**ALU PROJECT**

**Introduction**

The Arithmetic Logic Unit (ALU) is a crucial component in digital systems and serves as the computational brain of any processor. It performs all arithmetic and logical operations necessary for program execution. This project focuses on the design, implementation, and verification of a flexible and efficient ALU using Verilog, a hardware description language widely used in digital design.

The primary goal of this project is to build a modular and synthesizable ALU that can perform a variety of operations including arithmetic (addition, subtraction, increment, etc.), logical (AND, OR, XOR), and bit-manipulation functions (shifts and rotates). Additionally, the ALU is designed to compute and output multiple flags such as carry-out (COUT), overflow (OFLOW), and comparison flags (EGL - equal, greater, less), which are essential in control and decision-making in a processor pipeline.

The design follows a combinational logic approach, meaning the output is immediately responsive to input changes without requiring clock synchronization. This makes the ALU fast and suitable for integration into combinational datapaths or as part of a larger pipelined system.

The ALU project follows a structured development process encompassing three major phases, the Design Phase, where functional specifications are translated into modular Verilog code, the Simulation Phase, where correctness is verified using testbenches, waveforms, and expected outputs, and the Analysis Phase, where performance is validated for correctness, extensibility, and synthesis readiness. The ALU accepts two 8-bit operands (OPA and OPB), a command signal (CMD) to select operations, a mode selector (MODE) to differentiate arithmetic from logical functions, and control signals such as CE (clock enable), IN\_VALID, and CIN (carry in). It produces an 16-bit result (RES) along with intelligent status flags like COUT, OFLOW, and EGL, reflecting the outcome of operations. These features make the ALU not only capable of computation but also highly testable and suitable for real-world embedded or processor-based systems. A comprehensive test plan documented in an Excel sheet ensures coverage of all functional and edge-case scenarios. Overall, the project enhances understanding of digital system design and verification while emphasizing modular development and practical implementation using Verilog.

**Objective**

• The design if flexible for the any data width, as it parameterized.

• ALU performs specific operations depending on the mode and commands which is selected.

• ALU is designed to generate the output with one clock cycle delay.

• Flags (OFLOW, COUT, ERR) are used to check the status of the obtained output.

• A testbench for the same is written in Verilog to check the proper function of the design.

**Design architecture**

The below diagram respresents the architecture of the ALU design.

