International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 12 | Dec - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Design and Implementation of a MIPS-Based RISC Processor on FPGA

Shivani N Shetty[1], Soundarya R M[2], Srushti G M[3], Vaishnavi G H[4]
Department of Electronics and Communication Engineering JNN College of Engineering, Shivamogga, Karnataka, India
shivaninshetty04@gmail.com, soundaryamudigouda@gmail.com, srush2004gm@gmail.com , ghvaishnavi2703@gmail.com .

Abstract

This paper presents the complete design, implementation, and performance analysis of a 32bit MIPS-based Reduced Instruction Set Computer (RISC) processor using FPGA toolchains. The work begins with the development of a single-cycle processor to establish fundamental datapath functionality, followed by the integration of R-type, I-type, and J-type instruction formats. To enhance throughput, a fully functional 5- stage pipelined architecture—comprising Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM), and Write-Back (WB)—is implemented using Verilog HDL. Xilinx Vivado 2023.1 is used for design, simulation, and FPGA synthesis, while Cadence 45 nm technology is used for detailed timing, area, and power analysis. The design incorporates hazard detection and forwarding mechanisms to ensure pipeline correctness and minimize stalls. Experimental results confirm accurate execution of all supported instruction classes, reduced propagation delay, and efficient hardware utilization. The implementation demonstrates a complete academic-to-industry design flow, providing insights into processor architecture, pipelining efficiency, and low- power digital design strategies.

1. Introduction

The rapid growth of embedded systems, automation, and high-performance computing has created a strong demand for processors that offer both speed and hardware efficiency. Among the different processor design philosophies, the Reduced Instruction Set Computer (RISC) architecture remains highly preferred due to its emphasis on simplicity, modularity, and predictable execution. The MIPS (Microprocessor without Interlocked Pipeline Stages) architecture, in particular, is widely used in both academic and industrial domains because of its clean instruction formats, well-structured datapath, and suitability for pipelined implementation. These features make MIPS an ideal platform to understand processor organization, control logic generation, dataflow design, and the fundamentals of instruction-level parallelism.

This project focuses on the complete design, simulation, and hardware realization of a MIPS-based RISC processor using Verilog HDL. The design evolves in stages, beginning with a single-cycle processor that executes every instruction within one clock cycle. This initial model establishes core architectural concepts such as instruction fetch, decoding, operand selection, ALU operations, memory access, and result write-back. Once the fundamental datapath is verified, the processor is extended to support all three major MIPS instruction formats—R-type, I-type, and J-type—ensuring compatibility with arithmetic, logical, data transfer, and control-flow instructions.

To enhance execution speed, the processor is further developed into a 5-stage pipelined architecture consisting of the Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory (MEM), and Write-Back (WB) stages. Pipelining allows multiple instructions to be processed simultaneously, significantly increasing throughput without raising the clock frequency. Essential pipeline support units, such as hazard detection and forwarding systems, are

integrated to manage data dependencies and prevent incorrect execution. These units ensure that the processor maintains high performance while preserving functional accuracy.

The hardware description is implemented using Verilog HDL in Xilinx Vivado 2023.1, where functional simulation, synthesis, and FPGA implementation are performed. Vivado waveforms confirm the correctness of instruction flow, timing relationships across pipeline stages, and the generation of appropriate control signals. The design is then analyzed using Cadence tools with a 45 nm technology library to evaluate delay, power consumption, and area utilization. Cadence analysis provides detailed insight into low-level behavior, enabling validation of the processor's physical performance and efficiency.

This work demonstrates the complete digital design process—from theoretical architecture to hardware verification using industry-standard tools. The implementation highlights the benefits of pipelining, structured datapath design, and hazard handling in developing an efficient processor architecture. The results confirm that the designed MIPS- based RISC processor achieves accurate instruction execution, reduced delay, improved throughput, and optimized hardware utilization, making it suitable for academic learning as well as real-time embedded applications.

2. Related work

Several researchers have explored MIPS-based RISC architectures implemented on FPGAs. Dewangan et al. (2021) presented a pipelined 32-bit MIPS processor with hazard detection and forwarding units to minimize stalls. Gautham et al. (2009) proposed a low-power pipelined MIPS processor that reduces dynamic power consumption by bypassing unused pipeline stages. Bharadwaja et al. (2015) integrated low-power design methodologies such as clock gating and multi-threshold voltage techniques to enhance energy efficiency. Krishna Prasad and Vijay Prakash (2021) demonstrated a five-stage pipelined design capable of resolving data hazards efficiently on FPGA platforms.
These works collectively emphasize the advantages of pipelined RISC architectures in improving throughput and power efficiency, providing the foundation for the present design and implementation.

