Table of Contents

CHAPTER 1

PROJECT OVERVIEW AND SPECIFICATIONS

1.1 Project Overview

This project focuses on verifying a parameterized ALU capable of performing various arithmetic and logical operations. It supports single and two-operand instructions like ADD, SUB, SHIFT, ROTATE, COMPARE, and bitwise logic, along with error handling for invalid input formats.

A modular, SystemVerilog testbench is developed and built using mailbox-based communication. This structure enables constrained-random testing, accurate output comparison via a reference model, and robust functional coverage, ensuring thorough validation of the ALU's behavior.

This verification plan targets an Arithmetic Logic Unit (ALU) design capable of performing a variety of arithmetic and logical operations. It supports:

- Two operands (OPA, OPB)
- A command code (CMD)
- A mode signal (MODE) to switch between arithmetic (1) and logic (0)
- Input validity flags (INP_VALID)
- Clock enable (CE)
- Carry-in (CIN)
- Output result and status flags (COUT, OFLOW, G, L, E, ERR)

The ALU also has specific behavior such as:

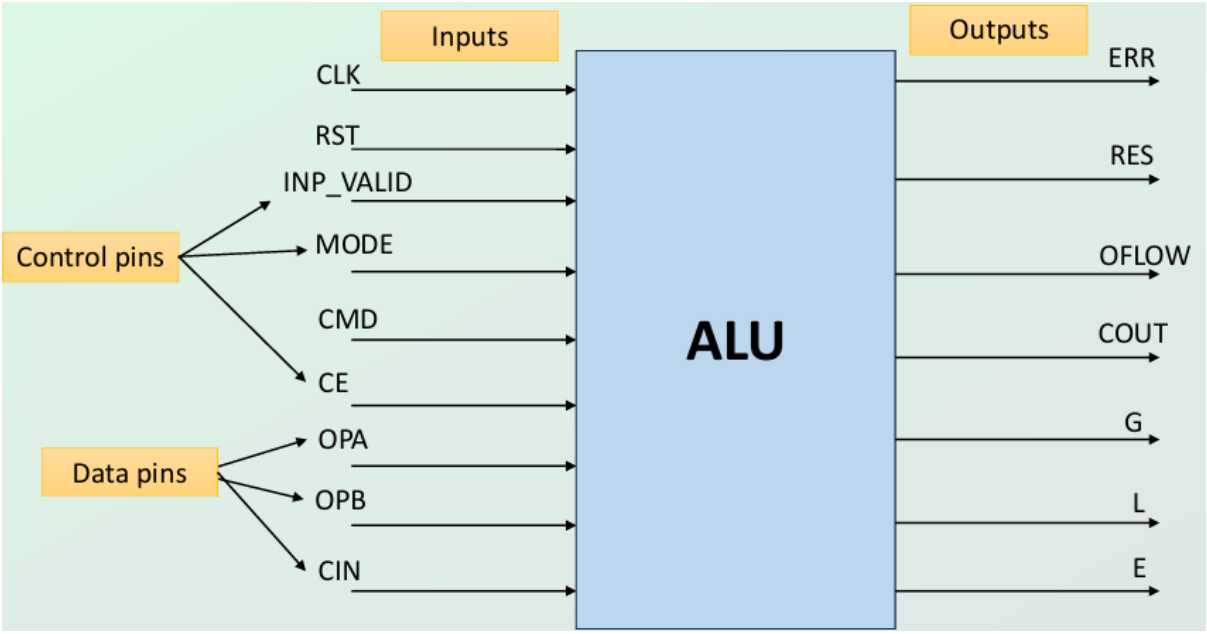
- Waiting 16 clock cycles for the second operand if only one is valid
- Error signaling for invalid rotate commands or timeout

1.2 Verification Objectives

- **Ensure Functional Correctness**:
  Verify that the ALU performs all supported arithmetic and logical operations correctly for valid input combinations under both single-operand and two-operand modes.

