Running Rust on an FPGA

Outline

- What is an FPGA
- What is a soft CPU
- FPGA Toolchain
- SVD files
- svd2rust
- Writing a hardware abstraction layer
- Interacting with your embedded code

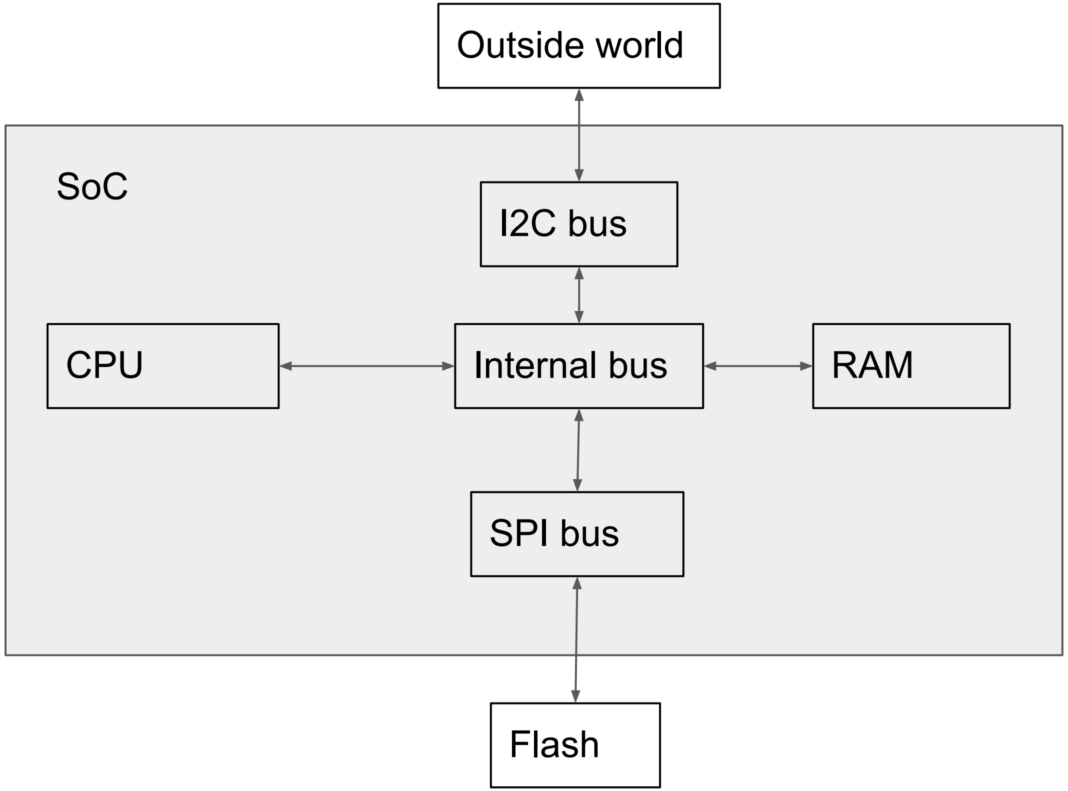
What is an FPGA?

- Field-Programmable Gate Array
- A collection of logic gates that can be programmed in the “field”
- Lets you create and run complex digital circuits without getting a custom chip made
- Modern FPGAs consist of a range of components:
  - Basic
    - Lookup tables - take a few (e.g. 4) bits and produce a 1 bit output
    - Flip-flops - store a single bit of information until the next clock cycle
    - IO - lets you control or read the data pins on the chip
  - Advanced
    - RAM - bulk storage - more efficient than using lots of flip-flops
    - Multiply-accumulate - more efficient than building your own out of lookup tables

Soft CPUs

- A software-defined CPU
- A CPU designed to run on an FPGA
- Often use RISC-V, since licensing is easier
- If you find a bug in your CPU, it can be fixed
- You can extend your CPU with your own custom instructions

SoC - System on a Chip



Open source FPGA toolchain

- LiteX - provides a lot of common components and makes them work together. e.g. soft CPU, busses, caches, ethernet, flash storage interface etc
- Amaranth - lets you write Python code that defines a digital design.
- Yosys - performs “synthesis” - converts your design into a network of components that are available on your particular FPGA
- NextPNR - performs place-and-route - tries to fit your design into the actual FPGA and checks that the design can operate at the requested frequency
- FPGA-specific bitstream generator - produces a binary that the FPGA can load

SVD files

- XML that documents the memory layout of a device. e.g.
    - Ethernet controller starts at address 0x1234_0000
        - Offset 0x00 is the control register
            - Bit 0 enables ethernet
            - Bit 1 enables high speed mode
- Chip vendors often supply SVD files for their chips
- LiteX can generate an SVD file for us based on the design we ask it to make

svd2rust

- Takes an SVD file and generates Rust code

Without svd2rust

```
const LED_ADDRESS: usize = 0x1234_0000;
unsafe {
    let value = core::ptr::read_volatile(
        LED_ADDRESS as *mut u32);
    core::ptr::write_volatile(
        LED_ADDRESS as *mut u32, value | 1);
};
```

With svd2rust

```
p.LEDS.out.modify(|_, w| w.led1().set_bit());
```

Building a hardware abstraction layer (HAL)

- Wrap the code that svd2rust generated in a nicer interface
- Use traits so that we can write tests that pass in a fake implementation
- Then we can write
  - `led1.turn_on();`

Interacting with your embedded code

- Build a host-side command line tool that:
  - Compiles your firmware
  - Flashes your firmware to the device
  - Lets you control your device
  - Lets you see panic messages
  - Lets you see diagnostic logs
- This tool can use whatever interface(s) you have available. e.g.:
  - JTAG/SWD
  - UART
  - I2C
  - OpenOCD

Resources

- CFU playground - helps with experimenting with custom instructions on FPGAs
- LiteX
- Amaranth
- hps-firmware - a project in ChromeOS that the author has worked on that uses all this stuff
- Contact @davidlattimore on github