ALU VERIFICATION DOCUMENT

CONTENTS

CHAPTER 1: PROJECT OVERVIEW AND OBJECTIVES

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 UVM 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:

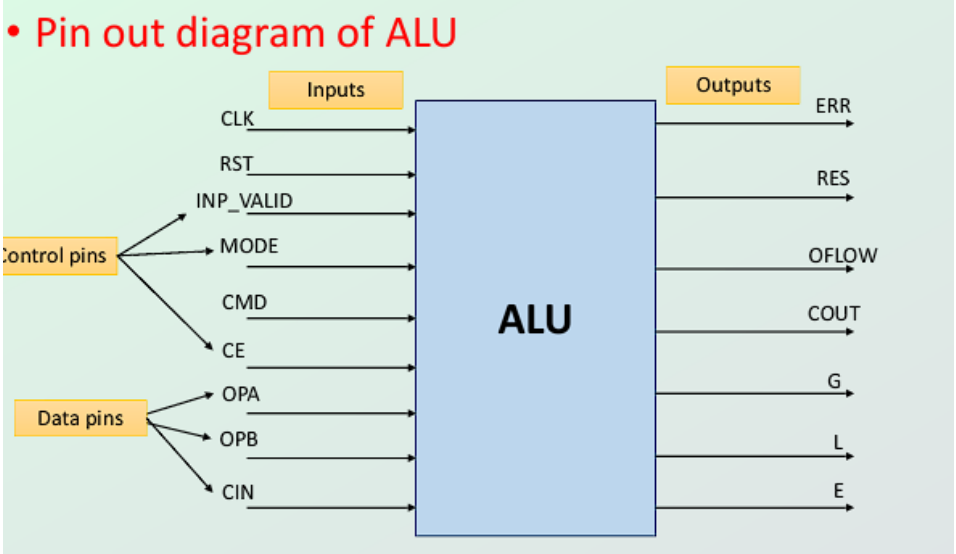
- **Functional Verification:**
  1. Arithmetic Operations.
  2. Logical Operations.
  3. Control and Interface:
     - MODE switching between arithmetic and logical modes.
     - INP_VALID combinations (00, 01, 10, 11) impact verification.
     - Clock enable and reset behavior testing.
     - 16-cycle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

