Framework and Tools for Undergraduates Designing RISC-V Processors on an FPGA in Computer Architecture Education

Peter Jamieson, Tyler McGrew, and Eric Schonauer
Electrical and Computer Engineering
Miami University
jamiespa@miamioh.edu

***Abstract*—Arguably, each computer engineer undergrad should build a simple processor in the pursuit of their degree to help them internalize the basic design principles and properties of a computer. With the proliferation of FPGAs in universities this is, easily, realizable in most undergraduate curricula. Many modern courses on computer architecture or organization rely on MIPS architectures (among others) as the base processor to learn with, but the MIPS architecture has little commercial success and real-world implementations that will allow students to get additional career benefit from building and learning about a used architecture. The increasing industrial interest of RISC-V ISA, its free availability, and its early success in real-world adoption makes this processor a great potential candidate in this educational space. This work provides suggestions on how undergraduates should build a RISC-V architecture on an FPGA, and a basic framework of tools and design principles for this exercise.**

I. INTRODUCTION

Undergraduate computer engineering curriculum [1] includes understanding the design and operation of a processor as well as the tools, software, and optimizations that relate to it. The details of understanding how a processor works is achieved in the suggested curriculum with 10 hours of study on computer organization and 10 hours of study with the instruction set architecture (ISA). This understanding can be achieved in a number of ways, but there is a popular belief that a future computer engineer should at least build one processor in their undergraduate education.

At Miami University, we do not force undergraduate computer engineers to build a processor, but we do suggest the value of such an exercise. However, our computer organization course is structured to allow students to pursue their interests via badge-based learning [2]. Students that select to build a processor can also choose what type of processor to build. In the past, students have built PIC [3], AVR [4], and MIPS [5] processors where the MIPS processor is the most popular choice since they, typically, start by learning the MIPS ISA for a basic understanding of assembly programming. With the invention of the RISC-V [6] processor and its potential adoption by a range of corporations due to the freely available instruction set makes it a great candidate processor for undergraduates to design on an FPGA. The main benefit of building a RISC-V processor is that a computer engineer gets the dual benefit of building a processor, and getting a better understanding of a machine that is used in the real-world including major industries.

In this work, we describe the tools and design choices used to create RISC-V processors on an FPGA. This includes the tools used to simulate the RISC-V programs, the tools used to create the processor on the FPGA, and the design choices made to make the design tractable for undergraduates. We provide links to two processors created in our course, and describe additional extensions that were done by these students to get advanced badges in the course. The hope of this work is to provide computer engineering educators and students with an idea on how to approach the base design of a RISC-V processor.

The remainder of this paper is organized as follows: Section II provides a brief description of previous work in this area. Section III describes the set of tools and the design choices made to create the exemplar architectures. Section IV describes are two implementations and extensions added to these processors. Finally, section V provides discussion and concludes the paper.

II. BACKGROUND - COMPUTER ARCHITECTURE IN COMPUTER ENGINEERING EDUCATION

A typical undergraduate course in computer architecture will use textbooks such as Patt and Patel's, "Introduction to Computing Systems - from bits & gates to C & beyond" [7] and Patterson and Hennessy's, "Computer Organization and Design - The Hardware/Software Interface" [8]. Both books are bottom up approaches to learning about computer architecture that start with an introduction or understanding of digital components that make up the system, and then continues with assembly language, moving to more and more complex topics in computer architecture until the student begins to understand how a high-level program, written in the C language [9], is mapped to a processor and executed. Another popular textbook and framework, by Nisan and Shockan [10], promotes a project based approach to systems from low-level gates all the way to a video game running on a system.

For decades, it has been debated among educators what and how to teach computer architecture and organization [11]. For example, one of the first questions in these courses is what architecture to use as the exemplar system [12]. Each of the above textbooks uses a different architecture (LC-2, MIPS, and Hack, respectively), and these 3 choices are just a small sample of other possibilities, which include ARM [13], PIC [14], and x86 [15]. With the choice of an architecture,

the next question is how to study the system with possibilities including assembly programming using simulators [16] [17] [18], FPGA implementations of processors [19] [20], or thought experiments of processor function [21]. After making the above two decisions, the approach to what students do in a course can vary from lectures, assignments, exams, projects, and other activities. For example, project-based learning [22] and experiential learning [23] has had a recent revival as universities look to student-centered approaches to learning - computer architecture is no exception to these trends [24] [2].

