Using FireSim to Enable Agile End-to-End RISC-V Computer Architecture Research

Sagar Karandikar, David Biancolin, Alon Amid, Nathan Pemberton, Albert Ou, Randy Katz, Borivoje Nikolic, Jonathan Bachrach, Krste Asanovic
{sagark,biancolin,alonamid,nathanp,aou,randykatz,bora,jrb,krste}@eecs.berkeley.edu
University of California, Berkeley

ABSTRACT

The explosive growth in the RISC-V ecosystem has brought about a multitude of open RTL SoC implementations, as well as broad software compatibility, presenting the opportunity to perform computer architecture research with direct impact using real implementations. However, putting these together in a research context with small, agile teams of developers has been challenging due to difficulty maintaining hardware compatibility with complicated software stacks, slow software RTL simulators, and poor introspection, productivity, and modeling-accuracy with FPGA prototyping.

While our prior work described FireSim's capabilities as an FPGA-accelerated cycle-exact datacenter simulation platform, in this paper, we delve into the internals of FireSim and walk through a case study simulating a novel hardware accelerator (Hwacha) that shows how a researcher would use FireSim as a tool for rapidly and cycle-exactly modeling their own systems that build on a single-node RISC-V SoC. We discuss how FireSim addresses the challenges of building a reliable, reproducible, and productive RISC-V research environment, including packaging standardized releases of compatible RISC-V software and hardware, automating the process of running cycle-exact simulations on cloud FPGAs that are orders of magnitude faster than any software simulator, and providing debugging tools that allow introspection capabilities not available in FPGA prototypes.

CCS CONCEPTS

• **Computing methodologies** → **Modeling and simulation**; **Modeling and simulation**; • **Computer systems organization**;

KEYWORDS

FPGA Simulation, Agile Hardware, Open-source Hardware, Performance Evaluation

ACM Reference Format:
Sagar Karandikar, David Biancolin, Alon Amid, Nathan Pemberton, Albert Ou, Randy Katz, Borivoje Nikolic, Jonathan Bachrach, Krste Asanovic. 2019. Using FireSim to Enable Agile End-to-End RISC-V Computer Architecture Research. In *CARRV '19: Third Workshop on Computer Architecture Research with RISC-V, June 22, 2019, Phoenix, AZ*. ACM, New York, NY, USA, 6 pages.

CARRV '19, June 22, 2019, Phoenix, AZ

1 INTRODUCTION

Driven by RISC-V, great strides have been made recently in the open-hardware community, with many new processor and accelerator designs, including some silicon and industry proven designs. However, translating open-hardware into tools for computer architecture research can be challenging. Integrating open-hardware and software components with a small agile team of researchers can be time consuming and difficult to maintain, in addition to being difficult to simulate. In particular, when using open-hardware platforms, a *simulation gap* exists—researchers have traditionally been left with slow software simulation or fast but inaccurate and complex FPGA prototyping. While commercial emulation tools exist, they are far too expensive to use in academia.

While previous work has covered the FireSim FPGA-accelerated hardware simulation platform in the context of thousand-node networked datacenter simulations [9], in this work we walk through using the FireSim flow to enable practical computer architecture research with real open-implementations running full software stacks at a variety of scales. To demonstrate what is possible with FireSim, we show how a researcher developing a hardware accelerator (e.g. Hwacha [11]) can use FireSim to simplify their research from early block-level development to running large benchmark suites on parallel FPGA simulations of the full SoC.

1.1 Why Not FPGA Prototyping?

In many cases, mapping an RTL design to an FPGA is not the end goal—instead, FPGAs are commonly used as a prototyping platform to enable running large software workloads on hardware designs before they are taped-out. However, FPGA-prototyping comes with several challenges. Often the prototype RTL must be substantially modified from that which will ultimately be taped-out. Directly attaching the I/Os of a design to FPGA I/Os, as is common with FPGA prototypes, also results in inaccurate I/O timing. Lastly, constructing an FPGA prototype also requires significant manual effort and the resulting prototype usually has poor introspection capabilities once deployed on the FPGA.

