ALU PROJECT

Introduction:

The Arithmetic Logic Unit (ALU) is a central building block in digital systems and processors. It is responsible for carrying out arithmetic operations such as addition and subtraction, as well as logical operations like AND, OR, and NOT. This project presents the implementation of a parameterized ALU designed using Verilog HDL. The ALU supports a wide range of operations including arithmetic, logical, shift, and rotate instructions. The goal is to create a modular, synthesizable, and simulation-verified ALU that can serve as a reliable component in larger digital design projects. The ALU design also includes comprehensive flag generation to support conditional operations in a processor pipeline.

Objectives:

- To design and implement a fully functional Arithmetic Logic Unit (ALU) using Verilog HDL, capable of executing a wide range of arithmetic and logical operations required in digital systems and processor data paths.
- To support core arithmetic operations, including:
- Unsigned addition, subtraction, increment, and decrement,
- Extended arithmetic operations like addition/subtraction with carry/borrow,
- Special multiplications such as (OPA + 1) * (OPB + 1) and (OPA << 1) * OPB.
- To implement signed arithmetic support, specifically:
- Signed addition using two's complement format, with overflow detection when operands have the same sign but result has the opposite sign.
- Signed subtraction, with overflow detection when operands have different signs and the result’s sign differs from the minuend.
- Proper setting of additional flags such as overflow, neg, zero, and comparison flags (G, L, E) for signed operations.

Incorporate logical operations, including:

- Bitwise operations: AND, OR, XOR, NAND, NOR, XNOR.
- Unary operations: NOT on operand A or B.
- Logical shifting (left and right) and rotation (left and right), with proper boundary checks and error handling.
- To implement a robust flag-generation system that provides:
- Arithmetic status flags such as carry-out (COUT), unsigned overflow (OFLOW), signed overflow (overflow), negative result (neg), and zero result (zero).
- Comparison flags like equal (E), greater-than (G), and less-than (L) to support decision-making logic.
- An error flag (ERR) to indicate illegal operations, invalid operand combinations, or unsupported commands.
- To support input validation and operand flexibility, using a 2-bit INP_VALID signal that determines whether one or both operands are valid, ensuring the ALU handles unary and binary operations correctly and safely.
- To verify the correctness and reliability of the ALU through:
- A comprehensive Verilog testbench simulating valid, invalid, and edge-case inputs.
- Detailed waveform analysis to validate output accuracy and correct flag behaviour under all conditions.
- Corner and toggle case testing to ensure complete logic coverage.

Architecture:

