Porting Debian to the RISC-V architecture

Tales from a long quest.

Karsten Merker <merker@debian.org>

February 2, 2019

What is RISC-V?

- **RISC-V** is a freely licensed **CPU Instruction Set Architecture** (ISA) originating from the University of California, Berkeley (UCB).
  - Everybody is free to implement processors based on the RISC-V ISA without royalty payments for using the ISA.
  - The ISA specification is available under CC-BY license.
  - Conformance to the specification is secured via trademarks. Only implementations that fully conform to the specification may call themselves “RISC-V”.
- Available in 32bit, 64bit and (not yet fully specified) 128bit flavours
- Designed to scale from microcontrollers to supercomputers by using a modular design.
- The ISA design aims at using only techniques that are patent-free, although there is no legal guarantee for that.

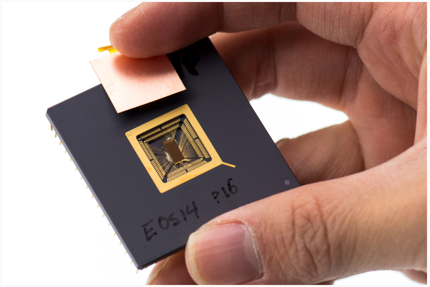
What is RISC-V not?

RISC-V

- is **not** a microprocessor **implementation**.
- **doesn't require implementations to be freely licensed**, i.e. there can be (and are) non-free RISC-V implementations.
- **doesn't have an open stewardship in the way open-source projects define "open", only in the way the semiconductor industry defines "open":**
  - The specifications are released under a DFSG-free license (CC-BY), but contributing to standard working groups and access to pre-release documents requires a RISC-V foundation membership.
  - The **RISC-V foundation** is a US 501(c)(6) non-profit (i.e. **an industry consortium**), not a US 501(c)(3) non-profit (i.e. **not a charity**).
  - Foundation membership requires signing an **NDA** ("Non-Disclosure Agreement").
  - Final decisions are in the hands of the board of directors, and **only platinum-level corporate members (paying US$ 25000 per year) are eligible for a board seat**.
  - **Only corporate members have voting rights**, individual members can take part in working groups, but cannot vote.

Why is it called “RISC-V”?

- It’s a classical **R**educed **I**nstruction **S**et **C**omputing architecture:
    - simple and fast instead of complex and slow instructions
    - fixed instruction-length (in base-ISA)
    - load/store architecture
    - 31 general-purpose registers, hardwired zero register
    - easy to pipeline
- It’s the fifth major RISC architecture developed by the University of California in Berkeley: RISC-I, RISC-II, SOAR, SPUR and finally RISC-V.



Early RISC-V prototype chip from 2013, photo by Derrick Coetzee, license: CC0 1.0
Image source: `https://en.wikipedia.org/wiki/File:Yunsup_Lee_holding_RISC_V_prototype_chip.jpg`

Other RISC architectures

Closed:

- MIPS (e.g. in DSL routers)
- ARM (e.g. in smartphones + SBCs)
- IBM Power (mainly in server systems)
- HP PA-RISC (mainly in server systems)

Semi-Open:

- Sun Microsystems SPARC (mainly in server systems, radiation-hardened variants used in spacecraft). Architecture published as IEEE standard 1754-1994, but the standard isn't DFSG-free.

Open:

- OpenRISC (mainly embedded controllers)

MIPS-based router, photo by Evan Amos, in the public domain.
Image source:
`https://commons.wikimedia.org/wiki/File:Linksys-Wireless-G-Router.jpg`

ARM-based single-board-computer, photo by Evan Amos, in the public domain.
Image source:
`https://commons.wikimedia.org/wiki/File:Raspberry-Pi-2-Bare-BR.jpg`

Why yet another RISC architecture?

