ALU Verification Project

Jason Linus Rodrigues

Emp id: 6078

CHAPTER 1:

PROJECT OVERVIEW AND SPECIFICATIONS.

1.1 Project Overview:

An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer.

The purpose of the ALU is to perform mathematical operations such as addition, subtraction, multiplication etc. Additionally, the ALU processes basic logical operations like AND/OR calculations etc. It serves as the computational hub of the Central Processing Unit (CPU) for a computer system. The performance and efficiency of the ALU directly impact the overall speed and capability of a computer system.

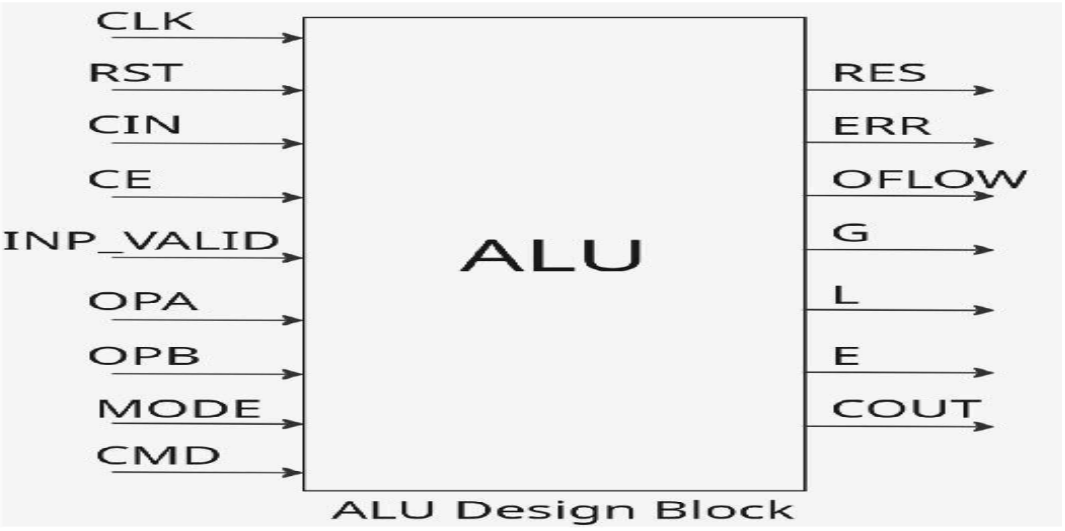


Figure 1: General ALU Design Block

1.2 Verification Objectives:

- To construct a verification planar that includes test plan, functional coverage plan and assertion plan.

- To Design and built a ALU Testbench architecture along with a structured code plan for testbench components.
- To implement functional coverage and assertions in the ALU testbench and validate the ALU's functional correctness along with its timings and find its design flaws.

1.3 DUT Interfaces:

Signals	Direction	Width-Size	Description
Clock	Input	1	Generates a Positive edge triggering signal
Reset	Input	1	Active high synchronous reset signal that sets all the output signals to 0
Clock Enable	Input	1	Active high clock enable signal that perform all ALU operation
Mode	Input	1	A 1bit select signal that perform arithmetic operation if mode signal is 1, else Logical operation if mode signal is 0
Command	Input	Parameterized Command Width Size	Parameterized (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: Operand A and B both incremented by 1, then multiplication performed. 10: Operand A left-shifted by 1, then multiplied by 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: ROL_A_B 13: ROR_A_B
Input Valid	Input	2	A 2 bit select signals that chooses the input operands. 00: no operand is selected 01: operand A is selected 10: operand B is selected 11: both operand A and operand B is selected
Operand A	Input	Parameterised width size	Parameterized operand A input data
Operand B	Input	Parameterised width size	Parameterized operand B input data
Carry in	Input	1	1-bit active high carry input signal
Result	Output	Parameterised width size	A Parameterized result when the specific operation is evaluated
Overflow	Output	1	A 1-bit signal indicates an output is overflow, during addition or subtraction or increment or decrement operation.
Carry out	Output	1	A 1-bit signal indicating a carry is generated during Addition and increment operation.
Equal	Output	1	A 1-bit comparator output signal, which indicates that the value of operand A is equal to the value of Operand B
Greater	Output	1	A 1-bit comparator output signal, which indicates that the value of operand A is greater than the value of Operand B
Lesser	Output	1	A 1-bit comparator output signal, which indicates that the value of

			operand A is lesser than the value of Operand B
Error	Output	1	A 1-bit error output signal if input_valid is 0, or if a logical operation (mode = 0) is requested with Command 12 or 13 and the required range [0 : $\log_2$(operandB)] is not provided or is invalid.

CHAPTER 2:

TESTBENCH ARCHITECTURE AND METHODOLOGY

2.1 Testbench Architecture:

General Testbench Architecture:

