An Agile Approach to Building RISC-V Microprocessors

Yunsup Lee, Andrew Waterman, Henry Cook, Brian Zimmer, Ben Keller, Alberto Puggelli, Jaehwa Kwak,
Ruzica Jevtić, Stevo Bailey, Milovan Blagojević, Pi-Feng Chiu, Rimas Avizienis, Brian Richards,
Jonathan Bachrach, David Patterson, Elad Alon, Borivoje Nikolić, Krste Asanović
EECS Department, University of California, Berkeley

The final phase of CMOS technology scaling provides continued increases in already vast transistor counts, but minimal improvements in energy efficiency, thus requiring innovation in circuits and architectures. However, even huge teams are struggling to complete large, complex designs on schedule using traditional rigid development flows. This article presents our agile hardware development methodology, which we have adopted for eleven RISC-V microprocessor tapeouts on modern 28 nm and 45 nm CMOS processes in the past five years, and discusses how it enabled small teams to build energy-efficient, cost-effective, and industry-competitive high-performance microprocessors in a matter of months. Our agile methodology relies on rapid iterative improvement of fabricatable prototypes using hardware generators written in Chisel, a new hardware description language embedded in a modern programming language. The parameterized generators construct highly customized systems based on the free, open, and extensible RISC-V platform. We present a case study of one such prototype featuring a RISC-V vector microprocessor integrated with a switched-capacitor DC-DC converter alongside an adaptive clock generator in a 28 nm fully-depleted silicon-on-insulator (FD-SOI) process.

1 Introduction

The end of Dennard scaling has drawn the rapid improvement of transistor performance to a close, increasing pressure on architects and circuit designers to improve energy efficiency. Modern systems-on-chip (SoCs) incorporate a large and growing number of specialized hardware units to execute specific tasks efficiently, often organized into multiple dynamic voltage and clock frequency domains to further save energy under varying computational loads. This growing complexity has led to a design productivity crisis, stimulating development of new tools and methodologies to enable the completion of complex chip designs on schedule and within budget.

We propose leveraging lessons learned from the software world by applying aspects of the software agile development model to hardware. Traditionally, software was developed via the waterfall development model, a sequential process that consists of distinct phases that rigidly follow one another, just as is done for hardware design today. Over-budget, late, and abandoned software projects were commonplace, motivating a revolution in software development, demarcated by the publication of the Agile Manifesto in 2001 [3]. The philosophy of agile software development emphasizes individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan. In practice, the agile approach leads to small teams iteratively refining a set of working-but-incomplete prototypes until the end result is acceptable.

Inspired by the positive disruption of the Agile Manifesto on software development, we propose a set of principles to guide a new agile hardware development methodology. Our proposal is informed by our experiences as a small group of researchers designing and fabricating eleven processor chips in five years. Lacking the massive resources of industrial design teams, we were forced to abandon standard industry practices and explore different approaches to hardware design. We detail the benefits of this approach with a case study of one of these chips,

Digital Object Indentifier 10.1109/MM.2016.11 0272-1732/$26.00 2016 IEEE

Raven-3, which achieved groundbreaking energy efficiency by integrating a novel RISC-V vector architecture with efficient on-chip DC-DC converters. Taken together, we feel our experiences developing these academic prototype chips make a powerful argument for the promise of agile hardware design in the post-Dennard era.

2 An Agile Hardware Manifesto

We present an agile hardware manifesto modeled suitable to the domain of complex SoC design. Borrowing heavily from the manifesto for agile software development, we propose the following principles for hardware design:

- **Incomplete fabricatable prototypes** over fully featured models.
- **Collaborative, flexible teams** over rigid silos.
- **Improving tools and generators** over improving the instance.
- **Responding to change** over following a plan.

This section explains the importance of each of these principles for building hardware, and contrasts them with the original principles of agile software development.

2.1 Incomplete fabricatable prototypes over fully featured models

We believe the primary benefit of adopting the agile model for hardware design is that it allows us to drastically reduce the cost of *verification* and *validation* by iterating through many working prototypes of our designs. In this

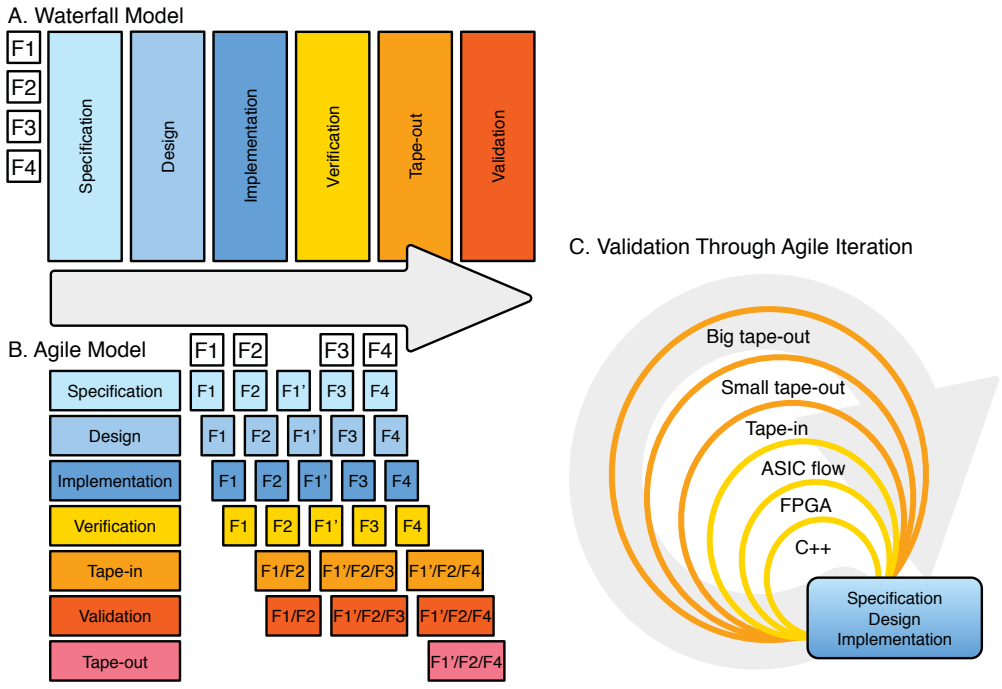


Figure 1: Contrasting the agile and waterfall models of hardware design. The labels F*N* represent various desired features of the design. A. The waterfall model steps all features through each activity sequentially, only producing a tapeout candidate once all features are complete. B. The agile model adds features incrementally, resulting in incremental tapeout candidates as individual features are completed, reworked, or abandoned. C. As validation progresses, designs are subjected to lengthier and more accurate evaluation methodologies.