Modeling I/O in an FPGA prototype is difficult to do accurately and deterministically. Consider an SoC that will eventually be taped out at 1 GHz, that includes a DRAM memory system with an average memory access time of 100ns. If this system is directly prototyped on an FPGA, the design will run at a lower frequency, e.g. 100 *MHz*. However, the DRAM of the SoC must be modeled using the physical DRAM attached to the FPGA, which presents the aforementioned "real world" latency of 100ns. If we measure performance of software running on the SoC design in these two cases, memory accesses will be *10 times* faster on the FPGA prototype than on the taped out system. In the least harmful case, this simply results in incorrect

system performance measurements. In the worst case, this can mask timing bugs in the FPGA-prototype version of the SoC that instead are caught in silicon—leading to a costly respin. Similar arguments can be made for other peripherals, like disks, networks, and UARTs.

Without introducing significant complexity, FPGA prototypes also limit designs to interfacing with the I/O interfaces available on a particular FPGA—if a user wants to model an SoC with an Ethernet NIC on an FPGA without Ethernet support, then they must manually develop some form of shim or purchase a new FPGA. Even when the correct I/Os are available, significant manual effort is required to connect the user's design with the FPGA manufacturer's IP that actually connects to the needed I/O.

FPGA-prototypes also generally suffer from poor introspection into a running design. Introspecting on an FPGA design is generally achieved using in-fabric debuggers like Vivado's Integrated Logic Analyzers, or ILAs, which can provide waveforms for a small selection of signals for a small number of cycles (e.g. 1000s of cycles). In addition to consuming non-trivial FPGA resources, since these debuggers provide poor visibility, choosing the right signals to tap is difficult: the designer is often forced to reselect and recompile the FPGA bitstream multiple times as they gradually identify the root cause.

FPGA-accelerated simulators address these issues, by exploiting *host-target decoupling*: they decouple clock ticks visible to the simulated system from the host clock of the FPGA. In doing so, they allow custom models to be attached to the design, which present accurate timing on all I/O interfaces. When new I/O peripherals are designed, the user does not need to purchase a new FPGA—they can simply write a new I/O model, which speaks to a standard simulation transport and is automatically shimmed out to simulation management software on the host machine attached to the FPGA, for further modeling. Because these models speak to standard interfaces exposed by the simulator environment (e.g. MMIO or a high-throughput queue interface in FireSim's case) they are easy to develop and test in software simulation, avoiding the commonly difficult to debug interfacing issues that show up with FPGA prototypes. Having this clean abstraction layer also simplifies porting to new FPGA platforms—the simulator environment itself is ported, rather than painstakingly porting each user design. This porting effort can be shared between all users of the FPGA platform. FPGA-simulation environments also add significant introspection capabilities that make debugging and evaluating a design considerably easier. We will discuss these in detail in Section 4.5.

These differences demonstrate that most architects don't really want FPGA prototypes—rather, FPGA-prototypes are a stopgap for the lack of a productive and open FPGA-accelerated cycle-exact simulation platform for their RTL designs.

1.2 Why use cloud FPGAs?

Cloud FPGAs provide significant benefits to the architecture research community as a platform for enabling agile design evaluation. Just like the general-purpose cloud, the FPGA cloud provides benefits in terms of elasticity, capital expenditure reduction, and ease-of-use [4].

The use of cloud FPGAs allows FPGA-simulation environments to be *elastic*. When a user needs to run several benchmarks in parallel in a short time period, e.g. before a paper or tape-out deadline, they can quickly scale to hundreds or thousands of FPGAs in the cloud for a short period of time. As we will see in later sections, the FireSim platform drastically simplifies the process of running large parallel FPGA simulations, for example running all ten benchmarks in SPECint17 Rate in parallel on ten FPGAs.

Using a cloud FPGA platform also reduces capital expenditure—many organizations cannot afford to purchase of 100s of large FPGAs (which cost thousands of dollars each). Affordability aside, purchasing a large FPGA farm is often cost ineffective as they are poorly utilized (the farm is completely utilized only near deadlines), must be constantly administered (faulty boards must be replaced, toolchains must be updated), and eventually become obsolete. All of these problems are resolved with cloud-hosted FPGAs, as they are essentially provided as a service, with the entire purchase and management lifecycle handled by the cloud provider. The user can simply request that some number of FPGAs are made available to them, and they appear as a resource on the cloud machines.

Because of these benefits, FireSim focuses on cloud platform support. However, when a user is in the initial stages of deploying FireSim for their design, or runs the same jobs constantly (e.g. for regression testing) maintaining a handful of local FPGAs is cost-effective. We plan to add support for local FPGAs to FireSim in the near future.

