RISC5: Implementing the RISC-V ISA in gem5

Alec Roelke
University of Virginia
ar4jc@virginia.edu

Mircea R. Stan
University of Virginia
mircea@virginia.edu

Abstract

We present an RISC5, an implementation of the RISC-V ISA in the gem5 simulator. Gem5 is a modular, open-source simulation platform that supports several ISAs such as x86 and ARM and includes system-level architecture and processor microarchitecture models. It also has advanced simulation features such as system call emulation and checkpointing that the Chisel C++ simulator lacks, increasing its usefulness for simulating entire RISC-V applications or using phase analysis to estimate system behavior. Gem5 also provides detailed performance data that can be used in power estimation tools such as McPAT, which require fine granularity to provide accurate output. RISC5 is validated against performance data from the Chisel C++ emulator and FPGA soft core and is shown to have less than 10% error on several performance statistics.

CCS Concepts • **General and reference** → **Performance**; • **Computer systems organization** → **Reduced instruction set computing**; • **Hardware** → Chip-level power issues;

Keywords RISC-V, gem5, soc simulation, simulation infrastructure, tool flow

ACM Reference format:
Alec Roelke and Mircea R. Stan. 2017. RISC5: Implementing the RISC-V ISA in gem5. In *Proceedings of Computer Architecture Research with RISC-V, Boston, Massachusetts USA, October 14, 2017 (CARRV'17),* 7 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

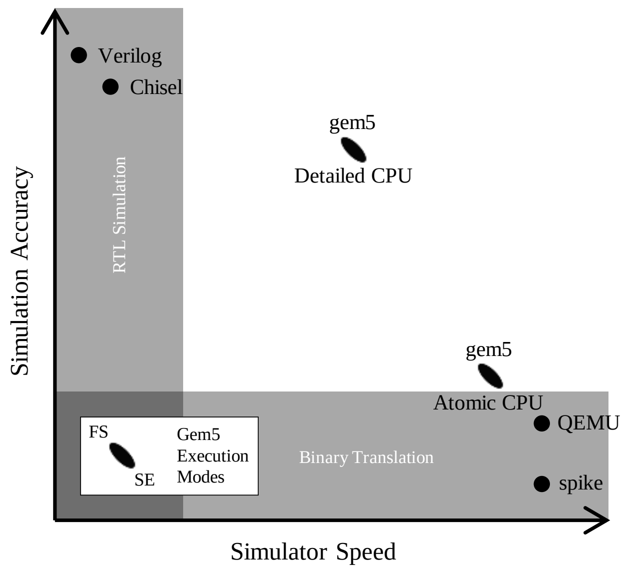

Figure 1. Illustration of simulation accuracy vs. speed. Without gem5, most simulators fall into two categories: high-accuracy RTL simulation or high-speed binary translation. With its high-level models of execution units and customization, gem5 can fall in between the two categories.

1 Introduction

As the number of transistors on a chip increases with Moore's Law, designing and simulating the chip becomes more complex and time-consuming. Additionally, proprietary libraries and architectures can impede collaboration between academic and industry researchers and engineers. In order to address this issue, the gem5 simulator [6] was developed to model a wide range of architectures in a way that is accessible to all researchers. Its modular design allows a researcher to focus on some aspects of a design without having to understand gem5's implementations of the rest and its open-source license allows easy collaboration.

Even so, many of the instruction sets gem5 supports are proprietary. Licenses for these ISAs can be costly and time-consuming to acquire and difficult to work with due to their complexity [3]. The RISC-V ISA [17] is designed to solve these problems by being simple, extensible, and free to use. Other open ISAs exist, but mistakes in their design have caused them to lose popularity [3]. RISC-V is designed to learn from past mistakes and maintain relevance by being extensible, agnostic of the type of core it runs on (such as in-order, out-of-order, or VLIW), and usable on real hardware running real workloads [17]. This way it can be extended to include future improvements in computer architecture and increases in data size.

CARRV'17, October 14, 2017, Boston, Massachusetts USA
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Several RISC-V designs [2, 7] are implemented using the hardware construction language Chisel [4], which takes advantage of programming concepts such as polymorphism and inheritance to create hardware generators that facilitate the creation of functional units. It can generate C++ code[1] that models a design and can be used in a simulator for cycle-accurate simulation along with Verilog HDL code that can be mapped to FPGA or used in an ASIC flow.

