Evaluation of RISC-V RTL with FPGA-Accelerated Simulation

Donggyu Kim, Christopher Celio, David Biancolin, Jonathan Bachrach, Krste Asanoviċ
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
{dgkim, celio, biancolin, jrb, krste}@eecs.berkeley.edu

ABSTRACT

This paper presents a fast and accurate simulation methodology for performance, power, and energy evaluation in the hardware/software co-design flow. Cycle-level microarchitectural software simulation is the bottleneck of the hardware/software co-design cycle due to its slow speed and the difficulty of simulator validation. While sampling methodologies can ameliorate some of these challenges, we show that it is often insufficient for rigorous design evaluations. To circumvent the limitations of software simulation and sampling, we employ MIDAS, which automatically generates FPGA-accelerated simulators from RTL. These simulators are not only up to three orders-of-magnitude faster than existing microarchitectural software simulators, but also truly cycle-accurate, as the same RTL is used to build the silicon implementation. MIDAS builds on the work of Strober, namely by increasing simulator execution rate (up to tenfold) and by including an abstract L2 cache model to simulate more realistic systems without the corresponding RTL implementations. We simulated the productized, in-order processor, Rocket, and the industry-competitive, out-of-order processor, BOOM. To our knowledge, this is the first paper to present performance, power, and energy evaluations from *RTL simulations running the whole SPEC2006int benchmark suite with its reference inputs.*

1 INTRODUCTION

Performance evaluation for next-generation hardware designs is a vital component of the system design process. To do so, cycle-level microarchitectural software simulators (e.g. [3, 15, 22]) are widely used by computer architects. However, these simulators are too slow to run real-world applications without sampling. Moreover, various kinds of errors can be introduced because of the abstractions used by these simulators [9].

Simulation sampling is another popular complement to microarchitectural software simulators. Phase-based sampling with basic block analysis [17] does not provide statistically-guaranteed error bounds. However, statistical sampling provides confidence intervals, but special care is necessary to address microarchitectural state warming problems [23]. To make matters worse, simulation sampling can be misleading for performance evaluations of managed-language applications whose execution paths depend on microarchitectural state of the system.

On the other hand, if RTL designs are available, they can be used to accurately evaluate not only cycle-level performance but also cycle time, area, and power using commercial CAD tools without introducing modeling errors. However, evaluating RTL designs using software simulation is very slow, preventing design space exploration of various hardware configurations with realistic software applications.

In this paper, we present a simulation methodology to evaluate RTL designs running real-world software, taking advantage of the RISC-V infrastructure. Unlike the previous simulation methodologies, neither simulation sampling nor more abstract modeling is adopted. Instead, we adopt the methodology described by Kim et al. in Strober [10]. First, FPGA-accelerated simulators are automatically generated from RTL designs. These simulators are up to three-orders-of-magnitude faster than existing cycle-level microarchitectural software simulators. Then, we take random RTL-state sample snapshots from the FPGA simulation, which are replayed in RTL or gate-level software simulation for power estimation.

The main contributions of this paper are as follows:

- We demonstrate the importance of FPGA-accelerated simulation of RTL designs in agile hardware design methodologies [11]. This is because microarchitectural software simulators are insufficiently fast and accurate for both traditional microprocessors and custom hardware accelerators running various kinds of software applications.
- We present an open-source FPGA-simulation methodology, MIDAS. [1] In MIDAS, we improve Strober [10] by increasing the speed of generated simlators rate by up to ten times. This enables designers to evaluate the hardware designs with the whole execution of realistic benchmark suites. We also include an abstract L2-cache model to simulate more realistic systems even without the corresponding RTL implementations.
- We present a case study in which we evaluate the performance, component-wise power, and the energy of a productized in-order processor, Rocket [1], and an industry-competitive superscalar out-of-order processor, BOOM [7], using their RTL implementations by running the SPEC2006int benchmark suite with its reference inputs.

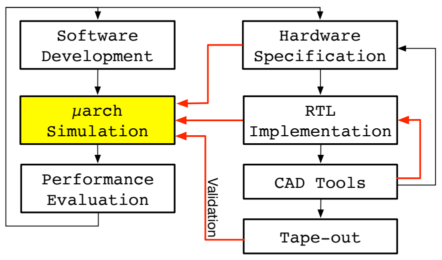
2 DESIGN EVALUATION PRACTICE

This section proposes a new design evaluation practice in the hardware/software co-design flow. Section 2.1 explains why performance, power, and energy evaluation using RTL designs instead of microarchitectural simulators is necessary for the recent hardware/software co-design trends. Section 2.2 demonstrates why the complete execution of real-world applications instead of simulation sampling is required for accurate evaluations in hardware/software co-optimizations.

