Embedded With Rust

Writing Embedded Software Using
The Rust Programming Language

Jacob Creedon

What’s wrong with C/C++?

- Problems with C/C++
  - Unsafe by default
  - Safety guidelines exist, (e.g. MISRA) but they require proprietary (and expensive) static analysis tools.
  - Tooling can require a lot of work to configure

- New systems programming language contenders
  - Go (Sponsored by Google)
  - D (Based on C++)
  - Rust (Sponsored by Mozilla)

Should I be starting a new codebase in C/C++?

What about embedded?

With Rust, maybe...

Why Rust?

- Zero-cost/low-cost abstractions (just like C++)
- Reduction of undefined behavior
- Focused on concurrency and memory safety
- Safe by default
- Ownership, borrowing, lifetimes
- No garbage collection
- Excellent build system and package manager (Cargo)

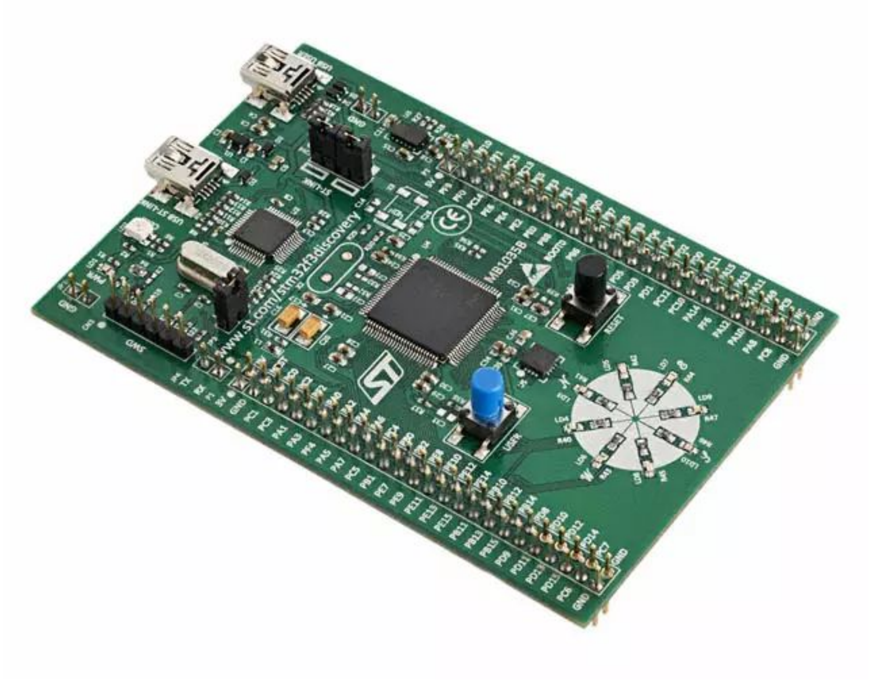
Project - MIDI Controller

- I need a little extra control when doing live events
- First get a blinking light
- Then get fader readings

- Outline
  - Hardware
  - Setup Dev Environment
  - Ecosystem Overview
  - Write Code
  - Flash and debug

Hardware Platform

- ST Micro F3 Discovery
- ~$15 from Digikey
- STM32F303VCT6
- 8 LEDs
- Accelerometer
- On board debugger

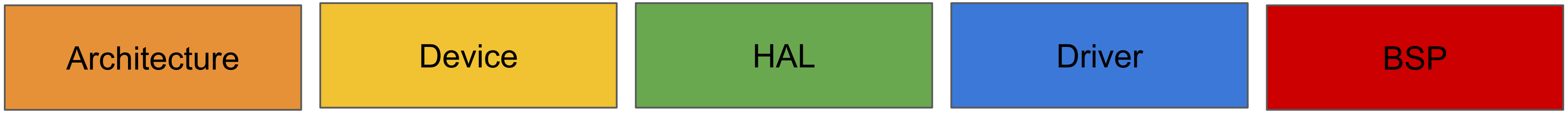


Install Rust and Toolchains

- Download and run rustup (www.rustup.rs)
  - Rustup is the toolchain manager
- Install the nightly toolchain
- Install GCC toolchain for arm-none-eabi-*
  - This is to get GDB
- Install GDB Server as needed
  - OpenOCD, JLink, etc.
  - Or just use a Black Magic Probe

