An FPGA-based Toolchain for Computer Architecture Courses

Dr. Ming Li, The University of Tulsa

Ming Li is an assistant professor of practice of Computer Science at The University of Tulsa in USA. His teaching interest is in Computer Architecture and Programming. He is currently the primary instructor for system courses offered at The University of Tulsa, including undergraduate courses Computer Organization and Assembler, and Operating Systems, and graduate course Computer Architecture. He is a licensed PE of Electrical Engineering in the state of Oklahoma, with 12 years R&D experience from electronics industry.

Kevin Garcia-Estala, The University of Tulsa

Kevin Garcia-Estala is an undergraduate from the University of Tulsa pursuing a degree in Computer Science. His academic interests include software engineering, electrical engineering, and game development. He is currently a tutor for the Computer Science department at The University of Tulsa, assisting students in undergraduate courses.

Issai Gutierrez, The University of Tulsa

Issai Gutierrez is an undergraduate from the University of Tulsa pursuing a degree in Computer Science. His research interests include Machine Learning, Electrical Engineering, and Genetic Algorithms. He is currently a tutor for the Computer Science department at The University of Tulsa, assisting students in undergraduate courses.

An FPGA-based Toolchain for Computer Architecture Courses

Abstract

Computer architecture (CA) courses are challenging for instructors and students in computer science and engineering disciplines. The amount of information to be covered in a semester is overwhelming, and the lack of opportunities for hands-on projects can easily discourage students from achieving course learning objectives. To address these challenges, we propose an FPGA-based toolchain that eases the design of the curriculum and enhances student learning experience in CA courses. The toolchain encompasses a synthesizable RISC-V soft core implemented with Verilog HDL, a custom RISC-V assembler, a framework to use ModelSim/Questa for simulating the execution of RISC-V machine code, and an Intel FPGA evaluation board for ultimate verification. Projects can be quickly created around the toolchain to help students grasp assembly programming and the principles of processor design. The toolchain is suitable for both introductory and advanced CA courses, having the potential to spark greater student interest and boost the teaching qualify of CA courses.

1 Introduction

It is well-known that computer architecture (CA) courses are the hardest ones to achieve course learning objectives in computer science or engineering programs [1–4]. The underlying reasons are two tiers. First, driven by Moore's law and the continuous demand for higher performance, the processor industry has experienced rapid growth over the past several decades. Condensing the large amount of ideas, concepts and principles into a semester-long class is a daunting task, much less the challenges faced by the teaching and learning process. Second, many undergraduate learners are software-oriented, and tend to underestimate the unique importance of CA courses, i.e. establishing a base of hardware knowledge to guide the design of architecture-aware, high-performance, and secure software. Without an interesting curriculum to spark interest, even the most lightweight CA courses become a burden to students.

Currently, most computer science or engineering programs offer two levels of undergraduate CA courses. An introductory CA course focusing on computer organization and design (COD) and taught in the early stage of the program, and an advanced course covering more sophisticated features of modern processor and targeting senior students. Regardless of difficulty level, it is a consensus that CA courses should be taught with a learning-by-doing approach.

As RISC-V emerges as an open-licensed ISA with strong interests from the industry and an ever-growing ecosystem to support design and development, the ISA taught in most CA

classrooms today has transitioned from the legacy MIPS to RISC-V. The RISC-V ISA features its simplicity and ease of implementation and extension. Its base integer subset RV32I consisting of about 40 instructions is described with less than 30 pages in the RISC-V Manual [5], making it the most popular ISA adopted by CA instructors [2–4, 6–8].

In this paper, we propose an FPGA-based open source toolchain (https://github.com/ca-course-new/risc-v.git) that aims to ease the creation of curricula and boost student learning experience in CA courses. The toolchain encompasses:

- A baseline soft processor that implements RV32I ISA with Verilog HDL. The datapath and control are meticulously designed to closely match the RV32I microarchitectures introduced in Patterson and Hennessy's classic textbook [9]. It is straightforward for students to apply what they learned in lectures in hands-on projects built around the toolchain.
- A Python-based custom assembler that translates RISC-V assembly programs into machine codes for simulation and execution. It can be easily extended to support extra instructions or used in student projects for assembler design exercise.
- A framework to use ModelSim/Questa to simulate the execution of RISC-V machine codes. The framework allows fast simulation speed, and monitoring the contents of general purpose registers and memories, which are convenient for debugging assembly programs.
- A customized low-cost FPGA evaluation board that serves as the verification tool for soft processors. The board connects with LCD and Wi-Fi modules, providing extra facilities for students to debug and test their designs. Its onboard SDRAM and SPI interface to an MicroSD card reader allow building memory hierarchy in course projects.

The rest of the paper is organized as follows. In Section 2, related research is surveyed. In Section 3, we introduce the software in the proposed toolchain. In Section 4, we introduce the hardware of the toolchain. In Section 5, we recommend the possible use of the toolchain and ideas of CA projects built around it. In Section 6, we draw conclusions and discuss future directions.

2 Related Research

Learning-by-doing is known as the best strategy to achieve learning objectives in different engineering disciplines. There have been intensive efforts incorporating the design of soft processors into CA curricula to motivate proactive learning and improve teaching quality. A soft processor is the HDL implementation of a specific ISA. It is an ideal learning target because of its flexibility for modification, simulation and synthesis. For example, MIPS soft processors implemented with Verilog HDL are adopted in student projects [10, 11]. Recent efforts have shifted to designing RISC-V processors in Verilog HDL [2–4, 6, 7, 12]. These pedagogical processors target RISC-V's integer ISA RV32I or extensions due to its simplicity and thorough coverage by CA textbooks. Verilog HDL is the most popular HDL to describe them, which can be attributed to the language's similarity to other programming languages like C and a relatively gentle learning curve. Other ISAs and HDLs are used too. For example, simplified ISAs with operand width less than 32 bits can be implemented [1, 13, 14]. The Davis DINO CPU [8] implements RV32I in Scala-based language Chisel.

