Department: Head
Editor: Name, xxxx@email

Accessible, FPGA Resource-Optimized Simulation of Multi-Clock Systems in FireSim

David Biancolin, Albert Magyar, Sagar Karandikar, Alon Amid, Borivoje Nikolić, Jonathan Bachrach, Krste Asanović
University of California, Berkeley

***Abstract*—Given the complexity of modern systems-on-chip (SoCs), hardware-assisted verification is an integral part of the chip-design process. However, chip designers often need to choose between richly featured but expensive emulation platforms or faster, cheaper, but less debuggable FPGA prototyping solutions. FireSim, an open-source, FPGA-accelerated hardware emulation platform hosted in the public cloud, attempts to accessibly offer the best of both worlds. This article highlights two new FireSim capabilities that help realize this goal: multi-cycle resource optimizations, which can enable an eight-fold increase emulated core count, and FPGA-agnostic support for multi-clock systems. These supplement existing FireSim features which provide a foundation for productive emulation, including a cloud manager to automatically scale out experiments and a rich debug toolkit.**

WHILE much can be said about the state of Moore's law, one unambiguous truth about chip design is that it remains expensive to build the first chip. Worse still, the drivers of this non-recurring engineering (NRE) cost are diverse—there appears to be no panacea for making chips cheap. However, with the explosion of new applications in the domains of artificial intelligence and the internet of things, specifically in energy-constrained environments, the need for custom silicon has never been greater.

One large driver of NRE cost, specifically of pre-silicon verification, validation, and software-development costs, is that of full-system simulation. With software-based simulation often being too slow or too inaccurate, SoC designers turn to hardware acceleration in the form of FPGA prototypes and hardware emulation. These technologies faithfully represent the chip while still executing quickly enough to run complete system stacks and applications. Of the two, FPGA prototypes tend to be faster and less expensive

 Published by the IEEE Computer Society

and therefore see extensive use in software development. Given its low barrier to entry, most academic SoC frameworks rely on FPGA prototyping for full-system evaluation [1], [2]; however, capacity constraints and limited debugging can slow design iteration in architecture research. Conversely, hardware emulators can be deterministic (when used as simulation accelerator) and provide a debuggable simulation environment akin to a very fast register-transfer level (RTL) or gate-level simulator, rendering them critical for pre-silicon verification.

Providing a hardware-accelerated, software-simulator-like emulation experience is no small undertaking for a number of reasons [3]:

- The scale of modern SoCs typically requires multi-chip systems to emulate, with cleverly designed intra-emulator interconnect to achieve good simulation performance.
- Mapping the chip to parallel emulator hardware is a fundamentally more difficult task than compiling software simulators, leading to long compile times.
- Chip primitives may not map cleanly to emulation hardware and may need to be substituted with emulation equivalents.
- Providing software-like debug features requires additional hardware support and tooling.

To meet these challenges, commercial hardware emulators use ASICs (e.g., Cadence Palladium), custom FPGAs (e.g., Mentor Veloce), or custom integration of commercial off-the-shelf (COTS) FPGAs (e.g., Synopsys ZeBu). To map user designs to customized hardware, vendors provide specialized compilers and broad suites of emulation IP libraries. While these extensive capabilities come at high cost, the unique feature-set delivered by commercial emulators makes them invaluable for increasingly expensive commercial SoC projects. Unfortunately, the cost of hardware emulation puts it out of reach for smaller companies and academics, and limits the scalability of its use at larger companies, where engineers must schedule time on limited emulation hardware.

With FireSim [4] (https://fires.im), we aim to democratize access to hardware emulation. First, we avoid the use of custom hardware in favor of single-instance, COTS FPGAs in the public cloud. Second, we have made FireSim's software ecosystem, which includes a compiler, emulation models, and an FPGA cluster manager, completely open-source. While FireSim was originally developed for simulating warehouse-scale computers by cycle-accurately networking together emulators across hundreds of FPGAs, here we expand on two new FireSim developments in service of making the underlying single-FPGA emulation technology more capable. First, to fit larger designs on a single FPGA without partitioning, we've introduced FPGA resource optimizations, notably instance multi-threading. Second, to enable rapid design space exploration of more realistic SoC clock organizations, we've introduced support for simulation of arbitrarily many fixed-frequency clocks. These capabilities, supplemented with FireSim's automation for managing emulator compilation and execution in the public cloud, and its extensive debug toolset, greatly expand the accessibility of hardware emulation.

A Primer on FireSim

Hardware emulation is typically deployed in one of two modes: *in-circuit emulation* replaces the SoC with an emulator and speed adapters to debug an application driving real-world I/O, whereas *transaction-based emulation* leverages the emulator as an accelerator for a deterministic, closed-world simulation. FireSim is designed for this second mode. It takes an input design, the *target*, and maps it to a hardware-accelerated platform, the *host*, to build an emulator that cycle-accurately represents the source. Like other emulation platforms, FireSim emulators are *co-hosted*: a CPU-hosted *driver* process interacts with a PCIe-attached, FPGA-hosted component. In order to support deterministic emulations, ease implementation over large FPGAs, and enable specialized resource optimizations, FireSim emulators are implemented as latency-insensitive bounded dataflow networks (LI-BDN) [5]. These networks consist of latency-insensitive *models*, the nodes of the graph, which communicate by exchanging *tokens*, messages that represent the value on a wire on a given cycle. Models can be hosted on the CPU or on the FPGA. FPGA-hosted models can be handwritten, emulation-only blocks or transformed from the target design. We depict a typical input system and its equiva-

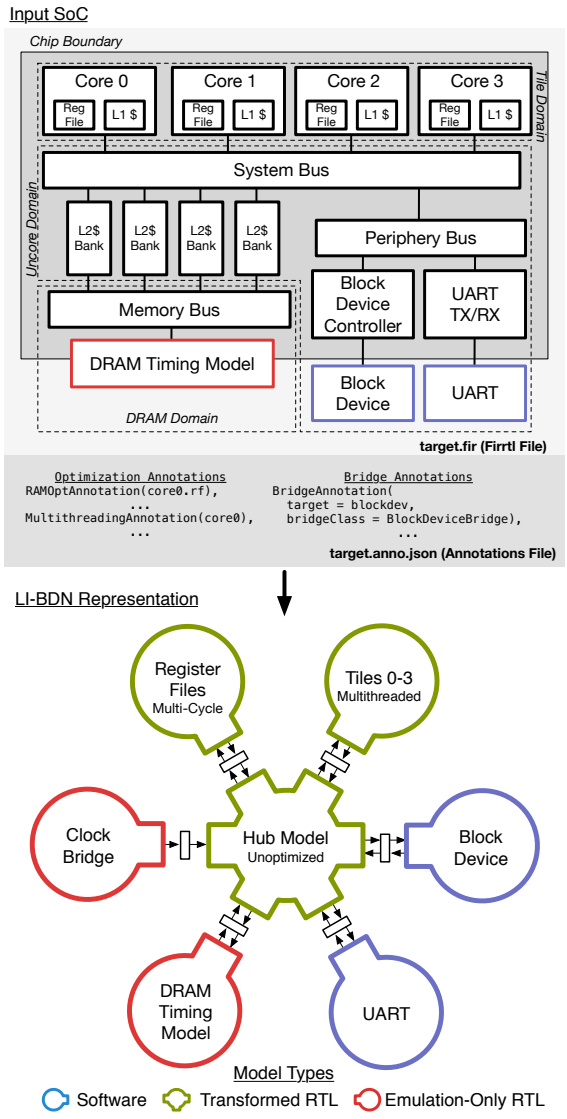

Figure 1. A example chip specification (top), and resulting LI-BDN after Golden Gate compilation (bottom). In FireSim, these networks always have a star topology: bridges and optimized models communicate directly with an unoptimized hub model (the remaining design after optimizable modules have been extracted).

lent LI-BDN representation in **Figure 1**.

FireSim's compiler, Golden Gate [6], is responsible for translating the target into an LI-BDN, performing optimizations on nodes of the network, and synthesizing auxiliary emulation infrastructure including debugging utilities. Golden Gate accepts input designs expressed in FIRRTL [7], an open intermediate representation for RTL circuits. These inputs include a description of the circuit and a set of *annotations*, which call out optimization opportunities (e.g, by labelling a multi-ported RAM in the IR) and identify modules that should be replaced with emulation-specific IP (referred to as *bridges*). FIRRTL is emitted natively by Chisel, a Scala-based HDL, but flows to translate other HDLs and hardware IRs to FIRRTL exist and could be used instead.

Golden Gate compilation proceeds in three phases. In *Target Transformation*, the design's module hierarchy is mutated to remove bridges and extract optimization candidates. In *Simulator Synthesis*, the design is first translated into a baseline LI-BDN, after which optimizations are then implemented as modular refinements on nodes of the network. In *Platform Mapping*, bridges, including hardware interfaces for software-hosted models, are synthesized and connected to host resources, such as off-chip memory. At this point, Golden Gate emits Verilog compatible with a host-FPGA shell project and a C++ header that describes the organization of the FPGA-hosted piece of the emulator and is linked into the driver process.

