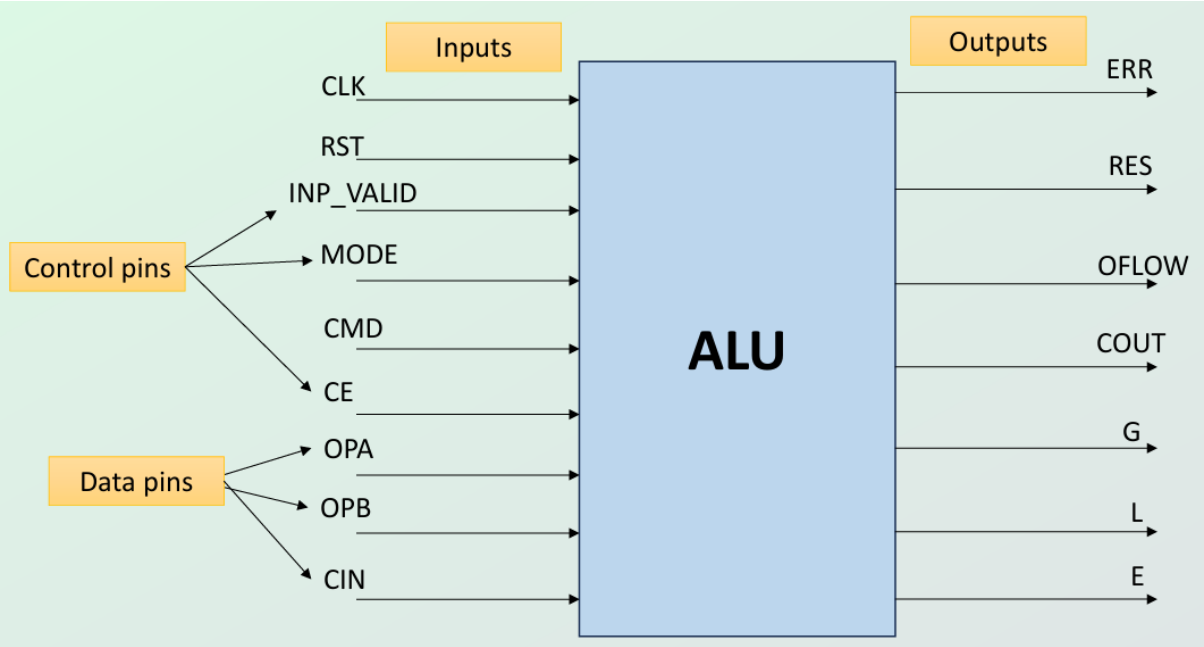
ALU PROJECT
DESIGN DOCUMENT

KUSUMANCHI ADITHYA

EMP ID: 6084

Introduction

The Arithmetic Logic Unit (ALU) is one of the core components in any digital processing system, responsible for executing arithmetic and logical operations on binary data. It acts as the computational engine of microprocessors, digital signal processors, and embedded systems. In this project, a versatile and parameterized ALU is designed using Verilog HDL. The design can perform a wide variety of operations including addition, subtraction, logical AND/OR/XOR, shift, and rotate functions.



The ALU supports both signed and unsigned data formats and is equipped with control signals that guide the operation mode (arithmetic or logic). It also provides status outputs such as carry out, overflow, equality, and comparison flags, which are essential for decision-making processes in higher-level modules. The design adheres to modular and synthesizable Verilog standards and is validated through simulation to ensure functional correctness.

Objectives

- To design and implement a parameterized ALU in Verilog HDL capable of executing multiple arithmetic and logical operations.
- To support both single-operand and dual-operand instructions with appropriate control Input valid signal
- To verify the functional correctness of the ALU using simulation and waveform analysis.
- To ensure the design meets timing by following 2 cycle and 3 cycle delay and logic requirements through structured input/output interfacing.

Architecture

The ALU architecture is modular and designed for clarity and scalability. It processes inputs through buffering stages, applies control signals, and routes data through arithmetic or logical execution blocks based on the operation mode. The key components include:

1. Input Buffers

- **Buffer 1** receives primary inputs: OPA, OPB, CIN, CE, MODE, INP_VALID, and CMD.
- These signals are synchronized and processed into internal signals (OPA_T, OPB_T, CE_T, MODE_T, etc.) before being routed to operation units.

2. Buffer 2

- Temporarily stores intermediate operand values (OPA_T1, OPB_T1) for use in multiplication.

3. Operation Units

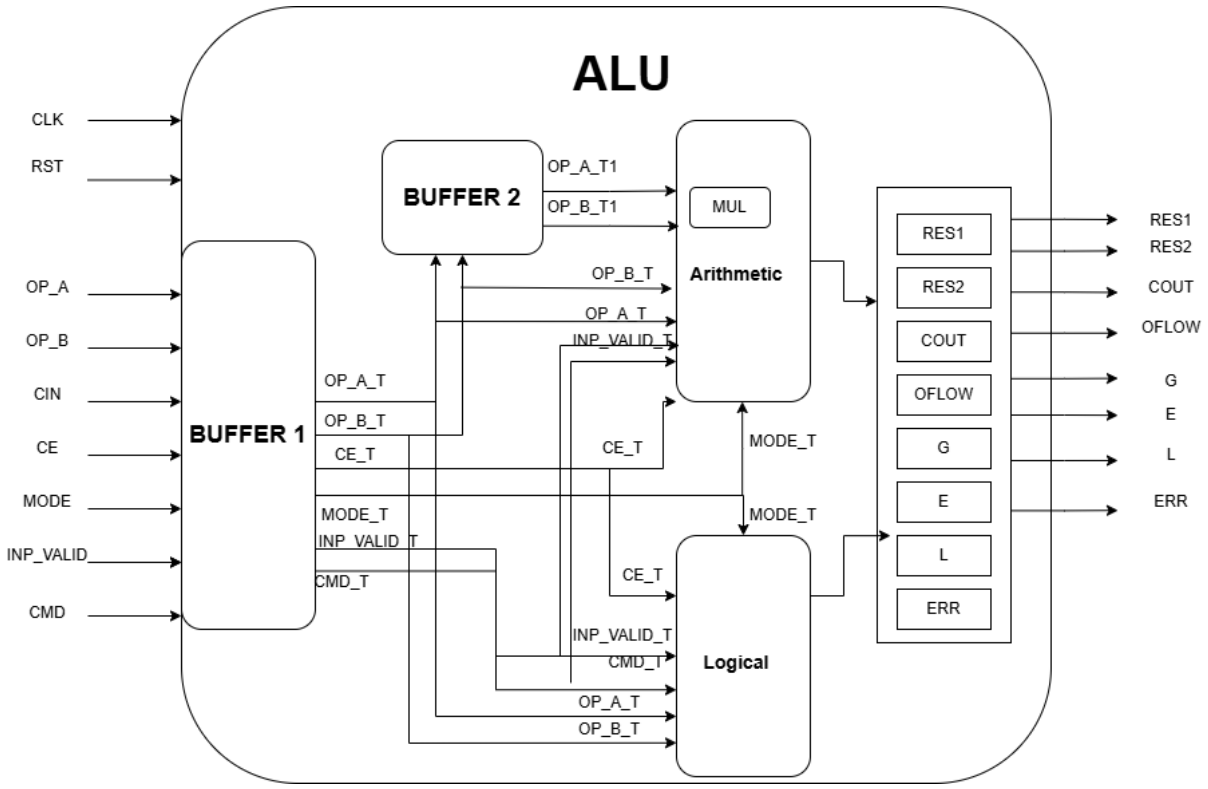
- **Arithmetic Block**: Handles arithmetic instructions such as addition, subtraction, increment, decrement, signed add,sub and multiplication. It receives operands, control signals, and produces results (RES1, RES2) along with status flags like COUT and OFLOW.
- **Logical Block**: Executes bitwise logical operations (AND, OR, XOR, NOT, etc.) and shift/rotate operations.
    - Operates under the control of CMD_T and MODE_T and receives validated operand signals.