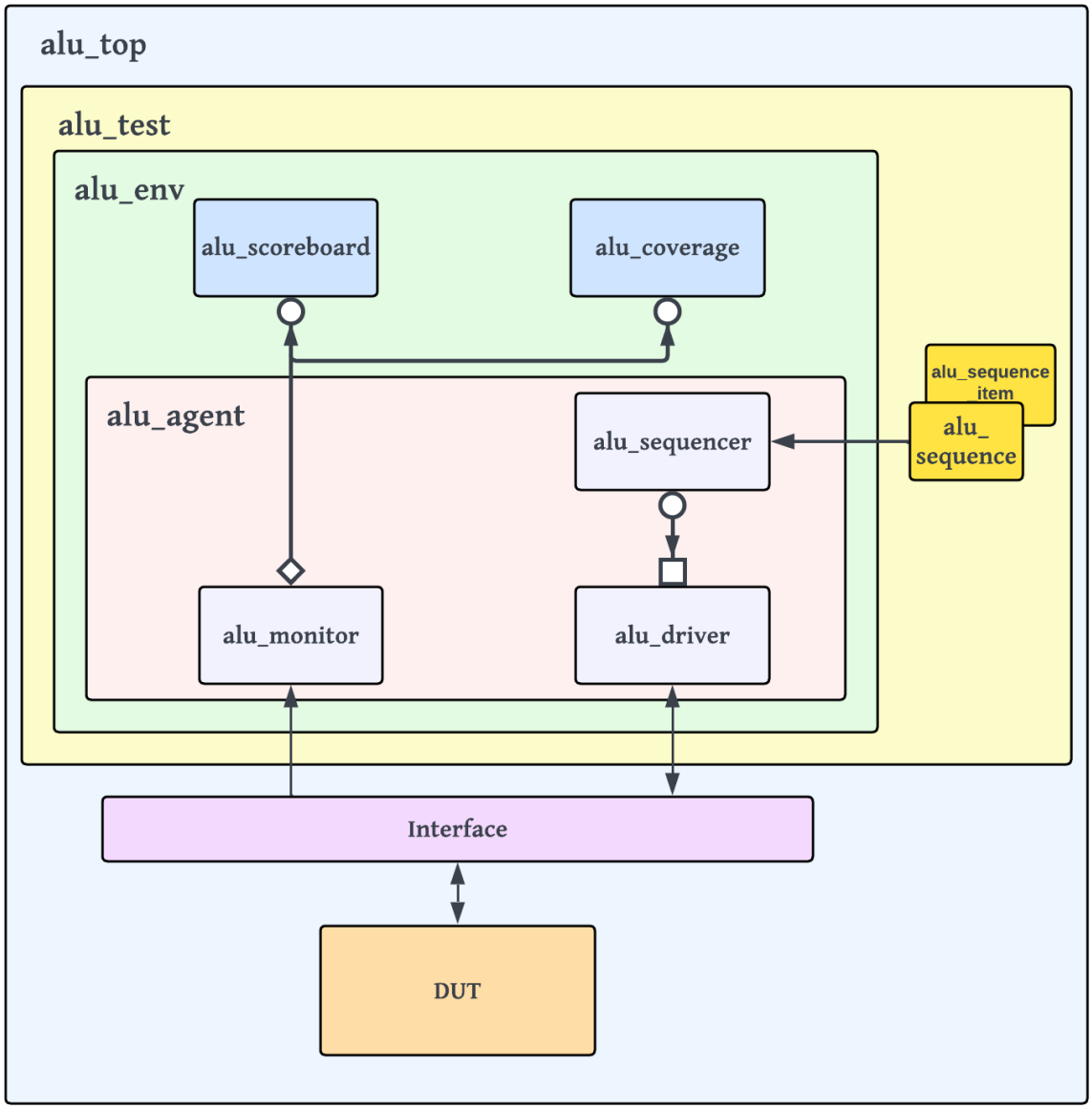
Signal Name	Type	Width (bits)	Description
INP_VALID	Input	2	Input valid signal - indicates when input data is valid
MODE	Input	1	Mode selection signal - determines ALU operation mode
CMD	Input	4	Command signal - specifies the specific ALU operation
OPA	Input	Parametrized	Operand A - first arithmetic/logic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

1.4 Verification architecture:



The figure shows the **Verification Architecture** for an Arithmetic Logic Unit (ALU) built using a **modular UVM testbench** approach.

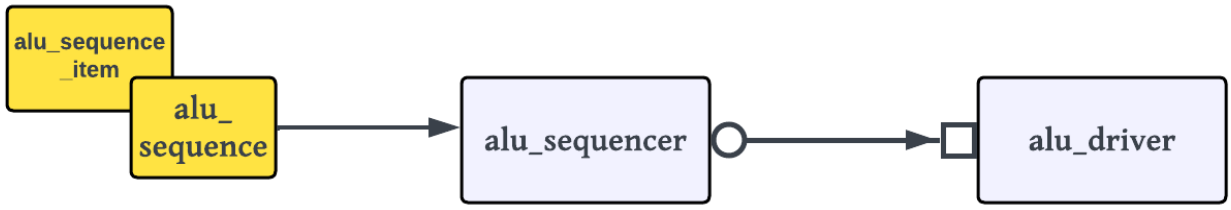
Key Components

- **Sequence Item** – A user-defined transaction object containing the fields for a single operation, such as input data and control signals.
- **Sequence** – Generates a series of sequence items to stimulate the DUT. It is launched from the **test** to initiate stimulus generation.
- **Sequencer** – Acts as a link between the sequence and the driver, transferring sequence items via a TLM export. It ensures that generated transactions are delivered to the driver.
- **Driver** – Converts the transactions from the sequencer (via a TLM port) into pin-level signals for the DUT, interacting through the **interface**.
- **Monitor** – Passively observes DUT outputs through the virtual interface, converts them into transaction format, and sends them to the scoreboard via a TLM port.
- **Agent** – A reusable block that encapsulates the driver, sequencer, and monitor.
- **Scoreboard** – Compares DUT outputs (from the monitor) with expected results from the **reference model**. Any mismatches are flagged as functional errors.
- **Coverage Component** – Records functional coverage to ensure test scenarios are well exercised. It connects to both the driver and monitor via TLM analysis ports.
- **Environment** – A container that organizes agents, scoreboards, and other verification components into a coherent structure.
- **Test** – The top UVM testbench component that builds the environment, sets configurations, and initiates stimulus generation.
- **Top** – Instantiates the interface and the DUT, and triggers the UVM phasing mechanism.
- **Reference Model** – A golden model that takes the same inputs as the DUT and produces the expected outputs for comparison in the scoreboard.
- **DUV (Design Under Verification)** – The ALU design being tested, which receives inputs from the driver and produces outputs for validation.

