**ALU VERIFICATION PLAN**

**1. Project Overview**

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

**2. Verification Objectives**

* Verify correct functionality of all ALU operations across both arithmetic and logical modes by comparing DUT output with expected values.
* Ensure output flags (G, L, E, COUT, OFLOW, ERR) are set correctly under relevant conditions and commands.
* Validate reset functionality by confirming all outputs are cleared when RST is asserted, regardless of current or past inputs.
* Check for correct 16-clock wait behavior when only operand A is valid (INP\_VALID = 2’b01), triggering ERR if operand B is not received within 16 cycles.
* Achieve 100% functional and cross coverage by sampling all meaningful combinations of inputs such as CMD, MODE, CIN, INP\_VALID, and control signals.
* Assert timing and protocol correctness using assertions, ensuring proper sequencing of signals, stable control signals, and correct response to invalid or boundary inputs.

**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) |

**4. Testbench**

**1. Top Module (top\_alu)**

Role:

* The simulation entry point that integrates the DUT (ALU) and verification environment.

Inputs:

* None directly (starts simulation)

Outputs:

* Instantiates the ALU design under verification (DUT)
* Instantiates the SystemVerilog interface
* Passes the virtual interface handle to the test class
* Launches the test using test\_alu

**2. Test Class (test\_alu)**

Role:

* Controls the execution of the test by creating the environment and starting its processes.

Inputs:

* Virtual interface from top\_alu

Outputs:

* Instantiates environment\_alu
* Calls the start() method to begin test execution

**3. Environment (environment\_alu)**

Role:

* Central hub that manages all components and connections.

Inputs:

* Virtual interface
* No direct mailbox inputs

Outputs:

* Instantiates:
  + generator\_alu
  + driver\_alu
  + monitor\_alu
  + reference\_model\_alu
  + scoreboard\_alu
* Sets up the following mailboxes:
  + gen2drv: Generator → Driver
  + drv2ref: Driver → Reference Model
  + ref2sb: Reference Model → Scoreboard
  + mon2sb: Monitor → Scoreboard

**4. Generator (generator\_alu)**

Role:

* Creates randomized or constrained input transactions.

Inputs:

* transactions\_alu class for field definitions and constraints

Outputs:

* Sends randomized transactions to Driver via gen2drv mailbox

**5. Driver (driver\_alu)**

Role:

* Drives the DUT signals and sends inputs to the reference model.

Inputs:

* Receives transactions from Generator via gen2drv mailbox

Outputs:

* Applies inputs to DUT through the virtual interface
  + Fields like: opa, opb, cmd, cin, mode, inp\_valid, ce
* Forwards same transaction to Reference Model via drv2ref mailbox
* Samples functional coverage for input conditions

**6. Reference Model (reference\_model\_alu)**

Role:

* Computes expected outputs based on the received inputs and ALU logic.

Inputs:

* Receives transactions from Driver via drv2ref mailbox

Outputs:

* Sends expected outputs (res, cout, oflow, G/L/E, err) to Scoreboard via ref2sb mailbox

**7. Monitor (monitor\_alu)**

Role:

* Passively observes DUT’s actual output behavior.

Inputs:

* Observes DUT signals via virtual interface (e.g., res, cout, G, L, E, err, etc.)

Outputs:

* Constructs an actual output transaction
* Sends it to Scoreboard via mon2sb mailbox

**8. Scoreboard (scoreboard\_alu)**

Role:

* Compares the DUT’s actual outputs with the expected outputs from the reference model.

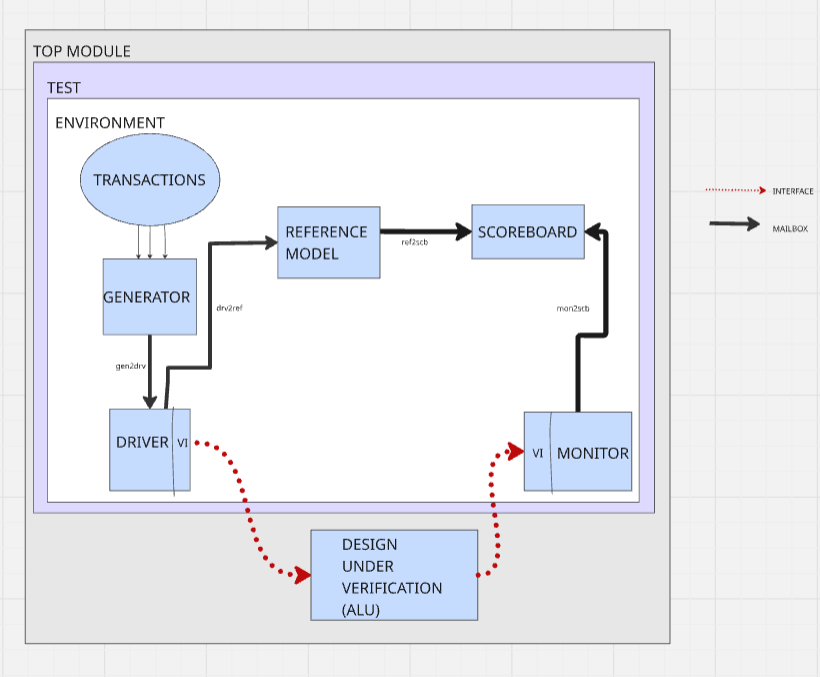
Inputs:

* Actual outputs from Monitor via mon2sb mailbox
* Expected outputs from Reference Model via ref2sb mailbox

Outputs:

* Reports mismatches, if any
* Confirms pass/fail status of each transaction

**Diagram**



**5. Test Plan**

**Test Scenarios**

* Apply all 14 ALU commands with a range of operands in both arithmetic and logical modes.
* Verify all input combinations of INP\_VALID including valid/invalid operand cases.
* Test with CIN set to 0 and 1 during ADD\_CIN and SUB\_CIN.
* Check behavior when CE is deasserted (no operation should take place).
* Apply edge cases such as maximum and minimum operand values (e.g., 0, mid-range, max).
* Confirm correct G, L, E flags when CMD\_CMP, CMD\_SIGN\_ADD, and CMD\_SIGN\_SUB are used.
* Trigger error condition by providing OPA only and delaying OPB beyond 16 clock cycles.
* Trigger error by giving invalid rotate values (non-zero bits in OPB[WIDTH-1:SHIFT+1]).
* Assert reset at various stages to check proper reset response.
* Confirm behavior when illegal combinations (like invalid CMD) are input.

**Functional Coverage Plan**

* CMD\_CP: Cover all 14 ALU commands.
* MODE\_CP: Cover both logical and arithmetic modes.
* INP\_VALID\_CP: Cover all operand validity combinations (00, 01, 10, 11).
* CE\_CP: Cover CE being 0 and 1 during operation.
* CIN\_CP: Test CIN = 0 and 1 for ADD\_CIN/SUB\_CIN commands.
* OPA\_CP and OPB\_CP: Cover operand edge values (e.g., 0, mid, max).
* RST\_CP: Test reset condition clearing the outputs.
* ERR\_CP: Cover both error and no-error scenarios (rotate and timeout).
* CMP\_OUT\_CP: Ensure CMP command sets correct G, L, E flags.

**Cross Coverage:**

* CMD × MODE
* CMD × INP\_VALID
* CMD × ERR
* CMD × CMP Output Flags
* CMD × Wait-16 behavior

**Assertions**

* CMD signal must remain stable when CE is low.
* ERR must never be asserted during valid input and legal command conditions.
* When RST is high, all outputs must reset to default.
* If INP\_VALID is 2'b01, a timer must count 16 clock cycles waiting for OPB; if not received, ERR must go high.
* During rotate operations (CMD = 12 or 13 in logical mode), if OPB[WIDTH-1:SHIFT+1] != 0, then ERR must assert.
* RES and flags must only update on the rising edge of CLK when CE is high and inputs are valid.