ALU VERIFICATION DOCUMENT

CONTENTS

CHAPTER 1

1.1 Project Overview:

This project describes the complete testing and validation process for a parameterized Arithmetic Logic Unit (ALU) utilizing an advanced, class-based SystemVerilog verification environment. The core purpose is to confirm the operational accuracy and temporal performance of the ALU implementation through a well-organized and scalable testing methodology**.**

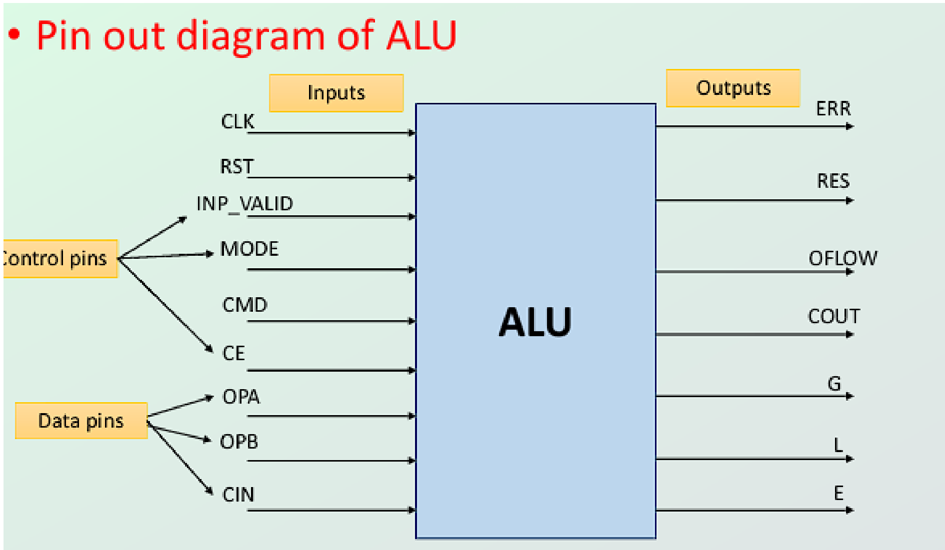
1.2 Verifcation Objectives:

- **Functional Verification:**
  1. Arithmetic Operations.
  2. Logical Operations.
  3. Control and Interface:
     - MODE switching between arithmetic and logical modes.
     - INP_VALID combinations (00, 01, 10, 11) impact verification.
     - Clock enable and reset behavior testing.
     - 16-cycle timeout mechanism for missing operands.
  4. Error Handling:

- **Functional Coverage**:
  - 100% command coverage across both modes
  - All INP_VALID state transitions
  - Comprehensive error condition coverage

- **Verification Stratergy:**
  - Constrained random stimulus generation
  - Self-checking testbenches with reference models
  - Directed tests for corner cases
  - Assertion-based protocol verification
  - Coverage-driven methodology

