System Verilog

Data Types & Constrained Random

Assignment 2

By: Magdy Ahmed Abbas Abdelhamid

Question 1

Dynamic Arrays

Data Types & Constrained Random Assignment

The purpose of this assignment is to:

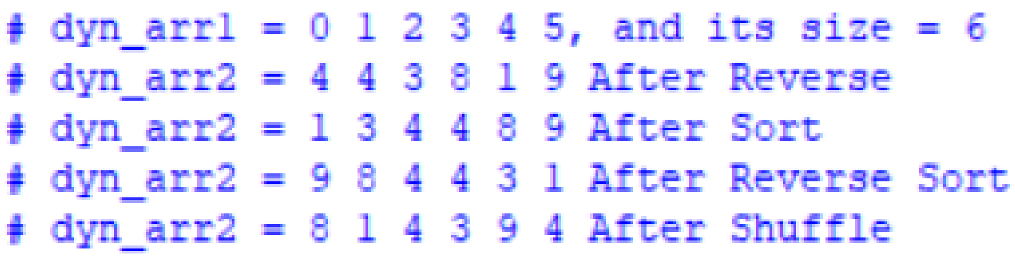
- Practice creating dynamic arrays, associative arrays, and queues
- Practice using $random & constrained randomization

1) Write a module to test dynamic array data type and its predefined methods. Run Questasim to make sure the display statements are working as expected.

- declare two dynamic arrays dyn_arr1, dyn_arr2 of type int
- initialize dyn_arr2 array elements with (9,1,8,3,4,4)
- allocate six elements in array dyn_arr1
- initialize array dyn_arr1 with index as its value
- display dyn_arr1 and its size
    - Expected output: (0,1,2,3,4,5), 6
- delete array dyn_arr1
- reverse, sort, reverse sort and shuffle the array dyn_arr2 and display dyn_arr2 after using each method (refer to slide 71 in the notes)
    - Expected output: (4,4,3,8,1,9), (1,3,4,4,8,9), (9,8,4,4,3,1), <shuffled_array>

Dynamic Arrays Testbench

```
module dynamic_array;

int dyn_arr1[];
int dyn_arr2[]  =  '{9,1,8,3,4,4};

initial begin
   dyn_arr1  =  new[6];

   foreach (dyn_arr1[i]) begin
      dyn_arr1[i]  =  i;
   end

   $display("dyn_arr1  =  %0p, and its size  =  %0d", dyn_arr1, $size(dyn_arr1));
   dyn_arr1. delete();

   dyn_arr2. reverse();
   $display("dyn_arr2  =  %0p After Reverse", dyn_arr2);

   dyn_arr2. sort();
   $display("dyn_arr2  =  %0p After Sort", dyn_arr2);

   dyn_arr2. rsort();
   $display("dyn_arr2  =  %0p After Reverse Sort", dyn_arr2);

   dyn_arr2. shuffle();
   $display("dyn_arr2  =  %0p After Shuffle", dyn_arr2);

   $stop;
end

endmodule
```



```
# dyn_arr1 = 0 1 2 3 4 5, and its size = 6
# dyn_arr2 = 4 4 3 8 1 9 After Reverse
# dyn_arr2 = 1 3 4 4 8 9 After Sort
# dyn_arr2 = 9 8 4 4 3 1 After Reverse Sort
# dyn_arr2 = 8 1 4 3 9 4 After Shuffle
```

Question 2

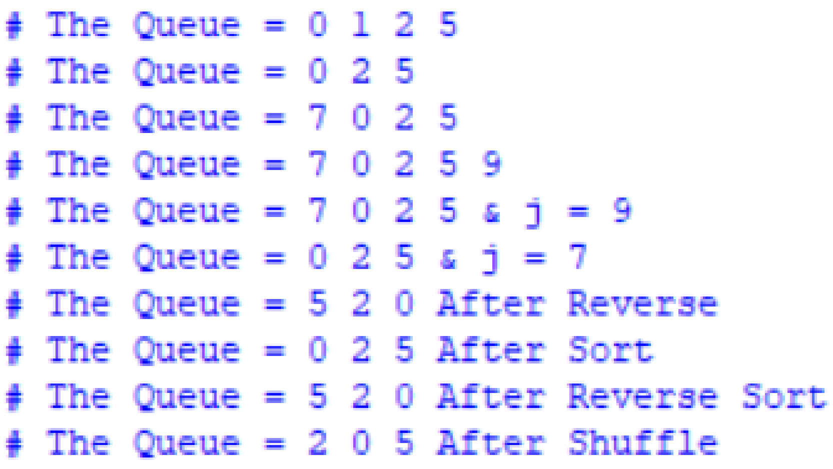
QUEUE

2) Write a module to test queue data type and its predefined methods. Run Questasim to make sure the display statements are working as expected.

- Declare int j and a queue q of type int
- initialize int j as 1 and queue q as (0, 2, 5)
- insert int j at index 1 in queue q and display q
- delete index 1 element from queue q and display q
- push an element (7) in the front in queue q and display q
- push an element (9) at the back in queue q and display q
- pop an element from back of queue q into j, display q, and j
- pop an element from front of queue q into j, display q, and j
- reverse, sort, reverse sort and shuffle the queue and display q after using each method

Queue Testbench

```
module queue;
int j = 1;
int q[$] = {0,2,5};
initial begin
  q.insert(1,j);
  $display("The Queue = %0p",q); // q = (0,1,2,5)
  q.delete(1);
  $display("The Queue = %0p",q); // q = (0,2,5)
  q.push_front(7);
  $display("The Queue = %0p",q); // q = (7,0,2,5)
  q.push_back(9);
  $display("The Queue = %0p",q); // q = (7,0,2,5,9)
  j = q.pop_back();
  $display("The Queue = %0p & j = %0d",q,j); // q = (7,0,2,5) & j = 9
  j = q.pop_front();
  $display("The Queue = %0p & j = %0d",q,j); // q = (0,2,5) & j = 7
  q.reverse();
  $display("The Queue = %0p After Reverse",q); // q = (5,2,0)
  q.sort();
  $display("The Queue = %0p After Sort",q); // q = (0,2,5)
  q.rsort();
  $display("The Queue = %0p After Reverse Sort",q); // q = (5,2,0)
  q.shuffle();
  $display("The Queue = %0p After Shuffle",q); // q = Random like (2,0,5)
  $stop;
end
endmodule
```



```
# The Queue = 0 1 2 5
# The Queue = 0 2 5
# The Queue = 7 0 2 5
# The Queue = 7 0 2 5 9
# The Queue = 7 0 2 5 & j = 9
# The Queue = 0 2 5 & j = 7
# The Queue = 5 2 0 After Reverse
# The Queue = 0 2 5 After Sort
# The Queue = 5 2 0 After Reverse Sort
# The Queue = 2 0 5 After Shuffle
```

Question 3

RAM

3) The design to be tested is a synchronous single-port 8-bit x64K (512kBit) RAM. The RAM will read on the positive edge of the clock when input read =1 and write on the positive edge of the clock when input write = 1. Write enable signal has a higher priority than the read enable signal and both write and read data from the RAM is not allowed at the same time. Even parity will be calculated on data written to the RAM and placed in the 9th bit of the memory. The partially completed memory model is below (add the memory declaration)

```
module my_mem(
                    input clk,
                    input write,
                    input read,
                    input [7:0] data_in,
                    input [15:0] address,
                    output reg [7:0] data_out
);

  // Declare a 9-bit associative array using the logic data type & the key of int datatype
  <Put your declaration here>

  always @(posedge clk) begin
    if (write)
      mem_array[address] = {~^data_in, data_in};
    else if (read)
      data_out =  mem_array[address];
  end

endmodule
```

Your testbench will:

1. Perform 100 writes of random data to random addresses followed by 100 reads to the same addresses in reverse order. To do this:
   a. Use a local parameter TESTS with the value 100 to define the size of the following dynamic arrays
   b. Create a dynamic array of addresses called *address_array*
   c. Create a dynamic array of data in for the RAM called *data_to_write_array*
   d. Create an associative array of data expected to be read indexed by the address read called *data_read_expect_assoc*
   e. Create a queue of data read called *data_read_queue*
   f. Create a task called stimulus_gen that will prepare the data to be written as well as the address using a for loop for the dynamic arrays.

   g. Create a task called "golden_model" that will fill the associative array by looping and filling the associative array with the expected values.
   h. Drive the DUT with data to be written
   i. Drive the data to read out the data, make your test-bench self-checking by calling a checking task that will compare the data read with the *data_read_expect_assoc* array. Maintain an error counter. Store the data read in the queue *data_read_queue.*
2. At the end of the test, print out the data read stored in the *data_read_queue* using while loop and pop_front method (search online how to do it). Also, print out the error and correct counter.

