CS2323: Computer Architecture

Pre-Final Implementation Status Report

Submitted as part of course CS2323 – Project Work (Nov 2025)

Design and FPGA Implementation of a 64-bit Multicycle RISC-V Processor

Submitted By:

Krishna H. Patil (EE24BTECH11036)
Arnav Yadnopavit (EE24BTECH11007)

Course Instructor:

Dr. Rajesh Kedia (CS2323)

December 18, 2025

Contents

1. Current Implementation Status

Since the submission of the Step-2 report, the project has progressed significantly beyond the initial single-cycle design. We now have a fully functional **5-stage pipelined RV64IM processor** with hazard detection, forwarding logic, branch prediction, stalling mechanisms, and an improved memory interface. The following subsections summarize the major advancements.

1.1 Instruction Set Support

The processor now supports the complete **RV64IM** instruction set:

- All RV64I integer instructions (R, I, S, B, U, J formats)
- M-extension: `MUL`, `MULH`, `MULHSU`, `MULHU`
- M-extension: `DIV`, `DIVU`, `REM`, `REMU`

All instructions have been functionally verified in simulation, including multi-cycle DIV behavior.

1.2 Pipeline Implementation (5 Stages)

The processor implements the standard RISC-V 5-stage pipeline:

$$\text{IF} \rightarrow \text{ID} \rightarrow \text{EX} \rightarrow \text{MEM} \rightarrow \text{WB}$$

with correct stage-wise register transfer and control signal propagation.

1.3 Hazard Detection and Data Forwarding

A full hazard management subsystem has been implemented:

- **Data forwarding** from EX/MEM/WB stages into EX stage
- **Load-use hazard stall unit** (1-cycle bubble insertion)
- **DIV staller** that dynamically inserts multiple stalls during long-latency divide operations

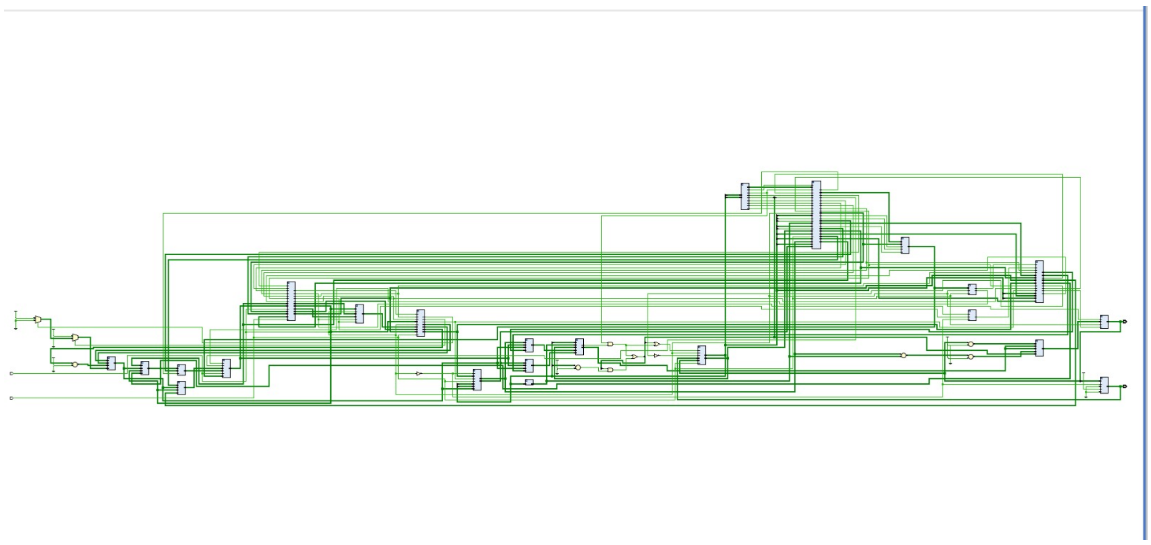


Figure 1: 5-stage pipelined datapath (implemented design).

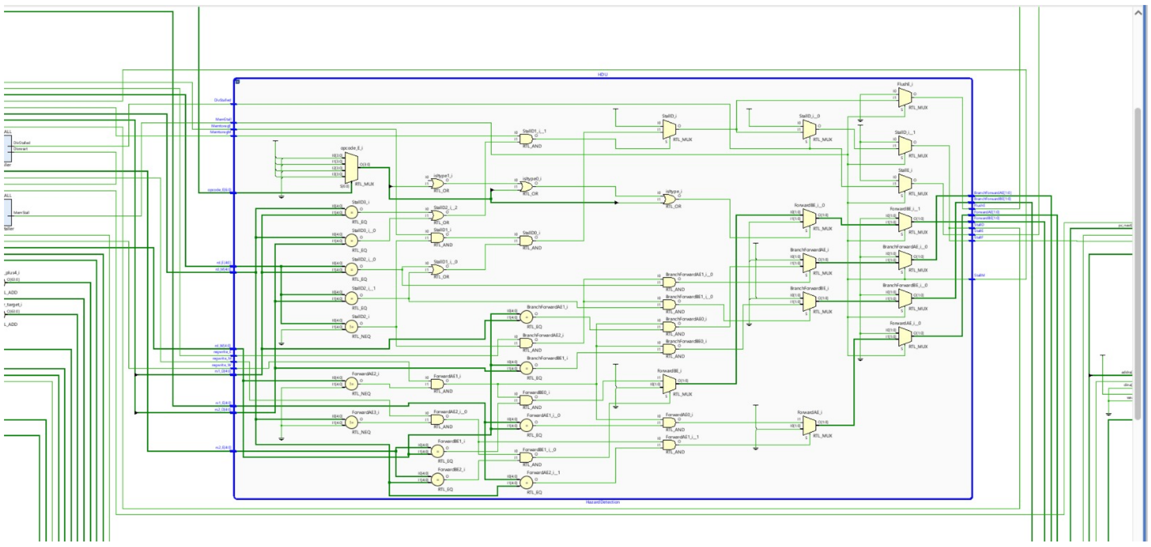


Figure 2: Forwarding unit and hazard detection logic (implemented design).

1.4 Branch Prediction

We implemented **early branch resolution** in the **Decode** stage:

- Reduces branch penalty from 3–4 cycles to 1 cycle
- Uses immediate evaluation and register read in ID stage
- Ensures compatibility with forwarding and load-use hazards

1.5 Memory Subsystem Fix and Redesign

The earlier Step-2 issue where IMEM/DMEM inferred LUTRAM instead of BRAM has been fully resolved.

We redesigned DMEM into three decoupled modules:

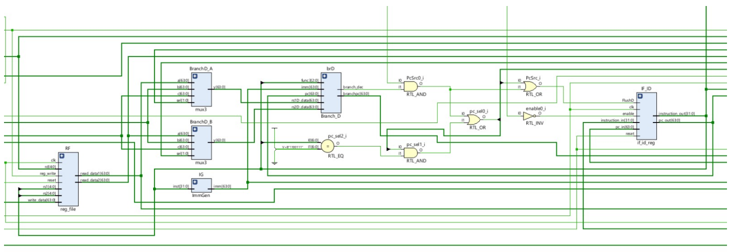


Figure 3: Branch resolution in Decode stage.

- **DMEM Load Unit** – handles data align/extend for loads
- **BRAM-backed Memory Array** – correctly inferred as block RAM
- **DMEM Store Unit** – handles byte/half/word/dword stores

Because block RAM cannot simultaneously support arbitrary timing for read and write in the same cycle, we introduced:

- **Memory stalls**
- **Timing-matched handshakes** between EX and MEM

This fixed all inference issues and made synthesis fully BRAM-based.

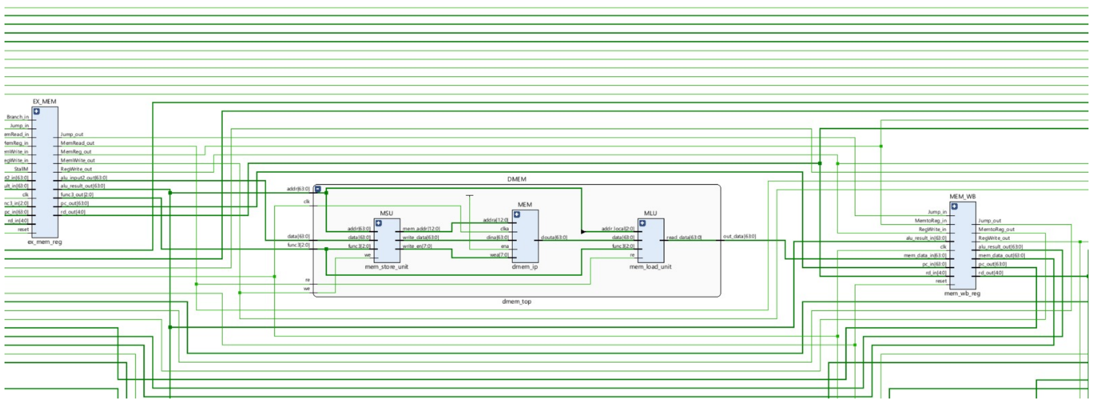


Figure 4: Split DMEM design with Load Unit, BRAM Memory, and Store Unit.

1.6 Integration of Mixed Architectural Models

The final pipeline architecture is based on a combination of:

- Harris and Harris's Model
- Patterson & Hennessy's pipeline diagrams

Both were combined where appropriate to achieve:

- Correct pipeline register behavior
- Timing-safe memory access logic
- Forwarding paths aligned with our CONTROL → ALU → MEM design

1.7 Pipeline Architecture Overview

The pipelined processor implemented in this project is based on a combination of two architectural references: (1) the textbook 5-stage RISC-V pipeline with full hazard handling (Figure 7.61), and (2) the Harris & Harris pipeline diagram (Figure 4.62). Both diagrams were used in complementary ways: the second provided structural clarity while the first provided complete hazard and control signal integration.