- Closed architectures are unusable for research purposes.
- SPARC is only semi-open and the SPARC designers have made a number of design choices that CPU designers would like like to avoid from today's point of view.
- OpenRISC is fully open, but at the time RISC-V was created, OpenRISC had two major drawbacks that got resolved only later on:
  - originally 32bit-only
  - no upstream gcc support

→ **New, clean-slate design without mandating a particular microarchitecture**

- 32bit, 64bit and 128bit support
- no condition codes, no branch-delay slots, no architectural register-windows
- extendable but at the same time easy-to-parse instruction encoding
- focus on avoiding patented technology in the ISA design
- strict upstreaming policy

A modular architecture - Part I

- The RISC-V ISA is split into a Base-ISA and so-called “Extensions” that are each assigned a character:
  - “I” - the base ISA providing integer functions
  - “M” - hardware multiply and divide support
  - “A” - support for atomic operations
  - “F” - single-precision IEEE 754 floating point support
  - “D” - double-precision IEEE 754 floating point support

  } “G” - the “general purpose” set

  - “C” - “compressed” instruction support (shorter encodings for often-used instructions)

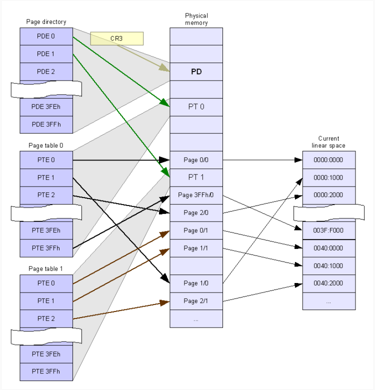
- A full ISA-Identifier consists of RV + base register size + extensions, e.g “RV64IMAFD” (=“RV64G”).

A modular architecture - Part II

- The RISC-V ISA has three privilege modes:
  - *machine mode* (highest privilege)
  - *supervisor mode*
  - *user mode* (lowest privilege)

  Only machine mode is mandatory; low-power microcontrollers commonly only implement machine mode. For running unix-style operating systems with virtual memory, a CPU has to implement all three privilege modes and a PMMU.

- The 32bit, 64bit and 128bit Base-ISAs are independent from each other, i.e. an RV64 CPU cannot automatically execute RV32 code. A CPU implementation can support more than one Base-ISA, but doesn't have to (similar to (64bit-)ARMv8 where support for (32bit-)ARMv7 code is optional).



Page-based virtual memory schema, by user "Jego.ruS" on Wikimedia Commons.
Image in the public domain, image source: `https://commons.wikimedia.org/wiki/File:Linear_addressing.png`

Meltdown, Spectre and open architectures

A common misconception: *"Open architectures like RISC-V are immune to Meltdown and Spectre".* This statement is nonsense!

- Meltdown and Spectre are vulnerabilities in specific processor **implementations**, not in an **ISA** per se.
- Meltdown and Spectre are side-channel attacks against externally visible effects of out-of-order/speculative code execution.
- All **currently** available RISC-V chips are in-order designs without speculative execution, so they are not vulnerable to meltdown- and spectre-style attacks.
- Out-of-Order RISC-V implementations (such as BOOM) are currently in development and those could in principle be vulnerable to meltdown- and spectre-style attacks.

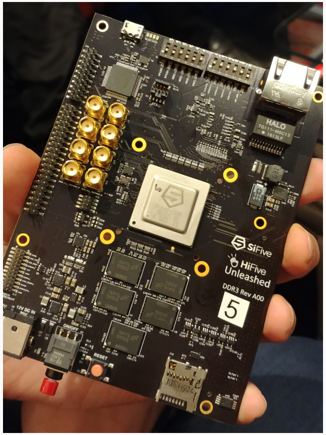
Meltdown and Spectre logos by Natascha Eibl, license: CC0 1.0, Image sources: `https://commons.wikimedia.org/wiki/File:Meltdown_with_text.svg` `https://commons.wikimedia.org/wiki/File:Spectre_with_text.svg`

