From C to Rust on the ESP32: A Developper's Journey into no_std

Alexis Lothoré

FOSDEM 2026

whoami

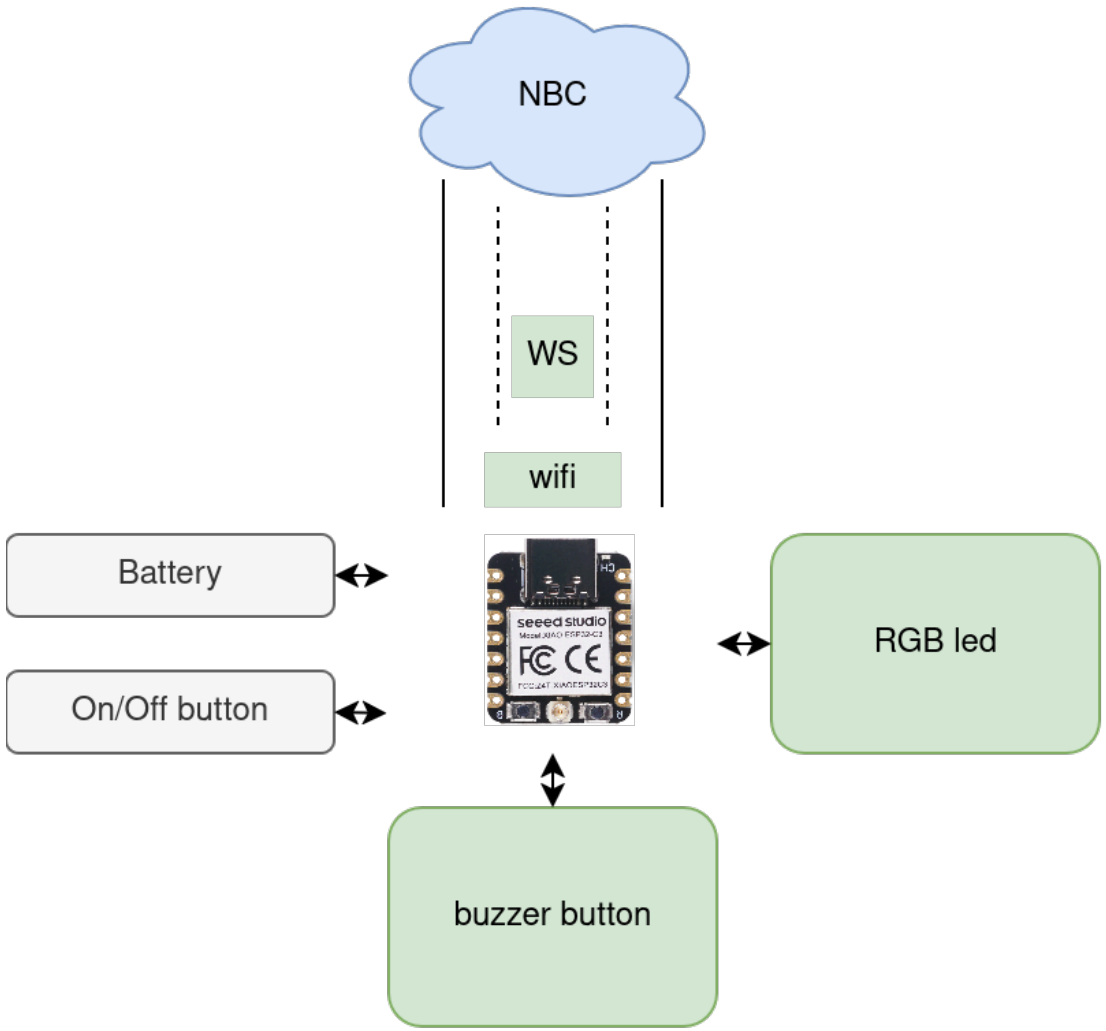
- Alexis Lothoré
- Linux engineer and trainer @ Bootlin during the day
  - Engineering company specialized in **Embedded Linux** and **Zephyr**
  - 28 people, mostly in France
  - Very strong open-source focus
  - We are **hiring**, including **interns**
- Hacker at night
  - electronics
  - (embedded) software
  - CAO/3d printing

The project: Neon Beat Buzzer

Neon Beat

- a custom Blind Test platform
  - each player or team gets a **buzzer** (physical button)
  - all buttons connect to the **Neon Beat controller** (**NBC**) hosting the game logic
  - a game master drives the game through a **dedicated web interface**
  - players can follow the game on a **shared screen**
  - players compete for the highest score

The Buzzer



- custom electronics
  - core: Xiao esp32c3
  - battery: 3.7V lithium battery, 320mAh
  - button: keyboard switch
  - led: WS2812
  - (coming soon: a proper PCB)
- custom casing
  - current: FreeCAD + 3D printing
  - WIP: wood work + molding

The challenge: oxydizing the firmware

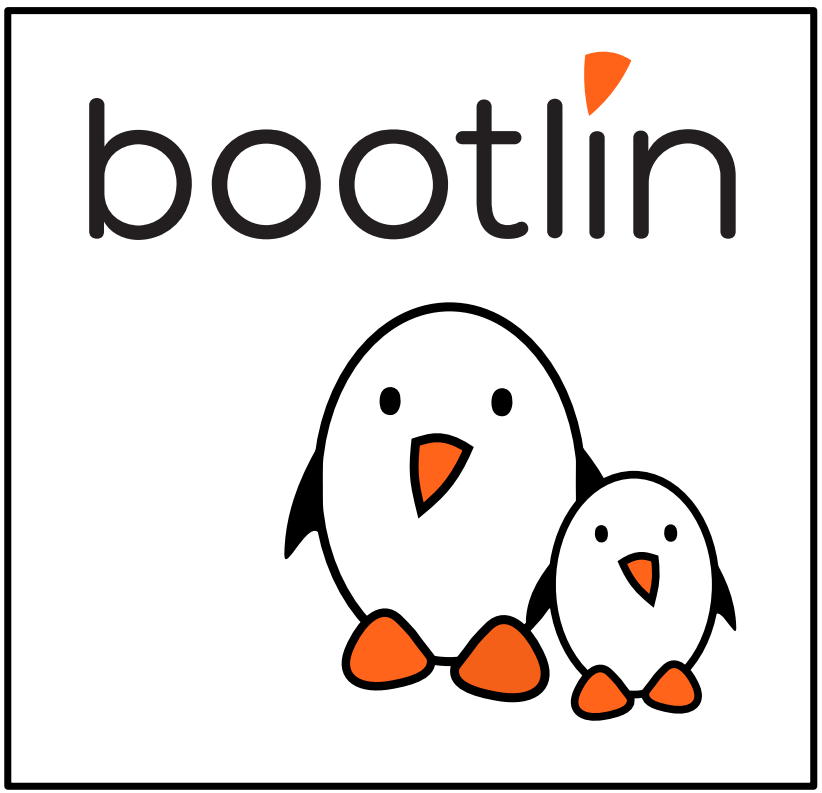
- The buzzer already runs a full custom firmware based on esp-idf (C)
- That's a perfect sandbox to practice **no_std** Rust
  - Rust, but without alloc, the fancy types, filesystems, concurrency, etc

The challenge: oxydizing the firmware

- The buzzer already runs a full custom firmware based on esp-idf (C)
- That's a perfect sandbox to practice **no_std** Rust
  - Rust, but without alloc, the fancy types, filesystems, concurrency, etc

- Expected outcome
  - Will the firmware become safe and bug-free ? => NO
  - Will the firmware become blazingly fast ? => NO
  - Will it be fun ? => LIKELY !
  - Will I learn things ? => DEFINITELY !

