Design & Implementation of 5-Stage 32-bit RISC-V Pipeline Processor on FPGA

Muhammad Zain Naveed[1], Muhammad Amar[2], Arshad Hussain, Zeeshan Akbar Awan
Department of Electronics
Faculty of Natural Sciences, Quaid-i-Azam University, Islamabad
zain_qau44@yahoo.com, amarkazi42@gmail.com, arshad@qau.edu.pk

ABSTRACT

The proposed work focuses on designing and implementing a Harvard Architecture-based 32-bit RISC-V processor, using the open instruction set to enhance speed and performance. The processor leverages pipelining, comprising five stages—instruction fetch, decode, execute, memory access, and write back—and operates within a single cycle to improve CPI (clock cycles per instruction). The control unit plays a crucial role in managing the smooth execution of each pipeline stage operation. A hazard detection unit is designed to address potential pipeline hazards, which utilizes registers to handle pipeline hazards like data dependency and prevent stalls. Branch instructions, which can introduce delays and reduce efficiency, are addressed through the Static Branch Predictor. The branch predictor reduces CPI by saving one clock cycle during branch execution, which enhances the processor's overall performance. All components of the processor are designed in Verilog and simulated in Vivado. For hardware implementation, this code is synthesized into a gate-level netlist, which optimizes the design for FPGA architecture. Implementation follows, where the netlist is mapped to FPGA resources and then routed to a specific location on board. This processor design is tested using the Xilinx Vivado toolchain to verify its functionality and efficiency. Finally, the bitstream is generated and loaded onto the ZedBoard Zynq Evaluation and Development Kit (xc7z020clg484-1) FPGA for real-time testing and evaluation. To visually demonstrate the instruction execution process, Two 7-segment displays are connected via Pmod headers on the FPGA board. These displays offer real-time feedback on the instruction sequence being executed, which allows for easy observation and verification of the processor's operational stages.

Keywords: RISC-V Processor, Pipeline, Static Branch Predictor, Xilinx Vivado, FPGA.

1. INTRODUCTION

The evolution of processors, from the large, slow vacuum tube-based designs of the 1940s to today's efficient microprocessors, has been driven by innovations in performance and versatility. Key milestones include the introduction of transistors in the 1950s and Intel's 4004 microprocessor in the 1970s, which led to the development of CISC and RISC architectures. RISC, especially with the open-source RISC-V introduced in 2010, has become a powerful architecture due to its simplicity, flexibility, and cost-effectiveness. This paper presents the design and implementation of a 5-stage

pipelined RISC-V processor based on the RV32I instruction set, optimizing processing speed through Static Branch Prediction and efficient component interaction. Implemented on a ZedBoard FPGA, this design offers insights into Power, Performance, and Area (PPA) factors, demonstrating the potential of FPGA-based prototyping for modern computing systems. The paper is structured to cover ISA, processor architecture, and evaluation of its real-world applications.

2. INSTRUCTION SET ARCHITECTURE (ISA)

All processor core instructions include a basic 32-bit integer instruction set known as RV32I. This open-standard ISA follows Reduced Instruction Set Computing (RISC) principles and features no branch delay slots, supporting optional variable-length instruction extensions. As a 32-bit processor, all RV32I instructions are 32-bit long, using three operand registers (two source and one destination). The design includes 32-bit registers (general purpose registers) and utilizes four different basic instruction formats. Separate instruction memory is designed, from where instructions are fetched based on the PC address. RISC-V's core features include a simple, clean-based ISA, flexibility through standard extensions, and an open, royalty-free model, driving its global adoption.

RISC-V Instruction Set

Core Instruction Formats

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

Figure 1 Image Provided by Chegg

3. PROCESSOR ARCHITECTURE

Pipelining is a technique in computer architecture that improves instruction throughput by overlapping instruction execution. It breaks the execution process down into discrete steps, including fetch, decode, execute, memory, and write back, enabling the processing of more than one instruction at the same time. Each stage deals with a part of an instruction, and this makes executions faster and saves cycles. In pipelining, by having several stages working simultaneously at different instructions, the efficiency and speed of CPUs are improved. Its block diagram is shown in Fig. 2.

