Engineering Educators Bringing the World Together
2025 ASEE Annual Conference & Exposition
Palais des congrès de Montréal, Montréal, QC • June 22–25, 2025 ASEE
Paper ID #48675

RISC-V System-on-Chip Design Textbook and Course

Dr. Rose Thompson, Oklahoma State University

Rose Thompson received her Ph.D. in Electrical Engineering from Oklahoma State University and two B.S. degrees in Electrical Engineering and Computer Engineering from the University of Washington. She has also designed chips at the Air Force Research Laboratory. Her professional interests include SoC design and verification, custom instruction set architectures, branch prediction, memory systems, and secure computing. Rose also enjoys biking, hiking, rock climbing, and playing the piano.

Prof. David L Harris, Harvey Mudd College

David Harris is the Harvey S. Mudd Professor of Engineering Design at Harvey Mudd College. He received his Ph.D. from Stanford University and S.B. and M.Eng. degrees from MIT. His interests are in the area of microprocessor design, computer arithmetic, and high speed and low power circuits. David is the author of Digital Design and Computer Architecture, CMOS VLSI Design, Logical Effort, and Skew-Tolerant Circuit Design, and he holds sixteen patents in the field. He has designed chips at Intel, Broadcom, Sun Microsystems, Hewlett-Packard, and elsewhere. David's other interests include writing Southern California hiking guidebooks, building experimental aircraft, and traveling with his family.

James Stine Jr., Oklahoma State University

I am a Professor at Oklahoma State University interested in pushing the frontiers of computation within digital logic for general-purpose and application-specific computer architectures. I have interests in logic design for high-speed and low-power arithmetic, VLSI, FPGA, memory architectures, divide and square root implementations, computer architectures, cryptographic implementations, and graphics applications.

I have also developed several design flows for use with Electronic Design Automation (EDA) tools, including the FreePDK with my colleagues Rhett Davis from NC State University and those at the Semiconductor Research Corporation (SRC), several Cadence Design Systems (CDS) flows including the GPDK and MOSIS flows for use with CDS, National Science Foundation-funded OpenRAM, and Mentor Graphics and Synopsys EDA flows. I have also developed design flows for Google, Skywater Technology, IBM, trusted foundry, and the US Air Force. I am committed to use my experience to help others learn these tools and help develop them to further research endeavors for everyone involved.

Prof. Sarah L Harris, University of Nevada - Las Vegas

Sarah L. Harris is a Professor of Electrical and Computer Engineering at the University of Nevada, Las Vegas. She completed her M.S. and Ph.D. at Stanford University. Before joining UNLV in 2014, she was a faculty member at Harvey Mudd College for ten years. She is the co-author of four textbooks, including Digital Design and Computer Architecture: RISC-V Edition (2021, Morgan Kaufmann). Her research interests include computer architecture and applications of embedded systems and machine learning to biomedical engineering and robotics.

RISC-V System-on-Chip Design Textbook and Course

Rose Thompson[1], David Harris[2], James E. Stine[1], Sarah L. Harris[3]
[1]Oklahoma State University, [2]Harvey Mudd College, [3]University of Nevada, Las Vegas

Abstract

We wrote a textbook, *RISC-V System-on-Chip Design*, to bridge the gap between learning about the theory of processor, computer architecture, and system-on-chip (SoC) design and being able to put these theories into practice by understanding, simulating, analyzing, and expanding a fully functional processor and SoC. The book begins with a brief history of processor design and an overview of the RISC-V architecture and then describes the tools needed to work with Wally, the open-source RISC-V SoC described in the book. These tools include GCC and the Sail, Spike, and Verilator simulators as well as best practices in hardware description language (HDL) design, design verification, and logic synthesis. The Wally SoC supports both of RISC-V's base integer instruction sets, RV32I and RV64I, caches, branch prediction, virtual memory, and many extensions, including the compressed (C), multiply/divide (M), floating-point (Zfh/F/D/Q), atomic (A), and bit manipulation (B) extensions. Wally supports the RVI20U32, RVI20U64, and RVA22S64 RISC-V profiles and can boot Linux with privilege modes and virtual memory and can run on an FPGA. The textbook can be used to teach courses in computer architecture, SoC design, design verification, embedded systems, or a subset of these in theory, practice, or both. We describe two types of courses we taught using a draft version of this textbook: a senior/master's level course that focused on all stages of SoC design and a second course taught at the sophomore/junior level that focused on computer architecture and processor design only. These courses used the labs, exercises, and Wally SoC that accompany the textbook. We expect this book and course, as well as Wally, to continue to evolve both in its capabilities and in the way that it is used and taught.

