Design & Verification of 16-Bit Risc Processor Using Vedic Mathematics

M. Jhansi Rani[1], P. Bhavani[2], P. Sanjay[3], P. Anantha Raghavan[4], N. Bharath 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-The rapid advancement of digital systems demands processors that are not only fast but also efficient in terms of power and hardware utilization. This work presents the design and verification of a 16-bit Reduced Instruction Set Computing (RISC) processor enhanced with Vedic mathematics techniques. RISC architecture is known for its simplified instruction set, enabling faster execution and easier pipelining. In this design, Vedic mathematical algorithms, particularly from ancient Indian computation methods, are incorporated into the Arithmetic Logic Unit (ALU) to improve arithmetic performance. The processor is model using hardware description language and verified through simulation tools to ensure functional correctness. The proposed approach demonstrates improvements in speed and computational efficiency while maintaining a compact hardware footprint. This makes it suitable for embedded systems and real-time applications where performance and resource optimization are critical.

Keywords-16-bit RISC Processor, Vedic Mathematics, ALU, FPGA, HDL, Processor Design, Verification, Embedded Systems.

I. INTRODUCTION

In modern computing, processors play a crucial role in executing instructions efficiently. Among various processor architectures, RISC has gained popularity due to its simplicity, reduced instruction complexity, and faster execution cycles. Unlike Complex Instruction Set Computing (CISC), RISC processors use a limited set of instructions, which allows optimized performance and easier hardware implementation.

A key challenge in processor design is improving arithmetic computation speed without increasing hardware complexity. Arithmetic operations such as multiplication and addition significantly affect processor performance. To address this, Vedic mathematics offers efficient algorithms that can be implemented in digital systems. These algorithms, derived from ancient Indian mathematics, provide faster computation techniques compared to conventional methods.

This project integrates Vedic mathematical principles into a 16-bit RISC processor design. The focus is on enhancing the ALU, which is responsible for executing arithmetic and logical operations. By combining RISC architecture with Vedic algorithms, the processor achieves better speed and efficiency. The design is implemented and verified using simulation tools to ensure reliability and correctness.

II. METHODOLOGY

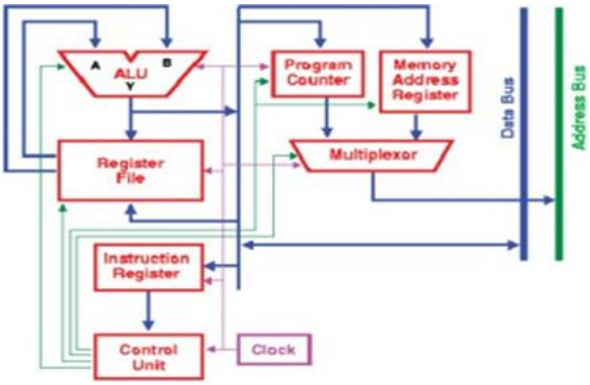

Fig 1: Block Diagram

1. ALU (Arithmetic Logic Unit): Performs arithmetic and logical operations like addition, subtraction, AND, OR.

2. Register File: Stores temporary data and operands used by the ALU.

3. Program Counter (PC): Holds the address of the next instruction to be executed.

4. Memory Address Register (MAR): Stores the address for memory access.

5. Multiplexer (MUX): Selects between different inputs for processing.

6. Instruction Register (IR): Holds the current instruction being executed.

7.Control Unit: Generates control signals to coordinate all operations.

8. Data Bus & Address Bus: Used to transfer data and addresses between processor and memory.

9.Clock: Synchronizes all operations.

The methodology for designing and verifying a 16-bit RISC processor using Vedic mathematics focuses on improving computational efficiency and hardware performance. The design begins with defining the processor architecture, including instruction set, register file, ALU, control unit, and memory interface. A simplified RISC architecture is adopted to ensure faster execution and reduced complexity. Vedic mathematics algorithms, particularly the Urdhva Tiryakbhyam (vertical and crosswise) technique, are integrated into the Arithmetic Logic Unit (ALU) to enhance multiplication speed and reduce delay.