I. Pipelined unit

The pipeline stages in a CPU break down an instruction into a sequence of steps and so it enhances the efficiency and speed of the processing. The IF cycle takes the PC and fetches an instruction from memory to execute yielding a new value for the PC. Instruction Decode (ID) fetches and decodes the instruction, identifying the opcode, registers, and any intermediate values as it produces control signals and data hazard signals. The Execute unit (EX) performs computation using the ALU, including the arithmetic and logic operations as well as the address calculation. Memory Access may access data from memory in 1 cycle or 2 cycles depending on actual memory cache misses or memory access time. Last of all, Write Back takes results from other stages and stores them in the register file for use by the next instructions while handling data hazards by stalling and forwarding.

II. Control unit

The control unit is addressed with the CPU to manage the instruction execution and generate the control signals. A control unit is designed in the decode state to decode instructions to understand the required operation, control the ALU for arithmetic and logic operations, and manage smooth data flow between registers and memory.

III. Hazard Detection Unit

Hazard detection and resolution are key components that are used in maintaining the efficiency and accuracy of the RISC V pipeline. Data hazards occur when an instruction depends on the result of another and cause delays if the data is not available, resolved by forwarding or pipeline stalling. Control hazards caused by branch instructions interfere with the program flow, and this can be prevented with branch prediction methods. Structural hazards occur when two instructions attempt to use a particular hardware resource, but such conflicts are not possible in the RISC-V pipeline.

IV. Branch Prediction unit

In pipelined processors, branch prediction plays an important role in modern computer architecture. They are effective in pipelined processors because they allow early prediction of branch instructions. Branch prediction is a technique implemented in pipelined processors to decide the branch prediction before full execution, to prevent pipeline delays and enhance performance. Without branch prediction, the CPU must stall the pipeline until the branch outcome is known, leading to inefficiency and delays. A static branch predictor is used in this processor to predict the branch, which reduces the latency and saves one clock cycle.

- Static Branch Predictor

A static branch predictor makes a simple, fixed prediction of whether a branch is taken or not. Unlike dynamic branch predictors, it doesn't use runtime feedback. Instead, it relies on programmed-based predictions, where the compiler takes branch hints from the instruction set architecture to predict the branch outcome. In the decode stage, the processor statically predicts branch and jump instructions by examining the two most significant bits (MSB) of the opcode in the control unit. If these 2 bits are `11` (binary), it identifies a branch instruction. Consequently, the processor flushes the last-fetched instruction. The program counter (PC) is then updated by adding the offset to its current value to form a new address. This new address is used to fetch the next instruction.

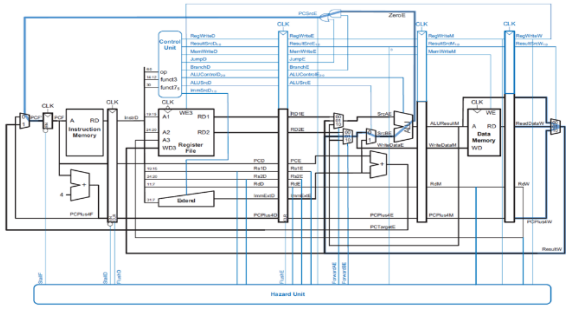

Figure 2 Pipeline RISC V with Hazard and Control Unit

4. DESIGN METHODOLOGY

It includes a few steps to design, Simulate, and Implement the RISC V processor. The few steps are:

I. Development Tools and Environment

Designing and Simulation of this processor is performed in Vivado. It developed by Xilinx, is a software

suite for designing FPGA and SoC hardware. It is a powerful tool for RTL coding, high-level synthesis, and FPGA implementation, optimizing designs for area, timing, and power. It supports robust simulation and system-level integration with custom and third-party IP cores, offering power analysis and strong community support. The processor design is implemented on the ZedBoard, a versatile development kit featuring an ARM Cortex A9 and Xilinx FPGA, with DDR3 memory and peripheral interfaces. This setup is ideal for prototyping embedded applications like IoT and digital signal processing, backed by Vivado's comprehensive FPGA development tools and resources.

