Comparative Performance Analysis of Single-Stage and Pipelined RISC-V Processors on FPGA

Vallisree S[1], Karthigaa S[2], Gangiesh S[3], Nisanth T[4]
[1,2,3,4]*Department of Electronics Engineering, Madras Institute of Technology, Anna University, Chennai, India*

***Abstract*—This paper presents a comparative performance analysis of two RISC-V processor architectures: a single-stage processor and a five-stage pipelined processor. Leveraging the open-source and modular nature of the RISC-V instruction set architecture, both processors were designed and implemented using Verilog HDL. The single-stage architecture offers design simplicity by executing one instruction per clock cycle, whereas the pipelined architecture divides instruction execution into five stages to improve instruction throughput. C test programs were compiled into RISC-V machine code using a cross-compilation toolchain and converted into Intel HEX (.hex) format for loading into the instruction memory. Both architectures were simulated and synthesized on an FPGA to evaluate key performance metrics such as clock period, resource utilization, and execution speed. The results show that the pipelined processor achieves a maximum operating frequency of 158.48 MHz, compared to 51.07 MHz for the single-stage design, resulting in a speedup of up to 3.1 times for long instruction sequences. This study highlights the trade-off between architectural simplicity and execution throughput and demonstrates the effectiveness of RISC-V as a flexible platform for processor design.**

***Index Terms*—RISC-V, Pipelining, Single-Stage Processor, FPGA (Field-Programmable Gate Array), Verilog HDL, Instruction Set Architecture (ISA), Hazard Mitigation.**

I. INTRODUCTION

Processor architecture plays a crucial role in determining the performance, power efficiency, and scalability of modern computing systems. With the increasing demand for customizable and open hardware platforms, Reduced Instruction Set Computer (RISC) architectures have gained significant attention due to their simplicity, efficiency, and suitability for implementation on Field-Programmable Gate Arrays (FPGAs).

Among these, RISC-V has emerged as a prominent open-source Instruction Set Architecture (ISA), offering modularity, extensibility, and freedom from proprietary licensing constraints.

RISC-V follows a clean load–store architecture with fixed-length instructions, enabling efficient hardware implementation and easy support for instruction-level parallelism. These features make RISC-V particularly suitable for academic research, processor prototyping, and FPGA-based design exploration. As processor performance is heavily influenced by architectural choices, it is important to analyse how different execution models impact speed and hardware utilization.

One of the most fundamental architectural comparisons is between single-stage and pipelined processor designs. In a single-stage processor, all instruction phases—fetch, decode, execute, memory access, and write-back—are completed within a single clock cycle. While this approach offers simplicity and ease of understanding, it results in long critical paths and limited operating frequency. In contrast, a pipelined processor divides instruction execution into multiple stages, allowing concurrent processing of multiple instructions and significantly improving throughput.

Understanding the impact of execution models on performance is fundamental in processor design. Analysing these architectural approaches helps in identifying performance bottlenecks, design trade-offs, and optimization opportunities in RISC-V-based processor systems.

II. RELATED WORKS

Several works have explored FPGA implementation of RISC-V processors, targeting simplicity, performance, or hardware acceleration. Poli et al. [1] presented a lightweight, single-stage RV64I processor on Basys 3 FPGA using only seven instructions. The design achieved low resource usage (322 LUTs, 229 FFs) and 2.276 DMIPS/MHz at 100 MHz, demonstrating efficiency for educational purposes.

Ahmed and Harun-Ur-Rashid [2] implemented a 32-bit, five-stage pipelined RV32I core with hazard handling and verified it on an Altera DE2-115 FPGA. The design is suitable for small FPGAs and low-power applications, with potential extensions for vector computation.

Markov and Romanov [3] extended a single-stage RISC-V processor with the Zbb bit-manipulation instructions. Their BBMU module increased FPGA resource usage but improved program execution speed by ~30% and reduced code size by ~37%, highlighting performance benefits of ISA extensions.