Digital Object Indentifier 10.1109/MM.2016.11 0272-1732/$26.00 2016 IEEE

article, we use the standard definition of *verification* as testing that determines whether each component and the assembly of components correctly meets its specification ("are we building the thing right?"). In hardware design, the term *validation* is usually narrowly applied to post-silicon testing to ensure that manufactured parts operate within design parameters. We use the broader and, to our minds, more useful definition of validation from the software world, where validation ensures that the product serves its intended purposes ("are we building the right thing?").

Figure 1 contrasts the agile hardware development model with the waterfall model. The waterfall model relies on Gantt charts, high-level architecture models, rigid processes such as RTL freezes, and CPU-centuries of simulations in an attempt to achieve a single viable design point. In our agile hardware methodology, we first push a trivial prototype with a minimal working feature set all the way through the toolflow to a point where it could be taped out for fabrication. We refer to this tape-out-ready design as a *tape-in*. Then we begin adding features iteratively, moving from one fabricatable tape-in to the next as we add features. The agile hardware methodology will always have an available tape-out candidate with some subset of features for any tape-out deadline, whereas the conventional waterfall approach is in danger of grossly overshooting any tape-out deadline due to issues at any stage of the process.

While the agile methodology provides some benefits in terms of verification effort, where it particularly shines is validation: agile designs are more likely to meet energy and performance expectations using the iterative process. Unlike conventional approaches that use high-level architectural models of the whole design for validation with the intent to later refine these models into an implementation, we emphasize validation using a full register-transfer level (RTL) implementation of each incomplete prototype with the intent to add features if there is time. Unlike high-level architectural models, the actual RTL is guaranteed to be cycle-accurate. Furthermore, the actual RTL design can be pushed through the entire VLSI toolflow to give early feedback on backend physical design issues that will affect the quality-of-results (QoR), including cycle time, area, and power. For example, Figure 1.B shows a feature F1 being reworked into feature F1' after validation showed it would not meet performance or QoR goals, while feature F3 was dropped after validation showed it exceeded physical design budgets or did not add sufficient value to the end system. Informed by the results of attempted physical design, these decisions can be made before the design of the final feature F4 is completed.

Figure 1.C illustrates our iterative approach to validation of a single prototype. The circumference of each circle represents the relative time and effort it takes to begin validating a design using a particular methodology. Although the latency and cost of these prototype modeling efforts increases towards the outer circles, our confidence in the validation results produced increases in tandem. Using fabricatable prototypes increases validation bandwidth, as a complete RTL design can be mapped to FPGAs to run end-application software stacks orders of magnitude faster than with software simulators. In agile hardware development, the FPGA models of iterative prototype RTL designs (together with accompanying QoR numbers from the VLSI toolflow) fulfill the same function as working prototypes in agile software development, providing a way for end-customers to give early and frequent feedback to validate design choices. This enables customer collaboration as envisioned in the original agile software manifesto.

2.2 Collaborative, flexible teams over rigid silos

Traditional hardware design teams are usually organized around the waterfall model, with engineers assigned to particular functions, such as architecture specification, microarchitecture design, RTL design, verification, physical design, and validation. This specialization of skills is more extreme than for pure software development, as

Digital Object Indentifier 10.1109/MM.2016.11 0272-1732/$26.00 2016 IEEE

reflected in the specificity of job titles held by designers (architect, hardware engineer, or verification engineer). In some cases, some functions, such as backend physical design, are even outsourced to different companies. As the design progresses through these functional stages, extensive effort is required to communicate design intent for each block, and consequently to understand the documentation received from the preceding stage. Misunderstandings can lead to subtle errors that require extensive verification to prevent, and QoR suffers as no engineer views the whole flow. End-to-end design iterations are too expensive to contemplate. Considerable effort is required to push high-level changes down through the implementation hierarchy, particularly across company boundaries. This leads to innovation-stunting practices such as the "RTL freeze", where only show-stopping bugs are allowed to unfreeze a design. In addition, balancing workload across the different stages is difficult as engineers focused in one functional area often lack the training and experience to be effective in other roles.

In the agile hardware model, engineers work in collaborative, flexible teams that can drive a feature through all stages of implementation, avoiding most of the documentation overhead required to map a feature to the hardware. Each team is able to iterate on the high-level architectural design with full visibility into low-level physical implementation, and can validate system-level software impacts on an intermediate prototype. Figure 1.B shows how multiple teams can work on different features in a pipelined fashion, each iterating through multiple levels of abstraction as shown in Figure 1.C. A given tape-in will accumulate some small number of feature updates and integrate these for whole-system validation. Engineers naturally develop a complete vertical skill set, and the focus of project management is prioritizing features for implementation, not communication between silos.

2.3 Improving tools and generators over improving the instance

Modern software systems could not be built economically without the extreme reuse enabled by extensive software libraries written in modern programming languages, or without functioning compilers and an automated build process. In contrast, most commercial hardware design flows rely on rather primitive languages to describe hardware, and frequent manual intervention on tool outputs to work around tool bugs and long tool runtimes. The poor languages and buggy tools hinder development of truly resuable hardware components, and lead to a focus on building a single instance instead of designing components that can be easily reused in future designs.

