Appears in the First Workshop on Computer Architecture Research with RISC-V (CARRV 2017), August 2017

Experiences Using the RISC-V Ecosystem to Design an Accelerator-Centric SoC in TSMC 16nm

Tutu Ajayi [1] Khalid Al-Hawaj [2] Aporva Amarnath [1] Steve Dai [2] Scott Davidson [4]
Paul Gao [4] Gai Liu [2] Anuj Rao [4] Austin Rovinski [1] Ningxiao Sun [4] Christopher Torng [2]
Luis Vega [4] Bandhav Veluri [4] Shaolin Xie [4] Chun Zhao [4] Ritchie Zhao [2]
Christopher Batten [2] Ronald G. Dreslinski [1] Rajesh K. Gupta [3] Michael B. Taylor [4] Zhiru Zhang [2]

[1]Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI
[2]School of Electrical and Computer Engineering, Cornell University, Ithaca, NY
[3]Department of Computer Science and Engineering, University of California, San Diego, CA
[4]Bespoke Silicon Group, University of Washington, WA and University of California, San Diego, CA

ABSTRACT

The recent trend towards accelerator-centric architectures has renewed the need for demonstrating new research ideas in prototype systems with custom chips. Unfortunately, building such research prototypes is tremendously challenging, but the emerging RISC-V open-source software and hardware ecosystem can partly address this challenge by reducing design, implementation, and verification effort. This paper briefly describes the Celerity system-on-chip (SoC), a 5×5 mm 385M-transistor chip in TSMC 16 nm, which uses a tiered parallel accelerator fabric to improve both the performance and energy efficiency of embedded applications. The Celerity SoC includes five RV64G cores, a 496-core RV32IM tiled manycore processor, and a complex BNN (binarized neural network) accelerator implemented as a Rocket custom co-processor (RoCC). We describe our experiences using the RISC-V ecosystem to build Celerity, and highlight both key benefits and challenges in leveraging the RISC-V instruction set, RISC-V software stack, RISC-V processor and memory system generators, RISC-V on-chip network interfaces, RISC-V verification suite, and RISC-V system-level hardware infrastructure. The RISC-V ecosystem played an important role in enabling a team of junior graduate students to design and tapeout the highest-performance RISC-V SoC to date in just nine months.

1 INTRODUCTION

The field of computer architecture has a long history of building computer architecture research prototypes that implement research ideas using custom-designed chips and systems [5, 11, 15, 22]. Putting principle into practice by building prototypes is one of the best ways to validate assumptions, measure real system-level performance and efficiency, gain intuition about physical design issues, build credibility with industry, and provide platforms for future software research. Contributing to building a prototype can have a transformative impact on young researchers, and lay the foundation for new research directions. The need for building prototypes has never been more urgent owing to the rise of dark silicon which in turn has motivated an increasing trend towards accelerator-centric architectures [7, 12–14, 16, 19, 21, 24]. These architectures can include a complex heterogeneous mix of both programmable and specialized accelerators, and this means traditional simulation-based evaluation methodologies based on general-purpose processors are no longer sufficient. Building accelerator-centric prototypes is an important complement to early simulation-based design-space exploration.

However, building research prototypes with custom-designed chips can be tremendously challenging. Simply gaining access to a reasonably modern technology node can require months of legal negotiation. Acquiring and managing the diverse array of electronic design automation tools for simulation, synthesis, place-and-route, analysis, and verification can require a full-time support engineer. Gaining access to and then instantiating all of the relevant physical intellectual property (IP) blocks (e.g., standard-cell libraries, I/O cell libraries, memory generators) is frustratingly difficult. Designing a new instruction set from scratch can require significant effort, but extending an existing instruction set is often prohibited due to licensing restrictions. Designing, implementing, and verifying general-purpose processor cores and on-chip networks (OCNs) at the register-transfer level (RTL) can also require significant effort, but acquiring third-party processor/OCN IP can be expensive and may again prohibit modifications. Developing the system-level hardware infrastructure to evaluate a prototype (e.g., board design, FPGA gateware) requires tedious engineering effort. Finally, bringing up a full-featured software stack including an assembler, compiler, standard C/C++ library, and operating system can require months or years of additional effort.