- **Validate Error Signaling**:
  Confirm that the ALU detects and signals appropriate error flags (ERR) when invalid or unsupported input formats or control signals are applied.
- **Input and Output Coverage**:
  Achieve comprehensive input space exploration using constrained-random stimulus to ensure that all operations, operand combinations, modes, and edge cases are exercised.
- **Reference Model Comparison**:
  Develop a cycle-accurate reference model that mirrors expected ALU behavior and use it for output comparison with DUT results through the scoreboard.
- **Layered Verification**:
  Build a structured testbench architecture that isolates stimulus generation, driving, monitoring, and checking, enabling modularity and ease of debugging.
- **Functional and Assertion Coverage**:
  Measure functional coverage (input combinations, operation types, flags) and integrate assertions to verify protocol and control correctness.

1.3 DUT Interfaces

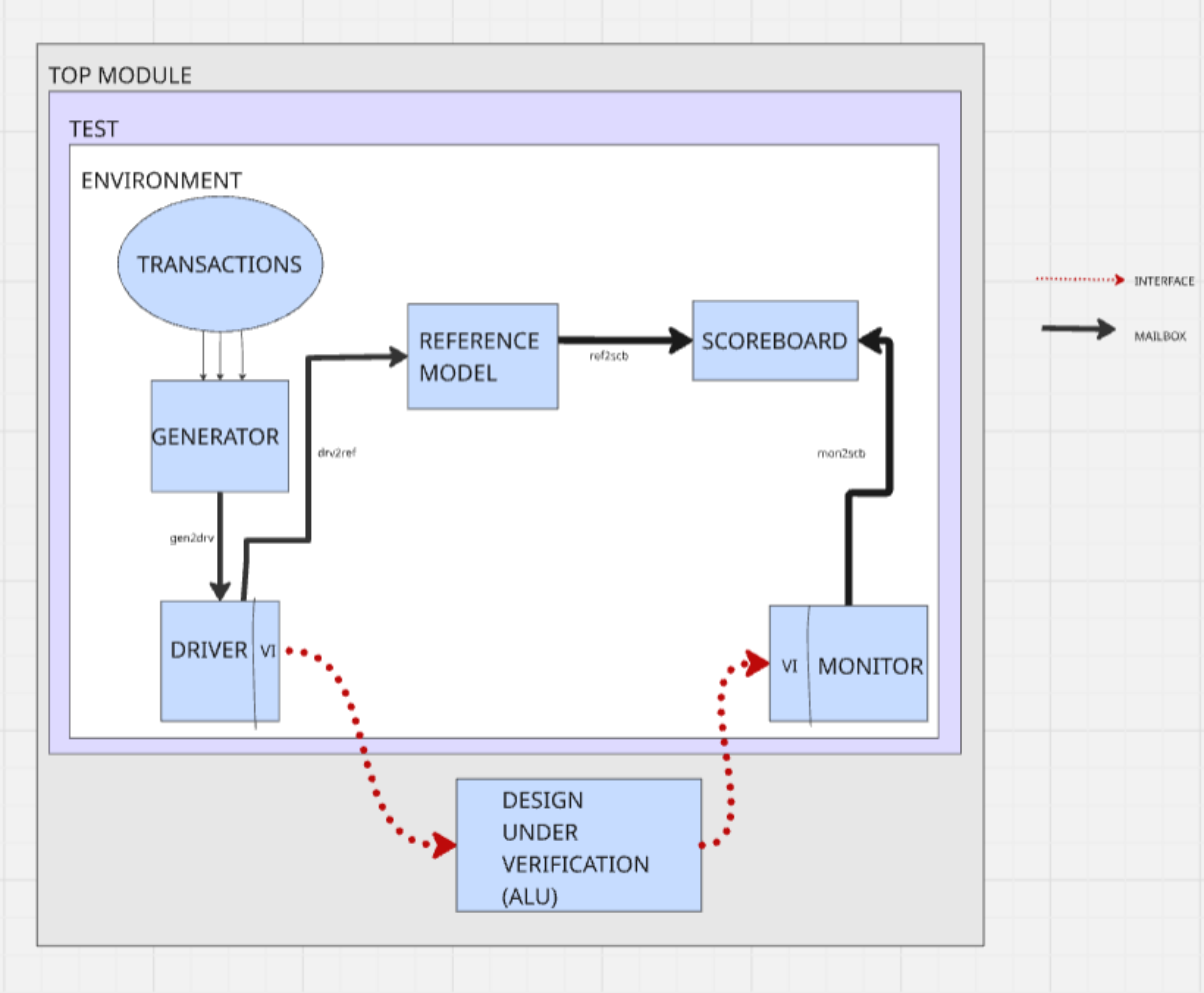


Signal	Direction	Bit-width	Description
OPA	Input	[WIDTH-1:0]	Operand A
OPB	Input	[WIDTH-1:0]	Operand B
CIN	Input	1 bit	Carry-in
CMD	Input	[CMD_WIDTH-1:0]	ALU operation command (0–13)
MODE	Input	1 bit	1 = Arithmetic mode, 0 = Logical mode
INP_VALID	Input	2 bits	Operand validity: bit[0] for OPA, bit[1] for OPB
CE	Input	1 bit	Clock Enable
CLK	Input	1 bit	Clock
RST	Input	1 bit	Asynchronous Reset
RES	Output	[RES_WIDTH-1:0]	Result of the ALU operation
COUT	Output	1 bit	Carry-out
OFLOW	Output	1 bit	Overflow flag
G	Output	1 bit	Greater-than flag from CMP
L	Output	1 bit	Less-than flag from CMP
E	Output	1 bit	Equal-to flag from CMP
ERR	Output	1 bit	Error flag (bad rotate or 16-cycle timeout)

CHAPTER 2

TESTBENCH ARCHITECTURE AND METHODOLOGY

