ALU Verification Plan

CHAPTER 1 – PROJECT OVERVIEW

This project details the comprehensive verification of a parameterized Arithmetic Logic Unit (**ALU**) using a modern, class-based System-Verilog testbench. The primary objective is to ensure the functional and timing correctness of the ALU design through a structured and reusable verification environment.

1.1 **Key design features**:

- Parameterized operand width (default 8-bit, scalable to 16, 32, 64, 128 bits).
- Dual operation modes: Arithmetic [MODE = 1] and Logical [MODE = 0].
- Comprehensive instruction set with different commands per mode.
- Advanced features: Rotate operations, comparison outputs, overflow detection etc.
- Timeout mechanism for operand validation (16 clock cycles).
- For [MODE = 1] Arithmetic operation, `Multiply` based command give output after 3 clock cycles and other commands take only 2 clock cycle for output.

1.2 **Verification Objectives**:

Functional correctness:

1. **Arithmetic Operations:** Verify that all arithmetic commands (ADD, SUB, ADD_CIN, SUB_CIN, INC_A, DEC_A, CMP, multiplication, etc.) produce the correct values for RES, COUT, OFLOW, and the comparator flags (G, L, E).
2. **Logical Operations:** Verify that all logical commands (AND, OR, XOR, NOT, NAND, NOR, etc.) produce the correct bitwise results including shift and rotate operations.
3. **Timing correctness:**
   - Latency: Verify that standard operations complete in the specified 2 clock cycles, and that multi-cycle instructions, like multiplication, complete 3 clock cycles.

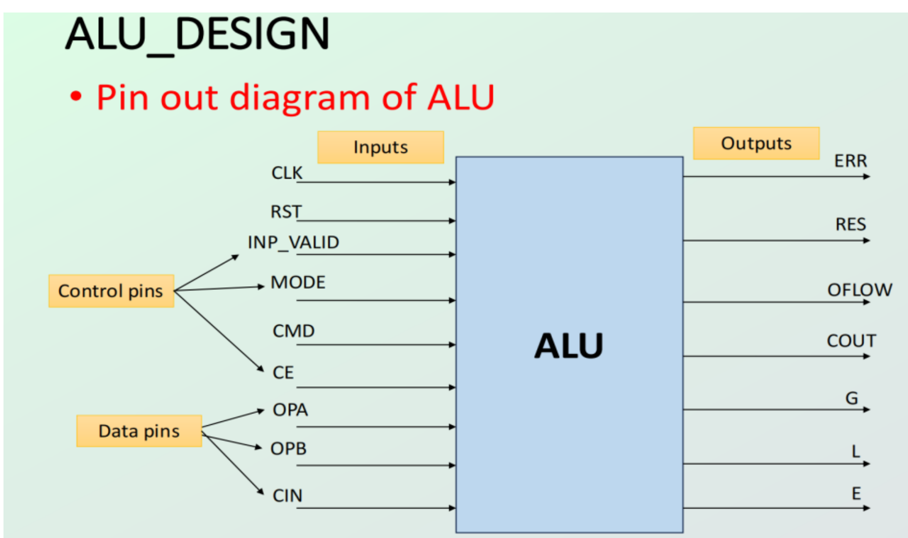
4. **Error handling**:

- Invalid Command Errors: Verify that the ERR signal is asserted correctly when an invalid operation is attempted, such as a rotate command where OPB[7:4] is non-zero.
- Operand Timeout: Verify that the design correctly triggers the 16-cycle timeout and asserts the ERR signal if a second required operand is not valid within 16 clock cycles.

1.3 Test, Coverage, and Assertion plan:

- **Test Plan Completion**: To successfully execute all tests outlined in the Test Plan with zero failures, errors, or scoreboard mismatches.
- **Functional Coverage:** Achieve 100% of the coverage goals defined in the Functional Coverage Plan. This includes hitting all defined coverpoints and crosses for every command, mode, and range of input operands.
- **Assertion Coverage**: Evaluate all assertions in the Assertion Plan, ensuring all properties have been active and tested without failure during simulation.

1.4 DUT Interface:



The **ALU** (Design Under Test) interface is composed of three primary categories of pins: data pins for the operands, control pins to manage the ALU's operation and timing, and output pins to present the result and status flags. A clear understanding of these interfaces is fundamental to creating an effective verification environment.