FireSim has been optimized to run on Amazon Web Services (AWS) EC2, with emulations hosted on EC2's F1 instances that feature Intel Xeon CPUs with PCIe-attached Xilinx VU9P FPGAs. FireSim provides a *manager* utility, which dynamically launches instances to parallelize compilation and emulation jobs (as described in *Cloud Accessibility*).

FireSim is integrated into the Chipyard SoC design framework [8], which contains a large corpus of SoC IP developed by a growing community of designers, including RISC-V processor cores, cache generators, hardware accelerators, and periphery IP. Chipyard SoC generators are written in Chisel, which makes it easy to elaborate many different SoC configurations across a large design space. As we will show, this dovetails with FireSim's ability to rapidly evaluate many designs in parallel using the public cloud.

The performance of a FireSim emulator varies as a function of the target SoC and its simulated behavior. FireSim emulators use a single emulation clock which, on a Xilinx VU9P, closes timing between 30 and 190 MHz. However, models may dynamically take more than one cycle to execute,

leading to runtime fluctuations in performance. This is quantified by the FPGA-to-Model-Cycles Ratio (FMR) [9]: the average number of FPGA cycles used to simulate a clock cycle. For unoptimized emulators of typical Chipyard SoCs, FMR tends to range between 1.0 and 1.5. For a particular run, dividing the FPGA frequency by FMR gives the effective emulation frequency, f_{emul}, of the target. As we will discuss, resource optimizations radically improve emulation capacity at the expense of FMR.

Resource Efficiency

With the increasing size and complexity of modern SoCs, the finite capacity of existing FPGA platforms has long been a significant hurdle to productive simulation. In contrast with software RTL simulators and processor-based emulation platforms like Palladium, FPGAs are generally constrained by the need to directly target design RTL specifications to FPGA implementations directly and at a fine granularity. This approach can result in simulators that require more resources than are provided by even an advanced, high-end FPGA device for moderately sized target systems. While this limitation is often addressed by partitioning simulators across multiple FPGA dies using either general-purpose FPGA EDA tools or specialized emulators' software flows, we incorporate an alternative approach that reflects common root causes of excessive resource utilization:

- Some microarchitectural primitives common in ASIC design map poorly to FPGAs, such as complex RAMs and CAMs [10].
- Increased parallelism in target systems often implies numerous instances of identical blocks, each consuming distinct emulator resources.

Notable examples of poorly mapped structures in ASIC RTL include register files in superscalar cores and reordering structures. While the underlying highly ported memories may be efficiently realized in ASICs with library IP, their RTL specifications cannot efficiently exploit FPGA memory structures and therefore consume significant logic resources. To avoid this bottleneck, the Golden Gate compiler may automatically optimize a simulator to dedicate fewer resources to modeling such FPGA-hostile memories by identifying each instance in the hierarchy, denoting it as a logical partition of the target, and mapping this partition to a multi-cycle decoupled model [6]. This model implements a cycle-exact simulation of the target memory in the LI-BDN protocol via efficient, serialized accesses to underlying two-port block RAMs. Since the Golden Gate compiler is capable of transforming arbitrary FIRRTL circuits, regardless of their application domain, this optimization can be universally applied to any FIRRTL input by labeling any FPGA-hostile memories with annotations resembling compiler directives in other languages. While the introduction of multi-cycle models may increase FMR and reduce throughput, it can significantly reduce the resources required to simulate targets with complex memories, allowing larger designs to fit on existing FPGAs.

Building upon this first optimization, we further exploit the use of multi-cycle models to reduce the resource footprint needed to simulate a ubiquitous feature of modern, highly parallel SoCs: sets of identical instances. Drawing inspiration from software simulators and specialized emulators, where common code may be repeatedly executed to model an arbitrary number of copies of a given block, we introduce *threaded models* to FireSim, where a single LI-BDN model amortizes the cost of a module's logic over multiple simulated instances, each an independent thread of simulation. As with the preceding memory optimization, this transformation can be applied to any FIRRTL circuit by merely listing the target set of instances in a compiler directive. However, rather than just specific memory ports, the threading optimization can broadly "de-duplicate" all logic resources used across a set of instances, including both lookup tables (LUTs) and digital signal processing (DSP) blocks. Though instance threading does not intrinsically reduce the number of state bits required to simulate the full set of instances–and indeed it carries the overhead of selecting the correct state and I/O for the active instance–it can even help reduce pressure on FPGA RAM resources by packing N copies of a target memory that was previously too shallow to efficiently exploit inflexible block RAM primitives. While this technique has been painstakingly implemented by hand in previous

