RV-IOV: Tethering RISC-V Processors via Scalable I/O Virtualization

Luis Vega
University of Washington

Michael Bedford Taylor
University of Washington

ABSTRACT

Recent interest in open source instruction set architecture (ISA) such as RISC-V has opened new horizons for computer systems research across operating systems, compilers, and hardware architectures. One fundamental aspect catalyzing these innovations is the ability to emulate a complete system. This allows researchers evaluate their ideas on real hardware without the hassle of building infrastructure. One interesting RISC-V core generator available is the Rocket chip generator [1], which generates RISC-V implementations using customizable parameters. There are currently three emulation systems for the Rocket core available to the community: Zybo, Zedboard, and ZC706. These systems are based on a heterogeneous multiprocessing chip composed of a general purpose processor running a host, aka front-end server, and programmable logic where the actual Rocket core is implemented. The drawback of these systems is the tight integration between the Rocket core and the host. This result in challenges when (i) a Rocket core is implemented in ASICs, and (ii) a Rocket core configuration requires more resources than available in current emulation systems. We propose hardware support, RV-IOV, that overcome these limitations and increase the number of prototyping targets for Rocket cores. RV-IOV decouples Rocket cores from the host using I/O virtualization and enables cores to be implemented in ASICs or larger FPGAs. We describe a case of study implementing RV-IOVs to enable five Rocket cores that share the same host in a novel system-on-chip design called Celerity [22], which is a tiered parallel RISC-V architecture implemented in TSMC 16 nm. Additionally, the system is further evaluated against multi-FPGA system, based on the Zedboard and an external open source emulation board called DoubleTrouble [4]. Finally, we measure performance overhead for two systems using seven benchmarks.

KEYWORDS

RISC-V, Rocket cores, I/O virtualization, prototyping, emulation

1 INTRODUCTION

Computer architects spend a considerable amount of time evaluating design properties and tradeoffs using high-level simulators [3, 5, 7] due to the exorbitant cost and time associated with building a prototype [6, 8]. These costs continue to grow after the chip is fabricated as it must be brought up and tested [21, 26]. Validation on real hardware is a necessity to ensure power and performance requirements have been met [22].

Building an ASIC prototype involves other challenges such as functional verification, which is achieved via simulation and oftentimes FPGA emulation. Processor simulation is extremely expensive for large designs and bounded to small programs running on the processor being tested. Therefore, FPGA emulation is employed for efficiently testing the processor under more complex workloads, such as running an operating system. In fact, FPGA emulation was essential for testing x86 operating systems including Linux and Windows on Intel processors [20, 27].

Design and verification are not the only incentives for building a real system. Researchers are also interested in multidisciplinary design exploration covering operating system and computer architecture [10].

For the reasons mentioned above, a complete hardware and software stack are indispensable in order to validate changes made across one or more layers. RISC-V [28], an open source instruction set architecture, has improved this situation by building a hardware and software ecosystem, based on chip generators [1, 19], hardware construction language [2, 12, 25], compilers [15], operating systems [16], and FPGA emulation infrastructure [13].

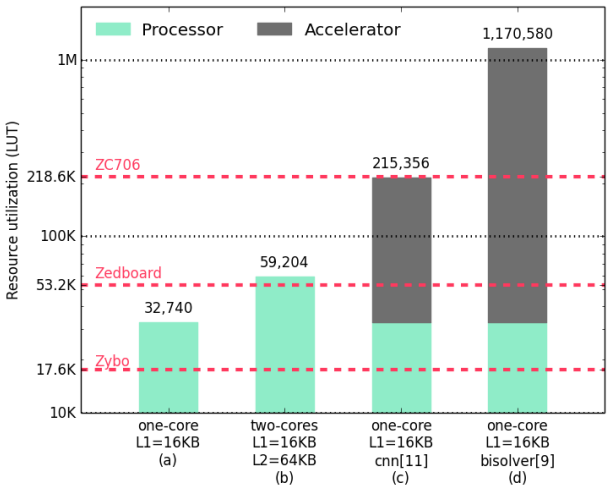

Figure 1: Resource utilization for four Rocket cores and hardware accelerator configurations (a, b, c, and d) and available resources for supported FPGA emulation boards Zybo, Zedboard, and ZC706.