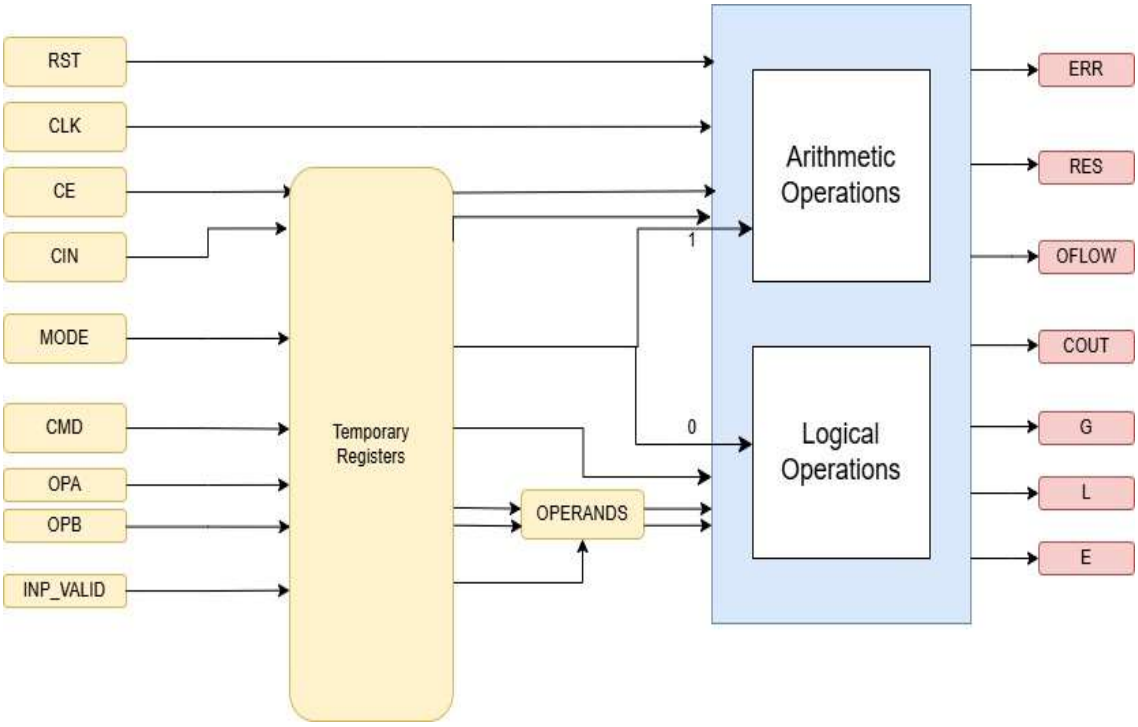

Figure 1: ALU architecture

- The ALU design includes input and output pins for operands, control signals, and result and flag outputs.
- The two input operands, OPA and OPB, are parameterized to allow flexibility in operand width.
- The ALU receives control signals such as the mode (to choose between arithmetic and logic operations), clock, reset, and enable.
- The command (CMD) signal selects the specific operation to be performed.
- Internal combinational logic processes the selected operation based on the mode and command inputs.
- The ALU generates output signals including the operation result (RES) and various status flags: carry out (COUT), overflow (OFLOW), greater (G), less (L), equal (E), and error (ERR).

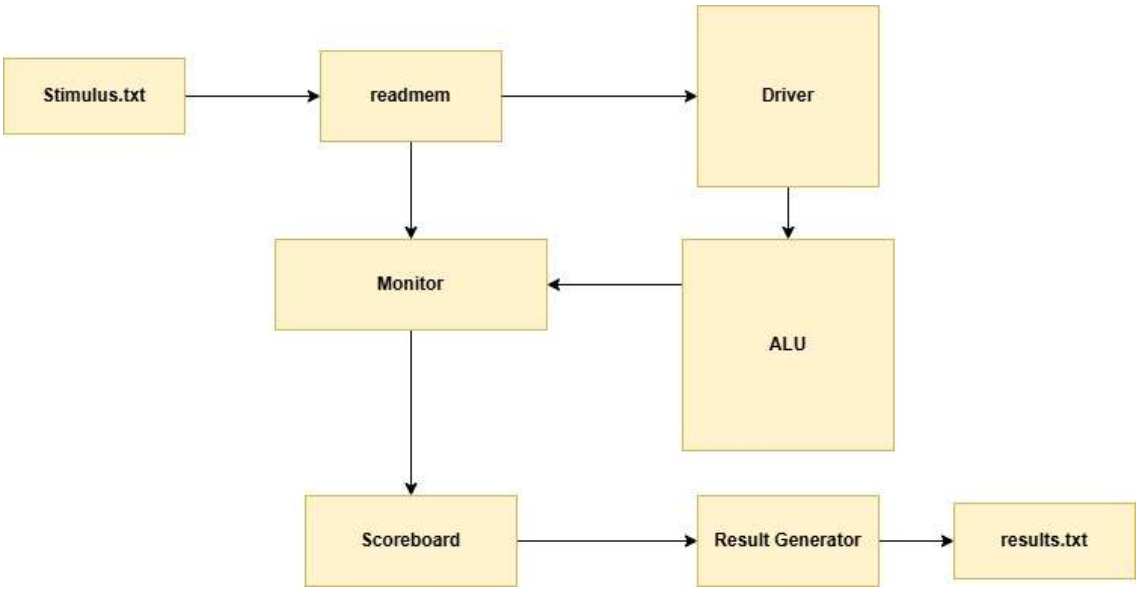

Figure 2: Testbench Architecture

The testbench architecture illustrated in Figure 2 is designed to verify the functionality of the ALU in a structured and modular way. The process begins with the Stimulus.txt file, which contains a set of predefined test cases, each specifying input values such as operands, operation codes, and control signals. These vectors are read into the simulation environment using the readmem block, which loads the data into a memory array or queue. This data is then passed to two key components: the Driver and the Monitor. The Driver is responsible for applying these inputs to the ALU under test in a synchronized manner, ensuring that each test vector is correctly issued according to the clock and enable signals.

