ALU DESIGN DOCUMENT

Introduction

The Arithmetic Logic Unit (ALU) is a fundamental building block of any digital processing system, responsible for performing a range of arithmetic and logical operations. This report presents the design and implementation of a parameterized ALU that supports both signed and unsigned operations for operations with some control features. The ALU design integrates arithmetic functions such as addition, subtraction, and increment/decrement operations, as well as logic functions including AND, OR, XOR, NOT, and their variants. Additionally, the design supports operations such as shift and rotate (both left and right), along with conditional comparisons and flag generation.

The ALU is designed to be flexible and configurable through a clearly defined set of control and data input pins, allowing it to operate under different modes (arithmetic or logic) and command types. It includes an input validation mechanism, flag outputs (carry, overflow, error, greater-than, less-than, equal), and a timing control interface using clock, reset, and clock enable signals.

Objectives

- To design a parameterized N-bit Arithmetic Logic Unit (ALU) using Verilog that supports a wide range of arithmetic and logical operations.
- To incorporate support for parameterized command sets, input/output widths, and operation modes (arithmetic or logic).
- To validate the functionality of the ALU through comprehensive simulation using various input combinations and comparing the results against expected outputs.
- To develop a synthesizable and reusable ALU design compliant with hardware design best practices.
- To handle invalid input conditions gracefully by asserting the appropriate error flag.

Architecture

The ALU is parameterized and when there is no input parameter change operates on 8-bit wide input data, allowing it to process two 8-bit operands and generate a 16-bit result. The ALU includes the following inputs:

- clk (Clock signal)
- rst (Reset signal)
- OPA and OPB (8-bit default input operands)
- CMD   (Command or opcode for selecting the operation)
- INP_VALID (Indicates when input data is valid)
- CE (Clock Enable signal)
- Mode   (Defines the operational mode is either arithmetic or logical)
- CIN (Carry-in for arithmetic operations)

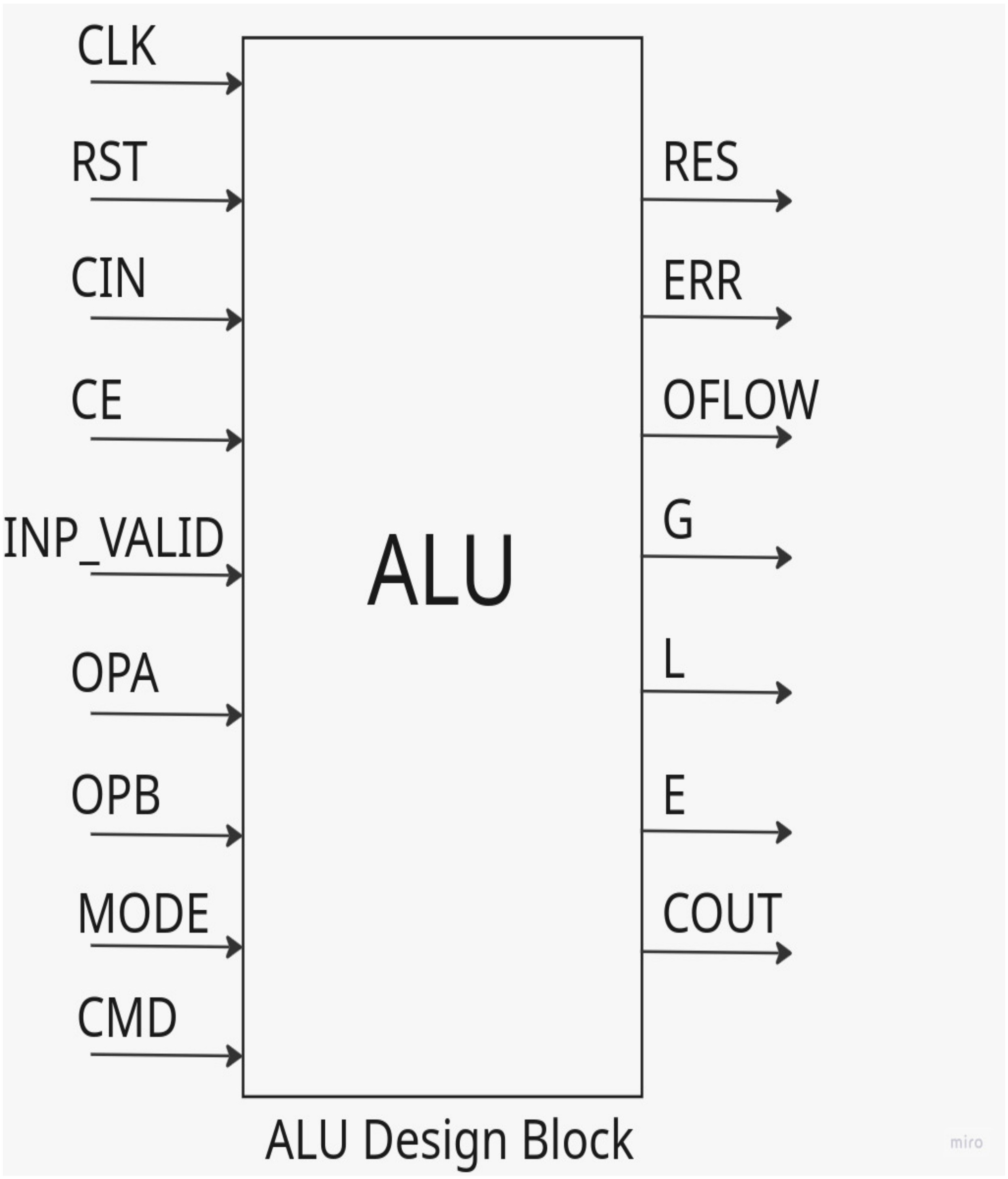
The output pins of the ALU include:

- RES (16-bit default result : parameter is given as ((2xWIDTH)-1))
- ERR (Error flag)
- OFLOW (Overflow flag)
- COUT (Carry-out flag)
- G (Greater-than flag)
- L (Less-than flag)
- E (Equal-to flag)

An input valid ensures that only valid data is captured into temporary registers, which provide stable inputs for the Arithmetic Logic Unit (ALU) during operations. It consists of 4 operations i.e,

- When the selector is 00, both the temporary register in A and B is set to zero.
- When the selector is 01, the temporary register of A stores the input operand A, and the temporary register of B is set to zero

- When the selector is 10, the temporary register of B stores the input operand B, and the temporary register of A is set to zero
- When the selector is 11, the temporary register of A stores the input operand A and the temporary register of B stores the input operand B.



Type	MODE	Operations	CMD	Descriptions
s		ADD	0	Adds OPA and OPB
		SUB	1	Subtracts OPA and OPB
		ADD_CIN	2	Adds OPA and OPB with carry input
		SUB_CIN	12	Subtracts OPA and OPB with carry input
		ADD_SIG	11	Adds signed or unsigned OPA and OPB
		SUB_SIG	5	Subtracts signed or unsigned OPA and OPB
Arithmetic	1	CMP	8	Compares OPA and OPB
		INC_A	4	Increments OPA by 1
		INC_B	5	Increments OPA by 1
		DEC_A	6	Decrement OPA by 1
		DEC_B	7	Decrement OPB by 1
		ADD_MUL	9	Add 1 to OPA and OPB and multiply
		SHIFTL_MUL	10	Shift OPA left by 1 and multiply
		AND	0	Does logical AND of OPA and OPB
		OR	2	Does logical OR of OPA and OPB
		NAND	1	Does logical NAND of OPA and OPB
		NOR	3	Does logical NOR of OPA and OPB
		XOR	4	Does logical XOR of OPA and OPB