Ram RTL Design

System Verilog Design Code:

```
module RAM(
   input clk,
   input write,
   input read,
   input [7: 0] data_in,
   // First Bug is that Address must be 19 Bits Not 16 Bits only to cover all memory locations "512 Kbit"
   input [18: 0] address,
   // Add extra bit for parity
   output reg [8: 0] data_out
);

// RAM memory declaration as associative array
logic [8: 0] mem_array [bit [18: 0]] ;

always @(posedge clk) begin
   if (write)
      // Second Bug is that Even Parity is the XOR(^) of data not XNOR(~^)
      mem_array[address]  =  {^data_in, data_in};
   else if (read)
      data_out  =  mem_array[address];
end

endmodule
```

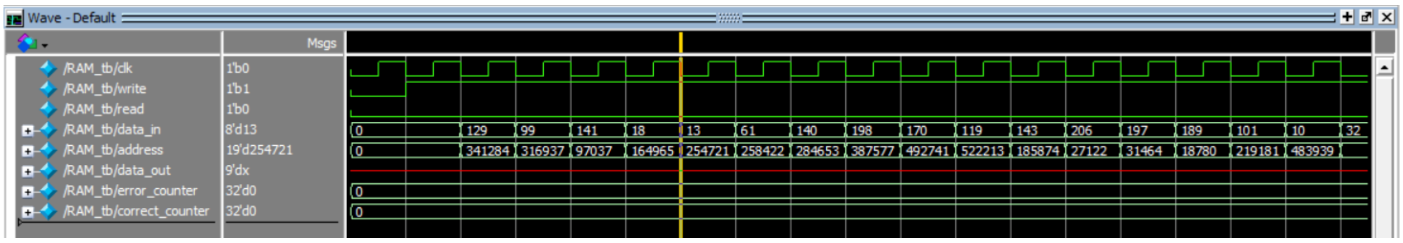
Do File for (RAM) Question 3

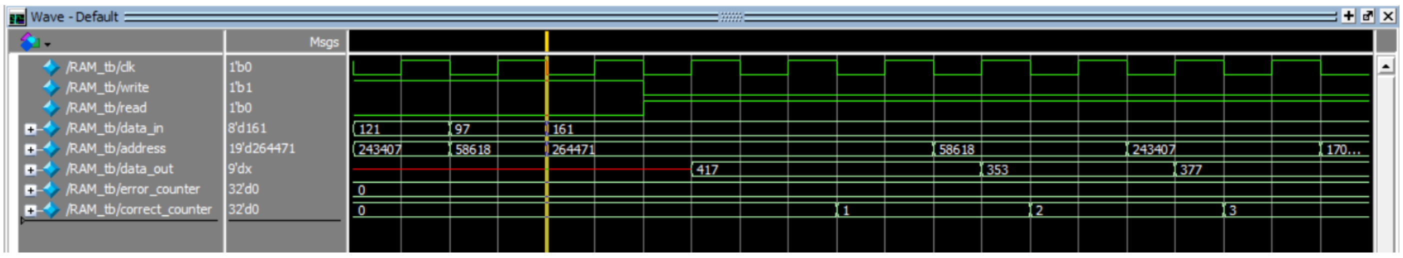
```
vlib work
vlog RAM. sv RAM_tb. sv  + cover  – covercells
vsim  – voptargs = +acc work. RAM_tb  – cover
add wave  *
coverage save RAM_tb. ucdb  – onexit  – du work. RAM
run  – all
# quit  – sim
# vcover report RAM_tb. ucdb  – details  – annotate  – all  – output RAM_Coverage_Rpt. txt
```

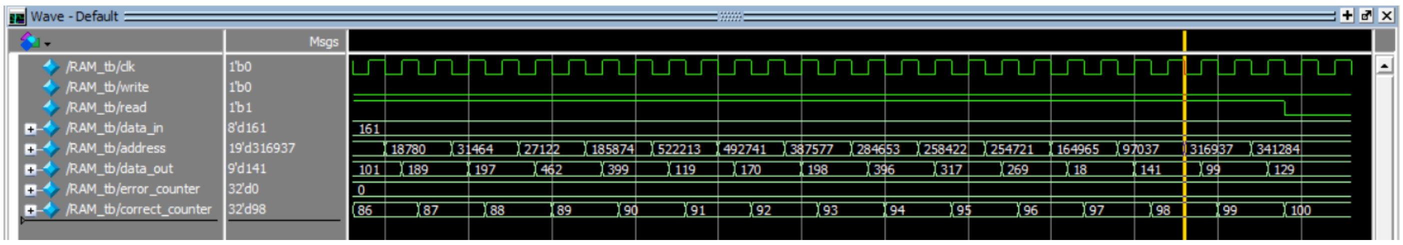
Ram Testbench

```
module RAM_tb;
// Input and Output Declaration
logic clk, write, read;  // Input
logic [7: 0] data_in;     // Input
logic [18: 0] address;    // Input
logic [8: 0] data_out;    // Output
// Size of the Two Dynamic Arrays
localparam TESTS = 100;
// Error & Correct Counters
integer error_counter, correct_counter;
// Dynamic Arrays Declarations
bit [18: 0] address_array[]; // To Store Address
bit [7: 0] data_to_write_array[]; // Store the Data That will be Written in RAM
// Associative Array Declaration
bit [8: 0] data_read_expect_assoc[bit [18: 0]]; // Store The Expected Read Data indexed by Read Address
// Queue Declaration
bit [8: 0] data_read_queue[$]; // Store Data Read From the RAM
// Design Module Instantiation
RAM DUT(.*);
// Clock Generation
initial begin
  clk = 0;
  forever
    #1 clk = ~clk; // CLK Period = 2ns
end
// Initial Values for Inputs & Main Block Operations
initial begin
  address_array = new[TESTS];
  data_to_write_array = new[TESTS];
  // Initialize all Counters & Inputs to ZERO
  error_counter = 0;
  correct_counter = 0;
  read = 0;
  write = 0;
  address = 0;
  data_in = 0;
  // Delay 1 Clock Cycles for Toggling from (0 -> 1)
  repeat(1) @(negedge clk);
  stimulus_gen();
  golden_model();
  // 100 Write Operations
  write = 1;
  for (int i = 0 ; i < TESTS ; i++) begin
    @(negedge clk);
    address = address_array[i];
    data_in = data_to_write_array[i];
  end
  // Delay 1 Clock Cycles to Allow final Data to be Written in RAM
  repeat(1) @(negedge clk);
  // 100 Read Operations
  write = 0;
  read = 1;
  address_array.reverse();
  for (int i = 0 ; i < TESTS ; i ++) begin
    @(negedge clk);
    address = address_array[i];
    check_result(address);
    data_read_queue.insert(i, data_out);
  end
  $display("Data Read From The Queue");
  while (data_read_queue.size())
    $display("At %0t ns: Data Read From Queue %0d", $time, data_read_queue.pop_front());
  read = 0; // To Achieve Toggling for Read Signal From 1 -> 0
  repeat(2) @(negedge clk);
  $display("%0t: At End of test error counter = %0d and
                                    correct counter = %0d", $time, error_counter, correct_counter);
  $stop;
end

task stimulus_gen();
  for (int i = 0 ; i < TESTS ; i ++) begin
    address_array[i] = $random;
    data_to_write_array[i] = $random;
  end
endtask

task golden_model();
  for (int i = 0 ; i < TESTS ; i ++)
    data_read_expect_assoc[address_array[i]]
              = {^data_to_write_array[i], data_to_write_array[i]};
endtask

task check_result(input bit [18:0] Expected_Address);
  @(negedge clk);
  if (data_read_expect_assoc.exists(Expected_Address)) begin
    if (data_out !== data_read_expect_assoc[Expected_Address]) begin
      $display("%0t: Error: For Read Data %0h should be equal to %0h but it is not equal",
                  $time, data_out, data_read_expect_assoc[address_array[Expected_Address]]);
      error_counter ++;
    end
    else
      correct_counter ++;
  end
endtask
endmodule
```

Waveform





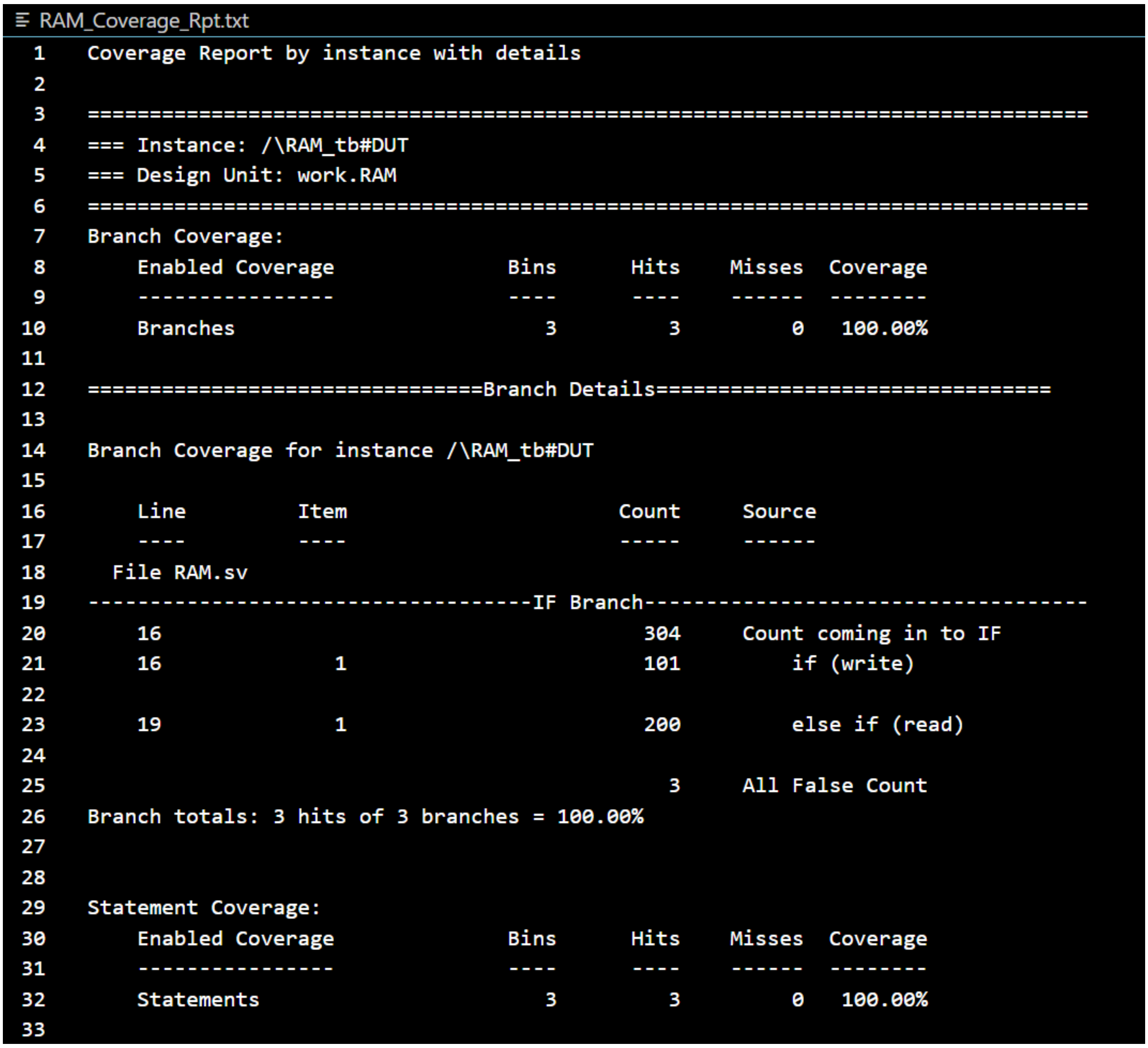


```
# Data Read From The Queue
# At 604 ns: Data Read From Queue 417
# At 604 ns: Data Read From Queue 353
# At 604 ns: Data Read From Queue 377
# At 604 ns: Data Read From Queue 219
# At 604 ns: Data Read From Queue 120
# At 604 ns: Data Read From Queue 476
# At 604 ns: Data Read From Queue 438
# At 604 ns: Data Read From Queue 192
# At 604 ns: Data Read From Queue 311
# At 604 ns: Data Read From Queue 282
# At 604 ns: Data Read From Queue 250
# At 604 ns: Data Read From Queue 442
# At 604 ns: Data Read From Queue 368
# At 604 ns: Data Read From Queue 317
# At 604 ns: Data Read From Queue 390
# At 604 ns: Data Read From Queue 297
# At 604 ns: Data Read From Queue 284
# At 604 ns: Data Read From Queue 46
# At 604 ns: Data Read From Queue 45
# At 604 ns: Data Read From Queue 136
# At 604 ns: Data Read From Queue 105
# At 604 ns: Data Read From Queue 260
# At 604 ns: Data Read From Queue 149
# At 604 ns: Data Read From Queue 202
# At 604 ns: Data Read From Queue 365
# At 604 ns: Data Read From Queue 77
# At 604 ns: Data Read From Queue 89
# At 604 ns: Data Read From Queue 147
# At 604 ns: Data Read From Queue 184
# At 604 ns: Data Read From Queue 312
# At 604 ns: Data Read From Queue 10
# At 604 ns: Data Read From Queue 78
# At 604 ns: Data Read From Queue 195
# At 604 ns: Data Read From Queue 154
```

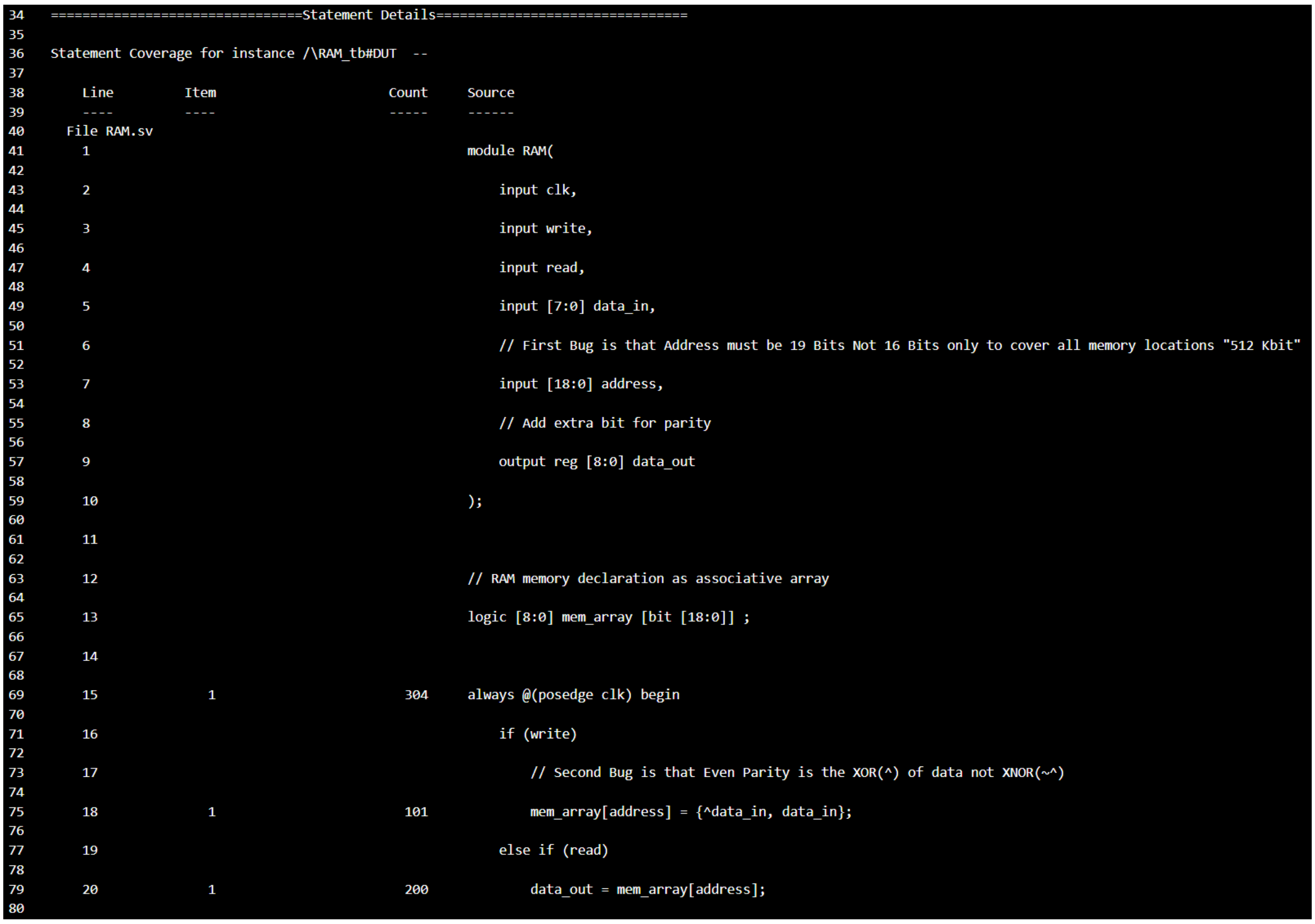
```
# At 604 ns: Data Read From Queue 270
# At 604 ns: Data Read From Queue 298
# At 604 ns: Data Read From Queue 68
# At 604 ns: Data Read From Queue 479
# At 604 ns: Data Read From Queue 101
# At 604 ns: Data Read From Queue 237
# At 604 ns: Data Read From Queue 90
# At 604 ns: Data Read From Queue 459
# At 604 ns: Data Read From Queue 68
# At 604 ns: Data Read From Queue 435
# At 604 ns: Data Read From Queue 163
# At 604 ns: Data Read From Queue 294
# At 604 ns: Data Read From Queue 156
# At 604 ns: Data Read From Queue 390
# At 604 ns: Data Read From Queue 298
# At 604 ns: Data Read From Queue 329
# At 604 ns: Data Read From Queue 120
# At 604 ns: Data Read From Queue 467
# At 604 ns: Data Read From Queue 57
# At 604 ns: Data Read From Queue 126
# At 604 ns: Data Read From Queue 317
# At 604 ns: Data Read From Queue 456
# At 604 ns: Data Read From Queue 12
# At 604 ns: Data Read From Queue 337
# At 604 ns: Data Read From Queue 464
# At 604 ns: Data Read From Queue 393
# At 604 ns: Data Read From Queue 92
# At 604 ns: Data Read From Queue 183
# At 604 ns: Data Read From Queue 399
# At 604 ns: Data Read From Queue 332
# At 604 ns: Data Read From Queue 473
# At 604 ns: Data Read From Queue 277
# At 604 ns: Data Read From Queue 58
# At 604 ns: Data Read From Queue 335
# At 604 ns: Data Read From Queue 113
# At 604 ns: Data Read From Queue 298
# At 604 ns: Data Read From Queue 430
# At 604 ns: Data Read From Queue 438
# At 604 ns: Data Read From Queue 393
# At 604 ns: Data Read From Queue 216
# At 604 ns: Data Read From Queue 394
# At 604 ns: Data Read From Queue 60
# At 604 ns: Data Read From Queue 10
# At 604 ns: Data Read From Queue 207
# At 604 ns: Data Read From Queue 430
# At 604 ns: Data Read From Queue 469
```

```
# At 604 ns: Data Read From Queue 83
# At 604 ns: Data Read From Queue 275
# At 604 ns: Data Read From Queue 413
# At 604 ns: Data Read From Queue 288
# At 604 ns: Data Read From Queue 10
# At 604 ns: Data Read From Queue 101
# At 604 ns: Data Read From Queue 189
# At 604 ns: Data Read From Queue 197
# At 604 ns: Data Read From Queue 462
# At 604 ns: Data Read From Queue 399
# At 604 ns: Data Read From Queue 119
# At 604 ns: Data Read From Queue 170
# At 604 ns: Data Read From Queue 198
# At 604 ns: Data Read From Queue 396
# At 604 ns: Data Read From Queue 317
# At 604 ns: Data Read From Queue 269
# At 604 ns: Data Read From Queue 18
# At 604 ns: Data Read From Queue 141
# At 604 ns: Data Read From Queue 99
# At 604 ns: Data Read From Queue 129
# 608: At End of test error counter = 0 and correct counter = 100
```

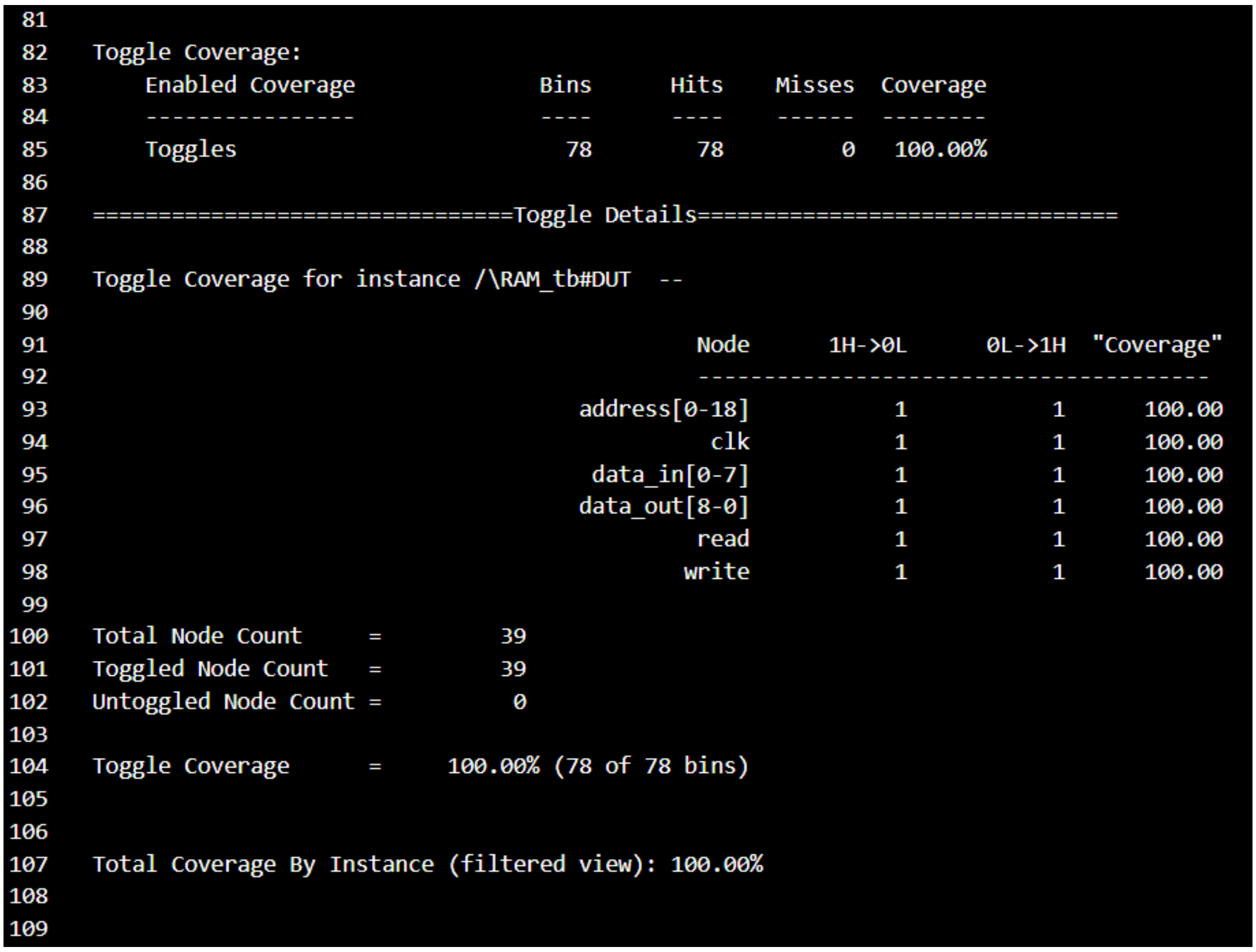
Code Coverage


RAM_Coverage_Rpt.txt

```
Coverage Report by instance with details

=================================================================================
=== Instance: /\RAM_tb#DUT
=== Design Unit: work.RAM
=================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         3         3         0   100.00%

================================Branch Details================================

Branch Coverage for instance /\RAM_tb#DUT

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File RAM.sv
------------------------------------IF Branch------------------------------------
    16                                       304     Count coming in to IF
    16              1                        101         if (write)

    19              1                        200         else if (read)

                                               3     All False Count
Branch totals: 3 hits of 3 branches = 100.00%


Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       3         3         0   100.00%

```



```
==============================Statement Details==============================

Statement Coverage for instance /\RAM_tb#DUT  --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File RAM.sv
    1                                                module RAM(

    2                                                    input clk,

    3                                                    input write,

    4                                                    input read,

    5                                                    input [7:0] data_in,

    6                                                    // First Bug is that Address must be 19 Bits Not 16 Bits only to cover all memory locations "512 Kbit"

    7                                                    input [18:0] address,

    8                                                    // Add extra bit for parity

    9                                                    output reg [8:0] data_out

    10                                               );

    11

    12                                               // RAM memory declaration as associative array

    13                                               logic [8:0] mem_array [bit [18:0]] ;

    14

    15              1                        304     always @(posedge clk) begin

    16                                                   if (write)

    17                                                       // Second Bug is that Even Parity is the XOR(^) of data not XNOR(~^)

    18              1                        101             mem_array[address] = {^data_in, data_in};

    19                                                   else if (read)

    20              1                        200             data_out = mem_array[address];

```



```

Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         78        78         0   100.00%

================================Toggle Details================================

Toggle Coverage for instance /\RAM_tb#DUT  --

                                              Node      1H->0L      0L->1H  "Coverage"
                                              ---------------------------------------
                                     address[0-18]           1           1      100.00
                                               clk           1           1      100.00
                                      data_in[0-7]           1           1      100.00
                                     data_out[8-0]           1           1      100.00
                                              read           1           1      100.00
                                             write           1           1      100.00

Total Node Count     =         39
Toggled Node Count   =         39
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (78 of 78 bins)


Total Coverage By Instance (filtered view): 100.00%


```

Question 4

ALU

4) We will test the same ALU given in assignment 1 but using randomization and typedef enum.

Follow the following steps:

- Create typedef enum for the opcode at the top of the file before the class or the module
- Create a class to randomize all ALU inputs
  - Constraint the reset to be low most of the time
  - Use the typedef enum for the opcode variable
- Create the testbench module
  - Use the typedef enum for the opcode variable
  - Your testbench will use constrained randomization to drive the stimulus
  - Make the testbench self-checking
  - Monitor the output to display errors if occurred
  - Use a do file to run the testbench
  - Generate a code coverage report (100% design code coverage is expected. Less than 100% must be justified.)

ALU RTL Design

Verilog Design Code:

```
module ALU_4_bit (
   input  clk,
   input  reset,
   input  [1: 0] Opcode,   // The opcode
   input  signed [3: 0] A,// Input data A in 2′s complement
   input  signed [3: 0] B,// Input data B in 2′s complement

   output reg signed [4: 0] C // ALU output in 2′s complement
  );

  reg signed [4: 0] Alu_out; // ALU output in 2′s complement

  localparam Add  =  2′b00;         // A  +  B
  localparam Sub  =  2′b01;         // A  −  B
  localparam Not_A  =  2′b10;        // ~A
  localparam ReductionOR_B  =  2′b11; // |B

  // Do the operation
  always @(∗) begin
    case (Opcode)
      Add: Alu_out  =  A  +  B;
      Sub: Alu_out  =  A  −  B;
      Not_A: Alu_out  =  ~A;
      ReductionOR_B: Alu_out  =  |B;
     default: Alu_out  =  5′b0;
    endcase
  end // always @ (∗)

  // Register output C
  always @(posedge clk or posedge reset) begin
    if (reset)
      C  <=  5′b0;
    else
      C  <=  Alu_out;
  end

endmodule
```

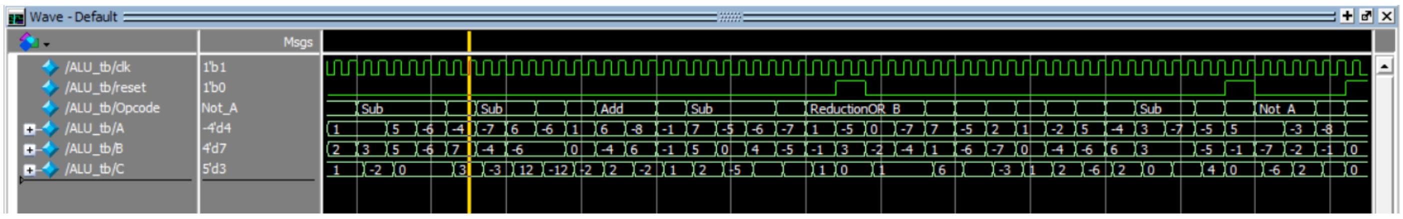
ALU Testbench

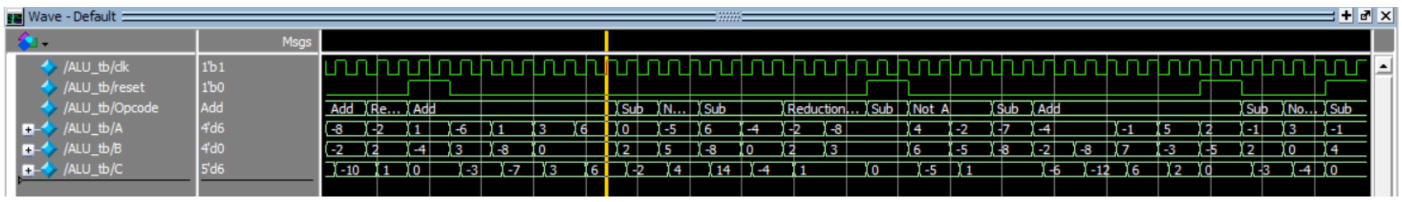
```
typedef enum logic [1: 0] {Add , Sub , Not_A , ReductionOR_B} opcode_e;
class ALU;
rand bit signed [3: 0] a , b;
rand opcode_e opcode;
rand bit rst;
constraint c_rst     {rst dist{0: = 90, 1: = 10}; }
constraint c_input { a dist{[−8: −1]:/50, [0: 7]:/50};
                     b dist{[−8: −1]:/50, [0: 7]:/50};  }
constraint c_opcode{opcode dist {Add ≔ 50, Sub ≔ 50, Not_A ≔ 30,
                                  ReductionOR_B: = 30 }; }
endclass
module ALU_tb ;
// input and output declaration
logic clk, reset;
opcode_e Opcode;
logic signed [3: 0] A , B;
logic signed [4: 0] C;
// creation of object from class
ALU alu_class;
//Module Instantiation
ALU_4_bit DUT(.*);

// clock declaration
initial begin
  clk  =  0;
forever
  #1 clk  =  ~clk;      // clock period  =  2 ns
end

initial begin
  // Check the Default Operation Where there is No Opcode & enum of time Logic ′x default′
  A  =  $random;
  B  =  $random;
  result_checking();
  alu_class  =  new();
  for(int i = 0; i < 100; i + +) begin
    @(negedge clk);
    assert(alu_class. randomize());
    A  =  alu_class. a;
    B  =  alu_class. b;
    Opcode  =  alu_class. opcode;
    reset  =  alu_class. rst;
    result_checking();
    $display("%0t: For Inputs A = %0d, B = %0d, Opcode = %0b,
                               rst = %0b, Output C = %0d", $time, A, B, Opcode, reset, C);
  end
  $stop;
end

task result_checking();
  logic signed [4:0] C_expected;
  @(negedge clk);
  if(reset)
    C_expected = 0 ;
  else begin
    case(Opcode)
      Add: C_expected = A + B;
      Sub: C_expected = A - B;
      Not_A: C_expected = ~A;
      ReductionOR_B: C_expected = |B;
      default : C_expected = 0 ;
    endcase
  end
  if(C !== C_expected) begin
    $display("Wrong ALU Design! ");
    $stop;
  end
endtask

endmodule
```

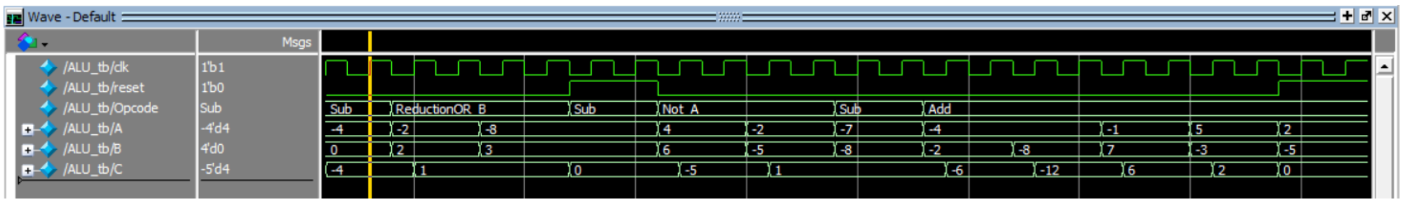
Do File for (ALU) Question 4

```
vlib work
vlog alu.v ALU_tb.sv +cover -covercells
vsim -voptargs=+acc work.ALU_tb -cover
add wave *
coverage save ALU_tb.ucdb -onexit -du work.ALU_4_bit
run -all
# quit -sim
# vcover report ALU_tb.ucdb -details -annotate -all -output ALU_Coverage_Rpt.txtvlib work
```

WAVEFORM



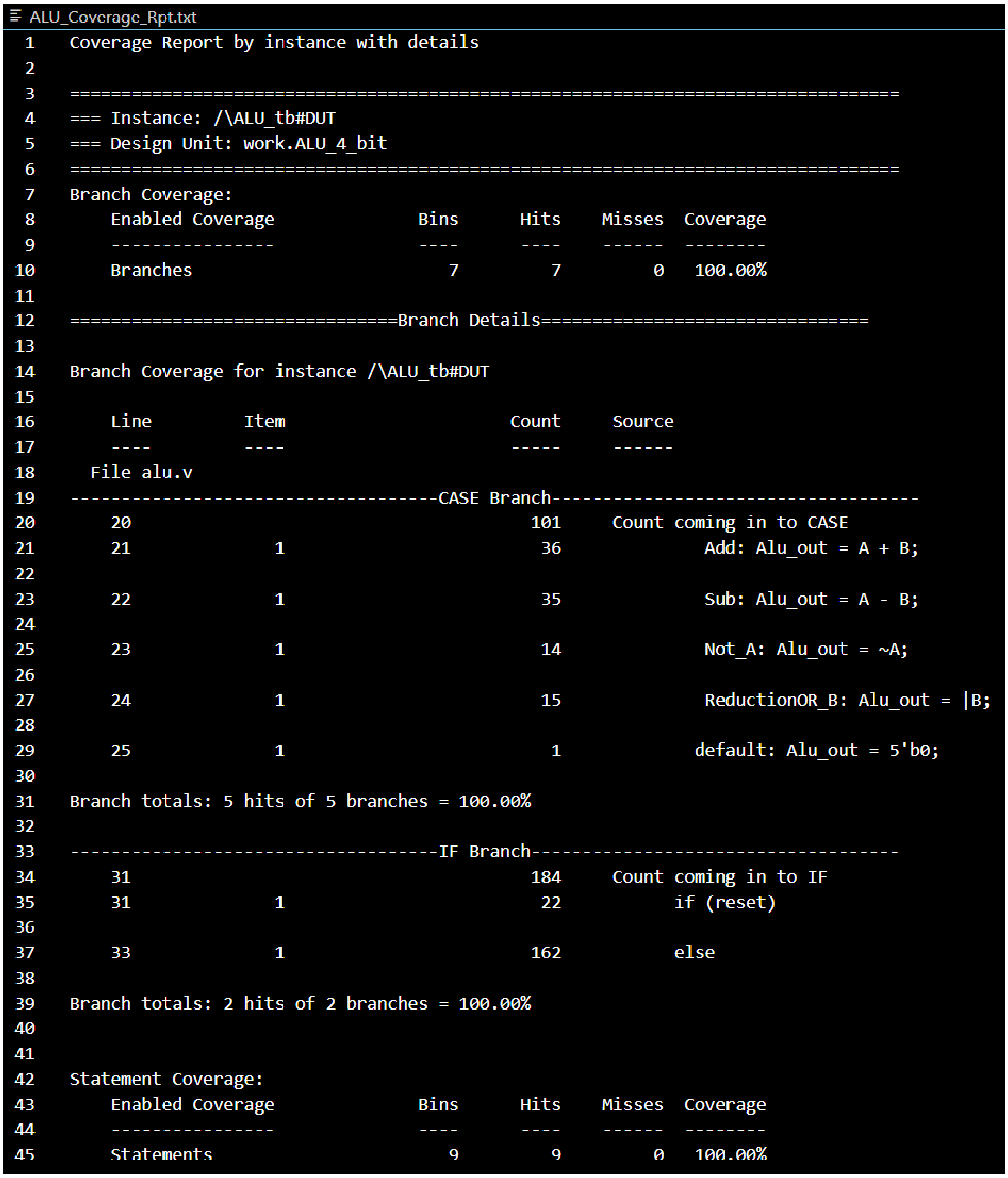




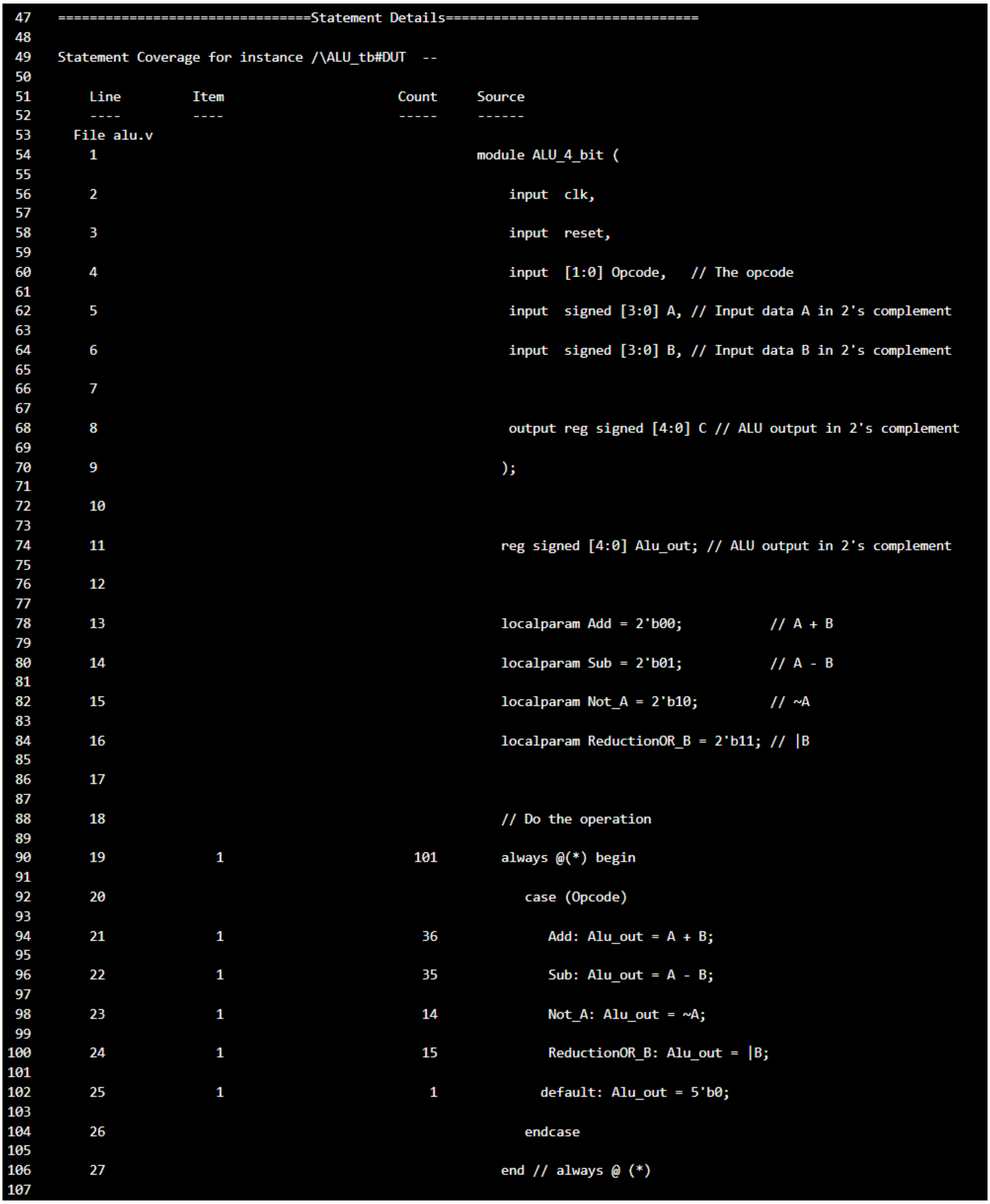
```
# 6: For Inputs A = 0, B = 0, Opcode = 11, rst = 0, Output C = 0
# 10: For Inputs A = -3, B = -6, Opcode = 1, rst = 0, Output C = 3
# 14: For Inputs A = -7, B = 6, Opcode = 11, rst = 0, Output C = 1
# 18: For Inputs A = 1, B = 2, Opcode = 11, rst = 0, Output C = 1
# 22: For Inputs A = 1, B = 3, Opcode = 1, rst = 0, Output C = -2
# 26: For Inputs A = 5, B = 5, Opcode = 1, rst = 0, Output C = 0
# 30: For Inputs A = -6, B = -6, Opcode = 1, rst = 0, Output C = 0
# 34: For Inputs A = -4, B = 7, Opcode = 10, rst = 0, Output C = 3
# 38: For Inputs A = -7, B = -4, Opcode = 1, rst = 0, Output C = -3
# 42: For Inputs A = 6, B = -6, Opcode = 1, rst = 0, Output C = 12
# 46: For Inputs A = -6, B = -6, Opcode = 0, rst = 0, Output C = -12
# 50: For Inputs A = 1, B = 0, Opcode = 10, rst = 0, Output C = -2
# 54: For Inputs A = 6, B = -4, Opcode = 0, rst = 0, Output C = 2
# 58: For Inputs A = -8, B = 6, Opcode = 0, rst = 0, Output C = -2
# 62: For Inputs A = -1, B = -1, Opcode = 11, rst = 0, Output C = 1
# 66: For Inputs A = 7, B = 5, Opcode = 1, rst = 0, Output C = 2
# 70: For Inputs A = -5, B = 0, Opcode = 1, rst = 0, Output C = -5
# 74: For Inputs A = -6, B = 4, Opcode = 1, rst = 0, Output C = -10
# 78: For Inputs A = -7, B = -5, Opcode = 0, rst = 0, Output C = -12
# 82: For Inputs A = 1, B = -1, Opcode = 11, rst = 0, Output C = 1
# 86: For Inputs A = -5, B = 3, Opcode = 11, rst = 1, Output C = 0
# 90: For Inputs A = 0, B = -2, Opcode = 11, rst = 0, Output C = 1
# 94: For Inputs A = -7, B = -4, Opcode = 11, rst = 0, Output C = 1
# 98: For Inputs A = 7, B = 1, Opcode = 1, rst = 0, Output C = 6
# 102: For Inputs A = -5, B = -6, Opcode = 0, rst = 0, Output C = -11
# 106: For Inputs A = 2, B = -7, Opcode = 10, rst = 0, Output C = -3
# 110: For Inputs A = 1, B = 0, Opcode = 0, rst = 0, Output C = 1
# 114: For Inputs A = -2, B = -4, Opcode = 1, rst = 0, Output C = 2
# 118: For Inputs A = 5, B = -6, Opcode = 10, rst = 0, Output C = -6
# 122: For Inputs A = -4, B = 6, Opcode = 0, rst = 0, Output C = 2
# 126: For Inputs A = 3, B = 3, Opcode = 1, rst = 0, Output C = 0
# 130: For Inputs A = -7, B = 3, Opcode = 1, rst = 0, Output C = -10
# 134: For Inputs A = -5, B = -5, Opcode = 10, rst = 0, Output C = 4
# 138: For Inputs A = 5, B = -1, Opcode = 0, rst = 1, Output C = 0
# 142: For Inputs A = 5, B = -7, Opcode = 10, rst = 0, Output C = -6
# 146: For Inputs A = -3, B = -2, Opcode = 10, rst = 0, Output C = 2
# 150: For Inputs A = -8, B = -1, Opcode = 1, rst = 0, Output C = -7
# 154: For Inputs A = -2, B = 0, Opcode = 10, rst = 1, Output C = 0
# 158: For Inputs A = 6, B = 0, Opcode = 1, rst = 0, Output C = 6
# 162: For Inputs A = -1, B = -8, Opcode = 0, rst = 0, Output C = -9
# 166: For Inputs A = -2, B = -5, Opcode = 1, rst = 0, Output C = 3
# 170: For Inputs A = 2, B = -7, Opcode = 0, rst = 0, Output C = -5
# 174: For Inputs A = 1, B = -2, Opcode = 11, rst = 0, Output C = 1
# 178: For Inputs A = 2, B = 4, Opcode = 1, rst = 0, Output C = -2
# 182: For Inputs A = 4, B = -4, Opcode = 0, rst = 0, Output C = 0
# 186: For Inputs A = -4, B = -5, Opcode = 0, rst = 0, Output C = -9
# 190: For Inputs A = 5, B = -5, Opcode = 1, rst = 0, Output C = 10
```

```
# 194: For Inputs A = -1, B = -3, Opcode = 1, rst = 0, Output C = 2
# 198: For Inputs A = -3, B = 5, Opcode = 0, rst = 0, Output C = 2
# 202: For Inputs A = -4, B = -8, Opcode = 1, rst = 0, Output C = 4
# 206: For Inputs A = 1, B = -2, Opcode = 0, rst = 0, Output C = -1
# 210: For Inputs A = -1, B = -6, Opcode = 1, rst = 0, Output C = 5
# 214: For Inputs A = 1, B = -4, Opcode = 1, rst = 0, Output C = 5
# 218: For Inputs A = -6, B = 4, Opcode = 0, rst = 1, Output C = 0
# 222: For Inputs A = 0, B = -6, Opcode = 11, rst = 0, Output C = 1
# 226: For Inputs A = 3, B = -2, Opcode = 0, rst = 0, Output C = 1
# 230: For Inputs A = -6, B = 0, Opcode = 0, rst = 0, Output C = -6
# 234: For Inputs A = -4, B = -7, Opcode = 11, rst = 0, Output C = 1
# 238: For Inputs A = 1, B = 1, Opcode = 0, rst = 1, Output C = 0
# 242: For Inputs A = -6, B = 5, Opcode = 1, rst = 0, Output C = -11
# 246: For Inputs A = -3, B = -5, Opcode = 10, rst = 0, Output C = 2
# 250: For Inputs A = -7, B = 2, Opcode = 0, rst = 1, Output C = 0
# 254: For Inputs A = 6, B = -2, Opcode = 10, rst = 0, Output C = -7
# 258: For Inputs A = 5, B = 5, Opcode = 0, rst = 0, Output C = 10
# 262: For Inputs A = 7, B = -2, Opcode = 1, rst = 0, Output C = 9
# 266: For Inputs A = -8, B = 6, Opcode = 0, rst = 0, Output C = -2
# 270: For Inputs A = 4, B = -1, Opcode = 0, rst = 0, Output C = 3
# 274: For Inputs A = -6, B = 1, Opcode = 1, rst = 0, Output C = -7
# 278: For Inputs A = -7, B = 0, Opcode = 11, rst = 0, Output C = 0
# 282: For Inputs A = -3, B = -6, Opcode = 0, rst = 1, Output C = 0
# 286: For Inputs A = 4, B = -3, Opcode = 1, rst = 0, Output C = 7
# 290: For Inputs A = 5, B = -4, Opcode = 1, rst = 0, Output C = 9
# 294: For Inputs A = 3, B = 3, Opcode = 1, rst = 0, Output C = 0
# 298: For Inputs A = 5, B = -7, Opcode = 0, rst = 0, Output C = -2
# 302: For Inputs A = -8, B = -2, Opcode = 0, rst = 0, Output C = -10
# 306: For Inputs A = -2, B = 2, Opcode = 11, rst = 0, Output C = 1
# 310: For Inputs A = 1, B = -4, Opcode = 0, rst = 1, Output C = 0
# 314: For Inputs A = -6, B = 3, Opcode = 0, rst = 0, Output C = -3
# 318: For Inputs A = 1, B = -8, Opcode = 0, rst = 0, Output C = -7
# 322: For Inputs A = 3, B = 0, Opcode = 0, rst = 0, Output C = 3
# 326: For Inputs A = 6, B = 0, Opcode = 0, rst = 0, Output C = 6
# 330: For Inputs A = 0, B = 2, Opcode = 1, rst = 0, Output C = -2
# 334: For Inputs A = -5, B = 5, Opcode = 10, rst = 0, Output C = 4
# 338: For Inputs A = 6, B = -8, Opcode = 1, rst = 0, Output C = 14
# 342: For Inputs A = -4, B = 0, Opcode = 1, rst = 0, Output C = -4
# 346: For Inputs A = -2, B = 2, Opcode = 11, rst = 0, Output C = 1
# 350: For Inputs A = -8, B = 3, Opcode = 11, rst = 0, Output C = 1
# 354: For Inputs A = -8, B = 3, Opcode = 1, rst = 1, Output C = 0
# 358: For Inputs A = 4, B = 6, Opcode = 10, rst = 0, Output C = -5
# 362: For Inputs A = -2, B = -5, Opcode = 10, rst = 0, Output C = 1
# 366: For Inputs A = -7, B = -8, Opcode = 1, rst = 0, Output C = 1
# 370: For Inputs A = -4, B = -2, Opcode = 0, rst = 0, Output C = -6
# 374: For Inputs A = -4, B = -8, Opcode = 0, rst = 0, Output C = -12
# 378: For Inputs A = -1, B = 7, Opcode = 0, rst = 0, Output C = 6
# 382: For Inputs A = 5, B = -3, Opcode = 0, rst = 0, Output C = 2
# 386: For Inputs A = 2, B = -5, Opcode = 0, rst = 1, Output C = 0
# 390: For Inputs A = -1, B = 2, Opcode = 1, rst = 0, Output C = -3
# 394: For Inputs A = 3, B = 0, Opcode = 10, rst = 0, Output C = -4
# 398: For Inputs A = -1, B = 4, Opcode = 1, rst = 1, Output C = 0
# 402: For Inputs A = -2, B = -4, Opcode = 0, rst = 0, Output C = -6
```

