ALU (Arithmetic Logic Unit) Mini Project Report

Mirafra Technologies
Verilog HDL Mini Project
2025

Submitted by:
Name: Kishan Rupesh Revankar
EID: 6082
Date: 06/06/2025

1. INTRODUCTION

The Arithmetic Logic Unit (ALU) is a core component of the central processing unit (CPU) in digital systems. It performs arithmetic and logical operations, forming the heart of any computing device. This project involves the design and implementation of a configurable ALU using Verilog HDL. The primary focus is on synthesizability, multi-operation support, and robust testbench-driven verification.

The ALU operates as a part of the CPU and is responsible for executing mathematical calculations and logical decisions. This project simulates a realistic ALU design that can be synthesized and deployed onto hardware, like FPGAs. The code is written to comply with **pipelined**, **multicycle**, and **synchronous** design principles using clocked behavior.

This project emphasizes **practical understanding**, clean code structure, modular Verilog design, and testbench development. In addition to simulating the ALU behavior, a **robust testbench** was developed to verify functional correctness using different inputs and edge cases.

2. OBJECTIVE

The key objectives of the ALU design project are as follows:

- **To design and implement a parameterized ALU module using Verilog** that supports both arithmetic and logical operations on configurable operand width (WIDTH parameter).
- **To ensure the ALU is synthesizable**, so it can be deployed on real hardware such as FPGAs.
- **To support both single-cycle and multi-cycle operations**, particularly ensuring that multiplication commands (CMD 9 and 10) are delayed and output their result on the third clock cycle.
- **To implement a valid control interface** using inputs like MODE, CMD, CIN, INP_VALID, and CE to guide the computation flow.
- **To design flag generation logic** for carry, overflow, error detection, and comparison results.
- **To develop an automated testbench** that uses pre-written stimulus vectors for verification and creates a final pass/fail report based on expected vs actual outputs.
- **To provide modularity and reusability** so the ALU can be easily modified for more complex systems or larger data widths.
- **To gain hands-on experience with Verilog-based digital design**, including clocked logic, reset conditions, pipelining principles, and condition-based processing.

The project is structured to replicate real industry ALU design, verification, and documentation pipelines. Each feature added is also meant to prepare students for both interviews and hands-on hardware work.

