Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs

Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, Paul Rigge, Colin Schmidt, John Wright, Jerry Zhao, Yakun Sophia Shao, Krste Asanović, and Borivoje Nikolić
University of California, Berkeley

***Abstract*—Continued improvement in computing efficiency requires functional specialization of hardware designs. Agile hardware design methodologies have been proposed to alleviate the increased design costs of custom silicon architectures, but their practice thus far has been accompanied with challenges in integration and validation of complex systems-on-a-chip (SoCs). We present the Chipyard framework, an integrated SoC design, simulation, and implementation environment for specialized compute systems. Chipyard includes configurable, composable, open-source, generator-based IP blocks that can be used across multiple stages of the hardware development flow while maintaining design intent and integration consistency. Through cloud-hosted FPGA accelerated simulation and rapid ASIC implementation, Chipyard enables continuous validation of physically realizable customized systems.**

Digital Object Identifier 10.1109/MM.2020.2996616
Date of publication 22 May 2020; date of current version 30 June 2020.

In the face of the slowdown in technology scaling, sustaining improvements in system capability requires a greater use of domain-specific

Published by the IEEE Computer Society

architectures. This era of specialization is being seen as a new golden age of computer architecture,[1] but creates the challenge of escalating development costs. Differentiated architectures require productive digital system design methods for architectural exploration, system integration, verification, validation, and physical design.

To reduce development costs, a generator-based agile design process for hardware has been proposed and demonstrated through a series of RISC-V microprocessor chips developed with small teams[2] and made available as a now widely used open-source codebase.[3] Designs captured as generators enable reuse through rich parameterization and incremental extension. Past efforts have focused on the module generator development and an implementation of relatively small, homogeneous processor architectures with uniform interconnect and core-level configurability. Enabling the development of complex heterogeneous systems-on-a-chip (SoCs) requires greater levels of coordination and synchronization between many disparate open-source designs and development tools, motivating the creation of a new unified open-source SoC design framework, which we call Chipyard.*

Chipyard provides a framework to bring together a collection of independently developed open-source tools and RTL generators, allowing development of heterogeneous SoCs through integrated design, simulation, and implementation environments. Chipyard helps alleviate many of the challenges that exist when using independent and uncoordinated open-source tools and designs, as often experienced in concurrent and nonuniform feature design iterations, typical in the agile design process.

Chipyard provides a framework to bring together a collection of independently developed open-source tools and RTL generators, allowing development of heterogeneous SoCs through integrated design, simulation, and implementation environments.

AGILE SoC DEVELOPMENT

A number of RISC-V test chips have been designed by small groups of students at the University of California, Berkeley over the past eight years. These designs have been based on the open-source Rocket Chip SoC generator[3] that includes Rocket, an in-order RISC-V core, and supports coherent caches and standard interconnects via the TileLink protocol. Customizations to test chips based on the Rocket Chip generator have been performed by adding multiple generations of vector processors, peripheral devices, and by replacing the standard in-order core with an out-of-order core. The design process outlined in[2] was gradually enhanced on the physical design front to support much larger silicon dies and many more placeable instances in the SoC, resulting in increasingly complex test chips (see Figure 1). The most recent test chips in this series are representative of modern SoCs composed of a diverse set of IP blocks and include multiple cores, accelerators, and complete analog subsystems, including high-speed serial links (SerDes), analog-to-digital converters (ADCs), and phased-locked loops (PLLs).

Verification and validation are major components of the SoC design cycle. By relying on many previously verified open-source components such as the Rocket core, and incrementally extending SoCs based on the Rocket Chip generator, it was often possible to sidestep rigorous verification and validation steps in this series of test chips. Each test chip has a focus on testing either a particular design feature of a module or a component of the design methodology, and therefore, the verification of a test chip was, respectively, limited to a small set of functionality.

This reliance on open-source generators shifts the verification challenge from block-level testing to system-integration verification and validation. Rich module-level configurability in the generator-based approach allows for quick, iterative design customization, enabled by expansive configuration and parameter systems. While these parameter systems enable quick iterations across many design points,