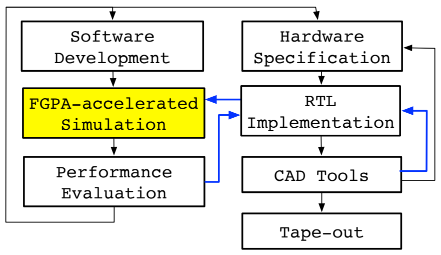
2.1 Microarchitectural Simulation vs. FPGA-Accelerated RTL Simulation

Figure 1 shows two alternatives for performance evaluation in the computer system design process. Historically, cycle-level microarchitectural software simulation (e.g. [3, 15, 22]) has been widely-used by computer architects for performance evaluation (Figure 1a). Whereas RTL implementation has been tedious, labor-intensive and

[1]Beta v0.1 is available at https://github.com/ucb-bar/midas



(a) Performance Evaluation using Microarchitectural Simulation.



(b) Performance Evaluation using RTL Designs.

Figure 1: Performance Evaluation Methodologies in the Hardware/Software Co-design Flow.

difficult to modify and verify, microarchitectural software simulators are flexible and easy to use. For this reason, high-level software simulation remains an important tool to guide system design in the early stages of system design, before RTL design has commenced. Ease-of-use concerns notwithstanding, evaluating RTL designs remains extremely slow when using commercial CAD tools, and thus, high-level simulation was necessary to evaluate the target systems with real-world applications.

However, *microarchitectural software simulation becomes the bottleneck of the recent hardware/software co-design cycles* for two main reasons. First of all, microarchitectural software simulators should be carefully validated against RTL designs and real systems. This is only feasible when new designs to not differ tremendously from either existing hardware, or similar designs from previous design cycles that have already been validated. As shown in Figure 1a, there are frequent feedback loops from the CAD tools to the RTL designs in the hardware design cycle in order to improve the quality of the silicon implementation. Whenever any changes are made to the RTL designs, microarchitectural software simulators should also be carefully tuned. Moreover, microarchitectural software simulators should be exhaustively validated against the silicon implementations running real-world software. Otherwise, various kinds of evaluation errors can be introduced by the abstraction and the modeling of the target systems [9]. Recent hardware design trends are moving toward heterogeneous SoCs with a plethora of custom hardware accelerators, has made simulator validation more difficult as it has become harder to find an existing system against which to validate the simulator.

To make matters worse, microarchitectural software simulation is too slow to run the full execution of the today's realistic workloads with very complicated hardware designs. In general, any non-trivial modern application executes trillions of dynamic instructions, making them practically impossible to simulate even with the fastest microarchitectural software simulator. For example, it takes roughly a month to simulate one trillion instructions with a 400 KIPS simulator [15]. Therefore, the slow speed of microarchitectural software simulation prevents an agile hardware development approach [11], which is required for rapid hardware/software co-optimizations.

In this paper, we propose *FPGA-accelerated RTL simulation* to resolve the difficulty of performance, power, and energy evaluation for the hardware/software co-design flow as shown in Figure 1b. First of all, RTL implementation is becoming more productive for computer architects thanks to various hardware construction languages (e.g. [2], [14], [16], [13], [19]) that greatly improve the expressivity over existing hardware description languages like Verilog and VHDL. In addition, there are an increasing number of open-source RISC-V RTL designs (e.g. [1, 7]) with which to bootstrap a new project, dramatically reducing RTL development time.

Next, this methodology is truly cycle-accurate as the FPGA-accelerated simulators are generated using automatic transformations on the same RTL design that will be consumed by VLSI CAD tools to produce he silicon implementations through the CAD tools. These transformations preserve the RTL behavior of the design, and thus do no introduce modeling errors. Thus, system designers do not need to re-validate their simulator after an RTL design change, greatly easing hardware / software co-optimization, where the RTL design may be frequently modified.

Finally, using FPGAs for performance evaluation increases the simulation speed by multiple orders of magnitude over existing cycle-level microarchitectural software simulation, which is well demonstrated by the previous work [10, 18]. Since FPGA-accelerated simulators can execute tens of MIPS, it becomes possible to evaluate complete benchmark suites, like SPEC2006int, on a cycle-accurate model of the design.

2.2 Simulation Sampling vs. Full Execution

Simulation sampling is a popular technique to overcome the sluggishness of cycle-level microarchitectural software simulators. There are two major approaches: phase-based sampling [17] and statistical sampling [23].