2.1 Testbench Architecture



2.2 Component Details and Flowchart

2.2.1 Transaction Class

The **transaction class** serves as the fundamental building block for stimulus representation in the verification environment. It encapsulates all input and expected output signals involved in a single ALU operation. The class is parameterized with a WIDTH field (default 8 bits) and supports randomization to enable constrained-random test generation.

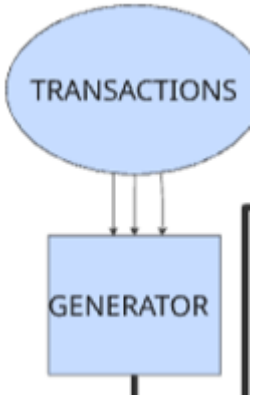
Key Fields:

- **Inputs**: ce, mode, cmd, inp_valid, opa, opb, cin, inject_err
- **Outputs (Predicted & Observed)**: res, cout, oflow, err, g, l, e

Features:

- **Random Constraints**: Used to ensure legal and illegal stimulus combinations are generated depending on the test type (functional or error-injection).
- **Copy Method**: Implements a deep copy function to clone transaction instances reliably when transferring between components.
- **Inheritance Support**: Specialized sub-classes (e.g., single_logical, two_arithmetic) apply specific constraints for targeted testing based on operation type.

This class acts as the **data carrier** across the testbench—generated by the Generator, driven by the Driver, observed by the Monitor, and compared in the Scoreboard. It provides a reusable and extensible mechanism to represent transactions uniformly across the verification components.



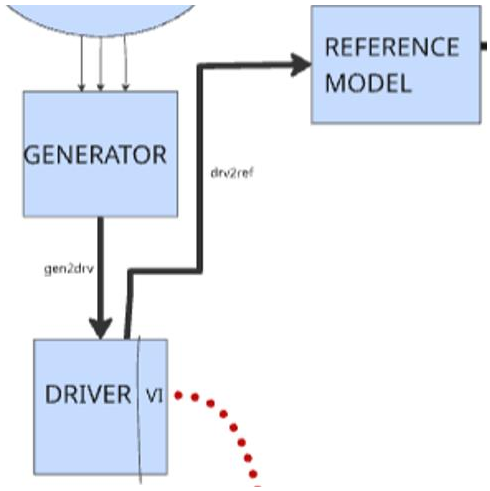
2.2.2 Generator

The **Generator** is responsible for creating randomized test stimulus in the form of transaction objects. It uses constraints within the transaction_alu class and its derived sub-classes to generate legal and illegal input patterns, depending on the test being executed.

Key Features:

- Implements a loop to generate a fixed number of transactions (no_of_trans).
- Randomizes the blueprint transaction and sends a copy of it to the **Driver** through a mailbox.
- For error testing, it leverages inject_err and constraint overrides to produce corner cases and invalid scenarios.
- Sends the same transaction to the **Reference Model** for expected output computation.

The Generator ensures systematic and randomized exploration of the ALU's functionality.



2.2.3Driver

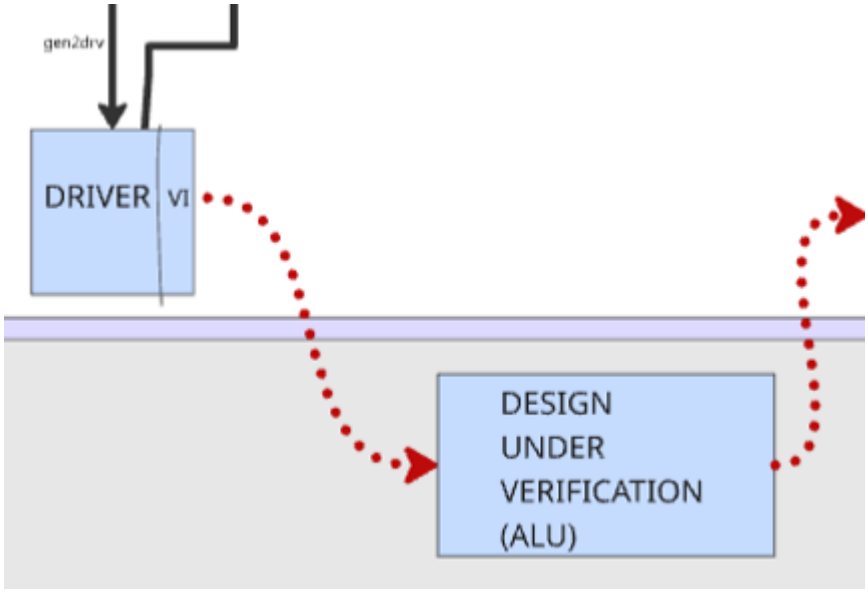
The **Driver** receives transactions from the Generator and converts them into pin-level signal activity to stimulate the DUT through the interface.

Key Responsibilities:

- Waits for valid transactions in the mailbox.

- Drives opa, opb, cin, ce, cmd, mode, and inp_valid to the DUT in alignment with clock and reset.
- Synchronizes with Monitor and Reference Model
- Maintains protocol integrity by honoring valid/ready-style control.

The Driver forms the physical interface between abstract test stimulus and the DUT's real ports.



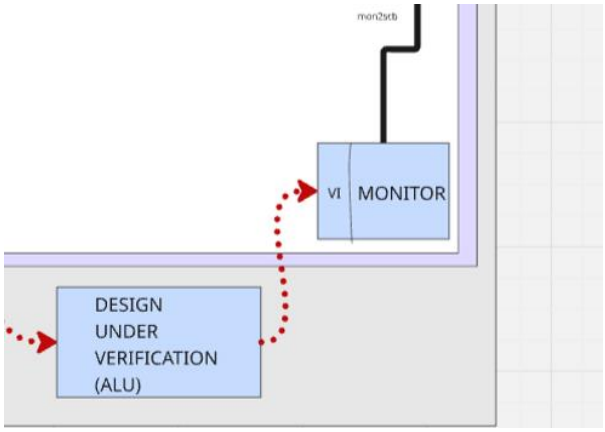
2.2.4 Monitor

The **Monitor** observes the DUT outputs non-intrusively and reconstructs the transaction for further analysis.