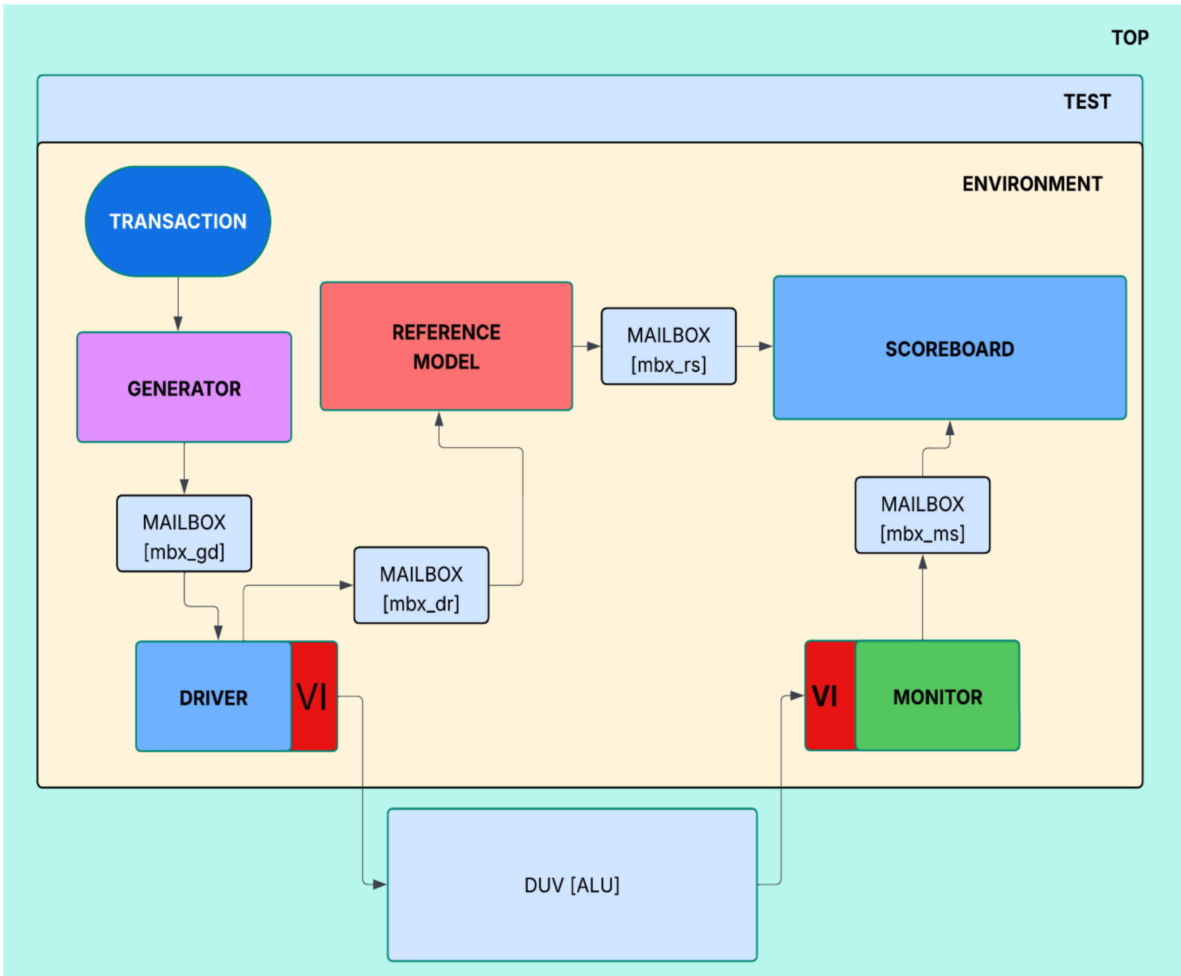
- **Data Pins:** These include the two primary parameterized operands, OPA and OPB, along with a 1-bit carry-in signal, CIN, used for arithmetic operations.
- **Control Pins:** These signals manage the ALU's state and behavior. They include the standard CLK and active-high asynchronous RST, a clock-enable [CE], an INP_VALID bus to signal when operands are ready, a MODE pin to switch between arithmetic and logical operations, and a CMD bus to select the specific instruction.
- **Output Pins:** The ALU's primary output is the RES bus, which is one bit wider than the operands to include a carry-out. Additional outputs provide critical status information, including
COUT (carry-out),
OFLOW (overflow),
comparator flags (G, L, E), and an ERR signal for illegal operations or timeouts.

Pin Name	Direction	Width (bits)	Function
OPA	INPUT	Parameterized	The first primary operand for the operation.
OPB	INPUT	Parameterized	The second primary operand for the operation.
CIN	INPUT	1	Carry-in signal used in certain arithmetic operations.
CLK	INPUT	1	The main clock signal for the synchronous design.
RST	INPUT	1	Active-high asynchronous reset.
CE	INPUT	1	Active-high clock enable signal. Operations are paused when low.
MODE	INPUT	1	Selects operation mode: 1-Arithmetic, 0-Logical.
INP_VALID	INPUT	2	Indicates which operands are valid on the data bus.

