```
/* Notes Section Styling */
.note {
  font-weight: bold;
  font-size: 1.1em;
  color: #333;
  padding: 10px;
  background-color: #f8f9fa;
  border-left: 4px solid #007bff;
  margin: 15px 0;
}

.note ul {
  margin-top: 5px;
  padding-left: 20px;
}
```

ALU Verification Assignment

Submitted to: Eng. Nour ElDeen ElHout

By: Amira Atef, Aya El Desouky and Mohamed Ayman

Note

This repo doesn't verify the shift and rotate operations due to the purpose of their exact functions being unclear.

Two files are included:

- **ALU_1**: Verifies the ALU design in one testbench file.
- **ALU_2**: Contains a preliminary testbench done using the SV Architecture.
- **ALU_3**: Contains a full testbench done using the SV Architecture and the mailbox built-in class.

Table of Contents

Signals Definition

Name	Direction	Length	Description
clk	input	1 bit	Clock
reset	input	1 bit	Active low async. reset
valid_in	input	1 bit	validate input signals
a	input	4 bits	port A
b	input	4 bits	port B
cin	input	1 bit	carry in
ctl	input	4 bits	opcodes
valid_out	output	1 bit	validate input signals
alu	output	4 bits	alu output
carry	output	1 bit	carry out
zero	output	1 bit	zero flag output

ALU Operations

OP Code	Function
4'b0000	Select data on port B
4'b0001	Increment data on port B
4'b0010	Decrement data on port B
4'b0011	ADD without CARRY
4'b0100	ADD with CARRY
4'b0101	SUB without BORROW
4'b0110	SUB with BORROW
4'b0111	AND
4'b1000	OR
4'b1001	XOR
4'b1010	Shift left
4'b1011	Shift right
4'b1100	Rotate left
4'b1101	Rotate right

```
module alu(
        //Input signals
        input       clk,
        input       reset,
        input       valid_in,   //validate input signals
        input [3:0] a,          //port A
        input [3:0] b,          //port B
        input       cin,        //carry input from carry flag register
        input [3:0] ctl,        //functionality control for ALU
        //Output signals
        output      valid_out,  //validate output signals
        output [3:0] alu,       //the result
        output      carry,      //carry output
        output      zero,       //zero output
        );
```

Figure 1: ALU Operations

Verification Environment Architecture

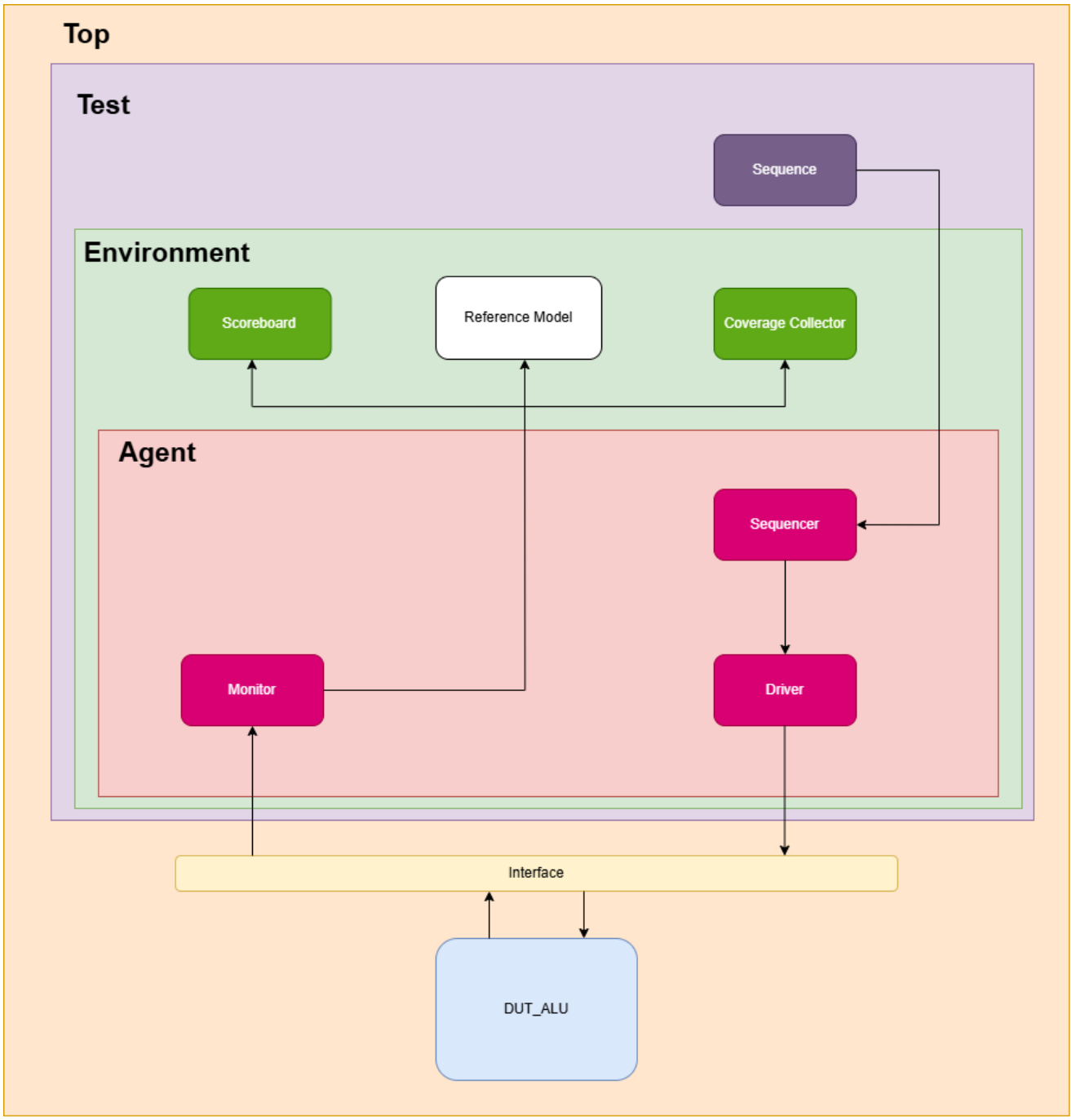


Figure 2: Verification Arichitecture

Wave Diagrams

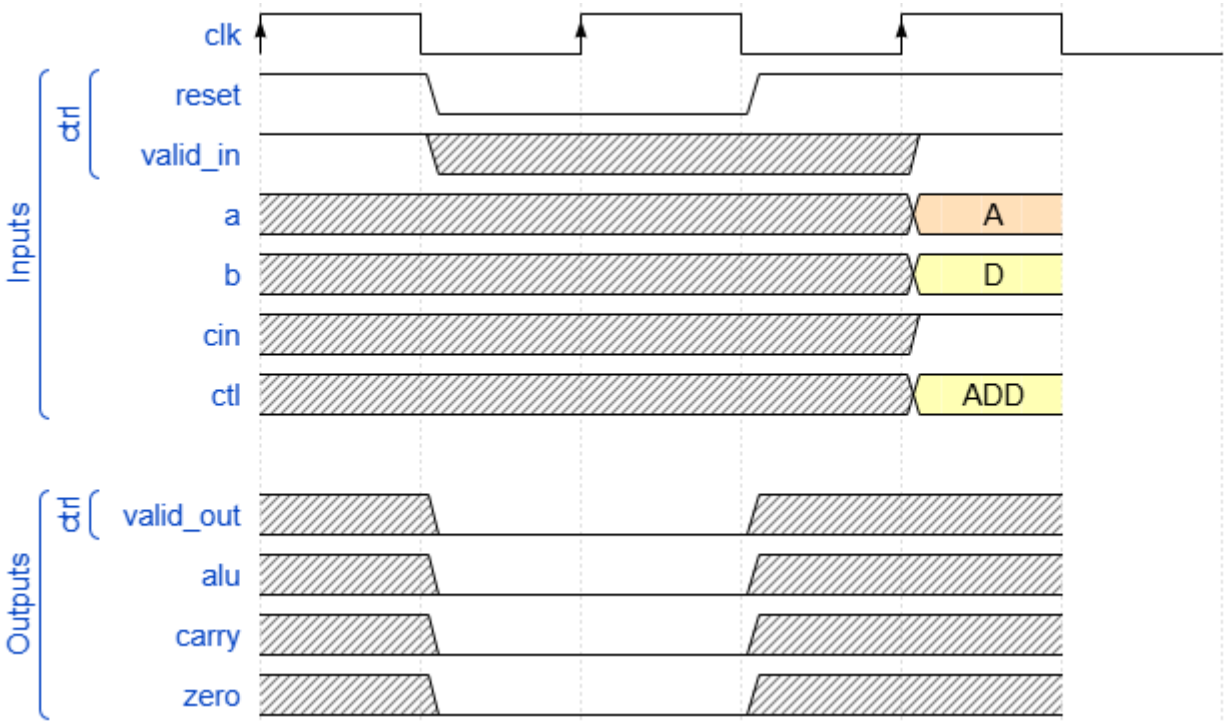


Figure 3: Reset

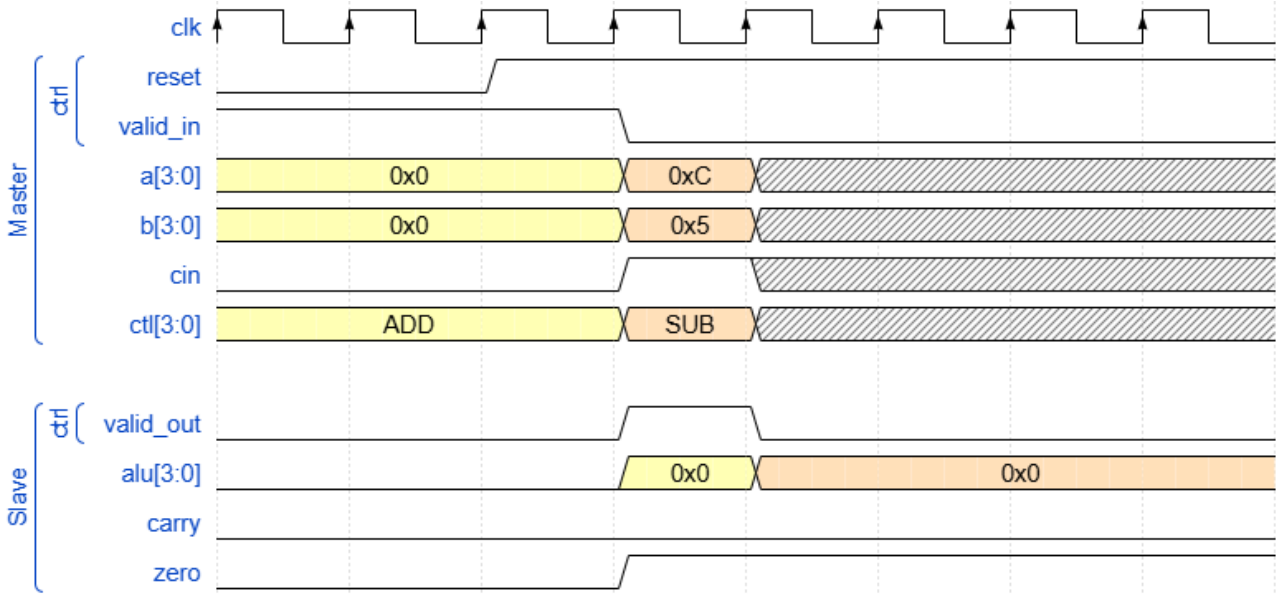


Figure 4: Valid_in

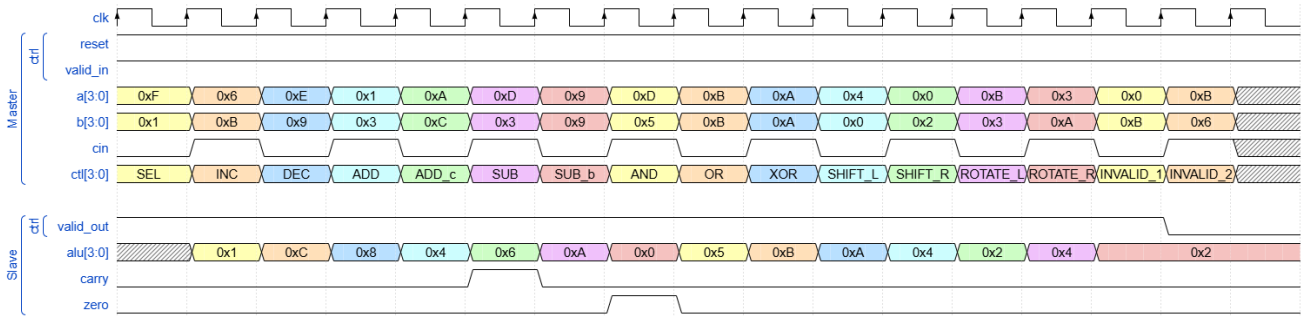


Figure 5: ALU Operations

Verification Plan

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
ALU_1	When the reset is deasserted, the outputs values should be low.	Directed at the start of the simulation, then randomized with constraint that drive the reset to be off(high) most of the simulation time.	-	A checker in the testbench to make sure the output is correct
ALU_2	Verifying extreme values on a and b, with OPCODE = ADD, SUB, ADD_c or SUB_b.	Randomization under constraints on the a and b signals to take values (MAX and ZERO) most of the simulation time.	Covered the extreme values, the rest of the values and walking ones for both a and b. Covered the ADD, SUB, ADD_c and SUB_b opcode values.	A checker in the testbench to make sure the output is correct
ALU_3	When OPCODE = 0 or 1 or 2 the operations are solely done on input b.	Randomization under constraints on signal b to take values (MAX and ZERO) most of the simulation time.	Covered the SEL, INC and DEC opcode values.	A checker in the testbench to make sure the output is correct
ALU_4	When OPCODE = AND, OR or XOR, the output out value must be the ANDing, ORing or XORing of a and b.	Randomization under constraints on the input B most of the time to have one bit high in its 3 bits while constraining the A to be low.	Covered the AND, OR and XOR opcode values.	A checker in the testbench to make sure the output is correct
ALU_5	When OPCODE = SHIFT_L, SHIFT_R, ROTATE_L or ROTATE_R, the output value must be shifted or rotated accordingly.	Randomization under constraints on the inputs a and b when opcode= shift or rotate, do not constraint the inputs a or b when the operation is shift or rotate.	Covered the SHIFT_L, SHIFT_R, ROTATE_L and ROTATE_R values for the opcode.	A checker in the testbench to make sure the output is correct
ALU_6	When OPCODE is != ADD, SUB, ADD_c or SUB_b, carry signal must be low.	Randomization under constraints on the opcode to be anything but ADD, SUB, ADD_c or SUB_b, do not constraint the inputs a or b.	Covered the SEL, INC, DEC, AND, OR, XOR, SHIFT_L, SHIFT_R, ROTATE_L and ROTATE_R values for the opcode.	A checker in the testbench to make sure the output is correct

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
ALU_7	When alu= 4'b0 and carry = 1'b0, zero flag should be 1.	Randomization	-	A checker in the testbench to make sure the output is correct
ALU_8	When Valid_in = 0, input is not valid, so output values don’t change and valid_out = 0.	Randomization	Covering valid inputs and outputs.	A checker in the testbench to make sure the output is correct
ALU_9	When Valid_in = 1, input is valid, so output values adjust and valid_out = 1.	Randomization	Covering valid inputs and outputs.	A checker in the testbench to make sure the output is correct
ALU_10	When OPCODE = 14 or 15. output stays the same and valid_out = 0.	Randomization	Covering ctl values that aren't opcodes.	A checker in the testbench to make sure the output is correct
CROSS 1	a, b at ADD, SUB, ADD_c or SUB_b.	Randomization	When the ALU is addition or subtraction, a and b should have taken all permutations of maxpos and zero.	Output Checked against reference model.
CROSS 2	cin	Randomization	When the ALU is addition w/ carry or subtracition w/ carry, cin should have taken 0 or 1	Output Checked against reference model.
CROSS 3	a, b at OR & XOR & AND	Randomization	When the ALU is AND, OR or XOR, then a and b take all walking one patterns (0001, 0010, 0100 and 1000).	Output Checked against reference model.

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
CROSS 4	a, b at OR & XOR & AND	Randomization	When the ALU is AND, OR or XOR, then a and b take all ones and all zeroz (0000 and 1111).	Output Checked against reference model.
Reference model	when the randomized inputs enter it, the output should be accurately equal to correct output of the design	Randomization	-	Output Checked against reference model.
Check Result	when the expected values from the golden model are not equal to the outputs of the DUT, the error_count increased, else correct_count increased.	Randomization	-	Output Checked against reference model.

QuestaSim Transcript Output

```
# Incorrect zero flag!,                9965000
# ** Error: Assertion error.
#    Time: 9970 ns Started: 9960 ns  Scope: alu_top.dut.SVA_inst File: alu_SVA.sv Line: 142
# Incorrect alu!,              9975000
# ** Error: Assertion error.
#    Time: 9980 ns Started: 9970 ns  Scope: alu_top.dut.SVA_inst File: alu_SVA.sv Line: 28
# ** Error: Assertion error.
#    Time: 9980 ns Started: 9970 ns  Scope: alu_top.dut.SVA_inst File: alu_SVA.sv Line: 142
# Incorrect valid_out!,              9985000
# ** Error: Assertion error.
#    Time: 9990 ns Started: 9980 ns  Scope: alu_top.dut.SVA_inst File: alu_SVA.sv Line: 156
# ** Error: Assertion error.
#    Time: 9990 ns Started: 9980 ns  Scope: alu_top.dut.SVA_inst File: alu_SVA.sv Line: 142
# Incorrect zero flag!,              9995000
# ** Error: Assertion error.
#    Time: 10 us Started: 9990 ns  Scope: alu_top.dut.SVA_inst File: alu_SVA.sv Line: 142
# Incorrect zero flag!,             10005000
# The simulation has been finished: Error Count = 992, Correct Count = 10
# ** Note: $stop    : alu_monitor.sv(91)
#    Time: 10005 ns  Iteration: 2  Instance: /alu_top/monitor
# Break in Module alu_monitor at alu_monitor.sv line 91
```

Figure 6: Transcript

QuestaSim Waveforms and Discovered Bugs

Bug 1 (SUB Operation):

- We can see that when the SUB operation is performed between 11 and 5 as highlighted below, the golden module produces the correct result which is 6 while the ALU produces an incorrect result which is 7.

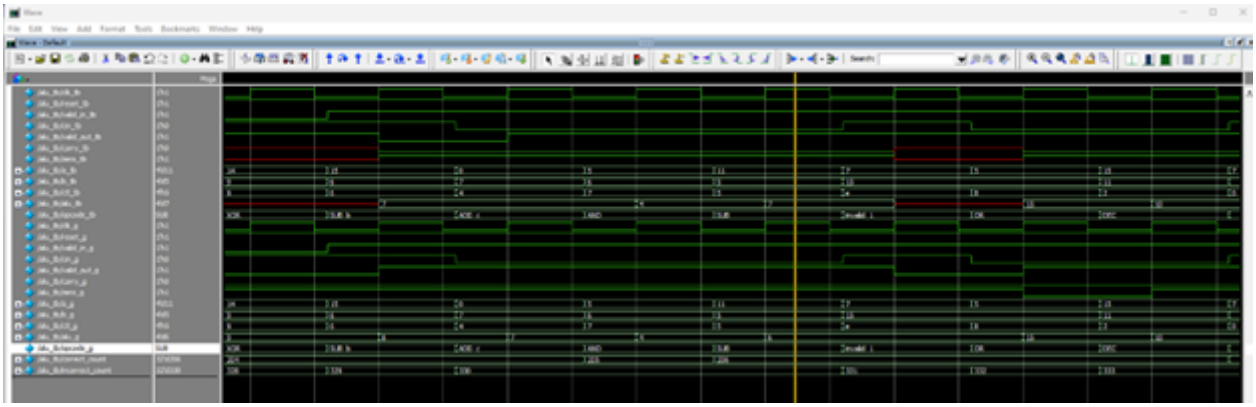


Figure 7: Waveform Snippet 1

Bug 2 (SUB w/ Borrow Operation):

-We can see that when the SUB with borrow operation is performed between 15 and 9 with cin = 0 as highlighted below, the golden module produces the correct result which is 6 while the ALU produces an incorrect result which is 5.

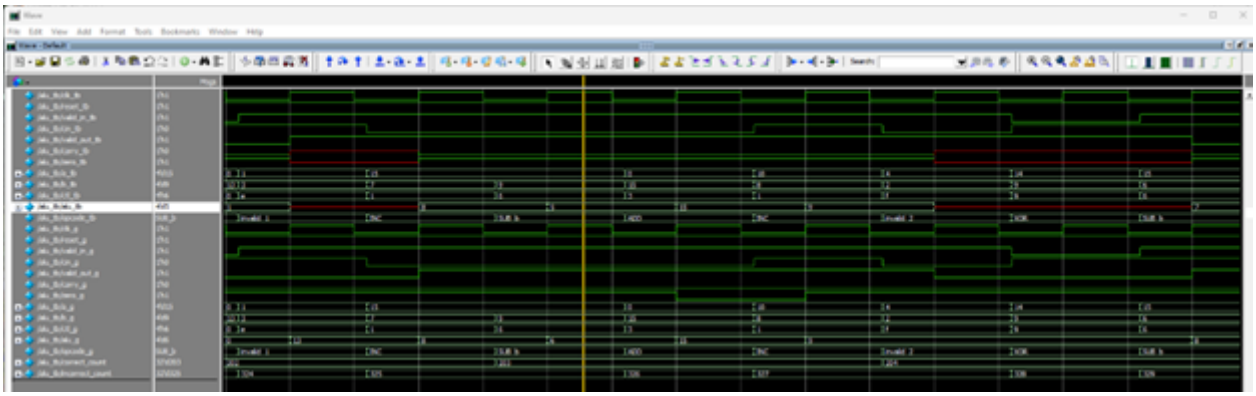


Figure 8: Waveform Snippet 2

Bug 3 (SEL Operation):

-We can see that when the SEL operation is performed with b = 10 as highlighted below, the golden module produces the correct result which is 10 (selects the value of b) while the ALU produces an incorrect result which is 7 (selects the value of a).

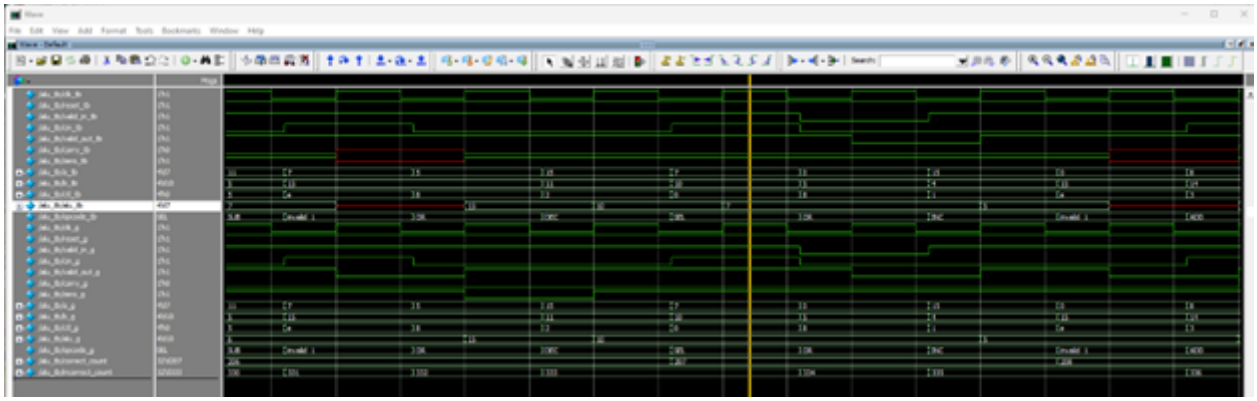


Figure 9: Waveform Snippet 3

Bug 4 (Valid out value in case of XOR Operation):

- We can see that when the XOR operation is performed with a = 14 and b = 15 as highlighted below, BOTH the ALU and the golden module produce the correct result which is 1, however the value of the valid out in the ALU is 0 which is incorrect while the value of the valid out in the golden module is 1 which is correct.

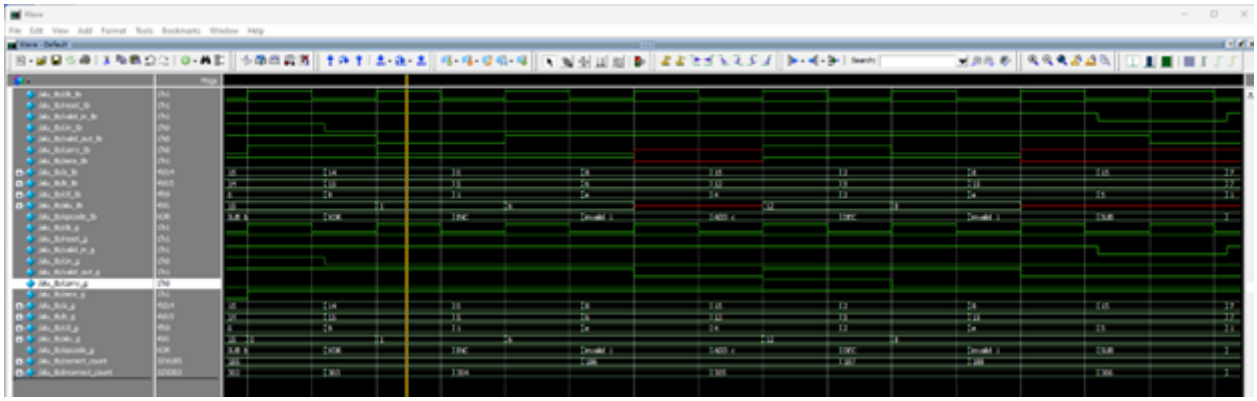


Figure 10: Waveform Snippet 4

Bug 5 (INVALID Operations):

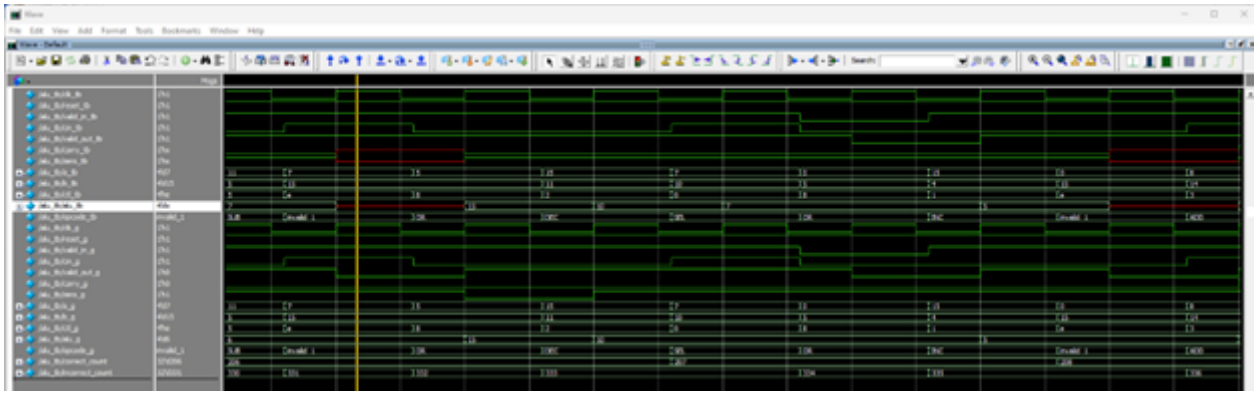


Figure 11: Waveform Snippet 5

Bug 6 (Zero Flag Value):

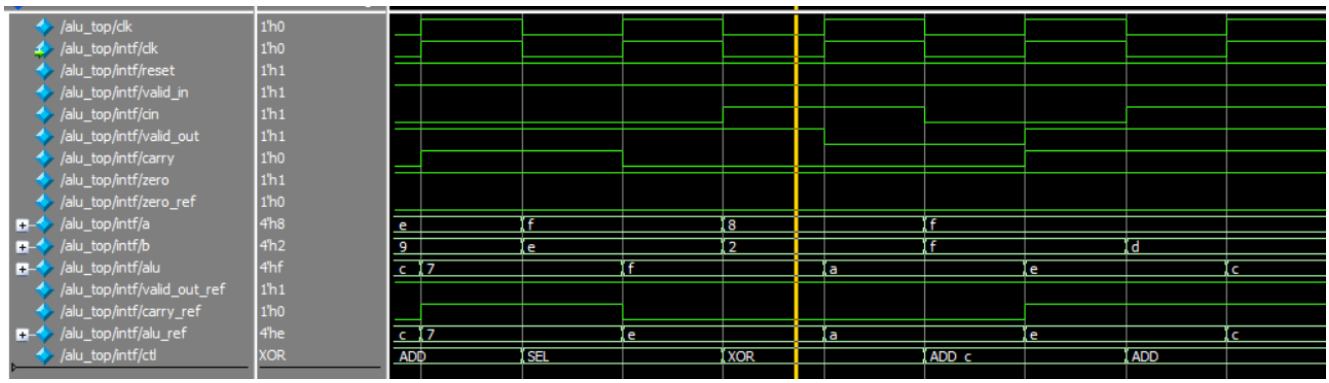


Figure 12: Waveform Snippet 6

Code Coverage Report

Statement Coverage

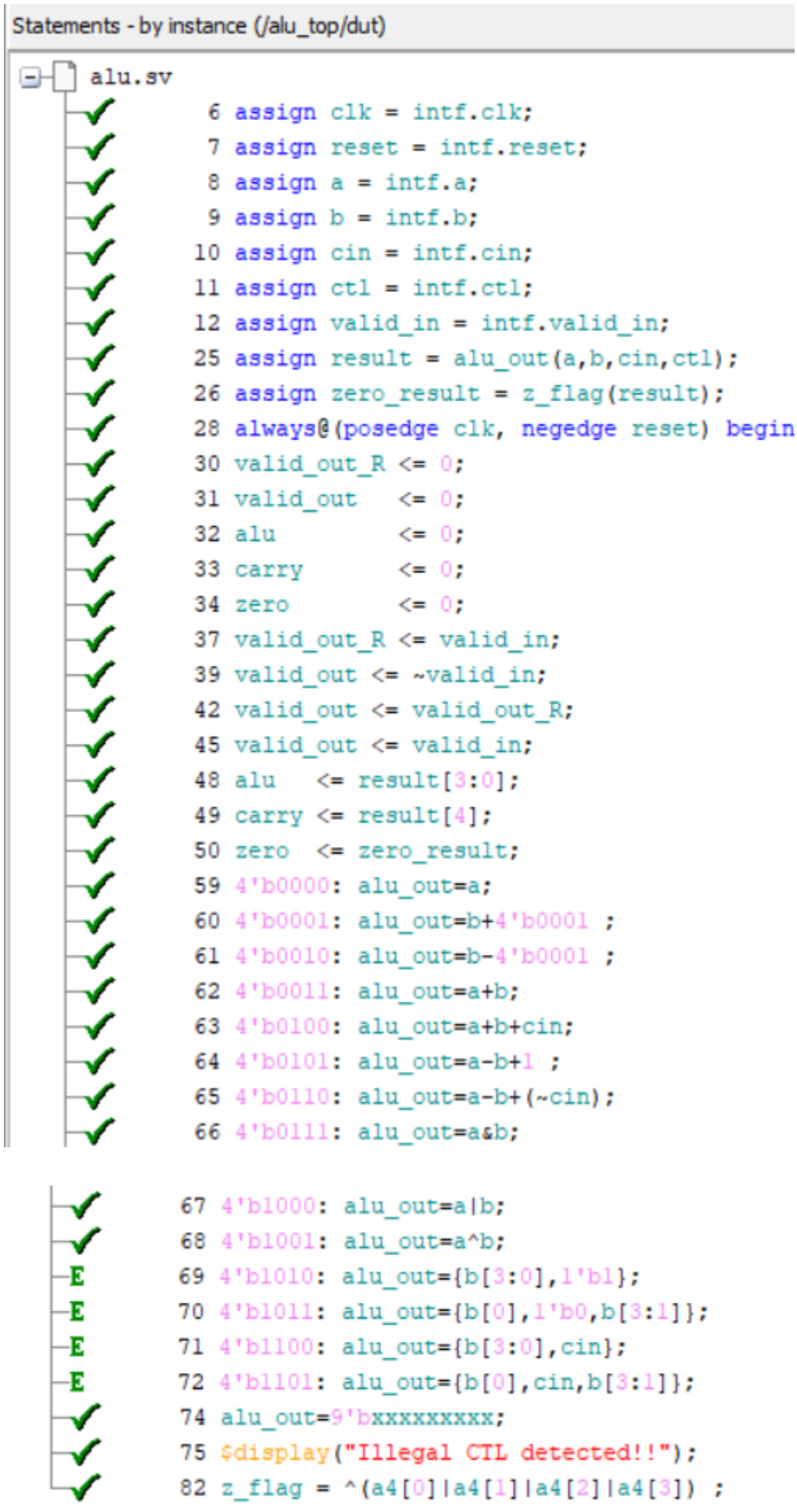


Figure 13: Statement Coverage

Branch Coverage

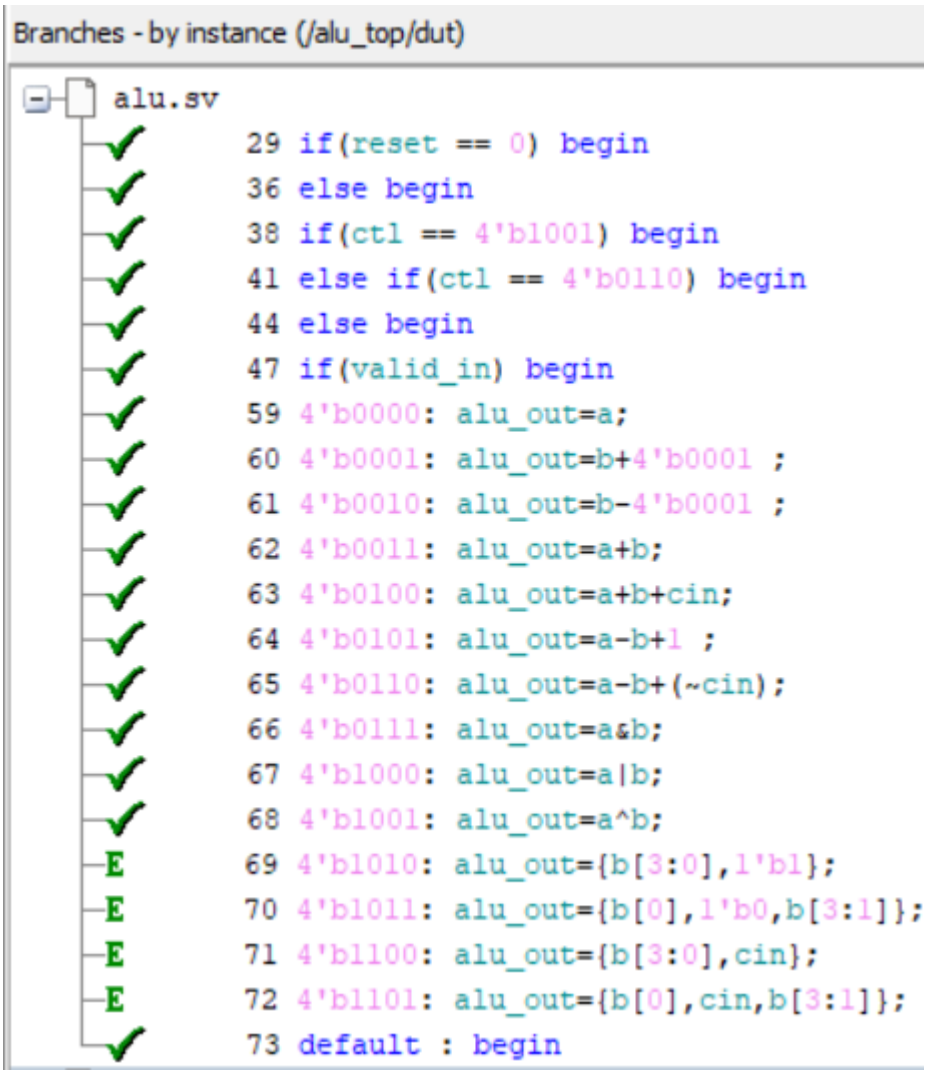


Figure 14: Branch Coverage

Toggle Coverage

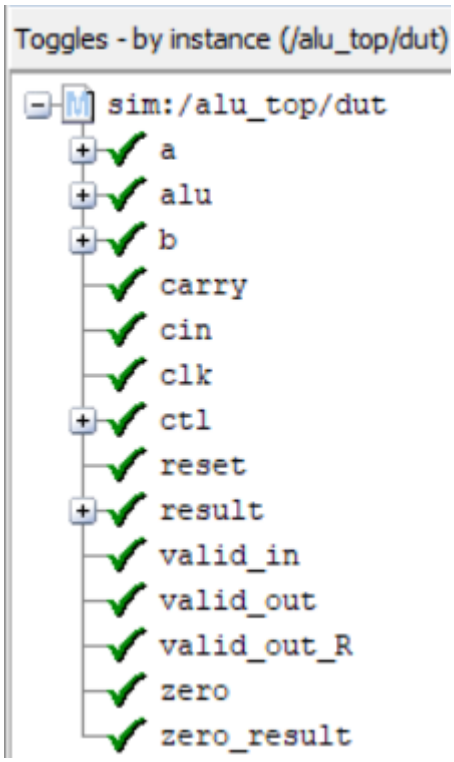
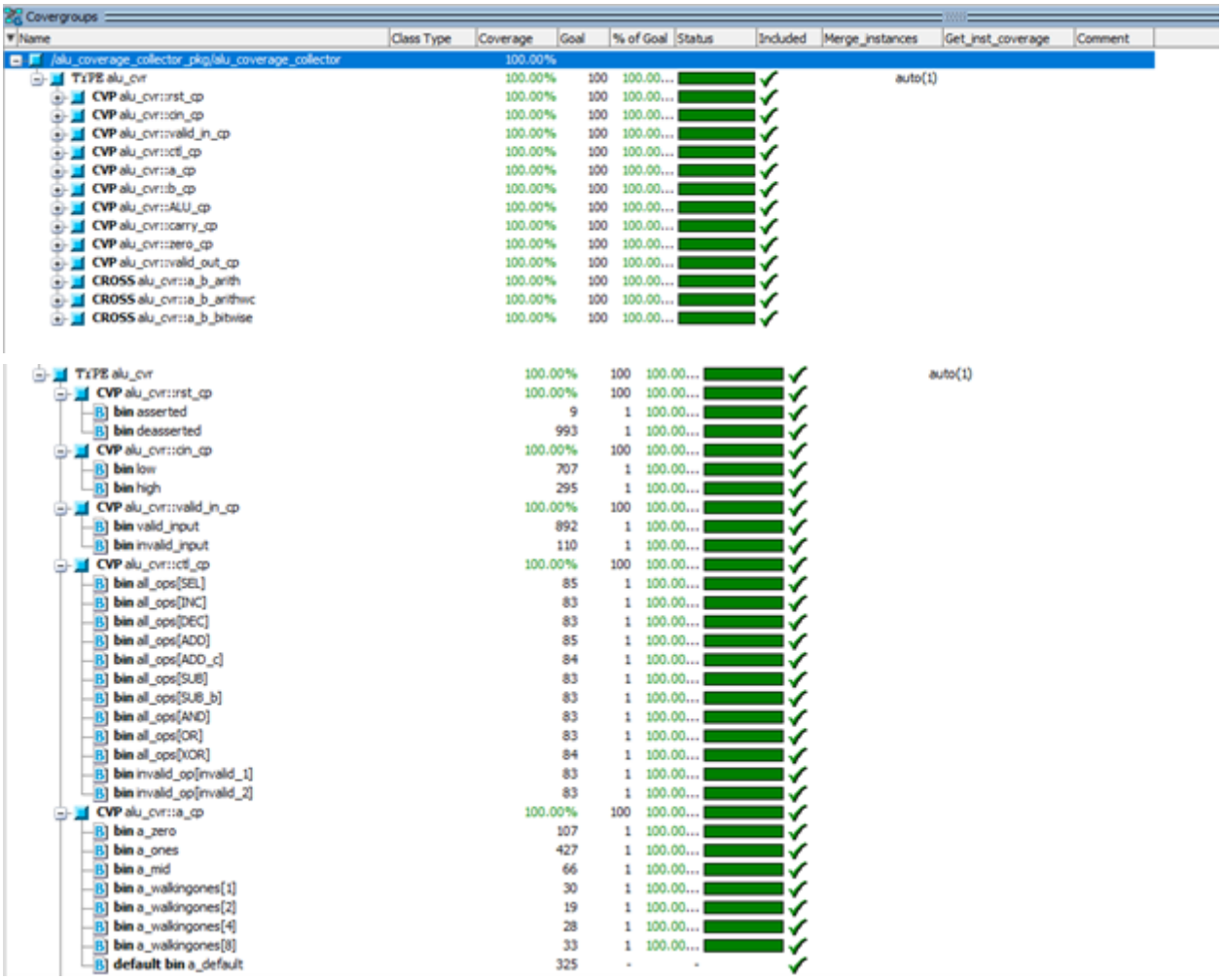


Figure 15: Toggle Coverage

Functional Coverage Report



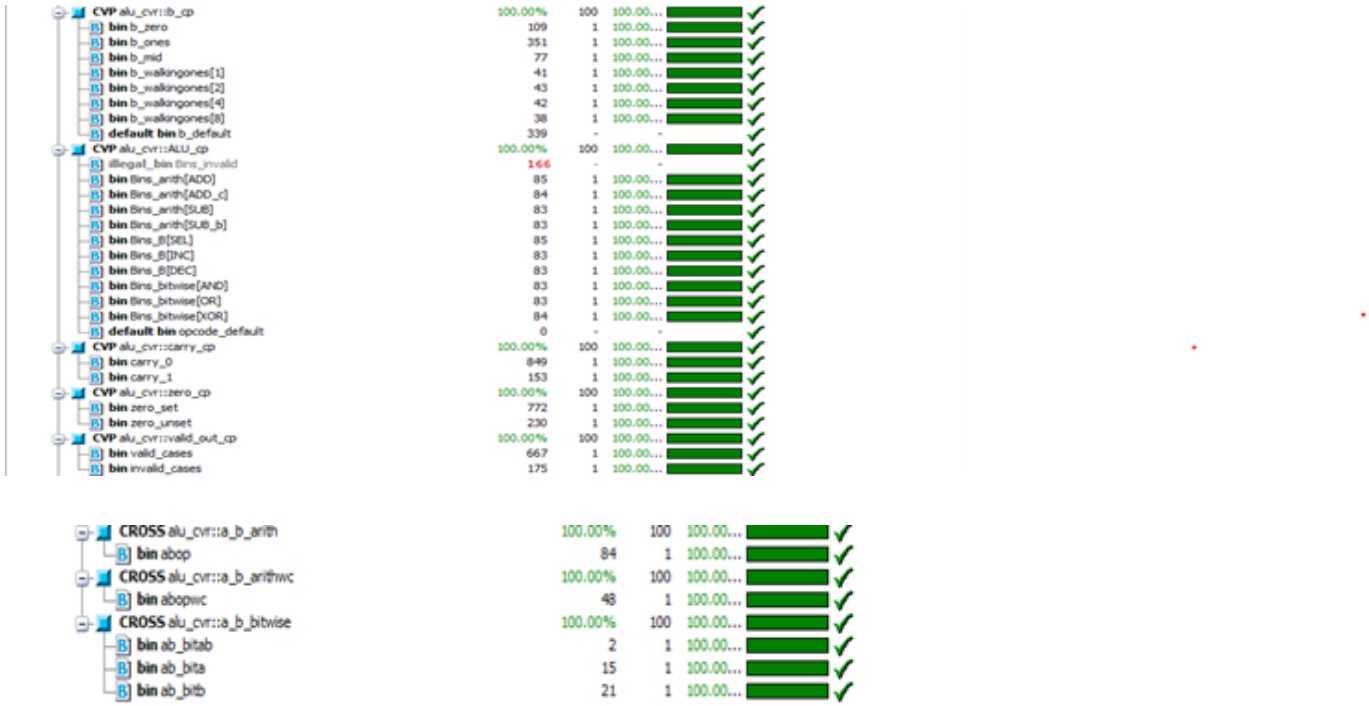


Figure 16: Functional Coverage Report

Assertions and Cover Directives

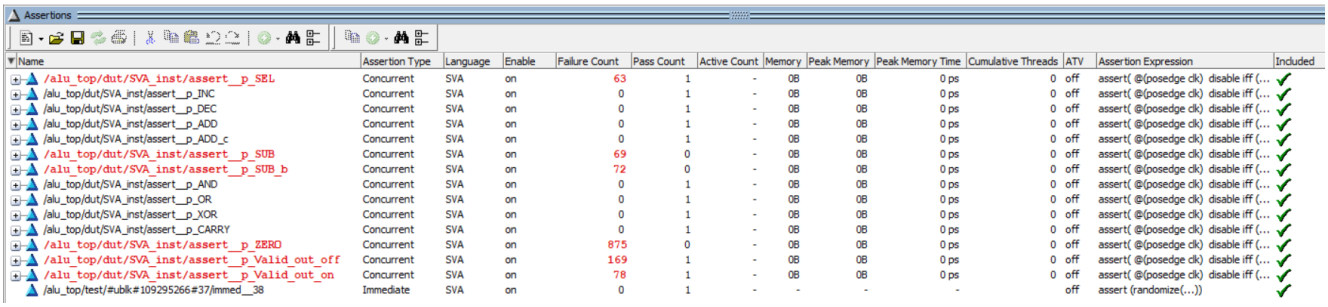


Figure 17: Assertions

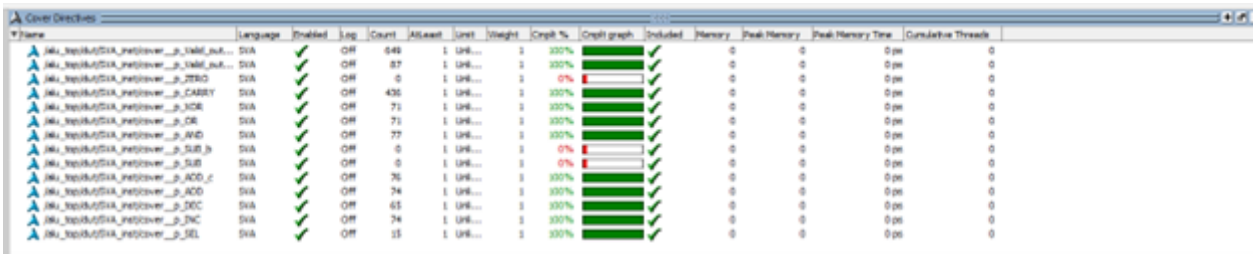


Figure 18: Cover Directives