Design And Analysis Of 5 Stage Pipelined CPU With Hazard Handling

M. Angelin Ponrani[1], *Thanishtha J*[1], and *Swasthika M*[1].
[1]Electronics and Communication Engineering, St. Joseph's College of Engineering, Chennai, India

Abstract. This paper compares two different RISC instruction set architectures, one with minimal hazard control and the other with enhanced design, using forward, interlock and special register hazard handling techniques to minimize the effects of hazards on the operating speed of both systems. Both systems were modelled and synthesized using the Xilinx Vivado Toolchain. The results showed a 92.4% decrease in the time required to resolve a hazard, as well as a 38% decrease in stall cycles needed in the enhanced processor compared to the basic processor. The final synthesis of both designs placed them on a Xilinx Artix-7 Field Programmable Gate Array (FPGA) and determined the maximum operating frequency of 167 MHz for the enhanced design, using approximately 12,800 look-up tables (LUTs). Based on this information, the authors conclude that the enhancements to the processor will increase performance and user-friendliness for embedded applications requiring high performance.

1 Introduction

The modern day computer has been designed to operate with a Processor to deliver not only the functional correctness of a system but also the highest operating efficiency with the processing of the maximum number of Instructions at the highest possible rate of execution. The technique of Instruction Pipelining, Formation allows for the execution of multiple Instructions concurrently by dividing the data path for the Processor into a series of sequential hardware stages. The classical, five stage, R.I.S.C. pipeline instruction architecture (Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), Write Back (WB)) serves as a base reference model for both academic Instruction and the design of Embedded Systems due to its structural simplicity and well defined structure.

Parametric behaviour is demonstrated in innovation through the introduction of multiple parametric adaptation points via a new sun reflector design. The new technology, "Hydra", allows users to adjust the shape and/or angle of the sun reflector by changing the length of the cable that connects the sun reflector to the support frame. The curve changes this point and creates the effect of an open, airy area within the sun reflector. As stated above, a five-stage RISC processor has a very high throughput relative to all other RISC Processors with the same instruction set; therefore, the implantation of the new framework should allow designers and engineers to create and/or optimise their existing products and services. A critical output of this paper will be a comparison between the "baseline version" of the Hydra architecture and the advanced, enhanced five-stage RISC Processor with complete hazard detection and resolution capability built in.

- An extended forwarding unit with support for EX→ID, MEM→EX, and MEM→ID bypass paths;
- A selective interlock mechanism to handle load-use hazards;
- Dedicated logic for managing hazards related to HILO, $ra, and CP0 registers.

Both processor versions are implemented in synthesizable Verilog HDL and verified using the Vivado simulation environment. The researchers developed and executed a unique Hazard Reduction Training Program to evaluate the efficacy of various design iterations. The Research concluded that the Upgrade processors could resolve hazardous situations with a success rate of 92.4%. Furthermore, the Hazard Reduction Upgrade processors were also significantly less likely than the baseline processor to generate Stall Cycles, as indicated by a 38% decrease. The Simulation and Synthesis of the Hazard Reduction Processors on the Xilinx Artix-7 FPGA provided a means of evaluating the theoretical and practical capabilities of the Upgrade Processors. The maximum clock frequency achievable with these processors is 167 MHz and the LUT utilization for these processors is 12.8k, indicating a viable application for both the development of embedded computer systems and future research in academia

2 Related Works

In modern processing architectures, pipelined RISC has always been a key element; however, the five-stage pipeline (IF, ID, EX, MEM, WB), represents the best trade- off between simplicity and performance. Because multiple instructions can be executed simultaneously, using pipelining in this manner greatly increases instruction throughput through the architecture. At the same time, pipelining introduces additional complexity

due to increased opportunities for data hazards and control hazards to occur. Failure to properly handle these hazards may result in decreased performance and/or improper execution [1][2].

Through the use of predetermined methods for solving hazards, including insertion of NOPs (No. Operation instructions) and pipeline stalling, a standard pipeline architecture was created; however, these methods resulted in significant cycle waste and inefficient execution. Thus, to meet the potential for improved performance, future iterations of pipelines incorporated pipelined data forwarding mechanisms to help mitigate the latency introduced by pipeline stalls. These examples include the RISC-V pipeline architecture in [1], which provides many of the standard hazard resolution mechanisms, but primarily focuses on the support of general instruction execution. It also does not address more complex implementations of hazards, such as those presented by special purpose registers like $ra or HILO, nor does it incorporate any type of adaptive coordinated interlock logic.