An agile hardware methodology relies on better hardware description languages to enable reuse. Instead of constructing an optimized instance, engineers develop highly parameterized generators that can be used to automatically explore a design space, or to support future use in a different design. Better reuse is also the primary way in which agile can reduce verification costs, as the verification infrastructure of these generators is also portable across designs.

An agile hardware methodology also strives to eliminate manual intervention in the chip build process. Replacing or fixing the many commercial ECAD tools required to complete a chip design is not practical, but most of a chip flow can be effectively automated with sufficient effort. Perhaps the most labor-intensive part of a modern chip flow is dealing with physical design issues for a new technology node. However, we note that most chip design starts are in older technologies as only a few products can justify the cost of using an advanced node. Also, as technology scaling continues to slow, backend flows and ECAD tools will have time to mature and should become more automatable even for the most advanced nodes. The end of scaling will also coincide with a greater need for a variety of specialized designs, where the agile model should show greater benefit.

Our emphasis on tools and generators is not at odds with the manifesto for agile software development, which emphasizes collaboration over tools. Our proposed methodology also values collaboration more than specific tools,

Digital Object Indentifier 10.1109/MM.2016.11 0272-1732/$26.00 2016 IEEE

but we feel that modern hardware design is far less automated and exhibits far less reuse than modern software development (or even software development in 2001). Accordingly, we feel the need to emphasize the importance of reuse and automation enabled by new tools in the hardware design sphere, as these universally-accepted software principles are not emphasized by many modern hardware design teams.

2.4 Responding to change over following a plan

This principle mirrors the the original manifesto. Unlike software, a chip design will ultimately be frozen into a mask set and mass-manufactured, with no scope for frequent updates. However, even over the timeframe of a single chip development, requirements will change due to market shifts, customer feature additions, or standards body deliberations. In agile design, change can also result from validation testing (as opposed to the waterfall model, in which designs will only change in response to verification problems).

An agile hardware development process embraces continual change in chip specifications as a means to enhance competitiveness. Every incremental fabricatable prototype, not only the most recent, becomes a starting point for the changed set of design requirements, with any change treated as a reprioritization of features to implement, drop, or rework. By first implementing features that are unlikely to change, the effort lost to late changes can be minimized.

3 Implementing the Agile Hardware Methodology

This section describes the steps we have taken towards implementing an agile hardware design process. Our RTL source code is developed using the open-source Chisel hardware construction language (Section 3.1), and expressed as libraries of highly-parameterized hardware generators (Section 3.3). The free and open RISC-V ISA (Section 3.2) forms the base for all processors, both general-purpose and specialized. We have developed a highly automated backend design flow to reduce manual effort in pushing designs through to layout (Section 3.4). The resulting chips from our agile hardware design process are then presented (Section 3.5).

3.1 Chisel: Constructing Hardware In a Scala Embedded Language

To facilitate agile hardware development by improving designer productivity, we have developed the Chisel hardware construction language [2] as a domain-specific extension to the Scala programming language. Chisel is not a high-level synthesis (HLS) tool in which hardware is inferred from Scala code. Rather, Chisel is intended to be a substrate that provides a Scala abstraction of primitive hardware components, such as registers, muxes, and wires. Any Scala program whose execution generates a graph of such hardware components provides a blueprint to fabricate hardware designs; the Chisel compiler translates a graph of hardware components into a fast, cycle-accurate, bit-accurate C++ software simulator, or low-level synthesizeable Verilog that maps to standard FPGA or ASIC flows.

Because Chisel is embedded in Scala, hardware developers can tap into Scala's modern programming language features such as object-oriented programming, functional programming, parameterized types, abstract data types, operator overloading, and type inference to improve designer productivity by raising the level of abstraction and increasing code reuse. Furthermore, metaprogramming, code generation, and hardware design tasks are all expressed in the same source language, encouraging developers to write parameterized hardware generators rather

Digital Object Indentifier 10.1109/MM.2016.11 0272-1732/$26.00 2016 IEEE

```
   idiom                                                  result
A  (1,2,3) map { n => n + 1 }                             (2,3,4)
B  (1,2,3) zip (a,b,c)                                    ((1,a),(2,b),(3,c))
C  ((1,a),(2,b),(3,c)) map { case (left,right) => left }  (1,2,3)
D  (1,2,3) foreach { n => print(n) }                      123
E  for (x <- 1 to 3; y <- 1 to 3) yield (x,y)             (1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)
```

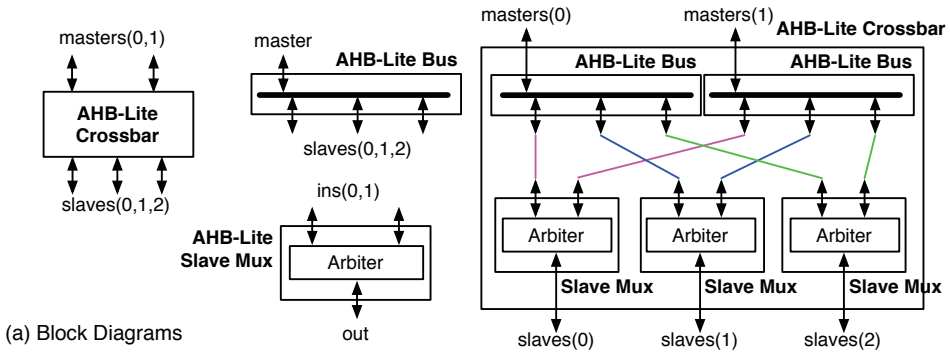
Figure 2: Generic functional programming idioms.

than discrete instances of individual blocks. This in turn improves code reuse within a given design and across generations of design iterations.

