A Modular Open-Source Pipeline for Teaching and Prototyping in Computer Architecture

D VENKATESH
Department of Electronics and Communiction Engineering
PSG Institute Of Technology and Applied Reseach
Coimbatore, India
venkateshd@psgitech.ac.in

AMIRTHA T
Department of Electronics and Communiction Engineering
PSG Institute Of Technology and Applied Reseach
Coimbatore, India
211104@psgitech.ac.in

K SHREE HARINI
Department of Electronics and Communiction Engineering
PSG Institute Of Technology and Applied Reseach
Coimbatore, India
211145@psgitech.ac.in

SHRIREKKHA K
Department of Electronics and Communiction Engineering
PSG Institute Of Technology and Applied Reseach
Coimbatore, India
211147@psgitech.ac.in

SOWJANYA R
Department of Electronics and Communiction Engineering
PSG Institute Of Technology and Applied Reseach
Coimbatore, India
211150@psgitech.ac.in.

DHANYA S
Department of Electronics and Communiction Engineering
PSG Institute Of Technology and Applied Reseach
Coimbatore, India
231124@psgitech.ac.in.

***Abstract*—This paper presents an open-source, modular five-stage pipelined RISC processor developed using Verilog for teaching and prototyping applications in computer architecture. Unlike many academic designs, this framework emphasizes modularity, extensibility, and accessibility, making it ideal for education and early-stage research. The design includes a custom instruction set, clear architectural documentation, and cycle-accurate simulations, enabling students and researchers to thoroughly explore processor behavior, pipelining concepts, and hazard scenarios. Extensive simulation-based validation provides detailed insight into instruction execution flow, control signals, and pipeline interactions. By bridging theory with practical, hands-on exploration through openly available source code and simulation resources, this work offers a reproducible and adaptable platform for students, educators, and researchers studying RISC architectures, pipelining, and hardware verification.**

Keywords—RISC Processor, Verilog HDL, Pipeline Architecture, Modular Design, Open-Source Hardware, Simulation-Based Learning

I. INTRODUCTION

A. Importance of Understanding Processor Pipelines in Education and Research

Processor pipelining is a foundational concept in computer architecture that bridges theoretical principles with practical hardware implementation. Understanding the flow of instructions through stages such as fetch, decode, execute, memory access, and write-back is critical for students, educators, and researchers alike. A clear grasp of pipelined design not only strengthens core digital design skills but also lays the groundwork for advanced concepts such as superscalar execution, hazard resolution, and branch prediction. In research, pipelining serves as a steppingstone for building domain-specific processors and optimizing custom instruction sets for specialized applications.

B. Gap in Accessible, Modular Platforms for Hands-On Learning

While textbooks and simulation tools provide theoretical knowledge, there is a noticeable lack of accessible, modular, and reproducible platforms that allow learners to experiment with processor internals. Many academic designs are either overly simplified—limiting exploration—or too complex, making them unsuitable for introductory or intermediate-level education. Furthermore, closed-source or hardware-specific implementations restrict customization and broader usage. This gap leaves students and researchers without a reliable, open platform to understand, test, and extend the concepts of pipelining in a hands-on manner.

C. Aim and Contribution of This Work

This work addresses the above gap by developing a **modular, open-source five-stage pipelined RISC processor** written in Verilog. The processor is designed with educational clarity and research extensibility as primary goals. It provides:

- A **fully modular architecture** that isolates functional blocks for easy understanding and debugging.
- A **customizable instruction set**, enabling exploration of both standard and domain-specific operations.
- **Simulation-based validation** with clear waveform visualization, allowing cycle-level analysis of pipeline behaviour and hazard scenarios.
- **Open-source accessibility**, ensuring reproducibility and ease of adoption in academic and research environments.

By bridging theory with practical simulation, this work provides a versatile platform for students, educators, and researchers to explore RISC architecture, pipelining techniques, and hardware verification with greater confidence and depth.