Existing RISC-V simulation tends to fit into two categories: detailed RTL simulation as discussed above or binary translation using emulators like spike or QEMU [5] (Figure 1). Using high-level models of architectural units and memory hierarchy, gem5 is uniquely capable of bridging the gap between these two categories by providing highly accurate simulation at much faster speeds than RTL simulation does. In Section 2, we introduce an RISC5, an implementation of RISC-V in gem5, and discuss its simulation features in Section 3 that allow it to bridge this gap. We validate it in Section 4 against Rocket [2] and compare its performance against the Chisel-generated C++ simulator and an FPGA soft core. Finally, we present future work in Section 5 and conclude in Section 6.

2 Implementation of RISC-V in gem5

RISC-V is divided into a base integer instruction set, which supports 32- and 64-bit address and data widths,[2] and several extensions that add additional instructions. These extensions include the multiply extension, which adds integer multiply and divide instructions; the atomic extension, which adds instructions to atomically read-modify-write data in memory; and single- and double-precision floating point extensions. Currently RISC5 implements these instruction sets for single-core simulations in system call emulation (SE) mode. Additional nonstandard extensions are available in [3], such as quad-precision floating point arithmetic. Of these, only the compressed instruction set is currently included in RISC5. In this section we present some details about the implementations of these instruction sets.

2.1 Integer and Multiply Instructions

Because MIPS is a RISC instruction set with many analogues to RISC-V [8], much of the code gem5 uses to implement MIPS was adapted to implement RISC-V. All instructions in RV64IM could be implemented either by referring to their definitions in [3] or by referring to their analogues in other ISAs from [8] (usually MIPS) when their implementations required information about gem5's internal behavior. For example, the RISC-V `fence` instruction is similar to the MIPS `sync` instruction, so the implementation of `fence` in gem5 is based off its implementation of `sync`. The only instruction not implemented is `eret`, which returns from a higher privilege level. Privilege levels do not exist in gem5's SE mode, so this instruction is unnecessary until full-system simulation is implemented.

2.2 Atomic Instructions

RISC-V follows the *release consistency* memory model [10], which ensures that the results of a writer's operations on a memory location are seen by a reader if the reader acquires that location after the writer releases it. To that end, RISC-V provides load-reserved and store-conditional instructions along with atomic read-modify-write (RMW) instructions that can be marked to acquire and/or release a memory location.

Each RMW instruction requires two memory accesses (read and write). Since gem5 does not support multiple memory accesses per instruction when simulating memory with timing, each atomic memory instruction had to be split into two micro-ops: one which would read from memory and one which would write the result back to memory. In order to enable the write micro-op of each atomic memory instruction to keep track of the data loaded from memory by the read micro-op, a new integer register had to be added to RISC-V. This register is only used for storing a value loaded by the first micro-op of an atomic memory instruction. Each micro-op is marked with an `ACQUIRE` or `RELEASE` flag to indicate if it is acquiring and/or releasing a memory location. Load-reserved and store-conditional instructions are similarly flagged. Because RISC5 only supports single-threaded simulation, the actual atomicity and memory consistency of the implementations of these instructions cannot be verified.

2.3 Floating Point Instructions

RISC-V conforms to the IEEE-754 2008 floating-point standard [1], which is slightly different than the standard that the x86 machine used for development conforms to. The IEEE 2008 floating point standard provides five modes for rounding results (*roundTowardPositive*, *roundTowardNegative*, *roundTowardZero*, *roundTiesToEven*, and *roundTiesToAway*) while the machine used for development has four (*roundTiesToAway* is missing). Attempting to use the missing rounding mode causes the simulation to halt.

We verified RISC5's floating point behavior by comparing its results with those from the RISC-V ISA simulator, spike, and from the Chisel-generated C++ simulator. While Spike emulates RISC-V instructions on a host system, the Chisel-generated C++ simulator makes use of a gate-level model of a RISC-V CPU and so performs its own floating point arithmetic. While the results of the floating-point computation were the same between both simulators, they disagreed in several cases about what exceptions should be generated and sometimes did not conform to the specifications in [17]

[1]This functionality was present in chisel2 and was replaced in chisel3 by translation of Verilog into C++ source.
[2]A provisional definition for 128-bit addresses and data exists in [17] but formal definitions for instructions have not been created.

