ALU VERFICATION PLAN DOCUMENT

Pratiksha P Shetty
6093

Table Of Contents

1. Introduction

An Arithmetic Logic Unit is a fundamental digital circuit embedded in every processor, responsible for executing essential arithmetic and logical operations. These include tasks such as addition, subtraction, increment, decrement, and bitwise operations like AND, OR, XOR, and NOT. In modern digital systems, the ALU is not only central to performing computations but also plays a critical role in comparisons and decision-making processes.

The ALU designed in this project is parameterized and supports a variety of command codes across arithmetic and logical modes. It includes advanced features such as:

- Comparison outputs: Greater-than (G), Less-than (L), Equal-to (E)
- Overflow (OFLOW) and carry-out (COUT) detection
- Input validation through control signals
- Error signaling for invalid operations or input conditions

This project focuses on the design and development of a custom ALU testbench using SystemVerilog. The primary objective is to ensure the ALU performs all intended operations accurately while maintaining a testbench structure that is clear, modular, reusable, and simulation-friendly.

2. Verification Objectives

1. Functionality Check:
   - Verify all arithmetic operations.
   - Verify all logical operations.
   - Confirm correct behaviour under different CMD and MODE combinations.
2. Flag and Output Verification:
   - Carry-out: Validate correct generation in arithmetic operations.
   - Overflow (OFLOW): Detect and verify overflow.
   - Validate the generation of comparison outputs: G (greater), L (less), E (equal).
   - Ensure outputs reflect accurate status based on operand values.
3. Control Signal Behaviour:
   - Verify the effect of INP_VALID, CE, and RST on operation execution.
   - Check whether the ALU holds or updates results based on control signals.
4. Parameterization:
   - Confirm correct behaviour across different parameter settings.
   - Verify flexibility and consistency of operation with varying configurations.
5. Input Validation:
   - Validate proper handling of input values and edge cases.
   - Test with invalid control codes or unexpected combinations.
6. Timing and Sequencing:
   - Ensure outputs are generated correctly with respect to clock cycles.
   - Check synchronous reset behaviour and proper initialization.
7. Coverage:
   - All command codes, modes, and input patterns are exercised.
   - All branches, conditions, and paths in the ALU RTL are fully verified.
   - Cross coverage is collected for CMD vs MODE combinations to detect corner-case gaps.

