

Oh crab!



My watch speaks Rust

Pierre-Yves Aillet

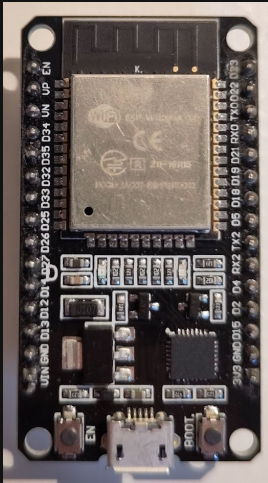
Quelques rappels et définitions sur l'embarqué

Pas d'OS "classique"

- GPIO: General Purpose Input/Output
- I2C: Inter-Integrated Circuit
- SPI: Serial Peripheral Interface
- UART: Universal Asynchronous Receive/Transfer

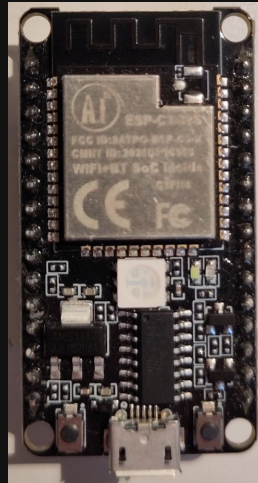
ESP32

ESP32 / ESP32-S*



Architecture: xtensa-lx6
Licensed from Tensilica

ESP32-C*



Architecture: RISC-V
Open standard instruction set architecture



Rappels sur Rust

Rust est un langage de programmation compilé multi-paradigme conçu et développé par Mozilla Research depuis 2010”, puis par la Rust Foundation

”

Wikipedia



Rappels sur Rust

- Memory safety - ownership
- Type safety - explicit error handling, safe refactoring
- Zero Cost Abstractions
- Fearless concurrency
- Great Developer eXperience



Rappels sur Rust

Structure d'un projet

```
.  
├─ Cargo.toml  
└─ src  
    └─ main.rs
```

Manipulations du projet

```
# Construction du projet  
$ cargo build  
  
# Lancement du projet  
$ cargo run  
  
# Formattage du code  
$ cargo fmt
```



Pourquoi Rust est adapté pour de l'embarqué ?

- no_std
 - core
 - alloc
 - std

hello

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

use esp32_hal::{clock::ClockControl, pac::Peripherals, prelude::*, Delay};
use esp_backtrace as _;
use std::fmt::println; // <= This is not possible with no_std :(

#[xtensa_lx_rt::entry]
fn main() -> ! {
    let peripherals = Peripherals::take().unwrap();
    let system = peripherals.DPORT.split();
    let clocks = ClockControl::boot_defaults(system.clock_control).freeze();

    let mut delay = Delay::new(&clocks);

    loop {
        println!("Hello world!");
        delay.delay_ms(1000u32);
    }
}
```



Pourquoi Rust est adapté pour de l'embarqué ?

- no_std
- Typestate pattern

blinky

```
fn main() -> ! {  
    let peripherals = Peripherals::take().unwrap();  
    let system = peripherals.DPORT.split();  
    let clocks = ClockControl::boot_defaults(system.clock_control).freeze();  
  
    let mut delay = Delay::new(&clocks);  
  
    let io = IO::new(peripherals.GPIO, peripherals.IO_MUX);  
    let mut led = io.pins.gpio4.into_push_pull_output();  
  
    loop {  
        led.toggle().unwrap();  
        delay.delay_ms(1000u32);  
        println!("Led is high: {}", led.is_high());  
    }  
}
```




Pourquoi Rust est adapté pour de l'embarqué ?

- no_std
- Typestate pattern

```
struct Output {}
struct Input {}
struct Gpio4<STATE> {
    _state: PhantomData<STATE>,
}

impl<T> Gpio4<T> {
    fn into_input(self) -> Gpio4<Input> {
        Gpio4 { _state: PhantomData }
    }
    fn into_output(self) -> Gpio4<Output> {
        Gpio4 { _state: PhantomData }
    }
}

impl Gpio4<Input> {
    fn is_high() -> bool { unimplemented!() }
}

impl Gpio4<Output> {
    fn toggle(&mut self) {}
}
```



Pourquoi Rust est adapté pour de l'embarqué ?

- no_std
- Typestate pattern
- Ownership, &mut unique