Table 1. Simulation Features and Compatibility

Feature	gem5	Chisel	spike	QEMU
Binary translation	✓		✓	✓
Checkpoints	✓			✓
Multicore simulation	✓[3]	✓	✓	✓
Performance statistics	✓	✓[4]		
RTL imulation		✓		
System call emulation	✓	✓[5]	✓[5]	✓
ASIC synthesis		✓		
FPGA tools		✓		
Phase analysis	✓			
Stats-based tools	✓	✓[4]		

and [1]. For example, Spike threw a divide-by-zero exception when performing a single-precision divide by zero, but the Chisel model did not. In these cases, gem5 conforms to the specifications without regard to other simulators. In the above example, a divide-by-zero exception will be thrown.

3 Comparison with Chisel

Prior to RISC-V's inclusion into gem5, existing simulation tools for RISC-V supported either slow but highly accurate RTL simulation or fast but low-detail binary translation (Figure 1). The former enables debugging and analysis of a design, but is hampered by long simulation times; the latter enables software development due to its high speed, but cannot provide information about an underlying system to inform design. In this section, we discuss gem5's ability to bridge the gap between the two simulation paradigms high-level architectural and memory models and advanced simulation features that enable it to achieve high accuracy while reducing simulation time. In doing so, we compare its features with those of Chisel and other methods of simulating RISC-V (Table 1), evaluate the compatibility of each system with external tools, and perform an example simulation flow.

3.1 Simulation Features

Gem5 provides several features to reduce simulation setup complexity and simulation time. Its system call emulation (SE) mode replaces a program's own system calls with ones to the host. As a result, the target system can make use of the host's file system; a kernel and boot device for the target system are not required. This reduces the overhead of booting a kernel and performing system calls, which is useful for measuring workload effects. Gem5 also provides a full-system mode if kernel simulation is desired.

It also contains several CPU models and supports saving simulation state. Four CPU models exist at varying levels of detail and allow exploitation of the tradeoff between simulation accuracy and speed: CPU models can be switched during simulation, allowing for faster simulation until a period of interest is reached followed by slower, but more detailed, simulation of that period. Similarly, at any point during simulation a checkpoint can be saved. Once the simulation is complete, gem5 can resume from any saved checkpoint without having to simulate up to that point first and can use a more detailed CPU model than the checkpoint was taken with. These two features combined allow a user to rapidly simulate a benchmark using a simple CPU model and save checkpoints at the start of each region of interest, then resume from each checkpoint and simulate using a detailed model until the region of interest is complete. This enables compatibility with techniques such as phase analysis [15] so the power and performance of a benchmark can be estimated without having to simulate it in its entirety.

Because Chisel only generates a C++ model of a design, it does not have advanced features such as those listed above. The simulation environment is left up to the user, who may decide which features are necessary but must implement them him- or herself. The user may choose to make use of an existing simulation environment such as the one in [2], which does support system call emulation through the use of the RISC-V proxy kernel, but does not support some of gem5's other features. Unlike gem5, however, the Chisel model can output a trace of the signals in a design for inspection and analysis with a waveform viewer, enabling RTL debugging.

3.2 Compatibility with External Tools

Gem5's output of performance statistics enables it to be used alongside tools that model other aspects of a design, such as power [14], temperature [12], and voltage noise [19]. These tools often require time traces of their inputs, and gem5 can provide those by outputting performance statistics mid-simulation. By default, Chisel's C++ model does not keep track of performance information. A design in Chisel can be modified to keep track of that information, but in order to make use of it the program being executed must contain code to read and output it at the desired points.

Even so, Chisel is able to generate Verilog HDL that can be used in ASIC synthesis or mapped to FPGA [4]. This way a designer can see not only the impact of a new idea on performance but also on power consumption, area, and reliability using existing synthesis tools. Similar simulations using gem5 and other tools can provide estimates, but cannot provide exact information about a particular design.

[3]Multicore simulation is generally supported by gem5 in SE and FS mode, but has not been implemented for RISC-V.
[4]Chisel can only output performance statistics if a design has means of counting and outputting them.
[5]System call emulation is supported via the RISC-V proxy kernel.