With a soft processor as the base, different learning activities can be created. The BRISC-V [4] is a web-based toolchain to study RISC-V ISA. Its CPU building tool BRISC-V Explorer can create the skeleton of a single-cored 5-stage pipeline, which is then used in student projects to build a working processor. In RVfpga [3], 20 labs are built around the VeeR EH1 core, an open-source soft CPU. These labs cover from running compiled C and assembly programs on the core, to the use of its peripherals, and to extending the core with new instructions and hardware performance counters. As described in [2], a pipelined RV32IM CPU is provided to students as the baseline processor in a design project. The hazard detection and handling units are absent and the students are tasked to fill in these missing units to support the normal execution of any program on the core.

After taking a COD course, students should be able to develop assembly programs. Using soft processors also provides new approaches for running and debugging such programs. Learners have the option to use mature HDL simulators such as Verilator, ModelSim, and Vivado Simulator [2, 3, 6, 12, 13]. Or they can develop their customized simulators tailored to meet the special requirements from learning [1, 3, 4]. Compared with traditional software simulators such as MARS for MIPS, and RARS for RISC-V, simulation tools built around HDL have the advantage of being more clock-cycle accurate and closer to the actual hardware. Users can freely add any of the processor's internal states to the signal list monitored during the simulation, which is not possible in traditional software simulators.

Last but not least, a soft processor can be synthesized and verified on a real IC. As FPGAs have fast development period and convenience of reprogramming, many CA courses use FPGA evaluation boards as the verification tool. For instance, an Altera DE2 board can be adopted in student projects to verify the functionality of MIPS soft processors [10]. As demonstrated in [6], the DE2-115 boards can be used in RV32I implementation projects, where students test their RISC-V cores with extended features like caching. Multiple FPGA boards can be organized as a cloud to provide service to larger classes. As shown in [15], an FPGA cloud consisting of 32 nodes of Xilinx FPGA boards is employed in a COD course to serve as the training ground for assembly program execution, individual component and CPU synthesis, and performance evaluation. Although not all the previous examples require an FPGA evaluation board in their learning activities [1–4, 8], we argue that the advantages of adopting it outweigh the increased cost to implementing CA courses. First, FPGA evaluation boards provide rich peripheral devices supporting displaying and communication, which make more visualized simulation possible. Second, verifying processors on an FPGA allows the leap from simulation- to application-oriented design. An FPGA device adds more constraints to the CPU implementation other than the basic requirements for correct datapath and control signals. Concepts like reset and clock signal, performance, memory-mapped I/O, and external interrupts can never become realistic to students until the soft processor is downloaded into an FPGA chip. Third, modern processors have rich interfaces to external storage, such as DRAM, NAND Flash, etc, which are important components to build the memory hierarchy. Even the best simulator cannot simulate the stringent timing requirements on the control signals to manipulate such external storage devices. Considering the heavy coverage of memory hierarchy in most CA courses, an FPGA evaluation board connecting different types of storage devices is the best tool to strengthen the understanding of such systems.

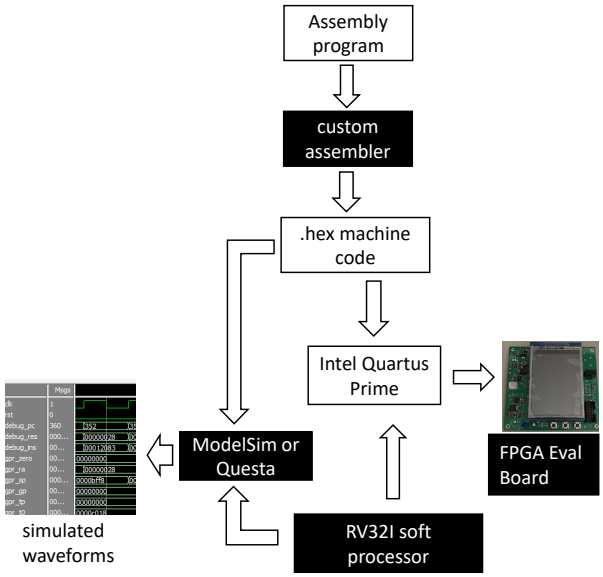

Figure 1: System diagram of the FPGA-based toolchain

3 Software of the FPGA-based ToolChain

In this section, we introduce the three software components in our FPGA-based toolchain: a soft processor implementing RV32I ISA with Verilog HDL, a custom assembler to translate RISC-V assembly programs, and a framework to use ModelSim/Questa to simulate the processor. The relationship between the three components and the FPGA evaluation board in CA learning activities is illustrated in Figure 1.

3.1 RV32I Soft Processor

To provide a base design, the RISC-V integer ISA RV32I is implemented as a 5-stage pipelined soft processor, which is structured to closely match the datapath and control described in the textbook [9]. A notable enhancement is that our version supports 37 RV32I instructions, only excluding FENCE, FENCE.TS0, PAUSE, ECALL and EBREAK. All the components are individual Verilog HDL modules invoked in top-level design files. We provide two formats of the top-level file: one written with Verilog HDL, and the other created as a schematics diagram. In simulation-oriented learning activities, the Verilog format with a pre-written testbench can be simulated directly in ModelSim or Questa. Typical constraints on requested resources during synthesis are not applied in this format. The schematics format lays out the components as they are introduced in the textbook. Instructors can open the schematics in an Intel Quartus Prime project. In-class demonstrations using this format can help students connect dots and lines. More importantly, this format can be synthesized in the Quartus IDE.

