

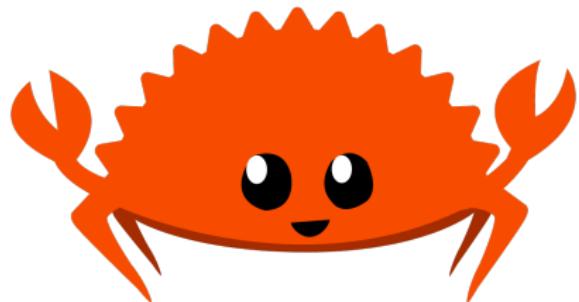
RustIEC - Rust ecosysteem

RustIEC team

November 24, 2022

OVERVIEW

1. Rust in general
2. Rust for web
3. Rust for embedded



Rust ecosystem: general remarks

CARGO

Cargo is your friend!



Cargo is your friend!



`Cargo.toml:`

```
1 [package]
2 name = "whisperfish"
3 version = "0.6.0-dev"
4 authors = ["Ruben De Smet
    ↵ <ruben.de.smet@rubdos.be>"]
5 description = "Private messaging
    ↵ ..."
6
7 [dependencies]
8 async-trait = "0.1"
9 bincode = "1.2.1"
10 actix = "0.12"
```

Cargo is your **new best friend**:

- ▶ <https://crates.io>
(e.g., crates.io/crates/ecdsa)

The screenshot shows the crates.io website interface. At the top, there's a search bar with placeholder text "Press ⌘ to focus this search...", a "Browse All Crates" link, and a "Log in with GitHub" button. Below the header, the title "ecdsa v0.14.8" is displayed, along with a brief description: "Pure Rust implementation of the Elliptic Curve Digital Signature Algorithm (ECDSA) as specified in FIPS 186-4 (Digital Signature Standard), providing RFC6979 deterministic signatures as well as support for added entropy". A "crypto" dependency badge is present. Below the title, there are tabs for "Readme", "58 Versions", "Dependencies", and "Dependents".

The main content area features a section titled "RustCrypto: ECDSA" with a summary: "Elliptic Curve (Digital Signature Algorithm) (ECDSA) as specified in FIPS 186-4 (Digital Signature Standard)." It includes a "Documentation" link and an "About" section. The "About" section details the crate's purpose: "This crate provides generic ECDSA support which can be used in the following ways:

- Generic implementation of ECDSA usable with the following crates:
 - `sign` (RFC6979)
 - `sign` (NIST P-256)
 - `sign` (NIST P-384)
- Other crates which provide their own complete implementations of ECDSA can also leverage the types from this crate to export ECDSA functionality in a generic, interoperable way by leveraging `sign::Signature` with the `signatures::Signer` and `signatures::Verifier` traits.

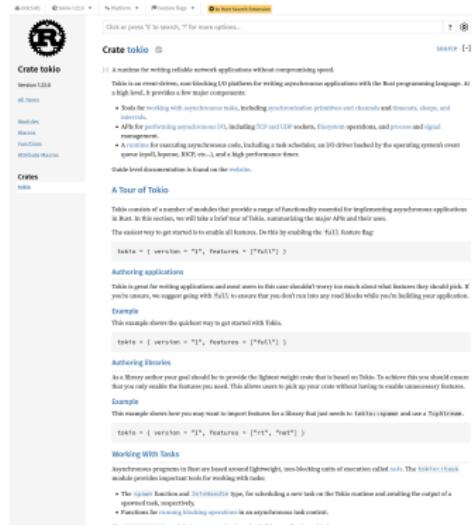
"

A "Security Warning" section notes: "The ECDSA implementation contained in this crate has never been independently audited for security." A note at the bottom states: "This crate contains a generic implementation of ECDSA which must be".

On the right side of the page, there are sections for "Metadata" (with links to the crate's GitHub repository and license information), "Install" (with a command-line example), "Documentation" (link to the crate's documentation), "Repository" (link to the GitHub repository), "Owners" (links to profiles for Tony Ansari and RustCrypto/signatures), and "Categories" (links to Cryptography and No standard library). A "Feedback" button is located at the bottom right.