Keywords
RISC-V, computer architecture, microarchitecture, system-on-chip, SoC, verification, synthesis, Linux, FPGA

1. Introduction

The RISC-V architecture was introduced as an open and royalty-free instruction set architecture in 2010 and it has evolved since then to become increasingly influential both academically and commercially [1, 2, 3]. Because of its commercial relevance and its lack of a license, RISC-V offers an ideal foundation for showing how to build a system-on-chip (SoC) based on a real-world computer architecture.

Existing textbooks and courses often introduce the theory of architecture or SoC design, but they focus only on the theory or the implementation or on a subset of topics, never tying all the topics together to build a fully functional SoC. Our textbook and course aim to bridge the gap between theory and practice and to demystify all levels of SoC design by teaching how to understand the theory and fundamentals of an SoC and its design and then build on that understanding to teach the practice of implementing, modifying, and verifying an SoC, starting with digital design and

verification fundamentals and proceeding through architecture, processor, and SoC design, with a full understanding of each building block and step.

RISC-V offers an ideal teaching platform because of its organization, including extensions, platforms, and its evolving and accessible ecosystem. Because the architecture and many aspects of its ecosystem, including compilers, are open source, this enables users to readily access tools to experiment with all aspects of the design, including understanding the existing hardware, compiling and running programs on the system and expanding it to support additional features. Our textbook teaches the fundamentals of the RISC-V architecture first, focusing on the RV32I and RV64I integer instruction sets, and then builds on that foundation to add extensions and features, including a system bus, external memory, and peripherals, to build up a fully functional SoC. We teach these all in the context of hardware design and verification. In addition to describing the architectural theory, our book and its accompanying open-source SoC, called Wally, describe and show in-depth implementation details of a real SoC design. The code for Wally is available as an open-source resource that accompanies the book. Wally is configurable and can range from a small microcontroller to a larger application processor that can run Linux, with caches, virtual memory, and a system bus.

The remainder of this paper describes the textbook, its accompanying resources, and the Wally SoC. We also discuss the organization, experiences, and results of two types of courses taught using the book and its resources and compare this textbook with existing RISC-V and SoC textbooks. We conclude by summarizing the textbook, its resources, and future directions.

2. RISC-V System-on-Chip Design Textbook Overview and Structure

Our textbook *RISC-V System-on-Chip Design* is being published by Morgan Kaufmann, an imprint of Elsevier, in 2025. It includes the chapters listed in Table 1. The first two chapters give an overview of computer design and the RISC-V architecture. Chapters 3-6 describe the tools used in the design flow, including the GCC compiler, simulators such as Sail, Spike, and Verilator, the SystemVerilog hardware description language (HDL), and design verification and synthesis. Chapters 7-13 describe the RISC-V processor that is the heart of the Wally SoC. These chapters first describe the pipelined core then add support for privileged operations, an AHB bus interface, caches, a memory management unit (MMU), the load/store and instruction fetch units (LSU and IFU), and branch predictor. By adding a bus interface to the processor, the system can add peripherals such as a UART, SPI, and external memory to become an SoC. Chapters 14-19 expand the SoC to support RISC-V extensions, including compressed, multiply and divide, floating-point, atomic, bit manipulation, cryptography, and other instructions. The final four chapters (20-23) show how to add peripherals to Wally, measure Wally's performance using benchmarking, load and run Linux on Wally, and implement Wally on an FPGA. The appendices, like grades, are lettered A-D and F. Appendix A gives a summary of Wally features, including its configuration parameters, supported features and profiles, and main diagrams. Appendices B-D show how to use some of the tools and platforms needed to work with Wally: Linux, Git, and Tcl. The final appendix describes the Wally floating-point implementation, which is a follow-on to Chapter 16's discussion of floating-point theory.