To see how these language features interact in real designs, we describe a crossbar that connects two AHB-Lite master ports to three AHB-Lite slave ports in Figure 3(a). The high-level block diagram is on the left, the two building blocks (AHB-Lite buses and AHB-Lite slave muxes) are in the middle, and the final design is shown on the right. To help understand the Chisel code, some generic functional programming idioms are shown in Figure 2. Idiom A maps a list of numbers to an expression that adds 1 to them. Idiom B zips two lists together. Idiom C iterates over a list of tuples, uses Scala's case statement to provide names of elements, and then operates on those names. Idiom D is used to iterate over a list when the output values are not collected. Idiom E is a for-comprehension with a yield statement.

Figure 3(b) shows the Chisel code that builds the crossbar. Line A declares the AHBXbar module with two parameters: number of master ports (`nMasters`) and slave ports (`nSlaves`). Lines C-F declares the I/O ports of the module. Lines H and I instantiates collections of AHB-Lite buses and AHB-Lite slave muxes. Lines K-N connects the building blocks together. Note, the Chisel operator <> connects module I/O ports together. Line K extracts the master I/O port of each bus, zips it to the corresponding master I/O port of the crossbar itself, and then connects them together. Line L does the same thing for each slave mux output and each slave I/O port of the crossbar. Lines M and N use for-comprehension to generate the cross product of all port combinations, and uses each pair to connect a specific slave I/O port of a specific bus to the matching input of the corresponding mux.

The brevity of this code ensures that it is easily verified by inspection. While the example shows two buses and three slave muxes, these 10 lines of code would be identical for any number of buses and slave muxes. In



(a) Block Diagrams

```
A  class AHBXbar(nMasters: Int, nSlaves: Int) extends Module
B  {
C    val io = new Bundle {
D      val masters = Vec(new AHBMasterIO, nMasters).flip
E      val slaves = Vec(new AHBSlaveIO, nSlaves).flip
F    }
G
H    val buses = List.fill(nMasters){Module(new AHBBus(nSlaves))}
I    val muxes = List.fill(nSlaves){Module(new AHBSlaveMux(nMasters))}
J
K    (buses.map(b => b.io.master) zip io.masters) foreach { case (b, m) => b <> m }
L    (muxes.map(m => m.io.out)    zip io.slaves ) foreach { case (x, s) => x <> s }
M    for (m <- 0 until nMasters; s <- 0 until nSlaves) yield {
N      buses(m).io.slaves(s) <> muxes(s).io.ins(m)
O    }
P  }
```

(b) Chisel Source Code

Figure 3: A crossbar switch written in Chisel.

Digital Object Indentifier 10.1109/MM.2016.11 0272-1732/$26.00 2016 IEEE

contrast, a Verilog or VHDL implementation of this crossbar would require many more lines of code. The same module written in traditional hardware description languages would be difficult to generalize into a parameterizable crossbar generator. This example is a small example of the power of Chisel to bring modern programming language constructs to hardware development, increasing code maintainability, facilitating re-use, and enhancing designer productivity.

3.2 RISC-V: A Free, Open, and Extensible Instruction Set

RISC-V [11] is a free, open, and extensible instruction set architecture (ISA) that forms the basis of all of our programmable processor designs. Unlike many earlier efforts that designed open processor cores, RISC-V is an ISA specification, intended to allow many different hardware implementations to leverage common software development. RISC-V is designed as a modular architecture, with variants covering 32-bit, 64-bit, and 128-bit address spaces. The base integer instruction set is lean, requiring less than 50 user-level hardware instructions to support a full modern software stack, enabling microprocessor designers to quickly bring up fully functional prototypes and add additional features incrementally. In addition to simplifying the implementation of new microarchitectures, the RISC-V design provides an ideal base for custom accelerators. Not only can accelerators reuse common control processor implementations, they can share a single software stack, including the compiler toolchain and operating system binaries. This dramatically reduces the cost of designing and bringing up custom accelerators.

Further information on RISC-V is available from the non-profit RISC-V Foundation, which was incorporated in August 2015 to help coordinate, standardize, protect, and promote the RISC-V ecosystem [7]. A free and open ISA is a critical ingredient in an agile hardware methodology, as having an active and diverse community of open-source contributors amortizes the overhead of maintaining and updating the software ecosystem, which makes it possible for smaller teams to focus on developing custom hardware [8]. A free ISA is also a prerequisite for shared open-source processor implementations, which can act as a base for further customization [1].

3.3 Rocket Chip Generator

Chisel's first-class support for object-orientation and metaprogramming allows hardware designers to write generators, rather than individual instances of designs, which in turn encourages the development of families of customizable designs. By encoding microarchitectural and domain knowledge these generators, we can quickly create different chip instances customized for particular design goals and constraints [9]. As constructing hardware generators requires support for reconfiguring individual components based on the context in which they are deployed, a particular focus of Chisel is to improve upon the limited module parameterization facilities of traditional hardware description languages.

The Rocket chip generator is written in Chisel and constructs a RISC-V-based platform. The generator consists of a collection of parameterized chip-building libraries that we can use to generate different SoC variants. By standardizing the interfaces that are used to connect different libraries' generators to one another, we have created a plug-and-play environment in which it is trivial to swap out substantial components of the design simply by changing configuration files, leaving the hardware source code untouched. We can also both test the output of individual generators as well as perform integration tests on the whole design, where the tests are also parameterized so as to exercise the entire design-under-test.

Digital Object Indentifier 10.1109/MM.2016.11 0272-1732/$26.00 2016 IEEE

This article has been accepted for publication in IEEE Micro but has not yet been fully edited.
Some content may change prior to final publication.

