

## Assignment 1

### 1- Adder

#### I. Verification plan

| 1  | LABEL    | Description   | Stimulus Generation                     | Function check  |
|----|----------|---|---|---|
| 2  | ADDER_1  | when the reset is asserted . The output C value must be low           | directed at the start of the simulation | A checker in the testbench to make sure the output is correct |
| 3  | ADDER_2  | verifying maximum negative value on A and maximum negative value on B | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 4  | ADDER_3  | verifying maximum negative value on A and ZERO value on B             | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 5  | ADDER_4  | verifying maximum negative value on A and maximum positive value on B | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 6  | ADDER_5  | verifying ZERO value on A and maximum negative value on B             | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 7  | ADDER_6  | verifying ZERO value on A and ZERO value on B                         | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 8  | ADDER_7  | verifying maximum positive value on A and maximum negative value on B | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 9  | ADDER_8  | verifying maximum positive value on A and ZERO value on B             | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 10 | ADDER_9  | verifying maximum positive value on A and maximum positive value on B | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 11 | ADDER_10 | verifying ZERO value on A and maximum positive value on B             | directed during the simulation          | A checker in the testbench to make sure the output is correct |

#### II. Testbench

|    |                               |    |                         |
|----|-------------------------------|----|-------------------------|
| 1  | module adder_tb ();           | 34 | reset = 0 ;             |
| 2  |                               | 35 | A = MAXNEG ;            |
| 3  | localparam MAXPOS = 7 ;       | 36 | B = MAXNEG ;            |
| 4  | localparam MAXNEG = -8 ;      | 37 | no_case = no_case + 1 ; |
| 5  |                               | 38 | check_result(-16);      |
| 6  | logic clk ;                   | 39 |                         |
| 7  | logic reset ;                 | 40 | A = MAXNEG ;            |
| 8  | logic signed [3:0] A ;        | 41 | B = 0 ;                 |
| 9  | logic signed [3:0] B ;        | 42 | no_case = no_case + 1 ; |
| 10 | logic signed [4:0] C ;        | 43 | check_result(-8);       |
| 11 |                               | 44 |                         |
| 12 | logic [4:0] no_case ;         | 45 |                         |
| 13 | integer correct_counter = 0 ; | 46 | A = MAXNEG ;            |
| 14 | integer error_counter = 0 ;   | 47 | B = MAXPOS ;            |
| 15 | adder dut (                   | 48 | no_case = no_case + 1 ; |
| 16 | .clk(clk),                    | 49 | check_result(-1);       |
| 17 | .reset(reset),                | 50 |                         |
| 18 | .A(A),                        | 51 |                         |
| 19 | .B(B),                        | 52 | A = 0 ;                 |
| 20 | .C(C)                         | 53 | B = MAXNEG ;            |
| 21 | );                            | 54 | no_case = no_case + 1 ; |
| 22 |                               | 55 | check_result(-8);       |
| 23 |                               | 56 |                         |
| 24 | always #20 clk = ~clk ;       | 57 |                         |
| 25 |                               | 58 | A = 0 ;                 |
| 26 | initial                       | 59 | B = 0 ;                 |
| 27 | begin                         | 60 | no_case = no_case + 1 ; |
| 28 | clk = 0 ;                     | 61 | check_result(0);        |
| 29 | A = 0 ;                       | 62 |                         |
| 30 | B = 0 ;                       |    |                         |
| 31 | no_case = 0 ;                 |    |                         |
| 32 | rest();                       |    |                         |
| 33 |                               |    |                         |

```

62
63     A = MAXPOS ;
64     B = MAXNEG ;
65     no_case = no_case + 1 ;
66     check_result(-1);
67
68     A = MAXPOS ;
69     B = 0 ;
70     no_case = no_case + 1 ;
71     check_result(7);
72
73
74     A = MAXPOS ;
75     B = MAXPOS ;
76     no_case = no_case + 1 ;
77     check_result(14);
78
79     A = 0 ;
80     B = MAXPOS ;
81     no_case = no_case + 1 ;
82     check_result(7);
83
84
85     $display ("error_counter = %0d " ,error_counter );
86     $display ("correct_counter = %0d " ,correct_counter );
87     $stop ;
88
89 end
90

