**ALU VERIFICATION**

Havala P N

havalapn22@gmail.com

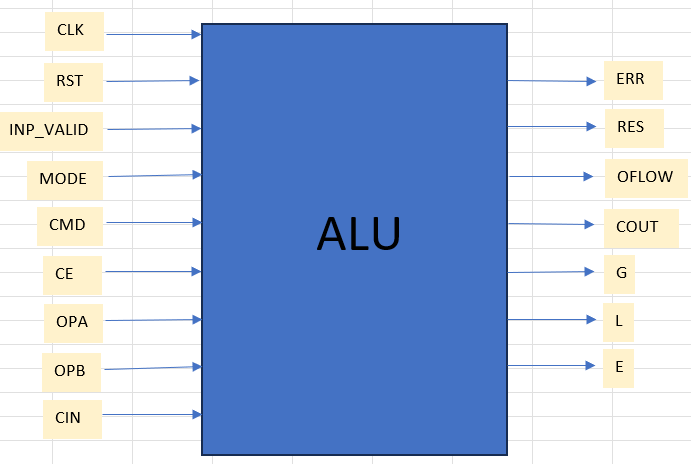
**CHAPTER 1: PROJECT OVERVIEW**

The ALU Verification project aims to test and validate a parameterized Arithmetic Logic Unit (ALU) using a SystemVerilog based testbench. The ALU supports various arithmetic and logical operations along with various flags. A constrained randomisation, coverage group is used along with components like generator, driver, monitor and scoreboard to stimulate and check the DUT. Functional and code coverage are used for testing.

**CHAPTER 2: VERIFICATION OBJECTIVES**

1. To ensure that the ALU performs all arithmetic, logical and shift operations correctly.
2. To verify that the output of the ALU matches the expected result for each operation.
3. To check that status flags such as carry, overflow, zero, equal, greater and less are generated correctly.
4. To test the ALU with a wide range of input values, including both normal and boundary cases.
5. To use assertions and checks to catch unexpected or incorrect behaviour during simulation.
6. To achieve sufficient functional and code coverage to ensure complete verification.