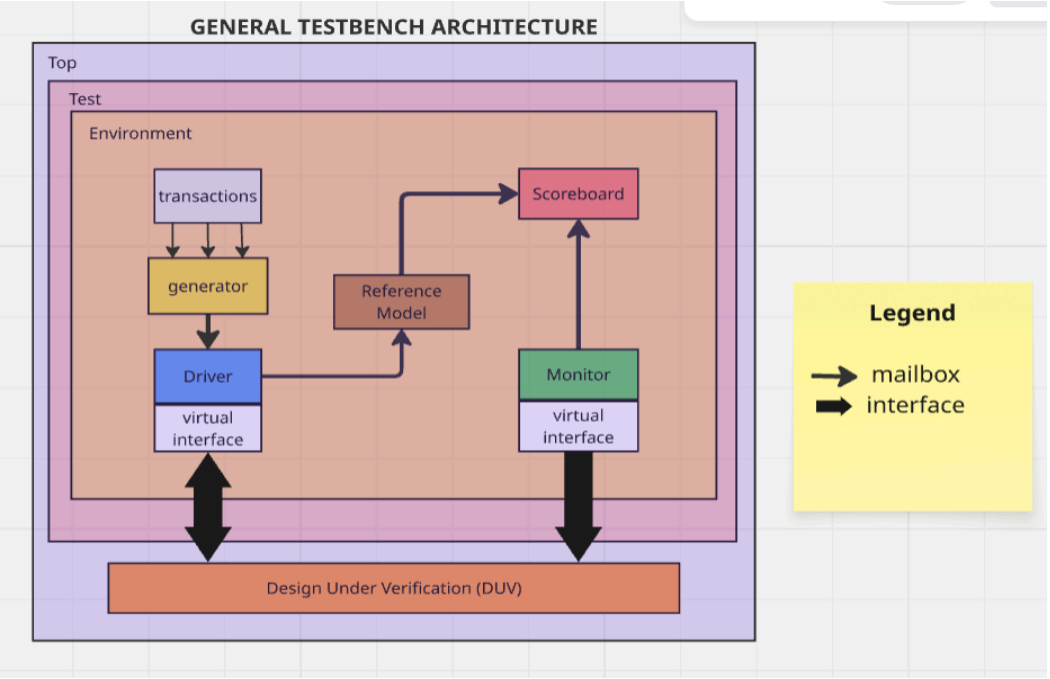


Figure 2: General Testbench Architecture

Proposed Testbench Architecture:

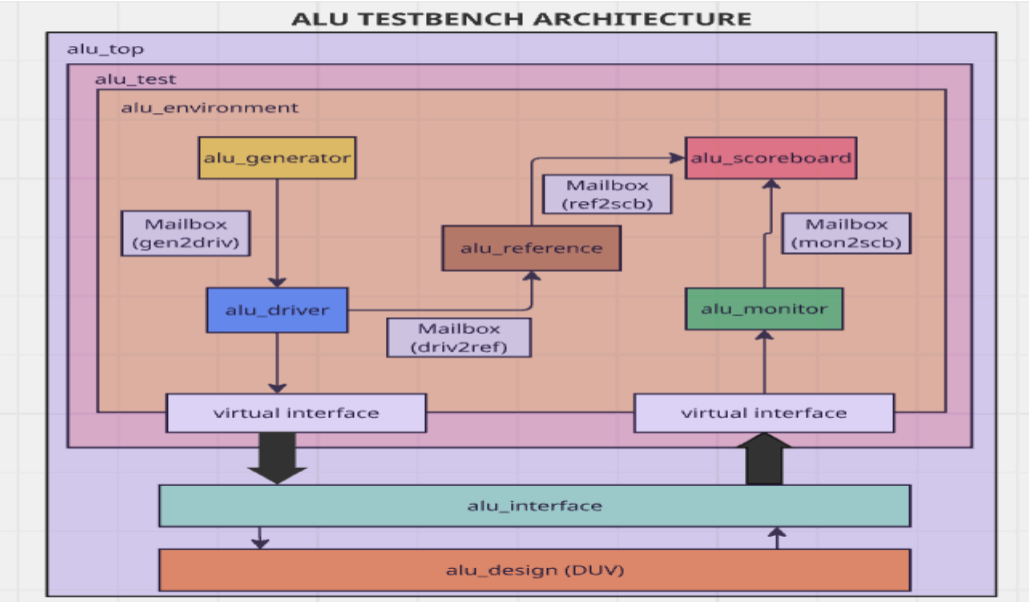


Figure 3: Proposed ALU Testbench Architecture

2.2 Component Details and Flowchart

a) Transaction:

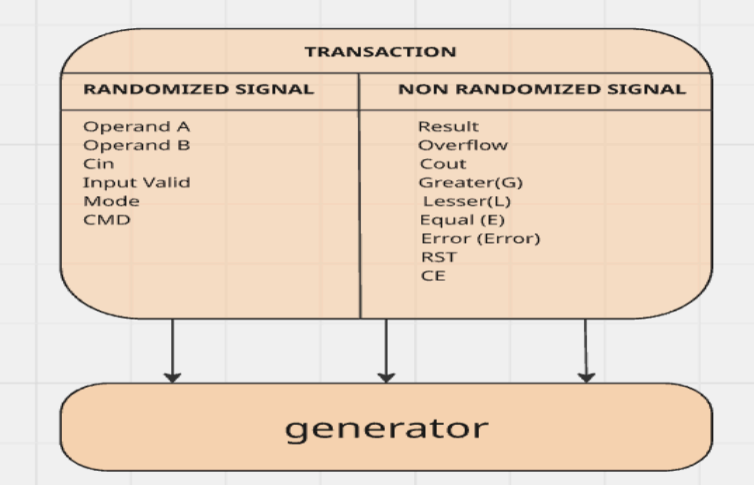
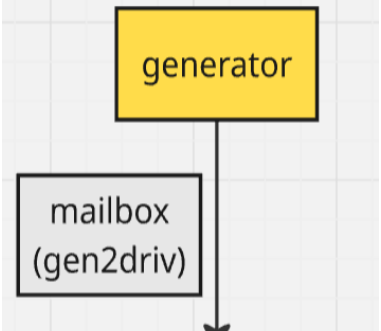


