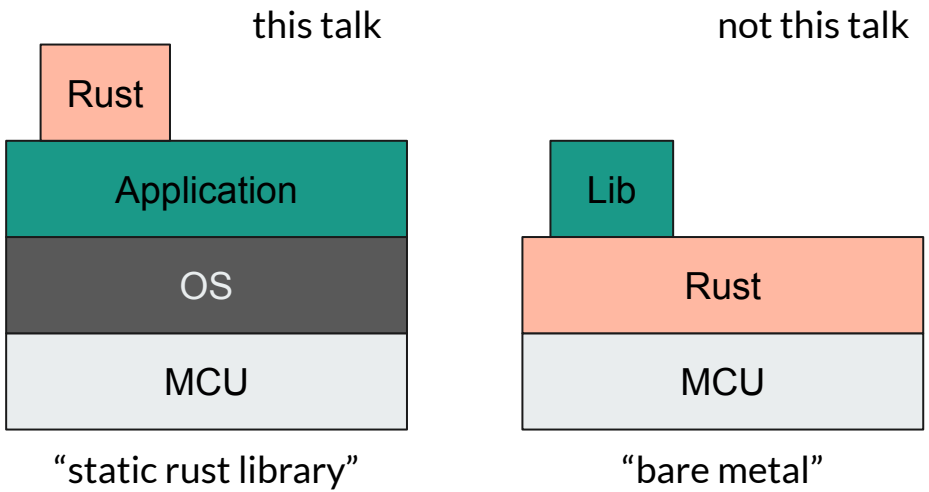
Embedding Rust

Adding Rust to embedded projects

Embedding Rust

- Calling Rust static library
- From existing C-ABI application



Reasoning

There are good reasons why we not always choose “bare metal” Rust:

- Existing project
- Vendor environment (STM-IDE, NRF Connect SDK, ESP-IDF)
- OS (FreeRTOS, Zephyr, etc.)
- Certification (of RF stack)

Reasoning

However we might still want to use Rust for some parts:

- Strict
- Safe
- Efficient
- Crates
- Async

tm1637-gpio-driver v2.0.6

Generic GPIO driver for the TM1637 micro controller, primarily for educational purpose. Doesn't need std-lib and you can choose any GPIO interface/library you want.

Homepage Documentation Repository

All-Time: 2,414
Recent: 86
Updated: about 1 month ago

aht20-driver v1.2.0

Rust embedded-hal driver for the AHT20 temperature and humidity sensor.

Homepage Documentation Repository

All-Time: 383
Recent: 35
Updated: 8 months ago

ws2818-rgb-led-spi-driver v2.0.0

Simple, stripped down, educational, no_std-compatible driver for WS28XX (WS2811/12) RGB LEDs. Uses SPI device for timing/clock, and works definitely on Linux/Raspberry Pi.

Homepage Documentation Repository

All-Time: 2,542
Recent: 178
Updated: over 1 year ago

xca9548a v0.2.1

Platform-agnostic Rust driver for the TCA954xA and PCA954xA I2C switches/multiplexers.

Homepage Documentation Repository

All-Time: 2,343
Recent: 130
Updated: about 2 years ago

embedded-graphics v0.7.1

Embedded graphics library for small hardware displays

Documentation Repository

All-Time: 289,725
Recent: 38,014
Updated: over 1 year ago

stepper-driver v0.1.0

A49xx and DRV88xx stepper motor driver.

Repository

All-Time: 432
Recent: 9
Updated: over 4 years ago

Difficulty

Rust makes it easy!

- Extern: just like a C-style function
- No-mangle: exact symbol names
- #[repr(C)]: memory layout is identical to C

Difficulty

Rust makes it sometimes not so easy…

- Need to check a lot more (but you should've probably checked that regardless)
- Memory ownership is complicated (but it always was)
- Rust can "panic"