*https://github.com/ucb-bar/chipyard/

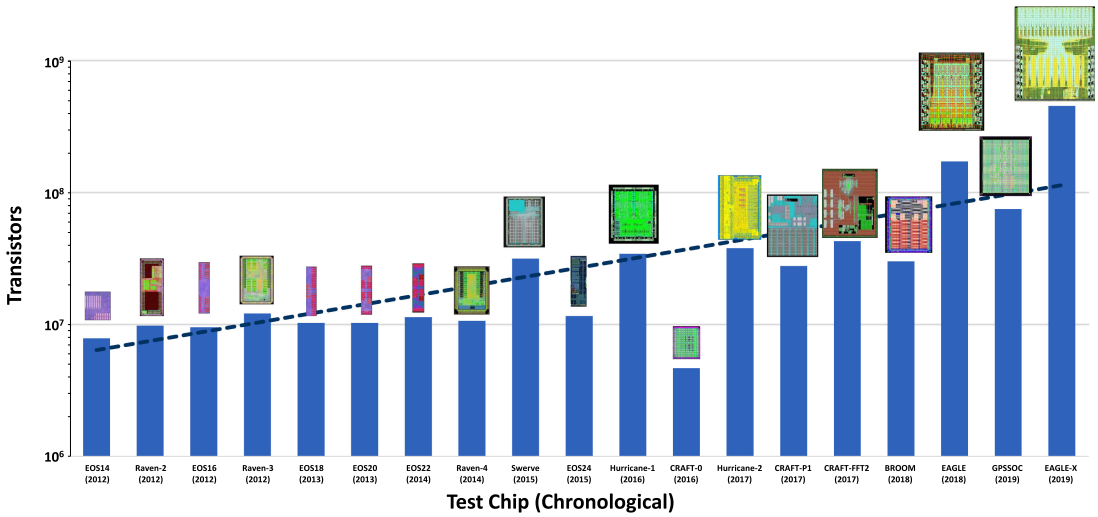

Figure 1. Increasing complexity of custom RISC-V SoC test chips built at Berkeley using the Rocket Chip SoC generator since 2012.

their flexible nature makes them prone to misconfiguration, underscoring the need for continuous full-system integrated validation and verification. Rich module-level parameterization makes it possible to continuously update the SoC design in an agile way, but does not inherently aid the design verification or presilicon validation of the chip performance. In addition, issues relating to on- and off-chip interfaces, clock domain crossings, third-party IP integration, and power management are all vulnerable to insufficient full-system verification coverage. Our previous chips have encountered both verification and validation gaps of this type: one chip's cache capacity was inadvertently reduced to half of its desired size, when the configuration was changed late in the design process to meet physical design constraints.

Although design and verification executed by a small team in an agile manner has a high appeal for small companies and industrial and academic research labs, our experiences with test chips developed through an agile process also identify some challenges in execution. The increasing complexity of test chips makes it difficult to parallelize and distribute effort among the small number of designers, as architecture definition, RTL implementation, physical design, verification, and validation all take varying amounts of time, which is difficult to account for. Furthermore, it is difficult to maintain institutional memory of good design practices, especially in academic environments and when working with complex physical design tool flows in deeply scaled technologies.

Finally, transitioning between process technologies is a significant undertaking in both system and test chip design. The test chips in this retrospective have been designed in several different process technologies, including IBM 45nm SOI, ST 28nm FD-SOI, TSMC 28nm, TSMC 16nm FFC, and Intel 22nm FFL. The generator-based approach simplifies process technology transition since it allows for adjustment of design parameters through high-level descriptions. However, this flexibility does not propagate across the abstraction layers to the physical design process. Each custom test chip requires significant manual effort in mapping RTL abstractions to process-specific components (such as memory and register-file macros), as well as meeting the design rules. When generator parameters change between design iterations, a new rigid physical design script is often created.

While some of the issues described in this section are not specific to agile hardware design methodologies, their visibility increases as the cost and complexity of other parts of the flow diminish. Nevertheless, a common theme throughout our observations relates to integration and partitioning—on the chip level, and on the methodology and flow level.

CHIPYARD FRAMEWORK

Chipyard provides a unified framework and work flow for agile SoC development. Multiple separately developed and highly parameterized IP blocks can be configured and interconnected to form a complete SoC design. The SoC design can be verified and validated through both FPGA-accelerated and standard software simulations, then pushed through portable VLSI design flows to obtain tapeout-ready GDSII data for various target technologies. Chipyard also provides a workload management system to generate software workloads to exercise the design.

Chipyard Front-End RTL Generators

The front end of the Chipyard framework is based on the Rocket Chip SoC generator.[2,3] Chipyard inherits Rocket Chip's Chisel-based parameterized hardware generator methodology,[3] including a Scala-based parameter-negotiation framework, Diplomacy,[4] that negotiates mutually compatible parameterizations and interconnections across all IP blocks in a design. A unified top-level SoC generator enables the generation of heterogeneous systems based on parameterized configurations. Chipyard also allows IP blocks written in other hardware languages, e.g., Verilog, to be included via a Chisel wrapper.

Chipyard adds a large corpus of open-source IP generators to the existing Rocket Chip base library, allowing for the construction of modern digital SoCs. These include the Berkeley Out-of-Order Machine (BOOM) generator,[5] the Ariane core,[6] the Hwacha vector-unit generator,[7] digital signal processing modules, domain-specific accelerators (including machine learning and cryptography), memory systems, and peripherals. The majority of these generators have silicon-proven instances through the aforementioned series of test chips.

While some commercial IP vendors have large collections of proprietary configurable IP for certain portions of an SoC, Chipyard provides a publicly extensible open-source alternative for complete SoCs to support continuing research and development of specialized state-of-the-art SoCs.