Cargo is your **new best friend**:

- ▶ <https://crates.io>
(e.g., crates.io/crates/ecdsa)
- ▶ <https://docs.rs>
(e.g., docs.rs/tokio)



Cargo is your **new best friend**:

- ▶ <https://crates.io>
(e.g., crates.io/crates/ecdsa)
- ▶ <https://docs.rs>
(e.g., docs.rs/tokio)
- ▶ <https://lib.rs>^a
(e.g., lib.rs/crates/actix)

^aunofficial

The screenshot shows the official Actix website. At the top, it says "Actix" and "Actor framework for Rust". Below that, it says "Actix is a Actor framework for Rust. by Nikolay Klim, Petr Lhotáček, tokio-rs authors and over 100 contributors. Co-owned by Actix." It features a navigation bar with links for "Home", "About", "References", "Status (beta)", and "www (beta)".

The main content area includes:

- Statistics:** 51 releases, 232K+ Mar 1, 2020, 632K+ Jun 8, 2021, 810K+ Mar 23, 2021, 832K+ Sep 11, 2019, 63.3 Nov 21, 2019.
 - RDF in Asynchronous**: 23000 54.5M+ Used in 199 crates (21 direct).
 - MFTI Apache**: 23000 54.5M+ Used in 199 crates (21 direct).
- Actix**: Actor framework for Rust.
- Documentation**: User Guide, API Documentation, API Documentation (master branch).
- Features**: Actor and sync actors, Actor communication in a local/thread context, Uses futures for asynchronous message handling, Actor supervisor, Send messages to any type!, Run on stable Rust 1.6+.
- Usage**: To use actix, add this to your Cargo.toml:

```
[dependencies]
actix = "0.10.2"
```
- Initialize Actix**: In order to use actix you first need to create a System.

```
fn main() {
    let system = actix::System::new();
```

Cargo is your **new best friend**:

- ▶ <https://crates.io>
(e.g., crates.io/crates/ecdsa)
- ▶ <https://docs.rs>
(e.g., docs.rs/tokio)
- ▶ <https://lib.rs>^a
(e.g., lib.rs/crates/actix)

^aunofficial

The screenshot shows the official Actix website. At the top, it says "Actix" and "Actor framework for Rust". Below that, it says "by Nikolay Klim, Petr Lhotáček, Lukáš Černohorský and over 100 contributors. Co-owned by Actix." It features a navigation bar with links for "Home", "API reference", "Status (beta)", and "www (beta)".

The main content area includes:

- Metrics:** 51 releases, 232K+ Mar 1, 2023, 632K+ Jun 8, 2021, 810K+ Mar 23, 2021, 810K+ Sep 11, 2019, 632K+ Sep 21, 2017. It also mentions 100.0% downloadable pre-releases and being Used in 199 crates (21 direct).
- Actix:** Actor framework for Rust.
- Documentation:** User Guide, API Documentation, API Documentation (master branch).
- Features:** Actor and sync actors, Actor communication in a local/thread context, Uses futures for asynchronous message handling, Actor supervisor, Send messages to any type!, Run on stable Rust 1.6+.
- Usage:** To use actix, add this to your Cargo.toml:

```
[dependencies]
actix = "0.10.2"
```
- Initialize Actix:** In order to use actix you first need to create a `System`.

```
fn main() {
    let system = actix::System::new();
```

Rust search extension <https://rust.extension.sh/>

INTEGRATED DEVELOPMENT ENVIRONMENTS

```
use crate::SqlPeError;
use argon2::{Argon2, ParamsBuilder};
use rand::Rng, RngCore;
use serde::{Deserialize, Serialize};

use super::*;

#[derive(Clone, Serialize, Deserialize, Debug)]
pub struct PasswordAuthenticationMethod {
    salt: Vec<u8>,
    version: u32,
    m_cost: u32,
    t_cost: u32,
    p_cost: u32,
    encrypted_secret_encryption_key: Vec<u8>,
}

impl RoleManager {
    pub fn authenticate_by_password(
        &mut self,
        name: &str,
        password: impl AsRef<[u8]>,
    ) -> Result<&AuthenticatedRole, SqlPeError> {
        if self.is_authenticated(name) {
```

Lapce (<https://lapce.dev/>)