Pin Name	Direction	Width (bits)	Function
CMD	INPUT	Parameterized (by default: 4)	Command code selecting the instruction to be executed.
RES	OUTPUT	Parameterized + 1	Result of the performed ALU instruction.
OFLOW	OUTPUT	1	Indicates overflow during a subtraction operation.
COUT	OUTPUT	1	Carry-out from addition operations.
G	OUTPUT	1	Comparator output; asserted when OPA > OPB.
L	OUTUT	1	Comparator output; asserted when OPA < OPB.
E	OUTPUT	1	Comparator output; asserted when OPA = OPB.
ERR	OUTPUT	1	Asserted on illegal command or timeout occurrence.

CHAPTER 2 - Verification Architecture

2.1 Verification Architecture :

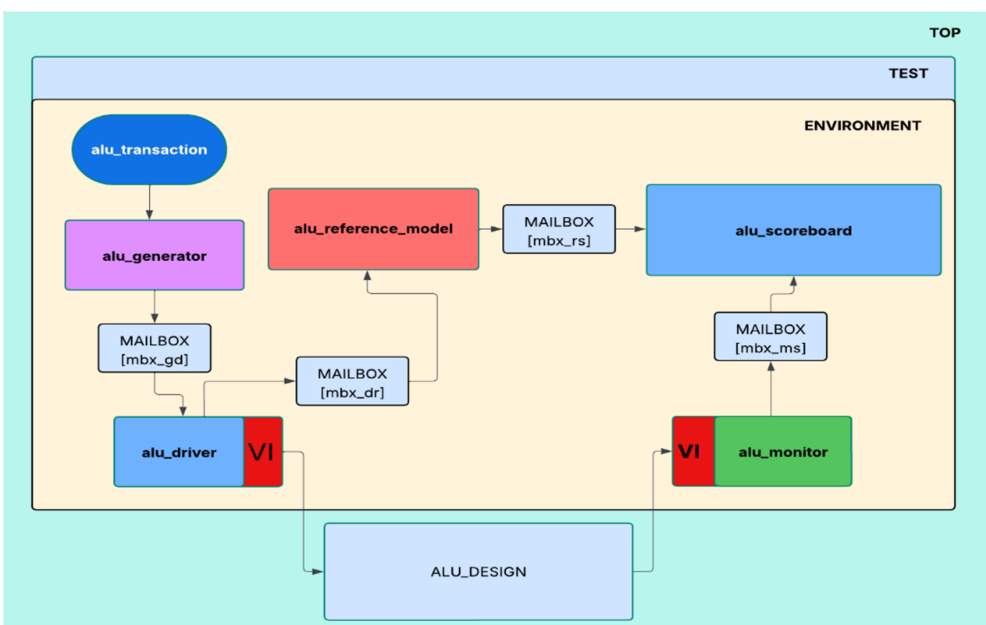


Generalized Testbench Architecture

Component	Role / Description	Functionality
Test	Top-level controller	Initializes environment, triggers stimulus generation and test scenario setup.
Environment	Wrapper containing all verification components	Manages **generator, driver, monitor, reference model, scoreboard, interfaces**.

Component	Role / Description	Functionality
Generator	Creates stimulus (input transactions)	Generates random or directed tests to verify different functional behaviours.
Mailbox	Communication mechanism between components	Transfers data between components safely (e.g., gen → drv, mon → scb).
Driver	Translates transactions into signal-level DUT inputs	Applies pin-wiggling protocols as dictated by interface timing and spec.
DUT (Device Under verification)	Core design block or RTL under verification	Receives inputs from the driver and produces outputs to be checked.
Monitor	Passive observer of DUT signals	Samples DUT outputs, extracts meaningful transaction data for checking.
Reference Model	Predicts expected behaviour/output for given inputs	Acts as the "golden model" generating correct responses based on stimulus.
Scoreboard	Compares DUT output and reference model output	Detects mismatches, reports pass/fail decisions, logs errors.

<u>2.2 Verification Architecture for ALU:</u>



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

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).**

• **alu_design**:
The actual ALU design [**DUV (Design Under Verification)**] 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.