Currently, there are three FPGA emulation platforms based on Xilinx FPGAs that supports tethered Rocket cores, ranging from low power (Zybo, Zedboard) to high performance (ZC706) FPGA boards. Support for these platforms comes as a collection of implementation scripts, processor configurations, hardware modules, and software libraries that automates and simplifies the creation of a complete system. However, the emulation platform has two major limitations. First, all three emulation platforms are based on a single chip emulation environment, resulting in challenges for testing Rocket cores implemented in external chips such as FPGAs or ASICs. Second, the number of programmable logic cells (LUT)

available in these platforms is limited. For example, the Zybo board has 17.6 KLUT while the ZC706 board has 218.6 KLUT.

To quantify these limitations, we obtained resource utilization numbers based on programmable logic cells, for different standalone Rocket core configurations and recent hardware accelerators found in the literature [9, 11].

In Figure 1, we evaluated four configurations (a, b, c, and d) against the available resources present in Zybo, Zedboard, and ZC706 emulation boards. A single Rocket core with 16 KB of L1 cache, represented by configuration (a), already takes up to 61.54% of the resources for the medium range emulation platform or the Zedboard. Whereas configuration (b) based on a Rocket dual-core is only feasible in the most expensive platform (ZC706). Also, we estimate how many resources a single Rocket core described in (a) uses when it is connected to two different hardware accelerators (c, d). Configuration (c) shows that a single Rocket core connected to a convolutional neural network accelerator [11] barely fits in the ZC706 board. Conversely, a bilateral solver accelerator [9], used for virtual reality video, does not fit in any supported emulation board as shown in configuration (d).

In this paper, we propose a hardware mechanism called RV-IOV that extends available platforms by decoupling Rocket cores from the host. Thus, larger systems such as the ones mentioned above are realizable while reusing existing emulation infrastructure. Furthermore, processor decoupling is achieved using I/O virtualization, allowing multiple isolated Rocket cores to be implemented on the same ASIC or FPGA and shared the same host, making RV-IOV ideal for collaborative research projects sharing the same silicon die.

The contributions of this paper are:

- We propose hardware support, called RV-IOV, for implementing multiple Rocket cores on ASIC prototypes or FPGA emulation boards.
- We describe how RV-IOVs enable five Rocket cores share the same host on a novel system-on-chip design called Celerity implemented in TSMC 16 nm.
- We demonstrate the flexibility of RV-IOVs by implementing a Rocket core in a Multi-FPGA emulation environment based on the Zedboard and the DoubleTrouble board [4].
- We evaluate the overhead of RV-IOV based systems, under two FPGA configurations, using seven benchmarks available to the RISC-V community [18].

2 EMULATION INFRASTRUCTURE

Current RISC-V prototyping efforts cover both hardware and software domains. The hardware support for RISC-V is provided by a chip generator [1, 19], hardware construction language [2, 12, 25], and hardware emulation infrastructure [13]. In addition to hardware support, there are software tools and libraries for compiling RISC-V programs [15], managing a tethered Rocket core [14], handling system calls [17], and synthetic benchmarks [18]. The following paragraphs cover essential components for emulating a Rocket core in the available emulation platforms including Zybo, Zedboard, and ZC706.

Hardware support. The hardware architecture consists of a host processor (ARM) and programmable logic (FPGA) where the actual Rocket core is implemented as shown in Figure 2. A tethered Rocket core has two interfaces: host and memory. The host interface is used by the host processor to control the Rocket core. Host transactions are initiated by the host processor, the master, and acknowledged by the Rocket core, the slave. The flow control used by host interface in the Rocket core is based on valid and ready, while the host processor uses AXI4 [29]. Therefore, a flow control *translation* is required as shown in Figure 2.

Unlike the host interface, the memory interface for the Rocket core is AXI4 compliant and allows direct connection to the host processor. In contrast to the host, the Rocket core creates memory requests as a master while the host processor is in charge of handling these requests and creating responses from main memory, as a slave. These memory operations are entirely managed in hardware without any software intervention.

