ALU

INTRODUCTION

An Arithmetic Logic Unit (ALU) is a fundamental combinational circuit widely used in digital systems and processors. It is designed to perform a variety of arithmetic operations such as addition, subtraction and multiplication, as well as logical operations like AND, OR, XOR and NOT. The ALU receives two input operands along with control signals such as opcode and mode that determines which operation to execute. Based on the control input, it processes the operands and generates an output result accordingly.

In Verilog, the ALU is typically implemented using case statements or conditional structures that select the correct operation based on the control code. Along with the result, the ALU can also produce important status flags such as carry-out (from arithmetic operations), overflow (in signed operations), EGL (from comparison operations) and error (for invalid operations). These flags are useful for decision-making in higher-level modules.

OBJECTIVES

- Develop a multifunctional ALU using Verilog HDL that supports a comprehensive set of arithmetic and logic operations.
- Incorporate configurable parameters for operand size and operation codes to promote adaptability and easy integration into various digital systems.
- Implement logic for handling both signed and unsigned computations, including precise generation of carry, overflow, and comparison outputs (e.g., equal, greater than, less than).
- Maintain consistent output timing, ensuring single-cycle latency for most functions and dedicated multi-cycle support for complex operations like multiplication.
- Conduct rigorous functional testing through simulation in a Verilog environment, applying diverse test cases to ensure correctness and reliability.

ARCHITECTURE

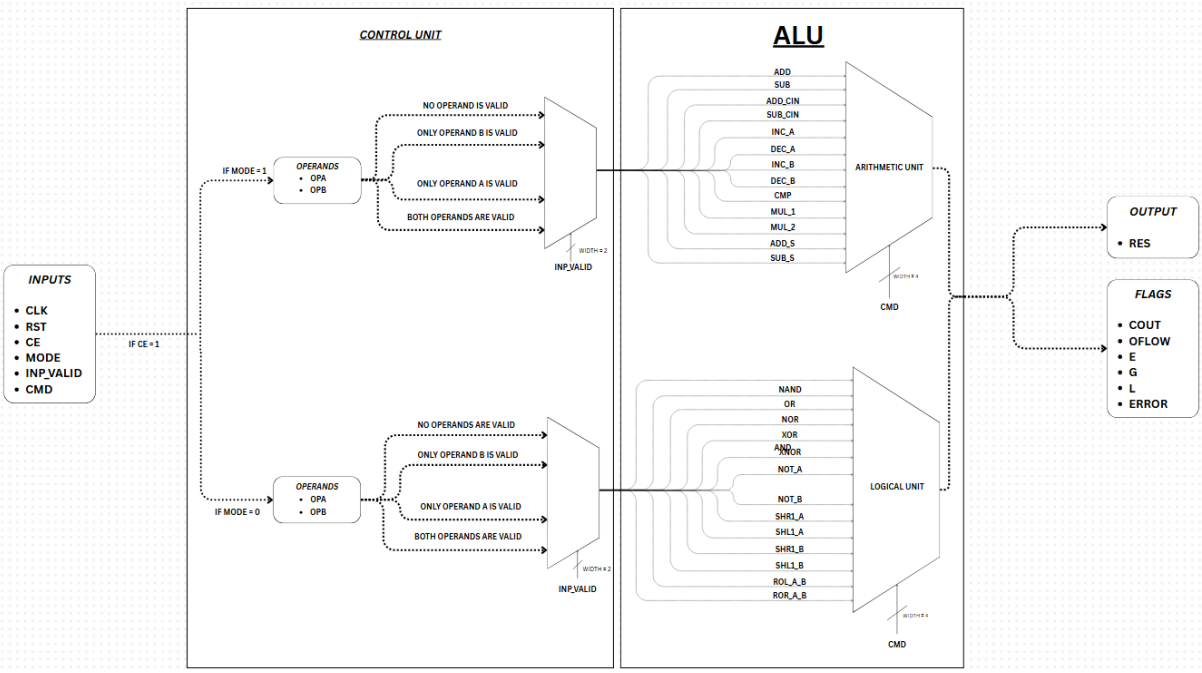
Inputs:

- CLK - This is the clock signal to the design and it is edge sensitive.