1.3 FireSim

FireSim [2, 9] is an easy-to-use, open-source, FPGA-accelerated, cycle-exact hardware simulation platform that runs on cloud FPGAs. FireSim automatically transforms and instruments open-hardware designs (e.g. RISC-V Rocket Chip [5], BOOM [7], and accelerators attached to RISC-V SoCs, such as Hwacha [11] and the NVDLA [1, 8]) with the MIDAS framework into fast (10s-100s MHz), deterministic, FPGA-based simulators that enable productive pre-silicon verification and performance validation.

To model I/O, FireSim includes synthesizeable and timing-accurate models for standard interfaces like DRAM [6], Ethernet, UART, and others. By providing a framework to automatically manage cloud FPGA infrastructure, FireSim lets hardware developers run large parallel workloads on their new hardware designs and lets software developers get a head-start on building software for a novel hardware design, all running on the same infrastructure. In effect, both hardware and software developers work from a single source of truth: the RTL for the hardware design.

Originally developed to simulate new datacenter architectures [9], FireSim is capable of scaling to simulating thousands of multi-core compute nodes, with an optional cycle-accurate network simulation tying them together. Leveraging AWS EC2 F1, FireSim removes the high capex traditionally involved in large-scale FPGA-based simulation, democratizing access to realistic pre-silicon hardware modeling of new designs. For designs that contain RISC-V SoCs, FireSim also provides compatible Linux distributions (Buildroot, Fedora) and automates the process of building complex workloads on top of these Linux distributions and deploying these workloads to hundreds of FPGAs in parallel. By harnessing a standardized

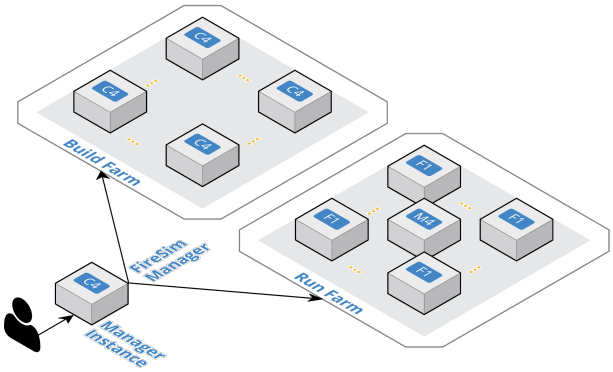

Figure 1: FireSim Host Environment

host platform and providing a large amount of automation/tooling, FireSim drastically simplifies the process of building and deploying large-scale FPGA-based hardware simulations.

2 USING FIRESIM AS PART OF THE AGILE ARCHITECTURE RESEARCH FLOW

Below, we show how FireSim integrates into the agile hardware design stack. We estimate that most computer architecture researchers are interested primarily in evaluating the performance, area, and power of their designs, in that order. FireSim plays a key role in this stack, because it drastically improves on the state of the art for obtaining performance results for novel hardware designs. We consider steps 1 and 7 below to be out of the scope of this paper, but describe some high-level integrations with the tapeout flow at the end of this paper.

(1) Iterate on high-level architectural tradeoffs with a high-level simulator. This is outside the scope of this paper.
(2) Start developing RTL for your design, debug small-scale bugs with software RTL simulation.
(3) Integrate your design into FireSim to simulate your design automatically on cloud FPGAs.
(4) Use FireMarshal to build reproducible RISC-V-based software stacks for your hardware design.
(5) Use FireSim's debugging features to debug your design at FPGA-speeds.
(6) Integrate with Hammer to get area numbers.
(7) Iterate on physical design/tape-out your design. This is outside the scope of this paper.

In the remainder of this paper, we focus on how a user can use FireSim to enable agile development and evaluation of a custom accelerator attached to Rocket Chip over the RoCC interface, by walking through this flow.

3 FIRESIM BASICS

Before we walk through using FireSim to simulate a novel accelerator design, let us clarify some FireSim environment-specific terms. Figure 1 describes what FireSim infrastructure looks like at a high-level on a cloud platform.