• **alu_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 :

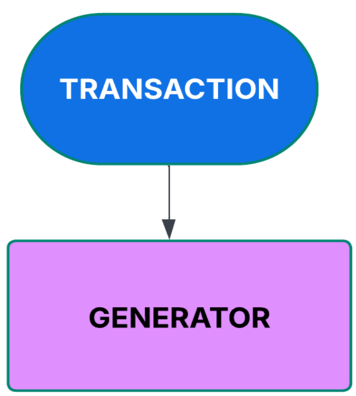
1. **Transaction class:**

Input Phase:
- Declare randomizable fields (data, address, control signals)
- Define constraints for valid input ranges
- Implement randomization methods

Processing Phase:
- Apply randomize() function to generate valid stimulus
- Validate constraint satisfaction
- Pack data into transaction object format

Output Phase:
- Provide accessor methods for field values
- Implement display/print functions for debugging
- Support DEEP/SHALLOW copy operations

2. **Generator class:**



Input Phase:
- Receive test configuration parameters
- Get scenario-specific directives from test class
- Access constraint settings and randomization seeds

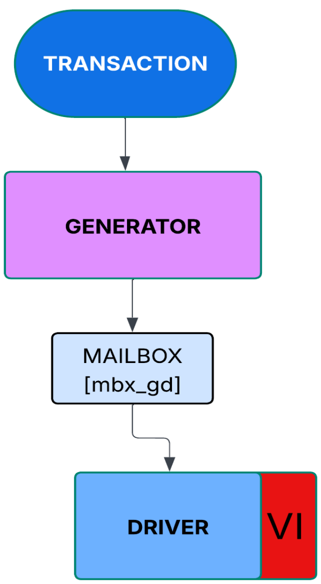
Processing Phase:

- Create transaction objects in loops or based on scenarios
- Apply randomize() to generate stimulus variations
- Implement timing control between transaction generations
- Handle corner cases and directed test patterns

Output Phase:

- Send completed transactions to driver via mailbox
- Log generation statistics and coverage information
- Signal completion to environment controller

3. Driver class:



Input Phase:

- Receive transactions from generator via mailbox
- Get interface configuration and timing parameters
- Access virtual interface handle to DUT signals

Processing Phase:

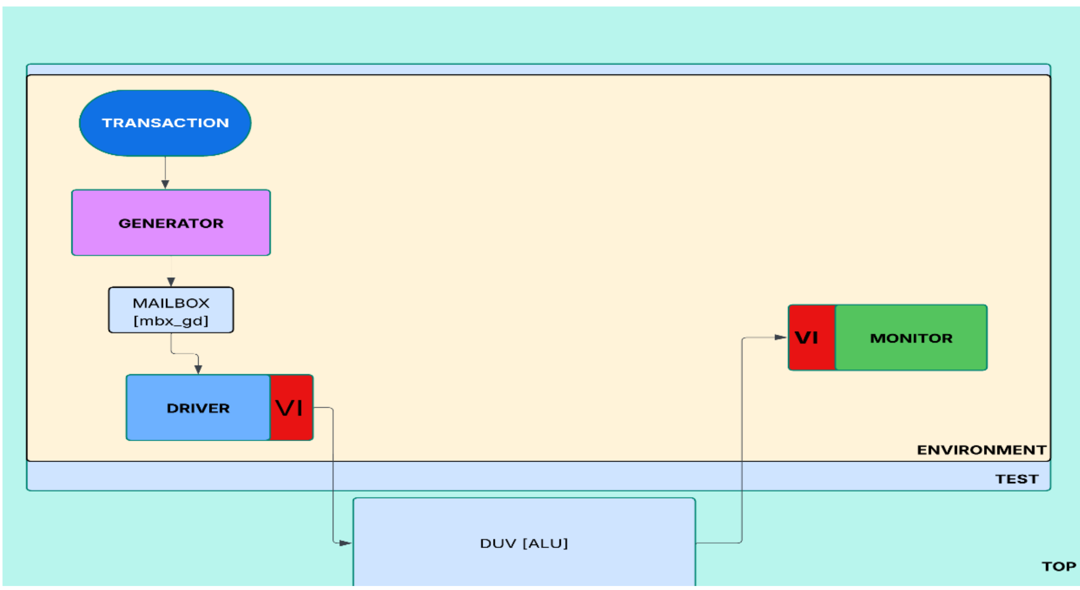
- Convert transaction fields to signal-level values
- Implement protocol-specific handshaking
- Apply proper timing relationships (setup/hold times)
- Handle clock synchronization and edge alignment

- Manage reset and error recovery scenarios

Output Phase:

- Drive signals onto DUT/DUV input pins
- Send transaction copy to reference model
- Report successful stimulus application to environment
- Update driver status and statistics

4. monitor class:



Input Phase:

- Continuously sample DUT output signals via virtual interface
- Receive clock and control signal information
- Get protocol configuration for signal interpretation

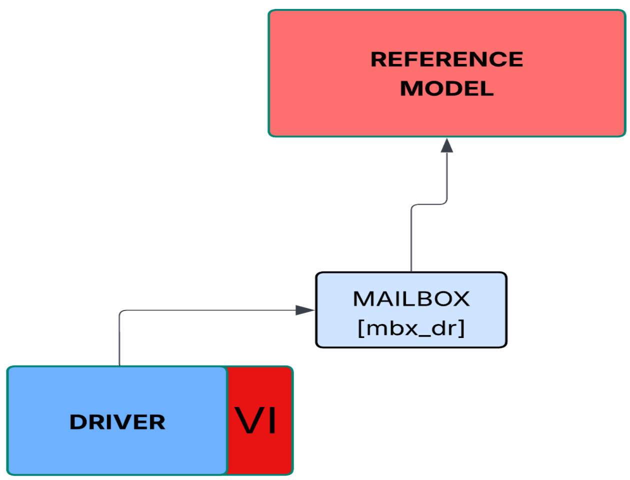
Processing Phase:

- Detect valid transaction boundaries using protocol signals
- Extract and reconstruct transaction data from sampled signals
- Validate protocol compliance and signal integrity
- Apply filtering and data preprocessing
- Handle metastability and timing variations

Output Phase:

- Forward reconstructed transactions to scoreboard
- Send coverage information to coverage collectors
- Log monitored transactions for debug purposes
- Generate protocol violation warnings when detected

5. Reference model class:



Input Phase:

- Receive same input transactions sent to DUT (from driver)
- Get configuration parameters matching DUT setup
- Access behavioral model parameters and lookup tables

Processing Phase:

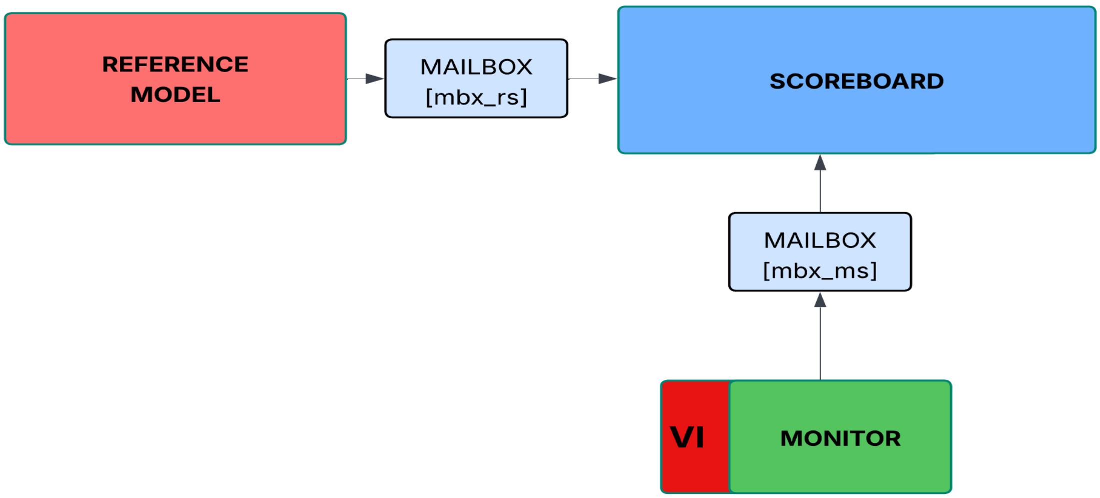
- Execute ideal/golden behavior algorithms
- Perform functional calculations without timing delays
- Handle all input combinations and corner cases
- Apply behavioral modeling (no RTL timing constraints)
- Generate expected output responses

Output Phase:

- Send predicted results to scoreboard for comparison
- Provide expected transaction objects with timing information
- Log reference model calculations for debug analysis

- Report any internal model errors or warnings

6. Scoreboard class:



Input Phase:

- Receive actual output transactions from monitor.
- Receive expected output transactions from reference model.

Processing Phase:

- Perform field-by-field comparison of actual vs expected.
- Handle timing differences and reordering scenarios.
- Track pass/fail statistics and error categories.

Output Phase:

- Generate pass/fail reports for each comparison.
- Log detailed mismatch information for debugging.
- Update overall test statistics (error count, pass rate)
- Trigger test termination on critical failures.
- Provide summary reports at test completion.