ALU Verification Plan

Swathi Rathod
Emp id - 6116

PROJECT OVERVIEW:

An Arithmetic Logic Unit (ALU) is a basic combinational circuit used in digital systems to perform arithmetic and logical operations on binary numbers. It can handle operations like addition, subtraction, AND, OR, and NOT. This project is a parameterized ALU design using System Verilog. The ALU is capable of performing a variety of operations, including arithmetic, logical, shift, and rotate functions. The main aim is to create a modular and reusable ALU that works correctly in both simulation and synthesis, and can be used as a building block in larger digital design projects.

VERIFICATION OBJECTIVES :

The main goal of verifying the ALU is to make sure it performs all its operations correctly. This means checking that arithmetic operations like addition, subtraction, and increment, along with logical ones like AND, OR, and NOT, give the right results for different inputs. It is also important to see if the output flags—like overflow, carry, and comparison flags (greater than, less than, equal)—are being set correctly based on the operation. The ALU is tested in different scenarios, to ensure it behaves reliably in all cases. Along with this, the verification process checks whether all operations are properly covered (functional coverage) and if any mistakes are caught through the written checks (assertions).

DUT INTERFACES :

The alu interface defines a SystemVerilog interface that encapsulates all input and output signals between the testbench and the DUT (Design Under Test). This interface ensures clean, modular, and synchronized communication using clocking blocks and modports.

The ALU interface include:

INPUTS:

• OPA, OPB: Operand inputs (parameterized width N)

• CMD: Operation command (4 bits to select the desired ALU operation)

• INP_VALID: Indicates which operands are valid (00, 01, 10, 11)

• MODE: 1 for arithmetic, 0 for logical

• CIN: Carry input

• CLK: Clock signal, positive edge triggered

• RST: Asynchronous reset

• CE: Clock enable

OUTPUTS:

• RES: Result of the ALU operation

• COUT: Carry-out from addition/subtraction

• OFLOW: Indicates overflow

• ERR: Indicates invalid conditions (e.g. wrong rotation inputs)

• G, L, E: Comparison flags (Greater, Less, Equal)

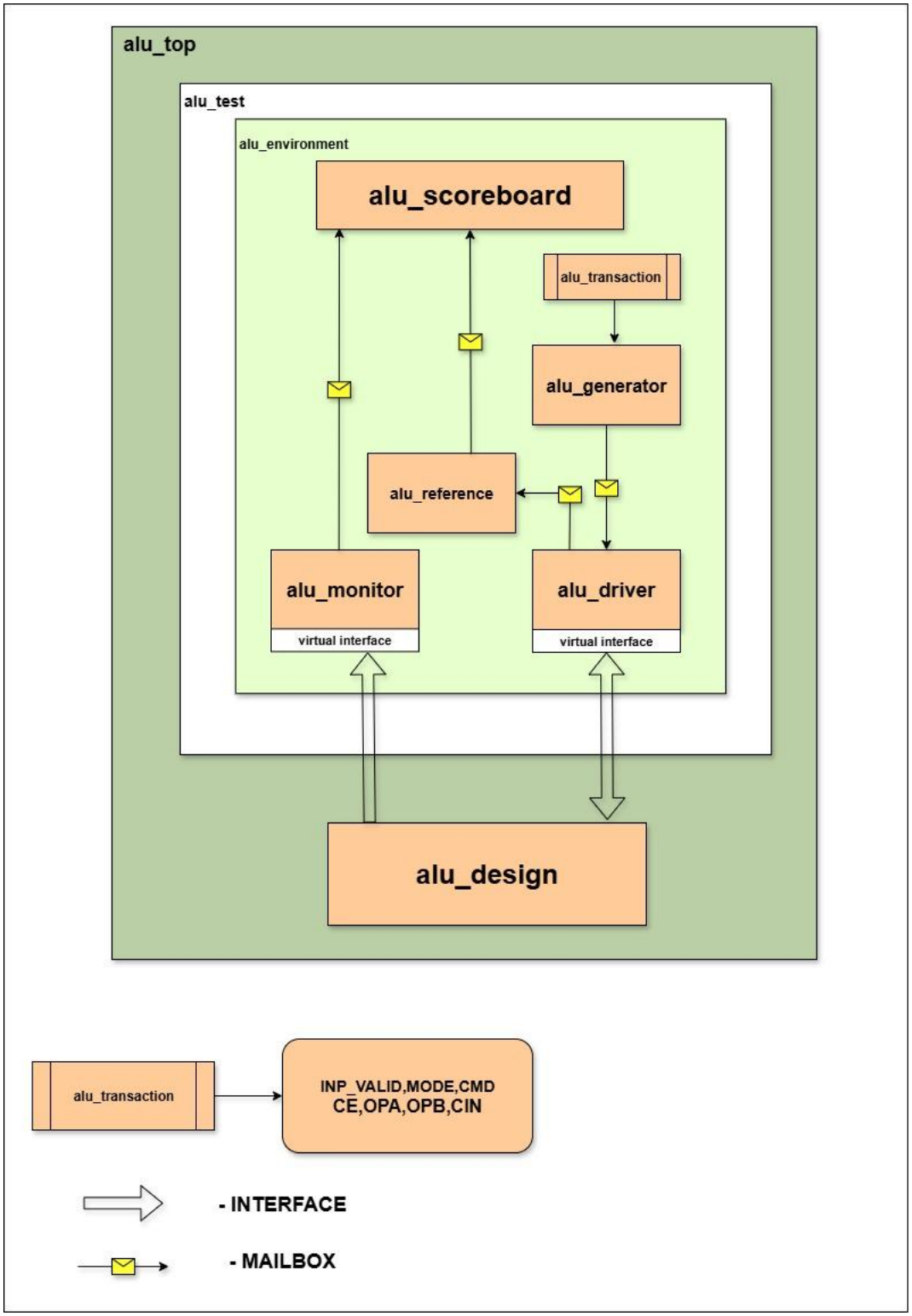
MODPORTS:

The modport groups and specifies the direction to the signals specified within the interface.

CLOCKING BLOCK:

A clocking block in SystemVerilog defines the timing and synchronization for signal interaction between the testbench and DUT. It specifies a clock event as a reference, the signals to be sampled or driven, and the timing of these actions relative to the clock. This helps ensure stable and predictable communication between the testbench and DUT.

TESTBENCH ARCHITECTURE :



1. Top Module (alu_top)

The top module is the top-level wrapper that instantiates the test environment, connects it to the DUT.

2. Test (alu_test)

Test contains environment.

3. Environment (alu_environment)

The environment contains all the essential components such as the driver, monitor, generator, scoreboard, reference model, and mailboxes. These elements work together to provide stimulus, capture outputs, generate expected results, and perform verification comparisons.

4. Generator (alu_generator)

The Generator creates transactions including fields like opa, opb, mode, cmd, inp_valid, ce, cin, etc. It sends the transactions to the Driver via the mailbox mbx_gd.

5. Driver (alu_driver)

The Driver receives the randomized values from the generator and drives it to the DUT through the interface. It also sends the values to the reference model through mailbox.

6. Monitor (alu_monitor)

The Monitor observes and captures the DUT outputs such as res, cout, oflow, err, g, l, and e. It sends this data to the Scoreboard (via mbx_ms) for comparison.

7. Reference Model (alu_reference)

It receives transactions and performs the expected operation, and sends the results to the Scoreboard.

8. Scoreboard (alu_scoreboard)

The Scoreboard performs comparison between actual DUT outputs and the expected outputs from the Reference Model. It logs pass/fail information for each transaction and reports mismatches.

9. Mailboxes

Mailboxes provide a transaction-level communication channel between various testbench components. Total 4 mailboxes are there,they are from:

- Generator to Driver
- Driver to Reference Model
- Reference Model to Scoreboard
- Monitor to Scoreboard

10. Design Under Test (DUT)

The DUT receives driven inputs from the Driver and sends responses back to the Monitor. It is treated as a black-box connected through virtual interfaces.

11.Interface

An interface is a bundle of signals or nets through which a testbench communicates with a design. The interface construct is used to connect the design and testbench.

12.Virtual interface

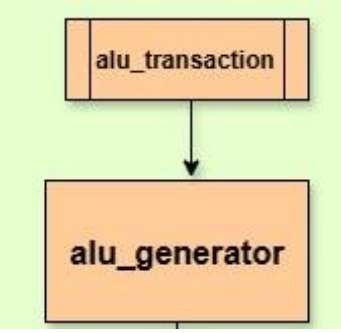
A virtual interface is a variable of an interface type that is used in classes to provide access to the interface signals. A virtual interface is a variable that represents an interface instance.

13.Transaction

Transaction consists of ALU inputs that need to be randomized and outputs that are not randomized.

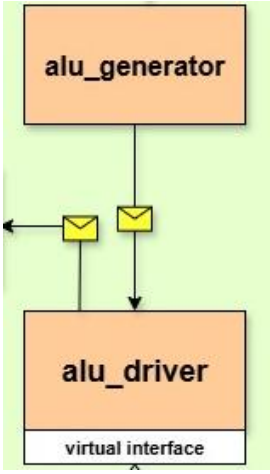
FLOWCHART OF SV COMPONENTS :

1.Transaction class



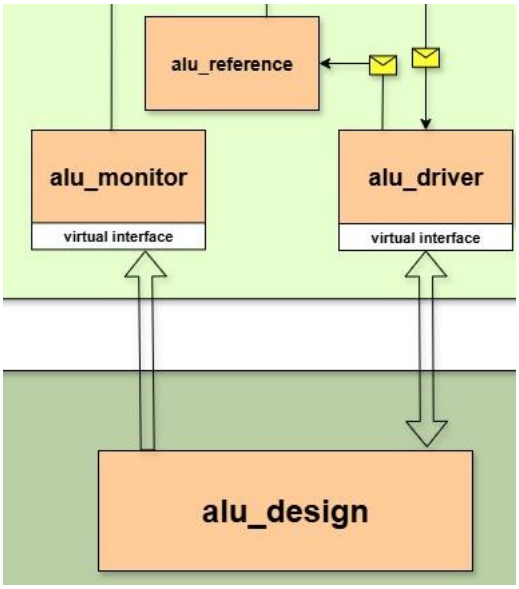
The transaction class consists of randomized inputs and non-randomized outputs. It consists of constraints and a deep copy function for creating transaction copies.