- Install Editor or IDE
  - Visual Studio Code
    - “Rust (rls)” Plugin
  - IntelliJ IDEA
    - Intellij-rust plugin

Cargo, Crates, and Layers of Abstraction



- Cargo is the Rust package manager and build system
- Packages are called “Crates”
- Downloads and builds dependencies then builds your project

- Architecture Crate
- Device Crate
- Board Support Crate
- HAL Implementation Crate
- Driver Crate

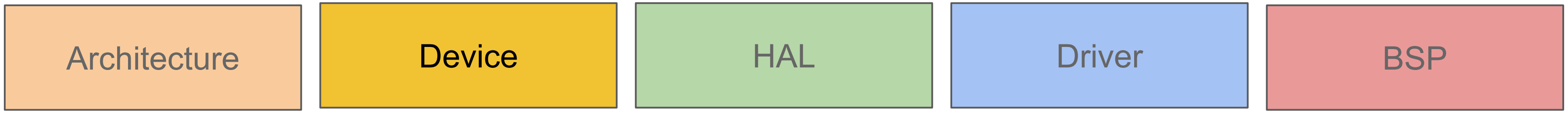
Cargo, Crates, and Layers of Abstraction

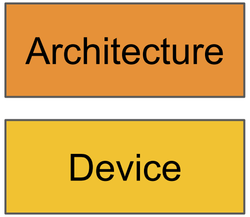
Architecture | Device | HAL | Driver | BSP

Architecture

- Registers common to the architecture
- Example, Cortex-M:
  - SYSTICK
  - ITM
  - etc.

Cargo, Crates, and Layers of Abstraction

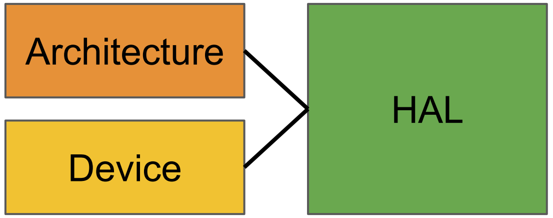




- Registers specific to the chip and its peripherals
- Crates can be auto-generated from SVD files
- Depending on the SVD, registers have meaningful names with named bitfields

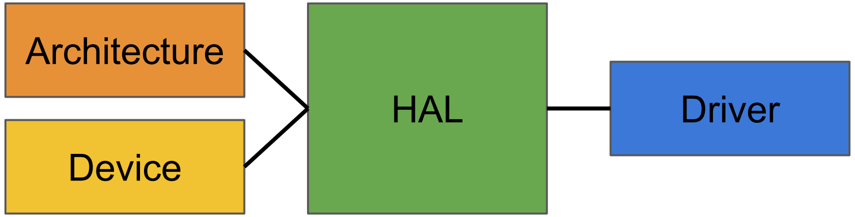
Cargo, Crates, and Layers of Abstraction

Architecture
Device
HAL
Driver
BSP



- Abstraction on top of common device peripherals to provide a consistent interface, e.g.:
  - SPI, I2C, Serial
- This is kind of like how Arduino acts as a HAL
- Implement logic not represented in SVD File

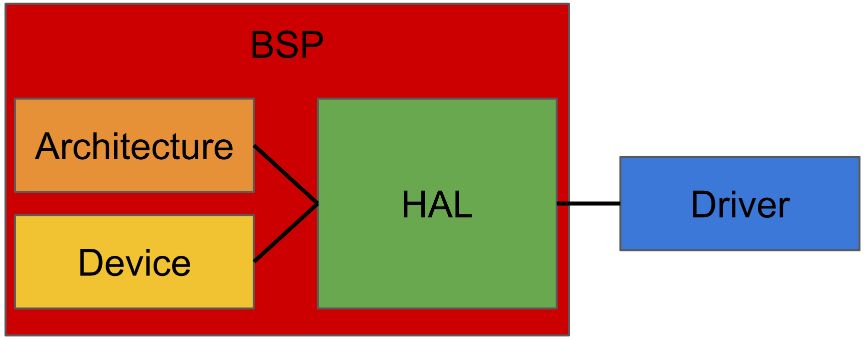
Cargo, Crates, and Layers of Abstraction



