ALU VERIFICATION PLAN

EMP ID -6091

PRATHIKSHA

1.PROJECT OVERVIEW

The Arithmetic Logic Unit (ALU) is a key part of any computer's processor. It is responsible for performing both arithmetic and logic operations. Arithmetic operations include adding, subtracting, increasing (increment), and decreasing (decrement) numbers. Logic operations help the processor make decisions and include functions like AND, OR, NOT, and XOR. Besides these, the ALU can also compare two numbers to find out if one is greater than, less than, or equal to the other. These comparison results are important for making decisions in programs, like running a certain part of code only if a condition is true.
To make sure the ALU works correctly in every situation, a verification plan is used. This plan checks how the ALU behaves with different types of inputs and under various conditions, including different timing scenarios. By testing it thoroughly, we can be confident that the ALU performs correctly and reliably in real-world use.

2.VERIFICATION OBJECTIVES

The main objective of verifying the ALU is to check that it performs all its supported operations correctly. This includes making sure that arithmetic operations like addition, subtraction, and increment, as well as logical operations like AND, OR, and NOT, produce the correct output for all types of inputs. It is also important to check that the output flags—such as overflow, carry, and comparison flags (greater, less, equal)—are set properly depending on the operation.
Another key part of verification is checking how the ALU behaves when given wrong or incomplete inputs. For example, if one of the required operands is missing or arrives late, the ALU should raise an error. The rotate operations are also tested carefully to make sure they work correctly and raise errors if used with invalid input patterns.
Additionally, the goal is to test the ALU under different conditions, such as changing inputs quickly or giving the same input in different ways, to ensure the design is stable and works in all possible situations. Apart from functional checks, the verification also includes making sure that all types of operations are tested (functional coverage) and that written checks (assertions) are triggered correctly when something goes wrong.

3.DUT INTERFACES

The ALU module interfaces include:

INPUTS:

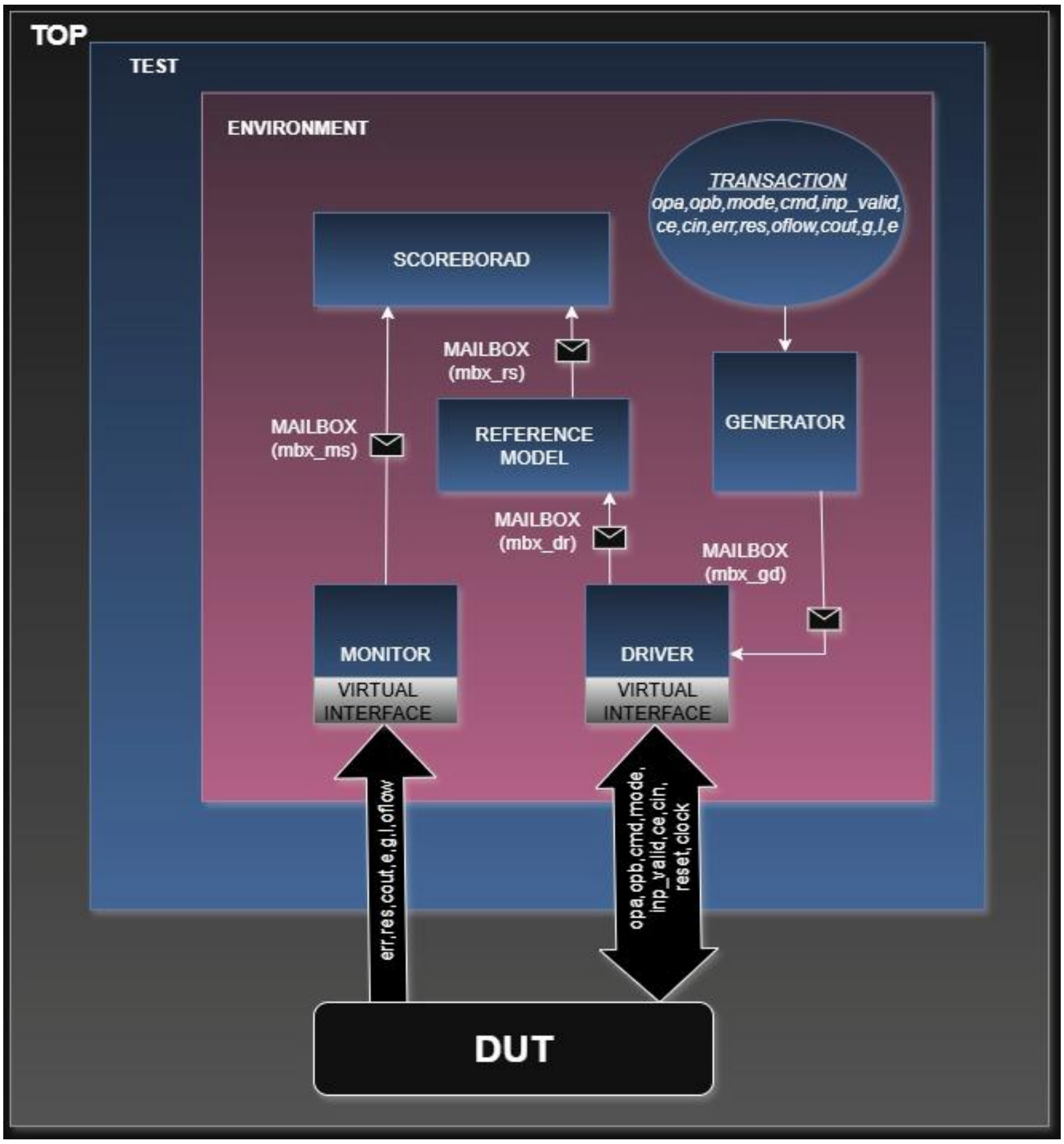
- OPA, OPB: Operand inputs (parameterized width N)
- CMD: Operation command (4 bits to select the desired ALU operation)
- INP_VALID: Indicates which operands are valid (00, 01, 10, 11)
- MODE: 1 for arithmetic, 0 for logical operations
- CIN: Carry input used in arithmetic operations
- CLK: Clock signal, 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 port directions to the wires/signals declared within the interface.
Clocking blocks- clocking block specifies timing and synchronization for a group of signals.
Clock skew-Clocking skew specifies the moment w.r.t clock edge at which input and output clocking signals are to be sampled or driven respectively.

