ALU MINI PROJECT REPORT

Submitted by:

Name: Nishanth Gowda C J

Company:Mirafra Technologies

EMP ID: 6089

ABSTRACT

This project focuses on the design and verification of a parameterized Arithmetic Logic Unit (ALU) that performs various arithmetic and logical operations based on input commands. The ALU accepts control and data inputs to carry out functions such as addition, subtraction, logical operations, shifts, and rotates. The implementation ensures modularity and scalability with parameterized data widths, suitable for integration into larger digital systems

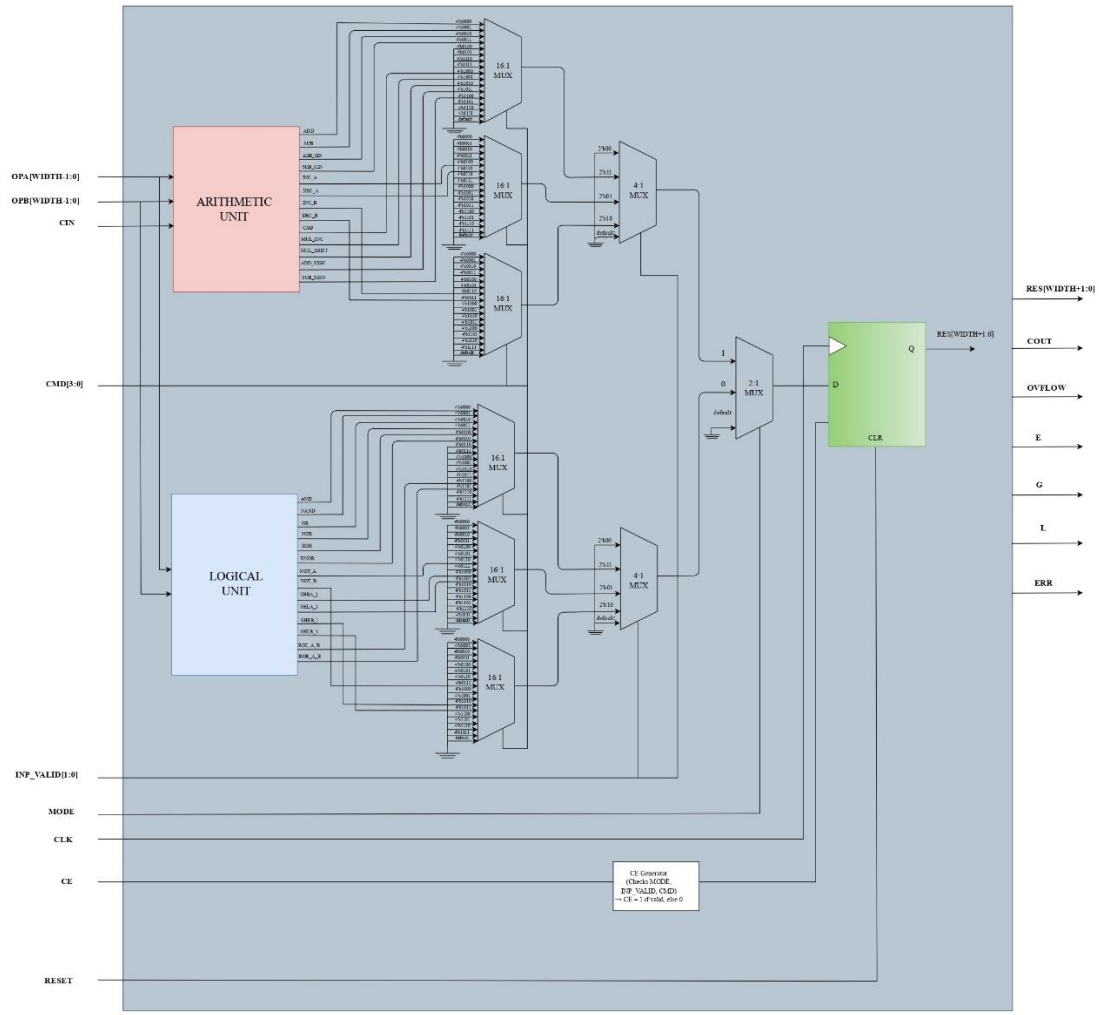
INTRODUCTION

The Arithmetic Logic Unit (ALU) is a core component of a processor used for arithmetic and logical computations. The ALU designed in this project supports both signed and unsigned operations and features parameterized inputs for flexible data width. The architecture is designed to interpret various control signals, enabling diverse operations with accurate flag outputs

OBJECTIVES:

- To design a parameterized ALU supporting arithmetic and logic operations.
- To implement extended functionalities such as shift, rotate, compare.
- To ensure modularity for easy testing and integration.
- To generate status flags like carry, overflow, and comparison outputs.

ARCHITECTURE



The ALU design consists of the following key signals:

Control Signals:-

- CLK: Clock signal
- RST: Reset (active high)
- CE: Clock Enable
- MODE: 1 for Arithmetic, 0 for Logic
- INP_VALID: 2-bit input validity for operands
- CMD: 4-bit operation command

Data Signals:

- OPA, OPB: Operands
- CIN: Carry-in input

Outputs:

- RES: Operation result
- OFLOW: Overflow
- COUT: Carry out
- G, L, E: Comparator outputs
- ERR: Error signal for rotate command

Functional Description:

- Arithmetic Operations (when MODE = 1)
- CMD Operation
- 0 ADD
- 1 SUB
- 2 ADD with CIN
- 3 SUB with CIN
- 4-7 INC_A, DEC_A, INC_B, DEC_B
- 8 CMP (compare OPA and OPB)
- 9-12 Multiply or complex arithmetic with flags

Logical Operations (when MODE = 0):

- CMD Operation 0-5 AND, NAND, OR, NOR, XOR, XNOR
- 6-7 NOT_A, NOT_B
- 8-11 SHL1, SHR1 for A and B
- 12-13 ROL_A_B, ROR_A_B (rotate based on OPB)

Error Handling: If rotate commands (CMD=12 or 13) are selected and upper 4 bits of OPB are non-zero, ERR is raised

WORKING

The ALU is implemented using a 3-stage pipelined architecture to support efficient execution of arithmetic and logic operations. The control and data flow depend on clock (CLK), reset (RST), clock enable (CE), input validity (INP_VALID), and operation mode (MODE).

Basic Control Flow

- Reset (RST):
    - When RST = 1, the ALU is asynchronously reset.
    - All internal registers and outputs are cleared immediately.
- Clock Enable (CE):
    - ALU performs any operation only when CE = 1 and RST = 0.
    - If CE = 0, the design holds previous values and doesn't perform any operation.
- Input Validity (INP_VALID):
    - It's a 2-bit input:
        - 00: No operand is valid → ALU does not perform any operation.
        - 01: Only Operand A is valid → Valid for single-operand commands only.
        - 10: Only Operand B is valid → Valid for single-operand commands only.
        - 11: Both operands are valid → Required for dual-operand commands (e.g., ADD, SUB, AND, etc.)
    - Any mismatch (e.g., INP_VALID = 01 used with ADD command) is treated as incorrect and may lead to undefined behavior.
- Operation Mode (MODE):
    - MODE = 1: Arithmetic operations
    - MODE = 0: Logical operations

Pipeline Stages

stage 1: Input Capture

- Activated on positive clock edge if CE = 1 and RST = 0.
- Inputs such as OPA, OPB, CMD, MODE, CIN, and INP_VALID are latched.
- These are stored in internal registers for further decoding and processing.
- Waveform behavior: Inputs become stable in this stage.

Stage 2: Operation Execution (Except Multiply)

- Based on latched CMD and MODE, the ALU performs:
    - Arithmetic (ADD, SUB, INC, DEC, etc.)
    - Logic (AND, OR, XOR, NOT, SHIFT, etc.)
- This stage also evaluates comparator outputs (G, L, E) and flags (COUT, OFLOW).
- Waveform behavior: RES, flags, and comparators are updated in this stage (except for multiply).

