CEC Continuing Education Center

DesignNews

Writing Microcontroller Drivers in Rust

DAY 2 : The Peripheral Access Crate

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.

CEC Continuing Education Center

BENINGO EMBEDDED GROUP

DigiKey

THE SPEAKER

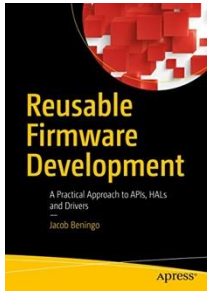
Jacob Beningo

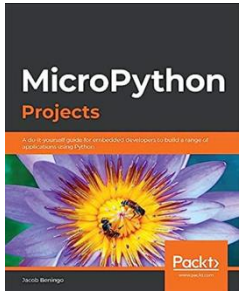
Jacob@beningo.com

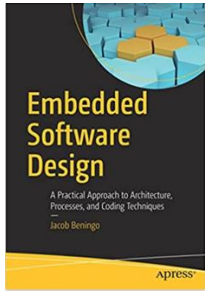
Beningo Embedded Group – CEO / Founder

Focus: Embedded Software Consulting and Training

Help teams deliver higher-quality embedded software faster. We specialize in creating and promoting embedded software excellence in businesses around the world.







Blogs for:

- DesignNews.com
- Embedded.com
- EmbeddedRelated.com
- MLRelated.com

Visit **www.beningo.com** to learn more

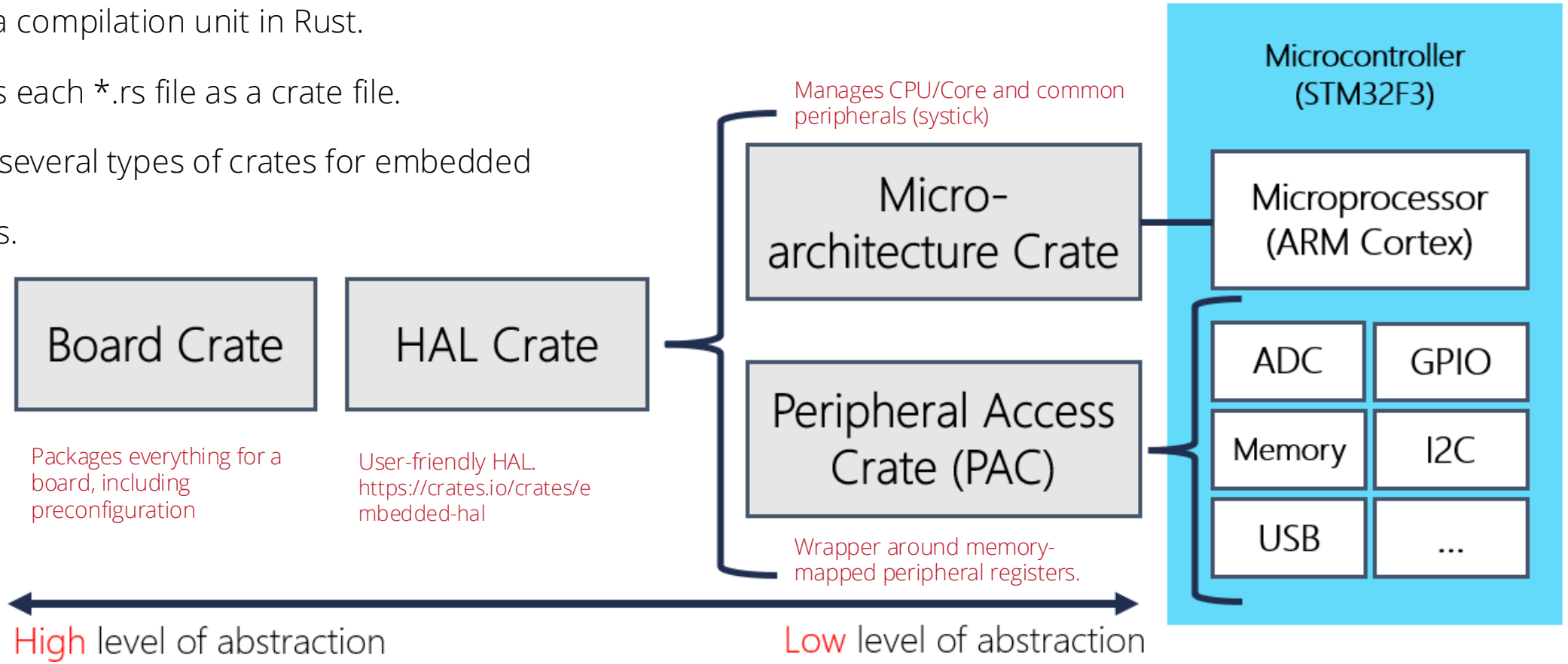
The Peripheral Access Crate (PAC)