1.3 DUT Interface:



- **Input Ports**

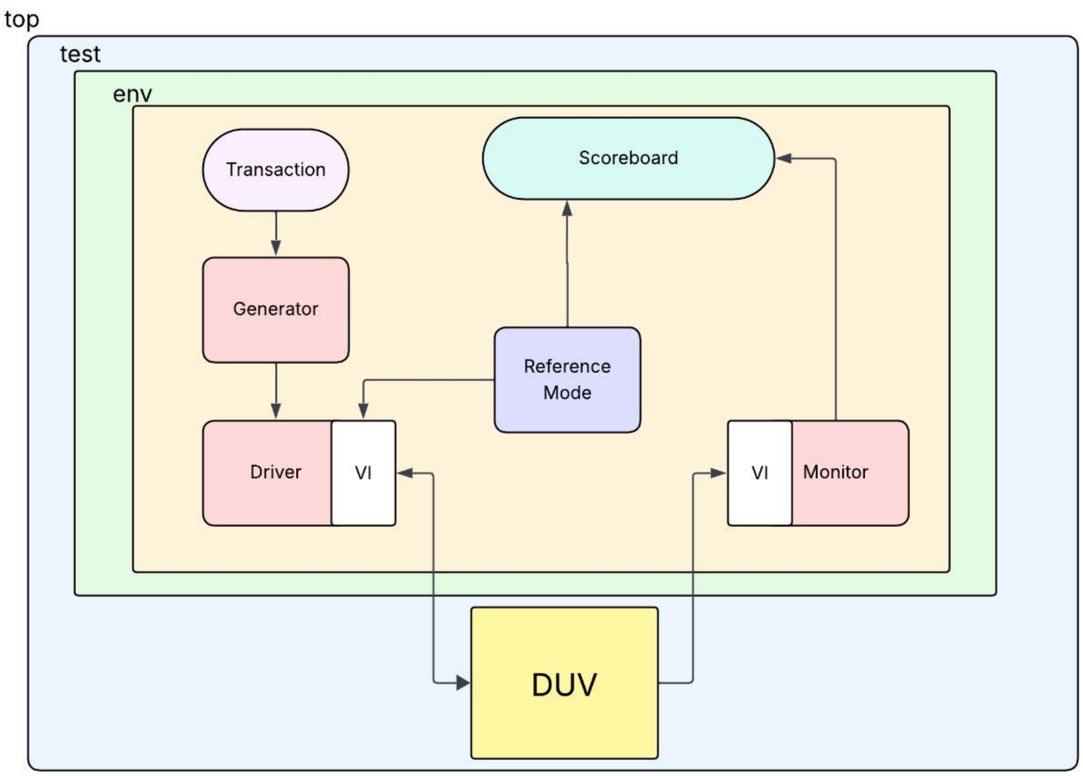
Signal Name	Type	Width (bits)	Description
INP_VALID	Input	2	Input valid signal - indicates when input data is valid
MODE	Input	1	Mode selection signal - determines ALU operation mode
CMD	Input	4	Command signal - specifies the specific ALU operation
OPA	Input	Parametrized	Operand A - first arithmetic/logic operand
OPB	Input	Parametrized	Operand B - second arithmetic/logic operand
CIN	Input	1	Carry In - input carry for arithmetic operations

- **Output ports:**

Signal Name	Type	Width (bits)	Description
ERR	Output	1	Error signal - indicates if an error occurred during operation
RES	Output	Parametrized	Result - the output result of the ALU operation
OFLOW	Output	1	Overflow - indicates arithmetic overflow condition
COUT	Output	1	Carry Out - output carry from arithmetic operations
G	Output	1	Greater than - comparison result flag
L	Output	1	Less than - comparison result flag
E	Output	1	Equal - comparison result flag

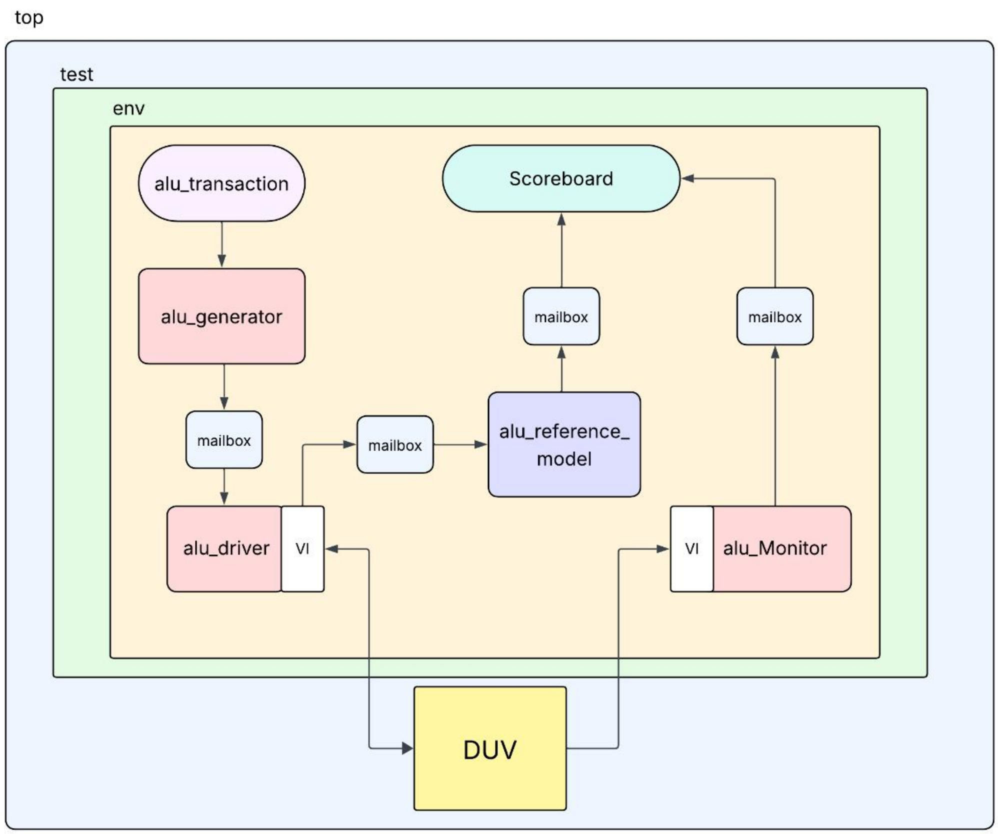
CHAPTER 2 - Verification Architecture

2.1 Verification architecture:



The figure shows general testbench architecture used for verifying digital designs. At the top module is the Design Under Verification (DUV). The test has the testbench environment that includes a transaction generator that creates and manages the input testcases for testing. These transactions are randomized within the generator and then transmitted to the driver, which applies them as signals to the DUV through a virtual interface. Outputs from the DUV are monitored by a monitor, which forwards this data to a scoreboard for checking correctness against expected results calculated by a reference model. The environment block encapsulates all these components, ensuring coordination among them, while the scoreboard oversees the entire verification process to validate the DUV.

2.2 Verification ALU architecture:



The figure illustrates the **Verification Architecture** for an Arithmetic Logic Unit (ALU) using a modular testbench environment.

Key Components:

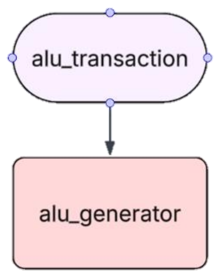
- **alu_transaction**:
  Defines the input and output of transactions (e.g., operands and operations) exchanged between components.
- **alu_generator**:
  Randomly creates test scenarios by generating transactions. These are sent to the driver for processing.

- **alu_driver**:
  Receives transactions from the generator and drives them to the DUV and reference model using a **virtual interface (VI)**
- **DUV (Design Under Verification)**:
  The actual ALU design that receives inputs from the driver and generates outputs for validation.
- **alu_monitor**:
  Observes and captures the output signals from the DUV through the virtual interface. It converts them back into transaction format and forwards them to the scoreboard.
- **alu_reference_model**:
  Serves as a golden model that receives the same input as the DUV and produces the expected output for comparison.
- **Scoreboard**:
  Compares the output from the DUV (via the monitor) with the expected output from the reference model. Any mismatches are flagged as functional errors.
- **Mailboxes**:
  Facilitate communication between generator, driver, monitor, reference model, and scoreboard by passing transactions.

2.3 FLOW CHART OF SV COMPONENTS :

1. Transaction Class:

The **`alu_transaction`** class encapsulates all ALU input stimuli and output responses for verification purposes.



Components

Randomized Input Stimuli: The transaction contains ALU input signals declared with the `rand` keyword for automatic randomization:

- INP_VALID, MODE, CMD, OPA, OPB, CIN: Input signals that will be randomized by the generator

Non-Randomized Output Signals

Output signals are declared without `rand` as they represent the ALU's response:

- ERR, RES, OFLOW, COUT, G, L, E: Output signals monitored for verification

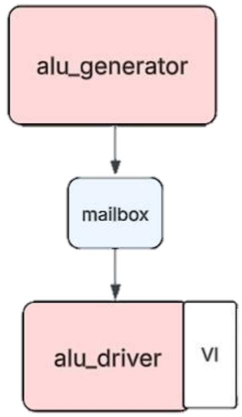
Constraints

The class implements mode-dependent constraints for realistic test scenarios:

Deep Copy Method

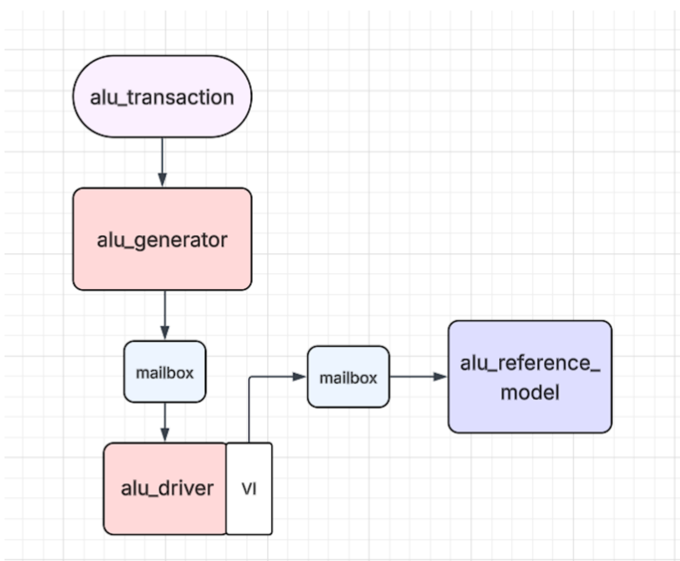
Implements a `copy()` function following the blueprint pattern for creating independent transaction copies used across testbench components.

2. Generator Class:



The generator consists of the ALU transaction class handle, the mailbox handle which connects to the driver, and the start() task which randomizes transactions and sends the randomized transactions to the driver through the mailbox.

3. Driver Class:



Key Features Implemented:

1. Two Mailboxes

- `gen_to_drv_mbox`: Transfers transactions from generator to driver
- `drv_to_ref_mbox`: Sends transactions from driver to reference model

2. Virtual Interface (Dynamic)

- `alu_if`: SystemVerilog interface with clocking blocks
- `valu_if`: Virtual interface for dynamic access
- Communicates between testbench and DUV with proper synchronization

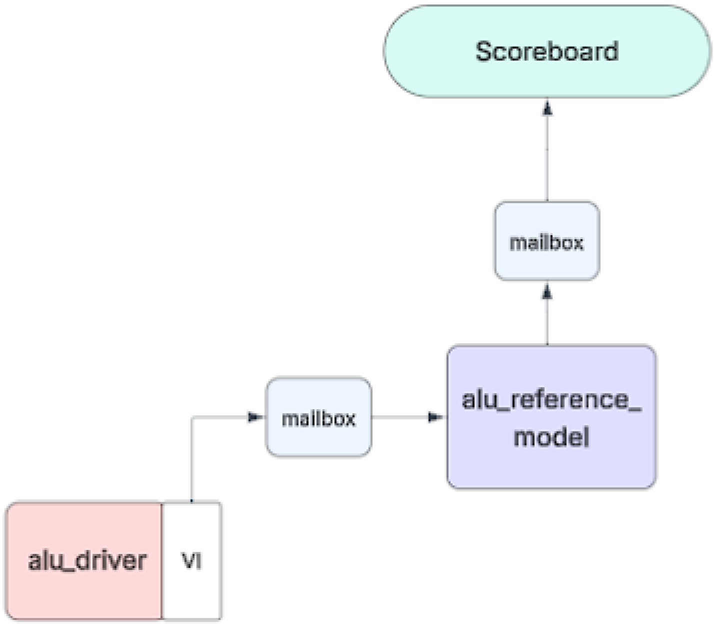
3. Functional Coverage Group

- Covers all input combinations (operands a, b, operation).
- Edge case coverage (zero results, overflow, carry conditions).

4. Drive Task in ALU Driver

- `drive_transaction()`: Drives stimuli to the DUV.

4. Reference Model:



Serves as a golden model that receives the same input as the DUV and produces the expected output for comparison.