Stage 3: Multiply Operation Execution

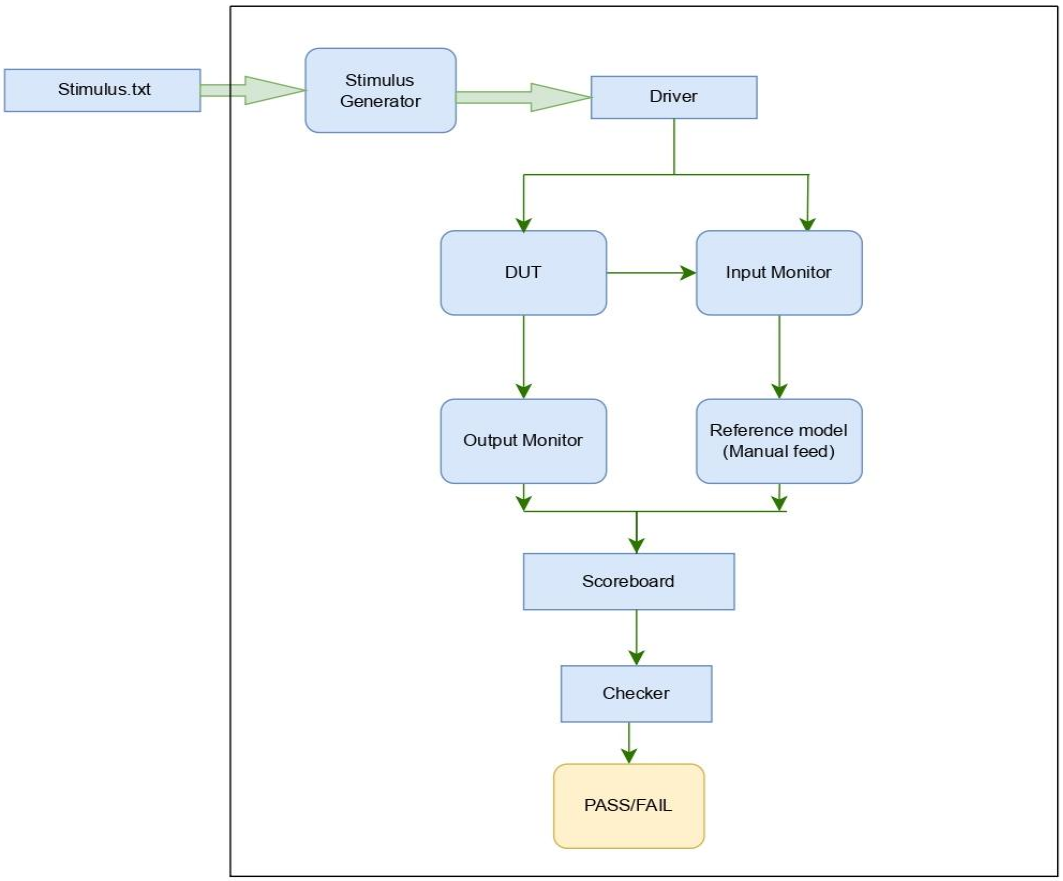
- Multiply operations (CMD 9, 10) are more complex and take extra cycles.
- Their outputs (RES) are computed and updated in this final stage.
- Ensures pipeline timing is maintained for lighter operations.
- Waveform behavior: RES for multiply commands is updated here with 1-cycle delay from stage 2.

Special Condition: Rotate Commands (CMD 12, 13)

- If CMD = 12 or 13 and MODE = 0, the ALU performs rotate left/right on OPA based on OPB

- If OPB[7:4] contains any 1, then ERR = 1.

TESTBENCH ARCHITECTURE



A structured, file-driven testbench is used to automate functional verification of the ALU.

1. stimulus.txt (Test Vector File)

- External file with **59-bit vectors** defining test cases
- Easy to update, supports regression testing

2. Stimulus Generator

- Reads stimulus.txt via $readmemb
- Loads test vectors into memory and supplies them sequentially to the drive

3. ALU DUT (Design Under Test)

- Receives inputs from the driver
- Produces outputs to be checked (e.g., RES, COUT, OFLOW, ERR, G, L, E)

4. Driver

- Extracts and applies inputs from each test vector
- Ensures proper timing and control signal delivery

5. Monitor

- Captures DUT outputs for each test case
- Non-intrusively records behavior for comparison

6. Scoreboard

- Compares DUT outputs with expected results
- Records pass/fail status for each test case

7. Report Generator

- Summarizes overall results
- Writes a results.txt file with pass/fail outcomes

SIMULATION AND VERIFICATION

To validate the functionality of the ALU, a parameterized testbench was created that feeds 40-bit input packets into the ALU. Each packet encapsulates control, data, and verification fields, making the simulation compact and traceable.