RISC-V is a new open instruction set architecture (ISA) that is serving as the foundation for a rapidly developing open-source software and hardware ecosystem [3, 18]. This ecosystem includes: the RISC-V ISA specification; OCN specifications; a complete software stack for both embedded and general-purpose computing; various RISC-V processor and OCN implementations; and system-level hardware infrastructure for RISC-V processors. While the RISC-V ecosystem cannot solve all of the challenges in building research chips in academia, the hope is that this ecosystem can at least partly reduce the design, implementation, and verification effort required for building accelerator-centric prototypes. For example, the open RISC-V ISA enables researchers to easily adopt the base ISA for the portions of the prototype that are relatively standard, and then to modify and/or extend this base ISA with new research ideas (e.g., new custom instructions). A complete off-the-shelf RISC-V software stack (e.g., binutils, GCC, newlib/glibc, Linux kernel, Linux distributions) enables rapidly bringing up initial workloads on new prototypes before modifying and/or extending this software

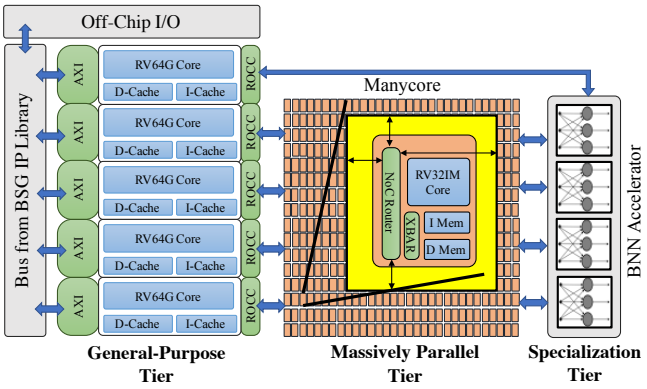

Figure 1: Celerity SoC Architecture

stack to support new software research ideas. Complete off-the-shelf RISC-V processor and memory system implementations (e.g., Rocket chip SoC generator) enable rapidly deploying traditional processors before modifying and/or extending these initial implementations with new hardware research ideas. Similarly, OCN IP that is designed within the RISC-V ecosystem (e.g., NASTI, TileLink) can reduce system-level integration effort. Standard verification test suites can greatly simplify developing new processor microarchitectures, and turn-key FPGA gateware (e.g., framework for running Rocket cores on various Xilinx Zynq FPGA boards) can help reduce engineering effort. The entire RISC-V ecosystem has a strong emphasis on open-source software and hardware which facilities modifying and/or extending just the component of interest in the context of a given research idea.

In this paper, we describe our experiences using the RISC-V ecosystem to build Celerity, an accelerator-centric system-on-chip (SoC) which uses a tiered accelerator fabric to improve energy efficiency in the context of high-performance embedded systems [1]. The *general-purpose tier* includes a few fully featured RISC-V processors capable of running general-purpose software including an operating system, networking stack, and non-critical control/configuration software. This tier is optimized for high flexibility, but of course at the cost of energy efficiency. The *massively parallel tier* includes hundreds of lightweight RISC-V processors, a distributed, non-cache-coherent memory system, and a mesh-based interconnect. This tier is optimized for efficiently executing applications with fine-grain data- and/or thread-level parallelism. The *specialization tier* includes application-specific accelerators (possibly generated using high-level synthesis). This tier is optimized for extreme energy efficiency, but of course at the cost of flexibility. We envision a three-step process for mapping algorithms to such fabrics. *Step 1:* Implement the algorithm using the general-purpose tier. *Step 2:* Accelerate the algorithm using either the massively parallel tier OR the specialization tier. *Step 3:* Improve performance and efficiency by cooperatively using both the specialization AND the massively parallel tier. A key feature of tiered accelerator fabrics is the use of high-throughput parallel links to inter-connect all three tiers.