Logical	0	X_NOR	5	Does logical XNOR of OPA and OPB
		NOT_A	6	Gives the complement of OPA
		NOT_B	7	Gives the complement of OPB
		SHR1_A	8	Shifts OPA right by 1
		SHL1_A	9	Shifts OPA left by 1
		SHR1_B	10	Shifts OPB right by 1
		SHL1_B	11	Shifts OPB left by 1
		ROL_A_B	12	Rotates OPA left by OPB LSB value
		ROR_A_B	13	Rotates OPA right by OPB LSB value

Working

The Arithmetic Logic Unit (ALU) is designed to perform a range of arithmetic and logical operations based on the control signals clk, INP_VALID, CE and rst. It accepts two primary operands (opa and opb) of configurable bit-width (WIDTH) and uses a command signal (CMD) to select the desired operation. The operation mode is controlled using the mode input: when mode is high (1), the ALU executes arithmetic operations, and when low (0), it performs logical operations. Additionally, the INP_INVALID signal indicates the validity of operands — for instance, 2'b11 means both operands are valid, while 2'b10 or 2'b01 implies that only one of the operands (opa or opb, respectively) is valid.

The ALU uses a large res register (of width 2*WIDTH + 1) to store the output of operations,

Even though only necessity of the output width is WIDTH + 1 the final output is given as 2*WIDTH as a design has to be static and cannot be changed dynamically. Operations like addition, subtraction, and multiplication are handled in arithmetic mode i.e… mode == 1.

Depending on the command, carry-out (cout) and overflow (OFLOW) flags are updated. carry-out (cout) and overflow (OFLOW) flags are updated only for arithmetic operations and OFLOW == COUT is given for the unsigned operations while for signed operations with CMD 11 and 12, in the commands 11 and 12 where signed or unsigned ADD_SIG i.e… signed addition takes place the G, L, E output flags are updated by comparing the RES with 0.

Operations SUB_CIN and ADD_CIN consider the carry-in (CIN) value. The comparison operation (CMP) outputs flags g, l, and e to indicate whether opa is greater than, less than, or equal to opb. Additional arithmetic operations such as signed addition (ADD_SIG) and signed subtraction (SUB_SIG) use two's complement arithmetic and include overflow detection logic.

In logical mode (mode = 0), the module performs bitwise logical operations such as AND, OR, NAND, NOR, XOR, and XNOR. It also supports bitwise NOT, single-bit left/right shifts, and rotate operations. The rotate operations (ROL_A_B, ROR_A_B) use a portion of the bits from opb to determine the shift amount. However, the implementation for rotate operations has inconsistencies: the second line in each rotate case incorrectly overwrites the result with a reduction OR operation, potentially indicating a bug or incomplete logic.

The ALU also handles operand invalidation conditions. If only one operand is valid (INP_INVALID is 2'b10 or 2'b01), the ALU restricts operations to unary arithmetic or logical instructions such as increment (INC_A, INC_B) and NOT/shift operations on a single operand. If an unsupported operation is attempted or both operands are invalid, the ALU sets an error flag (ERR) and clears the result.

The design is clocked synchronously with clk, and the operation is gated by the CE (Clock Enable) signal. A synchronous reset (rst) clears all internal states. The combinational block assigns the intermediate results and flags to the output ports, ensuring the output reflects the most recent operation.

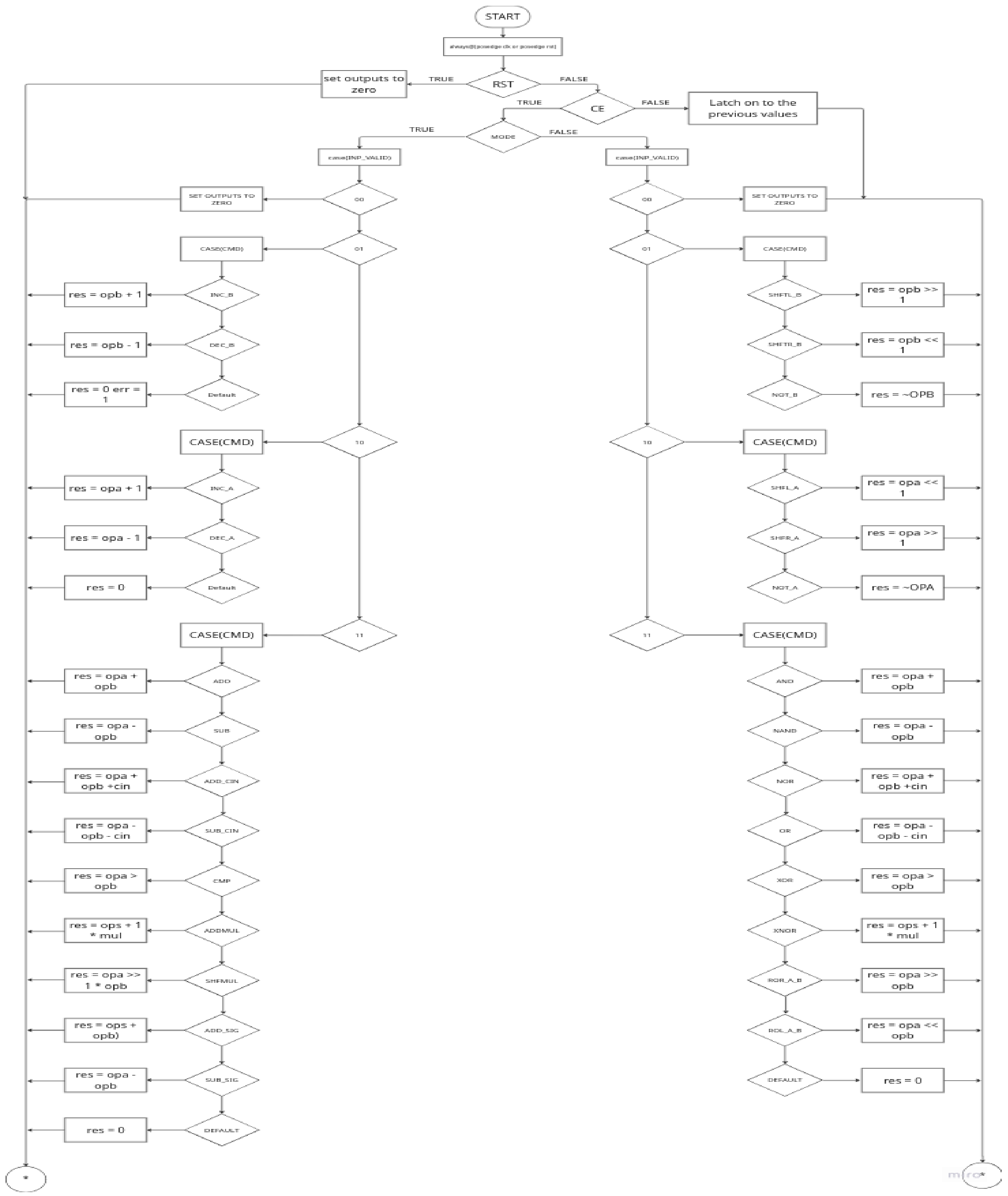
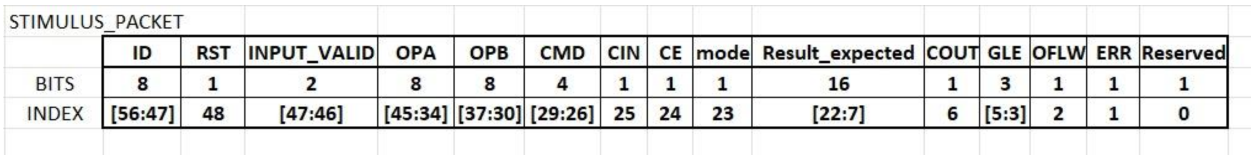