INTEGRATED DEVELOPMENT ENVIRONMENTS

```
15         })
14     });
13     sender.send_contact_details(&local_addr, None, contacts, false, true).await?
12     t?;
11 },
10 Type::Groups => {
9   use libsignal_service::sender::GroupDetails;
8   let sessions = storage.fetch_group_sessions();
7
6   let groups = sessions.into_iter().map(|session| {
5     let group = session.unwrap_group_v1();
4     let members = storage.fetch_group_members_by_group_v1_id(&group.id);
3     GroupDetails {
2       name: Some(group.name.clone()),
1       members_e164: members.iter().filter_map(|(member, recipient)| recipient.e164.clone()).collect(),
618     // XXX: update proto file and add more.
1     // members: members.iter().filter_map(|(member, recipient)| Member{ r { e164: recipient.e164}.collect()},
2       // avatar, active?, color, ..., many cool things to add here!
3       // Tagging issue #264
4       ..Default::default()
5     });
6   });
8     sender.send_groups_details(&local_addr, None, groups, false).await?;
9   }
10 Type::Blocked => {
11   anyhow::bail!("Unimplemented {:?}", req.r#type());
12 }
```

whisperfish/src/worker/client.rs 618,25 31% whisperfish/Cargo.toml 29,2 10%

```
13 [dependencies]
12 async-trait = "0.1"  ◊ 0.1.58
11 bincode = "1.2.1"  ◊ 1.3.3
10 actix = "0.12"  ◊ 0.12.0  ◊ 0.13.0
9 actix-web = "0.12"  ◊ 0.7.3  ◊ 0.8.5
8 parking_lot = "0.12"  ◊ 0.12.1
7 uuid = "version = \"1\", features=[\"v4\"]"  ◊ 1.2.2
6 nime = "0.3.16"  ◊ 0.3.18
5 nime_guess = "2.8"  ◊ 2.8.4
4 nime_classifier = "0.0.1"  ◊ 0.0.1
3 chrono = "0.4"  ◊ 0.4.23
2 tokio = { version="1.17.0", features=["time", "io-util", "net", "sync"] },  ◊ 1.22.0
1 futures = "0.3"  ◊ 0.3.25
29 tokio.extra = "1.2.8"  ◊ 1.2.8
1
2 # 1.9.1 requires Rust 2021
3 indexmap = ">=1.8.1"  ◊ 1.8.1  ◊ 1.9.2
4
5 # 3.0.0-rc.1 requires Rust 1.54
6 actix-httplc = "3.0.0-rc.1"  ◊ 3.0.0-beta.19  ◊ 3.2.2
7
8 libsignal-service = { git = "https://github.com/Whisperfish/libsignal-service-rs", branch = "master" }
9 libsignal-service-actix = { git = "https://github.com/Whisperfish/libsignal-service-actix", branch = "master" }
10
11 libsignal-protocol = { git = "https://github.com/Signalapp/libsignal", tag = "v0.21.1" }
12 zkgroup = { git = "https://github.com/Signalapp/libsignal", tag = "v0.21.1" }
```

whisperfish | check computing...

```
Compiling fastrand v1.8.0
Compiling remove_dir_all v0.5.3
Compiling fixedbitset v0.4.2
Compiling petgraph v0.6.2
Compiling tempfile v3.3.0
error: could not compile `tokio`
```

Caused by:
process didn't exit successfully: 'rustc --crate-name tokio --edition=2018 /home/rsmet/.cargo/registry/src/github.com-1eccb299db9ec823/tokio-1.20.1/src/lib.rs --error-format=json'
Hit q to quit, w to wrap lines, ? for help

VIM (bacon and coc.nvim)