3. DUT Interfaces

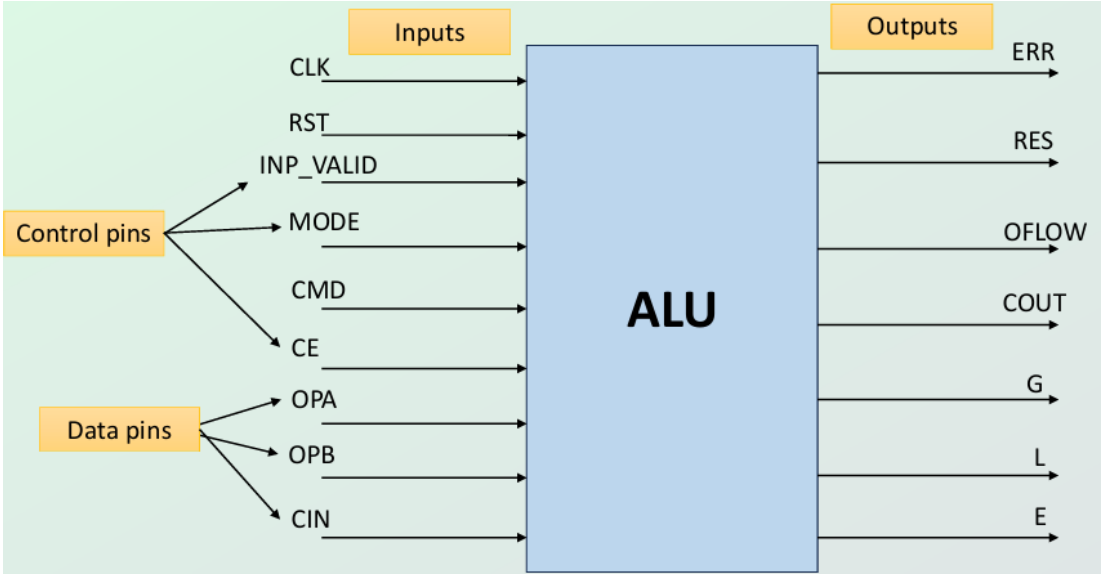


Fig 1: ALU Functional Interface with Control and Data Pins

SI.NO	PIN NAME	DIRETION	NO OF BITS	FUNCTION
1	OPA	INPUT		Parameterized operand 1
2	OPB	INPUT		Parameterized operand 2
3	CIN	INPUT	1	Active high carry in input signal
4	CLK	INPUT	1	Clock signal to the design and its edge sensitive
5	CE	INPUT	1	Active high clock enable signal
6	RST	INPUT	1	Active high asynchronous reset to the design
7	MODE	INPUT	1	
8	INP_VALID	INPUT	2	Operands are valid as per below table 00: No operand is valid 01: Operand A is valid 10: Operand B is valid 11: Operand A and B bot
9	CMD	INPUT		Parametrized (4-bit default) Arithmetic Commands CMD = 0 :ADD 1: SUB 2: ADD_CIN 3: SUB_CIN 4 :INC_A 5:DEC_A 6: INC_B 7: DEC_B, 8: CMP, 9: oprand A and B both incr by 1 and then multiplication performed.

				10: Operand A left shift by 1 and then multiply with B. Logical Commands CMD = 0: AND, 1: NAND 2:OR 3: NOR 4: XOR 5: XNOR 6: NOT_A 7: NOT_B 8: SHR1_A 9: SHL1_A 10: SHR1_B 11: SHL1_B 12: if operandB [7:4] any bit is 1 then its error. 13: if operandB [7:4] any bit is 1 then its error.
10	RES	OUT		This is the total parameterized plus 1-bit result of the instruction performed by the ALU.
11	OFLOW	OUT	1	indicates an output overflow, during Addition/Subtraction
12	COUT	OUT	1	This is the carry out signal of 1-bit, during Addition/Subtraction
13	G	OUT	1	This is the comparator output of 1-bit, which indicates that the value of OPA is lesser than the value of OPB
14	L	OUT	1	This is the comparator output of 1-bit, which indicates that the value of OPA is equal to the value of OPB
15	E	OUT	1	This is the comparator output of 1-bit, which indicates that the value of OPA is equal to the value of OPB
16	ERR	OUT	1	When Cmd is selected as 12 or 13 and mode is logical operation, if 4th ,5th ,6th and 7th bit of OPB are 1, then ERR bit will be 1 else it is high impedance.

4. Testbench Architecture

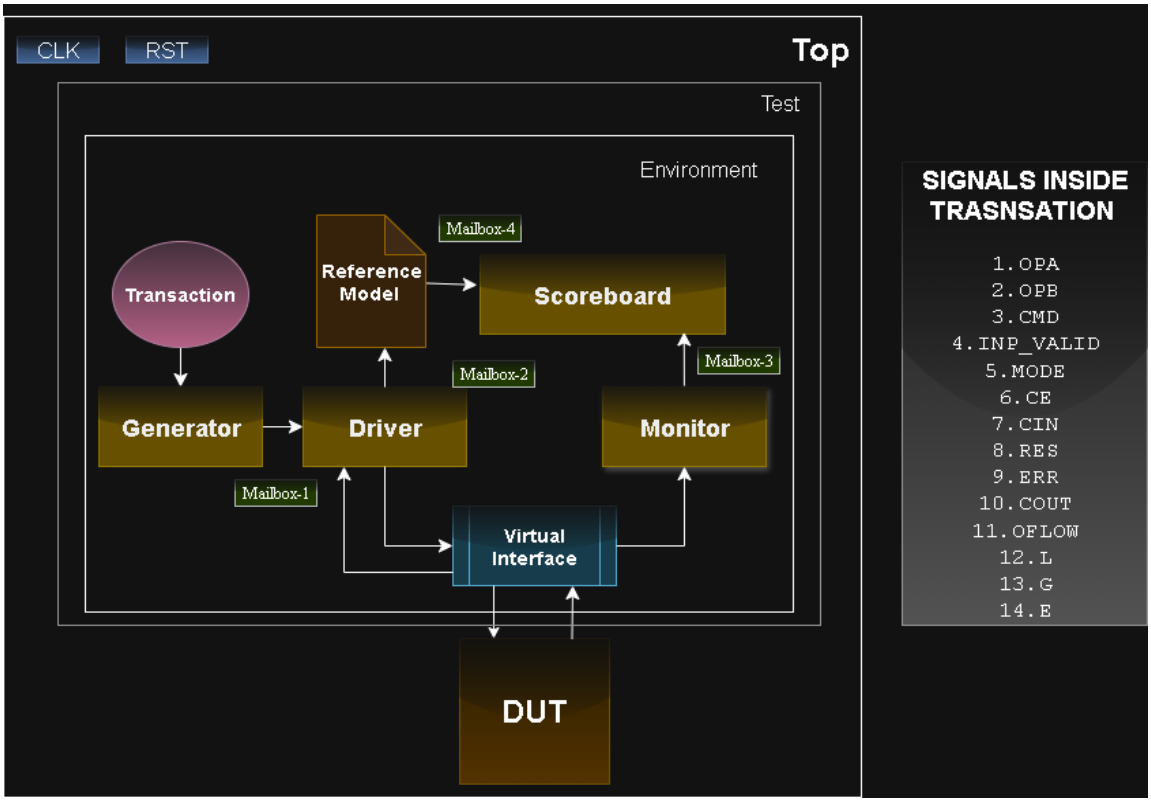
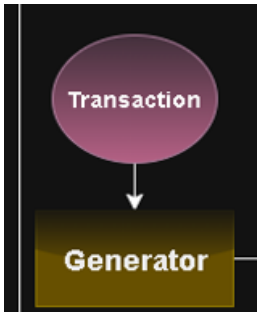