4.TESETBENCH ARCHITECTURE


TOP
TEST
ENVIRONMENT
TRANSACTION
opa,opb,mode,cmd,inp_valid,
ce,cin,err,res,oflow,cout,g,l,e
SCOREBORAD
MAILBOX
(mbx_rs)
GENERATOR
MAILBOX
(mbx_ms)
REFERENCE
MODEL
MAILBOX
(mbx_dr)
MAILBOX
(mbx_gd)
MONITOR
VIRTUAL
INTERFACE
DRIVER
VIRTUAL
INTERFACE
err,res,cout,e,g,l,oflow
opa,opb,cmd,mode,
inp_valid,ce,cin,
reset,clock
DUT

1. Top Module – TEST
The TEST module acts as the top-level wrapper that instantiates the test environment, connects it to the DUT.

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

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

4. Driver
Driver receives random values through mailbox from the generator and sends it to reference model and also to the DUT through interface.

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

6. Reference Model
The Reference Model is the behavioral model of the ALU. It receives inputs from driver and performs the expected computation, and sends the results to the Scoreboard.

7. 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.

8. Mailboxes
Mailboxes provide a transaction-level communication channel between various testbench components. They include:

- mbx_gd: Generator to Driver
- mbx_dr: Driver to Reference Model
- mbx_rs: Reference Model to Scoreboard
- mbx_ms: Monitor to Scoreboard

9. 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.

10. Transaction

Transcations consist of inputs signals that needs to be randomized and it also consists of output signals.

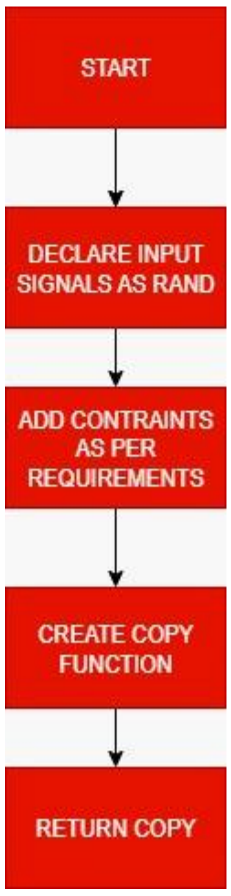
11. Interface

An interface is a group of signals bundled together to connect different parts of a design.

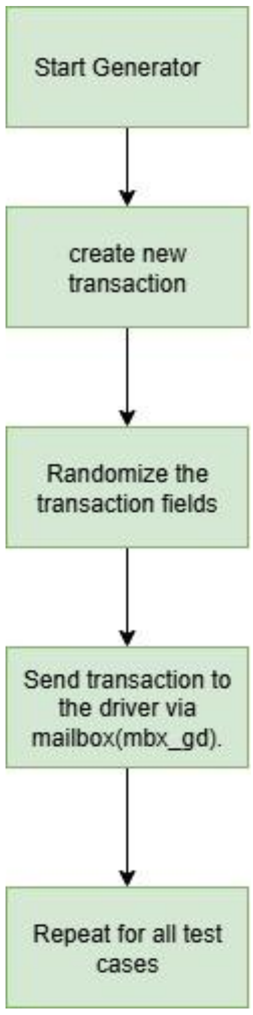
12. Virtual Interface

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

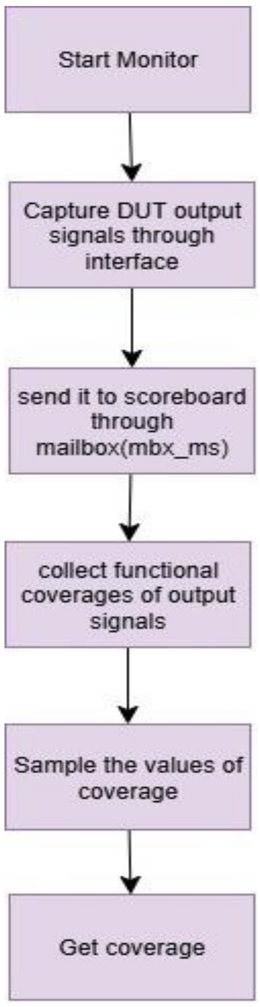
FLOW CHART


START
DECLARE INPUT SIGNALS AS RAND
ADD CONTRAINTS AS PER REQUIREMENTS
CREATE COPY FUNCTION
RETURN COPY

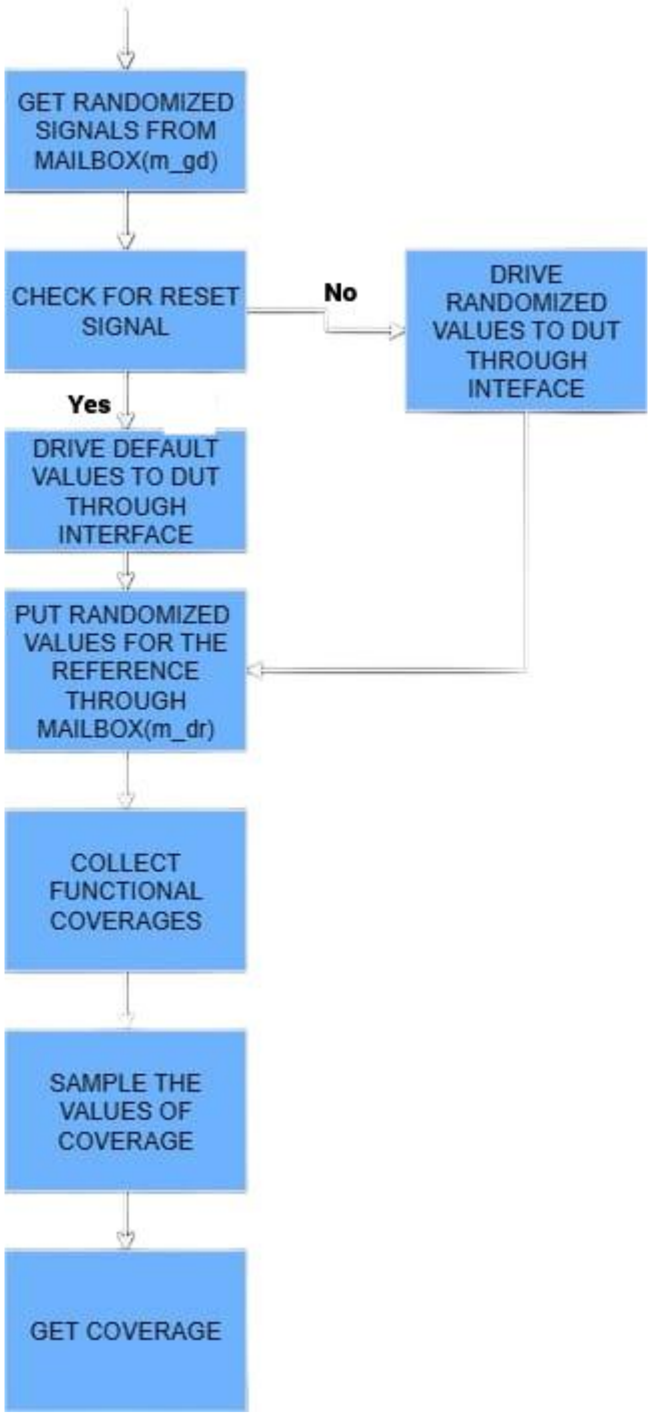
a.TRANSACTION

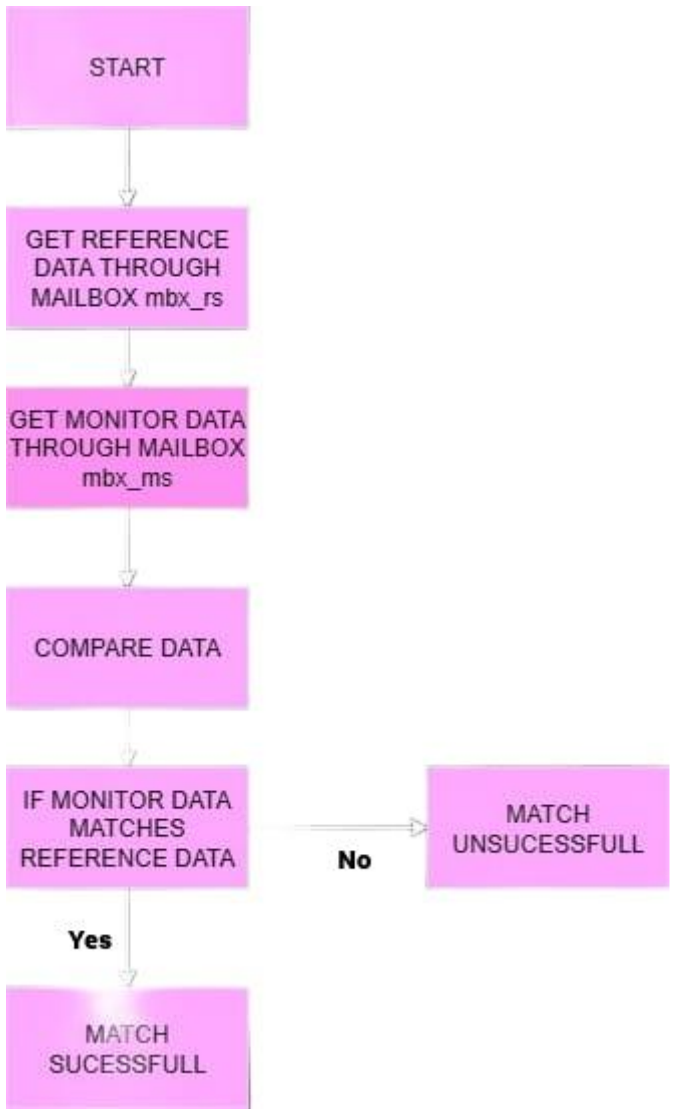

Start Generator
create new transaction
Randomize the transaction fields
Send transaction to the driver via mailbox(mbx_gd).
Repeat for all test cases

b.GENERATOR


Start Monitor
Capture DUT output signals through interface
send it to scoreboard through mailbox(mbx_ms)
collect functional coverages of output signals
Sample the values of coverage
Get coverage

c.MONITOR


GET RANDOMIZED SIGNALS FROM MAILBOX(m_gd)
CHECK FOR RESET SIGNAL
No
DRIVE RANDOMIZED VALUES TO DUT THROUGH INTEFACE
Yes
DRIVE DEFAULT VALUES TO DUT THROUGH INTERFACE
PUT RANDOMIZED VALUES FOR THE REFERENCE THROUGH MAILBOX(m_dr)
COLLECT FUNCTIONAL COVERAGES
SAMPLE THE VALUES OF COVERAGE
GET COVERAGE


START
GET REFERENCE DATA THROUGH MAILBOX mbx_rs
GET MONITOR DATA THROUGH MAILBOX mbx_ms
COMPARE DATA
IF MONITOR DATA MATCHES REFERENCE DATA
No
MATCH UNSUCESSFULL
Yes
MATCH SUCESSFULL

e.SCOREBOARD

d.DRIVER