**CHAPTER 3: DUT INTERFACE**

****

**Inputs:**

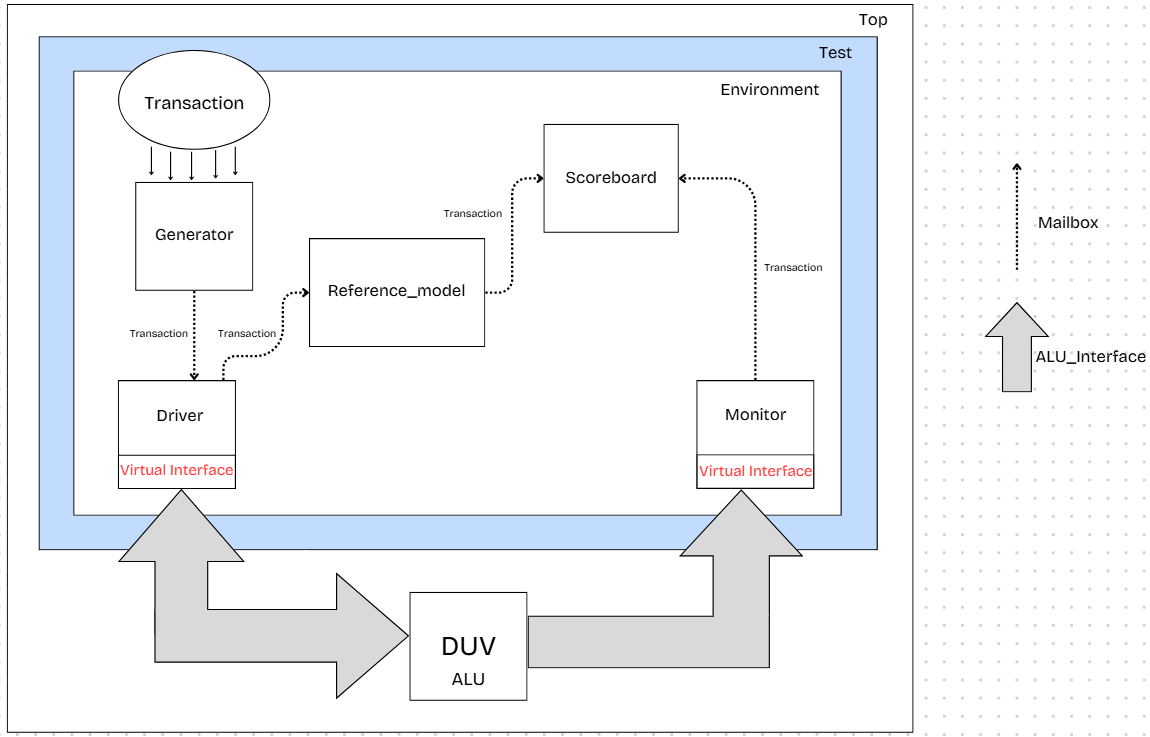
* CLK - This is the clock signal to the design and it is edge sensitive.
* RST - This is the active high asynchronous reset to the design. When RST is high, all outputs are reset. When RST is low i.e. RST = 0, the CE (clock enable) is checked.
* CE - This is the active high clock enable signal 1 bit. When CE = 0, there is no change in the output as the operands are not evaluated whereas when CE = 1, the MODE is checked.
* MODE – Control signal that determines if the design has to perform either Arithmetic or Logical operations. If MODE = 1, the ALU performs Arithmetic operations and if the MODE = 0, it performs Logical operations.
* OPA, OPB: Parameterised input operands
* INP\_VALID - 2’b00: No operand is valid
* 2’b01: Only operand A is valid. Operations are performed only on OPB
* 2’b10: Only operand B is valid. Operations are performed only on OPA
* 2’b11: Both operands are valid
* CMD: Defines the operation to be executed in conjunction with the MODE signal.
* CIN: Carry-in bit given in addition with OPA and OPB in operations such as ADD and SUB.

**Outputs:**

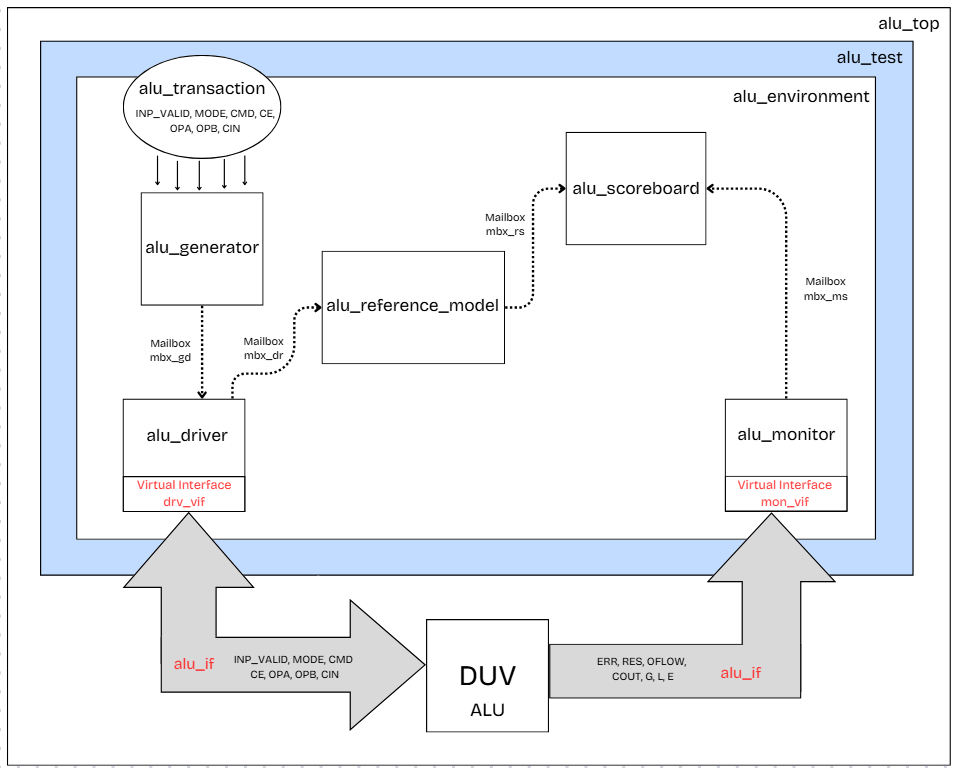
* RES - Result of ALU operation.
* COUT - Carry-out Flag.
* OFLOW - Overflow Flag.
* G, L and E - Flags indicating greater-than, less-than, or equal condition between operands (from comparisons).
* ERR – Error Flag.

**CHAPTER 4: VERIFICATION ARCHITECTURE**

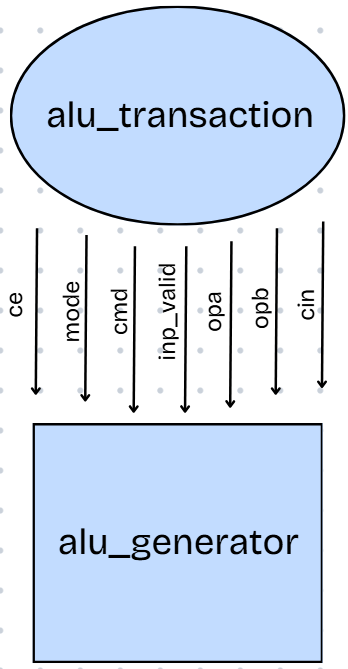
**4.1: TESTBENCH ARCHITECTURE**



**4.2: ALU TESTBENCH ARCHITECTURE**

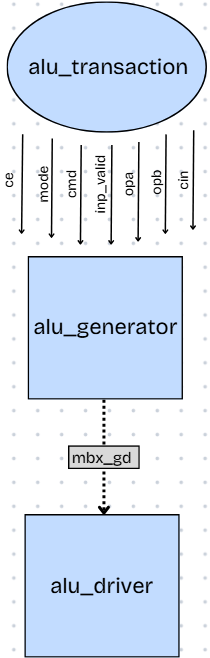
****

**4.2.1: TRANSACTION**

****

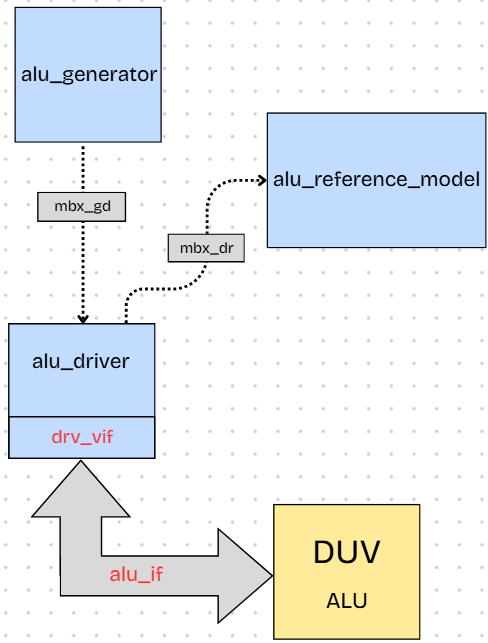
* Transaction includes all the necessary input signals and expected output signals required to test the ALU. The constraints for randomization are given in the transaction.
* The generator creates transactions with either random or directed.

**4.2.2: GENERATOR**

****

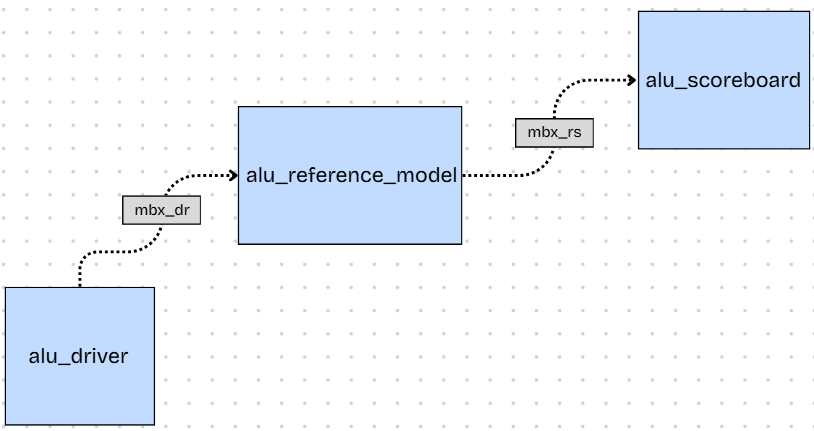
* The generator is responsible for creating ALU transactions randomly.
* It uses randomize () method to generate random values for inputs like opa, opb, cmd, mode, etc.
* After generating a valid transaction, it places the transaction into a mailbox to be picked up by the driver.
* The process is repeated in a loop to generate multiple test cases.

**4.2.3: DRIVER**

****

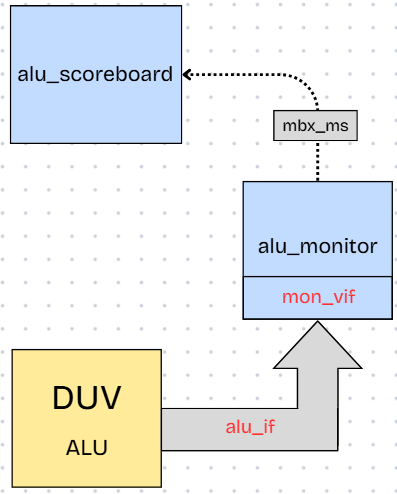
* The driver gets ALU transactions from the generator via a mailbox.
* The values obtained through the mailbox are applied to the DUV (ALU) using the virtual interface signals.
* The driver sends the transaction to the reference model. This means, both the reference model and the DUV receive the same set of inputs simultaneously.

**4.2.4: REFERENCE MODEL**

****

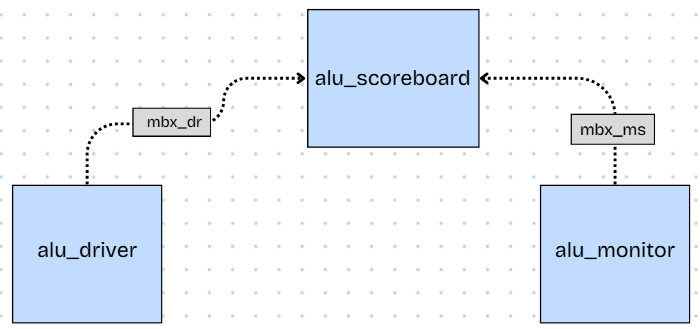
* The reference model gets a copy of the transaction from the driver through a mailbox.
* It mimics the behaviour of the DUV and based on the inputs and operation, it calculates the expected result and status flags.
* The expected output and flags are written back into the transaction object.
* The updated transaction is then sent to the scoreboard for comparison with the actual DUV output.

**4.2.4: MONITOR**

****

* The monitor watches the ALU outputs through the virtual interface.
* The values captured by the monitor are sent to the scoreboard for comparison through a mailbox.

**4.2.5: SCOREBOARD**

****

* The scoreboard gets the expected output from the reference model and the actual output from the monitor.
* If any value doesn’t match, it flags a mismatch and logs the error for debugging.
* Each test case is recorded as pass if all outputs match, or fail if any mismatch is found.

**CHAPTER 5: TEST PLAN**

**5.1: Test Scenarios:**

[Testcases](https://docs.google.com/spreadsheets/d/1n7TZ7ERN6w-v39LnOZK00wL89nI7R4qg/edit?gid=887637571%23gid=887637571)

**5.2: Coverage plan:**

[Coverage Plan](https://docs.google.com/spreadsheets/d/1wOJnZYcWnBawjc_NTwAHQMston4fM6TL/edit?gid=1097903250%23gid=1097903250)

**5.3: Assertion Plan:**

[Assertion Plan](https://docs.google.com/spreadsheets/d/1PcE_BNzbui1xjvLluUTAyP48c8hTYBHT/edit?gid=1422700542%23gid=1422700542)