The Rocket Chip Generator

Krste Asanovi
Rimas Avizienis
Jonathan Bachrach
Scott Beamer
David Biancolin
Christopher Celio
Henry Cook
Daniel Dabbelt
John Hauser
Adam Izraelevitz
Sagar Karandikar
Ben Keller
Donggyu Kim
John Koenig

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-17
http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

April 15, 2016

The Rocket Chip Generator

Krste Asanović, Rimas Avižienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Palmer Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Benjamin Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, Andrew Waterman

Electrical Engineering & Computer Sciences Department
University of California
Berkeley, California

April 15, 2016

Abstract

Rocket Chip is an open-source Sysem-on-Chip design generator that emits synthesizable RTL. It leverages the Chisel hardware construction language to compose a library of sophisticated generators for cores, caches, and interconnects into an integrated SoC. Rocket Chip generates general-purpose processor cores that use the open RISC-V ISA, and provides both an in-order core generator (Rocket) and an out-of-order core generator (BOOM). For SoC designers interested in utilizing heterogeneous specialization for added efficiency gains, Rocket Chip supports the integration of custom accelerators in the form of instruction set extensions, coprocessors, or fully independent novel cores. Rocket Chip has been taped out (manufactured) eleven times, and yielded functional silicon prototypes capable of booting Linux.

1 Introduction

Systems-on-chip (SoC) leverage integration and customization to deliver improved efficiency. *Rocket Chip* is an open-source SoC generator developed at UC Berkeley suitable for research and industrial purposes. Rather than being a single instance of an SoC design, Rocket Chip is a design generator, capable of producing many design instances from a single high-level source. It produces design instances consisting of synthesizable RTL, and multiple functional silicon prototypes have been manufactured. Extensive parameterization makes it flexible, enabling easy customization for a particular application. By changing a single configuration, a user can generate SoCs ranging in size from embedded microcontrollers to multi-core server chips. Rocket Chip is open-source and available under a BSD license on Github[1]. For increased modularity, many of the component libraries of Rocket Chip are available as independent repositories, and we use `git` submodules to track compatible versions. Rocket Chip is stable enough to produce working silicon prototypes, and we continue to expand the space of designs it can express with new functionality. This report briefly catalogues the Rocket Chip features available as of April 2016.

[1] `https://github.com/ucb-bar/rocket-chip`

2 Background

Rocket Chip is based on the RISC-V Instruction Set Architecture (ISA) [11]. RISC-V is an ISA developed at UC Berkeley and designed from the ground up to be clean, microarchitecture-agnostic and highly extensible. Most importantly, RISC-V is free and open, which allows it to be used in both commercial and open-source settings [2]. It is under the governance of the RISC-V Foundation[2] and is intended to become an industry standard.

Using RISC-V as an ISA removes potential licensing restrictions from Rocket Chip and allows the same ISA and infrastructure to be used for a wide range of cores, from high-performance out-of-order designs (Section 5) to small embedded processors (Section 6). RISC-V is flexible due to its modular design, which features a common base of roughly 40 integer instructions (I) that all cores must implement, with ample opcode space left over to support optional extensions, of which the most canonical have already been standardized. Existing extensions include *multiply and divide* (M), *atomics* (A), *single-precision* (F) and *double-precision* (D) floating point. These common extensions (IMAFD) are collected into the (G) extension that provides a general-purpose, scalar instruction set. RISC-V provides 32-bit, 64-bit, and 128-bit address modes. A *compressed* (C) extension provides 16-bit instruction formats to reduce static code size. Opcode space is also reserved for non-standard extensions, so designers can easily add new features to their processors that will not conflict with existing software compiled to the standard. RISC-V's User-level ISA is frozen and described in the official RISC-V Instruction Set Manual [11]. The privileged ISA and platform specification are currently under review, with draft specifications available [10].

A large and growing software ecosystem is available for RISC-V. This includes the *GCC* and *LLVM* compilers (and their supporting infrastructure such as *binutils* and *glibc*), ports of the *Linux* and *FreeBSD* operating systems and a wide range of software through the Yocto Project's Linux distribution generator, *poky*. Software simulation is available through *QEMU* and *Spike* (homegrown functional simulator).