Compared with the pipeline from the textbook, the major adaptation in our base design is made to the memory for the convenience of synthesis. The overall memory structure is illustrated in Figure 2. Both the instruction and data memory are implemented with Verilog's memory

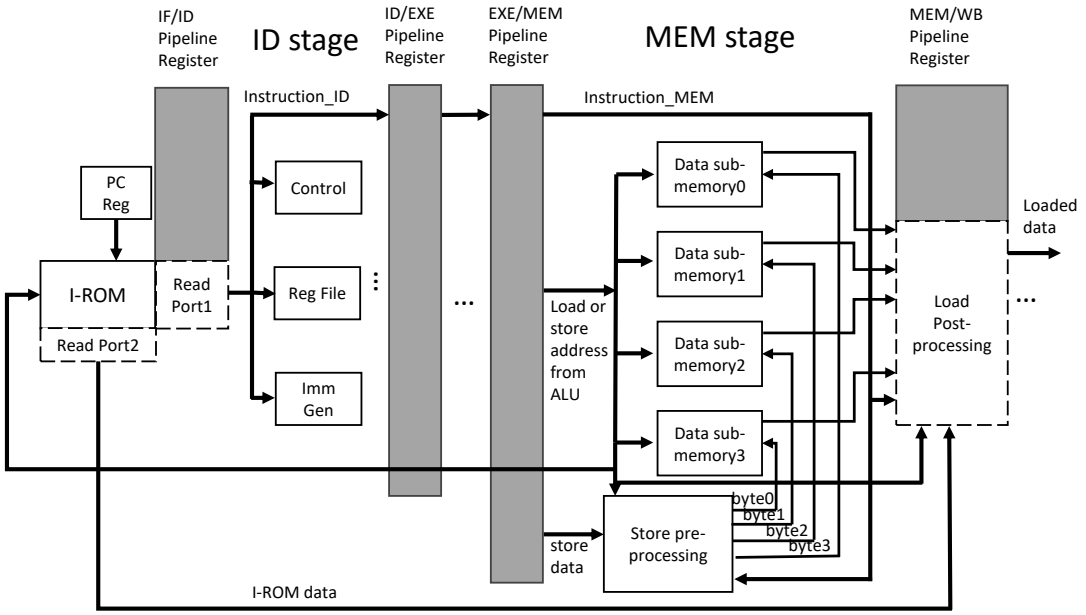

Figure 2: Memory diagram in the 5-stage pipelined soft processor. For simplicity, irrelevant components and signals are not included.

grammar and always structure. During synthesis, they are successfully recognized by the synthesizing tool as inferred memory to use Intel FPGA's BRAM. The instruction memory is created as a ROM considering that user programs are not often overwritten by the code itself. Two read ports are offered. One port for loading instructions for the ID stage, and the other for reading constant data in the MEM stage. The data memory is separated from the instruction memory, with a pair of read and write ports only accessible to the MEM stage. To support variable data widths requested by LB/SB, LH/SH and LW/SW, the data memory is split into four one-byte wide sub-memory components. A store pre-processing component examines the type of the store instruction and determines which one(s) of the sub-memory components should be active during the store operation. A load post-processing module examines the type of the load instruction and determines which byte(s) should be connected to the load data bus. One issue is reported in [6] about the difficulty to merge synthesized memory into their pipeline as Intel Quartus Prime can only create two-cycle memory to use FPGA's BRAM. The first cycle latches address into the memory's address register, and the next latches the data into an output buffer. We overcome this difficulty by making the output port of the synthesized memory as a part of the pipeline register (see the components with dashed outlines in Figure 2). As a result, any instruction or data item read out of the memory will be directly used by the subsequent pipeline stage, and no extra pipeline stages are required.

3.2 Assembler

A Python-based custom assembler is developed to translate RISC-V assembly programs into machine code, which can be loaded by Verilog HDL testbenches for simulation, or downloaded into an Intel FPGA's built-in BRAM to execute when the soft processor is programmed into the

FPGA chip. The workflow of the assembler is illustrated in Figure 3. The user assembly program is pre-processed first to remove all comments and empty lines. The updated program is then parsed for the first time with each assembly instruction translated into one machine code instruction, and stored as a 32-bit hexadecimal into a temporary hex file (temp.hex). The first parsing correctly encodes most assembly instructions, except for some jump and branch instructions with unresolved labels, which are due to targeting a later instruction in the program. Therefore, a second parsing reads the temp.hex, resolves all unresolved labels, and generates the machine code (machine_code.hex) ready for execution.

Our assembler supports two modes to name RV32I registers in user programs: either with X0-X31, or with RISC-V calling conventions. Grammar errors such as incorrect instruction format, and undefined or redefined labels can be caught and reported to the programmer. The current version is being improved to provide support for pseudo instructions. Besides being used to translate student assembly programs, this custom assembler can also be provided to students in a reduced form, for example, by removing the lines to translate certain types of instructions. Students are then tasked to complete the assembler in a project. Such projects are natural exercises that improve their understanding of important low-level concepts in CA courses, such as instruction formats, addressing modes, jump/branch target addresses, etc.

3.3 Simulation Framework Using ModelSim/Questa

To simulate the execution of assembly programs, CA courses typically resort to software simulators, such as RARS, Ripes and MARS. If the processor is a soft one implemented with HDL, HDL simulators can also be used, for example, Verilator, ModelSim/Questa, and Vivado Simulator. HDL simulators have advantages over software simulators in that they are closer to the hardware, and flexible to monitor any internal states or signals of the processor during simulation. As our RV32I soft processor is implemented with Verilog HDL, we choose ModelSim/Questa as the simulation environment. During simulation, the machine code translated from user assembly program will be loaded into the simulated instruction ROM. The clock signal for the simulated processor is set with a period of 20 ns to be consistent with the hardware. The reset signal is set active in the first 40 ns, after which the first rising edge of the clock signal will load the first instruction into the pipeline.