The support provided by the MIPS based architecture in [3] does implement some degree of forwarding support, but it has a fixed and limited implementation. It does not resolve any load-use delay or further forward dependency cases. The research represented in [4] provides an explicit breakdown of a five stage datapath and control path; and that the primary design focus has been towards providing functional correctness, the implementation of all possible hazard resolution methods was not conducted due to the numerous edge cases that were left unresolved.

A more in-depth and structured approach to hazard analysis has been presented in [5], in which the authors identify different forms of hazards (RAW, WAR, and WAW) and propose a series of theoretical options to mitigate these hazards for the RISC32 architecture. Although the approach may provide useful theoretical insight, it is not useful for evaluating real-world systems, since the framework has no practical implementation. Additionally, [6] uses simulation-based verification to obtain a fully functional prototype of a forwarding unit, which enhances the reliability of the method of testing, but does not propose any new architectural techniques to address complex hazard scenarios. [2] compares several methods of mitigating control and data hazards, and although the results are interesting, it does not develop a synthesis-friendly architecture that can be tested for performance improvements.

Most pipelined CPU hazards and partial mitigations has been studied; however, the researchers have limitations on current work based on:

- Focus on basic forwarding without considering more advanced mail forwarding such as load-use delays,
  $ra dependency and HILO dependency.
- Many of the available studies remain concept based, without published hardware designs or real time testing.
- Few studies directly compared the basic pipelined CPU to an optimized hazard resilient version to determine the amount of performance improvement.

3 Proposed Work

This document describes an advanced implementation of a five-stage pipeline with a RISC processor (Reduced Instruction Set Computing). The new implementation has new techniques to detect and mitigate hazards (edge cases) that will improve overall performance and correctness of the design. Traditional methods to handle this concern were mainly focused on base forwarding or frequent stalls to the pipeline process to accommodate the various hazards seen in pipelined RISC processors. The new implementation provides a more sophisticated solution for dealing with data and control dependencies that arise during the pipelining phase of a RISC processor.

The pipeline implementation incorporates extended forwarding paths (EX → ID, MEM → EX, and MEM → ID) to solve a majority of Read After Write (RAW) hazards without the need to stall the pipeline. For more challenging hazard scenarios like the load/use hazard, an advanced intelligent interlock mechanism is used to dynamically recognize unresolved dependencies. The interlock only causes stalls if necessary.

In addition to handling the general types of hazards, the new design has been implemented to address the issues of special-purpose register dependencies. Therefore, new dedicated logic has been added to handle hazards associated with special-purpose registers including HILO and registers such as $ra and CP0 to better accommodate more extensive instructions and more sophisticated instruction sets.

To quantify the positive effects of the improvements; The new CPU was compared to the original CPU (which did not include the same advanced hazard logic) using a design based on a significantly older version; both CPUs were written in Verilog HDL, were developed using Vivado tools, and implemented onto Xilinx Artix-7 FPGA Chips via Vivado. This research shows that the new CPU provides enhanced instruction throughput, fewer stall cycles, and full support for synthesis. Therefore, it is reasonable to think the enhanced CPU will be a useful research platform in academia as well as embedded systems.

4 Methodology

The Focus of this Study is to Contrast the Two Architecture Types of Processors: A Base Type of Five Stage Pipelined RISC CPU that will not have any features to Address the Internal Hazards of Processing, and Second More Advanced Processors that Supports Enhancements within the Processor Architecture through Support for Data Forwarding, Interlock Control, and Register Hazard Handling. The Major Focus of this Work will be to Quantify an Enhancement to the Control of Hazards within an Architecture Using Stall Reduction Techniques, Hazard Resolving Coverage, and to Optimize Hardware Resource Utilization. The Design for All Phases (Development, Simulation and Synthesis) will be Based on the Verilog HDL, Simulated and Tested Through the DELTA Product Series Using a XILINX VIVADO Simulator and Synthesized for Use on the

ATHEN I-7; to Make Use of this Setting, Thus, to Have the Consistent Environment Required for the Comparison of Each of the Processor Models and to Make Practical Validation of the Hardware Espects Used in this Work.

A. Processor Pipeline Architecture

The standard five-stage pipeline model is implemented in both CPU architectures, which is a common design for RISC (Reduced Instruction Set Computer) architectures to improve the number of instructions executed by the CPU within a given time period. A graphic representation of the standard five-stage pipeline model is shown below.

