ALU Project Report

Krishna Negi

6083

1. Introduction

This project details the design, implementation, verification, and results of a parameterized Arithmetic Logic Unit (ALU) implemented in Verilog. The ALU serves as a core combinational block capable of executing diverse arithmetic and logic operations, with synchronous control via input sampling registers and status flag generation for carry, overflow, comparison, and error conditions.

2. Objectives

The primary objectives of this project are:

• Design a modular Verilog ALU supporting 20+ operations with parameterized operand width.

• Implement input validation via INP_VALID gating for flexible operand availability.

• Generate accurate status flags: COUT (carry), OV (overflow), G/L/E (comparison), ERR (invalid).

• Develop a file-driven, cycle-accurate testbench for exhaustive verification of all operations.

• Ensure synthesizability and readiness for pipelined extension.

3. Architecture

The ALU architecture is segmented into distinct blocks:
• Input Sampling Registers: Latch inputs on clock edge when CE=1 to isolate combinational path.
• Control Decoder: Interprets 4-bit CMD and 2-bit INP_VALID to route signals to the proper functional unit.
• Combinational Core: Executes arithmetic (add, sub, signed, multiply variants) and logic (AND/OR/XOR, shifts, rotates) operations.
• Flag Generation: Computes carry, overflow, and comparison outcomes within the combinational block.
• Output Registers: Synchronize outputs to clock edge, ensuring glitch-free delivery.

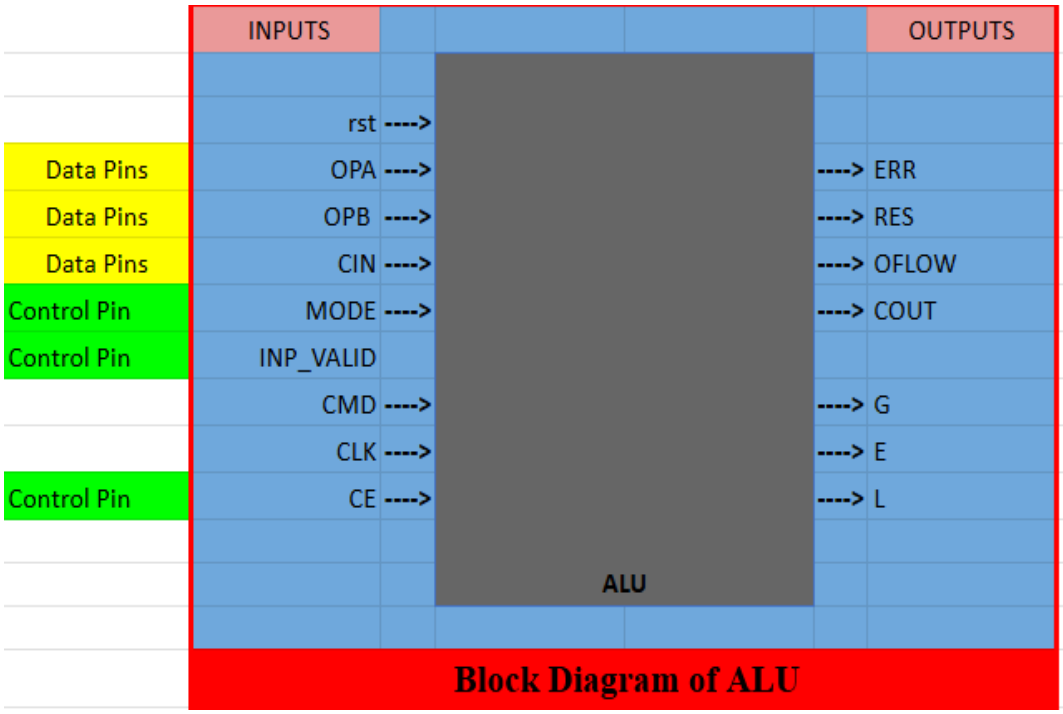


Figure 1 Block Diagram

4. Port Description

Port	Direction	Width	Description
clk	in	1	System clock
rst	in	1	Asynchronous reset
ce	in	1	Clock enable
mode	in	1	1=Arithmetic, 0=Logic
inp_valid	in	2	Operand validity indicator
cmd	in	4	Operation code
opa/opb	in	8	Operand data buses
cin	in	1	Carry-in
res	out	8-16	Computation result

cout	out	1	Carry-out flag
ov	out	1	Overflow flag
g/l/e	out	1 each	Comparison flags
err	out	1	Error flag for invalid ops

5. Implementation Details

5.1 Input Sampling

The inputs are registered using an always block sensitive to clk and rst. When CE is asserted, inputs are latched into internal registers (opa_p, opb_p, cmd_p, etc.), decoupling combinational logic.