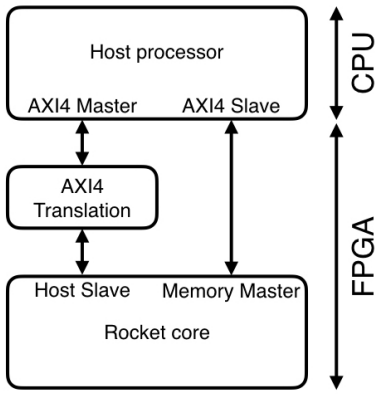

Figure 2: Rocket core emulation platform.

Software tools and libraries. A small set of software libraries running on the host processor are responsible for extending Rocket core capabilities via the host interface. There are two libraries needed by the Rocket core to run programs during emulation, a front-end server (fesvr) [14], and a proxy-kernel (pk) [17]. The front-end server can be considered as low-level library that implements basic functions required by the tethered Rocket core. This includes loading ELF binaries, emulating peripheral devices, and terminating RISC-V programs when they finish running on the Rocket core. With these features in place, bare-metal programs can be properly executed and tested. Finally, the proxy-kernel handles system calls and allows more complex programs to run on the Rocket core.

3 RV-IOV HARDWARE SUPPORT

RISC-V I/O virtualization or RV-IOV is a hardware mechanism developed in SystemVerilog RTL that allows tethered Rocket cores to be decoupled from its host. This is a required feature for Rocket prototyping infrastructure in cases when Rocket cores are implemented in ASICs, while still reusing available host emulation infrastructure. Additionally, FPGA emulation systems benefit from RV-IOV because it enables emulation of larger Rocket core configurations. These designs can be implemented in external platforms that support millions of programmable logic cells such as the ones found in latest FPGAs [30].

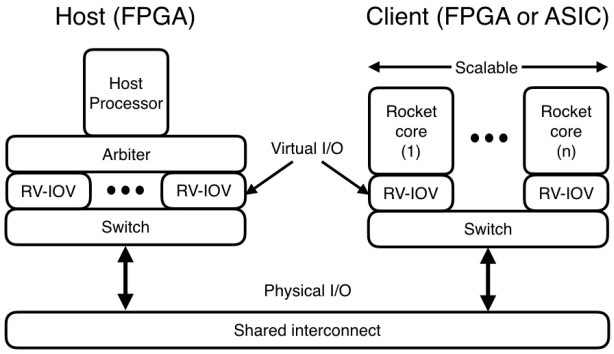

Figure 3: High level view of a RV-IOV based system.

A system level view of RV-IOV is given in Figure 3. Each Rocket core requires a pair of RV-IOVs, one for the host and another for the client connected to the core. The RV-IOV instantiated in the host is implemented in an emulation platform similar to the Zedboard or ZC706, and the RV-IOV placed in the client is implemented in the same technology as the Rocket core either FPGAs or ASICs. This end-to-end hardware mechanism, from host to client, provides I/O virtualization for both the host and Rocket core. We achieve scalability by adding tags to packets according to core id using a switch as shown in Figure 3. This allows multiple Rocket cores to operate under the illusion of having multiple host processors while sharing the same host.

The architecture of the RV-IOV for both the host and client is described in Figure 4. There are two major functions executed by a RV-IOV, memory serialization and stream interleaving.

Memory serialization. In this stage, the memory protocol based on AXI4 is serialized or de-serialized depending on whether the module is located in the client or the host respectively. The RV-IOV in the client merges the five AXI4 channels into a single bi-directional one, while the RV-IOV in the host converts it back to AXI4. Figure 5 shows a finite state machine for write and read operations.

A write operation starts with an address request packet (`aw`), follow by eight data packets (`w`), and finishes with an acknowledge packet (`b`). Similar to the write operation, the read operation starts with an address request packet (`ar`), and waits until the requested data packets, eight in this case, are available (`r`). In contrast to writes, read operations are not acknowledged. In addition to these procedures, we considered two policies to implement these operations. One policy is prioritize writes over reads. If there is a write and a read at the same time, then the write is taken over the read. The second policy is that operations are non-preemptive, i.e. , write and read operations are never interrupted until completion.

Stream interleaving. After memory serialization, memory and host streams can be interleaved over a shared interconnect as shown in Figure 4. On the client side, streams are tagged and queued on intermediate buffers. Next, buffers are dequeued in a round-robin fashion into a single stream. Later in the host, packets are mapped to the corresponding buffer according to their type, either host or memory. Additionally, buffers are sized with respect to bandwidth

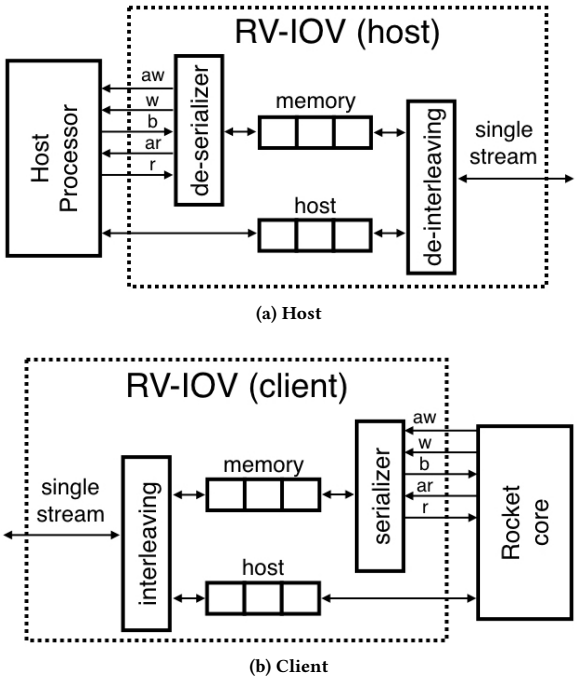

Figure 4: RV-IOV client and host architecture.

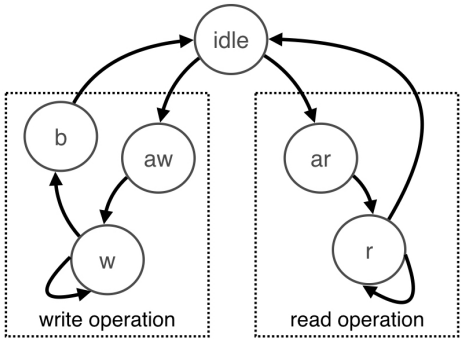

Figure 5: Memory serialization.

delay product between the host and client. Finally, end-to-end flow control, between RV-IOVs, is achieved using a credit protocol.

4 ASIC IMPLEMENTATION

In this section, we briefly explain the Celerity design and how RV-IOVs were used. A more detailed description of Celerity SoC can be found in [22, 23]. The design consist in a 5 x 5 mm 350M-transistor system-on-chip (SoC) implemented in TSMC 16 nm. The SoC is based on a tiered parallel architecture, providing an interesting fabric for embedded applications. The architecture is composed of three tiers, a general purpose tier, a massively parallel tier, and a specialization tier with RISC-V accelerators. The general purpose tier has five 64-bit Rocket cores generated using the rocket chip generator [1, 19] and the hardware construction language Chisel [2,

12, 25]. Next, there are 496 32-bit RISC-V processors defining the massively parallel tier developed using SystemVerilog RTL. Lastly, the specialization tier is based on binarized neural network (BNN) built using Cadence StratusHLS tool. The Celerity chip was taped out in May 2017, and it is expected to return from foundry in September 2017.

Five client RV-IOVs are used in Celerity in the general purpose tier, connected to the five 64-bit Rocket cores. These cores together with RV-IOVs work as a gateway to the other two tiers through the RISC-V accelerator interface or RoCC. Therefore, a single host emulation platform such as the Zedboard can manage the five Rocket cores. The Zedboard provides DRAM memory, storage, and other resources to the Celerity SoC.

5 FPGA EVALUATION

In this section, we explain how the baseline platform Zedboard was extended to evaluate different emulation scenarios for RV-IOV. We use the Zedboard as our baseline platform, because it is supported by RISC-V community, falls into mid-size category, and is common in academia. One option for extending the Zedboard is by the FPGA Mezzanine Card or FMC, which is a high speed I/O mezzanine port that expands the board via a daughter board [24]. We extend the Zedboard by attaching, through FMC, an open source emulation platform called DoubleTrouble [4] as shown in Figure 6.

