Teaching Digital Logic and Computer Architecture Using Open Source Tools

Bill Siever
Michael Hall
James Feher
Roger Chamberlain

Bill Siever, Michael Hall, James Feher, and Roger Chamberlain, “Teaching Digital Logic and Computer Architecture Using Open Source Tools,” in *Proc. of 22nd ACM International Conference on Computing Frontiers Workshops and Special Sessions*, May 2025, pp. 53-56.
DOI: 10.1145/3706594.3726971

Presented at 3rd Open Source Hardware Workshop (OSHW), Cagliari, Sardinia, Italy.

Dept. of Computer Science and Engineering
and
Dept. of Electrical and Systems Engineering
Washington University in St. Louis

Teaching Digital Logic and Computer Architecture Using Open Source Tools

Bill Siever
Washington University in St. Louis
St. Louis, Missouri, USA
bsiever@gmail.com

Michael Hall
Washington University in St. Louis
St. Louis, Missouri, USA
mhall24@wustl.edu

James Feher
Washington University in St. Louis
St. Louis, Missouri, USA
jdfeher@wustl.edu

Roger Chamberlain
Washington University in St. Louis
St. Louis, Missouri, USA
roger@wustl.edu

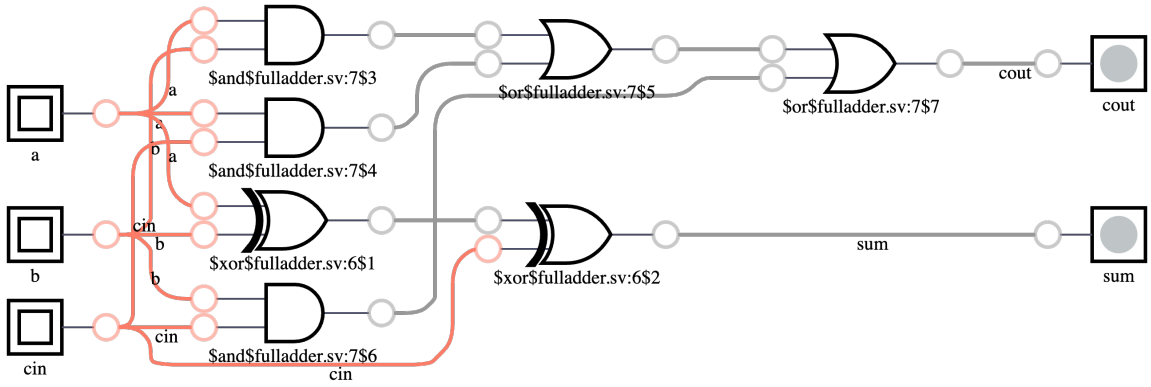

Figure 1: Full adder visual simulation.

Abstract

A recent redesign of the digital logic and computer architecture courses at Washington University in St. Louis exploits developer containers and open source tools to provide students with a low-cost, portable tool suite for hardware design. Here, we describe the tool flow used by the students and articulate the motivation for and benefits from utilizing this approach.

CCS Concepts

• **Hardware → Electronic design automation**; • **Software and its engineering → Open source model**; • **Social and professional topics → Computer engineering education**.

Keywords

Digital Logic, Computer Architecture, Field Programmable Gate Arrays, FPGAs, Developer Containers, Dev Containers, Visual Studio Code, VS Code, RISC-V

CF Companion '25, Cagliari, Italy

ACM ISBN 979-8-4007-1393-4/25/05
https://doi.org/10.1145/3706594.3726971

ACM Reference Format:
Bill Siever, Michael Hall, James Feher, and Roger Chamberlain. 2025. Teaching Digital Logic and Computer Architecture Using Open Source Tools. In *22nd ACM International Conference on Computing Frontiers (CF Companion '25), May 28–30, 2025, Cagliari, Italy*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3706594.3726971

1 Introduction

In the fall semester of 2024, the McKelvey School of Engineering at Washington University in St. Louis engaged in a refresh of two of the fundamental courses in the BS Computer Engineering program: the introduction to digital logic course and the computer architecture course. As part of that refresh, we decided to make a serious attempt at using fully open source tools in the delivery of the course material. The resulting tool suite was the subject of a demo at this year's Technical Symposium on Computer Science Education [16].

