Getting started with embedded Rust

microcontrollers should be fun, actually

2024-10-14

Sandro Stikić

Table of Contents

1 Preamble

Goals

- Dip our toes into the embedded world
- Setup our breadboard
- Setup our Rust toolchain
- Program our microcontroller
- Debug our program

2 Hardware

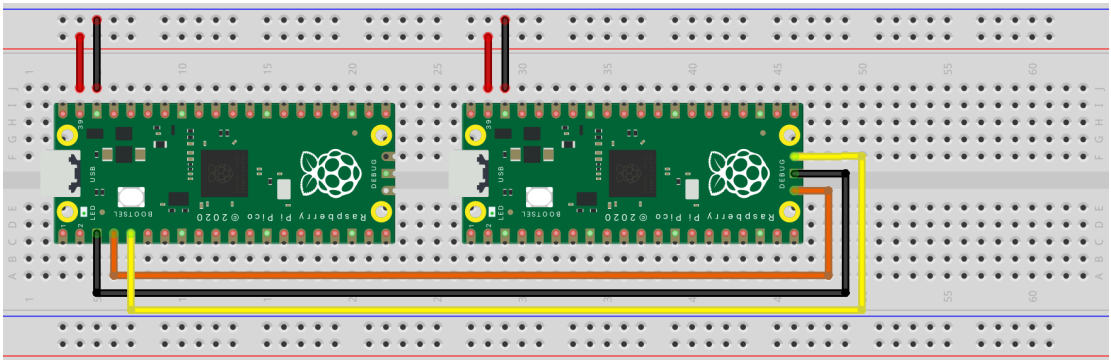
What you'll need

- 2x Raspberry Pi Pico microcontroller with presoldered headers
  - This is the microcontroller we'll be programming—cheap, abundant, and powerful!
- Breadboard
  - Used to quickly prototype electronics projects
- Jumper wire kit
  - Used to make connections in our breadboard
- DuPont wire kit (male to male, male to female, female to female)
  - Used to make connections in our breadboard (cursed!)
- Micro USB cable (power *and* data!)
  - Used to power, debug, and flash our microcontroller

Wiring it up

- We're using two Picos in our breadboard
  - The probe Pico is responsible for flashing and debugging and target Pico
- Connect the debug pins:
  - Connect `GP2` on the probe to `SWCLK` on the target
  - Connect `GP3` on the probe to `SWDIO` on the target
  - Connect `GND` from the probe to the target
- Power the target via the probe:
  - Connect `VSYS` on both the probe and target to the breadboard power rail
  - Connect `GND` on both the probe and target to the breadboard ground rail
  - Connect your computer to the probe via Micro USB

Circuit diagram



- You **must** use a common ground in your circuit
- Powering another Pico via `VSYS` is fine, powering a thirsty peripheral via `VSYS` is not

Debug probe firmware

- Download the official debug probe firmware
- Press and hold the `BOOTSEL` button on the probe
- While holding `BOOTSEL` down, connect the probe to your computer via Micro USB
  - Ensure the USB cable supports power *and* data
- You should see a new storage device mount called `RPI-RP2`, you can now release `BOOTSEL`
- Copy `picoprobe.uf2` to the `RPI-RP2` storage device
- The `RPI-RP2` device should automatically unmount
- Check the LED on the probe is lit, if so, the flashing succeeded!

3 Software

Library or framework?

- There's two main options for working with the Pico in Rust:
  - rp-hal
    - Abstraction over the RP2040 microcontroller, more akin to a library
    - Your code is "tied" to the HAL
    - Relatively easy to setup
    - Supports stable *or* nightly Rust
  - embassy
    - Embedded application framework
    - Lots of board support, more abstract, easier to port[1], async support!
    - Painful setup[2], the interface is unstable
    - Only supports nightly Rust

[1] In practice, the nature of embedded devices means porting code is not straightforward

[2] The setup docs are incomplete and using embassy requires patching crates.io

Setup

- Install Nix:

```
curl --proto '=https' --tlsv1.2 -sSf -L https://install.determinate.
systems/nix | sh -s -- install
nix profile install nixpkgs#direnv
echo 'eval "$(direnv hook bash)"' >> ~/.bashrc
```

- Clone the template[3]:

```
git clone https://github.com/opeik/rp2040-project-template
cd rp2040-project-template
# nix environment loads automatically, wow!
```

[3] My fork adds Nix support and fixes the CPU halting when you start debugging

Flash it!

```
❯ cargo run
    Finished `dev` profile [optimized + debuginfo] target(s) in 0.17s
     Running `probe-rs ...`
      Erasing ✔
  Programming ✔
INFO  Program start
└─ rp2040_project_template::__cortex_m_rt_main @ src/main.rs:27
INFO  on!
└─ rp2040_project_template::__cortex_m_rt_main @ src/main.rs:68
INFO  off!
└─ rp2040_project_template::__cortex_m_rt_main @ src/main.rs:71
```

Debug it!

- Open VSCode from the Nix shell:

```
code .
```

- Press F5, or click Run and Debug > Start Debugging
- That's it!