Other open-source SoC design frameworks focus on tile-granularity customization in many-core architectures,[8,9] or rely on rigid subsystems and proprietary IP.[10] The generator approach used in Chipyard does not rely solely on static interfaces for integration of IP blocks, but allows for dynamic customization of encodings, memory maps, and buses during the hardware generation stage, enabling custom components to be created and integrated at various levels, including in the memory mapped IO (MMIO) periphery, as tightly integrated accelerators, and as heterogeneous cores and controllers. For example, through its fine-grained intracore parameter system, the Rocket core can be used for different purposes in an SoC. Specifically, in several of our recent test chips, the application cores consist of fully Linux-capable Rocket cores supporting RV64GC with floating-point units and virtual memory support, whereas the controller cores (e.g., a power-management unit) are a Rocket core supporting only RV64IMAC, with significantly smaller branch prediction and cache resources and no virtual memory. In essence, a significant portion of the microcontrollers running firmware on the SoC can be implemented using variants of Rocket or BOOM cores. This common core configuration interface for all software-managed controllers within the system improves designer productivity, compared to alternative SoC development frameworks that enable drop-in replacement core options or only coarse-grained sizing. Similarly, machine-learning accelerators have been integrated with Rocket and other core generators as tightly integrated accelerators,[11] as well as in the form of MMIO periphery accelerators,[12] demonstrating the various levels of possible customization in the Chipyard framework. Figure 2 demonstrates the various degrees of customization on the core, tile, and SoC levels, enabled by the Chipyard framework.

FIRRTL Intermediate Representation

The Chipyard framework currently integrates tools to address the three main activities within the custom SoC design cycle: front-end RTL design, system validation/verification, and back-end chip physical design. These different stages of the SoC design flow require different levels of description of the design. For example, while front-end RTL descriptions usually use abstract notions of memory and I/O, back-end RTL