Praveen et al. [4] designed a single-stage RV32I processor for Spartan-6 FPGA with full instruction cycle completion. Verification against the Venus simulator confirmed correctness, and synthesis reports detailed minimal hardware footprint.

Nguyen-Hoang et al. [5] implemented a 32-bit VexRiscV processor with AES, SHA-1, and RSA accelerators on Nexys4DDR FPGA and 65 nm ASIC. Custom ISA instructions controlled the accelerators, achieving up to 4× throughput improvement for AES, demonstrating integration of cryptographic hardware with RISC-V cores.

III. SINGLE-STAGE AND PIPELINED RISC-V PROCESSOR DESIGN

A single-stage RISC-V processor is a fundamental implementation of the RISC-V architecture in which each instruction is executed entirely within a single clock cycle. This design executes instruction fetch, decode, execution, memory access, and write-back in one clock period. Although simple, it effectively demonstrates the basic flow of instruction execution in RISC-V.

A. Working of Single-Stage Processor

Each clock cycle performs the following sequence of operations:

- Instruction Fetch (IF): The Program Counter (PC) holds the address of the next instruction. The instruction memory reads the instruction, and the PC is incremented by 4, as each RISC-V instruction is 32 bits.
- Instruction Decode (ID): The Control Unit decodes the fetched instruction and generates the necessary control signals. Simultaneously, the Register File reads the source registers (rs1 and rs2) specified by the instruction.
- Execution (EX): The ALU performs the required operation such as addition, subtraction, or comparison. For branch instructions, the ALU computes the target address and determines whether the branch condition is met.
- Memory Access (MEM): For load instructions, data is read from memory; for store instructions, data from a register is written to memory. ALU results for other instruction types bypass this stage.
- Write-Back (WB): The result from the ALU or memory is written back into the destination register (rd) of the register file.

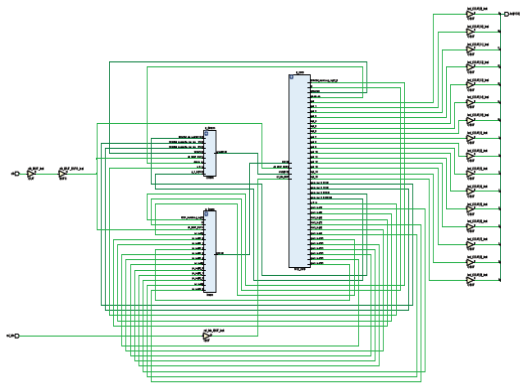

Fig. 1 Single-stage RISC-V Block Diagram

The overall flow is illustrated in Fig. 1, which shows the internal data path and control-signal interactions of a single-stage RISC-V processor. The diagram highlights the connectivity between components, enabling instructions to travel from fetch to write-back within one clock cycle. The PC continuously feeds the Instruction Memory, ensuring the processor always retrieves the correct instruction. The Control Unit interprets the opcode and generates signals to guide the ALU, Register File, Data Memory, and multiplexers (MUXes).

The ALU performs arithmetic and logic operations and generates the Zero flag for branch decisions. The Immediate Generator provides extended immediate values for I-type, S-type, B-type, U-type, and J-type instructions. MUXes positioned before the ALU and memory select appropriate inputs based on the instruction type, ensuring accurate data flow.

B. Pipelined RISC-V Processor Design

The limitations of the single-stage processor, particularly its long critical path and inefficient hardware utilization, motivate the adoption of pipelining. Pipelining is a fundamental architectural technique that improves processor throughput by partitioning instruction execution into multiple stages and allowing multiple instructions to be processed concurrently. The pipelined processor conforms to the RV32I ISA and demonstrates improved performance through instruction-level parallelism without modifying instruction semantics.

The architecture divides instruction execution into five stages: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write-Back (WB). Pipeline registers are inserted between successive stages to isolate their operations and enable multiple instructions to reside in different stages simultaneously.

