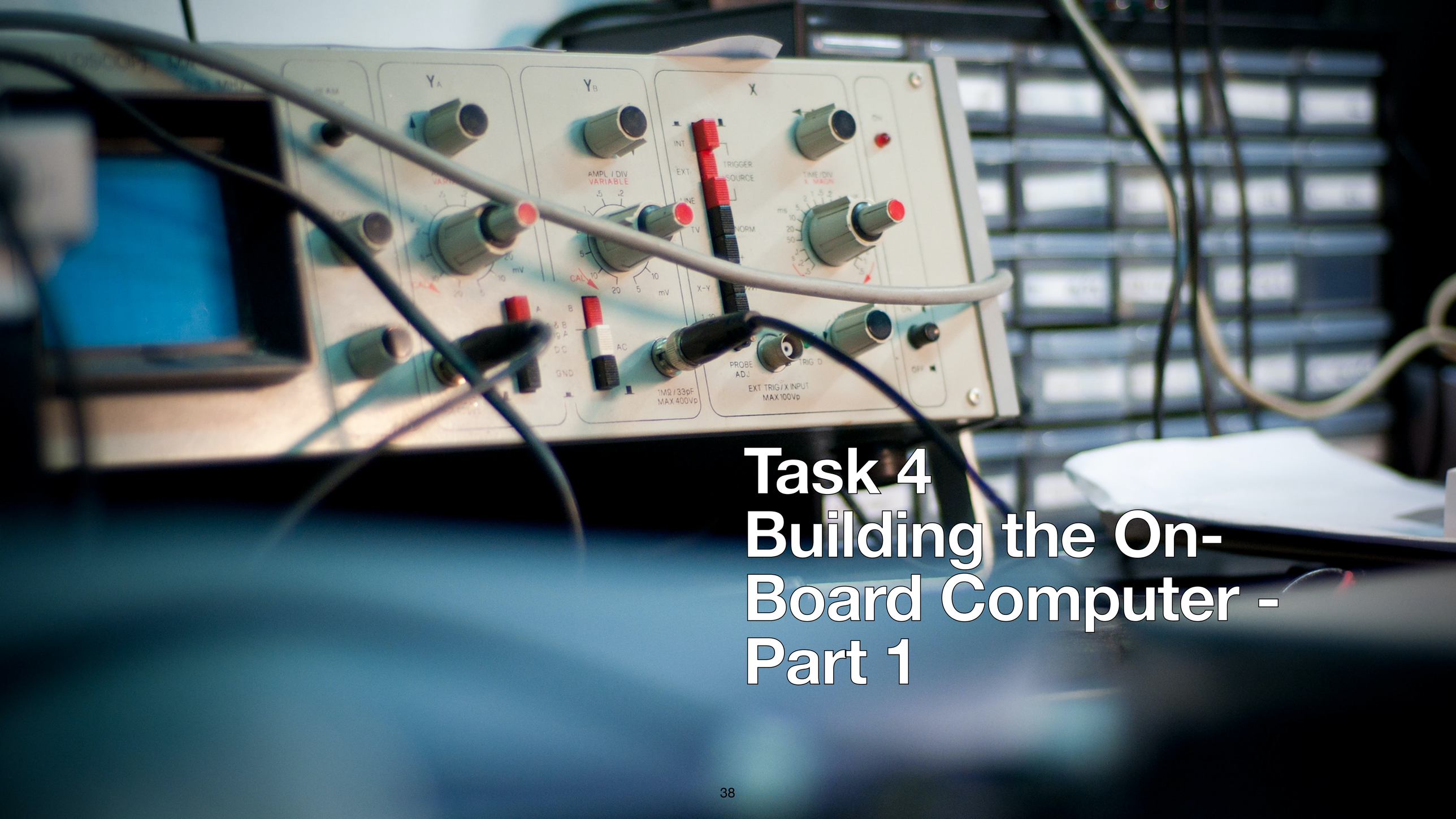
```
static BATT: AtomicU8 = AtomicU8::new(0);
```

Safe Access:

```
BATT.load(Ordering::Relaxed)
```

&

```
BATT.store(data_u8, Ordering::Relaxed)
```



Task 4
Building the On-Board Computer - Part 1

Introduction

Requirements

- **The power system needs to read the satellite system battery level sent from the power system**