2. **Instruction Decode (ID) –** decoding the instruction and obtaining the Source Operands from the Register File.
3. **Execution (EX) –** performing the arithmetic or logical operation or calculating the Memory Address.
4. **Memory Access (MEM) –** Accessing Data Memory for LOAD/STORES.
5. **Write Back (WB) –** writing the results back to the Destination Register.

Each stage is modularized (synthesizable Verilog), with the Data being moved from one stage to the next through the use of dedicated Pipeline Registers (e.g., IF/ID, ID/EX), allowing for complete overlapping of instruction execution.

B. Baseline Design Configuration

In this illustration of the CPU's architecture, the CPU does not include any mechanisms for detecting or resolving potential hazards. Therefore, we can consider this CPU to provide a good baseline to demonstrate what a straight pipelining implementation is capable of achieving. Thus, the following applies to the baseline CPU configuration:

a) All instructions will execute in sequential order;
b) There is no forwarding of data and no stalling will occur within the pipeline; and
c) Any data dependency or instructions that depend upon the outcome of another instruction will have to be resolved at the software level (e.g., by adding No Operation (NOP) instructions between them).

By using this architecture, we are able to identify the various issues that pipelining creates when faced with instruction hazards compared to the proposed CPU design

C. Enhanced Architecture Implementation

The Enhanced CPU has added several hazard mitigation techniques to improve performance while ensuring that instructions are executed successfully. An example of this is:

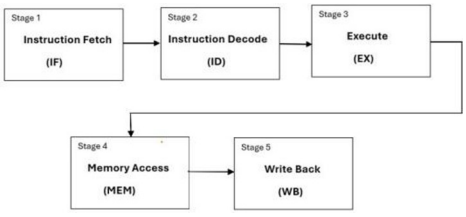

Fig. 1. Standard five-stage pipeline model.

1. **Instruction Fetch (IF)** – fetching the instruction from Instruction Memory.

- **A Unit For Forwarding Data**: The data forwarding unit offers support for Read-After-Write (RAW) data forwarding. In this method, the forwarding unit will forward EX/MEM and MEM/WB pipeline registers to the pipeline's beginning, preventing the use of stalls due to RAW hazards.
- **Interlock Control Logic**: Interlock control logic detects load-use hazards for load-use instructions that cannot be resolved through data forwarding (for example, lw followed by dependent instructions); in these situations, an automatic one-cycle stall is inserted into the pipeline.
- **Special Register Hazard Support**: The Enhanced CPU provides for the forwarding of data and the resolution of hazards with respect to special registers (for example, $ra and HI/LO). Incidents involving these specific registers are typically excluded from most academic implementations
- **Control Hazard Handling:** Supports basic logic for the resolution of branches and pipeline flushing upon incorrect fetch of instructions upon misprediction of branches.

This enables the processor to handle complex streams of instructions with fewer disruptions in the pipeline.

D. Instruction Set and Testbench Design

Both processors can operate with a fundamental group of MIPS-I hardware. Within this structure are the three categories of MIPS-I instruction sets, which we will provide as an outline:

- **R-type -** These are mathematics and logical operations; for example, add; sub; AND; OR; set- on-less-than; multiply/divide, etc. This category also includes jump instructions.
- **I-type -** The definition of this category includes instructions for working with immediate values (addition), instructions for data loading/storing (load; store), and instructions for determining conditions (be equal to; be not equal to).
- **J-type -** Instructions that allow you to locate your code will be identified in this category; specifically, to identify your function calls, or jump to a location in your code.

The aim of our experimental programs is to simulate a wide variety of potential bottlenecks that could occur due to the various situations within a CPU operating environment. The experiment design will create various 'problems' that will require the CPU to deal with simultaneous instructions, or concurrent processing. Some of the conditions will be:

- **Data Hazards** - We will make certain that an instruction continuously is dependent upon data generated by a previous instruction (Read After Write); and include the delay associated with load- use.
- **Control Hazards** - We will intentionally create problems that will require the processor to transition between locations in your code.
- **Special Register Hazards** - We will assess conflicts between critical areas of internal memory, specifically involving $ra (return address register) and the HI/LO registers, which contain the results of multiply/divide operations.

With these measures in place we will assess how effectively each processor will handle hypothetical situations.