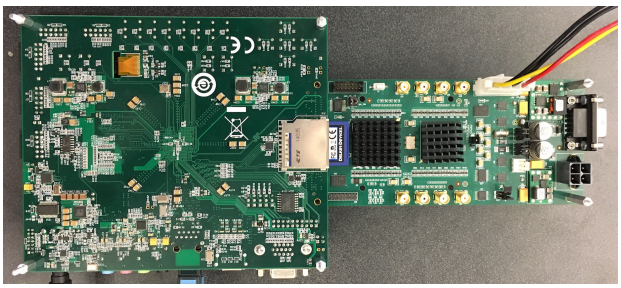

Figure 6: RV-IOV FPGA evaluation system, Zedboard (left) and DoubleTrouble (right).

The DoubleTrouble board is a cost-effective and open-source emulation platform based on two Xilinx Spartan6 FPGAs, one connected to the FMC port called gateway FPGA and the other called client FPGA. The purpose of this board is to provide emulation infrastructure that can be later leveraged for ASIC prototypes.

The gateway and client FPGAs are based on a 45 nm chip that supports I/O voltages ranging from 1.2 V up to 3.3 V, providing support for a wide range of ASIC technology nodes. Additionally, the Spartan6 family offers advaced high-speed I/O features such as SerDes, input and output delay lines, and differential signaling at a reasonable price. Next to the gateway, the client FPGA emulates the ASIC design. The purpose of the client FPGA is to emulate the high risk implementation, which will later be replaced by the ASIC tapeout while reusing the rest of the system. This reduces considerably the overhead of bringing up the chip.

Interestingly, the Zedboard and DoubleTrouble board allow us to evaluate two emulation scenarios for the Rocket core. First, a two FPGA system called one-hop system, composed of a host and gateway FPGA. The Rocket core is implemented in the gateway FPGA. The second system, two-hop, is based on three FPGAs. This system adds a client FPGA to the one-hop system, located in the DoubleTrouble board. Unlike the one-hop system, the two-hop system emulates the Rocket core in the client FPGA. Furthermore, we implemented and evaluated a single Rocket core with 16 KB of L1 cache for both systems.

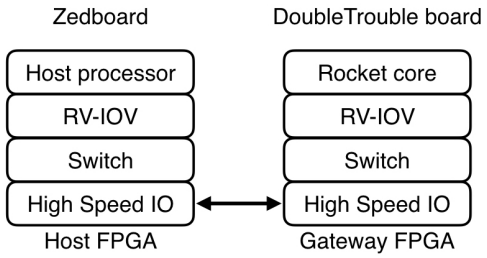

Figure 7: System level view of the one-hop system. Host FPGA (Zedboard) and Gateway FPGA (DoubleTrouble)

One-hop system. A high-level view of this system is described in Figure 7. Here, the host processor is connected to the host RV-IOV, which send and receives Rocket packets from and to a switch. Next, a high-speed IO module transfer packets from the host FPGA to the gateway FPGA at 8.8 Gbps using SerDes working in DDR mode. Later in the client, Rocket packets are processed by the high-speed IO module, and then forwarded to the switch and client RV-IOV. This process happens in both directions between the Rocket and host processors. As a result, the Rocket core implemented in the gateway FPGA operates seamlessly one-hop away from its host.

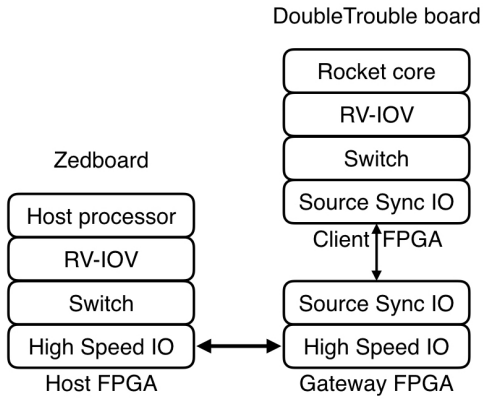

Figure 8: System level view of the two-hop system. Host FPGA (Zedboard) and Gateway/Client FPGA (DoubleTrouble)

Two-hop system. This system adds another FPGA to the one-hop system, a client FPGA, as shown in Figure 8. One key difference compared to the one-hop system is that the gateway FPGA together with the client FPGA implement a source synchronous IO protocol. For these experiments, the data rate for this IO protocol was set at 1.28 Gbps with capabilities of up to 10x higher. Additional properties

of this protocol include error detection/correction and channel calibration. These properties comes at the expense of performance but ensures reliability and flexibility for ASIC prototypes.