In ModelSim/Questa, the processor's internal signals, such as the PC value, the instruction read out of the instruction memory, the ALU's input operands, etc, can be added to the list of signals monitored by the simulation. By doing so, the simulation vividly shows how different components are coordinated to execute the instruction. As our RV32I soft processor is pipelined, internal signals are sourced from different pipeline stages. For example, an R-type instruction has its PC value generated in the IF stage, instruction word available in the ID stage, and ALU operands ready in the EXE stage. When students have not learned pipelines, this inconsistency

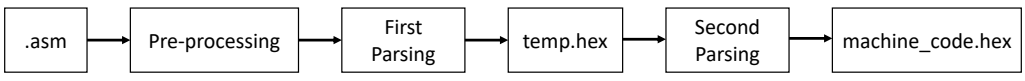

Figure 3: Workflow of our custom assembler

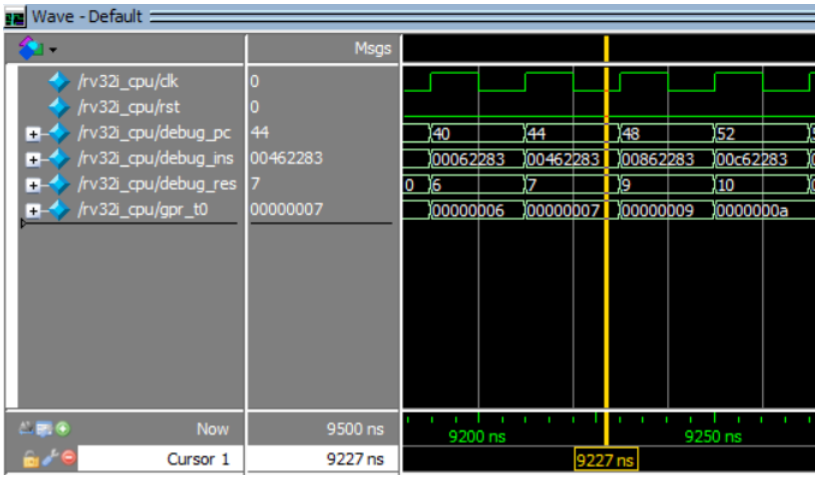

Figure 4: Simulating the execution of a sorting program on the soft processor in ModelSim. Monitored signals are: clock signals (clk), reset signal (rst), pc (debug_pc), instruction (debug_ins), ALU's output (debug_res) and GPR T0 (gpr_t0). The sorted numbers are 6, 7, 9 and 10.

will cause difficulty to interpreting the simulation waveforms. We solve this by relaying all such signals to the output of the WB stage through pipeline registers. As a result, students can view them in the same cycle, and the pipeline simulation will appear to be conducted on an unpipelined processor.

Using the simulation under ModelSim as an example, Figure 4 shows the result of running a sorting program. The waveforms are generated in 5 seconds after entering a "run 9500" command in the ModelSim command line, which simulates the first 9500 ns of the program execution. If single-step is required in debugging, we can instead enter a "run 130" command at the beginning of the simulation, which will only progress the simulation to the 5th rising edge of the clock signal (clk) after the reset (rst) goes inactive. Command "run 20" can then be repeatedly entered, and each will add the results of one instruction to the waveform window.

4 FPGA Evaluation Board

A low-cost FPGA evaluation board is designed as the hardware part of our toolchain. In student projects, this board serves as the verification tool. Different learning activities targeting synthesizable designs can be created around it.

4.1 Circuit Design

The block diagram of the PCB is illustrated in Figure 5. Detailed schematics and PCB layout can be found in our GitHub repo. The central IC is a low-cost Intel Cyclone IV FPGA: EP4CE15E22C7N, which contains 15,408 LEs, 516,096-bit on-chip memory and 82 GPIOs. After synthesis, our soft processor with peripheral driving modules typically needs less than 50% of the LEs. Each RV32I instruction word is 32-bit wide, and the on-chip memory allows a maximum of 516096/32=16,128 instruction and data words. Most student programs are unlikely to exceed this limit. The synthesizable version of our soft processor (available in our GitHub repo

