Project Report On

Design and Implementation of RISC-V Processor with 5 Stage Pipelining

Submitted in Partial Fulfillment for the award of

Post Graduate Diploma in VLSI Design

(PG-DVLSI)

from

C-DAC, ACTS (Pune)

Guided by:

Dr. Sajish Chandrababu

Presented by:

Ms. Namrata Chande	**PRN: 250240133006**
Mr. Gourav Jalan	**PRN: 250240133010**
Ms. Preeti Sawant	**PRN: 250240133021**
Mr. Sarvesh Mane	**PRN: 250240133024**
Mr. Sirigidi Prashanth	**PRN: 250240133027**

Centre for Development of Advanced Computing
(C-DAC), Pune

ACKNOWLEDGEMENT

This project "Design and Verification of High Performance AXI4 Memory Interface" was a great learning experience for us and we are submitting this work to Advanced Computing Training School (CDAC ACTS).

We are very glad to mention the name of *Dr. Sajish Chandrababu* for their valuable guidance to work on this project. Their guidance and support helped me to overcome various obstacles and intricacies during the course of project work.

Our heartfelt thanks goes to *Ms. Namratha and Ms.Swati S* (Course Coordinator, *PG-DVLSI*) who gave all the required support and kind coordination to provide all the necessities like required hardware, internet facility, and extra Lab hours to complete the project and throughout the course up to the last day here in C-DAC ACTS, Pune.

TABLE OF CONTENTS

Abstract

With the rapidly growing adoption of RISC-V as an open-source instruction set architecture (ISA), this project presents the design and implementation of a 32-bit RISC-V processor featuring a classical 5-stage pipeline: instruction fetch, decode, execute, memory access, and writeback. The architecture aims to optimize performance by enabling instruction-level pipelining, hazard detection, and forwarding mechanisms.

The project involves designing synthesizable RTL code for pipeline stages and control units following the RISC-V RV32I base integer instruction set. The verification is done using simulation synthesis results are obtained on Xilinx Vivado targeting specified FPGA devices. The pipeline design incorporates control and data hazard mitigation techniques to ensure seamless instruction execution and throughput.

The report details the architecture design, pipeline stages, hazard mitigation, verification methodologies including UVM guidance, and presents simulation and synthesis analysis showcasing performance and resource utilization outcomes.

Keywords: RISC-V, 5-stage pipeline, hazard detection, forwarding, RTL design, UVM verification.

INTRODUCTION

The Reduced Instruction Set Computer V (RISC-V) is an open, extensible Instruction Set Architecture (ISA) designed to address modern computing needs from embedded to high-performance systems. The RISC-V specification promotes simplicity, modularity, and scalability, making it a popular choice for education, research, and commercial processor designs.

This project focuses on designing and implementing a 32-bit single-core RISC-V processor utilizing the classical five-stage pipeline architecture. Pipelining improves performance by overlapping the execution of instructions, thereby increasing throughput. The design employs comprehensive pipeline stages: Instruction Fetch (IF), Instruction Decode/Register Fetch (ID), Execute (EX), Memory Access (MEM), and Write Back (WB).

We present a synthesizable RTL design, integrating control logic, hazard mitigation units such as forwarding and stalling, and a register file, adhering to the RV32I base ISA. Verification is conducted using RTL simulations in ModelSim/QuestaSim and synthesis using Xilinx Vivado targeting FPGA platforms.

This report details the theoretical background, system design, optimization, verification methodology, and results validating our RISC-V pipeline implementation.

LITERATURE SURVEY

The evolution of RISC architectures has been pivotal in increasing processor efficiency and simplifying instruction sets. RISC-V, emerging from this lineage, stands out with its open and extensible approach. Earlier processors followed multi-cycle or single-cycle architectures, trading complexity for performance.

Pipelining is a classic enhancement that allows overlapping instruction phases. The canonical five-stage pipeline has been widely adopted and studied since the MIPS R2000 processor. Various works have addressed pipeline hazards such as data dependencies and control flow changes. Techniques include forwarding, stalling, and branch prediction.

Several RISC-V implementations in literature focus on modular design, verification using Universal Verification Methodology (UVM), and hazard control to balance complexity and throughput. Projects demonstrate efficient handling of R-type, I-type, branch, load/store instructions, with some extending the base RV32I set.

Research also emphasizes modular verification environments for RISC architectures, employing UVM sequences, drivers, monitors, and scoreboards for comprehensive functional and protocol compliance checking.

This project leverages these foundations, developing a pipelined RISC-V core validated through simulation and synthesis.

AIM OF PROJECT

- To design and implement a synthesizable 32-bit single-core RISC-V processor with a classical five-stage pipeline.
- To implement pipeline hazard detection and forwarding units to ensure data consistency and throughput.
- To develop a modular RTL design of pipeline stages alongside control units supporting the RV32I base instruction set.
- To verify the design using simulation tools and synthesize targeting FPGA platforms.
- To analyze results including simulation waveforms, resource utilization, and timing performance.
- To document the entire design process for educational and reference purposes.

SCOPE AND OBJECTIVE

Scope :

- Single-core 32-bit RISC-V processor implementing RV32I instructions.
- Five pipeline stages: IF, ID, EX, MEM, WB with pipeline registers.
- Hazard detection to handle load-use and branch hazards.
- Data forwarding to reduce pipeline stalls.
- Verification by RTL simulation.
- Synthesis on Xilinx FPGA for resource and speed analysis.

Objectives :

- Design RTL modules for each pipeline stage and supporting units.
- Implement control logic for instruction decoding and signal generation.
- Develop hazard detection and data forwarding modules.
- Verify functionality via testbenches and functional simulations.
- Synthesize design and analyze area, timing, and power.
- Prepare documentation and test reports.

THEORETICAL DESCRIPTION OF PROJECT

1. RISC-V Architecture Overview

RISC-V is a modular ISA comprising a base integer instruction set (RV32I) with optional extensions (multiplication, floating point, atomic instructions). This project targets the RV32I subset supporting 32-bit fixed-length instructions, arithmetic, logical, control flow, and load/store operations.

Key features :

- Load/store architecture: All data manipulation done via registers; memory accessed through loads/stores.
- Simple orthogonal instruction formats (R-type, I-type, S-type).
- 32 registers (x0 to x31) where x0 is hardwired to zero.
- Fixed 32-bit instruction width promoting straightforward decoding.