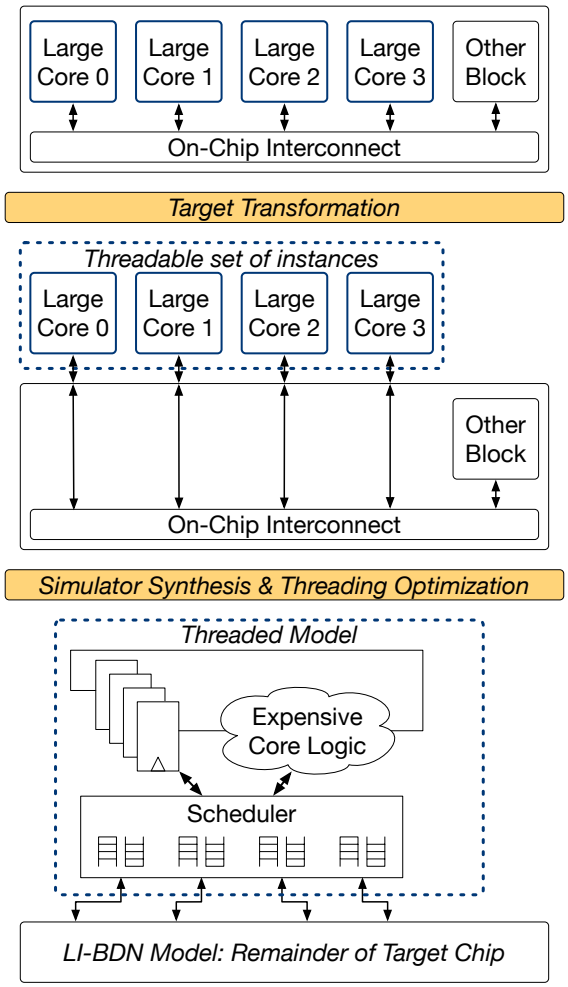

Figure 2. Optimizing a emulator of a four-core target system with instance threading. Rather than mapping each target core to its own costly FPGA implementation, the full set of identical instances is simulated with a threaded model. Each cycle, this model schedules a thread of execution corresponding with a particular instance on a single copy of the underlying logic, saving FPGA resources.

handwritten microarchitectural simulators [11], [12], the Golden Gate compiler is the first tool to automatically translate repeated instances in RTL designs to emulators incorporating threaded models, as shown in **Figure 2**.

As with multi-cycle memory models, the underlying serialization of instance threads reduces emulation throughput; however, the latency-hiding benefits of threaded schedules can reduce the relative overhead of introducing further optimizations or co-simulated software models. Taken in combination, these two optimizations can dramatically reduce the resource footprint of FPGA-based emulators by trading off space for throughput. By exposing these automated transformations through simple flags, Golden Gate provides a user-friendly mechanism to avoid "cliffs" in device capacity.

To demonstrate this capability, we explore how Golden Gate optimizations can increase the effective capacity of commodity AWS-hosted FPGAs to allow emulating larger multi-core SoCs. In particular, we evaluate emulators of two Chipyard-based systems:

- A *general-purpose* SoC incorporating varying numbers of five-issue configurations of the open-source BOOM out-of-order processor.
- A *domain-specific accelerator* with multiple tiles, each pairing an in-order Rocket RISC-V control processor with a linear algebra co-processor based on an 8x8 systolic array of bfloat16 processing elements (PEs).

Figure 3 shows the number of FPGA resources used to implement varying core counts–either BOOM cores or systolic array accelerator cores–both with and without the two Golden Gate resource optimizations enabled. Using standard FPGA mapping techniques an AWS-hosted Xilinx VU9P FPGA can fit only a two-core BOOM target or a single accelerator instance; higher core counts of each design require more LUTs than are available in the host VU9P FPGA, rendering them infeasible to implement even before placement or routing. However, applying both optimizations extends the simulation capacity to sixteen BOOM cores or eight accelerator cores; not only do these large, optimized simulators with instance threading and multi-cycle models reduce LUT utilization to a feasible amount, but they successfully close timing at 50MHz for all but the eight-core accelerator target, which runs at 35MHz. While adding resource threading has some small overhead that causes LUT utilization to grow nominally as core count increases, it is interesting to note that the eight-core, threaded simulation of the multi-accelerator system uses the exact same number of FPGA DSP block resources (594 in total) as an unoptimized simulator of a single accelerator. Finally, even though the threading optimization does not intrinsically reduce the the-

oretical memory footprint of multi-core designs, the ability to better pack target memories into fixed-size primitives reduces BRAM utilization in optimized BOOM simulators.