- RST - This is the active high asynchronous reset to the design. When RST is high, all outputs are reset. When RST is low i.e. RST = 0, the CE (clock enable) is checked.
- CE - This is the active high clock enable signal 1 bit. When CE = 0, there is no change in the output as the operands are not evaluated whereas when CE = 1, the MODE is checked.
- MODE – Control signal that determines if the design has to perform either Arithmetic or Logical operations. If MODE = 1, the ALU performs Arithmetic operations and if the MODE = 0, it performs Logical operations.
- OPA, OPB: Parameterised input operands
- INP_VALID - 2'b00: No operand is valid
  - 2'b01: Only operand B is valid. Operations are performed only on OPB
  - 2'b10: Only operand A is valid. Operations are performed only on OPA
  - 2'b11: Both operands are valid
- CMD: Defines the operation to be executed in conjunction with the MODE signal.
- CIN: Carry-in bit given in addition with OPA and OPB in operations such as ADD and SUB.

Outputs:

- RES - Result of ALU operation.
- COUT - Carry-out Flag.
- OFLOW - Overflow Flag.
- G, L and E - Flags indicating greater-than, less-than, or equal condition between operands (from comparisons).
- ERR – Error Flag.



WORKING

This module implements a parameterized Arithmetic Logic Unit (ALU) that performs a wide variety of arithmetic and logical operations based on control signals and input operands. The key parameters are:

- DATA_WIDTH: Width of input operands (Default value = 8)
- CMD_WIDTH: Width of the operation command (Default value = 4)

Arithmetic Operations (MODE = 1)

- If MODE is set to 1, the ALU enters arithmetic mode.
- Based on the CMD value, the following operations are performed:
    - **Addition and Subtraction**: With or without CIN and unsigned variations.
    - **Comparison** (CMP): Checks whether OPA equals, is greater than, or less than OPB, and sets flags respectively.
    - **Increment and Decrement**: Increases or decreases OPA/OPB by 1.

- **Multiplication**:
  - MUL_1: Multiplies OPA+1 and OPB+1
  - MUL_2: Multiplies (OPA >> 1) with OPB
  - These are pipelined operations that take 3 clock cycles.
- **Signed Arithmetic**: ADD_S and SUB_S work on signed values and check for overflow.

- If the CMD or INP_VALID combination is incorrect, the error flag is raised.

Logical Operations (MODE = 0)

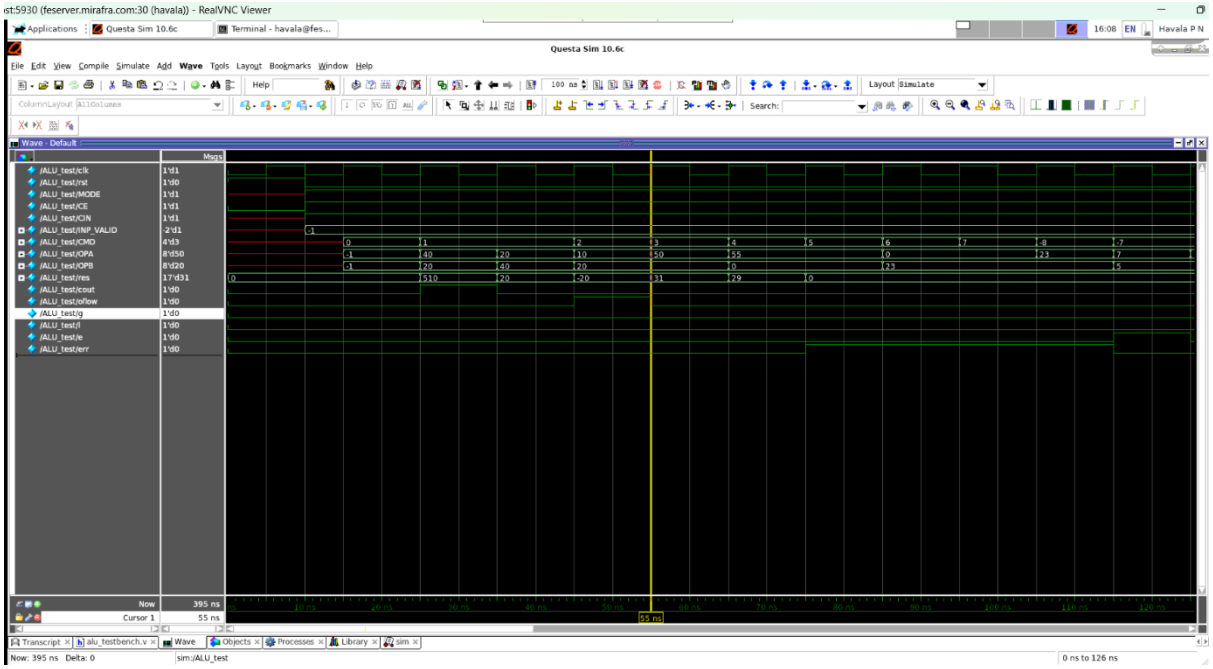
- If MODE is 0, the ALU performs bit-level operations:
  - **Bitwise operations**: AND, OR, XOR, NAND, NOR, XNOR
  - **Shifts**: Left or right shift of 1 bit for OPA or OPB
  - **Rotates**:
    - ROL_A_B (rotate left) and ROR_A_B (rotate right) rotate OPA based on OPB's value.
    - If OPB has invalid shift bits (upper nibble is non-zero), the error flag is raised.
  - **NOT operations**: Invert only OPA or OPB when only one input is valid.