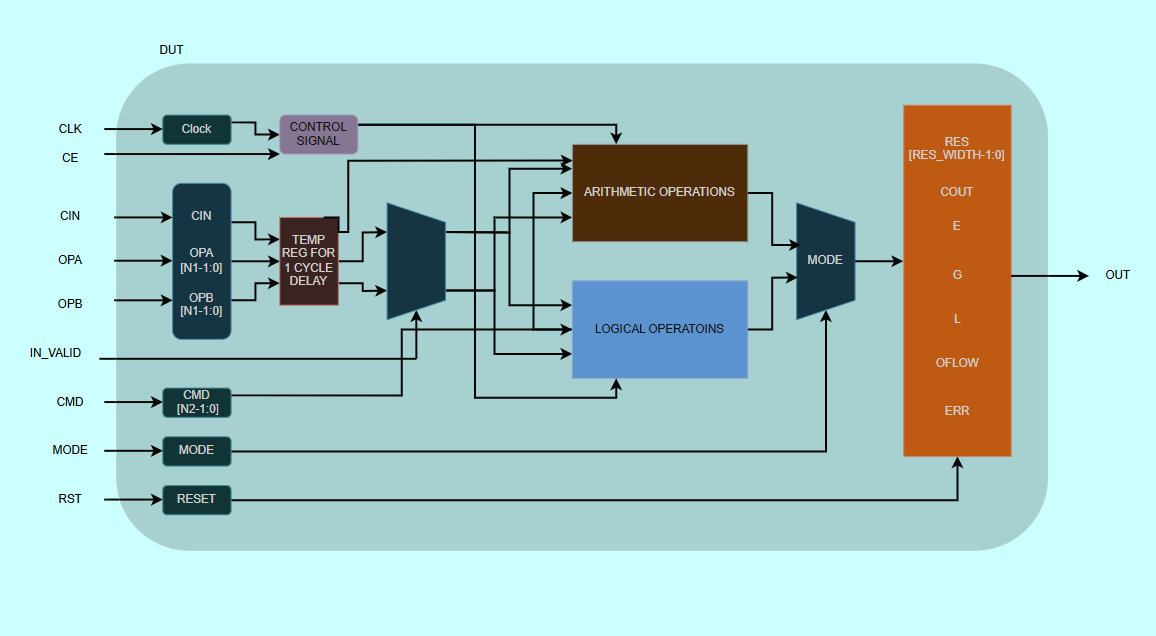


Fig 1:ALU Design architecture

Pin description:The desgin has several input and output pins, the brief description of each of these pins is given below.

INPUT PORTS:

1. CLK : The clock signal on which the design perform function during positive edge.
2. RST : ALU is designed with the asynchronous active high reset signal
3. CE : Clock enable is active high signal.
4. MODE : It is 1 bit signal. Arithmetic (MODE=1) or Logical (MODE=0) operations are performed, based on this signal.
5. IN\_VALID : It is a 2-bit input valid signal to check the validity of the input operands.
6. CMD : CMD is the command input, which tells which operation has to be performed, depending on, in which MODE it is present.
7. OPA : Parameterized input operand A.
8. OPB :Parameterized input operand B.
9. CIN : 1 bit carry in signal.

OUPUT PORTS:

1. RES : Paramterized output signal.
2. COUT : 1 bit carry-out signal used in addition/subtraction.
3. OFLOW : 1 bit overflow signal used in addition/subtraction.
4. ERR : 1 bit error signal, flag is raised to indicate errors.
5. G : 1 bit output raised if OPA is greater than OPB.
6. L : 1 bit output raised if OPA is lesser than OPB.
7. E : 1 bit output raised if OPA and OPB are equal.

Tesbench architecture:

The below diagram shows the simple test bench architecture for the ALU design.

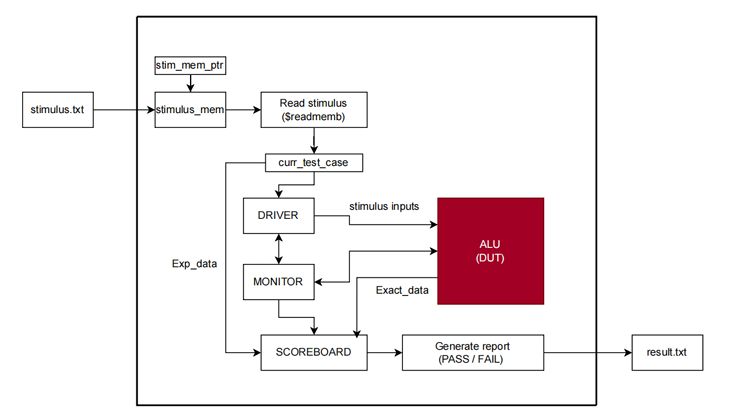


Fig 2:Testbench architecture

The testbench architecture contains blocks such as driver, monitor, scoreboard to check display the data.

1. stimulus.txt is a file containing the test case data in the format as mentioned in Fig.2.
2. Each test cases is stored in stimulus\_mem, and it is fetched accordingly.
3. Single test case (curr\_test\_case) is driven by the driver and inputs are fed into the DUT.
4. The obtained output from DUT is sent to scoreboard.
5. Based on the expected data (Exp\_data) and obtained data (Exact\_data), the report is generated, and for each feature id, the report is updated in result.txt file.
6. Monitor monitors the data and prints it.

**Working**

The ALU is clocked on the rising edge and includes an asynchronous, active‐high reset and an active‐high clock‐enable. Whenever RESET is asserted high, all outputs are immediately cleared to zero. Once RESET is released low, CE (clock‐enable) governs operation: if CE = 1, the ALU executes the selected function on the inputs; if CE = 0, it simply makes the output to zero.

A single‐bit MODE signal selects between arithmetic and logical operations. When MODE = 1, the ALU performs arithmetic functions; when MODE = 0, it carries out logical operations. Within the chosen mode, the specific operation is indicated by the 4-bit CMD opcode.