There are two reasons for evaluating this particular system. First, we demonstrate that Rocket packets can be split and merged through source synchronous IO like interfaces without affecting the functionality of RV-IOV nor the Rocket core. Second, we show the flexibility provided by the RV-IOV to the Rocket core; there is no limitation on where the Rocket core is implemented as long the RV-IOV mechanism is used.

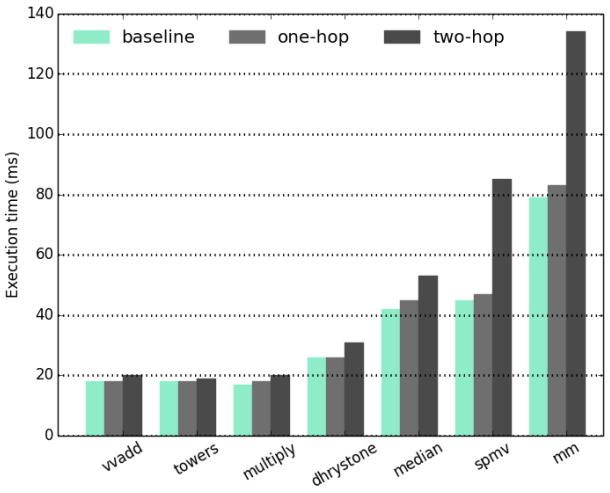

Figure 9: Execution time measured in the host for the two evaluated systems and the baseline.

6 PRELIMINARY RESULTS

We run seven bare-metal benchmarks available for RISC-V processors [18] and measure the execution time for the two implemented systems and the baseline emulation system based on only the Zedboard (without RV-IOV). Results are shown in Figure 9.

It is important to note that this is the default VLSI configuration for the Rocket core. This configuration is based on single-core with 16 KB of L1 cache and without L2 cache. Additionally, the source synchronous IO used in the two-hop system is not optimized and not working at full speed. Therefore, IO intensive benchmarks are impacted significantly for the two-hop system.

7 CONCLUSION

This paper presents a new hardware mechanism called RV-IOV that provides I/O virtualization support to Rocket cores and increases implementation flexibility for both ASIC and FPGA based designs. RV-IOV improves current emulation infrastructures by allowing larger Rocket core configurations, i.e. multi-core and many-core Rocket systems, implemented on more resourceful emulation platforms. The benefits of RV-IOVs are not exclusive to standalone Rocket cores but can be applied to extend Rocket cores with more powerful hardware accelerators.

ACKNOWLEDGMENTS

This research was supported in part by DARPA CRAFT Award HR0011-16-C-0037 and NSF Awards 1059333, 1563767, and 1565446. We thank the many contributors to the open-source RISC-V software and hardware ecosystem, originally created at U.C. Berkeley. We also thank our collaborators on the Celerity SoC project, who taped out this work in the 16nm SoC.

REFERENCES

[1] Krste Asanović, 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, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. The rocket chip generator. Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley, Apr 2016.

[2] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović. Chisel: Constructing hardware in a scala embedded language. In *DAC Design Automation Conference 2012*, pages 1212–1221, June 2012.

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[4] BSG. BaseJump: Open Source Components for ASIC Prototypes. http://bjump.org, 2017.

[5] HP. CACTI 7.0. https://github.com/HewlettPackard/cacti, 2017.

[6] Moein Khazraee, Lu Zhang, Luis Vega, and Michael Bedford Taylor. Moonwalk: Nre optimization in asic clouds. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, pages 511–526, New York, NY, USA, 2017. ACM.

[7] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. The mcpat framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing. *ACM Trans. Archit. Code Optim.*, 10(1):5:1–5:29, April 2013.

[8] Ikuo Magaki, Moein Khazraee, Luis Vega Gutierrez, and Michael Bedford Taylor. Asic clouds: Specializing the datacenter. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 178–190, Piscataway, NJ, USA, 2016. IEEE Press.

[9] Amrita Mazumdar, Armin Alaghi, Jonathan T. Barron, David Gallup, Luis Ceze, Mark Oskin, and Steven M. Seitz. A hardware-friendly bilateral solver for real-time virtual reality video. In *Proceedings of High Performance Graphics*, HPG '17, pages 13:1–13:10, New York, NY, USA, 2017. ACM.