The processor model using a hardware description language such as Verilog or VHDL. Each module—such as instruction decoder, ALU, and registers—is developed and tested individually before integration. After completing the RTL design, simulation is performed using tools like Xilinx Vivado to verify functional correctness. Test benches are created to validate different instruction executions and arithmetic operations.

For verification, both functional and timing analysis are carried out to ensure correct operation under various conditions. Synthesis and implementation are then performed on FPGA to evaluate area utilization, power consumption, and speed. Finally, the results are analyzed and compared with conventional designs to demonstrate the performance improvement achieved using Vedic mathematics.

III. IMPLEMENTED DESIGN

The implemented design of the 16-bit RISC processor focuses on achieving high-speed computation with optimized hardware utilization by integrating Vedic mathematics into the Arithmetic Logic Unit (ALU). The processor follows a simple and efficient RISC architecture, consisting of key modules such as the instruction fetch unit, instruction decoder, register file, ALU, and control unit. Each module is designed to execute instructions in a streamlined manner, reducing complexity and improving execution speed.

A significant enhancement in this design is the use of Vedic multiplication techniques, particularly the *Urdhva Tiryakbhyam* method, within the ALU. This approach enables faster arithmetic operations compared to conventional multipliers by allowing parallel generation of partial products. As a result, the processor achieves reduced delay and improved performance.

The register file stores intermediate data, while the control unit manages instruction execution and data flow between components. The entire design is implemented using Hardware Description Language (HDL) and verified through simulation tools to ensure correct functionality. The processor is then synthesized and tested on FPGA, confirming its efficiency in terms of speed, area, and power consumption.

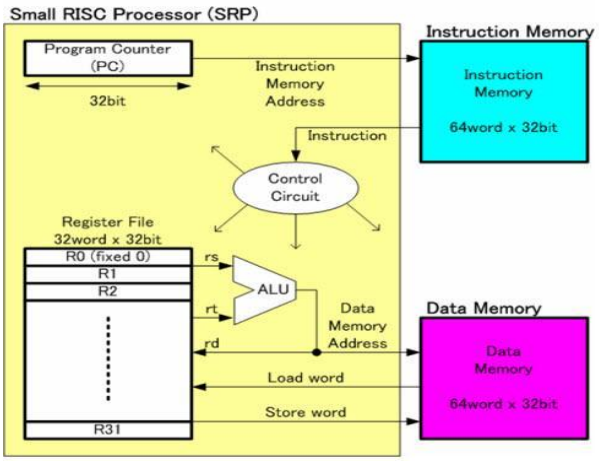

Fig 2: RISC Processor

The process begins with the Program Counter (PC), which holds the address of the next instruction. This address is sent to the instruction memory, where the Instruction Fetch Unit (IFU) retrieves the instruction.

The fetched instruction is then passed to the Instruction Decoder, which interprets the opcode and generates the required control signals.

Next, the decoded instruction accesses the Register File, where operands are read from selected registers. These operands are forwarded to the Arithmetic Logic Unit (ALU). In this design, the ALU is enhanced with a Vedic multiplier based on the *Urdhva Tiryakbhyam* algorithm, enabling faster multiplication through parallel partial product generation.

The Control Unit plays a crucial role by coordinating all operations. It controls data movement between registers, ALU operations, memory access, and updates to the PC. Depending on the instruction type (arithmetic, logical, load/store), the control signals adjust the data path accordingly.

IV. SIMULATION RESULTS

1)Wave Form

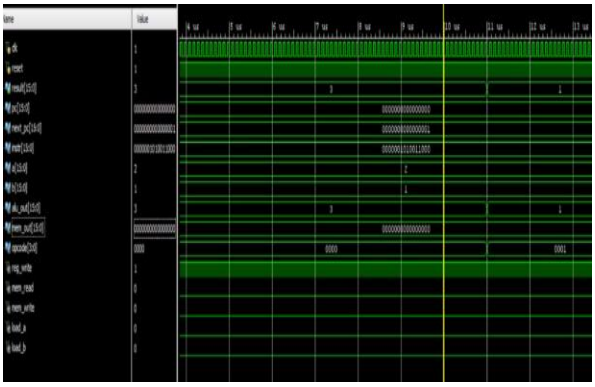

Fig 3: Result Waveform

The simulation output of a 16-bit RISC processor, where different signals correspond to instruction execution over time. The horizontal axis indicates clock cycles, and each transition reflects changes in control and data signals triggered by opcodes.