- Drivers for external components accessible via the HAL
- Example: a LSM303DLHC I2C Accelerometer driver uses the HAL

Cargo, Crates, and Layers of Abstraction

Architecture
Device
HAL
Driver
BSP

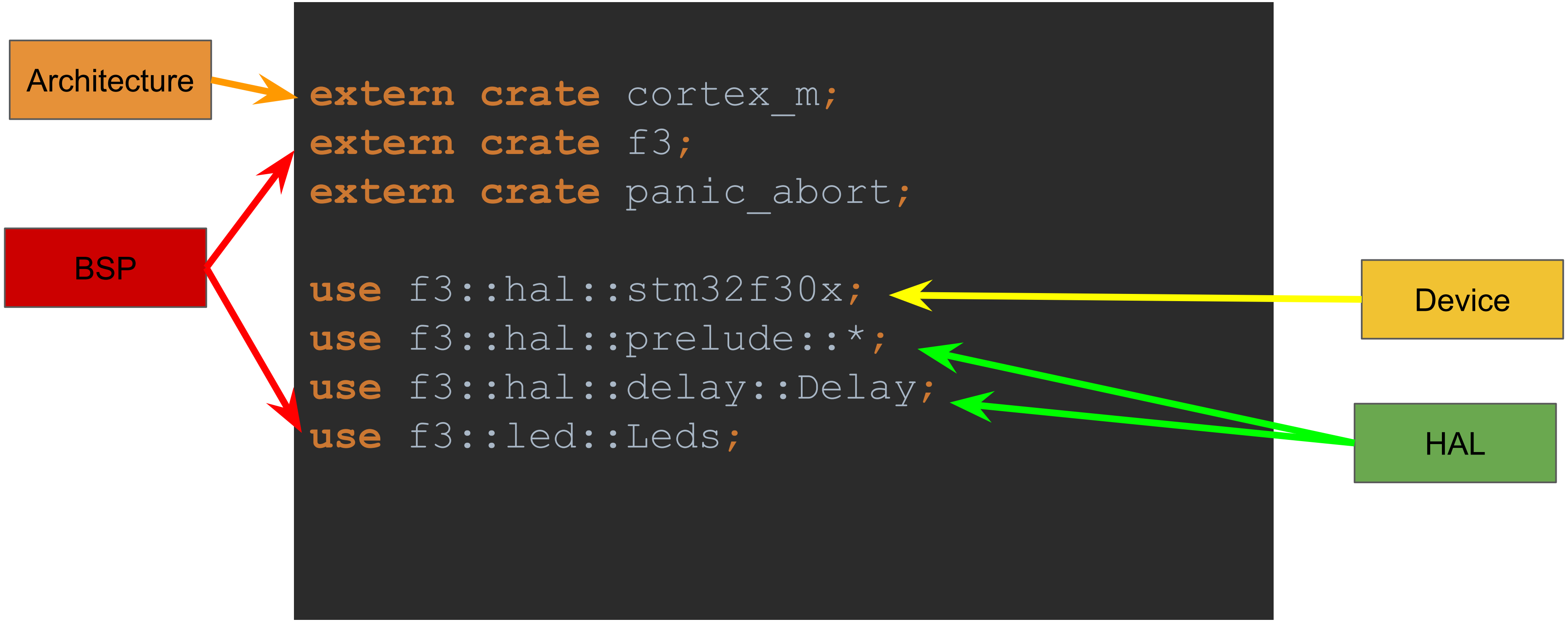


- Acts as a collection of crates necessary for working with a particular development board
- Often has aliases for board features e.g. led -> PC13
- Very few of these around, but useful for hitting the ground running if available

