Hijacking radio joysticks

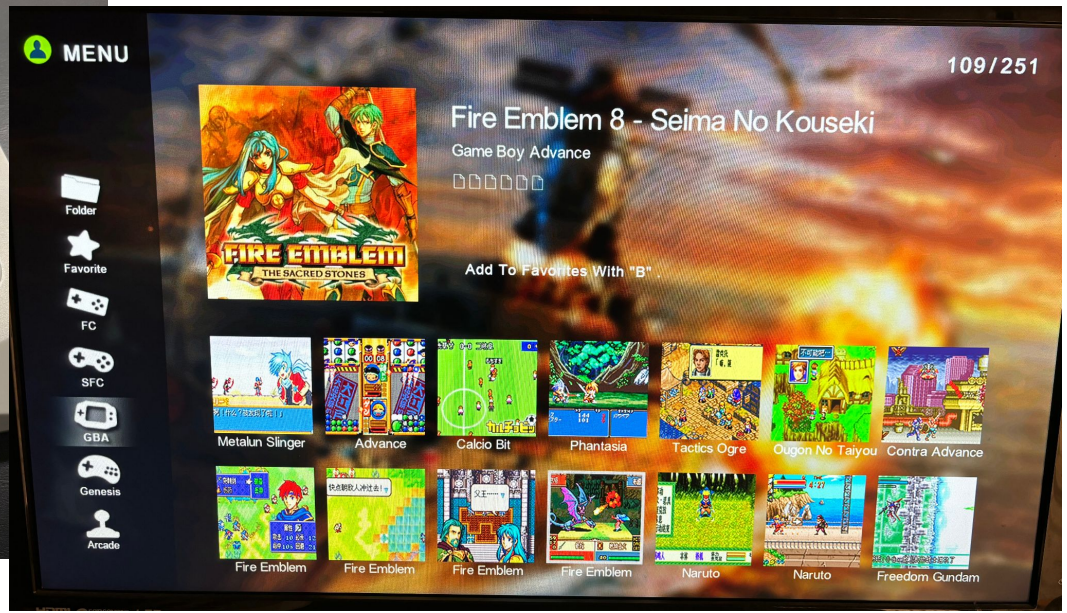
RP2040 + Rust = ❤️

What it will be about?

- Brief recap of one of my evening project: story about exploring and hacking a retro gaming console from Aliexpress

- Rust ecosystem overview for hobby-grade embedded development and my impressions of it