Phase-based sampling [17] provides simulation points through phase analysis on dynamic basic-block traces. Here, the underlying assumption is the dynamic execution of software consists of short periods of phases and each phase will exhibit similar microarchitectural behavior, and thus IPC, whenever repeated. The simulation points produced by this methodology are those centered about the basic blocks that are most frequently visited. This approach provides no guaranteed error bounds.

However, this is not necessarily true because the periods of phases can be lengthy and the performance characteristics of each phase depend on the dynamic state of the systems. Figure 2 shows a periodic sampling average of IPC every 100-million cycles over the entire execution of `401.bzip2` in the SPEC2006int benchmark suite with its reference input, running on BOOM-2w (Table 1). Even though there are some distinct execution phases, each phase has a non-trivial length period and shows different performance

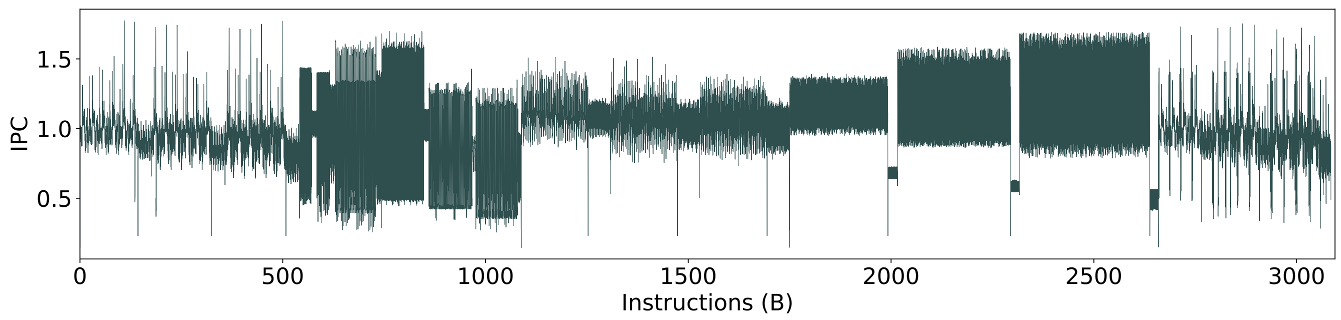


Figure 2: A periodic sampling of the IPC of BOOM-2w (see Table 1) running `401.bzip2`. A sample was collected every 100 million cycles: each point represents the average IPC over that 100 million cycle interval. Samples were collected from the FPGA as the simulation executed.

characteristics when repeated. Thus, *running a couple of million instructions in dozens of representative points from phase analysis is not enough to characterize the performance of real-world applications.*

On the other hand, statistical simulation sampling [23] provides performance estimates with statistically-bounded errors. The main idea is the intervals between detailed simulation points are fast-forwarded using fast functional simulation. However, for this approach to be effective, the microarchitectural state of the machine must be reconstituted in a "warming" phase.This requires considerable execution time in the detailed simulator even before evaluating the sample point.

Various methods (e.g. [22]) have been suggested to address this state warming problem, most of which propose keeping track of some microarchitectural state in functional simulation. Of course, this slows down the fast functional simulation, making it difficult to regenerate samples points of the machine. Thus, architects must choose between either the increased design iteration time to perform sample recollection, or potential simulation inaccuracy that may result from using stale samples. represent the machine. Finally, it is not clear what subset of state should be warmed for non-traditional hardware designs such as custom hardware accelerators.

In conclusion, there are often cases simulation sampling is not the right tool for design evaluation in the hardware/software co-design flow. In these cases, we suggest *the complete simulation of realistic workloads with fast and cycle-accurate FPGA-accelerated simulator generated from RTL designs* for performance, power, and energy evaluation.

3 SIMULATION METHODOLOGY

This section explains how MIDAS is used in performance, power, and energy evaluations of an RTL design. Section 3.1 shows how an arbitrary RTL design is mapped to an FPGA-hosted simulation model. Section 3.2 then explains how this is composed with CPU-hosted software models, and abstract FPGA-hosted timing models within a FPGA-accelerated simulator.

3.1 FPGA-Accelerated Simulators from RTL

For the performance evaluation methodology suggested in Section 2 to be feasible, we improve Strober [10] to automatically generate an FPGA-accelerated simulator from any RTL design. Any MIDAS-generated simulator is a *FAME1* simulator [18] whose cycle-level timing is modeled as decoupled timing-token exchanges between simulation models.