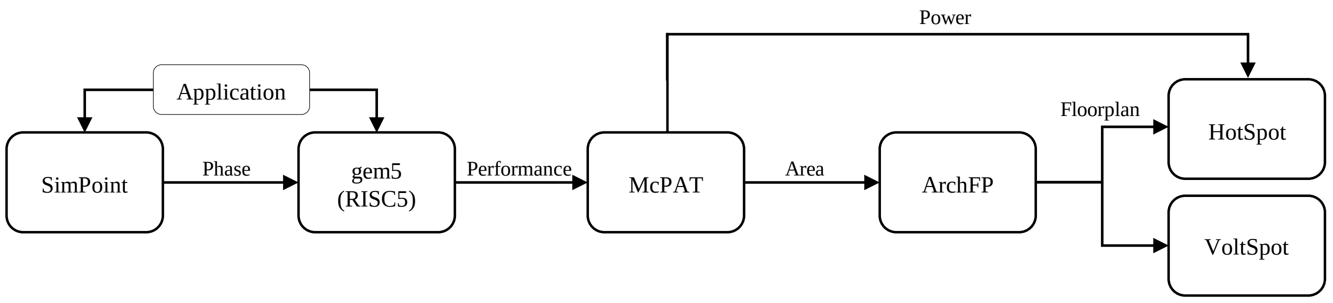

Figure 2. Tool flow for simulating an application running on a chip. In our example, we use SimPoint [15] to find a region of interest of the *libquantum* benchmark [11], simulate it using gem5, estimate its area and calculate its power using McPAT [14], create a floorplan using ArchFP [9], and then calculate temperature using HotSpot [12]. VoltSpot [19] can also be used to calcluate voltage droop.

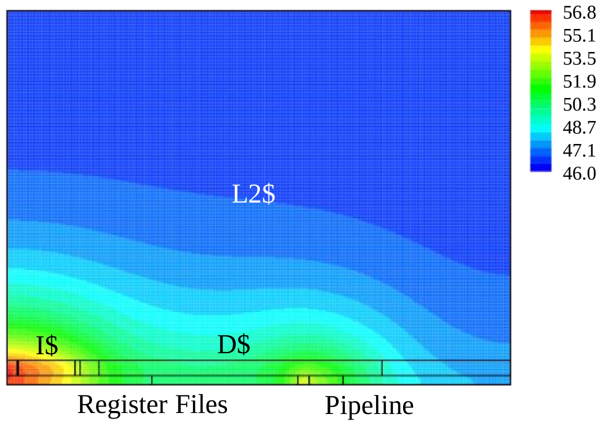

Figure 3. Floorplan for Rocket [2] overlaid with a heat map for a section of *libquantum* [11]. Blue indicates temperatures closer to 45.99 °C and red indicates closer to 56.65 °C. The total estimated area of the chip is 4.175 mm^2.

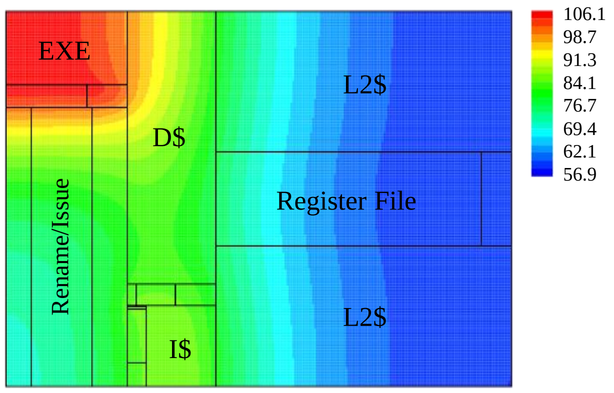

Figure 4. Floorplan for BOOM [7] overlaid with a heat map for a section of *libquantum* [11]. Blue areas indicate temperatures closer to 57.19 °C and red indicates closer to 106.1 °C. The total estimated area of the chip is 1.367 mm^2.