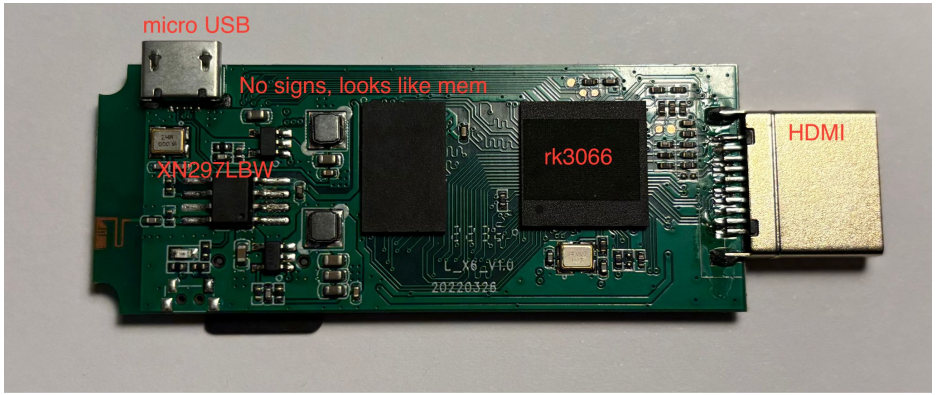
One day I’ve got a present

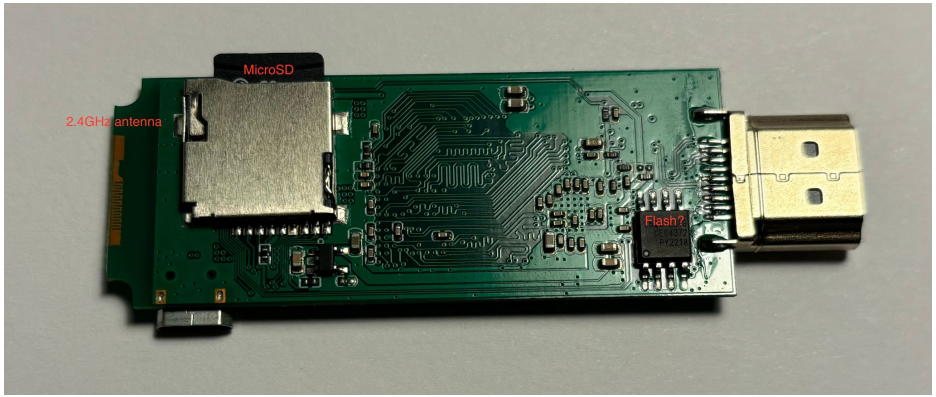


Project goals

- Connect joysticks to a PC
  - Windows and linux
  - Preferably without extra drivers
  - Keep joystick wireless
- Keep original device working
  - Avoid hardware modifications if possible

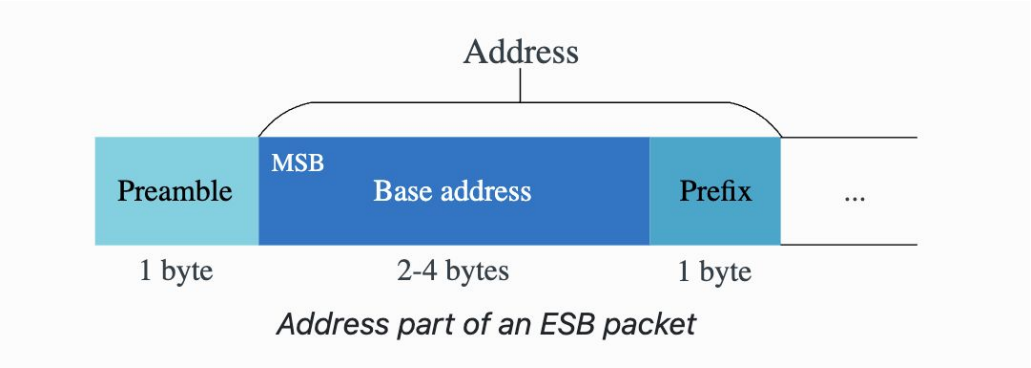
Exploring





- Seems a linux box with retroarch like environment
  - Rockchip rk3036, Cortex-A7
  - In theory, possible to execute arbitrary code
  - Cumbersome without additional IO interfaces
- No interfaces aside HDMI and SD
  - USB wired only for power
- XN297LBW transceiver
  - Uses SPI for communication and configuration
  - nRF24 clone (not compatible via air)
  - Datasheet exists
- Transmitter chip inside joysticks does not have any signs
  - But it seems to be XN297L in different packaging

nRF24L and its clones



Address part of an ESB packet

- nRF24L - Nordic Semiconductor's chip for wireless communication
  - Seems common across Arduino community for organizing wireless communication
  - Uses its own L2 protocol - Enhanced ShockBurst (ESB)
  - Supports star network topology with typically one receiver and multiple transmitters
  - Reasonably low latency
- A lot of clones on market right now
  - XN297, RFM73, RFM75, LCX24G
  - Common across cheap rc devices, like drones, rc cars and so on
  - Uses ESB variations
  - Not fully compatible with nRF24L
  - XN297 specifically applies scrambling to addresses
  - Clones can have additional features
  - https://github.com/roboremo/ChiNRF

No luck with nRF24L :(

Despite of compatibility on SPI side, does not compatible via air due to different preambles and address scrambling

What is SPI?