- **FireSim Manager** (`firesim`). This program automates the work required to launch FPGA builds and run simulations. Most users will only have to interact with the manager most of the time. This program is similar to the `vagrant` and `docker` tools, but for FPGA simulators instead of VMs/containers.
- **Manager Instance**. This is the AWS EC2 instance that users SSH-into and do work on. The FireSim Manager running on this instance will deploy builds/simulations to other EC2 nodes.
- **Build Farm**. These are instances that are automatically started/terminated by the FireSim manager when you run FPGA builds. The manager will automatically ship source for builds to these instances and run the Verilog to FPGA Image process on them, then copy back build results.
- **Run Farm**. These are a tagged collection of F1 (and M4) instances that the manager automatically launches and deploys simulations onto. You can launch multiple Run Farms in parallel, each with their own tag, to run multiple separate simulations in parallel.

To disambiguate between the hardware designs being simulated and the machines doing the simulating, we also define:

- **Target**. The design and environment under simulation. Generally, a group of one or more multi-core RISC-V microprocessors with or without a network between them.
- **Host**. The computers executing the FireSim simulation—the Run Farm from above.

We frequently prefix words with these terms. For example, software can run on the simulated RISC-V system (*target*-software) or on a host x86 machine (*host*-software).

4 USING FIRESIM TO SIMULATE A CUSTOM ACCELERATOR

As a concrete example of the FireSim flow, we describe how to integrate and simulate a novel hardware accelerator design into FireSim. For this paper, we use the Hwacha vector accelerator [11].

4.1 Initial RTL development and simulation

Initial RTL development works much in the same way as any other system. The developer first chooses a target platform. An example target platform is FireChip, which is the default target in FireSim. FireChip is built on the open-source Rocket Chip generator [5] and integrates various other peripherals, including an Ethernet NIC, a block device controller, a UART (from SiFive's `sifive-blocks` [3]), and other peripherals. FireChip also supports replacing the in-order Rocket cores included with Rocket Chip with Berkeley Out-of-Order Machine (BOOM) cores [7]. As a Rocket Chip-based system, FireChip can include RoCC accelerators. At this stage in the process of developing and integrating a custom accelerator like Hwacha, the user can use the FireChip environment to run small benchmarks on their system in software RTL simulation, using proprietary environments or open tools like Verilator [12]. To distinguish from other forms of software simulation we will discuss in the following section, we call this *target-level* software simulation. Once a

developer is satisfied with their design running their specified microbenchmarks, they can move on to integration into FireSim.

4.2 Integrating a design into FireSim and building FPGA images

Once a user has faith in the basic functionality of their RTL, they can rapidly build a variety of configurations within the FireSim environment. To target their design in FireSim, users can simply drop-in their fork of FireChip and choose the appropriate top-level design to build with three parameter settings given to the manager:

- `DESIGN`
- `TARGET_CONFIG`
- `PLATFORM_CONFIG`

Before running an FPGA-build, FireSim provides two additional levels of software-simulation to allow users to sanity-check that their design has been properly integrated into the FireSim environment: *MIDAS-level* software simulation and *FPGA-level* software simulation.

Generally, MIDAS-level software simulation is the only form of software-simulation that will be utilized by users. This level of software-simulation acts as a sort of integration test of the design into the FireSim environment. This takes the design and wraps it in the FPGA-simulation infrastructure, and then runs the same microbenchmarks that the user was previously running in *target-level* software simulation. This ensures that common mistakes like incorrect I/O model selection are discovered before going through time consuming FPGA-builds.

FPGA-level software simulation is primarily targeted at FireSim developers. This level of software simulation is primarily useful when porting to new FPGA platforms, or when incorporating previously unused features of an existing FPGA platform. This relies on the FPGA vendor's simulation environment (e.g. Xilinx's XSim, provided with Vivado), to faithfully model all of the I/Os exposed by the FPGA platform. This level of simulation is generally the slowest, but is unnecessary for users who are not changing the simulation environment's interfaces with the FPGA host.

Once the user is satisfied that their design has been properly incorporated into the FireSim environment, they can rapidly run many parallel FPGA-bitstream builds. While FireSim does not improve on FPGA-bitstream build time, it does include significant automation to take advantage of parallelism in the cloud. For each `DESIGN`, `TARGET_CONFIG`, and `PLATFORM_CONFIG` triplet that a user requests to be built, FireSim can spin up a Vivado build instance on EC2 for that configuration. Thus, an essentially unlimited number of FPGA bitstream builds can run in-parallel. One the bitstream process is completed, the FireSim manager emails the user with the bitstream's identifier for each configuration, which we will use later to deploy simulations. For example, if we consider a design space where the user has two `DESIGN`s (e.g. two different kinds of accelerator pipeline), two `TARGET_CONFIG`s (e.g. 10 vs. 20 functional units with the design), and two different memory systems to experiment with, represented in the `PLATFORM_CONFIG` (e.g. DDR3 vs. DDR3 + LLC), the FireSim manager can automatically build all eight of these configurations at once, in-parallel, on eight build instances. Since the cost of builds is measured in machine-hours on AWS, running these builds in parallel costs no more than running them serially.