2. Five Stage Pipeline Architecture

The classic five-stage pipeline decomposes instruction execution into:

- Instruction Fetch (IF): Fetch instruction from memory using the program counter (PC).
- Instruction Decode/Register Fetch (ID): Decode fetched instruction, read operands from register file.
- Execute (EX): Perform ALU operations or calculate address/branch targets.
- Memory Access (MEM): Access data memory for loads and stores.
- Write Back (WB): Write results back to register file.

Each stage is separated by pipeline registers holding intermediate values. This design supports instruction-level parallelism where multiple instructions reside in various pipeline stages simultaneously, improving throughput over single-cycle designs.

3. Pipeline Hazards and Mitigation

Pipelining introduces hazards:

- Data hazards: Result from data dependencies between instructions. Mitigation via forwarding or stalls.
- Control hazards: Due to branch/jump instructions altering control flow. Mitigated by pipeline flush or branch prediction (basic design uses flushing).
- Structural hazards: Conflicts for resources, avoided here by design.

This design implements:

- Forwarding Unit: Routes data from MEM or WB pipeline stages back to EX stage to resolve data dependencies.
- Hazard Detection Unit: Detects load-use hazard and stalls pipeline as necessary.
- Branch Hazard Handling: Pipeline flush/stall on branch instructions to maintain correctness.

4. Control Unit and Datapath Overview

The control unit decodes opcode, funct3, and funct7 to generate control signals:

- Register write enable.
- ALU operation controls.
- Memory read/write enables.
- Branch and jump controls.
- Multiplexer select signals.

The datapath integrates:

- Register file.
- ALU with control inputs.
- Sign extension for immediate operands.
- Pipeline registers between stages.
- Hazard and forwarding units for safe and efficient data flow.

SYSTEM DESIGN AND OPTIMIZATION

1. Design Information

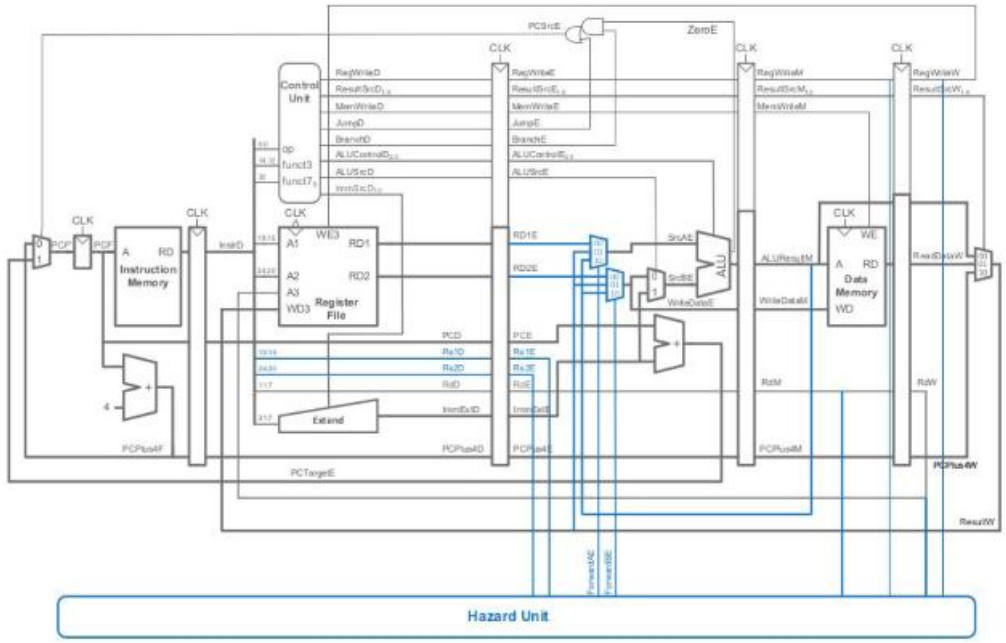
The design is partitioned into modules corresponding to pipeline stages and functional units:

- **IF Stage** : Program counter and instruction memory interface.
- **ID Stage** : Instruction decoder, register file, sign extender, and control units.
- **EX Stage** : ALU and branch address calculator.
- **MEM Stage** : Data memory interface.
- **WB Stage** : Multiplexers to select data for register write-back.

Pipeline registers capture outputs between stages maintaining state synchronization.

Optimization considerations:

- Minimized combinational delay within stages.
- Efficient hazard detection logic to restrict stall cycles.
- Forwarding paths implemented with minimal multiplexers.
- Resource sharing in FPGA synthesis with modular coding style.



The block diagram depicts the data and control flow through the five pipeline stages. Control signals from the control unit branch out to multiplexers, register file write enables, and ALU

controls. Hazard detection module monitors ID stage inputs and EX/MEM stage pipeline registers to detect stalls. Forwarding logic dynamically selects correct ALU inputs to avoid stalls.

2. **Implementation Considerations**

- Registers and pipeline registers are edge-triggered on the clock.
- Reset signal initializes all registers and stages.
- Hazard detection applies stalls only when necessary to maximize pipeline utilization.
- Forwarding logic prioritizes more recent data in MEM or WB stages.
- Instruction and data memories are modeled as synchronous RAM blocks.
- All modules parameterized to support easy scaling.