```
struct Blinker {  
    led: Gpio4<Output>  
}  
  
impl Blinker {  
    fn run() {  
        // [...]  
    }  
}  
  
fn main() -> ! {  
    // [...]  
  
    let mut led = io.pins.gpio4.into_push_pull_output();  
    let blinker = Blinker { led };  
    blinker.run();  
  
    led.into_input();  
  
    // [...]  
}
```



Pourquoi Rust est adapté pour de l'embarqué ?

- no_std
- Typestate pattern
- Ownership, &mut unique
- Gestion des dépendances et compilation conditionnelle

```
[package]
name = "blinky"
version = "0.1.0"
authors = ["Pierre-Yves Aillet <pyaillet@gmail.com>"]
edition = "2021"
license = "MIT OR Apache-2.0"

[dependencies]
esp32-hal = "0.5.0"

esp-backtrace = { version = "0.2.0", features = ["esp32"] }

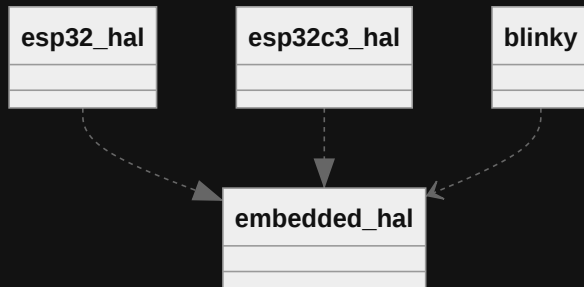
xtensa-lx-rt = { version = "0.13.0", features = ["esp32"] }

[features]
default = ["rt"]
rt = ["xtensa-lx-rt"]
```



Pourquoi Rust est adapté pour de l'embarqué ?

- no_std
- Typestate pattern
- Ownership, &mut unique
- Gestion des dépendances et compilation conditionnelle
- Abstractions





Pourquoi Rust est adapté pour de l'embarqué ?

- no_std
- Typestate pattern
- Ownership, &mut unique
- Gestion des dépendances et compilation conditionnelle
- Abstractions

```
// in embedded-hal
pub trait OutputPin {
    fn set_high(&mut self);
}

// in esp32-hal
impl OutputPin for Gpio4 {
    fn set_high(&mut self) {
        // specific code
    }
}

// in blinky
fn blink(mut led: impl OutputPin) {
    led.set_high();
}
```

😬 Mais...

- Il manque certaines abstractions
 - de base : interrupt, DMA, ...
 - de haut niveau : Bluetooth, WiFi[1]

[1] c'est en cours... embedded_svc



Qu'est-ce que c'est que cette montre ?

T-Watch 2020

LILYGO®

TFT_MISO	NULL
TFT_MOSI	I019
TFT_SCLK	I018
TFT_CS	I005
TFT_DC	I027
TFT_RST	NULL
TFT_BL	I012

ST7789V
1.54 LCD

BMA423
AxisSensor

Interrupt	I039
I2C_SDA	I021
I2C_SCL	I022

I2S Class
Max98357A

I2S_BCK	I026
I2S_WS	I025
I2C_DOUT	I033

Vibration
Motor:GPIO4

Power Button
2S ON,6S OFF

PMU:AXP202

Interrupt	I035
I2C_SDA	I021
I2C_SCL	I022

IR:I013

USB Port

Touch Board

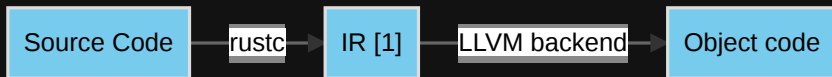
Interrupt	I038
I2C_SDA	I023
I2C_SCL	I032

Interrupt
RTC:I037

La construction du firmware, comment ça fonctionne ?

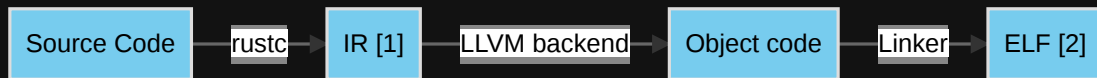
Source Code

La construction du firmware, comment ça fonctionne ?



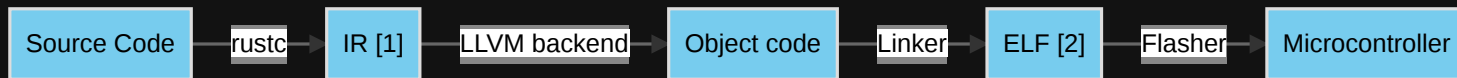
- [1] IR: Intermediate Representation

La construction du firmware, comment ça fonctionne ?



- [1] IR: Intermediate Representation
- [2] ELF: Executable and Linkable Format

La construction du firmware, comment ça fonctionne ?



- [1] IR: Intermediate Representation
- [2] ELF: Executable and Linkable Format

Tooling

- toolchains et crosscompilation