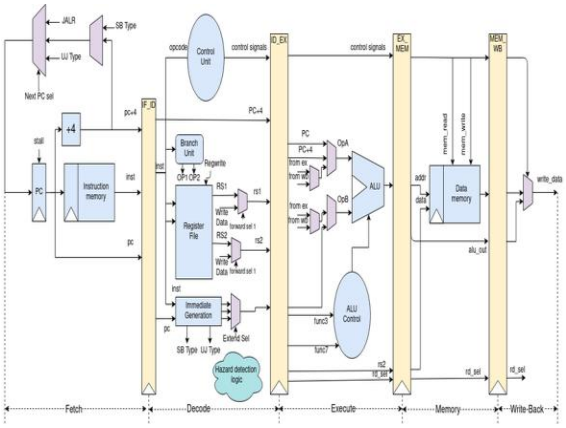

Fig. 2 Pipelined RISC-V Block Diagram

IV. INSTRUCTION SET IN RISC-V

The structure of the 32-bit RISC-V instruction formats is illustrated in Fig. 3. The figure shows how instruction fields are organized to represent different instruction types. These fields form the foundation of all 32-bit RISC-V instructions and ensure a consistent and modular encoding scheme. By rearranging field positions, RISC-V efficiently supports arithmetic operations, memory access, branching, and control-flow instructions. The standardized format simplifies hardware decoding, enabling faster and more reliable instruction execution. Additionally, the inclusion of immediate fields across multiple instruction formats enhances flexibility by allowing operations on both register-based and constant operands.

```
Disassembly of section .init:

00000000 <_start>:
   0:   02000117            auipc   x2,0x2000
   4:   40010113            addi    x2,x2,1024 # 2000400 <__stack>
   8:   02000197            auipc   x3,0x2000
   c:   7f818193            addi    x3,x3,2040 # 2000800 <__global_pointer$>
  10:   0040006f            jal     x0,14 <_cstart>

00000014 <_cstart>:
  14:       0b8002ef        jal     x5,cc <__riscv_save_0>
  18:       02000537        lui     x10,0x2000
  1c:       00000613        addi    x12,x0,0
  20:       17800593        addi    x11,x0,376
  24:       00050513        addi    x10,x10,0 # 2000000 <__bss_end>
  28:       104000ef        jal     x1,12c <memcpy>
  2c:       02000537        lui     x10,0x2000
  30:       00000593        addi    x11,x0,0
  34:       00000613        addi    x12,x0,0
  38:       00050513        addi    x10,x10,0 # 2000000 <__bss_end>
  3c:       114000ef        jal     x1,150 <memset>
  40:       02000537        lui     x10,0x2000
  44:       00050513        addi    x10,x10,0 # 2000000 <__bss_end>
  48:       124000ef        jal     x1,16c <_set_tls>
  4c:       00000593        addi    x11,x0,0
  50:       00000513        addi    x10,x0,0
  54:       008000ef        jal     x1,5c <main>
  58:   /-- 0000006f        jal     x0,58 <_cstart+0x44>
```

Fig. 3 RISC-V Instruction Format

V. SOFTWARE TOOLCHAIN AND MEMORY FILE GENERATION

To verify and execute programs on the Verilog-based RISC-V processor, executable machine code was generated and loaded into the instruction memory. Since the target processor follows the RV32I architecture, a RISC-V cross-compilation flow was required. This process was carried out on a Windows host system using the Ubuntu environment provided by Windows Subsystem for Linux (WSL), which enables native Linux tool support without virtualization overhead.

Within the Ubuntu WSL environment, a RISC-V GNU cross-compilation toolchain was used to compile C programs into RISC-V executable code suitable for bare-metal execution. The compilation flow translated high-level C source files into RISC-V machine instructions and generated memory initialization files

compatible with FPGA-based instruction memory. A custom linker configuration was employed to map program code and data into predefined memory regions corresponding to the processor's instruction and data memories.

The compilation process produced a disassembly listing file for debugging and verification, and a hexadecimal memory file containing the final machine code. The disassembly file was used to inspect the generated RISC-V instructions and confirm correct code generation, while the hexadecimal file was used to initialize the instruction memory during simulation and FPGA implementation. Sample outputs of these files are shown in Fig. 4 and Fig. 5, respectively.

This software toolchain forms a critical bridge between high-level program development and hardware execution, enabling functional validation of the RISC-V processor design on both simulation and FPGA platforms.

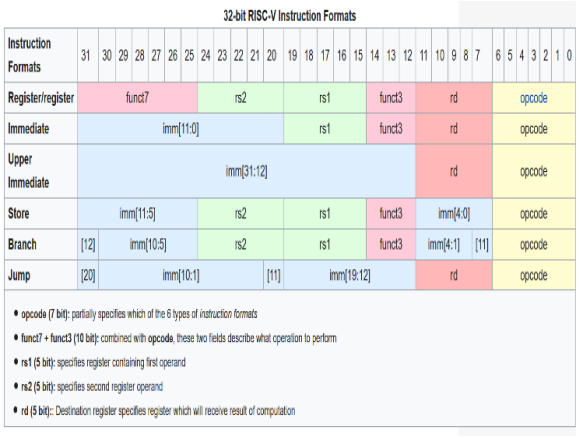

Fig. 4 The generated linker script file

```
@00000000
02000117 40010113 02000197 7F818193
0040006F 11C002EF 02000537 00000613
20000593 00050513 18C000EF 02000537
00000593 00000613 00050513 19C000EF
02000537 00050513 1AC000EF 00000593
00000513 008000EF 0000006F FE010113
00112E23 00812C23 02010413 020007B7
FEF42223 00600793 FEF42623 00100793
FEF42423 0240006F FEC42583 FE842503
100000EF 00050793 FEF42423 FEC42783
FFF78793 FEF42623 FEC42783 FCF04EE3
FE442783 FE842703 00E7A023 FE842783
00078513 01C12083 01812403 02010113
00008067 FC010113 00000313 01B12623
00C0006F FC010113 FF000313 01A12823
01912A23 01812C23 01712E23 00C0006F
FC010113 FE000313 03612023 03512223
03412423 03312623 03212823 02912A23
02812C23 02112E23 40610133 00028067
FF010113 01212023 00912223 00812423
00112623 00028067 00C12D83 01010113
00012D03 00412C83 00812C03 00C12B83
01010113 00012B03 00412A83 00812A03
00C12983 01010113 00012903 00412483
00812403 00C12083 01010113 00008067
```

Fig. 5 The generated hexadecimal file

VI. RESULTS AND DISCUSSION

A. Functional Verification

Functional verification was carried out using the Vivado simulator by executing representative C programs compiled into RISC-V machine code and loaded into the instruction memory.

For the single-stage processor, a “sum of n numbers” program was executed for n = 10. The final result was correctly stored in data memory as 0x37 (decimal 55), confirming correct arithmetic and control-flow operation. A factorial program was also tested, and for n = 10, the processor produced the correct output of 10! = 3,628,800, represented as 0x375F00. The corresponding simulation waveforms are shown in Fig. 6 and Fig. 7.

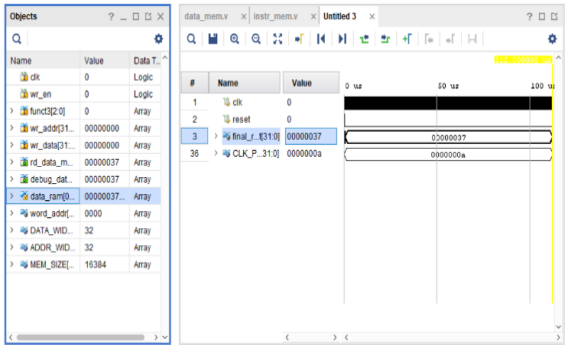

Fig. 6 Simulated output of sum of 1 to 10 numbers in single-stage RISC-V processor

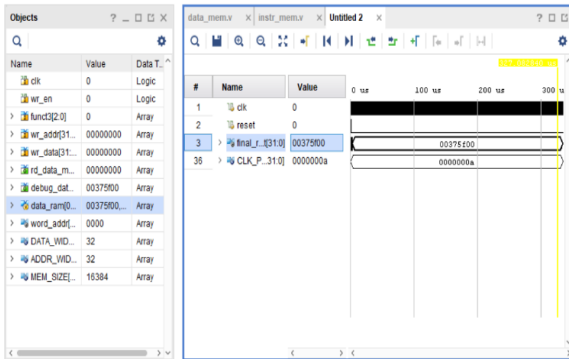

Fig. 7 Simulated output of factorial of 10 in single-stage RISC-V processor

For the pipelined processor, the sum of numbers program was executed for n = 11, yielding the correct result of 0x42 (decimal 66) without pipeline hazards affecting correctness. Additionally, a factorial program for n = 5 produced the expected output of 5! = 120, stored as 0x78 in data memory. Simulation results for the pipelined design are shown in Fig. 8 and Fig. 9. These results confirm correct pipeline operation and data forwarding.

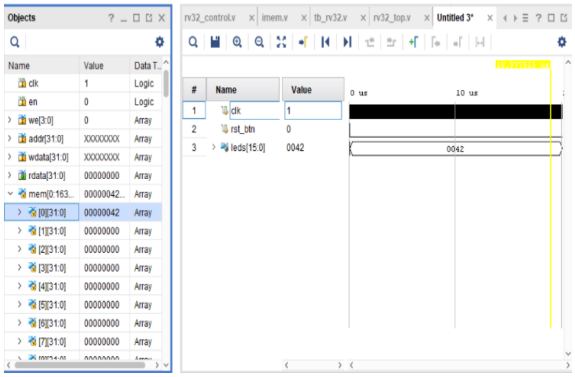

Fig. 8 Simulated output of sum of 1 to 11 numbers in pipelined RISC-V processor

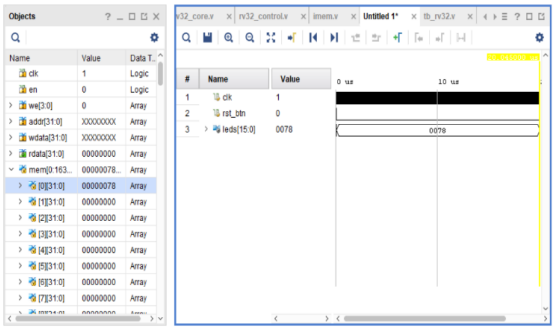

Fig. 9 Simulated output of factorial of 6 in pipelined RISC-V processor

B. Hardware Implementation

Both processors were synthesized and implemented on the Basys 3 FPGA. For the single-stage processor, the sum of numbers programs correctly displayed the final result 0x37 on the on-board LEDs, as shown in Fig. 10.

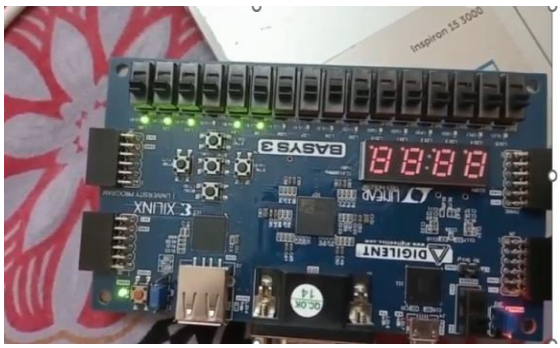

Fig. 10 FPGA implementation of addition of 1 to 10 numbers in single-stage RISC-V processor

For the pipelined processor, the sum of numbers and factorial programs were successfully executed on hardware, displaying results 0x42 and 0x72, respectively, on the LEDs (Fig. 11 and Fig. 12). These results validate the correctness of the hardware implementation and the consistency between simulation and FPGA execution.

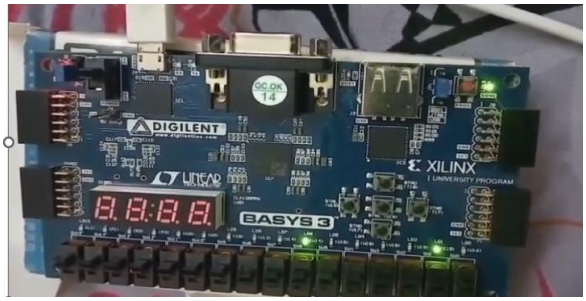

Fig. 11 FPGA implementation of addition of 1 to 11 numbers in pipelined RISC-V processor

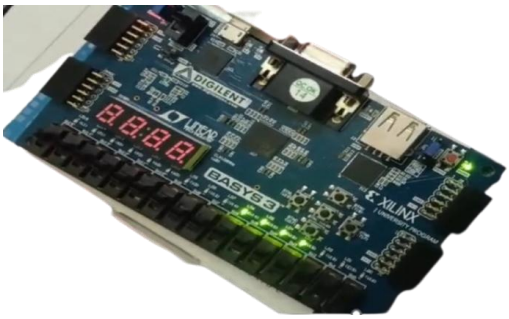

Fig. 12 FPGA implementation of factorial of 6 in pipelined RISC-V processor

C. Maximum Frequency Analysis

The maximum operating frequency was derived from post-synthesis timing reports. The critical path delay and maximum frequency are given by:

$$T_{oriti}{}^{k}{}_{al} = T_{oonstraint} - WNS \quad \text{..... (Eq. 1)}$$

$$F_{max} = 1 / T_{oriti}{}^{k}{}_{a} \quad \text{..... (Eq. 2)}$$

For the single-stage processor, the critical path delay was 19.58 ns, resulting in a maximum frequency of F_{max} = 51.07 MHz. For the pipelined processor, the critical path delay was significantly reduced to 6.31 ns, yielding F_{max} = 158.48 MHz. This demonstrates the effectiveness of pipelining in shortening the critical path and improving clock frequency.

D. Speedup Factor

The architectural speedup was evaluated using slack-adjusted clock periods. The execution time for single-stage and pipelined processors is given by:

$$T_{sin}{}^{k}{}_{le} = n \times T_s \quad \text{.....(Eq. 3)}$$

$$T_{pipeline} = (k + n - 1) \times T_p \quad \text{..... (Eq. 4)}$$

$$Speedup = T_{sin}{}^{k}{}_{le} / T_{pipeline} \quad \text{..... (Eq. 5)}$$

where n is the number of instructions and k = 5 is the pipeline depth. For n = 10 instructions, the measured speedup was 2.2×. For n = 1000 instructions, the speedup increased to 3.09×. These results show that while pipelining provides moderate improvement for short programs, its performance benefits become significantly more pronounced for longer instruction sequences.

VII. CONCLUSION

Table I presents a quantitative comparison between the single-stage and five-stage pipelined RISC-V processor implementations in terms of performance, resource utilization, and power consumption.

TABLE I. COMPARISON OF SINGLE-STAGE AND PIPELINED PROCESSOR

Metric	Single-Stage Processor	Pipelined Processor
Target Clock Period	20 ns (50 MHz)	20 ns (50 MHz)
Max Frequency	51.07 MHz (19.579 ns)	158.48 MHz (6.31 ns)
Worst Negative Slack (WNS)	0.421 ns	13.690 ns
Timing Constraints Met?	Yes	Yes
Slice LUTs	1979	212
Slice Registers	2864	342
BLOCK RAM (BRAM) Tiles	0	16
Total On-Chip Power	0.193 W	0.189 W
Dynamic Power	0.121 W (at 50 MHz)	0.116 W (at 100 MHz)
Static Power	0.072 W	~0.073 W
Instruction Memory	64 KB	64 KB
Data Memory	64 KB	64 KB

The single-stage processor achieves a maximum operating frequency of 51.07 MHz, limited by the long critical path that spans all instruction phases within a single clock cycle. In contrast, the pipelined processor attains a maximum frequency of 158.48 MHz, providing an approximate 3.1× improvement. The significantly higher Worst Negative Slack (13.690 ns) for the pipelined design further confirms its strong timing margin.

Despite incorporating additional pipeline registers and hazard-handling logic, the pipelined design uses substantially fewer Slice LUTs and registers than the single-stage implementation. This is primarily due to efficient memory inference: the pipelined processor maps its instruction and data memories to dedicated Block RAM (BRAM) resources, while the single-

stage processor relies on LUT-based distributed memory.

Power analysis results indicate that both designs exhibit similar static power consumption, which is largely determined by the FPGA device itself. However, the pipelined processor demonstrates lower dynamic power consumption even when operating at a higher clock frequency. Overall, the results confirm that the five-stage pipelined RISC-V processor offers superior performance, scalability, and energy efficiency compared to the single-stage design, making it a more suitable choice for high-performance FPGA-based RISC-V implementations.

REFERENCES

[1] S. Poli, S. Saha, K. Zhai, and K. McDonald-Maier, "Design and implementation of a lightweight RISC-V processor on FPGA for educational purposes," IEEE Access, vol. 9, pp. 118456–118468, 2021.

[2] M. Ahmed and M. H. Ur-Rashid, "Design, implementation, and verification of a five-stage pipelined RISC-V core (RV32I ISA)," IEEE Open Journal of the Computer Society, vol. 6, pp. 45–56, 2025.

[3] I. Markov and A. Romanov, "Implementation of the RISC-V architecture with the extended Zbb instruction set," Microprocessors and Microsystems, vol. 92, p. 104561, 2022.

[4] A. Praveen, R. Kumar, and S. Natarajan, "RISC-V microarchitecture design on FPGA," in Proceedings of the International Conference on VLSI Design and Embedded Systems, Chennai, India, 2024, pp. 112–117.

[5] T. Nguyen-Hoang, P. Coussy, and E. Casseau, "Implementation of a 32-bit RISC-V processor with cryptography accelerators on FPGA and ASIC," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 8, pp. 1245–1256, Aug. 2022.

[6] A. Kumar and P. Reddy, "Implementation of a pipelined RISC-V processor with hazard detection and forwarding," International Journal of Innovative Research in Computer and Communication Engineering, vol. 9, no. 11, pp. 8701–8709, 2021.

[7] A. B. Varma and V. R. K. M. Rao, "Performance analysis of single-cycle and pipelined RISC-V processors using FPGA synthesis tools," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2021.

[8] D. M. Harris and S. L. Harris, Digital Design and Computer Architecture: RISC-V Edition, 2nd ed., Morgan Kaufmann, 2021.

[9] J. E. Kim, S. H. Lee, and Y. D. Kim, "A pipelined RISC-V processor design with low-latency memory access for FPGA implementation," IEEE Access, vol. 11, pp. 1–12, 2023.

[10] L. Poli, S. Saha, X. Zhai, and K. D. McDonald-Maier, "Design and implementation of a RISC-V processor on FPGA," in Proceedings of the 17th International Conference on Mobility, Sensing and Networking (MSN), Exeter, UK, 2021, pp. 1–6.

[11] K. Asanovic et al., "The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10," RISC-V Foundation, 2017.

[12] K. Khatri and A. Patel, "Comparative Analysis of Single-Cycle and Pipelined RISC Processors," International Journal of Scientific and Research Publications, vol. 11, no. 2, pp. 101–107, 2021.