Fig 2: Component-Level Architecture of ALU Verification Environment

Detailed explanation of each component:

1. *TRANSACTION:*



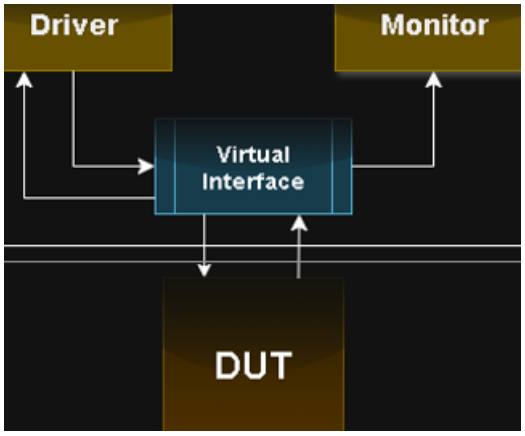
The Transaction component encapsulates all input and output signals required to stimulate and observe the ALU. Each transaction includes operands, control signals, and expected result fields, enabling consistent handling of stimuli across components like the generator, driver, monitor, and scoreboard.

These include OPA and OPB as the primary operand inputs, and CMD to specify the operation to be performed. Control signals such as INP_VALID indicate when the

inputs are valid, MODE determines whether the operation is arithmetic or logical, CE enables clock-based evaluation, and CIN represents the carry in bit used in certain arithmetic operations which are randomized. On the output side, the transaction captures RES for the operation result, ERR for error detection, COUT for carry-out, and OFLOW for overflow indication. The comparison flags L, G, and E reflect less-than, greater-than, and equal conditions respectively, which are not randomized.

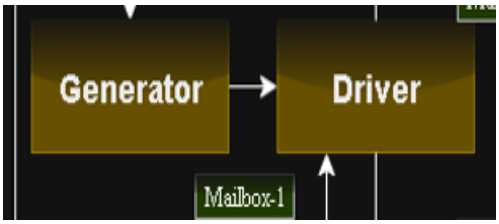
The copy() function in the transaction class is used to create an exact duplicate of a transaction object, allowing safe transfer of data between components without unintentional modification of the original.

2. *INTERFACE:*



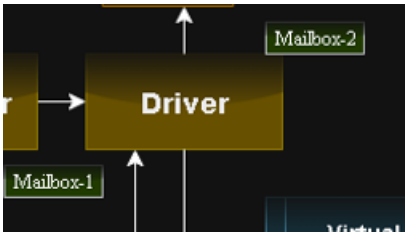
The virtual interface serves as a communication bridge between the testbench and the ALU design under test. It encapsulates all the input and output signals of the DUT into a single construct, enabling efficient access and manipulation from within the testbench. The interface includes inputs such as OPA, OPB, CMD, MODE, CIN, CE, INP_VALID, along with the clock and reset signals. It also carries outputs like RES, ERR, COUT, OFLOW, G, L, and E. To ensure synchronized interaction, a clocking block is defined within the interface, allowing components like the driver and monitor to drive or sample signals at specific clock edges.