Field	Width	Description
Feature ID	8 bits	Uniquely identifies the test case or operation
INP_VALID	2 bits	Validity of operands: 00, 01, 10, 11
OPA	4 bits	Operand A (parameterized width)
OPB	4 bits	Operand B (parameterized width)
CMD	4 bits	Command for operation (arithmetic/logical)
CIN	1 bit	Carry-in signal

Field	Width	Description
CE	1 bit	Clock enable (must be 1 to allow operation)
MODE	1 bit	1: Arithmetic, 0: Logical
RESERVED	4 bits	Reserved for future use or alignment
RESULT	5 bits	Expected result of the operation (width + 1)
COUT	1 bit	Expected carry out flag
E, G, L	3 bits	Expected comparator flags: Equal, Greater, Less
OFLOW	1 bit	Expected overflow flag

Simulation Flow

- The testbench reads input packets line-by-line from a structured stimulus.txt file.
- It decodes the fields and drives corresponding signals into the ALU.
- A monitor captures the ALU's output and compares it with the expected result and flags.
- Each test case logs a PASS/FAIL result based on the match.

Verification

The ALU is 3-stage pipelined:

Stage 1: Input latching if CE=1 and RST=0

Stage 2: Result and flag update for normal ops

Stage 3: Multiply operations produce results (e.g., CMD=9 or 10)

Waveform analysis :