This paper examines how students can use the RISC-V as the modern processor to implement on an FPGA. The reasons for considering another processor in the already heavily populated educational processor space includes:

- A RISC processor similar to MIPS
- A free and open ISA
- A modern processor being adopted in industry in academia
- A suite of tools for simulation and compilation
- A easily supported software stack

III. Tools and Design Choices for RISC-V Processors

A. Simulator and Assembler

The first tools needed to understand an architecture is a simulator and assembler. In many cases, the simulator and assembler are included in the same software tool. For example, MARS [25] simulator is a popular simulation tool for MIPS that includes simulation and assembling of MIPS programs among other tools for educational purposes.

For RISC-V, there are a number of choices available on the RISC-V site (riscv.org/software-status/). The one which we use in our cases was designed by Morten B. Petersen called "Ripes" and available at github.com/mortbopet/Ripes as of the writing of this paper. Ripes is a simple tool that simulates a 5-stage pipelined RISC-V machine (not necessarily helpful for the actual design on the FPGA) and is easy to use to get machine encoded data for the assembly programs.

Other software tools such as a cross-compiler version of "gcc" and "gdb" are available to take high-level code and compile and debug the RISC-V architecture. These tools, though useful, are not necessary for the FPGA implementation and testing described in this work.

B. FPGA tools

The tools we used for our FPGA implementations of a RISC-V processor are all Intel based. Note, however, any FPGA toolset can be used, and we will name the generic tool name and industrial tool name so that others can use similar tools. Also, our hardware implementation language is Verilog [26], which is a Hardware Description Language. This exercise can also be done in other HDLs, such as VHDL, in schematics, or in higher level hardware design languages. We, however, recommend using an HDL as a good middle ground for this where, hopefully, students have a good understanding of the circuits generated by their "synthesizable" HDL design.

The main tool in the Intel flow is Quartus which takes in a design and synthesizes for the Intel FPGAs. Our labs contain DE2-115 boards from Terasic, and these boards include a Cyclone FPGA. Quartus includes an FPGA programmer that allows the design to be synthesized and programmed to a real FPGA.

The Quartus tool can be setup to use ModelSim that allows the simulation of digital design of the processor. A user program is loaded with the bits that are used to configure the design on the FPGA (we describe the details of this later). Similarly, a tool called "SignalTap" allows the actual programmed processor on the FPGA to be analyzed where "SignalTap" is an on-chip logic analyzer. Both simulation and the logic analyzer are used to debug the design. Also, with the simulator alone, in theory, a student could complete most of this exercise and prove that their processor works without needing a physical FPGA. We, however, require a real FPGA programming to emphasize other aspects of real engineering including a prototpye board with limited clock frequency and silicon area as real constraints for a practicing engineer.

C. RISC-V Architecture Constraints

The RISC-V version 2.2 documentation has a frozen base ISA [6] that has to variants - 32 bit or 64 bit. For this work, the RV32I is the base ISA that students should start from, but the best starting definition is the Tiny RISC-V Instruction set architecture from Cornell (www.csl.cornell.edu/courses/ece5745/handouts/ece5745-tinyrv-isa.txt); the TinyRV2 within that document is a nice basic processor that can run simple C programs and includes 34 instructions in the following instruction categories:

- Control/Status Register Instructions
- Register-Register Arithmetic Instructions
- Register-Immediate Arithmetic Instructions
- Memory Instructions
- Unconditional Jump Instructions
- Conditional Branch Instructions

D. FPGA Implementation Details

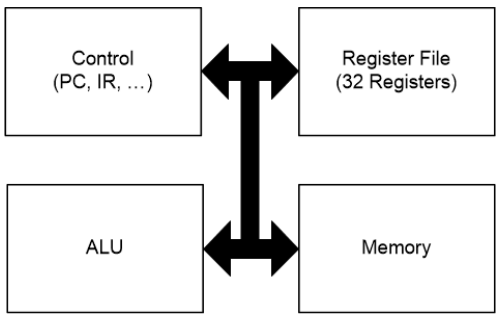

Fig. 1. A simple block diagram of the pieces of the RISC-V architecture

Figure 1 shows a basic block diagram of the major components of the RISC-V system. Note, that the details of the architecture are not observable from this diagram.