We use FIRRTL compiler [12] passes to transform RTL into an FPGA-hosted model. Unlike Strober [10], whose passes were embedded in the Chisel2 compiler, our methodology can be used on designs written any HDL that can be mapped to FIRRTL. The passes themselves, however, remain similar. As shown in Figure 3, *macro mapping* optionally maps technology-independent memory blocks to technology-dependent macro blocks for power estimation. A *FAME1 transform* is performed to decouple the target clock from the host clock. In addition, scan chains are optionally added to capture state snapshots for replays in software RTL / gate-level simulation for power estimation. Next, *simulation mapping* is applied to augment the logic for timing token communications. Finally, *platform mapping* adds platform-specific logic to the simulation modules.

For power estimation, we randomly sample 50 RTL state snapshots from FPGA-accelerated simulation. The target designs are synthesized [2] with the Synopses 32nm Educational Technology using the scripts generated by an open source CAD tool flow [8]. We replay the snapshots in software RTL simulation. [3] The post-synthesis design and the RTL signal activities from the sample replays are provided to power analysis tool. [4] Note that unlike statistical sampling for microarchitectural simulation [23], our methodology has no state warming problems in gate-level simulation as the exact RTL state snapshots are taken from FPGA simulation.

The details are also described by Kim et al. [10]

3.2 Mapping Simulation to the FPGA Host

Another challenge is how to map heterogeneous simulation models, including the transformed-from-RTL models, software models, and abstract timing models, to the FPGA host platform for fast simulation. Figure 4 shows how the target designs are mapped to the FPGA host platform. We use the Xilinx Zynq ZC706 boards for the case study but other FPGA platforms can be used in principle.

[2]Synopsys Design Compiler®Version L-2016.03-SP1.
[3]Synopsys VCS®Version L-2016.06-1.
[4]Synopsys PrimeTime® PX Version K-2015.12-SP3.

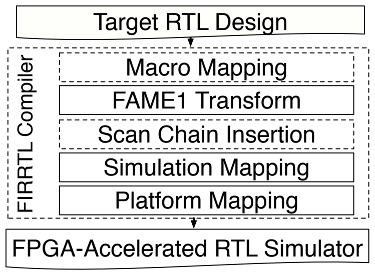


Figure 3: Custom Compiler Passes to Generate FPGA-Accelerated RTL Simulators.

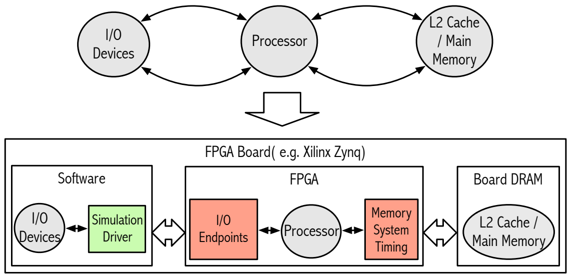


Figure 4: Target Design Mapping to the FPGA Host Platform.

The processor models are generated from RTL designs and mapped to the FPGA (Section 3.1). The I/O devices are modeled in software and run alongside the simulation driver as their transactions are infrequent. Kim et al. [10] report the simulation rate is 3.56 MHz, which is not fast enough to run real-world workloads to completion, due to the communication overhead between the CPU-hosted software models and the FPGA-hosted RTL models. We optimize the communications by having *I/O endpoints*, special hardware widgets that translate timing tokens to high-level transactions and vice versa. In other words, instead of exchanging low-level timing tokens, the I/O devices efficiently communicate the processor with high-level transactions through I/O endpoints. As a result, the average simulation rate of BOOM-2w is *18 MIPS* on average for the SPEC2006int benchmark suite.

We implement an abstract timing model for L2 caches and DRAM hosted on the FPGA, while the actual data are hosted on the board main memory. Using an abstract model for the L2 cache was compelling for three reasons. Firstly, the current version of RocketChip does not generate L2 caches, it is easier at first to write an abstract L2 models, than implemented a complete L2 cache. Two, abstract models can provide runtime configurable parameters with low overhead. In our case, the size, associativity, and the latency of L2 caches can be reconfigured without needing to recompile an FPGA bitstream. Finally, since we model the timing of the memory systems by only keeping the tags of L2 caches on the FPGA, we can model an L2 cache that would be too large to fit on our FPGA. However, the downside is the simulation needs to stall even with L2 cache hits to obtain the actual data from the board main memory, slightly slowing down the simulation rate. Of course, using an

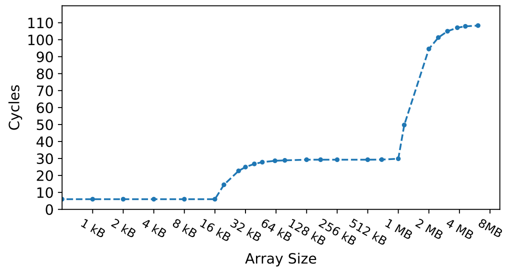


Figure 5: Memory System Timing Validation of BOOM-2w with a 16 KiB L1 Data Cache. A pointer-chase through increasing sizes of arrays demonstrates the load-to-load latency of different levels of the memory hierarchy.

abstract model introduces the aforementioned modeling errors and validation challenges.