```

```

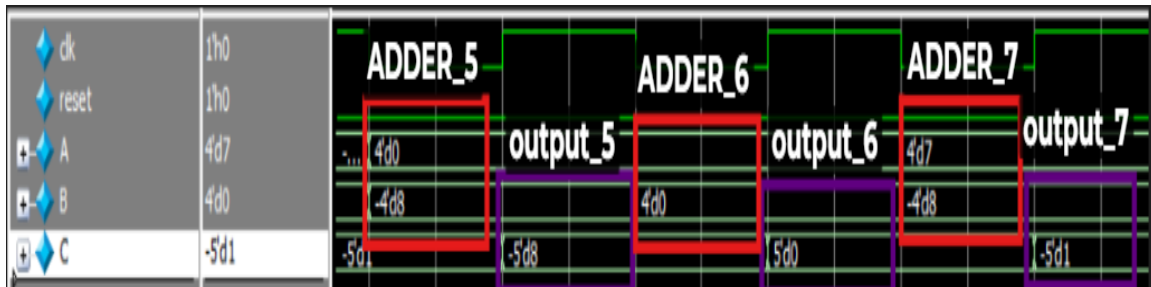
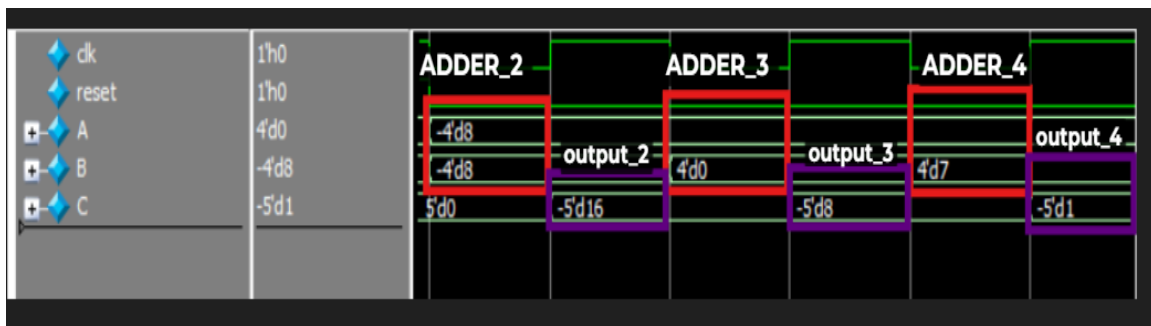
91
92 task check_result(input signed [4:0] expected_result );
93     @(negedge clk );
94     if(expected_result != C )
95     begin
96         error_counter = error_counter +1 ;
97         $display ("%t:error @ case = %0d ==> A = %0d , B = %0d ==> output equal C = %0d should be %0d " , $time , no_case ,A , B , expected_result , C );
98     end
99     else
100         correct_counter = correct_counter + 1 ;
101
102
103 endtask
104
105 task rest ();
106     reset = 0 ;
107     #1
108     reset = 1 ;
109     check_result(0);
110     reset = 0 ;
111
112 endtask
113
114
115 endmodule

```

### III. Do file

```
1  vlib work
2  vlog adder.v adder_tb.sv +cover -covercells
3  vsim -voptargs=+acc work.adder_tb -cover
4  add wave *
5  coverage save adder_tb.ucdb -onexit
6  run -all
```

### IV. Waveform



## V. Coverage

### 1- Branch

|                     |        |      |        |           |
|---------------------|--------|------|--------|-----------|
| Statement Coverage: |        |      |        |           |
| Enabled Coverage    | Active | Hits | Misses | % Covered |
| -----               | -----  | ---- | -----  | -----     |
| Stmts               | 3      | 3    | 0      | 100.0     |

=====Statement Details=====

Statement Coverage for file adder.v --

```
1      module adder (  
2          input  clk,  
3          input  reset,  
4          input  signed [3:0] A, // Input data A in 2's complement  
5          input  signed [3:0] B, // Input data B in 2's complement  
6          output reg signed [4:0] C // Adder output in 2's complement  
7      );  
8  
9          // Register output C  
10         always @(posedge clk or posedge reset) begin  
11             if (reset)  
12                 C <= 5'b0;  
13             else  
14                 C <= A + B;  
15             end  
16  
17     endmodule
```