TOOL USED AND RESULT

The following tools are used-

1. XILINX VIVADO 2019 2.

ELABORATED DESIGN

Elaborated Design of RISC-V 5 stage pipeline is as follows:

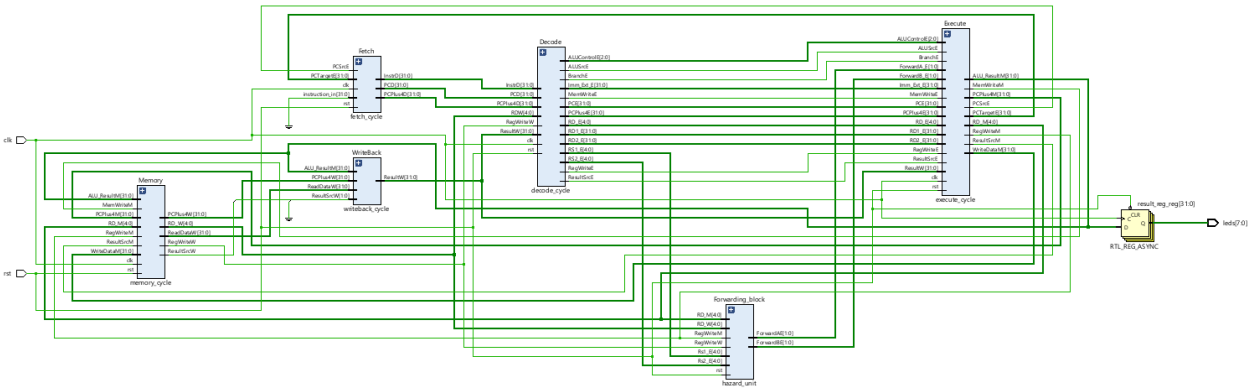
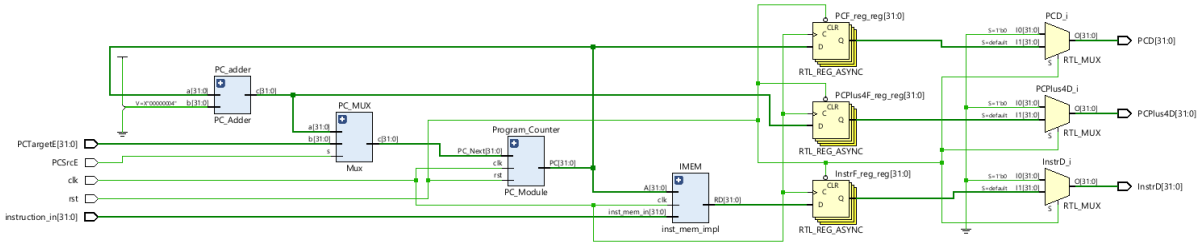
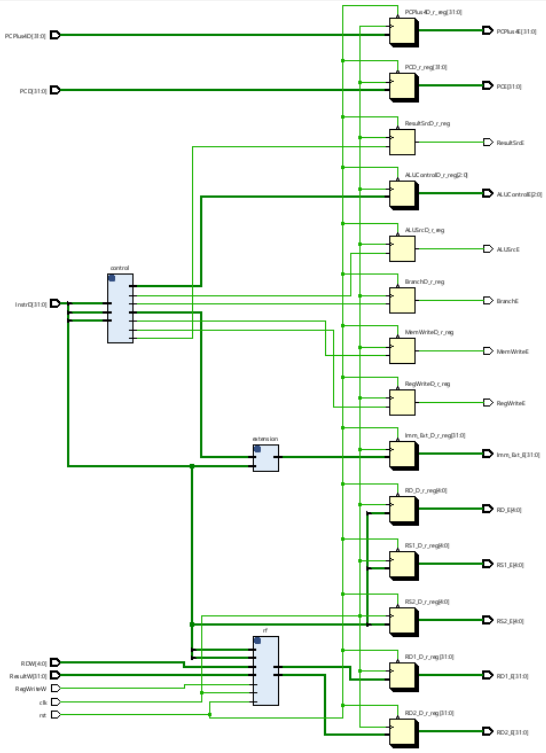


