F28HS Hardware-Software Interface: Systems Programming

Hans-Wolfgang Loidl

School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh

Semester 2 — 2023/24

[0]No proprietary software has been used in producing these slides

Outline

Lecture 9.
Miscellaneous Topics

Bare-metal programming

- **Bare-metal programming** means “programming directly on the hardware”, i.e. on a system that doesn’t run an operating system.
- This is the most common scenario for embedded systems programming.
- In this course we used Raspbian on the RPi2 mainly for **convenience** (tool support etc)
- Embedded systems in industry usage are often too small to run any OS
- For time-critical operations you don’t want an OS because in order to meet **real-time** constraints.

What’s different?

A lot:

- You have to control the boot process yourself
- You have to manage all aspects of the hardware directly:
  - memory (no virtual memory!)
  - external devices
- You need to produce stand-alone executables, i.e. no dynamically linked libraries
- You typically need to cross-compile your code

What are the advantages?

- You have direct control over the hardware:
  - For our LED etc examples, you **don't need** `mmap` to access the devices, rather you directly write to the hardware registers.
  - You can access aspects of the hardware that might not be accessible otherwise.
- Better suited for real-time constraints: no OS overhead, predictable performance
- Very small code size of the entire application
- Typically lower energy consumption

How does the application code differ?

Looking at our example code from the course

- No `mmap` is needed to access the GPIO pins
- You can't use external libraries: everything must be part of the application
- This means that in general you need to write your own device drivers for external devices such as a monitor
- The code typically needs to be cross-compiled, i.e. the machine that you are **compiling on** is different from the machine that you are **compiling for**.

And of course there are a lot of differences in terms of usability.

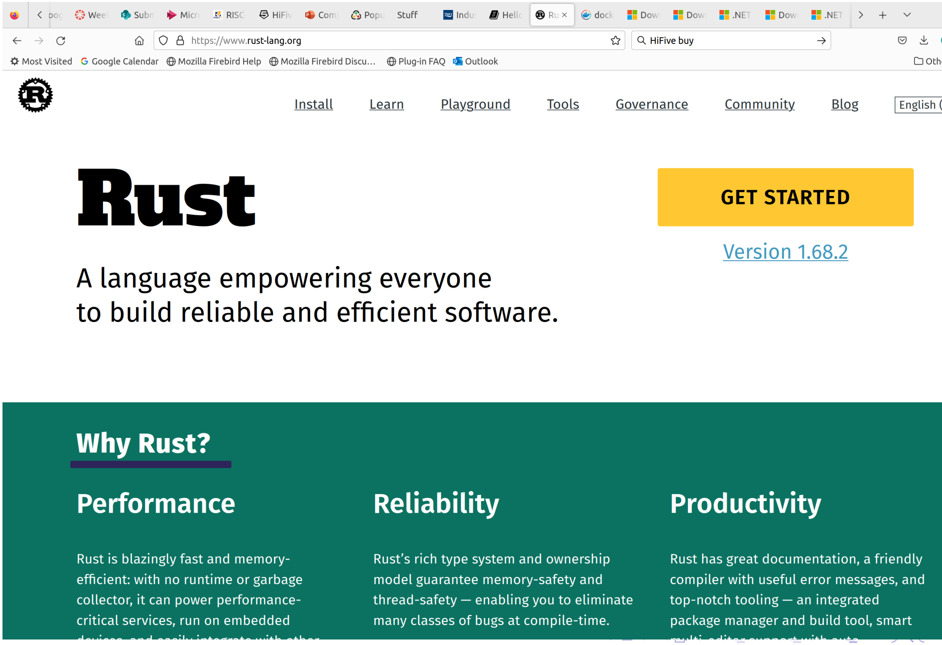
Further Reading & Deeper Hacking

- *"Embedded Linux"*, by Jürgen Quade (Textbook on embedded systems programming, using a bare-metal approach)
- Baking Pi, by Alex Chadwick (a course on bare-metal programming on the Rasbperry Pi at Cambridge University (only for RPi1))
- Valvers: Bare Metal Programming in C

Rust: an alternative systems programming language

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

[0] Rust



Rust

A language empowering everyone
to build reliable and efficient software.

Why Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded

Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart

Rust History

- Rust has been a very active topic in the programming languages community since at least 2013
- Rust builds on deep concepts such as:
  - region-based memory management (in Cyclone)
  - uniqueness types (in Clean)
- Rust has a very active community and very good tool support
- Since Oct 2022 (Open Source Summit) it is a **first-class language in the Linux kernel**

Rust Highlights

- A safer version of C as a low-level systems programming language
- Rust is to C, what Typescript is to Javascript
- Delivers high performance and low level control, with more memory safety
- Introduces the notion of **ownership** on pointers
- Three pillars of Rust:
  - Code correctness
  - Performance
  - Memory Safety

See Software Engineering master-class by Nathanel Brown, March 2023.