Table 2. Chip Parameters

	Rocket [2]	**BOOM [7]**
Process Size (nm)	45	45
Frequency (MHz)	1500	1500
External Memory Size (MB)	4096	4096
Inst. Cache Size (kB)	16	32
Inst. Cache Associativity	4	8
Data Cache Size (kB)	16	32
Data Cache Associativity	4	8
L2 Cache Size (kB)	2048	512
L2 Cache Associativity	8	8

Table 3. Simulation Flow Results Summary

	Rocket [2]	**BOOM [7]**
Area (mm^2)	4.175	1.367
Area w/o L2 (mm^2)	0.2946	0.7683
Power (W)	1.0215	5.1114

3.3 Example Tool Flow

To illustrate gem5's ability to simulate power and performance metrics alongside other tools, an example flow is presented in Figure 2. The flow begins by using gem5 in tandem with SimPoint [15] to determine representative phases of the application of interest. Once those phases have been determined, gem5 can simulate them in a detailed fashion and create performance data for power estimation with McPAT [14]. With the performance data and with chip information determined by a designer, McPAT can estimate area and power consumption. Another option for computing power is Strober [13], which can characterize arbitrary RTL for power. Area is input into ArchFP [9], which produces a floorplan that, alongside McPAT's power calculation, is input into HotSpot [12] to calculate chip temperature. Although we do not include it in our example, VoltSpot [19] can also use McPAT's power data and ArchFP's floorplan to estimate voltage droop.

We execute this flow using two RISC-V designs: Rocket and BOOM [7]. Using Rocket to guide parameters for gem5 and using SimPoint to find the most representative region, we simulated one million instructions of the *libquantum* benchmark to get performance data for input into McPAT. Using McPAT's area estimation, a floorplan was created with ArchFP that was used alongside McPAT's power estimation to create a temperature map of the system using HotSpot (Figure 3). This process was repeated using the same benchmark and region for BOOM, whose results are shown in Figure 4. The values of parameters from each design can be found in Table 2 and summaries of the calibrated area estimates and power calculations can be found in Table 3.

McPAT has known inaccuracies [18] which affect its power and area calculations. These inaccuracies can be reduced using fixes from [18] and by calibrating results against real data. We applied the fixes and calibrated McPAT's area estimates using the BOOM's floorplan from [7], excluding the area labeled "uncore," which resulted in an area of 1.367 mm^2 (Figure 4). To create the floorplan for Rocket, we scaled McPAT's area estimation using BOOM's calibrations, which resulted in an area of 4.175 mm^2 (Figure 3). Surprisingly, Rocket is much larger than BOOM, which has a larger register file and additional units for out-of-order execution. However, Rocket's L2 cache has about four times the capacity of BOOM's L2 cache. Indeed, when comparing the areas of the two chips without L2 cache, BOOM is over twice as large.

Even though it is smaller, BOOM consumes more power than Rocket. Because BOOM is out-of-order, its expanded register file and additional units add power and area overhead. According to the temperature maps produced by HotSpot, assuming that higher temperature is caused by higher power consumption from higher activity, Rocket has overall lower activity than BOOM. BOOM has multiple pipelines that enable higher core utilization through simultaneous in-flight instructions, increasing power consumption and temperature. Its higher capacity and associativity also consume more energy per instruction than a similar access pattern would in Rocket, leading to high data cache temperature from the frequent memory accesses in the benchmark region. Overall, BOOM is significantly hotter than Rocket; not only is its maximum temperature almost twice as high as Rocket's, but its minimum temperature is higher than Rocket's maximum.

Table 4. Version Information

Component	Version
gem5	`ffc29f2d9a5a`[6]
RISC-V User-level ISA [17]	2.1
RISC-V Privileged ISA [16]	N/A
Rocket Chip	`73e9508`
Chisel	`b18e98b`

4 Validation and Performance

We simulated selected benchmarks from the SPEC CPU2006 suite [11] on gem5, the Chisel-generated C++ simulator, and on a RISC-V soft core on an FPGA. Version information on the tools used is contained in Table 4. In order to keep track of performance statistics in Figure 5, we modified the Chisel code to add additional performance counters and each benchmark's source to read and print their values. The Chisel code was further modified to pause these counters during system calls to improve the quality of comparison with gem5 and account for the fact that gem5's SE mode does not count statistics during system calls since it outsources them to the host. We ran gem5 and the C++ simulator on a four-core machine running at 3.7 GHz with 32 kB, 256 kB, and 10 MB L1, L2, and L3 caches, respectively, and 32 GB of main memory. Rocket includes a flow for configuring a Xilinx Zynq FPGA as a RISC-V core with a clock rate of 1 GHz and up to half of the board's 512 MB of DRAM for the core. We used this flow to create a RISC-V soft core on the Zynq on which we also ran the benchmarks. Figure 5 shows each performance statistic obtained from gem5's output normalized to the value extracted from the C++ simulator and FPGA.