From a design standpoint we recommend that a designer first create their own version of Figure 1 with more detailed control lines and hardware components such as multiplexers (muxes), additional registers, and other arithmetic operators. The main reason for this is so that the designer can understand what is the data path and what are the control signals for this data path for their processor implementation. Also, this provides a useful high-level design that an instructor can look at and quickly assess if the designer understands all the components of the system and how they will interact.

Next, each of the major components, excluding the control, should be created and tested, and each should be implemented as a separate module in Verilog. Generally, we recommend building the arithmetic logic unit (ALU), the register file, and then the memory in order. In each case, the module input and outputs will correspond to signals in the designers initial diagram, and eventually these pieces can be treated as clear boxes in the design.

For the memory module, the university version (as well as the professional version) of Quartus has an "IP Catalog" tool that allows a designer to create on-chip memory. This tool creates a memory and shows a schematic of the memory module including signals going in and out of the module and any included registers on these paths. Also, within this tool you can specify a memory initialization file (mif) that will initialize the memory after programming the design onto the FPGA. This is how a user program can be loaded into memory, and for simplicity, we assume that the program start is at address 0. Alternatively, an off-chip SRAM memory chip is available on the DE2-115 and could be interfaced with the processor. In this case, however, a programmer also needs to be designed to initialize that memory with the program data.

The register file can also be implemented as an IP memory module with two read ports. We, however, recommend that a register file is implemented as a Verilog module using the register bits available in the soft-logic of the FPGA. There are two reasons for this. First, the access time of the register file will be one clock cycle, which is similar to a real ASIC design, and second, the design process helps student's understand the design of memories.

Once the major components are designed, then a top level module can be created with a finite state machine (FSM), muxing logic for the datapaths, and control signal generation from an instruction register to control the remainder of the processor. A simple four stage (fetch, decode, execute, write) model can be built within the FSM. This exercise is not simple, but is a function of understanding the control signals in the system and identifying which signals need to be set for each instruction. Branching and program counter (PC) incrementing is the last piece of the exercise that needs to be designed in the control.

IV. Implementations and Extensions to the RISC-V processors

In 2019, the computer organization course at Miami was taught as a badge-based course, and two students chose to implement the RISC-V processor (and are co-authors of this paper). These students have already completed basic programming courses and have completed a 2nd year digital systems course, which introduced them to Verilog. In both cases, the above described tools were used to design the processors and map them to DE2-115 prototyping boards. Table I provides basic information on the two implementations. Column 1 shows the last name of the designer, column 2 shows the ISA, column 3 provides the url to the code repository for the design files, and column 4 describes extensions to the architecture that students do to get an advanced badge in the course.

Extensions to the base hardware can include caches, pipeline, multicore system, and operating system functionality [27] [28] [29]. For the two extensions to the base processor in this course, McGrew implemented a simple cache, and Schonauer implemented a multicore system with a fence for access to shared memory. The memory IP from quartus typically take 2 cycles to access the data from in load operations. The goal of the cache is to build the hardware to make loads only cost 1 cycle on cache hits. The multicore implementation, by Schonauer, demonstrates a simple shared memory parallel machine with multiple cores that use an arbitrator to implement fence-like operations in the ISA.