The Celerity SoC is a 5×5 mm 385M-transistor chip in TSMC 16 nm designed and implemented by a team of over 20 students and faculty from the University of Michigan, Cornell University, and the Bespoke Silicon Group at the University of Washington and the University of California, San Diego, as part of the DARPA Circuit Realization At Faster Timescales (CRAFT) program. Figure 1 illustrates the SoC architecture. The Celerity SoC includes five Chisel-generated Rocket RV64G cores in the general-purpose tier, a 496-core RV32IM tiled manycore processor in the massively parallel tier, and a complex HLS-generated BNN (binarized neural network) accelerator implemented as a Rocket custom co-processor (RoCC) in the specialization tier. Celerity also includes tightly integrated Rocket-to-manycore communication channels, manycore-to-BNN high-speed links, sleep-mode subsystem with ten RV32IM cores, fully synthesizable phase-locked-loop clocking subsystem, and digital low-dropout voltage regulator. The chip was taped out in May 2017, and it will return from the foundry in the fall. The Celerity SoC is an open-source project, and links to all of the source files are available online at `http://opencelerity.org`.

In the rest of the paper, we describe each of the three tiers in more detail by answering four key questions: What did we build in that tier? How did we build it? How did we leverage the RISC-V ecosystem to facilitate design, implementation, and verification in that tier? and What were the challenges in leveraging the RISC-V ecosystem in that tier? Overall, the RISC-V ecosystem played an important role in enabling a team of junior graduate students to design and tapeout the highest-performance RISC-V SoC to date in just nine months.

2 GENERAL-PURPOSE TIER WITH RV64G CORES

The general-purpose tier uses fully featured RISC-V processors to execute general-purpose software. This tier is optimized for high flexibility, at the potential expense of energy efficiency.

What Did We Build? – The Celerity SoC general-purpose tier includes five RV64G cores. The RV64G instruction set is comprised of approximately 150 instructions for 64-bit integer arithmetic, single and double-precision floating-point arithmetic, memory access, unconditional and conditional control flow, and atomic memory operations. The cores use a relatively simple five-stage, single-issue, in-order pipeline. The RV64G core includes a memory management unit and support for RISC-V machine, supervisor, and user privilege levels. It can be used in either a bare-metal mode, with a proxy kernel (i.e., system calls are proxied to a separate host machine), or with a RISC-V port of the Linux operating system. The RV64G cores serve as the interface between the other tiers and the off-chip "northbridge" which is implemented as gateware in an FPGA. The northbridge includes support for initial boot-up, off-chip DRAM, and other I/O. Each RV64G core includes a 16 KB four-way set-associative instruction cache and a 16 KB four-way set-associative data cache. There is no on-chip L2 cache. The five RV64G cores are not cache-coherent, and thus only support running five independent instances of non-SMP Linux. Limited communication between the RV64G cores is possible using software-managed coherence and special support in the northbridge.

How Did We Build It? – We used the Berkeley Rocket chip SoC generator to create the RV64G core [2]. The Rocket chip SoC generator is written in Chisel [4], a hardware construction language embedded in Scala. Generating an RV64G core simply required

setting the appropriate configuration options and running the generator to create the corresponding SystemVerilog RTL. This RTL could then be integrated into the rest of the SoC using standard SystemVerilog RTL design methodologies. We chose a very specific commit of the Rocket chip SoC generator which corresponded to a recent tapeout by U.C. Berkeley to increase the chances of a fully functional core design. We leveraged the open-source BaseJump hardware component library [20] including the BaseJump front-side bus (a pipelined top-level SoC interconnect) and the BaseJump high-speed FPGA bridge (a high-speed source-synchronous DDR off-chip I/O interface). The BaseJump RV-IOV [23] I/O virtualization package (`http://bjump.org/rv-iov`) tunneled the five AXI4-like Not A STandardized Interface (NASTI) and host interfaces over the BaseJump FSB to the BaseJump FPGA Bridge and to the northbridge. For verification, we modified the open-source RISC-V framework for running Rocket cores on various Xilinx Zynq FPGA boards; our adapters converted BaseJump back into NASTI within the northbridge to then interface with the RISC-V Zynq FPGA framework. This infrastructure enabled full-system simulation to verify the Celerity SoC in-situ with the northbridge. We made extensive use of the RISC-V assembly test suite to help verify our designs, before moving to more complex C-based tests. A RV64G core was pushed through the standard-cell-based ASIC toolflow to create a hard-macro which was then instantiated five times in the SoC.

RISC-V Ecosystem Successes – Leveraging the RISC-V ecosystem paid significant dividends within the general-purpose tier. We were able to start from a well-specified instruction set, and just as importantly, a high-quality reference implementation. By using a core generator as opposed to a specific design instance, we were able to configure the core with the features we wanted (e.g., floating-point support, specific cache configuration). We were also able to turn to the broader RISC-V community for guidance on specific design issues. For example, we knew from public discussion of previous RV64G prototypes from U.C. Berkeley that the critical path was likely to be through the floating-point unit (FPU), so we were able to register retime the FPU to replicate U.C. Berkeley's previous success without much further effort on our part. We also used spike, the RISC-V ISA simulator, to count instructions of programs early on in the project to help estimate the expected run-times of the longest programs we hoped to run in RTL and gate-level simulation. NASTI provided a relatively simple OCN interface, and this enabled straight-forward integration with other IP. The ability to leverage the RISC-V Zynq FPGA framework to quickly create a full-featured verification environment supporting both bare-metal and proxy kernel operation was also a significant benefit. Being able to reuse the RISC-V assembly test suite simplified verification, and being able to use a mostly unmodified software stack greatly simplified porting applications. Several of the faculty involved in the project have had experience using modified-MIPS instruction sets, ARM instruction sets, and ARM IP in academic research prototypes. The RISC-V ecosystem provided a nice balance between the extensibility offered when using a modified-MIPS approach vs. the standardized software environment offered when using an ARM approach.

RISC-V Ecosystem Challenges – However, there were some challenges in using the RISC-V ecosystem in the general-purpose tier. Perhaps one of the most significant challenges was simply the rapid pace of development both within each RISC-V project and across the full RISC-V ecosystem. For example, the migration from NASTI to TileLink forced us to use an older version of the Rocket chip SoC generator, since the TileLink interface and implementation were rapidly changing. The bleeding edge version of the many RISC-V projects failed to work correctly together, and it was difficult to find a consistent view across all of the projects which ensured we were starting with a fully functioning ecosystem. Documentation across RISC-V projects is either lacking or completely missing. While Chisel certainly enables a powerful approach to hardware generation, it also presents a significant barrier to adoption. Ultimately, we were forced to minimize any changes to the RISC-V cores used in the general-purpose tier simply because we did not have time to become experts in Chisel.

One key challenge related to the verification methodologies and invariants used in the Rocket chip. The generated RTL explicitly used random state initialization to avoid any source of X-pessimism, and there were a few sources of microarchitectural non-determinism (e.g., front-end timing depended on the potentially-invalid contents of the data cache tags on squashed load or store instructions, the branch history table was not initialized on reset). These issues essentially prevented true four-state RTL simulation, which is important for robustly verifying the SoC comes out of reset correctly. In addition, the Chisel-generated non-blocking data cache generated simultaneous reads and writes to the same address in the dual-ported SRAMs which complicated the process of mapping to physical SRAMs. Finally, a few X-pessimism issues resulted from reconvergence of combinational logic. We were able to fix all of these issues by modifying Rocket's Chisel code.

3 MASSIVELY PARALLEL TIER WITH RV32IM CORES

The massively parallel tier uses hundreds of lightweight RISC-V processors to exploit fine-grain data- and thread-level parallelism. This tier is optimized to provide a balance of flexibility vs. efficiency.

What Did We Build? – The Celerity SoC massively parallel tier uses the open source BaseJump Manycore design (`http://bjump.org/manycore`), instantiating 496 lightweight RV32IM cores to form a manycore processor. The 496 RV32IM cores are 8× more dense than the Rocket cores. The cores use a five-stage, single-issue, fully forwarded, in-order pipeline. The cores do not use interrupts or a translation lookaside buffer (TLB). The manycore uses a fully distributed memory system with no caching; each RV32IM core includes a 4 KB instruction memory and a 4 KB data memory. The RV32IM cores are interconnected using a mesh on-chip network with XY-dimension-ordered routing, credit-based flow control, and 80 Gb/s full-duplex channels between adjacent cores. The manycore is highly parameterizable and is a general framework to efficiently stitch together heterogeneous cores and/or small accelerators.