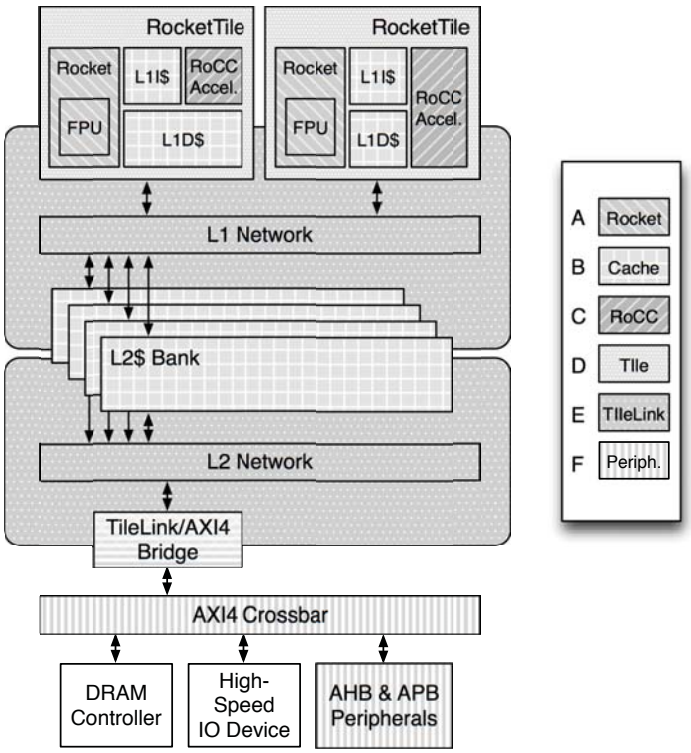


Figure 4: The Rocket chip generator consists of the following sub-components: A) Core generator B) Cache generator C) RoCC-compatible coprocessor generator D) Tile generator E) TileLink generator F) Peripherals

Figure 4 presents the collection of library generators and their interfaces within the Rocket chip generator. These generators can be parameterized and composed into a wide variety of SoC designs. Here is a summary of their current capabilities:

- **Core**: The Rocket scalar core generator with an optional FPU, configurable pipelining of functional units, and customizable branch prediction structures.
- **Caches**: A family of cache and TLB generators with configurable sizes, associativities, and replacement policies.
- **RoCC**: The Rocket Custom Coprocessor interface, a template for application-specific coprocessors which may expose their own parameters.
- **Tile**: A tile-generator template for cache-coherent tiles. The number and type of cores and accelerators are configurable, as is the organization of private caches.
- **TileLink**: A generator for networks of cache coherent agents and the associated cache controllers. Configuration options include the number of tiles, the coherence policy, the presence of shared backing storage, and the implementation of underlying physical networks.
- **Peripherals**: Generators for AXI4/AHB-Lite/APB compatible buses and a variety of converters and controllers.

3.4 Physical Design Flow

We frequently translate Chisel's Verilog output into a physical design through an ASIC CAD toolflow. We have automated this flow and optimized it to generated GDS in less than a day, without designer intervention. Automatic nightly regressions provide the designer regular feedback on cycle time, area, and energy consumption, and they

Digital Object Indentifier 10.1109/MM.2016.11 0272-1732/$26.00 2016 IEEE

also detect physical design problems early in the design cycle. Automatic verification and reporting on quality of results also allows more points in the design space to be evaluated, and reduces the barrier to implementing more significant changes at later stages in the development process.

One major hindrance to agile methodology for hardware design is CAD tool runtime. The "deployment" process for hardware is significantly more time consuming than for software. Using an initial hardware description to generate a GDS that is electrically and logically correct can require many iterations. Since each iteration takes many hours to run to completion, this process could require weeks or months for larger designs. To avoid the productivity loss of these long-latency iterations, we initially focus on smaller designs, from which GDS can be produced much more quickly. Iterating on these smaller-scale tape-ins has proven valuable, as major problems are uncovered much earlier in the design process, even more so when we target a new process technology. Additionally, the parameterizable nature of hardware generators reduces the effort to scale up to larger designs later in the design process. Regardless of design size, we ensure that each tape-in passes static timing analysis, is functionally equivalent to the RTL model, and has only a small number of design rule violations. Together, these properties ensure that the design is indeed tape-out-ready.

As a particular tapeout deadline approaches, we evaluate whether to tapeout the most recently taped-in design, usually based on whether the set of new features validated by the tape-in is sufficiently large. If we decide to tape out, we clean up the remaining design rule violations and run final checks on the resulting GDS. The full cycle of taping out the GDS and testing the resulting chip is more involved than a tape-in, but qualitatively it is just another, longer iteration in the agile methodology. Thus, we focus on producing lineages of increasingly complicated functioning chips, incorporating feedback from the previous chip into the next in the series. We believe that this rapid, iterative process is essential to improve the productivity of hardware designers and enable iterative design-space exploration of alternative system microarchitectures.

3.5 Silicon Results

Figure 5 presents a timeline of UC Berkeley RISC-V microprocessor tapeouts that were created using earlier versions of the Rocket-Chip generator. The 28 nm Raven chips combine a 64-bit RISC-V vector microprocessor with on-chip switched-capacitor DC-DC converters and adaptive clocking (see Section 4). The 45 nm EOS chips integrate a 64-bit dual-core RISC-V vector processor with monolithically-integrated silicon photonic links [6, 10]. In total, we have taped out four Raven chips on STMicroelectronics' 28 nm FD-SOI process, six EOS chips on IBM's 45 nm SOI process, and one SWERVE chip on TSMC's 28 nm process. The Rocket chip generator was