Rocket Chip itself is implemented in *Chisel*[3], an open-source hardware construction language embedded in Scala [3]. While Chisel describes synthesizable circuits directly—and so more closely resembles traditional hardware description languages like Verilog than high-level synthesis systems—it makes the full Scala programming language available for circuit generation, enabling functional and object-oriented descriptions of circuits. Chisel also has additional features not found in Verilog, such as a rich type system with support for structured data, width inference for wires, high-level descriptions of state machines, and bulk wiring operations. Chisel generates synthesizable Verilog code that is compatible FPGA and ASIC design tools. Chisel can also generate a fast, cycle-accurate RTL simulator implemented in C++, which is functionally equivalent to but significantly faster than commercial Verilog simulators and can be used to simulate an entire Rocket Chip instance.

3 Rocket Chip Generator

The Rocket Chip generator is written in Chisel and constructs a RISC-V-based platform. The generator consists of a collection of parameterized chip-building libraries that we can use to generate different SoC variants. By standardizing the interfaces that connect different libraries' generators to one another, we have created a plug-and-play environment in which it is trivial to swap out

[2] `http://riscv.org`

[3] `http://chisel.eecs.berkeley.edu`

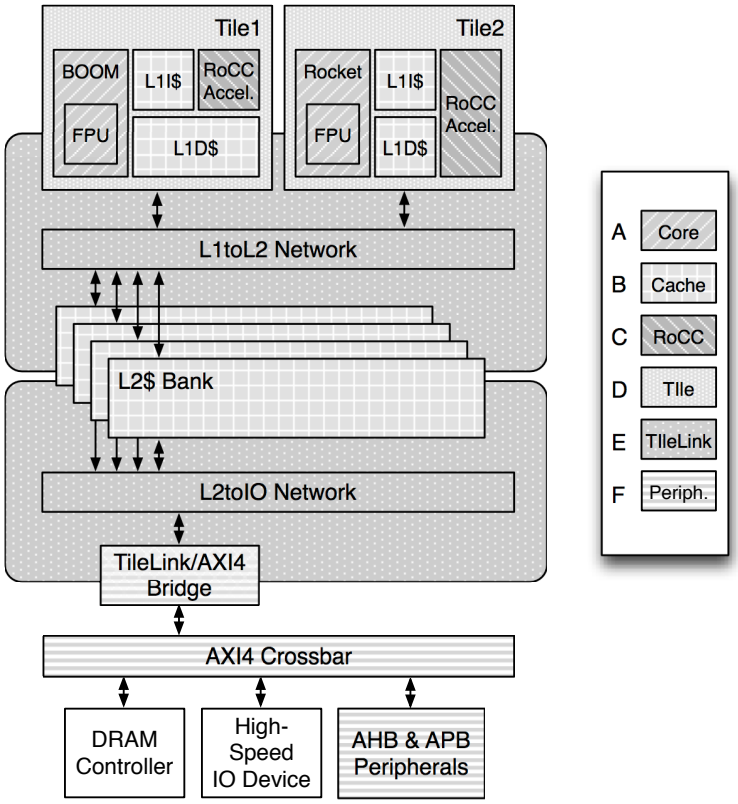

Figure 1: The Rocket Chip generator consists of the following sub-components: A) Core generator B) Cache generator C) RoCC-compatible coprocessor generator D) Tile generator E) TileLink generator F) Peripherals

substantial design components simply by changing configuration files, leaving the hardware source code untouched. We can also both test the output of individual generators as well as perform integration tests on the whole design. The tests, too, are parameterized so as to provide maximal coverage.

Figure 1 is an example of a Rocket Chip instance. It features two tiles attached to a 4-bank L2 cache that is itself connected to the external I/O and memory systems with an AXI interconnect [1]. Within Tile 1 is an out-of-order BOOM core with an FPU, L1 instruction and data caches, and an accelerator implementing the RoCC interface (Section 4). Tile 2 is similar, but it uses a different core, Rocket, and has different L1 data cache parameters. In general, Rocket Chip is a library of generators that can be parameterized and composed into a wide variety of SoC designs. Here is a summary of the current capabilities of the generators and interfaces:

- **Core**: The Rocket scalar core generator and BOOM out-of-order superscalar core generator, both of which can include an optional FPU, configurable functional unit pipelines, and customizable branch predictors.
- **Caches**: A family of cache and TLB generators with configurable sizes, associativities, and replacement policies.
- **RoCC**: The Rocket Custom Coprocessor interface, a template for application-specific coprocessors which may expose their own parameters.
- **Tile**: A tile-generator template for cache-coherent tiles. The number and type of cores and accelerators are configurable, as is the organization of private caches.