The manycore provides support for a heterogeneous remote-store programming model. Each core can store to any other core's data memory but can only load from its own local data memory. It extends the RV32IM instruction set by adding new instructions to facilitate synchronization in the context of the remote-store programming model. The manycore is directly connected to four of the

RV64G cores in the general-purpose tier through the RoCC command interface. This interface allows RV64G cores to use custom RISC-V instructions to read/write instructions/data in the manycore through the mesh network. For example the four RV64G cores can write a manycore program into each RV32IM core's instruction memory, write initial data into each RV32IM core's data memory, signal the manycore to begin computation, and then use the same RoCC command interface to retrieve the results of the computation.

How Did We Build It? – The BaseJump Manycore RV32IM core implementation leverages the BaseJump STL library (`http://bjump.org/stl`), a comprehensive library of parameterized SystemVerilog components that raises the level of abstraction of hardware design, both in the RV32IM core and the on-chip mesh network. We adapted the RV32IM assembly test suite from the V-scale project to help verify the RV32IM core in isolation. The manycore's on-chip network was tested in isolation using network traffic generators, and numerous C-programs for integration testing. Each RV32IM core and associated network interface was pushed through the standard-cell-based ASIC toolflow to create a hard-macro, and then this macro was instantiated 496 times to create the manycore processor.

RISC-V Ecosystem Successes – Leveraging the RISC-V ecosystem continued to provide benefits within the massively parallel tier. Again, starting from a well-defined instruction set simplified the core specification and enabled reusing the RISC-V assembler and C compiler. Reusing the V-scale assembly tests reduced verification effort. We also found it very useful to leverage the Berkeley RoCC command interface and associated custom instructions already included within the RV64G cores in the general-purpose tier. This enabled us to connect the general-purpose and massively parallel tiers without any modifications to the RV64G core microarchitecture. The RISC-V instruction set was designed with extensibility as a key design goal, and we found this to greatly simplify adding new custom instructions for remote-store programming.

RISC-V Ecosystem Challenges – The RISC-V ecosystem provided less benefit in the massively parallel tier compared to the general-purpose tier. We had hoped to reuse an open-source RV32IM implementation. We considered using the Z-scale or V-scale RV32IM implementations, but neither was being actively maintained. We also considered using the Rocket chip SoC generator to create a lightweight RV32IM core, but the generator could not meet our design requirements (e.g., generating instruction/data scratchpads instead of caches). Chisel was an even higher barrier to adoption, since we expected to make more extensive modifications to the RV32IM core. Ultimately, we had no choice but to implement the RV32IM core in-house.

Due to the sparse and scattered nature of RoCC interface documentation, we invested significant effort reverse engineering RoCC examples to understand this interface. To address this issue, we wrote and made public a RoCC user guide and a more sophisticated RoCC example, along with examples that show how to integrate SystemVerilog accelerators with RoCC. The RoCC user guide and examples are available at `http://bjump.org/rocc_doc`. One subtle challenge involved the Rocket chip SoC generator's instantiation of a dedicated RoCC co-processor module inside the Rocket core. For most SoC designs, it is more convenient to connect these blocks at the top-level. This allows reuse of the Rocket hard macro even when the Rockets are connected to different kinds of accelerators, or multiple Rockets are connected to a single accelerator. To address this issue, we modified the Rocket chip SoC generator to expose the RoCC interface at the top-level of the RV64G core. This enabled composition of multiple RV64G cores, the manycore, and the BNN within SystemVerilog.

4 SPECIALIZATION TIER WITH ROCC ACCELERATORS

The specialization tier uses application-specific accelerators to achieve extreme energy efficiency, but of course at the cost of flexibility.

What Did We Build? – Deep convolutional neural networks (CNNs) are now the state-of-the-art for image classification, detection, and localization tasks. However, using CNN software implementations for real-time inference in embedded platforms can be challenging due to strict power and memory constraints. This has motivated significant interest in hardware acceleration for CNN inference. Most of the prior work still requires large traditional on- and off-chip memories to store fixed-point weights and activations and carefully hand-crafted digital VLSI architectures [6, 8, 25]. Recent work on binarized neural networks (BNNs) have demonstrated that binarized weights and activations (i.e., +1, -1) can, in certain cases, achieve accuracy comparable to full-precision floating-point CNNs [9, 10, 17]. We have recently explored FPGA-based BNN accelerators [26], which motivated our interest in implementing a BNN accelerator in the specialization tier of the Celerity SoC.

We use the CIFAR-10 dataset to drive the design of our BNN accelerator, although our accelerator is flexible enough to be used for other similar small-image classification problems. Figure 2 illustrates the specific model we used for CIFAR-10 classification and includes six convolutional, three max pooling, and three dense (fully connected) layers. The input image is quantized to 20-bit fixed-point, and the first convolutional layer takes this representation as input. All remaining layers use binarized weights and feature maps. BNN-specific optimizations include eliminating the bias, reducing the complexity of the batch norm calculation, and carefully managing convolutional edge padding. This network can achieve 89.8% accuracy on the CIFAR-10 dataset. Figure 3 shows the BNN accelerator architecture. The BNN accelerator consists of a module for fixed-point convolution (i.e., first layer), a module for binarized convolution, a module for processing dense layers, weight and feature-map buffers, and a direct memory access engine to move data in and out of the buffers. The BNN accelerator processes one layer of one image at a time before moving onto the next layer. Any non-binarized computation is performed completely within each module to limit the amount of non-binarized intermediate data stored in the accelerator buffers and/or memory system. The BNN accelerator was pushed through the standard-cell-based ASIC toolflow to create a hard-macro, and this macro was instantiated along with the RV64G and RV32IM macros at the top level.

As with the massively parallel tier, a RV64G core is directly connected to the BNN accelerator through the RoCC command interface. In addition, the BNN accelerator makes use of the RoCC memory interface to read the input image and per-layer binarized weights from the memory subsystem in the general-purpose tier.

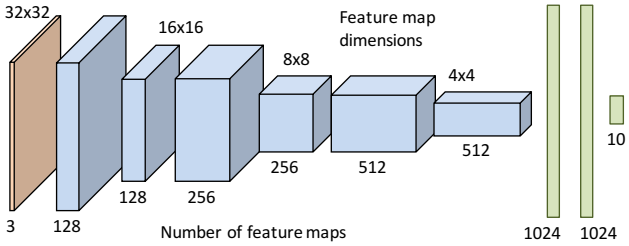

Figure 2: BNN CIFAR-10 Network

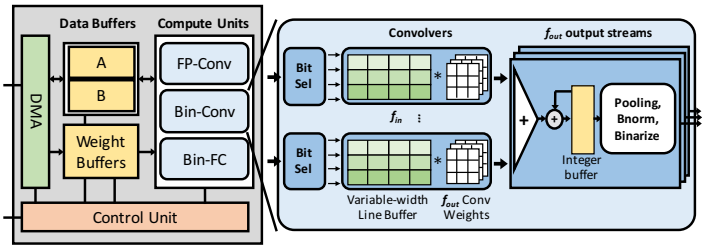

Figure 3: BNN Accelerator Architecture

The 14 Mb of binarized weights cannot fit in the general-purpose memory system resulting in non-trivial performance and energy overhead. To address these issues, we implemented a novel technique where the BNN accelerator can use the storage in the massively parallel tier as a software programmable streaming scratchpad. The RV64G cores first load all of the weights into the manycore. We then execute a simple program on the manycore which causes each RV32IM core to sequentially stream weights from its local memory to the BNN using the remote-store programming model. The BNN can be configured to retrieve its weights either from the general-purpose tier or the massively parallel tier. We used detailed post-place-and-route gate-level simulation to compare the performance of the various tiers in the Celerity SoC. Our BNN accelerator in isolation improves performance by 200× over an optimized software running on a RV64G core, and cooperatively using both the BNN accelerator and manycore improves performance by 1,220× and performance per Watt by 250× over the software baseline.