Host and project setup



High level SBOM

- `esp-hal` crate
  - safe APIs for peripherals access
- `esp_radio`
  - exposes wifi/ble
  - needs `esp-hal` unstable feature
  - needs an `alloc` crate: `esp-alloc`
  - needs `esp-rtos`
- `embassy` to write async code
  - `esp-rtos` provides the glue between `esp-hal` and `embassy`
- plenty of docs and examples:
  - https://docs.espressif.com/projects/rust/
  - https://github.com/esp-rs/esp-hal/tree/main/examples#examples

From zero to a working setup

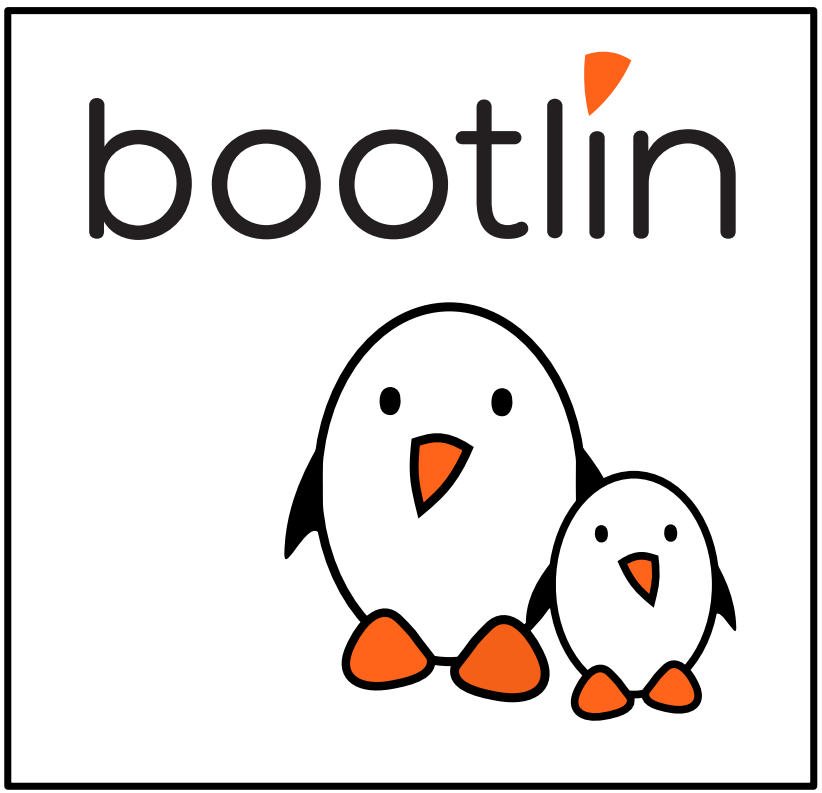
- get and run `rustup` (Rust programming language installer): http://rustup.rs
  - installs rust: `rustc`, `stdlib`, `cargo`, additional tooling
  - follow post-install instructions to correctly set ENV variables
- get `esp-generate`: `cargo install esp-generate --locked`
  - used to generate a project from a template
- run `esp-generate` to create your project. A TUI will guide you to select:
  - the platform (eg: `esp32c3`)
  - the wanted features
  - some extra tooling like `esp-flash` or `esp-config`
- and voila, you now have a ready-to-flash example:
  - `cargo run`
  - will automatically download the needed target toolchain variant

Default code (simplified)

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

#[panic_handler]
fn panic(_: &core::panic::PanicInfo) -> ! {
    loop {}
}

#[esp_rtos::main]
async fn main(spawner: Spawner) -> ! {

    let config = esp_hal::Config::default().with_cpu_clock(CpuClock::max());
    let peripherals = esp_hal::init(config);

    esp_rtos::start(timg0.timer0, sw_interrupt.software_interrupt0);

    let radio_init = esp_radio::init().expect("Failed to initialize Wi-Fi/BLE controller");
    let (mut _wifi_controller, _interfaces) = esp_radio::wifi::new(&radio_init, peripherals.WIFI, Default::default())
            .expect("Failed to initialize Wi-Fi controller");

    loop {
        Timer::after(Duration::from_secs(1)).await;
    }
}
```

Embassy tasks

```rust
#[embassy_executor::task]
async fn keepalive_message(timeout: Duration) {
    loop {
        info!("Firmware is running...");
        Timer::after(timeout).await;
    }
}

#[esp_rtos::main]
async fn main(spawner: Spawner) -> ! {

    let duration = Duration::from_secs(5);
    if let Err(e) = spawner.spawn(keepalive_message(duration)) {
        warn!("Failed to spawn the keepalive task: {e}");
    }

    loop {
        Timer::after(Duration::from_secs(1)).await;
    }
}
```



Implementation

Basic wifi connection (1/2)

```rust
#[embassy_executor::task]
async fn net_task(mut runner: Runner<'static, WifiDevice<'static>>) {
    runner.run().await
}

#[embassy_executor::task]
async fn connection(mut controller: WifiController<'static>) {
    loop {
        if esp_radio::wifi::sta_state()  == WifiStaState::Connected {
                controller.wait_for_event(WifiEvent::StaDisconnected).await;
        }
        if !matches!(controller.is_started(), Ok(true)) {
            let client_config = ModeConfig::Client( ClientConfig::default()
                    .with_ssid("nb_ap".into())
                    .with_password("nb_ap14789".into()),
            );
            controller.set_config(&client_config).unwrap();
            controller.start_async().await.unwrap();
        }

        if let Err(e) = controller.connect_async().await {
            info!("Failed to connect to wifi: {e:?}");
        } else {
            info!("Wifi connected!");
        }
    }
}
```

Basic wifi connection (2/2)

```rust
#[esp_rtos::main]
async fn main(spawner: Spawner) -> ! {

    /* [...] */

    let esp_radio_ctrl = &*mk_static!(Controller<'static>, esp_radio::init().unwrap());
    let (controller, interfaces) = esp_radio::wifi::new(&esp_radio_ctrl, peripherals.WIFI, Default::default()).unwrap();

    let config = embassy_net::Config::dhcpv4(Default::default());
    let rng = Rng::new();
    let seed = (rng.random() as u64) << 32 | rng.random() as u64;
    let (stack, runner) = embassy_net::new(interfaces.sta, config, mk_static!(StackResources<3>, StackResources::<3>::new()),
        seed);

    spawner.spawn(connection(controller)).expect("Can not spawn net task");
    spawner.spawn(net_task(runner)).expect("Can not spawn wifi task");

    while stack.config_v4().is_none() {
        Timer::after(Duration::from_millis(500)).await;
    }

    info!("Buzzer connected to NBC");