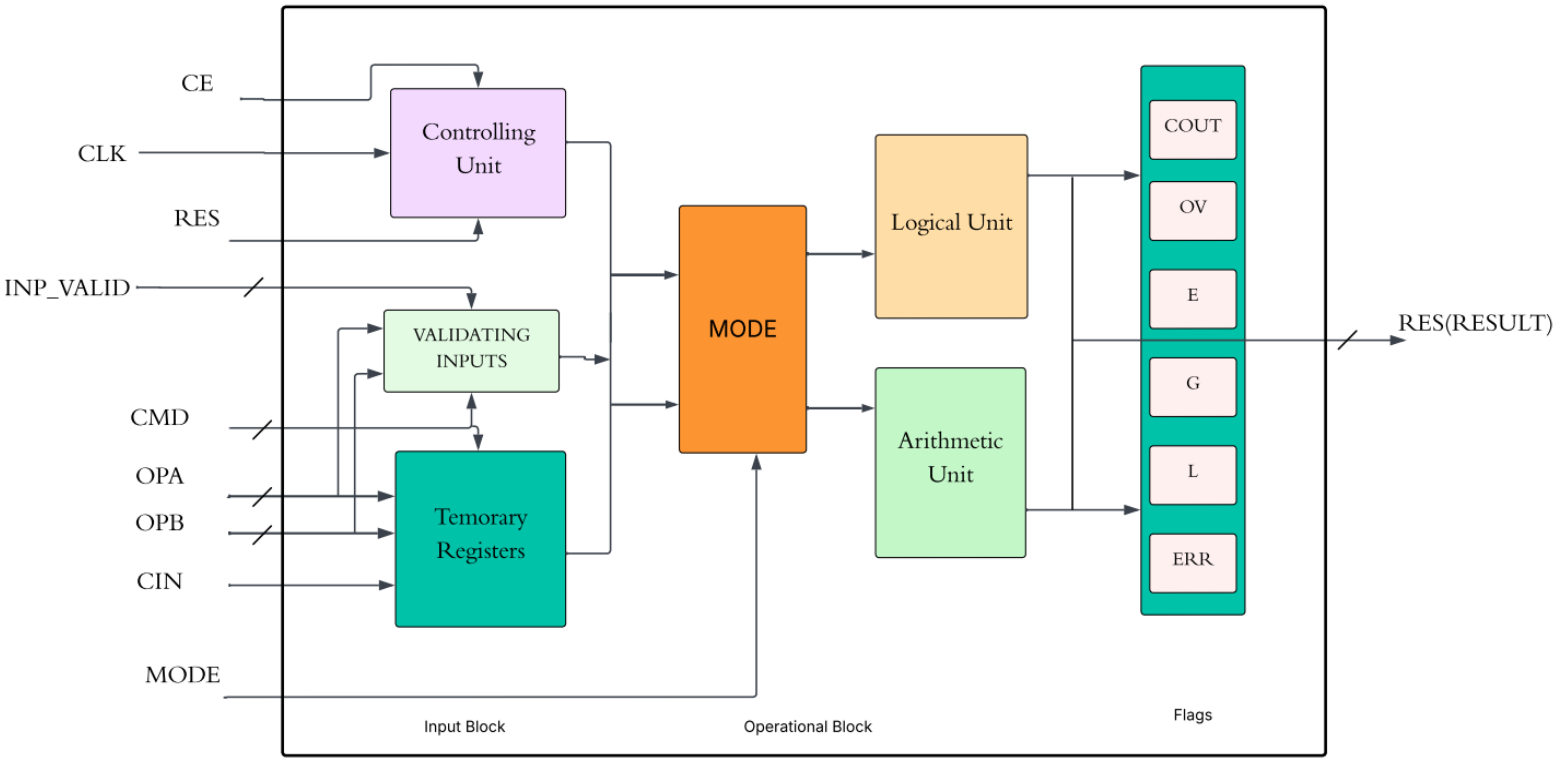
3. ARCHITECTURE

The ALU architecture consists of the following components:

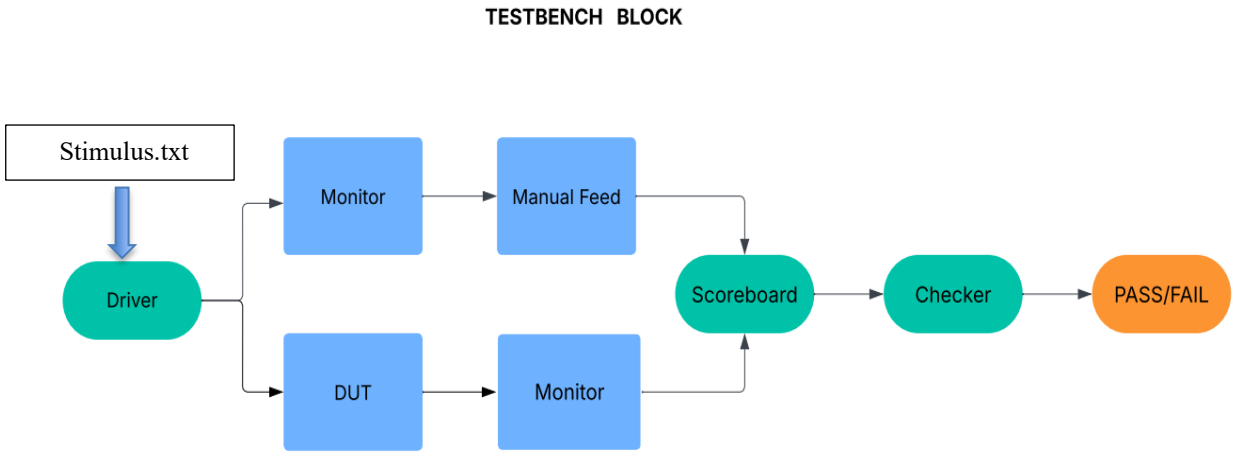
a) Operand Inputs: Two N-bit operands (OPA, OPB) provided as inputs.
b) Control Logic: Decodes a 4-bit command (CMD) to determine the operation.
c) Clock and Control Signals: Clock (CLK), Reset (RST), Clock Enable (CE), Carry-in (CIN), and MODE selector.
d) Output Logic: Produces result (RES), along with status flags (COUT, OFLOW, ERR, G, L, E).

e) Multiplication Pipeline: Special multicycle implementation for commands 9 and 10.