Fig: Schematic of RISC-V 5 stage Pipeline

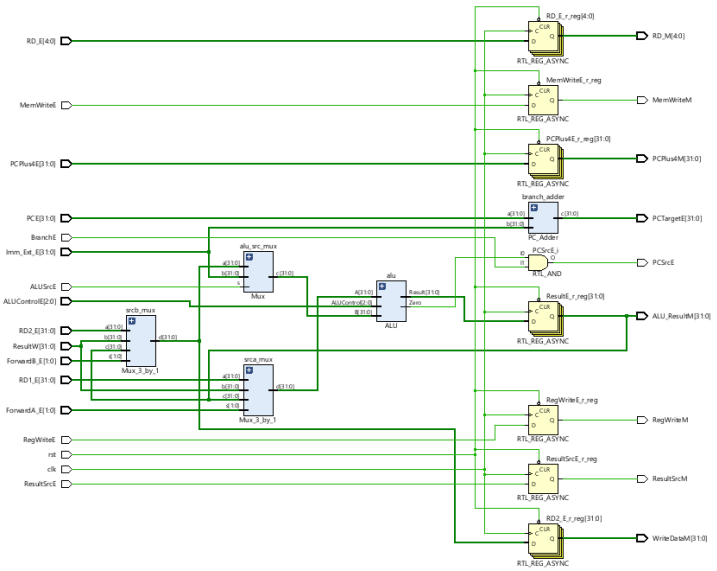
1. Fetch Cycle Schematic



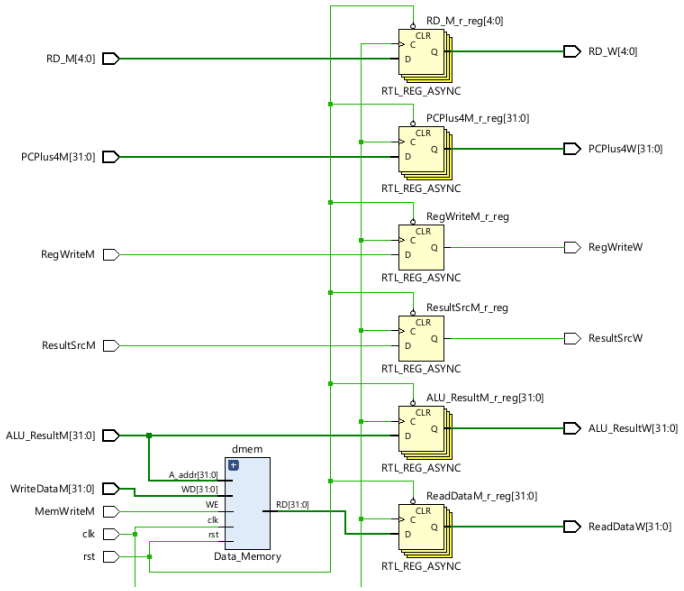
2. Decode Cycle Schematic



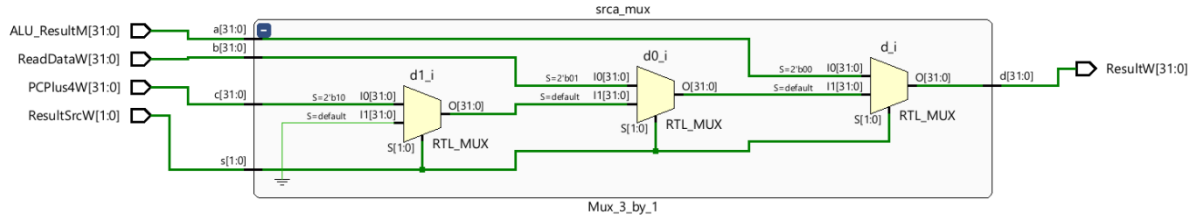
3. Execute Cycle Schematic



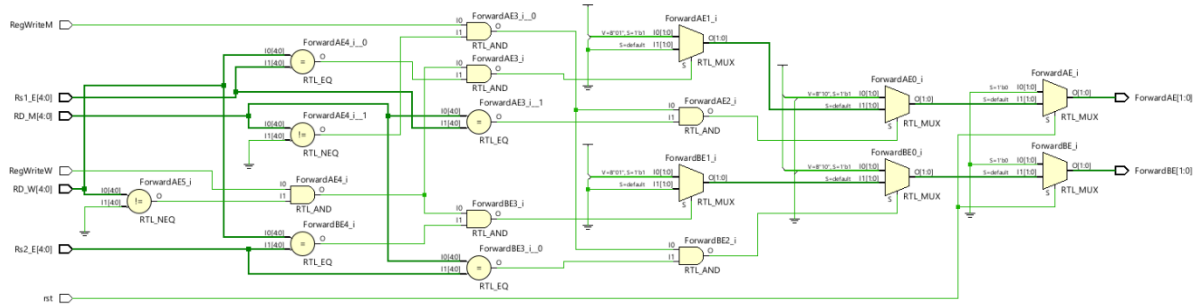
4. Memory Cycle



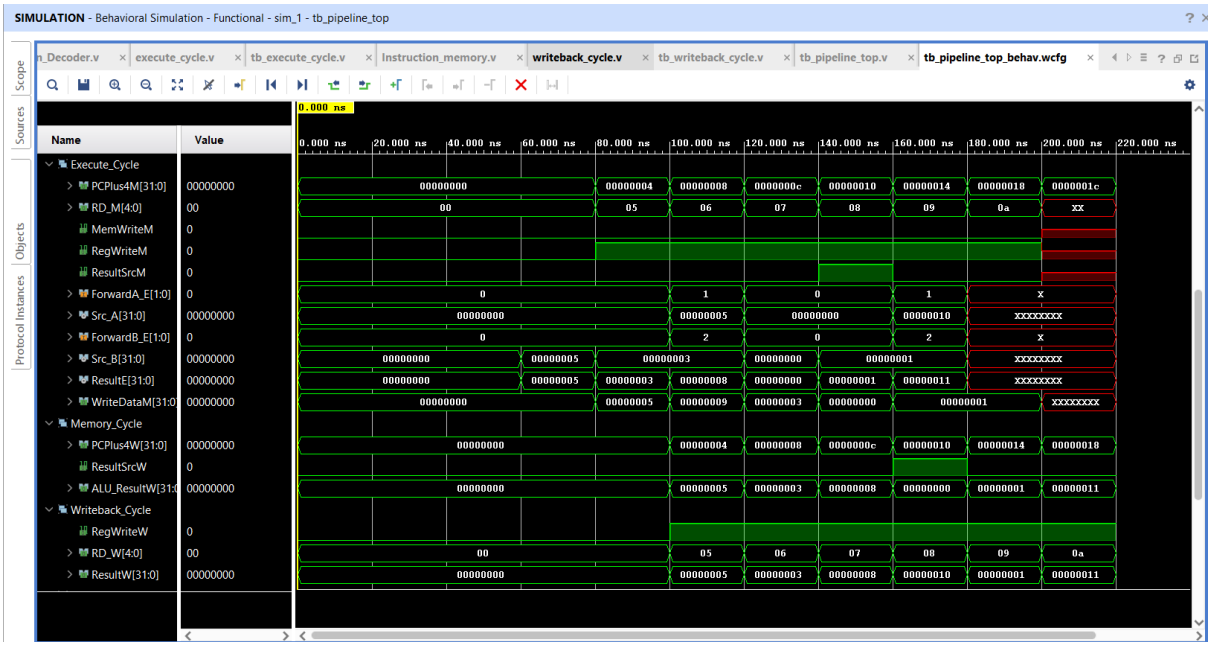
5. Writeback Cycle



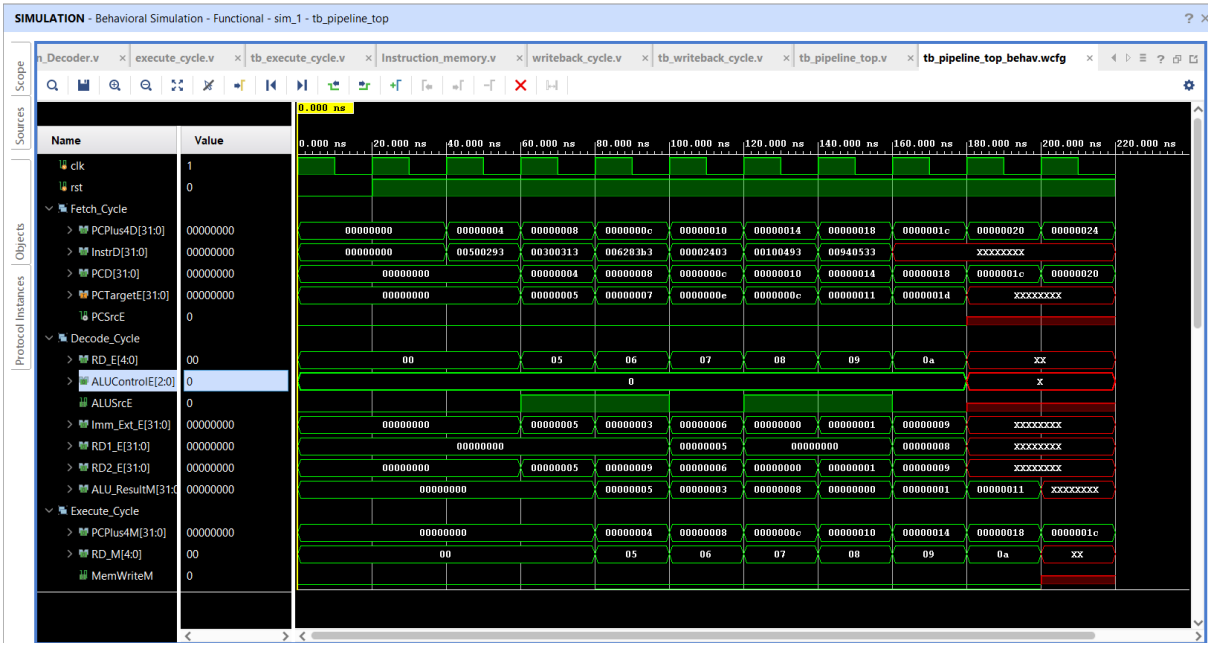
6. Hazard Unit



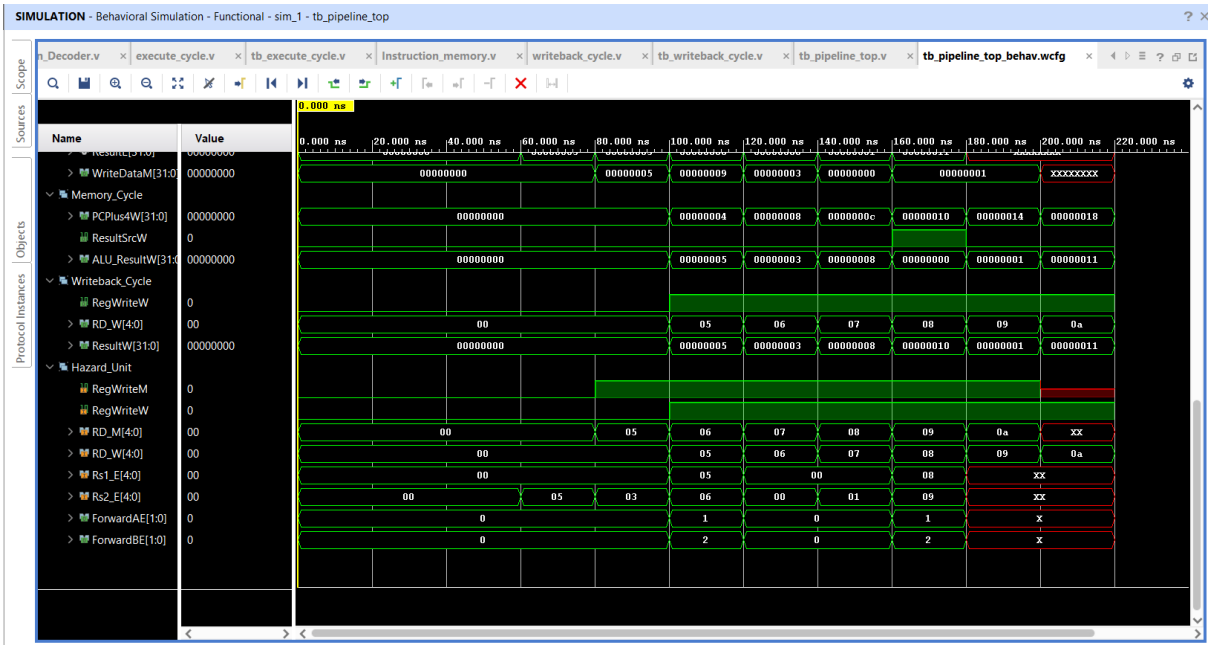
SIMULATION RESULTS



(1)



(2)



(3)