Figure 4: Transaction flowchart

The transaction consists of set of randomized and non - randomized signals shown in the above flowchart. It also has a set of constraint conditions necessary for ALU verification followed by the copy function and display function. Here the transaction creates a set of input signal whenever a randomized function is called.

b) Generator:



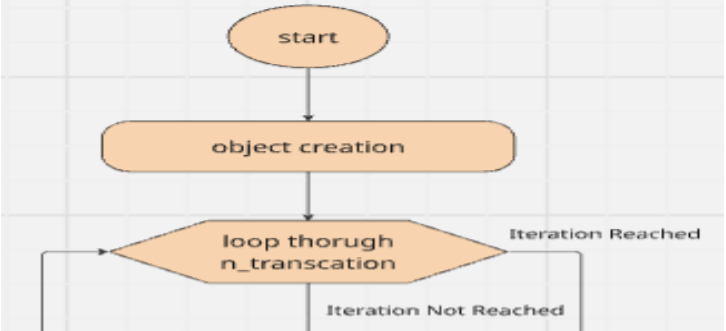
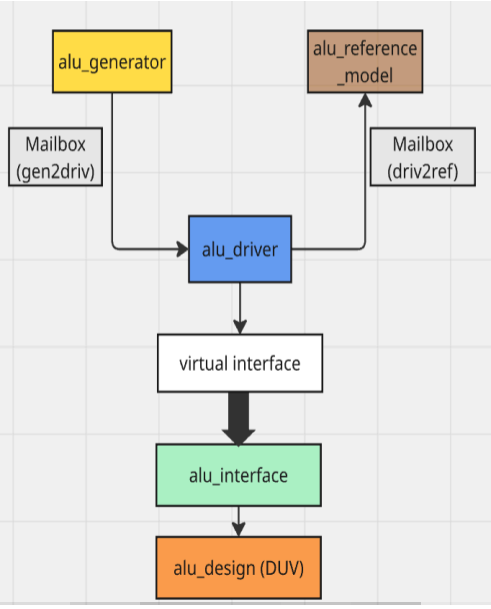


Figure 6: Working of Generator Class

The generator flowchart represents the process of generating and sending randomized transactions in a testbench. It begins with object creation, where the required transaction object is instantiated. A loop is executed for a specified number of transactions (n_transaction). Inside each iteration, randomization of transaction fields is performed. The randomized transaction is then sent through a mailbox for further processing by other components like a driver. This loop continues until all iterations are completed, after which the process stops.

c) Driver:



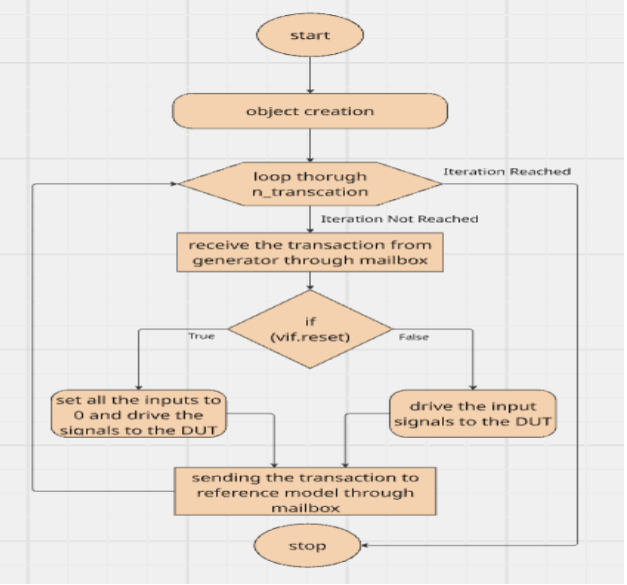
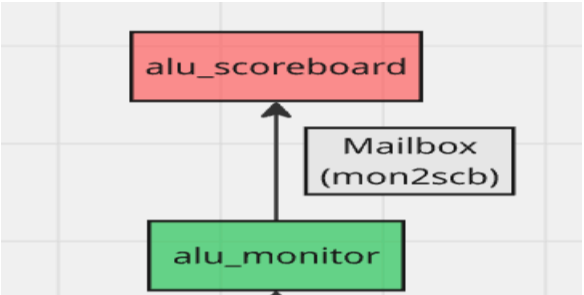


Figure 8: Working of Driver Class

The Driver flowchart illustrates the driving operation in the testbench. The process begins with object creation and enters a loop for the specified number of transactions (n_transaction). For each iteration, a transaction is received from the generator via a mailbox. The driver checks the reset condition (vif.reset); if reset is active, all inputs are set to zero and driven to the DUT. If reset is inactive, the driver applies the transaction signals to the DUT. Finally, the transaction is sent to reference model through the mailbox, and the loop continues until all iterations are complete, after which the process stops

d) Monitor:



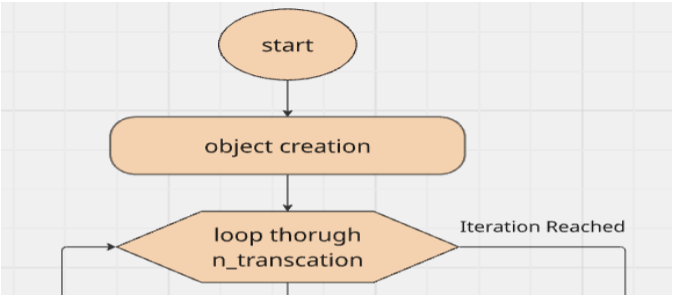
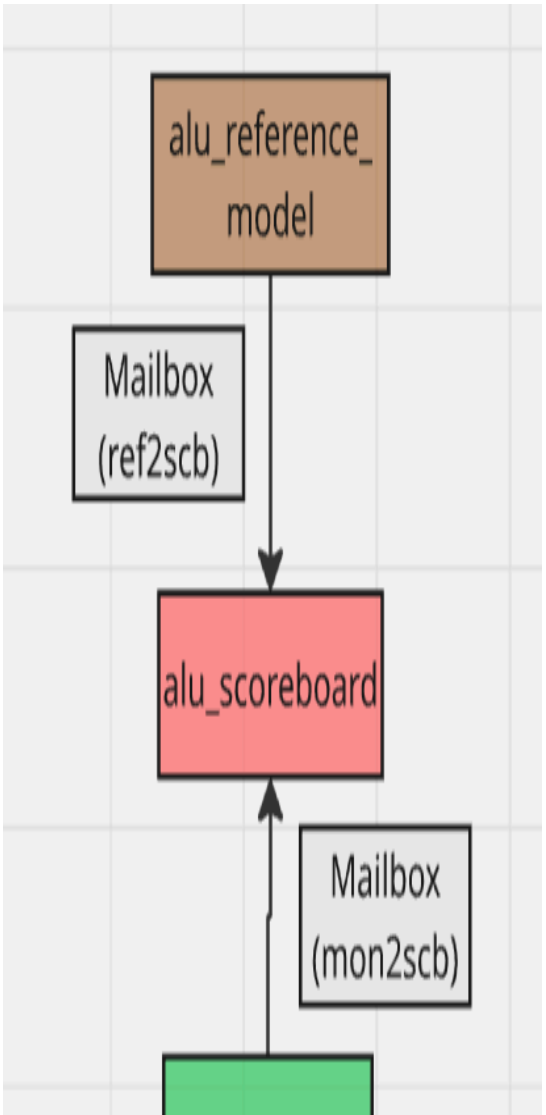


Figure 10: Working of Monitor Class

The Monitor flowchart describes the monitor operation in a testbench. The process starts with object creation and enters a loop for the defined number of transactions (n_transaction). In each iteration, the monitor extracts the output signals from the DUT (Design Under Test). These captured signals are then packaged into a transaction and sent to the scoreboard through a mailbox for result checking. The loop continues until all transactions have been processed. Finally, the process ends when all iterations are complete.

e) Scoreboard:

The Scoreboard Flowchart illustrates the scoreboard operation in a testbench environment. It starts with the creation of objects for communication with the reference model and monitor. Two packets are generated to store the reference results and monitor results**,** along with initializing pass and fail counters**.** For each transaction, results from the reference model and the monitor are received through mailboxes and stored in their respective packets. The scoreboard then performs a comparison between the reference and monitor packets. If the results match, the pass count is incremented; otherwise, the fail count is updated. The process repeats for all transactions and stops after all comparisons are complete.



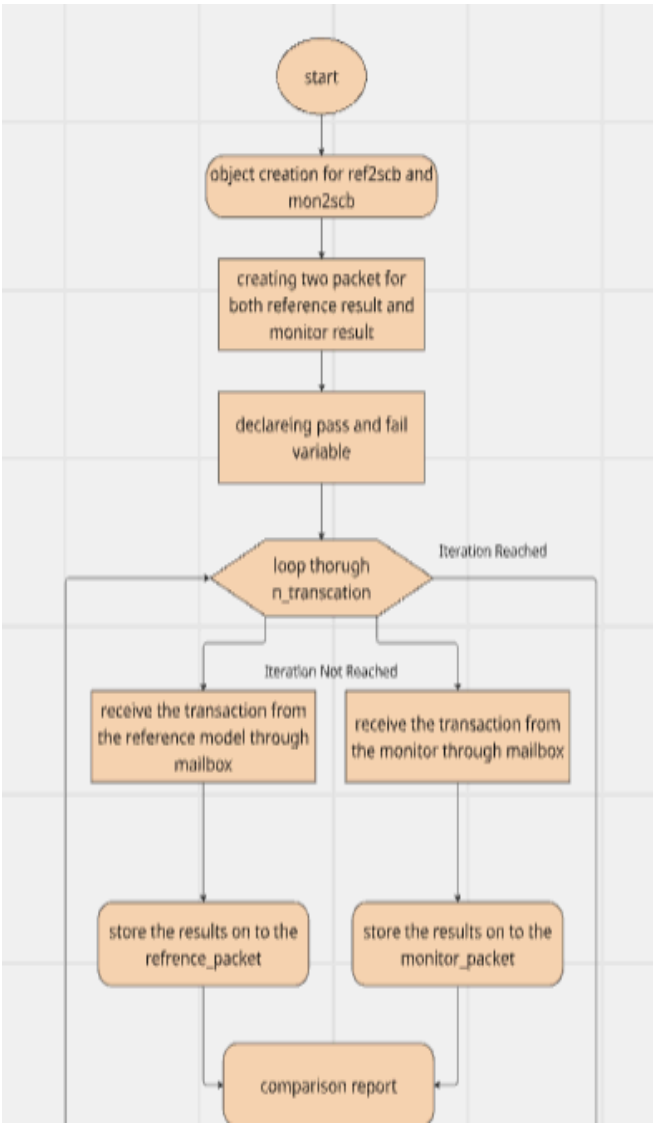
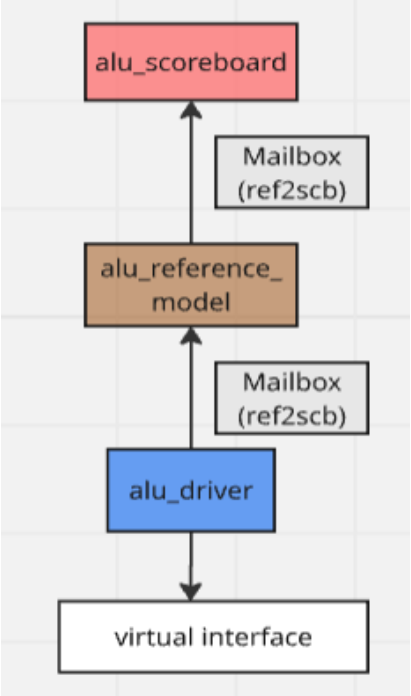


Figure 12: Working of Scoreboard Class

f) Reference model:



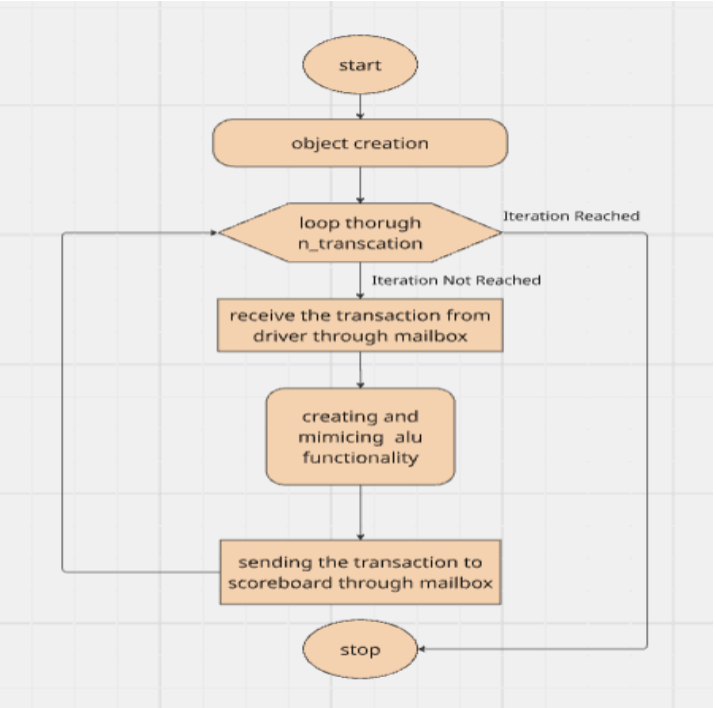
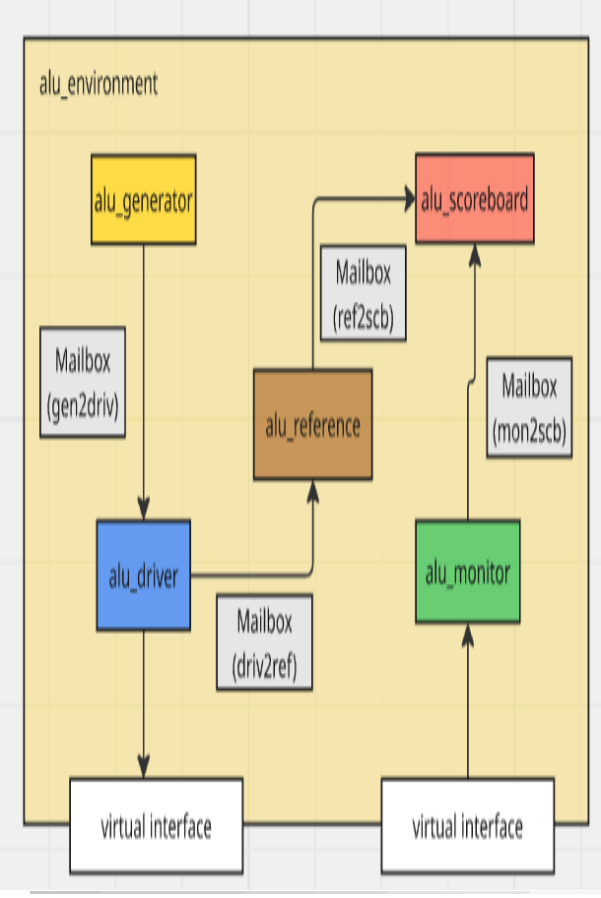


Figure 14: Working of Reference Model Class

The Reference model flowchart represents the operation of the dummy ALU model in the testbench. It begins with object creation and enters a loop for the specified number of transactions (n_transaction). For each iteration, the reference model receives a transaction from the driver via a mailbox. It then mimics the ALU functionality by computing the expected result based on the input operation and operands. The computed result is sent to the scoreboard through a mailbox for comparison with the DUT output. The process repeats until all transactions are processed, after which the process stops.

g) Environment:

The Environment flowchart represents the initialization and execution of testbench components. Here it Integrates all the test components and connects them using mailboxes. The process begins by a build task that define the creation of mailbox objects for communication between different components. Next, objects for all testbench components (generator, driver, monitor, scoreboard, and reference model) are created using a constructor. A start task is then defined to coordinate the flow. The start task sequentially initiates the generator, driver, monitor, scoreboard, and reference model tasks. These tasks run in parallel to perform the verification process. Finally, the process stops after all components complete their respective operations.



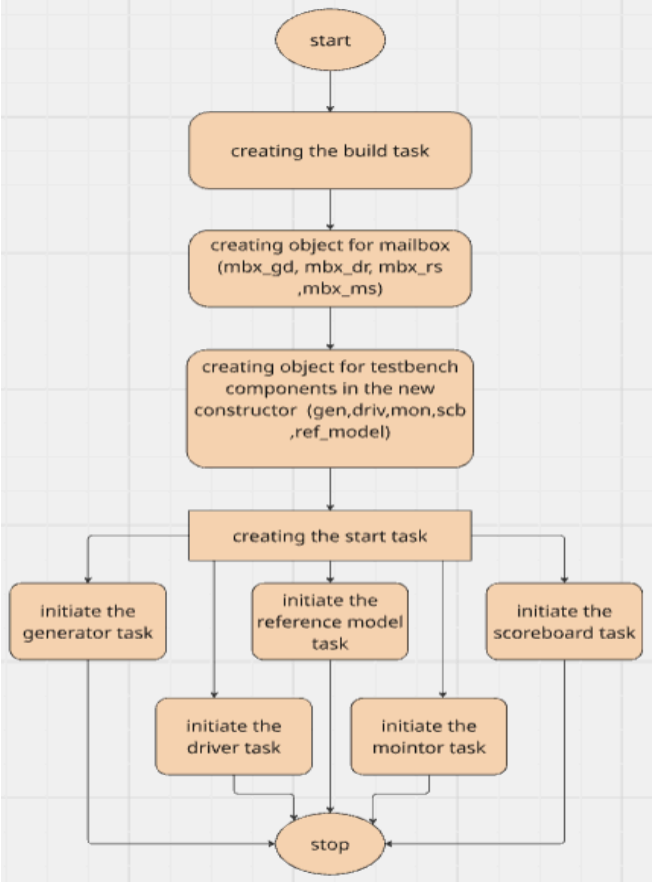
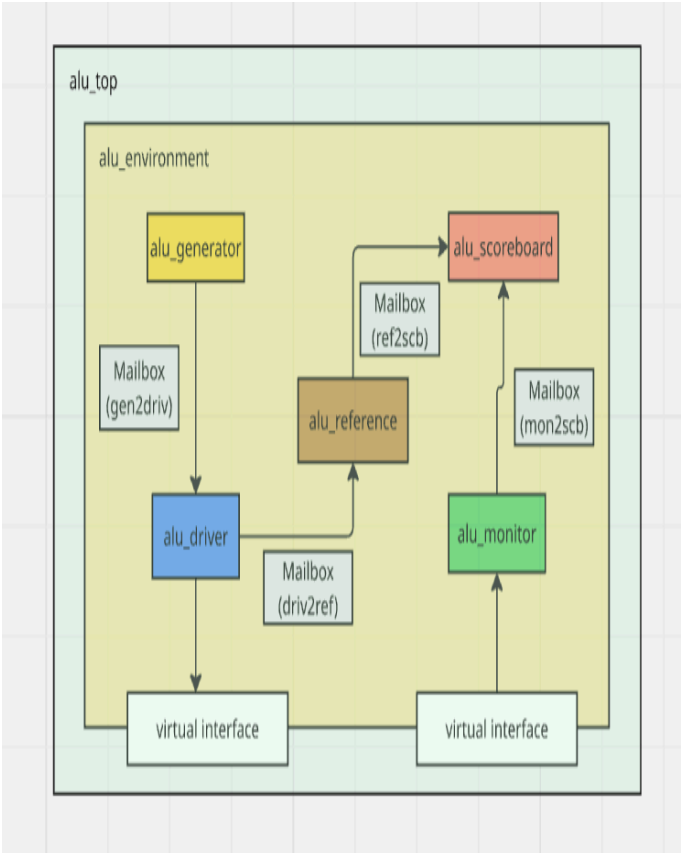


Figure 15: Working of Environment Class

h) Test:

The Test flowchart represents the execution flow for running the testbench. Mainly used to configure and coordinate all testbench components. It begins by creating the run task**,** which serves as the entry point. Next, an environmental handle object (env**)** is created using a constructor to manage all testbench components. The build task is initiated**,** which sets up and constructs the testbench elements. After the build phase, the start task is executed to trigger all verification components. Finally, the process ends with the completion of all tasks and the simulation stops.