Both of these courses are required for computer engineering students, digital logic (a sophomore level course) is required for electrical engineering students and is an optional technical elective for computer science students, while computer architecture (typically taken in the junior year) is an optional technical elective for both electrical engineering and computer science students. Digital logic is taught both fall and spring (averaging just over 80 students per semester this past year) and computer architecture is taught in the fall (with 25 students in the first offering since the refresh). As part of the refresh effort, each of the offerings of both courses have

had two instructors assigned, all of whom are co-authors on this paper. Both the Department of Computer Science and Engineering and the Department of Electrical and Systems Engineering share curricular responsibility for the computer engineering program and the contents of both courses.

In this paper, we describe the tool flow used by the students to perform digital hardware design from design entry through simulation, synthesis, place & route, to deployment on an FPGA. Almost 100% of the infrastructure used to support this tool flow comes from the open source community, and the number of distinct tools is considerable. Detailed usage instructions are provided at [15].

2 Past Practice and Present Motivation

Both of the two courses are fairly traditional in the topics covered and their overall scope. The digital logic course covers binary number systems, Boolean algebra, combinational and sequential system design, an introduction to a simple instruction set architecture, and a basic fetch-execute engine. The computer architecture course covers single-cycle, multi-cycle, and pipelined implementations of an instruction set architecture in addition to topics in memory subsystems (including virtual memory), I/O subsystems, and cache coherence protocols.

Historically, these courses used VHDL as the hardware description language and the Xilinx proprietary FPGA tool flow for design and implementation of digital designs. A fresh redesign of the two courses was triggered by 3 concurrent events: (1) a long-serving member of the faculty who had taught both courses for a number of years retired, (2) the textbook used in the computer architecture course went out of print, and (3) the FPGA development boards (used in both courses) were discontinued by the manufacturer.

As part of the redesign, an early decision was to go in the direction of open source tools, which motivated a number of the follow-on decisions. The redesign team committed early to the use of the RISC-V instruction set architecture (given that both the ISA and any number of available designs are open source). Given the proliferation of RISC-V designs in SystemVerilog, and its support in the open source community, switching away from VHDL was deemed appropriate. The above choices led us to the Harris and Harris text, *Digital Design and Computer Architecture: RISC-V Edition* [4], which we are now using in both courses (supplementing with some additional material in the computer architecture course).

The above decisions made, we were next challenged to put together a tool flow guided by the following goals. First, *authenticity*, have students use contemporary tools and deploy their designs on real hardware. Second, *cost*, use low-cost readily available hardware. Third, *complexity*, keep the installation and tool use overheads low. Fourth, *focus*, ensure the students are using their available cognitive load on the subject matter of the course, not on the distractions inherent in fully-featured commercial tools. Finally, *portability*, support the variety of execution platforms used by students, specifically including Linux, Windows, and MacOS. A number of commercial tools are limited in the platforms they support.

The Harris and Harris text (and accompanying resource materials) support the Intel Quartus development environment, and we explored the use of that tool suite, but ultimately decided that we wanted to go the open source route, driven by the above goals.

3 Tool Flow

Here we articulate the specific tools used in the design flow by the students, including containerization, design entry, HDL simulation, synthesis, place & route, FPGA deployment, ISA-level simulation, and high-level language compilation. The set of tools represents a broad spectrum of open source contributions by the community from around the world. The tool flow and hardware used rely heavily on ideas from Bell [1].

3.1 Containers

The use of development containers (or dev containers for short) are central to the utility of the tool suite we have put together. One of the real challenges associated with supporting multiple execution platforms (e.g., Linux, Windows, MacOS) is the complexity associated with deploying a broad set of tools on each platform. Koskinen et al. [7] describe how containers are used throughout software engineering, and Pahl et al. [10] review the use of containers in the cloud.

By deploying each of the tools described below in a dev container, there are three options for students to utilize the tool flow.

(1) The entire tool flow can be executed in the cloud, with the development environment accessed via a web browser.
(2) The entire tool flow can be executed natively on the student's local machine. As stated above, all of Linux, Windows, and MacOS are supported.
(3) A blended execution option is available, in which the development environment is run locally, yet the individual tools are executed in the cloud.

The above options provide maximum flexibility for students, enabling the tools to be used in a wide variety of contexts.

3.2 Design Entry and HDL Simulation

Early in the digital logic course, students use schematic capture for design entry. They then move to the use of hardware description languages, specifically SystemVerilog. The computer architecture course explicitly uses SystemVerilog for design entry.

The schematic capture and simulation tool used is JLS: Java Logic Simulator. Originally authored by Poplawski [11], it has been updated and binaries are provided for Windows and MacOS, as well as a generic JAR for all platforms [14]. Students use JLS for schematic-level designs of both combinational and sequential circuits. They can test the behavior and correctness of their work by applying test vectors. Moreover, JLS can be used for automated unit tests on student work.