Table 1. Textbook Chapters

Number	Title
1	A Brief History of Computer Design
2	Introduction to RISC-V
3	RISC-V Software Tool Flow
4	HDL Design Practices
5	Design Verification
6	Logic Synthesis
7	Pipelined Core
8	Privileged Operations
9	Bus Interface
10	Caches
11	Memory Management Unit
12	Load/Store Unit
13	Instruction Fetch Unit
14	Extensions: C (Compressed)
15	Extensions: M (Multiply and Divide)
16	Extensions: F/D/Q/Zfh/Zfa (Floating-Point)
17	Extensions: A (Atomic)
18	Extensions: Zb* & Zk* (Bit Manipulation and Cryptography)
19	Other Extensions
20	Peripherals
21	Benchmarking
22	Linux
23	FPGA Implementation
A	Wally Synopsis
B	Hitchhiker's Guide to Linux
C	Version Control using Git
D	Tcl Book of Armaments
F	Floating Point Implementation

The textbook also includes lecture slides, lab assignments, and exercises that will be available as an online resource on the textbook's companion website. Table 2 lists the available labs, which are typically followed by a final project, and Table 3 lists a brief synopsis of some example exercises.

Table 2. Labs

Number	Name
0	Getting Started with Wally & Tool Setup
1	Programming
2	Verification
3	Debug
4	Synthesis
5	Design

Table 3. Example Exercises

Chapter	Name
2	Write a RISC-V program for loading the following immediate into register `s2`: 0x89ABCDEF.
4	When designing combinational logic in SystemVerilog, when is it best to use `assign` statements? When is it best to use `always_comb` with `case` statements?
16	The current floating-point divide/square root finite state machine (FSM) takes longer than strictly necessary because of the transition through the DONE state. Optimize the FSM to save a cycle, both during normal and early termination.

3. Overview of the Wally SoC

The Wally SoC that we designed and describe in the book supports both of the base integer instruction sets, RV32I and RV64I, caches, branch prediction, virtual memory, and many extensions, including the compressed (C), multiply/divide (M), floating-point (Zfh/F/D/Q), atomic (A), and bit manipulation (B) extensions. Wally's full source code is open source and is hosted by OpenHW Group at https://github.com/openhwgroup/cvw.

To simplify describing which extensions and features are supported by a system, RISC-V International, which governs RISC-V standards and ratifications, defined profiles to summarize them [4]. Wally configurations support the RVI20U32, RVI20U64, and RVA22S64 RISC-V profiles, can boot Linux with privilege modes and virtual memory, and can run on an FPGA.

Figure 1 shows the Wally SoC. It consists of the processor core (everything below the EBU), an external bus unit (EBU), and the uncore, which includes all peripherals and external memory that may be on or off chip. The core has the typical five pipeline stages: Fetch, Decode, Execute, Memory, and Writeback, as indicated at the bottom of Figure 1. The processor fetches the instruction from memory and decodes it, with registers being read, in the first two stages. Then the processor performs the operation in the Execute stage and memory is potentially read or written in the Memory stage. Finally, the processor writes the instruction's result to the register file in the Writeback stage.

The main processor modules, drawn in light blue boxes in Figure 1, are the: IFU (instruction fetch unit), IEU (instruction execute unit), LSU (load/store unit), MDU (multiply/divide unit), FPU (floating-point unit), and the privileged unit. The textbook devotes one chapter to describe each of these units as well as the EBU and caches (Chapters 7-13 and 16). The hazard unit is also part of the processor; it is introduced in Chapter 7 and expanded throughout the book.

The IFU is in the Fetch stage and it includes the PC and supporting logic (labeled PC), branch predictor (BP), instruction cache (I$), and the IFU's memory management unit (MMU). The Decode stage reads registers from the integer or floating-point register file (IRF or FRF) and converts compressed, 16-bit, instructions to regular 32-bit instructions using the decompress (decomp) unit. The Execute stage includes the arithmetic logic unit (ALU) in the IEU as well as the first stages of the multiply/divide (MDU), and floating-point (FPU) calculations. The