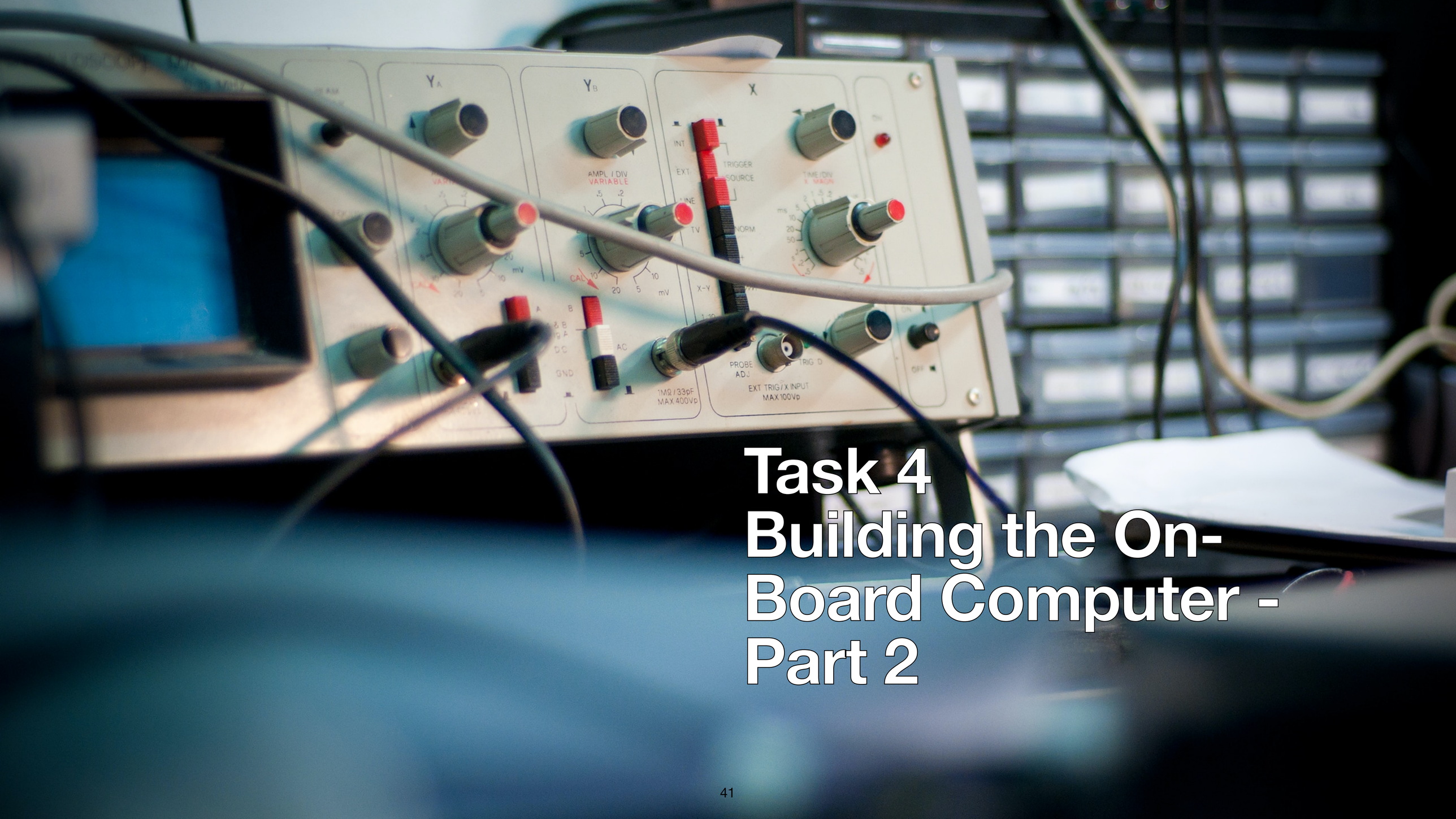
  UART Tx/Rx pins are gpio21 (Tx) and gpio20 (Rx)

Introduction

Tasks 👷

What You Need to Do:

1️⃣ In the `uart_comm` task read the battery percentage value every 1s and print it to the console



Task 4 Building the On-Board Computer - Part 2

Introduction

Requirements

- The onboard computer will parse GNSS messages using the `nmea` crate every 2 seconds from a log
- The onboard computer will extract (and print) from GNS messages the number of satellites and from GSV messages the satellites in view

Introduction

Tasks 👷

What You Need to Do:

1️⃣ In the `gnss` task read lines randomly from the `nmea.log` file in the `tests` folder

2️⃣ In the same task parse the read line using the `parse_str` from the `nmea` crate

3️⃣ Extract and print `nsattelites` from GNS messages and `_sats_in_view` from GSV messages

Final Feedback Form

🔗 https://forms.gle/cMEZHHRnzigPBhfCA