International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 04 | April - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Implementation of a Reconfigurable RV32I RISC-V with Five-Stage Pipelining Technique

Dr. B. Vasudeva[1], Attili Geetha Laxmi[2], Guntupalli Tejaswini[3], Kuruti Chandu[4], Chaviti Anil Kumar[5]
[1]*Department of ECE & GMR Institute of Technology*
[2]*Department of ECE & GMR Institute of Technology*
[3]*Department of ECE & GMR Institute of Technology*
[4]*Department of ECE & GMR Institute of Technology*
[5]*Department of ECE & GMR Institute of Technology*

Abstract - RISC-V has found increased application in embedded systems and research work due to the growing demand for open, flexible, and low-cost CPU architectures. In an effort to improve hardware utilization and improve processor performance, this paper demonstrates the development and realization of the RV32I RISC-V core on FPGA. The processor employs a five-stage pipeline that consists of the required stages for a pipelined processor: Instruction Fetch, Instruction Decode, Execute, Memory Access, and Write Back stages. Supports the RV32I instruction set architecture. It also incorporates a custom- made hazard control unit that uses pipeline stalling and data forwarding to handle data and control hazards. It is implemented on a Zynq 7000 FPGA and modeled in Verilog HDL. Simulation was used to test the processor. To examine the functionality of the processor, execution of instructions, arithmetic and loop-oriented programs were used. This processor is suitable for real-world FPGA- based applications and processor development, since the pipelined architecture improves instruction throughput compared to non-pipelined execution while maintaining low estimated on-chip power and moderate resource usage.

Key Words: RISC-V, FPGA-Based Processor, Five-Stage Pipeline, RV32I Instruction Set, Hazard Control Unit, Pipelined Architecture**.**

1. INTRODUCTION

The increasing demand for digital electronics and advanced computing systems has led to a growing need for processors that are fast, energy-efficient, scalable, and cost-effective. This is especially true in the area of embedded systems, Internet of Things (IoT), and real-time computing, where processors are needed to deliver trustworthy computation with low power consumption and flexible architecture. Therefore, processor design and implementation has become a major research area in academia and industry [3], [9]. The modern embedded system needs processors that can provide efficient computation and design flexibility and scalability for future technological advancements [10]. The conventional processor architectures, such as x86 and Advanced RISC Machine (ARM), are widely used in commercial applications owing to their high performance and optimized design. Nevertheless, these architectures are proprietary and need to be licensed, thus hindering their adaptability in academic research and experimental development. The closed design of these architectures hampers their customization and flexibility for a particular application. As such, researchers are currently concentrating on open and flexible processor architectures that facilitate innovation and customization without any licensing restrictions [2]. Reduced Instruction Set Computer - Fifth Generation (RISC-V) has been identified as a strong open-source Instruction Set Architecture (ISA) that overcomes the above-mentioned problems by providing a flexible platform for processor development. The ease of the instruction set, architecture, and scalability of RISC-V make it a popular choice for embedded systems, research, and Field Programmable Gate Array (FPGA) development. The 32-bit base integer instruction set of RISC-V architecture (RV32I) has been widely accepted due to its simplicity and sufficient functionality for various computing tasks [3], [4]. It has been proved in various research papers that RISC-V processors are efficient for embedded systems and FPGA development, highlighting their flexibility, scalability, and performance improvements [1], [9], [10].

The first generation of RISC-V processors was developed using single-cycle processor architecture because of its simplicity. In single-cycle processors, the entire process of instruction execution is completed within a single clock cycle. The simplicity of hardware design increases the complexity of the processor. Some research has been conducted on the implementation and analysis of single-

cycle RISC-V processors and their FPGA implementation, proving the feasibility of single-cycle RISC-V processors in educational and simple embedded systems, as well as showing their limitations in terms of performance [5], [6], [7].

To reduce these limitations, pipelining methods have been widely used in contemporary processor designs. In pipelining, the execution of instructions is divided into several stages, allowing for the simultaneous execution of multiple instructions, thus increasing the instruction throughput and processor speed. However, pipelined processors also handled some challenges with respect to data hazards, control hazards, and structural hazards, which must be rectified carefully to get the desired output. Some techniques used to handle hazards in pipelined processors, such as data forwarding, pipeline stalling, and hazard control units, have been used in pipelined RISC-V processors to handle these challenges [8], [15]. Research has also been carried out on different pipelined processor designs and techniques to optimize performance and power consumption for embedded systems [11].

There have been a few FPGA-based designs of pipelined RISC-V processors proposed to improve performance and hardware resource utilization. The pipelined five-stage architecture has been used widely due to its optimal trade-off between performance and complexity. The pipelined architecture stages are Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB) stages, which support parallel functioning of instructions. Recent research works have shown the design and FPGA implementation of five-stage pipelined RISC-V processors using Verilog HDL, which revealed the improvements in performance, speed, and power consumption over single-cycle architectures [12]. Other research works have also shown similar FPGA implementation and performance analysis, thus proving the efficiency of pipelined architectures in embedded systems and real-time applications [13], [14].

Moreover the five-stage pipelined architectures, other research papers on enhanced hazard-free pipelined architectures and efficient high-performance RISC-V processor designs have also been conducted to further improve the performance and power efficiency of IoT and embedded systems [11], [15]. Other research work on the implementation of pipelined RISC processor using Verilog Hardware Description Language (HDL) on FPGA platforms also provides important insights into pipelined architecture design and hardware optimization techniques [16].

In this paper, a 32-bit FPGA-based RISC-V processor with RV32I instruction set is designed and implemented using a five-stage pipelined architecture. The processor is designed with a Harvard memory architecture to avoid structural hazards by using separate instruction and data memories. Data hazards are reduced using data forwarding techniques, and control hazards are managed using pipeline stalling techniques with a custom-designed hazard control unit. The processor is described using Verilog HDL, simulated for functional verification, and implemented on an FPGA platform to analyze its performance using arithmetic and loop-based programs [12], [13], [14].

2. RV32I Architecture

2.1. Instruction Set Architecture (ISA)

The architecture of this design is compatible with contemporary operating systems and to serve as a target for compilers. It is designed to maintain simple hard-ware, making it simpler to develop small and efficient processors. Every architecture of RISC-V must contain RV32I as a core component, and additional functionality can be layered on top of that. The address of the next instruction is held by the program counter to be executed. The instruction from memory is fetched by this address, and then the instruction is analyzed to obtain the results from register memory. An Arithmetic Logical Unit (ALU) performs the mathematical and logical operations. According to the instruction's type, the result is either retained back in the register file or in memory. RV32I instruction set has six types of instructions including I-type, U-type, B-type, S-type, J- type, R-type.

2.2. RISC-V Processor Design

The major functional blocks of the proposed RISC-V processor are described below. Program Counter (PC): PC contains 32-bit register with subsequent instruction's address to be carried out. Normally, it updates by adding four (PC + 4) to move to the next instruction. However, for branch and jump instructions, it gets a new address, which changes the order in which instructions are executed.

- PC Adder: The PC Adder calculates the address of the next instruction. It keeps adding four to the current PC value, which helps in finding the next instruction's address in sequence.
- Instruction Memory: It is a type of read-only memory containing 64 KB of size (2048 × 32 bits) that holds the instructions given by program counter. It utilizes the address provided by the PC to fetch

instructions continuously while the program runs.

- Register File: This file comprises of 32 general pur- pose registers, each of which is 32 bits wide. It has dual independent read ports that are asynchronous and a write port that is synchronous with the clock. The write operation is controlled by a wr_en signal provided by the specific instruction decoding logic. The register x0 is hardwired to a constant zero value, while the remaining 31 registers are available for general-purpose computation.
- Immediate Value Block: The Immediate Value Block extracts and creates immediate values from instructions based on their format. These values are sign-extended and sent to the next pipeline stage for arithmetic or address calculations.
- Arithmetic Logic Unit: An ALU of size 32-bit per- forms arithmetic–logic processing operations like subtraction, bitwise operations, addition, shifts, and comparisons. The specific task is chosen with the help of a 4-bit control line from instruction decoder. The ALU is also leveraged to compute branch and jump addresses.
- Extension Block: The Extension Block ensures that data is properly formatted during memory operations. Regarding LW and SW instructions, it uses the entire 32-bit data. In the case of half-word or byte instructions, it chooses the appropriate 16-bit or 8-bit data and sign-extends it before processing or storing it back.
- Data Memory: The Data Memory is a read-write memory with a size of 16 KB (512 × 32 bits). It supports asynchronous read operations and synchronous write operations. This module is activated during load and store instructions.
- Branch Comparator Block: The Branch Comparator compares two register values and produces signals like equality or greater-than. These signals help the instruction decoder decide whether a branch instruction is met.
- Instruction Decoder: The Instruction Decoder takes the fetched instruction and produces all the required control signals for the processor. It includes a main control unit that generates pipeline control and selection signals, as well as an ALU decoder that creates specific control signals for the ALU.

2.3. Pipeline Stages of the Proposed RISC-V Processor

A five-stage pipeline is used in the suggested processor to enhance overall speed as well as instruction throughput. During the IF level, program counter is incremented while the instruction is accessed from memory. The ID stage examines register operands and decodes the instruction. The EX-stage performs logical or mathematical operations. In the case of load and store instructions, MEM reads or writes to data memory, while the final phase stores the result to the register file. This pipelining design is more efficient because it eliminates delays and enables the processor to execute multiple instructions simultaneously.

A. Instruction Fetch (IF)

This stage accesses the next instruction from memory. It includes the Instruction Memory, PC, and PC adder, along with the branch control logic. The PC keeps the address of the instruction being executed and is updated by adding four (PC+ 4) in sequence. When there is a branch command recognized in the decode phase of the instruction, the pipeline stalls for three clock cycles, with a No Operation (NOP) instruction inserted to avoid incorrect execution. The fetched instruction is moved to the next stage.

A. Instruction Decode (ID)

During the fetching of an instruction, the Instruction Decode phase understands the instruction and pre- pares for the necessary actions. This stage includes the Register file, Immediate Value Generator, Branch Comparator, and Instruction Decoder. The file provides source operands, while the immediate block produces necessary immediate values based on the instruction format. Branch decisions are made at this stage. When a branch is detected, the Hazard Control Unit generates halt signals to ensure the correct functioning of the pipeline. All control signals and operational values are forwarded to the next stages through pipeline registers.

B. Execute (EX)

In Execution phase all mathematical and logical operations are carried out. It contains blocks like ALU and extension. Arithmetic Logic unit uses values from register and immediate block or from the PC based on the requirement. The output may be arithmetic, logical or memory address, which is given to data file or register file in the following stages.

C. Memory Access (ME)

It contains the Data file and performs the execution of Load Word (LW) and Store Word (SW) instructions. For a LW instruction, the data is read from memory asynchronously and for store instruction, the data is written to memory synchronously, depending on the write enable signal. If the instruction didn't have memory

access, the ALU's output is passed to the next stage without accessing data memory.

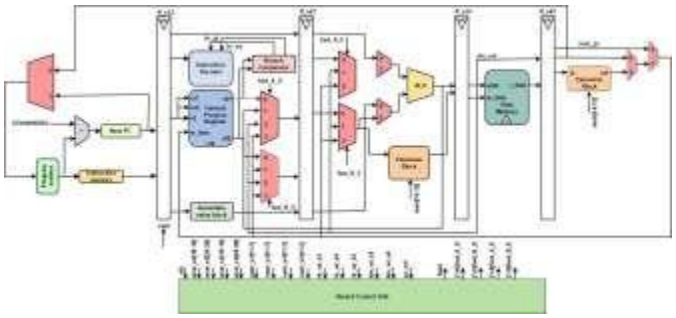

Fig-1: Architecture of Five-Stage Pipelined RV32I RISC-V Core

D. Write Back (WB)

It completes the instruction's execution and writes back final result into the register file. It includes multiplexers and an extension block that selects the correct data to be written. The write-back data can be an output from the ALU, data memory or the next PC value for JAL and JALR instructions. Depending on the instruction type (like, lh, or lb), the data is formatted correctly before being written into the destination register.

This stage accesses the next instruction from memory. It includes the Instruction Memory, PC, and PC adder, along with the branch control logic. The PC keeps the address of the instruction being executed and is updated by adding four (PC+ 4) in sequence. When there is a branch command recognized in the decode phase of the instruction, the pipeline stalls for three clock cycles, with a No Operation (NOP) instruction inserted to avoid incorrect execution. The fetched instruction is moved to the next stage.