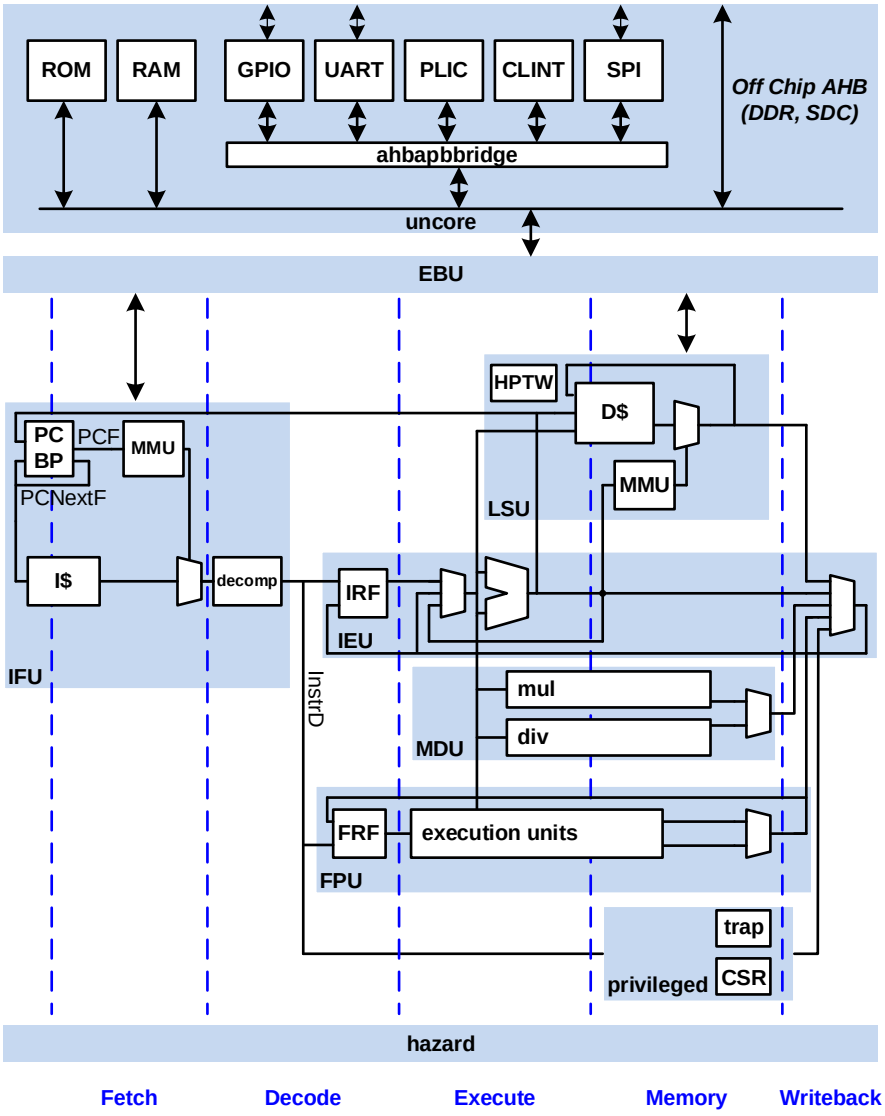

Figure 1. Wally SoC

Memory stage accesses memory using the LSU's data cache (D$) and MMU and checks privilege levels. The Writeback stage writes the calculated result back to the IRF or FRF. Note that the caches may be replaced by on-chip instruction and data memories (IROM and DTIM), depending on how Wally is configured. Wally is also referred to as Core-V Wally (cvw) because it has five (V) stages.

The EBU manages communication between the core and the uncore peripherals and memory using an AHB bus. The peripherals use the simpler APB bus, so the uncore includes an AHB-to-APB bridge (module `ahbapbbridge`). The uncore currently supports general-purpose I/O (GPIO), a UART, interrupt controllers (PLIC and CLINT), Serial Peripheral Interface (SPI), and access to external memories.

Figure 2 shows a zoomed-in picture of a simple Wally core configured to support only RISC-V's integer instructions (RV32I or RV64I). This figure is similar to the simple 5-stage pipeline discussed in *Digital Design and Computer Architecture, RISC-V Edition* [5], but it has some subtle differences: the on-chip instruction and data memories (IROM and DTIM) are shown on top of the pipeline registers because their addresses set up one cycle before they are read or written; the core also uses a separate comparator for branches and jumps (`comp`) instead of using the ALU; and `PCLinkE` is calculated again in the Execute stage. Here the instruction and data memories are shown as on-chip memories (IROM and DTIM) instead of caches as in Figure 1, but the core could also be configured to use caches instead.