Figure 1: Flow diagram of the ALU



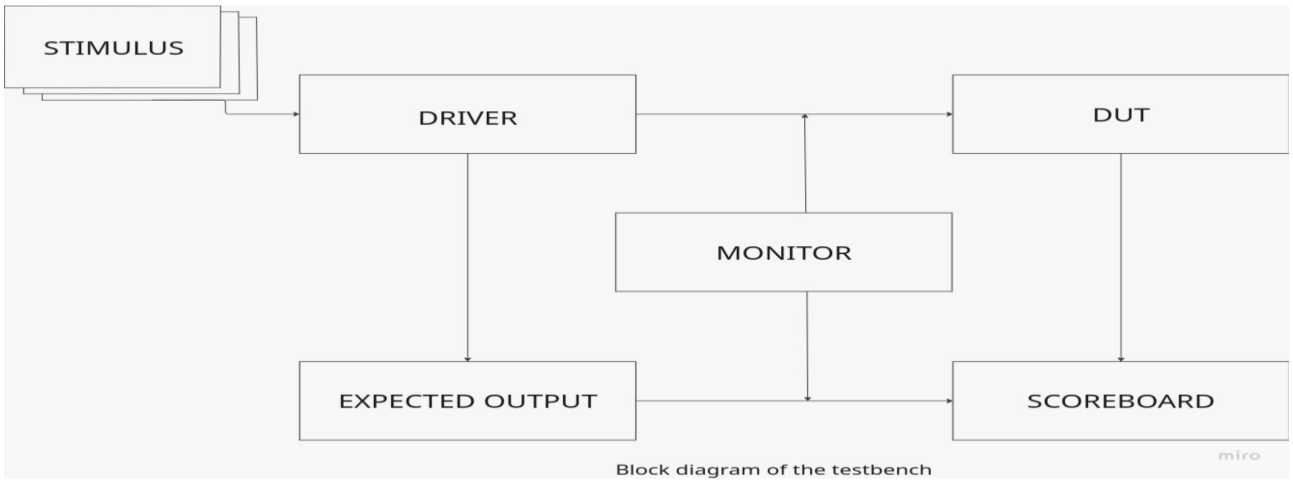
STIMULUS_PACKET

	ID	RST	INPUT_VALID	OPA	OPB	CMD	CIN	CE	mode	Result_expected	COUT	GLE	OFLW	ERR	Reserved
BITS	8	1	2	8	8	4	1	1	1	16	1	3	1	1	1
INDEX	[56:47]	48	[47:46]	[45:34]	[37:30]	[29:26]	25	24	23	[22:7]	6	[5:3]	2	1	0

Figure 2: Stimulus packet

The figure above illustrates the structure of the stimulus packet and how data is distributed and applied during simulation. It also depicts the overall flow of verification using a testbench architecture comprising three major components: the DUT, driver, monitor, and scoreboard. The driver is responsible for generating and applying valid input signals to the Design Under Test (DUT), which in this case is the alu module. These inputs are driven based on a stimulus packet, which not only provides the necessary operand and control values but also includes the expected output results and a unique identifier (ID) for each test case. This ID helps in tracking and differentiating between multiple test scenarios during simulation.

The monitor observes and captures the actual inputs sent to the DUT and the corresponding outputs generated. It ensures that all signal transitions are recorded accurately without affecting the functional behavior of the DUT. Meanwhile, the scoreboard acts as the verification module. It receives the expected outputs from the stimulus packet and compares them with the actual outputs received from the DUT via the monitor. If any mismatch is detected, it flags the test case as a failure, otherwise, it is marked as passed. The results of each comparison, along with relevant test case details, are written to an external result text file for post-simulation analysis and debugging.



Result

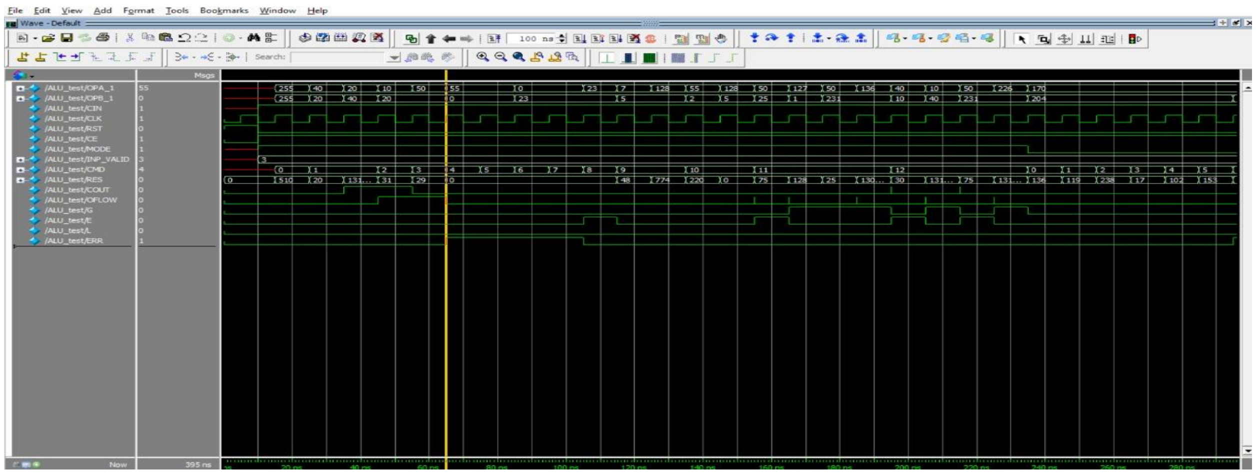


Figure 3: Arithmetic simulation output on Questasim

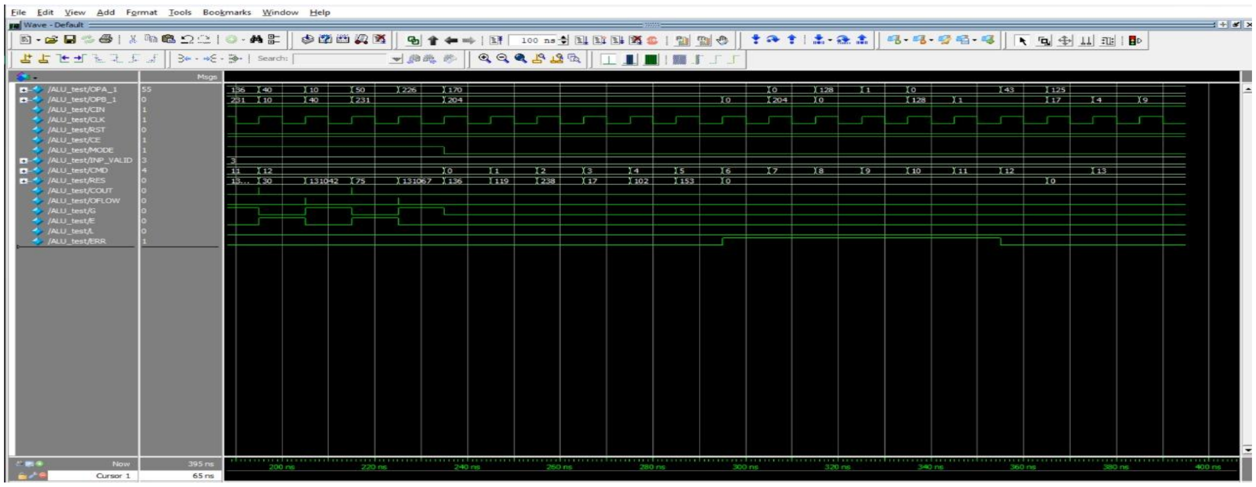


Figure 4: Logical part of the ALU simulation output

The ALU simulation was conducted using QuestaSim. A dedicated testbench was used to verify the functionality of the ALU module, and an additional testbench was employed to validate the behavior of specific ALU operations. These testbenches applied stimulus packets to the DUT (Design Under Test) through a driver, while outputs were captured and logged into a file for further analysis.

The applied stimulus contained operand values, operation commands, and expected results, enabling comprehensive validation. The simulation results showed that the ALU output (RES) was generated in the same clock cycle as the input when the INP_VALID signal was high. This immediate response contradicts the design specification, which likely intended a multi-cycle delay in the output. The cause of this behavior is most likely the use of non-blocking assignments (<=) in the RTL, which allows output updates to occur in the same simulation cycle as the input.

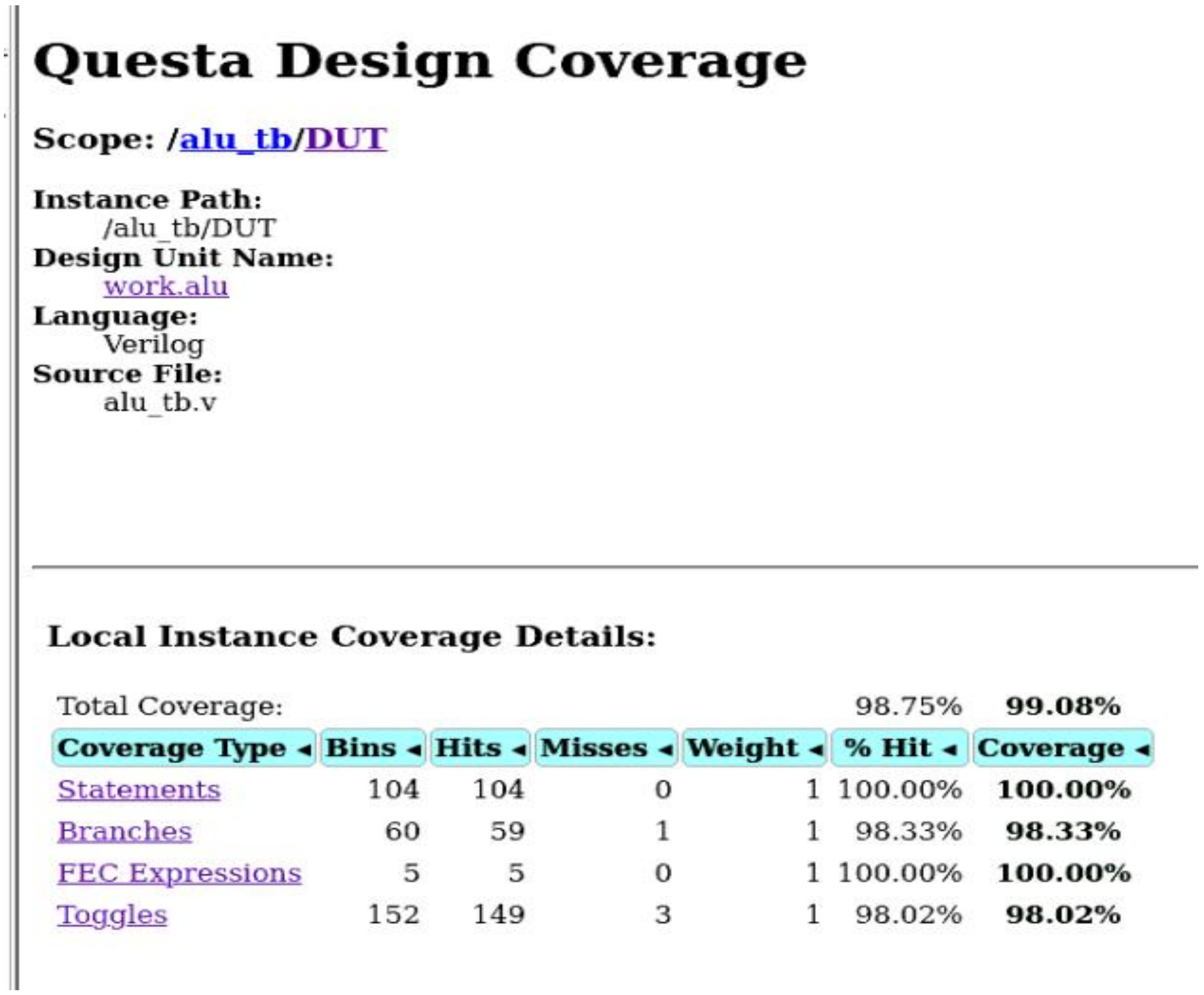
Functionally, the ALU correctly performs a variety of arithmetic and logical operations for both signed and unsigned values. Operands such as OPA_1 = 255 and OPB_1 = 0 produce accurate results (e.g., 255), and operations like subtraction (e.g., 50 - 20 = 30) also behave as expected. The status flags behave appropriately across different operations. The COUT flag is updated during unsigned addition when a carry-out occurs, and in subtraction when a borrow situation is detected. The OFLOW flag is triggered during signed overflow scenarios, specifically when results exceed the valid signed 8-bit range (-128 to 127). For other commands such as logical operations, the results lie within the range 0–255.