```
> rustup target list
aarch64-unknown-linux-gnu
aarch64-unknown-linux-musl
aarch64-unknown-none
aarch64-unknown-none-softwarefloat
riscv32i-unknown-none-elf
thumbv6m-none-eabi
wasm32-unknown-emscripten
wasm32-unknown-unknown
wasm32-wasi
x86_64-apple-darwin (installed)
x86_64-apple-ios
x86_64-fortanix-unknown-sgx
x86_64-fuchsia
x86_64-linux-android
```

Tooling

L'architecture xtensa n'est pas supportée par le compilateur rustc...

Parce que le backend xtensa n'existe pas côté llvm...

MAIS !

C'est en cours...

Depuis 3 ans...

Malgré tout, il y a eu des progrès prometteurs dernièrement 🎉

– <https://discourse.llvm.org/t/rfc-request-for-upstream-tensilica-xtensa-esp32-backend/65355>

– <https://lists.llvm.org/pipermail/llvm-dev/2019-March/130796.html>

Tooling

- Toolchain custom avec un fork maintenu de llvm et rustc
 - <https://github.com/esp-rs/rust-build.git>
 - <https://github.com/espressif/llvm-project>
 - <https://github.com/esp-rs/rust>
- Installation de la toolchain custom avec un outil dédié [espup](#) :

```
cargo install espup --git https://github.com/esp-rs/espup
espup install
. export-esp.sh
```

Tooling

Création d'un nouveau projet

- `cargo generate https://github.com/esp-rs/esp-template`

Avec tout ça on est prêts ?

Pas vraiment...

La gestion du WiFi et du Bluetooth, c'est compliqué...

FreeRTOS et esp-idf à la rescousse

FreeRTOS is a market-leading real-time operating system (RTOS) for microcontrollers and small microprocessors.

<https://freertos.org>

ESP-IDF is the official development framework for the ESP32, ESP32-S and ESP32-C Series SoCs.

<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/>

FreeRTOS et esp-idf à la rescousse

Application Example

GPIO output and input interrupt example: [peripherals/gpio/generic_gpio](#).

API Reference - Normal GPIO

Header File

- [components/driver/include/driver/gpio.h](#)

Functions 🔗

```
esp_err_t gpio_config(const gpio_config_t *pGPIOConfig)
```

GPIO common configuration.

Configure GPIO's Mode, pull-up, PullDown, IntrType

Parameters: **pGPIOConfig** – Pointer to GPIO configure struct

Returns:

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

FreeRTOS et esp-idf à la rescousse

- SPI/I2C/I2S
- Gestion des tasks
- Eventloop
- Bluetooth/BLE
- WiFi
- Timer
- ...



esp-idf-sys v0.31.10

Bindings for ESP-IDF (Espressif's IoT Development Framework)

[#sys](#) [#esp32](#) [#esp-idf](#) [#idf](#)

[Readme](#)

[71 Versions](#)

[Dependencies](#)

[Dependents](#)

Rust bindings for ESP-IDF

(Espressif's IoT Development Framework)

 [passing](#)  [esp-rs](#)

The ESP-IDF API in Rust, with support for each ESP chip (ESP32, ESP32S2, ESP32S3, ESP32C3, etc.) based on the Rust target.


For more information, check out:

- The [Rust on ESP Book](#)
- The [esp-idf-template](#) project
- The [esp-idf-svc](#) project
- The [esp-idf-hal](#) project
- The Rust for Xtensa toolchain
- The [Rust-with-STD demo](#) project

Table of contents

- [Build](#)
- [Features](#)
- [sdkconfig](#)
- [Build configuration](#)
- [Extra esp-idf components](#)
- [Conditional compilation](#)
- [More info](#)

Metadata

 7 days ago

 MIT or Apache-2.0


 42.4 kB

Install

Add the following line to your Cargo.toml file:

```
esp-idf-sys = "0.31.10"
```

Documentation

 esp-rs.github.io/esp-idf-sys

Repository

 github.com/esp-rs/esp-idf-sys

Owners

 [esp-rs/espessif](#)

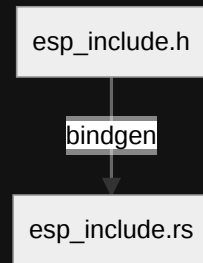
 [Scott Mabin](#)

 [sapis](#)

 [Ivan Markov](#)

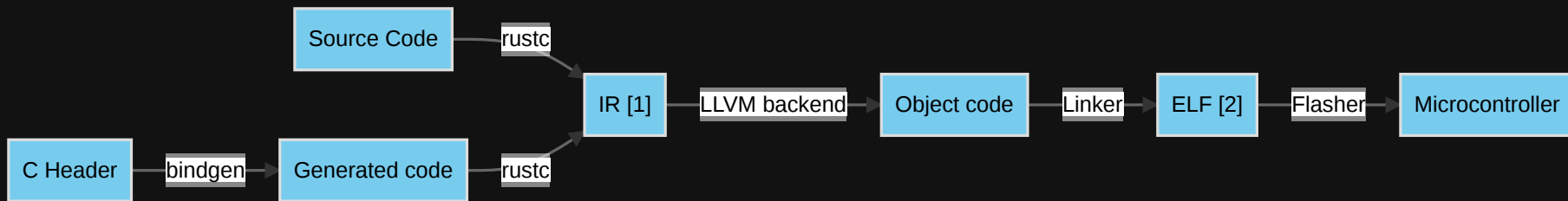
Tooling

- toolchains et crosscompilation
- bindgen



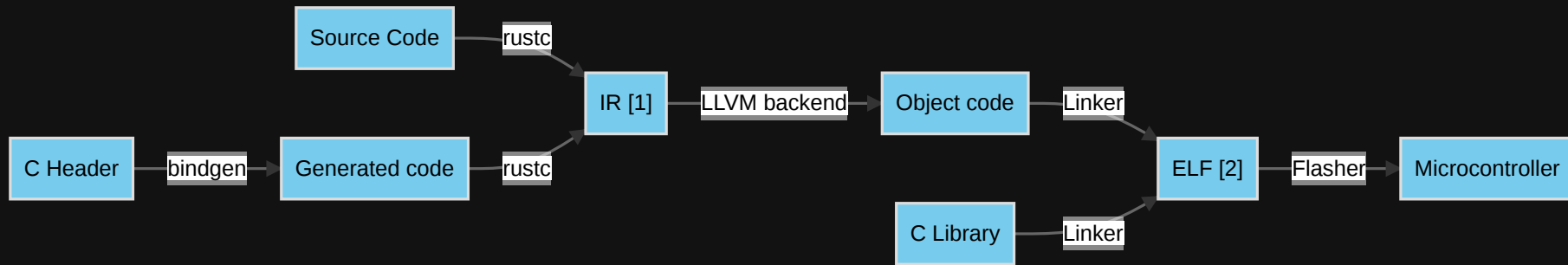
Tooling

- toolchains et crosscompilation
- bindgen



Tooling

- toolchains et crosscompilation
- bindgen
- build.rs



Tooling

- toolchains et crosscompilation
- bindgen
- build.rs
- esptool et esptool.py

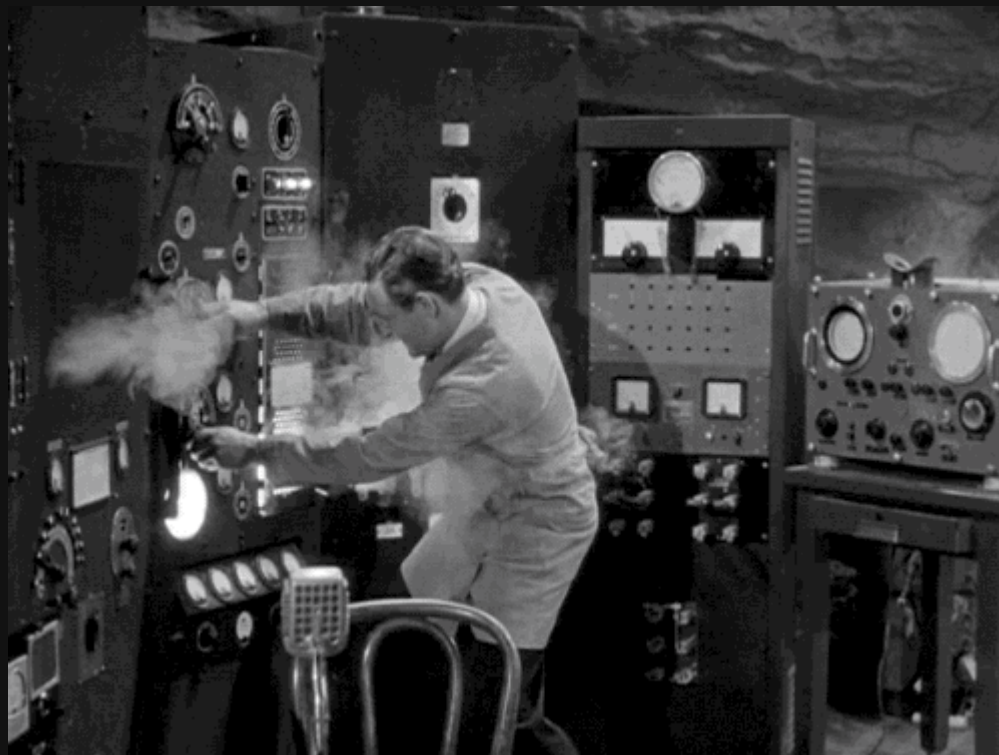
```
pyaillet@MacBook-Pro:~/Projets/esp32/twatch-rust/ttgo-display-idf-rs
> cargo esptool --monitor --speed 921600
Detected 4 serial ports. Ports which match a known common dev board are highlighted.