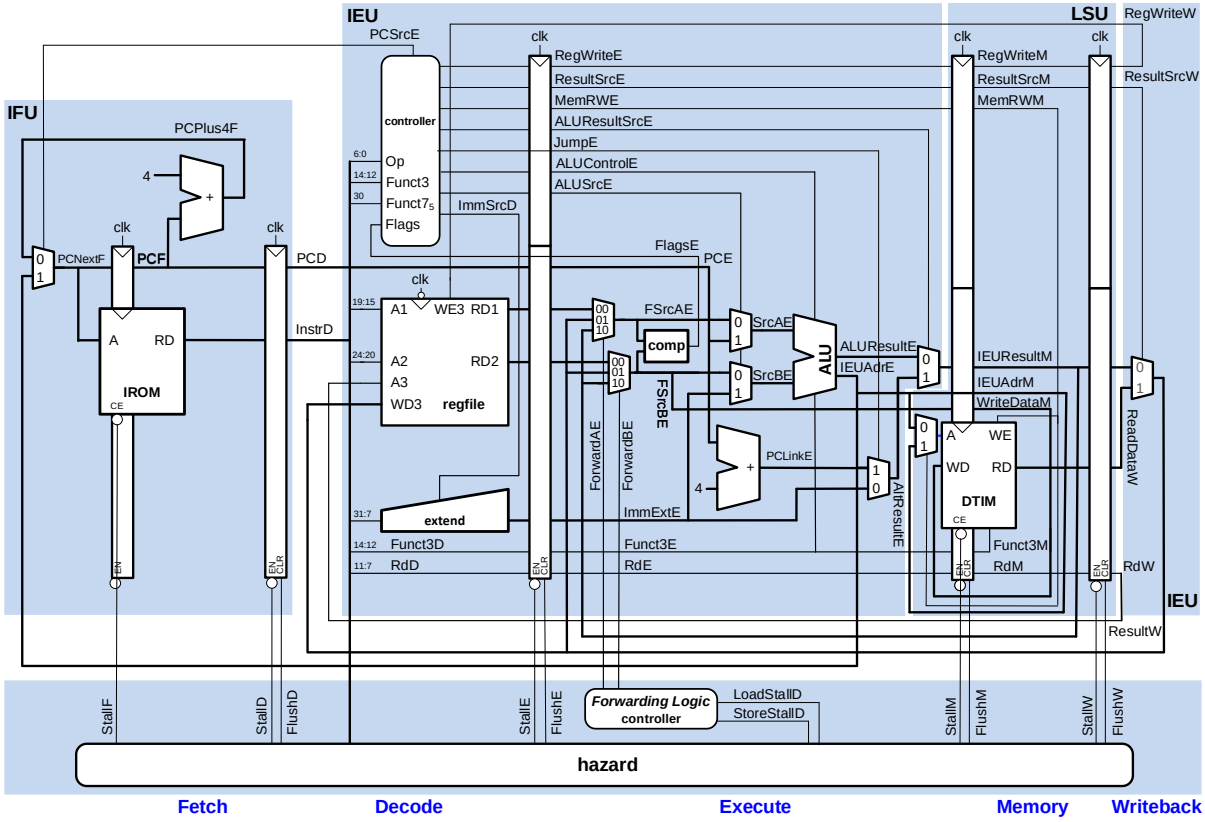

Figure 2. Wally Core

The Wally core is introduced in Chapter 7 and then expanded throughout the book to support additional extensions and features. For example, consider how the PC logic for computing the next PC (`PCNextF`) evolves throughout the book. The upper left of Figure 2 shows the simple PC logic: an adder for computing PC+4 (`PCPlus4F`) and a multiplexer to select between that and the branch or jump target address, `IEUAdrE`, produced by the ALU. As the book progresses and supports more extensions and features, this simple PC logic (adder and multiplexer) expands to the more complex logic shown in Figure 3, which supports branch prediction (`bpred`), traps (as managed by the privileged unit), fences, reset, stalls, and compressed instructions. The output is now called `PCPlus2or4F` to indicate that the logic supports both compressed (PC+2) or regular (PC+4) instructions.