    loop { /* [...] */ }
}
```

False start

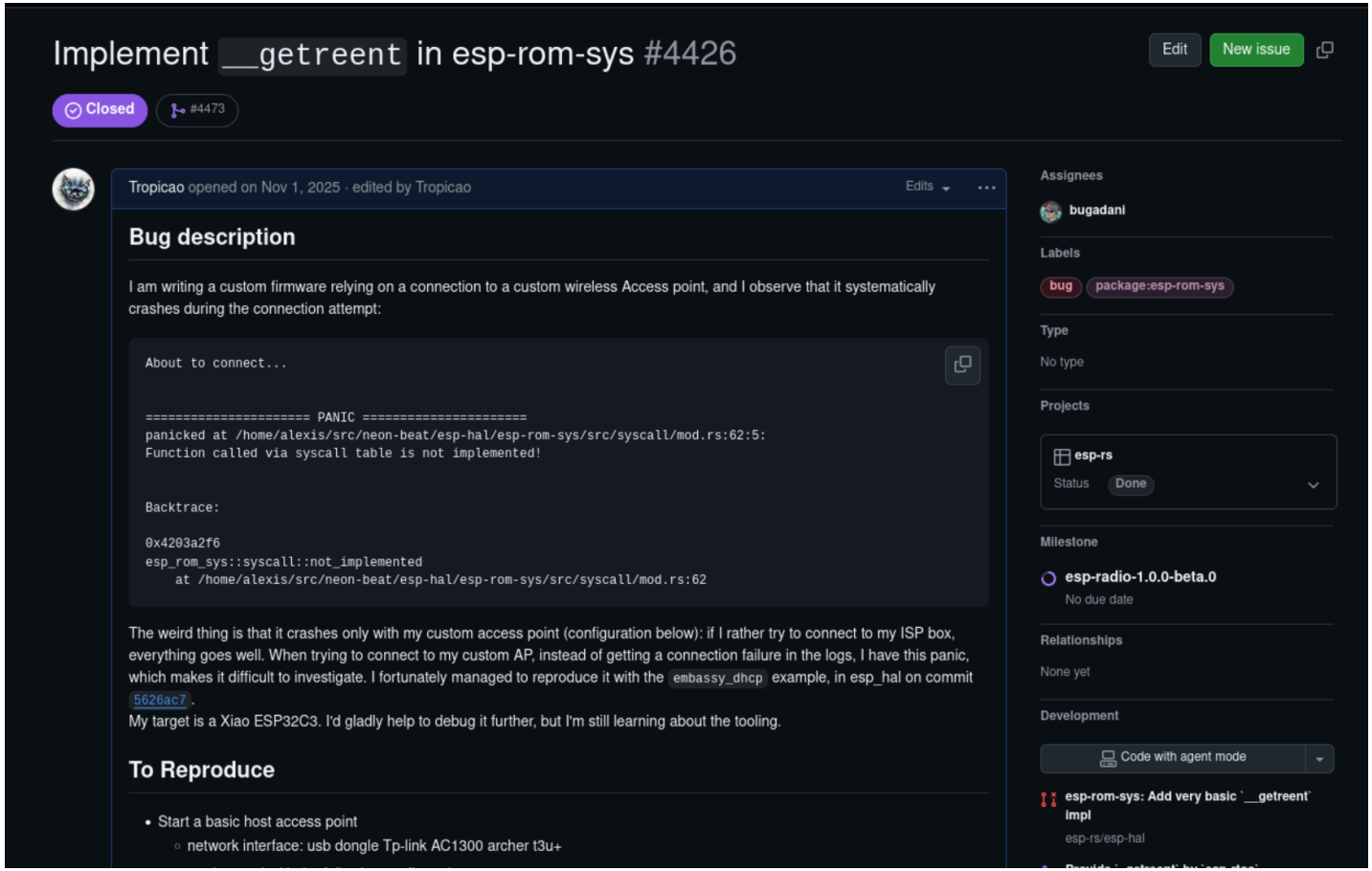
```
ESP-ROM:esp32c3-api1-20210207
Build:Feb  7 2021
rst:0x15 (USB_UART_CHIP_RESET),boot:0x9 (SPI_FAST_FLASH_BOOT)
Saved PC:0x40380862
SPIWP:0xee
mode:DIO, clock div:2
load:0x3fcd5820,len:0x15c4
load:0x403cbf10,len:0xc84
load:0x403ce710,len:0x2fd0
entry 0x403cbf1a
[...]
```

```
====================== PANIC ======================
panicked at /home/alexis/src/cargo/registry/src/index.crates.io-1949cf8c6b5b557f/esp-rom-sys-0.1.3/src/syscall/mod.rs:62:5:
Function called via syscall table is not implemented!
```

```
Backtrace:

0x42039cc8
esp_rom_sys::syscall::not_implemented
    at /home/alexis/src/cargo/registry/src/index.crates.io-1949cf8c6b5b557f/esp-rom-sys-0.1.3/src/syscall/mod.rs:62
```

- reproducible with `examples/wifi/embassy_dhcp` from `esp-hal`

Calling for help



Implement `__getreent` in esp-rom-sys #4426

Edit | New issue

Closed | #4473

Tropicao opened on Nov 1, 2025 · edited by Tropicao — Edits

Bug description

I am writing a custom firmware relying on a connection to a custom wireless Access point, and I observe that it systematically crashes during the connection attempt:

```
About to connect...


==================== PANIC ====================
panicked at /home/alexis/src/neon-beat/esp-hal/esp-rom-sys/src/syscall/mod.rs:62:5:
Function called via syscall table is not implemented!


Backtrace:

0x4203a2f6
esp_rom_sys::syscall::not_implemented
    at /home/alexis/src/neon-beat/esp-hal/esp-rom-sys/src/syscall/mod.rs:62
```

The weird thing is that it crashes only with my custom access point (configuration below): if I rather try to connect to my ISP box, everything goes well. When trying to connect to my custom AP, instead of getting a connection failure in the logs, I have this panic, which makes it difficult to investigate. I fortunately managed to reproduce it with the `embassy_dhcp` example, in esp_hal on commit `5626ac7`.
My target is a Xiao ESP32C3. I'd gladly help to debug it further, but I'm still learning about the tooling.

To Reproduce

- Start a basic host access point
  - network interface: usb dongle Tp-link AC1300 archer t3u+

Assignees: bugadani

Labels: bug, package:esp-rom-sys

Type: No type

Projects: esp-rs — Status: Done

Milestone: esp-radio-1.0.0-beta.0 — No due date

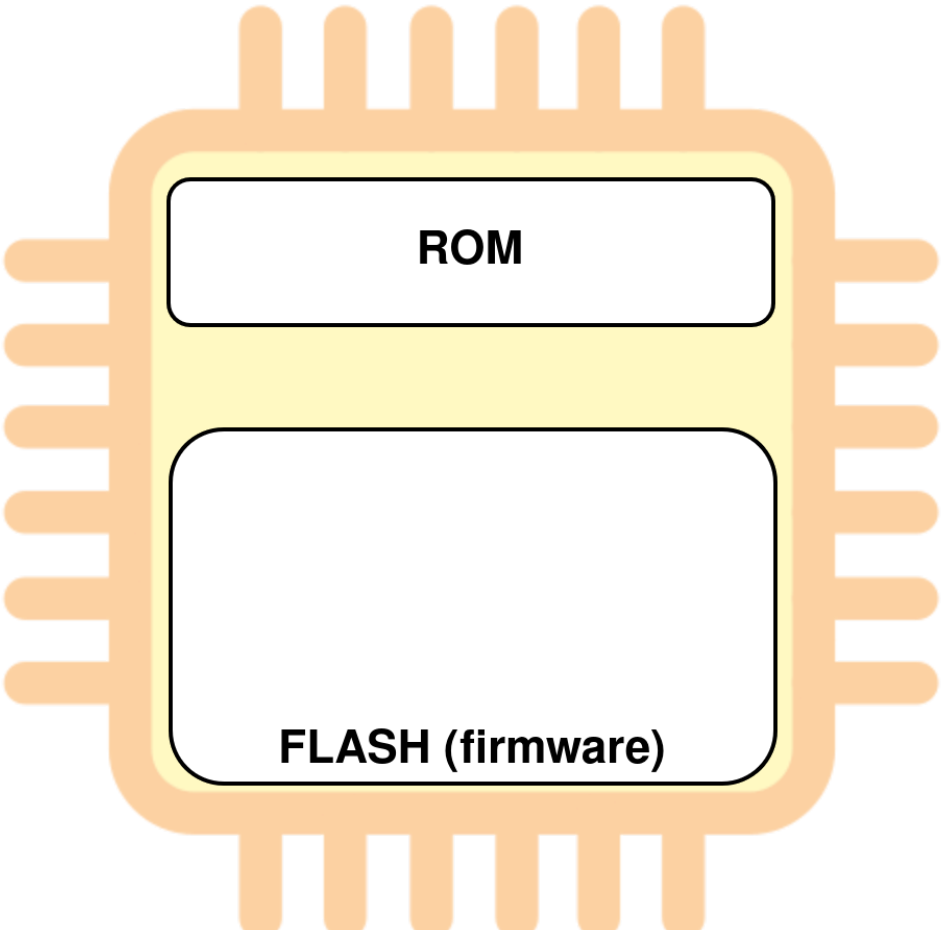
Relationships: None yet

Development: Code with agent mode

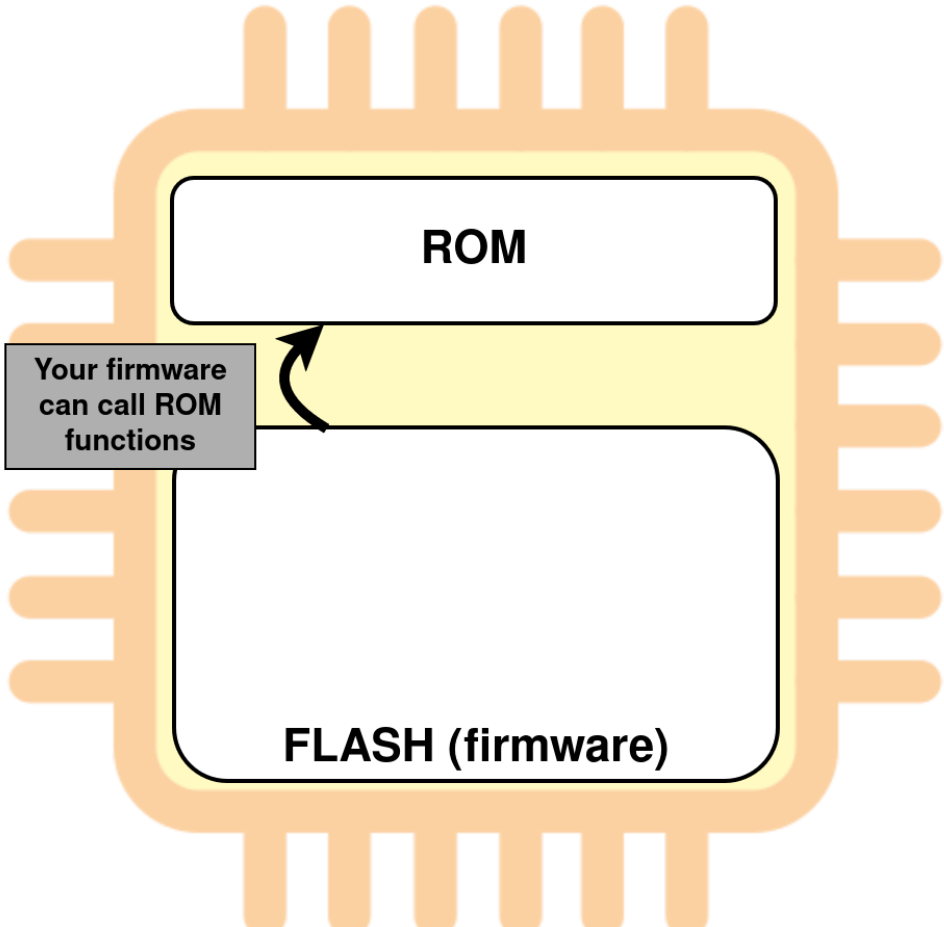
esp-rom-sys: Add very basic `__getreent` impl
esp-rs/esp-hal

https://github.com/esp-rs/esp-hal/issues/4426

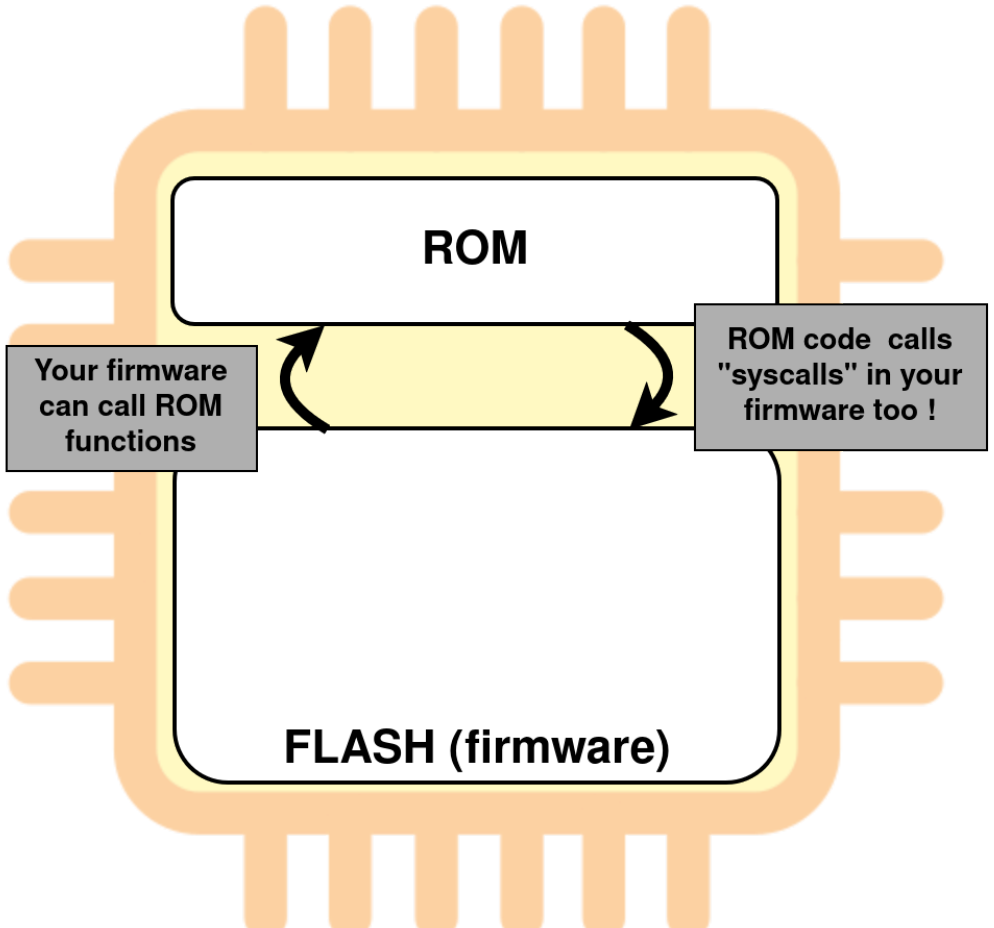
Espressif chips ROM code



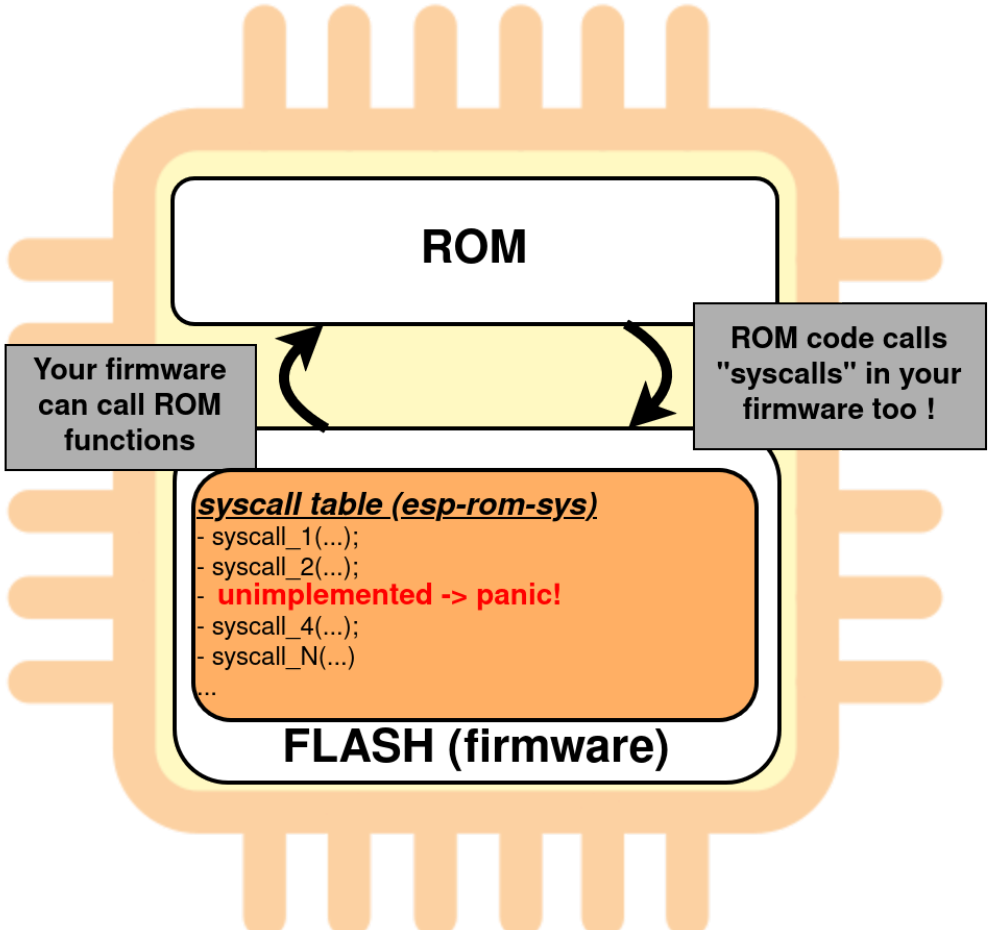
Espressif chips ROM code



Espressif chips ROM code



Espressif chips ROM code



Espressif help

- after a few rounds on Github, and in a bit less than two weeks:
  - issue is identified in `esp-rom-sys`
  - the `__getreent` syscall is implemented
  - new tests show that `_malloc_r` and `_free_r` also need to be implemented
  - finally:

```
ESP-ROM:esp32c3-api1-20210207
Build:Feb  7 2021
rst:0x15 (USB_UART_CHIP_RESET),boot:0x8 (SPI_FAST_FLASH_BOOT)
[...]
I (114) esp_image: segment 2: paddr=00030020 vaddr=42020020 size=71e68h (466536) map
I (218) esp_image: segment 3: paddr=000a1e90 vaddr=3fc8a6a0 size=00ff0h (  4080) load
I (220) esp_image: segment 4: paddr=000a2e88 vaddr=40380000 size=09784h ( 38788) load
I (235) boot: Loaded app from partition at offset 0x10000
I (235) boot: Disabling RNG early entropy source...
INFO - IPv4: DOWN
INFO - Waiting on link up...
INFO - Waiting on link up...
INFO - Waiting on link up...
INFO - Waiting on link up...
INFO - link_up = true
INFO - IPv4: DOWN
INFO - Wifi connected!
INFO - Buzzer connected to NBC
```

@bugadani

@JurajSadel

@MabezDev

I owe you a beer 🍻

Targeting specific crates revisions

- fixes are merged but not released yet on crates.io
- no problem, we can use temporary remotes:

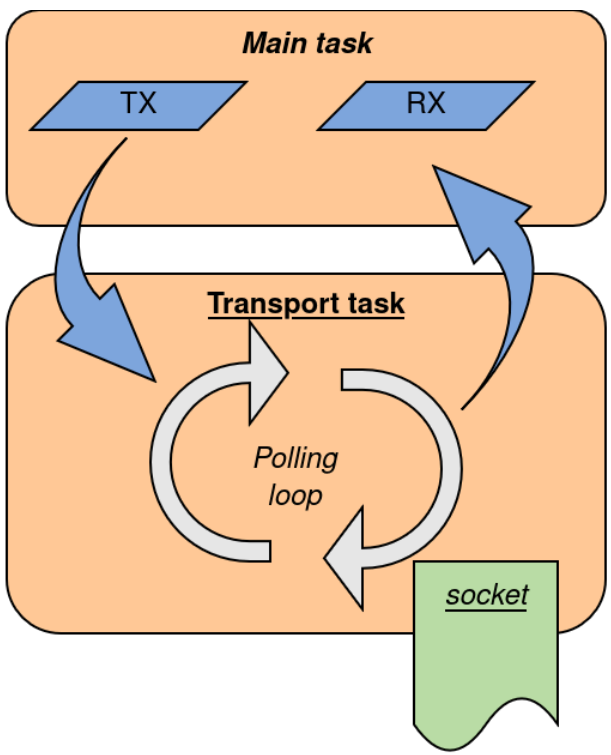
```
# in Cargo.toml
[patch.crates-io]
esp-hal = { git = "https://github.com/esp-rs/esp-hal", rev="223815270092663682a151a1b285665587a3d5dd" }
esp-rtos = { git = "https://github.com/esp-rs/esp-hal", rev="223815270092663682a151a1b285665587a3d5dd" }
esp-bootloader-esp-idf = { git = "https://github.com/esp-rs/esp-hal", rev="223815270092663682a151a1b285665587a3d5dd" }
esp-alloc = { git = "https://github.com/esp-rs/esp-hal", rev="223815270092663682a151a1b285665587a3d5dd" }
esp-radio = { git = "https://github.com/esp-rs/esp-hal", rev="223815270092663682a151a1b285665587a3d5dd" }
```

Basic socket management