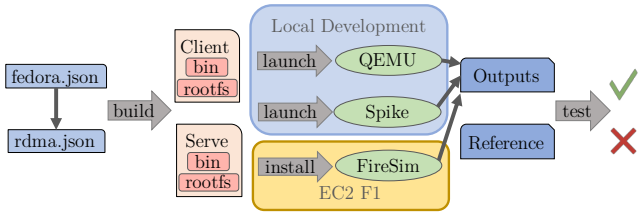

Figure 2: FireMarshal flow. Configuration files are built into a boot binary and rootfs for each job in the workload. The jobs can then be launched in either functional simulation with FireMarshal or cycle-exact simulation with the FireSim manager.

4.3 Using FireMarshal to build reproducible RISC-V based software stacks

An agile hardware design flow requires testing and benchmarking with real software workloads. For example, a project may want a bare-metal unit testing workload utilizing the RISC-V proxy kernel, performance regressions using a Buildroot-based Linux distribution, and end-to-end distributed benchmarks using Fedora. These varied and complex software stacks need to be built consistently and reproducibly by different developers, be compatible between functional and cycle-exact simulation, and limit complexity to project-specific features. In FireSim, workloads are managed by a tool called FireMarshal. Under FireMarshal, workloads can be tracked in a version-controlled repository and reproduced as-needed. Configuration files typically specify a workload to base off of, and any workload-specific changes that must be made to that base. FireMarshal comes with several standard software distributions that are configured to work in the RISC-V SoCs that FireSim supports and receive regular updates (bare-metal, Buildroot, and Fedora). Complex projects may create hierarchical workloads, where common options are defined once and inherited by many workloads (e.g. unit tests and benchmarks likely share startup code). Once the workload is specified, FireMarshal can be used to automatically `build`, `launch`, and `test` the workload in functional simulation. When the workload is ready, the `install` command links the workload into FireSim for cycle-exact simulation. Figure 2 shows a typical usage pattern.

4.3.1 Hwacha End-to-End Testing Workload. The software development process for a custom accelerator like Hwacha begins with a modified Spike functional simulator that acts as a golden model for RTL development, and a high-performance platform for software development. We then define a number of specialized workloads including unit tests, end-to-end regression tests, and complete benchmarks. Listing 1 shows the FireMarshal configuration for an end-to-end testing workload. This workload uses the standard Fedora distribution as a base, then adds a custom Linux kernel (Hwacha changes the context-switch code in Linux), an overlay containing benchmark sources, and a bootstrap script that natively compiles benchmarks and downloads any required packages (`guest-init` scripts run exactly once natively on the workload in QEMU during building). We also specify a run script that executes the benchmark

every time the workload is launched. Finally, the workload configuration includes a testing reference directory that FireMarshal will use to verify the results of the runTest.sh script.

```
{
  "base" : "fedora",
  "run" : "runTest.sh",
  "linux-src" : "hwacha-linux/",
  "overlay" : "hwacha-overlay",
  "guest-init" : "bootstrap.sh",
  "spike" : "hwacha-spike/",
  "testing" : { "refdir" : "e2eRes/"}
}
```

Listing 1: Example benchmark workload configuration. This workload inherits the standard Fedora configuration, adds a custom Linux kernel, file-system overlay, a script to download packages and natively compile benchmarks, and a modified Spike functional model. When launched, this configuration runs the "runTest.sh" benchmark and compares its output to the "e2eRes/" folder (if launched via the test command).