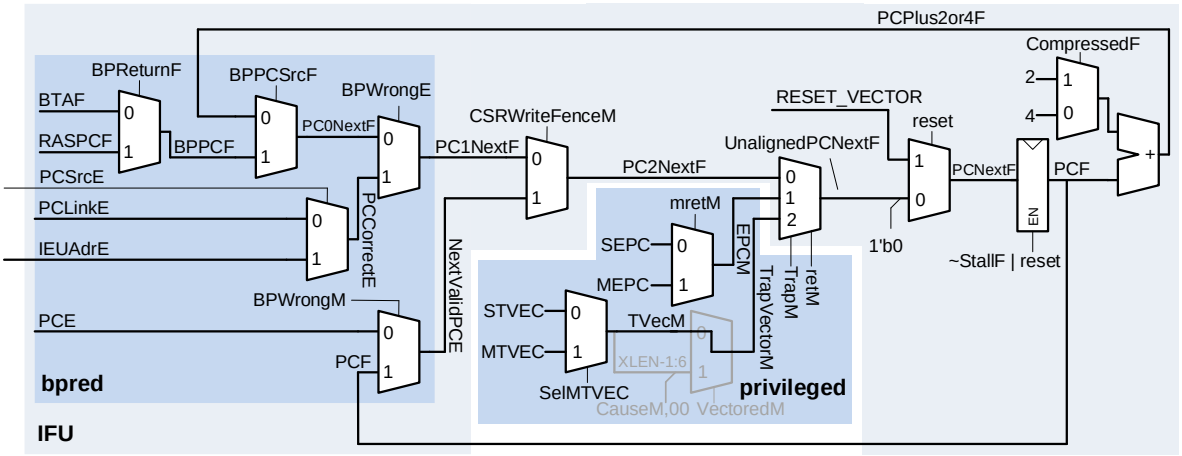

Figure 3. Expanded PC Logic

4. Courses

Over the past three years, two types of semester-long courses have been taught using drafts of the textbook and the Wally source code. The more advanced course, taught to seniors and master's-level students, covered the entire textbook and used all companion resources (lecture slides, labs, exercises, and the Wally SoC code) to teach computer architecture and SoC design, test, and verification. The lower-level course taught to sophomores used the textbook, the Wally SoC, and a subset of resources, focusing on computer architecture and processor design only.

We first focus on the senior/master's-level course, which was taught three times at Harvey Mudd College (Mudd) and twice at Oklahoma State University (OSU). The Mudd course was taught once per year from 2023-2025 to 13, 5, and 35 students in each course; the OSU course was taught once per year from 2023-2024 to 16 and 18 students. Table 4 shows the syllabus for this advanced course. The students were expected to have a basic understanding of digital design, HDLs – ideally SystemVerilog or Verilog, and computer architecture. For example, they were expected to understand the topics covered in [5].

Table 4. Course Syllabus

Week	Topic	Reading	Assignment
1	Introduction to RISC-V architecture	Ch. 1-2	
2	Programming Wally using Assembly; Linux; Git	Ch. 3, B, C	Lab 0: Tools
3	Programming Wally using C; HDL design practices	Ch. 4-5, D	Lab 1: Programming
4	Chip implementation; Single-cycle RISC-V processor	Ch. 6-7	Lab 2: Verification
5	Pipelined RISC-V processor; privileged unit	Ch. 7-8	Lab 3: Debug
6	Multiply & divide; FPU: FMA (FP multiply-accum.)	Ch. 15-16	Lab 4: Synthesis
7	FPU: fpdivsqrt (FP divide/square root)	Ch. 16	Lab 5: Design
8	Bus; Cache	Ch 9-10	Project: Overview
9	MMU; LSU	Ch. 11-12	Project: FP multiplies
10	IFU; compressed	Ch. 13-14	Project: FP add
11	Atomic; Bit manipulation; Peripherals	Ch. 17-20	Project: FP mult-add
12	Benchmarking; Linux	Ch. 21-22	Project: Special cases
13	FPGA Implementation	Ch. 23	Project: Rounding
14	RISC-V Market & Careers		Project: Report due

The instructor spent the first 7 weeks describing the Wally core while, at the same time, the students installed and used the tools and experimented with Wally using the labs. For the last 7 weeks, the instructor taught about the cache, the bus, peripherals, and extensions while students worked on their final project. In the syllabus above, students designed and implemented their own FMA (floating-point multiply/add) unit in Wally. Because of this course's final project topic, the floating-point lectures were moved earlier in the semester, to Weeks 6 and 7. Instructors may choose different topics for the final project. For example, during one of the Mudd courses, students focused on code coverage and verification for their final project. Many final project topics are possible, including: designing and implementing a new branch prediction strategy; expanding Wally to support another RISC-V extension; changing the cache replacement policy; adding custom instructions and measuring Wally's performance on target applications with and without the added instructions; and so on.