Difficulty

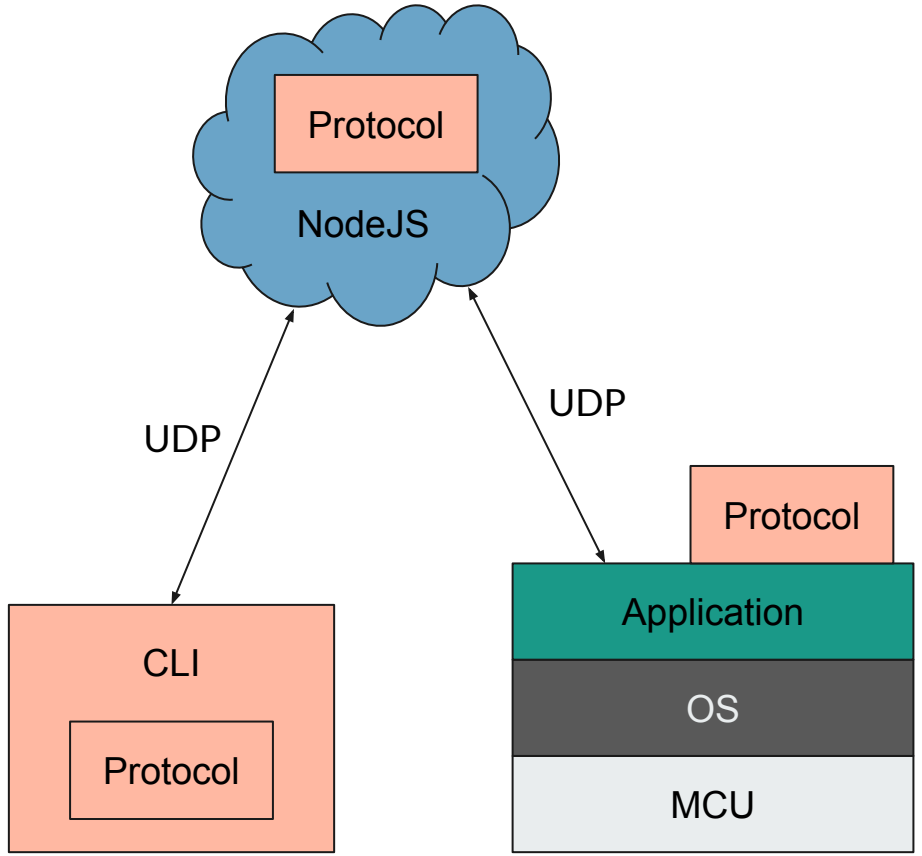
What makes Rust loved is laid bare when integrating into foreign systems (FFI).

Because now you have to do a lot of it manually.

Cases - Protocol

Implementation of byte-packet protocol in Rust.

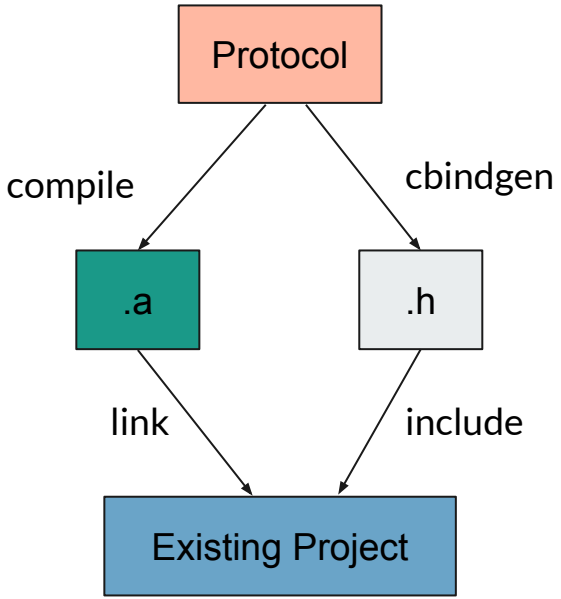
- Embedded in C
- Server in NodeJS
- CLI in Rust



