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.

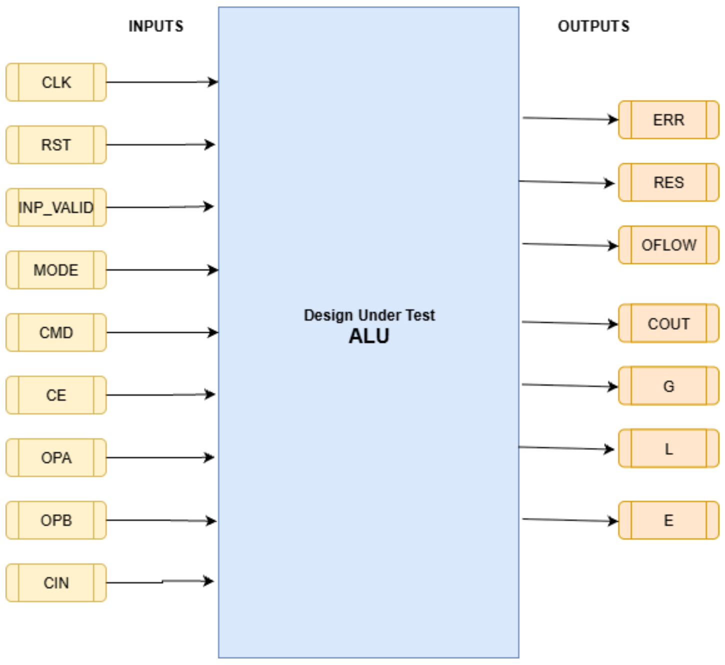

Figure 1. ALU under test

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

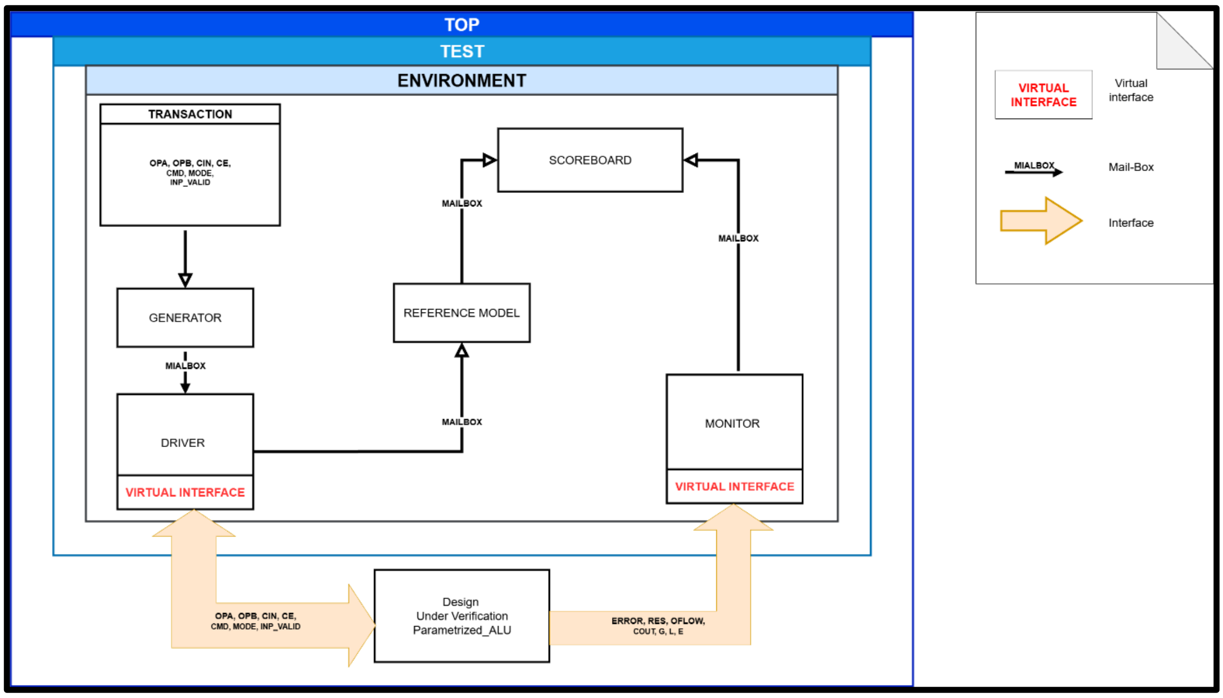

Figure 1. ALU Testbench Architecture

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).

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.