- **TileLink**: A generator for networks of cache coherent agents and the associated cache controllers. Configuration options include the number of tiles, the coherence policy, the presence of shared backing storage, and the implementation of underlying physical networks.
- **Peripherals**: Generators for AMBA-compatible buses (AXI, AHB-Lite, and APB) and a variety of converters and controllers, including the Z-scale processor.

To support diverse workloads and to improve energy efficiency, Rocket Chip supports heterogeneity. Not only can the SoC be composed of different tiles, but there is also support for adding custom accelerators. Rocket Chip supports three mechanisms for integrating accelerators, depending on how tightly coupled the accelerator is to the core. The easiest and most tightly coupled option is to expose the accelerator by extending RISC-V and attaching the accelerator directly to the core's pipeline. For more decoupling, the accelerator can act as a coprocessor and receive commands and data from the processor via the RoCC interface. Fully decoupled accelerators can be instantiated in their own tiles and connect coherently to the memory system using TileLink. Furthermore, these techniques can be combined, as in the case of the Hwacha vector-thread accelerator [6], which receives commands via RoCC, but directly attaches to TileLink to bypass the processor's L1 data cache to obtain greater memory bandwidth.

4 Rocket Core

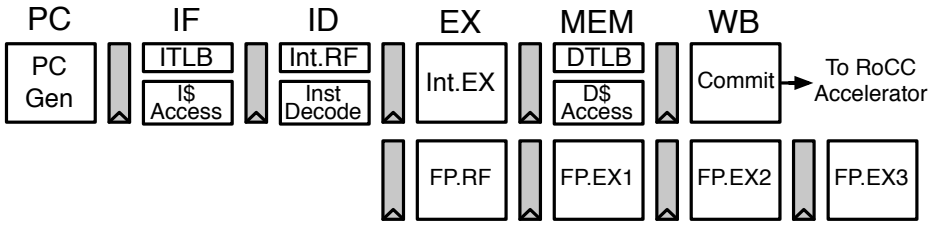

Figure 2: The Rocket core pipeline.

Rocket is a 5-stage in-order scalar core generator that implements the RV32G and RV64G ISAs[4]. It has an MMU that supports page-based virtual memory, a non-blocking data cache, and a front-end with branch prediction. Branch prediction is configurable and provided by a branch target buffer (BTB), branch history table (BHT), and a return address stack (RAS). For floating-point, Rocket makes use of Berkeley's Chisel implementations of floating-point units[5]. Rocket also supports the RISC-V machine, supervisor, and user privilege levels. A number of parameters are exposed, including the optional support of some ISA extensions (M, A, F, D), the number of floating-point pipeline stages, and the cache and TLB sizes.

Rocket can also be thought of as a library of processor components. Several modules originally designed for Rocket are re-used by other designs, including the functional units, caches, TLBs, the page table walker, and the privileged architecture implementation (i.e., the control and status register file).

[4] `https://github.com/ucb-bar/rocket`
[5] `https://github.com/ucb-bar/berkeley-hardfloat`

The *Rocket Custom Coprocessor Interface (RoCC)* facilitates decoupled communication between a Rocket processor and attached coprocessors. Many such coprocessors have been implemented, including crypto units (e.g., SHA3[6]) and vector processing units (e.g., the Hwacha vector-fetch unit [6]). The RoCC interface accepts coprocessor commands generated by committed instructions executed by the Rocket core. The commands include the instruction word and the values in up to two integer registers, and commands may write an integer register in response. The RoCC interface also allows the attached coprocessor to share the Rocket core's data cache and page table walker, and provides a facility for the coprocessor to interrupt the core. These mechanisms are sufficient to construct coprocessors that participate in a page-based virtual memory system. Finally, RoCC accelerators may connect to the outer memory system directly over the TileLink interconnect, providing a high-bandwidth but coherent memory port. The Hwacha vector-fetch unit makes use of all of these features and has driven the development of RoCC into a sophisticated coprocessor interface.

5 Berkeley Out-of-Order (BOOM) Core

BOOM is an out-of-order, superscalar RV64G core generator[7]. The goal of BOOM is to serve as a baseline implementation for education, research, and industry and to enable in-depth exploration of out-of-order micro-architecture. An independent specification available elsewhere[8] elucidates BOOM's design and rationale.