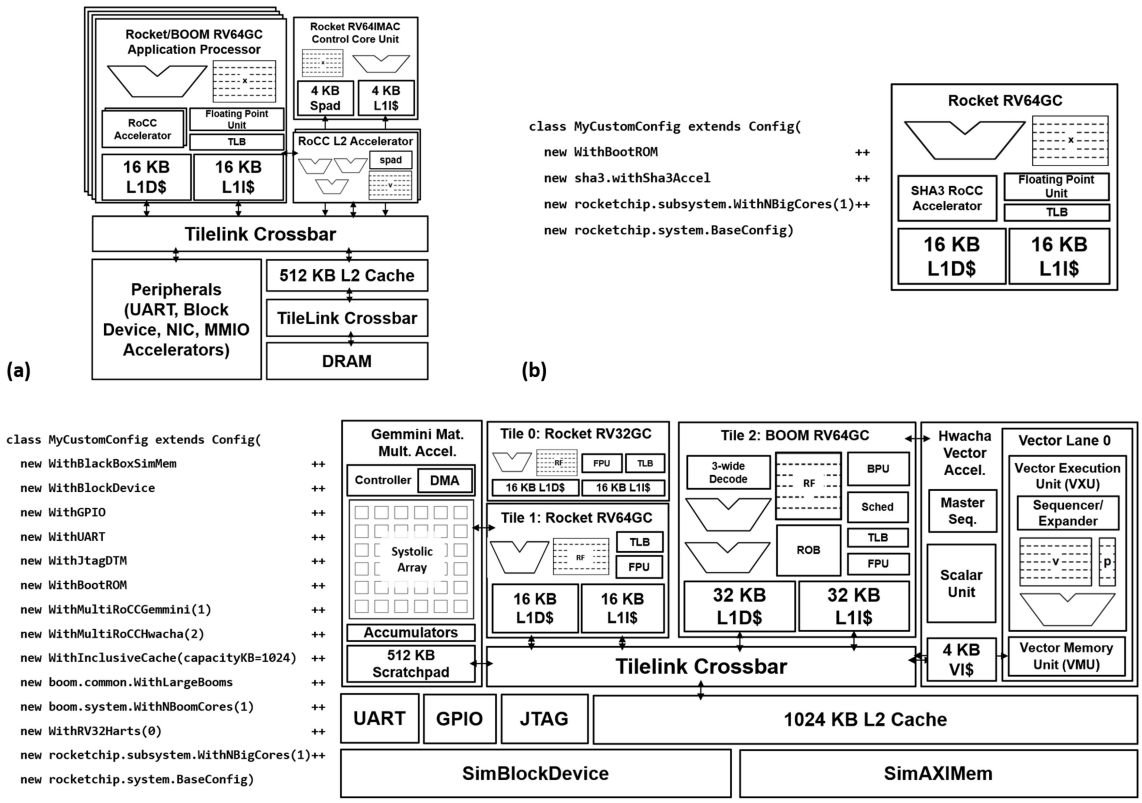

Figure 2. SoC customization in Chipyard using the Rocket Chip generator configuration system. (a) A typical Chipyard SoC can include multiple heterogeneous application cores, as well as multiple configurations of BOOM or Rocket cores that can serve various purposes within the SoC, such as a controller unit. Customization can be performed across various levels of the SoC, including Rocket Custom Coprocessor (RoCC) accelerators, MMIO accelerators, and peripherals. (b) An example configuration of a simple design, consisting of a single Rocket in-order processor core and a custom educational SHA3 tightly integrated accelerator. (c) An example configuration of a more complex SoC, consisting of a multiple cores of various capabilities (RV32 Rocket in-order control core, RV64 Rocket in-order application core, RV64GC BOOM 3-wide out-of-order application core), custom accelerators attached to each core, a standard set of peripherals, and a deeper memory hierarchy with an L2 cache. This example also includes a simulated block device and simulated backing DRAM memory.

requires more precise descriptions mapped to the underlying process technology. Similarly, FPGA emulation will require the digital design to interact with FPGA-specific interfaces, periphery, and internal components, which requires particular collateral. Cosimulation also requires additional hardware clock gating to control simulation progress.

Chipyard elaborates the front-end RTL design into a FIRRTL[13] intermediate representation. Custom FIRRTL transformations convert the generated FIRRTL design to drive the different flows used at different stages of the design cycle. Using FIRRTL transformations to enable multiple concurrent design flows from the same shared code repository and source RTL helps to reduce and amortize the environment setup costs incurred with frequent iterations between development stages, as is needed for an agile methodology. This approach is demonstrated in Figure 3.

While Chisel is the primary language for design entry in Chipyard using the FIRRTL compiler, a FIRRTL-based flow can integrate Verilog IP through either "Blackbox" IP integration or Verilog-to-FIRRTL support by certain

Verilog elaboration tools.[14] Furthermore, while the Verilog outputs of various stages of a FIRRTL-based flow can be integrated into standard dynamic verification environments or compared using logical equivalence checking, tools for both simulation and temporal property checking of "FIRRTL-native" circuits are openly available.[15]

Verilog or SystemVerilog-based design frameworks[8,10] must rely on design-specific custom scripts or interface adjustments when transitioning between emulation, simulation and physical design. In contrast to alternative hardware package management systems[16] or integration standards like IP-XACT, which focus on metadata associated with particular IP components to target different electronic design automation (EDA) flows, the FIRRTL transformations used by Chipyard can be applied to any Chisel design integrated with the Chipyard framework.

Software RTL Simulation

Software-based RTL simulators are a critical tool in most phases of the design process. Compiling a software simulator of a top-level design, including various IP components, peripheral and memory models, and an external test harness can be a time-consuming engineering task. Chipyard provides build flows for both the open-source Verilator simulator and proprietary commercial simulators. Open-source RTL simulators such as Verilator are also used in industry[17] to provide efficient and cost-effective digital design verification. Chipyard provides Makefile wrappers for direct generation of a simulation executable, which simulates tethered designs with emulated peripherals. The Makefile wrappers generate the top-level design and matching test harnesses based on the SoC configuration. Tethered designs use a host to send transactions that bring up the simulated SoC and load programs. These software RTL simulation wrappers enable quick design cycles and execution of RISC-V binaries in simulation. While tethered designs are the default form to generate software RTL simulations in Chipyard, Chipyard also supports un-tethered SoC configurations in which the SoC can boot standalone using a boot ROM.

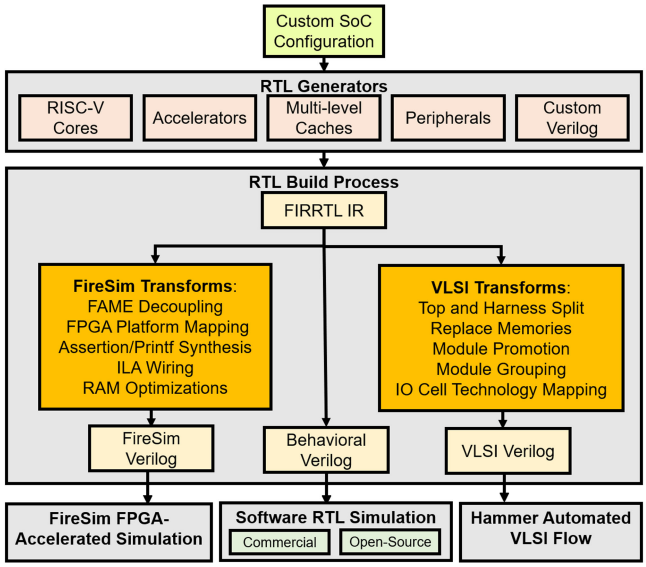

Figure 3. Multiple disparate design flows supported by the Chipyard framework through generators and transformations. A series of FIRRTL transformations consumes generators with a custom configuration, and outputs appropriate Verilog and associated collateral for different design stage platforms.

