The CORE-V software ecosystem: Ten lessons learned from developing vendor specific compiler tool chains

Jeremy Bennett[1]

[1]Embecosm Limited

Abstract

CORE-V is a family of RISC-V processor cores available as free silicon IP from the Open Hardware Group (abbreviated here as OpenHW Group). The OpenHW Group is a consortium of 102 industrial, academic and other organizations creating free and open source RISC-V IP that is verified and commercially robust. The CORE-V family makes heavy use of both non-standard and standard ISA extensions. In this talk we look at the challenges is creating the software tool chains to go alongside this silicon IP and present a series of lessons learned. This talk is illustrated by reference to both GCC and Clang/LLVM tool chains for CORE-V, particularly the 32-bit CV32E40Pv2 processor.

Introduction

The CV32E40P processor is derived from the ETH Zürich PULP RI5CY processor. The original project developed a standard RISC-V `RV32IMAC_zicsr` core. The CV32E40Pv2 project extends this processor with the standard `F` and `zfinx` extensions along with 8 ISA extensions from the ETH Zürich and the University of Bologna PULP project [2]:

- `Xcvmem` post-incrementing load/store (25);
- `Xcvhwlp` hardware loops (6);
- `Xcvalu` general ALU operations (31);
- `Xcvmac` multiply-accumulate (22);
- `Xcvbi` immediate branching (2);
- `Xcvelw` event load (1);
- `Xcvbitmanip` PULP bit manipulation (16); and
- `Xcvsimd` PULP SIMD (220).

The numbers in parentheses refer to the number of instructions in each extension. In addition CORE-V is a proving ground for official extensions prior to ratification. The CV32E41P is a Technology Readiness Level (TRL) 3 demonstrator of the `Zc*` extension, version 1.0.1, which is also supported in the compiler tool chain (see [3] for a description of TRLs). Over the past three years a team of engineers from Embecosm, ETH Zürich, TU Tübingen, University of Bologna and ISCAS PLCT have been developing GCC and Clang/LLVM tool chains for CORE-V.

We take it as axiomatic, that the CORE-V tool chains must eventually be supported upstream. Vendor specific variants of tool chains are supported by both GCC and Clang/LLVM upstream, and this is provided for in the RISC-V standards.

In this talk we explore the following areas:

- the technical implementation of the tool chain ISA extensions;
- compliance with RISC-V ISA encoding;
- upstreaming vendor specific extensions; and
- the rôle of CORE-V as a proving ground for pre-ratification RISC-V features.

We focus on the 32-bit CV32E40Pv2 processor, although the topics apply to all other 32- and 64-bit CORE-V processors. Throughout we highlight the lessons learned from our experience.

Technical implementation

For the extended abstract we provide only the lessons learned in each section. The full talk will provide the detailed technical explanation.

1.1 The code bases

For both GCC and Clang/LLVM, we maintain out of tree trackers of the upstream repositories on GitHub.

Lesson learned #1: The RISC-V tool chains are under active development. Rebasing from a very old mirror is a laborious job, requiring a lot of rewriting of the CORE-V specific patches. Rebase often!

1.2 Identifying CORE-V in the code

We have submitted the two letter prefix "cv" to the official RISC-V tool chain conventions repository [2].

Lesson learned #2: It takes a long time to get this accepted. RISC-V subcommittees are a bottleneck, and their involvement should be kept to a minimum.

1.3 Instruction encoding

The ISA extensions as originally defined by the PULP project pre-date the finalization of the RISC-V ISA, and its support for non-standard ISA extensions.

For CORE-V, the PULP instructions were all re-encoded to use custom-0 through custom-3 fields, thereby becoming RISC-V compliant. Some of these, while RISC-V compliant do not use existing instruction formats.

1.4 Assembler support

Lesson learned #3: The GNU assembler lacks the infrastructure to support variant encodings for multiple versions of ISA extensions. Plan for a single version of any ISA extension. If you need to update in the future, give it a new name.

Lesson learned #4: The CORE-V ISA extensions were specified using mixed case, which is not supported in the GNU assembler. Only define assembler instructions in lower case!

1.5 Linker support

The presence of new instruction formats can require new linker relocations. This is needed for the CORE-V hardware loop instructions. There are only 64 vendor specific linker relocations available for all vendors in the world. There is a proposed solution which has been in the works for two years.

Lesson learned #5: Comprehensive vendor specific relocation support is an absolute requirement if vendor specific tool chains are to be supported upstream.

1.6 Defining builtin functions

Builtin functions are the first step towards full compiler support for ISA extensions. While they look superficially like ordinary functions, they are actually integrated within the compiler itself, and are amenable to direct optimization within the back-end of the compiler.

Lesson learned #6: Don't slavishly define one builtin per ISA extension opcode. Define builtin functions to serve the user not the ISA extension.

Lesson learned #7: Defining a builtin function is not always appropriate. Sometimes the new instructions can be used to improve existing standard builtin functions.

1.7 Code generation

At this point things do get easier. This is just a matter of extending back-end code generation patterns to take advantage of the new instructions when these are available. We can already generate post-incrementing load/store for loops where appropriate.

1.8 Upstreaming

Following upstream convention, the CORE-V tool chains are built using `corev` as the vendor field in the target triplet. Thus we have `riscv32-corev-elf-gcc` rather than the generic `riscv32-unknown-elf-gcc`. This results in `VENDOR_COREV` being defined, which is used to gate all CORE-V specific code (all in the `config/risc-v` directory).

Our aspiration had been to submit all ISA extensions at the same time. However we have recognized that this would represent i) a huge reviewing demand and ii) only be possible once the vendor specific relocation issue is resolved.

Another effect of this is that many of the CORE-V ISA extensions have been complete (at least in the assembler) for over a year. We have been continually having to rebase and update these patches. Had the code been upstream, this job would have been much easier.

Inevitably the commits as we develop code have not been perfect first time. Before upstreaming, we have to recast all the work into a rational set of commits, suitable for upstream review. A rigorous requirement of commits to follow the upstream tool coding conventions makes this task less demanding.

Lesson learned #8: Upstream early, in small, well-defined chunks.

Lesson learned #9: Ensure your out-of-tree commits follow the upstream coding conventions at all time.

Discussion

We have presented the technical lessons we learned as we have developed the CORE-V tool chains. However these processor cores often incorporate upstream extensions before they are finally ratified. Upstream GCC in particular has very strict rules about not-accepting patches until an extension is ratified, or at least frozen pending ratification.

The CORE-V mirrors do not have this constraint. Thus we can develop compiler code support earlier on. This in turn allows extension developers early visibility of how their extension may work in the compiler. For example the `Zc*` extension has been supported in the CORE-V GNU tool chain since draft 0.7.5. The latest tool chain supports the frozen draft 1.0.1, and is in the process of being submitted upstream for inclusion in GCC 13.1.

Lesson learned #10: Organizations like the OpenHW Group can be a valuable proving ground for pre-ratification RISC-V extensions.

Summary

We have presented 10 lessons learned by the multi-national team developing GNU and Clang/LLVM tool chains for the CORE-V family of processors. We hope these will be of value to the wider RISC-V community.

References

[1] RISC-V Toolchain Conventions. Offical RISC-V GitHub repository: github.com/riscv-non-isa/riscv-toolchain-conventions.

[2] PULP: Parallel Ultra Low Power. https://iis-projects.ee.ethz.ch/index.php/PULP. Retrieved 20 March 2023.

[3] Technology Readiness Level. https://en.wikipedia.org/wiki/Technology_readiness_level. Retrieved 10 May 2023.