This resource-efficiency advantage does come at a tradeoff in simulation speed; while the unoptimized simulators all run at an observed FMR of 1.62 FPGA cycles per emulated cycle, the serialization imposed by sharing sets of resources across larger targets further slows the optimized simulators. The optimized simulators range in performance based on both the number of threaded instances and the presence of combinational paths connecting optimized models; while it takes an average of only eight FPGA cycles (a net simulation rate of 4.38MHz) to simulate a cycle of the eight-accelerator system, the extra serialization of complex BOOM register files requires 32 cycles to simulate a cycle of the 16-core BOOM system, yielding an overall throughput of 1.56MHz for the slowest simulator configuration. However, this tradeoff is ultimately balanced by the ability to accessibly simulate target designs that were previously impossible to map to a single FPGA without the expense or extreme slowdown of previous techniques such as partitioning. Furthermore, as an orthogonal dimension to partitioning, the ability to increase the simulation capacity of each individual FPGA carries the potential to benefit both single- and multi-FPGA platforms.

Cloud Accessibility

Traditionally, the complexity of FPGA platform bringup and management has hampered broader adoption of FPGA-based simulation tools, especially outside of large commercial users. However, the introduction of FPGAs in the public cloud in 2017 (with Amazon Web Services' EC2 F1 instances) drastically improved the practicality of managing and deploying FPGAs at scale, even for small organizations. This scalability was critical in enabling FireSim's original purpose of simulating warehouse-scale machines from the ground-up [4].

To productively harness the capabilities of cloud FPGA platforms, FireSim includes the *manager*, a tool that orchestrates the deployment of emulations across clusters of FPGA-accelerated compute instances (*Run Farms*) and emulator builds (i.e., FPGA synthesis/P&R) across clusters of standard compute instances (*Build Farms*).

Most FireSim users interact with only the manager instance. Configuring a simulation run requires only specifying parameters of the simulation in a set of `.ini` configuration files and running a sequence of three `firesim` commands: `launchrunfarm`, which creates instances of the requested type; `infrasetup`, which copies all emulation collateral to these instances; and `runworkload`, which launches emulations and copies all remote results back to the manager instance for analysis. Similarly, to run parallel builds, users need only specify a list of configurations to the manager. The user then runs the `buildafi` command, which automatically launches build instances, completes the entire build process, tears down the instances, and finally reports the AWS identifiers of the final bitstreams.

Flexible Multi-clock Emulation

The enormous scale of modern chips not only demands large emulation capacity but begets complex clocking organizations that can be challenging to map onto an emulator. As an interim step towards emulating these directly, FireSim allows the user to simulate a user-defined set of fixed-frequency clocks. To do so, users instantiate a single *clock bridge* in their design from which they source clocks for their system. Clock-domain crossing (CDC) circuitry, assuming it is FPGA synthesizable, may be left unchanged. The clock bridge generates an infinite stream of *clock tokens*: each token encodes which clocks have an edge in a given timestep.

Multi-clock designs still use a single emulation clock: multiple target clocks are emulated by independently clock-gating the emulation clock for each target domain based on the current clock token. To maintain compatibility with FireSim debug features and optimizations, all extracted satellite models remain synchronous to a single clock, and thus, are conventional primitive LI-BDNs. Only the hub model, which is never subject to optimizations, implements multi-clock emulation semantics. In the hub, a clock token is processed over a two-stage control pipeline. In the first stage, clock edges are launched and output

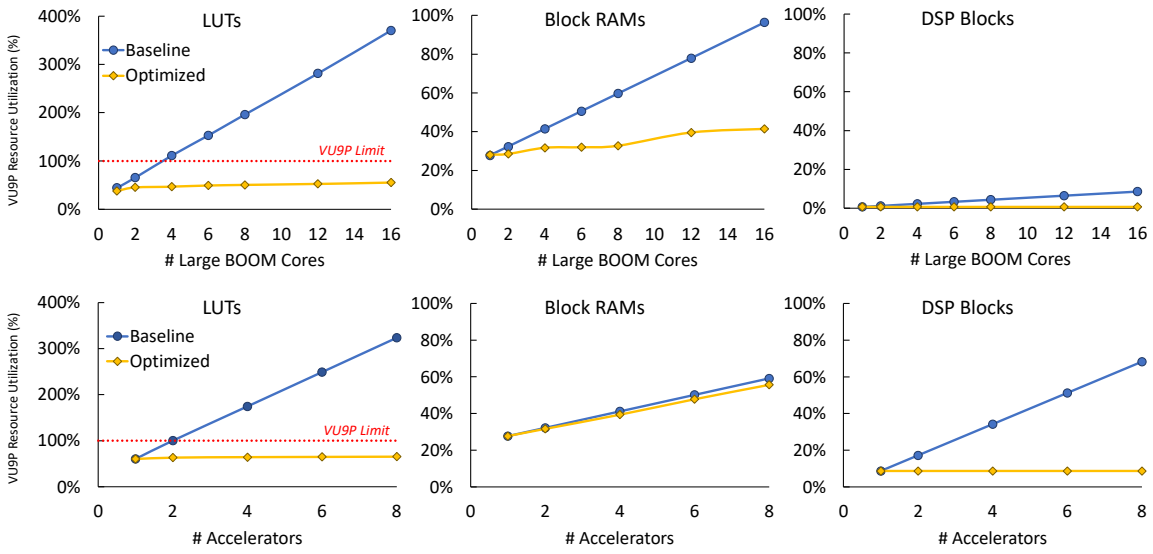

Figure 3. A comparison of the number of FPGA lookup table (LUT), block RAM (BRAM), and digital signal processing (DSP) block resources required to implement emulators of Chipyard-based target designs with varying numbers of identical instances of coherent cores. Each core of the Large BOOM is a five-issue configuration of the BOOM RISC-V out-of-order core, while each instance of the Accelerator core combines an in-order Rocket RISC-V control processor with an 8x8 systolic array coprocessor to target linear algebra on bfloat16 data. Implementation results results were obtained using Vivado 2018.3 for both a *Baseline* unoptimized simulator configuration and an *Optimized* flow with both multi-cycle memory models and instance threading. The red dotted line depicts the number of LUTs available in the Xilinx VU9P devices hosted in AWS F1 instances. Together, the two optimizations extend simulation capacity from two large BOOM cores to sixteen and from one accelerator instance to eight.

FSMs in the launching domains are reset; then in the second stage, multi-model combinational paths, if they exist, are allowed to resolve. Under this implementation, if all simulated clocks are integer divisions of the fastest clock in the system, the fastest clock in the system can be emulated with unity FMR. On a Xilinx VU9P FPGA and the AWS-provided shell project, our current implementation, which uses global clock buffers to implement clock gating, scales reliably to twelve target clocks before running into placement challenges. In the future, we plan to expand this by using finer grained clock-gating schemes for smaller target clock domains and more relaxed clock placement and timing constraints.

This flexible approach hides any FPGA-specific handling of multiple clocks from the user and, in modern FPGAs, is scalable to systems with many clocks. Combined with FireSim's automation, this makes it easy to explore performance tradeoffs between different clock domain organizations. To illustrate this, we evaluated the SPECspeed 2017 integer on three different RISC-V chip configurations (illustrated in the block diagram at the top of Figure 1): a completely synchronous system running at 1.5 GHz (*Synchronous*), a two-domain system (*Two Domains*) with a DDR3 memory system running at 1.0 GHz behind an asynchronous crossing, and a three-domain system (*Three Domains*), in which we introduce a rational crossing between the processor cores and the uncore (running at 1.5 GHz and 750 MHz, respectively). All three systems consist of four Rocket in-order cores with 16 KiB L1 instruction and data caches and a shared 256 KiB L2 cache. To rapidly obtain these results, we ran each benchmark in the suite in parallel (requiring 11 emulators since we split `625.xz`'s inputs in two to halve its runtime), on each configuration in parallel, using a total of 33 emulators.

The runtime of the simplified clock organizations relative to the three-domain configu-

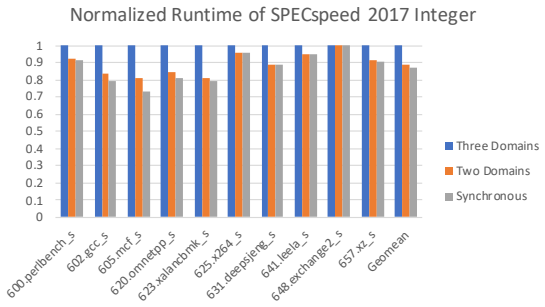

Figure 4. SPECspeed 2017 runtime of simplified clock organizations: a completely synchronous design (*Synchronous*) and one with just the DRAM in a separate domain (*Two Domains*), normalized against a realistic one (*Three Domains*).