II. Hardware Description Language

Verilog is a type of HDL, which is used in designing and implementing digital circuits such as RISC-V processors. It helps designers to model and simulate processors at the RTL level and gate level so that designers can test and optimize constituent components such as the ALU control unit and memory interfaces. Due to its support for synthesis tools, Verilog is suitable for the development of systems using RISC-V technology.

5. SIMULATION AND VERIFICATION

Vivado Compiler/Simulator was used to implement the design. Because of the language's versatility and the availability of these tools through university resources, Verilog was chosen over VHDL. In this design technique, Verilog behavioral modelling was initially utilized to develop each processor component, which was then tested using a Verilog test bench application. Any computing system's processor, which is executing necessary duties and executing instructions, is its beating heart. Individual components must seamlessly integrate to produce a processor that executes instructions correctly. Zed board FPGA has a clock frequency of 100 MHZ, equal to 10 ns. So, considering the hardware implementation, we used to provide a clock time of 10 ns. The total Simulation time is 300 ns. For the first 150 ns, reset is 0, which disables all components' functionality. For the remaining half-time, reset is 1, which enables the hardware components of the processor and instructions to start executing. Its simulation result is shown in Fig. 3.

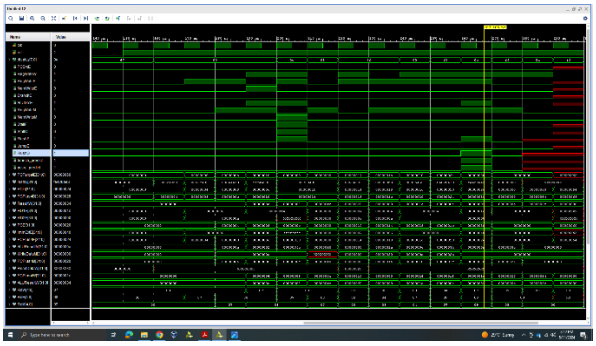

Figure 3 RISC V pipeline simulation results with the Control Hazard unit and Branch Predictor

6. FPGA IMPLEMENTATION

FPGA implementation involves several key steps. First, understanding FPGA technology is crucial; FPGAs are programmable devices featuring Configurable Logic Blocks (CLBs), Input/Output Blocks (IOBs), memory blocks, and routing fabric, making them versatile for various applications. The design flow begins with selecting the right FPGA board, such as the ZedBoard Zynq-7000, and setting up the development environment using Xilinx Vivado

software. Designers then create the Pipeline RISC-V processor using HDL, define I/O constraints in an XDC file, and proceed with synthesis to convert HDL code into a gate-level netlist. Implementation translates this netlist into a format suitable for placement and routing on the FPGA. After synthesis and implementation, the bitstream file is generated to program the FPGA, followed by using the hardware manager to load and configure the bitstream. External peripherals like 7-segment displays are connected to visualize results. The hardware implementation is shown in Figure 4.

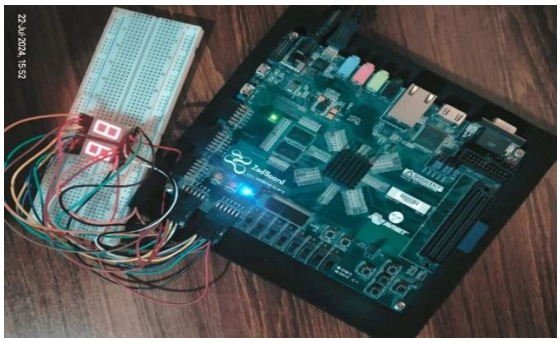

Figure 4 External 7-segment display connected with Pmod headers of zed board FPGA