Serial port: /dev/cu.wchusbserial531C0180031
Connecting...

WARN setting baud rate higher than 115200 can cause issues.
Finished dev [optimized + debuginfo] target(s) in 0.22s
Chip type: ESP32 (revision 3)
Crystal frequency: 40MHz
Flash size: 4MB
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
MAC address: 94:b9:7e:d3:d9:98
App/part. size: 238688/1048576 bytes, 22.76%
[00:00:00] ##### 16/16 segment 0x1000
[00:00:00] ##### 1/1 segment 0x8000
[00:00:02] ##### 131/131 segment 0x10000
Flashing has completed!
Commands:
CTRL+R Reset chip
CTRL+C Exit

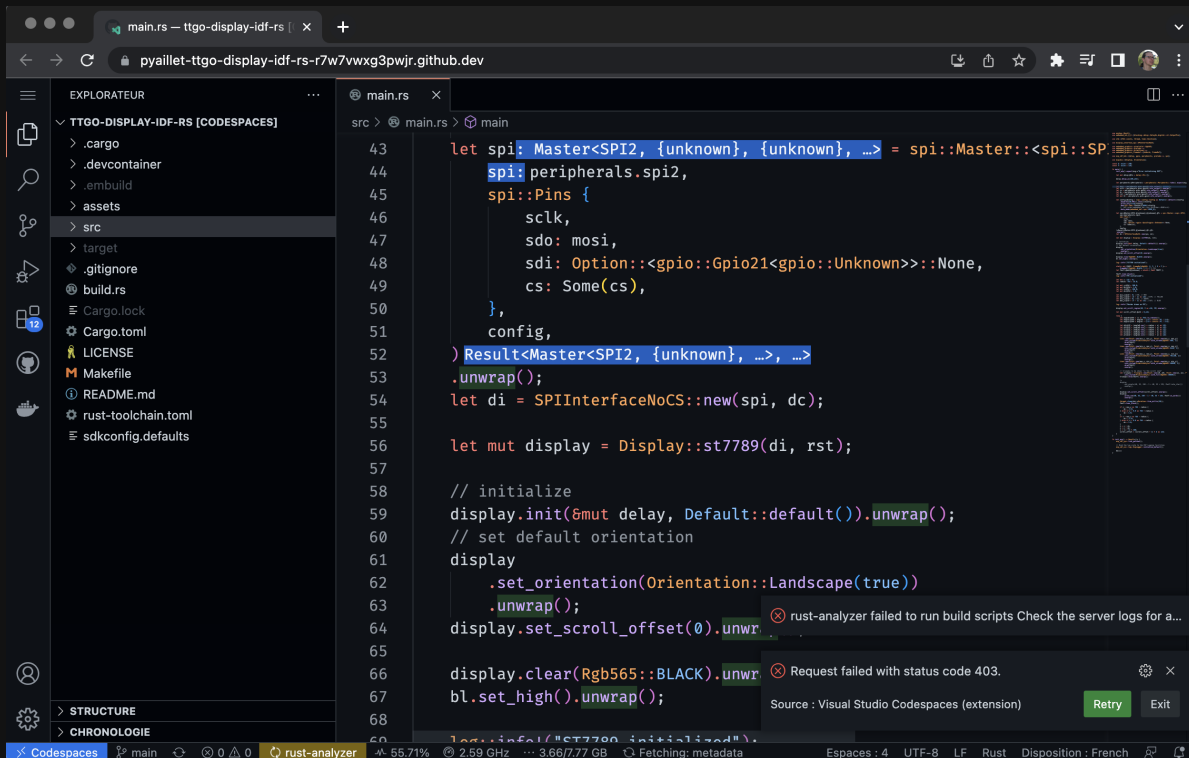
ets Jul 29 2019 12:21:46
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:6660
load:0x40078000,len:14848
0x40078000 - __udivmoddi4
at ??:??
ho 0 tail 12 room 4
load:0x40080400,len:3792
0x40080400 - _invalid_pc_placeholder
at ??:??
entry 0x40080694
0x40080694 - _iram_text_start
at ??:??
I (29) boot: ESP-IDF 08fa67f 2nd stage bootloader
I (29) boot: compile time 15:49:04
I (29) boot: chip revision: 3
I (32) boot_comm: chip revision: 3, min. bootloader chip revision: 0
I (39) boot.esp32: SPI Speed : 40MHz
I (44) boot.esp32: SPI Mode : DIO
I (48) boot.esp32: SPI Flash Size : 4MB
I (53) boot: Enabling RNG early entropy source...
I (58) boot: Partition Table:
I (62) boot: ## Label Usage Type ST Offset Length
I (69) boot: 0 nvs WiFi data 01 02 00009000 00006000
I (77) boot: 1 phy_init RF data 01 01 0000f000 00001000
I (84) boot: 2 factory factory app 00 00 00010000 00100000
I (92) boot: End of partition table
I (96) boot_comm: chip revision: 3, min. application chip revision: 0
I (103) esp_image: segment 0: naddr=00010020 vaddr=3f400020 size=0ae8ch ( 44684) map
```


Démo



Tooling