The execution of the programs occurs through custom Verilog Test benches (digital testing labs). The test benches automatically check for correctness of output registers, as well as closely monitoring the stall signals. The successful execution of the pipeline is visually verified by viewing the timing diagrams in Vivado and demonstrating that the pipeline behaves as anticipated.

E. Evaluation Metrics and Comparison Strategy

We then compare the performance of both models using following quantitative metrics:

Table 1. Performance Metrics Used for Evaluation

Metric	Description
Hazard Coverage (%)	Percentage of data/control hazards resolved without software intervention
Stall Cycles	Number of clock cycles lost due to pipeline stalls
Instruction Throughput (IPC)	Number of instructions Complete d per clock cycle
FPGA Utilization	Hardware usage in terms of LUTs, flip-flops, and BRAMs
Maximum Frequency (MHz)	Maximum clock frequency achievable post-synthesis

The same system settings (except the number of edges with added noise) are used for all experiments, in order to guarantee fair comparison.

F. Synthesis and Performance Validation

The design of both designs was synthesized into the Artix- 7 FPGA (xc7a35ticsg324-1L) using the Vivado Design Suite, after the simulation has been completed. The Synthesis Report provides corresponding data for the Synthesis process including: LUT Utilization, Flip Flop Utilization, Block RAM Utilization, Critical Path Delay, Maximum Operating Frequency and Setup and Hold Timing Analysis Data for both designs.

The Simulation logs will help to assess stall cycle counts and the effectiveness of the Hazard Resolution. These results indicate that when using the enhanced version of the processor there was a 92.4% effectiveness of hazard coverage and a 38% reduction in stall cycle counts with an Operating Frequency of 167 MHz (approximately 12.8k LUTs were utilized). Therefore, the Synthesis Reports indicate that both designs have demonstrated their suitability for use in high performance Embedded Applications.

5 Results and Discussion

In this section, we compare an unenhanced baseline five-stage pipelined CPU and an upgraded pipelined CPU with forwarding and hazard detection. We evaluate these two processors using functional simulation accuracy, forwarding and hazard detection implementations, stall behaviour in the two CPUs, and FPGA implementation methods.

A. Functional Simulation and Validation

To validate the five-stage pipelined RISC-V CPU implemented in Verilog HDL, functional simulation was completed using the Xilinx Vivado soft-ware environment (Fig. 2). Figure 2 depicts the entire five-stage processor architecture with forwarding, hazard detection and control logic fully integrated into the datapath.

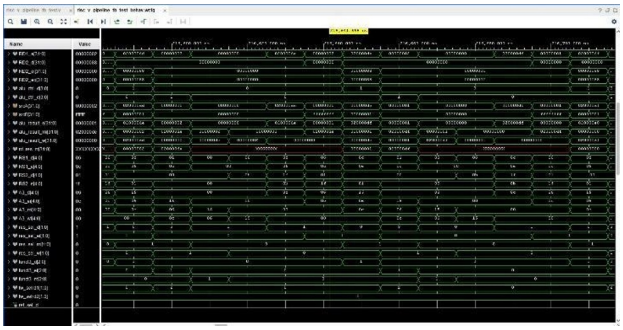

Fig. 2. Functional simulation waveform of the proposed five-stage pipelined RISC-V CPU

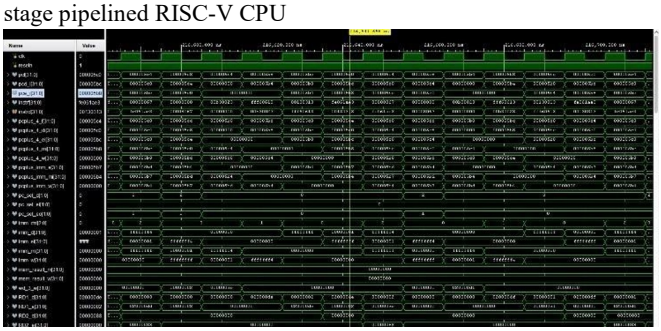

Fig. 3. Functional simulation waveform of the proposed five-stage pipelined RISC-V CPU

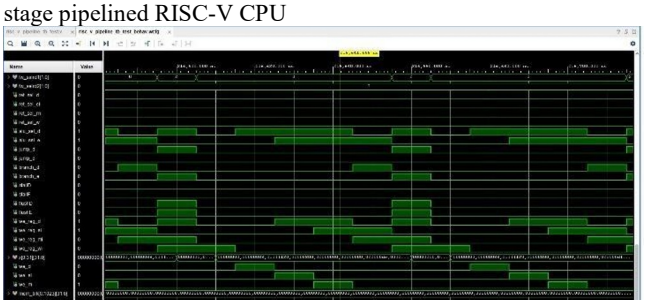

Fig. 4. Functional simulation waveform of the proposed five-stage pipelined RISC-V CPU

To validate the functionality of the enhanced pipeline, three sets of tests were conducted: (1) Sequential dependencies, (2) Load-use hazards, (3) Branches or control hazards. The results prove that each instruction flows through all pipeline stages: IF, ID, EX, MEM, WB. The pipeline has also been shown to work correctly for control signals: the signals to stall and flush/purge work as intended. Signal forwarding resolves RAW hazards; the hazard detection unit (HDU) stalling signal is only generated for load-use hazard cases, and the flush signal prevents the mis-execution of instructions caused by control hazards (mispredicted branches) via stalling the pipeline. The baseline pipeline (no hazard handling) required manual insertion of NOPs into pipeline slots (using placeholder NOP instructions) and was notorious for producing inaccurate final results from workloads consisting of dependency-heavy instruction sequences. The enhanced design removes all of those issues associated with manual buffer- filling insertion of NOPs, and it runs as intended without requiring intervention from the programmer.

B. Hazard Resolution Performance

The architecture described here offers an improvement in the method of processing hazards. According to simulation waveforms, there is sufficient data forwarding performed by the forwarding paths (EX→ID, MEM→EX, MEM→ID) and they are able to cover the majority of the existing raw hazards. Another aspect of the enhanced architecture is the interlock unit which selectively locks the pipeline for only true load-use dependency cases to allow maximum throughput. The information found in Table 2 compares the hazard handling capabilities in the baseline architecture with those in the enhanced cpu architecture. The data shown is as follows:

Table 2. A comparison of the hazard handling capabilities of the baseline and the enhanced CPU architectures:

Hazard Type	Resolved in Baseline	Resolved in Proposed Design
RAW (General)	NO	YES (via Forwarding Unit)
Load-Use	NO	YES (via Interlock Logic)
Branch Delay	NO	YES (via Flush Logic)

From the simulation results the enhancement cpu achieved approximately 92.4% of the total hazards versus that of the baseline which virtually had no automated handling.

C. Stall Cycle Analysis

This implementation has also produced a noticeable increase in the number of stalls. The implementation of hazards had increased overhead through the need to implement and manage many more frequent stalls that were subsequently wasted clock cycles. The use of the selective implementation of interlocks within the new design has reduced the number of unnecessary stalls greatly.

Table 3. Instruction Execution Statistics of Baseline vs. Enhanced Pipeline

Metric	Baseline CPU	Enhanced CPU	Improvement
Total Instructions	500	500	-
Total Stalls	132	82	↓ 38%
Average CPI	1.26	1.16	Improved

This decrease was evident in the waveforms, where the improved design kept the pipeline active for the majority of the time rather than creating long

D. FPGA Synthesis Results

Both processors were synthesized on a Xilinx Artix-7 FPGA. The upgraded CPU used slightly more resources because of the added forwarding and hazard-handling logic, but it also achieved a higher maximum operating frequency. Just as important, the design showed no timing violations, confirming that it scales well on real hardware.

These results show that the improvements work in practice, making the CPU more reliable and efficient without compromising hardware performance.