FPGA-Accelerated Simulation With FireSim

For full-system validation and evaluation, the Chipyard framework harnesses the FireSim[18] open-source FPGA-accelerated simulation platform using the AWS EC2 public cloud. In contrast with FPGA *prototyping*, FPGA-accelerated *simulation* correctly models timing behavior of not only 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).

Originally developed as a platform to enable scale-out simulation for datacenter architecture research on hundreds of cloud FPGAs, FireSim necessarily automates the infrastructure management and simulation mapping necessary to automatically run high-performance simulations. As part of the agile chip-design stack, this automation and integration reduces the level of expertise required to harness cloud FPGAs for emulation purposes and thus increases the accessibility of high-performance full-system simulation to a broad spectrum of designers.

FireSim has been useful in presilicon verification, validation, and software development. From the perspective of small agile teams with limited resources, FireSim provides many of the features available in costly commercial emulation platforms. In contrast with prior FPGA-accelerated simulation tools, the accessibility of FireSim through FPGA instances on the AWS public cloud, together with the automation of host-target interfaces with the FPGA, have made FireSim a popular tool within Berkeley and other academic hardware development users, as well as emerging startup companies.

FireSim enables codevelopment of software and hardware simultaneously, allowing for quick software adjustment turnarounds based on hardware modifications. Furthermore, FireSim plays a major role in the performance and functional validation of processors, since it enables the identification of bugs deep into simulation execution time thanks to FPGA acceleration with appropriate peripheral modeling. Unlike many other open-source hardware development platforms with FPGA support, FireSim's focus on simulation and emulation as opposed to prototyping enables true presilicon performance evaluation and validation in a full-system context within the Chipyard framework. While maintaining its standalone operation as an architectural research platform, FireSim was transformed into a library that is integrated into the broader Chipyard framework. As such, FireSim can now consume design configurations composed within the Chipyard framework, and transform them into FPGA-accelerated simulations. Furthermore, the FireSim Golden Gate compiler has been integrated into the Chipyard framework, so it can now consume arbitrary FIRRTL as its input, as well as external Verilog components necessary for broader system integration.

Back-End Physical Design With Hammer

For back-end physical design, Chipyard includes a modular VLSI flow named Hammer.[19] The Hammer VLSI flow provides an abstraction layer above process-technology- and EDA-tool-specific concerns, with the goal of increasing reuse and modularity of vendor-specific components of the physical design flow. To this end, the Hammer VLSI flow utilizes separate vendor-specific process technology plug-ins and EDA-tool-specific plug-ins, which implement abstracted software APIs to generate design flow collateral like Tcl scripts, clock constraints, and power specifications based on higher level design inputs. For example, Hammer will emit process- and vendor-specific macro placement, obstruction, and power-strap placement commands from a high-level process- and vendor-agnostic description of the design. This separation of abstraction layers between design, process technology, and EDA tool vendor enables faster adoption of open-source components.

The Hammer flow aspires to support open-source tools in conjunction with commercial and proprietary tools using common levels of abstraction. As such, while the first Hammer-based designs were implemented using proprietary process technologies, a plug-in for the ASAP7[20] open-source predictive process design kit (PDK) was created in only a few weeks and is now included in the core Hammer repository. With this, small teams and academic users can prototype design flows and experiment with RTL designs using predictive or simple physical design kits while being able to reuse similar Hammer descriptions for chip fabrication using advanced process nodes.

Hammer was designed to support hierarchical physical design flows. Hierarchical physical design flows are of particular importance in highly complex custom SoCs, composed of multiple specialized blocks with a variety of physical design constraints. Decomposing a design into these smaller hierarchical components not only improves the quality of results emitted by EDA tools, but it also allows the distribution of physical design tasks among multiple hardware developers, which is important for agile design. FIRRTL-based grouping and flattening transformations in Chipyard further assist the hierarchical physical design flow in Hammer by enabling users to specify one logical hierarchy in the source RTL while choosing a different hierarchy for physical boundaries through automated transformations.

Workload Management With FireMarshal

Finally, no hardware development environment is complete without relevant software interfaces. In order to support continuous and concurrent full-system validation, Chipyard

enables shared software development across the feature design stages through the FireMarshal software workload generation tool. Shared software development is especially important in integrated validation of specialized and custom designs. FireMarshal provides a standard and version-controlled format for software workload descriptions and automates the generation of these workloads for various simulation targets. Software developers can begin work as soon as a functional model is available (e.g., in the Spike RISC-V ISA simulator or the QEMU emulator). Those workloads can then be used without any modification on RTL simulations and FireSim simulations. In this way, the complex task of software development and porting (particularly for Linux-based workloads) can be reused by anyone on the design team without requiring special expertise. FireMarshal includes several examples and templates for Linux-based workloads, enabling fast ramp-up of software development across the various simulation and emulation targets using preset Linux kernel configurations and base distribution images with matching drivers. Chipyard provides a versioned set of standard RISC-V software development tools (e.g., GNU toolchain, QEMU, Spike), as well as a set of equivalent nonstandard RISC-V development tools for nonstandard extensions of custom IP blocks. The two software development tool sets can be used interchangeably in the framework. The software development structure in Chipyard is illustrated in Figure 4.