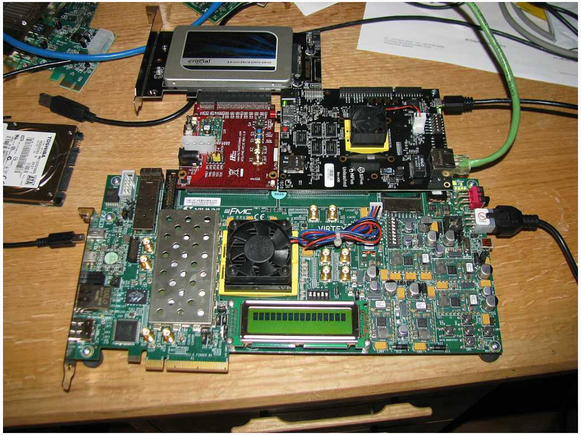
Hardware?

Having an open ISA is nice, but how to actually use it? Options for running RISC-V code include:

- Emulation via qemu (both user-mode as well as full system emulation).
- Linux-capable softcores under DFSG-free licenses on FPGAs (Rocket, BOOM, Ariane).
- ASICs, i.e. “real” CPU chips
    - SiFive Freedom U540 SoC (quad-core RV64GC), first silicon presented at FOSDEM 2018, development boards with first-generation chips available for purchase by the general public since March 2018 (although still very expensive).
    - LowRISC - a project to produce an open-hardware RV64 “community SoC”, but still probably 3-5 years away from being able to publicly tape-out a chip in numbers.
    - SHAKTI - a project at the IIT Madras to design a family of BSD-licensed RISC-V-based SoCs covering the full range from microcontrollers to desktop/server class systems.

Hardware!


SiFive
HiFive Unleashed
DDR3 Rev A00
5
HALO
RESET
ON/OFF
12V DC IN


crucial
TOSHIBA
FMC
VIRTEX

Why porting Debian to RISC-V?

- Fits perfectly from a philosophical standpoint - free software on free hardware.
- There have been other open ISAs before (e.g. OpenRISC), but RISC-V is probably the first open ISA that has a realistic chance of gaining significant market traction:
  - Modern and clean scalable ISA design with 64bit support
  - Proper upstream toolchain support
  - Industry support from major players (among them Google, Marvell, Micron, Microsemi, Nvidia, NXP, Qualcomm, Rambus, Samsung, Western Digital, Allwinner, Espressif, Globalfoundries, Hitachi, Huawei, IBM, IDT, Lattice, MediaTek, Seagate, Siemens and SK Hynix)
- RISC-V comes at a time where community-led production of actual silicon slowly becomes feasible and the ongoing reverse-engineering efforts for FPGA bitstreams might enable a wider adoption of CPU designs on FPGAs in the free software world during the next years.
- The usefulness of an architecture very much depends on the software ecosystem available for it.

RISC-V software development: The dark ages - Part I

Involvement of Debian developers in RISC-V started in early 2015. At that time, a lot of things were still in flux:

- The RISC-V privileged specification wasn't stable and incompatible changes could happen at any time (and did happen).
- Binutils and GCC weren't upstream
- The glibc and Linux ports were in their infancy and the ABI wasn't stable.
- "Culture clash" in the RISC-V project between people from the hardware/ISA-design world and people from the software development world (an example for a particularly hot topic was the "RISC-V config string vs. device-tree" debate).

RISC-V software development: The dark ages - Part II

- Due to the changing specifications and unstable ABIs, there were version interdependencies between glibc, binutils, gcc, Linux and spike (the only RISC-V CPU emulator available at that time), so that having to rebuild the whole world to accomodate for changes in one component was a common occurance.
- Spike was designed as a CPU emulator that replicated the existing hardware at the time of its creation, which means that emulation of peripherals was only available over the "Host-Target-Interface" used for "tethering" early RISC-V CPU hardware implementations to an existing (in most cases ARM-based) CPU. HTIF supported only a "dumb" console and a single block-storage device, but no network interface, which made userland software development with spike an extremely tedious task.