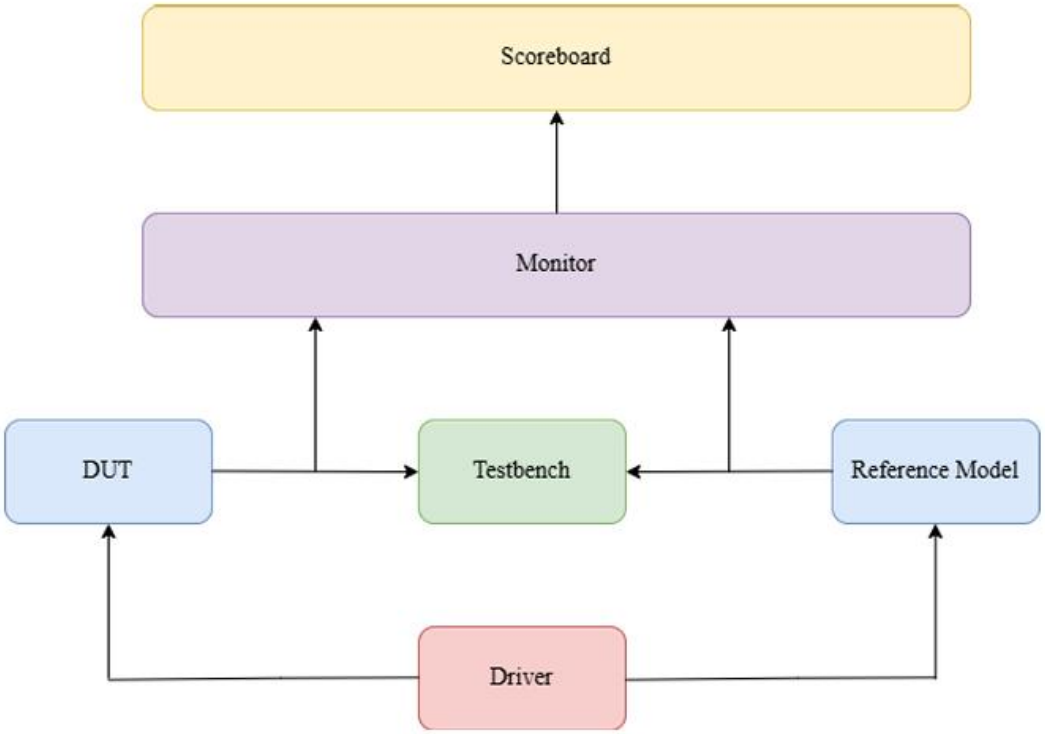
4. Control Path

- The control logic directs data flow based on the MODE_T (Arithmetic/Logical) and CMD_T inputs.
- The signal INP_VALID_T ensures valid operand input combinations per operation requirements.

5. Output Unit

- Collects results from arithmetic and logical blocks.
- Outputs include:
    - **RES1, RES2**: Operation results.
    - **COUT, OFLOW**: Carry-out and overflow flags.
    - **G, L, E**: Comparator flags (Greater than, less than, Equal).
    - **ERR**: Indicates command or input-related errors (e.g., invalid rotate settings).


ALU
CLK
RST
OP_A
OP_B
CIN
CE
MODE
INP_VALID
CMD
BUFFER 1
BUFFER 2
OP_A_T1
OP_B_T1
MUL
Arithmetic
OP_B_T
OP_A_T
INP_VALID_T
OP_A_T
OP_B_T
CE_T
MODE_T
INP_VALID_T
CMD_T
CE_T
MODE_T
MODE_T
CE_T
INP_VALID_T
CMD_T
OP_A_T
OP_B_T
Logical
RES1
RES2
COUT
OFLOW
G
E
L
ERR
RES1
RES2
COUT
OFLOW
G
E
L
ERR


Scoreboard
Monitor
DUT
Testbench
Reference Model
Driver

The testbench architecture shown above is a self-checking, modular verification environment designed to validate the functionality of the ALU. It consists of the following key components:

• **Driver**: Responsible for generating stimulus by driving random and directed test cases to both the Design Under Test (DUT) and the Reference Model. It controls input signals like operands, command, mode, and control flags.

• **Testbench:** Acts as a wrapper that instantiates the DUT, Reference Model, Driver, Monitor, and Scoreboard, facilitating communication among them.