As the ALU processes the inputs, it generates outputs such as the result and various flags (e.g., carry, overflow, zero). These outputs are continuously observed by the Monitor, which not only captures the output responses but also records the inputs applied for completeness. The Monitor sends this information to the Scoreboard, which plays a crucial role in checking the correctness of the ALU's behaviour. The Scoreboard compares the actual results with either expected values or computed golden references and determines whether each test passes or fails. These outcomes are then forwarded to the Result Generator, which formats the results and writes them into results.txt, a final report that summarizes the verification process.

Working:

The parameterized ALU module is designed to perform a variety of arithmetic and logical operations on input operands of configurable bit width N. It takes in two primary operands (OPA and OPB), a carry-in (CIN), a 4-bit command signal (CMD), a mode selector (MODE), input validity flags (INP_VALID), and standard clocking and control signals (CLK, RST, CE). All input values are latched into internal registers on the rising edge of the clock when clock enable (CE) is active. If reset (RST) is asserted, all internal registers, flags, and counters are cleared to zero to initialize the system.

The ALU operates in two distinct modes, controlled by the MODE signal. When MODE is high, the ALU performs arithmetic operations. Within this mode, the behaviour is determined by the CMD and INP_VALID signals. If both inputs are valid (INP_VALID = 2'b11), the ALU can perform standard unsigned addition and subtraction, as well as extended operations such as addition with carry, subtraction with borrow, signed addition and subtraction (with proper overflow, negative, and zero flag handling), and two multiplication variants: (OPA + 1) * (OPB + 1) and (OPA << 1) * OPB. These multiplication operations are implemented using a three-cycle delay mechanism with internal state tracking to support sequential execution. Additionally, the ALU supports comparison operations that set the greater (G), less (L), or equal (E) flags depending on the relationship between operands.

If only one operand is valid (i.e., INP_VALID = 2'b10 for OPA or 2'b01 for OPB), the ALU performs single-operand operations such as increment and decrement. These operations are bounded, and the ALU raises the ERR flag if the operand is at its maximum (for increment) or zero (for decrement), preventing overflow or underflow.

When MODE is low, the ALU switches to logical operation mode. With both inputs valid, it performs operations including AND, OR, XOR, NAND, NOR, and XNOR. It also supports bitwise rotation (ROL and ROR), where OPB specifies the number of bits to rotate OPA. If the rotation amount exceeds the operand width N, the ALU flags an error. When only one input is valid, unary logical operations such as bitwise NOT, and single-bit shift operations (left or right) are supported. These too are bounded and checked for validity.

For every operation, appropriate output flags are set to indicate result status. The ALU produces a wide result (RES) of size 2N+1 to accommodate extended operations like multiplication. Carry (COUT) and overflow (OFLOW) flags are used for unsigned operations, while signed operations set the overflow, neg, and zero flags. The error (ERR) flag is asserted when an invalid command is issued or when operand validity does not match the operation's requirements. All operations are combinational in nature, with the exception of multiplication, which is pipelined over two cycles to simulate a multicycle unit. This design ensures a clean, synthesizable, and testable ALU architecture suitable for integration into larger digital systems.

Results:

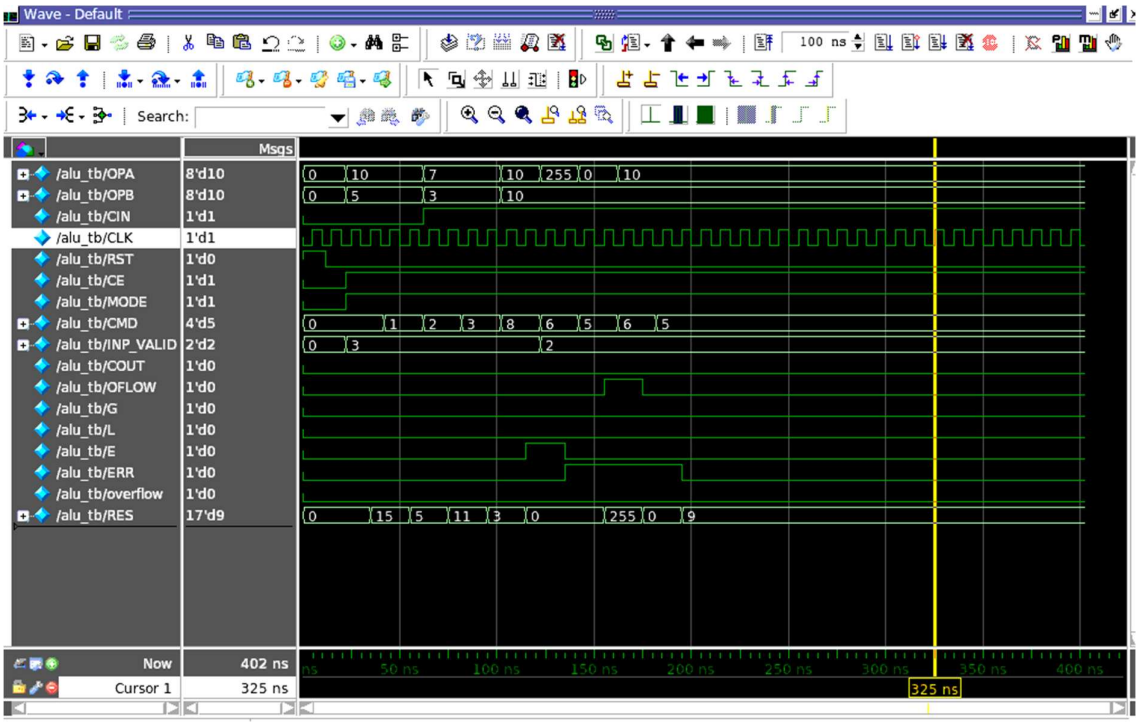

Figure 3: Arithmetic operations

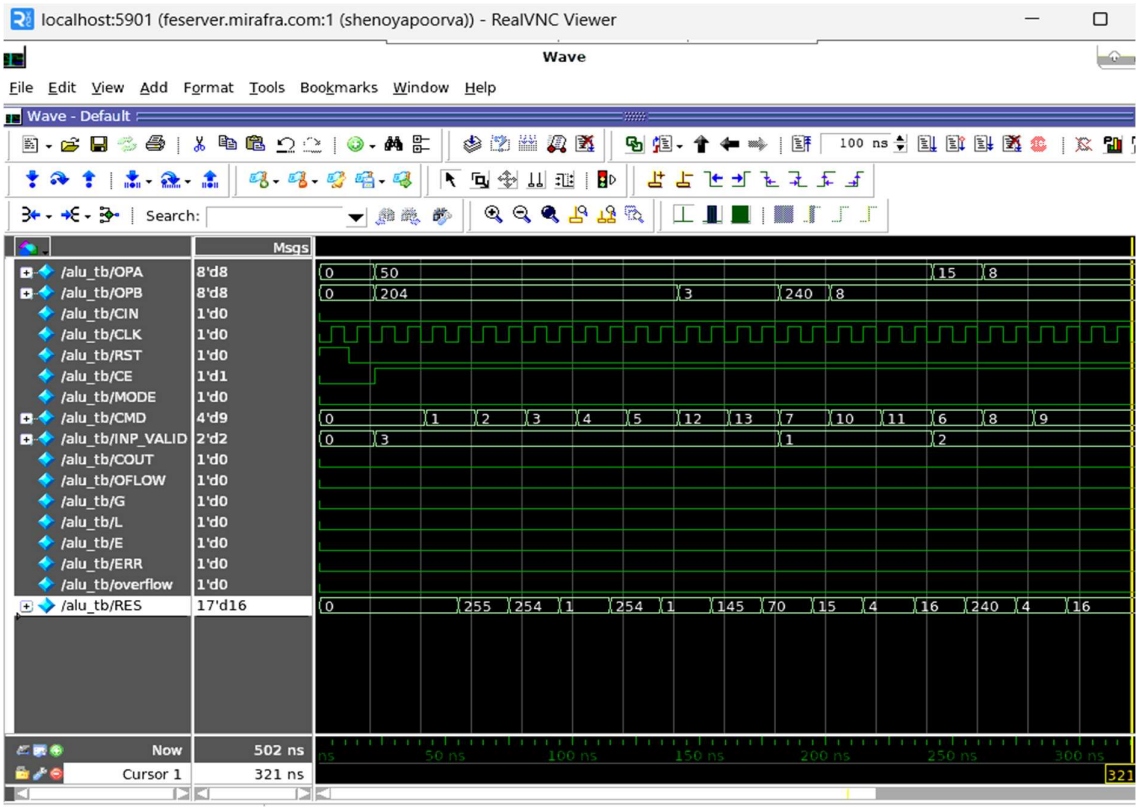

Figure 4: Logical Operations

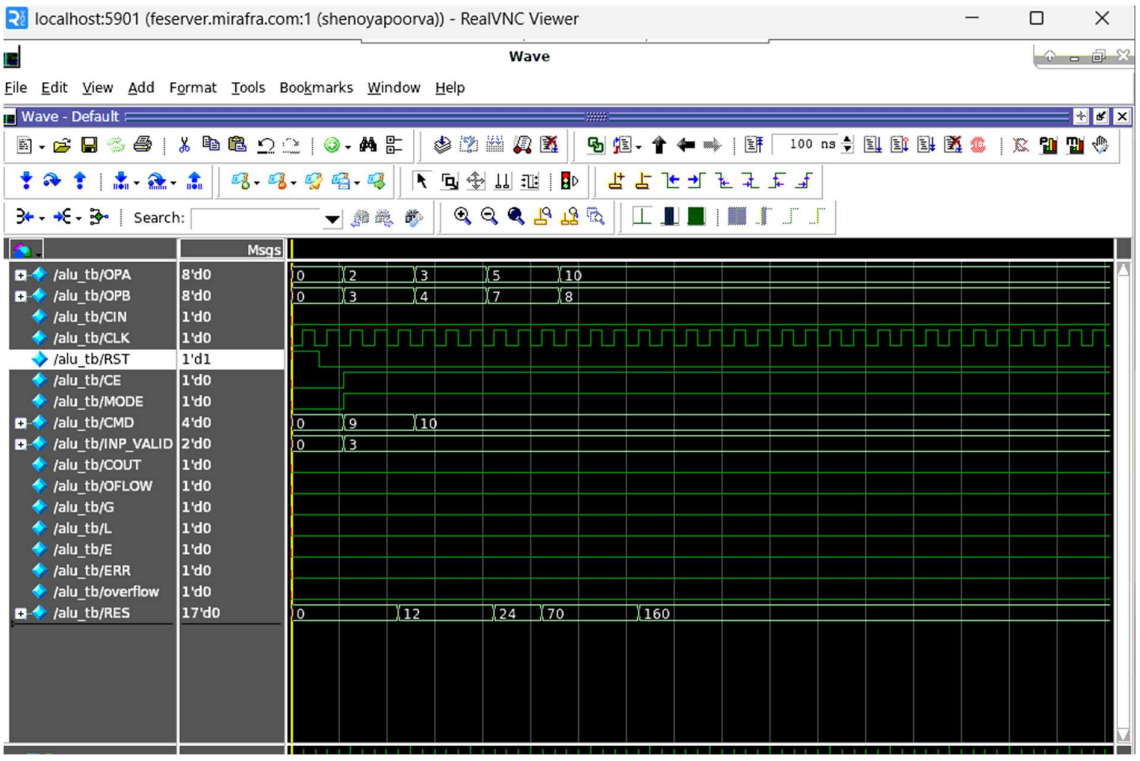

Figure 5: Multiplication operation

Conclusions:

In conclusion, the ALU was successfully implemented in Verilog and verified through simulation. It supports a comprehensive range of operations, from simple arithmetic and logical instructions to more advanced shift, rotate, and comparison functions. The design includes detailed flag generation and error detection, making it suitable for integration into more complex systems such as processors or controllers. The code is modular and parameterized, ensuring reusability and ease of expansion. Simulation results confirm the functional correctness of the design and validate that all requirements were met.

Future Improvements:

Introduce pipelining to break the ALU into multiple stages (fetch, decode, execute, write-back), enabling higher throughput and making it suitable for use in CPUs and high-performance systems

Expand data width support by parameterizing the ALU to handle 16-bit, 32-bit, or 64-bit operands, which will allow the design to scale for more complex processing needs.

Add advanced operations, such as:

- Signed/unsigned multiplication
- Division and modulo operations
- Barrel shifters and dynamic shift support
- Support floating-point arithmetic**.**