4.4 Running simulations with an accelerator design in FireSim on cloud FPGAs

Once the hardware and software components have been built, the next step is to deploy simulations on FPGAs. By modifying a few configuration files, users can control the entire FireSim simulation environment, including the design under simulation and the host platform. The hardware design being simulated is set in these configuration files and heterogeneous configurations are also possible. The hardware configuration can also add a cycle-accurate network simulation between simulated nodes, provided that the target design includes a network interface. On the software side, a workload can be similarly selected—the user can choose any relevant workload that has been defined using FireMarshal. Combining these components, the FireSim manager can be easily configured to run both large-scale networked simulations, as well as many parallel single-node benchmarks. For example, the `spec17-intrate.json` workload included in the open version of FireSim runs all ten benchmarks included in the SPECint17 Rate benchmark suite in parallel on 10 FPGAs. Once the configuration files are appropriately set, only three commands are required to run an FPGA simulation, with no other intervention by the user:

(1) `$ firesim launchrunfarm`: This launches the Run Farm instances automatically, based on user-supplied counts of the types of instances needed. In the case of running SPEC, this will automatically launch 10 `f1.2xlarge` instances, which each have one FPGA, so that we can run the ten benchmarks in parallel.

(2) `$ firesim infrasetup`: This automatically builds all required software components to run a workload on a particular hardware configuration and copies all necessary infrastructure to each of the F1 (FPGA) instances previously launched.

(3) `$ firesim runworkload`: This starts the actual FPGA simulations and presents a monitoring interface that shows where each simulation is running. It will also automatically copy back results of completed simulations to the manager instance for analysis by the user. With a setting specified in an `ini` file, this can also automatically terminate instances as soon as simulations are computed, for maximum cost efficiency.

4.5 Using FireSim's debugging features to debug a design at FPGA speeds

A common argument against FPGA prototyping is the low visibility into the simulation as compared with software-based RTL simulation or architectural simulation. In the case that we encounter a situation in which a simulation hangs or produces an incorrect result, FireSim offers several debugging features to help users understand what is happening in their hardware design when running on an FPGA. These include assertion and `printf` synthesis [10], automatic integrated logic analyzers (ILA) insertion, and processor commit trace logging.

Assertion and `printf` synthesis were originally introduced in DESSERT [10] and are now available in open-source FireSim. When assertion synthesis is enabled, assertion statements present in the target source RTL, which would traditionally be ignored by an FPGA flow, are wired up to a custom FireSim widget. This widget tracks the firing of assertions within the target design and reports assertions that trigger when running on the FPGA. Designs like Rocket Chip and BOOM by default contain hundreds of assertions in their source RTL, and including assertions in new custom designs is a standard practice. Synthesis of `printfs` is another useful debugging tool. While traditional FPGA flows ignore `printfs` in RTL, `printf` synthesis allows wiring up `printfs` to a widget on the FPGA which transports the information in the `printf` to the host machine, where it is then printed as if the user were running a software RTL simulation. This greatly simplifies the process of extracting information that lives deep within a design for FPGA-speed debugging.

Automatic ILA insertion is another debugging feature available in FireSim. This feature, called AutoILA, assists in bridging the gap between high-level generator descriptions written in Chisel and the ILA infrastructure available from commercial FPGA vendors. FireSim's AutoILA support allows users to apply a Chisel annotation to a signal in a design, which is then automatically wired out to the top of the design by a FIRRTL transformation. From there, Xilinx ILA resources are automatically generated, added, and wired into the simulator RTL, and then built as part of the regular FPGA-image build-flow. At runtime, users can then attach the standard Vivado GUI ILA interface from their manager instance to the F1 instance running the simulator to be debugged and use standard ILA features, like trigger-based recording. Altogether, ILAs enable waveform debugging in an environment similar to software RTL-simulation.

An additional debugging feature currently available in FireSim is processor commit trace logging. FireSim includes a hardware widget that attaches to the committed instruction trace port available on

Rocket Chip and BOOM to capture information about the software executing on a simulated RISC-V design and log it to disk on the host machine. These logs allow users to understand and analyze the software their simulated SoC design is running, down to the individual instruction.

With these debugging tools, FireSim provides drastically greater design introspection capabilities at runtime compared to FPGA prototypes, while enabling orders-of-magnitude better simulation performance than fully observable software RTL simulators.

4.6 Measuring ASIC QoR with Hammer.

Of course, FireSim can only shed light on two of the three terms of the Iron Law of processor performance—to measure cycle time (and area and power) the designer must also push their design through an ECAD flow targeting a specific process technology. To this end, users can take the same design they simulated with FireSim (e.g. their fork of FireChip) and, with little modification, pass it off to Hammer, an open-source, agile, vendor-agnostic ECAD flow. While users can use their own flow, we are working on providing much tighter integration with Hammer in future FireSim releases, providing a highly streamlined, end-to-end process for evaluating novel processor and accelerator designs.