INTEGRATED DEVELOPMENT ENVIRONMENTS

The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar with a tree view of project files. The main area displays a code editor with several tabs open at the top: `ipv6.rs`, `ipv6routing.rs`, `sixlowpan.rs` (the active tab), `ipv6option.rs`, and `mod.rs`. The code in the editor is for the `sixlowpan.rs` file, specifically the `dispatch_ieee802154` function. The code handles packet fragmentation and transmission. A tooltip is visible in the editor area, stating: "You, last month * Split interface into multiple files". The code editor has syntax highlighting for Rust.

```
// The packet does not fit in one IEEE802154 frame, so we need fragmentation.  
// We do this by emitting everything into the 'out_packet.buffer' from the interface.  
// After emitting everything into that buffer, we send the first fragment here.  
// When 'poll' is called again, we check if out_packet was fully sent, otherwise we  
// call 'dispatch_ieee802154_out_packet', which will transmit the other fragments.  
  
// 'dispatch_ieee802154_out_packet' requires some information about the total packet size,  
// the link local source and destination address...  
let SixlowpanOutPacket {  
    buffer: &mut ManagedSlice<u8>,  
    packet_len: &mut usize,  
    datagram_size: &mut u16,  
    datagram_tag: &mut u16,  
    sent_bytes: &mut usize,  
    fragn_size: &mut usize,  
    ll_dst_addr: &mut Address,  
    ll_src_addr: &mut Address,  
    datagram_offset: &mut usize,  
    ..  
} = &mut _out_packet.unwrap().sixlowpan_out_packet;  
  
if buffer.len() < total_size {  
    net_debug!("6LoWPAN: Fragmentation buffer is too small");  
    return Err(Error::Exhausted);  
}  
  
*ll_dst_addr = ll_dst_a;  
*ll_src_addr = ll_src_a;  
  
let mut iphc_packet: Packet<&mut [u8]> =  
    SixlowpanIphcPacket::new_unchecked(buffer: &mut buffer[..iphc_repr.buffer_len()]);  
iphc_repr.emit(&mut iphc_packet);  
  
let mut b: &mut [u8] = &mut buffer[iphc_repr.buffer_len() ..];
```

VS Code (<https://code.visualstudio.com/>)

rust.

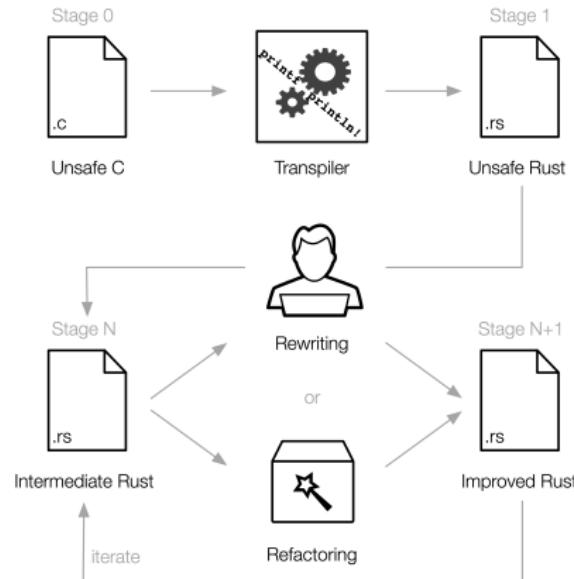
analyzer

General rule: **rust-analyzer**
(<https://github.com/rust-lang/rust-analyzer>)

RUST AND C

"I have C code, and I want it to become Rust code":

- ▶ c2rust (<https://github.com/immunant/c2rust>)



RUST AND C

"I have C code, and I want it to become Rust code":

- ▶ c2rust (<https://github.com/immunant/c2rust>)

"I have Rust code, and I need to call it from C or C++":

- ▶ cbindgen (<https://github.com/eqrion/cbindgen/>)

```
1 #[no_mangle]
2 pub unsafe extern "C" fn sqlpe_spsm_validate_trust(
3     spsm_ptr: *const SqlPrivilegeStateMachine,
4 ) -> c_int {
5     let spsm = spsm_ptr.as_ref().expect("valid spsm");
6     ...
7 }
```

PARALLELISM AND CONCURRENCY

RAYON: PARALLEL ITERATORS

```
1 use rayon::prelude::*;
2 fn sum_of_squares(input: &[i32]) -> i32 {
3     input.par_iter() // <-- just change that!
4         .map(|&i| i * i)
5         .sum()
6 }
```

PARALLELISM AND CONCURRENCY

CROSSBEAM: LOW-LEVEL SYNCHRONISATION PRIMITIVES

```
1 use crossbeam_queue::ArrayQueue;
2
3 let q = ArrayQueue::new(2);
4
5 assert_eq!(q.push('a'), Ok(()));
6 assert_eq!(q.push('b'), Ok(()));
7 assert_eq!(q.push('c'), Err('c'));
8 assert_eq!(q.pop(), Some('a'));
```

PARALLELISM AND CONCURRENCY

ASYNC-AWAIT

```
1  async fn foo() -> MyObject {  
2      // Long running execution...  
3      let bytes = fetch_from_the_internet().await;  
4      decrypt_on_threadpool(bytes).await  
5 }
```