Figure 5 shows the timing validation of the memory systems using a pointer-chase benchmark [4]. The target design observes the 16KiB L1 data cache (6 cycles), the 1MiB L2 data cache (6 + 23 cycles), and the main memory (6 + 23 + 80 cycles) as configured in Table 1.

4 TARGET DESIGNS AND BENCHMARKS

Parameter	Rocket	BOOM-2w
Fetch-width	1	2
Issue-width	1	3
Issue slots	-	20
ROB size	-	80
Ld/St entries	-	16/16
Physical registers	32(int)/32(fp)	110
Branch predictor	-	gshare: 16 KiB history
BTB entries	40	40
RAS entries	2	4
MSHR entries	2	2
L1 $ capacities	16 KiB or 32 KiB	
ITLB and DTLB reaches	128 KiB / 128 KiB	
L2 $ capacity and latency	1 MiB / 23 cycles	
DRAM latency	80 cycles	

Table 1: The Parameters for the Two Target Processors.

Section 4.1 explains RocketChip [1] and BOOM [7] used as the target hardware designs in this paper. Section 4.2 covers the SPEC2006int benchmark suite used for the case study.

4.1 Target Designs

RocketChip [1] is an open-source SoC generator suitable for research and industrial purposes. Rather than being a single instance

Benchmarks	Instructions (T)	Benchmarks	Instructions (T)
400.perlbench	2.48	*458.sjeng*	2.85
401.bzip2	3.08	*462.libquantum*	2.09
403.gcc	1.37	*464.h264ref*	5.07
429.mcf	0.29	*471.omnetpp*	0.61
445.gobmk	2.04	*473.astar*	1.05
456.hmmer	2.95	*483.xalanbmk*	1.10

Table 2: Dynamic Instruction Counts with RISC-V ISA.

of an SoC design, RocketChip is a hardware design generator, capable of producing many design instances from a single piece of Chisel [2] source code. Multiple industry products as well as silicon prototypes are manufactured using RocketChip. A RocketChip instance generally consists of three major components: processors, a cache hierarchy, and an uncore.

RocketChip instantiates an in-order processor, Rocket, by default, but also supports various core implementations including an out-of-order processor, BOOM. Rocket is a 5-stage in-order processor that implements the RISC-V ISA [20, 21]. It has an MMU that supports page-based virtual memory, a non-blocking data cache, and a frontend with branch prediction. Branch prediction is configurable and provided by a branch target buffer (BTB) with its associative branch history table (BHT), and a return address stack (RAS). BOOM [7] is a superscalar out-of-order processor with a unified physical register file with configurable fetch widths, issue widths, and instruction window sizes. BOOM supports full branch speculation using a BTB, RAS, and a parameterizable backing predictor. BOOM is written in only 14K lines of Chisel code as it reuses many of RocketChip's components.

A RocketChip cache hierarchy can include L1 instruction caches, L1 blocking or non-blocking data caches, and TLBs with configurable sizes, associativities, and replacement policies. A RocketChip uncore consists of networks of cache coherent agents and the associated cache controllers for multi-core systems. These components are shared across both BOOM and rocket-based instances.

Table 1 shows the base parameters for two target processors used in the case study unless otherwise stated.

4.2 Benchmarks

The SPEC2006int benchmark suite is widely-used for computer architecture research as well as performance evaluation of real systems. However, only small fractions of the whole benchmarks are evaluated in computer architecture research using microarchitectural software simulators due to the non-trivial execution lengths as shown in Table 2. All benchmarks are compiled using Speckle [5], and run on RISC-V Linux kernel version 4.6.2. For each benchmark, we built a BusyBox image including all necessary files for a given benchmark within an initramfs. Unfortunately, `445.gobmk`, `456.hmmer`, and `462.libquantum` fail on the current version of BOOM-2w, and thus are excluded in its evaluations.

5 CASE STUDY

Figure 6 shows the IPCs of Rocket and BOOM-2w with the configurations in Table 1 for the SPEC2006int benchmark suite with its reference inputs. The IPCs are computed from the complete execution of each benchmark. Note that the parameters of BOOM-2w are chosen to approximate the configuration of the ARM Cortex-A9 processor. For reference, the IPCs of ARM Cortex A9 for the SPEC2006int benchmark suite are also presented in Figure 6.