- Serial Peripheral Interface
- Short distance wired communication protocol
- Synchronous
- Typically 4-wires, but 3-wires variants exist
- Full-duplex (not for 3 wires)
- Pretty mature and well known

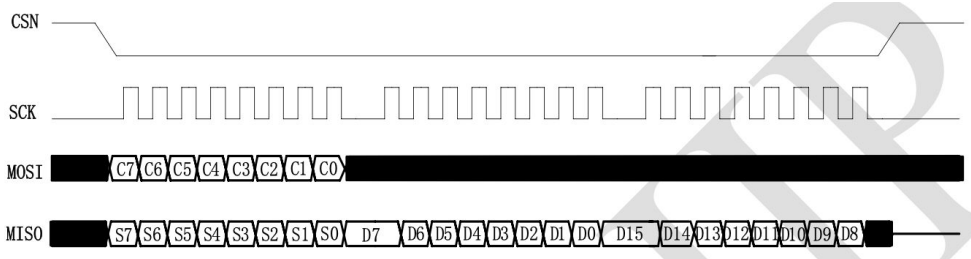


Figure 7-1 SPI read operation

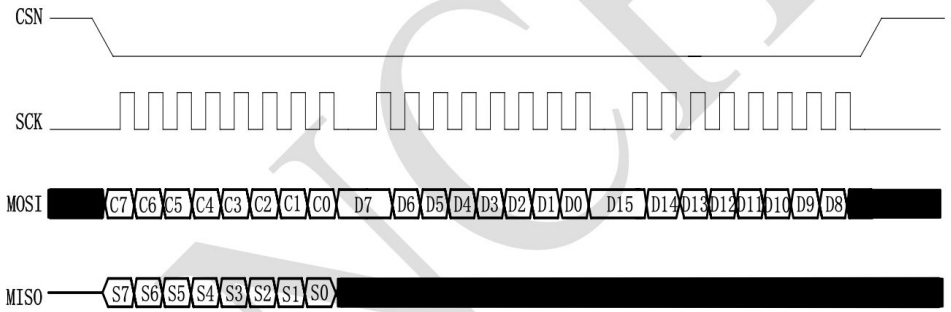
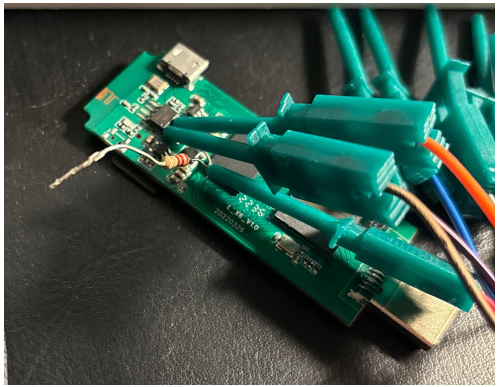
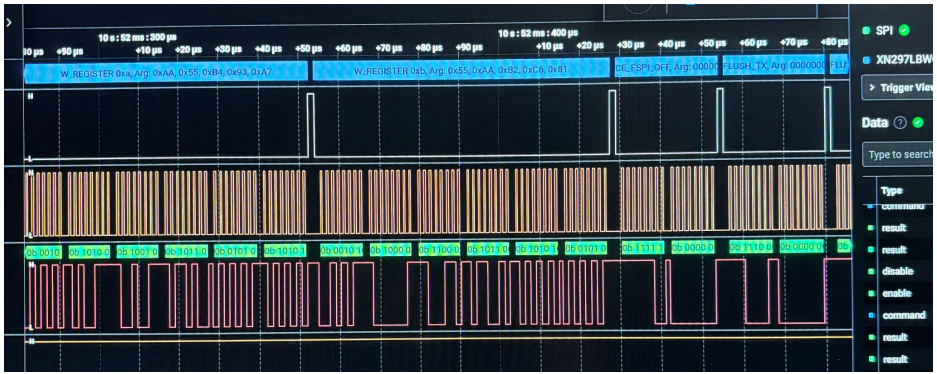