V. Discussion and Conclusion

Project-based learning continues as a pedagogically sound approach to most of engineering education, and computer architecture education is no exception. The concepts an undergraduate needs to learn for computer architecture is very broad, but the experience and skills learned by creating a processor and implementing it on an FPGA are highly valuable. In this work, we provide a description of tools, design constraints, and ideas in how an undergraduate can design a RISC-V architecture and map it to an FPGA. The key benefit of choosing the RISC-V architecture is that it is a freely available ISA that is simple for students to understand. Additionally, there are a number of mature tools that have been created for the ISA, and this processor is getting more and more traction in industry and academia. For these reasons, we believe that if undergraduates are going to implement an architecture (and we believe that all of them should do this exercise) then RISC-V is the best choice.

One concern for educators with these types of exercises is will students actually perform the exercise, or will the students access the vast array of resources (including the two implementations listed here) and use someone else's work. For example, the BRISC-V tools [30] could be used by a student to quickly build a processor for submission to a course. This problem becomes worse as the number of students in computer architecture course increases, and project-based learning approaches become less and less viable for teachers. Our simple solution to this problem is to predefine either the ALU or a special instruction for each students design.

Of the two solutions, we believe that creating a black-box ALU with varying parameters is the easiest solution to implement. For example, the instructor could design a

TABLE I
DETAILS OF THE TWO IMPLEMENTATIONS

Creator	ISA	web hosting at: github.com/	Extension	Notes
McGrew	TinyRV2	tymcgrew/RISC-V	Cache	No CSRR, CSRW instructions
Schonauer	TinyRV2	EricSchonauer/RISC-V-Processor	Multicore with Fence	No CSRR, CSRW instructions

set of ALUs that have different ways of control such as an accumulator based ALU, a registered ALU, a sequential single-bit sequential ALU [31], and an 8-byte ALU. The control signals to and from each of these ALUs will create changes in the datapath that will challenge students in copying other designs without deeper understanding of the architecture.

From a future work perspective, the perfect educational extension to this work is the integration of a number of exercises after the creation of the processor that would allow students to improve their base architecture, create a software tool chain including compiling and linking for their processor, and create system software that would manage their system (similar to [10]). These exercises, over a number of courses, would give an undergraduate a deep understanding of the modern computer and its tools.

REFERENCES

[1] J. Impagliazzo, S. Conry, E. Durant, J. L. Hughes, and R. Meier, "Launching curricular guidelines for computer engineering: Ce2016," in *Frontiers in Education Conference (FIE), 2016 IEEE*. IEEE, 2016, pp. 1–4.

[2] P. Jamieson, "Does badge-based learning buck the grading curve? an educational experiment in computer architecture," in *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)*, 2014.

[3] M. Bates, *PIC microcontrollers: an introduction to microelectronics*. Elsevier, 2011.

[4] N. Alizadeh, "Avr enhanced risc microcontrollers data book," *ATML corporation*, vol. 33, p. 102, 1996.

[5] J. Hennessy, N. Jouppi, S. Przybylski, C. Rowen, T. Gross, F. Baskett, and J. Gill, "Mips: A microprocessor architecture," in *MICRO 15: Proceedings of the 15th annual workshop on Microprogramming*. Piscataway, NJ, USA: IEEE Press, 1982, pp. 17–22.

[6] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic, "The risc-v instruction set manual, volume i: Base user-level isa," *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62*, vol. 116, 2011.

[7] Y. N. Patt and S. J. Patel, *Introduction to Computing Systems: From Bits & Gates to C & Beyond*. New York, NY, USA: McGraw-Hill, Inc., 2004.

[8] D. Patterson and J. Hennessy, *Computer Organization and Design: The Hardware/software Interface*. Morgan Kaufmann, 2005.

[9] B. W. Kernighan and D. M. Ritchie, *The C programming language*. Prentice-Hall, 1978.

[10] N. Nisan and S. Schocken, *The elements of computing systems: building a modern computer from first principles*. MIT press, 2005.

[11] A. Clements, "The undergraduate curriculum in computer architecture," *IEEE Micro*, vol. 20, no. 3, pp. 13–21, 2000.

[12] ——, "Selecting a processor for teaching computer architecture," *Microprocessors and Microsystems*, vol. 23, no. 5, pp. 281–290, 1999.

[13] ——, "Arms for the poor: Selecting a processor for teaching computer architecture," in *2010 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2010, pp. T3E–1.

[14] M. Smolnikar and M. Mohorcic, "A framework for developing a microchip pic microcontroller based applications," *WSEAS Transactions on Advances in Engineering Education*, vol. 5, no. 2, pp. 83–91, 2008.

[15] M. D. Black and P. Komala, "A full system x86 simulator for teaching computer organization," in *Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, 2011, pp. 365–370.

[16] G. S. Wolffe, W. Yurcik, H. Osborne, and M. A. Holliday, "Teaching computer organization/architecture with limited resources using simulators," in *ACM SIGCSE Bulletin*, vol. 34, no. 1. ACM, 2002, pp. 176–180.

[17] C. Yehezkel, W. Yurcik, M. Pearson, and D. Armstrong, "Three simulator tools for teaching computer architecture: Little man computer, and rtlsim," *Journal on Educational Resources in Computing (JERIC)*, vol. 1, no. 4, pp. 60–80, 2001.

[18] B. Nikolic, Z. Radivojevic, J. Djordjevic, and V. Milutinovic, "A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization," *IEEE Transactions on Education*, vol. 52, no. 4, pp. 449–458, 2009.

[19] Y. Li and W. Chu, "Using fpga for computer architecture/organization education." in *WCAE@ HPCA*. Citeseer, 1996, p. 5.

[20] M. Holland, J. Harris, and S. Hauck, "Harnessing fpgas for computer architecture education," in *Proceedings 2003 IEEE International Conference on Microelectronic Systems Education. MSE'03*. IEEE, 2003, pp. 12–13.

[21] P. Jamieson, D. Davis, and B. Spangler, "The mythical creature approach-a simulation alternative to building computer architectures." in *FECS*, 2010, pp. 23–28.

[22] J. Macias-Guarasa, J. Montero, R. San-Segundo, A. Araujo, and O. Nieto-Taladriz, "A project-based learning approach to design electronic systems curricula," *Education, IEEE Transactions on*, vol. 49, no. 3, pp. 389 –397, 2006. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1668283

[23] D. A. Kolb *et al.*, *Experiential learning: Experience as the source of learning and development*. Prentice-Hall Englewood Cliffs, NJ, 1984, vol. 1.

[24] A. Martínez-Monés, E. Gómez-Sánchez, Y. A. Dimitriadis, I. M. Jorrín-Abellán, B. Rubia-Avi, and G. Vega-Gorgojo, "Multiple case studies to enhance project-based learning in a computer architecture course," *IEEE Transactions on Education*, vol. 48, no. 3, pp. 482–489, 2005.

[25] D. K. Vollmar and D. P. Sanderson, "A mips assembly language simulator designed for education," *Journal of Computing Sciences in Colleges*, vol. 21, no. 1, pp. 95–101, 2005.

[26] *Verilog Hardware Description Reference*, Open Verilog International, March 1993.

[27] J. H. Lee, S. E. Lee, H. C. Yu, and T. Suh, "Pipelined cpu design with fpga in teaching computer architecture," *IEEE Transactions on Education*, vol. 55, no. 3, pp. 341–348, 2012.

[28] J. Gray, "Hands-on computer architecture: teaching processor and integrated systems design with fpgas," in *Proceedings of the 2000 workshop on Computer architecture education*. ACM, 2000, p. 17.

[29] Z. Tan, A. Waterman, R. Avizienis, Y. Lee, H. Cook, D. Patterson, and K. Asanović, "Ramp gold: an fpga-based architecture simulator for multiprocessors," in *Proceedings of the 47th Design Automation Conference*. ACM, 2010, pp. 463–468.

[30] S. Bandara, A. Ehret, D. Kava, and M. Kinsy, "Brisc-v: An open-source architecture design space exploration toolbox," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2019, pp. 306–306.

[31] J. Robinson, S. Vafaee, J. Scobbie, M. Ritche, and J. Rose, "The supersmall soft processor," in *2010 VI Southern Programmable Logic Conference (SPL)*. IEEE, 2010, pp. 3–8.