BOOM supports full branch speculation using a BTB, RAS, and a parameterizable backing predictor. Some of the available backing predictors that can be instantiated include a *gshare* predictor and a TAGE-based predictor. BOOM uses an aggressive load/store unit which allows loads to execute out-of-order with respect to stores and other loads. The load/store unit also provides store data forwarding to dependent loads. As shown in Figure 1, a BOOM tile is fully I/O-compatible with a Rocket tile and fits seamlessly into the Rocket Chip memory hierarchy.

BOOM is written in 10k lines of Chisel code. BOOM is able to accomplish this low line count in part by instantiating many parts from the greater Rocket Chip repository; the front-end, functional units, page table walkers, caches, and floating point units are all instantiated from the Rocket and hardfloat repositories. Chisel has also greatly facilitated making BOOM a true core generator - the functional unit mix is customizable, and the fetch, decode, issue and commit widths of BOOM are all parameterizable.

6 Z-scale Core

Z-scale is a 32-bit core generator targeting embedded systems and micro-controller applications. It implements the RV32IM ISA and is designed to interface with AHB-Lite buses, making it plug-compatible in a manner analogous to the ARM Cortex-M series. Z-scale uses a 3-stage, single-issue in-order pipeline that supports the machine and user privilege modes. As it interfaces directly with AHB-Lite buses, it is not a direct replacement for a Rocket (or BOOM) application core and it does not fit into the Rocket Chip cache coherent memory hierarchy. Z-scale is available both in

[6] `https://github.com/ucb-bar/rocc-template`

[7] `https://ucb-bar.github.io/riscv-boom`

[8] `https://ccelio.github.io/riscv-boom-doc/`

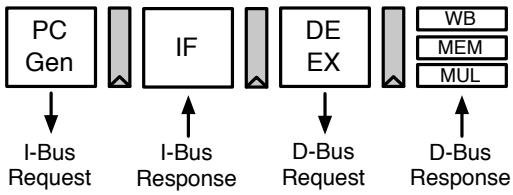

Figure 3: The Z-scale core pipeline.

Chisel,[9] like the rest of Rocket Chip, and also Verilog[10].

7 Uncore and TileLink

Rocket Chip includes generators for a shared memory hierarchy of coherent caches interconnected with on-chip networks. Configuration options include the number of tiles, the coherence policy, the presence of a shared L2 cache, the number of memory channels, the number of cache banks per memory channel, and the implementation of the underlying physical networks. These generators are all based around *TileLink*, a protocol framework for describing a set of cache coherence transactions that implement a particular cache coherence policy.

TileLink's purpose is to orthogonalize the design of the on-chip network and the implementation of the cache controllers from the design of the coherence protocol itself. This separation of concerns improves the modularity of the HDL description of the memory hierarchy, while also making validation and verification of individual memory system components more tractable. Any cache coherence protocol that conforms to TileLink's transaction structure can be used interchangeably alongside the physical networks and cache controller generators we provide. By supplying a framework to apply transactional coherence metadata updates throughout the memory hierarchy, TileLink enables simplified expressions of the coherence policies themselves. Conversely, as long as newly designed controllers and networks make certain guarantees about their behavior, system-on-chip designers can be confident that incorporating them into their TileLink-based memory hierarchy will not introduce coherence-protocol-related deadlocks.

TileLink is designed to be extensible, and supports a growing family of custom cache coherence policies implemented on top of it. TileLink also codifies a set of transaction types that are common to all protocols. In particular, it provides a set of transactions to service memory accesses made by agents that do not themselves have caches containing coherence policy metadata. These built-in transactions make TileLink a suitable target for the memory interfaces of accelerators, coprocessors and DMA engines, and allow such agents to automatically participate in a global shared memory space.

Within the framework provided by TileLink, we provide generators for cache controller state machines as well as data and metadata memory arrays. By composing different sets of features using Chisel, we are able to generate outer level caches with reverse-mapped directories as well as broadcast-based snooping networks. While the extant Rocket Chip uses crossbars to connect tiles, alternate physical network designs can be plugged into our framework, and we are working to

[9] `https://github.com/ucb-bar/zscale`
[10] `https://github.com/ucb-bar/vscale`

interface with designs produced by RapidIO [5] and OpenSoC [4].

To support memory-mapped IO (MMIO), we provide a TileLink interconnect generator. This generator takes an address map description and uses it to produce a system of routers for directing MMIO requests to the correct memory-mapped peripheral. Rocket Chip also provides converters from TileLink to the AXI4, AHB-Lite, and APB protocols in order to allow interfacing with external third-party peripherals and memory controllers.