RISC-V software development: From dark ages to renaissance - Part I

- Early attempts at building a Debian userland for RISC-V in 2015/2016 (as well as similar efforts by Fedora developers) were thwarted by further ABI breaks (glibc/kernel), rendering all created binary packages useless.
- In consequence, both Debian and Fedora developers involved in RISC-V development decided to put all work on building binaries on hold until the ABI was declared stable.
- Work now concentrated on getting binutils, gcc, glibc and Linux upstream and thereby getting the ABI stabilized.
- Debian and Fedora developers contributed to the upstreaming process by providing feedback, code review and patches. This included (among other things) areas like the dynamic linker (to make Debian-style multiarch setups possible) and getting device-tree accepted as the standard RISC-V hardware description format.

RISC-V software development: From dark ages to renaissance - Part II

At the beginning of 2016 an unexpected legal roadblock occured:

- Some of the authors of the RISC-V support code for binutils, gcc and glibc were employed by the University of California, Berkeley (UCB) at the time the code was written, which gave UCB the copyright in parts of the code.
- The FSF insisted on a copyright assignment for inclusion of the RISC-V support code in upstream binutils/gcc/glibc.
- The authors were willing to sign the copyright assignment for their code but...
- ...that required a legal release from UCB. Unfortunately while UCB had no problem with the code being published under BSD/GPL, their lawyers denied assigning the copyright to the FSF for reasons undisclosed to the public. It took most of the year 2016 for this issue to be resolved so that binutils support could go upstream in November 2016.

RISC-V software development: Renaissance - Part I

Once the legal roadblock was cleared, upstreaming of the toolchain made good progress:

- RISC-V support for binutils was merged on 2016-11-01 and released with binutils 2.28 on 2017-03-02.
- RISC-V support for gcc was merged on 2017-02-06 and released with gcc 7.1 on 2017-05-02.

Unfortunately things took significantly longer for glibc:

- Glibc upstream required the kernel ABI to be final and part of an upstream Linux kernel release (or at least a release candidate) before accepting the glibc patchset.
- The first kernel patchset was submitted on 2017-05-22, the parts of the RISC-V support that are relevant for UAPI/ABI have been merged on 2017-11-15 and released with kernel 4.15 on 2018-01-28.

RISC-V software development: Renaissance - Part II

- RISC-V support for glibc has been gradually merged during January 2018 and released with glibc 2.27 on 2018-02-02 (just in time for FOSDEM 2018)
- RISC-V support for qemu has been submitted upstream on 2018-01-03, merged on 2018-03-09 and released as part of qemu 2.12 on 2018-04-24.

What's still missing?

- GDB Linux support (bare-metal support is available)
- LLVM/Clang (currently work-in-progress)
- Java/OpenJDK JIT support (interpreter support is available)
- Rust
- Golang
- FreePascal
- V8 (for node.js)

Distribution bootstrapping preparations

- How to create a base system for a new architecture from thin air?
  - ⇒ Cross-compilation and making use of the Debian multiarch support.
- Requirement: a multiarch-aware cross-toolchain for the new architecture.
  - ⇒ Get support for the new architecture into the binutils, gcc, glibc and Linux packages as well as into dpkg.
- Problem: glibc and Linux had just been freshly released upstream and the versions with RISC-V support weren't in the archive because no other architecture used them yet.
  - ⇒ Package newer versions of all toolchain bits and upload them to the "experimental" suite.

Determining the minimal package set for a port

- "What is the smallest possible Debian system?"
  - $\Rightarrow$ The set of packages with priority "required".
- "That's just a handful of packages, isn't it?"
  - $\Rightarrow$ Around 55. But we also need their (recursive) dependencies (libraries etc.).
- "But that's it?"
  - $\Rightarrow$ We also need the (recursive) build-dependencies of those dependencies...

In the end, the full list of transitive (build-)dependencies contains several hundred packages, including most of X11, qt and gtk and a bunch of databases. This happens over dependency chains like meson $\rightarrow$ qt $\rightarrow$ X11 or meson $\rightarrow$ python $\rightarrow$ databases.

