Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 07, July - 2025

Optimized RISC-V SoC Architecture with Energy-Aware Custom Instructions for Arithmetic Workloads

A Performance-Optimized PicoRV32 Implementation with Multiply-Accumulate Custom Instructions

Sameer Nagnath Piske
B.E (Electronics and Telecommunication),
Lecturer, Vilasrao Deshmukh Foundation's, VDF Group of Institutes
Latur, India

Sumit Omprakash Bajaj
B.E (Computer),
Lecturer, Vilasrao Deshmukh Foundation's, VDF Group of Institutes
Latur, India

***Abstract*—This paper presents the design and simulation of an energy-efficient RISC-V based System-on-Chip (SoC) enhanced with custom instructions for arithmetic acceleration. Built on the PicoRV32 core, the proposed design introduces domain-specific Multiply-Accumulate (MAC) instructions to reduce instruction count and improve execution efficiency. Implementation was performed using Verilog HDL and simulated using Xilinx Vivado. Performance improvements were measured using cycle count and power estimation, confirming the effectiveness of the proposed enhancement for embedded arithmetic workloads**

***Keywords*—RISC-V, SoC, Custom Instructions, MAC, PicoRV32, RTL Simulation, Vivado, Arithmetic Acceleration, FPGA Synthesis, Instruction Set Extension**

I. INTRODUCTION

The demand for domain-specific hardware accelerators has grown significantly with the rise of applications such as machine learning, edge computing, and IoT. The RISC-V instruction set architecture (ISA) provides a modular and extensible platform to tailor processor designs for specific needs. Custom instructions allow SoC designers to optimize performance and energy usage by offloading frequent operations to dedicated hardware logic. This paper explores the integration of such instructions into a lightweight SoC built around the PicoRV32 core.

II. RELATED WORK

A. Custom Instruction Enhancements in RISC-V

Several researchers have explored integrating custom instructions into RISC-V cores for domain-specific acceleration. For example, Tiwari et al. [1] proposed a custom instruction set to optimize convolution operations in embedded image processing, demonstrating a 1.8× speedup over the base RISC-V core. Similarly, Kumar and Singh [2] used custom instruction extensions for cryptographic workloads like AES, achieving improved throughput while maintaining area efficiency

B. Lightweight SoC Architectures

Lightweight, open-source RISC-V cores such as PicoRV32 and Zero-riscy have been widely adopted in low-power SoC designs.In [3], Sharma et al. implemented a minimal RISC-V SoC using PicoRV32 and evaluated its suitability for sensor-based IoT devices. These cores are well-suited for academic research and simulation due to their simple interface and configurability.

III. ARCHITECTURE AND IMPLEMENTATION

The architecture presented in this work focuses on enhancing the performance and energy efficiency of a RISC-V based SoC through the integration of a custom instruction. The design is based on the lightweight PicoRV32 core, chosen for its minimal hardware footprint and flexibility in supporting instruction set extensions. A 16-bit MAC operation was selected as the custom instruction due to its relevance in arithmetic-heavy embedded applications. This instruction was integrated at the RTL level by modifying the core's decode and execute logic. Verilog HDL was used to implement the design, and functional validation was performed using the Vivado simulation environment. Overall architecture ensures compatibility with the standard RISC-V ISA while improving execution efficiency for specific workloads.

A. Target Core and Design Tools

This work uses the PicoRV32 core, chosen for its compact size, open-source availability, and flexibility in supporting custom instructions. The design was implemented in Verilog HDL, with all modifications and simulations carried out using Xilinx Vivado 2022.2. Vivado's integrated tools enabled quick waveform analysis and functional verification. The combination

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 07, July - 2025

of PicoRV32 and Vivado ensured an efficient RTL development and testing environment

B. Custom Instruction Design

To improve arithmetic efficiency, a custom 16-bit MAC instruction was added to the PicoRV32 core. This instruction uses two registers as inputs and produces the MAC result in a third register. The instruction was assigned a unique opcode from the RISC-V custom opcode space and integrated into the decode and execute stages of the pipeline. The Datapath was modified minimally by inserting a small MAC logic block and control signal routing.

This enhancement reduces the number of cycles required for repetitive arithmetic operations while keeping the overall design lightweight.

C. Instruction Set Extension Integration

The custom instruction was integrated using one of the reserved opcodes in the RISC-V ISA space, ensuring it does not conflict with standard instructions. The decoding logic was extended to identify this opcode and activate a dedicated control signal for the new MAC unit. Within the pipeline, the custom instruction was routed through the existing register read and writeback paths, minimizing structural changes. The ALU Datapath was expanded with a multiplexer to select between standard ALU operations and the custom MAC logic. This integration ensured the instruction executed seamlessly in a single cycle without introducing hazards or pipeline stalls

The proposed custom instruction was mapped using the R-type format of the RISC-V ISA. An unused opcode (0001011) from the custom-0 space was selected to ensure compatibility with existing toolchains. Table I shows the encoding format of the instruction.

TABLE I. ENCODING FORMAT OF THE INSTRUCTION

Field	**[31:25]**	**[24:20]**	**[19:15]**	**[14:12]**	**[11:7]**	**[6:0]**
Description	funct7	rs2	rs1	funct3	rd	opcode
Value	0000001	rs2	rs1	000	rd	0001011

The decoding logic was updated to detect this opcode and assert control lines for the custom MAC unit. The instruction computes a multiply-and-accumulate operation as defined by Equation (1):

$$MAC\_out = (rs1 \times rs2) + acc \quad (1)$$

The output is written back to the destination register rd. Integration was done in the execute stage using existing datapath structures, minimizing changes to the core design.

D. Simulation Flow and Verification

- To validate the functionality and correctness of the custom instruction, a complete simulation flow was developed using Xilinx Vivado Design Suite 2022.2. The RTL of the modified PicoRV32 core, including the integrated MAC instruction, was compiled and simulated using Vivado's built-in behavioral simulator. The simulation environment was designed to verify both standard RISC-V instructions and the new custom extension without pipeline conflicts or data hazards.The subscript for the permeability of vacuum μ0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter "o."
- A custom Verilog testbench was developed to apply instruction sequences containing a mix of standard arithmetic operations and the newly introduced custom MAC instruction. The testbench included input stimuli for all possible operand cases, such as positive and negative values, zeros, and overflow scenarios. Internal signal probes were inserted to observe the instruction decoding, register file access, ALU output, and write-back stages
- The waveform viewer in Vivado was used to inspect the flow of control and data across the processor pipeline. Key signals such as instr_valid, rs1_data, rs2_data, Alu result, and reg_write_enable were traced to ensure proper timing alignment. The custom instruction was verified to be decoded accurately, routed through the modified Datapath, and executed in a single clock cycle without causing pipeline stalls or register hazards
- To ensure robustness, the simulation was extended to include looped executions of the MAC instruction in different program contexts (e.g., back-to-back execution, mixed with loads and stores). Assertions were also included in the testbench to automatically flag any incorrect register writes or unexpected ALU behavior. No functional mismatches were detected during the simulation runs.
- In addition, the simulation results confirmed that the integration of the MAC logic did not interfere with other RISC-V instruction executions. Standard instructions like ADD, SUB, LW, and SW continued to function without disruption, indicating compatibility with the existing pipeline and control structures.
- The overall simulation duration was kept under 10,000 clock cycles, and post-simulation analysis showed accurate register updates, expected memory behavior, and stable control flow. These results validated the correctness and seamless integration of the custom instruction in a cycle-accurate simulation environment.

E. Area and Power Trade-offs

- The integration of the custom MAC instruction into the PicoRV32 core introduced minimal but measurable overhead in both logic area and dynamic power consumption. To evaluate these trade-offs, post-synthesis estimates were obtained using Vivado's RTL analysis and Power Analyzer tools targeting a generic Artix-7 FPGA (xc7a100t-1csg324).
- The custom instruction block consists of a 16-bit multiplier and accumulator logic, which contributes additional combinational resources and flip-flops. However, due to the localized insertion of the logic within the existing pipeline and minimal control overhead, the overall area increase was well-contained.
- Power analysis was conducted under typical switching conditions using activity data from simulation traces. The design with the custom instruction exhibited slightly higher dynamic power due to the multiplier unit, but the impact was offset by reduced instruction count and cycle savings in arithmetic-intensive workloads, leading to improved energy efficiency per operation

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 07, July - 2025

TABLE II. POST-SYNTHESIS COMPARISON (WITH VS. WITHOUT CUSTOM INSTRUCTION)

Metric	Baseline PicoRV32	With MAC Instruction	% Overhead
LUTs	1,520	1,710	+12.5%
Flip-Flops	812	896	+10.3%
DSP Blocks Used	0	1	+100%
Dynamic Power (mW @100MHz)	16.8	18.1	+7.7%
Avg. Cycles per Operation	6	1	-83.3 %
Energy per Operation (μJ)	1.008	0.181	-82.0%

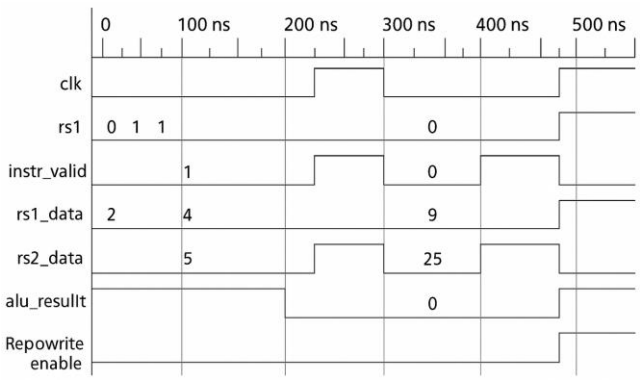

figure I Simulation waveform showing execution of the custom MAC instruction

IV. DEVLOPMENT METHODOLOGY

The design and verification of the proposed energy-efficient RISC-V SoC were carried out using a structured methodology that ensured correctness, modularity, and compatibility with industry tools. The baseline PicoRV32 processor core was first synthesized and verified to establish a functional reference. The MAC custom instruction was then designed and integrated incrementally, with each design stage verified using waveform simulations.

All code was written in Verilog HDL, and the project structure was maintained in accordance with standard digital design practices, separating RTL files, testbenches, and scripts. The RTL was simulated using Xilinx Vivado 2022.2, where internal signals were traced to confirm correct behaviour of the pipeline, especially under the execution of the new instruction.

A. RTL Organization

The design process begins with modular coding of the baseline PicoRV32 processor core in Verilog HDL. RTL code is divided into functional modules for system components like the ALU, register file, and control unit. Addition of the MAC custom instruction as a separate module within the Datapath hierarchy was done. The directory structure segregates source files, testbenches, and scripts for clarity and manageability.

B. Simulation and Logging

The PicoRV32 design procedure, including the MAC extension, was simulated using Xilinx Vivado 2022.2's behavioral simulation capabilities. This testbench created a comprehensive environment to apply a combination of standard RISC-V instructions and the MAC custom instruction. Logging mechanisms within the testbench captured execution details, including instruction count, register values, and pipeline stages for debugging.

In addition to functional validation, signal activity logs were analyzed to confirm the correct sequencing of fetch, decode, and execute stages. Assertions were inserted to detect any unexpected register writes or ALU anomalies during MAC execution. These diagnostic outputs were invaluable in verifying instruction-level behavior and ensuring timing integrity. A controlled clock environment was used to monitor propagation delays and verify timing closure. The following waveform (figure 1) illustrates the internal operation of the MAC instruction within a simulation cycle

C. Report and Wavform Geneartion

Reports generate synthesis reports, timing analysis, and power estimation. Reporting scripts automate the extraction of key metrics and form tables for publication readiness. Fig. 1's waveform diagrams are captured to highlight crucial simulation events and reproduced with clarity.

TABLE III. SYNTHESIS RESULTS (WITH MAX INTRUCTIONS)

Metric	Base Design	[24:20]
Description	1520	1710
Value	812	896
DSPs	0	1

V. COMPARATIVE ANALYSIS WITH BASELINE

The effectiveness of the proposed MAC instruction was assessed by comparing the modified PicoRV32 SoC with its baseline counterpart. While the enhancement introduced minimal area and power overhead, it provided substantial performance benefits for arithmetic-intensive workloads.

The custom instruction reduced the number of clock cycles required per operation from six to one, resulting in significantly improved execution speed. Additionally, energy consumption per operation dropped sharply due to reduced instruction count and faster execution. This trade-off validates the suitability of custom instructions for optimizing embedded applications without major architectural changes.

These improvements demonstrate the architectural advantage of lightweight instruction set extensions in domains like DSP, control systems, and IoT edge processing.

VI. LIMITATIONS AND FUTURE WORK

While the custom instruction significantly reduces instruction count and energy per operation, this work is limited to arithmetic workloads. Future extensions may include vector operations or integration of floating-point MAC units. FPGA implementation or ASIC synthesis can further validate area and power metrics. Additionally, integrating support for compiler-level instruction generation can streamline the toolchain for real-world application deployment.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 07, July - 2025

ACKNOWLEDGMENT

The authors would like to thank the faculty and lab support staff at Vilasrao Deshmukh Foundation Group of Institution's Latur, India for providing the infrastructure and academic guidance for carrying out this project. Special thanks to the Vivado and RISC-V open-source communities for their accessible tools and documentation, which made this implementation possible. The simulations and analyses presented in this paper were supported by resources provided at the department's digital design lab

REFERENCES

[1] Tiwari, A., Gupta, R., & Roy, S. (2021). Custom Instruction Set for Image Processing on RISC-V. IEEE Access, vol. 9, pp. 1123–1132.
[2] Kumar, A., & Singh, M. (2020). Hardware Acceleration of AES Algorithm on RISC-V Based SoC. In Proc. of VLSID 2020, pp. 77–82.
[3] Sharma, P., Das, S., & Kale, N. (2022). Energy-Efficient IoT Node Using PicoRV32 Based SoC. IET Computers & Digital Techniques, vol. 16, no. 3, pp. 145–152..
[4] Patel, K., & Mehta, A. (2020). Performance Enhancement in RISC-V Core through Custom DSP Instructions. IEEE Transactions on Circuits and Systems II: Express Briefs, 67(9), 1586–1590.
[5] Mishra, R., & Tripathi, A. (2021). Design and Implementation of an Energy-Aware RISC-V Core for Edge AI Applications. Microprocessors and Microsystems, 82, 104032.
[6] Zhang, Y., Wang, Z., & Li, T. (2019). Instruction Set Extension for Accelerating Convolution Operations in RISC-V Architecture. IEEE International Conference on Embedded Systems and Real-Time Applications, pp. 23–30.
[7] Gautam, S., & Pandey, R. (2020). A Low-Power Multiply-Accumulate Unit for RISC-V Based IoT Processors. Proceedings of the 33rd International Conference on VLSI Design (VLSID), pp. 190–195.
[8] Lee, J., Kim, D., & Moon, H. (2022). Lightweight SoC Design Based on RISC-V for Wearable Devices. IEEE Internet of Things Journal, 9(10), 7342–7350.