RISC-V Microprocessor System-On-Chip Design Book Proposal

David Harris, James Stine, Sarah Harris
with contributions from Teo Ene, Ross Thompson, Kaveh Pezeshki, Noah Boorstin, Katherine Parry

Objectives & Audience

This book is intended for a graduate or undergraduate course or self-study. The reader will learn to design and optimize microprocessors and use them in a system-on-chip, applying contemporary design and verification tools. Much of the book describes detailed designs and tradeoffs for the functional blocks in and around a processor. There is presently no textbook on the market that describes microarchitecture in sufficient detail to fully implement a non-toy system. The reader should have had at least a first course in digital design and computer architecture, and experience programming in C in a Linux environment.

The book is associated with an open-source RISC-V microprocessor and system-on-chip project including SystemVerilog files, test suites, benchmarking, peripherals, Linux boot, and a design flow for an FPGA board and for the Skywater open-source fabrication process, targeting a 45 nm process.

Table of Contents

1. Introduction [DH]
    a. Overview
    b. A Brief History
    c. RISC-V Architecture [SH - Dec 1, 2022 or summer?]
        i. RV32/64/128
        ii. Integer Core Instructions
        iii. Extensions
            1. Single/Double Precision FP
            2. MUL
            3. Compressed
            4. Atomic
        iv. Privileged Operations

    *** include lots of quotes from RISC-V reader about simplicity
    *** include lessons from RISC-V reader including bits in address space, importance of lots of registers, no need for ARM shifter or condition codes, avoid branch delay slot, value of compressed instructions & sharing semantics exactly, placing multiply result in regular register, ability to move between integer and fp, etc. (look through book again)
    *** maybe compare with IA, ARM as done in RISC-V reader?

d. Simplified Single Cycle Processor Example [*** can we adapt anything from Borje's class at Berkeley?, see emails of June 20ish, 2021. **Or single cycle from DDCA**?]
    i. ISA
        1. Mention RV32/64, extensions
    ii. Microarchitecture
    iii. Verilog
    iv. Test Program
    v. FPGA Implementation
    vi. System-on-chip Implementation
        1. Performance, Power, Area (define FO4 delays for cycle times)

e. Wally Processor
    i. Configurable
    ii. Block Diagram
    iii. Top-Level Interface
    iv. Verification
f. Justification for both commercial and open-source tools and their implementations using System on Chip and FPGA design
    i. Moore's Law
        1. Need for accelerators
    ii. Abandonment of ITRS
    iii. Movement towards RISC-V and open-access design ecosystems
    iv. Need for new technologies and better collaboration between technologies (?)
    v. Create a new initiative that connects software and hardware foundations.
    vi. Design enablement for emerging technologies as well as education and workforce development.
    vii. Goal of open RISC-V processor that researchers can use to benchmark designs, fold new and better execution units into the processor

Part 1: Design Practices

2. Tool Flow [DH]
    i. Platform requirements
        1. Which version of Linux to use? (RedHat 8 / Rocky 8)
        2. RedHat8 is $27 per FTE for site subscription - academic pricing but no support. $1000 per year for virtualization).
    ii. riscv-wally Quick Start
        1. Check out repository
        2. Install tools
        3. Build test cases
        4. Simulate Verilog
        5. Boot Linux in QEMU and ModelSim
    iii. Hitchhiker's Guide to Linux [move to appendix]

1. x2go / some VNC?
2. Getting Around
    a. ls, cd, mv, cp, mkdir, rm, rm -rf *, chmod
3. Handy Commands
    a. grep
    b. find
4. Editing Files ?
    a. Visual Studio Code
5. Working with Git Repositories

iv. Editing Files
1. Visual Studio Code

v. Compiling, Assembling, and Disassembling Programs
1. Compiling, Assembling, and Linking
    a. C
    b. Assembly
        i. Register conventions (table B.4 from DDCA)
        ii. Pseudoinstructions
        iii. Assembler Directives
    c. Object Files (.elf)
    d. Linker (see RISC-V Reader)
    e. Compiler Flags
2. Disassembling
    a. objdump/binutils
    b. debug file (see riscv-arch-test)

vi. Simulating and Debugging Programs
1. Introduction to Simulation
2. ISA Simulators
    a. Spike - currently being used (10/20/21)
    b. QEMU
    c. Sail
    d. riscvOVPsimPlus
    e. Whisper?
3. Debuggers
    Simulation and Debug (how to run in Spike and/or QEMU) (not doing: OVPSim (because need to download every 30 days), Sail - need to evaluate - listed by RISC-V, may need to build to run RISC-V tests - David will look into)
    a. gdb (integration through VisualStudio)
    b. ddd (?)