Fig: 5-stage Pipelining

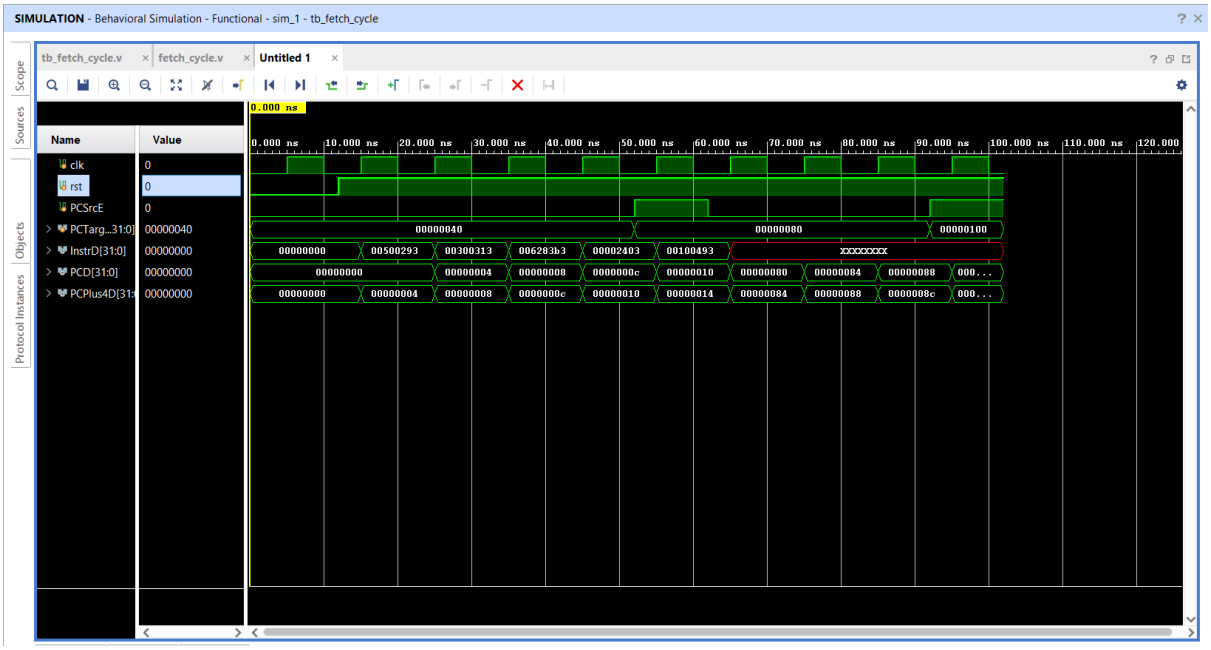


Fig: Fetch_Cycle

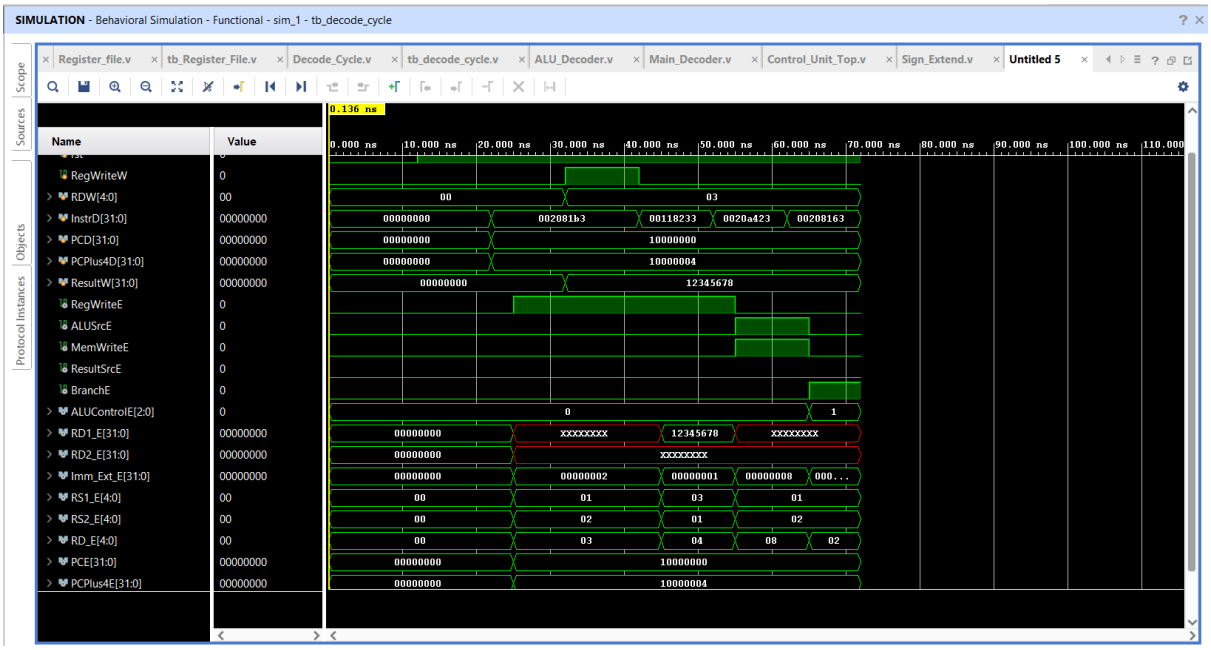


Fig: Decode_Cycle

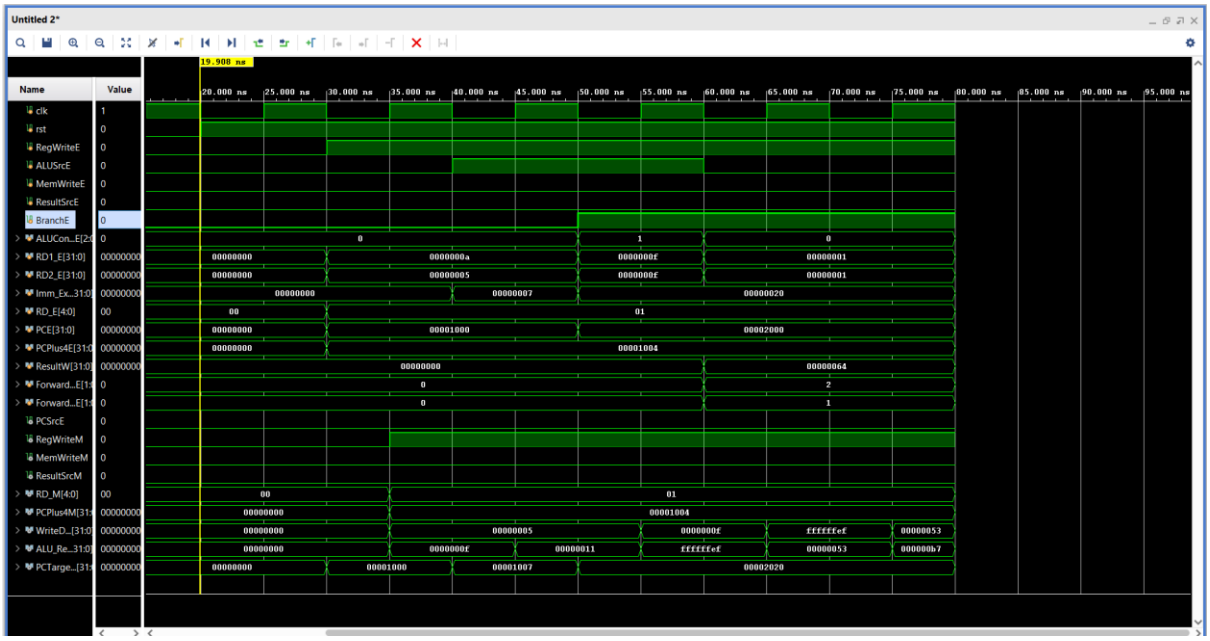


Fig: Execute_Cycle

S

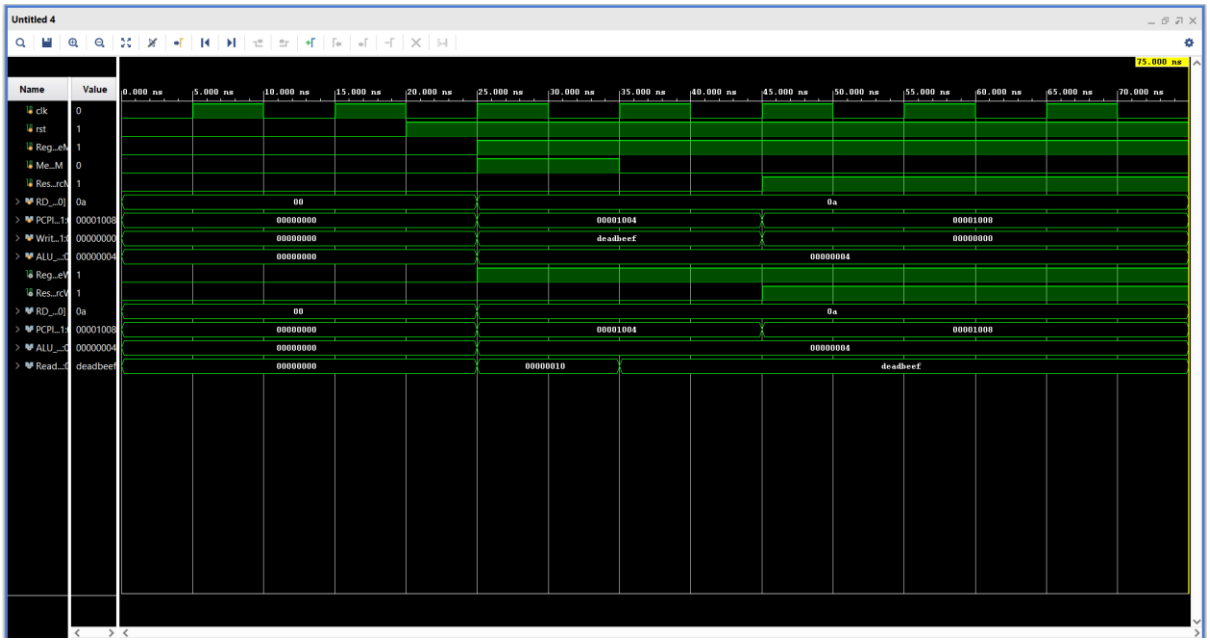


Fig: Memory_Cycle

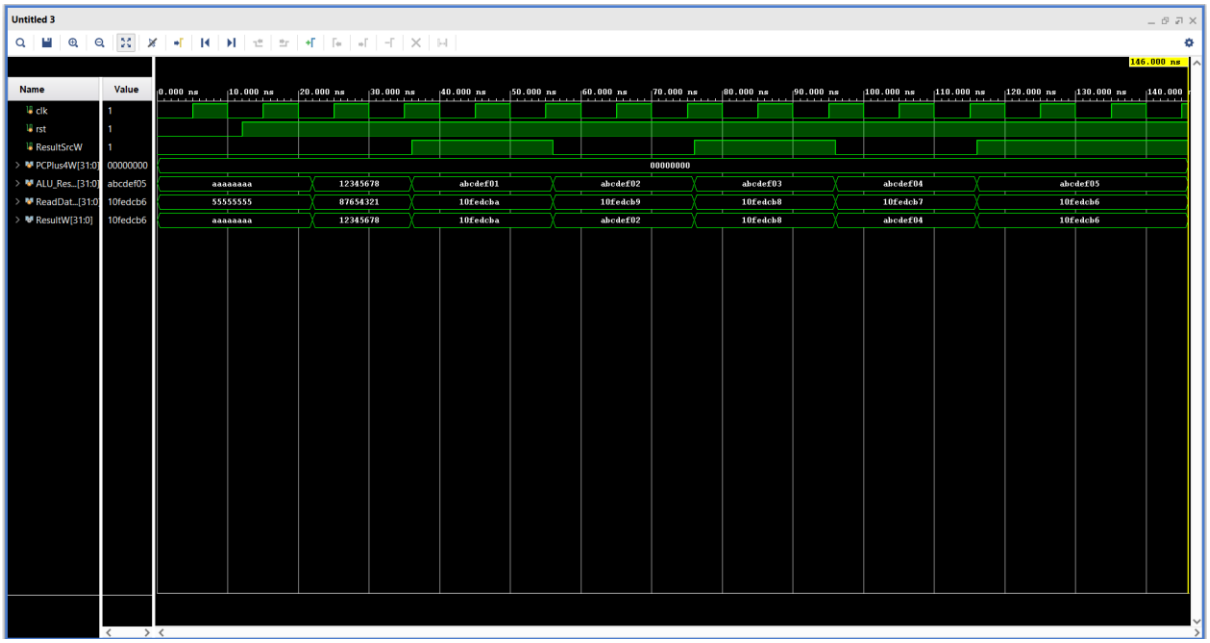
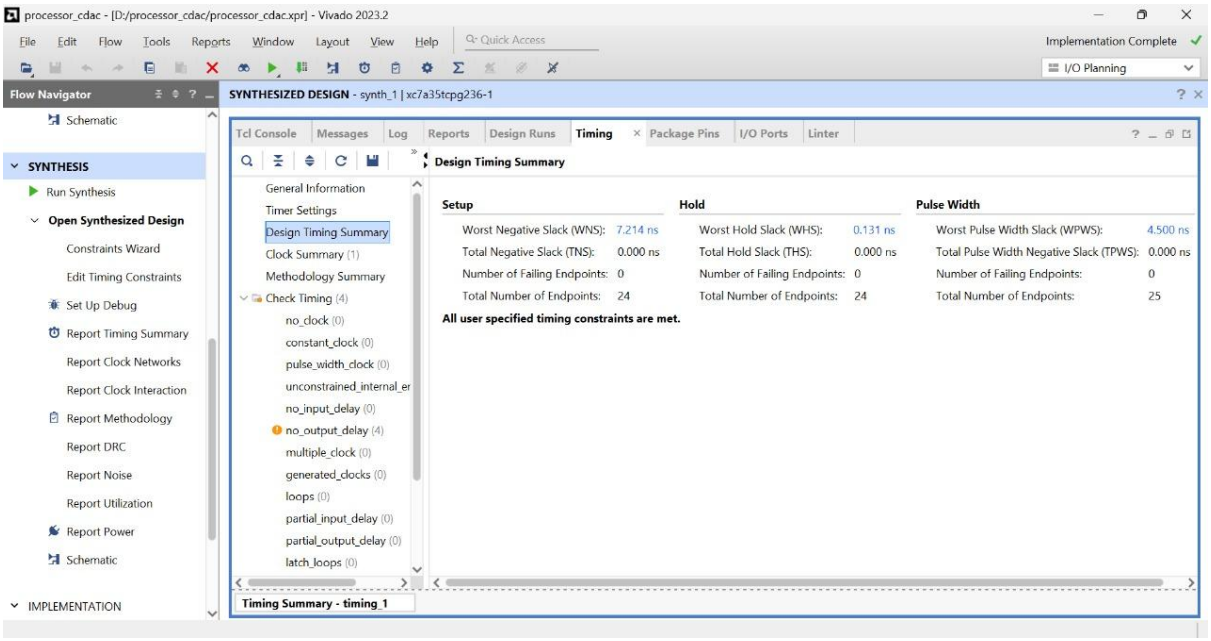