```
loop {
    /* Don't do this at home, please be gentle with your stack */
    let mut rx_buffer:[u8;512] = [0;512];
    let mut tx_buffer:[u8;512] = [0;512];

    let mut socket = TcpSocket::new(stack, &mut rx_buffer, &mut tx_buffer);
    socket.set_timeout(Some(Duration::from_secs(10)));
    socket.set_keep_alive(Some(Duration::from_secs(8)));
    let remote = (Ipv4Addr::new(192, 168, 66, 1), 80);
    let res = socket.connect(remote).await;
    if let Err(e) = res {
        error!("Failed to connect to TCP server: {:?}", e);
        continue;
    }
    while socket.state() == embassy_net::tcp::State::Established {
        info!("Waiting for some data...");
        Timer::after(Duration::from_secs(5)).await;
    }
}
```

Rust ~~inflexibility~~ strength

- many parts of the firwmare want to send/receive data
- but only one task can **own** the socket
- let's use **channels** to share the transport layer



First attempt

```rust
#[embassy_executor::task]
pub async fn socket_task(
    stack: embassy_net::Stack,
    tx_chan: embassy_sync::channel::Receiver<NoopRawMutex, Message, 1>,
    rx_chan: embassy_sync::channel::Sender<NoopRawMutex, Command, 1>,
) {
    loop {
        match select(socket.read(&mut buf), tx_chan.receive()).await {
            Either::First(count) => {
                info!("Received {count} bytes");
                /* [...] */
                rx_chan.send(Command::LedOn).await;
            },
            Either::Second(status) => {
                info!("Sending {status} to NBC");
                /* [...] */
                socket.write(&buf).await.expect("Failed to send message");
            }
        }
    }
}
```

```rust
#[derive(Debug)]
enum Message {
    Identify,
    Buzz,
}

#[derive(Debug)]
enum Command {
    LedOn,
    LedOff,
}
```

```rust
let tx = Channel::new();
let rx = Channel::new();
spawner
    .spawn(socket_task(stack,
                       tx.receiver(),
                       rx.sender()))
    .expect("Failed to spawn tcp task");
/*[...]*/
tx.send(Message::Identify).await;
```

Note the `select` call: `socket.read()` and `tx_chan.receive()` return Futures ! We have to `await` them !

So it begins

```
   Compiling neon-beat-buzzer v0.1.0 (/home/alexis/src/neon-beat/neon-beat-buzzer)
error[E0726]: implicit elided lifetime not allowed here
   --> src/main.rs:123:14
    |
123 |     tx_chan: embassy_sync::channel::Receiver<NoopRawMutex, Message, 1>,
    |              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ expected lifetime parameter
    |
help: indicate the anonymous lifetime
    |
123 |     tx_chan: embassy_sync::channel::Receiver<'_, NoopRawMutex, Message, 1>,
    |                                             +++

error[E0726]: implicit elided lifetime not allowed here
   --> src/main.rs:124:14
    |
124 |     rx_chan: embassy_sync::channel::Sender<NoopRawMutex, Command, 1>,
    |              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ expected lifetime parameter
    |
help: indicate the anonymous lifetime
    |
124 |     rx_chan: embassy_sync::channel::Sender<'_, NoopRawMutex, Command, 1>,
    |                                           +++
```

Second attempt

```
#[embassy_executor::task]
pub async fn socket_task(
    stack: embassy_net::Stack,
    tx_chan: embassy_sync::channel::Receiver<'_, NoopRawMutex, Message, 1>,
    rx_chan: embassy_sync::channel::Sender<'_, NoopRawMutex, Command, 1>,
) {
  /* [...] */
}
```

Second attempt

```
#[embassy_executor::task]
pub async fn socket_task(
    stack: embassy_net::Stack,
    tx_chan: embassy_sync::channel::Receiver<'_, NoopRawMutex, Message, 1>,
    rx_chan: embassy_sync::channel::Sender<'_, NoopRawMutex, Command, 1>,
) {
  /* [...] */
}
```

```
    Checking neon-beat-buzzer v0.1.0 (/home/alexis/src/neon-beat/neon-beat-buzzer)
error: Arguments for tasks must live forever. Try using the `'static` lifetime.
   --> src/main.rs:123:46
    |
123 |     tx_chan: embassy_sync::channel::Receiver<'_, NoopRawMutex, Message, 1>,
    |                                              ^^

error: Arguments for tasks must live forever. Try using the `'static` lifetime.
   --> src/main.rs:124:44
    |
124 |     rx_chan: embassy_sync::channel::Sender<'_, NoopRawMutex, Command, 1>,
    |                                            ^^
```

Third attempt

```
#[embassy_executor::task]
pub async fn socket_task(
    stack: embassy_net::Stack,
    tx_chan: embassy_sync::channel::Receiver<'static, NoopRawMutex, Message, 1>,
    rx_chan: embassy_sync::channel::Sender<'static, NoopRawMutex, Command, 1>,
) {
  /* [...] */
}
```

Third attempt

```
#[embassy_executor::task]
pub async fn socket_task(
    stack: embassy_net::Stack,
    tx_chan: embassy_sync::channel::Receiver<'static, NoopRawMutex, Message, 1>,
    rx_chan: embassy_sync::channel::Sender<'static, NoopRawMutex, Command, 1>,
) {
  /* [...] */
}
```

```
   Compiling neon-beat-buzzer v0.1.0 (/home/alexis/src/neon-beat/neon-beat-buzzer)
error[E0597]: `tx` does not live long enough
   --> src/main.rs:98:35
    |
 95 |     let tx = Channel::new();
    |         -- binding `tx` declared here
...
 98 |         .spawn(socket_task(stack, tx.receiver(), rx.sender()))
    |                ------------------^^-------------------------
    |                |                 |
    |                |                 borrowed value does not live long enough
    |                argument requires that `tx` is borrowed for `'static`
[...]
```

Fourth attempt

```
static TX: Channel<NoopRawMutex, Message, 1> = Channel::new();
static RX: Channel<NoopRawMutex, Command, 1> = Channel::new();

#[esp_rtos::main]
async fn main(spawner: Spawner) -> ! {
/* [...] */
spawner
    .spawn(socket_task(stack, TX.receiver(), RX.sender()))
    .expect("Failed to spawn tcp task");
TX.send(Message::Buzz);
```

Hmmm, why did we decide to stop coding in C, again ?

Fourth attempt

```
static TX: Channel<NoopRawMutex, Message, 1> = Channel::new();
static RX: Channel<NoopRawMutex, Command, 1> = Channel::new();

#[esp_rtos::main]
async fn main(spawner: Spawner) -> ! {
/* [...] */
spawner
    .spawn(socket_task(stack, TX.receiver(), RX.sender()))
    .expect("Failed to spawn tcp task");
TX.send(Message::Buzz);
```

Hmmm, why did we decide to stop coding in C, again ?