vii. Execution Environment [DH: 12/31/21]
viii. *** see RISC-V spec
b. Bare Metal, OS, Hypervisors
c. Memory map: VM layout with text, data, heap, stack, kernel for Linux earlier

d. Privileged operation
    i. *** writing exception handlers/privilege change in C and Assembly
e. Compiling and Running Code for Wally Verilog Simulation *** maye in Verification?
    i. Imperas directories
    ii. Assembly and C
    iii. Startup Code
    iv. checking code

*** reference trap handler, supporting regular traps, interrupts
*** how to terminate a program in sims: ecall with gp=1?
*** how to get sims to work that have ecalls in them

3. HDL Design Practices [DH: 10/31/21]
    i. *** say Verilog is shorthand for SystemVerilog
b. Signal Naming
c. Configurable Designs
d. Synthesizable Designs
e. SystemVerilog Simulation
    f. Verilator
        i. Lint
    g. ModelSim
    h. https://riscv.org/wp-content/uploads/2015/01/riscv-software-stack-bootcamp-jan2015.pdf

F. Wally Verification
    i. Lint
    j. Test Suites
        i. riscv-arch-test
        ii. Imperas
        iii. wally-riscv-arch-test
        iv. Linux
    k. Regression

H. Chip Implementation Quick Start
    l. FPGA Flow
    m. Open Source SoC Flow
        i. Skywater 130 nm
        ii. Yosys/OpenRoad
    n. Commercial SoC Flow
        i. MUSE/TSMC 28 nm
        ii. Synopsys Design Compiler
        iii. Cadence Innovus
iv. PPA

Part 2: RISC-V Microarchitecture

*** add benchmarks to each section to show improvements from new features

4. RISC-V Pipelined Microarchitecture [DH: 12/31/21]
    a. Hart
        i. Integer Execution Unit
        ii. Instruction Fetch Unit
            1. how to deal with FSM that handles cache hits/misses, page table walking
            2. Dealing with two fetches on line wrap / spill
        iii. Load/Store Unit (only TIM and SWR/SWW, others grayed out)
        iv. IFU
        v. Other units (to discuss later)
        vi. Stalls and Flushes
            1. Should CSR accesses flush rather than stall?
    b. Uncore
5. Privileged Operations [DH 12/31/21]
6. AHB Lite Interface [Ross & DH 1/7/21]
    a. Support fast cache line fills - bursts?
    b. Adding bus in exclusion then in parallel with dtim/irom.
7. Caches [DH Intro, Ross write sections on testing and implementation target Mid-November
    a. Principles
    b. RISC-V Practices
        i. fence.i
    c. Test Plan
        i. TBD (maybe summer student)
    d. Wally Implementation
8. Memory Management Unit [DH] July21
    a. Discuss changes of having MMU
9. LSU [RT]
    a. MMU in diagram but grayed out
10. IFU [RT]
    a. Principles: Branch Prediction
        i. RISC-V branch prediction: resolved in execute, not decode
    b. Test Plan
        i. Specific to verifying branch prediction
    c. Wally Implementation
        i. Integrate cache
        ii. Integrate MMU
        iii. Integrate Branch Precition
        iv. compressed grayed out
11. Extensions: Compressed Instructions [DH 1/15/22]
    a. RISC-V Principles
        i. Uncompression
        ii. Spills

