Design and Simulation of 8-bit RISC Processor using Verilog HDL

Mrs. J. Sravanthi[1], P. Radhika[2], N. Pavan kumar[3], M. Shiva kumar[4], M. Shashi Reddy[5]
[1] *Assistant Professor, Dept. of ECE, TKR College of Engineering and Technology*
[2,3,4,5] *Student, Dept. of ECE, TKR College of Engineering and Technology*

***Abstract*—8-bit Reduced Instruction Set Computing (RISC) processor implemented using Verilog Hardware Description Language (HDL). The proposed processor adopts a Harvard architecture with separate instruction and data memories to enhance execution efficiency. A pipelined architecture is incorporated to improve overall system performance, enabling the execution of one instruction per clock cycle under ideal conditions. The processor consists of key functional units, including an arithmetic logic unit (ALU), register file, control unit, program counter, and memory modules. A simple yet efficient instruction set is designed to support fundamental arithmetic, logical, and data transfer operations. The design is described using Verilog HDL and simulated using industry-standard simulation tools to verify functional correctness.**

***Index Terms*—8-bit RISC Processor, Verilog HDL, Harvard Architecture, Pipelining, FPGA Implementation, Arithmetic Logic Unit (ALU), Instruction Set Architecture (ISA), Simulation.**

I. INTRODUCTION

The increasing demand for high-performance and energy-efficient computing systems has led to significant advancements in processor design methodologies. Reduced Instruction Set Computing (RISC) architecture has emerged as a popular approach due to its simplicity, faster execution, and efficient hardware utilization. Unlike Complex Instruction Set Computing (CISC), RISC processors use a limited set of instructions that can typically be executed within a single clock cycle, improving overall system speed. This paper focuses on the design and simulation of an 8-bit RISC processor using Verilog Hardware Description Language (HDL), which is widely used for modeling and implementing digital systems. The proposed processor adopts a Harvard architecture, where separate memory units are used for instructions and data, allowing simultaneous access and improving throughput. To further enhance performance, pipelining is incorporated, enabling multiple instructions to be processed in different stages of execution concurrently. The processor design includes essential components such as an Arithmetic Logic Unit (ALU), register file, control unit, program counter, and memory modules, all integrated to perform efficient computation. Verilog HDL is used to describe the structural and behavioral aspects of the processor, ensuring modularity and scalability in design. Simulation plays a crucial role in validating the correctness and performance of the processor before hardware implementation. The proposed 8-bit RISC processor serves as a foundational model for educational purposes and can be extended for more complex embedded system applications. The modular design approach adopted in this work allows individual components of the processor to be tested and optimized independently, improving design reliability and ease of debugging. Furthermore, the proposed design can be implemented on FPGA platforms, providing a practical and cost-effective solution for real-time hardware validation and prototyping.

II. METHODOLOGY

The design of the 8-bit RISC processor is carried out using a structured and modular approach in Verilog HDL. The processor architecture follows the Harvard model, consisting of separate instruction memory and data memory to allow parallel access and improve execution speed. The overall system is divided into several functional blocks, each responsible for a specific operation, and all modules are interconnected to ensure smooth data flow and control.

At the core of the processor is the Program Counter (PC), which holds the address of the next instruction to be executed. The PC updates sequentially or branches to a new address based on control signals generated during execution. The instruction memory stores the program instructions, which are fetched using the PC during the instruction fetch stage.

The Instruction Memory use to Stores all program instructions in a fixed 24-bit format for execution by the processor. It is a read-only memory unit during normal operation, ensuring that instructions are not modified while executing. The memory is organized as 256 locations, each capable of storing one instruction. It is accessed using the instruction address provided by the Program Counter (PC). It supports sequential instruction fetching, where instructions are read one after another.

The fetched instruction is passed to the Instruction Decoder (Control Unit), which interprets the opcode and generates the necessary control signals for other components. These signals determine the type of operation to be performed, such as arithmetic, logical, or data transfer operations. The control unit plays a critical role in coordinating all stages of the processor. The Control Signal Generator Acts as the central unit that generates control signals to manage and coordinate all processor operations. It receives decoded information from the Instruction Decoder and determines the required control actions. It controls the operation of key components such as the ALU, register file, memory units, and multiplexers

The Register File consists of a set of general-purpose registers used to store intermediate data and operands. It supports read and write operations, allowing quick access to data required for processing.

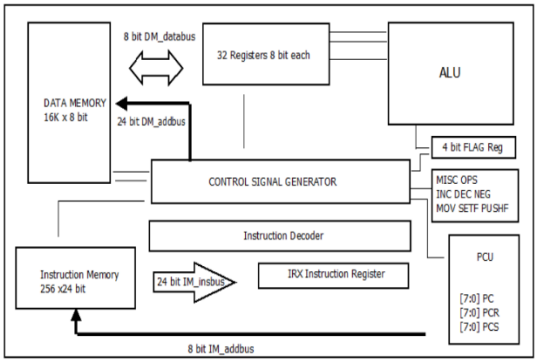

Fig 1: 8-bit RISC processor

The operands from the register file are fed into the Arithmetic Logic Unit (ALU), which performs operations such as addition, subtraction, AND, OR, and comparison.