```
    Checking neon-beat-buzzer v0.1.0 (/home/alexis/src/neon-beat/neon-beat-buzzer)
error[E0277]: `*mut ()` cannot be shared between threads safely
   --> src/main.rs:51:12
    |
 51 | static TX: Channel<NoopRawMutex, Message, 1> = Channel::new();
    |            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ `*mut ()` cannot be shared between threads safely
    |
    = help: within `NoopRawMutex`, the trait `Sync` is not implemented for `*mut ()`
note: required because it appears within the type `PhantomData<*mut ()>`
   --> /home/alexis/src/rustup/toolchains/stable-x86_64-unknown-linux-gnu/lib/rustlib/src/rust/library/core/src/marker.rs:822:12
    |
822 | pub struct PhantomData<T: PointeeSized>;
    |            ^^^^^^^^^^^
note: required because it appears within the type `NoopRawMutex`
   --> /home/alexis/src/cargo/registry/src/index.crates.io-1949cf8c6b5b557f/embassy-sync-0.7.2/src/blocking_mutex/raw.rs:70:12
    |
 70 | pub struct NoopRawMutex {
    |            ^^^^^^^^^^^^
    = note: required for `embassy_sync::blocking_mutex::Mutex<NoopRawMutex, RefCell<embassy_sync::channel::ChannelState<Message, 1>>>` to implement `Sync`
```

Fifth (and final) attempt

```
static TX: StaticCell<Channel<NoopRawMutex, Message, 1>> = StaticCell::new();
static RX: StaticCell<Channel<NoopRawMutex, Command, 1>> = StaticCell::new();

#[esp_rtos::main]
async fn main(spawner: Spawner) -> ! {

let tx: &'static mut _ = TX.init(Channel::new());
let rx: &'static mut _ = RX.init(Channel::new());
spawner
    .spawn(socket_task(stack, tx.receiver(), rx.sender()))
    .expect("Failed to spawn tcp task");
```

Fifth (and final) attempt

```
static TX: StaticCell<Channel<NoopRawMutex, Message, 1>> = StaticCell::new();
static RX: StaticCell<Channel<NoopRawMutex, Command, 1>> = StaticCell::new();

#[esp_rtos::main]
async fn main(spawner: Spawner) -> ! {

let tx: &'static mut _ = TX.init(Channel::new());
let rx: &'static mut _ = RX.init(Channel::new());
spawner
    .spawn(socket_task(stack, tx.receiver(), rx.sender()))
    .expect("Failed to spawn tcp task");
```

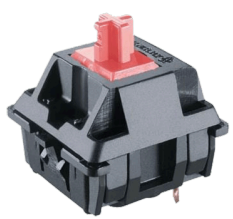
```
Compiling neon-beat-buzzer v0.1.0 (/home/alexis/src/neon-beat/neon-beat-buzzer)
 Finished `dev` profile [optimized + debuginfo] target(s) in 0.24s
```

Alleluia

Button management: finally something simple

```
use embassy_sync::{blocking_mutex::raw::NoopRawMutex, channel::Sender};
use esp_hal::gpio::{AnyPin, Input, InputConfig, Pull};
use log::info;

#[embassy_executor::task]
pub async fn button_task(pin: AnyPin<'static>, sender: Sender<'static, NoopRawMutex, bool, 1>) {
    let config = InputConfig::default().with_pull(Pull::Up);
    let mut button = Input::new(pin, config);
    loop {
        button.wait_for_falling_edge().await;
        info!("Button pushed!");
        sender.send(true).await;
    }
}
```

- `esp-hal` exposes async APIs for GPIOs
- it even handles automatically interrupt configuration !

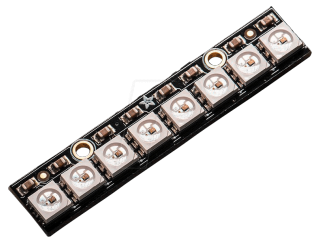


Button management: finally something ~~simple~~ almost simple

```rust
const DEBOUNCE_MS: u64 = 100;

#[embassy_executor::task]
pub async fn button_task(pin: AnyPin<'static>, sender: Sender<'static, NoopRawMutex, bool, 1>) {
    let config = InputConfig::default().with_pull(Pull::Up);
    let mut pushed = false;
    let mut button = Input::new(pin, config);
    loop {
        button.wait_for_falling_edge().await;
        if !pushed {
            info!("Button pushed!");
            sender.send(true).await;
        }
        /* Quick and dirty deboucing, enough as long as we only need to
         * detect single, short presses
         */
        Timer::after_millis(DEBOUNCE_MS).await;
        pushed = button.is_low();
    }
}
```

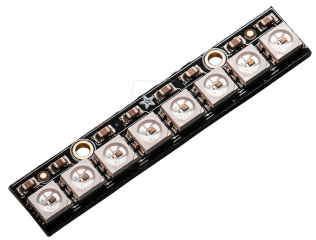
LED management

- WS2812 is controlled with a specific serial protocol
- Most ESP32 chips have a **RMT peripheral** available
  - Generally used to control infrared transceivers
  - disable carrier modulation, and voila, you know how to talk to a led



LED management

- WS2812 is controlled with a specific serial protocol
- Most ESP32 chips have a **RMT peripheral** available
  - Generally used to control infrared transceivers
  - disable carrier modulation, and voila, you know how to talk to a led



- Once again, let's benefit from **existing crates**:

```
cargo add smart-leds
cargo add esp-hal-smartled
```

```
use esp_hal_smartled::{SmartLedsAdapterAsync, smart_led_buffer};
use smart_leds::{RGB, brightness, SmartLedsWriteAsync};

let mut buffer = smart_led_buffer!(1);
let rmt = Rmt::new(peripherals.RMT, Rate::from_mhz(80)).expect("Failed to initialize RMT controller");
let red: RGB<u8> = RGB::new(255, 0, 0);

let mut led = SmartLedsAdapterAsync::new(rmt.into_async().channel0, peripherals.GPIO3, &mut buffer);
led.write(brightness([red].into_iter(), 255)).await.expect("Failed to set led on");
```

Exchanging messages with the NBC

- The buzzer sends status messages to the controller:


```
{"type":"identification","id":"64e833b6ab18"}
```


- The controller send LED commands over websocket as json payloads:


```
{"pattern":{"type":"blink","details":{"duration_ms":1000,"period_ms":200,"dc":0.5,"color":{"h":125.0,"s":1.0,"v":1.0}}}}
```


- wouldn't it be nice to have a crate/framework to automatically {de}serialize messages ?

Exchanging messages with the NBC

- The buzzer sends status messages to the controller:

```
{"type":"identification","id":"64e833b6ab18"}
```

- The controller send LED commands over websocket as json payloads:

```
{"pattern":{"type":"blink","details":{"duration_ms":1000,"period_ms":200,"dc":0.5,"color":{"h":125.0,"s":1.0,"v":1.0}}}}
```

- wouldn't it be nice to have a crate/framework to automatically {de}serialize messages ?

*Time for some **Serde** goodness !*

Serde

- allows {de}serializing plenty of standard formats (json, CBOR, Yaml, CSV, toml...)
- you can also write your own {de}serializer
- `serde`: the main crate
  - contains the `Serialize` and `Deserialize` traits
  - able to handle plenty of standard types

Serde

- allows {de}serializing plenty of standard formats (json, CBOR, Yaml, CSV, toml...)
- you can also write your own {de}serializer
- `serde`: the main crate
  - contains the `Serialize` and `Deserialize` traits
  - able to handle plenty of standard types
- `serde_json`: a serde-based {de}serializer
  - allows "anonymous" or "strongly types" deserialization
  - depends on `std` by default
  - can work without `std`, but still needs `alloc`

Serde

- allows {de}serializing plenty of standard formats (json, CBOR, Yaml, CSV, toml...)
- you can also write your own {de}serializer
- `serde`: the main crate
  - contains the `Serialize` and `Deserialize` traits
  - able to handle plenty of standard types
- `serde_json`: a serde-based {de}serializer
  - allows "anonymous" or "strongly types" deserialization
  - depends on `std` by default
  - can work without `std`, but still needs `alloc`
- `serde_json_core`: a no_std, no alloc, serde-based {de}serializer
  - only supports "strongly typed" deserialization
  - does not handle as many types as full serde
  - but hey, we're doing embedded development !

$\Longrightarrow$ *`cargo add serde && cargo add serde_json_core`*

Handling status and commands (1/3)

```
{
  "pattern":{
    "type":"blink",
    "details":{
      "duration_ms":1000,
      "period_ms":200,
      "dc":0.5,
      "color":{
        "h":125.0,
        "s":1.0,
        "v":1.0
      }
    }
  }
}
```

⟹

```
#[derive(Deserialize, Debug)]
pub struct MessageLedPattern<'a> {
    #[serde(borrow)]
    pattern: MessageLedType<'a>,
}

#[derive(Deserialize, Debug)]
struct MessageLedType<'a> {
    r#type: &'a str,
    details: MessageLedDetails,
}

#[derive(Deserialize, Debug)]
struct MessageLedDetails {
    duration_ms: u32,
    period_ms: u32,
    dc: f32,
    color: MessageLedColor,
}

#[derive(Deserialize, Debug)]
struct MessageLedColor {
    h: f32,
    s: f32,
    v: f32,
}
```

Handling status and commands (2/3)

- to parse a received command:

```
let cmd = serde_json_core::from_slice::<MessageLedPattern>(&[..msg_len])
if cmd.is_err() { /* [...] */ }
match cmd.pattern.r#type {
  "blink" => { /* [...] */ },
  "wave" => { /* [...] */ },
  _ => { /* [...] */ }
}
```

Handling status and commands (2/3)

- to parse a received command:

```
let cmd = serde_json_core::from_slice::<MessageLedPattern>(&[..msg_len])
if cmd.is_err() { /* [...] */ }
match cmd.pattern.r#type {
  "blink" => { /* [...] */ },
  "wave" => { /* [...] */ },
  _ => { /* [...] */ }
}
```

- to serialize a status:

```
struct StatusMessageData<'a, 'b> {
    r#type: &'a str,
    id: &'b str,
}
```

```
let mut buffer = [u8;512];
let ident = StatusMessageData {
    r#type: "identification",
    id: "deadbeefcafe"
};
if let Ok(count) = serde_json_core::to_slice(&ident, &mut buffer) {
  socket.write(buffer[..count])
}
```

Handling status and commands (3/3)

- But some messages have a slightly different format !

```
{"pattern":{"type":"blink","details":{"duration_ms":1000,"period_ms":200,"dc":0.5,"color":{"h":125.0,"s":1.0,"v":1.0}}}}
```

```
{"pattern":{"type":"off"}}
```

Handling status and commands (3/3)

- But some messages have a slightly different format !

```
{"pattern":{"type":"blink","details":{"duration_ms":1000,"period_ms":200,"dc":0.5,"color":{"h":125.0,"s":1.0,"v":1.0}}}}
```

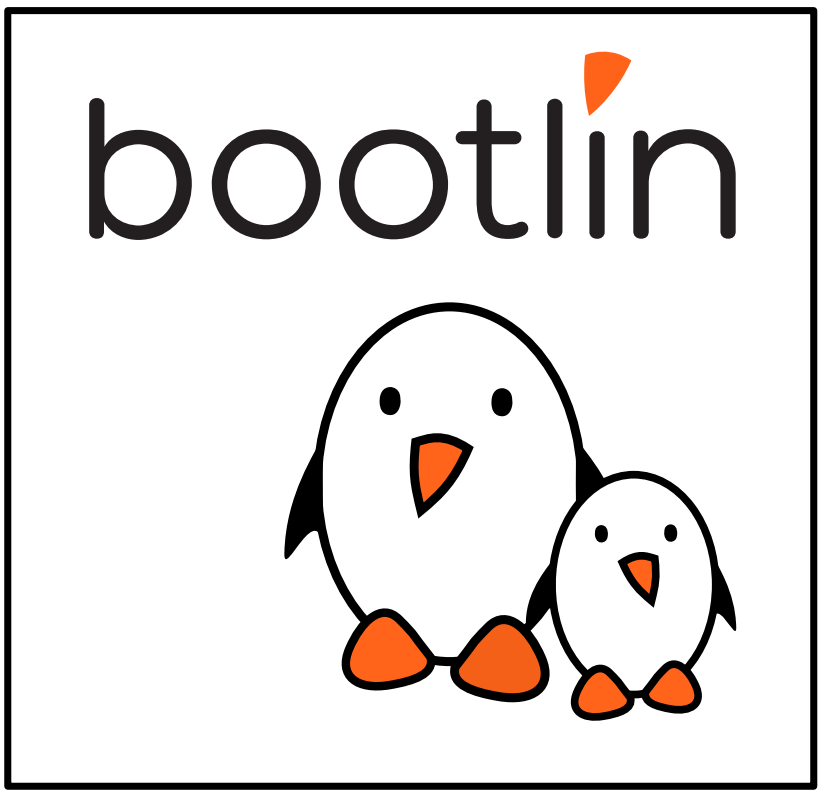
```
{"pattern":{"type":"off"}}
```

```
#[derive(Deserialize, Debug)]
pub struct MessageLedPattern<'a> {
    #[serde(borrow)]
    pattern: MessageLedType<'a>,
}

#[derive(Deserialize, Debug)]
struct MessageLedType<'a> {
    r#type: &'a str,
    details: MessageLedDetails,
}
```

```
#[derive(Deserialize, Debug)]
pub struct MessageLedPattern<'a> {
    #[serde(borrow)]
    pattern: MessageLedType<'a>,
}

#[derive(Deserialize, Debug)]
struct MessageLedType<'a> {
    r#type: &'a str,
    details: Option(MessageLedDetails)
}
```

```
if value.pattern.type != "off" {
  let details = value.pattern.details.ok_or(PatternError::MissingDetails)?;
  /* [...] */
}
```

Show time



Show time

```
cargo run
```

Next steps and improvements

- final binary size (1.3MB 😳)
- better websocket management
- more idiomatic error management
- tests !

Thank you!

Questions?

https://github.com/neon-beat/neon-beat-buzzer-fw.git

Alexis Lothoré

https://bootlin.com/pub/conferences/