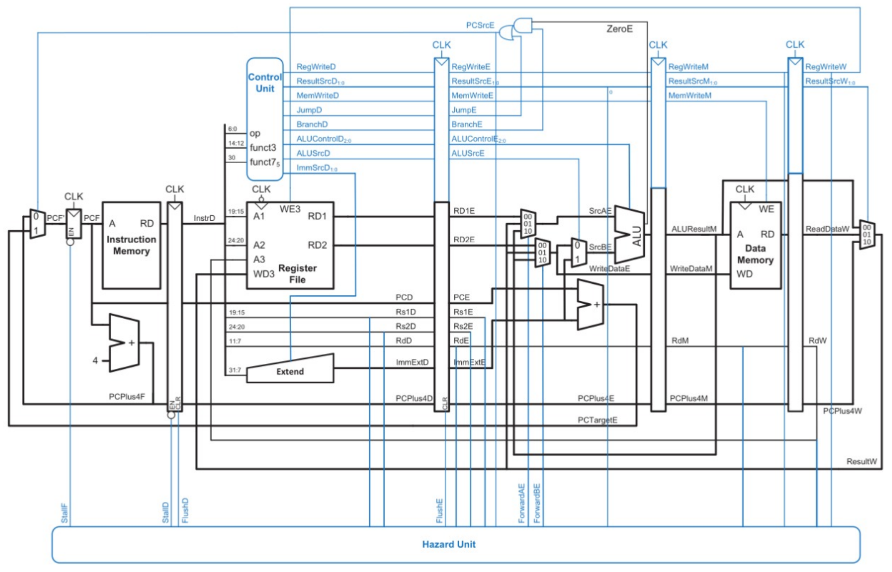


Figure 7.61 Pipelined processor with full hazard handling

Figure 5: Implemented 5-stage pipelined processor (main reference architecture).

The actual implementation follows the standard five pipeline stages:

$$\text{IF} \rightarrow \text{ID} \rightarrow \text{EX} \rightarrow \text{MEM} \rightarrow \text{WB}.$$

Major contributions of each reference diagram:

- The full hazard-handling diagram (Figure 7.61) provided the structural basis for the forwarding paths, stall signals, register pipeline boundaries, and control-signal propagation across stages.
- Patterson's stylized pipeline (Figure 4.62) provided the concept of early hazard outputs, a compact control-flow structure, and simplified muxing patterns that shaped our control-unit wiring.

1.8 Branch Resolution Moved to Decode Stage

Branch handling is a major improvement over the textbook EX-stage branch evaluation. To reduce branch penalty, we partially adopted the design from Figure 4.62 and modified the branch datapath so that the branch comparison occurs in the **Decode (ID) stage** rather than EX. For this purpose:

- A small comparator block was added in ID stage.
- Immediate generation, register file outputs, and branch condition evaluation are all available early.
- Branch target address calculation (PC + immediate) occurs in parallel in ID.
- A flush signal is generated immediately upon misprediction, giving a 1-cycle penalty.

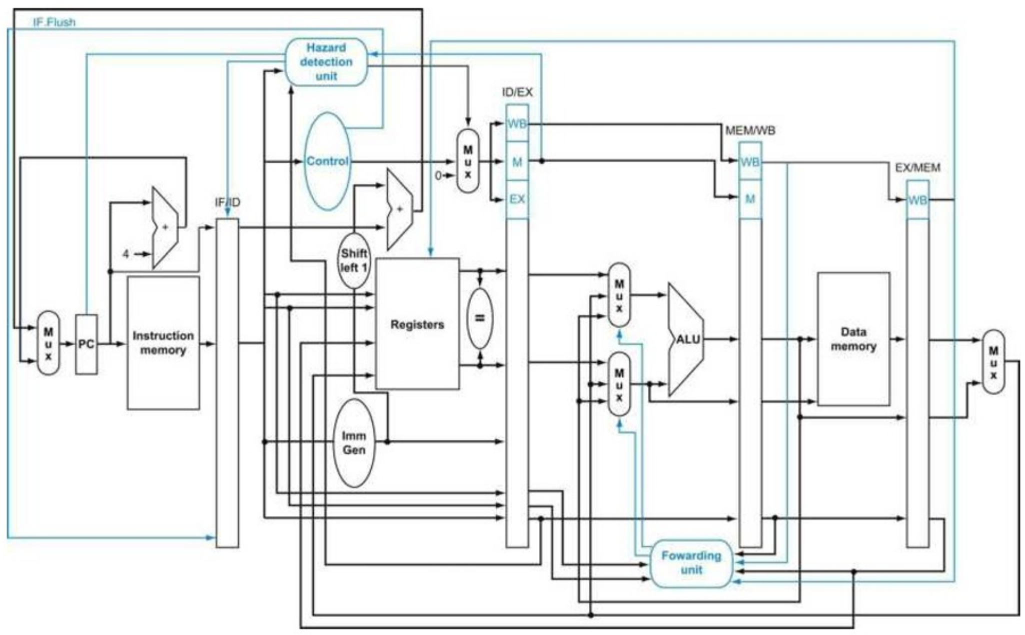


FIGURE 4.62 The final datapath and control for this chapter.
Note that this is a stylized figure rather than a detailed datapath, so it's missing the ALUsrc Mux from Figure 4.55 and the multiplexor controls from Figure 4.49.

Figure 6: Branch datapath used to move branch resolution to the Decode stage.

This architectural change significantly improves pipeline efficiency, especially for branch-heavy workloads.

1.9 Hazard Unit and Data Forwarding

A complete hazard detection unit has been implemented. It handles:

- **Load-use hazards**: a 1-cycle stall is inserted when EX-stage instruction depends on a MEM-stage load.
- **Branch hazards**: automatic detection and flush of the IF/ID pipeline register.
- **DIV hazards**: dedicated multi-cycle stalling (stall signal holds PC and IF/ID until division completes).

The forwarding unit forwards:

- EX/MEM → EX stage
- MEM/WB → EX stage

The pipeline supports all RAW hazards without compiler scheduling.

1.10 Division Staller and Timing Alignment

Since the division operations (`DIV`, `DIVU`, `REM`, `REMU`) take multiple cycles, we implemented:

- a **division staller** to hold the pipeline during multi-cycle DIV execution,
- handshake signals between EX and MEM to prevent overwriting ALU results,
- cycle-accurate alignment so that DIV results write back correctly through MEM and WB.

