ALU VERIFICATION PLAN

-SHREYA

6108

CONTENTS	PAGE NUMBER

CHAPTER 1 - PROJECT OVERVIEW AND OBJECTIVES

ALU:

The Arithmetic Logic Unit, commonly known as the ALU, is a digital circuit that performs both arithmetic operations like addition and subtraction, and logical operations like AND, OR, and XOR. It forms a fundamental part of every computer system, microcontroller, or processor where data processing is involved.

1.1 PROJECT OVERVIEW:

The goal of this project is to verify ALU that can perform various arithmetic and logic functions. It includes custom commands, shift/rotate functionality, and intelligent error detection.

We started by understanding the functional requirements such as the need for ADD, SUB, CMP, INC, DEC, AND, OR, XOR, SHL, SHR, ROL, and ROR. These were mapped to the CMD signal with proper encoding for both arithmetic and logical modes.

Inputs like operand A and B, clock, reset, input validity, and carry-in are driven into the ALU. Based on the command selected and the operation mode, the ALU generates results along with flags like overflow, equality, carry, and error.

If the operand A is valid and if we get operand B with in or at 16^{th} clock cycle then the particular operation is valid otherwise the error flag will set high.

In this project, our main focus is on verifying it using a System Verilog based testbench. This ensures our ALU behaves correctly for all supported operations and under all possible input conditions.

1.2 VERIFICATION OBJECTIVE:

The primary goal of this verification is to ensure the correctness, reliability, and completeness of the ALU functionality as described in the design document. The verification process will validate each supported operation, check proper flag generation, handle error scenarios, and ensure the design meets timing and interface requirements.

A key objective is to test all arithmetic and logical operations under various input combinations. This includes operations like ADD, SUB, INC, DEC, AND, OR, XOR, shifts, and rotations. Both valid and invalid command sequences will be applied to confirm correct outputs and flag behavior. Each CMD value will be verified for both arithmetic (MODE=1) and logic (MODE=0) modes.

Another major objective is to verify error-handling logic. The timeout condition (if a second operand is not received within 16 clock cycles) must raise an error, and late input values should be handled correctly using last-priority logic.

The verification must also confirm correct behavior of output flags: COUT, OFLOW, G, L, and E. These outputs are essential for downstream decision-making and must reflect the operation result accurately. For example, the ALU must correctly set the OFLOW flag when an overflow occurs during addition or subtraction, and the comparator flags (G, L, E) must correctly indicate operand relationships.

1.3 DUT INTERFACE:

The ALU has a clearly defined set of input and output signals. The primary inputs are OPA and OPB (the two operands), CMD (the command), MODE (arithmetic/logical), INP_VALID (validity of operand input), CLK (clock), RST (reset), CIN (carry in), and CE (clock enable).

On the output side, RES gives the result of the operation, and several 1-bit status outputs (OFLOW, COUT, ERR) indicate special conditions like overflow, carry out, or invalid operation. The comparison outputs G, L, and E show the result of comparing operand A and B.

CHAPTER 2 TESTBENCH ARCHITECTURE AND METHODOLOGY

2.1 TESTBENCH ARCHITECTURE:

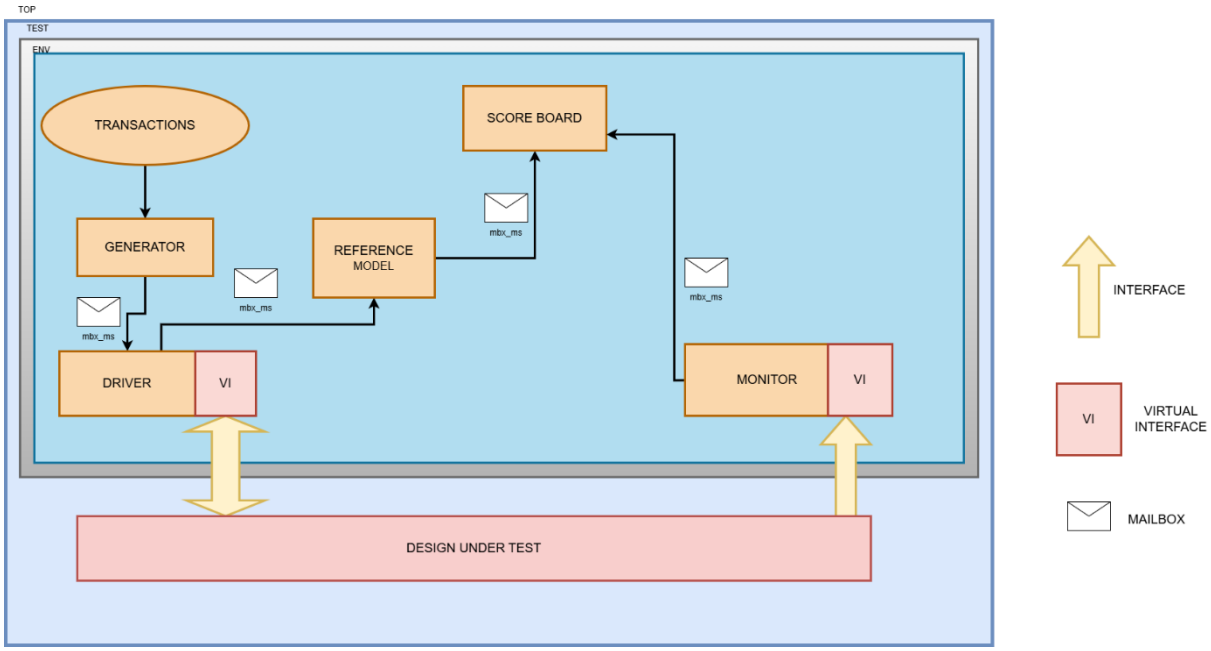


Fig 1: Testbench Architecture

The testbench is used to check if a digital design (called the Design Under Verification or DUV) works correctly. It includes a transaction generator that creates random test inputs, which are sent to a driver. The driver turns these inputs into signals and sends them to the DUV through a virtual interface. The DUV's output is observed by a monitor, which sends the data to a scoreboard. The scoreboard compares the actual output with the expected results from a reference model to see if the DUV is working properly. All these parts are organized and managed within an environment block to ensure they work together smoothly.

2.1 ALU TESTBENCH ARCHITECTURE:

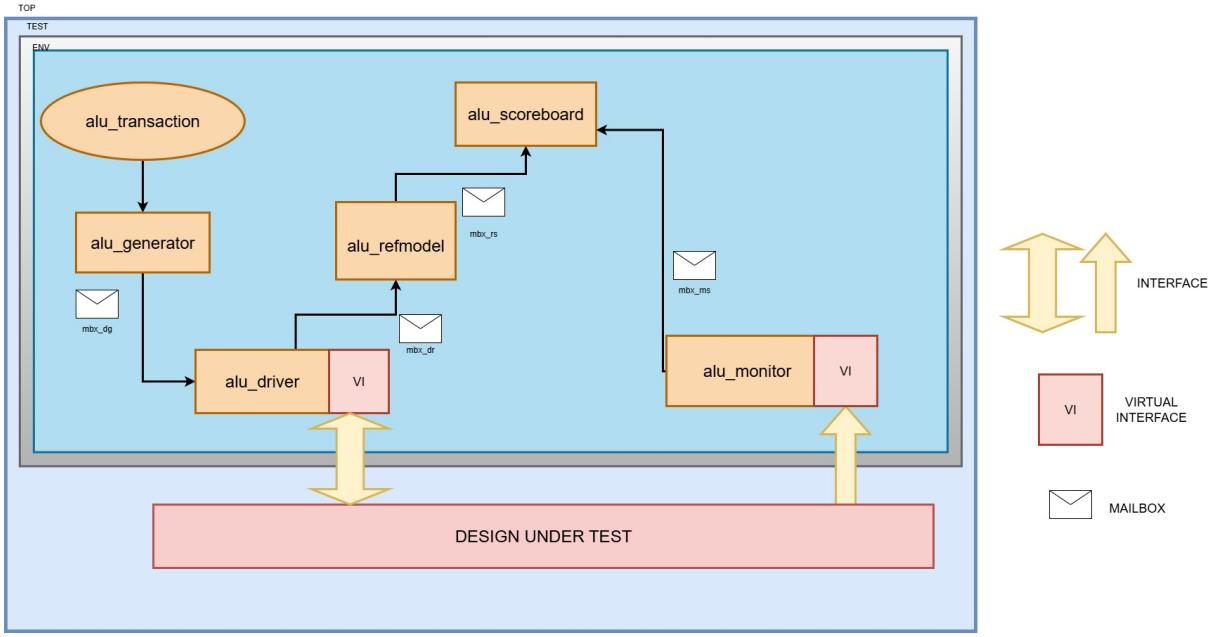


Fig 2: verification architecture

2.2 COMPONENT DETAILS AND FLOW CHART

Top:

*This is the top most module of the SV testbench architecture where control signals like clock and reset are generated

* Its code is written inside a module and the interface, the DUV and the test are instantiated in it

Test:

*This is the component where different test cases are written and run

* Here the environment is instantiated and built.

Environment:

*This is the component of the testbench which instantiates the generator, driver, monitor, reference model and scoreboard.

Transactions:

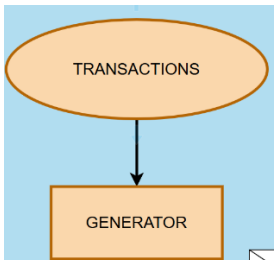


Fig 2 Transaction

* All the inputs and outputs of the Design Under Verification (DUV) are written inside the transaction class, excluding the clock and reset signals that are generated in the top module.

Generator:

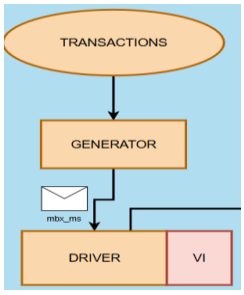


Fig 3 Generator

*This is a component of the testbench which generates constrained random stimuli (transactions) for the DUV.

* The generator sends the generated stimuli to the driver through a mailbox

Driver:

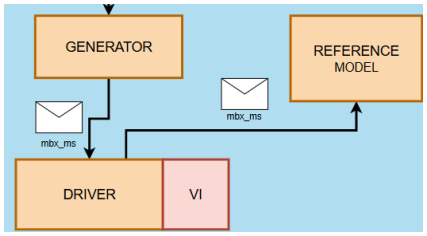


Fig 4 Driver

* It receives the generated transactions from the generator through a mailbox and drives it to the DUV through a virtual interface according to the DUV protocol

*It also sends the received transactions from the generator to the reference model through another mailbox.

Monitor:

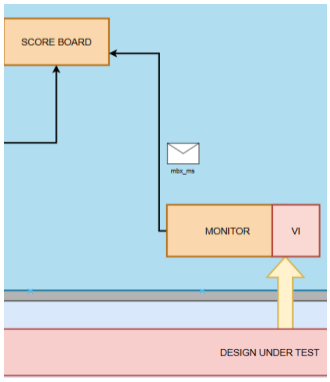


Fig 5 Monitor

* It collects the outputs of the DUV through a virtual interface and sends it to the scoreboard through a mailbox

Reference Model:

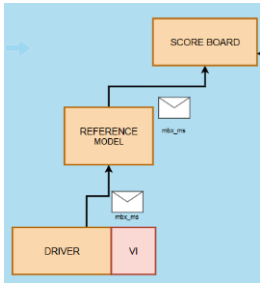


Fig 6 Reference Model

* This is a ‘golden’ model that mimics the functionality of the DUV and generates reference results used for comparison against simulation results.

Scoreboard:

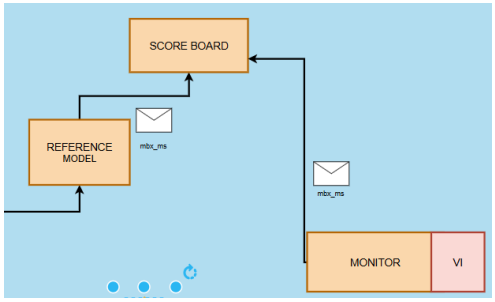


Fig 7 Scoreboard

*This is the component that collects the transactions (expected results) from the reference model through a mailbox.

* It collects the transactions (actual results) from the monitor through another mailbox.

* It compares the expected transactions with the actual transactions and generates a report.

Interface and Virtual interface:

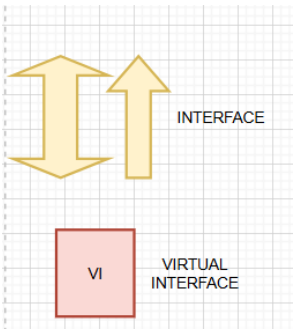


Fig 8 Interface

*The interface consists of all the input and output pins of the ALU

* The interface communicates with the driver and monitor in the testbench architecture

* The interface is static, whereas the testbench components driver and monitor which are written inside classes are dynamic. Therefore to connect a static interface with a dynamic testbench component, a Virtual Interface (VI) is used.

Mailbox:

Mailboxes provide a traction-level communication channel between various testbench and components.

CHAPTER 3 VERIFICATION RESULTS AND ANALYSIS

3.1 Design Bugs

1. 16-Clock Cycle Timeout Logic

Error flag is not set even after waiting for 16 clock cycles when the expected timeout condition occurs.

2. Single-Operand Operations

DUT does not perform increment/decrement operations when only the corresponding input valid signal is high. It only works when INP_VALID == 2'b11, which violates the design specification.

3. Carry-Out Logic for Increment Operations

COUT logic is incorrect or missing.

It remains at the default value.

COUT flag should assert when operand reaches the maximum value (e.g., 255 incrementing to 256).

4. Increment A Operation

INC_A operation holds the same value as OPA, instead of incrementing it.

This results in no change to the operand value.

5. Increment B and Decrement B Operations

INC_B operation incorrectly decrements the value instead of incrementing.

6. Shift Operation

Right shift operations on both OPA and OPB fail.

Left shift OPB operation also fails to execute correctly.

7. Decrement B Operations

DEC_B operation incorrectly increments the value instead of decrementing.

8. Overflow Logic for Decrement Operations

OVER_FLOW logic is incorrect or missing.

It remains at the default value.

OVER_FLOW flag should assert when operand at minimum value (e.g., 0) is decremented to -1.

9. Carry-In Signal Timing

CIN signal appears with a one-clock-cycle delay in addition to CIN and subtraction with CIN operations.

This causes timing misalignment.

10. Rotate Right A by B Error Condition

ROR_A_B error condition detection is incorrect and always remains zero.

It fails to set appropriate error flags even when an error condition occurs.

11. Multiplication Operation

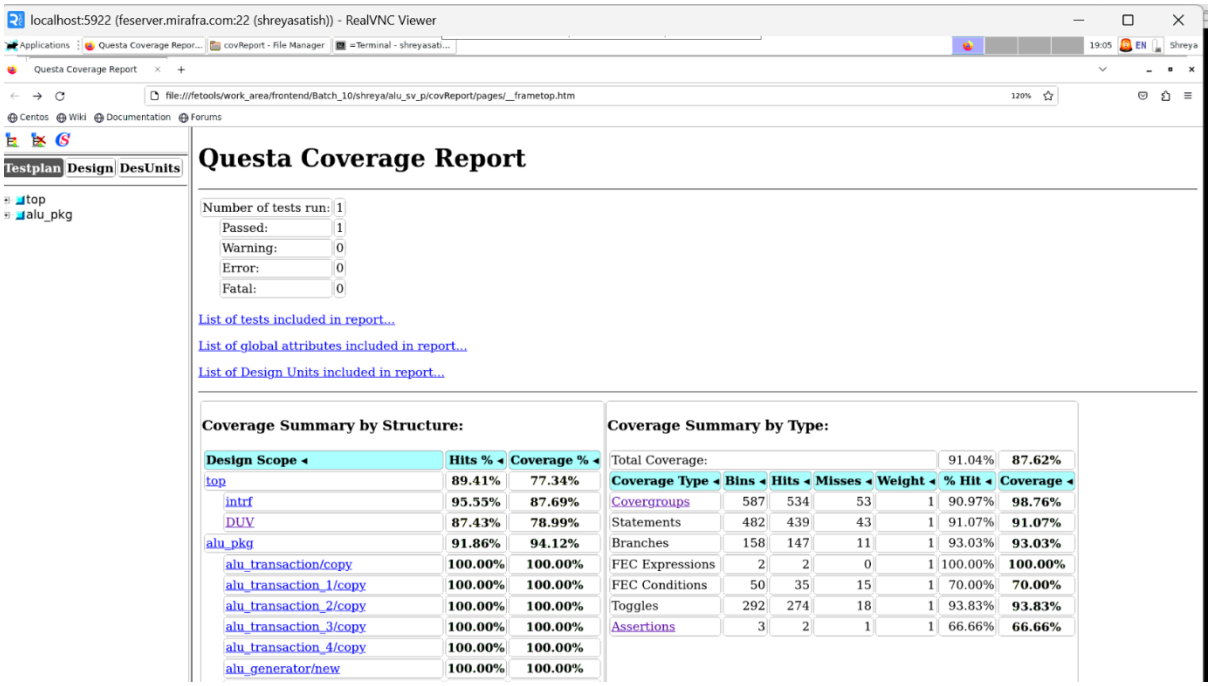
Issues exist in the multiplication functionality that affect correct arithmetic results.

12. Logical OR

Logical OR operation fails to execute correctly even with valid inputs and proper timing.

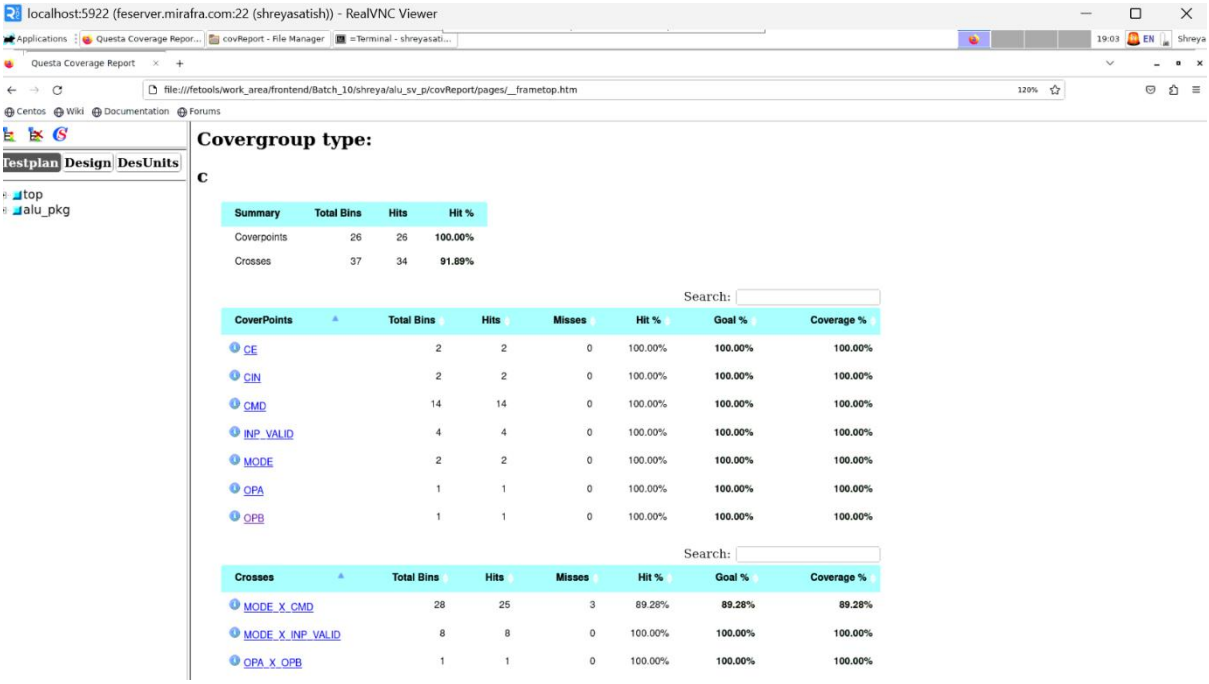
3.2 COVERAGE REPORT:

- **Overall Coverage:**

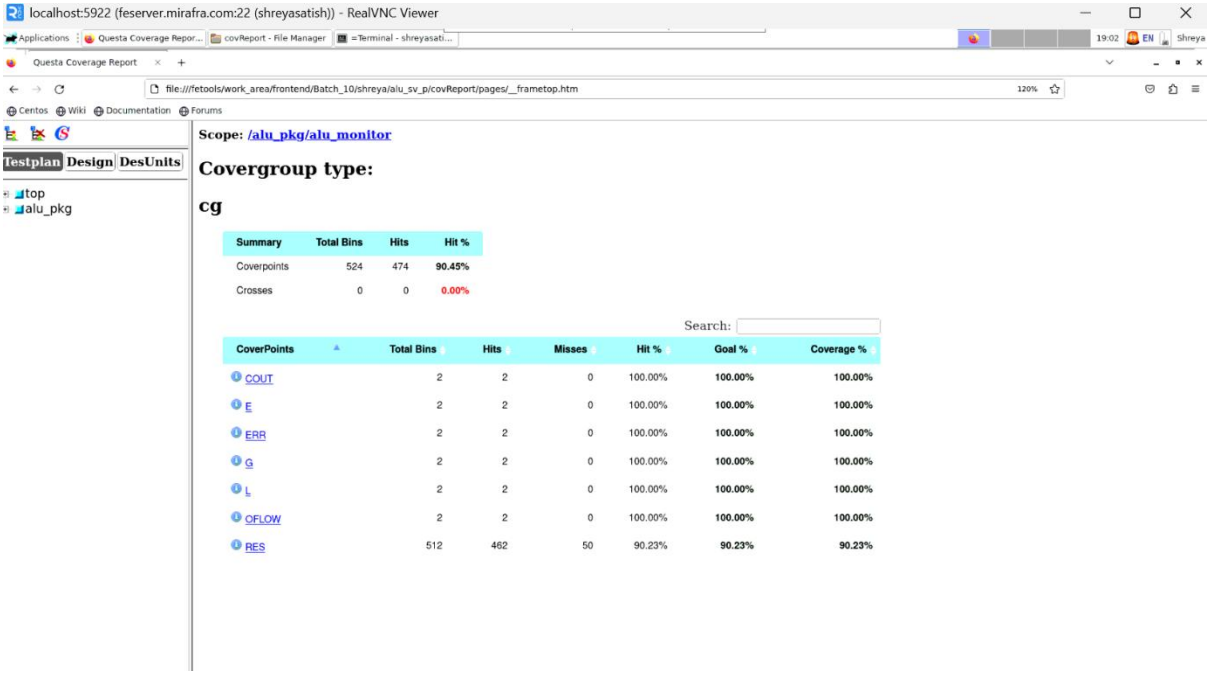


- **Functional Coverage:**

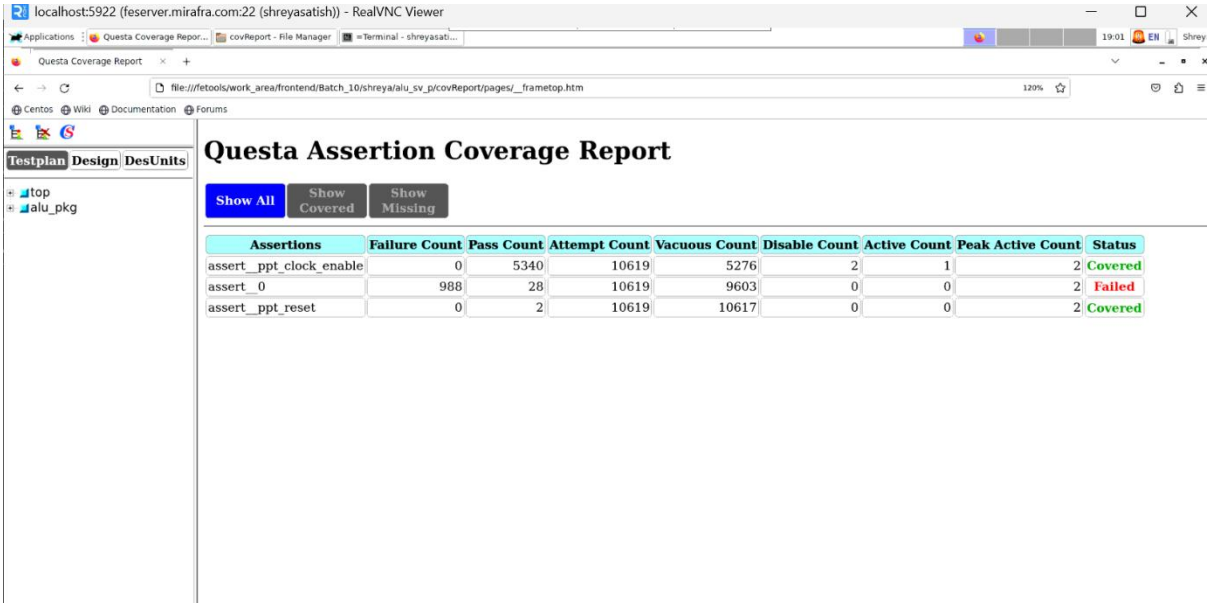
Input coverage:



Output coverage:



- **Assertion Coverage:**



Assertions	Failure Count	Pass Count	Attempt Count	Vacuous Count	Disable Count	Active Count	Peak Active Count	Status
assert__ppt_clock_enable	0	5340	10619	5276	2	1	2	Covered
assert__0	988	28	10619	9603	0	0	2	Failed
assert__ppt_reset	0	2	10619	10617	0	0	2	Covered

3.3 Output waveform: