Vertically Integrated Computing Labs Using Open-Source Hardware Generators and Cloud-Hosted FPGAs

Alon Amid, Albert Ou, Krste Asanović, Yakun Sophia Shao, Borivoje Nikolić
University of California, Berkeley
{alonamid,aou,krste,ysshao,bora}@berkeley.edu

***Abstract*—The design of computing systems has changed dramatically over the past decade, but most courses in advanced computer architecture remain unchanged. Computer architecture education lies at the intersection between computer science and electrical engineering, with practical exercises in classes based on appropriate levels of abstraction in the computing system design stack. Hardware-centric lab exercises often require broad infrastructure resources and tend to navigate around tedious practical implementation concepts, while software-centric exercises leave a gap between modeling and system implementation implications that students later need to overcome in professional settings. Vertical integration trends in domain-specific compute systems, as well as software-hardware co-design, are often covered in classroom lectures, but are not reflected in laboratory exercises due to complex tooling and simulation infrastructure. We describe our experiences with a joint hardware-software approach to exploring computer architecture concepts in class exercises, by using open-source processor hardware implementations, generator-based hardware design methodologies, and cloud-hosted FPGAs. This approach further enables scaling course enrollment, remote learning and a cross-class collaborative lab ecosystem, creating a connecting thread between computer science and electrical engineering experience-based curricula.**

***Index Terms*—Computer Architecture, Education, FPGA, Simulation**

I. Introduction

The study of computer architecture traditionally traverses the hardware-software interface: first, it informs programmers about the logical structure of the computing hardware executing their software, so as to explain its performance characteristics and guide the optimization of software; and second, it educates hardware engineers about the principles, techniques, and trade-offs that permeate the design and implementation of computer systems. However, the development of domain-specific computing systems in recent years requires a change in exploration of vertically integrated computing architectures and software/hardware co-design. This emphasis is reflected in explicit hardware and programming models for parallel computing through differentiated hardware architectures such as FPGAs, GPUs, and other custom accelerators in both edge and cloud computing platforms. While traditional computer architecture education has been able to develop analytical models and rules-of-thumb for processor performance based on the simple single-thread software model, modern computer architecture education requires a much more empirical approach, and relies upon a higher degree of empirical experience and simulation.

Approaches to computer architecture education can be paralleled to the Iron Law of Processor Performance [1], [2], which factors processor performance into three components: the number of instructions executed, cycles per instruction, and cycle time. These succinctly capture the contributions from various layers of the computing stack, spanning from algorithms to circuits. While all computer architecture curricula broadly cover fundamentals of the Iron Law, individual courses tend to emphasize either a *software-centric* or *hardware-centric* perspective. A software-centric approach focuses on the interaction between the programming model and microarchitecture, associated with the first two components of the Iron Law: the number of instructions executed and the cycles per instruction. Experiments typically rely on abstract functional or cycle-approximate simulators. Knowledge of RTL description languages is generally not required for most computer architecture courses. A hardware-centric approach focuses on the physical implications of computer architecture, represented by the latter two factors of the Iron Law: the cycles per instruction and the cycle time. Usually tightly entwined with the digital design curricula, this highlights the influence of technological constraints: for example, the relationship between pipelining and register timing; the considerations of SRAM organization in cache and register file design; and the limits on microarchitectural complexity due to wire congestion and power consumption. Lab assignments often involve the