How Did We Build It? – The tight design timeline and the need to perform extensive design-space exploration to effectively map an emerging application onto an accelerator inspired us to use a high-level synthesis (HLS) methodology. The architecture shown in Figure 3 was written completely in SystemC and then synthesized into RTL using the Cadence StratusHLS tool. An HLS methodology had the added benefit of reducing simulation time since significant verification could take place using SystemC. HLS tools also enabled rapid iteration on timing closure, e.g., we were able to improve the clock frequency by 43% in just a few days by aggressively pushing the tools. This HLS-based design methodology enabled three graduate students with near-zero neural network experience to rapidly design, implement, and verify a complex application-specific accelerator.

RISC-V Ecosystem Successes – Even though the specialization tier did not include any general-purpose processors, we still found significant benefit in using the RoCC command and memory interface to interconnect the BNN accelerator with the RV64G core. As in the massively parallel tier, we were able to connect the accelerator with no changes to the RV64G core.

RISC-V Ecosystem Challenges – We faced similar challenges related to the RoCC interface as mentioned in the massively parallel tier. One key new challenge related to how a RoCC accelerator interacted with the memory management unit in the RV64G core. This interaction was poorly documented, and we eventually discovered that the RoCC interface in the older Rocket chip SoC generator uses physical addresses. Modifying the RV64G core to support memory translation and protection through the RoCC interface would require significant design effort. We used software workarounds to enable RoCC accelerators to operate correctly in both bare-metal mode and with the proxy kernel, but using RoCC accelerators in Linux will require more work. We are encouraged to see that many of these issues have been addressed in more recent versions of Rocket chip SoC generator.

5 CONCLUSIONS

The Celerity SoC is a 5 × 5 mm 385M-transistor chip in TSMC 16 nm which includes five Chisel-generated Rocket RV64G cores, a 496-core RV32IM tiled manycore processor, complex HLS-generated BNN (binarized neural network) accelerator implemented as a Rocket custom co-processor (RoCC), high-speed links between the RV64G, manycore, and BNN accelerator, and sleep-mode subsystem with ten RV32IM cores. The Celerity SoC is an open-source project, and links to all of the source files are available online at `http://opencelerity.org`.

The RISC-V ecosystem played an important role in enabling a team of junior graduate students to design and tapeout a complex RISC-V SoC in just nine months. We were able to leverage the RISC-V instruction set, RISC-V software stack, RISC-V processor and memory system generators, RISC-V on-chip network interfaces, RISC-V verification suite, and RISC-V system-level hardware infrastructure. While ultimately a success, we still faced some challenges including limited documentation, lack of reference implementations in an industry standard hardware description language, and the lack of a stable release schedule across the entire ecosystem.

ACKNOWLEDGMENTS

This research was supported in part by DARPA CRAFT Award HR0011-16-C-0037 and NSF CRI Award #1059333, NSF CRI Award #1512937, NSF SaTC Award #1563767, and NSF SaTC #1565446. We thank Synopsys, Cadence, and Mentor for EDA tool donations, ARM for physical IP donations, and Xilinx for both EDA tool and FPGA development board donations. We thank Ian Galton, Julian Puscar, Patrick Mercier, and Loai Salem for designing the analog portions of the Celerity chip. We thank the many contributors to the open-source RISC-V software and hardware ecosystem. We also acknowledge U.C. Berkeley's contributions to forming the RISC-V ecosystem, and for developing the the Rocket chip SoC generator and RoCC interface.

REFERENCES

[1] T. Ajayi, K. Al-Hawaj, 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, and R. Zhao. Celerity: An Open-Source RISC-V Tiered Accelerator Fabric. *Symp. on High Performance Chips (Hot Chips)*, Aug 2017.

[2] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman. The Rocket Chip Generator. Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley, Apr 2016.

[3] K. Asanovic and D. 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.

[4] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avizienis, J. Wawrzynek, and K. Asanović. Chisel: Constructing Hardware in a Scala Embedded Language. *Design Automation Conf. (DAC)*, Jun 2012.

[5] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff. OpenPiton: An Open Source Manycore Research Framework. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Apr 2016.