Figure 7-2 SPI write operation

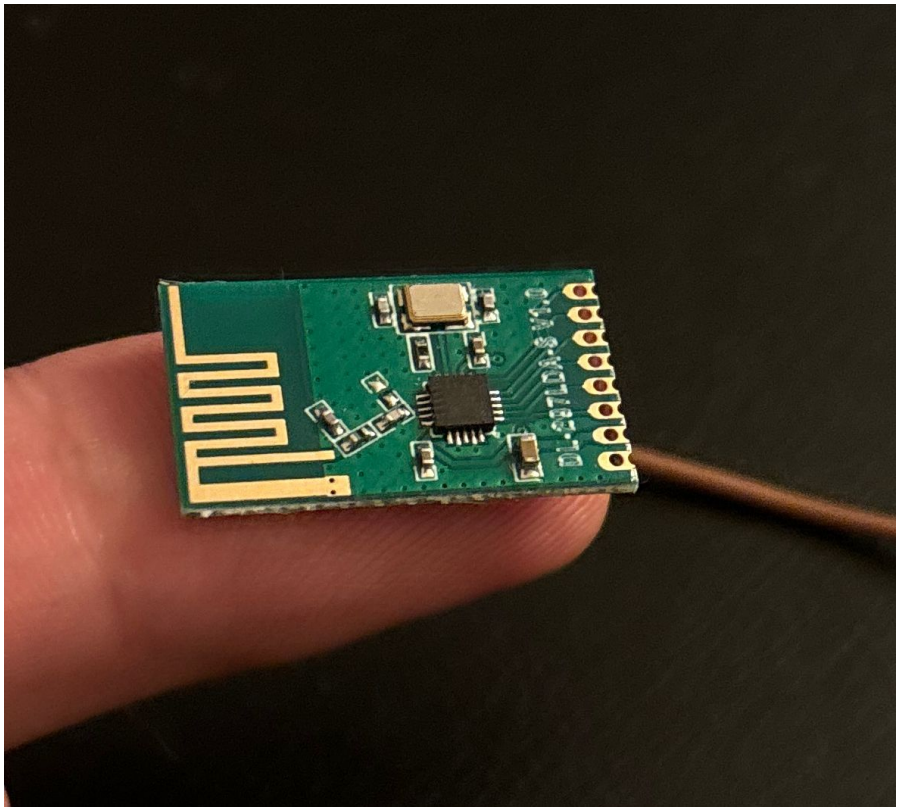
Go a bit deeper and do some decoding





- Logic analyzer to the rescue
- Logic2 has plugins system
- Raw SPI transmissions can be decoded with a bit of Python code with a help of a datasheet
- Sometimes datasheets are incomplete :(

XN297 + Arduino prototype



- Got XN297 module from the depths of Aliexpress
- Entire module costs less than euro
- Built first prototype with Arduino Uno using the exact parameters from the SPI communication dump
  - Didn’t manage to fully understand some parts of communication, some registers are not documented even in chinese datasheet
  - Some register values are applicable only to 3-wire spi model
- First successful receiving!

RP2040, Picoprobe

- A lot of ready-to-go and quite **cheap** boards
- Small size
- RP2040 - Dual core ARM Cortex M0
- Some of boards are on-chip debugging ready with a bit of help of external debugger

- Another RP Zero can be used as debugger adapter
- Nothing extra aside wires
- https://github.com/raspberrypi/debugprobe
- OpenOCD for remote GBD sessions: https://openocd.org/
- Debugging with OpenOCD works well with Rust

Rust?

- Guess everyone heard of it already
  - Rich type system simplifies development
  - Compile time static analysis is really helpful
- Becomes big, gain maturity
  - Linux kernel project accepted it as a second language for writing modules
  - Big tech companies are slowly but surely adopting it
- Great community
  - Feels it’s pretty much open source/github centric
- Doing pure rust on embedded devices is possible
  - ARM Cortex M are covered pretty well
  - A lot of materials and examples online
- It’s fun :)

Why is it so cool? Cargo!

- Integral part of the Rust ecosystem
- Best tool I’ve seen for dependency management and build manipulation
- Extensible with plugins, some of them are great (`cargo clippy` for example)
- Good dependency manager for a system language? Sounds tremendous to me!