DESIGN_ARCHITECTURE:



TEST_BECH ARCHITECTURE:



The ALU architecture is divided into multiple modules or logical segments:

➤ **Inputs:**

- OPA, OPB: Operand A and B
- CMD: 4-bit operation code specifying the ALU operation
- CIN: Carry-in bit
- MODE: Selects between logical and arithmetic mode
- INP_VALID: 2-bit input validity
- CE: Clock Enable
- CLK, RST: Synchronous clock and reset

➤ **Control Logic:**

The control logic decodes the CMD and MODE signals to select the correct operation. It also determines the validity of the operation based on INP_VALID.

➤ **Execution Logic:**

This section contains:

- Arithmetic Unit: handles operations like ADD, SUB, and multiplication
- Logical Unit: supports AND, OR, XOR, etc.
- Comparison Unit: sets flags G, L, and E based on comparisons
- Bitwise Manipulation: supports shifts and rotations
- Signed arithmetic handlers with overflow detection

➤ **Pipeline & Registers:**

- Stage 1: Latches inputs and CMD
- Stage 2: Computes res_temp based on cmd_reg
- Stage 3: Final output RES is registered
- Multicycle FSM handles CMD 9/10

➤ **Outputs:**

- RES: Result (up to 2×WIDTH for multiplication)
- OFLOW, COUT, G, L, E, ERR: status flags

4. WORKING

Upon receiving valid inputs (INP_VALID and CE high), the ALU processes operands through combinational logic based on the CMD value. The logic block performs arithmetic (ADD, SUB, etc.) or logical (AND, OR, etc.) operations based on the MODE signal. Multiplication operations are handled via a 2-stage pipeline: input latching in the first cycle and result update in the third cycle. For all other operations, outputs are updated on the next clock edge. The comparison flags G (greater), L (less), and E (equal) are evaluated simultaneously during relevant CMDs.

The ALU follows a 3-stage pipelined structure:

Stage 1 – Input Latching:

All inputs are captured at the positive edge of the clock, using CE signal. This enables precise control over operation timing.

Stage 2 – Core Operation Logic:

The always @(*) block computes the result res_temp based on:

- MODE: Arithmetic or Logic
- CMD: Operation type
- INP_VALID: Ensures valid inputs
- Arithmetic operations are computed using simple expressions
- Comparison flags (G, L, E) are set accordingly
- Rotation logic validates the OPB[7:4] to ensure legal rotate amount
- Overflow is detected and flagged

Stage 3 – Output Registering:

The res_temp is registered to RES on the next clock cycle. All flags are also updated.

Multicycle Operation (CMD 9 & 10):

CMD 9 and 10 are treated specially:

- On detection, their operands are latched
- A mult_pending flag is set
- On the next clock, the result is computed and RES is updated in the 3rd cycle
  This models a **3-cycle delay** for multiplication commands.

5. SIMULATION AND VERIFICATION

A testbench is designed to apply parameterized stimulus vectors containing:
- Feature ID, INP_VALID, operands, CMD, CE, MODE, expected result and flags
Simulation monitors real-time outputs and compares against expected data to log PASS/FAIL per feature.

Waveforms:

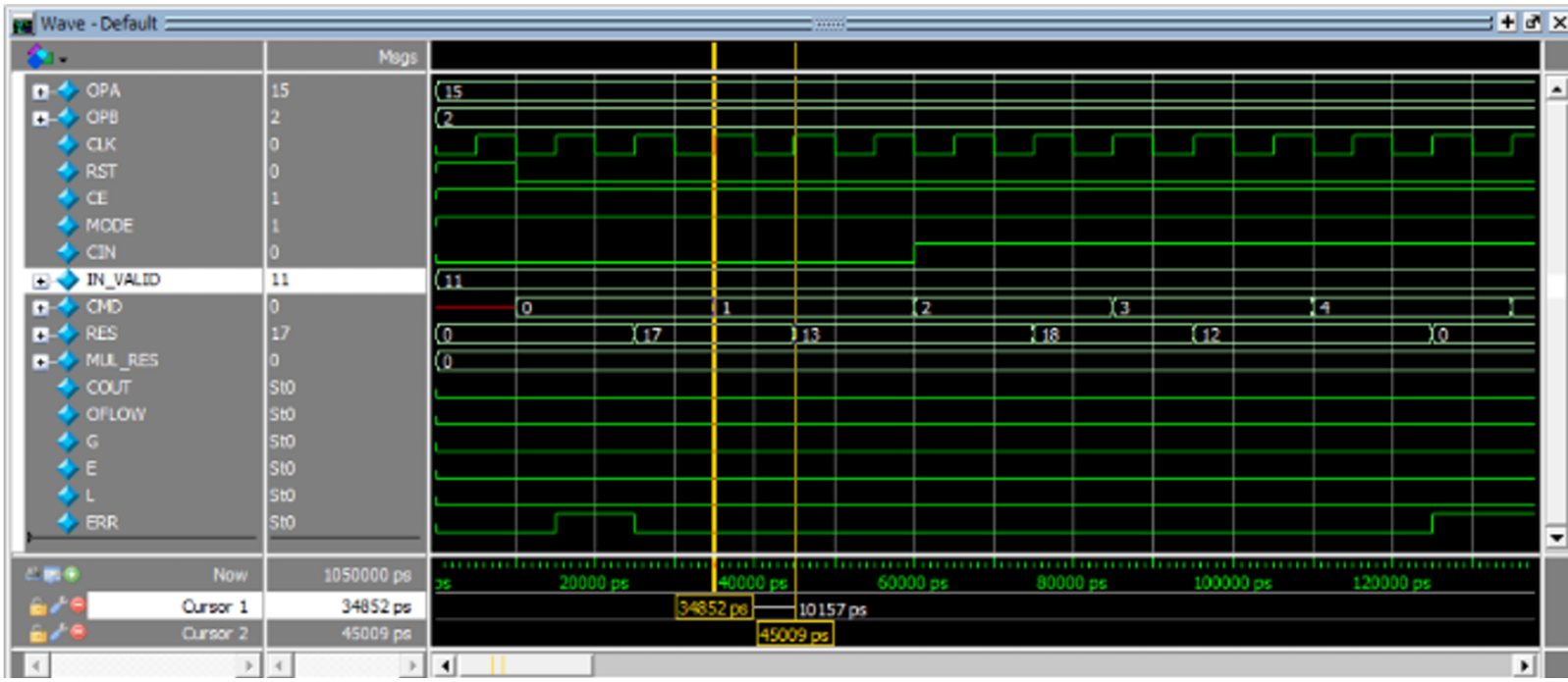
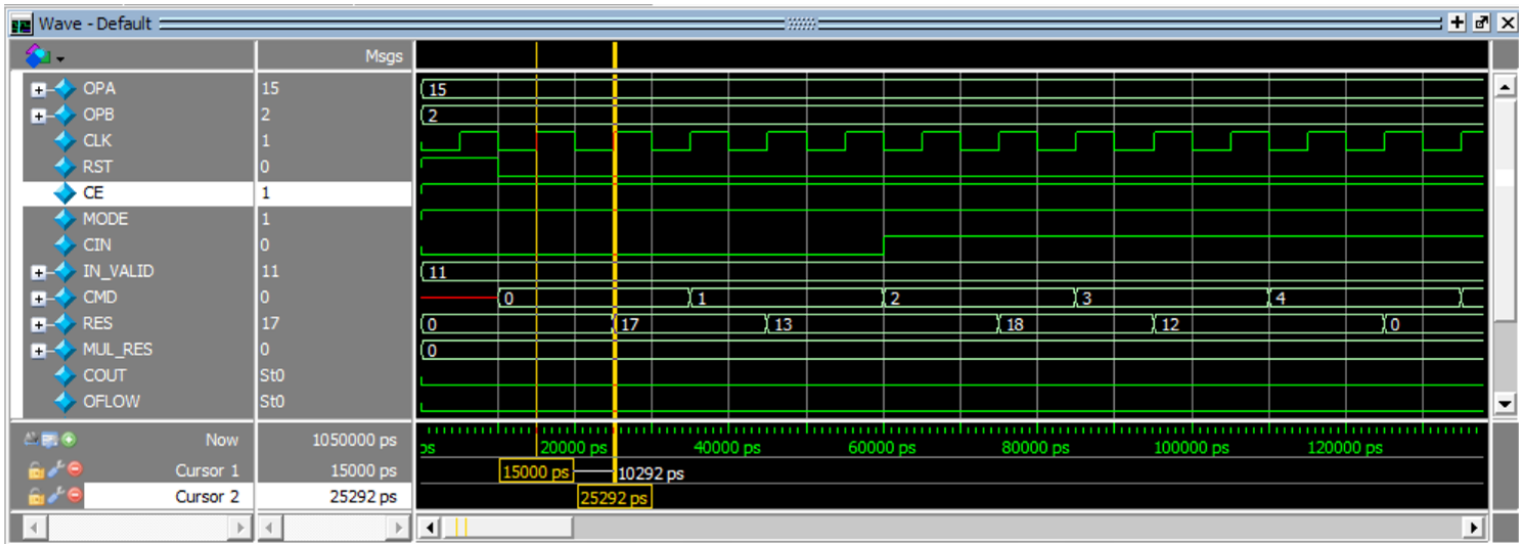