        iii. Interaction with cache
    b. Test plan
        i. spills
    c. Wally Implementation
12. Extensions: Multiplication and Division DH 10/31/21] Aug21
    a. Multiplication
        i. Carry-save addition
    b. Division
        i. By subtract and shift in non-redundant form, replicated for 2 bits per cycle
    c. Forward reference to Floating Point Division chapter for integer SRT division
13. Extensions: Floating Point [DH/JS late October James draft clean]
    a. Floating Point Representation
    b. Addition/Subtraction
    c. Multiplication
    d. FMA
    e. Division and Square Root by Digit Recurrence
    f. Division and Square Root by Iterative Approximation
    g. Other Operations
    h. Softfloat Test Suite (PARANOIA)
14. Extensions: Atomic ***
    a. LR/SC
    b. AMO operations
    c. using atomic operations for synchronization: locks, semaphores
        i. See Linux book Ch 1 for summary
        ii. Look at how Linux uses this
        iii. Lotteries
15. Peripherals [DH Spring]
        i. GPIO
        ii. CLINT
        iii. PLIC
        iv. UART
        v. SD Card [Ross]
        vi. SPI for 2nd edition?
        vii.
16. High Performance Microarchitecture [SH: 12/1/21 or summer] Adapt from Hennessy & Patterson?
    - Introduction of each processor (show figures & tables plus few paragraph overview)
    a. Superscalar
    b. Out-of-Order - talk about in more detail.
    c. SIMD
    d. Vector

Later:

e. Survey of Microarchitectures [SH, DH:]
    i. Various SiFive
        1. P650
    ii. SweRV
    iii. Other RISC-V
    Andes
    Alibaba
    https://github.com/T-head-Semi?spm=a2cl5.14290816.0.0.d3ef1ae6lip5dk

    survey of RISC-V implementations?

Part 3: Implementation

17. Benchmarking [DH, RT]
    a. CoreMark
        i. https://docs.google.com/document/d/1ZQA4TA-P2a-yO-jHqiQ7Lo-i-D80jkAEoHTtcHbdHZI/edit
        ii. (get steps from Abraham)
            1. How to download
            2. What has to change
            3. How to compile & run it

                cd tests/riscv-coremark/coremark
                make (builds and runs)
                cd ..
                ./transferobjdump.sh
                cd ../../wally-pipelined/regression
                vsim -c -do wally-coremark.do
                Notes: Western Digital says IPC is about 1.03
                Their CoreMark is good because of compiler flags.
    b. Embench - later - Skylar may work on this
            1. How to download
            2. What has to change
            3. How to compile & run it
            4. Look at
            5. https://riscv.org/wp-content/uploads/2019/12/12.10-12.50a-Code-Size-of-RISC-V-versus-ARM-using-the-Embench™-0.5-Benchmark-Suite-What-is-the-Cost-of-ISA-Simplicity.pdf
            6. https://github.com/embench/embench-iot-results/blob/master/details/eh2-rv32imc-gcc-10.2-o2.mediawiki
            7.
            8.
    c. Optimizations

18. Linux Boot [KP, NB]
    a. https://archive.fosdem.org/2020/schedule/event/riscv_bootflow/attachments/slides/4205/export/events/attachments/riscv_bootflow/slides/4205/FOSDEM_2020_Atish.pdf
    b. Boot Loader
    c. Device Tree
    d. Virtual Memory
    e. Privilege Modes
    f. Peripherals
19. FPGA Implementation [JS/SH/KP]
    a. Motivations
    b. FPGA Design Flow
    c. Bus Interface
    d. External Memories
    e. Optimization
20. CMOS for Microarchitects [DH]
    a. CMOS Transistors
    b. CMOS Layout
    c. Example: Skywater 45 (or 22?) nm Process
    d. Cell Design
    e. Floorplanning
    f. Performance
        i. RC Delay Model
        ii. Logical Effort
        iii. Wire Delay
        iv. Repeaters
    g. Power
        i. Dynamic Power
        ii. Static Power
    h. Area
    i. Building Blocks
    j. Memories
    k. Phase Locked Loops
    l. Scan Registers
    m. System-on-Chip Design
        i. Moore's Law
        ii. Scaling
        iii. Wires
        iv. CAD Tools
        v. Economics
            1. [***maybe get estimate from David Garrett about costs for startup]
        vi. IP Reuse
        vii. Dark Silicon (IEEE UDP)