Current progress of rust-embedded community

- Working group established: https://github.com/rust-embedded/wg
- Things are developing rapidly
- People seems keen to keep maintaining things, I didn’t run into abandoned repos so far
- Multiple frameworks are actively developing. Notable repos:
  - https://github.com/embassy-rs/embassy
  - https://github.com/rtic-rs/rtic
- A lot of things are on quite early stages and considered unstable
- No official support from vendors

embedded-hal

https://github.com/rust-embedded/embedded-hal

- HAL - stands for Hardware Abstraction layer
- Foundation for build hardware agnostic drivers
- HAL implementation crates provides wrappers around peripherals

```
let spi_miso = pins.gp12.into_function::<hal::gpio::FunctionSpi>();
let spi_sclk = pins.gp10.into_function::<hal::gpio::FunctionSpi>();
let spi_mosi = pins.gp11.into_function::<hal::gpio::FunctionSpi>();

let mut spi_csn = pins.gp5.into_push_pull_output();
spi_csn.set_high().unwrap();

let mut spi_ce = pins.gp3.into_push_pull_output();
spi_ce.set_low().unwrap();

let spi = hal::spi::Spi::<_, _, _, 8>::new(pac.SPI1, (spi_mosi, spi_miso, spi_sclk)).init(
    &mut pac.RESETS,
    clocks.peripheral_clock.freq(),
    4.MHz(),
    embedded_hal::spi::MODE_0,
);
```

BSPs

- BSP - board specific package
- Wraps peripherals even further for a specific board for convenience
  - Can contain board specific drivers as well
- Exists for a lot of popular boards
  - Depends on MCU, but most popular seems covered quite well

```
1  #![no_main]
2  #![no_std]
3
4  extern crate panic_halt as _;
5
6  use cortex_m_rt::entry;
7  use microbit::hal::prelude::*;
8  use microbit::Board;
9
10 #[entry]
11 fn main() -> ! {
12     let mut board = Board::take().unwrap();
13
14     board.display_pins.col1.set_low().unwrap();
15     board.display_pins.row1.set_high().unwrap();
16
17     loop {}
18 }
```

USB device stack?

Libraries exists.

- https://github.com/rust-embedded-community/usb-device
  - A lot of features are missing
  - No host mode
  - A bit of hacking might be necessary
- https://github.com/embassy-rs/embassy
  - Feature-rich asynchronous framework
  - Has USB stack implementation
  - Has its own HAL which causes extra effort with `embedded-hal` based drivers

Can I do it with XYZ microcontroller?

Highly likely!

A lot of implementation of traits from `embedded-hal` for various MCUs

- https://github.com/esp-rs - ESP32
- https://github.com/stm32-rs - STM32
- https://github.com/rp-rs - Raspberry Pi
- https://github.com/Rahix/avr-device - AVR / Arduino

Is it production ready?

Rather No, despite it's well suitable for hobby-grade development. Questionable for production.

- Vast majority of libraries around did not hit 1.0+ versions and considered unstable
- Things are changing rapidly, breaking changes happens in foundational crates
  - embedded-hal 1.0 for example
- Almost no support from major vendors
  - Everything came from the community so far
- No language standardization (like ISO/IEC)
  - Might be a big issue for critical applications

Materials

- nRF24L
  - https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/protocols/esb/index.html
  - https://github.com/roboremo/ChiNRF - Great repo for diving deeper into nRF24L clones, a bit outdated, but has a lot of useful info in code

- Rust: Maintained/curated by `rust-embedded` team
  - https://docs.rust-embedded.org/book/ - Rust embedded book
  - https://docs.rust-embedded.org/embedonomicon/ - The Embedonomicon
  - https://github.com/rust-embedded/awesome-embedded-rust

Final result

Sources and spi captures:

https://github.com/lobziik/2040UsbJoysticksReceiver

And the most important gaming device function, of course

Sources and spi captures:

https://github.com/lobziik/2040UsbJoysticksReceiver