Write Code

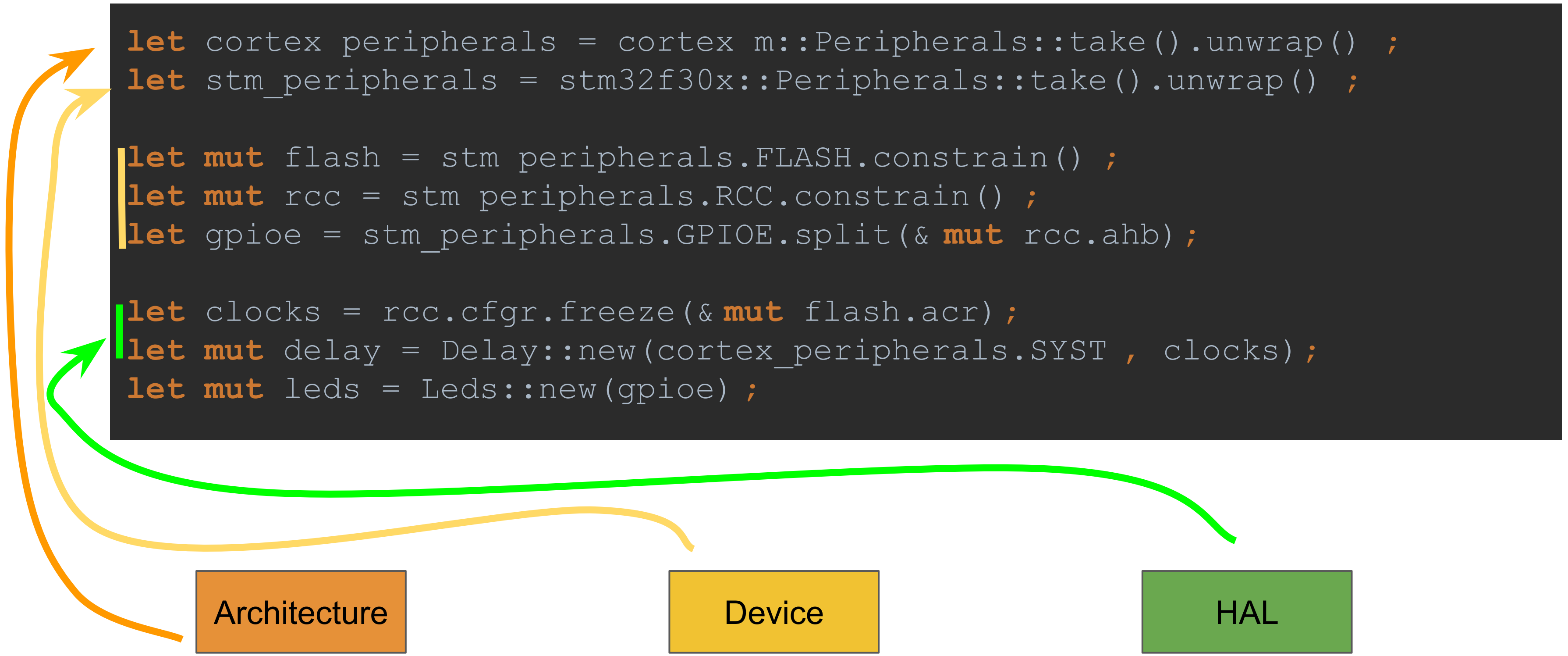
```
#![no std]
#![deny(unsafe code)]
#![deny(warnings)]

extern crate cortex_m;
extern crate f3;
extern crate panic_abort;

use f3::hal::stm32f30x;
use f3::hal::prelude::*;
use f3::hal::delay::Delay;
use f3::led::Leds;

fn main() {
   let cortex peripherals = cortex m::Peripherals::take().unwrap();
   let stm_peripherals = stm32f30x::Peripherals::take().unwrap();

   let mut flash = stm peripherals.FLASH.constrain();
   let mut rcc = stm peripherals.RCC.constrain();
   let gpioe = stm_peripherals.GPIOE.split(&mut rcc.ahb);

   let clocks = rcc.cfgr.freeze(&mut flash.acr);
   let mut leds = Leds::new(gpioe);
   let mut delay = Delay::new(cortex_peripherals.SYST, clocks);

   loop {
       leds[0].on();
       delay.delay ms(500_u16);
       leds[0].off();
       delay.delay_ms(500_u16);
   }
}
```

Write Code - Imports



```rust
extern crate cortex_m;
extern crate f3;
extern crate panic_abort;

use f3::hal::stm32f30x;
use f3::hal::prelude::*;
use f3::hal::delay::Delay;
use f3::led::Leds;
```

