ALU UVM DESIGN REQUIRMENTS PDF

Timescale & Basic Info

Property	Value	Description
Timescale	1ns/1ps	System timing precision
Clock Edge	Positive	All operations synchronous to rising edge
Reset Type	Asynchronous	Active low reset (rst_n)

Parameters

Parameter	Default Value	Description	Usage
INPUT_WIDTH	5	Width of input operands	Defines bit width of A and B inputs
OUTPUT_WIDTH	6	Width of output	Defines bit width of C output
A_OP_WIDTH	3	A operation width	Defines operation selection range for A
B_OP_WIDTH	2	B operation width	Defines operation selection range for B
EN_WIDTH	1	Enable width	Defines width of enable signals

Interface Signals

Signal	Direction	Width	Type	Default Radix	Description
rst_n	Input	1	Wire	unsigned	Active low reset
clk	Input	1	Wire	unsigned	System clock
ALU_en	Input	1	Wire	unsigned	ALU enable
a_en	Input	[EN_WIDTH-1:0]	Wire	unsigned	Enable signal controlling which opcode mode operates
b_en	Input	[EN_WIDTH-1:0]	Wire	unsigned	Enable signal controlling which opcode mode operates
a_op	Input	[A_OP_WIDTH-1:0]	Wire	unsigned	Opcode, each opcode corresponds to a certain operation
b_op	Input	[B_OP_WIDTH-1:0]	Wire	unsigned	Opcode, each opcode corresponds to a certain operation
A	Input	[INPUT_WIDTH-1:0]	Wire	Signed	Signed input operand A
B	Input	[INPUT_WIDTH-1:0]	Wire	Signed	Signed input operand B
C	Output	[OUTPUT_WIDTH-1:0]	Reg	Signed	Signed output, ALU result

Design Constraints

Signal	Direction	Width	Constraints	Invalid Range
A	Input	[INPUT_WIDTH-1:0]	[-15:15]	-16
B	Input	[INPUT_WIDTH-1:0]	[-15:15]	-16
C	Output	[OUTPUT_WIDTH-1:0]	[-31:31]	-32

Solution?

Constraints to ensure said values are not hit
(a_op == A_XNOR && a_en && ~b_en) -> (A^B) != 5'b11111;
(b_op == B01_NAND && ~a_en && b_en) -> (A&B) != 5'b11111;
(b_op == B11_XNOR && a_en && b_en) -> (A^B) != 5'b11111;

ALU UVM DESIGN REQUIRMENTS PDF

Verification Test Names & Description

TEST_NAME	Description
random_test	randomizes inputs according to specs & drives them
repitition_test	focuses on repeating each operation of each opcode of each mode 10* times
error_test	Focuses on driving random inputs but without specification constraints, trying to see how the dut behaves (if it hangs or not and trigger assertion CEXs

Special Conditions

Condition	Behavior
rst_n = 0	Output C cleared to 0 asynchronously
ALU_en = 0	Output C maintains current value
Invalid op codes	Display warning message
Operation timing	One clock cycle latency

Opcodes

Enable Mode	Operation Select	Function	Description
MODE_A (a_en & ~b_en)	0	A + B	Addition
	1	A - B	Subtraction
	2	A ^ B	XOR
	3,4	A & B	AND
	5	A | B	OR
	6	~(A ^ B)	XNOR
	7	NULL	return 0
MODE_B01 (~a_en & b_en)	0	~(A & B)	NAND
	1,2	A + B	Addition
	3	NULL	return 0
MODE_B11 (a_en & b_en)	0	A ^ B	XOR
	1	~(A ^ B)	XNOR
	2	A - 1	Decrement A
	default	B + 2	Increment B by 2
MODE_IDLE (~a_en & ~b_en)	N/A	C <= C	Hold current value

ALU UVM DESIGN REQUIRMENTS PDF

Verification Plan				
Plan	Description	Tests	Coverage	Assertion
Reset DUT	Resetting the DUT using two methods (directly from the topmodule and from reset_sequence being called within the virtual sequences) The output is being asserted during this time period through an assertion based coverage at the sva_interface	Random Test & Repetition Test & Error Test	✗	✓
ALU_en	Sending transactions while ALU_en is ON & OFF and checking the behaviour of the DUT against the expected behaviour in the Scoreboard	Random Test	✓	✓
A_en & B_en	Checking that the correct mode of operation is actually being handled correctly accoridng to the values of the A_en & B_en	Random Test & Repetition Test & Error Test	✓	✓
A_op VALID Opcodes	Checking that all the VALID A_op Opcodes are behaving correctly against the expected behaviour	Random Test & Repetition Test	✓	✓
B_01op VALID Opcodes	Checking that all the VALID B_01op Opcodes are behaving correctly against the expected behaviour	Random Test & Repetition Test	✓	✓
B_11op VALID Opcodes	Checking that all the VALID B_11op Opcodes are behaving correctly against the expected behaviour	Random Test & Repetition Test	✓	✓
Randomizing VALID inputs	Randomly Resetting the DUT during the entire simulation & Randomizing VALID inputs & Driving them while simultaniously checking the output of the DUT against the expected behaviour	Random Test	✓	✓
Repeating VALID inputs	Randomly Resetting the DUT during the entire simulation & Repeating VALID inputs & Driving them while simultaniously checking the output of the DUT against the expected behaviour	Repetition Test	✓	✓
A_op INVALID Opcodes	Checking that all the INVALID A_op Opcodes are detected & behaving correctly against the expected behaviour without causing the design to hang!	Error Test	✓	✓
B_01op VALID Opcodes	Checking that all the INVALID B_01op Opcodes are detected & behaving correctly against the expected behaviour without causing the design to hang!	Error Test	✓	✓
Driving INVALID input operands	Checking that all the INVALID A & B operands are detected and are behaving correctly against the expected behaviour without causing the design to hang!	Error Test	✓	✓
Driving VALID inputs that generate INVALID Outputs	Checking that all the VALID inputs that could generate INVALID OUTPUTS are detected and are behaving correctly against the expected behaviour without causing the design to hang!	Error Test	✓	✓
Adding a TIMEOUT mechanism to make sure the Environment does not HANG	Controlling the TIMEOUT threshold through the topmodule to make sure the environment does not hang and cause the tool to crash for myself & my colleagues.	Random Test & Repetition Test & Error Test	N/A	N/A
Functional Coverage	Ensuring all opcodes/values are met and tried using Functional Coverage and Fixing the DUT if not	Random Test & Repetition Test & Error Test	N/A	N/A
Code Coverage	Ensuring all the DUT's code lines were HIT using code coverage and making better test cases if not.	Random Test & Repetition Test & Error Test	N/A	N/A
Back & Forth between Code & Functional Coverage	Going back & Forth between code & functional coverage until the desired goal (100% or close to it) is achieved	Random Test & Repetition Test & Error Test	N/A	N/A
Checking Functional Coverage = 100% for each test	Adding a mechanism to make sure FUNCTIONAL COVERAGE = 100% for each test!	Random Test & Repetition Test & Error Test	✓	N/A
Checking Code Coverage = 100% for each test	Through checking the generated reports for the VALID tests as well as the ERROR tests	Random Test & Repetition Test & Error Test	✓	N/A