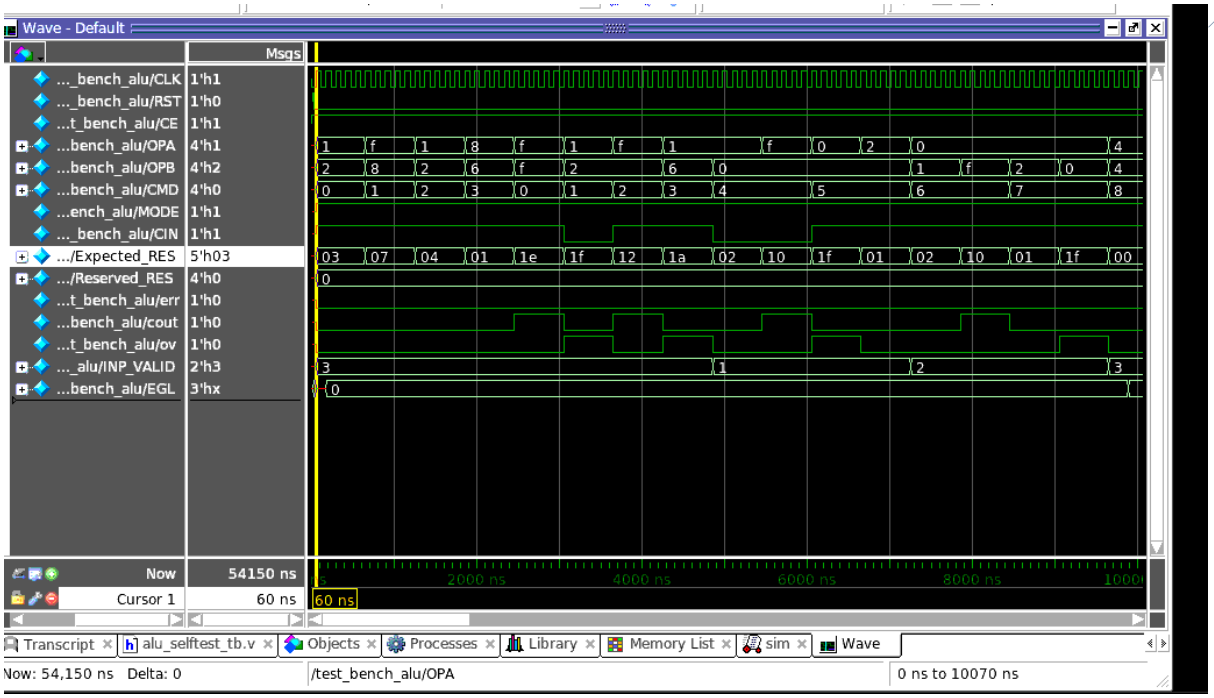


Figure 1: ALU Operation

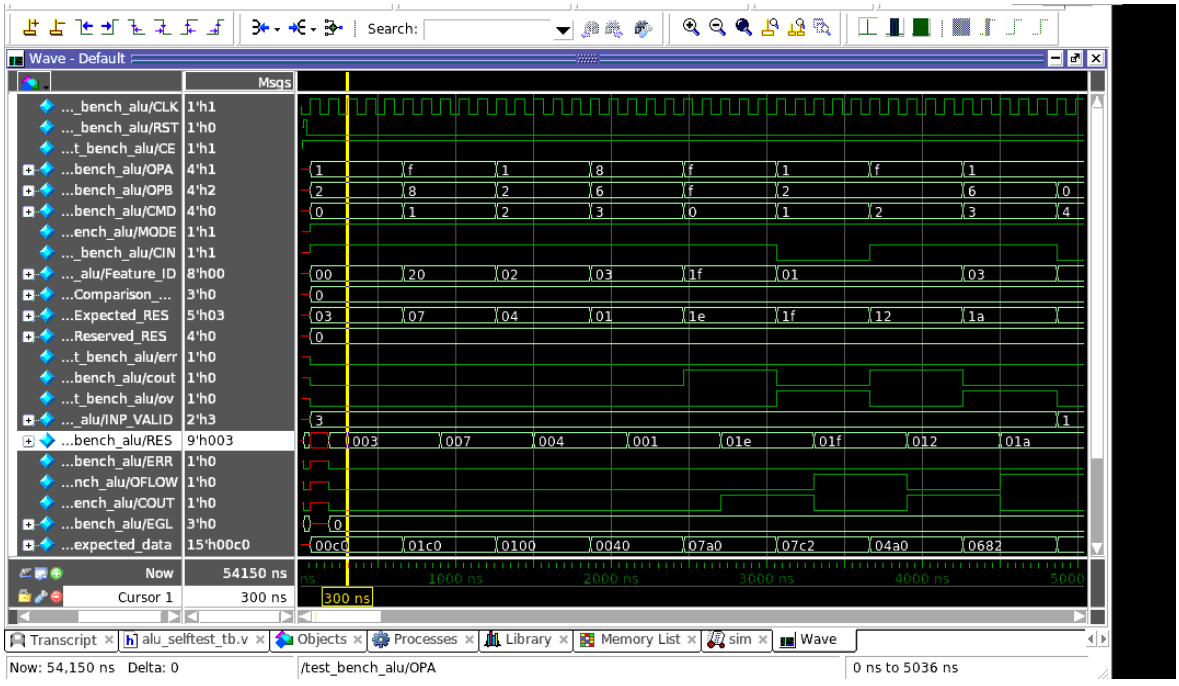


Figure 2: ALU Operation – ADD

In figure 2 OPA=1,OPB=2 input updated on 1 clk cycle RES =3 ,RES updated on next clk cycle.