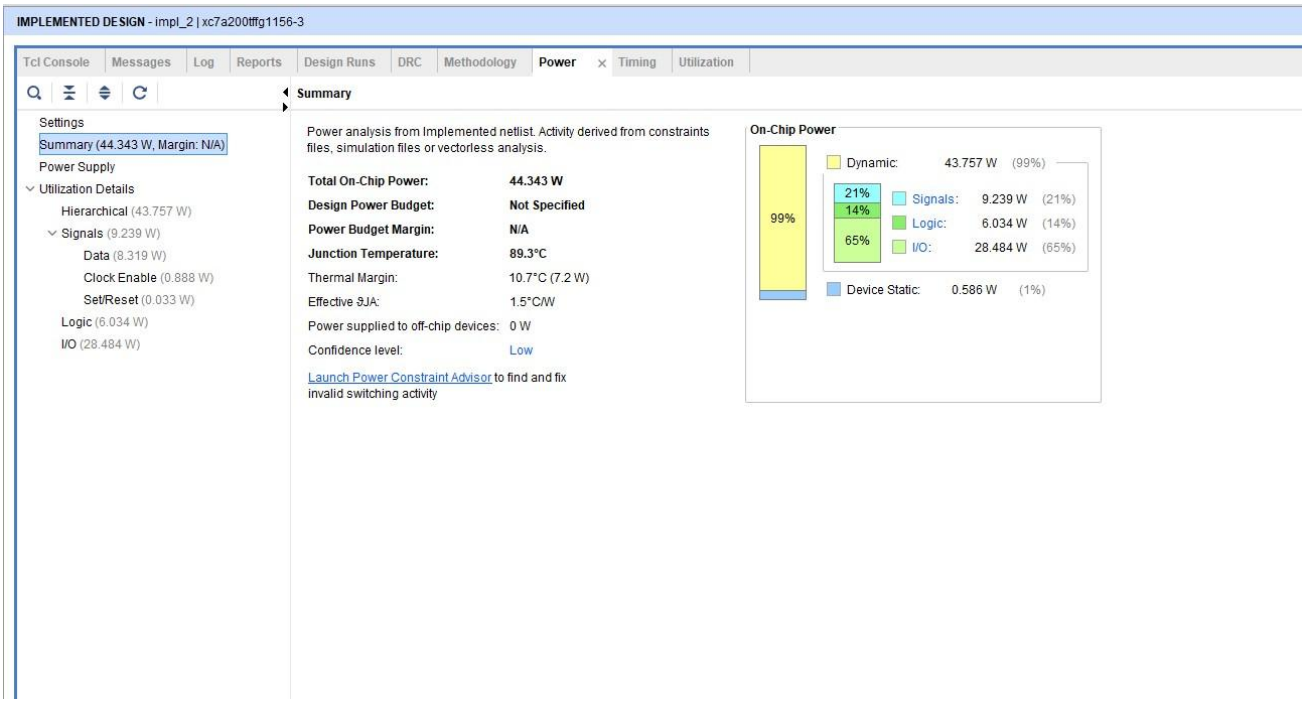
Fig: Writeback_Cycle

SYNTHESIS RESULT

TIMING SUMMARY



POWER



UTILIZATION REPORT

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	559	0	0	20800	2.69
LUT as Logic	383	0	0	20800	1.84
LUT as Memory	176	0	0	9600	1.83
LUT as Distributed RAM	176	0			
LUT as Shift Register	0	0			
Slice Registers	322	0	0	41600	0.77
Register as Flip Flop	322	0	0	41600	0.77
Register as Latch	0	0	0	41600	0.00
F7 Muxes	64	0	0	16300	0.39
F8 Muxes	32	0	0	8150	0.39

CONCLUSION

This project successfully designed and implemented a 32-bit single-core RISC-V processor featuring a classical five-stage pipeline architecture. The modular RTL design includes a comprehensive control unit, hazard detection, and forwarding logic ensuring correct and efficient instruction execution.

Simulation and synthesis results demonstrate functional correctness and meet target timing goals. The design supports a core subset of RV32I instructions facilitating arithmetic, logic, control flow, and load/store operations.

The project establishes a foundation for future enhancements such as multi-core extension, advanced branch prediction, and comprehensive verification leveraging UVM.

FUTURE SCOPE

- Multi-core RISC-V Processor: Extending the design to a multi-core system with cache coherence protocols.
- Advanced Branch Prediction: Implement dynamic and static branch predictors to reduce control hazards.
- Expanded ISA Support: Adding floating-point, multiplication, atomic instructions conforming to RISC-V extensions.
- Cache Integration: Incorporate instruction and data caches improving average memory access latency.
- Power Optimization: Design clock gating and power-saving mechanisms to reduce dynamic power consumption.
- Enhanced Verification: Develop a complete UVM-based verification environment supporting randomized and directed testing with coverage analysis.
- SoC Integration: Integrate with peripherals and AXI-based interconnects for complete system demonstration.

References

1. Patterson, David A., and John L. Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware Software Interface.* Morgan Kaufmann, 2017.
2. Waterman, Andrew, et al. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA Version 2.2.* RISC-V Foundation, 2017.
3. Krste Asanović et al., *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture*, RISC-V Foundation, 2020.
4. Michael D. Ciletti. *Advanced Digital Design with the Verilog HDL*, 3rd Edition, Pearson, 2010.
5. Chris Spear and Greg Tumbush. *SystemVerilog for Verification*, Springer, 2007.
6. Accellera Systems Initiative, *Universal Verification Methodology (UVM) 1.2 Specification*, 2015.
7. Xilinx Vivado Design Suite User Guide.