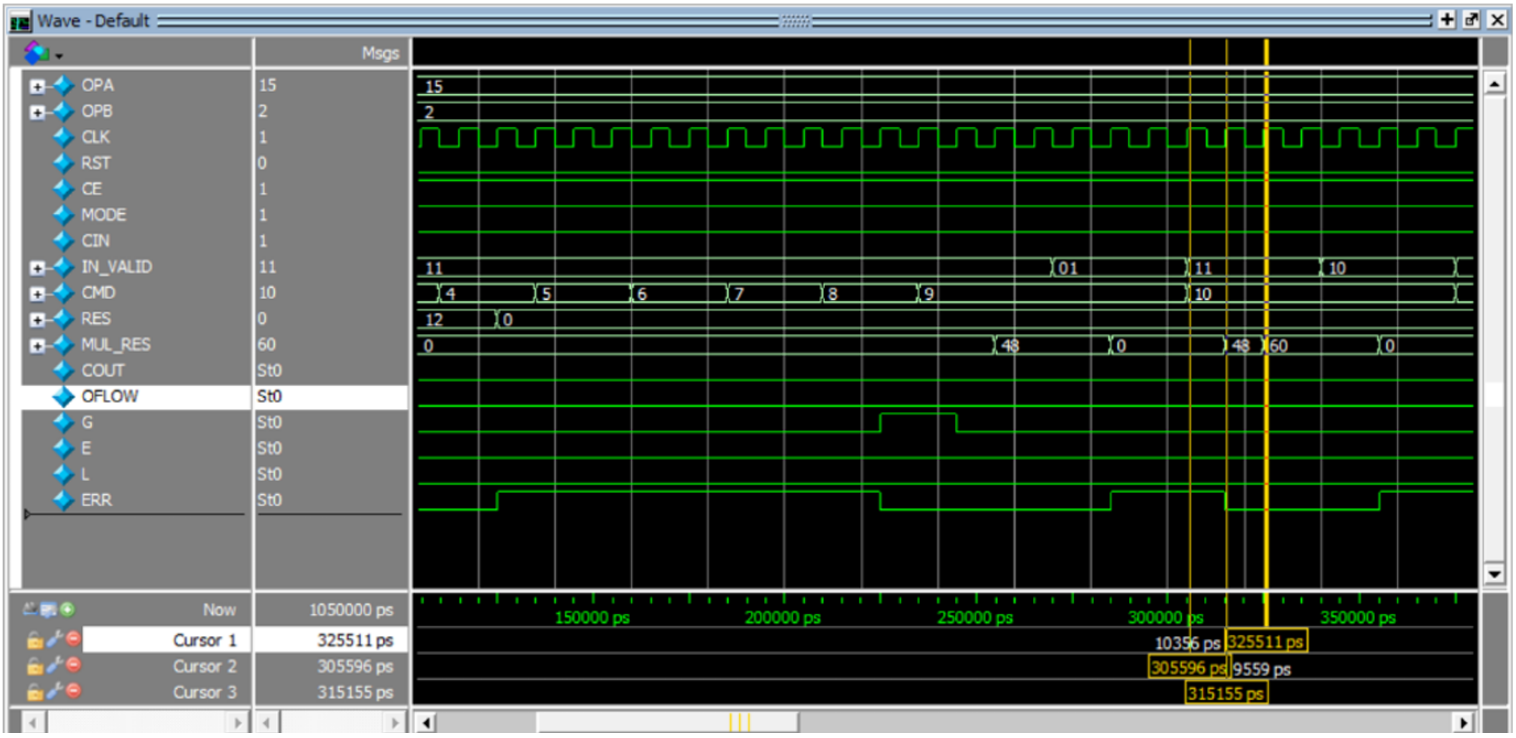