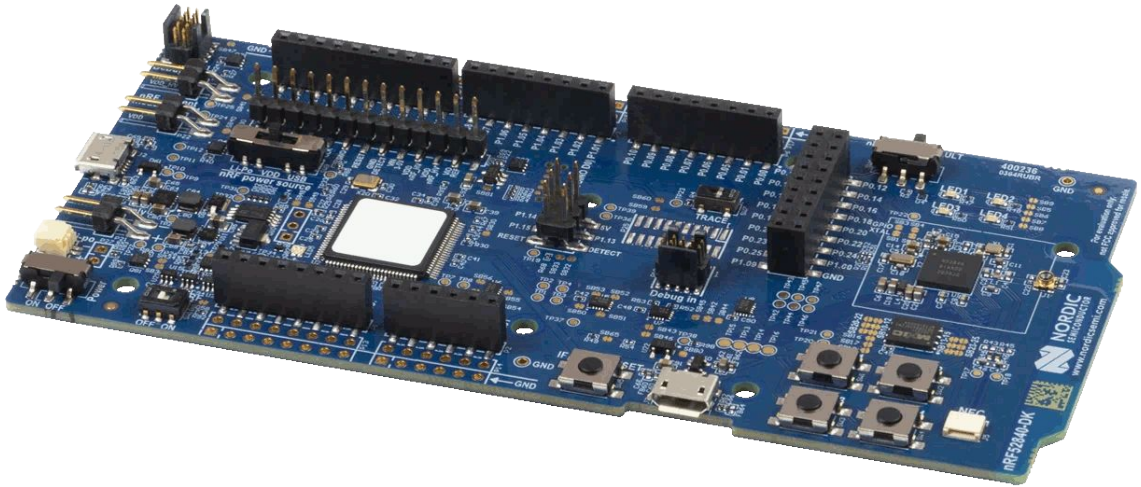
Approach

Presentation of a simplified protocol.

- Define FFI boundary
- Compile static `.a` library
- Run `cbindgen`, get header file
- Include headers & link to library
- Call Rust code!



Cases - Protocol



Cases - Protocol

```rust
#![no_std]

use serde::Serialize;

#[derive(Serialize)]
#[repr(C)]
pub enum Level {
    Info,
    Debug,
}

#[derive(Serialize)]
#[repr(C)]
pub struct Message {
    recipient: Recipient,
    ttl: u16,
    level: Level,
}
```

```c
#include <stdarg.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdlib.h>



typedef enum Level {
  Info,
  Debug,
} Level;

typedef struct Message {
  struct Recipient recipient;
  uint16_t ttl;
  enum Level level;
} Message;
```

```rust
#[repr(C)]
pub struct Recipient {
    // Starting pointer to a C-style string representing the name of the recipient.
    ptr: *const c_uchar,
    /// Length of the C-style string in bytes, including null character at end.
    len: usize,
}

impl Serialize for Recipient {
    fn serialize<S>(&self, serializer: S) -> Result<S::Ok, S::Error>
    where
        S: serde::Serializer,
    {
        let slice = unsafe { core::slice::from_raw_parts(self.ptr, self.len) };
        if let Ok(Ok(s)) = core::ffi::CStr::from_bytes_with_nul(slice).map(|cstr| cstr.to_str()) {
            serializer.serialize_str(s)
        } else {
            serializer.serialize_none()
        }
    }
}
```

Cases - Protocol

```rust
#[no_mangle]
pub extern "C" fn serialize_message(
    msg: *const Message,
    dst_ptr: *mut u8,
    dst_len: usize,
) -> usize {
    let slice = unsafe { core::slice::from_raw_parts_mut(dst_ptr, dst_len) };
    if let Ok(len) = serde_json_core::to_slice(unsafe { &*msg }, slice) {
        len
    } else {
        0
    }
}
```

```c
size_t serialize_message(const struct Message *msg, uint8_t *dst_ptr, size_t dst_len);
```

Cases - Protocol

```toml
[package]
name = "protocol"
version = "0.1.0"
edition = "2021"

[dependencies]
serde = { version = "1.0", features = ["derive"], default-features = false }
serde-json-core = { version = "0.4", default-features = false }

[build-dependencies]
cbindgen = "0.24"

[lib]
crate-type = ["staticlib", "lib"]
```

Cases - Protocol

```
wouter@wgeraedts-sleipnir:~/sarif/embedded-integration/examples/protocol$ cargo build --release
   Compiling protocol v0.1.0 (/home/wouter/sarif/embedded-integration/examples/protocol)
error: `#[panic_handler]` function required, but not found

error: could not compile `protocol` due to previous error
```

```
1 #![no_std]
2
3 extern crate panic_halt;
```

Cases - Protocol

```
wouter@wgeraedts-sleipnir:~/sarif/embedded-integration/examples/protocol$ ls -la ./protocol.h
-rw-rw-r-- 1 wouter wouter 732 nov  4 09:41 ./protocol.h
wouter@wgeraedts-sleipnir:~/sarif/embedded-integration/examples/protocol$ ls -la target/thumbv7em-none-eabi/release/libprotocol.a
-rw-rw-r-- 2 wouter wouter 6708386 nov  4 11:06 target/thumbv7em-none-eabi/release/libprotocol.a
```