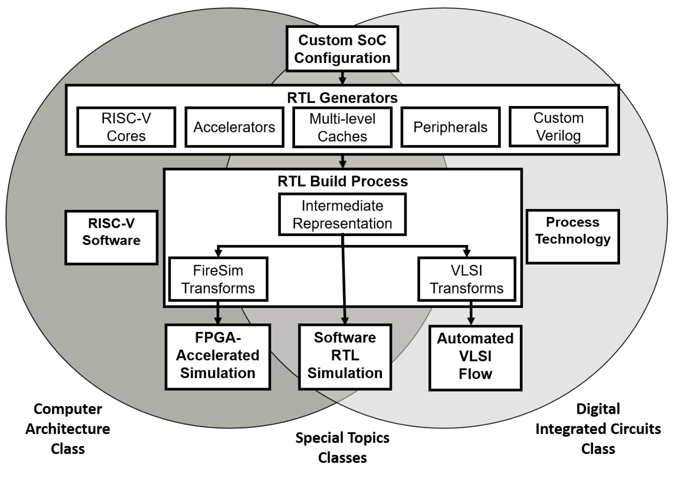


Fig. 1. Multi-Class Flow Using Unified Generator-Based System Framework

manipulation of RTL and associated methodologies [3], [4].

The multi-faceted approach presented in this paper (and summarized in Figure 1) traverses the software-hardware interface, enabling a complete system view of processor performance and efficiency through practical exercises.

II. Generator-based Hardware Design

Generator-based approaches to processor and digital system design have been under development in recent years [5]–[9]. In a generator-based approach, highly parameterized and modular implementations of digital designs using high-level programming language abstractions enable generation of a broad range of RTL designs. Generators describe digital designs at the RTL level in conjunction with additional primitives from functional programming and metaprogramming to encode a high level of parameterization and modularity. In contrast to high-level synthesis (HLS) approaches, hardware generators do not raise the level of abstraction above the RTL level. The *design* of hardware generators still requires a thorough understanding of RTL-based digital design considerations. However, the *usage* of hardware generators is significantly simplified through modularity and parameter configuration schemes. Within the research and development community, these usage models enable additional re-use of building blocks towards specialized computing systems. Within the education community, these simpler usage models open up new educational opportunities for demonstrating complex system interactions through modular and parameter-based lab exercises. Similar approaches for computer architecture education have been proposed in the realm of embedded systems [10], [11], where instead of RTL generators, Architecture Description Languages (ADLs) can be used to compose and configure processor structures at a higher level of abstraction. Unfortunately, unlike RTL, ADLs do not have tooling support to be synthesized into gate-level digital integrated circuits, and therefore their usage is still restricted to only modeling rather than implementation.

The RISC-V ISA is a free and open ISA specification that has garnered industrial and academic momentum in helping lead the open source hardware movement. The rapid adoption of RISC-V in the open-source community is evolving with a strong open-source software ecosystem, as well as accessible open-source processor implementations. The emergence of both open-source hardware and software ecosystems makes RISC-V an excellent tool for computer architecture education [12]–[14]. Prior to the emergence of RISC-V, the primary ISAs used in computer architecture education were MIPS, LC3, x86, and ARM. The leading industry-standard ISAs (x86, ARM) have grown to be too complex to be covered in a course, and they lack open-source hardware processor implementations for students to interact with. On the other hand, the simple RISC ISAs that enable open-source educational hardware implementations lack a sufficiently supportive software ecosystem to later demonstrate more advanced topics and emerging frontiers in computer architecture such as parallel and distributed computing, domain-specific acceleration, and hardware-related security vulnerabilities.

The RISC-V ISA specification has enabled the emergence of many open-source processor implementations that conform to various subsets of the ISA specification – from simple microcontrollers [15], [16] to complete application-class processors with Linux support [6], [17]–[19]. These open source processor implementations provide students unprecedented access and visibility to the nuances behind practical processor implementations. The Rocket Chip generator [6] is a RISC-V-based generator with a sufficient maturity level in terms of functionality and parameterization to be used across industry, research, and education. It consists of Rocket, a highly parameterized in-order RISC-V core, and supports coherent caches and standard interconnects via the TileLink protocol. As the tooling ecosystem around the Rocket Chip generator has evolved, it has become more amenable to be used in a variety of classes to demonstrate various computer architecture phenomena. This ecosystem consists of two primary open-source tools: FireSim FPGA-accelerated simulation, and the Chipyard SoC research and development framework.

The Chipyard framework [20] brings together open source hardware generators and digital design tooling for SoC development. The Chipyard unified generator builds upon Rocket Chip's Chisel-based [21] parameterized hardware generator methodology, and adds a large corpus of open-source IP generators to the existing base library, including an out-of-order core generator [17], a vector-unit generator [22], digital signal processing modules, domain-specific accelerators, memory systems, and peripherals. Chipyard further adds tightly integrated support for simulation and implementation tools through automated HDL transformations and build processes.

In a generator-based joint hardware-software exercises approach using the Chipyard framework, students modify the configuration of an RTL hardware generator and execute an RTL-level simulation through Chipyard's automated build process. Figure 2 demonstrates example configurations used in a lab exercise on the topics of memory hierarchies and caches. In this lab, students change cache parameters and memory hierarchy compositions and observe their interaction with cache-tiling of matrix operations. The configuration interface is similar to dedicated caching simulators, but the generator emits complete SoC RTL which is compiled into a cycle-accurate simulation executing the software workload as it would be executed on actual silicon.

Prior work has surveyed many existing simulators [23], [24], with classifications based on traditional usage within the computer architecture research community as well as through an educational and pedagogical point of view [25]–[27], with some surveys proposing criteria for the evaluation of their suitability for teaching courses in computer architecture. Within the proposed criteria, the generator-based approach demonstrated in this work would be considered under the "Advanced Architecture (AA)" category with "Design Support." The joint model we propose balances the trade-offs between simulation granularity and the level of implementation detail by using a finer scale than those used in prior simulator surveys for education.

III. Accessible FPGA Emulation

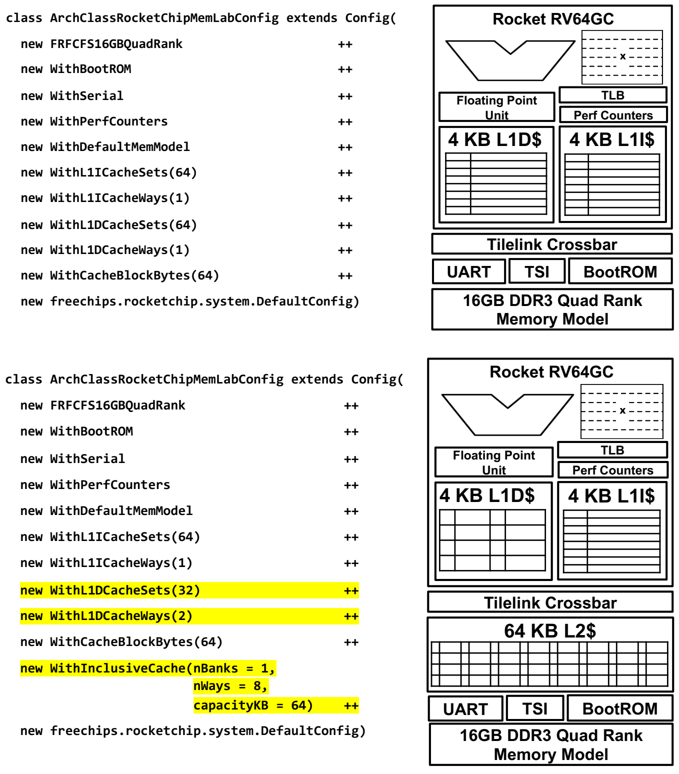
The additional major challenge with RTL-based simulations for computer architecture education is the impact of simulation runtime on the possible scope of software workloads that can be characterized and analyzed by students. Since software RTL simulations are highly detailed, their runtime can be very long – RTL simulations run at rates of 1,000-10,000 cycles per second, which means that software running on the simulated hardware is 10,000× slower than if running on actual silicon. As such, characterizing a program that would take mere seconds to run in real time may take several hours in software RTL simulation. Higher-level computer architecture simulators bypass this constraint by modeling only the relevant details for the particular type of simulation (cache policy, branch prediction, pipeline behavior), while ignoring extra system details. This type of simulation solution is indeed valuable and sufficient in introductory courses, but it leaves something to be desired in system-oriented advanced computer architecture curricula which highlight the nuances of interactions between system components. Some hardware-centric approaches to computer architecture utilize FPGA prototyping as part of the class curricula. However, in addition to the overhead of creating the RTL description of the design and learning the tooling for FPGA prototyping, the students face the challenge of distorted timing accuracy at the periphery and system level of FPGA systems. We use the FireSim [28] FPGA-accelerated simulation platform to enable execution of long-running RTL-based simulations. FireSim is a research platform, originally designed to simulate data-center scale computing clusters at cycle-accuracy with RTL-level detail.

FireSim implements FPGA-accelerated *simulation*, as opposed to FPGA *prototyping* used in hardware-centric versions of the curricula. FPGA-accelerated *simulation* correctly models timing behavior not only of the design under test but also the I/Os and peripherals of the SoC. FPGA-accelerated simulation enables deterministic and reproducible evaluation with a realistic system environment, as opposed to FPGA prototyping where each execution is sensitive to the FPGA environment, and timing depends on FPGA peripheral device performance (e.g. DRAM performance). This determinism is important in order to obtain reproducible course results and automated grading mechanisms. Furthermore, the construction of deterministic simulation also necessitates FireSim to automate the interaction between the host machine and the FPGA, which means that FireSim users do not need to directly interact with the FPGA toolchain and the FPGA-specific configurations.

FireSim is principally designed to operate with cloud-hosted FPGAs, specifically F1 instances in the Amazon Web Services (AWS) Elastic Compute Cloud (EC2). Since the FPGAs are in the cloud, students do not interact with the complexities of any physical FPGAs. Furthermore, class size is not limited by the availability of physical FPGAs for hands-on lab exercises. AWS F1 instances enable scaling of FPGA-based lab exercises to large class sizes, since FPGA access is not limited by the number of physical FPGAs in the lab.

As an example, in the architectural component of a Hardware for Machine Learning class [29], students were instructed to complete an RTL implementation of a machine learning accelerator with a software-managed scratchpad memory, and then study the effects of tiling and scheduling of DNN models using that implementation. The study of end-to-end performance of domain specific accelerators requires application-level evaluation using non-standard hardware implementations. The implementation stages of the assignment were performed by using the Chipyard framework and the base SoC and accelerator components it provides. Since DNN models take many hours to run in software RTL simulation, FPGA-accelerated simulation was crucial for studying the tiling and scheduling aspects of the ML system. FireSim allowed students to perform fast experimental iterations with accurate performance evaluations.

During the first use of FireSim within an advanced computer architecture class in Spring 2019 [30], a memory hierarchy analysis lab used similar configurations to the ones in Figure 2 to run tests out of two benchmark suites used for evaluation of commercial processors: SPEC CPU2017 intspeed [31] and GAPBS [32]. Although FPGA-accelerated simulation is orders of magnitude faster than software RTL simulation, the programs in the SPEC CPU2017 intspeed and GAPBS still



```
class ArchClassRocketChipMemLabConfig extends Config(
  new FRFCFS16GBQuadRank                         ++
  new WithBootROM                                ++
  new WithSerial                                 ++
  new WithPerfCounters                           ++
  new WithDefaultMemModel                        ++
  new WithL1ICacheSets(64)                       ++
  new WithL1ICacheWays(1)                        ++
  new WithL1DCacheSets(64)                       ++
  new WithL1DCacheWays(1)                        ++
  new WithCacheBlockBytes(64)                    ++
  new freechips.rocketchip.system.DefaultConfig)
```

```
class ArchClassRocketChipMemLabConfig extends Config(
  new FRFCFS16GBQuadRank                         ++
  new WithBootROM                                ++
  new WithSerial                                 ++
  new WithPerfCounters                           ++
  new WithDefaultMemModel                        ++
  new WithL1ICacheSets(64)                       ++
  new WithL1ICacheWays(1)                        ++
  new WithL1DCacheSets(32)                       ++
  new WithL1DCacheWays(2)                        ++
  new WithCacheBlockBytes(64)                    ++
  new WithInclusiveCache(nBanks = 1,
                         nWays = 8,
                         capacityKB = 64)        ++
  new freechips.rocketchip.system.DefaultConfig)
```

Fig. 2. Example generator configurations used in a memory hierarchy lab exercise. Using a simple configuration-based user interface, the students are instructed to change the internal parameters of the caches such as associativity and cache size, as well as adding and removing components from the memory hierarchy. The generator emits synthesizable Verilog which is then automatically compiled and simulated using a software RTL simulator or an FPGA-accelerated simulator for the students to characterize program execution behaviors across the different memory hierarchies

take considerable time to run (50-80 minutes each on FPGA-accelerated simulation), while the relevant memory hierarchy effects could be demonstrated using shorter and more succinct programs. A retrospective comparison between the two example usages of FireSim FPGA-accelerated simulation in classes illustrates the trade-off in educational usage of this type of tool: Simulation time must be appropriate to the level of the phenomena to be demonstrated in the exercise.

IV. Vertical Curricula Span

The use of the Chipyard environment has allowed us to create an interconnecting thread between electrical engineering and computer science classes. By using this framework, students who choose to take classes that straddle both fields are able to break through the levels of abstraction and see how different components of the hardware-software stack relate to each other by using the same environment for labs and assignments across various classes. Figure 1 illustrates the multi-class flow enabled by a unified generator-based system framework. Computer-science-oriented students do not need to interact with RTL, as they only change configuration files and associated software. Digital-integrated-circuits students get a baseline design to explore VLSI optimizations. In the overlap section between standard classes, special topics classes such as hardware for machine learning or hardware for digital signal processing can utilize the unified framework to integrate unique software algorithms mapped onto custom hardware architectures in order to demonstrate the interactions between the two with complete system design flows.

During the Spring semester of 2020, the Chipyard framework was used in three concurrent classes at the same university, as demonstrated in Figure 1: A computer architecture class, a digital-integrated-circuits design class, and a special topics class about hardware for machine learning. Approximately 15% of students in the special topics class were also enrolled in the computer architecture class in the same semester, while 10% of students in the special topics class were also enrolled in the digital integrated circuits design class in the same semester. The unified assignments framework reduced the ramp-up time for students. Those who took two classes concurrently were able to amortized their infrastructure setup learning curve across classes. As all three are advanced classes, they also include a project component. The unified infrastructure enabled students to complete more comprehensive class projects, focusing on different components of the framework based on the topic of the class. Students who utilized the framework for class projects in multiple classes demonstrated class project results that included both architectural innovations and characterizations with complete software evaluation, as well as integrated circuits analysis with accurate VLSI power and area comparisons. These integrated projects effectively increase the capacity of a program to teach required system skills beyond the scope of a single class.

V. Remote Instruction

FireSim and Chipyard enable a high level of accessibility and scalability of classes, as there is no physical infrastructure requirement. Class size is essentially limited by the cost of AWS credits (for cloud-hosted FPGA-accelerated simulation) and commercial EDA license usage for the VLSI flow. The lack of physical infrastructure requirements, and in particular the usage of cloud-hosted FPGAs, present additional instructional benefits with respect to remote instruction. While the COVID-19 pandemic disrupted the Spring 2020 semester, the lab assignments in all three classes which used the joint framework continued uninterrupted despite the transition to remote instruction. However, while the possibility of large remote class exercises using these tools is within reach and attractive, actual deployment of these tools requires additional testing and infrastructure enhancements to improve their robustness and identify appropriate assignment usage models.