In this case study, we are mainly interested in the performance impact of the increase in the L1 cache sizes from 16 KiB to 32 KiB. As seen from Figure 6, there are big performance improvements for several benchmarks (e.g. `400.perlbench`, `458.sjeng`, `464.h264ref`) from this change. Therefore, it is desirable to have 32 KiB L1 caches unless it lengthens the critical path.

To understand the performance evaluations more deeply, we collect more performance statistics from the performance counters. Figure 7 shows the misses per kilo instructions (MPKIs) of BOOM-2w. Notably, some benchmarks such as `471.omnetpp` and `483.xalancbmk` suffer from a large number of indirect branch mispredicts, which indicates the BTB size need to be increased.

We can also figure out the pipeline utilization by examining the issue queue utilization. Intuitively, issued slots per cycle of the issue queue are equal to IPC. As shown in Figure 8, more than 60 % of issue slots are wasted. The first case is issue slots are empty, which is caused by the frontend hazards including branch mispredicts and instruction cache misses, and/or lack of pipeline resources such as physical registers and ROB entries. For example, `403.gcc` exhibits lots of empty slots even though it has relatively small frontend MPKIs, which implies the benchmark wants more pipeline resources. Another case is issue slots are neither empty nor issued due to the backend hazards. For instance, `429.mcf` and `471.omnetpp` suffer from high data cache miss rates, therefore having a large fraction of non-empty non-issued slots.

We can check whether or not the designs are realistic by examining their power consumption. Figure 9 shows the power breakdowns for Rocket and BOOM-2w with 32 KiB L1 caches for the SPEC2006int benchmarks, which are computed using 50 random sample snapshots with 95% confidence. We can see out-of-order processors can easily be power inefficient. Specifically, the register file and the rename logic are unrealistic as they consume up to 40% of the total power. Improving energy efficiency is a key motivation of a new version of BOOM [6], which has a different microarchitecture from the current version.

Figure 10 shows the energy efficiencies of Rocket and BOOM-2w with 32 KiB L1 caches. BOOM-2w is much less energy efficient than Rocket: BOOM-2w burns more power (Figure 9) without proportionally increasing IPC (Figure 6). Notably, the energy efficiency of `429.mcf` is worse than any other benchmarks although it consumes the least power. Therefore, evaluating just performance or power but not both is insufficient if energy efficiency is a primary concern, as it is for most classes of computer systems. The methodology and the tools suggested in this paper along with the RISC-V infrastructure enables fast and accurate evaluations on energy efficiency.

6 CONCLUSION

In this paper, we presented MIDAS, an open-source, performance, power, and energy evaluation methodology using RTL designs by running realistic workloads to completion in FPGA simulation. We demonstrated cases in which microarchitectural software simulation with simulation sampling is not sufficiently fast and accurate for the hardware/software co-design flow. To address these issues, we suggest using FPGA-accelerated simulators that have been generated from the RTL design. These simulators are both fast, up to three orders-of-order-magnitude faster than cycle-level microarchitectural software simulators, and truly cycle-accurate, as they use RTL identical to that used in the silicon implementation. This enables the full execution of the full-software stack on long-running applications with little loss of fidelity. We demonstrated how this methodology can be employed in practise by running the SPEC2006int benchmark suite with its reference inputs to completion on Rocket and BOOM.

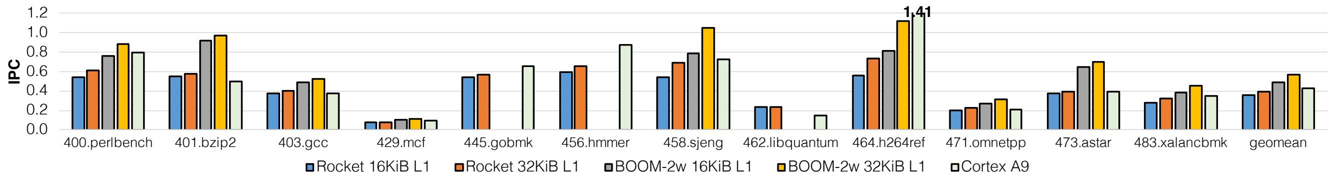


Figure 6: The IPCs of Rocket, BOOM-2w, and Cortex A9 for the SPEC2006int Benchmarks.