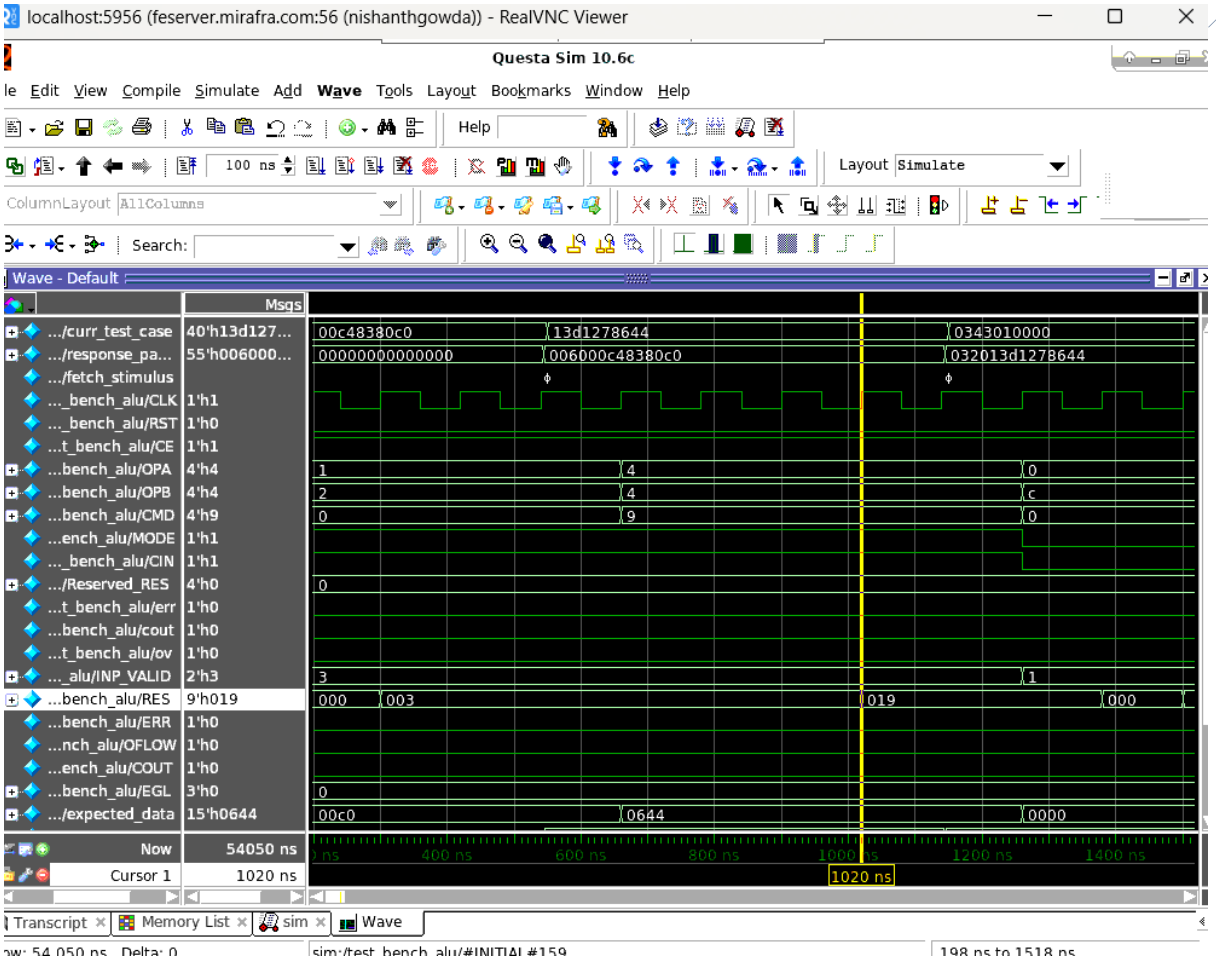


Figure 3: ALU Operation – MUL

In figure 3 OPA=4,OPB=4 input updated on 1 clk cycle and it will be increment by than opa and opb will be 5 , RES =19(in hexadecimal) ,RES updated on 3 clk cycle.

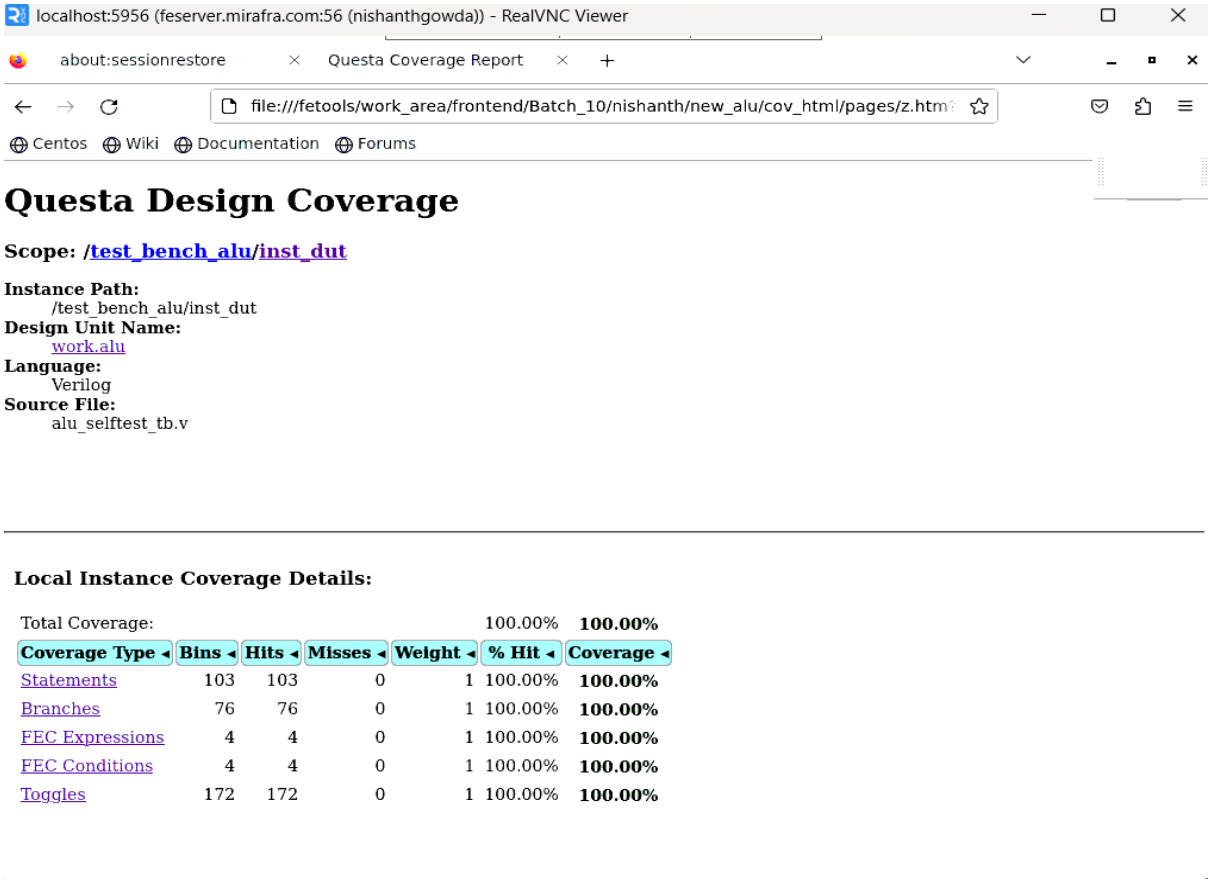
RESULTS

The ALU design was fully verified through comprehensive simulation using a modular testbench. Functional correctness was confirmed for all arithmetic, logical, and special operations, including edge cases like overflow, underflow, and invalid inputs. Waveform analysis validated correct timing, with single-cycle outputs for standard operations and multi-cycle delays for multiplication as designed.

Flag behaviors (COUT, OFLOW, G, L, E, ERR) were accurately triggered under appropriate conditions. The testbench automatically checked outputs and flags against expected results, logging pass/fail status.

After iterative refinement of stimulus vectors, 100% code coverage was achieved, ensuring all functional scenarios and corner cases were thoroughly exercised. The design is fully synthesizable and parameterized for scalability, making it robust and reusable. Overall, the simulation confirms the ALU meets all design specifications reliably.

Code coverage:



CONCLUSION

The ALU design successfully meets all functional requirements with correct implementation of arithmetic and logical operations. It is modular, pipelined, synthesizable, and handles multicycle operations efficiently. Verified using a robust testbench, waveform inspection. the ALU is hardware-ready and suitable for FPGA deployment. This project also deepened understanding of timing, FSMs, and verification methods.

FUTURE IMPROVEMENTS

- Optimize all operations to complete in 3 clock cycles.

- Add floating-point and complex number support.
- Implement AI-based testbench generation and intelligent debugging tools.
- Enhance formal verification with assertions and coverage-driven techniques.