This ensures architectural correctness without corrupting pipeline state.

1.11 Memory Subsystem: Three-Part DMEM Redesign

To resolve earlier synthesis and BRAM inference problems, DMEM was redesigned as three components:

- **DMEM Load Unit** – handles sign/zero extension, alignment.
- **BRAM-backed memory array** – now always inferred as block RAM.
- **DMEM Store Unit** – handles byte/half/word/dword stores.

Stalls are inserted during memory operations to match BRAM timing requirements.

The final pipelined memory system is now timing-clean and resource-optimized for FPGA implementation.

1.12 Summary

The current pipeline integrates:

- A full RV64IM instruction set,
- 5-stage pipelining with forwarding and hazard detection,
- Branch resolution moved to ID,
- Full stall management including multi-cycle DIV stalls,
- A BRAM-accurate memory system with load/store units.

This constitutes a complete and functional pipelined RISC-V implementation suitable for FPGA demonstration.

1.13 Waveform and Testbench Verification

All major modules and pipeline paths have been verified via:

- Individual module testbenches (ALU, RegFile, ImmGen, CU, PC, IMEM/DMEM)
- Full pipeline-level testbenches
- Multi-instruction programs (ADD chains, load/store sequences, branch loops)
- Full RV64IM test sequences

1.14 Testbench Generation Note

Initial versions of the testbenches were generated with the assistance of ChatGPT and later refined, corrected, and extended by us according to module interfaces and debugging needs.

.

2. Current Implementation Status

2.1 Single-Cycle Processor (Completed Earlier)

The initial single-cycle RV64I processor was fully completed during Step-2 of the project. This version included:

- Full datapath for RV64I integer instructions
- Working control unit, ALU, immediate generator, register file
- Separate IMEM and DMEM with `$readmemh` initialization

This stage served as the foundation for all subsequent enhancements.

2.2 Transition Away from Multicycle Design

Although an intermediate multicycle design was part of the initial plan, we decided to skip the multicycle stage entirely and directly implement a **5-stage pipelined processor**. This decision was motivated by:

- The larger benefit of focusing on a complete pipelined RV64IM core
- Time saved by reusing the single-cycle datapath components
- Better alignment with modern RISC-V processor architecture

2.3 Pipelined Processor (RV64IM) — Fully Implemented

We now have a complete, synthesizable **5-stage pipelined RV64IM processor** with:

- Instruction Fetch (IF)
- Instruction Decode (ID)
- Execute (EX)
- Memory (MEM)
- Writeback (WB)

Major pipeline features:

- **Full forwarding** (EX/MEM to EX, MEM/WB to EX)
- **Load-use hazard stall** (1-cycle bubble)
- **Branch resolution moved to Decode stage**
- **Automatic branch flush logic**
- **Pipeline register bundles between all stages**

2.4 M Extension (Multiply/Divide)

All RV64IM multiply and divide instructions were fully implemented:

- `MUL`, `MULH`, `MULHU`, `MULHSU`
- `DIV`, `DIVU`, `REM`, `REMU`

The multiply operations complete in a single cycle using DSP blocks. The division operations are implemented as a multi-cycle iterative algorithm, managed through a:

- **DIV staller** that holds IF/ID and ID/EX until the divider finishes
- handshake between EX and MEM to maintain result alignment

2.5 FPU Integration (Deferred)

Although originally planned, the RV64F floating-point extension was **intentionally deferred**. This decision was based on the following considerations:

- The FPU requires additional floating-point register paths and specialised control logic.
- Integrating add/sub/mul floating-point units would require modifying the EX and MEM stages and reworking hazard detection.
- The complexity multi-cycle FPU behavior would not be feasible within the remaining course deadline.

Given these constraints, the focus was shifted toward completing a fully functional and stable RV64IM pipelined core and preparing it for FPGA implementation.

2.6 Memory System and Cache (Deferred Cache, Improved DMEM)

Cache implementation (L1/L2) was also **deferred**. However, the memory system was significantly enhanced:

- DMEM split into three parts:
  - Load Unit
  - BRAM-backed memory
  - Store Unit
- Timing-correct BRAM behavior with stall signals
- IMEM and DMEM now reliably inferred as Block RAM

This ensures correct memory access timing on the FPGA.

2.7 Synthesis and FPGA Testing

The full RV64IM pipelined design successfully synthesizes in Vivado with:

- Correct BRAM inference for IMEM and DMEM
- DSP usage for multiplication
- LUT usage within Arty A7 limits

Current FPGA status:

- Bitstream generation successful

- IMEM initialized using `$readmemh` (confirmed via simulation and Vivado logs)
- LED/UART integration planned for final program output
- On-board testing underway with small test programs

The focus now is completing hardware validation and preparing the demonstration sequence.

2.7.1 Testbench Automation Improvements

As part of the pre-final development phase, we also began improving the structure of our testbenches. Earlier, each new RISC-V program required manual modifications to the testbench, including changes to file paths, initial conditions, and internal signals. This made iterative testing slower and more error-prone.

To address this, we are redesigning the testbench framework so that:

- the same testbench can be reused for all RISC-V programs,
- only the instruction memory hex file needs to be changed for each new test,
- the testbench automatically loads the hex file using `$readmemh`,
- no internal modifications to the testbench are required when switching programs.

This improvement greatly simplifies simulation workflow and makes it easier to validate more complex programs, especially before running them on the FPGA. It also supports rapid debugging of pipeline behavior, hazard response, and multi-instruction sequences.