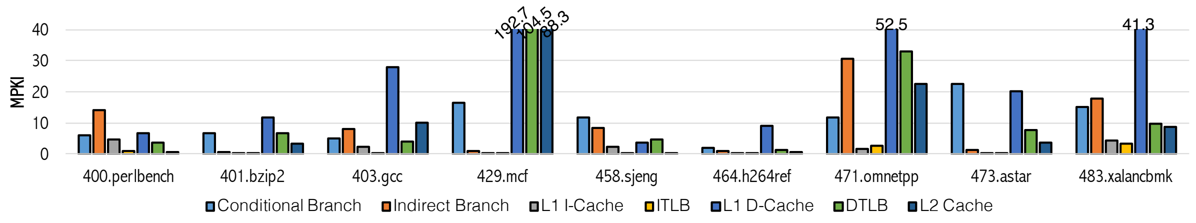


Figure 7: The MPKIs of BOOM-2w for the SPEC2006int Benchmarks.

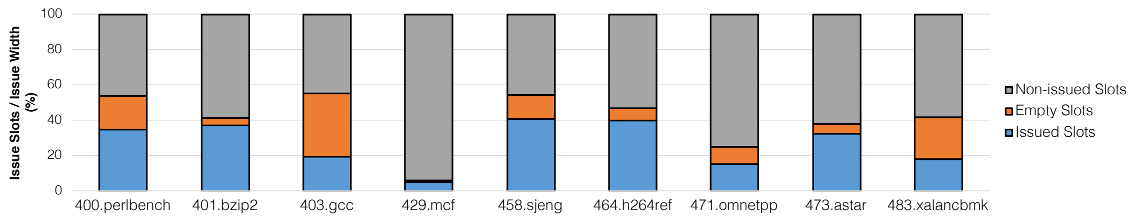


Figure 8: The Issue Queue Utilizations of BOOM-2w for the SPEC2006int Benchmarks.

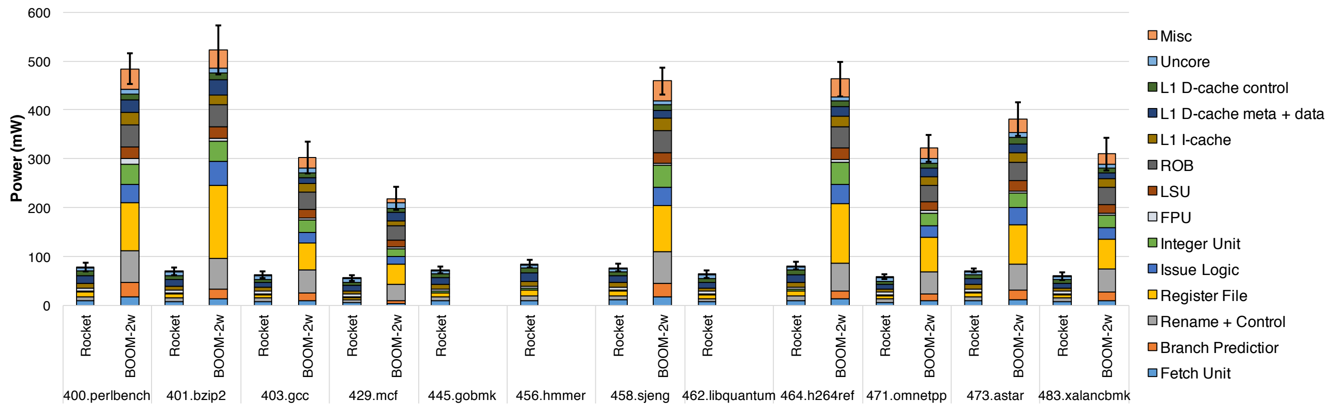


Figure 9: The Power Estimates of Rocket and BOOM-2w with 32 KiB L1 Caches for the SPEC2006int Benchmarks.

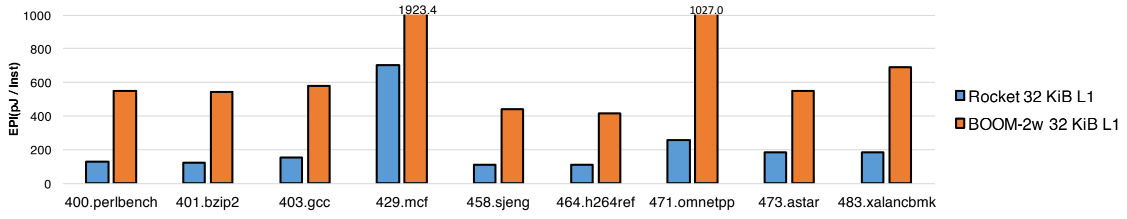


Figure 10: The Energy Efficiencies of Rocket and BOOM-2w with 32 KiB L1 Caches for the SPEC2006int Benchmarks.