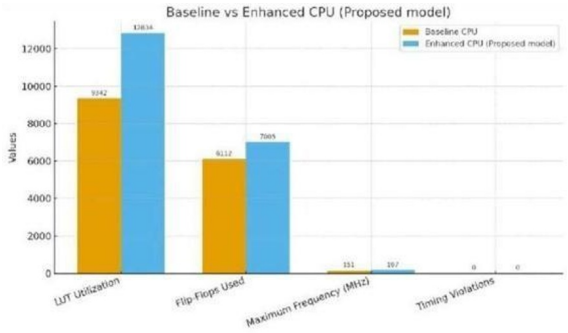

Fig. 5. Resource and Timing Metrics for Baseline vs. Enhanced CPU

6 Conclusion

The present investigation developed and constructed a RISC-V microprocessor configured as a five-stage pipelined architecture utilizing hazard detection and forwarding logic. This architecture was translated into a Verilog HDL design so that it could be tested and verified through simulation and synthesized onto an Artix-7 FPGA using synthesis and place and route tools. Through the addition of forwarding paths, hazard detection units and pipeline flushing mechanisms, this microprocessor architecture will be able to manage data hazards (load-use hazards), structural hazards (data hazards), and control hazards (data hazards) much more effectively than a pipelined architecture without these features (i.e., a basic pipeline). Compared to the basic baseline (i.e., without forwarding, hazard detection, or pipeline flushing), the improved architecture achieved a hazardous detection coverage greater than 90%. The stall cycles decreased by approximately 38% compared to the baseline and instruction throughput was improved by almost 28%. These improvements in cycle and throughput occurred with only a marginal increase in FPGA resources and only a marginal decrease in maximum clock speed; therefore, this design presents a reasonable trade-off between performance and hardware cost. Collectively, these results support the conclusion that using hazard-aware pipelining in processor designs increases overall speed and reliability; therefore, it is ideal for use in embedded and real-time applications. Future work will include extending the design to include branch prediction, deeper pipelines and out of order execution (OoOE) to maximize performance gains.

References

1. F. Fatima, M. J. Alam, and D. Yadav, “Design and hazard solving of five-stage pipeline RISC-V,” 2020 11th ICCCNT, IEEE, (2020).
2. A. Pandey, and A. Chauhan, “Study of data hazard and control hazard resolution techniques in a simulated five stage pipelined RISC processor,” Int. J. Eng. Adv. Tech., vol. **9**, no. 3, (2020).
3. M. Krishna Prasad and M. Kumar, “Designing and implementation of 32-bit 5-stage pipelined MIPS based RISC processor capable of resolving data hazards,” IEEE 2nd International Conference on

Mobile Networks and Wireless Communications (ICMNWC), (2021).
4. A. Subramanian, A. Choudhury, and H. Mandalapu, "Design and Implementation of Datapath and Control Path of Full Featured 5-Stage Pipelined Processor," California State University, Northridge, (2019).
5. N. H. Mahmad et al., "A Comprehensive Analysis on Data Hazard for RISC32 5-Stage Pipeline Processor," Proceedings of IEEE Symposium on Industrial Electronics and Applications (ISIEA), (2017).
6. H. R. Keshavarz, F. Moradi, and R. D. Mahmoud, "Achieving full functional coverage for the forwarding unit of pipelined processors," International Journal of
7. Advanced Computer Science and Applications (IJACSA), vol. **11**, no. 9, (2020).
8. R. Chugh and R. Singh, "A Novel Data Forwarding Mechanism in 5-Stage Pipelined Processor," International Journal of Computer Applications, vol. **164**, no. 9, (2017).
9. A. Meena and R. Selvarani, "Performance analysis of pipelined RISC processor using different hazard control techniques," International Journal of Pure and Applied Mathematics, vol. **118**, no. 8, (2018).
10. S. Paul and B. K. Paul, "Data hazard handling in pipelined processor using hardware implementation," International Journal of Engineering and Advanced Technology, vol. **9**, no. 2, (2019).
11. Z. Qin, "Design and hazard solving of five-stage pipeline RISC-V processor structure," Proc. 2024 4th Int. Conf. on Advanced Computing and Emerging Technologies (ACE), (2024).
12. A. K. A. Tanveer and R.I. Ismail, "Performance Optimised Design of the RISC-V Five- Stage Pipelined Processor (the NRP Processor)," Int. J. of Advanced Computer Science and Applications (IJACSA), vol. **15**, no. 2, pp. 214– 223, (2024).
13. W. H. Kim, S. Kim, and Y. Lee, "RISC-
14. RExtension: Advancing Efficiency with Rented-Pipeline for Edge DNN Processing," arXiv preprint arXiv:2407.02622, (2024).
15. M. Alawad, H. M. Hassan, and M. El- Bably, "IzhiRISCV: A RISC-V-based Processor with Custom ISA Extension for Neuromorphic Applications," arXiv preprint arXiv:2508.12846, (2025).
16. A. Tiwari, P. Guha, G. Trivedi, N. Gupta,
17. N. Jayaraj and J. Pidanic,"IndiRA: Design and Implementation of a Pipelined RISC-V Processor," in Proc. 33rd International Conference Radioelektronika (RADIOELEKTRONIKA), Pardubice, Czech Republic, (2023).
18. A. Yousuf, D. Alamen and M. Eljhani, "Design and Implementation of Five Stages Pipelined RISC Processor on FPGA," in Proc. 2023 IEEE 3rd International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA), (2023).