21. CMOS Implementation [TE]
    a. Motivations
    b. CMOS Design Flow
        i. Standard cells
        ii. Memories
        iii. I/O
    c. Low Power Design
    d. Design for Test
    e. Power, Performance, and Area Optimization
        i. See report from Shreya nd Kip in slack synth channel Dec 27 2021
    f. Low Power Methodology
    g. Packaging
    h. Commercial RISC-V Implementations
        i. SiFive
        ii. SWERVE

Second Edition

22. RISC-V Superscalar Microarchitecture [DH with BB] [2nd edition?]
23. RISC-V Out-Of-Order Microarchitecture [Ross?] [2nd edition?]
24. RISC-V Threaded Microarchitecture [JS] [2nd edition?]
25. More Bus Interfaces [SH, JS] [2nd edition?]
    a. APB
    b. AXI
    c. Wishbone
    d. Network on Chip
26. Debug Interface [2nd edition]
27. Multicore [SH 2nd Edition]
    a. Cache Coherence
    b. Synchronization: Atomic Instructions
    c. Bus Arbitration
28. SIMD [2nd edition]
29. Vector [2nd edition]
30. Bit Manipulation [2nd edition]
31. Crypto (K) [2nd edition]
32. Security [2nd edition]
    a. Hardware features for security
    b. Side channel attacks

Peripherals

        i. SPI
        ii. PWM?

33. Silicon Debug [?]

Supplements

Tool installation: gcc, ovp, lint (verilator), modelsim, Synopsys & Cadence
Imperas tests, git repo

OVP: Regarding riscvOVPsimPlus.exe, go to and choose version
https://www.ovpworld.org/library/wikka.php?wakka=riscvOVPsimPlus
Or switch to free simulator

Other Notes

The authors would like the book to be available in printed format and in permanent (non-rental) ebook format at reasonable prices for students.

The length is still TBD but expected to be 400-800 pages. Target delivery September 2022.

An external website and open-source tools will be provided to accompany the text.

We plan to produce the manuscript in Google Docs and produce most figures in Visio and most equations in MathType.

Author Bios

David Harris is the Harvey S. Mudd Professor of Engineering and Associate Engineering Chair at Harvey Mudd College. He received his S.B. and M. Eng. degrees in Mathematics and Electrical Engineering/Computer Science from MIT and his Ph.D. from Stanford University. His research interests focus on digital systems. He is the author of *Digital Design and Computer Architecture*, *CMOS VLSI Design*, *Logical Effort*, *Skew-Tolerant Circuit Design,* as well as various Southern California hiking guidebooks. He has worked as a microprocessor designer at Intel, Sun Microsystems, Hewlett-Packard, Broadcom, and elsewhere. He holds more than a dozen patents in the field.

James Stine is the Edward Joullian Professor of Engineering at Oklahoma State University. His area of research is in computer arithmetic, memory architectures, and Electronic Design Automation (EDA) design flow. He is author of numerous articles on optimization of architectures for use with computer arithmetic as well as interfacing to memory architectures. He is author of three texts *Digital Datapath Computer Arithmetic with Verilog*, A*dder Architectures for VLSI Implementations* and *System on Chip Design Flow and Standard-Cell Library*.

Sarah Harris is an Associate Professor of Electrical and Computer Engineering at the University of Nevada, Las Vegas. She graduated with her PhD from Stanford University in 2005 and is the author of the *Digital Design and Computer Architecture* group of textbooks. She has also worked at Hewlett Packard, Nvidia, and the Technical University of Darmstadt and has collaborated with other companies including Southwest Research Institute, Intel, and Imagination Technologies. Her research areas include computer architecture, reconfigurable

computing, and applications of embedded systems and machine learning to biomedical engineering and robotics.

Writing Notes

Look at **https://inst.eecs.berkeley.edu/~cs152/sp21/** for ideas, lots of images that would make good sidebars.