5.2 Combinational Core

A single combinational always block evaluates operations based on mode and valid signals. Nested case statements ensure clear separation of arithmetic and logic flows. Key operations include:

• Unsigned Addition/Subtraction

• Signed Arithmetic with Overflow Detection

• Increment/Decrement

• Bitwise Logic (AND, OR, XOR, XNOR)

• Shift Left/Right

• Rotate Left/Right with Error Check

5.3 Opcode Definitions

Mnemonic	Code (binary)	Function
ADD	0000	Unsigned add
SUB	0001	Unsigned subtract
ADD_CIN	0010	Add with carry
SUB_CIN	0011	Subtract with borrow
INC_A/DEC_A	0100/0101	Increment/Decrement A
INC_B/DEC_B	0110/0111	Increment/Decrement B

CMP	1000	Compare A vs B
MULT_INC	1001	Multiply after increment
MULT_SHIFT_A	1010	Shift A then multiply
S_ADD/S_SUB	1011/1100	Signed add/sub
AND	0000	Bitwise AND
OR	0010	Bitwise OR
XOR	0100	Bitwise XOR
NOR/NAND	0011/0001	Bitwise NOR/NAND
SHL/SHR	1001/1000	Shift
ROL/ROR	1100/1101	Rotate

6. Testbench Methodology

The testbench is implemented in Verilog using a file-driven approach:
- $readmemb reads 'stimulus.txt' containing 74 test vectors.
- 'driver' task fetches and applies inputs at the positive clock edge.
- 'monitor' task captures outputs after a fixed delay and packs them for comparison.
- 'score_board' task compares actual vs expected and logs results.
All possible opcode and mode combinations are included, with edge cases for error flags.