Key Features:

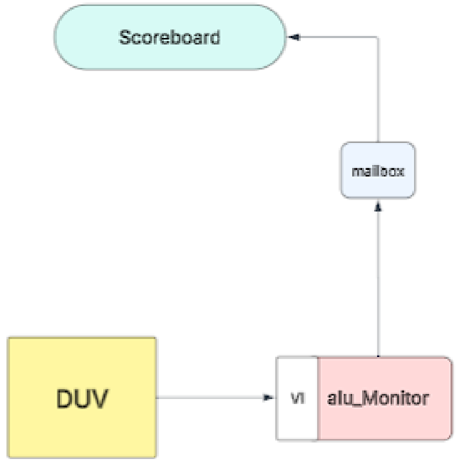
1. Two Mailboxes

1. **drv_to_ref_mbox**: Receives transactions from the driver with input stimuli
2. **ref_to_scb_mbox**: Sends processed transactions with expected results to the scoreboard

2. Functionality

- Task **: start()**, Continuously gets transactions from driver, computes expected results, and forwards to scoreboard
- Task to permorm operations**: task compute_expected_result()**: Implements golden reference for all ALU operations (ADD, SUB, MUL, DIV, AND, OR, XOR, NOT)

5. Monitor:



The ALU Monitor captures transactions from the DUV interface and forwards them to the scoreboard for result comparison.

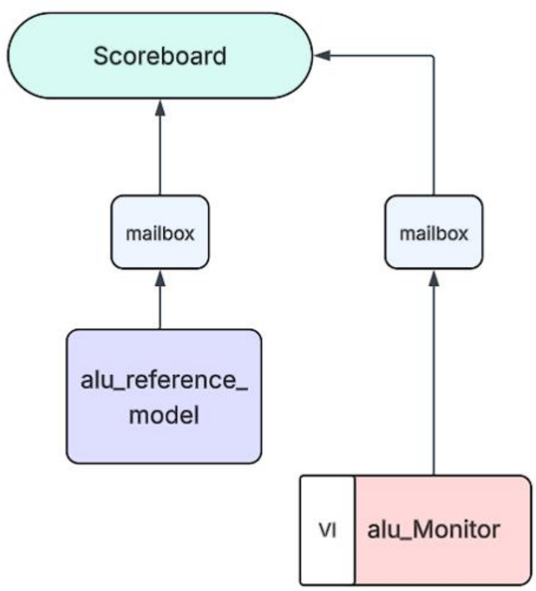
Mailbox Communication

- **mon_to_scb_mbox**: Sends captured DUV transactions to the scoreboard

Functionality

- **start()**: Continuously monitors DUV interface and captures completed transactions
- Samples DUV outputs when valid transaction completes and creates transaction objects

6. Scoreboard:



The ALU Scoreboard compares actual DUV results against expected results from the reference model to determine test pass/fail status.

Mailbox Communication

- **`ref_to_scb_mbox`**: Receives expected results from the reference model
- **`mon_to_scb_mbox`**: Receives actual results from the monitor

Functionality

- **`compare_results()`**: Compares expected vs actual results and reports mismatches
- **`report()`**: Provides final test statistics (pass/fail counts, coverage)

CHAPTER 3 :

3.1.1 Design Bugs:

1. The 16-clock cycle logic does not work because the error flag is not set even after waiting for 16 clock cycles.
2. Additionally, the DUT does not perform single-operand operations, such as increment or decrement, even when the corresponding input valid signal is high. It only works when INP_VALID == 2'b11. However, the design specification states that for commands associated with single-operand operations, only that operand and its corresponding valid signal should be driven.
3. The COUT logic is incorrect/missing in the case of an increment operation,it is always at default value. If the operand is at its maximum value (e.g., 255), incrementing it results in 256, which the RES register can accommodate. However, the COUT flag should still be asserted in this case.
4. While performing an increment operation with INC_A, the result holds the same value as OPA instead of incrementing it.
5. For INC_B, the operation incorrectly decrements the value, and for DEC_B, it incorrectly increments the value.
6. The OVER_FLOW logic is incorrect/missing for the decrement operation, it is always at default value. If the operand is at its minimum value (e.g., 0), decrementing it results in -1, which the RES register can represent in two’s complement. However, the OVER_FLOW flag should still be asserted in this scenario.
7. The CIN signal appears with a one-clock-cycle delay in the case of addition with CIN and subtraction with CIN.
8. The ROR_A_B error condition is incorrect, as it always remains zero even when the error condition occurs.
9. There are issuses in the multiplication operations.