Write Code - Initialization



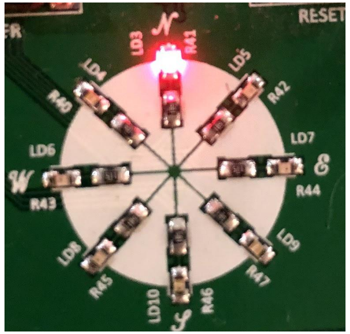
```
let cortex peripherals = cortex m::Peripherals::take().unwrap() ;
let stm_peripherals = stm32f30x::Peripherals::take().unwrap() ;

let mut flash = stm peripherals.FLASH.constrain() ;
let mut rcc = stm peripherals.RCC.constrain() ;
let gpioe = stm_peripherals.GPIOE.split(& mut rcc.ahb);

let clocks = rcc.cfgr.freeze(& mut flash.acr);
let mut delay = Delay::new(cortex_peripherals.SYST , clocks);
let mut leds = Leds::new(gpioe) ;
```

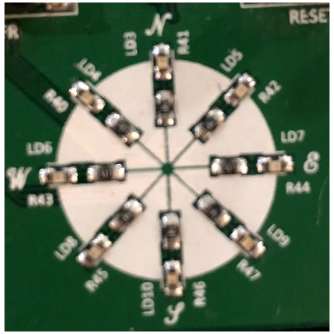
Write Code - Main Loop

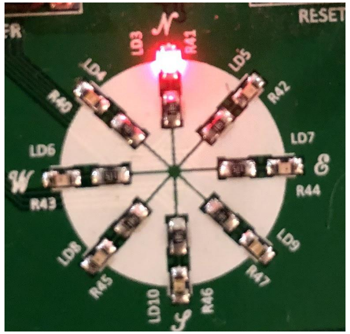
```
loop {
    leds[0].on();
    delay.delay_ms(500_u16);
    leds[0].off();
    delay.delay_ms(500_u16);
}
```

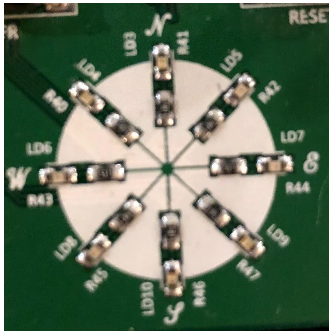
Build, Flash, Debug

- `cargo build`
- `openocd -f interface/stlink.cfg -f target/stm32f3x.cfg`
- `arm-none-eabi-gdb target/thumbv7em-none-eabihf/debug/blinky`
  - `target remote :3333`
  - `load`
  - `continue`