The main teaching strategy was to describe the computer architecture or SoC design concepts and theory using the textbook and lecture slides and then engaging the students in hands-on exercises and labs to explore and experiment with those concepts by simulating Wally, writing and running test programs, and also by modifying the Wally SoC. The student learning outcomes were for them to:

- Understand all units within a processor
- Understand how to build an SoC around a processor
- Be able to modify an SoC to extend its capabilities
- Analyze algorithmic complexity of digital logic and assess power, performance, and area for digital logic

We measured these student outcomes by the number of students who completed the final project, as a summative measure at the end of the course. All students at Mudd and OSU, 87 total students, successfully completed their final projects.

Student comments also qualitatively reflected how much they met the student learning outcomes. Many students stated how it was exciting to work on a current processor and to understand the fine details of its design. Some specific student comments were:

"*It was a challenge to work with so much code, but so rewarding to see Linux boot messages come across the screen.*"

"*Challenging labs, but the professor was dedicated to helping us learn the how the load/store unit worked.*"

"*Lots of work for the project but very cool and interesting. Homework is the right amount to help with the exams.*"

"*The course is great, and I have learned a ton! I also enjoy 'bit-banging' and figuring out how the computer parts work!*"

The lower-level course, which was taught at OSU in spring 2024 to 50 students, focused on chapters 7-13 to teach computer architecture and processor design. For this course, the students had not yet taken an introductory computer architecture course but had some experience with digital design. After learning computer architecture and processor design principles, the students used the labs to simulate the Wally SoC and completed some textbook exercises, but for their final project they only modified Wally's cache replacement policy. They implemented their designs on an FPGA and ran benchmarks before and after their changes to determine how performance was affected.

The main challenges of the two courses were:

- Installing and using the tools needed to work with Wally, especially for those unfamiliar with Linux.
- Synthesizing Wally targeted to an FPGA.
- Booting Linux, which has many failure points and can overwhelm students.
- Selecting which aspects to have the students complete hands-on because the field is so broad.

As a result, we have simplified the installation of multiple critical tools and added continuous integration into Wally's GitHub repository to proactively discover if an updated tool breaks installation across any supported OS. We have also clarified the steps for targeting Wally to an FPGA.

In summary, our textbook can be used to teach architecture, microarchitecture, SoC design, embedded systems, or a subset of these topics in theory, practice, or both. For lower-level classes, instructors may choose to use the book, or even a subset of the book, to focus on the theory, instead of delving into the implementation. By varying the topics covered, the number of exercises and labs used, and the inclusion or level of the project, the course can be adapted to a lower-level or advanced course.

5. Comparison with Existing RISC-V and SoC Textbooks

Many books on computer architecture exist as well as some on SoC design. *Computer Architecture: A Quantitative Approach* [6] is a seminal text in the field of computer architecture and covers similar topics as our textbook. However, it does not provide implementation details or show how all the pieces fit together. It also does not include an SoC to experiment with the principles described in the book. *The RISC-V Reader* [7] is an excellent synopsis of the RISC-V architecture and is a good companion to the RISC-V manuals [2, 3]; however, it focuses on the architecture only, not a complete SoC, and assumes some prior knowledge of theory. It also does not provide a fully functional processor or SoC. *Digital Design and Computer Architecture: RISC-V Edition* [5] describes both digital design and computer architecture but does not discuss SoC design. It also discusses and provides code for a processor, but one that implements only a subset of RISC-V instructions.