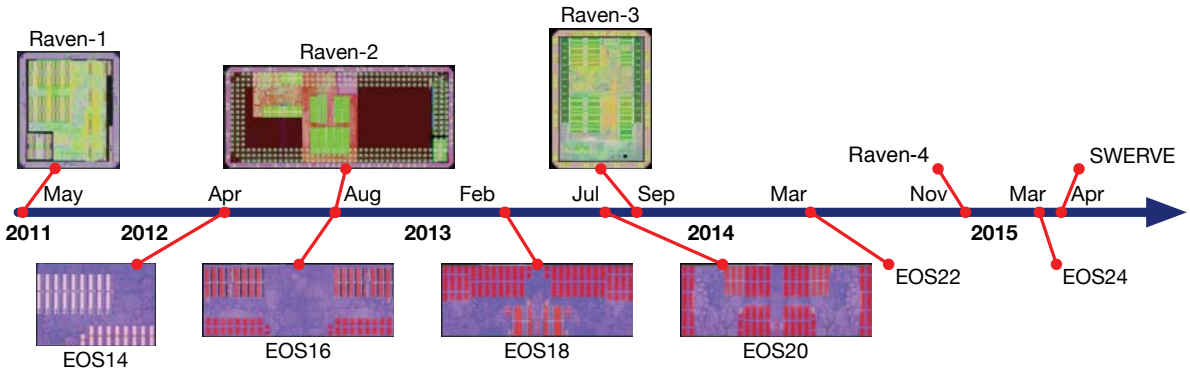


Figure 5: Recent UC Berkeley tapeouts using RISC-V processors.

Digital Object Indentifier 10.1109/MM.2016.11 0272-1732/$26.00 2016 IEEE

able to play the role of shared code base for these eleven distinct systems; the best ideas from each design were incorporated back into the code base, ensuring maximal re-use even though the three distinct families of chips were specialized differently to test out distinct research ideas.

4 Case Study: RISC-V Vector Microprocessor with On-Chip DC-DC Converters

This section details the application of our agile design methodology to Raven-3, a 28 nm microprocessor that integrates switched-capacitor (SC) DC-DC converters, adaptive clocking circuitry, and low-voltage SRAM macros with a RISC-V vector processor instantiated by the Rocket chip generator [12]. The successful development of the Raven-3 prototype demonstrates how architecture and circuit research was supported by the combination of the freely extensible RISC-V ISA, the parametrizable Chisel-based hardware implementation, and fast feedback via iteration and automation as part of the agile methodology.

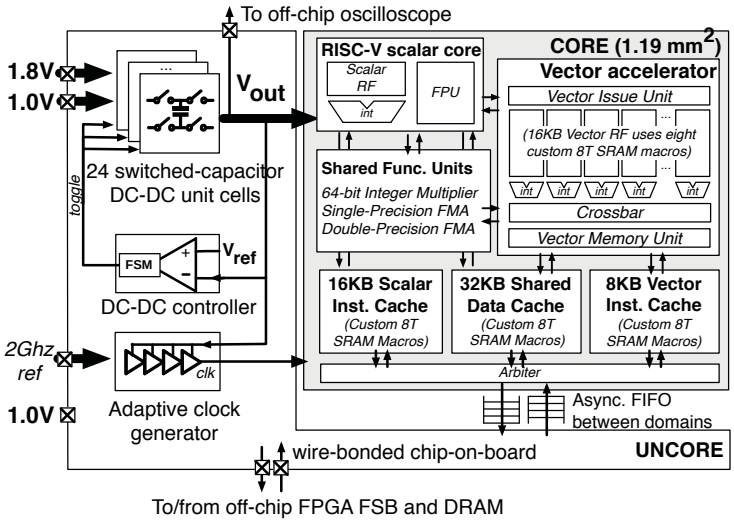


Figure 6: Raven-3 system-level block diagram.

Figure 6 shows the Raven-3 system-level block diagram. The Raven-3 microprocessor instantiates the upper-left RocketTile of Figure 4 with the Hwacha vector unit as a RoCC accelerator. We did not include an L2 cache in this iteration of the design, leaving this feature for implementation in future tape-ins and tape-outs. The chip consists of two voltage domains; the varying core domain and the fixed uncore domain. The core domain is connected to the output of fully-integrated non-interleaved SC DC-DC converters [5], and powers the processor and its L1 caches. The fixed domain powers the uncore and remains fixed at 1 V. The DC-DC converters generate four voltage modes between 0.5 V and 1 V from 1 V and 1.8 V supplies; they achieve better than 80% conversion efficiency, while requiring no off-chip passives, greatly simplifying package design [12]. The clock generator selects clock edges from a tunable replica circuit powered by the rippling core voltage that mimics the critical path delay of the processor, and adapts the clock frequency to the instantaneous voltage [4].

The design of Raven-3 benefitted greatly from the agile hardware design methodology. By aspiring to build an incomplete prototype (rather than, for instance, a manycore design implementing these technologies), we dramatically improved the odds of project success. The unique integration of the power delivery, clocking, and processor systems was only possible because the design team broadly shared expertise about all aspects of the project. The

Digital Object Indentifier 10.1109/MM.2016.11 0272-1732/$26.00 2016 IEEE

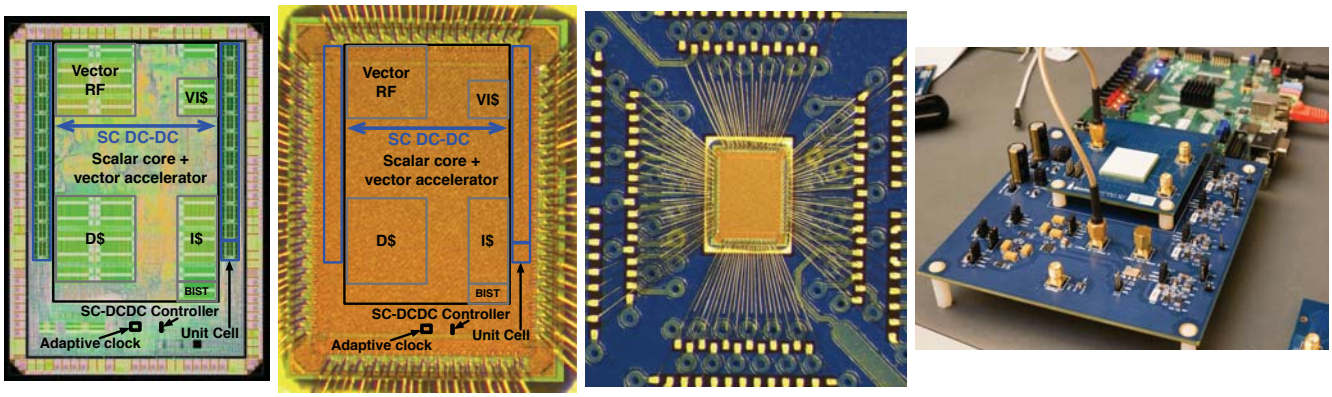


Figure 7: Raven-3 Implementation.

processor RTL could largely be reused from previous tape-outs, with only incremental improvements and configuration changes required to validate it for the particular feature set desired. As physical design on partial feature sets gave incremental results, we were able to adjust design features to improve feasibility and QoR.