3. *GENERATOR:*



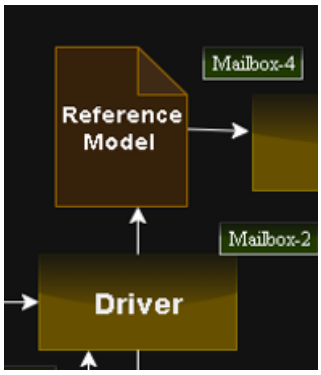
The Generator is responsible for creating stimulus for the ALU by producing a sequence of transaction objects containing randomized or constrained random values. These transactions represent different input scenarios, covering a wide range of arithmetic and logical operations, boundary cases, and invalid conditions. Once a transaction is generated, it is passed to the Driver through a mailbox (1) for execution. The Generator plays a crucial role in achieving thorough functional coverage by systematically or randomly exploring various combinations of operands, command codes, modes, and control signals.

4. *DRIVER:*



The Driver receives transaction objects from the generator using the mailbox (1) and translates them into pin-level signal activity by driving the corresponding inputs onto the DUT through the virtual interface. It synchronizes with the clock to apply signals such as OPA, OPB, CMD, MODE, CIN, CE, and INP_VALID at the appropriate times. Has 16 clock cycles wait mechanism. The driver forwards a copy of the same transaction to the Reference Model using a mailbox (2), ensuring both the DUT and the model operate on identical input data.

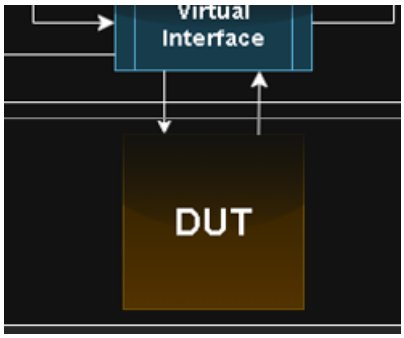
5. *REFERENCE MODEL:*



The Reference Model is a high-level behavioural implementation of the ALU that generates expected outputs based on the same inputs provided to the DUT. It receives a copy of each transaction from the driver and performs the corresponding operation

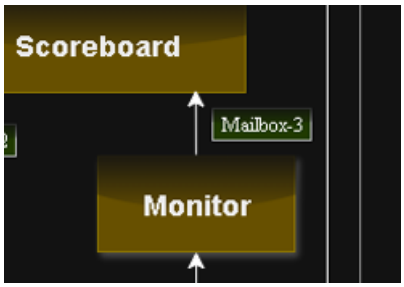
using SystemVerilog arithmetic and logical operators. To accurately mirror the timing behaviour of the DUT especially for multi cycle operations like multiplication the reference model also includes a 16-cycle delay mechanism, ensuring that output generation aligns with when the DUT is expected to produce results. The model computes outputs such as RES, COUT, OFLOW, ERR, and comparison flags - G, L, E, then sends them to the Scoreboard via a mailbox (4).

6. *DUT:*



The Design Under Test is the parameterized Arithmetic Logic Unit being verified. It accepts a range of inputs including operands, operation command, mode selection, control signals, and carry-in. The ALU produces outputs such as the result, error flag, carry-out, overflow flag, and comparison results. Designed to support both arithmetic and logical operations, the DUT may require multiple clock cycles to complete certain operations like multiplication. The DUT is instantiated in the top module and is driven by stimulus from the testbench through a virtual interface, while its outputs are monitored and validated against a reference model to ensure functional correctness.

7. *MONITOR:*



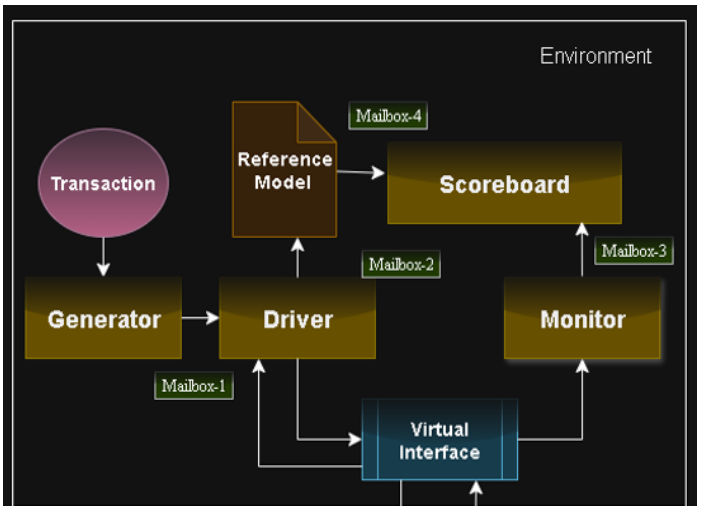
The Monitor observes the outputs of the DUT through the virtual interface without driving any signals. It observes the relevant outputs such as RES, ERR, COUT, OFLOW, G, L, and E, at appropriate clock edges and packages them into a transaction object. This observed transaction is then sent to the Scoreboard via a mailbox (3) for

comparison against the expected results generated by the reference model. The monitor ensures accurate capture of DUT behaviour during simulation while maintaining signal integrity. It plays a crucial role in validating output correctness and collecting coverage data.

8. *SCOREBOARD:*

The Scoreboard is responsible for checking the functional correctness of the ALU by comparing the DUT outputs - which are received from the monitor via mailbox (3) with the expected outputs which are received from the reference model via mailbox (4). It performs field-by-field comparisons on signals such as RES, COUT, OFLOW, ERR, G, L, and E. Any mismatch is flagged as an error, while matching results are logged as a pass. The scoreboard provides detailed feedback, helping to identify functional bugs and ensuring that the ALU behaves as specified under all tested conditions.

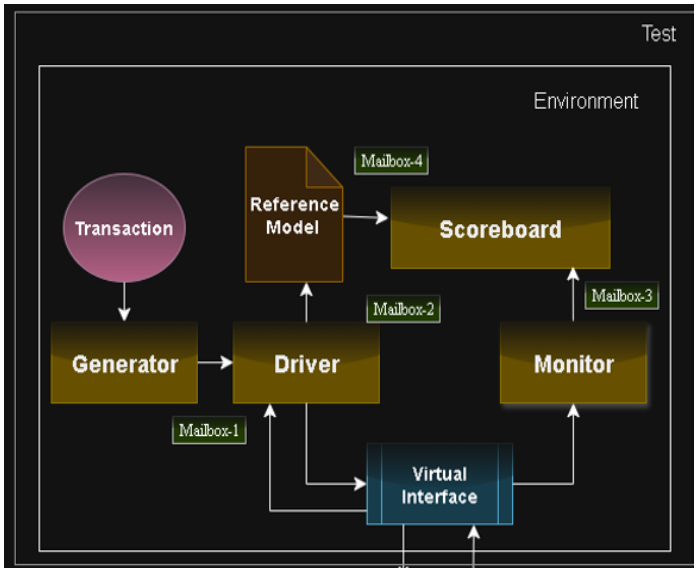
9. *ENVIRONMENT:*



The Environment serves as a container that integrates all key components of the testbench, including the Generator, Driver, Monitor, Reference Model, and Scoreboard. It sets up the interconnections between these components using mailboxes for communication and binds them to the virtual interface for synchronized access to the

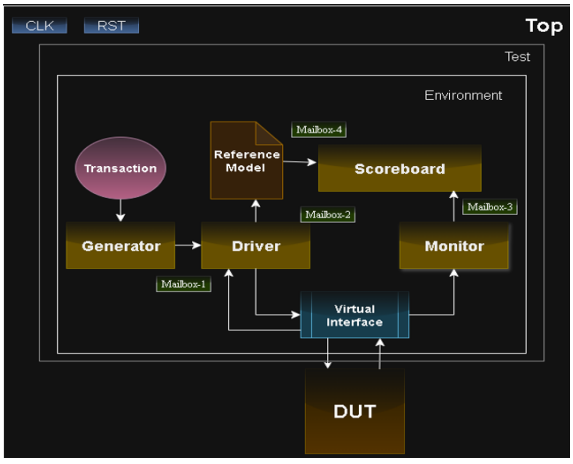
DUT. The environment manages the flow of transactions generating stimulus, applying inputs, observing outputs, and verifying results.

10. TEST:



The Test is the topmost class that controls the overall execution of the verification process. It instantiates the Environment, configures it, and triggers its operation. The test is responsible for starting the stimulus generation, managing simulation time, and applying global settings such as constraints or specific seed values. It includes specific scenarios, corner-case configurations, or directed sequences to target critical functionality.

11. TOP:



The Top Module acts as the integration point between the DUT and the testbench environment. It instantiates the DUT and the interface, which encapsulates all input and output signals. A virtual interface handle is passed from the top module to the environment, allowing testbench components like the driver and monitor to access DUT signals in a synchronized and modular manner. The top module is responsible for generating the clock and reset signals, which are essential for driving the sequential behaviour of the ALU. The clock is typically generated using a simple toggling mechanism with a fixed time delay, while the reset ensures the DUT and testbench components are properly initialized before simulation begins.

5. Plans

1. Below is the link for the detailed test plan.

tplan.xlsx

2. Below is the link for the detailed assertion plan.

assertion_plan.xlsx

3. Below is the link for the detailed coverage plan.

coverage_plan.xlsx