II. LITERATURE REVIEW

This section surveys representative educational cores and RISC pipeline projects across ARM-compatible, MIPS-style, and RISC-V designs, as well as HLS- and FPGA-centric implementations. Collectively, prior work demonstrates functional processors and specific microarchitectural features (e.g., hazard handling, prediction, forwarding). However, most efforts are either tightly coupled to particular toolchains/boards or focus on peak performance rather than modularity, open-source accessibility, and classroom integration. Our work targets this gap with a simulation-first,

openly available, easily extensible 5-stage pipeline oriented toward teaching workflows.

[1] Geun-young Jeong and Ju-sung Park designed and verified a 32-bit ARM7TDMI-compatible core with a 3-stage pipeline, six register banks, a 32-bit ALU, and a 4-cycle MAC unit, validated against a commercial ARM simulator. They demonstrated real-time ADPCM/SOLA and accurate (non-real-time) MP3 decoding. While robust and application-driven, the work is not positioned as a modular educational platform or open resource, and its latch-based, ARM-specific flow limits classroom extensibility.

[2] Bhardwaj and Murugesan presented a 32-bit MIPS-style RISC processor in Verilog, emphasizing ISA simplicity and correct simulation. The study is valuable for introductory pipeline exposure but offers limited guidance on modular decomposition, instructor-ready artifacts, or reproducible teaching labs.

[3] Yuan and Zhu proposed a flexible FPGA-based SoC with configurable ISA, enabling multiple MCUs tailored to applications and efficient resource use at low clock rates. They also implemented a 5-stage WIPS core relying on compile-time scheduling to mitigate hazards. The platform is configuration-rich but remains FPGA-centric; classroom adoption is constrained by hardware availability and less emphasis on step-by-step pedagogical materials.

[4] A 32-bit pipelined design employing intermediate storage ("pipes") reduced execution time from 2500 ns to 1000 ns and supported R/I/J formats. The focus is throughput gains and embedded suitability (e.g., PDAs, image processing), with less attention to open-source packaging, modular classroom labs, or simulation-first exploration.

[5] Kulshreshtha et al. compared 16-bit (non-pipelined Harvard) and 32-bit (MIPS-inspired pipelined) RISC processors, revealing ~70% faster operation for 32-bit at higher power. This informs architectural trade-offs but does not provide a reusable, modular teaching framework or openly shared design artifacts.

[6] A 5-stage 32-bit RISC-V processor (RV32I) in Bluespec SystemVerilog targeted Virtex-6 and ASIC (65/130 nm), integrating branch prediction, forwarding, and IEEE-754 FPU. While technically comprehensive and open-ISA aligned, the toolchain (BSV) and hardware emphasis may raise barriers for Verilog-centric courses and simulation-only labs.

[7] Singh et al. analysed a RV32I core with a six-stage folded pipeline on Virtex-7/UltraScale, offering parameterized control (prediction, dual-side cache, AHB-Lite3). Though modular in features, the study optimizes power/performance on specific FPGAs; turnkey materials for hands-on learning and code-level pedagogy are not the central aim.

[8] Mantovani et al. introduced HL5, a RV32IM core created via SystemC/HLS, showing reduced design complexity and competitive performance with RTL. The work advances HLS-based methodology but depends on HLS tools and FPGA/CMOS flows, which may not align with Verilog-only curricula or low-overhead classroom setups.

[9] A dual-pipeline, in-order superscalar RV32IM[A] core on Virtex-7 achieved 3.84 CoreMark/MHz with prediction, ECC, VM, and split caches. It demonstrates strong performance and modular peripherals, yet its complexity and hardware reliance make it less accessible for early-stage teaching of baseline pipeline concepts.

[10] Islam et al. detailed a high-speed pipelined execution unit (0.13 μm) reaching 714 MHz via efficient data-flow control and dedicated forwarding hardware. The contribution is performance-oriented and technology-specific rather than an open, pedagogy-ready platform.

[11] A 32-bit 5-stage RV32I Harvard processor with static branch prediction and hazard detection improved CPI from 1.53 to 1.26 in Vivado tests, suitable for educational demos. However, materials emphasize RTL correctness and FPGA outcomes more than modular, simulation-first learning scaffolds and extensible lab exercises.