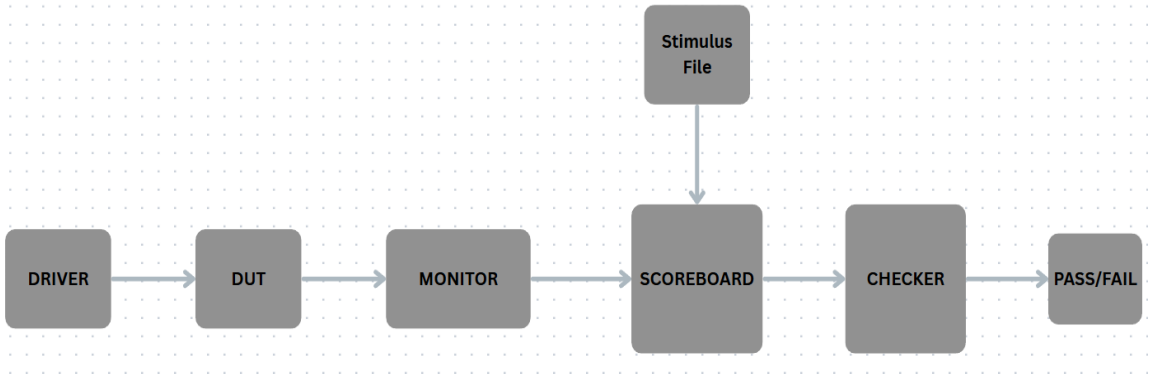


Figure 2 Testbench Architecture

7. Simulation Results

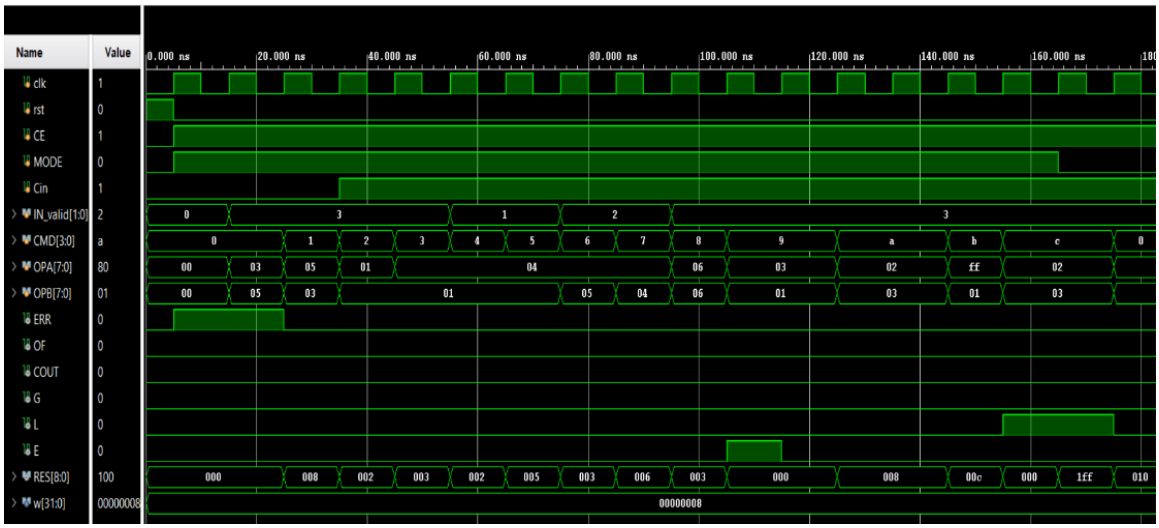


Figure 3 Output of Arithmetic Operations

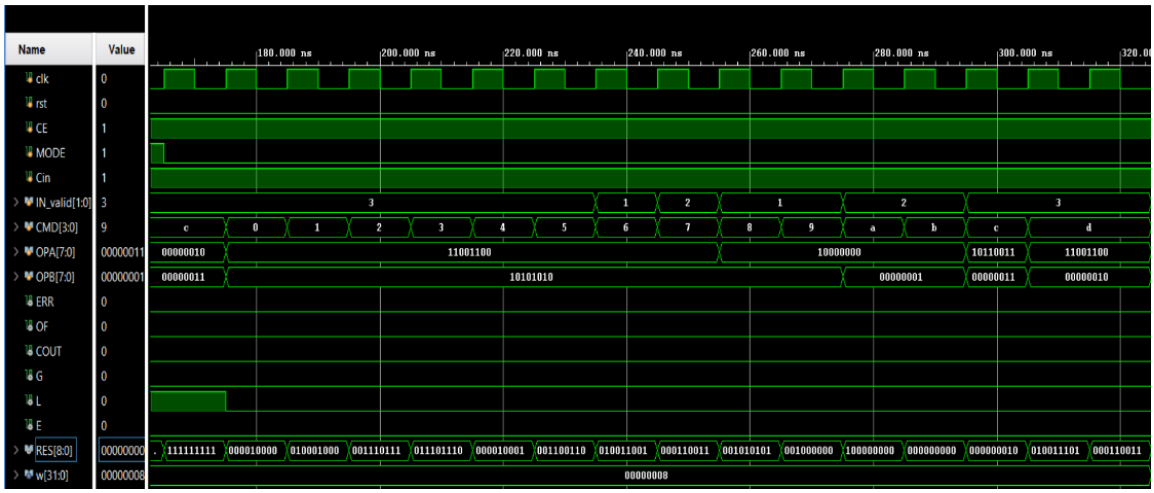
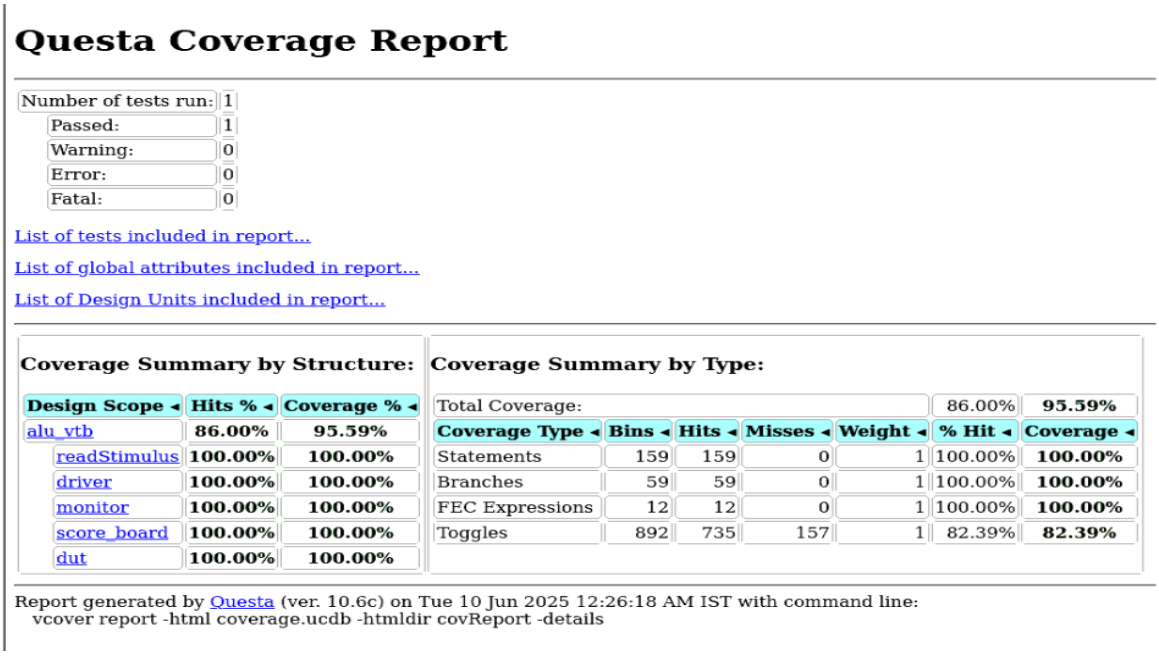


Figure 4 Output of Logical Operations



Questa Coverage Report

Number of tests run:	1
Passed:	1
Warning:	0
Error:	0
Fatal:	0

List of tests included in report...