Several books on SoC design also exist, including [8-12]. *Architecting and Building High-Speed SoCs* focuses on integrating existing IP blocks to build an SoC on big FPGAs and developing software for the SoC rather than computer architecture and processor design [8]. *Modern System-*

on-Chip Design on ARM also only integrates IP blocks to build an SoC [9]. *Computer System Design* [10] and *System-on-Chip (SoC) Architecture* [11] focus on higher-level concepts and don't cover implementation. *Application-Specific Integrated Circuits* is a comprehensive but dated book on ASIC design, but it doesn't emphasize architecture [12].

None of these textbooks compete directly with ours because we describe all stages of SoC design from digital design to computer architecture to processor and SoC design, simulation, implementation, testing, and verification. We also accompany the textbook with the source code for the fully functional SoC that we describe in the book to give a ready platform for hands-on learning and expanding on the principles discussed in the textbook.

6. Conclusions and Future Directions

We have developed the Wally RISC-V SoC and an accompanying textbook that describes the theory and practice of processor and SoC design at all stages of design, simulation, verification, and implementation, starting at logic design and verification and continuing through computer architecture, SoC design, and implementation. The Wally SoC can run Linux and is configurable, from a small embedded core to a high-performance application processor. The textbook provides in-depth explanations as well as detailed diagrams and open-source code. We also include lecture slides, exercises, and labs and have used these to teach two types of courses: an advanced course that covers the entire textbook and a lower-level course that taught computer architecture and processor design only. Using the book, users are able to understand computer architecture and SoC design topics, including a detailed understanding of the theory and implementation of each building block needed to design an SoC. The open-source code for the fully functional and configurable Wally SoC provides a platform for hands-on learning and experimentation with the concepts taught in the textbook. In the future, we plan on expanding the textbook and Wally's capabilities to support more advanced computer architecture topics such as superscalar and out-of-order execution, multithreading, and multicore implementations. Because the Wally SoC is open source and is supported by detailed explanations of theory and implementation in the textbook, we also look forward to seeing how others use and expand on our work.

Acknowledgments
We would like to acknowledge the support and contributions of many collaborators and reviewers, including Krste Asanović, Andrea Bartolini, Allen Baum, Duncan Bees, Jeremy Bennett, Javier Bruguera, Daniel Chaver, Simon Davidmann, Ashish Dixit, Michael Engel, Andrea Gallo, Duncan Graham, Firas Hassan, Brian "Redbeard" Herrington, Pawan Kumar, Andreas Koch, Bill McSpadden, Lee Moore, Simon Moore, Robert Mullins, Torbjørn Ness, John Nestor, Rick O'Connor, David Patterson, Xiaoning Qi, Alex Solomatnikov, Dwight Sunada, Aimee Sutton, Mike Thompson, Andrew Waterman, Peiyi Zhao, and many of our students at Harvey Mudd College, Oklahoma State University, and the University of Nevada, Las Vegas.

References

[1] RISC-V International, https://riscv.org/. Accessed May 1, 2025.

[2] RISC-V International, "The RISC-V Instruction Set Manual, Volume I: Unprivileged Architecture." https://riscv.org/specifications/ratified/. Accessed May 1, 2025.

[3] RISC-V International, "The RISC-V Instruction Set Manual, Volume II: Privileged Architecture." https://riscv.org/specifications/ratified/. Accessed May 1, 2025.

[4] RISC-V International, "RISC-V Profiles." [Online]. Available: https://riscv.org/specifications/ratified/

[5] S. Harris and D. Harris. *Digital Design and Computer Architecture: RISC-V Edition*. Morgan Kaufmann, 2021.

[6] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed., Morgan Kaufmann, 2019.

[7] D. Patterson D, A. Waterman, *The RISC-V Reader: an Open Architecture Atlas*. Strawberry Canyon, 2017.

[8] M. Maaref, *Architecting and Building High-Speed SoCs: Design, Develop, and Debug Complex FPGA Based Systems-on-Chip*. Packt Publishing, 2022.

[9] D. Greaves. *Modern System-on-Chip Design on ARM*. Arm Education media, 2021.

[10] M. Flynn, W. Luk. *Computer System Design: System-on-Chip*. John Wiley & Sons, 2011.

[11] V. Chakravarthi and S. Koteshwar. *System-on-Chip (SoC) Architecture: A Practical Approach*. Springer Nature, 2023.

[12] M. Smith. *Application specific integrated circuits*. Addison-Wesley Professional, 1997.