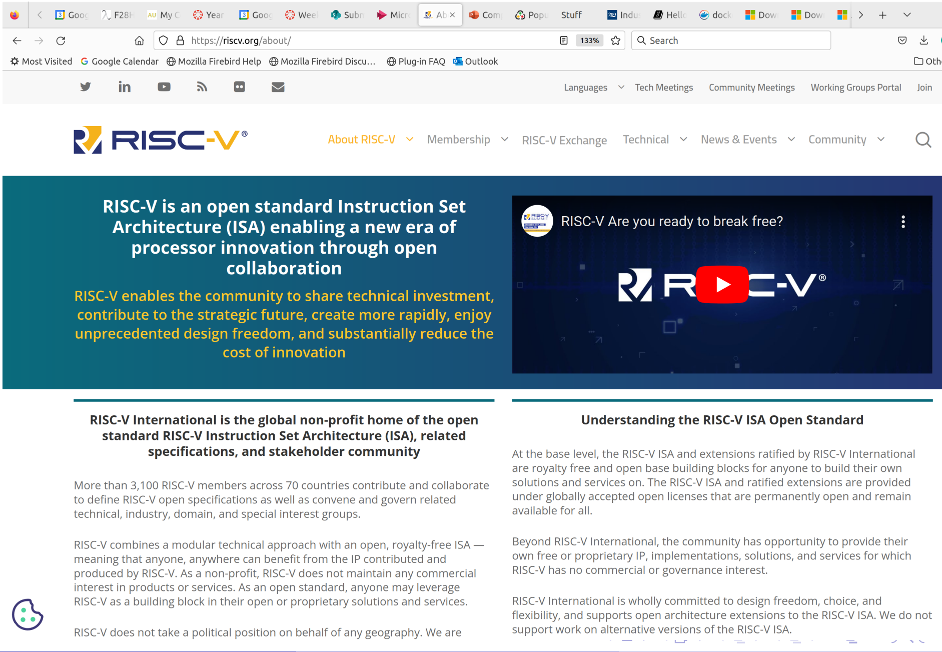
Rust Features

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference
- minimal runtime
- efficient C bindings

Risc V

- Risc V is a **open hardware standard** based on a RISC design
- Originally developed around 2011 at the University of Berkeley
- Item aims to provide non-proprietary chip design, unlike ARM or Intel designs
- It has a very active community, with start-up companies building on that design
- It enables in hardware design what Linux has enabled in software/OS design
- **Risc V is the future of hardware design!**

More info at https://riscv.org/.



RISC-V is an open standard Instruction Set Architecture (ISA) enabling a new era of processor innovation through open collaboration

RISC-V enables the community to share technical investment, contribute to the strategic future, create more rapidly, enjoy unprecedented design freedom, and substantially reduce the cost of innovation

RISC-V International is the global non-profit home of the open standard RISC-V Instruction Set Architecture (ISA), related specifications, and stakeholder community

More than 3,100 RISC-V members across 70 countries contribute and collaborate to define RISC-V open specifications as well as convene and govern related technical, industry, domain, and special interest groups.

RISC-V combines a modular technical approach with an open, royalty-free ISA — meaning that anyone, anywhere can benefit from the IP contributed and produced by RISC-V. As a non-profit, RISC-V does not maintain any commercial interest in products or services. As an open standard, anyone may leverage RISC-V as a building block in their open or proprietary solutions and services.

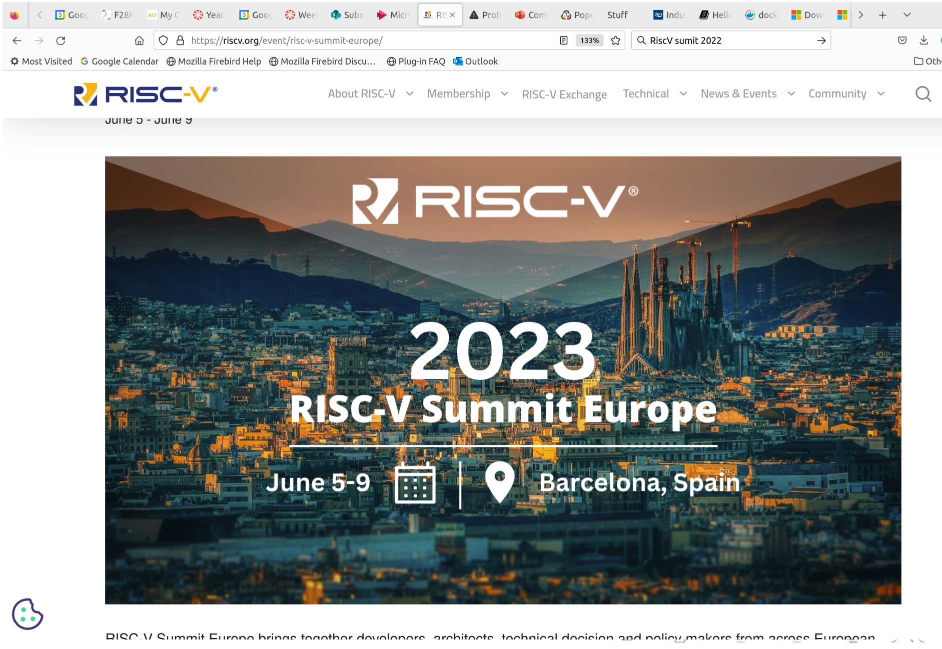
RISC-V does not take a political position on behalf of any geography. We are