3. Challenges Faced

The transition from a single-cycle CPU to a complete RV64IM pipelined processor introduced several non-trivial challenges. Many of these required deeper architectural understanding, careful debugging, and multiple redesign iterations.

3.1 Pipeline Implementation Complexity

Designing the 5-stage pipeline was significantly more difficult than expected. Key challenges included:

- determining the exact set of signals that needed to be carried across each pipeline stage,
- calculating the widths of IF/ID, ID/EX, EX/MEM, and MEM/WB pipeline registers,

- ensuring that no necessary control or data signal was accidentally omitted,
- identifying which signals were redundant and removing them to avoid resource waste,
- maintaining timing consistency across stage boundaries.

This phase required repeated refinement of datapath diagrams and debugging through waveform inspection.

3.2 Interconnection and Signal Propagation Issues

A major portion of the debugging time was spent ensuring correct wiring of signals between stages. Typical problems included:

- incorrect or inconsistent signal naming,
- mismatched bus widths across modules,
- late discovery that some signals needed to be forwarded to later stages,
- duplicated or missing control signals that caused subtle pipeline bugs.

Many of these issues became apparent only during integrated simulation and required multiple iterations of correction.

3.3 Integrating M-Extension Instructions

The integration of the M-instructions was particularly challenging. While `MUL` instructions were straightforward, the division instructions introduced several complexities:

- handling signed vs. unsigned division and remainder instructions,
- designing a multi-cycle iterative divider compatible with the pipeline,
- managing pipeline stalls during division to avoid corrupting subsequent instructions,
- aligning divider output timing with MEM/WB stages,
- ensuring correct flag generation and operand selection.

The M-extension integration required substantial redesign of the EX-stage logic and hazard-handling unit.

3.4 Timing Mismatches and Unexpected Pipeline Behavior

Certain bugs arose from incorrect assumptions about timing:

- control signals reaching a stage earlier/later than data signals,
- pipeline registers capturing values before some modules had updated their outputs,
- signed/unsigned interpretation mismatches causing arithmetic failure,
- off-by-one cycle errors that produced subtle writeback bugs.

Some of these issues were the result of minor mistakes (our own "dumbness"), but resolving them provided deeper insight into cycle-by-cycle hardware timing.

3.5 Memory Timing and BRAM Behavior

The need to restructure DMEM into load/store/BRAM units emerged because:

- BRAM has fixed read/write latency that doesn't match textbook single-cycle memory models,
- naive memory integration caused timing mismatch between EX and MEM results,
- unaligned loads/stores required careful data alignment logic.

This required significant re-engineering of the MEM stage and addition of stalls.

4. Early Results

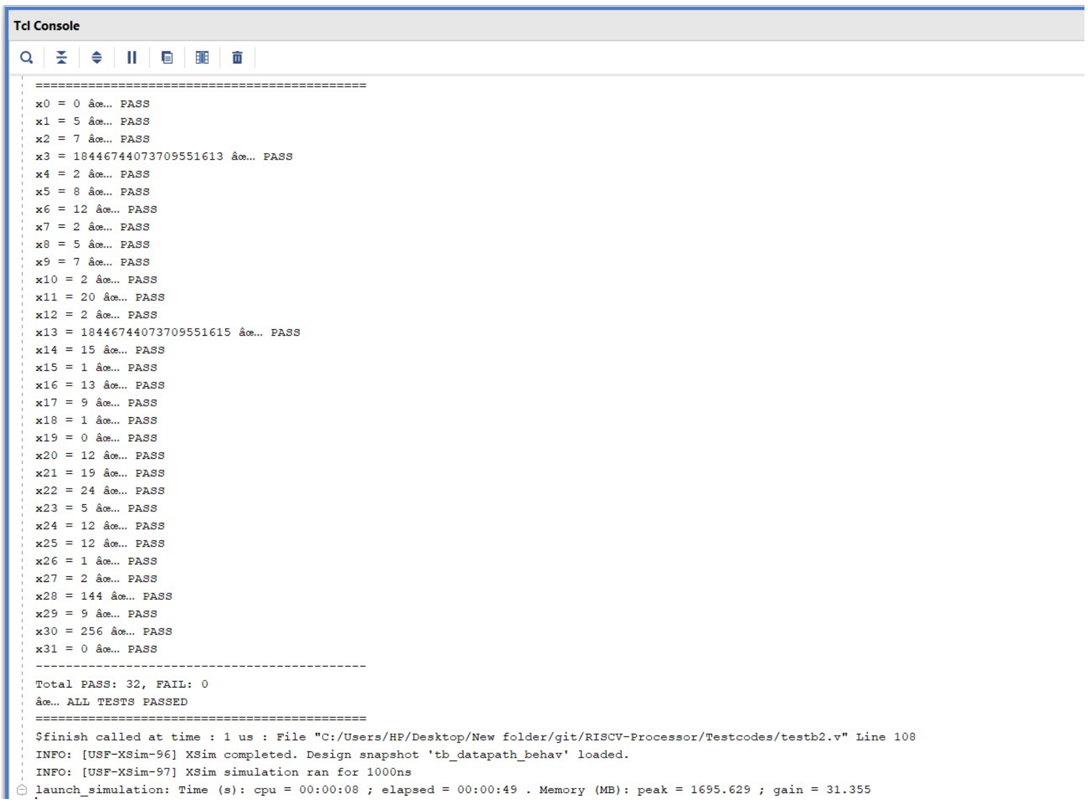
Extensive simulation-based testing was performed on the implemented RV64IM pipelined processor using XSIM. The objective was to validate correctness of the ISA behavior, pipeline control logic, hazard management, memory subsystem, and the M-extension. All programs were assembled into hex format and loaded into IMEM using `$readmemh`. Register-file contents were automatically checked after execution.