Dependency issues

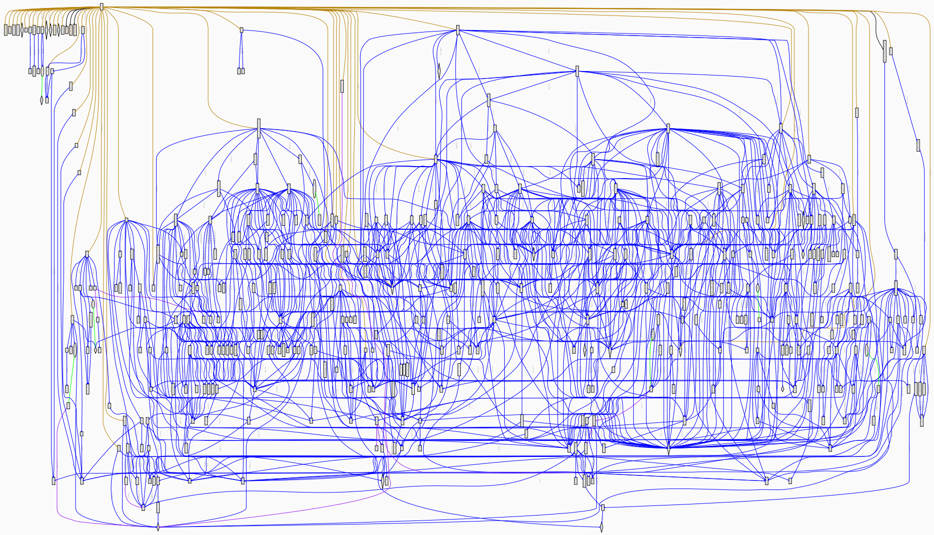
- **Deep dependency chains** cause seemingly simple packages to transitively (build-)depend on everything and the kitchen sink. This happens often when packages depend on either python or qt, as both python and qt offer lots of bindings which cause them to build-depend on lots of other packages.

- **Circular dependency chains** are chains that lead back to their starting point. Example:

  audit $\xrightarrow[\textit{depends}]{\textit{build-}}$ openldap $\xrightarrow[\textit{depends}]{\textit{build-}}$ cyrus-sasl $\xrightarrow[\textit{depends}]{\textit{build-}}$ pam $\xrightarrow[\textit{depends}]{\textit{build-}}$ audit.

- Circular dependencies are often added over time due to “feature creep”.
- Workaround: staged builds with reduced featuresets and therefore reduced (build-)dependencies.
- Staged builds traditionally require a lot of manual work. Debian is trying to address this with the introduction of build-profiles. The end-goal is the ability to do a fully automatic bootstrap, but there is still a lot of work to do to get there.

Firefox-ESR dependency graph



Generated with ‘‘debtree --no-provides --no-conflicts --no-recommends -b firefox-esr | dot -Teps’’

Cross-build issues

Quite a number of packages don't cross-build properly, and some of those are essential for other packages (e.g. perl, gobject-introspection). Typical issues include:

- Cross-build support varies quite a bit between build systems:
  - GNU autotools-based projects usually just worked.
  - Quite a number of CMake- and Meson-based projects didn't cross-build out of the box. Both CMake and Meson generally support cross-building, but the practical experience with real-world projects using them hasn't been particularly great.
  - The perl build system claims to support cross-building, but is completely broken for the case of bootstrapping a new architecture.
- Makefiles try to execute code that they have just built. That works for native builds, but not for cross-builds. Partial solution: qemu-user.
- Improper separation between build-arch and host-arch tools, e.g. Makefiles run the build-arch pkg-config instead of the host-arch pkg-config in a cross-build setting.
- Lack of multiarch co-installability - in a cross-build setting, sometimes the build-arch and the host-arch version of a library have to be installed at the same time, but some libraries are not multi-arch co-installable with themselves.

General portability issues - Part I

When porting software to a new architecture, a number of generic portability issues need to be handled. Examples of issues that commonly need to be addressed include:

- Architecture-dependent data type sizes
- Endianess
- Outdated config.sub/config.guess files in the upstream sources
- Software using wrong flags for atomics and pthread support
    - Proper: -pthread
    - Improper: -lpthread

  The former sets appropriate platform-specific flags and automatically links in libatomic when necessary, while the latter doesn't. The improper version works on some architectures (e.g. i386/amd64), but not on others (among them RISC-V).
- Upstream sources that require creating per-arch header files by hand. Example: libgpg-error.

General portability issues - Part II

- Upstream not taking care of the fact that some architectures have type constraints for atomic operations (i.e. native support is only available for word-sized atomics but not for atomic operations on sub-word sizes). Libatomic abstracts such differences away, but upstream often doesn't use libatomic.
- Upstream build systems making decisions based on a finite list of architectures instead of making them based on specific properties, so every new architecture to build the code for requires patching the sourcecode.
- Upstream shipping configure scripts that cannot be re-built from source because they have been hand-edited by upstream after initially having been created from source. Example: perl
- Missing language/compiler support for new platforms, e.g. Rust (required for building Firefox).

(Semi-)Automatic bootstrapping?

- Bootstrapping a distribution manually is a rather tedious task.
- Fully automating it is a lot more complicated than one would guess at first sight.
- Bootstrapping happens in Debian/unstable $\Rightarrow$ moving target.
- Foundational work by Johannes Schauer in his master thesis "Solving the Bootstrap Problem for Free and Open Source Binary Distributions" from 2013.
- Automation of the early stages of cross-bootstrapping a new architecture with the "rebootstrap" tool by Helmut Grohne.

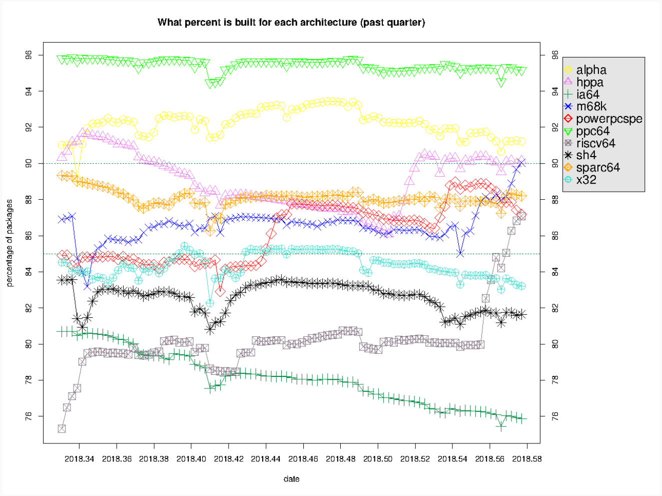
Debian infrastructure requirements for a new port

1. Becoming a “debian-ports” architecture (i.e. a second-class architecture where bugs don’t influence the ability of packages on release architectures to migrate from unstable to testing and ultimately to stable) requires:
    - Upstream toolchain support available.
    - Support for the new architecture in dpkg/unstable available.
    - Enough buildds available to cope with the flow of new package uploads, but buildds can be hosted outside of DSA-managed infrastructure and may be run in qemu.

    For “debian-ports” architectures, uploads of packages with arch-specific modifications to the “unreleased” suite are permitted; packages are kept in a separate archive run by a mini-dak instance. **That’s where we are right now.**

2. Becoming a regular Debian architecture in addition requires:
    - The architecture is able to build more than 95% of the archive.
    - The archive is managed by the main dak instance, all packages are in unstable (no “unreleased” suite).
    - Support for the architecture is available in dpkg/stable.
    - All buildds run on DSA-managed physical hardware, porterboxes are available.

Debian ports archive build status



Debian-Ports archive per-architecture build status.
Non-copyrightable graph, programmatically generated from factual data.
Image source: `https://buildd.debian.org/stats/graph-ports-quarter-big.png`

Debian infrastructure requirements for a release architecture

Each architecture in Debian must re-qualify as a “release architecture” for each stable release. In addition to the previous requirements, a release architecture must fulfill the following points:

- It must have Debian-Installer support.
- It must have enough buildd-redundancy.
- Hardware managed by DSA must fullfill certain requirements (rack-mountable, out-of-band remote-management capabilities, easily available replacement hardware in case of failure).
- It must have at least 3 porters who take care of architecture-specific issues over the whole lifetime of the stable release.