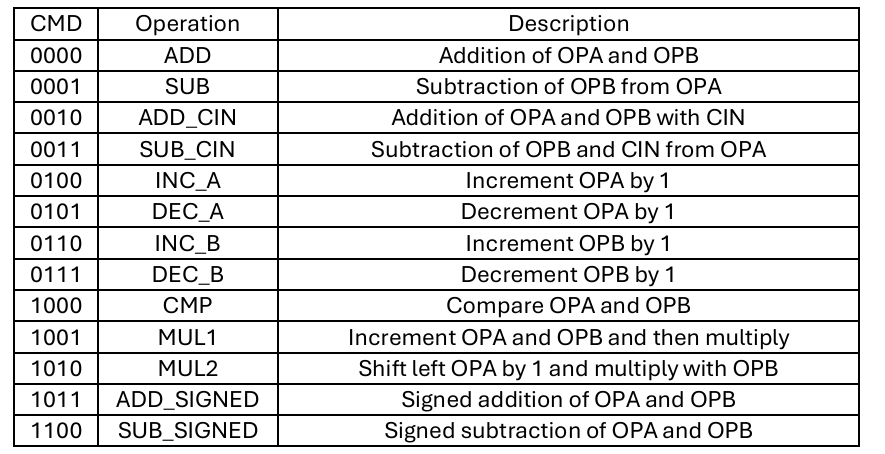
The IN\_VALID bus is two bits wide and tells the ALU which operands are valid for the next computation:

* 2’b00: neither OPA nor OPB is valid (no operation),
* 2’b01: only OPA is valid,
* 2’b10: only OPB is valid,
* 2’b11: both OPA and OPB are valid.

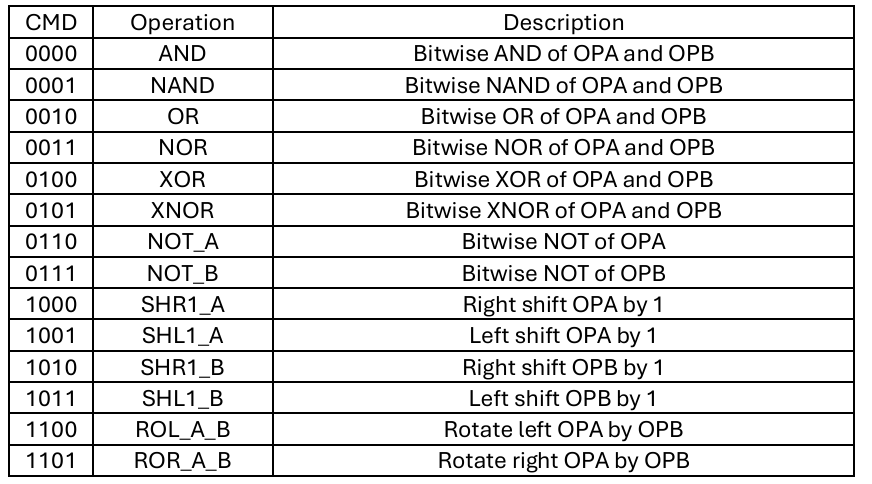
With MODE = 1 (arithmetic), the ALU can perform addition, subtraction, increment, decrement, etc., based on CMD. In logical mode (MODE = 0), it executes bitwise operations such as AND, OR, XOR, shifts, and rotates, again selected by the CMD code.

CMD is the 4-bit opcode, which tell the ALU to perform the operations as below,

In MODE 1, i.e., arithmetic operation,



In MODE 0, i.e., logical operation,



**Results**

The ALU was successfully designed and simulated using Verilog. Functional verification was performed through a comprehensive testbench that applied a variety of input combinations across all supported operations. Simulation waveforms confirmed correct execution of arithmetic and logical functions, including flag generation (Carry Out, Overflow, Greater, Equal, Less). The design responded accurately to control signals and reset conditions, validating its correctness and robustness under various scenarios.

For normal operations like arithmetic and logical the output appears at one clock cycle delay as given in the diagram below

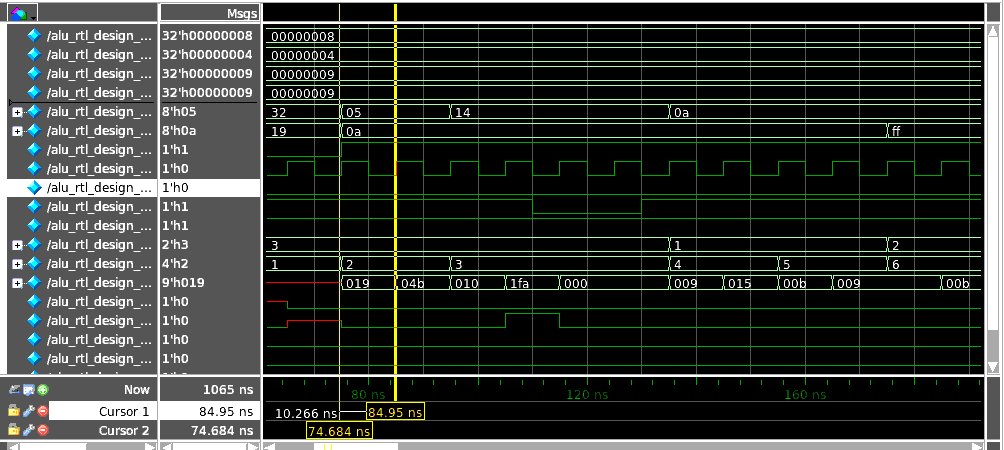


Fig 3:Waveform for normal operation

For multipication operatons the result will appear in a clock cycle delay of two clock cycles as shown below

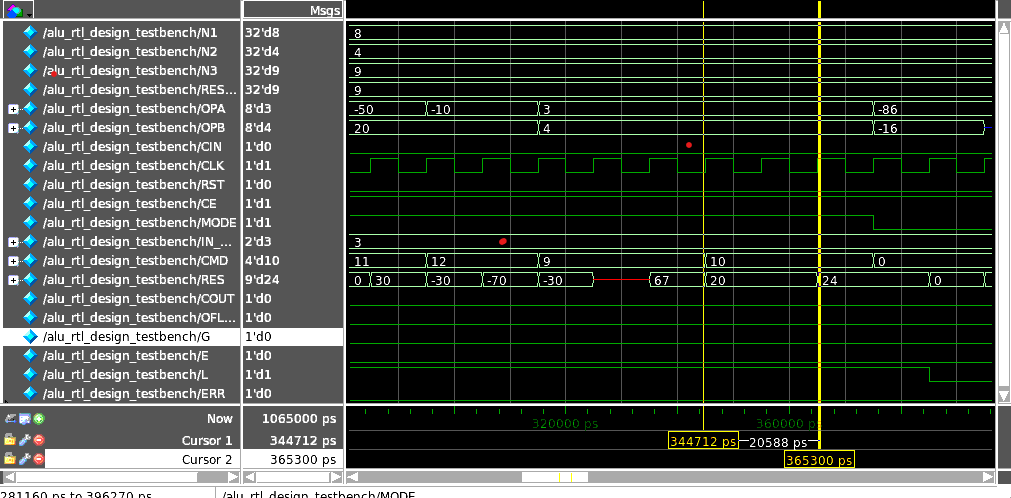


Fig 4: Waveform for the multiplication operatons

**Conclusion**

The implementation of the ALU and its verification through a structured testbench ensures that the design meets functional requirements. By handling both arithmetic and logical operations effectively, integrating status flag management, and employing a systematic verification approach, this ALU guarantees computational accuracy, reliability, and seamless integration within digital systems. The use of a stimulus driven testbench, golden reference models, and comparison techniques with a scoreboard mechanism

ensures robust validation and detection of potential errors, improving overall design efficiency.

**Future improvement**

* High-Speed Optimization: Improve architecture with pipeline execution to boost efficiency and reduce latency.
* Expanded Functionality: Add floating-point, complex number, and cryptographic operations for advanced computing.
* Automated Verification: Use AI-driven testbench generation and debugging tools to enhance accuracy and reduce development time.
* By integrating these improvements, the ALU can evolve into a more powerful, efficient, and adaptive computing unit, catering to advanced digital applications while maintaining high accuracy and performance.