3.2 Coverage Report:

- **Overall coverage:**

Questa Coverage Report

Number of tests run:	1
Passed:	1
Warning:	0
Error:	0
Fatal:	0

List of tests included in report...

List of global attributes included in report...

List of Design Units included in report...

Coverage Summary by Structure:

Design Scope	Hits %	Coverage %
top	**93.64%**	**86.36%**
inf	**96.55%**	**98.21%**
DUT	**92.47%**	**85.87%**
alu_pkg	**100.00%**	**100.00%**
alu_driver	**100.00%**	**100.00%**
alu_monitor	**100.00%**	**100.00%**

Coverage Summary by Type:

Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Total Coverage:					94.96%	**89.09%**
Covergroups	140	140	0	1	100.00%	**100.00%**
Statements	150	146	4	1	97.33%	**97.33%**
Branches	73	68	5	1	93.15%	**93.15%**
FEC Conditions	18	11	7	1	61.11%	**61.11%**
Toggles	294	276	18	1	93.87%	**93.87%**

Report generated by Questa (ver. 10.6c) on Mon 28 Jul 2025 02:55:56 PM IST with command line:
vcover report -html coverage.ucdb -htmldir covReport -details

- **Code coverage:**

Questa Design Coverage

Scope: /top/DUT

Instance Path:
/top/DUT
Design Unit Name:
work.ALU_DESIGN
Language:
Verilog
Source File:
top.sv

Local Instance Coverage Details:

Total Coverage:					92.47%	**85.87%**
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Statements	117	113	4	1	96.58%	**96.58%**
Branches	73	68	5	1	93.15%	**93.15%**
FEC Conditions	18	11	7	1	61.11%	**61.11%**
Toggles	204	189	15	1	92.64%	**92.64%**

- **Functional Coverage:**

Scope: /alu_pkg/alu_driver

Covergroup type:

drv_cg

Summary	Total Bins	Hits	Hit %
Coverpoints	28	28	**100.00%**
Crosses	101	101	**100.00%**

Search:

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
CIN_CP	2	2	0	100.00%	**100.00%**	**100.00%**
CMD_CP	14	14	0	100.00%	**100.00%**	**100.00%**
INP_VALID_CP	4	4	0	100.00%	**100.00%**	**100.00%**
MODE_CP	2	2	0	100.00%	**100.00%**	**100.00%**
OPA_CP	3	3	0	100.00%	**100.00%**	**100.00%**
OPB_CP	3	3	0	100.00%	**100.00%**	**100.00%**

Search:

Crosses	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
CMD_X_IP_V	56	56	0	100.00%	**100.00%**	**100.00%**
MODE_X_CMD	28	28	0	100.00%	**100.00%**	**100.00%**
MODE_X_INP_V	8	8	0	100.00%	**100.00%**	**100.00%**
OPA_X_OPB	9	9	0	100.00%	**100.00%**	**100.00%**

Scope: /alu_pkg/alu_monitor

Covergroup type:

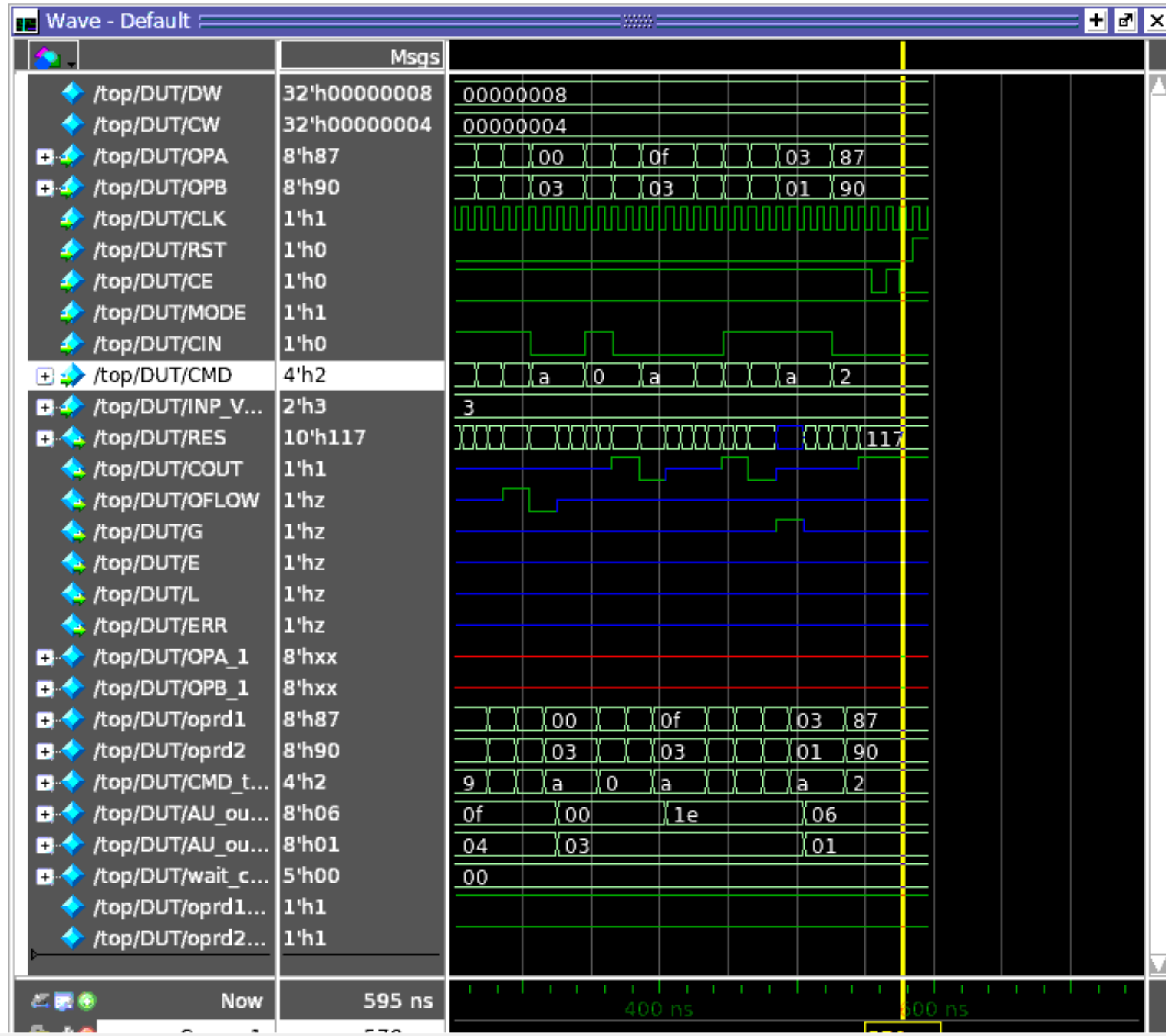
mon_cg

Summary	Total Bins	Hits	Hit %
Coverpoints	11	11	**100.00%**
Crosses	0	0	**0.00%**

Search:

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
COUT_CP	2	2	0	100.00%	**100.00%**	**100.00%**
E_CP	1	1	0	100.00%	**100.00%**	**100.00%**
ERR_CP	2	2	0	100.00%	**100.00%**	**100.00%**
G_CP	1	1	0	100.00%	**100.00%**	**100.00%**
L_CP	1	1	0	100.00%	**100.00%**	**100.00%**
OV_CP	2	2	0	100.00%	**100.00%**	**100.00%**
RES_CP	2	2	0	100.00%	**100.00%**	**100.00%**

3.3 Output waveform:



3.4 Verification Plan link:

Verification_Plan