The CORE-V Software Ecosystem

Ten lessons learned from developing vendor specific compiler tool chains

Jeremy Bennett
OpenHW Group and CORE-V

CORE-V: A family of 32- and 64-bit RISC-V cores & system software
About Embecosm

- Open source software consultancy, focusing on
  - compiler tool chains
  - pre-silicon models
  - operating system bring up
  - Bayesian inference AI/ML

- R&D centers in Southampton, Paris and Nürnberg
Lesson 1: Rebase Often

- RISC-V tools are under active development
- Rebasing from an old mirror is laborious
- You want to build on the latest developments
- See lesson 8...

Image: git-scm.com CC-BY-NC-SA
Lesson 2: Avoid Committees

- RISC-V standardization needs committees
- Committee decisions take hard work = time
- Avoid dependencies on a committee decision

Image: Firkin at openclipart.org
Lesson 3: Single Version of Extensions

- Multiple version support in tools is incomplete
- Variant instruction encodings not supported
- CORE-V had to abandon its original encodings

Image: horse50 at openclipart.org
Lesson 4: Use Lower Case Assembler

16-Bit x 16-Bit Multiplication

- GNU assembler assumes lower case
  - not deliberate, but no-one had ever tried mixed-case
  - too much effort to fix
- CORE-V had to change mixed-case instructions
RISC-V psABI provides 64 vendor relocations
- enough for one vendor, but not all vendors

Add a new 32-bit relocation to identify vendor
- use in pairs

Needs standardizing
- reference implementation in development
Lesson 6: Use Builtin Functions Wisely

- Don’t slavishly create one builtin per ISA opcode
- Define builtins to suit the user
- Example:
  __builtin_riscv_cv_simd_shuffle_sci_b
  - maps to one of four CORE-V instructions

C and C++ Functions

Builtin Functions

User Defined Functions
Lesson 7: Not Everything Needs Builtins

- Reuse standard builtins where possible
  - e.g. CORE-V ALU extension reuses `__builtin_abs`
- Some operations don’t work well as builtins
  - flow of control instructions
  - load/store with novel addressing modes
Lesson 8: Upstream Early & Often

- Out-of-tree is expensive
  - see Lesson 1
- Instead become part of the upstream code base
  - benefit from community interaction
- GCC and LLVM have long accepted vendor variants
Lesson 9: Use Upstream Conventions

- Common coding styles make code easier for everyone
  - even if you have a better style
  - otherwise patches may be rejected
  - clang-format is your friend

- This is more than just code style
  - patch format conventions
  - submission and review process

- Use the same out-of-tree

---

Image: u/trutheality at reddit.com
Lesson 10: Use Proving Grounds

- Upstream is the goal
  - tool-centric view of the world

- Initially a target centric view helps
  - gather consortia to work on the project
  - attract new engineers to tool development
  - move code upstream ASAP

Image: www.openhwgroup.org
Thank You

jeremy.bennett@embecosm.com

embecosm.com
openhwgroup.org