ALU - VERIFICATION

BHUVANESH

27bhuvanesh5@gmail.com

PROJECT OVERVIEW

The project focuses on the verification of a parameterized Arithmetic Logic Unit (ALU) which supports a wide range of arithmetic and logical operations. ALU are an integral part of any SOC that performs Arithmetic and logical operations. The ALU supports variety of functions including arithmetic operations such as addition, subtraction, increment, decrement, and multiplication, as well as logical operations such as AND, OR, XOR, NOT, NAND, NOR, and XNOR. In addition, it supports shift and rotate operations. The design also has comparator functions and error checking for invalid command conditions.

VERIFICATION OBJECTIVE

The objective of the project is to:

- Verify functional correctness of all ALU operations — including arithmetic, logical, comparison, and shift/rotate — as determined by CMD and MODE.
- Ensure input protocol compliance by checking that INP_VALID reflects operand availability (2'b01, 2'b10, or 2'b11) and operations proceed only when valid.
- Verify that when only one operand is valid (INP_VALID = 2'b01 or 2'b10), the ALU waits up to 16 clock cycles for the second operand, and asserts ERR if it doesn't arrive.
- Confirm timing behaviour, ensuring results are available after 1 or 2 clock cycles, depending on the operation.
- Apply constrained-random testing and functional coverage to explore all valid/invalid input combinations, including edge cases like overflow, underflow, and invalid rotate commands.

DUT INTERFACES

Block diagram of parameterized ALU is shown in Figure 1.

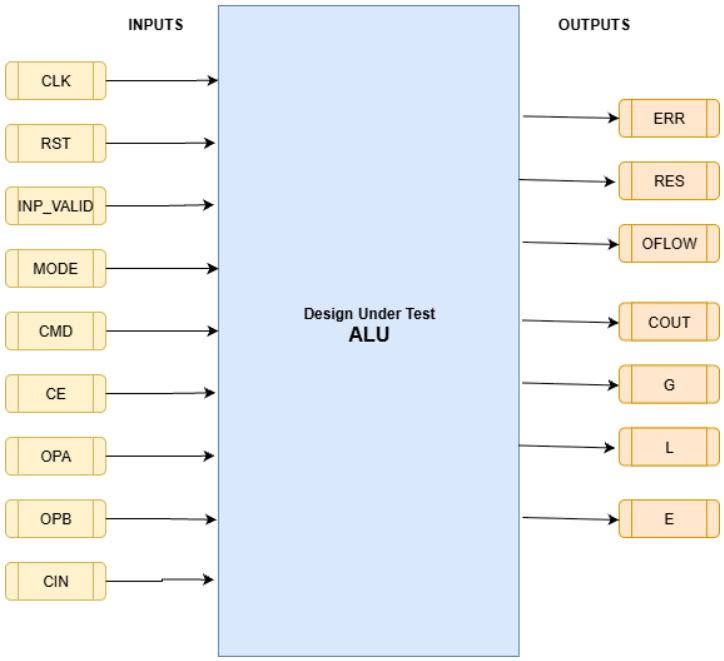

Figure 1. ALU under test

Below is the table which describes about the pins in the ALU which is design under test.

	PIN NAME	PIN DESCRIPTION
INPUTS	CLK (Clock)	Synchronous clock
	RST (Reset)	Asynchronous reset
	CE (Clock Enable)	Enables the ALU to perform operations on the clock edge
	CDM	A 4-bit command input that specifies the operation to be performed
	MODE	Determines the type of operation i.e. if MODE = 1 then Arithmetic operation else Logical operation
	CIN (Carry In)	Used for addition/subtraction with carry/borrow
	OPA / OPB [Width-1:0]	Parameterized inputs for operand A and operand B
	INP_VALID	It is a 2-bit input which indicates validity of input operands i.e. 00: No operand is valid 01: Operand A is valid 10: Operand B is valid 11: Both operands are valid
OUTPUTS	ERR	Error signal for invalid operations
	RES [Width-1:0]	Result from the logical or arithmetic operation
	G, L, E	Comparator outputs indicating relationship between operands i.e. greater-than, less-than and equal-to
	COUT	Carry out for arithmetic operation
	OFLOW	Overflow flag for arithmetic operation

Below is the table for the ALU showing specific arithmetic operation cases for given inputs, where the resulting outputs must be carefully considered as follows:

Command number	Command Operation	Description	ALU Behaviour / Operation	Flags Affected
0	ADD	Unsigned Addition	RES = OPA + OPB	COUT, OFLOW
1	SUB	Unsigned Subtraction	RES = OPA - OPB	COUT, OFLOW
2	ADD_CIN	Addition with Carry-In	RES = OPA + OPB + CIN	COUT, OFLOW
3	SUB_CIN	Subtraction with Borrow (Carry-In)	RES = OPA - OPB - CIN	COUT, OFLOW
4	INC_A	Increment A	RES = OPA + 1	OFLOW
5	DEC_A	Decrement A	RES = OPA - 1	OFLOW
6	INC_B	Increment B	RES = OPB + 1	OFLOW
7	DEC_B	Decrement B	RES = OPB - 1	OFLOW
8	CMP	Compare A and B	Sets G, L, E based on comparison	G, L, E
9	INC_A_B_MUL	Increment A and B, then multiply	RES = (OPA + 1) * (OPB + 1)	-
10	SHL_A_MUL_B	Left shift A by 1, then multiply with B	RES = (OPA << 1) * OPB	-
11	ADD_SIGNED	Signed Add, sets all relevant flags	RES = OPA + OPB with signed consideration	COUT, OFLOW, G, L, E
12	SUB_SIGNED	Signed Subtract, sets all relevant flags	RES = OPA - OPB with signed consideration	COUT, OFLOW, G, L, E

The designed ALU supports various logical operations when the signal MODE is deasserted which includes AND, OR, NAND, NOR, XOR, XNOR, NOT_A, NOT_B, SHR1_A, SHL1_A, SHR1_B, SHL1_B, ROL_A_B (Rotate Left A by bits specified in B), ROR_A_B (Rotate Right A by bits specified in B)

Below is the table for the ALU showing specific logical operation cases for given inputs, where the resulting outputs must be carefully considered as follows:

Command number	Command Operation	Description	ALU Behaviour / Operation	Flags Affected
0	AND	Bitwise AND between A and B	RES = OPA & OPB	-
1	NAND	Bitwise NAND between A and B	RES = ~ (OPA & OPB)	-
2	OR	Bitwise OR between A and B	RES = OPA | OPB	-
3	NOR	Bitwise NOR between A and B	RES = ~ (OPA | OPB)	-
4	XOR	Bitwise XOR between A and B	RES = OPA ^ OPB	-
5	XNOR	Bitwise XNOR between A and B	RES = ~ (OPA ^ OPB)	-
6	NOT_A	Bitwise NOT of A	RES = ~OPA	-
7	NOT_B	Bitwise NOT of B	RES = ~OPB	-
8	SHR1_A	Shift Right A by 1	RES = OPA >> 1	-
9	SHL1_A	Shift Left A by 1	RES = OPA << 1	-
10	SHR1_B	Shift Right B by 1	RES = OPB >> 1	-
11	SHL1_B	Shift Left B by 1	RES = OPB << 1	-
12	ROL_A_B	Rotate Left A by bits specified in B	Rotate A left by the lowest bits of B needed to cover the operand width; set ERR if any higher bits of B, are nonzero.	ERR

13	ROR_A_B	Rotate Right A by bits specified in B	Rotate A right by the lowest bits of B needed to cover the operand width; set ERR if any higher bits of B, are nonzero.	ERR

Testbench Architecture

General Testbench Architecture

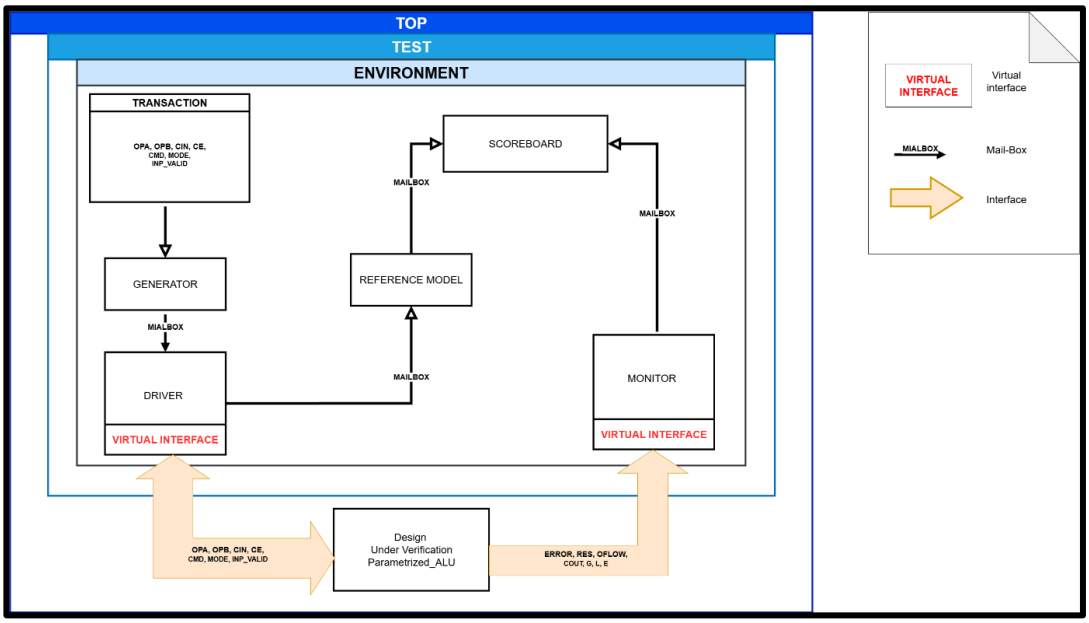

Figure 1. General Testbench Architecture

ALU Testbench Architecture

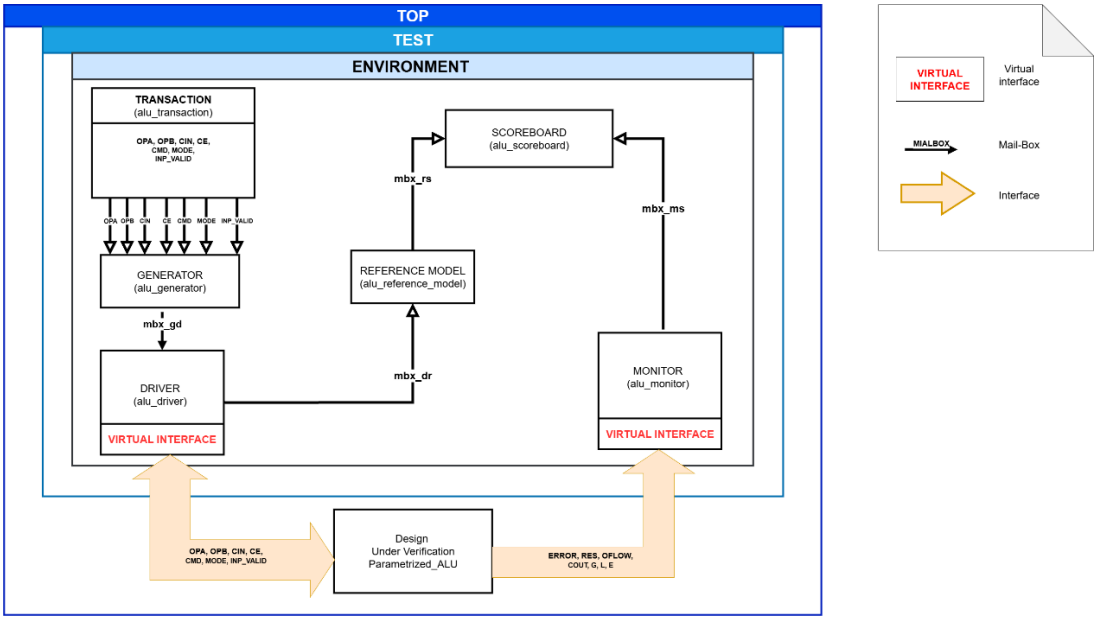

Figure 2. ALU Testbench Architecture

The ALU testbench architecture consist as follows:

Top:

- The top module is the entry point of the simulation.
- It generates clock and reset signal, instantiates the interface, ALU design (DUT), and the test class.
- The test is started from here using an initial block, making this the launch point of the simulation.

Interface:

- The interface connects between the testbench and the ALU design (DUT).
- It groups all the input and output signals of the ALU into a single unit so that they can be easily accessed and controlled by the driver and monitor.
- It includes inputs like OPA, OPB, CIN, CMD, MODE, INP_VALID, CE, and outputs like RES, COUT, OFLOW, ERR, G, L, E.
- Clock and reset signals are also included in the interface for synchronization.
- The interface is instantiated in the top module and is passed into the testbench components using a virtual interface handle.
- This virtual interface is used by the driver to drive signals to the DUT and by the monitor to observe outputs from the DUT.
- The interface also provides tasks and functions (optional) to encapsulate common behaviour, such as applying inputs or capturing outputs.

Transaction:

- All the inputs and outputs used by the ALU (like OPA, OPB, CIN, CE, CMD, MODE, and INP_VALID) are included in the transaction class.
- Clock and reset signals are excluded since they are generated in the Top module.
- The transaction class can also contain constraints on the input values to generate meaningful scenarios (e.g., valid operand ranges, operation types).

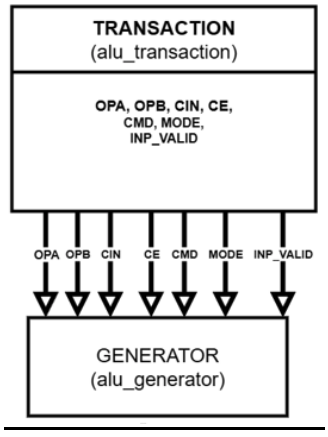

Figure 3. ALU_Transaction block

Generator:

- This component creates random input combinations for the ALU, respecting the constraints defined in the transaction class.
- It generates transactions and sends them to the driver using a mailbox.
- The generator helps cover different kinds of operations and edge cases.

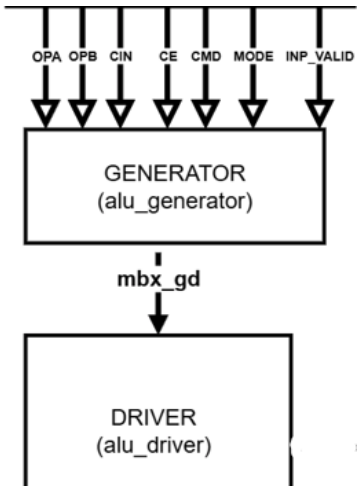

Figure 2. ALU_Generator Block

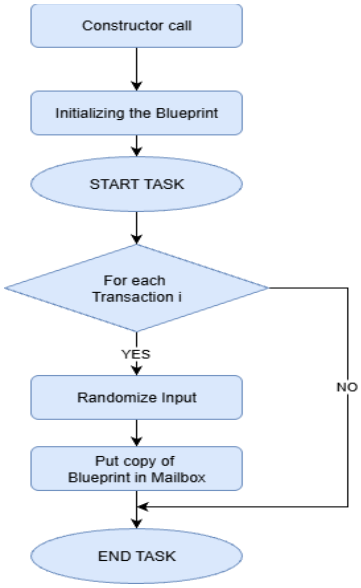

Figure 3. Flowchart of ALU_Generator

Driver:

- The driver receives transactions from the generator and applies them to the ALU inputs using a virtual interface.
- It translates the high-level transaction data into signal activity on the ALU pins.
- It also forwards the same transactions to the reference model, which is used for checking expected behaviour.

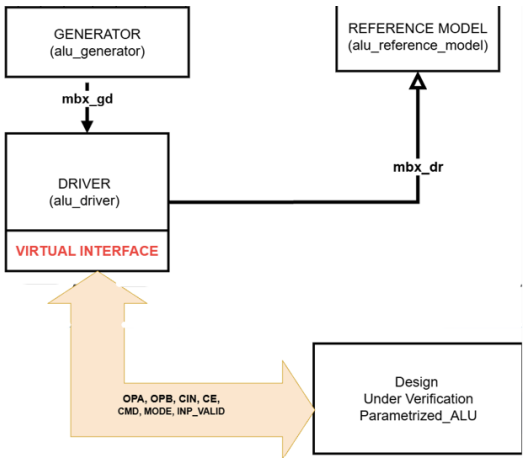

Figure 3. ALU_Driver Block

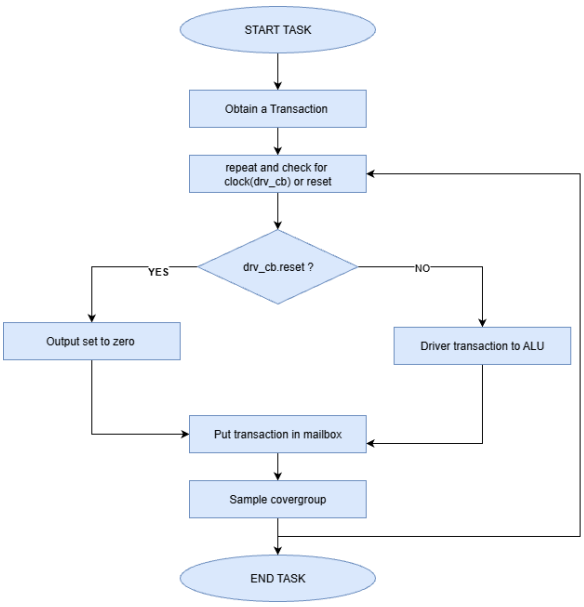

Figure 4. Flowchart of ALU_Driver

Monitor:

- The monitor observes the outputs of the ALU (like RES, COUT, OFLOW, ERR, G, L, and E) using a virtual interface.
- It converts these low-level signals into a high-level transaction and sends them to the scoreboard through a mailbox.

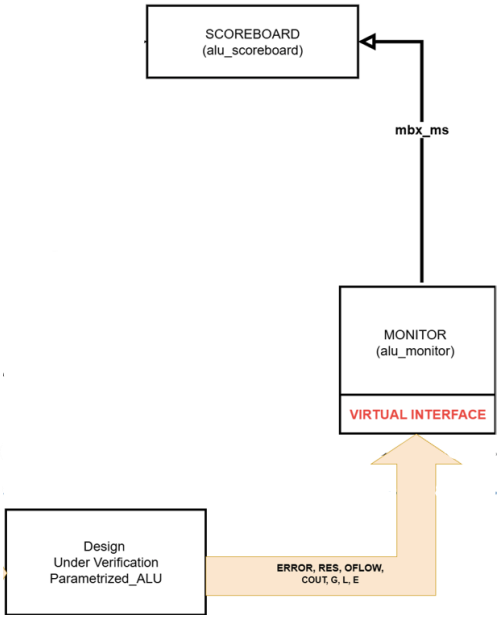

Figure 4. ALU_Monitor Block

Reference Model:

- This is the ideal or "golden" model of the ALU, implemented at a high level.
- It performs the same operation as the real ALU using the input transaction and produces the expected output.
- The input to the reference model comes from the driver, and its output is passed to the scoreboard for comparison.

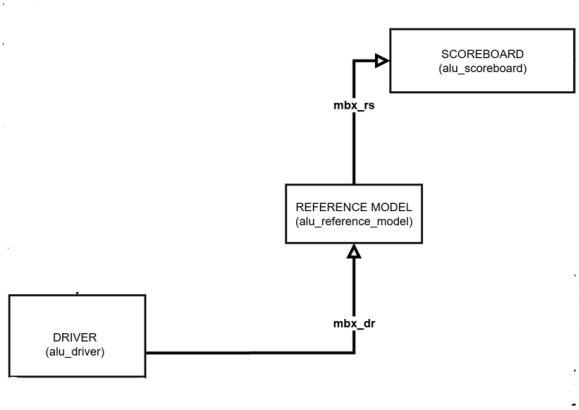

Figure 5. ALU_Reference_Model Block

Scoreboard:

- The scoreboard compares the expected output from the reference model with the actual output from the monitor.
- If the values match, the test passes; otherwise, it logs a mismatch and raises an error.

- The scoreboard is central to identifying bugs or mismatches in ALU functionality.

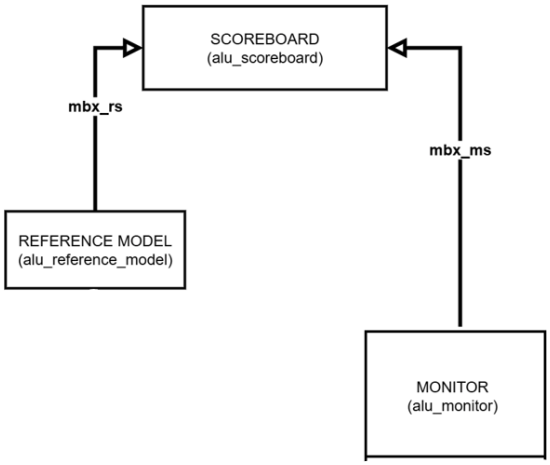

Figure 6. ALU_ScoreBoard

Environment:

- The environment contains all the above components: generator, driver, monitor, reference model, and scoreboard.
- It handles their construction, configuration, and connection using mailboxes and virtual interfaces.
- The environment acts as a container for building the full testbench.

Test:

- This is where different test cases are defined (e.g., testing addition, subtraction, overflow, rotate, etc.).
- The test instantiates the environment and runs specific scenarios.
- It is written inside a class and helps run both directed and randomized tests.

Test Plan

Click on the link to access the test plan of the ALU : ALU_TEST_PLAN

Functional Coverage Plan

Click on the link to access the coverage plan of the ALU : ALU_COVERAGE_PLAN

Assertions Plan

Click on the link to access the assertion plan of the ALU : ALU_ASSERTION_PLAN