List of global attributes included in report...

List of Design Units included in report...

Coverage Summary by Structure:

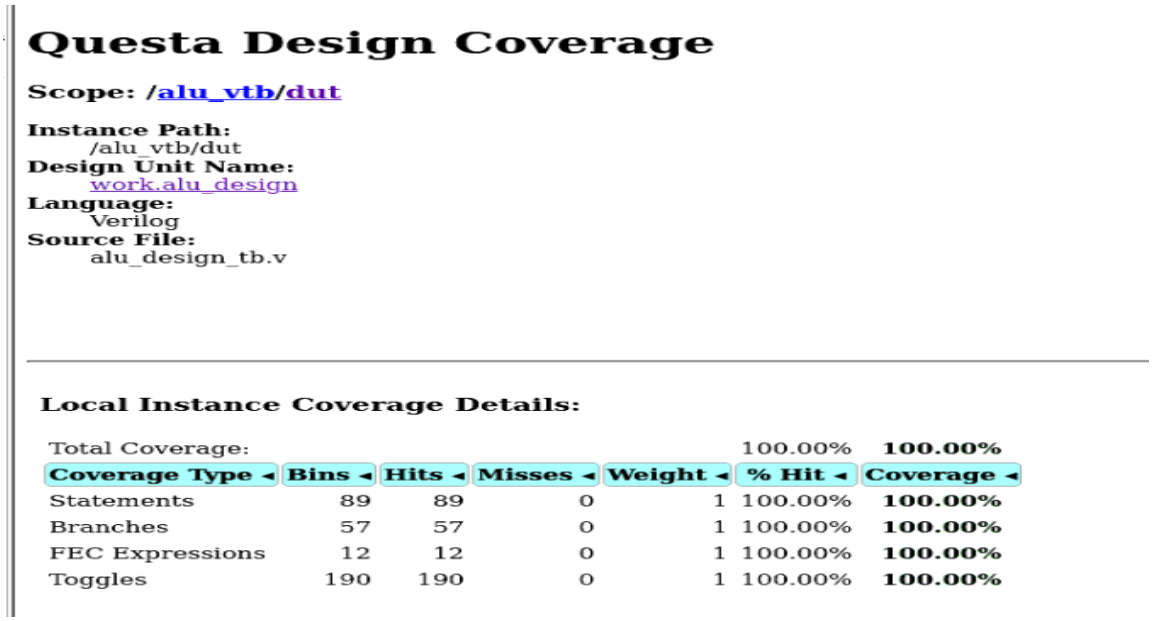
Design Scope	Hits %	Coverage %
alu_vtb	86.00%	95.59%
readStimulus	100.00%	100.00%
driver	100.00%	100.00%
monitor	100.00%	100.00%
score_board	100.00%	100.00%
dut	100.00%	100.00%

Coverage Summary by Type:

Total Coverage:					86.00%	95.59%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Statements	159	159	0	1	100.00%	100.00%
Branches	59	59	0	1	100.00%	100.00%
FEC Expressions	12	12	0	1	100.00%	100.00%
Toggles	892	735	157	1	82.39%	82.39%

Report generated by Questa (ver. 10.6c) on Tue 10 Jun 2025 12:26:18 AM IST with command line:
vcover report -html coverage.ucdb -htmldir covReport -details

Figure 5.1 Coverage Report



Questa Design Coverage

Scope: /alu_vtb/dut

Instance Path:
/alu_vtb/dut
Design Unit Name:
work.alu_design
Language:
Verilog
Source File:
alu_design_tb.v

Local Instance Coverage Details:

Total Coverage:					100.00%	100.00%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Statements	89	89	0	1	100.00%	100.00%
Branches	57	57	0	1	100.00%	100.00%
FEC Expressions	12	12	0	1	100.00%	100.00%
Toggles	190	190	0	1	100.00%	100.00%

Figure 5.2 Coverage Report

8. Conclusion and Future Work

The designed ALU is verified, synthesizable, and ready for integration into larger datapaths. Future enhancements include:

• Multi-stage pipelining to increase throughput.

- Extended operand widths (16/32/64-bit).
- Additional operations: division, modulo, population count.
- Formal verification using SystemVerilog Assertions.
- UVM-based random testbench for higher coverage metrics.