Once development moves to SystemVerilog, the students use an IDE, working extensively in VS Code with HDL support for syntax highlighting, etc., by Hiramori [5]. Linting and simulation is performed using Verilator [17] and/or Icarus Verilog [23].

An important component in the use of hardware description languages is the utility of generating testbenches. Currently testbench generation is all done via SystemVerilog; however, we are investigating the potential use of cocotb [2] to enable the development of testbenches using Python.

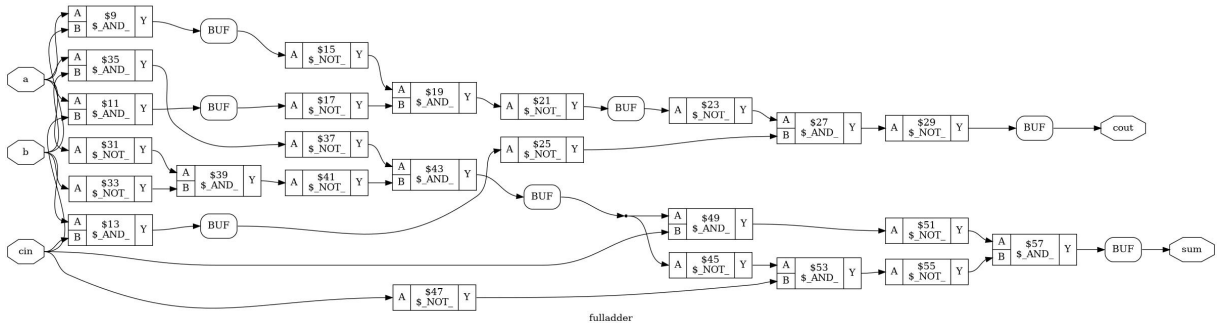

Figure 2: Synthesized full adder.

3.3 Synthesis and Place & Route

Synthesis and place & route use Yosys + nextpnr [13, 24]. Once synthesized, DigitalJS [8] is used for visual interactive simulation (see Figure 1). By clicking on inputs, students can see the effects of the input changing through the circuit.

Figure 2 shows the synthesized full adder. Note that this represents a different implementation relative to the full adder of Figure 1, using different primitive gates.

3.4 Deployment to an FPGA

The physical FPGA board that we use is the open source UPduino platform [20]. The specific FPGA is a Lattice Semiconductor UltraPlus ICE40UP5K. The board, shown in Figure 3, features an on-board FPGA programmer, flash and LED with all 32 GPIO pins on 0.1" headers.

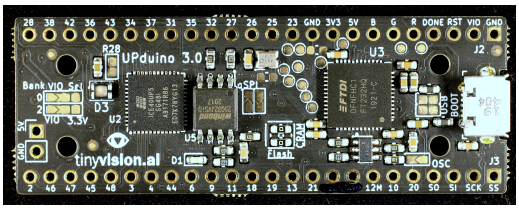

Figure 3: UPduino Lattice UltraPlus iCE40 FPGA board [20]. Image credit: Gregory Benjamin.

Once a bitstream file has been created, it is loaded onto the UPduino using openFPGALoader [3], which has native installs for Linux, Windows, and MacOS. When the development environment is running in the cloud, YoWASP [26] enables loading from the browser through the use of WebAssembly.

Initially, the students construct simple digital I/O features (e.g., switch inputs, LED outputs) to exercise their deployed bitstreams. Figure 4 is an example from an assignment in the digital logic course. Subsequently, they graduate to a more full-featured I/O board (see Figure 5), that contains 7-segment LEDs and multiple buttons. It interfaces to the FPGA using the TM1638 driver chip via a synchronous serial bus. A locally developed driver (available at [15]) provides digital input and output ports for a design on the FPGA and transitions those ports to the physical inputs and outputs on the I/O board.

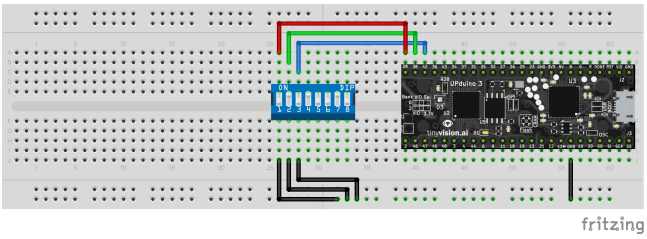

Figure 4: Wiring diagram for switch inputs (drawn using the fritzing [6] package).