MISCELLANEOUS CRATES

logging log and env_logger

Example:

```
1 log::info!("Starting application v{version}");
2 log::error!("Error connecting to server at {url}!");
```

MISCELLANEOUS CRATES

logging log and env_logger
errors thiserror and anyhow

Example:

```
1 #[derive(thiserror::Error, Debug)]
2 pub enum AwcWebSocketError {
3     #[error("Could not connect")]
4     ConnectionError(#[from]
5         → awc::error::WsClientError),
6     #[error("Websocket connection failed")]
7     ProtocolError(#[from] WsProtocolError),
8 }
```

MISCELLANEOUS CRATES

logging `log` and `env_logger`
errors `thiserror` and `anyhow`
serialization `serde-family`

Example:

```
1 #[derive(Debug, Deserialize, Serialize)]
2 #[serde(rename_all = "camelCase")]
3 pub struct PreKeyEntity {
4     pub key_id: u32,
5     #[serde(with = "serde_base64")]
6     pub public_key: Vec<u8>,
7 }
```

LISTS FOR ECOSYSTEM QUALITY

AREWE*YET.ORG

- ▶ <https://AreWeAsyncYet.rs/>
- ▶ <https://AreWeGameYet.rs/>
- ▶ <https://AreWeWebYet.org/>
- ▶ <https://www.AreWeLearningYet.com>
- ▶ ...

<https://wiki.mozilla.org/Areweyet>

Rust for the Web and Cloud back-end ecosystem: web and cloud

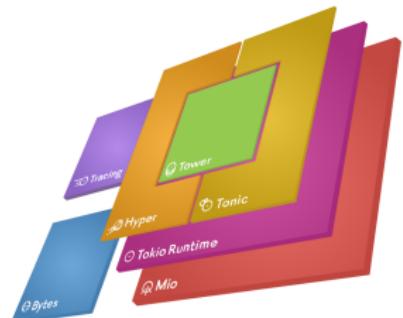
BACK-END ECOSYSTEM

TOKIO

Multi-layer approach:

- ▶ Built on Mio: “Metal IO”

<https://github.com/tokio-rs/mio>



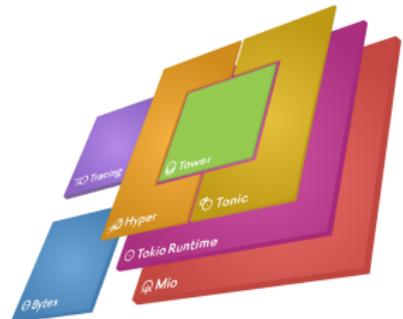
<https://tokio.rs/>

BACK-END ECOSYSTEM

TOKIO

Multi-layer approach:

- ▶ Built on Mio: “Metal IO”
<https://github.com/tokio-rs/mio>
- ▶ Tokio asynchronous **runtime**:
I/O, timer, filesystem,
synchronization, and scheduling



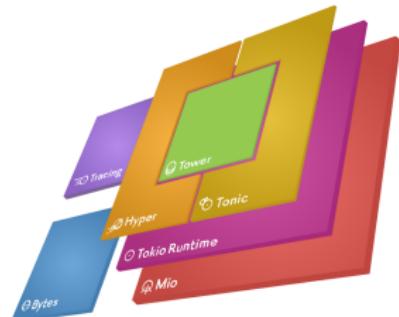
<https://tokio.rs/>

BACK-END ECOSYSTEM

TOKIO

Multi-layer approach:

- ▶ Built on Mio: “Metal IO”
<https://github.com/tokio-rs/mio>
- ▶ Tokio asynchronous **runtime**:
I/O, timer, filesystem,
synchronization, and scheduling
- ▶ HTTP implementations:
 - ▶ Hyper
<https://github.com/hyperium/hyper>
 - ▶ Actix Web
<https://actix.rs>



<https://tokio.rs/>

Rust for the Web and Cloud front-end ecosystem

FRONT-END ECOSYSTEM

EXAMPLE: FLICK.RS

- ▶ wasm-pack
 - <https://github.com/rustwasm/wasm-pack/>
- ▶ React
- ▶ Client-side cryptography in Rust



<https://flickr.s.opencloudedge.be>

FRONT-END ECOSYSTEM

EXAMPLE: FLICK.RS

- ▶ wasm-pack
<https://github.com/rustwasm/wasm-pack/>
- ▶ React
- ▶ Client-side cryptography in Rust

Alternative, 100 % Rust: <https://yew.rs/>



[https://flickr.s.
opencloudedge.be](https://flickr.s.opencloudedge.be)

Rust for embedded programming

1. Install IDE and tools for specific microcontroller;
2. Use Hardware Abstraction Layer (HAL) provided by manufacturer;
3. Write firmware ...

1. Use any preferred IDE and tools¹ for specific microcontroller;
2. Search for HAL on `crates.io / lib.rs / ...`;
Manufactures don't support Rust yet;
3. Write firmware ...

¹Most of the time, cargo can be used for anything

SUPPORTED MICROCONTROLLERS

- Peripheral Access Crates
 - Microchip
 - Nordic
 - NXP
 - Raspberry Pi Silicon
 - SiFive
 - Silicon Labs
 - STMicroelectronics
 - Texas Instruments
 - MSP430
 - Espressif
 - Ambiq Micro
 - GigaDevice
 - XMC
 - Vorago
 - Wiznet
- HAL implementation crates
 - OS
 - Microchip
 - Nordic
 - NXP
 - Raspberry Pi Silicon
 - SiFive
 - STMicroelectronics
 - Texas Instruments
 - MSP430
 - Espressif
 - Silicon Labs
 - XMC
 - GigaDevice
 - Vorago

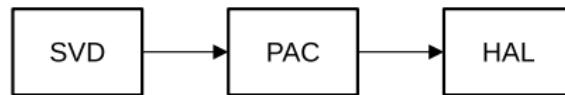
<https://github.com/rust-embedded/awesome-embedded-rust>

PAC OR HAL NOT AVAILABLE?

SVD2RUST

PAC OR HAL NOT AVAILABLE?

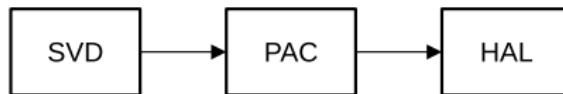
SVD2RUST



SVD: System View Description file

PAC OR HAL NOT AVAILABLE?

SVD2RUST



SVD: System View Description file

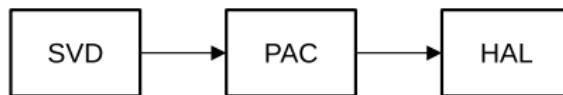
↓ **svd2rust**

PAC: Peripheral Access Crate

```
$ svd2rust -i STM32F30x.svd
$ form -i lib.rs -o src/ && rm lib.rs
$ cargo fmt
```

PAC OR HAL NOT AVAILABLE?

SVD2RUST



SVD: System View Description file

↓ **svd2rust**

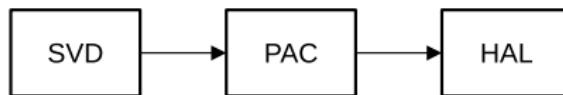
PAC: Peripheral Access Crate

↓ developer

HAL: Hardware Abstraction Layer

PAC OR HAL NOT AVAILABLE?

SVD2RUST



SVD: System View Description file

↓ **svd2rust**

PAC: Peripheral Access Crate

↓ developer

HAL: Hardware Abstraction Layer

Trait² definitions for embedded systems.

²Interfaces in Java or Abstract class in C#

Trait² definitions for embedded systems.

Example for SPI:

```
1 pub trait Spi<Word> {
2     type Error;
3     fn read(&mut self) -> Result<Word, Self::Error>;
4     fn send(&mut self, word: Word) -> Result<(), 
5         Self::Error>;
}
```

²Interfaces in Java or Abstract class in C#

Trait² definitions for embedded systems.

Example for SPI:

```
1 pub trait Spi<Word> {
2     type Error;
3     fn read(&mut self) -> Result<Word, Self::Error>;
4     fn send(&mut self, word: Word) -> Result<(), 
5         Self::Error>;
}
```

Consequences:

- ▶ driver authors can support any number of target platforms;
- ▶ HAL developers unlock all these drivers for their platform.

²Interfaces in Java or Abstract class in C#

PROBE-RS

CARGO REALLY IS YOUR BEST FRIEND!



Flash programming

Downloading a binary to your MCU is easy with probe-rs. We support ELF, ihex and plain binary.