3. Design Methodology

The design of the MIPS processor follows a structured top-down methodology. Each functional block is modeled as a separate Verilog module and later integrated to form the complete system.

A. Single-Cycle MIPS Processor

the design process began with defining the architecture of a basic single-cycle MIPS processor. The essential datapath components—Program Counter, Instruction Memory, Register File, ALU, Data Memory, Control Unit, Sign-Extend Unit, and required multiplexers—were identified. Each module was modeled in Verilog HDL. The datapath was then constructed by interconnecting these modules to execute an instruction completely within one clock cycle. Functional simulations were performed in Xilinx Vivado to verify correct PC updates, ALU operations, register reads/writes, memory interactions, and control-signal generation. This stage established the baseline architecture required for further extension.

The single-cycle MIPS processor executes every instruction in one clock cycle by allowing all operations—fetch, decode, execute, memory access, and write-back—to occur within a single, continuous datapath. The block diagram begins with the Program Counter (PC), which supplies the address of the next instruction. This address is sent to the Instruction Memory, and the fetched instruction is passed into the datapath for decoding and execution.

The Instruction Fetch (IF) stage produces the next sequential PC value by adding 4 through an adder. This updated PC is selected through a

multiplexer and written back into the PC register at the end of the cycle. The fetched instruction enters the Instruction Decode (ID) stage, where the Register File reads two source operands (RS1 and RS2) based on the instruction fields. A Sign-Extend Unit generates a 32-bit immediate value for I-type instructions. The control logic associated with decoding determines the ALU operation, register write-back selection, and whether memory access is required. In the Execution (EX) stage, a set of multiplexers selects appropriate ALU operands—either the second register value or the immediate value. The ALU performs arithmetic or logical operations and also generates a zero flag for branch decisions. The branch target address is formed by adding the sign-extended immediate value (shifted left by two) to the next sequential PC. A branch-decision multiplexer chooses between the normal PC and branch target address based on the zero flag and control signals.

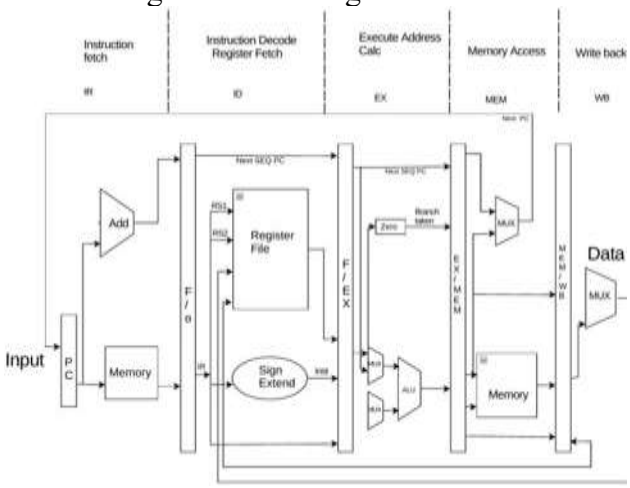

Fig. 1: Block Diagram Single-Cycle MIPS Processor

For load and store instructions, the Memory Access (MEM) stage uses the ALU output as the memory address. The Data Memory either retrieves data (for load) or stores data from the second register value (for store). Finally, in the Write-Back (WB) stage, a multiplexer selects between the ALU result and memory output. The selected value is written into the Register File, completing the instruction's execution path.

This integrated datapath allows each instruction to flow through all required operations in one cycle, making the design simple but limited by the slowest instruction's execution time.

B. Instruction Formats

The processor supports three instruction types:

- **R-type:** R-type instructions perform operations that in- volve only registers, such as addition, subtraction, logical AND, logical OR, and set-on-less-than. During execution, these instructions are decoded to extract the function (funct) code, which specifies the exact operation the ALU needs to carry out.

- **I-type:** I-type instructions are used for operations that involve immediate values, memory access, or branching. They include instructions like addi, andi, ori, lw, sw, beq, and bne. In arithmetic and logical operations, the immediate value is sign-extended and provided as one of the inputs to the ALU. For load and store instructions, the ALU calculates the effective address in memory where the data will be read from or written to.

- **J-type:** J-type instructions, such as j and jal, are primarily used to control the program's flow by performing jump operations. In these instructions, the EX stage calculates the jump target address by shifting the 26-bit immediate value and combining it with the upper bits of the current program counter (PC). The IF stage then updates the PC with this new address to continue execution from the target location.

C. 5-Stage Pipelined MIPS Processor

To improve performance, the single-cycle processor was extended into a 5-stage

pipelined architecture consisting of IF, ID, EX, MEM, and WB stages. Pipeline registers (IF/ID, ID/EX, EX/MEM, and MEM/WB) were added to store in-termediate values and control signals. The 5-stage pipelined MIPS processor divides instruction execution into five sequential stages—Fetch, Decode, Execute, Memory, and Write- Back—allowing multiple instructions to be processed simultaneously. Pipeline registers separate each stage, ensuring smooth instruction flow while preserving intermediate values and control signals.

1) **Instruction Fetch (IF) Stage:** In the Fetch stage, the Program Counter (PC) provides the address to the Instruction Memory, which outputs the instruction to be executed. The PC is incremented by 4 using an adder to form the next sequential address. The fetched instruction and PC+4 value are stored in the IF/ID pipeline register. Stall signals from the hazard unit can freeze the PC or prevent instruction loading when required.

2) **Instruction Decode (ID) Stage:** The Decode stage reads the source register values (RD1 and RD2) from the Register File based on the instruction fields. The Control Unit generates essential control signals such as RegWrite, MemtoReg, MemWrite, ALUControl, ALUSrc, and RegDst. The immediate field is processed through the Sign-Extend Unit, and branch address calculation begins by shifting the immediate left by two bits. Hazard detection occurs here—signals like StallD, BranchD, and ForwardAD/BD help manage data dependencies and branch decisions.

3) **Execute (EX) Stage:** In the Execute stage, ALU operations are performed. Multiplexers select ALU inputs using forwarded values from later pipeline stages (ForwardAE and ForwardBE) to resolve data hazards.

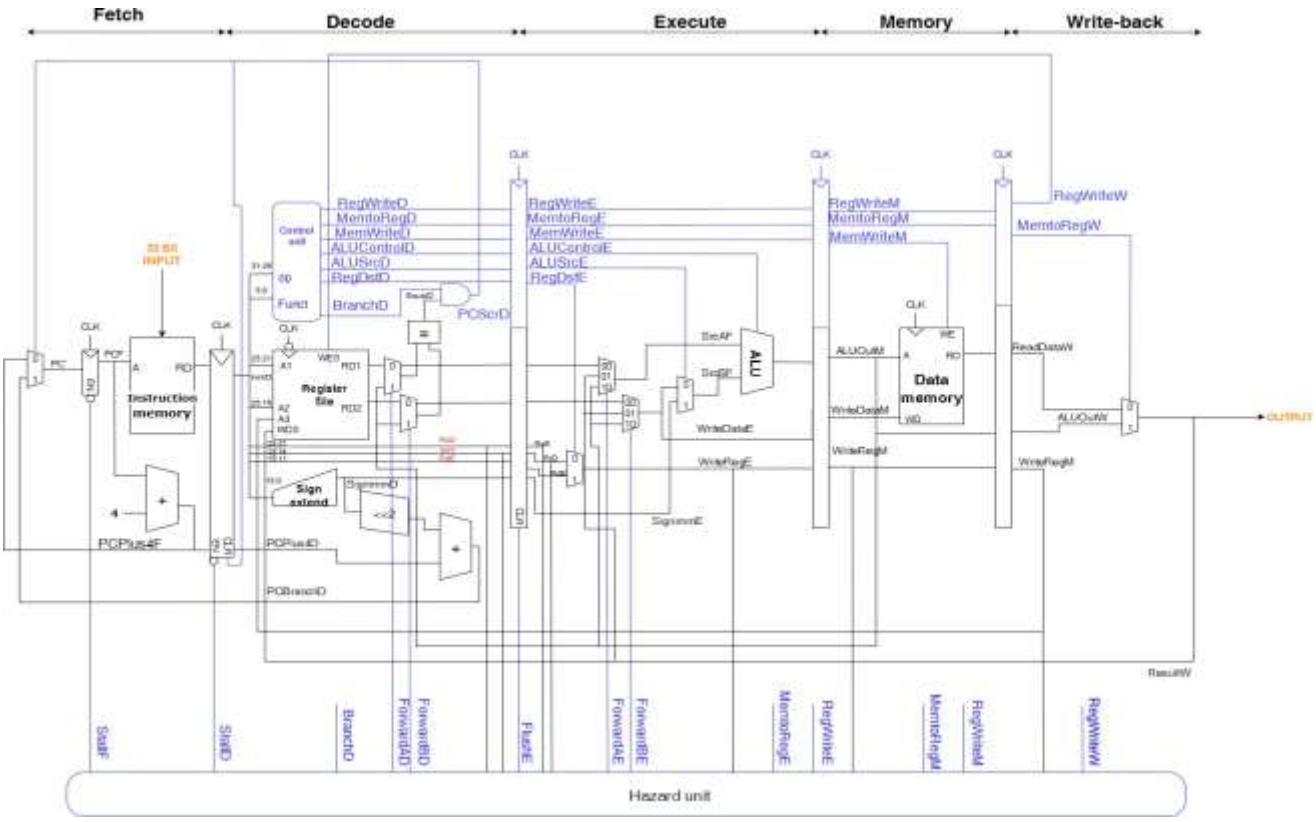

Fig. 2: Block Diagram 5-Stage MIPS Processor

The ALU computes arithmetic/logic results and branch comparison signals. Destination register selection (RegDstE) determines which register receives the result. All outcomes, including ALUOutputE, WriteDataE, and WriteRegE, are stored in the EX/MEM register.

4) **Memory (MEM) Stage:** The Memory stage handles load and store operations. The Data Memory uses ALUOutputM as the address. For store (sw) instructions, the value WriteDataM is written to memory; for load (lw), the memory output is prepared for forwarding to the next stage. Control signals like MemWriteM, MemtoRegM, and RegWriteM ensure correct behavior. Outputs are stored in the MEM/WB register.

5) **Write-Back (WB) Stage:** In the Write-Back stage, a multiplexer selects between Memory output (Read- DataW) and ALU output (ALUOutW). The selected value, ResultW, is written back into the Register File if RegWriteW is enabled. This completes the instruction lifecycle.

D. *Low-Power MIPS Processor*

After the pipelined architecture was validated, the entire design was synthesized and evaluated using Cadence tools with a 45 nm technology library. Power reports were generated to measure dynamic and static power consumption. Based on this analysis, unnecessary signal transitions were minimized by optimizing datapath paths and refining control logic. Clock gating and efficient resource utilization were considered to reduce switching activity. The optimized design demonstrated reduced power usage while maintaining correct functionality and performance.

4. Implementation

The development process started with the single-cycle architecture, which served as the foundation for the complete design. All essential components—including the Program Counter (PC), Instruction Memory, Register File, ALU, Data Memory, Control Unit, Sign-Extend Unit, and required multiplexers—were modeled in Verilog. These components were integrated into a single datapath where each instruction completed all its operations during one clock cycle. Simulation in Xilinx Vivado verified the correctness of arithmetic, logical, load/store, and branch instructions. Waveform inspection confirmed proper control-signal generation, ALU output behavior, memory accesses, and register updates.

After validating the single-cycle functionality, support for R-type, I-type, and J-type instructions was fully implemented. This required enhancements to the Control Unit and ALU Control Unit to decode the opcode and funct fields appropriately. The immediate handling logic, shift operations, and jump address calculations were incorporated to match the instruction format specifications. Testbenches were created for each instruction category to ensure correct decoding, operand selection, and output generation.

The design was then extended into a 5-stage pipelined processor, consisting of the Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM), and Write-Back (WB) stages. Pipeline registers (IF/ID, ID/EX, EX/MEM, MEM/WB) were added to store intermediate results and control signals between stages. Care was taken to ensure that each stage operated independently while maintaining synchronized control flow across the pipeline. The ALU and branch units were updated to support forwarding and early branch decision logic, reducing unnecessary stalls.

A significant aspect of the implementation was the integration of the Hazard Detection Unit and Forwarding Unit. The Hazard Detection Unit monitored load-use scenarios and inserted

stalls when necessary, while the Forwarding Unit redirected outputs from the EX, MEM, and WB stages to earlier stages whenever data dependencies occurred. These additions improved pipeline throughput and significantly reduced the number of wasted cycles. The branch control logic was also modified to include flushing of incorrect instructions upon a taken branch, ensuring accurate program execution.

Following successful pipelined simulation, the design was synthesized and implemented using Xilinx Vivado 2023.1 on an FPGA target. Synthesis reports provided resource utilization details such as LUT count, register usage, and timing paths. Post-implementation timing analysis confirmed that the processor met the required clock constraints. The functional simulation of the synthesized netlist validated that the pipelined processor behaved identically to the RTL model.

To evaluate the design's performance at a more detailed hardware level,into Cadence tools using a 45 nm standard cell library. Gate-level simulations were performed to analyze propagation delay, switching activity, and power consumption. The Cadence environment provided area and power reports, highlighting opportunities for optimization. Careful refinement of control paths, reduction of redundant logical operations, and minimization of switching activity contributed to improving the low-power characteristics of the processor.