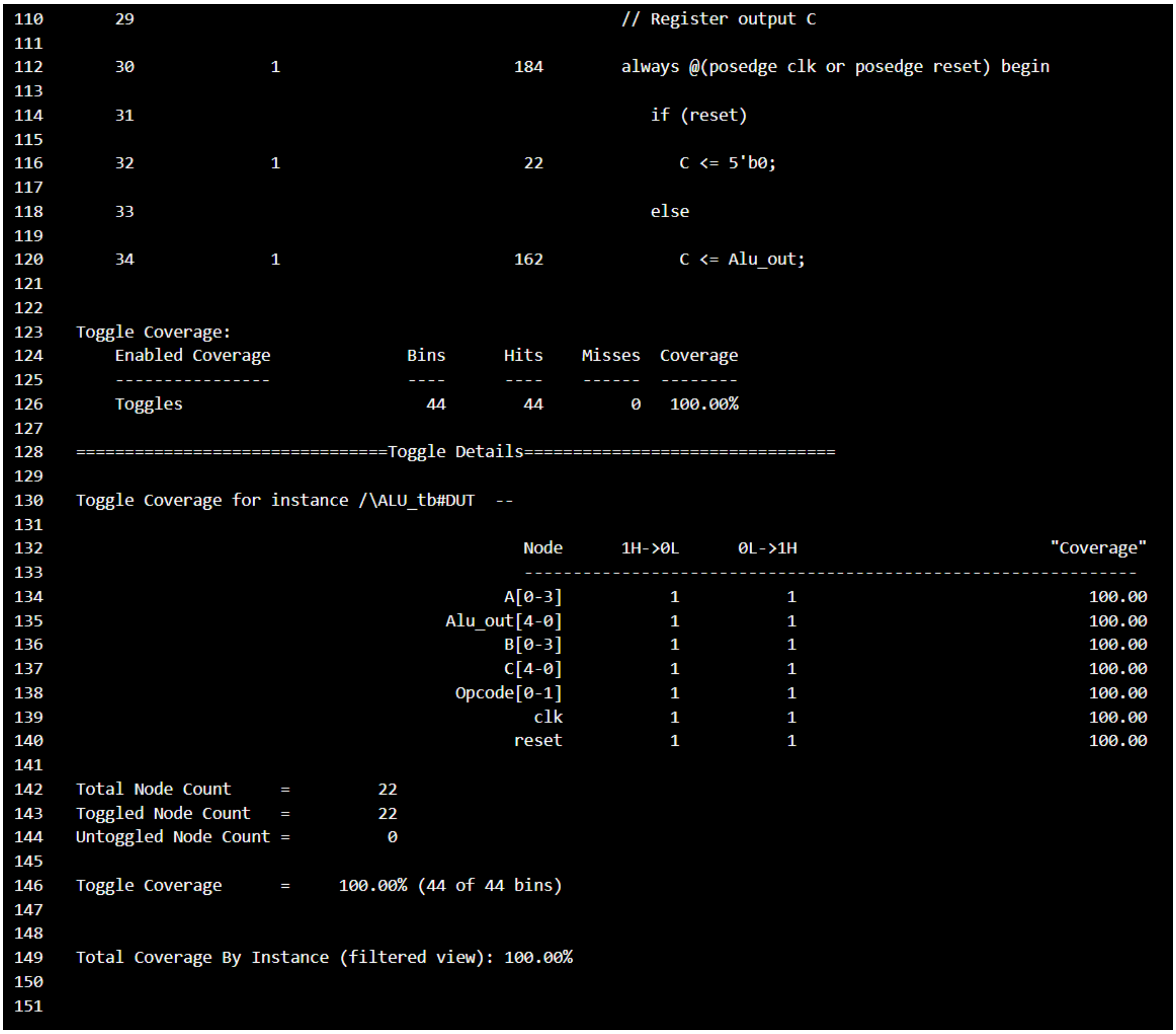
Code Coverage



```
Coverage Report by instance with details

=================================================================================
=== Instance: /\ALU_tb#DUT
=== Design Unit: work.ALU_4_bit
=================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         7         7         0   100.00%

================================Branch Details================================

Branch Coverage for instance /\ALU_tb#DUT

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File alu.v
------------------------------------CASE Branch------------------------------------
    20                                       101     Count coming in to CASE
    21              1                         36              Add: Alu_out = A + B;

    22              1                         35              Sub: Alu_out = A - B;

    23              1                         14              Not_A: Alu_out = ~A;

    24              1                         15              ReductionOR_B: Alu_out = |B;

    25              1                          1             default: Alu_out = 5'b0;

Branch totals: 5 hits of 5 branches = 100.00%

------------------------------------IF Branch------------------------------------
    31                                       184     Count coming in to IF
    31              1                         22           if (reset)

    33              1                        162           else

Branch totals: 2 hits of 2 branches = 100.00%


Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       9         9         0   100.00%
```



```
================================Statement Details================================

Statement Coverage for instance /\ALU_tb#DUT  --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File alu.v
    1                                                module ALU_4_bit (

    2                                                    input  clk,

    3                                                    input  reset,

    4                                                    input  [1:0] Opcode,   // The opcode

    5                                                    input  signed [3:0] A, // Input data A in 2's complement

    6                                                    input  signed [3:0] B, // Input data B in 2's complement

    7

    8                                                    output reg signed [4:0] C // ALU output in 2's complement

    9                                                   );

    10

    11                                                  reg signed [4:0] Alu_out; // ALU output in 2's complement

    12

    13                                                  localparam Add = 2'b00;           // A + B

    14                                                  localparam Sub = 2'b01;           // A - B

    15                                                  localparam Not_A = 2'b10;         // ~A

    16                                                  localparam ReductionOR_B = 2'b11; // |B

    17

    18                                                  // Do the operation

    19               1                       101        always @(*) begin

    20                                                     case (Opcode)

    21               1                        36              Add: Alu_out = A + B;

    22               1                        35              Sub: Alu_out = A - B;

    23               1                        14              Not_A: Alu_out = ~A;

    24               1                        15              ReductionOR_B: Alu_out = |B;

    25               1                         1             default: Alu_out = 5'b0;

    26                                                     endcase

    27                                                  end // always @ (*)
```



```
    29                                              // Register output C

    30              1                       184         always @(posedge clk or posedge reset) begin

    31                                                     if (reset)

    32              1                        22               C <= 5'b0;

    33                                                     else

    34              1                       162               C <= Alu_out;


Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         44        44         0   100.00%

===============================Toggle Details===============================

Toggle Coverage for instance /\ALU_tb#DUT  --

                                              Node      1H->0L      0L->1H                         "Coverage"
                                              ---------------------------------------------------------------
                                            A[0-3]           1           1                             100.00
                                      Alu_out[4-0]           1           1                             100.00
                                            B[0-3]           1           1                             100.00
                                            C[4-0]           1           1                             100.00
                                       Opcode[0-1]           1           1                             100.00
                                               clk           1           1                             100.00
                                             reset           1           1                             100.00

Total Node Count     =         22
Toggled Node Count   =         22
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (44 of 44 bins)


Total Coverage By Instance (filtered view): 100.00%

```