01

Crates

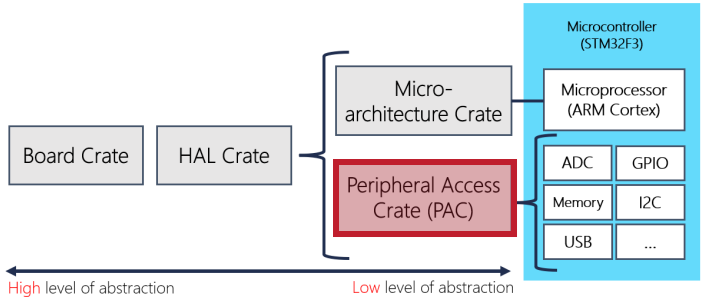
Overview

- A crate is a compilation unit in Rust.
- Rust treats each *.rs file as a crate file.
- There are several types of crates for embedded developers.



The Peripheral Access Crate (PAC)

Defined



What is the PAC?

- PAC stands for Peripheral Access Crate.
- Provides direct, low-level access to a microcontroller's peripherals.
- Auto-generated from the microcontroller's SVD (System View Description) files, ensuring accuracy and completeness.

Key Features

- Type Safety: Utilizes Rust's type system to prevent common bugs (e.g., invalid register access).
- Memory Safety: Ensures safe access to peripheral registers, mitigating risks of memory corruption.
- Concurrency Safety: Facilitates safe sharing of peripherals between tasks in concurrent environments.

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

use panic_halt as _; // panic handler

use cortex_m_rt::entry;
use tm4c123x;

#[entry]
pub fn init() -> (Delay, Leds) {
    let cp = cortex_m::Peripherals::take().unwrap();
    let p = tm4c123x::Peripherals::take().unwrap();

    let pwm = p.PWM0;
    pwm.ctl.write(|w| w.globalsync0().clear_bit());
    // Mode = 1 => Count up/down mode
    pwm._2_ctl.write(|w| w.enable().set_bit().mode().set_bit());
    pwm._2_gena.write(|w| w.actcmpau().zero().actcmpad().one());
    // 528 cycles (264 up and down) = 4 loops per video line (2112 cycles)
    pwm._2_load.write(|w| unsafe { w.load().bits(263) });
    pwm._2_cmpa.write(|w| unsafe { w.compa().bits(64) });
    pwm.enable.write(|w| w.pwm4en().set_bit());
}
```

The Peripheral Access Crate (PAC)

How it works

How PACs Work:

- Each PAC corresponds to a specific microcontroller or family, encapsulating all peripheral definitions.
- Developers interact with hardware registers directly, using strongly-typed Rust structs and enums.
- Provides the foundation for higher-level abstractions, like HALs (Hardware Abstraction Layers).

Benefits:

- Accuracy: Reflects the microcontroller's hardware design accurately, enabling precise control over hardware features.
- Efficiency: Low overhead, direct manipulation of hardware registers without intermediate abstractions.
- Flexibility: Allows for advanced techniques and optimizations specific to the hardware's capabilities.

Use Cases:

- Custom driver development for peripherals not covered by existing libraries.
- High-performance, low-level embedded applications requiring fine-grained hardware control.
- Learning and teaching purposes, providing insights into microcontroller architecture and peripheral programming.

Audience POLL Question

Which of the following is a key feature of the PAC?

a) Type safety
b) Memory safety
c) Concurrency safety
d) All the above
e) None of the above

Introducing svd2rust

02

The Peripheral Access Crate (PAC)

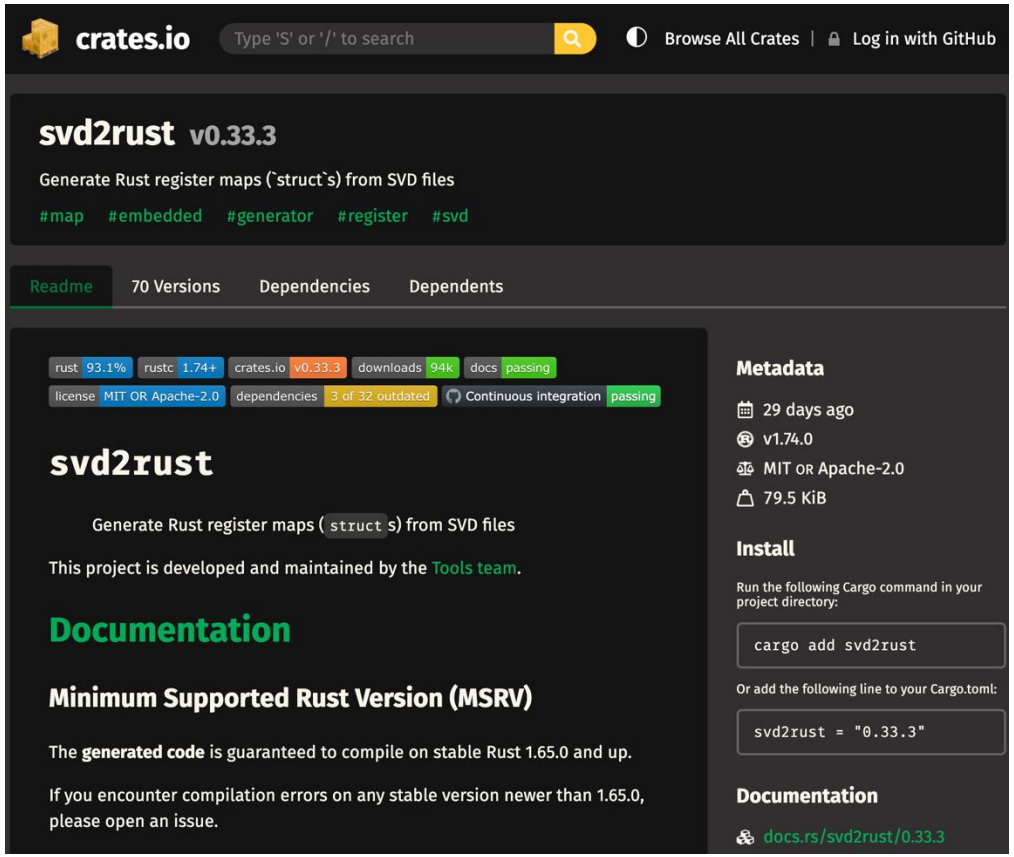
Peripheral Access Crates