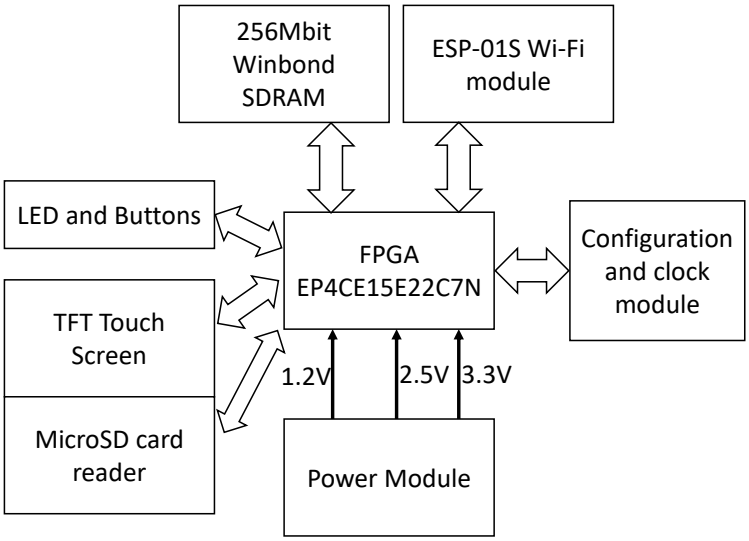

Figure 5: Block diagram of the FPGA evaluation board

as an Intel Quartus Prime project) partitions the memory space as follows: the lowest 8K words are reserved for instruction ROM with the first instruction located at address 0; The next 6K words are reserved for data memory. Both instruction and data memory use FPGA on-chip memory. The address space above the lower 14K words is reserved for peripherial devices, whose control and data registers are mapped into the processor's address space for memory-mapped I/O operation. In addition to on-chip resources, this FPGA comes with a simple 144-pin LQFP package, which allows manual soldering on PCB to avoid going through reflow soldering that will increase the preparation period and cost of the board. Note that as the FPGA chip needs 1.2V, 2.5V and 3.3V DC power supplies, we use two AD2302 step-down regulators in the power module to generate the 1.2V and 3.3V from external power source, and a TLV1117LV25 to generate the 2.5V supply. The clock signal of the FPGA is provided by a 50 Mhz (20ns clock period) active crystal oscillator. Both the AS and JTAG ports are wired to connector sockets to allow programming and debugging the FPGA. Based on the primary topics covered by COD and advanced CA courses, we select a set of peripheral devices to connect with the FPGA's 82 GPIOs. These devices can be used in student projects for display, communication, storage and control purposes.

4.2 ELEGOO UNO R3 2.8-inch TFT Touch Screen

This 320*240 LCD module is compatible with Arduino UNO. To interface with it, we developed its driving module in Verilog HDL, whose control and data buffers are mapped into the processor's address space. We translated its C source code available in Arduino IDE into RV32I assembly procedures, and built on top of them two wrapper procedures: one to display a single ASCII symbol at a user specified location, and the other to display the contents of any GPR as an 8-digit hexadecimal value. We provide these procedures to students to be called in their own programs. Each ASCII symbol is displayed with a resolution of 16*12 pixels, and the LCD screen is partitioned into grids of 20 rows and 20 columns of displaying positions. The bitmaps of the 96 visible ASCII symbols are stored in a font ROM written with Verilog HDL and attached to processor's address space. Programs directly accessing the font just need to use RV32I load

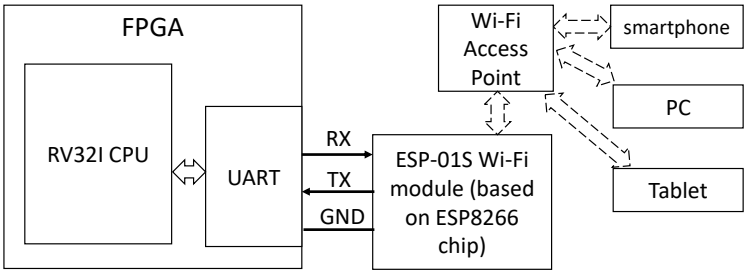

Figure 6: The Wi-Fi module ESP-01S connects the RV32I soft processor to a network.

instructions. With this LCD as the primary display device, projects requiring visualized demonstration can be created, motivating students to develop more interesting programs.

4.3 ESP-01S Wi-Fi module

The module is based on ESP8266, a low-cost microchip with a built-in TCP/IP protocol stack. Figure 6 illustrates the relationship between this module, the FPGA and other devices connected to the same network. The module connects with the FPGA via a 3-wire serial port (RX, TX and GND), and is configured to work in different modes by receiving "AT" commands from the serial port [16]. We developed a UART component in Verilog HDL with its control registers, sending and receiving buffers mapped into the address space of the RV32I processor. The UART coordinates all communication between the processor and the Wi-Fi module. For the processor to send network data, the UART receives a sequence of "AT" commands along with the data words from the processor and relays them to the Wi-Fi module, which then sends the data to the targeted network device. To receive network data, the UART also relays the processor's "AT" commands to the Wi-Fi module, which then responds by sending back the received network data to the UART. The processor lastly reads the network data (if any) from the UART's internal buffer. In our GitHub examples, we provide RV32I assembly procedures to help instructors and students use this Wi-Fi module. The integration of this module into the FPGA toolchain enables novel applications. For example, the FPGA's on-chip memory can be updated by receiving programs and data from the Wi-Fi module. A device such as a laptop, connecting to the same LAN or via the Internet, can reprogram the soft processor implemented in the FPGA without needing a physical connection to the FPGA board through a firmware download cable. To serve such or similar purposes, the Wi-Fi module needs to be configured as a server, with certain ports open for other devices to access. To prevent potential port scans or DoS attacks, especially those originating from the Internet, we recommend either disallowing the connection to the WAN at the Wi-Fi access point, which will make the board only accessible by devices in the same LAN, or implementing additional measures such as adding firewalls, which makes the Internet connection more secure.

4.4 W9825G6KH SDRAM and MicroSD card reader

Considering the importance of memory hierarchy in COD and advanced CA courses, our FPGA board encompasses an on-board SDRAM and an SPI interface to connect with an Micro-SD card reader, making it a suitable platform to build a memory hierarchy. The Micro-SD card reader is

an extra function provided by the TFT touch screen. It is controlled by an SPI port, connecting with 4 FPGA GPIOs. Example C code is available in Arduino IDE to perform read and write operations on inserted Micro-SD cards. We are currently translating the C code into RV32I assembly procedures and making a Verilog HDL control module for the card reader, both of which will be added to our example project on GitHub. The onboard SDRAM W9825G6KH is organized as 4M words * 4 banks * 16 bits, providing a total capacity of 256 Mbits. Its address, data and control pins are directly connected to 39 of the FPGA's GPIOs. Its package TSOP II 54-pin is suitable for manual soldering to the board like the FPGA chip. In the example in GitHub repo, we include a primitive memory management unit (MMU) to control the interface to the SDRAM. The MMU's internal registers and read/write buffers are mapped into the processor's address space, hiding the control details from the programmer. Upon power-up, the MMU's finite state machine (FSM) automatically initializes the SDRAM and enters a waiting-for-request state afterwards. An internal timer periodically triggers the auto-refresh of the SDRAM. Both read and write requests from the processor are circularly buffered by the MMU, which will notify the processor upon the request being granted. The current MMU does not support all the features described in the SDRAM's manual, nor is it controlling the SDRAM like a main memory in a real memory hierarchy. However, it does prove the feasibility of making the SDRAM a backup storage for the FPGA's on-chip memory. We plan to make the implementation of a fully operational MMU and further the memory hierarchy as student projects in our advanced CA courses.

4.5 Other Aspects of the Board

There are three switch buttons connected to the FPGA to allow user programs to scan for press/release actions. For projects using external control, such actions can be interpreted as operations like opening/closing a menu, incrementing/decrementing a variable, etc. There is only one indicator LED onboard as we already have the interface to the TFT LCD. However, for a fast test of the FPGA board's basic functionality, one can always download a simple module into the FPGA to blink the LED.

The design of this FPGA evaluation board aims at making the board open-source, affordable, and

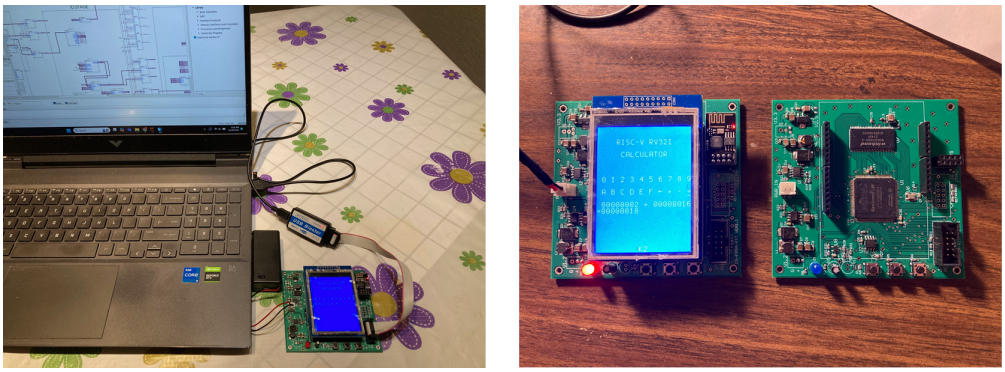

Figure 7: Demonstration of the FPGA evaluation board. The left figure shows the connection of the board with a laptop via a programming cable. In the right figure, the left circuit shows the FPGA board mounted with LCD hosting the RV32I soft processor to run a hexadecimal calculator. The right circuit shows the board without LCD mounted.

easy to solder and debug. In our GitHub repo, the schematics and the PCB layout are both open-source, allowing instructors and students to use them as a base design and develop their own evaluation boards. The aforementioned off-board devices can be acquired from major E-commerce platforms like Amazon and eBay. The onboard components can be ordered from electronics distributors like Digi-Key and Mouser. With all peripheral devices and a power cord included, the estimated cost per board is 139 USD, which is cheaper than most mainstream commercial FPGA boards. As for assembling the boards for a class, we recommend that instructors find TAs or interested students with PCB soldering experience to help, especially those who soldered surface mounted devices before. However, instructors can also consider designing a project-oriented CA course, in which everything must be built from scratch. Students will acquire full experience from building their own board to implementing the soft processor. As a demonstration, Figure 7 shows the board connecting with a laptop for programming and debugging, and it hosting our RV32I soft processor that is executing a hexadecimal calculator written with RV32I assembly language. A rich collection of projects and learning activities can be created to target this board and the proposed toolchain. We present some ideas in the next section.

5 Use of the Toolchain in CA Learning Activities

The FPGA-based toolchain aims to instill learning-by-doing into the curricula of CA courses. In this section, we propose the preparation work instructors and students need to do to use the toolchain in a semester long CA course. We also recommend ideas of hands-on projects built around the toolchain for both COD and advanced CA courses.

5.1 Preparation

The RV32I soft processor in our toolchain is built to closely match with the textbook [9]. For COD courses adopting the toolchain, the textbook [9] and a supplementary RISC-V manual [5] should suffice to explain the ISA and the processor's datapath and control. However, one issue might arise to compromise the benefits brought by including the toolchain. Students may not have taken a digital design course as a prerequisite. As the design of the soft processor entails a base knowledge about combinational and sequential circuits, a deficiency in digital design will make it hard to achieve learning objectives. Instructors should dedicate 1-2 weeks to fill the knowledge gap if this is the case.

On the other hand, covering Verilog HDL design for another 1-2 weeks afterwards might be more rewarding than complicating the curriculum. As mentioned in previous sections, the design work to develop individual components for the processor is mostly done. Without revealing the underlying Verilog HDL code, the top level schematics can already be used for in-class demonstration. Yet, it is impossible for students to just play blocks in a project-oriented COD course. The modification of the processor requires the ability to read and modify Verilog HDL modules. We recommend that instructors at the minimum cover the Verilog HDL representation of multiplexers, encoders and decoders, registers, counters, memory, and FSMs.

As for the adoption of the toolchain in advanced CA courses, instructors should expect that students already have finished a low-level CA course. For students who might have forgotten

minor details, instructors should encourage them to catch up within a short time. The projects in advanced CA courses aim to implement caching, dynamic prediction, and other modern architecture features, which are difficult and time-consuming. Students should be allowed to start their project as soon as the relevant knowledge base is covered.

5.2 COD Projects

For the COD course with sophomore and junior students as the potential audience, the projects created around the toolchain aim to help consolidate their understanding of the core knowledge. At the minimum, they should be able to complete an unfinished processor in student projects. Instructors can remove the Verilog HDL code partially or completely for some modules, which are left for the students to complete. Components like ALU, register file and main control unit are especially suitable for such exercises due to their importance in the architecture and more modularized code structure. Instructors can also remove the codes for all but one instruction from a certain instruction group, and let the student do the missing ones with the given one as a reference. As the translation of assembly instructions to machine code requires bit manipulations, working on a project to complete a partially designed assembler can greatly enhance student understanding of RISC-V instruction formats, addressing modes and jump/branch target calculation. In addition to the Python version, we are currently adapting our custom assembler with other languages like Java and C++. Considering that most students in their sophomore year have taken at least one programming class, using high-level languages like Python/Java/C++ to complete an assembler will be a rewarding task with manageable difficulty.

After adopting our toolchain, the sharpest contrast to a COD using only software simulations comes from using the FPGA evaluation board. According to the order of the topics covered by the textbook [9], we recommend the following assembly programming projects with the FPGA evaluation board as the ultimate verfication tool.

- Project idea 1: Calculator. This project requires students to design a calculator supporting arithmetic and logic operations. The assembly program should scan the actions on the three switch buttons for input values and operators, perform the operation, and display the results on the LCD. This project examines the use of different R-type/I-type instructions, which are in the group of instructions typically introduced first in a COD course. To allow students to focus on the use of these instructions, the project should provide ASCII symbol display, register display, and key scanning and anti-bouncing in pre-built procedures.
- Project idea 2: Sorting. This project requires students to sort a sequence of values and display the result on the LCD. Both brute-force and efficient sorting algorithms can be selected as the project's goals. Since sorting algorithms are memory-intensive, implementing them provides students with valuable hands-on experience in using load and store instructions. Brute-force algorithms, such as Selection Sort and Insertion Sort, serve as a foundation for practicing loops and data swapping. Efficient algorithms like Merge Sort, Quick Sort, and Radix Sort challenge students to develop effective memory management strategies for handling 1D and 2D arrays, especially when the memory is provided by the physical FPGA chip with limited capacity. The recursive nature of Merge Sort and Quick Sort further makes their implementation an ideal exercise for deepening

students' understanding of low-level concepts, including callers and callees, leaf and non-leaf procedures, and stack operations like push and pop.

- Project idea 3: Processor Modification and Hamming Code. This project requires the user program to receive a codeword sent from an upstream device, apply the principle of Hamming code (a topic of COD) to detect possible errors, and return a corrected codeword to the device. Procedures for reading and writing the Wi-Fi module will be provided as the pre-built. In addition to developing the assembly program, this project can provide opportunities for students to modify the processor. For instance, by asking them to add support for instructions that must be used in the program, such as xor and shift instructions.

There are many other possibilities for COD projects built around our toolchain. For instance, projects that implement forwarding and hazard detection units. While aiming to make these projects more interesting and inspiring to the students, instructors should always keep them on a difficulty level proper for the students and provide enough in-class and out-of-class support.

5.3 Advanced CA Projects

Advanced CA courses are often open to senior undergraduate and graduate students. While topics like performance, trends in processor industry and benchmark programs can be learned without too much challenge, student performance starts to diverge after ISA and pipeline basics are recapped in a very short time. Subsequent topics like cache optimization and dynamic branch prediction raise the level of abstraction immediately, and only covering their relevant textbook sections [17] without hands-on projects can be detrimental to the learning effects. Our proposed toolchain can serve as a baseline for instructors to design such projects to enrich student learning experience. We recommend a few project ideas as follows, although the flexibility of the proposed toolchain definitely allows readers to be more creative than we are.

- Project idea 4: Cache System. The soft processor in the toolchain uses FPGA's on-chip memory as the main memory, which is accessed in the MEM stage of the pipeline without any caching. The peripheral devices connected to the FPGA board make it possible to build a real memory hierarchy in student projects. The Winbond 256Mbit SDRAM on the FPGA board can be used as the main memory. A MicroSD card inserted into the card reader can be used as the backup storage of the SDRAM. The FPGA's on-chip memory can therefore be used as single- or multi-level caches, or instruction and data caches. Many benefits are brought by accomplishing this project. For instance, achieving profound understanding of memory-related concepts, and having a platform to compare performance under different design decisions on structuring the memory hierarchy. On a moderate difficulty level, such a design project needs to provide the SDRAM and MicroSD card reader's driving modules and example reading/writing programs in the starter package, allowing students to just focus on structuring the memory hierarchy. For more ambitious students, however, a bonus project to build their own storage driving modules can be offered.
- Project idea 5: Branch Prediction System. Dynamic branch prediction is often taught in advanced CA courses with lectures accompanied by exercises on reference access sequences. Students can easily figure out the prediction sequence and calculate the accuracy of the prediction. Although dynamic branch prediction is known to be more

effective, students rarely have a chance to see how it works in real CPUs. The proposed FPGA evaluation board is a perfect platform to make it realistic. In a project designing dynamic branch predictors, students can be tasked to add branch prediction buffers and extra supporting logic to proper pipeline stages. To see how much improvement is made, they will also need to add hardware counters accessible to the processor in the form of special purpose registers. The most challenging part of this project will lie in designing proper logic to correct mis-predictions.

- Project idea 6: Out-of-Order Processors. Out-of-order (OOO) processors are covered as a sub-topic of instruction level parallelism (ILP). Contents like Tomasulo's algorithm, reservation stations, common data buses, speculative execution and reorder buffer add further confusions to learners for lack of an effective way for demonstration. After studying the entirety of the proposed soft processor and finishing one or two preparatory exercises, students will feel more comfortable to read the schematics of processors, modify and redesign their datapath and control, and even re-organize their architecture. This is the point to assign more ambitious projects, like building an OOO processor. Although the proposed soft processor is in-order, its modules can be reused in student projects to develop OOO processors. The first goal of such projects should be the design of an OOO processor that can correctly run user programs. Subsequently, students can add hardware counters to compare between OOO and in-order processors.

In addition to the above ideas, multi-cored processors, vector processors and cache coherence protocols are also candidates to make design projects using our toolchain. Due to the inherent challenges in advanced CA courses and the tight schedule in a typical 4-month semester, we recommend that instructors make such projects as group projects. At the beginning of the course, students can be divided into groups of two or three, and each group should be assigned with one FPGA board before instructors recap ISA and pipeline basics. While senior undergraduate and graduate students are expected to learn project materials more proactively, extra support from instructors in the form of out-of-class tutoring and short workshops will help address technical issues in-time and guarantee the smoothness of the project-based learning activities.

6 Conclusions and Future Work

In this paper, we propose an FPGA-based toolchain to enhance student learning experience in computer architecture courses. The toolchain includes a soft processor implementing RISC-V RV32I ISA, a Python-based custom RISC-V assembler, a framework to use ModelSim/Questa to simulate and debug the processor, and a customized low-cost evaluation board with Intel's FPGA device. We also propose a series of project ideas to use the toolchain in both introductory and advanced CA courses.

For future work, we plan to deploy the toolchain in both COD and advanced CA courses at our university. We will evaluate the effectiveness of adopting the toolchain via performance analysis and student surveys. We plan to finish adapting our custom assembler to support more languages like Java and C++, we also plan to add more pre-built Verilog HDL modules to better support the use of the peripheral devices on the FPGA board. We will build new pre-written procedures for faster code development in student projects.

References

[1] M. Doğan, K. Öztoprak, and M. R. Tolun, "Teaching computer architecture by designing and simulating processors from their bits and bytes," *PeerJ Computer Science*, vol. 10, p. e1818, 2024.

[2] S. A. Zekany, J. Tan, J. A. Connelly, and R. G. Dreslinski, "Risc-v reward: Building out-of-order processors in a computer architecture design course with an open-source isa," in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pp. 1096–1102, 2021.

[3] S. L. Harris, D. Chaver, L. Piñuel, O. Kindgren, and R. Owen, "Rvfpga: Computer architecture course and mooc using a risc-v soc targeted to an fpga and simulation," in *ASEE Annual Conference and Exposition, Conference Proceedings*, 2023.

[4] R. Agrawal, S. Bandara, A. Ehret, M. Isakov, M. Mark, and M. A. Kinsy, "The brisc-v platform: A practical teaching approach for computer architecture," in *Proceedings of the Workshop on Computer Architecture Education*, pp. 1–8, 2019.

[5] "RISC-V Instruction Set Manual." https://drive.google.com/file/d/1uviu1nH-tScFfgrovvFCrj7Omv8tFtkp/view. last accessed: 2024-12-01.

[6] P. Jamieson, H. Le, N. Martin, T. McGrew, Y. Qian, E. Schonauer, A. Ehret, and M. A. Kinsy, "Computer engineering education experiences with risc-v architectures—from computer architecture to microcontrollers," *Journal of Low Power Electronics and Applications*, vol. 12, no. 3, p. 45, 2022.

[7] M. H. B. Saif, N. U. Sadad, and M. N. I. Mondal, "Fpga implementation of educational risc-v processor suitable for embedded applications," in *2023 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pp. 1–5, IEEE, 2023.

[8] J. Lowe-Power and C. Nitta, "The davis in-order (dino) cpu: A teaching-focused risc-v cpu design," in *Proceedings of the Workshop on Computer Architecture Education*, pp. 1–8, 2019.

[9] D. A. Patterson and J. L. Hennessy, *Comptuer Organization and Design RISC-V Edition: The Hardware Software Interface*. Morgan Kaufmann, 2020.

[10] 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, 2011.

[11] F. Passe, M. Canesche, O. P. V. Neto, J. A. Nacif, and R. Ferreira, "Mind the gap: Bridging verilog and computer architecture," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2020.

[12] V. Lalu *et al.*, "Design and implementation of 5-stage pipelined risc-v processor on fpga," in *2024 28th International Symposium on VLSI Design and Test (VDAT)*, pp. 1–6, IEEE, 2024.

[13] K. C. Lau, M. A. Rahman, and C. U. E. Kan, "Design and implementation of pipelined risc processor for educational purpose," *Evolution in Electrical and Electronic Engineering*, vol. 5, no. 2, pp. 369–377, 2024.

[14] M. D. L. Á. Cifredo-Chacón, Á. Quirós-Olozábal, and J. M. Guerrero-Rodríguez, "Computer architecture and fpgas: A learning-by-doing methodology for digital-native students," *Computer Applications in Engineering Education*, vol. 23, no. 3, pp. 464–470, 2015.

[15] K. Zhang, Y. Chang, M. Chen, Y. Bao, and Z. Xu, "Computer organization and design course with fpga cloud," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pp. 927–933, 2019.

[16] "ESP8266 Introduction." https://www.instructables.com/Getting-Started-With-the-ESP8266-ESP-01/. last accessed: 2024-12-01.

[17] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2019.