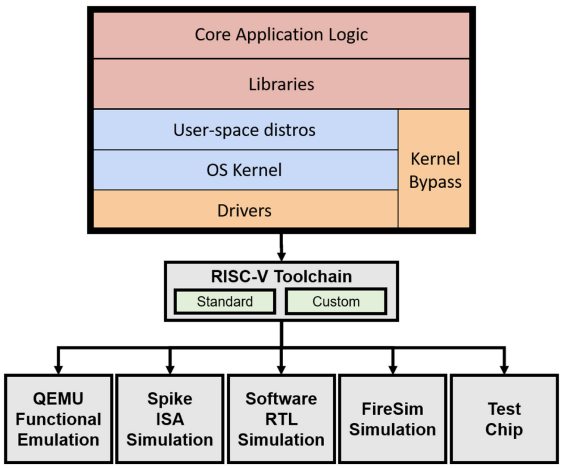

Figure 4. Shared software development in Chipyard using FireMarshal. Designers can use the standard RISC-V software toolchains, or custom software toolchains. While the core application logic and libraries are consistent with the SoC design, kernel and driver configuration may change based on the target platforms or tethered systems. FireMarshal automates workload generation to enable targeting multiple platforms using a single description.

Concurrent Agile Hardware Development

Together, the components of the Chipyard framework enable continuously integrated development of custom and specialized SoCs, taking advantages of the accessibility of open-source tooling. The Chipyard framework enables further adoption of agile hardware development methodologies, by lowering the barrier to entry and providing support and integration of development environments for different stages of the design process.

Chipyard is designed for concurrent development of multiple custom SoC features by multiple members of a small team, with each feature going through iterative design cycles, which include modeling, simulation, system integration, FPGA emulation and validation, and physical design. The structure of Chipyard enables independent development of the various IP modules and/or configured SoC instances, with continuous integration using configured workloads for system validation and verification.

OPEN-SOURCE SoC DEVELOPMENT

Chipyard aims to be a fully open-source SoC development framework. However, while open-source tools and projects within the digital-logic abstraction layer of the hardware design stack have been gaining traction and trust in research and industrial communities, design aspects that are closer to underlying technologies such as analog and mixed-signal (AMS) modules and the manufacturing interfaces still need to identify appropriate open-source development and integration models.

In the analog domain, new generator-based approaches such as the Berkeley Analog Generator (BAG)[21] are envisioning a portable and process-technology agnostic generator-based approach to AMS design. However, this approach requires tight integration with digital SoC components. Future integration of analog

generators and their simulation environments, such as BAG, into the Chipyard framework will help support common interfaces and integration between analog and digital components through integrated tools and automation of design collateral such as appropriate generated behavioral models of analog blocks, as well as matching physical design constraints.

In the digital domain, there is a large interdependence of standard commercial EDA tool stacks with tightly controlled NDA-only physical design kits due to the complexities of submicron process technologies. This issue has been identified in the past,[22] and open-source hardware projects choose to address this challenge in various ways: some publish "patches" to the common flow scripts provided by EDA vendors,[8] or partially associate hardware design template implementations with particular process technologies,[23] but are largely obscured elsewhere. The approach used in the Hammer VLSI flow within Chipyard is a "plug-in" model. These plug-ins provide a mapping between the Hammer vendor-agnostic level of abstraction, to the proprietary vendor-specific APIs. Open-source physical design initiatives such as the OpenROAD project[24] are making encouraging progress toward addressing this challenge.

In this article, we presented the Chipyard framework that was developed to provide an integrated design, simulation, and implementation environment to support the growing complexity and differentiation of custom SoCs.

CONCLUSION

In this article, we presented the Chipyard framework that was developed to provide an integrated design, simulation, and implementation environment to support the growing complexity and differentiation of custom SoCs. Through integration with the Rocket Chip generator ecosystem, Chipyard provides a large number of easily composable and extensible open-source digital IP blocks. The additional integration of multiple simulation and implementation tools enables continuous and simultaneous development for higher-quality verification, validation, and system integration. Future integration with analog generator frameworks will open the door to breaking the digital–analog divide, and the enabling of complete open-source custom SoC development solutions.