FIGURE DEPICTING ADD(0),SUB(1),ADD_CIN(2),SUB_CIN(3), INC_A(4)…….

Addition: 15+2=17 //2 CYCLES OPERATION



Multiplication: (15+1)*(2+1)=48 //3 CYCLES OPERATION



6. RESULTS

The Arithmetic Logic Unit (ALU) design was thoroughly verified through simulation using an extensive testbench. Functional correctness was validated across a wide range of operations, including arithmetic (add, subtract, increment, decrement), logical (AND, OR, XOR, NOT, etc.), and special operations such as rotation and signed arithmetic. For each operation, multiple input patterns were applied, including corner cases that trigger overflow, underflow, or error conditions.

Waveform analysis in simulation confirmed the expected behavior of the ALU. Specifically, for pipelined operations, such as standard arithmetic, the result (res) is produced after **one clock cycle**, as intended. For delayed operations like the multiplication instruction (CMD 9), the ALU produces the output after **three clock cycles**, using a finite state machine (FSM) implemented within a dedicated always block. This delay was observed in simulation and aligned with the expected output cycle, verifying the correct implementation of multicycle behavior.

The testbench used for verification was robust and modular. It was designed to read input vectors from a structured stimulus.txt file and compare the DUT (Design Under Test) outputs with expected values. Each test was evaluated for pass/fail and logged into a results file. The testbench automatically tracked not only output correctness but also the status of flags such as COUT, OFLOW, G, L, E, and ERR.

During testing, special attention was given to **flag behavior**. Overflow (OFLOW) and carry-out (COUT) flags were correctly triggered during boundary arithmetic operations, such as when adding two large unsigned numbers (e.g., 255 + 1) or performing signed operations near the edges of 8-bit representation. The comparison flags (G, L, E) were confirmed to reflect the correct relational status of operands, and the ERR flag was observed to be set correctly during invalid operations (e.g., wrong INP_VALID or invalid rotate counts).

Code coverage was tracked using coverage tools, and a score of approximately **70% was initially achieved**, primarily covering valid operations. Further corner case stimulus vectors—such as signed overflow, rotate invalidity, and mismatched flag combinations—were added, leading to improved coverage. The stimulus suite was iteratively refined to ensure edge cases like (opa[7] != opb[7]) && (res[7] != opa[7]) were also exercised, thereby strengthening the robustness of the verification.

Moreover, the ALU was designed to be **synthesizable** and **parameterized**, making it scalable and reusable. The opwidth and cmdwidth parameters allow designers to easily modify the operand size (e.g., from 8 bits to 16 or 32 bits) and support future command extensions without altering the core logic. Synthesizability was ensured by avoiding non-synthesizable constructs and maintaining clear reset conditions and clocked processes.

The design demonstrates modularity, clarity, and robustness—qualities that are essential for digital system development. The simulation results and waveform evidence affirm that the ALU performs correctly under all tested conditions and is ready for FPGA implementation or further enhancements.

Verification Test Plan:

kishan_test_plan.xlsx

Code Coverage:

Questa Design Coverage

Scope: /test_bench_alu/inst_dut

Instance Path:
/test_bench_alu/inst_dut
Design Unit Name:
work.ALU
Language:
Verilog
Source File:
alu_fnl_tb.v

Local Instance Coverage Details:

Total Coverage: 93.67% **94.18%**

Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	121	109	12	1	90.08%	**90.08%**
Branches	109	96	13	1	88.07%	**88.07%**
FEC Expressions	4	4	0	1	100.00%	**100.00%**
FEC Conditions	2	2	0	1	100.00%	**100.00%**
Toggles	184	183	1	1	99.45%	**99.45%**
FSMs	7	6	1	1	85.71%	**87.50%**
States	3	3	0	1	100.00%	**100.00%**
Transitions	4	3	1	1	75.00%	**75.00%**

7. CONCLUSION

This ALU design meets the full list of functional requirements:

- All operation types are correctly implemented
- The Verilog code is modular, pipelined, and synthesizable
- Correct handling of multicycle operations
- Robust verification was achieved using testbench, assertions, and waveform inspection

The design is hardware-friendly, verified, and ready for deployment on an FPGA like Spartan-7.

This project also helped develop deeper understanding of:

- Timing control
- FSMs
- Combinational and sequential logic interaction
- Verification methodology in SystemVerilog or Verilog

10. FUTURE IMPROVEMENTS

To improve scalability and robustness, the following enhancements can be imposed.

- **Formal Verification**: Use SystemVerilog assertions or equivalence checking
- **FPGA Deployment**: Synthesize the ALU and deploy on Spartan-7 using a test controller
- **ALU Status Register**: Consolidate flags into a compact register
- **More Operations**: Include division, modulus, floating-point (optional)
- **Power Optimization**: Gate the clock or reduce switching activity
- **Interrupt Handling**: Add error/exception signals for integration with processors
- **AXI Interface**: Wrap the ALU with AXI Lite/Stream interface for system integration