Design	Run Time (h)	f_{emul} (MHz)
Three Domain	44.2	81.0
Two Domain	36.4	80.7
Synchronous	28.0	104.5

Table 1. Emulation performance summary of the three SoC configurations running SPECspeed.

ration (**Figure 4**) precisely reveals the performance costs of introducing more clock domains. Applications whose working set does not fit in L1 cache see a large slowdown when a core-to-uncore crossing exists. Similarly, those whose working set does not fit in the L2 cache, notably `605.mcf_s`, see another, generally smaller, performance penalty when an uncore-to-DRAM crossing is present. Crucially, FireSim let us collect these results quickly: **Table 1** reports f_{emul} of the core clock and the wall-clock time of the longest-running benchmark. While these emulators close timing at 110 MHz, emulation performance falls to approximately 80 MHz in the multi-clock designs because the DRAM controller clock frequency (1 GHz) is not an integer division of the core clock frequency (1.5 GHz). As a result, the core clock can only fire in three of every four emulator time steps, since one time step is spent launching a DRAM domain clock edge when there is no concurrent core clock edge. This bounds the best-case f_{emul} to 82.5 MHz. Nonetheless, this is still sufficiently fast such that, when combined with the parallelism of using emulation in the cloud, and the ease of mapping different clock domain organizations to an emulator, FireSim enables rapid and precise design space exploration of SoC clock organizations.

Debugging and Visibility

One of the main differentiators between hardware emulators and FPGA prototypes is the various debugging and visibility features afforded to the user. In FPGA prototypes, users typically rely on vendor-provided in-circuit debugging blocks, such as Xilinx's integrated logic analyzers (ILA). Unfortunately, limitations on the number of recorded signals, finite buffer size, and non-determinism at the FPGA I/O boundary significantly increase debugging effort. In contrast, custom hardware emulators often provide complete, reproducible waveform visibility, print statements, and assertions over the entire design. To bridge this gap, FireSim's debugging features bring several emulator-like debugging capabilities to commodity FPGAs.

As with FPGA prototyping tools, FireSim provides compiler directives to automate ILA integration for traditional in-circuit waveform debugging. However, FireSim also provides interfaces to synthesize target-design printf and assertion statements through compiler directives labeling either individual constructs or entire modules. These features, originally introduced in DESSERT [13], provide FireSim with debugging visibility closer to software RTL simulation, enabling a simulation to halt on assertion failures, or to print a specific signal value upon selected events. These features support traditional, user-friendly techniques like "printf debugging" or invariant checks.

Finally, FireSim provides a collection of system-level debugging features, collectively named FirePerf [14]. These include instruction trace generation and visualization as well as out-of-band performance counter generation. These features can be customized to generate output only for particular windows of execution specified by a simulation trigger that may incorporate both cycle count and values of target-specific signals like program counters. These features enable system-level analysis of specific regions of interest in long-running emulations.

Together, these debugging features enable the identification and resolution of hardware bugs trillions of cycles into emulation time with both convenience and determinism. FireSim's debug-

ging capabilities provide an accessible and affordable middle ground between fast but low-visibility FPGA prototypes and slow but highly introspective software RTL simulation.

Conclusion

With its use of FPGAs in the public cloud, open-source toolchain, and unique combination of features, such as multi-cycle resource optimizations, FireSim expands accessibility to hardware emulation capabilities to a broader audience of chip designers. The FireSim community is working towards further improving this accessibility by adding new features such as support for local FPGAs, detailed emulation of targets that dynamically scale frequencies, integration of a Verilog-to-FIRRTL flow to make it possible to optimize currently blackboxed Verilog modules, and state snapshotting and replay for full visibility debugging. These features will further drive FireSim adoption and help put hardware emulation in the hands of users for whom it was previously out of reach.