To enhance performance, a pipelining technique is implemented, typically consisting of stages such as Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB). This allows multiple instructions to be processed simultaneously at different stages, thereby increasing throughput.

Each module is designed and coded separately in Verilog HDL and then integrated to form the complete processor. Simulation is performed to verify the functionality of each block and the overall system behavior. Test benches are used to validate different instruction executions and ensure correct timing and data flow across the processor.

III. IMPLEMENTED DESIGN

In an 8-bit RISC processor, pipelining is commonly implemented using a 5-stage pipeline: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB). Between each stage, pipeline registers are used to store intermediate results and control signals, ensuring smooth data flow.

Once the pipeline is fully filled, the processor can complete one instruction per clock cycle, which greatly improves efficiency compared to non-pipelined execution.

However, pipelining also introduces challenges called hazards. These include:

- Data hazards, where an instruction depends on the result of a previous instruction
- Control hazards, caused by branch instructions
- Structural hazards, due to hardware resource conflicts

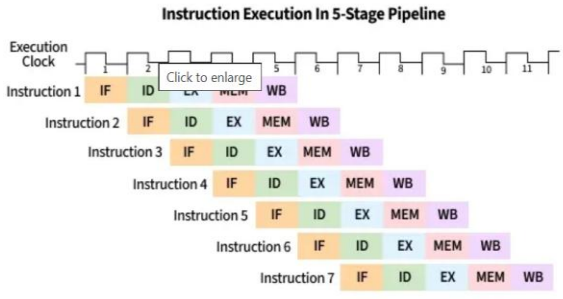

Fig 2: 5-stage pipeline

i. Instruction Fetch (IF)
In the Instruction Fetch stage, the processor retrieves the instruction from the instruction memory using the address stored in the Program Counter (PC). The PC acts as a pointer that always holds the address of the next instruction to be executed.

ii. Instruction Decode (ID)In the Instruction Decode stage, the fetched instruction is analyzed to determine the type of operation to be performed. The instruction is divided into fields such as opcode, source registers, and destination register.

iii. Execute (EX)
In the Execute stage, the actual computation takes place using the Arithmetic Logic Unit (ALU). Based on the control signals, the ALU performs arithmetic operations like addition and subtraction or logical operations like AND, OR, and XOR

iv. Memory Access (MEM)
In the Memory Access stage, the processor interacts with data memory depending on the type of instruction. For load instructions, data is read from memory using the address computed in the Execute stage. For store instructions, data is written into memory.

v. Write Back (WB)
In the Write Back stage, the final result of the instruction is written back into the register file. The data written may come either from the ALU (for arithmetic and logical operations) or from memory (for load instructions).

IV. RESULT & DISCUSSION

4.1 Simulation Results
The simulation is carried out using a Verilog testbench. Where it consists of four main stages: Instruction Fetch, Decode, Execute, and Write Back. Each stage is connected through pipeline registers, allowing multiple instructions to be processed simultaneously. The ALU performs arithmetic and logical operations, while multiplexers control data flow based on control signals generated in the decode stage.

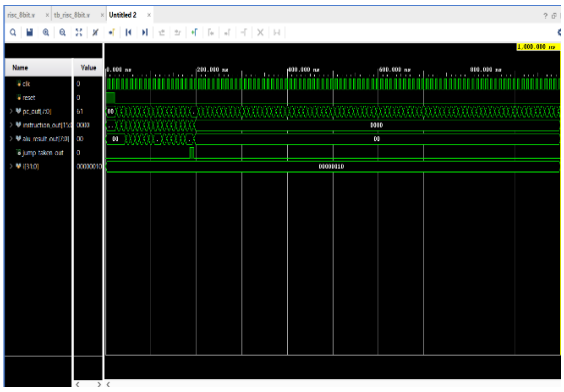

Fig 3: Waveforms

The waveform shows how different signals in the 8-bit RISC processor change over time during simulation. The clock signal continuously toggles, driving all operations. The reset starts low, allowing the processor to run normally. The program counter (pc_out) keeps increasing, indicating instruction flow. The instruction output remains mostly zero, suggesting limited or repeated instructions in memory. The ALU result also stays zero, meaning no arithmetic operation is producing a visible change.

4.2 RTL Analysis
The schematic represents the internal architecture of an 8-bit pipelined RISC processor. It consists of four main stages: Instruction Fetch, Decode, Execute, and Write Back. Each stage is connected through pipeline registers, allowing multiple instructions to be processed simultaneously. The ALU performs arithmetic and logical operations, while multiplexers control data flow based on control signals generated in the decode stage. The design improves performance by enabling parallel execution of instructions.