• **DUT (Design Under Test):** The actual Verilog ALU being verified for functional correctness under various scenarios.

• **Reference Model:** A golden model that mimics the ALU functionality and produces expected outputs for comparison. It is assumed to be bug-free and helps in validating DUT output accuracy.

• **Monitor:** Observes outputs from both the DUT and Reference Model. It captures responses and forwards them to the scoreboard.

• **Scoreboard:** Compares the outputs from the DUT and the Reference Model. Any mismatch is flagged, helping in pinpointing incorrect behaviour in the DUT.

This architecture enables automation, reusability, and scalability while ensuring accurate and efficient validation. The modular design also supports future extension to incorporate new commands, data types, or operational modes with minimal changes to the testbench structure.

Working:

The ALU is designed as a combinational circuit that performs operations synchronized with the positive edge of the clock. It supports both arithmetic and logical operations based on the state of control and data signals. The following describes the working behaviour of the ALU in detail:

Reset and Enable Behaviour:

• If the asynchronous RST signal is high, all outputs of the ALU are reset to zero.

• If the Clock Enable (CE) signal is low during operation, the ERR flag is set high, and

the result output is forced to zero, indicating that the ALU is disabled.

Operation Mode and Operand Validity:

• The MODE signal (1-bit) determines the operation type:

o MODE = 1: Arithmetic operation

o MODE = 0: Logical operation

• The INP_VALID signal determines the validity of the operands:

o 00: No operand is valid

o 01: Only Operand A is valid

o 10: Only Operand B is valid

o 11: Both operands A and B are valid

Command Execution and Timing:

• All inputs must be correctly driven based on the CMD:

For operations requiring two operands (e.g., addition or subtraction), both operands must be valid (IN_VALID = 11). Any mismatch, such as only one operand being valid, will result in the ERR flag being set high.

For operations requiring one operand, only that operand and its corresponding valid bit should be driven. Incorrect IN_VALID values will also raise the ERR flag.

• Execution Latency:

For multiplication commands (CMD = 9 or 10) under MODE = 1, the result is produced after three clock cycles.

For all other operations, the result is generated after two clock cycles.

Command Restrictions and Error Handling:

• Under logical mode (MODE = 0), if the CMD value is greater than 13, it is treated as

an invalid operation, and the ERR flag is raised.

• Under arithmetic mode (MODE = 1), if the CMD value exceeds 12, the ERR flag is set.

• For logical operations with CMD = 12 or 13, if bits 4 to 7 of OPB are all set to 1, the

ERR flag is asserted.

Compare Operation:

• When CMD = 8 in MODE = 1, the ALU performs a comparison between Operand A

and Operand B.

• Based on the result:

o G is set if A > B

o L is set if A < B

o E is set if A = B

This structured behaviour ensures that the ALU handles a wide range of valid operations while robustly detecting and flagging invalid input scenarios through the ERR output.

Working of the Self-Checking Testbench:

To verify the functional correctness of the ALU design, a self-checking testbench has been implemented using a modular verification environment, as depicted in the block diagram. The testbench architecture follows a structured approach with key components that collaborate to automate and validate ALU operations against expected outcomes.

Driver:

The driver is responsible for generating a variety of test scenarios. It drives input stimuli such as operands, control signals (CMD, MODE, IN_VALID, etc.) - to both the Device Under Test (DUT) and the Reference Model. This ensures consistency in input conditions across both entities.

Testbench:

The testbench acts as the central module coordinating the signal routing and execution flow. It receives inputs from the Driver and applies them to the DUT and Reference Model simultaneously, maintaining the same environment for both.

DUT and Reference Model:

• The DUT is the actual ALU design being tested.

• The reference model is a golden model that performs the same operations as the DUT but is known to produce correct results. It serves as the baseline for comparison.

Monitor:

The monitor captures the outputs from both the DUT and the Reference Model. It collects the resulting values (such as RES, ERR, flags like COUT, OFLOW, etc.) and packages them for validation.