2.Generator class



This component generates randomized input values and is sent to the driver through a mailbox.

3.Driver class



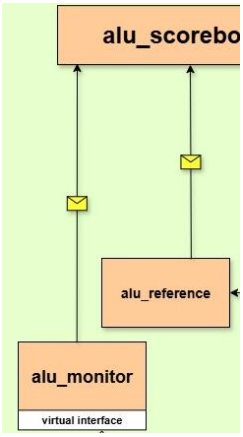
It consists of two mailbox:

Generator to driver mailbox: It transfers randomized transactions from the generator to the driver

Driver to reference model mailbox: It sends the same transactions to the reference model

The task applies the stimulus to the DUV based on the received transaction

4.Monitor class



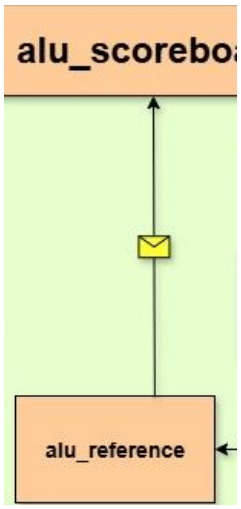
The ALU Monitor observes the DUT's outputs and converts them into transaction objects for validation.

It consists of one mailbox:

Monitor to Scoreboard Mailbox: It transfers captured output transactions to the scoreboard

Detects valid output transactions, samples the result, and sends it as a transaction object to the scoreboard

5.Reference model



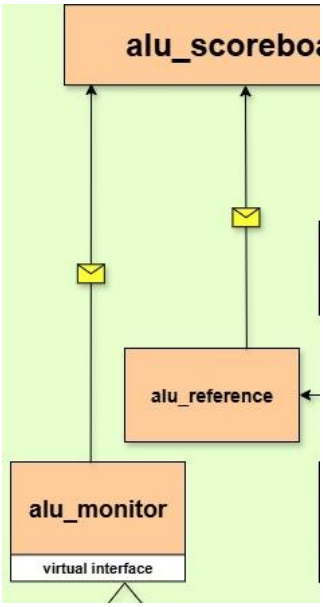
Acts as the reference for result validation. It processes the same inputs as the DUV and produces the expected outputs.

It consists of two Mailboxes:

Driver to reference model mailbox: It receives transactions from the driver

Reference to scoreboard mailbox: It sends computed expected results to the scoreboard

6. Scoreboard



The scoreboard verifies the correctness of the DUV by comparing its outputs with the reference model's expected results.

It consists of two mailbox:

Reference model to scoreboard mailbox : It receives the expected results from the reference model

Monitor to Scoreboard mailbox: It receives actual results from the monitor

Compares the actual and expected results, and flags mismatches and generates final statistics, including pass/fail counts and functional coverage details.

TEST PLAN:

Functional Coverage Plan:

The ALU functional coverage plan includes monitoring of key input, output, and control signals to ensure comprehensive verification. Coverage for INP_VALID ensures all possible values (00, 01, 10, 11) are exercised. The CMD signal must cover all 14 ALU operations (0 to 13). Control signals like CE, CIN, and RESET are checked for toggle activity between 0 and 1. The MODE signal ensures both arithmetic and logic modes are tested. Operand inputs OPA and OPB should each span the full value range from 0 to (2^N)-1, while the result RES must cover from 0 to (2^(N+1))-1. Output flags such as ERR, COUT, and OFLOW are verified for correct assertion under relevant conditions. Comparison flags—E, G, and L—are checked for proper behavior when OPA == OPB, OPA > OPB, and OPA < OPB, respectively. Lastly, cross coverage between CMD and MODE (CMD_X_MODE) ensures that each operation is tested under both arithmetic and logic modes, totaling 28 cross bins.

Assertions:

Signal validity is checked by checking that CMD stays within its range and INP_VALID correctly reflects input readiness. Timing alignment assertions ensure CMD, OPA, and OPB arrive together as expected. Reset (RST) behaviour assertions guarantee an immediate clear of all outputs and flags, regardless of the clock. For functional checks, logical modes keep COUT and OFLOW low, and comparison flags (G, L, E) are mutually exclusive. An error condition is asserted if CMD is 12 or 13 while OPB[7:4] isn’t 0000, forcing ERR high. Finally, result stability is checked so RES remains unchanged when CE is low or INP_VALID is 00.