ACKNOWLEDGEMENT

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, HP, Huawei, LGE, Nokia, NVIDIA, Oracle, and Samsung. 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] Krste Asanović et al. 2015. *The Rocket Chip Generator.* Technical Report UCB/EECS-2016-17.

[2] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. Chisel: constructing hardware in a scala embedded language. In *DAC.*

[3] Nathan Binkert et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39 (Aug 2011).

[4] Christopher Celio. 2010. The ccbench micro-benchmark collection. (2010). https://github.com/ucb-bar/ccbench/wiki

[5] Christopher Celio. 2014. Speckle: A wrapper for the SPEC CPU2006 benchmark suite. (2014). https://github.com/ccelio/Speckle.git

[6] Christopher Celio, Pi-Feng Chiu, Borivoje Nikolić, David A. Patterson, and Krste Asanović. 2017. BOOMv2: an open-source out-of-order RISC-V core. In *First Workshop on Computer Architecture Research with RISC-V (CARRV).*

[7] Christopher Celio, David A. Patterson, and Krste Asanović. 2015. *The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor.* Technical Report UCB/EECS-2015-167.

[8] Palmer Dabbelt. 2017. *PLSI: A Portable VLSI Flow.* Master's thesis. University of California, Berkeley.

[9] Anthony Gutierrez, Joseph Pusdesris, Ronald G. Dreslinski, Trevor Mudge, Chander Sudanthi, Christopher D. Emmons, Mitchell Hayenga, and Nigel Paver. 2014. Sources of error in full-system simulation. In *ISPASS.*

[10] Donggyu Kim, Adam Izraelevitz, Christopher Celio, Hokeun Kim, Brian Zimmer, Yunsup Lee, Jonathan Bachrach, and Krste Asanović. 2016. Strober : Fast and Accurate Sample-Based Energy Simulation for Arbitrary RTL. In *ISCA.*

[11] Y. Lee et al. 2016. An Agile Approach to Building RISC-V Microprocessors. *IEEE Micro* 36, 2 (Mar 2016), 8–20.

[12] Patrick S Li, Adam M Izraelevitz, and Jonathan Bachrach. 2016. *Specification for the FIRRTL Language.* Technical Report UCB/EECS-2016-9.

[13] Derek Lockhart, Gary Zibrat, and Christopher Batten. 2014. PyMTL : A Unified Framework for Vertically Integrated Computer Architecture Research. In *MICRO.*

[14] R. Nikhil. 2004. Bluespec System Verilog: efficient, correct RTL from high level specifications. In *MEMOCODE.*

[15] Avadh Patel, Furat Afram, and Shunfei Chen. 2011. MARSSx86: A full system simulator for x86 CPUs. In *DAC.*

[16] Ofer Shacham, Sameh Galal, Sabarish Sankaranarayanan, Megan Wachs, John Brunhaver, Artem Vassiliev, Mark Horowitz, Andrew Danowitz, Wajahat Qadeer, and Stephen Richardson. 2012. Avoiding game over: Bringing design to the next level. In *DAC.*

[17] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. 2002. Automatically characterizing large scale program behavior. In *ASPLOS.*

[18] Zhangxi Tan, Andrew Waterman, Henry Cook, Sarah Bird, Krste Asanović, David Patterson, and David Patterson Zhangxi Tan, Andrew Waterman, Henry Cook, Sarah Bird, Krste Asanović. 2010. A Case for FAME: FPGA Architecture Model Execution. In *ISCA.*

[19] J.I. Villar, J. Juan, M.J. Bellido, J. Viejo, D. Guerrero, and J. Decaluwe. 2011. Python as a hardware description language: A case study. In *Southern Conference on Programmable Logic (SPL).*

[20] Andrew Waterman, Yunsup Lee, Krste Asanović, and David Patterson. 2016. *The RISC-V Instruction Set Manual: Privileged Architecture Version 1.9.1.* Technical Report UCB/EECS-2016-161.

[21] Andrew Waterman, Yunsup Lee, David Patterson, and Krste Asanovi. 2016. *The RISC-V Instruction Set Manual: User-level ISA Version 2.1.* Technical Report UCB/EECS-2016-118.

[22] Thomas F T.F. Wenisch, R.E. Roland E R.E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and James C J.C. Hoe. 2006. SimFlex: Statistical Sampling of Computer System Simulation. *IEEE Micro* 26, 4 (Jul 2006), 18–31.

[23] R.E. Wunderlich, T.F. Wenisch, B. Falsafi, and J.C. Hoe. 2003. SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling. In *ISCA.*