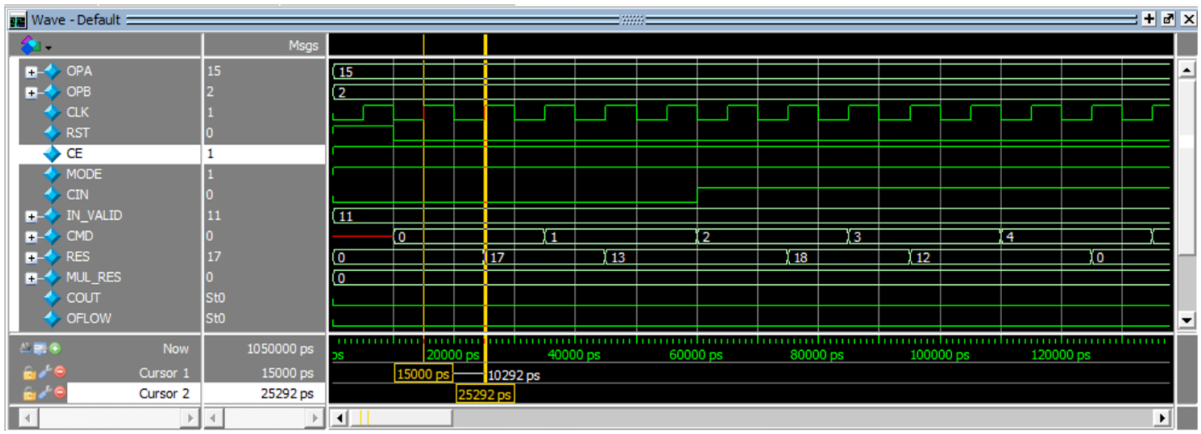
Scoreboard:

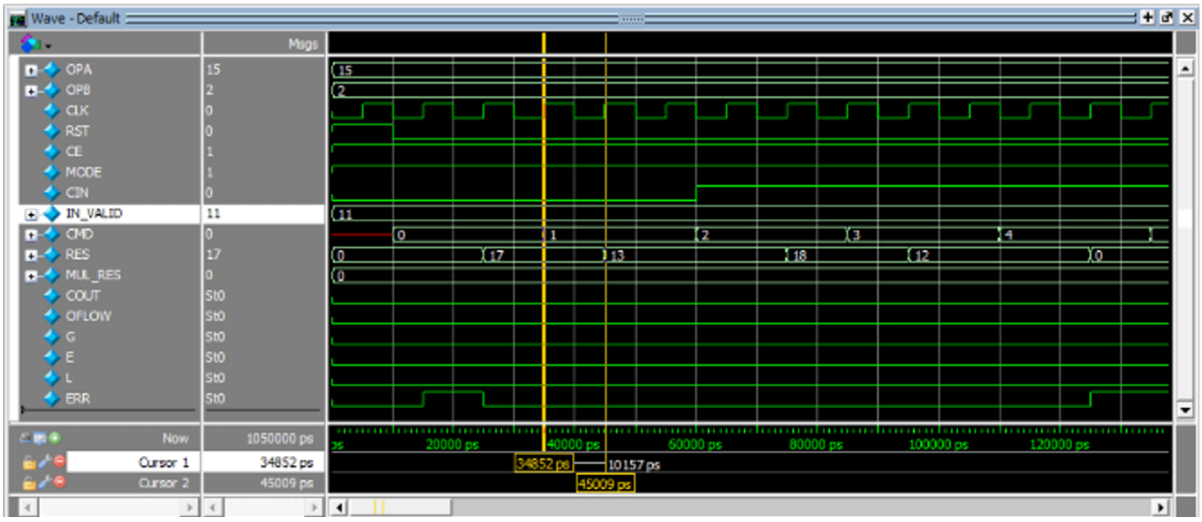
The scoreboard is the decision-making component. It compares the results captured by the Monitor from the DUT against the expected results from the Reference Model. Any mismatches are flagged as functional errors, while matches confirm correct behaviour.

This automated self-checking testbench enables comprehensive testing by running multiple test cases without manual intervention. It enhances verification efficiency, ensures correctness of the ALU design under different operation modes and edge cases, and facilitates early detection of bugs in the hardware logic.