4.1 RV64I Functional Tests

The first test sequence evaluated the core RV64I instruction set, covering:

- arithmetic and logical operations,
- shift operations,
- immediate instructions,
- register initialization and sanity checks.

All 32 registers matched their expected values, confirming full correctness of the base integer datapath.



```
============================================
x0 = 0 âœ… PASS
x1 = 5 âœ… PASS
x2 = 7 âœ… PASS
x3 = 18446744073709551613 âœ… PASS
x4 = 2 âœ… PASS
x5 = 8 âœ… PASS
x6 = 12 âœ… PASS
x7 = 2 âœ… PASS
x8 = 5 âœ… PASS
x9 = 7 âœ… PASS
x10 = 2 âœ… PASS
x11 = 20 âœ… PASS
x12 = 2 âœ… PASS
x13 = 18446744073709551615 âœ… PASS
x14 = 15 âœ… PASS
x15 = 1 âœ… PASS
x16 = 13 âœ… PASS
x17 = 9 âœ… PASS
x18 = 1 âœ… PASS
x19 = 0 âœ… PASS
x20 = 12 âœ… PASS
x21 = 19 âœ… PASS
x22 = 24 âœ… PASS
x23 = 5 âœ… PASS
x24 = 12 âœ… PASS
x25 = 12 âœ… PASS
x26 = 1 âœ… PASS
x27 = 2 âœ… PASS
x28 = 144 âœ… PASS
x29 = 9 âœ… PASS
x30 = 256 âœ… PASS
x31 = 0 âœ… PASS
--------------------------------------------
Total PASS: 32, FAIL: 0
âœ… ALL TESTS PASSED
============================================
$finish called at time : 1 us : File "C:/Users/HP/Desktop/New folder/git/RISCV-Processor/Testcodes/testb2.v" Line 108
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_datapath_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:08 ; elapsed = 00:00:49 . Memory (MB): peak = 1695.629 ; gain = 31.355
```

Figure 7: RV64I test output – All registers PASS (32/32).

4.2 Forwarding and Load-Use Hazard Tests

A dedicated test program stressed the forwarding unit and hazard detection logic. The test included:

- RAW hazards generated by back-to-back dependent instructions,
- ALU→ALU forwarding,
- MEM→ALU forwarding,
- load-use hazards requiring a one-cycle stall,
- dependent additions forming a chain.

The pipeline correctly inserted stalls for load-use hazards and forwarded data from EX/MEM and MEM/WB as intended. No mismatches were detected.

4.3 Branch and Control Hazard Tests

Branch prediction was implemented by resolving branches in the Decode (ID) stage. This test validated:

- correct early branch decision,

- IF/ID flush on misprediction,
- correct behavior of taken and not-taken branches,
- JAL/JALR link register correctness.

All branch paths behaved as expected, demonstrating that the Decode-stage branch unit and hazard logic function correctly.

4.4 Memory Subsystem Tests

A BRAM-backed DMEM implementation was tested using known load/store patterns. The results verified:

- correct sign/zero-extension for loads,
- correct byte/half/word/dword store handling,
- correct BRAM timing using the stall mechanism,
- proper interaction with EX/MEM pipeline registers.

4.5 M Extension (Multiply/Divide) Tests

A comprehensive test validated all eight M-extension instructions:

- **Multiply**: `mul`, `mulh`, `mulhsu`, `mulhu`
- **Divide**: `div`, `divu`
- **Remainder**: `rem`, `remu`

Both signed and unsigned corner cases were tested. The multi-cycle divider and DIV staller worked correctly, inserting stalls until the division unit completed and correctly routing the result to the MEM/WB stage.

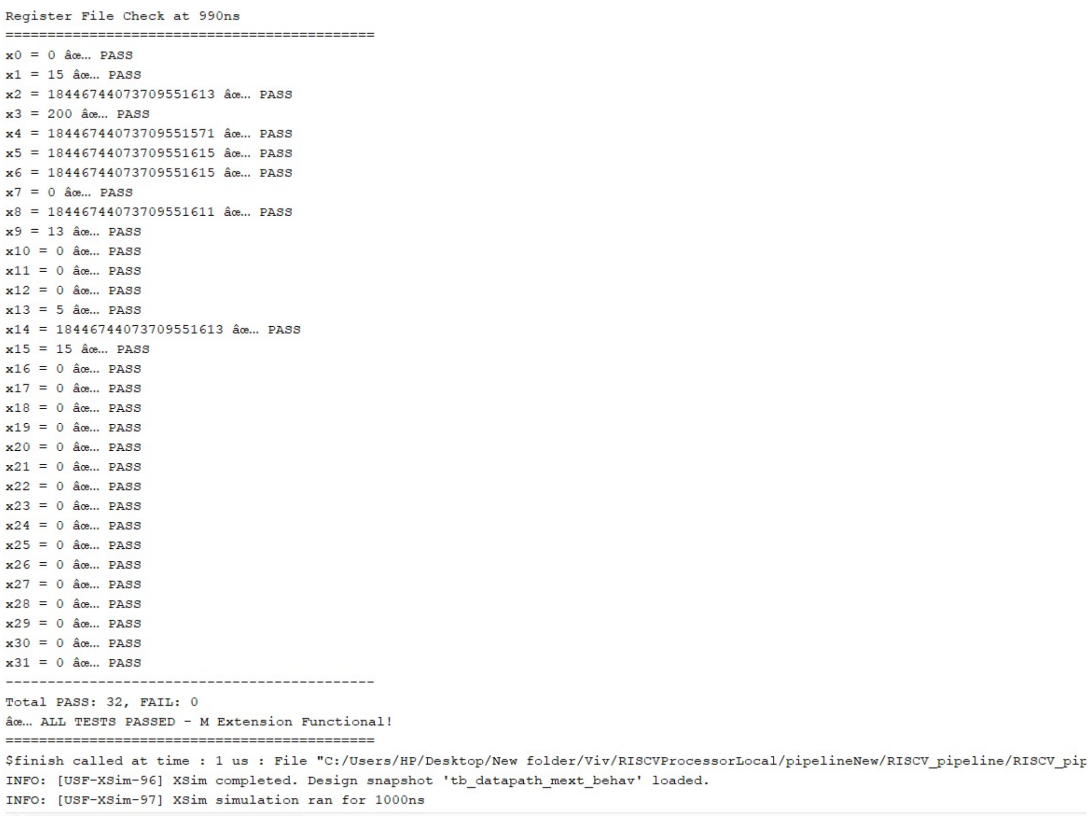
All registers matched the expected values.