ACKNOWLEDGMENTS

This work was supported in part by the Defense Advanced Research Projects Agency through the Circuit Realization at Faster Timescales Program under Grant HR0011-16-C0052; in part by the Advanced Research Projects Agency-Energy, U.S. Department of Energy, under Award DE-AR0000849; and in part by ADEPT Lab industrial sponsors and affiliates. The authors would like to thank Intel, STMicroelectronics, and TSMC for donating prototypes fabrication. The views and opinions of authors expressed herein do not necessarily state or reflect those of the U.S. Government or any agency thereof.

REFERENCES

1. J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, no. 2, pp. 48–60, Jan. 2019.
2. Y. Lee *et al.*, "An agile approach to building RISC-V microprocessors," *IEEE Micro*, vol. 36, no. 2, pp. 8–20, Mar. 2016.
3. K. Asanović *et al.*, "The Rocket Chip generator," Elect. Eng. Comput. Sci. Dept., Univ. California–Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2016-17, Apr. 2016.
4. H. Cook, W. Terpstra, and Y. Lee, "Diplomatic design patterns: A TileLink case study," in *Proc. 1st Workshop Comput. Archit. Res. RISC-V*, 2017.
5. C. Celio, D. A. Patterson, and K. Asanović, "The Berkeley out-of-order machine (BOOM): An industry-competitive, synthesizable, parameterized RISC-V processor," Elect. Eng. Comput. Sci. Dept., Univ. California–Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2015-167, 2015.
6. 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 Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 11, pp. 2629–2640, Nov. 2019.

7. Y. Lee, C. Schmidt, A. Ou, A. Waterman, and K. Asanović, "The Hwacha vector-fetch architecture manual, version 3.8. 1," Elect. Eng. Comput. Sci. Dept., Univ. California–Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2015-262, 2015.
8. J. Balkind *et al.*, "OpenPiton: An open source manycore research framework," in *Proc. 21st Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2016, pp. 217–232.
9. L. P. Carloni, "Invited—The case for embedded scalable platforms," in *Proc. 53rd Annu. Design Autom. Conf.*, 2016, pp. 17:1–17:6.
10. P. Whatmough, M. Donato, G. Ko, S. K. Lee, D. Brooks, and G.-Y. Wei, "CHIPKIT: An agile, reusable open-source framework for rapid test chip development," *IEEE Micro*, vol. 44, no. 4, Jul./Aug. 2020. doi: 10.1109/MM.2020.2995809.
11. H. Genc *et al.*, "Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures," 2019, *arXiv:1911.09925*.
12. F. Farshchi, Q. Huang, and H. Yun, "Integrating NVIDIA deep learning accelerator (NVDLA) with RISC-V SoC on FireSim," in *Proc. 2nd Workshop Energy Efficient Mach. Learn. Cogn. Comput. Embedded Appl. HPCA*, 2019.
13. A. Izraelevitz *et al.*, "Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations," in *Proc. 36th Int. Conf. Comput.-Aided Design*, 2017, pp. 209–216.
14. C. Wolf, "Yosys open synthesis suite—write_firrtl—Write design to a FIRRTL file," 2018. [Online]. Available: http://www.clifford.at/yosys/cmd_write_firrtl.html
15. A. Magyar, D. Biancolin, J. Koenig, S. Seshia, J. Bachrach, and K. Asanovic, "Golden gate: Bridging the resource-efficiency gap between ASICs and FPGA prototypes," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2019, pp. 1–8.
16. O. Kindgren, "Invited paper: A scalable approach to IP management with FuseSoC," in *Proc. Workshop Open Source Design Autom.*, 2019.
17. D. Das Sarma and G. Venkataramanan, "Compute and redundancy solution for Tesla's full self driving computer," Hot Chips 31: Stanford Memorial Auditorium, Stanford, CA, USA, Aug. 18–20, 2019. [Online]. Available: https://www.hotchips.org/hc31/HC31_2.3_Tesla_Hotchips_ppt_Final_0817.pdf
18. S. Karandikar *et al.*, "FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit.*, Jun. 2018, pp. 29–42.
19. E. Wang *et al.*, "A methodology for reusable physical design," in *Proc. 21st Int. Symp. Qual. Electron. Design*, Mar. 2020.
20. L. T. Clark *et al.*, "ASAP7: A 7-nm FinFET predictive process design kit," *Microelectron. J.*, vol. 53, pp. 105–115, 2016.
21. E. Chang *et al.*, "BAG2: A process-portable framework for generator-based AMS circuit design," in *Proc. IEEE Custom Integr. Circuits Conf.*, Apr. 2018, pp. 1–8.
22. G. Gupta, T. Nowatzki, V. Gangadhar, and K. Sankaralingam, "Kickstarting semiconductor innovation with open source hardware," *Computer*, vol. 50, no. 6, pp. 50–59, 2017.
23. M. B. Taylor, "Basejump STL: Systemverilog needs a standard template library for hardware design," in *Proc. 55th Annu. Design Autom. Conf.*, 2018, pp. 73:1–73:6.
24. T. Ajayi *et al.*, "Toward an open-source digital flow: First learnings from the OpenROAD project," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 76:1–76:4.