Cases - Protocol

```c
#include <zephyr/zephyr.h>
#include <string.h>

#include <protocol.h>

static const size_t BUF_LEN = 128;

void main(void)
{
    char *str = "recipient A";

    Message message = {
        .recipient = {
            .ptr = str,
            .len = strlen(str) + 1,
        },
        .ttl = 5,
        .level = Debug,
    };

    uint8_t buf[BUF_LEN];
    size_t len = serialize_message(&message, buf, BUF_LEN);

    if (len == 0)
    {
        printk("Serialization failed\n");
    }
    else
    {
        buf[len] = 0;
        printk("%s\n", buf);
    }
}
```

Cases - Protocol

```
wouter@wgeraedts-sleipnir: ~/sarif/embedded-integration/examples/integrated/thread-node
wouter@wgeraedts-sleipnir:~/sarif/embedded-integration/examples/integrated/thread-node$ west flash
-- west flash: rebuilding
ninja: no work to do.
-- west flash: using runner nrfjprog
Using board 683958415
-- runners.nrfjprog: Flashing file: /home/wouter/sarif/embedded-integration/examples/integrated/thread-node/build/zephyr/zephyr.hex
[ #################### ]   0.927s | Erase file - Done erasing
[ #################### ]   0.399s | Program file - Done programming
[ #################### ]   0.388s | Verify file - Done verifying
Enabling pin reset.
Applying pin reset.
-- runners.nrfjprog: Board with serial number 683958415 flashed successfully.
wouter@wgeraedts-sleipnir:~/sarif/embedded-integration/examples/integrated/thread-node$
```

```
wouter@wgeraedts-sleipnir: ~
*** Booting Zephyr OS build v3.1.99-ncs1  ***
{"recipient":"recipient A","ttl":5,"level":"Debug"}
```

Conclusions

- Calling Rust from your embedded C is easy.
- But there are some catches and caveats, so be careful.
- Rust can be a great addition to your project.

Further Reading

Interoperability: “a little Rust with your C”:

https://docs.rust-embedded.org/book/interoperability/rust-with-c.html

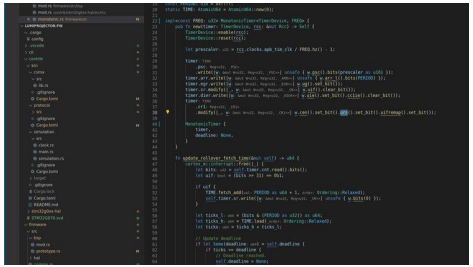
Tips for C developers:

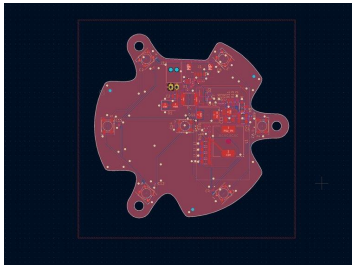
https://docs.rust-embedded.org/book/c-tips/index.html

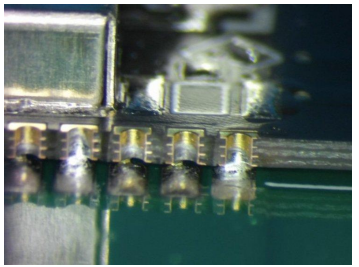
Need help?

Freelance embedded software & hardware engineer

https://sarif.nl







```rust
type TransmitFunction = unsafe extern "C" fn(*const u8, usize) -> usize;

#[repr(C)]
pub struct Parameters {
    transmit: TransmitFunction,
}

pub struct Context {
    parameters: Parameters,
    state: State,
}

#[no_mangle]
pub extern "C" fn context_size() -> usize {
    core::mem::size_of::<Context>()
}

#[no_mangle]
pub extern "C" fn create_context(parameters: Parameters, context: *mut Context) {
    let context: *mut MaybeUninit<Context> = unsafe { core::mem::transmute(context) };
    let context: &mut MaybeUninit<Context> = unsafe { context.as_mut() }.unwrap();

    let state: State = State::default();
    context.write(Context {
        parameters,
        state
    });
}
```