Furthermore, the use of cloud-hosted FPGA instances teaches students to interact with the public cloud, which is an increasingly desirable skill in the current professional computing world. Nevertheless, cloud-hosted FPGAs also come with challenges of managing variable spending among students in the class due to the dynamic nature of billing and usage. With traditional physical lab-based FPGAs and university instructional machines, students are only limited by their own time or availability of resources that have already been acquired. In contrast, when using cloud-based resources, there are virtually no availability constraints, rather only cost constraints, where cost is directly tied to usage. This raises interesting questions of responsibility and incentive models: Should resources be managed centrally for the entire class, or should students be allocated fixed amounts to “fund” their lab assignment? Should resource usage be incorporated or reflected in assignment feedback? What is the trade-off between students spending more time to further improve on their assignment, while at the same time spending more resources in order to do so? While these questions are not unique to cloud-hosted FPGAs, we believe they take on new meaning due to the traditional usage model of FPGAs in classes.

VI. Conclusion

We present our experiences with using open-source hardware generators and FPGA-accelerated simulation in advanced computer architecture classes. We show how these tools help mitigate some of the concerns of RTL-based simulation in software-centric computer architecture educational approaches, while helping contextualize the educational assignment objectives within a system-level view. We demonstrate the benefits of this approach with respect to vertical multi-class curricula across both electrical engineering and computer sciences classes and remote instruction. As the RISC-V ecosystem evolves and more commercial products appear on the market, we contend these hands-on skills can hold even more practical relevance to students and instructors alike.

VII. Acknowledgment

This work was supported in part by the NSF CCRI Award 2016662 and in part by ADEPT Lab industrial sponsors and affiliates.

References

[1] J. S. Emer and D. W. Clark, "A characterization of processor performance in the vax-11/780," in *Proceedings of the 11th Annual International Symposium on Computer Architecture*, ser. ISCA '84. New York, NY, USA: Association for Computing Machinery, 1984, p. 301–310.

[2] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Elsevier, 2011.

[3] D. L. Knox, "Integrating design and simulation into a computer architecture course," in *Proceedings of the 2nd Conference on Integrating Technology into Computer Science Education*, ser. ITiCSE '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 42–44.

[4] J. A. Nestor, "Teaching computer organization with hdls: an incremental approach," in *2005 IEEE International Conference on Microelectronic Systems Education (MSE'05)*, 2005, pp. 77–78.

[5] B. Nikolic, E. Alon, and K. Asanovic, "Generating the next wave of custom silicon," in *ESSCIRC 2018 - IEEE 44th European Solid State Circuits Conference (ESSCIRC)*, 2018, pp. 6–11.

[6] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The rocket chip generator," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[7] Y. Lee, A. Waterman, H. Cook, B. Zimmer, B. Keller, A. Puggelli, J. Kwak, R. Jevtic, S. Bailey, M. Blagojevic, P. Chiu, R. Avizienis, B. Richards, J. Bachrach, D. Patterson, E. Alon, B. Nikolić, and K. Asanović, "An agile approach to building RISC-V microprocessors," *IEEE Micro*, vol. 36, no. 2, pp. 8–20, Mar 2016.

[8] O. Shacham, O. Azizi, M. Wachs, W. Qadeer, Z. Asgar, K. Kelley, J. P. Stevenson, S. Richardson, M. Horowitz, B. Lee, A. Solomatnikov, and A. Firoozshahian, "Rethinking digital design: Why design must change," *IEEE Micro*, vol. 30, no. 6, pp. 9–24, 2010.

[9] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14.

[10] S. N. Soares and F. R. Wagner, "T&D-bench—innovative combined support for education and research in computer architecture and embedded systems," *IEEE Transactions on Education*, vol. 54, no. 4, pp. 675–682, 2011.