[12] RISC32 (MIPS-compatible, 5-stage) analysed and resolved data hazards through prioritized forwarding with fallback stalls, targeting correctness for IoT-scale workloads. The hazard study is a strong reference, yet the work centers on correctness/performance rather than packaging as an open, modular teaching platform. For output on the A4 paper size. If you are using US letter-sized paper, please close this file and download the Microsoft Word, Letter file.

III. SYSTEM DESIGN AND METHODOLOGY

The complete Verilog source code, testbenches, and .mem files, along with the detailed ISA documentation, are available in the open-source repository: https://github.com/Amirthathenappa/open-source-processor-code

A. Architecture Overview

The proposed processor is a **32-bit, five-stage pipelined RISC core** designed entirely in Verilog HDL. It follows the classic **instruction fetch (IF), instruction decode (ID), execution (EX), memory (MEM), and write-back (WB)** pipeline model, enabling concurrent instruction execution and improved throughput over a single-cycle architecture. The design supports a **custom instruction set** tailored for educational use, with support for arithmetic, logical, memory, branch, and basic I/O operations. By maintaining a clean and modular implementation, the processor architecture facilitates

both easy comprehension for beginners and rapid prototyping for advanced users.

B. Five-Stage Pipeline

The processor pipeline is divided into five distinct stages:

TABLE I. Five-Stage Pipeline

STAGE	FUNCTION
Instruction Fetch (IF)	Fetches instruction from instruction memory and increments the program counter.
Instruction Decode (ID)	Decodes the instruction, reads operands from the register file, and generates control signals.
Execution (EX)	Performs arithmetic or logical operations using the ALU and evaluates branch conditions.
Memory Access (MEM)	Accesses data memory for load/store instructions and handles I/O operations if applicable.
Write-Back (WB)	Writes the result of computation or data fetch back to the register file.

This structure ensures cycle-level concurrency, where up to five instructions can be in different stages simultaneously, improving throughput without significant design complexity..

C. Modular Design Approach

A key strength of the processor lies in its **fully modular architecture**, where each stage and functional block is implemented as an independent Verilog module. Major modules include:

- **Program Counter (PC) Unit**
- **Instruction Fetch and Decode Units**
- **Register File**
- **ALU and Execution Logic**
- **Data Memory and Memory Access Unit**
- **Write-Back Unit**
- **Control and Hazard Handling Logic**

- This modularity provides several advantages:

- **Ease of Debugging:** Each module can be simulated and verified independently, simplifying error localization.
- **Extensibility:** New instructions, peripherals, or functional units can be integrated with minimal changes to existing modules.
- **Pedagogical Clarity:** Clear boundaries between stages help students map theory to implementation more effectively.

D. Development Tools and Workflow

The processor was developed and validated using a combination of open and industry-standard tools:

- **HDL Development:** Verilog HDL was used for design and testbench creation, with optional SystemVerilog constructs for advanced verification scenarios.
- **Simulation:**
  - **EDA Playground** for quick functional verification and waveform visualization.
  - **Vivado Simulator** for timing-aware simulation and synthesis testing.
- **Testbenches:** Custom testbenches were created to validate individual modules and full-pipeline integration, with step-by-step instruction flow observation.
- **Waveform Analysis:** Signal behaviour at each pipeline stage was analysed using simulation waveforms to verify control flow, data paths, and hazard handling.

This **simulation-first approach** ensures that the processor design is fully reproducible and accessible without requiring dedicated FPGA hardware, making it particularly suitable for academic labs and remote learning environments.

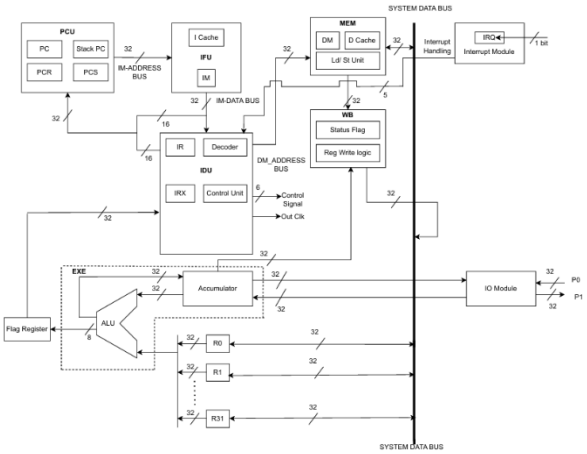

Figure 1: Modular Five-Stage RISC Processor Architecture.

IV. EDUCATION RELEVANCE AND USE-CASES

A. Laboratory Applications

The processor was designed with **teaching and prototyping** in mind. Its modular architecture and simulation-based workflow make it highly suitable for undergraduate and early research labs where hardware availability is limited. Students can:

- Write custom assembly programs, convert them to hexadecimal, and load them into the .mem file.
- Simulate the processor and observe real-time updates in the register file and pipeline stages.
- Modify modules to experiment with custom instructions or hazard-handling techniques.

This interactive process bridges theoretical concepts with practical exposure, reinforcing topics like **instruction execution, pipelining, and hazard resolution**.

B. ISA Customisation

One of the key educational benefits of this design is the **easeof ISA customization**. Students and researchers can:

- Extend the **R-Type instructions** to include specialized arithmetic or logical operations.
- Add **new branch or memory operations** to explore control hazards or pipeline stalls.
- Modify encoding to study the trade-offs in instruction length, opcode space, and decoding complexity.

This makes the platform not only a **learning tool** but also a **testbed for architectural experimentation**.

C. Integration with Verification Environments

The processor integrates smoothly with simulation platforms like **EDA Playground** and **Vivado**. Waveform analysis allows students to:

- Visualize data flow between pipeline stages.
- Verify correctness of custom instructions.
- Explore timing issues and hazard resolutions in a controlled environment.

This combination of modular design and robust simulation support allows even beginners to explore advanced architecture concepts without the need for FPGA deployment.

V. ISTRUCTION SET ARCHITECTURE (ISA)

This processor implements a **32-bit Reduced Instruction Set Computer (RISC)** architecture organized into **five instruction types**:

- **R-Type (Register–Register)**
- **I-Type (Register–Immediate)**
- **J-Type (Jump)**
- **B-Type (Branch)**
- **M-Type (Memory Access)**

All instructions are **32 bits wide**, with fixed field positions for consistent decoding and pipeline control. The hexadecimal version of each instruction is written in the .mem file for simulation.

A. R-Type instruction (Register-Register)

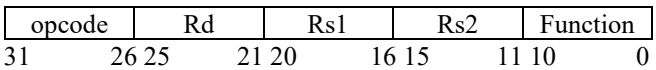

Figure 2: R-Type Instruction Format

TABLE 2. R-Type Instructions

Mnemonic	Opcode	Function Code	Description
ADD	000000	00000000001	rd = rs1 + rs2
SUB	000000	00000000010	rd = rs1 - rs2
MUL	000000	00000000011	rd = rs1 * rs2
AND	000000	00000000100	rd = rs1 & rs2
OR	000000	00000000101	rd = rs1 | rs2
XOR	000000	00000000110	rd = rs1 ^ rs2
SHL	000000	00000000111	rd = rs1 << rs2
SHR	000000	00000001000	rd = rs1 >> rs2
CMP	000000	00000001001	Set flags based on rs1 - rs2
MOV	000000	00000001010	rd = rs1

B. I-Type Instructions (Register-Immediate)

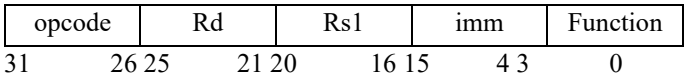

Figure 3: I-Type Instruction Format

TABLE 3. I-Type Instructions

Mnemonic	Opcode	Function	Description
ADDI rd, rs1, imm	000001	0001	rd = rs1 + imm
SUBI rd, rs1, imm	000001	0010	rd = rs1 - imm
MULI rd, rs1, imm	000001	0011	rd = rs1 * imm
ANDI rd, rs1, imm	000001	0100	rd = rs1 & imm
ORI rd, rs1, imm	000001	0101	rd = rs1 | imm
XORI rd, rs1, imm	000001	0110	rd = rs1 ^ imm
SHLI rd, rs1, imm	000001	0111	rd = rs1 << imm
SHRI rd, rs1, imm	000001	1000	rd = rs1 >> imm
MOVI rd, imm	000001	1001	rd = imm

C. J-Type Instruction (Jump Instruction)

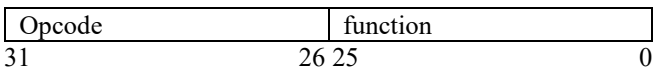

Figure 4: J-Type Instruction Format

TABLE 4. J-Type Instructions

Mnemonic	Opcode	Function	Description
JUMP addr	000010	Anythin except 26'b0	PC = addr

Mnemonic	Opcode	Function	Description
RET	000010	00000000000000000000000000	PC = Stack_PC (Return from Subroutine)

D. B-Type Instruction

opcode	Rs1	Rs2	function	Offset address
31 26	25 21	20 16	15 14	13 0

Figure 5: B-Type Instruction Format

TABLE 5. B-Type Instructions

Mnemonic	Opcode	Function	Description
BEQ rs1, rs2, offset	000011	00	if (rs1 == rs2) PC += offset
BNE rs1, rs2, offset	000011	01	if (rs1 != rs2) PC += offset
BGT rs1, rs2, offset	000011	10	if (rs1 > rs2) PC += offset
BLT rs1, rs2, offset	000011	11	if (rs1 < rs2) PC += offset

E. M-Type Instruction

opcode	f	Rd	Offset	Rs1
31 26	25 23	24 20	19 4	5 0

Figure 6: M-Type Instruction Format

TABLE 6. M-Type Instructions

Mnemonic	Opcode	Function	Description
LW rd, offset(rs1)	000100	f_mem = 1	Load rd = Mem[rs1 + offset]
SW rs1, offset(rs2)	000100	f_mem = 0	Store Mem[rs2 + offset] = rs1

VI. RESULTS AND ANALYSIS

A. Simulation Waveforms

The processor was functionally verified through behavioral simulation in Vivado. The .mem file was loaded with machine instructions encoded in hexadecimal format. Upon execution, register values and bus activities were monitored through the testbench environment.

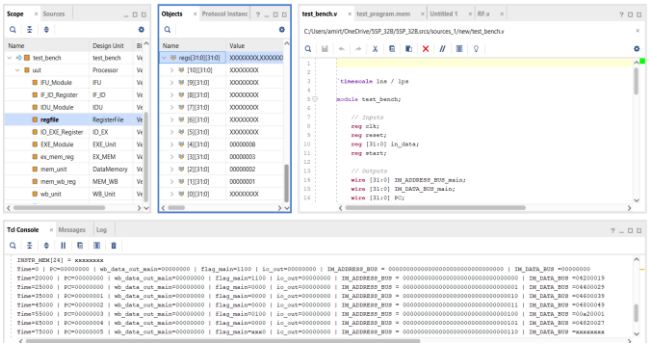

Figure 2 shows the simulation output for a sample program, where the **instruction memory**, **register file**, and **program counter (PC)** updates can be observed in real time. Key observations include:

- Proper fetching of instructions from the **instruction memory (IM_DATA_BUS)**.
- Sequential PC increments verifying pipeline fetch–decode–execute stages.
- Correct updates in the **register file (regfile)** with expected computed values.
- Flags (flag_main) and I/O signals (io_out) responding correctly to arithmetic and branch instructions.

This verifies the **end-to-end execution flow** of the processor, demonstrating the correctness of instruction decoding, execution, and write-back operations.

B. Performace Insights

- **Instruction Execution:** The processor successfully executed a sequence of **R-type, I-type, and memory access instructions** without hazards or functional mismatches.
- **Debug Visibility:** The modular design allowed for clear observation of data flow through different pipeline stages, simplifying the debugging and verification process.

VII. DISCUSSION

A. Bridging Theory and Practice

This design effectively bridges the gap between theoretical instruction pipeline concepts and hands-on processor implementation. Students can visualize how fetch, decode, execute, memory, and write-back stages interact, reinforcing topics like data hazards, forwarding, and branching that are often challenging to grasp from textbooks alone.

By simulating real programs, learners gain an intuitive understanding of instruction flow, timing behavior, and performance trade-offs, enabling deeper conceptual clarity and readiness for advanced digital design courses or industry applications.

B. Modularity and Extensibility

The **modular architecture** of the design provides significant benefits for both educational and research use:

- Individual modules (e.g., ALU, register file, memory units) can be modified or replaced without affecting the overall pipeline.

- Facilitates **ISA extensions**, such as adding floating-point support or custom instructions for specialized applications.
- Eases debugging and incremental improvements by isolating errors to specific blocks during simulation or FPGA deployment.

This flexibility makes the processor a **scalable platform** for both undergraduate projects and research-driven enhancements.

C. Challenges and Lessons Learned

During the development and verification phases, several challenges were encountered:

- **FPGA Resource Constraints:** Deploying the full five-stage pipeline on smaller devices required optimization of register usage and synthesis settings.
- **Debugging Complexity:** Identifying subtle timing mismatches or pipeline hazards during simulation demanded careful signal monitoring and structured testbenches.
- **Toolchain Limitations:** Transitioning from simulation to hardware occasionally exposed differences in behavior, highlighting the need for **robust verification environments** and well-documented workflows.

These challenges not only improved the design but also provided valuable insights into **real-world hardware development practices** such as iterative refinement and systematic debugging.

VIII. LIMITATIONS AND FUTURE WORK

A. Current Limitation

While the proposed processor design achieves the goals of modularity, educational usability, and functional correctness, some limitations were observed:

- **Basic Hazard Handling:** The current pipeline design does not include advanced **data hazard detection** or **forwarding mechanisms**, which may cause stalls in certain instruction sequences.
- **Limited Memory Features:** The memory interface supports basic load/store operations but lacks features like **cache hierarchies** or burst transfers, which could improve execution performance.
- **Performance Constraints:** On FPGA implementations, the achievable clock frequency and throughput are limited by resource availability and routing delays.
- **Minimal Peripheral Support:** Interaction with external peripherals is basic and does not fully leverage potential I/O operations or interrupt-driven workflows.

B. Future Enhancement

To enhance both **functionality and educational value**, several improvements are proposed:

- **Advanced Hazard Detection and Forwarding:** Implementing real-time hazard detection and bypassing units to reduce pipeline stalls and improve CPI.
- **Cache Integration:** Adding instruction and data caches to emulate real-world processor performance and teach concepts like locality and cache coherence.
- **Pipeline Optimization:** Exploring deeper pipelines or superscalar execution to increase instruction throughput and benchmark performance.
- **Peripheral Expansion:** Introducing UART, GPIO, or SPI modules to demonstrate system-level integration and real-world interfacing.
- **Enhanced Debugging Support:** Adding internal performance counters and debug registers to make educational analysis more intuitive.
- **Custom ISA Extensions:** Allowing students or researchers to define and integrate custom instructions for specialized domains like DSP or cryptography.
- **ASIC Implementation Exploration:** Migrating the design to ASIC flow to evaluate area, power, and timing trade-offs beyond FPGA constraints.

IX. CONCLUSION

This work presents an open-source, modular five-stage pipelined 32-bit RISC processor, developed in Verilog to bridge the gap between theoretical computer architecture concepts and hands-on implementation. Its modular structure simplifies debugging and encourages extensibility, making it a practical platform for education, experimentation, and early-stage research.

Through simulation and verification, the processor demonstrates functional correctness and provides a foundation for students to explore key concepts such as pipelining, instruction execution, and ISA customization. The design's accessibility—combined with compatibility with widely used tools like Vivado and EDA Playground—further enhances its usability in academic settings.

Looking forward, the framework offers significant potential for future enhancements, including advanced hazard handling, cache integration, and peripheral interfaces, making it an adaptable resource for deeper research and performance optimization.

REFERENCES

[1] [1] G.-y. Jeong and J.-s. Park, "Design of 32-bit RISC processor and efficient verification," *Proc. of 7th Korea-Russia International Symposium on Science and Technology (KORUS)*, Ulsan, South Korea, 2003, pp. 222–227, doi: 10.1109/KORUS.2003.1214839.

[2] [2] P. Bhardwaj and S. Murugesan, "Design and simulation of a 32-bit RISC-based MIPS processor using Verilog," *International Journal of Research in Engineering and Technology (IJRET)*, vol. 5, no. 11, pp. 10–15, 2016, doi: 10.15623/ijret.2016.0511030.

[3] [3] S.-Y. Yuan and B.-Y. Zhu, "A Flexible FPGA-Based ISA Configurable SoC platform," *arXiv preprint*, arXiv:2105.12678, 2021. [Online]. Available: https://arxiv.org/abs/2105.12678

[4] [4] S. Aruna, K. S. Naik, D. Madhusudan, and V. Venkatesh, "Implementation of 5-stage 32-bit microprocessor based on WIPS architecture," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 1, pp. 4557–4561, Nov. 2019, doi: 10.35940/ijitee.A4899.119119.

[5] [5] A. Kulshreshtha, A. Moudgil, A. Chaurasia, and B. Bhushan, “Analysis of 16-bit and 32-bit RISC processors,” in *Proc. of 7th Int. Conf. on Advanced Computing and Communication Systems (ICACCS)*, Coimbatore, India, 2021, pp. 1318–1324, doi: 10.1109/ICACCS51430.2021.9441873.

[6] [6] A. Raveendran, V. B. Patil, D. Selvakumar, and V. Desalphine, “A RISC-V instruction set processor-microarchitecture design and analysis,” in *Proc. of Int. Conf. on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA)*, Bengaluru, India, 2016, pp. 1–7, doi: 10.1109/VLSI-SATA.2016.7593047.

[7] [7] A. Singh, N. Franklin, N. Gaur, and P. Bhulania, “Design and implementation of a 32-bit ISA RISC-V processor core using Virtex-7 and Virtex UltraScale,” in *Proc. of 5th IEEE Int. Conf. on Computing, Communication and Automation (ICCCA)*, Greater Noida, India, 2020, pp. 126–130, doi: 10.1109/ICCCA49541.2020.9250850.

[8] [8] P. Mantovani, R. Margelli, D. Giri, and L. P. Carloni, “HL5: A 32-bit RISC-V processor designed with high-level synthesis,” in *Proc. of IEEE Custom Integrated Circuits Conf. (CICC)*, Boston, MA, USA, 2020, pp. 1–8, doi: 10.1109/CICC48029.2020.9075913.

[9] [9] T. Gokulan, A. Muraleedharan, and K. Varghese, “Design of a 32-bit dual pipeline superscalar RISC-V processor on FPGA,” in *Proc. of 23rd Euromicro Conf. on Digital System Design (DSD)*, Kranj, Slovenia, 2020, pp. 340–343, doi: 10.1109/DSD51259.2020.00062.

[10] [10] S. Islam, D. Chattopadhyay, M. K. Das, V. Neelima, and R. Sarkar, “Design of high-speed pipelined execution unit of 32-bit RISC processor,” in *Proc. of Annual IEEE India Conference (INDICON)*, New Delhi, India, 2006, pp. 1–5, doi: 10.1109/INDCON.2006.302780.

[11] [11] M. Z. Naveed, M. Amar, A. Hussain, and Z. A. Awan, “Design and implementation of 5-stage 32-bit RISC-V pipeline processor on FPGA,” *Preprint*, 2024.

[12] [12] W. P. Kiat, K. M. Mok, W. K. Lee, H. G. Goh, and I. Andonovic, “A comprehensive analysis on data hazard for RISC32 5-stage pipeline processor,” *Unpublished manuscript*, 2017.

[13] [13] Your Name(s), “A Modular Open-Source Pipeline for Teaching and Prototyping in Computer Architecture,” *[Conference/Journal Name]*, Year (to be updated upon submission).