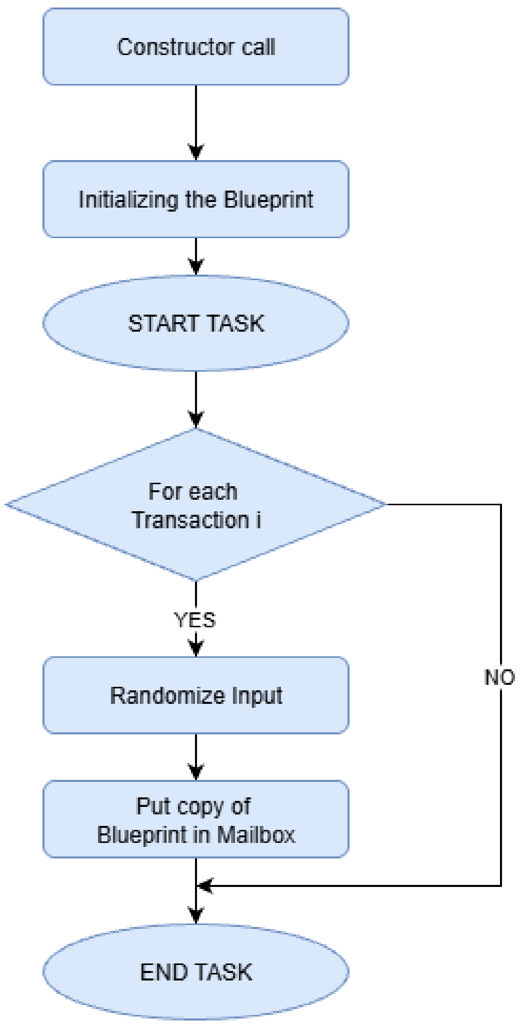

Figure 2. Flowchart of ALU_Generator

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.

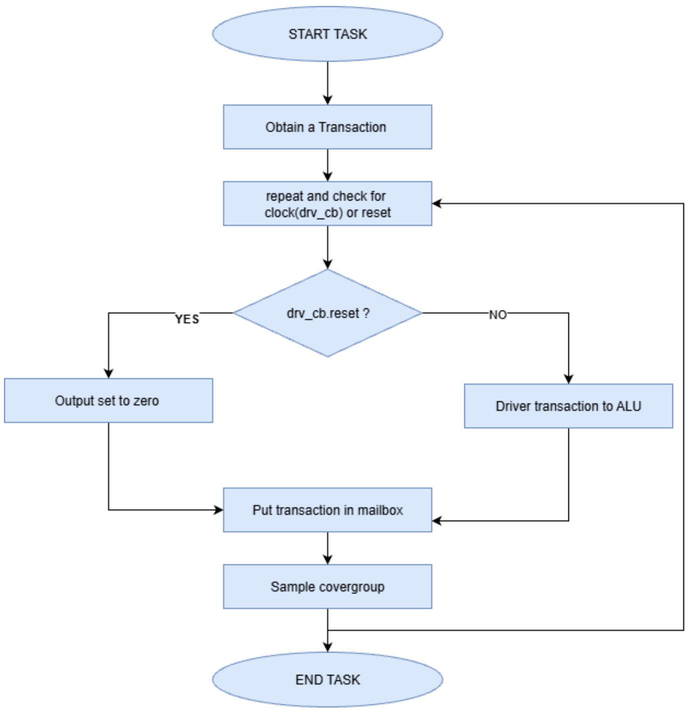
START TASK
Obtain a Transaction
repeat and check for clock(drv_cb) or reset
drv_cb.reset ?
YES
NO
Output set to zero
Driver transaction to ALU
Put transaction in mailbox
Sample covergroup
END TASK

Figure 3. Flowchart of task in the 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.

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.

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.

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

Below is the test plan for the general signal.

Feature Name	Feature Description	TestCase	Expected Output
CLK Toggle	Check if the clock is toggling	---	clk = ~clk
RST	When reset is set the output signals set to zero	RST = 1	RES= 0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0 ,
CE	n CE(Clock_Enable) is set to 1 the ALU operates the specific operation at positive edge of the CLK	CE = 1	J operates the specific operation at positive edge of the CLK
INP_VALID	if the INP_VALID is set to Zero the output signals are set to zero	INP_VALID == 2'b00	RES= 0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0 ,
	if the INP_VALID is set to One ,ALU performs operation that requires single operand on OPA	INP_VALID == 2'b01	ration that require single operand are executed after One CLK cycles
	if the INP_VALID is set to Two ,ALU performs operation that requires single operand on OPB	INP_VALID == 2'b10	
	k if the INP_VALID is set to Three ,ALU performs operation that requires both OPA and OPB	INP_VALID == 2'b11	form all the operations on both the operand whose output is yed by One CLK cycle - {except multiplication whose output is obtained after two CLK cycle}
MODE	heck when MODE is set to One Arithmetic operation is implimented on the operand	MODE == 1	Arithmetic Operation
	heck when MODE is set to Zero Arithmetic operation is implimented on the operand	MODE == 0	Logical Operation

Below is the test plan for arithmetic operation.