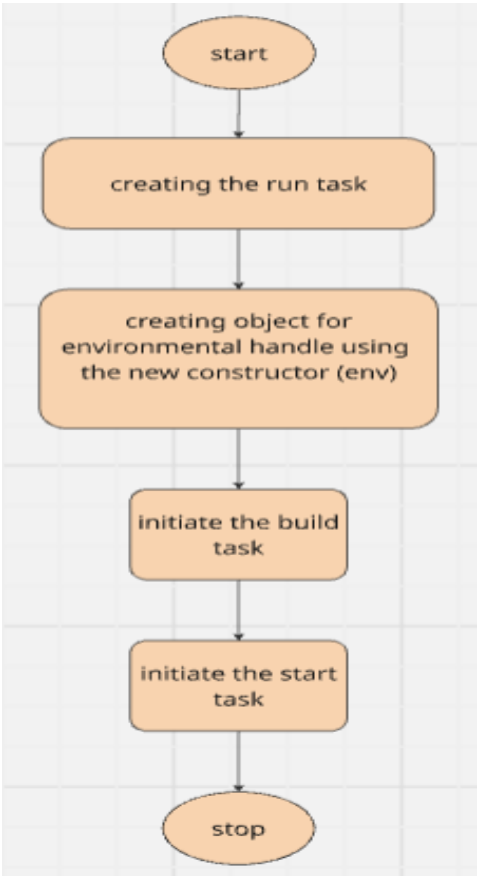
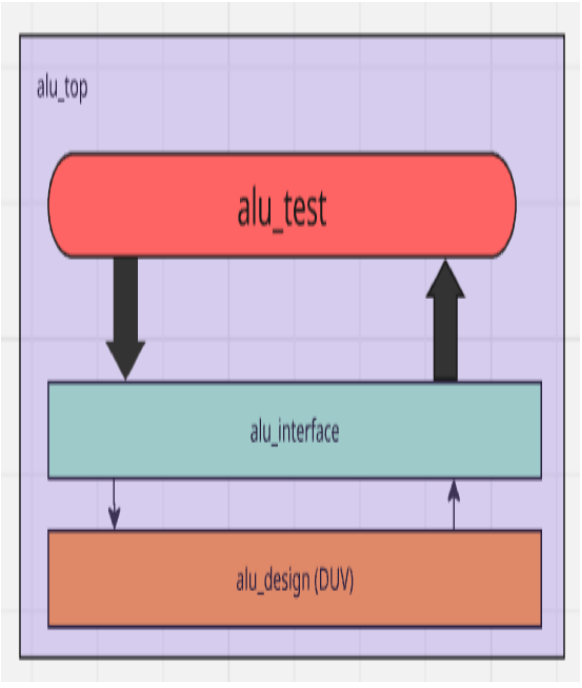


Figure 18: Working of Test Class

i) Top:



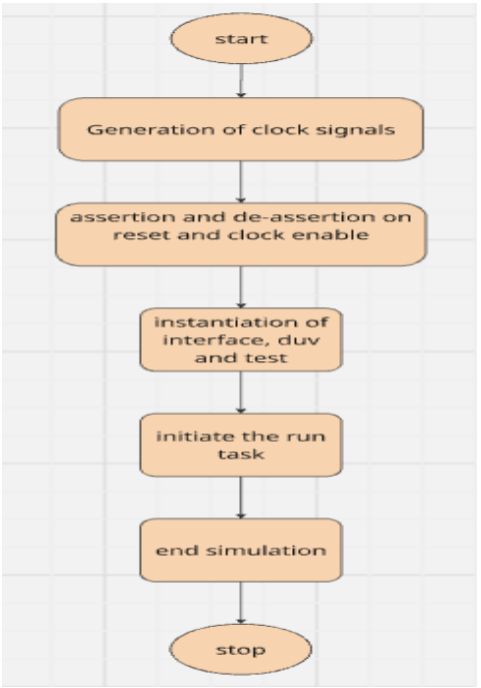


Figure 20: Working of Top Module

The Top flowchart outlines the top-level simulation process. The sequence starts with the generation of clock signals required for the testbench. Next, reset and clock enable signals are asserted and de-asserted to initialize the design. The interface, design under verification (DUV), and test components are instantiated. After this setup, the run task is initiated to execute the verification process. Finally, the simulation is terminated once all tasks are complete, leading to the stop state

j) Interface:

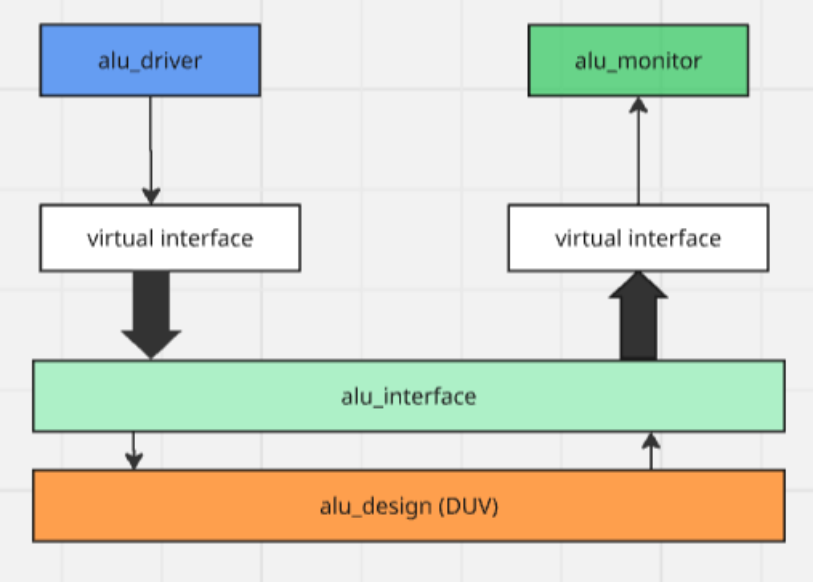


Figure 21: Interface Flowchart

The interface acts as a communication bridge between the testbench top module and the Design Under Verification (DUV), bundling all related signals into a single structured block. It includes the declaration of all input and output signals. Additionally, the interface defines clocking blocks for components like the driver, monitor, and reference model to ensure synchronized signal sampling and driving. Mod ports are also specified to control the direction and access of signals for each testbench component.

k) DUV (Design Under Verification):

The Design Under Verification (DUV) refers to the ALU module whose functional correctness is being tested against the design specifications. It is connected to the testbench through an interface module that links the DUV ports with the testbench components, such as the driver and monitor.