2.4. Results

A. Functional Verification of the Proposed Processor

The proposed Five-Stage Pipelined RV32I RISC-V Core was functionally verified with the help of behavioral simulation in Xilinx Vivado. The simulation ensures the correct flow of instructions through all phases of the pipeline. Figure 2 represents the simulation waveform for the execution of several RV32I instructions. Each instruction passes through the pipeline stages consecutively showing proper pipeline behavior. The program counter increases correctly, and instruction values are observed at every stage without unexpected pauses or incorrect changes. The exact functioning of the proposed processor is also confirmed by observing the execution of specific instructions from the program sequence during the simulation. Some of the basic instructions are as follows:

- 00800413 (ADDI x8, x0, 8) - This instruction sets register x8 with the immediate value 8. During the EX-stage, the ALU performs the addition, and the result is written back to the x8 register in final stage.
- 00402183 (LW x3, 4(x0)) - The load instruction determines the effective memory location in the EX - phase and then fetches values from data file in the MEM stage. The value is written to register x3 in the WB stage.
- 00218233 (ADD x4, x3, x2) - This instruction performs addition between two registers x3 and x2 using the ALU. The result is generated in the EX -stage and stored in register x4 during the write- back, confirming correct ALU and forwarding operation.

Fig-2: Simulation waveform demonstrating instruction flow and register–memory activity across pipeline stages.

B. Register and Data Memory Analysis

reg_mem[0:31][31:0]	00000000,00000000,00000002,000000...
[0][31:0]	00000000
[1][31:0]	00000000
[2][31:0]	00000002
[3][31:0]	00000005
[4][31:0]	00000007
[5][31:0]	00000000
[6][31:0]	00000009
[7][31:0]	00000000
[8][31:0]	00000008
[9][31:0]	00000000
[10][31:0]	00000000

Fig-3: Register file value updates during instruction execution.

To examine data movement in the processor, the contents of register and data memory were monitored during the simulation. The register values and data memory updates are observed at the appropriate clock cycles, matching the execution of load, store, and arithmetic instructions. Fig. 3 and Fig. 4 illustrate

the changes in register and data memory values during instruction execution.

data_mem[0:511][31:0]	00000008,00000000,00000000,000000...
[0][31:0]	00000008
[1][31:0]	00000000
[2][31:0]	00000000
[3][31:0]	00000000
[4][31:0]	00000008
[5][31:0]	00000000
[6][31:0]	00000008
[7][31:0]	00000000
[8][31:0]	00000000
[9][31:0]	00000009
[10][31:0]	00000000

Fig-4: Data memory value updates during instruction execution

The correct registers are updated in the WB phase, and data file is modified only during valid store operations. This confirms that the hazard control unit and pipeline registers function correctly, preventing data corruption and ensuring proper synchronization between pipeline stages.

B. FPGA Implementation of Fibonacci Code

Fig-5: Implementation of Fibonacci series on FPGA

The Fibonacci code was created and run on the FPGA board. The processor calculated the Fibonacci number successions using a set of instructions, and the FPGA generated the expected results. This ensures that register updates are handled correctly in each step to ensure arithmetic and control operations on the FPGA semi-conductor device. The experiment also demonstrates the dependability of the RISC-V core in performing iterative algorithms, validates that data route operations and control signals function properly in a hardware-accelerated environment, and highlights efficient use of FPGA resources.

C. FPGA Resource Utilization Analysis

After simulation verification, the processor was synthesized using Xilinx Vivado. The synthesis report shows that the design utilizes 6190 Look-Up Tables (LUTs) out of 53200 available, resulting in 11.64% utilization. It also uses 17304 flip-flops (FFs) out of 106400, corresponding to 16.26% utilization. Additionally, the design consumes 12 I/O pins out of 200, which is 6.00% utilization.

Resource	Utilization	Available	Utilization %
LUT	6190	53200	11.64
FF	17304	106400	16.26
IO	12	200	6.00

Fig-6: Resource Utilization of Five-Stage Pipelined RV32I RISC-V Core

The relatively low utilization of resources indicates that the processor design is area-efficient and does not heavily consume FPGA resources. This confirms that the proposed five-stage pipelined RV32I processor can be successfully synthesized and easily fit into the target FPGA with sufficient resources remaining for further extensions.

D. Timing Performance Analysis

Setup		Hold	
Worst Negative Slack (WNS):	8.612 ns	Worst Hold Slack (WHS):	0.394 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	1	Total Number of Endpoints:	1

Fig-7: Timing Analysis of FPGA Implementation

The synthesized design was analyzed for timing with a 100 MHz clock constraint of 10 ns period. The timing analysis result shows that all user-defined constraints are met with a worst negative slack (WNS) of +8.612 ns and total negative slack (TNS) of 0.000 ns. There were no failing endpoints, which verifies that the design satisfies setup timing at the desired frequency. The design also satisfies hold timing with a worst hold slack (WHS) of

+0.394 ns. This result shows that the processor is functional at 100 MHz. The throughput advantage of the proposed design comes from pipelining. Since the processor uses a five-stage pipeline, different stages of multiple instructions can execute concurrently. After the pipeline is filled, the processor can ideally complete close to one instruction per cycle for instruction streams with minimal hazards. Thus, the pipelined model enhances instruction throughput than non-pipelined execution while maintaining low-cost.

E. Power Analysis of the FPGA Implementation

Power was calculated using Vivado based on the synthesized netlist. The total estimated on-chip power

consumption is 0.33 W. This includes 0.221 W dynamic power and 0.109 W static device power. The dynamic power is mainly contributed by internal signal switching and logic activity, while clock power remains low. The obtained values provide a useful estimate of power con- Sumption and show that the design can operate within a low-power range suitable for FPGA-based embedded applications

Fig-8: Power analysis of five stage pipelined RV32I RISC-V core

F. Comparison with Existing FPGA-Based RISC-V Implementations

To evaluate the proposed processor, a comparison was performed with existing FPGA-based RISC-V implementations. The comparison is based on FPGA resource usage and power values reported in the respective works. As shown in Table I, single-cycle processors typically require fewer flip-flops because they do not include pipeline registers. However, pipelined processors need additional registers between stages, which increases flip flop usage but improves throughput by allowing parallel instruction execution. The proposed five-stage pipelined design provides moderate LUT utilization and a low estimated power of 0.33 W, showing feasibility for FPGA-based embedded processor applications.

Table -1: Comparison of Resource Utilization and Power Consumption

REFERENCES	PIPELINE	LUTS	FF	POWER(WATTS)
[5]	Single cycle	4671	3964	NA
[12]	Single cycle	7963	17442	0.244
[9]	4-stage	7534	3967	0.268
Proposed	5-stage	6190	17304	0.33

3. CONCLUSIONS

In this study, the architecture is built on the proposed instruction set and is used to create and realize a 32-bit FPGA-based RISC-V core. Verilog was used to design the CPU, and both simulation and FPGA implementation were used to test and confirm its functionality. To manage data and control hazards and ensure correct instruction execution and stable pipeline operation, a special hazard control unit was used. This unit combined data forwarding and pipeline stalling techniques. The processor's functional correctness and pipeline behavior were confirmed through the successful execution of benchmark programs that included arithmetic and iterative operations. In contrast with a single-cycle processor design, the experimental results show higher instruction throughput and lower power consumption. The proposed processor is well-suited for embedded applications, processor design education, and academic research because it offers a well-balanced performance and hardware usage.

ACKNOWLEDGEMENT

The authors would like to express their sincere gratitude to the project guide, Dr. B. Vasudeva, for his valuable guidance, continuous support, and encouragement throughout the completion of this work.
The authors also extend their thanks to the Head of the Department, Dr. V. Jagan Naveen, for providing the necessary facilities and support.
The authors are thankful to all the faculty members for their guidance and cooperation.
Finally, the authors express their gratitude to their family and friends for their constant support and motivation.

REFERENCES

1. A. Birari, P. Birla, K. Varghese and A. Bharadwaj, "A RISC-V ISA Compatible Processor IP," 2020 24th International Symposium on VLSI Design and Test (VDAT), Bhubaneswar, India, 2020, pp. 1-6, doi: 10.1109/VDAT50263.2020.9190558.
2. A. Raveendran, V. B. Patil, D. Selvakumar and V. Desal- phine,"A RISC-V instruction set processor-micro-architecture design and analysis," 2016 International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA), Bengaluru, India, 2016, pp. 1-7, doi: 10.1109/VLSI- SATA.2016.7593047.