The G (greater than), L (less than), and E (equal to) flags are updated based on the result value (RES). These flags reflect the relationship between the result and zero, which is particularly useful in signed comparisons. The update of these flags confirms that the ALU includes post-processing logic to determine comparison outcomes after arithmetic operations. It's also observed that there is no dedicated flag or output signal for multiplication, indicating a potential area for improvement in future versions of the ALU design.

A key observation from the waveform analysis is the behavior of the ERR flag in corner cases. Specifically, when OPA is subtracted by a larger OPB value (i.e., OPA < OPB), the ERR flag is raised, or an unexpected result (garbage value) is produced due to underflow. This behavior matches the expected handling of error scenarios, although it also suggests that stronger boundary checks or saturation arithmetic could improve robustness.

The CMD input cycles through values from 0 to 13, each corresponding to a specific ALU operation such as ADD, SUB, AND, OR, XOR, and various shift operations. The results and status flags correctly reflect the outcomes of each operation, confirming that the ALU's instruction decoding and execution logic are functioning as intended.

In summary, the ALU design performs its expected arithmetic and logical operations correctly, and the testbenches validate its behavior across a wide range of input conditions. All critical flags (COUT, OFLOW, G, L, E, ERR) update appropriately according to the result values and conditions. The only deviation from the expected design behavior lies in the timing of the output, which occurs immediately rather than being delayed over multiple clock cycles. This can be addressed by redesigning the output logic to include registered stages or introducing a pipelined structure. The addition of a dedicated output flag for multiplication and improved handling of corner cases would enhance the design's completeness and reliability.



Questa Design Coverage

Scope: /alu_tb/DUT

Instance Path:
/alu_tb/DUT
Design Unit Name:
work.alu
Language:
Verilog
Source File:
alu_tb.v

Local Instance Coverage Details:

Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Total Coverage:					98.75%	**99.08%**
Statements	104	104	0	1	100.00%	**100.00%**
Branches	60	59	1	1	98.33%	**98.33%**
FEC Expressions	5	5	0	1	100.00%	**100.00%**
Toggles	152	149	3	1	98.02%	**98.02%**

Figure 5: Coverage report of the ALU

Conclusion

- Successfully designed a parameterized ALU which is completely synthesizable and lint violation free.

- Tested and functionally verified using a testbench environment
- Performed coverage and obtained the reports on the design code.

Future Improvement

- Correct and complete the design according to the time specification.
- Use blocking statements to implement the ALU.
- Optimize the design with the inclusion of the flag parameter corrections.
- Ensure all status flags are updated synchronously to avoid glitches, especially during complex operations.