Understanding the RISC-V ISA Open Standard

At the base level, the RISC-V ISA and extensions ratified by RISC-V International are royalty free and open base building blocks for anyone to build their own solutions and services on. The RISC-V ISA and ratified extensions are provided under globally accepted open licenses that are permanently open and remain available for all.

Beyond RISC-V International, the community has opportunity to provide their own free or proprietary IP, implementations, solutions, and services for which RISC-V has no commercial or governance interest.

RISC-V International is wholly committed to design freedom, choice, and flexibility, and supports open architecture extensions to the RISC-V ISA. We do not support work on alternative versions of the RISC-V ISA.



June 5 - June 9

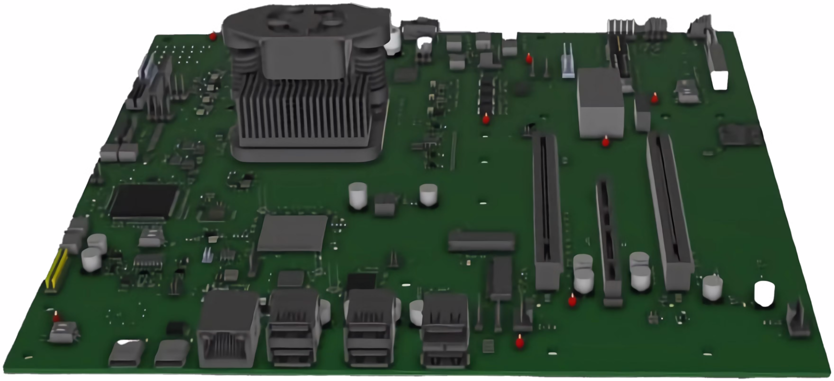
RISC-V Summit Europe brings together developers, architects, technical decision and policy makers from across European

RISC-V Users

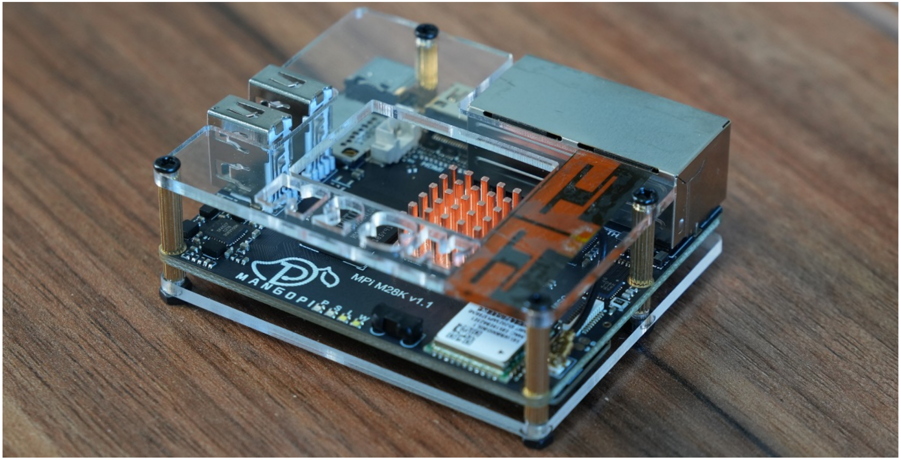
RPi alternatives:

- **HiFive Pro P550** is another commercially available single-board-computer by HiFive
- includes a BXE-4-32-MC1 graphics unit, to support proper desktop functionality
- supports AES encoding for security
- the usual connectors: HDMI-2.0, 4$\times$ USB-3.0, Ethernet
- GPIO pins similar to the RPi
- URL: https://www.sifive.com/boards/hifive-pro-p550
- **Mango-Pi**, based on an Alibaba Allwinner D1 chip
- this is targeting the Smartphone market, with an Android port of RISC-V
- URL: https://mangopi.org/