svd2rust is a command-line tool in the Rust embedded ecosystem designed to generate Rust Peripheral Access Crates (PACs) from System View Description (SVD) files.

SVD files are XML documents that describe the hardware peripherals of a microcontroller, including registers and their bitfields, in a standardized format.

Key Features:

- Code generation
- Documentation
- Zero-Run-Time Cost Abstractions

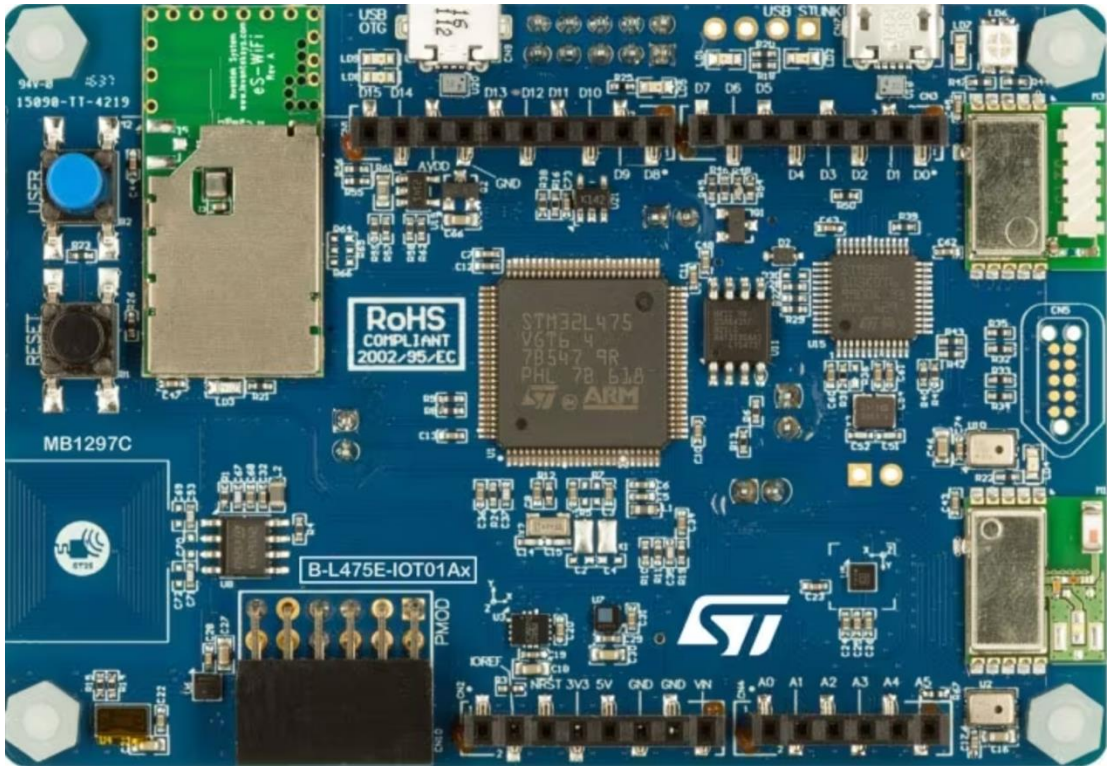


Session Goals

The STM32L475 IoT Discovery Board

A feature-rich development tool designed for IoT applications, leveraging the power of the STM32L475VGT6 microcontroller.

- Equipped with an ARM Cortex-M4 core that operates at up to 80 MHz with 1 MB of Flash memory and 128 KB of SRAM.
- Includes modules for Bluetooth® Low Energy (BLE), Sub-GHz RF, Wi-Fi, and a dynamic NFC-tag with a printed antenna
- A comprehensive collection of built-in sensors for motion, gesture, and environmental sensing, including a MEMS accelerometer, gyroscope, magnetometer, barometric pressure sensor, temperature/humidity sensor, and more.

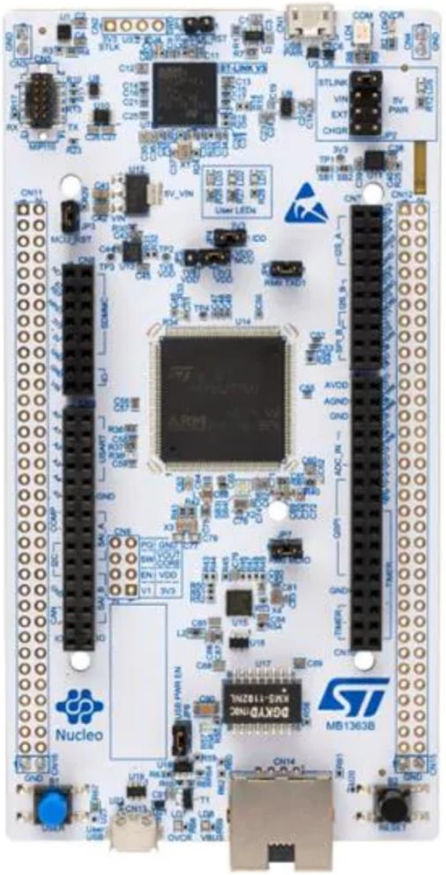


Creating a PAC with svd2rust

The STM32U575 Nucleo Board

A feature-rich development tool designed for IoT applications, leveraging the power of the STM32U575ZI-Q microcontroller.

- Equipped with an ARM Cortex-M33 core that operates at up to 160 MHz with 2 MB of Flash memory and 784 KB of SRAM.
- Includes Red, Green, and Blue user LEDs, USB, Arduino Headers, user push button, and more.



Audience POLL Question

Will you be trying to generate your own PAC?

a) Yes
b) No

CEC Continuing Education Center

BENINGO EMBEDDED GROUP

DigiKey

03 Creating a PAC with svd2rust

Creating a PAC with svd2rust

Running svd2rust

Update Rust:

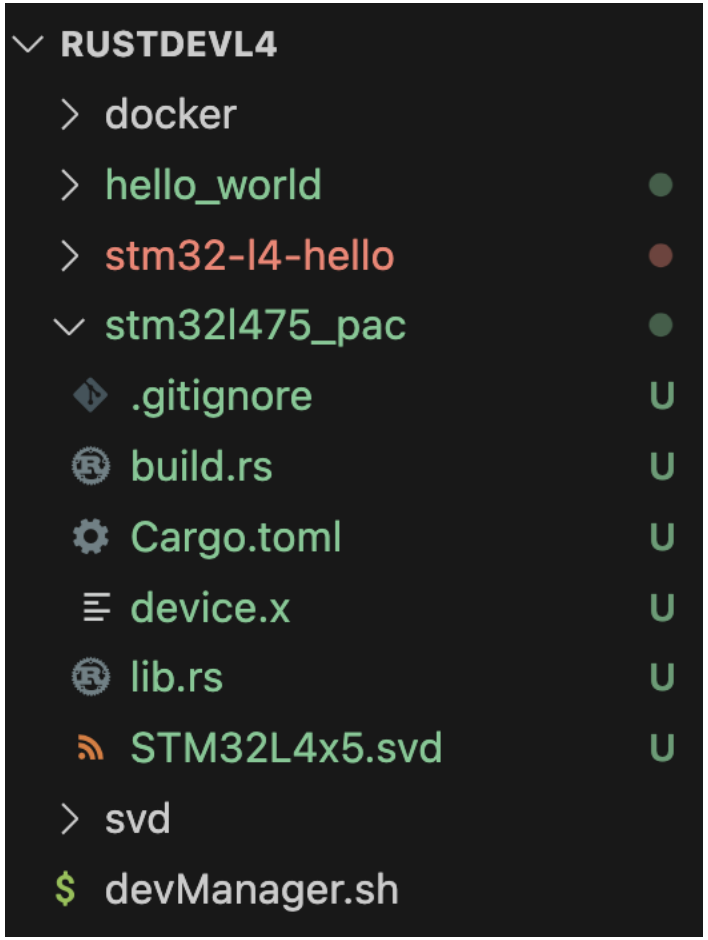
```
rustup update
```

Install the tool:

```
cargo install svd2rust
```

Create a new library project:

```
cargo new stm32u575_pac --lib
```



Creating a PAC with svd2rust

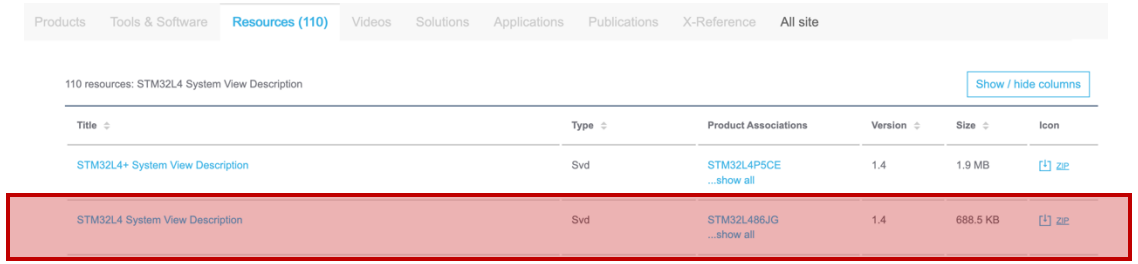
Getting your SVD file

- SVD Files are available from Silicon Vendor
- Place the SVD into your project directory
- Run the tool

    svd2rust –i <device>.svd

```
root@5c09a94cbcc6:/home/app/stm32l475_pac# svd2rust -i STM32L4x5.svd
[INFO  svd2rust] Parsing device from SVD file
[INFO  svd2rust] Rendering device
root@5c09a94cbcc6:/home/app/stm32l475_pac#
```

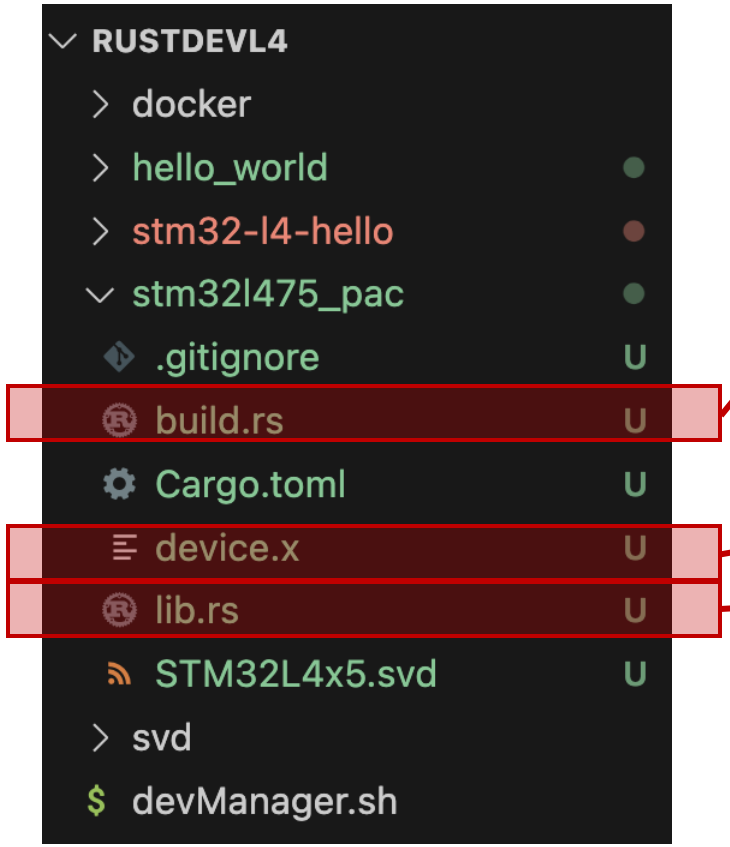
STM32L4x5.SVD Download



https://www.st.com/content/ccc/resource/technical/ecad_models_and_symbols/svd/group0/9c/fe/a9/98/0f/5a/42/b6/stm32l4_svd.zip/files/stm32l4_svd.zip/jcr:content/translations/en.stm32l4_svd.zip

Creating a PAC with svd2rust

Understanding svd2rust output



Build script

- Conditional compilation
- Device.x handling
- Linker configuration
- Build dependency tracking

Linker fragment for interrupts

Peripheral Implementation

- Not “clean”, or easily readable

Creating a PAC with svd2rust

Separate and Format Modules

Install form (if you don't have it already)

```
cargo install form
```

Remove src

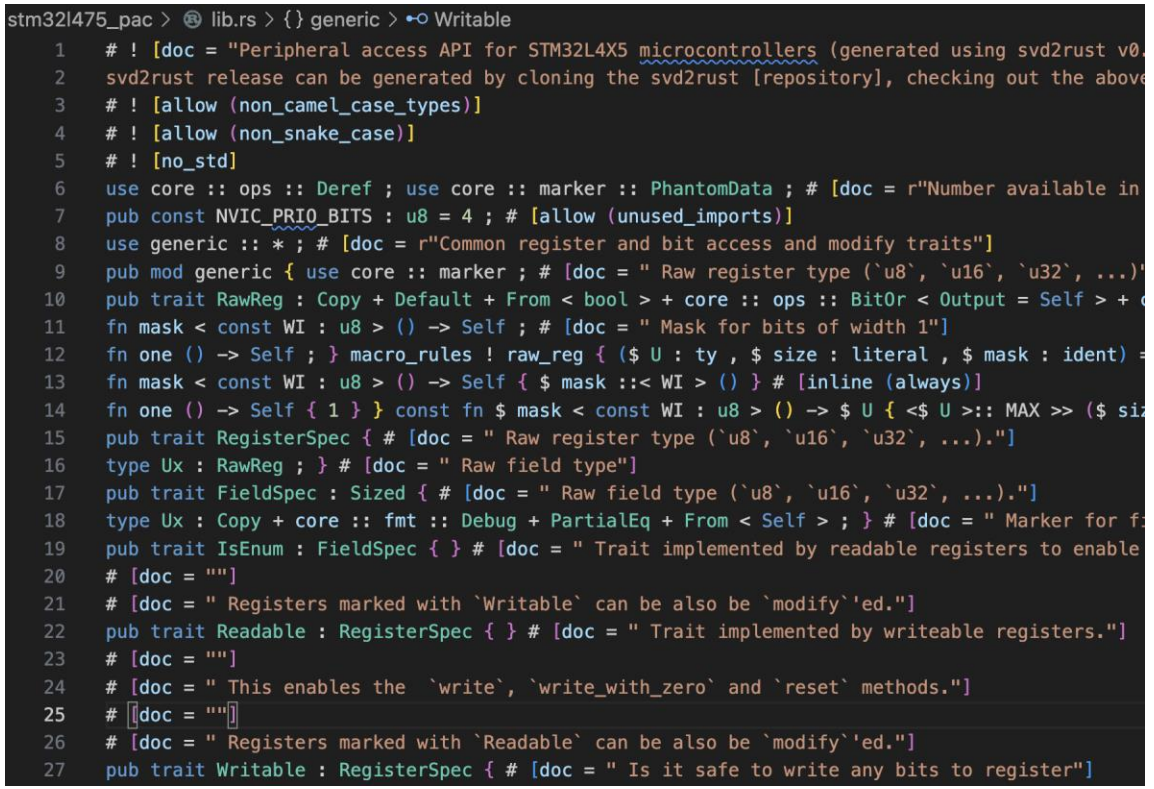
```
rm -rf src
```

Split lib.rs into separate modules

```
form -i lib.rs -o src/ && rm lib.rs
```

Format to be human readable and "clean"

```
cargo fmt
```



Creating a PAC with svd2rust

Add dependencies to the toml

Add to TOML file:

```
[dependencies]
critical-section = { version = "1.1.2", optional = true }
cortex-m = "0.7.7"
cortex-m-rt = { version = "0.7.3", optional = true }
vcell = "0.1.3"

[features]
rt = ["cortex-m-rt/device"]
```

- Provides a mechanism for creating critical sections within your code, allowing you to prevent race conditions and ensure data consistency when accessing shared resources from multiple contexts, such as interrupt service routines (ISRs) and the main program.
- Offers low-level access to core ARM Cortex-M functionality, such as registers and instructions specific to these microcontrollers
- Provides runtime support for ARM Cortex-M microcontrollers, including startup code, linker scripts, and definitions for the interrupt vector table.
- Offers volatile cell types, enabling safe read and write operations to memory-mapped peripheral registers.
- This feature flag is typically used to conditionally compile parts of the application that require runtime support, such as startup code and interrupt handling.

Check Rust Documentation for latest version #'s

Creating a PAC with svd2rust

Build the crate

Build the crate:

```
cargo build -r
```

```
(base) beningo@Jacobs-MacBook-Pro stm32u575_pac % cargo build -r
   Compiling proc-macro2 v1.0.85
   Compiling semver-parser v0.7.0
   Compiling unicode-ident v1.0.12
   Compiling syn v1.0.109
   Compiling cortex-m-rt v0.7.3
   Compiling cortex-m v0.7.7
   Compiling nb v1.1.0
   Compiling vcell v0.1.3
   Compiling void v1.0.2
   Compiling bitfield v0.13.2
   Compiling stm32U575_pac v0.1.0 (/Users/beningo/Projects/03-Rust/svd2rustU5/stm32u575_pac)
   Compiling volatile-register v0.2.2
   Compiling nb v0.1.3
   Compiling embedded-hal v0.2.7
   Compiling semver v0.9.0
   Compiling rustc_version v0.2.3
   Compiling bare-metal v0.2.5
   Compiling quote v1.0.36
   Compiling cortex-m-rt-macros v0.7.0
    Finished release [optimized] target(s) in 23.50s
(base) beningo@Jacobs-MacBook-Pro stm32u575_pac %
```

Audience POLL Question

What is the device.x file?

a) Build script
b) Linker fragment
c) Peripheral implementation
d) None of the above

Next Steps

04

Embedded Rust Docker Container

- https://mailchi.mp/beningo/embedded_rust_docker_container
  - Rust Toolchain
  - Embedded Tools

Beningo Rust Docker Container

Additional Resources

Please consider the resources below:

- Jacob's Blogs
- Jacob's CEC courses
- Embedded Software Academy

- Embedded Bytes Newsletter
  - http://bit.ly/1BAHYXm

Consulting | Coaching | Training

www.beningo.com

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets