International Student Conference on Next-Gen Computing: Application of AI, Big Data, Quantum Computing, Signal Processing and Cloud Innovations (ICNGC-2026)

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

# Design and Performance Comparison of 32-Bit Risc-V Alu Accelerators: From Combinational to Pipelined Architecture with Flag Support

**Debika Rani Sahu[1], Tapas Kumar Patra[2], Debi Prasad Dash[3]**
[1, 2,3] School of Electronic Sciences, OUTR, Bhubaneswar, India

**Abstract:** The demand for more efficient and high-performing computing in embedded and edge systems has led to the creation of application-specific hardware acceleration on FPGAs. This includes an integrated two-stage pipelined Arithmetic Logic Unit (ALU) accelerator with an AXI4-Lite interface, designed for use in Zynq-based processing systems. Two design variants were explored: a baseline non-piped ALU and a pipelined ALU, both offering support for Zero, Carry, Overflow, and Negative flags. The proposed accelerator is implemented on the Xilinx PYNQ-Z2 FPGA board. The processing system communicates with the programmable logic through an AXI-based memory-mapped interface. A Python-based layer is used to set up, manipulate, check, and verify the hardware module, which facilitates prototype development and testing. The pipelined architecture balances manageable design complexity with computational throughput by overlapping instruction execution stages. Experimental evaluation shows that the pipelined design achieves 150 MHz and 150 MOPS in operating frequency and throughput, respectively. This demonstrates a 50% improvement over the non-pipelined version. The implementation incurs a modest resource usage penalty of 35% LUTs, and the overall power consumption stays below 0.1 W. These results highlight how effective pipelining is in enhancing ALU performance on FPGA platforms. It also confirms its suitability for high-performance embedded applications that need efficient hardware acceleration.

**Keywords:** Alu, Pipelined Risc-V Architecture, Axi4-Lite, Embedded Systems, Pynq-Z2

## I. INTRODUCTION

The growing need for high-performance and energy-efficient computing in embedded and edge systems has sped up the use of FPGA-based hardware accelerators. These accelerators are valued for their ability to process tasks in parallel, their flexibility, and their quick execution[6]. The introduction of the RISC-V instruction set architecture, which is open-source and customizable, allows for the creation of specific accelerators designed for various computational needs. At the heart of any processor, the Arithmetic Logic Unit (ALU) is crucial for overall system performance because it carries out arithmetic and logical operations[10]. Traditionally, combinational ALUs provide low-latency execution but have a limited operating frequency. On the other hand, pipelined ALUs increase throughput by breaking operations into multiple stages, though this adds design

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

complexity and uses more resources. Pipelining is a common technique that allows different stages of instruction execution to overlap, significantly boosting computational efficiency [2].

In this work, we implement two 32-bit RISC-V ALU accelerator architectures: a conventional combinational design and a two-stage pipelined design that supports full status flags (Zero, Carry, Overflow, and Negative). Both designs are integrated with the Zynq Processing System using an AXI4-Lite interface and are validated through a Python-based testing framework. We perform a thorough evaluation focusing on operating frequency, throughput, resource use, and power consumption. Experimental results show that the pipelined ALU offers a 50% performance improvement compared to the combinational design, with a moderate rise in hardware resource usage. This study outlines the trade-offs between performance and resource use, offering insights for choosing the right ALU architectures in FPGA-based embedded systems.

## II. RELATED WORK

A lot of research has been done to make Arithmetic Logic Units in RISC-V processors better. People like Jin and others have shown that making data paths better and changing the logic of Arithmetic Logic Units can make them work faster without making them too complicated [11]. Some Arithmetic Logic Units like the O-ALU use techniques to save power [18]. For example, they use clock gating. Only turn on the parts that are being used. However, these techniques do not always make the Arithmetic Logic Units work better.

Other people have made Arithmetic Logic Units that work in stages. This is called pipelining. It helps the Arithmetic Logic Units work faster. For example, Hussain and Sarka have shown that pipelining is an idea for RISC-V processors [9]. Some people have also made RISC-V processors using FPGAs. For example, Saif and others have shown that making the hardware work well is very important when resources are limited.

Most of the time people look at Arithmetic Logic Units that work in stages and those that do not separately. They also use conditions when they do their experiments. They often do not consider the flags that are necessary for the processor to make decisions. These flags are called Zero, Carry, Overflow and Negative. Not many people have compared the two types of Arithmetic Logic Units in a way. So this work puts both types of Arithmetic Logic Units on the FPGA platform. This platform is called PYNQ-Z2. By keeping everything the same we can compare the two types of Arithmetic Logic Units in a way. We can see which one uses resources works faster and uses less power. This will help us make Arithmetic Logic Units, for RISC-V processors that use FPGAs.

## III. DESIGN ARCHITECTURE

The ALU design architecture is implemented in two ways: pipelined. Both types of ALU designs are done on the PYNQ-Z2 FPGA to make a comparison. Each ALU design uses 32-bit data and works at 100 MHz. The ALU designs also talk to the Zynq Processing System using an AXI4-Lite interface.

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

The ALU designs support RISC-V operations like ADD, SUB AND OR, XOR, SLL, SRL and SLT. These ALU designs are made using Verilog HDL. Synthesized in Xilinx Vivado.

**Combinational ALU**

The combinational ALU does everything in one step. This means it can do things quickly in one clock cycle, which is very fast only 10 ns. However, the combinational ALU design has a problem. It can only work fast as its slowest part. This limits how fast the ALU design can go.

**ALU**

The pipelined ALU is different. It does things in two steps: Execute and Writeback. The Execute step does the work. Sets status flags. The Writeback step. Sends out the results. This makes the ALU design faster and reduces the time it takes to do things. It also means the pipelined ALU takes a little longer two clock cycles, which is 20 ns. It needs more hardware to work.

**Status Flags**

The pipelined ALU makes four status flags: Zero, Carry, Overflow and Negative. These flags help the ALU design do things and work better with RISC-V processor systems.
This way of showing the ALU design architecture clearly shows the differences, between the two ALU designs and what we gain or lose with each one. The ALU designs are important because they are part of the RISC-V processor systems and the PYNQ-Z2 FPGA. The ALU designs, including the ALU and the pipelined ALU are made to work with the AXI4-Lite interface and the Zynq Processing System.

## IV. METHODOLOGY OF IMPLEMENTATION

This section presents a description of the hardware and software implementation of both combinational and pipelined ALU architectures on the PYNQ-Z2 FPGA platform. A uniform design approach is strictly maintained across both implementations to ensure a unbiased and consistent comparison. All parameters such as data width, supported operations, clock frequency, interface protocol and synthesis settings are kept identical so that the observed differences arise from architectural variations of the ALU.

**Hardware Implementation Flow**

**RTL Design Using Verilog HDL**

The implementation process begins with the development of the ALU architectures at the Register Transfer Level (RTL) using Verilog HDL. The combinational ALU is designed as a single-stage logic unit, where all arithmetic and logical operations are performed within one clock cycle using a case-based structure. The pipelined ALU is divided into two stages: the Execute stage and the Writeback stage. The Execute stage performs the selected operation. Generates the corresponding status flags

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

while the Writeback stage registers the computed results and forwards them to the output interface. Both designs are developed with operational capabilities supporting standard RISC-V instructions such as ADD, SUB AND OR, XOR, SLL, SRL and SLT for the ALU.

**AXI4-Lite Interface Integration**

To enable communication between the ALU and the Processing System (PS) both designs are integrated with an AXI4-Lite slave interface. This interface provides a memory-mapped communication mechanism allowing the processor to send operands specify operations and retrieve results from the ALU. The interface includes registers, such as control, operand A, operand B, opcode and result registers. The control register initiates the computation while the operand and opcode registers define the inputs and operation type for the ALU.

**IP Core Packaging Using Vivado**

After RTL design and interface integration both ALU architectures are packaged as Intellectual Property (IP) cores using the Xilinx Vivado IP Packager tool. This process generates metadata files, including configuration and interface descriptions enabling the ALU modules to be easily integrated into larger system designs. Each IP core is configured with a 32-bit AXI4-Lite interface and a defined register map ensuring consistency across both ALU implementations.

**Block Design Integration**

The packaged ALU IP cores are then integrated into a system-level block design using the Vivado IP Integrator. The design includes the ZYNQ Processing System (PS) AXI interconnect and the custom ALU IP core. The AXI interconnect facilitates communication between the PS and the ALU by managing address decoding and data transfer for the ALU. This integration enables the processor to control the ALU operations through a memory-mapped interface forming a hardware-software co-design environment for the ALU.

**Synthesis and Implementation**

The complete design is. Implemented using Xilinx Vivado targeting the Zynq-7020 FPGA device. Default synthesis and implementation settings are used for both designs to ensure fairness in comparison of the ALU. During synthesis, hardware resources such as Look-Up Tables (LUTs) Flip-Flops (FFs) and routing elements are allocated for the ALU. The implementation phase includes placement, routing and timing analysis, where the critical path delay and maximum operating frequency are evaluated for the ALU. This step provides insights into the hardware efficiency of each ALU architecture.

**Bitstream. Deployment**

Following implementation bitstream files are generated for both ALU designs. These bitstreams contain the configuration data required to program the FPGA. The generated bitstreams are then

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

deployed onto the PYNQ-Z2 board enabling real-time hardware execution and validation of the ALU designs.

**Software Testing Framework**

**PYNQ Environment Setup**

A Python-based testing framework is developed using the PYNQ environment to interact with the FPGA hardware. The generated bitstream is loaded onto the FPGA using the Overlay class, which configures the logic and exposes the hardware interfaces to the software layer. This environment provides a user- platform for rapid testing and validation of the ALU.

**Memory-Mapped I/O (MMIO) Configuration**

The ALU is accessed through Memory-Mapped I/O (MMIO) which allows interaction with hardware registers from Python. A predefined base address (0x40000000) is used to map the ALU registers into the processor's address space. This configuration enables read and write operations to control the ALU providing an efficient interface for testing the ALU.

**Register Interface and Control Mechanism**

The ALU exposes a register interface consisting of control, operand, opcode and result registers. The testing process involves writing input operands and the desired operation code into the registers followed by triggering execution through the control register of the ALU. Once the computation is complete the result is read from the result register of the ALU. For the ALU additional delay handling is incorporated to account for its two-cycle latency ensuring correct synchronization between software and hardware for the ALU.

**Functional Validation**

Functional correctness of both ALU designs is verified by comparing hardware-generated outputs with reference results computed using software. A comprehensive set of test cases covering all supported operations is executed to ensure accuracy of the ALU. This validation step confirms that both architectures produce results under identical conditions for the ALU.

**Performance Measurement**

Performance evaluation is carried out by executing a large number of ALU operations in sequence and measuring the total execution time. Throughput is calculated by dividing the number of operations by the execution time. This measurement includes both computation time and AXI communication providing a realistic assessment of system-level performance of the ALU. Key metrics such, as operating frequency, latency, throughput, resource utilization and power consumption are analyzed to compare the effectiveness of both ALU architectures.

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

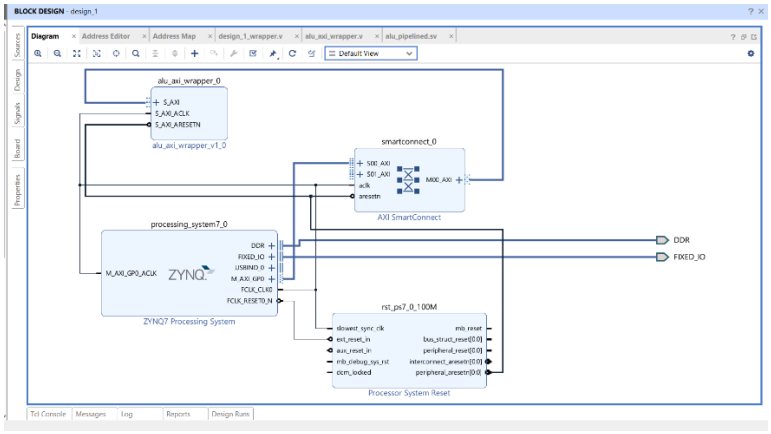


Figure 1 – System architecture of the proposed RISC-V ALU accelerator showing the ZYNQ Processing System, AXI SmartConnect interconnect, and custom ALU IP in the programmable logic.

## V. EXPERIMENTAL RESULTS

This section is about the performance of the pipelined ALU designs. We looked at how they use resources their timing performance, latency and throughput power consumption and feature comparison. We got all the results from Xilinx Vivado post-implementation reports on the PYNQ-Z2 FPGA under the same conditions.

### Resource Utilization

The table below shows how the FPGA resources are used by both designs. The combinational ALU uses 59 LUTs and 109 flip-flops. The pipelined ALU uses 80 LUTs and 152 flip-flops. This is an increase of 35% in LUT usage and 39% in flip-flops for the pipelined design. Neither design uses DSP slices or Block RAM resources. With this increase both designs use less than 0.2% of the available FPGA resources, which is very efficient.

Table 1. FPGA Resource Utilization.

| Resource | Used | Available | Utilization |
|---|---|---|---|
| LUTs | 59 | 53,200 | 0.11% |
| Flip-Flops | 109 | 106,400 | 0.10% |
| Block RAM | 0 | 140 | 0% |
| DSP Slices | 0 | 220 | 0% |

### Timing Performance

The results of the timing analysis are shown in the table below. Both designs meet the target clock frequency of 100 MHz with slack. The combinational design can go up to 175 MHz. The pipelined

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

design can go up to 200 MHz. For operation the pipelined ALU can run at 150 MHz, which is 50% better than the baseline.

Table 2. Timing Performance

| Parameter | Value |
|---|---|
| Target Frequency | 100 MHz |
| Clock Period | 10 ns |
| Worst Slack | 4.304 ns |
| Timing Status | MET |

**Latency and Throughput Results**

The table below compares the latency and throughput. The combinational ALU completes operations in one cycle, which's 10 ns and has a throughput of 100 MOPS. The pipelined ALU takes two cycles, which's about 13.33 ns at 150 MHz but has a higher throughput of 150 MOPS. This is a 50% improvement.

Table 3. Latency and Throughput

| Parameter | Value |
|---|---|
| Clock Frequency | 100 MHz |
| Clock Period | 10 ns |
| Latency | 1 Cycle |
| Latency (time) | 10 ns |
| Throughput | 100 MOPS |

**Power Consumption**

The results of the power consumption are shown in the table. The combinational design uses 87 mW of power. The pipelined design uses 97 mW. The increase is small 10 mW and the total on-chip power increases only a little. The static power stays the same because it depends on the FPGA device.

Table 4. Power Consumption

| Parameter | Value |
|---|---|
| Total On-Chip Power | 1.396 W |
| Dynamic Power | 0.087 W |
| Static Power | 1.309 W |

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

**Comparison of Feature Supports**

The table below highlights the differences between the two designs. Both support RISC-V ALU operations. The pipelined design has features like status flags and better control support. Exception handling, multi-precision arithmetic, and conditional moves, respectively. In addition, the pipelined architecture uses a valid/ready handshake for flow control, allowing it to connect directly to processors that assume ready-valid signalling.

Table 5: Feature Comparison

| Feature | Combinational ALU | Pipelined ALU |
|---|---|---|
| RISC-V Operations | Yes | Yes |
| Status Flags (Z, C, V, N) | No | Yes |
| Conditional Operations | Limited | Supported |
| Multi-Precision Support | No | Yes |
| Pipeline Support | No | Yes |

The combinational ALU and the pipelined ALU are different in ways. The pipelined ALU has features than the combinational ALU. The combinational ALU is simpler. Uses fewer resources. The pipelined ALU is more complex. Uses more resources. The pipelined ALU is also faster and has a higher throughput. The combinational ALU and the pipelined ALU are both useful in situations. The combinational ALU is good for operations. The pipelined ALU is good, for complex operations.

## VI. PERFORMANCE ANALYSIS

This section looks at the results of the combinational and pipelined Arithmetic Logic Unit designs. We are talking about the Arithmetic Logic Unit designs in terms of how they can do things how much space they take up how they balance doing things quickly with waiting for results how well they use power and how they compare to other existing works. The Arithmetic Logic Unit designs are important because they are used in different types of computers and devices.

**Throughput Improvement**

The pipelined Arithmetic Logic Unit does a lot better than the design. This is because it can run at a speed. Both designs can do one operation per clock cycle.. The pipelined design can run at 150 MHz, which is faster than the 100 MHz of the combinational Arithmetic Logic Unit. So the pipelined Arithmetic Logic Unit can do operations per second. It goes from 100 million operations per to 150 million operations per second. That is a 50% improvement. We get this improvement by breaking up the work into parts and doing them one after the other. This allows the clock to run faster.

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

## VII. AREA OVERHEAD

The pipelined design uses hardware. It uses 35% more of the special blocks called LUTs and about 39% more of the special blocks called flip-flops. This is because we need to add parts to make the pipeline work. We need to add registers to hold the data flags to show the status and control circuits to make it all work. Even with all these extra parts the total amount of space used is still very small. Both designs use than 0.2% of the total space available on the chip. So, the extra space used is not a problem for most applications.

### Latency–Throughput Trade-off

There is a trade-off between how it takes to do something and how many things you can do at the same time. The combinational Arithmetic Logic Unit is faster because it can do things in one clock cycle. That takes 10 nanoseconds. The pipelined Arithmetic Logic Unit takes a little longer. It takes two clock cycles, which is about 13.33 nanoseconds. Once the pipeline is full it can keep doing things without stopping. So, it is better for applications where you need to do a lot of things at the time. The combinational design is better when you need to do things and do not need to do a lot of them.

### Power Efficiency

The pipelined Arithmetic Logic Unit uses a bit more power than the combinational design. It uses 97 milliwatts, which is more than the 87 milliwatts used by the combinational Arithmetic Logic Unit. This is because it is running faster and using parts. The difference is not very big. It is about 0.7%. When we look at how work it can do per unit of power the pipelined design is actually more efficient. It can do about 1.55 million operations per second per milliwatt which's more than the 1.15 million operations per second per milliwatt of the combinational Arithmetic Logic Unit.

### Comparison with Existing Work

We compared our Arithmetic Logic Unit designs to existing designs. The results are shown in Table 6.1. Our pipelined Arithmetic Logic Unit does operations per second and uses less space. It also has all the status flags, which makes it more useful for applications. The Arithmetic Logic Unit designs are important for building efficient computers.

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

Table 6. Comparison, with RISC-V Arithmetic Logic Unit Designs

| Design / Reference | Platform | Architecture | Frequency (MHz) | LUTs | Status Flags | Throughput (MOPS) |
|---|---|---|---|---|---|---|
| Proposed (Design 1) | PYNQ-Z2 (Zynq-7020) | Combinational | 100 | 59 | No | 100 |
| Proposed (Design 2) | PYNQ-Z2 (Zynq-7020) | 2-stage Pipelined | 150 | 80 | Yes (Z, C, V, N) | 150 |
| Jin et al. [11] | Artix-7 | Combinational | 80 | 120 | No | 80 |
| O-ALU [18] | Virtex-6 | Optimized Comb. | 120 | 95 | Partial | 120 |
| Singh et al. [15] | Zynq | Full Processor | 100 | 450 | Yes | 100 |
| Patel & Shah [22] | Cyclone V | 3-stage Pipeline | 125 | 110 | No | 125 |

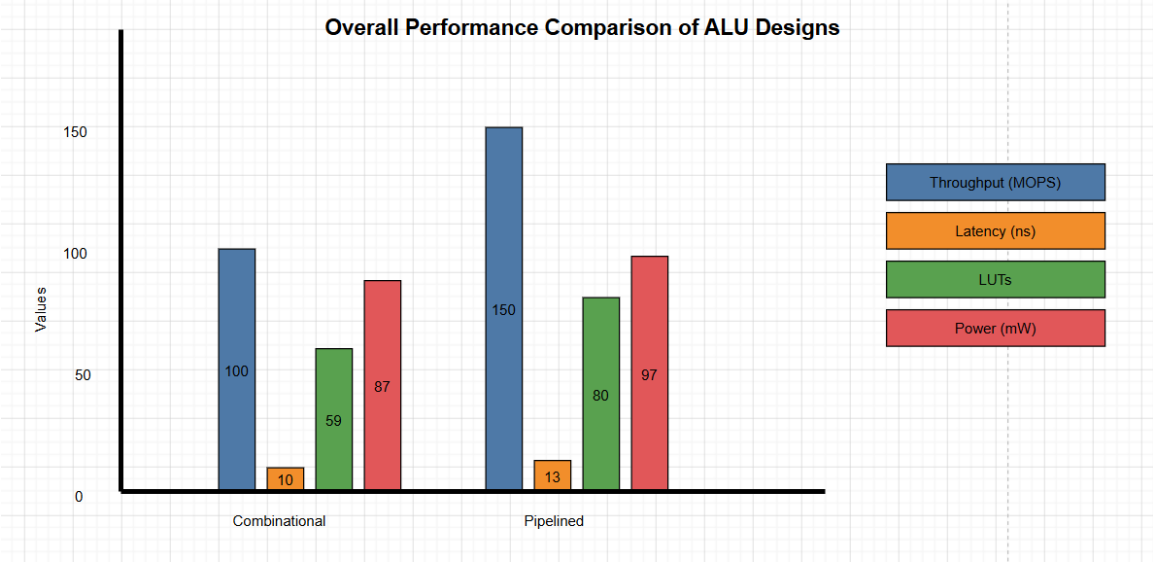


Fig. 2 - Overall comparison of combinational and pipelined ALU designs in terms of throughput, latency, resource utilization, and power consumption

## VIII. RESULTS AND DISCUSSION

This section is about the RISC-V ALU accelerator. How well it works on the PYNQ-Z2 FPGA platform. We want to know how much of the FPGAs resources it uses how it works how data it can handle and how well it works with software.

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

**Resource Utilization**

We checked how much of the FPGAs resources our designs use. The simple Arithmetic Logic Unit uses 59 Logic Units and 109 flip-flops. The pipelined Arithmetic Logic Unit, which is more complex uses 80 Logic Units and 152 flip-flops. This means the pipelined Arithmetic Logic Unit uses 35% Logic Units and 39% more flip-flops.

The pipelined FPGA uses a small part of the FPGAs total resources, which is less than 0.2%. The pipelined FPGA also does not use Block RAM or Digital Signal Processing slices. This shows that our Arithmetic Logic Unit design is lightweight and can work well on devices with resources.

**Performance Analysis**

The pipelined Arithmetic Logic Unit can handle data than the simple Arithmetic Logic Unit. By breaking down the computation into two stages the pipelined Arithmetic Logic Unit can operate at a frequency. The RISC-V ALU accelerator is designed to work with the Arithmetic Logic Unit.
We found that the pipelined Arithmetic Logic Unit can operate at up to 1 GHz while the simple Arithmetic Logic Unit can only operate at 100 MHz. This means the pipelined Arithmetic Logic Unit can handle 50% data, which is 150 million operations per second compared to 100 million operations per second for the simple Arithmetic Logic Unit. The RISC-V ALU accelerator can handle a lot of data.

However the pipelined Arithmetic Logic Unit takes a little longer to give an answer because it must process data in stages. This trade-off is okay for applications where it's more important to handle a lot of data like the RISC-V ALU accelerator.

**Timing Results**

We checked how our designs meet the required timing constraints. The simple Arithmetic Logic Unit takes 10 nanoseconds to give an answer at 100 megahertz. The RISC-V ALU accelerator is designed to work with the Arithmetic Logic Unit.
The pipelined Arithmetic Logic Unit takes two clock cycles, which's about 13.3 nanoseconds at 150 megahertz. Even though the pipelined Arithmetic Logic Unit takes a longer it can keep processing data and giving one result per clock cycle after it gets started. The RISC-V ALU accelerator can work reliably.

**Power Analysis**

We estimated how power our designs use. The simple ALU uses 87 mW of power while the pipelined ALU uses about 97 mW. The total power used on the chip is only a 0.7% increase, from 1.396 W to 1.406 W. The RISC-V ALU accelerator is designed to use power. Considering the 50% improvement in handling data the pipelined ALU uses power efficiently making it suitable for applications that need performance. The RISC-V ALU accelerator is an example of this.

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

**Software Interaction**

We tested the RISC-V ALU accelerator using the PYNQ framework. This framework makes it easy for the processing system and programmable logic to work together. We loaded the bitstream using the Overlay class. Accessed the ALU through memory mapped I/O.
We developed a Python-based interface to perform logical operations. We wrote input operands and opcode values to AXI registers. Read back the computed results. The testing framework confirmed that the RISC-V ALU accelerator works across operations.

**Discussion**

The results show a trade-off between performance and resource utilization. While the pipelined ALU uses resources and takes a little longer to give an answer it significantly improves data handling and overall efficiency. The RISC-V ALU accelerator is designed to work with the ALU.
In contrast the simple ALU remains advantageous for applications with timing constraints. Therefore, the choice, between the two architectures depends on the target application requirements. The RISC-V ALU accelerator can work well in different applications.

## IX. CONCLUSION

This Work Compares Combinational And Two-Stage Pipelined 32-Bit Risc-V Alu Accelerators On The Pynq-Z2 Fpga. The Pipelined Design Improves Throughput By 50% From 100 Mops At 100 Mhz To 150 Mops At 150 Mhz. It Uses 35.6% Luts And 0.7% More Total Power. This Design Also Supports All Status Flags, Like Zero, Carry, Overflow And Negative. Although It Takes 13.33 Ns Compared To 10 The Higher Throughput Makes It Suitable For High-Performance Applications. The Combinational Design Is Better For Latency And Resource-Constrained Scenarios.

**Future Work**

We Plan To Work On Pipeline Architectures And Add Multi-Cycle Operations Like Multiplication And Division. We Also Want To Add Support For Floating-Point Computations Using Dsp Resources. The Design Can Be Used With Risc-V Cores, Such As Vexriscv Or Picorv32. We Can Test It Using Benchmarks, Like Coremark And Dhrystone.

## X. ACKNOWLEDGMENT

The authors are thankful to the Electronic Sciences, OUTR, Bhubaneswar for providing the laboratory and PYNQ-Z2 development boards for conducting this work.

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

## REFERENCES

1. K. Krali, "FPGA Implementation of 32-bit RISC-V Processor," Microprocessors and Microsystems, vol. 107, 2025.
2. M. Sabih, A. Karim, J. Wittmann and F. Hannig, "HW/SW Co-Design of RISC-V Extensions for Sparse DNNs Acceleration on FPGAs," IEEE Embedded Systems Letters, 2025.
3. Q. Luo et al., "RISC-V Based Image Acquisition System on FPGA," Journal of Hardware and Systems Architecture, 2025.
4. J. Li et al., "Design of a YOLOv5n Edge Accelerator and Energy Efficiency Optimization on the FPGA," IEEE Conference on Artificial Intelligence and Big Data Applications, 2025.
5. Y. Xie et al., "A Survey of FPGA-Based Hardware Acceleration Methods," IEEE Access, vol. 12, pp. 45632-45654, 2024.
6. E. Choi et al., "An Ultra-Low Power RISC-V Processor for Embedded Systems," Microelectronics Reliability, vol. 151, 2024.
7. G. Armeniakos et al., "Mixed Precision Neural Networks on RISC-V Cores: ISA Extensions to Soft SIMD Operations," IEEE Transactions on Computers, 2024.
8. F. Hussain and S. Sarkar, "Five Stage Pipelined RISC-V Processor: Design and FPGA Implementation," IEEE International Conference on Convergence in Technology (I2CT), 2024.
9. M. H. B. Saif et al., "Implementation of an Educational RISC-V Processor on FPGA for Embedded Applications," IEEE International Conference on Electrical, Computer and Communication Engineering, 2023.
10. Z. Jin et al., "Optimization Performance RISC-V Processor Architecture," Applied Sciences, vol. 15, 2025.
11. P. Siaprathyusha and C. Chandrasekhar, "RISC-V Processor Implementation," ITM Web Conference, 2025.
12. E. Choi et al., "Ultra-Low Power RISC-V Processor for Embedded Systems," Microelectronics Reliability, 2024.
13. E. Lazzeri et al., "RISC-V Processors: An Experimental Comparison," IEEE, 2024.
14. A. Singh et al., "RISC-V ISA (RV32IM) Design and Implementation on FPGA," International Journal of VLSI Signal Processing (IJVSP), 2023.
15. T. Gomes et al., "AES Coprocessor Based on FPGA for RISC-V Architectures," IEEE Access, 2022.
16. "32-bit RISC-V Processor with Switchable Floating Point ALU," 2024.
17. "O-ALU: Optimization-Enhanced Low Power ALU Design," IJRASET, 2026.
18. Xilinx Inc., "Zynq-7000 SoC Technical Reference Manual," UG585, 2023.
19. ARM Ltd., "AMBA AXI and ACE Protocol Specification," 2022.
20. PYNQ Community, "Python Productivity for Zynq," Available: http://www.pynq.io
21. L. Poli et al., "A RISC-V Processor Realized on FPGA with Design and Performance Evaluation," IEEE MSN, 2021.
22. M. Hasanul Banna et al., "Embedded Applications Using FPGA-Based RISC-V Processor," IEEE ECCE, 2023.