4.6 Vivado Synthesis Results

The final RV64IM pipelined CPU was synthesized for the Arty A7-100T. Resource utilization is as follows:

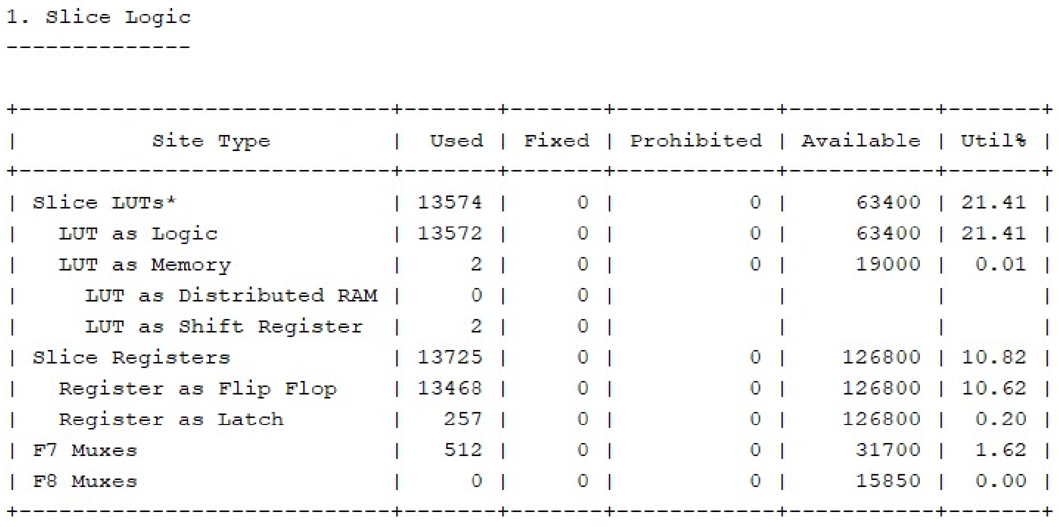
- **Slice LUTs**: 13,574 (21.41%)
- **Slice Registers**: 13,725 (10.82%)
- **BRAM Tiles**: 32 (23.7%)
- **DSP48E1 blocks**: 54 (22.5%)

This confirms that the design fits comfortably in the FPGA with space for UART and additional I/O.



```
Register File Check at 990ns
===========================================
x0 = 0 âœ… PASS
x1 = 15 âœ… PASS
x2 = 18446744073709551613 âœ… PASS
x3 = 200 âœ… PASS
x4 = 18446744073709551571 âœ… PASS
x5 = 18446744073709551615 âœ… PASS
x6 = 18446744073709551615 âœ… PASS
x7 = 0 âœ… PASS
x8 = 18446744073709551611 âœ… PASS
x9 = 13 âœ… PASS
x10 = 0 âœ… PASS
x11 = 0 âœ… PASS
x12 = 0 âœ… PASS
x13 = 5 âœ… PASS
x14 = 18446744073709551613 âœ… PASS
x15 = 15 âœ… PASS
x16 = 0 âœ… PASS
x17 = 0 âœ… PASS
x18 = 0 âœ… PASS
x19 = 0 âœ… PASS
x20 = 0 âœ… PASS
x21 = 0 âœ… PASS
x22 = 0 âœ… PASS
x23 = 0 âœ… PASS
x24 = 0 âœ… PASS
x25 = 0 âœ… PASS
x26 = 0 âœ… PASS
x27 = 0 âœ… PASS
x28 = 0 âœ… PASS
x29 = 0 âœ… PASS
x30 = 0 âœ… PASS
x31 = 0 âœ… PASS
-------------------------------------------
Total PASS: 32, FAIL: 0
âœ… ALL TESTS PASSED - M Extension Functional!
===========================================
$finish called at time : 1 us : File "C:/Users/HP/Desktop/New folder/Viv/RISCVProcessorLocal/pipelineNew/RISCV_pipeline/RISCV_pip
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_datapath_mext_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

Figure 8: M-extension results – MUL/DIV/REM operations fully correct (32/32 PASS).



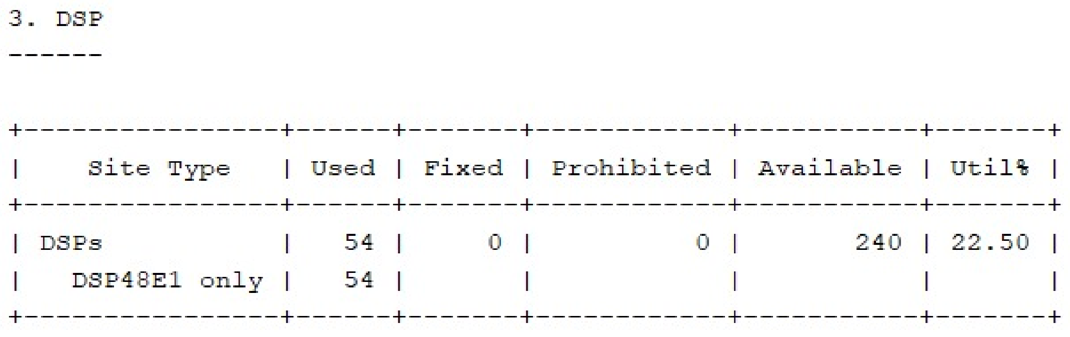
```
1. Slice Logic
--------------