DD UNSIGNED	if when input valid is 2'b11 ,addition of randomized OPA and OPB must take place	VALID=2'b11 , CMD=0000	n of two unsigned numbers whose result(RES) is obtained after One Clock cycle | COUT = 0/1 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
JB UNSIGNED	heck if when input valid is 2'b11 ,subtraction of randomized OPA and OPB must take place	VALID=2'b11 , CMD=0001	action of two unsigned numbers whose result(RES) is obtained after One Clock cycle | OFLOW =0/1 | COUT= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0

_CIN UNSIGNED	if when input valid is 2'b11 ,addition of randomized OPA and OPB must take place	_VALID=2'b11 , CMD=0010	on of two unsigned numbers whose result(RES) is obtained after One Clock cycle | COUT = 0/1 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
	ne operand is valid then it will wait for the other nd to be valid with in 16 clock cycles then addition opeartion will take place	ALID=2'b10 ---within 16 clk -- _VALID=2'b01/2'b11 , CMD = 0000	on of two unsigned numbers whose result(RES) is obtained after One Clock cycle | COUT = 0/1 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
_CIN UNSIGNED	heck if when input valid is 2'b11 ,subtraction of randomized OPA and OPB must take place	_VALID=2'b11 , CMD=0011	action of two unsigned numbers whose result(RES) is obtained after One Clock cycle | OFLOW =0/1 | COUT= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
INC_A	heck if when input valid is 2'b11, increment the randomized value of OPA by 1	VALID = 2'b11 , CMD = 0100	nented value of OPA must be present in the RES after one clock cycle | COUT=1/ 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
	heck if when input valid is 2'b01, increment the randomized value of OPA by 1	VALID = 2'b01 , CMD = 0100	mented value of OPA must be present in the RES after one clock cycle | COUT= 1/0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
DEC_A.	heck if when input valid is 2'b11, decrement the randomized value of OPA by 1	VALID = 2'b11 , CMD = 0101	nented value of OPA must be present in the RES after one clock cycle | OFLOW= 1/0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
	heck if when input valid is 2'b01, decrement the randomized value of OPA by 1	VALID = 2'b01 , CMD = 0101	nented value of OPA must be present in the RES after one clock cycle | OFLOW= 1/0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
INC_B	heck if when input valid is 2'b11, decrement the randomized value of OPB by 1	VALID = 2'b11 , CMD = 0110	mented value of OPB must be present in the RES after one clock cycle | COUT= 1/0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
	heck if when input valid is 2'b10, decrement the randomized value of OPB by 1	VALID = 2'b10 , CMD = 0110	mented value of OPB must be present in the RES after one clock cycle | COUT= 1/0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
DEC_B	heck if when input valid is 2'b11, decrement the randomized value of OPB by 1	VALID = 2'b11 , CMD = 0111	nented value of OPB must be present in the RES after one clock cycle | OFLOW= 1/0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
	heck if when input valid is 2'b10, decrement the randomized value of OPB by 1	VALID = 2'b10 , CMD = 0111	nented value of OPB must be present in the RES after one clock cycle | OFLOW= 1/0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
CMP	heck if when input valid is 2'b11, compare the randomized value of OPA and OPB	VALID = 2'b11 , CMD = 1000	parision output is obtained after one clock cycle RES= 0 , COUT= 0 ,OFLOW= 0 ,G=1/0 ,E=1/0 ,L=1/0 ,ERR= 0 ,
NC_A * INC_B	heck if when input valid is 2'b11 , increment the mized value of OPA and OPB and then multiply the incremented value	VALID = 2'b11 , CMD = 1001	result of the product is obtained after two clock cycles COUT= 0 ,OFLOW= 1/0 ,G=0 ,E=0 ,L= 0 ,ERR= 0
SHL1_A * B	ck if when the input valid is 2'b11 ,increment the ndomized value of OPA by 1 and multiply with randomized value of OPB	VALID = 2'b11 , CMD = 1010	result of the product is obtained after two clock cycles COUT= 0 ,OFLOW= 1/0 ,G=0 ,E=0 ,L= 0 ,ERR= 0