- toolchains et crosscompilation
- bindgen
- build.rs
- espflash et espmonitor
- devcontainer + esp-web-flash



```
43 let spi: Master<SPI2, {unknown}, {unknown}, ...> = spi::Master::<spi::SP
44 spi: peripherals.spi2,
45 spi::Pins {
46     sclk,
47     sdo: mosi,
48     sdi: Option::<gpio::Gpio21<gpio::Unknown>>::None,
49     cs: Some(cs),
50 },
51 config,
52 ) Result<Master<SPI2, {unknown}, ...>, ...>
53 .unwrap();
54 let di = SPIInterfaceNoCS::new(spi, dc);
55
56 let mut display = Display::st7789(di, rst);
57
58 // initialize
59 display.init(&mut delay, Default::default()).unwrap();
60 // set default orientation
61 display
62     .set_orientation(Orientation::Landscape(true))
63     .unwrap();
64 display.set_scroll_offset(0).unwr
65
66 display.clear(Rgb565::BLACK).unwr
67 bl.set_high().unwrap();
68
```

rust-analyzer failed to run build scripts Check the server logs for a...

Request failed with status code 403.

Source : Visual Studio Codespaces (extension) [Retry] [Exit]

Codespaces main rust-analyzer 55.71% 2.59 GHz 3.66/7.77 GB Fetching: metadata Espaces : 4 UTF-8 LF Rust Disposition : French

Tooling

- toolchains et crosscompilation
- bindgen
- build.rs
- espflash et espmonitor
- devcontainer + esp-web-flash
- wokwi

The screenshot shows the Wokwi web IDE interface. The left pane displays the Rust source code for a project named 'esp32c3-display'. The code includes headers for embedded graphics and SPI, defines a font, and sets up a display using the ILI9341 driver. The right pane shows a simulation of the hardware, including an ESP32C3 microcontroller and an ILI9341 display. The simulation output at the bottom shows the boot process and the initialization of the SPI LED driver.