7. RESULTS AND DISCUSSION

The RISC-V architecture is an open instruction set architecture that has achieved remarkable success in industry and academia. Let's examine the

I. Performance Analysis

Pipeline efficiency in a RISC-V processor is heavily affected by control hazards, particularly from branch instructions, which can cause performance reduction and pipeline stalls. To mitigate this, a Control Hazard Unit and a Static Branch Predictor are employed. The Static Branch Predictor predicts branch outcomes during the decode stage, reducing the number of clock cycles wasted and minimizing pipeline stalls. Without prediction, two clock cycles are lost per branch instruction due to flushing instructions already fetched and decoded. The static branch predictor, with a 100% accuracy rate, helps save one clock cycle per branch instruction, thereby improving the overall pipeline throughput.

II. Challenges and Solutions

In pipelined processors, hazards occur when instructions depend on others, causing data or control issues. Techniques like forwarding and stalling in the control hazard unit address these. An ideal pipeline has a CPI of 1, but mispredicted branches raise CPI. As pipelines grow more complex, mispredictions delay resolution, leading to instruction flushing. A total no. of 15 instructions (4 R-type, 3 I-type, 4 S-type, 4 branch instruction) is stored initially in instruction memory and after execution, achieves CPI of 1.53 which causes a reduction in performance. To counter this, a static branch prediction technique is used in the decode stage to identify branches early, reducing delays. This is covered in detail in the branch prediction section. After implementing the static branch predictor, CPI improves from 1.53 to 1.26 with the same instructions.

8. CONCLUSION

This paperinocuses on designing, simulating, and implementing a pipeline RISC-V processor using FPGA, to educate

beginners on processor architecture and Verilog coding. Key limitations include static branch mispredictions, delays from complex instructions, and increased power consumption from a deep pipeline. Future research should improve branch prediction, optimize pipeline depth, enhance memory access, prioritize energy efficiency, and refine exception handling. Additionally, multi-core pipeline designs and advancing the RISC-V ecosystem through standardization and better toolchain support are suggested for broader adoption and improved performance.

9. REFERENCES

[1] **Patterson, D.A**., **et al 2013**, Computer Organization and Design RISC-V Edition: The Hardware/Software Interface. Morgan Kaufmann.

[2] **Harris, D.M, et al 2012,** Digital Design and Computer Architecture. Morgan Kaufmann

[3] **Saveau, A. 2023,** Branch Prediction in Hardcaml for a RISC-V 32im CPU. ArXiv. /abs/2312.10426

[4] **Sharma, S. 2024,** RISC-V Architecture: A Comprehensive Guide to the Open-Source ISA. Wevolver. https://www.wevolver.com/article/risc-v-architecture

[5] **Kumar, M. et al, 2011**, FPGA based implementation of 32-bit risc processor." International Journal of Engineering Research and Applications 1, no. 3 pp. 1148-1151.

[6] **Katke, et al, 2012,** Design and implementation of 5 stages pipelined architecture in 32-bit RISC processor, International Journal of Emerging Technology and Advanced Engineering 2, no. 4, pp. 340-346.

[7] **Rathi, et al, 2020**, Design and development of an efficient branch predictor for an in-order RISC-V processor, Journal of Nano-and Electronic Physics 12, no. 5

[8] **P. S. Mane, et a;, 2006**, Implementation of RISC Processor on FPGA, 2006 IEEE International Conference on Industrial Technology, Mumbai, India, 2006, pp. 2096-2100, doi: 10.1109/ICIT.2006.372448.

[9] **Zain. (n.d.), 2024.** GitHub - zain2244/RISC-V-FPGA. GitHub. https://github.com/zain2244/RISC-V-FPGA

10. ACKNOWLEDGMENT

This research work was supported by System-on-Chip Design Laboratory (SoC), Department of Electronics, Faculty of Natural Sciences, Quaid-i-Azam University, Islamabad, Pakistan.