Key Responsibilities:

- Continuously samples DUT outputs at positive clock edges.
- Captures response fields like res, cout, err, oflow, g, l, e along with corresponding input context.
- Builds a mirrored transaction and forwards it to the **Scoreboard**.
- In parallel, updates the output functional coverage model using covergroups.

The Monitor provides visibility into DUT behavior and ensures observed outputs are accurately captured for validation.



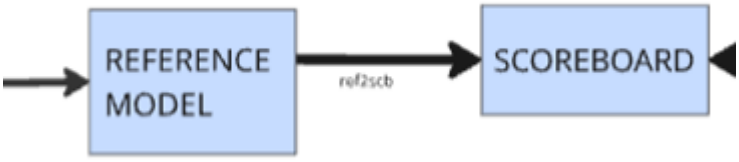
2.2.5 Reference Model

The **Reference Model** serves as the golden model for computing expected results for each transaction.

Key Responsibilities:

- Receives transactions from the Generator.
- Mimics ALU behavior in synthesizable logic using algorithmic computation.
- Computes correct values for result (res), comparison flags (g, l, e), carry (cout), overflow (oflow), and error (err) conditions.
- Sends the calculated outputs to the **Scoreboard** for comparison.

This component ensures correctness by providing an independently verified computation baseline for every test scenario.



2.2.6 Scoreboard

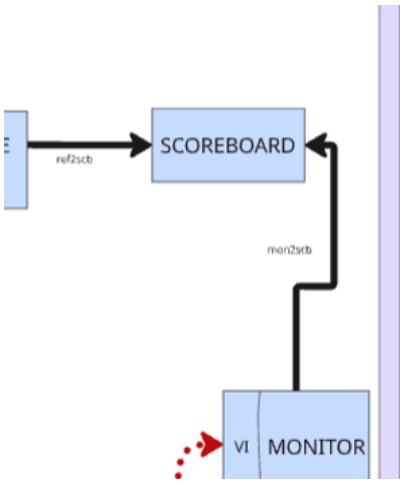
The **Scoreboard** is the decision-making unit that performs output comparison and logs the pass/fail status of each transaction.

Key Responsibilities:

- Receives DUT output from the Monitor and reference output from the Reference Model.
- Matches them field-by-field (res, err, cout, etc.) and flags mismatches.

- Maintains a transaction count and logs discrepancies with time and input context.
- Tracks the coverage and test completeness statistics across the simulation.

It plays a vital role in determining functional correctness and identifying design bugs.



CHAPTER 3

VERIFICATION RESULTS AND ANALYSIS

3.1 Error in the DUT

A. Arithmetic Mode Issues

- **INC (CMD-4)**: Increment operation on OPA does not yield the expected result.
- **DEC (CMD-5)**: Decrement on OPA functions only when INP_VALID = 3; fails for INP_VALID = 1.
- **INC (CMD-6)**: Increment operation on OPB is incorrect or non-functional.
- **DEC (CMD-7)**: Decrement on OPB does not perform as expected.
- **SHIFT by 1 & Multiply (CMD-10)**: Operation logic is incorrect or unimplemented.
- **ADD with CIN (CMD-2)**: COUT is not assigned correctly; MSB of RES is not propagated to carry.
- **SUB with CIN (CMD-3)**: When OPA = OPB and CIN = 1, result is -1 but overflow (OFLOW) flag is not set.

B. Logical Mode Issues

- **OR Operation (CMD-2)**: Performing logical AND instead of OR.
- **SHR1_A (CMD-8)**: Result is assigned directly from OPA without any shift.
- **SHR1_B (CMD-10)**: Left shift is performed instead of a right shift.
- **ROR (CMD-13)**: ERR flag is not asserted when invalid rotation bits OPB[4:7] are high.

C. General Functional Errors

- **Error Flag (ERR)**: Not raised when INP_VALID = 0.
- **Error Flag (ERR)**: Not triggered even when INP_VALID ≠ 3 for over 16 cycles.
- **Result Width**: Output result (RES) width is one bit wider than the specification allows.

3.2 Coverage Report

3.2.1 Overall Coverage