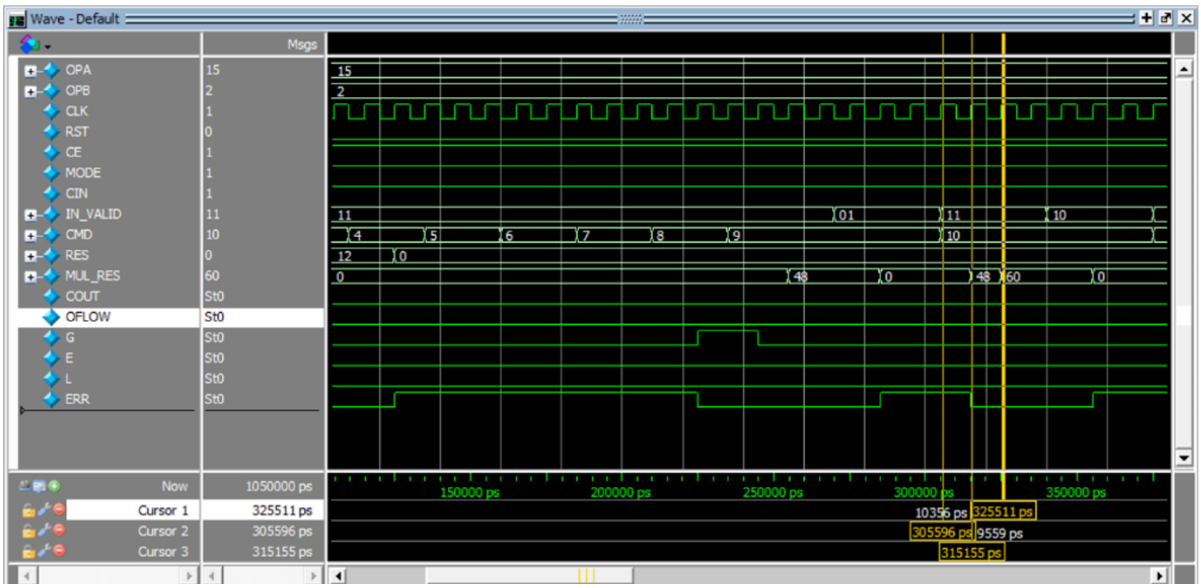
Result:

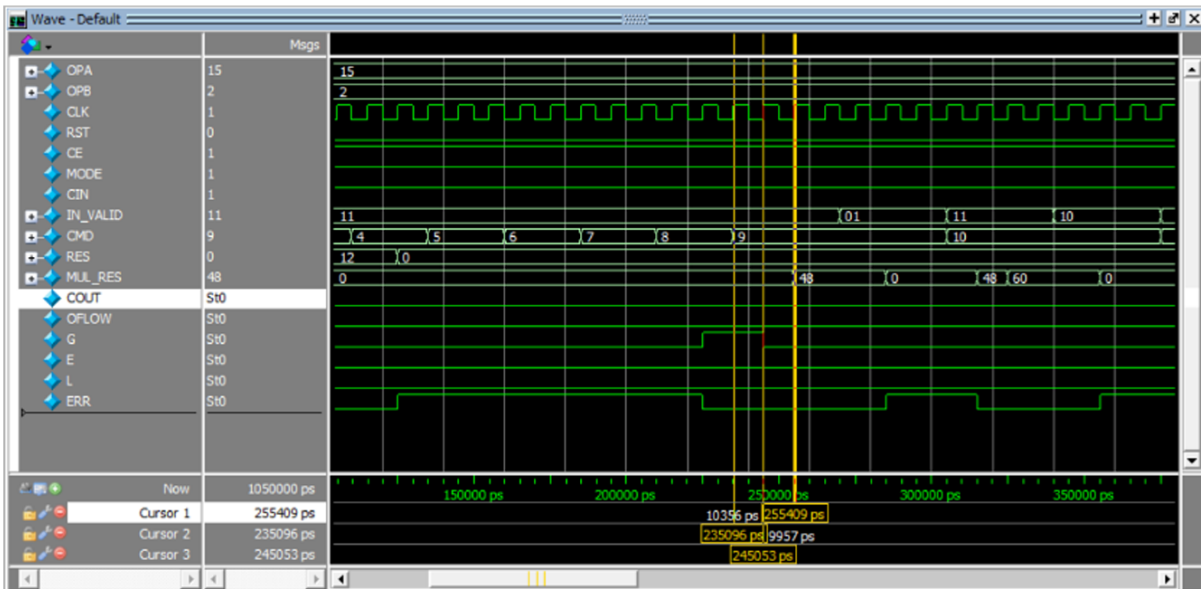
Operations that produce output after two clock cycles