Acknowledgments

The information, data, or work presented herein was funded in part by the Defense Advanced Research Projects Agency (DARPA) through the Circuit Realization at Faster Timescales (CRAFT) Program under Grant HR0011-16-C0052, by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number DE-AR0000849, and NSF CCRI Award 2016662. Research was partially funded by ADEPT Lab industrial sponsors and affiliates, and supported by gifts provided by Amazon Web Services. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

REFERENCES

1. P. Mantovani, D. Giri, G. D. Guglielmo *et al.*, "Agile soc development with open ESP : Invited paper," in *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2020, San Diego, CA, USA, November 2-5, 2020*. IEEE, 2020, pp. 96:1–96:9.
2. J. Balkind, M. McKeown, Y. Fu *et al.*, "OpenPiton: An open source manycore research framework," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '16. New York, NY, USA: ACM, 2016, pp. 217–232.
3. W. N. Hung and R. Sun, "Challenges in large fpga-based logic emulation systems," in *Proceedings of the 2018 International Symposium on Physical Design*, ser. ISPD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 26–33.
4. S. Karandikar, H. Mao, D. Kim *et al.*, "FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, June 2018, pp. 29–42.
5. M. Vijayaraghavan and A. Arvind, "Bounded dataflow networks and latency-insensitive circuits," in *Proceedings of the 7th IEEE/ACM International Conference on Formal Methods and Models for Codesign*, ser. MEMOCODE'09. IEEE Press, 2009, p. 171–180.
6. A. Magyar, D. Biancolin, J. Koenig *et al.*, "Golden gate: Bridging the resource-efficiency gap between asics and fpga prototypes," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.
7. A. Izraelevitz, J. Koenig, P. Li *et al.*, "Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations," in *Proceedings of the 36th International Conference on Computer-Aided Design*, ser. ICCAD '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 209–216.
8. A. Amid, D. Biancolin, A. Gonzalez *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.
9. M. Pellauer, M. Vijayaraghavan, M. Adler *et al.*, "A-port networks: Preserving the timed behavior of synchronous systems for modeling on fpgas," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, no. 3, Sep. 2009.
10. H. Wong, V. Betz, and J. Rose, "Comparing fpga vs. custom cmos and the impact on processor microarchitecture," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 5–14.
11. M. Pellauer, M. Adler, M. Kinsy *et al.*, "Hasim: Fpga-based high-detail multicore simulation using time-division multiplexing," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, 2011, pp. 406–417.
12. Z. Tan, A. Waterman, H. Cook *et al.*, "A case for fame: Fpga architecture model execution," in *Proceedings of*

the 37th annual international symposium on Computer architecture, 2010, pp. 290–301.

13. D. Kim, C. Celio, S. Karandikar *et al.*, "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)*, 2018, pp. 76–764.
14. S. Karandikar, A. Ou, A. Amid *et al.*, "FirePerf: FPGA-accelerated full-system hardware/software performance profiling and co-design," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 715–731. [Online]. Available: https://doi.org/10.1145/3373376.3378455

David Biancolin is currently a PhD candidate in the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. His dissertation work studies FPGA-based, discrete-event simulation techniques for SoCs with dynamically scaling clocks. He has a BASc in Engineering Science from the University of Toronto. Contact him at biancolin@berkeley.edu.

Albert Magyar is currently a PhD candidate in the ADEPT Lab at the University of California Berkeley. His research interests include increasing productivity of RTL design and improving the usability of FPGA simulation. He has a BS in Nuclear Engineering and a BA in Computer Science from the University of California, Berkeley. Contact him at albert.magyar@berkeley.edu.

Sagar Karandikar is currently a PhD student in the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. His research focuses on exploring hardware-software co-design in warehouse-scale machines. He has a BS and an MS in Electrical Engineering and Computer Science from the University of California, Berkeley. He is a member of the the Association for Computing Machinery (ACM) and the IEEE. Contact him at sagark@eecs.berkeley.edu.

Alon Amid is currently a PhD candidate in the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. His current research focus includes parallel and distributed computing, energy-efficient processors and architectures, and hardware-software co-design. He has a B.Sc in electrical engineering from Technion - Israel Institute of Technology, and an M.S. from the University of California, Berkeley. Contact him at alonamid@berkeley.edu.

Borivoje Nikolić is the National Semiconductor Distinguished Professor of Engineering at the University of California, Berkeley. He has a PhD in electrical and computer engineering from the University of California, Davis. He is a Fellow of the IEEE. Contact him at bora@eecs.berkeley.edu.

Jonathan Bachrach is currently an adjunct assistant professor in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley. He has a PhD in computer science from the University of Massachusetts, Amherst. Contact him at jrb@berkeley.edu.

Krste Asanović is currently a professor in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley. He has a PhD in computer science from the University of California, Berkeley. He is a Fellow of the IEEE and the Association for Computing Machinery (ACM). Contact him at krste@berkeley.edu.