The agile hardware design methodology was especially important in this design due to the complexity and risk introduced by the custom voltage conversion and clocking, as illustrated by the following example. Tape-ins of a simple digital system with a single DC-DC unit cell were small enough to simulate at the transistor level, and these simulations found a critical bug in the DC-DC controller. These transistor-level simulations showed that a large current event could cause the output voltage to remain below the lower-bound reference voltage even after the converter switched. A bug in the controller would cause the converter to stop switching and drop the processor voltage to zero for this case. This bug, which was fixed early in the design process by the addition of a simple state machine in the lower-bound controller, would have been found very late (if at all) with waterfall design approaches.

Figure 7 shows the (a) floorplan and (b) micrograph of the resulting chip, (c) the wire-bonded chip on the package board, and (d) the test setup. The resulting 2.37 mm^2 Raven chip is fully functional and boots Linux and executes compiled scalar and vector application code with the adaptive clock generator at all operating modes, while the DC/DC converter is able to transition between voltage modes in less than 20 ns. Raven-3 runs at a maximum clock frequency of 961 MHz at 1 V consuming 173 mW, and at a minimum voltage of 0.45 V at 93 MHz consuming 8 mW. Raven-3 achieves a maximum energy efficiency of 27 and 34 GFLOPS/W while running a double-precision matrix-multiplication kernel on the vector accelerator with and without the SC DC-DC converters respectively. These results validate the agile approach to hardware design.

5 Conclusion

Our agile hardware development methodology played an important role in successfully taping out eleven RISC-V microprocessors in five years at UC Berkeley. With a combination of our agile methodology, Chisel, RISC-V, and the Rocket chip generator, a small team of graduate students at UC Berkeley was able to design multiple microprocessors that are competitive with industrial designs. While there is much work left to do to show this approach can scale up to the largest SoCs, we hope these projects serve as an opportunity to rethink how modern SoCs are built, shed some light on inefficiencies in the current design process, and revitalize hardware design.

Digital Object Indentifier 10.1109/MM.2016.11 0272-1732/$26.00 2016 IEEE

Acknowledgements

The authors would like to thank Tom Burd, James Dunn, Olivier Thomas, and Andrei Vladimirescu for their contributions, and STMicroelectronics for fabrication donation. This work was funded in part by BWRC, ASPIRE, DARPA PERFECT Award Number HR0011-12-2-0016, STARnet C-FAR, Intel ARO, AMD, SRC/TxACE, Marie Curie FP7, NSF GRFP, and an NVIDIA Graduate Fellowship.

References

[1] K. Asanović and D. Patterson. The Case for Open Instruction Sets. *Microprocessor Report*, Aug 2014.

[2] 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 *Proc. Design Automation Conference*, pages 1216–1225, 2012.

[3] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mallor, K. Shwaber, and J. Sutherland. The Agile Manifesto. Technical report, The Agile Alliance, 2001.

[4] J. Kwak and B. Nikolić. A 550-2260MHz Self-Adjustable Clock Generator in 28nm FDSOI. In *Proc. IEEE Asian Solid-State Circuits Conference*, November 2015.

[5] H.-P. Le, S. Sanders, and E. Alon. Design Techniques for Fully Integrated Switched-Capacitor DC-DC Converters. *IEEE Journal of Solid-State Circuits*, 46(9):2120–2131, Sept 2011.

[6] Y. Lee, A. Waterman, R. Avižienis, H. Cook, C. Sun, V. Stojanović, and K. Asanović. A 45nm 1.3GHz 16.7 Double-Precision GFLOPS/W RISC-V Processor with Vector Accelerators. *Proc. IEEE European Solid-State Circuits Conference*, Sep 2014.

[7] R. Merritt. Google, HP, Oracle Join RISC-V. *EE Times*, Dec 2015.

[8] V. Patil, A. Raveendran, P. Sobha, A. David Selvakumar, and D. Vivian. Out of Order Floating Point Coprocessor for RISC V ISA. In *IEEE International Symposium on VLSI Design and Test*, pages 1–7, June 2015.

[9] O. Shacham, O. Azizi, M. Wachs, W. Qadeer, Z. Asgar, K. Kelley, J. Stevenson, S. Richardson, M. Horowitz, B. Lee, A. Solomatnikov, and A. Firoozshahian. Rethinking Digital Design: Why Design Must Change. *Micro, IEEE*, 30(6):9–24, November 2010.

[10] C. Sun, M. T. Wade, Y. Lee, J. S. Orcutt, L. Alloatti, M. S. Georgas, A. S. Waterman, J. M. Shainline, R. R. Avižienis, S. Lin, B. R. Moss, R. Kumar, F. Pavanello, A. H. Atabaki, H. M. Cook, A. J. Ou, J. C. Leu, Y.-H. Chen, K. Asanović, R. J. Ram, M. A. Popović, and V. M. Stojanović. Single-chip microprocessor that communicates directly using light. *Nature*, 528:534–538, 2015.

[11] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0. Technical Report UCB/EECS-2014-54, EECS Department, University of California, Berkeley, May 2014.