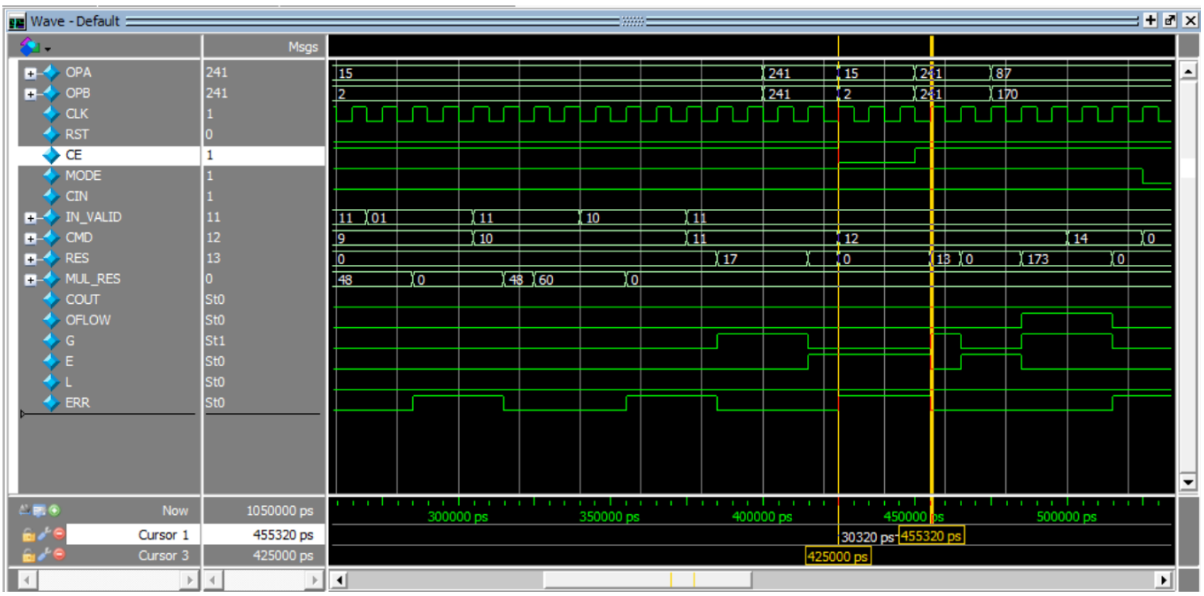


Operations that produce output after three clock cycle





ERR flag being set high when the CE is low



Conclusion:

The Verilog-based ALU design successfully meets all functional requirements, supporting a wide range of arithmetic and logical operations as specified. The implementation follows a modular, synthesizable, and testable architecture, ensuring clarity, maintainability, and hardware compatibility. A comprehensive self-checking testbench was developed to validate the design under various input scenarios, enabling automated verification through comparison with a reference model. All features and edge cases were thoroughly tested, and the functional correctness of the ALU was confirmed. Furthermore, the testbench achieved an impressive code coverage of 99.5% when analysed using QuestaSim, demonstrating the robustness and completeness of the verification process.

Future Improvement:

While the current ALU design already incorporates pipelined stages and supports wider data widths through parameterization, there are several directions for further enhancement. Future improvements may include:

• Expanding the Operation Set: Adding more complex operations such as division, square root, or trigonometric functions to increase ALU versatility.

• Formal Verification Integration: Incorporating formal verification techniques alongside simulation to mathematically prove correctness across all possible input combinations.

• Power and Area Optimization: Refining the logic to reduce dynamic power consumption and silicon area, making the design more suitable for low-power or resource-constrained environments.