[11] S. N. Soares and F. Wagner, "Design space exploration of embedded processors in computer architecture education using T&D-bench," in *Proceedings. Frontiers in Education. 36th Annual Conference*, 2006, pp. 19–24.

[12] R. Agrawal, S. Bandara, A. Ehret, M. Isakov, M. Mark, and M. A. Kinsy, "The BRISC-V platform: A practical teaching approach for computer architecture," in *Proceedings of the Workshop on Computer Architecture Education*, ser. WCAE'19. New York, NY, USA: Association for Computing Machinery, 2019.

[13] S. Wallentowitz, "RISC-V in practical education of computer architecture," in *Proceeding of the RISC-V Summit 2019*, 2019.

[14] RISC-V International, "RISC-V educational materials," https://riscv.org/educational-materials/, 2020, accessed: 2020-05-06.

[15] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[16] P. Davide Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, and L. Benini, "Slow and steady wins the race? a comparison of ultra-low-power RISC-V cores for internet-of-things applications," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2017, pp. 1–8.

[17] C. Celio, D. A. Patterson, and K. Asanovic, "The berkeley out-of-order machine (BOOM): An industry-competitive, synthesizable, parameterized RISC-V processor," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167*, 2015.

[18] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit RISC-V core in 22-nm FDSOI technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, Nov 2019.

[19] Z. Bandic and R. Golla, "SweRV cores roadmap," in *Proceeding of the RISC-V Summit 2019*, 2019.

[20] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanović, and B. Nikolić, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[21] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a scala embedded language," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 1216–1225.

[22] Y. Lee, C. Schmidt, A. Ou, A. Waterman, and K. Asanović, "The hwacha vector-fetch architecture manual, version 3.8. 1," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-262*, 2015.

[23] A. Akram and L. Sawalha, "A survey of computer architecture simulation techniques and tools," *IEEE Access*, vol. 7, pp. 78 120–78 145, 2019.

[24] A. Akram and L. Sawalha, "x86 computer architecture simulators: A comparative study," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, 2016, pp. 638–645.

[25] B. Nikolic, Z. Radivojevic, J. Djordjevic, and V. Milutinovic, "A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization," *IEEE Transactions on Education*, vol. 52, no. 4, p. 449–458, 11 2009.

[26] R. Hasan and S. Mahmood, "Survey and evaluation of simulators suitable for teaching for computer architecture and organization supporting undergraduate students at sir syed university of engineering technology," in *Proceedings of 2012 UKACC International Conference on Control*, 2012, pp. 1043–1045.

[27] P. W. C. Prasad, A. Alsadoon, A. Beg, and A. Chan, "Using simulators for teaching computer organization and architecture," *Computer Applications in Engineering Education*, vol. 24, no. 2, pp. 215–224, 2016.

[28] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra, Q. Huang, K. Kovacs, B. Nikolic, R. Katz, J. Bachrach, and K. Asanović, "Firesim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18. IEEE Press, 2018, p. 29–42.

[29] Y. S. Shao, A. Amid, and H. Genc, "EE290-2, spring 2020, hardware for machine learning," https://inst.eecs.berkeley.edu/~ee290-2/sp20/, EECS Department, University of California, Berkeley. Accessed: 2021-02-10.

[30] AWS Public Sector Blog, "University of california, berkeley uses AWS educate and amazon FPGA instances in undergraduate computer architecture course," https://aws.amazon.com/blogs/publicsector/university-of-berkeley-uses-aws-educate-for-amazon-fpga-accelerator-development-and-deployment-in-the-cloud/, accessed: 2020-05-06.

[31] Standard Performance Evaluation Corporation, "SPEC CPU 2017," https://www.spec.org/cpu2017/, 2017, accessed: 2020-05-06.

[32] S. Beamer, K. Asanović, and D. Patterson, "The GAP benchmark suite," *arXiv preprint arXiv:1508.03619*, 2015.