[6] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar 2014.

[7] Y. Chen, S. Lu, C. Fu, D. Blaauw, R. D. Jr, and T. M. H.-S. Kim. A Programmable Galois Field Processor for the Internet of Things. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2017.

[8] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *Int'l Solid-State Circuits Conf. (ISSCC)*, Feb 2016.

[9] M. Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training Deep Neural Networks with Binary Weights During propagations. *Conf. on Neural Information Processing Systems (NIPS)*, Dec 2015.

[10] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv*, arXiv:1602.02830, Feb 2016.

[11] R. G. Dreslinski, D. Fick, B. Giridhar, G. Kim, S. Seo, M. Fojtik, S. Satpathy, Y. Lee, D. Kim, N. Liu, M. Wieckowski, G. Chen, D. Sylvester, D. Blaauw, and T. Mudge. Centip3De: A 64-Core, 3D Stacked Near-Threshold System. *IEEE Micro*, 33(2):8–16, Mar/Apr 2013.

[12] N. Goulding, J. Sampson, G. Venkatesh, S. Garcia, J. Auricchio, J. Babb, M. Taylor, and S. Swanson. GreenDroid: A Mobile Application Processor for a Future of Dark Silicon. *Symp. on High Performance Chips (Hot Chips)*, Aug 2010.

[13] M. Khazraee, L. Zhang, L. Vega, and M. Taylor. Moonwalk: NRE Optimization in ASIC Clouds or, Accelerators Will Use Old Silicon. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Apr 2017.

[14] J. Kim, S. Jiang, C. Torng, M. Wang, S. Srinath, B. Ilbeyi, K. Al-Hawaj, and C. Batten. Using Intra-Core Loop-Task Accelerators to Improve the Productivity and Performance of Task-Based Parallel Programs. *Int'l Symp. on Microarchitecture (MICRO)*, Oct 2017.

[15] R. Krashinsky, C. Batten, and K. Asanović. Implementing the Scale Vector-Thread Processor. *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, 13(3), Jul 2008.

[16] I. Magaki, M. Khazraee, L. Vega, and M. Taylor. ASIC Clouds: Specializing the Datacenter. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2016.

[17] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. *arXiv*, arXiv:1603.05279, Mar 2016.

[18] RISC-V Foundation. `http://www.riscv.org`, 2017 (accessed Aug 15, 2017).

[19] S. Srinath, B. Ilbeyi, M. Tan, G. Liu, Z. Zhang, and C. Batten. Architectural Specialization for Inter-Iteration Loop Dependence Patterns. *Int'l Symp. on Microarchitecture (MICRO)*, Dec 2014.

[20] M. Taylor et al. BaseJump: Open-Source Components for ASIC Prototypes. `http://bjump.org`, 2017.

[21] M. B. Taylor. Is Dark Silicon Useful? Harnessing the Four Horsemen of the Coming Dark Silicon Apocalypse. *Design Automation Conf. (DAC)*, Jun 2012.

[22] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffmann, P. Johnson, W. Lee, A. Saraf, N. Shnidman, V. Strumpen, S. Amarasinghe, , and A. Agarwal. A 16-Issue Multiple-Program-Counter Microprocessor with Point-to-Point Scalar Operand Network. *Int'l Solid-State Circuits Conf. (ISSCC)*, Feb 2003.

[23] L. Vega and M. Taylor. RV-IOV: Tethering RISC-V Processors via Scalable I/O Virtualization. *Workshop on Computer Architecture Research with RISC-V (CARRV)*, Oct 2017.

[24] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor. Conservation Cores: Reducing the Energy of Mature Computations. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar 2010.

[25] P. N. Whatmough, S. K. Lee, H. Lee, S. Rama, D. Brooks, and G.-Y. Wei. A 28nm SoC with a 1.2GHz 568nJ/Prediction Sparse Deep-Neural-Network Engine with >0.1 Timing Error Rate Tolerance for IoT Applications. *Int'l Solid-State Circuits Conf. (ISSCC)*, Feb 2017.

[26] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang. Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs. *Int'l Symp. on Field Programmable Gate Arrays (FPGA)*, Feb 2017.