→ **RISC-V definitely won’t become a release architecture for “Buster”.**

Outlook - Software

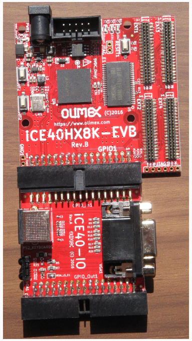
- **Debian-Installer**
  Basic RISC-V support is available in the Debian-Installer git repository, but further upstream work on the bootloader side of things (U-Boot + OpenSBI / BBL) and on elfutils is required before publishing a Debian-Installer release for RISC-V makes sense.

- **RV32 support**
  Currently Debian only supports RV64. Upstream glibc and kernel still lack full RV32 support (currently work-in-progress) and there is no Linux-capable RV32 hardware available yet. RedHat/Fedora is only interested in server-class RV64 hardware and definitely won't provide an RV32 port. Debian might look into an RV32 port in the future if interesting implementations should become available (cf. the next slide).

Outlook - RISC-V CPUs on FPGAs with open toolchains - Part I

While FPGAs provide the option of implementing RISC-V CPUs ("SoftCores") on them and doing so is a standard procedure in commercial CPU development, this hasn't been particularly interesting to the free-software world in the past because all FPGAs required using proprietary vendor toolchains and closed-source macros. Things are changing slowly now with more and more FGPA bitstream reverse-engineering projects:

- **Project IceStorm** (for the Lattice iCE40 FPGA series) provides a fully-featured DFSG-free toolchain (packaged in Debian).
- **Project Trellis** (for the Lattice ECP5 FPGA series) hasn't yet achieved feature-completeness but David Shah has already implemented a linux-capable mor1kx-based OpenRISC-SoC on an ECP5 FPGA with it.
  $\rightarrow$ "Project Trellis and nextpnr" talk in AW 1.125 on Sunday
- **Project X-Ray** (for the Xilinx 7-series FPGAs) is still in a comparatively early stage.



OSHW FPGA board with iCE40
(Photo by K. Merker/CC-BY-4.0)

Outlook - RISC-V CPUs on FPGAs with open toolchains - Part II

Unfortunately there are downsides to CPUs on FPGAs:

- Low clockrates (in the range between 50 and 200MHz, depending on the CPU design and the FPGA type).
- Fast FPGAs are expensive.
- High power consumption when compared to an ASIC.
- Attaching reasonable amounts of memory usually requires either using a proprietary and undocumented DDR-DRAM PHY or resorting to comparatively slow and expensive alternative memory types.
- The iCE40 FPGA series - which is the only one for which we have a feature-complete free toolchain right now - is only available in very limited sizes, so it is largely limited to microcontroller-class RV32 CPU designs. With the ongoing development of project Trellis, implementing a fully-fledged Linux-capable RISC-V CPU with supervisor mode support and virtual memory will probably become possible in the forseeable future, though.

Outlook - community-driven ASIC production?

The RISC-V project has inspired a number of initiatives aiming at making community-driven production of open-source chip designs possible:

- The LowRISC project is working on designing an RV64 “community SoC” - the current development version can run Debian on an FPGA.
- OnChip UIS, a spin-off of the “Universidad Industrial de Santander” in Colombia, develops open analog IP for open-source SoC-designs (e.g. power management unit, LDO, brownout-detection, DAC, ADC, RNG).
- The Libresilicon Alliance aims at developing a standardized, open silicon manufacturing process and a corresponding standard cell library to become independent from NDAed foundry standard cell libraries.
- The Chips4Makers project works on improving free software for ASIC design and tries to prove the feasibility of bringing free chip designs to fabrication with an all-open-source workflow. $\rightarrow$ “The future of Retro-uC and Chips4Makers” talk in AW 1.125 on Sunday.

The End

Questions?