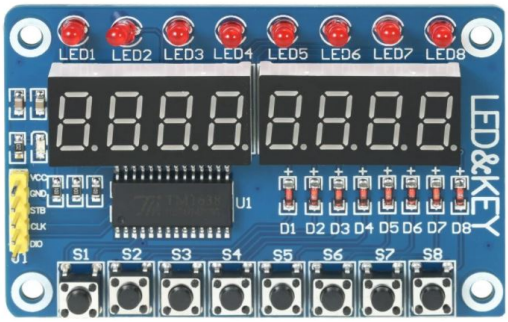

Figure 5: Digital I/O board.

3.5 RISC-V Tools

The digital logic course implements a subset of the RISC-V ISA on the FPGA, while the computer architecture course both expands this subset and includes the design and implementation of a cache (since the UPduino does not have an external memory, both main memory and cache are deployed in on-chip memory resources). In both cases, the students need to generate RISC-V instructions to execute on their constructed processor(s).

When studying the ISA and its associated assembly language, the students utilize the Venus RISC-V instruction set simulator [21], originally developed by Keyhan Vakil and more recently maintained by Steven K. (aka ThaumicMekanism). It has been integrated into VS Code [22], facilitating ease of use for the students. RISC-V assembly support for syntax highlighting is provided by Sun [19].

For programs written in a high-level language, compilation from C and/or C++ is supported by the GNU RISC-V compiler [12].

4 Discussion

The motivation for this work includes authenticity of the experience, cost to students, complexity of the tools, supporting a focus on the task at hand, and portability across multiple platforms.

- *Authenticity*. All of the tools used here are effective at digital system design and are used by a wide community, including many commercial firms. Even though some tools are designed explicitly as teaching tools, they still perform the functions appropriate to the task at hand. The result is a learning experience for students that is effective at giving them an authentic design experience.
- *Cost*. The only cost incurred by students is the physical hardware, as almost all of the software is open source and the rest is free to use. For example, if executing the tools in the cloud, the cloud provider, Microsoft, provides a modest time budget to students at no charge. In addition, the hardware is quite inexpensive, with the UPduino costing under $40 and the I/O board available for under $10.
- *Complexity*. The installation and tool use overheads are held to a minimum through the effective use of containers, which are designed for that very purpose.
- *Focus*. While our desire is to put together a tool suite that does what needs to be done, but not lots of other things to distract the students, this is inherently a difficult task. The current collection of tools is somewhat more organized (i.e., separate tools for separate tasks). However, the inherent differences that are present in the tasks to be performed (and the fact that they were developed by a diverse set of individuals) imply that there are variations in user interfaces that can be confusing. In our opinion, this is the one area where we were not uniformly satisfied with the end result, but we came away with a better understanding of its inherent challenges, and we also believe the result is an improvement on commercial tools.
- *Portability*. With containerization, the entire tool flow can run natively on Linux, Windows, MacOS, or the cloud. This is considerably broader coverage than is typically supported by many of the commercial tools.

5 Conclusions and Future Work

The result of this effort is a suite of FPGA development tools that are multi-platform (and therefore portable), inexpensive (thirteen of which are open source, all of which are free to use), and effective for the task at hand. These features enable their use in a broad set of contexts, ranging from large schools with plenty of instructional support, to small schools where the instructor is also responsible for tools used in the classroom. See [15] for detailed usage instructions and pointers to all the tools.

There are a number of directions we would like to expand this effort. We have already mentioned the desire to enable testbench development in Python, which is supported by cocotb [2]. In addition, the ability to observe internal signals in hardware via in-circuit debugging support can be extremely helpful, a capability supported by Manta [9]. Finally, high-level synthesis compilers have recently gotten to the point that they are effective in many contexts, e.g., spacecraft [18]. Our group is exploring open source HLS compilers such as ScaleHLS [25] in this area, and it would be of interest to see these tools also incorporated into the undergraduate computer engineering curriculum.

References

[1] Steven Bell. 2023. Reimagining the digital lab with $30 FPGAs. In *Proc. of ASEE Annual Conference & Exposition*. 16 pages.

[2] Cocotb. Accessed Feb. 2025. COroutine based COsimulation TestBench. https://www.cocotb.org/.

[3] Gwenhael Goavec-Merou et al. Accessed Feb. 2025. openFPGALoader: universal utility for loading FPGA. https://trabucayre.github.io/openFPGALoader/.

[4] Sarah Harris and David Harris. 2022. *Digital Design and Computer Architecture, RISC-V Edition*. Morgan Kaufmann.

[5] Masahiro Hiramori. Accessed Feb. 2025. Verilog-HDL/SystemVerilog/Bluespec SystemVerilog support for VS Code. https://marketplace.visualstudio.com/items?itemName=mshr-h.VerilogHDL.

[6] André Knörig, Reto Wettach, and Jonathan Cohen. 2009. Fritzing: a tool for advancing electronic prototyping for designers. In *Proc. of 3rd International Conference on Tangible and Embedded Interaction*. 351–358.

[7] Mikael Koskinen, Tommi Mikkonen, and Pekka Abrahamsson. 2019. Containers in software development: A systematic mapping study. In *Proc. of International Conference on Product-Focused Software Process Improvement*. Springer, 176–191.

[8] Marek Materzok. 2019. DigitalJS: A visual Verilog simulator for teaching. In *Proc. of 8th Computer Science Education Research Conference*. ACM, 110–115.

[9] Fischer Moseley. Accessed Feb. 2025. Manta: A Configurable and Approachable Tool for FPGA Debugging and Rapid Prototyping. https://github.com/fischermoseley/manta.

[10] Claus Pahl, Antonio Brogi, Jacopo Soldani, and Pooyan Jamshidi. 2017. Cloud container technologies: a state-of-the-art review. *IEEE Transactions on Cloud Computing* 7, 3 (2017), 677–692.

[11] David A Poplawski. 2007. A pedagogically targeted logic design and simulation tool. In *Proc. of Workshop on Computer Architecture Education*. 1–7.

[12] RISC-V Collaboration. Accessed Feb. 2025. RISC-V GNU Compiler Toolchain. https://github.com/riscv-collab/riscv-gnu-toolchain/.

[13] David Shah, Eddie Hung, Clifford Wolf, Serge Bazanski, Dan Gisselquist, and Miodrag Milanovic. 2019. Yosys+ nextpnr: an open source framework from Verilog to bitstream for commercial FPGAs. In *Proc. of 27th International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 1–4.

[14] Bill Siever. Accessed Feb. 2025. JLS: Java Logic Simulator. https://github.com/bsiever/JLS/.

[15] Bill Siever. Accessed Mar. 2025. Tools for Digital Logic and Computer Design. https://github.com/digital-logic-and-computer-design/upduino-devcontainer.

[16] Bill Siever, Michael Hall, Jim Feher, and Roger Chamberlain. 2025. Digital Logic, Computer Architecture, and Dev Containers: Supporting Schools from Little to Large. In *Proc. of 56th ACM Technical Symposium on Computer Science Education*, Vol. 2. 1737.

[17] Wilson Snyder. 2018. Verilator 4.0: open simulation goes multithreaded. In *Open Source Digital Design Conference (ORConf)*.

[18] Marion Sudvarg, Chenfeng Zhao, Ye Htet, Meagan Konst, Thomas Lang, Nick Song, Roger D. Chamberlain, Jeremy Buhler, and James H. Buckley. 2024. HLS Taking Flight: Toward Using High-Level Synthesis Techniques in a Space-Borne Instrument. In *Proc. of 21st Int'l Conference on Computing Frontiers*. ACM, 11 pages.

[19] Shao-Ce Sun. Accessed Feb. 2025. Most Comprehensive RISC-V ASM Highlighting. https://marketplace.visualstudio.com/items?itemName=sunshaoce.RISC-V.

[20] UPduino. Accessed Feb. 2025. UPduino-v3.0 and 3.1. https://github.com/digital-logic-and-computer-design/UPduino-v3.0/.

[21] Venus. Accessed Feb. 2025. Venus RISC-V instruction set simulator built for education. https://github.com/ThaumicMekanism/venus.

[22] Venus – VS Code. Accessed Feb. 2025. RISC-V Venus Simulator embedded in VS Code. https://github.com/hm-riscv/vscode-riscv-venus.

[23] Stephen Williams. Accessed Feb. 2025. The ICARUS Verilog Compilation System. https://github.com/steveicarus/iverilog.

[24] Clifford Wolf, Johann Glaser, and Johannes Kepler. 2013. Yosys-a free Verilog synthesis suite. In *Proc. of 21st Austrian Workshop on Microelectronics (Austrochip)*. 6 pages.

[25] Hanchen Ye, HyeGang Jun, Hyunmin Jeong, Stephen Neuendorffer, and Deming Chen. 2022. ScaleHLS: A scalable high-level synthesis framework with multi-level transformations and optimizations. In *Proc. of 59th ACM/IEEE Design Automation Conference*. ACM, 1355–1358.

[26] YoWASP. Accessed Feb. 2025. Unofficial WebAssembly-based packages for Yosys, nextpnr, and more. https://yowasp.org/.