As Figure 5 shows, gem5 is accurate in the number of instructions retired, number of memory operations performed, and number of branch instructions executed. Less accurate is the number of cycles it took to complete each benchmark and number of instructions fetched. This is due to differences between the microarchitectures and memories modeled by gem5 and implemented by Chisel. In particular, the branch prediction may be different, causing a significant difference in the number of instructions fetched and cycles while not affecting the number of retired instructions or memory operations. This also accounts for the slight difference in branch operations.

The FPGA, on average, took about 26.5 times less time to execute benchmarks than gem5 did while the Chisel simulator took, on average, about 32 times longer to perform each benchmark. As a result, several benchmarks took too long to execute and get data from, so they are excluded from Figure 5 for C++. This slowdown is due to the very high level of detail of the Chisel simulator, which enables the capability

[6]This changeset hash refers to gem5's older Mercurial repository. Gem5 has since been moved to GitHub.

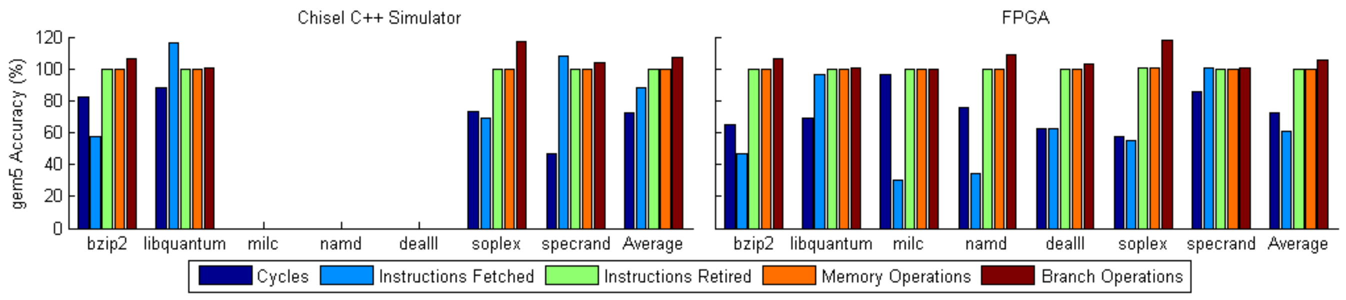

Figure 5. Validation of gem5 performance statistics against the same values from Chisel simulation (left) and FPGA (right). Some benchmarks are omitted due to excessively long simulation times. Gem5 is generally more accurate for statistics that accumulate later in Rocket's pipeline than for earlier.

for very accurate simulation but adds significant overhead. Gem5's abstraction of low-level signals using CPU models reduces overhead and allows it to run faster.

Since all benchmarks except *specrand* and *dealII* performed few system calls, the fact that gem5's SE mode does not simulate them on the target system while Chisel's C++ simulator does causes no significant slowdown. Of the two with many system calls, only *specrand* completed in the C++ simulator and had a much higher runtime compared to gem5 than the others (0.0041 slowdown compared to 0.0508 for *bzip2*, 0.0314 for *soplex*, and 0.0377 for *libquantum*) due to the significant overhead of having to simulate the system calls. This is not the case for the FPGA soft core because it ran both user-level code and system call code natively.

5 Future Work

There is still some functionality within gem5 that does not yet support RISC-V. The two most important contributions will be support for multithreaded execution in SE mode and for full-system simulation. Additionally, there are some smaller improvements to existing support, such as implementation of *roundTiesToAway* for floating-point operations, that will improve accuracy of simulation. Finally, it will be useful to characterize the differences between gem5's models and existing Chisel implementations and modify gem5 to more closely approximate real designs.

Gem5's ability to customize modeled hardware introduces a unique opportunity for ISA comparison. Because the system configuration is completely separated from the ISA, it is possible to simulate binaries compiled from the same code for different ISAs on systems that are identical except for the ISA they run. This allows direct comparison between ISAs that is difficult with tools that can only perform RTL simulation or binary translation.