3. E. Cui, T. Li and Q. Wei,"RISC-V Instruction Set Architecture Extensions: A Survey," in IEEE Access, vol. 11, pp. 24696- 24711, 2023, doi: 10.1109/ACCESS.2023.3246491

4. A. Waterman, Y. Lee, D. A. Patterson and K. Asanovic´, "The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2011-62, 2011.

5. A. La Gala, M. Chiariello, M. Malanchini, M. Tam- baro and M. De Matteis,"Design and Test-Verification of a Single-Cycle RISC-V Microprocessor on FPGA," 2024 31st IEEE International Conference on Electronics, Circuits and Systems (ICECS), Nancy, France, 2024, pp. 1-4, doi: 10.1109/ICECS61496.2024.10848919.

6. Anjana K. M, Anusha S. B, Dikshitha U and Shilpa V, "Implementation of RISC-V Single Cycle Core," International Journal of Advanced Research in Computer and Communica- tion Engineering (IJARCCE), vol. 14, no. 1, Jan. 2025, doi: 10.17148/IJARCCE.2025.14143.

7. D. K. Dennis et al.,"Single cycle RISC-V micro archi- tecture processor and its FPGA prototype," 2017 7th In- ternational Symposium on Embedded Computing and Sys- tem Design (ISED), Durgapur, India, 2017, pp. 1-5, doi: 10.1109/ISED.2017.8303926.

8. W. Mo, Y. Wang, H. Sun and J. Liu,"An SDPF RISC-
V Processor with Two-stage Pseudo-pipelined Architecture for IoT Applications," 2024 IEEE 17th International Con- ference on Solid-State Integrated Circuit Technology (IC- SICT), Zhuhai, China, 2024, pp. 1-3, doi: 10.1109/IC- SICT62049.2024.10832035.

9. K. Kırali and C. B. Fidan, "Implementation of FPGA based 32- bit RISC-V processor," Engineering Science and Technology, an International Journal, vol. 70, p. 102139, 2025.

10. A. Singh, A. Kumar, A. Singh, R. Anirudh and K. N. Pushpalatha, "Design and Implementation of RISC-V ISA (RV32IM) on FPGA," International Journal of VLSI Signal Processing, vol. 10, pp. 17–21, 2023.

11. G. Xie, Z. Wang, Y. Zhang, R. Li and X. Liu,"Compact and High-Performing Five-Stage Pipeline RISC-V Microprocessor Designed for IoT Applications," IEEE Design Test, vol. 42, no. 2, pp. 79-88, 2025, doi: 10.1109/MDAT.2024.3501952.

12. F. Hussain and S. Sarkar,"Design and FPGA Imple- mentation of Five Stage Pipelined RISC-V Processor," 2024 IEEE 9th International Conference for Convergence in Technology (I2CT), Pune, India, 2024, pp. 1-6, doi: 10.1109/I2CT61223.2024.10544184.

13. P. N. V. M and L. V,"Design and Implementation of 5-Stage Pipelined RISC-V Processor on FPGA," 2024 28th International Symposium on VLSI Design and Test (VDAT), Vellore, India, 2024, pp. 1-6, doi: 10.1109/VDAT63601.2024.10705665.

14. M. R. Maheshwar, E. Chitra, A. N. V. V. D. Manikanta Pavan and M. Reddy Balaji, "FPGA-Based Implementation of a Five-Stage Pipelined RISC-V Processor Using Xilinx Zynq- 7000," in Proc. International Conference on Information and Communication Technology for Intelligent Systems, 2025, pp. 511–521.

15. I. Thanga Dharsni, K. S. Pande and M. K. Panda,"Optimized Hazard Free Pipelined Architecture Block for RV32I RISC-V Processor," 2022 3rd International Conference on Smart Elec- tronics and Communication (ICOSEC), Trichy, India, 2022, pp. 739-746, doi: 10.1109/ICOSEC54921.2022.9952122.

16. J. Jeemon, "Pipelined 8-bit RISC processor design using Ver- ilog HDL on FPGA," 2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT), Bangalore, India, 2016, pp. 2023-2027, doi: 10.1109/RTEICT.2016.7808194.