```
25 use embedded_graphics::text::*;
26 use embedded_graphics::image::Image;
27 use embedded_graphics::geometry::*;
28
29 use profont::{PROFONT_24_POINT};
30 use embedded_graphics::draw_target::DrawTarget;
31
32 use esp_backtrace as _;
33 use riscv_rt;
34 use riscv_rt::entry;
35 use esp_println::println;
36
37 #[entry]
38 fn main() -> ! {
39     let peripherals = Peripherals::take().unwrap();
40     let mut system = peripherals.SYSTEM.split();
41     let mut clocks = ClockControl::boot_defaults(system.clock_control)
42
43     // Disable the RTC and TIMG watchdog timers
44     let mut rtc = Rtc::new(peripherals.RTC_CNTL);
45     let timer_group0 = TimerGroup::new(peripherals.TIMG0, &clocks);
46     let mut wdt0 = timer_group0.wdt;
47     let timer_group1 = TimerGroup::new(peripherals.TIMG1, &clocks);
48     let mut wdt1 = timer_group1.wdt;
49
50     rtc.rwdt.disable();
51     wdt0.disable();
52     wdt1.disable();
53
54     /* Some stuff for correct orientation and color on ILI9341 */
55     pub enum KalugaOrientation {
56         Portrait,
57         PortraitFlipped,
58         Landscape,
59         LandscapeVericallyFlipped,
60         LandscapeFlipped,
61     }
62
63     impl ILI9341 {
64         fn Mode for KalugaOrientation {
```

Simulation

ESP-ROM:esp32c3-apil-20210207
Build:Feb 7 2021
rst:0xc (RTC_SW_CPU_RST),boot:0x8 (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:1
load:0x3fcd6100,len:0x420
load:0x403ce000,len:0x90c
load:0x403d0000,len:0x2370
entry 0x403ce000
About to initialize the SPI LED driver ILI9341
Initialized



Next?

- Sur la montre...
 - BLE et connexion avec le téléphone
 - Détection d'activité
- Rust embedded
 - async/await avec embassy - <https://github.com/embassy-rs/embassy>
 - Autres cartes - Apache MyNewt + NimBLE - <https://mynewt.apache.org/>
 - RTIC - Real Time Interrupt driven Concurrency - <https://rtic.rs/1/book/en/>

Crédits

- Gentilhomme icônes créées par [Freepik - Flaticon](#)
- [#esp-rs:matrix.org](#)
- Developing Embedded Rust <https://www.youtube.com/watch?v=EughbCeVVxw>
- Rust sur de l'IOT ? <https://www.youtube.com/watch?v=pl60zczUXt0>
- Build scripts <https://www.youtube.com/watch?v=pePqWoTnSmQ>
- slidev <https://sli.dev/>
- Theme vuetiful <https://github.com/LinusBorg/slidev-theme-vuetiful>

Ressources / Références

- Embedded Rust
 - [Rust Embedded Workgroup](#)
 - [Embedded Rust book](#)
 - [Organisation esp-rs](#) (et tous les projets liés sur Github)
 - [Awesome embedded Rust](#)
- Espressif Rust
 - [Organisation esp-rs](#) (et tous les projets liés sur Github)
 - [esp-rs book](#)
 - [Ferrous systems Espressif Training](#)
 - [Awesome ESP Rust](#)
 - [Blog de Scott Mabin](#)
 - [Explication du Typestate pattern chez Cliff L. Biffle](#)
 - [Explication d'un principe identique au typestate pattern pour un builder chez Akanoa](#)
 - [Documentation officielle de esp-idf](#)
 - [FreeRTOS](#)
 - [Site officiel de la montre](#)

Projets

- Slides : <https://github.com/pyaillet/twatch-rust>
- Firmware de la montre : <https://github.com/pyaillet/twatch-idf-rs>
- Github : <https://github.com/pyaillet/>