8 Supporting Infrastructure

FPGA Support

We have released support for deploying Rocket Chip to a variety of Xilinx Zynq-7000 series FPGAs, such as the Xilinx ZC706 and ZedBoard[11]. This capability is primarily used for fast simulation and typically achieves clockrates of 25-100 MHz depending on the configuration and the FPGA. We provide infrastructure and documentation for three use cases:

(1) Deploying pre-built images, for users without Vivado licenses who want to try a RISC-V system on their FPGA

(2) Using existing FPGA infrastructure to deploy a custom SoC generated from Rocket Chip, for users making changes to the Rocket Chip RTL or configuration

(3) Building up the complete FPGA infrastructure from scratch, for users making involved changes to I/O or adding additional board support

The current infrastructure makes use of the hardened core inside the Zynq FPGA to provide support for proxied I/O devices, such as a console and block device.

Memory System Testing

In order to test behaviors in our memory hierarchy which are not easy or efficient to test in software, we have designed a set of test circuits called *GroundTest*[12]. These test circuits fit into the socket given to CPU tiles and directly drive different kinds of memory traffic to the L2 interconnect. The simplest test, *MemTest*, generates writes followed by reads at fixed strides across the address space. We also have a suite of regression tests which check for specific memory bugs which we have encountered in the past. This is especially useful for replicating ordering-dependent or timing-dependent bugs, since these are very difficult or impossible to reproduce in software. Finally, for more intensive testing, we generate random combinations of different memory operations and record a trace of the events. We then feed these traces into an external tool called Axe [8] in order to check for violations of the memory consistency model.

Core Testing

To help stress-test the cores via random testing, we utilize the *RISC-V Torture Tester*[13]. Torture picks randomly from a library of code sequences and stitches them together to form a RISC-V assembly program. The instructions from each code sequence are interleaved with different code sequences. Each randomly-generated test program is executed both on the core under test and the

[11] `https://github.com/ucb-bar/fpga-zynq`

[12] `https://github.com/ucb-bar/groundtest`

[13] `https://github.com/ucb-bar/riscv-torture`

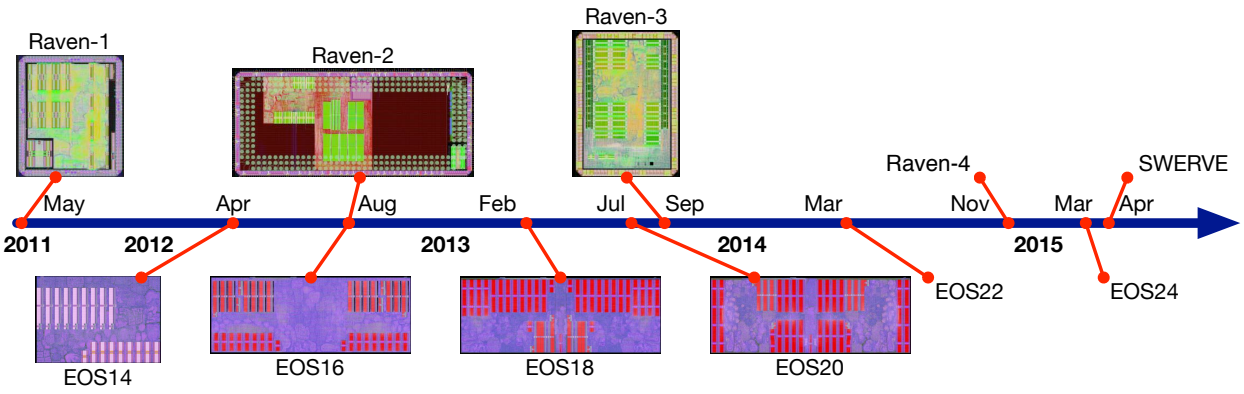

Figure 4: Recent UC Berkeley tapeouts using Rocket Chip.

Spike ISA simulator (RISC-V's "golden model"). At the end of the test program, the register state is dumped to memory and the output from the core is compared to the output of Spike. If Torture finds a test program that exhibits an error, it regenerates smaller versions of the test program until it can find the smallest test program that still exercises the failure. Torture can be run for hours, creating new tests, and storing any error-finding programs it finds.