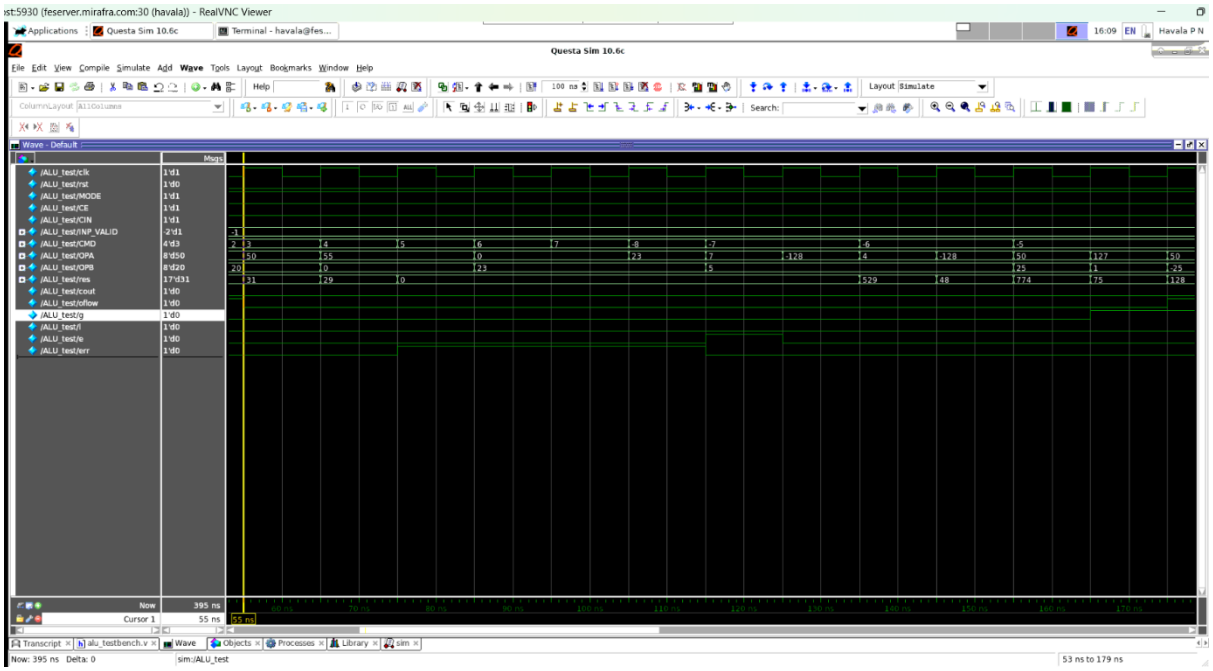
All operations produce their result with a delay of one clock cycle except multiplication operations, which require three clock cycles to produce the output due to the pipelined multiplication stages.

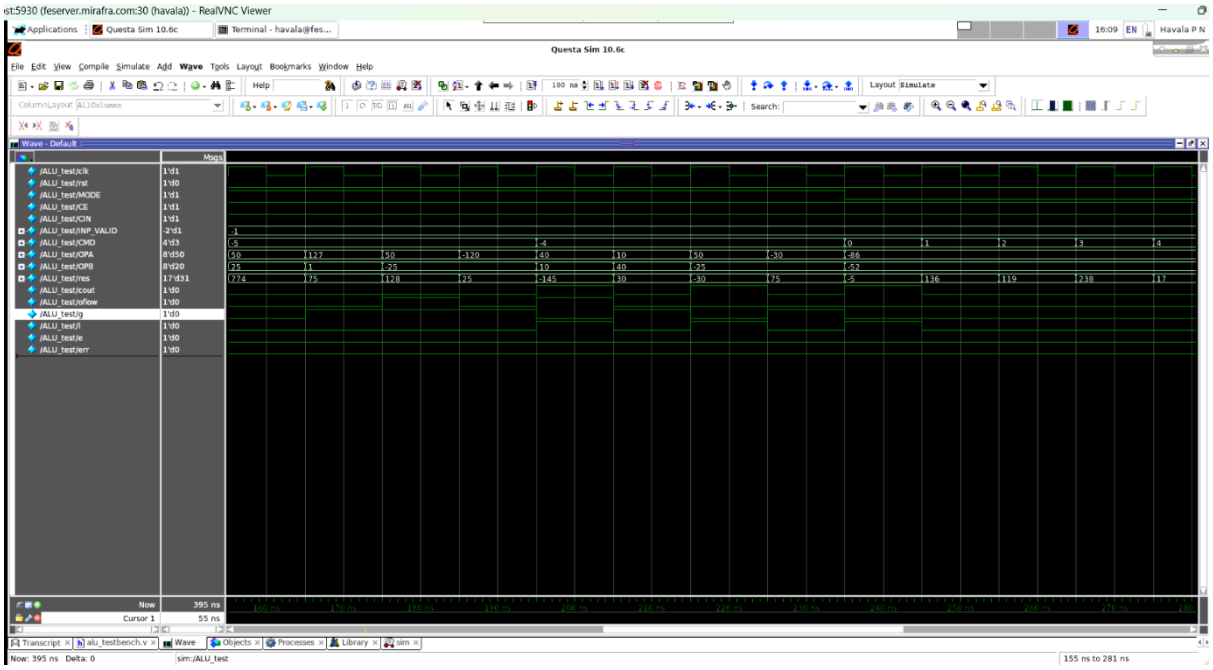
RESULT

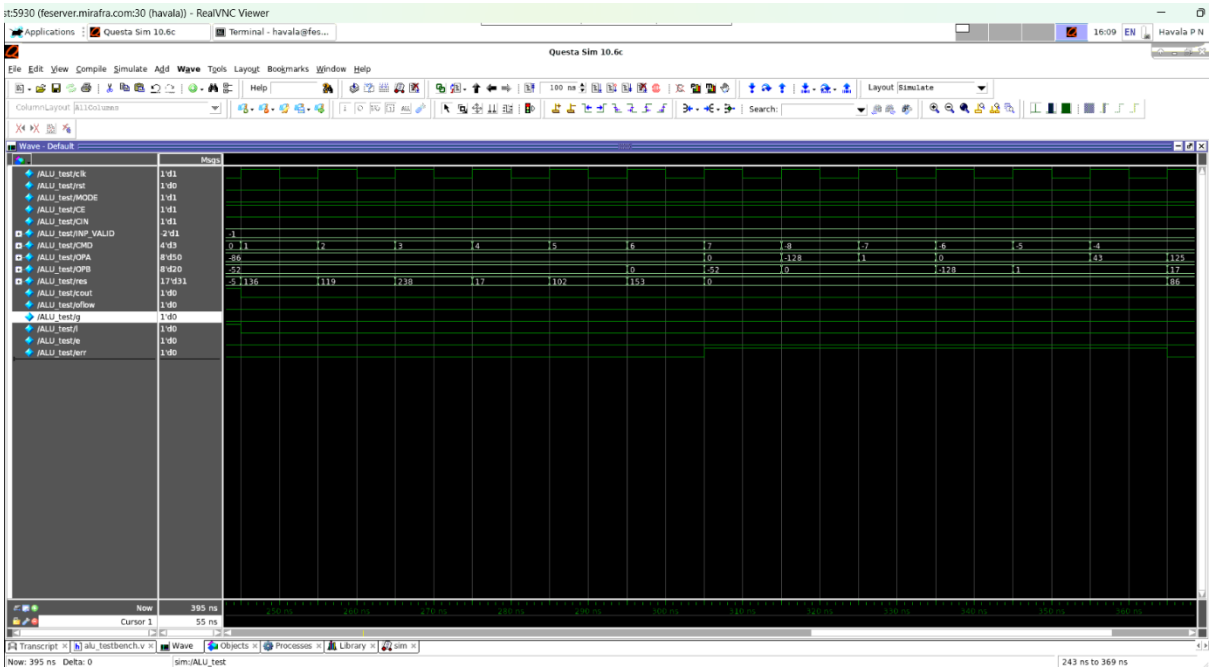
The designed ALU was tested using a Verilog testbench with various arithmetic and logical operations. The simulation confirmed that the ALU correctly performed addition, subtraction, comparison, multiplication (with pipelining), bitwise logic operations and shift/rotate functions. Flags like COUT, OFLOW, E, G, L and ERR responded correctly based on the operation and inputs. Multiplication produced results after a 3-cycle delay as expected with other operations having a delay of 1 clock-cycle. Invalid operations triggered the ERR flag,