4.3 Power Analysis
The power analysis report shows that the total on chip power consumption of the system is approximately 7.97W, which is medium for an FPGA-based implementation. The dynamic power consumption is minimal, indicating efficient switching activity within the design. Most of the power consumption is due to static (leakage) power, which is common in modern FPGA devices. The low power consumption makes the design suitable for energy efficient applications such as wireless and portable systems.

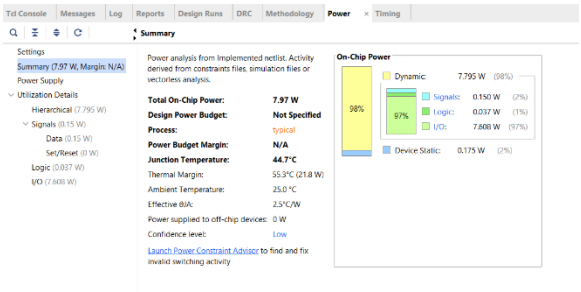

Fig 4: Power analysis

V. CONCLUSION

This "Design and Simulation of a 8-bit RISC Processor," focused on understanding the basic working of a processor and improving its performance using pipelining. The main goal of the project was to design a simple yet efficient 8-bit RISC processor and demonstrate how pipelining can increase the speed of instruction execution. By dividing the instruction process into multiple stages, the processor can handle several instructions at the same time, which leads to better overall performance. During the development of this project, several important tasks were completed step by step. First, the overall architecture of the 8-bit RISC processor was designed. Then, different modules such as the Arithmetic Logic Unit (ALU), registers, control unit, and memory interface were implemented. After that, the pipelining stages—Fetch, Decode, Execute, Memory Access, and Write Back—were developed and properly connected. Special attention was given to how data flows between these stages.

REFERENCES

[1] M. Morris Mano and Michael D. Ciletti, Digital Design with an Introduction to the Verilog/VHDL, Pearson Education, 2017. – Provides fundamentals of digital design and HDL-based processor implementation.

[2] Douglas L. Perry, VHDL Programming by Example, McGraw-Hill Education, 2017. – Covers VHDL modeling techniques useful for designing RISC processors.

[3] John L. Hennessy and David A. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 2019. – Explains RISC architecture concepts and performance optimization.

[4] IEEE Xplore Digital Library, "Design and Implementation of 8-bit RISC Processor using VHDL," 2018–2024. – Research papers on FPGA-based processor design and high-speed architectures.

[5] Springer, "FPGA-Based Embedded System Design Using VHDL," 2020. – Discusses FPGA implementation and optimization techniques.

[6] Elsevier Journal of Microprocessors and Microsystems, 2021–2025. – Contains studies on processor performance, pipelining, and embedded applications.

[7] Xilinx Documentation, Vivado Design Suite User Guide, 2022. – Practical reference for implementing VHDL designs on FPGA.

[8] Jikku Jeemon, "Low Powered Pipelined 8-Bit RISC Processor Design and Implementation on FPGA"(ICCICCT), pp.476-481, 2015.

[9] Akshatha S Patil, B G Shivaleelavathi, "Design and Implementation of Pipelined 8 bit RISC Processor using Verilog HDL on FPGA", (IRJET), 2017, Vol.4, Issue 6, pp.1753-1756, June 2017.

[10] Aishwarya H S, Sujatha Hiremath, "Design of Low Power High Speed 8-bit RISC Processor", International Research journal of Engineering and Technology (IRJET), Vol. 7, Issue 6, pp.1051-1056, June 2020.

[11] Ramandeep Kaur, Anuj, "8 Bit RISC Processor Using Verilog HDL", Anuj et al Int. journal of Engineering Research and Applications, Vol. 4, Issue 3, pp.417422, March 2014.

[12] R. Uma. "Design and Performance Analysis of 8-bit RISC Processor using Xilinx Tool," International Journal of Engineering

[13] Samiappa Sakthikumaran, S. Salivahanan and V.S. Kaanchana Bhaaskaran, June 2011, "16-Bit RISC Processor Design for Convolution Application" pp.394-397.

[14] Pravin S. Mane, Indra Gupta, M. K. Vasantha, Implementation of RISC Processor on FPGA, 1-4244- 0726-5/06, 2006 IEEE.

[15] Charles H. Roth, Jr., "Digital Systems Design using VHDL", The University of Texas at Austin. 2006 reprint, Thomson Asia Pte Ltd, Singapore

[16] Intel FPGA Documentation (Quartus Prime), 2023. – Guides for synthesis and high-speed hardware design using VHDL
[17] S. Knowles. A family of adders. In IEEE symposium on Computer Arithmetic.
[18] P.M.Kogge and H.S.Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. In IEEE Transactions on computers.
[19] Kevin Skahill, "VHDL for Programmable Logic", Pearson education, 2006
[20] "Design Hubs", https://www.xilinx.com/support/documentationnavigation/design-hubs/vivado-design-suite.ht