1. Test Plan

a) Test Scenarios:

Link:

https://docs.google.com/spreadsheets/d/1yfM7D9r4y9-2dNhEo2Sbz9A-pepWO2nO/edit?usp=drive_link&ouid=113766502478178390742&rtpof=true&sd=true

b) Functional Coverage Plan:

Link:

https://docs.google.com/spreadsheets/d/1yfM7D9r4y9-2dNhEo2Sbz9A-pepWO2nO/edit?usp=drive_link&ouid=113766502478178390742&rtpof=true&sd=true

c) Assertions:

Link:

https://docs.google.com/spreadsheets/d/1yfM7D9r4y9-2dNhEo2Sbz9A-pepWO2nO/edit?usp=drive_link&ouid=113766502478178390742&rtpof=true&sd=true

CHAPTER 3

VERIFICATION RESULTS AND ANALYSIS

3.1 Design Flaws:

During Normal Operation (without 16 clock cycle):

1) DUT Result width is not matching with the testbench result width.
2) DUT Functionality flaws:

Mode	Commands	Failure Reasons
Arithmetic	Sub	Overflow condition failed when both operands are equal
Arithmetic	Sub Cin	Overflow condition failed when both operands are equal
Arithmetic	INC_A	Increment A functionality failed
Arithmetic	INC_A	Failed to produce carry-out result on INC_A
Arithmetic	DEC_A	Failed to produce Overflow result on DEC_A
Arithmetic	INC_B	Increment B functionality failed
Arithmetic	INC_B	Failed to produce carry-out result on INC_B
Arithmetic	DEC_B	Decrement B functionality failed
Arithmetic	DEC_B	Failed to produce Overflow result on DEC_B
Arithmetic	Shift and multiply	Shift and multiply functionality failed
Logical	Logical OR	Failed to perform logical OR operation
Logical	Shift Right A	Shift right on A Operation failed
Logical	Shift Right B	Shift right on B Operation failed
Logical	Rotate right A on B	The error bit from [7:4] on operand B is failed

3) Input Valid Selection flaws:

   When input valid is 0, it will not produce the expected error result during single operand operation as well as two operand operation.

During 16 cycle Operation (with 16 clock cycle):

1) Whenever the 16-cycle operation is implemented on both arithmetic and logical operation it will not produce error result when input valid is 0 or 1 or 2. It will only pass when input valid is 3.

2) Whenever the count is greater than or equal to 16 and input valid is either 0, 1, or 2. It will not produce error result. It will pass when input valid is 3.

3.2 COVERAGE REPORT:

- **INPUT COVERAGE:**

Covergroup type:

input_coverage

Summary	Total Bins	Hits	Hit %
Coverpoints	39	39	100.00%
Crosses	16	16	100.00%

Search:

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
CARRY_IN	2	2	0	100.00%	100.00%	100.00%
CE	2	2	0	100.00%	100.00%	100.00%
COMMAND	25	25	0	100.00%	100.00%	100.00%
INP_VALID	4	4	0	100.00%	100.00%	100.00%
MODE	2	2	0	100.00%	100.00%	100.00%
OPA	1	1	0	100.00%	100.00%	100.00%
OPB	1	1	0	100.00%	100.00%	100.00%
RESET	2	2	0	100.00%	100.00%	100.00%

Search:

Crosses	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
CEXMODE	4	4	0	100.00%	100.00%	100.00%
INP_VALIDXMODE	8	8	0	100.00%	100.00%	100.00%
RESETXCE	4	4	0	100.00%	100.00%	100.00%

- **OUTPUT COVERAGE:**

Covergroup type:

output_coverage

Summary	Total Bins	Hits	Hit %
Coverpoints	13	10	76.92%
Crosses	0	0	0.00%

Search:

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
COUT	2	2	0	100.00%	100.00%	100.00%
EQUAL	2	1	1	50.00%	50.00%	50.00%
ERROR	2	2	0	100.00%	100.00%	100.00%
GREATER	2	1	1	50.00%	50.00%	50.00%
LESSER	2	1	1	50.00%	50.00%	50.00%
OFLOW	2	2	0	100.00%	100.00%	100.00%
RESULT	1	1	0	100.00%	100.00%	100.00%

- **ASSERTION COVERAGE:**

Assertions Coverage Summary:

Search:

Assertions	Failure Count	Pass Count	Attempt Count	Vacuous Count	Disable Count	Active Count	Peak Active Count	Status
/alu_top/assertion/ALU_LATCH	0	2276	10476	8200	0	0	2	Covered
/alu_top/assertion/ALU_reset	0	6476	10476	0	0	4000	4001	Covered
/alu_top/assertion/ALU_UNKNOWN	0	10475	10476	0	0	1	2	Covered
/alu_top_sv_unit/alu_generator/start/#anonblk#32361300#12#4#/#ublk#32361300#13/immed__14	0	2500	-	-	-	-	-	Covered
/work.alu_assertion/ALU_LATCH	0	2276	10476	8200	0	0	2	Covered
/work.alu_assertion/ALU_reset	0	6476	10476	0	0	4000	4001	Covered
/work.alu_assertion/ALU_UNKNOWN	0	10475	10476	0	0	1	2	Covered
/work.alu_top_sv_unit/alu_gene....361300#12#4#/#ublk#32361300#1/immed__14	0	2500	-	-	-	-	-	Covered

- **OVERALL COVERAGE:**

Total Coverage:					96.50%	**92.76%**
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Statements	120	117	3	1	97.50%	**97.50%**
Branches	73	70	3	1	95.89%	**95.89%**
FEC Conditions	20	16	4	1	80.00%	**80.00%**
Toggles	216	211	5	1	97.68%	**97.68%**

- **Overall Waveform:**