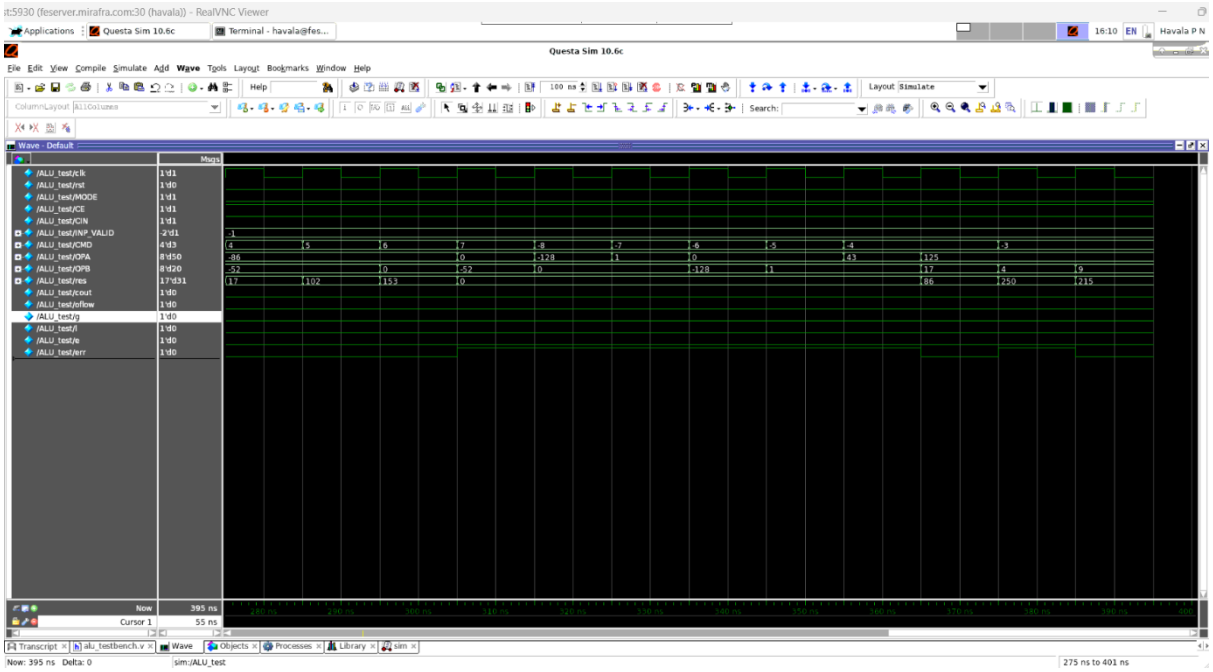
ensuring robust error handling. Overall, the ALU met all design specifications and passed most of the test cases in the simulation.

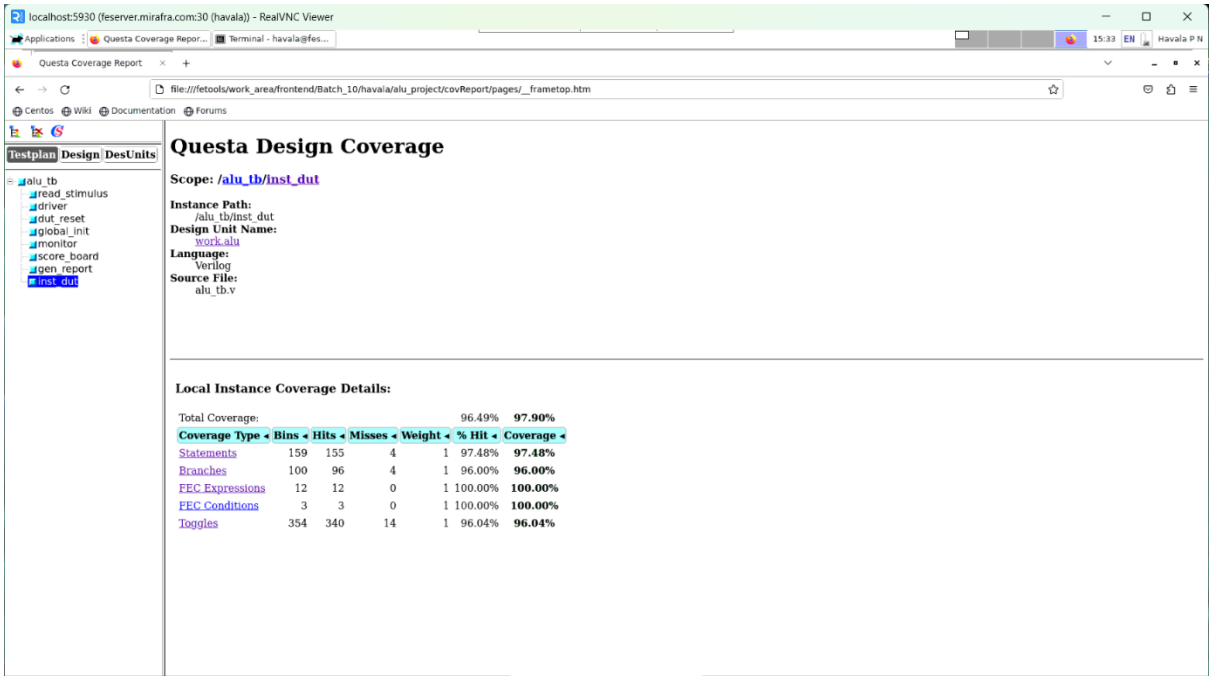





st:5930 (feserver.mirafra.com:30 (havala)) - RealVNC Viewer
Questa Sim 10.6c
Wave - Default
/ALU_test/clk
/ALU_test/rst
/ALU_test/MODE
/ALU_test/CE
/ALU_test/CIN
/ALU_test/INP_VALID
/ALU_test/CMD
/ALU_test/OPA
/ALU_test/OPB
/ALU_test/res
/ALU_test/cout
/ALU_test/oflow
/ALU_test/g
/ALU_test/l
/ALU_test/e
/ALU_test/err
Now: 395 ns Delta: 0
sim:/ALU_test
155 ns to 281 ns


st:5930 (feserver.mirafra.com:30 (havala)) - RealVNC Viewer
Questa Sim 10.6c
Wave - Default
/ALU_test/clk
/ALU_test/rst
/ALU_test/MODE
/ALU_test/CE
/ALU_test/CIN
/ALU_test/INP_VALID
/ALU_test/CMD
/ALU_test/OPA
/ALU_test/OPB
/ALU_test/res
/ALU_test/cout
/ALU_test/oflow
/ALU_test/g
/ALU_test/l
/ALU_test/e
/ALU_test/err
Now: 395 ns Delta: 0
sim:/ALU_test
243 ns to 369 ns





The overall functional code coverage obtained during simulation was **97.9%**, indicating that most of the design was exercised and verified through test cases, demonstrating strong verification quality.

CONCLUSION

The Verilog ALU meets all functional requirements with a modular, testable and synthesizable design, making it well-suited for practical hardware implementation. It supports a wide variety of arithmetic and logical operations, including addition, subtraction, comparison and bitwise

logic, while correctly handling both signed and unsigned inputs. The ALU efficiently processes control signals such as clock enable, mode selection and input validity to ensure accurate operation and output.

Status flags like carry-out, overflow, equality, greater-than, less-than and error detection are generated to provide detailed information about each operation's result. The design incorporates a pipelined multiplication operation, introducing a controlled three-clock cycle delay to balance performance and timing constraints, while all other operations complete in one clock cycle delay.

Comprehensive simulation and verification confirm that the ALU functions correctly across a wide range of input scenarios, including edge cases and error conditions.

FUTURE IMPROVEMENT

The ALU design can be improved by adding support for more complex operations such as division, modulus, and floating-point arithmetic to extend its functionality. Adding pipeline stages for various operations would improve performance and allow the ALU to work at higher speeds. Making the ALU more flexible by allowing different command sizes and easy switching between signed and unsigned operations would improve its usability. Lastly, combining the ALU with other parts like registers and control units would help build a complete and efficient processing system.