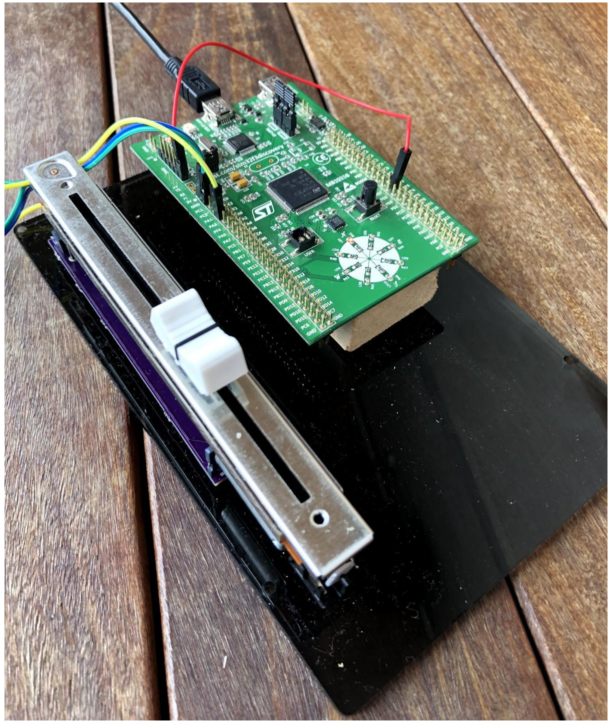






Hardware Platform - Part 2

- STM32F3DISCOVERY
- 100mm Slide Pot
- Let’s read a value and print it

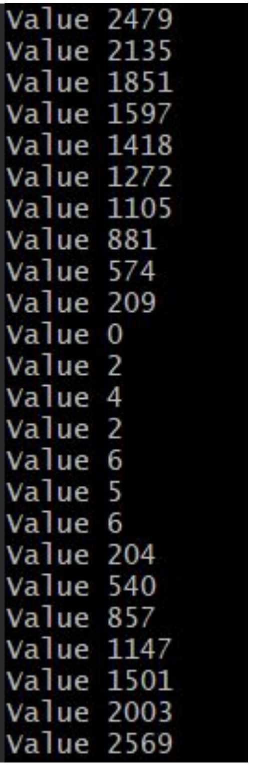


Setting a Register

```
let adc1 = stm_peripherals.ADC1;

adc1.cfgr.modify(|_, w| {
    w.align().clear_bit(); // Right data alignment
    w.cont().clear_bit(); // Single conversion mode
    w.ovrmod().set_bit() // Overwrite
});
```

Reading a Register

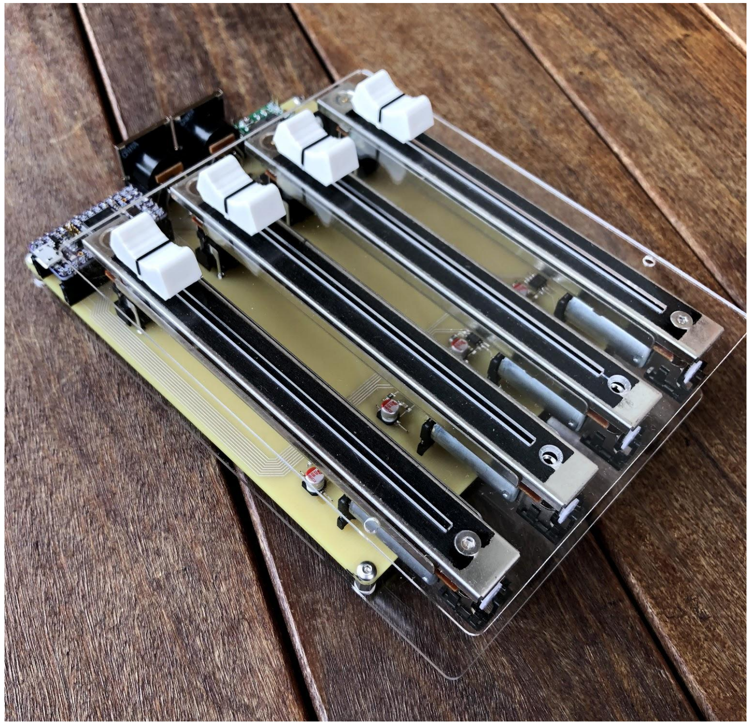
```rust
let mut itm = cortex_peripherals.ITM;
loop {
   adc1.cr.modify(|_, w| w.adstart().set_bit());
   while !adc1.isr.read().eos().bit() {}
   value = adc1.dr.read().regular_data().bits();
   adc1.isr.modify(|_, w| {
       w.eoc().clear_bit();
       w.eos().clear_bit()
   });
   iprintln!(&mut itm.stim[0], "Value {}", value)
   delay.delay_ms(500_u16);
}
```



```
Value 2479
Value 2135
Value 1851
Value 1597
Value 1418
Value 1272
Value 1105
Value 881
Value 574
Value 209
Value 0
Value 2
Value 4
Value 2
Value 6
Value 5
Value 6
Value 204
Value 540
Value 857
Value 1147
Value 1501
Value 2003
Value 2569
```

Hardware Platform - Part 3

- 1Bitsy
- MIDI In & Out
- 4 Motorized Faders



Why Not Rust?

- Still unstable, breaking changes still occur occasionally
- Examples are limited. What is available is often outdated because of the above
- Ecosystem is young and leans heavily towards Cortex-M, and then towards STM32

What to look forward to

- Embedded hitting Rust stable this year
- AVR support soon (already in LLVM trunk)
- Documentation is growing

Thank You

Jacob Creedon

@jacobcreedon
jcreedon@gmail.com

Questions?