Alon Amid is currently working toward the Ph.D. degree at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. His research interests include parallel and distributed computing, energy-efficient processors and architectures, and hardware–software codesign. Amid received the B.Sc. degree in electrical engineering from Technion—Israel Institute of Technology, and the M.S. degree from the University of California, Berkeley. Contact him at alonamid@berkeley.edu.

David Biancolin is currently working toward the Ph.D. degree at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. Biancolin received the BASc degree in engineering science from the University of Toronto. Contact him at biancolin@berkeley.edu.

Abraham Gonzalez is currently working toward the Ph.D. degree at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. His research interests include warehouse-scale computing, high-performance microarchitectures, and computer architecture tooling. Gonzalez received the B.S. degree in electrical and computer engineering from the University of Texas at Austin in 2018. Contact him at abe.gonzalez@berkeley.edu.

Daniel Grubb is currently working toward the Ph.D. degree at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. His current research focuses on navigation systems for autonomous robots and agile physical design methodologies. Grubb received the B.S. degree in electrical engineering and computer sciences from the University of California, Berkeley. Contact him at dpgrubb@eecs.berkeley.edu.

Sagar Karandikar is currently working toward the Ph.D. degree at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. His current research focuses on exploring hardware–software codesign in warehouse-scale machines. Karandikar received the B.S. and M.S. degrees in electrical engineering and computer sciences from the University of California, Berkeley. He is a member of the Association for Computing Machinery and IEEE. Contact him at sagark@eecs.berkeley.edu.

Harrison Liew is currently working toward the Ph.D. degree at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. His current research is at the intersection of the BWRC and ADEPT labs, focusing on signal processing generators for multiuser massive MIMO base stations and its implementation in integrated circuits using agile physical design frameworks and methodologies. Liew received the B.S. and M.S. degrees in electrical engineering from Columbia University in 2016. Contact him at harrisonliew@berkeley.edu.

Albert Magyar is currently working toward the Ph.D. degree at the ADEPT Lab, University of California, Berkeley, advised by K. Asanovic and J. Bachrach. His research interests include increasing productivity of RTL design and improving the usability of FPGA simulation. Contact him at albert.magyar@berkeley.edu.

Howard Mao is currently working toward the Ph.D. degree at the University of California, Berkeley. He is a member of the ADEPT Lab and is interested in designing microarchitectures for datacenter systems. Contact him at zhemao@eecs.berkeley.edu.

Albert Ou is currently working toward the Ph.D. degree at the University of California, Berkeley. His research interests include energy-efficient vector processor architectures, VLSI implementation, and data-parallel programming models. Ou received the B.S. and M.S degrees in electrical engineering and computer sciences from the University of California, Berkeley, in 2014 and 2015, respectively. He is a student member of IEEE. Contact him at aou@eecs.berkeley.edu.

Nathan Pemberton is currently working toward the Ph.D. degree at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. His current research focuses on computer architecture and operation systems for warehouse-scale computing. Pemberton received the B.S. degree in computer engineering from the University of California Santa Cruz and the M.S. degree in computer science from the University of California, Berkeley. He is a member of the Association for Computing Machinery. Contact him at nathanp@berkeley.edu.

Paul Rigge is currently working toward the Ph.D. degree at the University of California, Berkeley. His current research focuses on agile hardware methodologies for wireless systems. Rigge received the B.S. degree in electrical engineering and computer science from the University of Michigan in 2012. Contact him at rigge@berkeley.edu.

Colin Schmidt is currently working toward the Ph.D. degree at the University of California, Berkeley, where he works on architecting, implementing, and building software for vector accelerators. Schmidt received the B.S. degree in electrical and computer engineering and computer science from Cornell University. He is a student member of the Association for Computing Machinery. Contact him at colins@berkeley.edu.

John Wright is currently working toward the Ph.D. degree at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. Wright received the B.S. degree from the University of Kentucky in 2011 and the M.S. degree from the University of California, Berkeley, in 2017. He was previously with Cypress Semiconductor, San Jose, CA. Contact him at johnwright@berkeley.edu.

Jerry Zhao is currently working toward the Ph.D. degree at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. His current research focuses on microarchitecture of high-performance processors. Zhao received the B.S. degree in electrical engineering and computer science from the University of California, Berkeley. He is a student member of the Association for Computing Machinery and IEEE. Contact him at jzh@berkeley.edu.

Yakun Sophia Shao is currently an Assistant Professor at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. Shao received the Ph.D. degree in computer science from Harvard University. Contact her at ysshao@berkeley.edu.

Krste Asanović is currently a Professor at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. Asanovic received the Ph.D. degree in computer science from the University of California, Berkeley. He is a Fellow of IEEE and the Association for Computing Machinery. Contact him at krste@berkeley.edu.

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