HiFive Pro P550



Mango Pi



Unboxing and Review

- Wi-Fi Sheep Tech Channel
- NicoD's SBCs
- Chandler Klüser
- Armbian + MangoPi MQ Pro is GREAT!
- Article about using a Raspberry Pi as serial console for the MangoPi MQ-Pro
- MangoPi MQ-PRO Review: RISC-V Raspberry Pi Zero Alternative?
- MangoPi MQ Pro D1 Ubuntu (P)review

RISC-V Users

Many **embedded devices** in the Internet-of-Things (IoT) domain:

- Seagate is switching to RISC-V design for intelligent hard-disks
- used as controllers for future generations of hard-disk up to 40 TB
- also used as devices for **edge computing**, involving (small) local computations and encryption before communication
- major advantage is the availability of a **software ecosystem** for RISC-V
- these are low-cost devices in the range of $1
- also: Open Titan project, for secure booting through a root-of-trust

Internet of Things

- The amount of processors used in all kinds of settings is increasing rapidly.
- Examples are “smart homes” with configurable/programmable devices such as smart TVs etc
- These typically use small, embedded devices
- These devices want to exchange data, e.g. to monitor the environment and react to changes
- Therefore, these systems are inter-connected, building an **Internet of Things**
- These systems increasingly use a full operating system underneath
- Thus, a **RPi 4 or 5 running Raspberry OS is a good case study**

OS choices for the Internet of Things

- Rapberry OS, while useful as an interactive OS, comes with a lot of unnecessary packages if it should be used on one of these networked, embedded devices.
- Smaller, configurable Linux versions are often a better choice, e.g. Arch Linux (also available for RPi).
- These reduce the resource consumption of the system, and improve maintainability.
- Several new[1] OS's target this market: for example **MinocaOS**

[1] There are also several old OS's that fit this characterisation: see Minix and RISC OS.

Main features of MinocaOS

- MinocaOS is a completely new OS, matching standard interfaces such as POSIX.
- MinocaOS is advertised as: `Modular`, `Lean`, `Flexible`
- MinocaOS supports RPi2/3 in 2 different images that can be downloaded
- There is no 64-bit support available yet[2]
- MinocaOS is also provided as a Quemu-based virtual machine, for experimentation on a laptop
- MinocaOS has a very small resource footprint, and works well even on older RPi1's
- MinocaOS has good hardware support and fairly good tool support

[2] See the slides at the end for a link on how to build your own 64-bit kernel on an RPi3

MinocaOS

Some notable features of MinocaOS are:

- Most command-line tools are based on GNU versions: `bash`, `ls`, `cat`, `chmod`, `nano` (use `--help` to get info)
- It uses package management similar to Debian-based systems (`opkg` as package manager; packages have extension `.ipkg`)
- The list of available packages and repos can be edited in `/var/opkg-lists/`
- **No** graphical user interface at the moment (not necessary for IoT context)

A Guided Tour is available on the MinocaOS web page.

[2] Material from Raspberry Pi Geek 04/2017

MinocaOS

Some notable features of MinocaOS are:

- Most command-line tools are based on GNU versions: `bash`, `ls`, `cat`, `chmod`, `nano` (use `--help` to get info)
- It uses package management similar to Debian-based systems (`opkg` as package manager; packages have extension `.ipkg`)
- The list of available packages and repos can be edited in `/var/opkg-lists/`
- **No** graphical user interface at the moment (not necessary for IoT context)

A Guided Tour is available on the MinocaOS web page.

[2] Material from Raspberry Pi Geek 04/2017

UBOS: easy configuration

- UBOS is a Linux distribution for easy management of several web services on an Rpi.
- Very flexible, being based on Arch Linux
- Features (as advertised):
  - With UBOS, web applications can be installed, and fully configured with a single command.
  - UBOS fully automates app management at virtual hosts
  - UBOS pre-installs and pre-configures networking and other infrastructure.
  - Systems that have two Ethernet interfaces can be turned into a home router/gateway with a single command.
  - UBOS can backup or restore all, or any subset of installed applications on a device
  - UBOS uses a rolling-release development model
  - UBOS itself is all free/libre and open software.

3

[3] Material from Raspberry Pi Geek 04/2017

Compiling an 64-bit kernel for RPi3

In this course, we used RPi 2/3 or RPi 4 running in 32-bit mode.
RPi 3 onwards is a 64-bit architecture.
To fully exploit this architecture, the kernel needs to be compiled in 64-bit mode.
A detailed discussion on how to build a 64-bit kernel on a Rasberry Pi 3 is given in the Raspberry Pi Geek 04/2017.
A pre-pared 64-bit image for the RasPi 3 is here

Summary

- There are many more application scenarios for using RPi-like devices
- **IoT** is one huge application domain
- Programming IoT devices requires low-level systems programming skills
- **Rust** is modern systems programming language providing more memory security than C
- **Risc-V** is a hot area of architecture design (**open hardware**)
- To keep up-to-date, follow these directions beyond your studies!