- **Interface** – Connects testbench components to the DUT's signals.

2.2 FLOW CHART OF SV COMPONENTS

1) Sequence-Sequencer-Driver



The process begins with the **seq_item**, which specifies the transaction data, followed by the **alu_sequence**, which creates these transactions. These transactions are sent to the **alu_sequencer**, which manages their order and timing. The sequencer then delivers each transaction to the **alu_driver** through a handshake mechanism to maintain synchronization. Finally, the **alu_driver** translates the abstract transaction data into pin-level signals and drives them onto the DUT via the interface.

2) Monitor-Scoreboard:

In this stage of the flow, the **alu_monitor** observes the signal activity on the DUT interface and translates it into transaction-level data. It forwards these transactions to the **alu_scoreboard** via its TLM analysis port. The **alu_scoreboard**, through its TLM analysis implementation port, receives the transactions and uses an internal **reference model** to generate the expected output for the given inputs. It then compares the DUT's actual output (from the monitor) against the expected output (from the reference model) to verify the correctness of the design's functionality.

3) **Coverage**

In this part of the flow, the **alu_monitor** also sends transaction-level data to the **alu_coverage** component via their respective TLM analysis ports. The monitor provides observed DUT outputs and activity, which the **alu_coverage** component receives through its TLM analysis implementation ports. These transactions are then sampled in the component's **covergroups**, enabling the tracking of functional coverage for both input scenarios and output results. This ensures that all intended test cases and corner conditions are exercised during verification.

CHAPTER 3 : RESULTS AND ANALYSIS

3.1 Design Bugs:

S.No	Test Case Category	Reason for Failure
1	16-Clock Cycle Timeout Logic	Error flag is not set even after waiting for 16 clock cycles when expected timeout condition occurs.
2	Single-Operand Operations	DUT does not perform single-operand operations (increment/decrement) when only the corresponding input valid signal is high; only works when INP_VALID == 2'b11, violating design specification requirements.
3	Carry-Out Logic for Increment Operations	COUT logic is incorrect/missing in increment operations and remains at default value; COUT flag should be asserted when operand reaches 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, resulting 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 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/missing for decrement operations and remains at default value; OVER_FLOW flag should be asserted 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 with CIN and subtraction with CIN operations, causing timing misalignment.
10	Rotate Right A by B Error Condition	ROR_A_B error condition detection is incorrect and always remains zero even when the error condition occurs, failing to set appropriate error flags.
11	Multiplication Operation	Issues present in the multiplication operation functionality.
12	Logical OR	Logical OR operation fails to execute correctly even with valid inputs and proper timing.

3.2 Coverage Report:

- **Overall coverage**

Coverage Summary by Type:

Total Coverage:					95.38%	**88.85%**
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Covergroups	140	138	2	1	98.57%	**98.88%**
Statements	141	136	5	1	96.45%	**96.45%**
Branches	73	70	3	1	95.89%	**95.89%**
FEC Conditions	18	12	6	1	66.66%	**66.66%**
Toggles	294	280	14	1	95.23%	**95.23%**
Assertions	5	4	1	1	80.00%	**80.00%**

- **Code coverage:**

Local Instance Coverage Details:

Total Coverage:					94.41%	**88.65%**
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Statements	117	114	3	1	97.43%	**97.43%**
Branches	73	70	3	1	95.89%	**95.89%**
FEC Conditions	18	12	6	1	66.66%	**66.66%**
Toggles	204	193	11	1	94.60%	**94.60%**

- **Functional Coverage:**

Covergroup type:

drv_cg

Summary	Total Bins	Hits	Hit %
Coverpoints	28	28	**100.00%**
Crosses	101	99	**98.01%**

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	7	2	77.77%	**77.77%**	**77.77%**

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%

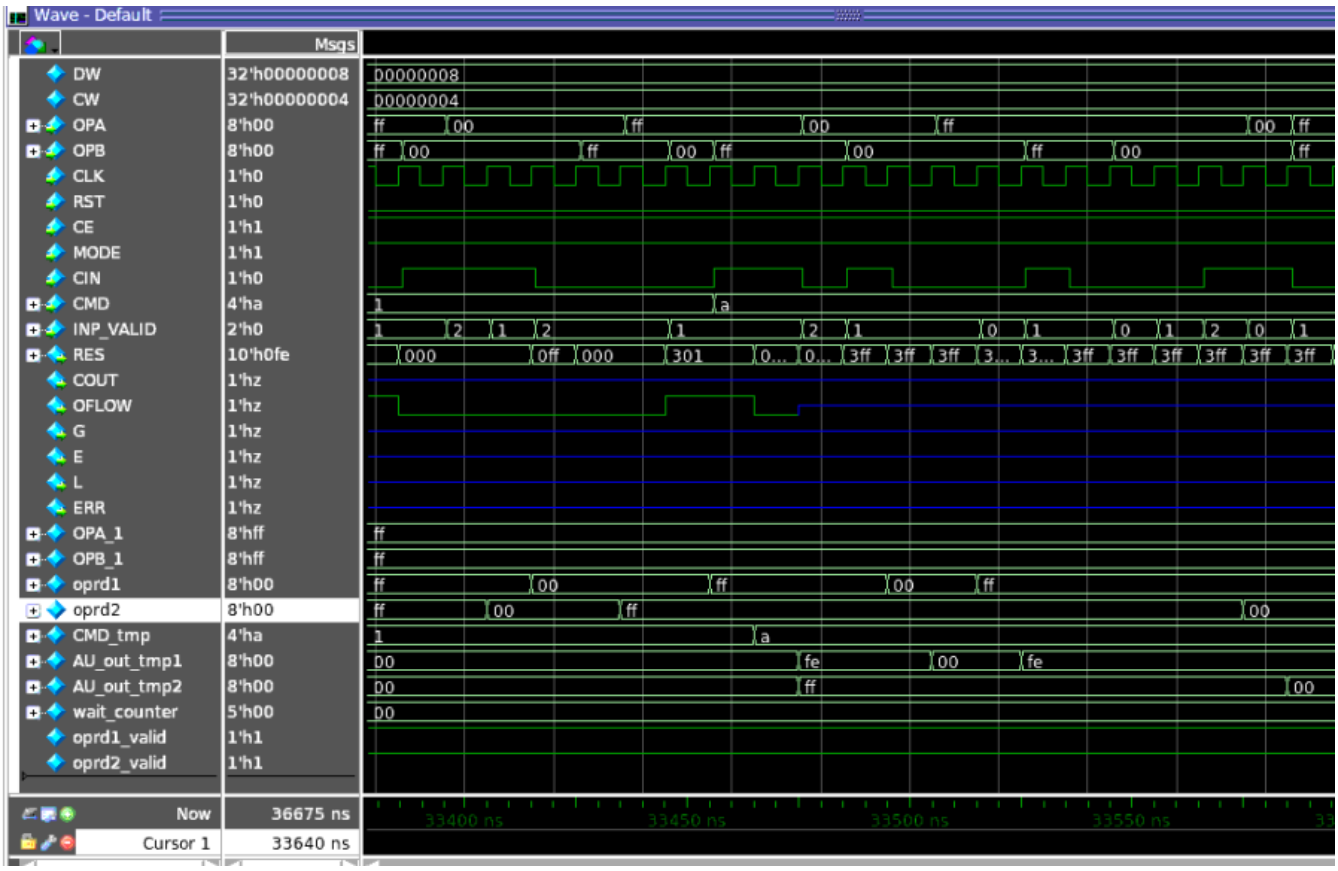
- **Assertion Coverage :**

Questa Assertion Coverage Report

Show All | Show Covered | Show Missing

Assertions	Failure Count	Pass Count	Attempt Count	Vacuous Count	Disable Count	Active Count	Peak Active Count	Status
assert__ppt_clock_enable	0	2	3849	3845	2	0	2	Covered
assert__ppt_timeout	170	1	3849	3676	2	0	12	Failed
assert__ppt_valid_inp	0	3843	3849	2	3	1	2	Covered
assert__ppt_reset	0	3	3849	3846	0	0	2	Covered

3.3 Output waveform:



3.4 Verification Plan link:

[Verification Plan]