6 Conclusion

We presented RISC5, an implementation of RISC-V in the gem5 simulator, that includes standard integer instructions, integer multiply instructions, atomic memory operations, and floating point instructions for use with gem5's SE mode simulating a single core. In comparison with Chisel, gem5 has a more robust simulation environment that supports extensive configuration with fast or detailed simulation, storing and resuming from checkpoints, and automatic tracking of performance statistics. On the other hand, Chisel enables fine control over a design and integrates better into a flow while still enabling detailed simulation through a C++ model or FPGA emulation. We then showed an example simulation flow of tools for modeling different aspects of a design, including ArchFP for floorplanning, gem5 for performance, McPAT for power and area estimation, and HotSpot for temperature calculation for two implementations of RISC-V: Rocket and BOOM. Finally, we showed gem5's accuracy and compared its runtimes for several benchmarks with those of a Chisel-generated FPGA mapping and C++ simulator. RISC5 is available as part of the main gem5 release at http://www.gem5.org.

Acknowledgments

We would like to acknowledge the members of the gem5 community, Schuyler Eldridge, Pradip Bose, Alper Buyuktosunoglu, and Matthew Ziegler from IBM, and Xinfei Guo, Vaibhav Verma, and Sergiu Mosanu from the University of Virginia for providing invaluable input into the implementation of RISC-V on gem5.

This work has been supported by NSF and SRC under the CLASH project and by CFAR, a STARNET center, by the National Science Foundation Graduate Research Fellowship under Grant No. DDGE-1315231, and by Defense Advanced Research Projects Agency, Microsystems Technology Office (MTO), under contract no. HR0011-13-C-0022. The views expressed are those of the authors and do not reflect the social policy or position of the Department of Defense or the U.S. Government.

References

[1] 2008. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008* (Aug 2008), 1–70. https://doi.org/10.1109/IEEESTD.2008.4610935

[2] Krste Asanović et al. 2016. *The Rocket Chip Generator.* Technical Report UCB/EECS-2016-17. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[3] Krste Asanović and David A. Patterson. 2014. *Instruction Sets Should Be Free: The Case For RISC-V.* Technical Report UCB/EECS-2014-146. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.html

[4] Jonathan Bachrach et al. 2012. Chisel: constructing hardware in a scala embedded language. In *Proceedings of the 49th Annual Design Automation Conference.* ACM, 1216–1225.

[5] Fabrice Bellard. 2005. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC '05).* USENIX Association, Berkeley, CA, USA, 41–41. http://dl.acm.org/citation.cfm?id=1247360.1247401

[6] Nathan Binkert et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

[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. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.html

[8] Tony Chen and David A. Patterson. 2016. *RISC-V Geneology.* Technical Report UCB/EECS-2016-6. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-6.html

[9] Gregory G Faust et al. 2012. ArchFP: Rapid prototyping of pre-RTL floorplans. In *VLSI and System-on-Chip (VLSI-SoC), 2012 IEEE/IFIP 20th International Conference on.* IEEE, 183–188.

[10] Kourosh Gharachorloo et al. 1990. Memory Consistency and Event Ordering in Scalable Shared-memory Multiprocessors. In *Proceedings of the 17th Annual International Symposium on Computer Architecture (ISCA '90).* ACM, New York, NY, USA, 15–26. https://doi.org/10.1145/325164.325102

[11] John L Henning. 2006. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News* 34, 4 (2006), 1–17.

[12] Wei Huang et al. 2006. HotSpot: A compact thermal modeling methodology for early-stage VLSI design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14, 5 (2006), 501–513.

[13] 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. *SIGARCH Comput. Archit. News* 44, 3 (June 2016), 128–139. https://doi.org/10.1145/3007787.3001151

[14] Sheng Li et al. 2009. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on.* IEEE, 469–480.

[15] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. 2002. Automatically characterizing large scale program behavior. In *ACM SIGARCH Computer Architecture News*, Vol. 30. ACM, 45–57.

[16] Andrew Waterman et al. 2016. *The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.9.1.* Technical Report UCB/EECS-2016-161. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-161.html

[17] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanović. 2016. *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. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.html

[18] Sam Likun Xi et al. 2015. Quantifying sources of error in McPAT and potential impacts on architectural studies. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on.* IEEE, 577–589.

[19] Runjie Zhang et al. 2014. Architecture implications of pads as a scarce resource. In *ACM SIGARCH Computer Architecture News*, Vol. 42. IEEE Press, 373–384.