[12] B. Zimmer, Y. Lee, A. Puggelli, J. Kwak, R. Jevtić, B. Keller, S. Bailey, M. Blagojević, P.-F. Chiu, H.-P. Le, P.-H. Chen, N. Sutardja, R. Avižienis, A. Waterman, B. Richards, P. Flatresse, E. Alon, K. Asanović, and B. Nikolić. A RISC-V vector processor with tightly-integrated switched-capacitor DC-DC converters in 28nm FDSOI. In *Proc. IEEE Symposium on VLSI Circuits*, June 2015.

Yunsup Lee is pursuing a PhD in computer science at the University of California, Berkeley. His research interests include computer architecture, VLSI design, and compilers. He has an MS in computer science from Berkeley and a BS in electrical engineering and computer science from Korea Advanced Institute of Science and Technology.

Andrew Waterman is pursuing a PhD in computer science at the University of California, Berkeley. His research interests include computer architecture, VLSI design, and operating systems. Waterman has an MS in computer science from Berkeley and a BSE in electrical and computer engineering from Duke University.

Digital Object Indentifier 10.1109/MM.2016.11 0272-1732/$26.00 2016 IEEE

Henry Cook is pursuing a PhD in computer science at the University of California, Berkeley. His research interests include computer architecture, compilers, and design space exploration. He has an MS in computer science from UC Berkeley and a BS in computer science from the University of Virginia.

Brian Zimmer has a PhD in electrical engineering and computer sciences from the University of California, Berkeley. His research interests include energy-efficient digital design and low-voltage SRAM design. He is currently a research scientist in the Circuits Research Group at NVIDIA.

Ben Keller is pursuing a PhD in electrical engineering at the University of California, Berkeley. His research interests include fine-grained dynamic voltage and scaling. Ben earned his BS in engineering from Harvey Mudd College and his MS in electrical engineering from the University of California, Berkeley.

Alberto Puggelli received the M.S. degree in computer science and the Ph.D. degree in electrical engineering from the University of California, Berkeley. His research interests include the design and control of power management techniques for energy-efficient SoCs. Dr. Puggelli currently works at Lion Semiconductor.

Jaehwa Kwak is pursuing a PhD in electrical engineering and computer sciences at University of California, Berkeley. His research interests include low power microprocessor design and the self-adjusting clocks. He received a BS and an MS in electrical engineering and computer sciences from Seoul National University.

Ružica Jevtić is an Assistant Professor at the Universidad de Antonio de Nebrija in Madrid. Her research interests include digital circuit design for low power and soft error circuit protection. She has a PhD in electrical engineering from Technical University of Madrid and was a postdoctoral fellow at the University of California, Berkeley.

Stevo Bailey is a PhD student at the University of California, Berkeley. His research interests include soft-error resilient logic design techniques, agile hardware design flows, and low-power digital ASIC designs. He received his BS degrees from the University of Virginia and his MS degree from the University of California, Berkeley.

Milovan Blagojević received the BSc and MSc degrees in electrical engineering from the University of Belgrade, Serbia. He is enrolled in a CIFRE PhD program realized in cooperation among three institutions: the Berkeley Wireless Research Center, STMicroelectronics, and Institut Supérieur d'Electronique de Paris.

Pi-Feng Chiu received her B.S. and M.S. degrees in electrical engineering from National Tsing Hua University, Taiwan. She is currently pursuing a PhD in electrical engineering at the University of California, Berkeley. Her current research interests are in nonvolatile memories and energy-efficient VLSI design.

Rimas Avižienis is pursuing a PhD in computer science at the University of California, Berkeley. His research interests include energy-efficient VLSI design and specialized processor architectures for software defined radio. He has an MS in computer science from the University of California, Berkeley.

Brian Richards received the BS degree in Electrical Engineering from the California Institute of Technology and the MS degree in Electrical Engineering and Computer Science from the University of California, Berkeley. In 1986, he joined the research staff at the University of California, Berkeley, where he worked on large scale digital system design projects. He is a founding member of the Berkeley Wireless Research Center.

Jonathan Bachrach is an Adjunct Assistant Professor of electrical engineering and computer science at University of California, Berkeley. The overall goal of his research is to introduce software techniques to accelerate the design of electromechanical systems: changing the way we design, what components we use, and what machines we use to fabricate them with. He holds a PhD and MS in computer science from University of Massachusetts, Amherst.

David Patterson is Pardee Professor of Computer Science at UC Berkeley, where he has been for 40 years since he received his BS, MS, and PhD from the University of California, Los Angeles. His best-known research projects are Reduced Instruction Set Computers (RISC), Redundant Arrays of Inexpensive Disks (RAID), and Network of Workstations. He is a Fellow of ACM and IEEE and was elected to the National Academy of Engineering, the National Academy of Sciences, and the Silicon Valley Engineering Hall of Fame.

Digital Object Indentifier 10.1109/MM.2016.11 0272-1732/$26.00 2016 IEEE

Elad Alon is an Associate Professor of Electrical Engineering and Computer Sciences at UC Berkeley and a co-Director of the Berkeley Wireless Research Center. His research focuses on energy-efficient integrated systems, including circuit, device, communications, and optimization techniques and methodologies used to design them. Alon has a PhD in electrical engineering from Stanford University, and is a Senior Member of IEEE.

Borivoje Nikolić is the National Semiconductor Distinguished Professor of Engineering at the University of California, Berkeley. He received the Dipl.Ing. and MSc degrees in electrical engineering from the University of Belgrade, Serbia, and the Ph.D. degree from the University of California, Davis. His research activities include the design of energy-efficient digital, analog and RF integrated circuits and systems.

Krste Asanović is a Professor in the Electrical Engineering and Computer Sciences Department at the University of California, Berkeley. He received a BA in Electrical and Information Sciences from Cambridge University and a PhD in Computer Science from the University of California, Berkeley. He is an ACM Distinguished Scientist and an IEEE Fellow.

Digital Object Indentifier 10.1109/MM.2016.11 0272-1732/$26.00 2016 IEEE