5 CONCLUSION

FireSim [2, 9] is an easy-to-use, open-source, FPGA-accelerated, cycle-exact hardware simulation platform that runs on cloud FPGAs. FireSim automatically transforms and instruments open-hardware designs (e.g. RISC-V Rocket Chip [5], BOOM [7], and accelerators attached to RISC-V SoCs, such as Hwacha [11] and the NVDLA [1, 8]) with the MIDAS framework into fast (10s-100s MHz), deterministic, FPGA-based simulators. FireSim can scale out simulations to arbitrarily many FPGAs in the public cloud, to enable productive design-space exploration without the capex of building a large FPGA farm. To manage the complexity of building and deploying simulations using a variety of different target-software toolchains, operating systems, and benchmark software, FireSim provides FireMarshal, an automated workload-generation tool. To ease the traditional challenges in debugging FPGA-based designs, FireSim provides multiple introspection tools, including Auto-ILA generation, assertion and `printf` synthesis, and trace log capture that give the designer considerably more insight into their design than would be possible with a conventional FPGA-prototype. Altogether, FireSim provides an unparalleled set of features among open-source, FPGA-based simulation frameworks with still more features yet to be released—we strongly encourage you to check out FireSim for your next RISC-V SoC project.

ACKNOWLEDGMENTS

Research partially funded by DARPA Award Number HR0011-12-2-0016 and ARPA-E Award Number DE-AR0000849. Research also partially funded by RISE Lab sponsor Amazon Web Services, ADEPT Lab sponsor Intel, and ADEPT Lab affiliates Google, Huawei, Siemens, SK Hynix, and Seagate. Any opinions, findings, conclusions, or recommendations in this article are solely those of the authors and do not necessarily reflect the position or the policy of the sponsors.

REFERENCES

[1] 2018. The NVIDIA Deep Learning Accelerator (NVDLA). http://nvdla.org/.
[2] 2019. FireSim: Easy-to-use, Scalable, FPGA-accelerated Cycle-accurate Hardware Simulation in the Cloud. https://github.com/firesim/firesim.
[3] 2019. SiFive Blocks: Common RTL blocks used in SiFive's projects. https://github.com/sifive/sifive-blocks.
[4] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2009. *Above the Clouds: A Berkeley View of Cloud Computing.* Technical Report UCB/EECS-2009-28. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html
[5] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. 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
[6] David Biancolin, Sagar Karandikar, Donggyu Kim, Jack Koenig, Andrew Waterman, Jonathan Bachrach, and Krste Asanović. 2019. FASED: FPGA-Accelerated Simulation and Evaluation of DRAM. In *The 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'19) (FPGA '19).* ACM, New York, NY, USA, 10. https://doi.org/10.1145/3289602.3293894
[7] Christopher Celio, Pi-Feng Chiu, Borivoje Nikolic, David A. Patterson, and Krste Asanović. 2017. *BOOM v2: an open-source out-of-order RISC-V core.* Technical Report UCB/EECS-2017-157. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-157.html
[8] Farzad Farshchi, Qijing Huang, and Heechul Yun. 2019. Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim. In *In proccedings of The 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications, at HPCA 2019.* http://arxiv.org/abs/1903.06495
[9] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, Qijing Huang, Kyle Kovacs, Borivoje Nikolic, Randy Katz, Jonathan Bachrach, and Krste Asanović. 2018. FireSim: FPGA-accelerated Cycle-exact Scale-out System Simulation in the Public Cloud. In *Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA '18).* IEEE Press, Piscataway, NJ, USA, 29–42. https://doi.org/10.1109/ISCA.2018.00014
[10] D. Kim, C. Celio, S. Karandikar, D. Biancolin, J. Bachrach, and K. Asanović. 2018. DESSERT: Debugging RTL Effectively with State Snapshotting for Error Replays across Trillions of Cycles. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL).* 76–764. https://doi.org/10.1109/FPL.2018.00021
[11] Yunsup Lee, Colin Schmidt, Albert Ou, Andrew Waterman, and Krste Asanović. 2015. *The Hwacha Vector-Fetch Architecture Manual, Version 3.8.1.* Technical Report UCB/EECS-2015-262. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-262.html
[12] Wilson Snyder. 2019. Introduction to Verilator. https://www.veripool.org/wiki/verilator.