At the beginning, the processor is in an idle or reset state, with most signals set to zero. Once the clock becomes active, the Program Counter (PC) starts incrementing, fetching instructions from memory. Each instruction is identified by its opcode, which determines the operation performed by the ALU.

For example, when the opcode corresponds to LOAD (e.g., 0001), the waveform shows memory read signals becoming active, and data is transferred into a register. For an ADD operation (e.g., 0010), the ALU inputs change, and the result is reflected in the output bus after a clock cycle. Similarly, SUB (0011) shows subtraction activity, where operand values change and the result is updated accordingly.

Control signals such as Reg Write, Mem Read, and Mem Write toggle based on the opcode, indicating whether data is being written to registers or memory. The stable flat lines represent holding states, while transitions indicate execution phases.

Overall, the waveform verifies correct instruction decoding and execution, confirming that each opcode triggers the intended operation within the processor.

2) RTL Schematic

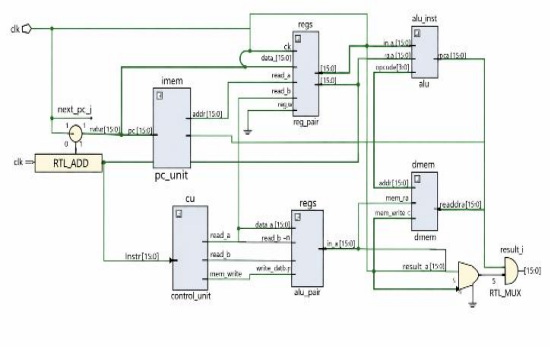

Fig 4: RTL schematic

The process begins with the Program Counter (PC), which generates the address of the next instruction. This address is passed to the instruction memory (imem), where the instruction is fetched. The fetched instruction is then sent to the control unit (CU), which decodes the opcode and generates the required control signals.

Based on these signals, the register file (regs) selects the appropriate operands. These operands are forwarded to the ALU (Arithmetic Logic Unit), where operations like addition, subtraction, or logical functions are performed depending on the instruction.

For memory-related instructions, the ALU output acts as an address for the data memory (dmem). Data can either be read from or written to memory depending on the control signals such as Mem Read and Mem Write.

Finally, a multiplexer (MUX) selects between the ALU result and memory output to produce the final result, which is written back into the register file. This completes one instruction cycle, ensuring smooth and efficient processor operation.

3) Power Utilization

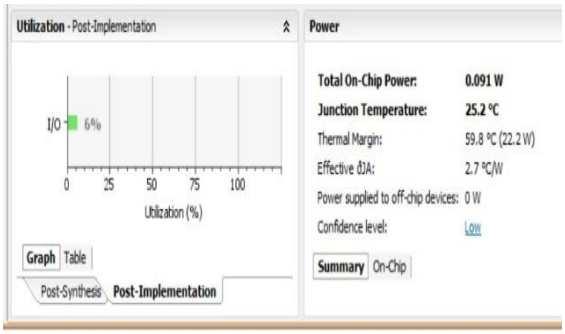

Fig 5: Power Utilization

The post-implementation utilization and power analysis of the designed system, which reflects how efficiently the hardware resources are used and how much power the circuit consumes after synthesis and implementation.

In the utilization section, the I/O usage is around 6%, indicating that only a small portion of available input/output pins on the FPGA are being used. This suggests that the design is compact and does not heavily depend on external interfacing, leaving room for future expansion or integration of additional modules.

On the right side, the power analysis provides important insights into the system's efficiency. The total on-chip power consumption is 0.091 W, which is very low and confirms that the processor design is energy-efficient. This is particularly beneficial for embedded systems where power saving is critical. The junction temperature is 25.2°C, which is close to room temperature, indicating that the device operates safely without overheating.

The thermal margin of 59.8°C shows a large safety buffer before reaching critical temperature limits. Additionally, the effective thermal resistance (θJA) of 2.7°C/W reflects good heat dissipation capability. Overall, the results demonstrate that the design is optimized for low power and reliable operation.

V. CONCLUSION

The project on design and verification of a 16-bit RISC processor using Vedic mathematics successfully demonstrates an efficient and optimized processor architecture. The integration of Vedic mathematical techniques, especially in the Arithmetic Logic Unit (ALU), improves computational speed while reducing hardware complexity. This approach enables faster arithmetic operations compared to conventional methods, making the processor suitable for high-performance and real-time applications.

The design was implemented using hardware description language and verified through simulation and synthesis tools. The waveform results confirm that all instructions are executed correctly based on their opcodes, ensuring proper functioning of modules such as the Program Counter, Control Unit, Register File, ALU, and Memory. Additionally, the post-implementation results show low resource utilization and minimal power consumption, which highlights the efficiency of the design.

Overall, the processor achieves a balance between speed, area, and power, making it ideal for embedded systems and digital applications. The project also provides a strong foundation for further enhancements, such as increasing bit width, adding pipeline stages, or integrating advanced instruction sets. This work proves that combining RISC architecture with Vedic mathematics can lead to improved processor performance and optimized hardware design.

REFERENCES

[1] Tiwari, K. S., "Design of Low-Power Vedic ALU Using Reversible Logic," *Microelectronics Journal*, 2025.
[2] Mugatkar, A., "Efficient Vedic Multiplier Architecture for High-Speed Applications," *International Journal of Modern Physics C*, 2023.
[3] Kumar, R. K., "Comparative Analysis of Vedic Multiplier Architectures," *Microprocessors and Microsystems*, 2024.
[4] Sona, M. K., "Vedic Multiplier Implementation in VLSI Systems," *Procedia Engineering*, 2020.
[5] Anjana, S., "High-Speed FloatingPoint Multiplier Using Vedic Mathematics," *Procedia Computer Science*, 2015.

[6] Rao,K. N., "Energy Efficient Vedic Multiplier Using GDI Technique," *IEEE Access*, 2024.
[7] Sathiya,A., "Modified Vedic Multiplier Using Nikhilam Sutra," *Scientific Reports*, 2025.
[8] Jaiswal, M. K., "Area Efficient Floating-Point Multiplier on FPGA," *Microelectronics Journal*, 2013.
[9] Kodali, R. K., "FPGA Implementation of Vedic Floating Point Multiplier," *IEEE Conference on SPICES*, 2015.
[10] Havaldar, S., "Design of IEEE 754 Vedic Multiplier," *IEEE RTEICT Conference*, 2016.
[11] Zhang, H., "Flexible Multiply-Accumulate Unit for Neural Networks," *IEEE Transactions on Computers*, 2020.
[12] Raman, A., "High-Speed FFT Design Using Vedic Mathematics," *arXiv*, 2010.
[13] Arish, S., "Floating Point Multiplier Using Karatsuba and Vedic Algorithms," *arXiv*, 2019.
[14] Thapliyal, H., "VLSI Implementation of RSA Using Vedic Mathematics," *arXiv*, 2006.
[15] "Design of High-Speed Vedic Multiplier," *International Journal of Scientific Research Publications*, 2012.
[16] "Design and Hardware Implementation of Vedic Multiplier," *IJERT*, 2017.
[17] "Low Power Multiplier Architectures Using Vedic Mathematics," *IJAREEIE*, 2017.
[18] "Comparative Study of Vedic Multipliers Using Adder Architectures," *ResearchGate Publication*, 2025.
[19] "Time-Area Efficient Multiplier Using Vedic Techniques," *Engineering Papers Journal*, 2018.
[20] Kumar, A. S., "Double Precision Floating Point Multiplier Using Vedic Mathematics," *Nature Scientific Reports*, 2026.