+----------------------------+-------+-------+------------+-----------+-------+
|          Site Type         |  Used | Fixed | Prohibited | Available | Util% |
+----------------------------+-------+-------+------------+-----------+-------+
| Slice LUTs*                | 13574 |     0 |          0 |     63400 | 21.41 |
|   LUT as Logic             | 13572 |     0 |          0 |     63400 | 21.41 |
|   LUT as Memory            |     2 |     0 |          0 |     19000 |  0.01 |
|     LUT as Distributed RAM |     0 |     0 |            |           |       |
|     LUT as Shift Register  |     2 |     0 |            |           |       |
| Slice Registers            | 13725 |     0 |          0 |    126800 | 10.82 |
|   Register as Flip Flop    | 13468 |     0 |          0 |    126800 | 10.62 |
|   Register as Latch        |   257 |     0 |          0 |    126800 |  0.20 |
| F7 Muxes                   |   512 |     0 |          0 |     31700 |  1.62 |
| F8 Muxes                   |     0 |     0 |          0 |     15850 |  0.00 |
+----------------------------+-------+-------+------------+-----------+-------+
```

Figure 9: Vivado synthesis report – resource utilization summary.

4.7 Summary of Early Results

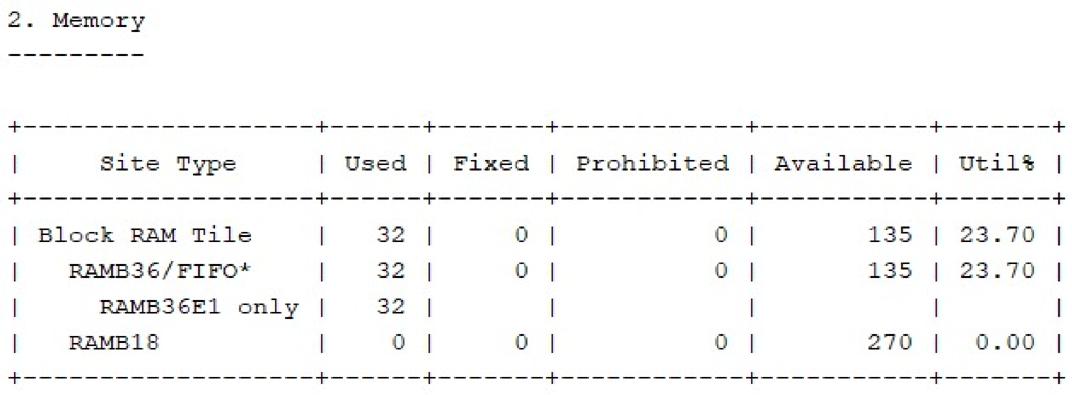
Table **??** summarizes all tested features.

.



```
3. DSP
------

+----------------+------+-------+------------+-----------+-------+
|    Site Type   | Used | Fixed | Prohibited | Available | Util% |
+----------------+------+-------+------------+-----------+-------+
| DSPs           |   54 |     0 |          0 |       240 | 22.50 |
|   DSP48E1 only |   54 |       |            |           |       |
+----------------+------+-------+------------+-----------+-------+
```

Figure 10: Vivado synthesis report – resource utilization summary.



```
2. Memory
---------

+-------------------+------+-------+------------+-----------+-------+
|     Site Type     | Used | Fixed | Prohibited | Available | Util% |
+-------------------+------+-------+------------+-----------+-------+
| Block RAM Tile    |   32 |     0 |          0 |       135 | 23.70 |
|   RAMB36/FIFO*    |   32 |     0 |          0 |       135 | 23.70 |
|     RAMB36E1 only |   32 |       |            |           |       |
|   RAMB18          |    0 |     0 |          0 |       270 |  0.00 |
+-------------------+------+-------+------------+-----------+-------+
```

Figure 11: Vivado synthesis report – resource utilization summary.

Test Category	**Status**
RV64I Functional Instructions	PASS
Forwarding Unit (RAW hazards)	PASS
Load-Use Hazard Detection	PASS
Decode-Stage Branch Prediction + Flush	PASS
Load/Store + BRAM Timing	PASS
RV64M (Multiply/Divide/Rem) Extension	PASS
Vivado Synthesis (Fit + DRC + Timing)	PASS

Table 1: Summary of simulation and synthesis results.