[10] Jeffrey C. Mogul, Andrew Baumann, Timothy Roscoe, and Livio Soares. Mind the gap: Reconnecting architecture and os research. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, HotOS'13, pages 1–1, Berkeley, CA, USA, 2011. USENIX Association.

[11] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pages 26–35, New York, NY, USA, 2016. ACM.

[12] RISC-V. chisel. https://github.com/ucb-bar/chisel, 2017.

[13] RISC-V. fpga-zync. https://github.com/ucb-bar/fpga-zynq, 2017.

[14] RISC-V. riscv-fesvr. https://github.com/riscv/riscv-fesvr, 2017.

[15] RISC-V. riscv-gnu-toolchain. https://github.com/riscv/riscv-gnu-toolchain, 2017.

[16] RISC-V. riscv-linux. https://github.com/riscv/riscv-linux, 2017.

[17] RISC-V. riscv-pk. https://github.com/riscv/riscv-pk, 2017.

[18] RISC-V. riscv-tests. https://github.com/riscv/riscv-tests, 2017.

[19] RISC-V. rocket-chip. https://github.com/freechipsproject/rocket-chip, 2017.

[20] Graham Schelle, Jamison Collins, Ethan Schuchman, Perrry Wang, Xiang Zou, Gautham Chinya, Ralf Plate, Thorsten Mattner, Franz Olbrich, Per Hammarlund, Ronak Singhal, Jim Brayton, Sebastian Steibl, and Hong Wang. Intel nehalem processor core made fpga synthesizable. In *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '10, pages 3–12, New York, NY, USA, 2010. ACM.

[21] O. Shacham, O. Azizi, M. Wachs, S. Richardson, and M. Horowitz. Rethinking digital design: Why design must change. *IEEE Micro*, 30(6):9–24, Nov 2010.

[22] A. Amarnath S. Dai S. Davidson P. Gao G. Liu A. Lotfi J. Puscar A. Rao A. Rovinski L. Salem N. Sun C. Torng L. Vega B. Veluri X. Wang S. Xie C. Zhao R. Zhao C. Batten R. Dreslinski I. Galton R. Gupta P. Mercier M. Srivastava M. Taylor T. Ajayi, K. Al-Hawaj and Z. Zhang. Celerity: An open-source risc-v tiered accelerator fabric. In *2017 IEEE Hot Chips 29 Symposium (HCS)*, Aug 2017.

[23] A. Amarnath S. Dai S. Davidson P. Gao G. Liu A. Rao A. Rovinski N. Sun C. Torng L. Vega B. Veluri S. Xie C. Zhao R. Zhao C. Batten R. Dreslinski R. Gupta M. Taylor T. Ajayi, K. Al-Hawaj and Z. Zhang. Experiences using the risc-v ecosystem to design an accelerator-centric soc in tsmc 16nm. In *First Workshop on Computer Architecture Research with RISC-V, CARRV'17*, Oct 2017.

[24] VITA. Fmc. http://www.vita.com/fmc, 2017.

[25] Huy Vo. Hardware construction in chisel. Master's thesis, EECS Department, University of California, Berkeley, May 2013.

[26] M. Wachs, O. Shacham, Z. Asgar, A. Firoozshahian, S. Richardson, and M. Horowitz. Bringing up a chip on the cheap. *IEEE Design Test of Computers*, 29(6):57–65, Dec 2012.

[27] Perry H. Wang, Jamison D. Collins, Christopher T. Weaver, Blliappa Kuttanna, Shahram Salamian, Gautham N. Chinya, Ethan Schuchman, Oliver Schilling, Thorsten Doil, Sebastian Steibl, and Hong Wang. Intel®atom™processor core made fpga-synthesizable. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '09, pages 209–218, New York, NY, USA, 2009. ACM.

[28] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanović. The risc-v instruction set manual, volume i: User-level isa, version 2.1. Technical Report UCB/EECS-2016-118, EECS Department, University of California, Berkeley, May 2016.

[29] Xilinx. AMBA AXI4 interface protocol. https://www.xilinx.com/products/intellectual-property/axi.html, 2017.

[30] Xilinx. UltraScale+ FPGAs Product Tables and Product Selection Guide. https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf, 2017.