Below is the test plan for logical operation

AND	Check if when input valid is 2'b11 ,AND eration for the randomized OPA and OPB must take place	P_VALID=2'b11 , CMD=0000	result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
NAND	heck if when input valid is 2'b11 ,NAND eration for the randomized OPA and OPB must take place	P_VALID=2'b11 , CMD=0001	result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
OR	if when input valid is 2'b11 ,OR operation the randomized OPA and OPB must take place	P_VALID=2'b11 , CMD=0010	esult(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
NOR	Check if when input valid is 2'b11 ,NOR eration for the randomized OPA and OPB must take place	P_VALID=2'b11 , CMD=0011	result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
XOR	Check if when input valid is 2'b11 ,XOR eration for the randomized OPA and OPB must take place	P_VALID=2'b11 , CMD=0100	result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
XNOR	heck if when input valid is 2'b11 ,XNOR eration for the randomized OPA and OPB must take place	P_VALID=2'b11 , CMD=0101	result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
NOT_A	eck if when input valid is 2'b11, take the npliment of the randomized value of OPA	_VALID = 2'b11 , CMD = 0110	A result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
	eck if when input valid is 2'b01,take the npliment of the randomized value of OPA	_VALID = 2'b01 , CMD = 0110	A result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
NOT_B	eck if when input valid is 2'b11, take the npliment of the randomized value of OPB	_VALID = 2'b11 , CMD = 0111	B result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
	eck if when input valid is 2'b01,take the npliment of the randomized value of OPB	_VALID = 2'b01 , CMD = 0111	B result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
SHR1_A	if when input valid is 2'b11, then shift the domized value of OPA to the right by 1 bit	_VALID = 2'b11 , CMD = 1000	right shift of OPA result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
	if when input valid is 2'b01, then shift the domized value of OPA to the right by 1 bit	_VALID = 2'b01 , CMD = 1000	right shift of OPA result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
SHL1_A	if when input valid is 2'b11, then shift the domized value of OPA to the left by 1 bit	_VALID = 2'b11 , CMD = 1001	left shift of OPA result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
	if when input valid is 2'b01, then shift the domized value of OPA to the left by 1 bit	_VALID = 2'b01 , CMD = 1001	left shift of OPA result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0

SHR1_B	if when input valid is 2'b11, then shift the domized value of OPB to the right by 1 bit	_VALID = 2'b11 , CMD = 1010	right shift of OPB result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
	if when input valid is 2'b10, then shift the domized value of OPB to the right by 1 bit	_VALID = 2'b10 , CMD = 1010	right shift of OPB result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
SHL1_B	if when input valid is 2'b11, then shift the domized value of OPB to the left by 1 bit	_VALID = 2'b11 , CMD = 1011	left shift of OPB result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
	if when input valid is 2'b10, then shift the domized value of OPB to the left by 1 bit	_VALID = 2'b10 , CMD = 1011	left shift of OPB result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
ROL_A_B	k if when input valid is 2'b11 ,roll left OPA by OPB times	P_VALID=2'b11 , CMD=1100	result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0
ROR_A_B	if when input valid is 2'b11 ,roll right OPA by OPB times	P_VALID=2'b11 , CMD=1101	result(RES) is obtained after One Clock cycle | COUT = 0 | OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=0

Corner conditions for the test plan are as below:

INP_VALID	if the INP_VALID is set to Zero the output signals are set to zero	INP_VALID == 2'b00	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
Arithmetic Operation for : MODE = 1 | RST=0 | CE=1			
ADD UNSIGNED		P_VALID == 2'b01 / 2'b10 , CMD = 0000	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
SUB UNSIGNED		P_VALID == 2'b01 / 2'b10 , CMD = 0001	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
ADD_CIN UNSIGNED		P_VALID == 2'b01 / 2'b10 , CMD = 0010	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
SUB_CIN UNSIGNED		P_VALID == 2'b01 / 2'b10 , CMD = 0011	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
CMP		P_VALID == 2'b01 / 2'b10 , CMD = 1000	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
INC_A * INC_B		P_VALID == 2'b01 / 2'b10 , CMD = 1001	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
SHL1_A * B	ut valid is 2'b01 / 2'b10 then error flag is raised	P_VALID == 2'b01 / 2'b10 , CMD = 1010	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
Logical Operation for : MODE = 0 | RST=0 | CE=1			
AND	ut valid is 2'b01 / 2'b10 then error flag is raised	P_VALID == 2'b01 / 2'b10 , CMD = 0000	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1

NAND	ut valid is 2'b01 / 2'b10 then error flag is raised	P_VALID == 2'b01 / 2'b10 , CMD = 0001	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
OR	ut valid is 2'b01 / 2'b10 then error flag is raised	P_VALID == 2'b01 / 2'b10 , CMD = 0010	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
NOR	ut valid is 2'b01 / 2'b10 then error flag is raised	P_VALID == 2'b01 / 2'b10 , CMD = 0011	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
XOR	ut valid is 2'b01 / 2'b10 then error flag is raised	P_VALID == 2'b01 / 2'b10 , CMD = 0100	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
XNOR	ut valid is 2'b01 / 2'b10 then error flag is raised	P_VALID == 2'b01 / 2'b10 , CMD = 0101	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
NOT_A	put valid is 2'b10 then error flag is raised	INP_VALID == 2'b10 , CMD = 0110	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
NOT_B	put valid is 2'b01 then error flag is raised	INP_VALID == 2'b01 , CMD = 0111	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
SHR1_A	put valid is 2'b10 then error flag is raised	INP_VALID == 2'b10 , CMD = 1000	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
SHL1_A	put valid is 2'b10 then error flag is raised	INP_VALID == 2'b10 , CMD = 1001	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
SHR1_B	put valid is 2'b01 then error flag is raised	INP_VALID == 2'b01 , CMD = 1010	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
SHL1_B	put valid is 2'b01 then error flag is raised	INP_VALID == 2'b01 , CMD = 1011	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
ROL_A_B	ut valid is 2'b01 / 2'b10 then error flag is raised	P_VALID == 2'b01 / 2'b10 , CMD = 1100	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1
ROR_A_B	ut valid is 2'b01 / 2'b10 then error flag is raised	P_VALID == 2'b01 / 2'b10 , CMD = 1101	0 , COUT=0 ,OFLOW= 0 ,G= 0 ,E=0 ,L= 0 ,ERR=1

FUNCTIONAL COVERAGE PLAN

Below is the coverage that has been implemented for the verification of ALU

Sl. No.	Coverpoints	Pin Name	Description
Simple Coverage			
1.1	CE_cp	CE	Check if CE has taken both values 1 and 0
1.2	MODE_cp	MODE	Check if both arithmetic (MODE == 1) and logical (MODE == 0) are covered
1.3	INP_VALID_cp	INP_VALID	Check if INP_VALID signals has covered all 3 cases {0,1,2,3}
1.4	CMD_1	CMD	if MODE==1 ,CMD has to cover 11 values (0 to 10)
1.5	CMD_0	CMD	if MODE==0 ,CMD has to cover 14 values (0 to 13)
1.6	A_cp	OPA	Check all bits are toggled
1.7	B_cp	OPB	Check all bits are toggled
1.8	CIN_cp	CIN	Check all bits are toggled
1.1	CE_cp	CE	Check if CE has taken both values 1 and 0
1.2	MODE_cp	MODE	Check if both arithmetic (MODE == 1) and logical (MODE == 0) are covered
1.3	INP_VALID_cp	INP_VALID	Check if INP_VALID signals has covered all 3 cases {0,1,2,3}
Cross Coverage			
1.1	MODE_cp_x_CMD_1	MODE_cp,CMD_1	Cross coverage between MODE and CMD (for Arithmetic operation)
1.2	MODE_cp_x_CMD_0	MODE_cp,CMD_0	Cross coverage between MODE and CMD (for Logical operation)

ASSERTIONS

Randomized

Sl. No.	Features	Signals	Scenarios	Description
Valid_Check				
1.1	Check input signals	INP_VALID	Check if signals are valid (except 'x' or 'z')	All the signals to be valid (except 'x' or 'z')
1.2		MODE		
1.3		CMD		
1.4		CE		
1.5		OPA		
1.6		OPB		
1.7		CIN		
Reset_Check				
1.1		RST	Check on asserting reset	when reset is asserted, the

				output is set to zero at posedge clk
1.2	Check reset signal	RST	Check on de-asserting reset	when reset is de-asserted, the output obtained is based on the particular operation
Timing_Check				
1.1	Check for stability.	CE	Check clock enable stability	CE must be high during the ALU operation
1.2	Check for stability.	MODE,CMD	Check mode and command are stable	MODE and CMD should be stable for respective of ALU operation
1.3	Waiting Period	INP_VALID	INP_VALID is 2'b01 or 2'b10	Wait for 16 clock cycles for INP_VALID to become 2'b11
1.4	Latch Check	CE	Clock enable de-assertion	When clock enable is de-asserted the output of the ALU is latched
1.5	One-Cycle Delay	ERR,RES,G,L,E,COUT	Except multiplication operation	The result for the particular operation is executed after one clock cycle delay
1.6	Two-Cycle Delay	ERR,RES,G,L,E,COUT	For multiplication operation	The result for the multiplication operation is executed after two clock cycle delay