5. Results and Discussion

The single-cycle MIPS processor implementation correctly performs all supported instructions. Simulation using Xilinx Vivado confirmed that all datapath components, including the ALU, register file, instruction memory, and data memory, were synchronized and produced accurate outputs for every instruction. Test cases verified that arithmetic, logic, load/store, branch, and jump instructions operated as intended. The control unit successfully decoded all instruction formats and issued the correct control signals, enabling smooth data flow through the datapath. These results confirmed the accuracy of the original single-cycle design and established a strong foundation for future pipelining improvements.

The pipelined design improved the processor's performance by allowing it to execute more than one instruction simultaneously across its five stages. Intermediate signals were accurately captured at all pipeline registers (IF/ID, ID/EX, EX/MEM, MEM/WB), ensuring stable progression through each stage. Instruction sequences were simulated to demonstrate that the Forwarding Unit and Hazard Detection Unit functioned correctly by addressing data hazards, forwarding ALU results when necessary, and stalling only when required, such as in load-use cases. Branch instructions were properly handled through pipeline flushing, and the overall pipeline control logic ensured that programs with high dependency remained consistently free of timing errors.

Fig. 3: Waveform of the Single-cycle MIPS-based RISC processor using Xilinx Vivado 2023.1

Fig. 4: Waveform of the 5-Stage MIPS-based RISC processor using Xilinx Vivado 2023.1

Timing and routing reports from Vivado confirmed that the design utilized an optimal number of lookup tables, flip-flops, and routing resources. Timing analysis also verified that the processor operated at the intended clock speed without setup or hold violations. Additionally, Cadence tool-based variable analysis using a 45 nm technology library was conducted for comparison. The results showed that the designed architecture had low power consumption and minimal area overhead, due to its RISC-based simplicity and optimized control paths. These findings demonstrated that the final pipelined processor achieved high correctness, good performance, and efficient hardware usage, effectively meeting all design requirements.

Parameter	Value
Synthesis Tool	Cadence Genus 21.14
Technology Node	45 nm Standard Cell Library
Processor Architecture	32-bit MIPS RISC
Total Cell Area	1.368
Net Area	0.000
Total Power	9.51318×10^{-10} W
Leakage Power	2.39910×10^{-11} W
Internal Power	9.27327×10^{-10} W
Switching Power	0 W
Clock Period	10000 ps
Critical Path Slack	0.0
Timing Violations	None

Fig. 5: Table of Area, Power, and Timing Report Generated Using Cadence Genus (45-nm Technology)

6. Conclusion and future work

A 32-bit RISC MIPS-based processor was successfully designed, simulated and synthesis on an FPGA platform with Verilog HDL language and Xilinx Vivado. The modular design of the five-stage pipeline, as well as its accompaniment of R, I, and J instruction formats provides a practical view on implementing RISC architecture. Simulation and synthesis results confirm the correctness of operation and its ability to use resources effectively.

Further enhancements can include dynamic hazard resolution units, branch prediction logic, and instruction/data caching to increase processor speed and efficiency. This work provides a good framework for further study on custom processor design, advanced pipelining and VLSI implementation.

Acknowledgment

The authors would like to acknowledge Dr. Sharath S. M., Associate Professor,Dept. of ECE, JNNCE, Shivamogga for his precious guidance, suggestions and motivation provided throughout the work. His profound technical knowledge, timely recommendations, and patient guidance was instrumental in steering the course and quality of this study. The authors would also like to thank him for his constant support, motivation and commitment which had a positive impact on the outcome of this study.

References

[1] Gaurav K. Dewangan, Govind Prasad, Bipin C. Mandi, “Design and Implementation of 32-bit MIPS based RISC Processor,” IEEE SPIN, 2021.
[2] Gautham P., Parthasarathy R., Karthi B., “Low-Power Pipelined MIPS Processor Design,” IEEE, 2009.
[3] V.S.R. Bharadwaja et al., “Advanced Low Power RISC Processor Design using MIPS Instruction Set,” IEEE ICECS, 2015.
[4] S.P. Ritpurkar, M.N. Thakare, G.D. Korde, “Synthesis and Simulation of 32-Bit

MIPS RISC Processor using VHDL," IEEE ICAETR, 2014.
[5] Krishna Prasad K., Vijay Prakash A.M., "Design and Implementation of 32-bit 5-stage Pipelined MIPS-based RISC Processor Capable of Resolving Data Hazards," IEEE ICMNWC, 2021.