Extensible

probe-rs can be used as a library, giving you even more flexibility



Compatible

ARM, RISC-V, CMSIS-DAP, STLink, JLink probe-rs supports it all. And many more to come!



GDB integration

probe-rs includes a GDB stub to integrate seamlessly into your usual workflow with common tools..



Cargo integration

With cargo-flash you get everything cargo run gives you, but for embedded targets. No compromises.



VSCode integration

We provide a Microsoft DAP server implementation with probe-rs to debug in VSCode.

PROBE-RS

CARGO REALLY IS YOUR BEST FRIEND!



Flash programming

Downloading a binary to your MCU is easy with probe-rs. We support ELF, ihex and plain binary.



Extensible

probe-rs can be used as a library, giving you even more flexibility



Compatible

ARM, RISC-V, CMSIS-DAP, STLink, JLink probe-rs supports it all. And many more to come!



GDB integration

probe-rs includes a GDB stub to integrate seamlessly into your usual workflow with common tools..



Cargo integration

With cargo-flash you get everything cargo run gives you, but for embedded targets. No compromises.



VSCode integration

We provide a Microsoft DAP server implementation with probe-rs to debug in VSCode.

```
cargo flash --release --chip <chip_name>
```

Real-Time Interrupt-driven
Concurrency (RTIC):

- ▶ Currently only Cortex-M

Embassy (Embedded + Async):

- ▶ Cortex-M, RISC-V, Xtensa
architectures

TOCK:

- ▶ Cortex-M, RISC-V, Xtensa
architectures



EMBASSY EXAMPLE

```
1  #[no_std]
2  #[no_main]
3
4  #[embassy_executor::main]
5  async fn main(_spawner: Spawner) {
6      let p = embassy_nrf::init(Default::default());
7      let mut led = Output::new(p.P0_13, Level::Low,
8          → OutputDrive::Standard);
9
10     loop {
11         led.set_high();
12         Timer::after(Duration::from_millis(300)).await;
13         led.set_low();
14         Timer::after(Duration::from_millis(300)).await;
15     }
}
```

DRIVERS

<https://github.com/rust-embedded/awesome-embedded-rust>

1. [AD983x](#) - SPI - AD9833/AD9837 waveform generators / DDS - [Intro blog post](#) - [crates.io v0.3.0](#)
2. [adafruit-alphanum4](#) - I2C - Driver for Adafruit 14-segment LED Alphanumeric Backpack based on the ht16k33 chip - [crates.io v0.1.2](#)
3. [ADS1x1x](#) - I2C - 12/16-bit ADCs like ADS1013, ADS1015, ADS1115, etc. - [Intro blog post](#) - [crates.io v0.2.2](#)
4. [ADXL313](#) - SPI - 3-axis accelerometer - [crates.io v0.2.4](#)
5. [ADXL343](#) - I2C - 3-axis accelerometer - [crates.io v0.8.0](#)
6. [ADXL355](#) - SPI - 3-axis accelerometer - [Intro blog post](#) - [crates.io v0.2.3](#)
7. [AHT20](#) - I2C - Humidity and temperature sensor - [github](#) - [crates.io v0.1.0](#)
8. [AHT20-driver](#) - I2C - Humidity and temperature sensor - [Intro blog post](#) - [github](#) - [crates.io v1.2.0](#)
9. [AnyLeaf](#) - I2C - pH sensor module - [github](#) - [crates.io v1.0.1](#)
10. [AT86RF212](#) - SPI - Low power IEEE 802.15.4-2011 ISM RF Transceiver - [Intro blog post](#) - [crates.io v0.2.0](#)
11. [BlueNRG](#) - SPI - driver for BlueNRG-MS Bluetooth module - [Intro post](#) [crates.io v0.1.0](#)
12. [BNO055](#) - I2C - Bosch Sensortec BNO055 9-axis IMU driver - [Intro post](#) [crates.io v0.3.3](#)
13. [CD74HC4067](#) - GPIO - 16-channel digital and analog multiplexer - [Intro blog post](#) - [github](#) - [crates.io v0.2.0](#)
14. [dht-sensor](#) - 1-Wire - DHT11/DHT22 temperature/humidity sensor driver - [github](#) - [crates.io v0.2.1](#)
15. [DRV8825](#) - DRV8825 Stepper Motor Driver (based on [Stepper](#)) - [Intro blog post](#) - [crates.io v0.6.0](#)
16. [DS1307](#) - I2C - Real-time clock driver - [Intro blog post](#) - [crates.io v0.4.0](#)
17. [EEPROM24x](#) - I2C - 24x series serial EEPROM driver - [Intro blog post](#) - [crates.io v0.5.0](#)
18. [embedded-ccs811](#) - I2C - Gas and VOC sensor driver for monitoring indoor air quality - [Intro blog post](#) - [crates.io v0.2.0](#)