9 Silicon Prototypes

Figure 4 presents a timeline of UC Berkeley RISC-V microprocessor tapeouts that were created using earlier versions of the Rocket Chip generator. The 28 nm Raven chips combine a 64-bit RISC-V vector microprocessor with on-chip switched-capacitor DC-DC converters and adaptive clocking. The 45 nm EOS chips feature a 64-bit dual-core RISC-V vector processor with monolithically-integrated silicon photonic links [7, 9]. In total, we have taped out four Raven chips on STMicroelectronics' 28 nm FD-SOI process, six EOS chips on IBM's 45 nm SOI process, and one SWERVE chip on TSMC's 28 nm process. The Rocket Chip generator successfully served as a shared code base for these eleven distinct systems; the best ideas from each design were incorporated back into the code base, ensuring maximal re-use even though the three distinct families of chips were specialized differently to evaluate distinct research ideas.

Acknowledgements

We thank the external users and contributors for helping to improve Rocket Chip. In particular, we are grateful for contributions from: Wei Song (Computer Laboratory, University of Cambridge), Matthew Naylor (Computer Laboratory, University of Cambridge), and Schuyler Eldridge (Boston University). This project would of course only be possible with both the RISC-V and Chisel projects and their contributors.

Research partially funded by DARPA Award Number HR0011-12-2-0016, the Center for Future Architecture Research, a member of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and ASPIRE Lab industrial sponsors and affiliates Intel, Google, Hewlett-Packard, Huawei, LGE, NVIDIA, Oracle, and Samsung. Earlier work was supported in part by Intel Corporation and DARPA awards W911NF-08-1-0134, W911NF-08-1-0139,

and W911NF-09-1-0342. Research also supported in part by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Any opinions, findings, conclusions, or recommendations in this paper are solely those of the authors and does not necessarily reflect the position or the policy of the sponsors.

References

[1] AMBA AXI and ACE protocol specification. Feb 2013.

[2] Krste Asanović and David A. Patterson. Instruction sets should be free: The case for risc-v. Technical Report UCB/EECS-2014-146, EECS Department, University of California, Berkeley, Aug 2014.

[3] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. Chisel: Constructing hardware in a Scala embedded language. In *Proc. Design Automation Conference*, pages 1216–1225, 2012.

[4] Farzad Fatollahi-Fard, David Donofrio, George Michelogiannakis, and John Shalf. Opensoc fabric: On-chip network generator: Using chisel to generate a parameterizable on-chip interconnect fabric. In *Proceedings of the 2014 International Workshop on Network on Chip Architectures*, pages 45–50. ACM, 2014.

[5] Sam Fuller. *RapidIO: The embedded system interconnect*. John Wiley & Sons, 2005.

[6] Yunsup Lee, Albert Ou, Colin Schmidt, Sagar Karandikar, Howard Mao, and Krste Asanović. The Hwacha microarchitecture manual, version 3.8.1. Technical Report UCB/EECS-2015-263, EECS Department, University of California, Berkeley, Dec 2015.

[7] Yunsup Lee, Andrew Waterman, Rimas Avižienis, Henry Cook, Chen Sun, Vladimir Stojanović, and Krste Asanović. A 45nm 1.3GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators. *Proc. IEEE European Solid-State Circuits Conference*, Sep 2014.

[8] Matthew Naylor. Axe memory consistency checker. `https://github.com/CTSRD-CHERI/axe`, 2016.

[9] Chen Sun, Mark T. Wade, Yunsup Lee, Jason S. Orcutt, Luca Alloatti, Michael S. Georgas, Andrew S. Waterman, Jeffrey M. Shainline, Rimas R. Avižienis, Sen Lin, Benjamin R. Moss, Rajesh Kumar, Fabio Pavanello, Amir H. Atabaki, Henry M. Cook, Albert J. Ou, Jonathan C. Leu, Yu-Hsin Chen, Krste Asanović, Rajeev J. Ram, Miloš A. Popović, and Vladimir M. Stojanović. Single-chip microprocessor that communicates directly using light. *Nature*, 528:534–538, 2015.

[10] Andrew Waterman, Yunsup Lee, Rimas Avizienis, David A. Patterson, and Krste Asanović. The RISC-V instruction set manual volume II: Privileged architecture version 1.7. Technical Report UCB/EECS-2015-49, EECS Department, University of California, Berkeley, May 2015.

[11] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanović. The RISC-V instruction set manual, volume I: User-level ISA, version 2.0. Technical Report UCB/EECS-2014-54, EECS Department, University of California, Berkeley, May 2014.