LIBRARIES FOR EMBEDDED

<https://github.com/rust-embedded/awesome-embedded-rust>

1. [adskalman](#): Kalman filter and Rauch-Tung-Striebel smoothing implementation. [crates.io](#) v0.14.0
2. [atomic](#): Generic Atomic wrapper type. [crates.io](#) v0.5.1
3. [bbqueue](#): A SPSC, statically allocatable queue based on BipBuffers suitable for DMA transfers - [crates.io](#) v0.5.1
4. [bitmatch](#): A crate that allows you to match, bind, and pack the individual bits of integers. - [crates.io](#) v0.1.1
5. [biquad](#): A library for creating second order IIR filters for signal processing based on Biquads, where both a Direct Form 1 (DF1) and Direct Form 2 Transposed (DF2T) implementation is available. [crates.io](#) v0.4.2
6. [bit_field](#): manipulating bitfields and bitarrays - [crates.io](#) v0.10.1
7. [bluetooth-hci](#): device-independent Bluetooth Host-Controller Interface implementation. [crates.io](#) v0.1.0
8. [bounded-registers](#) A high-assurance memory-mapped register code generation and interaction library. [bounded-registers](#) provides a Tock-like API for MMIO registers with the addition of type-based bounds checking. - [crates.io](#) v0.1.1
9. [cam-geom](#): Geometric models of cameras for photogrammetry. [crates.io](#) v0.13.0
10. [combine](#): parser combinator library - [crates.io](#) v4.6.6
11. [console-traits](#): Describes a basic text console. Used by [menu](#) and implemented by [vga-framebuffer](#).
[crates.io](#) not found
12. [cmim](#), or Cortex-M Interrupt Move: A crate for Cortex-M devices to move data to interrupt context, without needing a critical section to access the data within an interrupt, and to remove the need for the "mutex dance" - [crates.io](#) v0.2.1
13. [cmsis-dsp-sys](#) : Rust FFI bindings to the [Arm CMSIS_5](#) math library - [crates.io](#) v0.3.1
14. [dcmimu](#): An algorithm for fusing low-cost triaxial MEMS gyroscope and accelerometer measurements
[crates.io](#) v0.2.2

REFERENCES

- ▶ **crates.io**: <https://crates.io>
- ▶ **docs.rs**: <https://docs.rs>
- ▶ **lib.rs**: <https://lib.rs>
- ▶ **Rust Search Extension**: <https://rust.extension.sh>
- ▶ **cargo**: <https://doc.rust-lang.org/stable/cargo/>
- ▶ **Lapce**: <https://lapce.dev>
- ▶ **rust-analyzer**: <https://rust-analyzer.github.io/>
- ▶ **c2rust**: <https://github.com/immunant/c2rust>
- ▶ **cbindgen**: <https://github.com/eqrion/cbindgen>
- ▶ **rayon**: <https://github.com/rayon-rs/rayon>
- ▶ **AreWeYet**: <https://wiki.mozilla.org/Areweyet>

REFERENCES

- ▶ **crossbeam**:
<https://github.com/crossbeam-rs/crossbeam>
- ▶ **envlogger**: https://github.com/rust-cli/env_logger
- ▶ **thiserror**: <https://github.com/dtolnay/thiserror>
- ▶ **anyhow**: <https://github.com/dtolnay/anyhow>
- ▶ **serde**: <https://serde.rs/>
- ▶ **Awesome Embedded Rust**: <https://github.com/rust-embedded/awesome-embedded-rust>
- ▶ **svd2rust**: <https://github.com/rust-embedded/svd2rust>
- ▶ **embedded-hal**:
<https://github.com/rust-embedded/embedded-hal>
- ▶ **probe-rs**: <https://probe.rs/>
- ▶ **Tock**: <https://www.tockos.org/>
- ▶ **RTIC**: <https://rtic.rs/1/book/en/>
- ▶ **Embassy**: <https://embassy.dev/>