### 2- Statement

|                  |        |      |        |           |
|------------------|--------|------|--------|-----------|
| Branch Coverage: |        |      |        |           |
| Enabled Coverage | Active | Hits | Misses | % Covered |
| -----            | -----  | ---- | -----  | -----     |
| Branches         | 2      | 2    | 0      | 100.0     |

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

Branch Coverage for file adder.v --

-----IF Branch-----

|    |   |    |                       |
|----|---|----|-----------------------|
| 11 |   | 11 | Count coming in to IF |
| 11 | 1 | 2  | if (reset)            |
| 13 | 1 | 9  | else                  |

Branch totals: 2 hits of 2 branches = 100.0%

### 3- Toggle coverage

63 Toggle Coverage:
64 Enabled Coverage Active Hits Misses % Covered
65 -----
66 Toggle Bins 30 30 0 100.0
67
68 =====Toggle Details=====
69
70 Toggle Coverage for File adder.v --
71
72 Line Node 1H->0L 0L->1H "Coverage"
73 -----
74 2 clk 1 1 100.00
75 3 reset 1 1 100.00
76 4 A[3] 1 1 100.00
77 4 A[2] 1 1 100.00
78 4 A[1] 1 1 100.00
79 4 A[0] 1 1 100.00
80 5 B[3] 1 1 100.00
81 5 B[2] 1 1 100.00
82 5 B[1] 1 1 100.00
83 5 B[0] 1 1 100.00
84 6 C[4] 1 1 100.00
85 6 C[3] 1 1 100.00
86 6 C[2] 1 1 100.00
87 6 C[1] 1 1 100.00
88 6 C[0] 1 1 100.00
89

### 2- Priority encoder

#### I. Design

```
1  module priority_enc (
2  input  clk,
3  input  rst,
4  input  [3:0] D,
5  output reg [1:0] Y,
6  output reg valid
7  );
8
9  always @(posedge clk) begin
10     if (rst)
11         begin
12             Y <= 2'b00;
13             valid <= 1'b0 ;
14         end
15     else
16         begin
17             casex (D)
18                 4'b1000: Y <= 0;
19                 4'bX100: Y <= 1;
20                 4'bXX10: Y <= 2;
21                 4'bXXX1: Y <= 3;
22                 default: Y <= 2'b00;
23             endcase
24             valid <= (~|D)? 1'b0: 1'b1;
25         end
26     end
27 endmodule
```

Add the Valid signal is equal to zero if rst is high. And add default case in the case when the input “ D = 4'b0000 “

## II. Verification plan

| LABEL  | Description   | Stimulus Generation                     | Function check  |
|--------|---|---|---|
| case_1 | when the reset is asserted . The output C value must be low && valid must be equal 0      | directed at the start of the simulation | A checker in the testbench to make sure the output is correct |
| case_2 | when D = 4'b1000 . The output C value must be low && valid must be equal 1                | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| case_3 | when D = 4'b0100 . The output C value must be equal 1 && valid must be equal 1            | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| case_4 | when D = 4'b1010 . The output C value must be equal 1 && valid must be equal 1            | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| case_5 | when D = 4'b0101 . The output C value must be equal 1 && valid must be equal 1            | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| case_6 | when D = 4'b1000 . The output C value must be low && valid must be equal 1                | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| case_7 | when D = 4'b0000 . The output C value must be low "default case" && valid must be equal 0 | directed during the simulation          | A checker in the testbench to make sure the output is correct |

## III. Testbench

```

1  module priority_enc_tb ();
2
3
4      logic      clk      ;
5      logic      rst      ;
6      logic [3:0] D      ;
7      logic [1:0] Y      ;
8      logic      valid    ;
9
10
11  integer correct_counter = 0 ;
12  integer error_counter  = 0 ;
13
14  priority_enc dut (
15      .clk(clk),
16      .rst(rst),
17      .D(D),
18      .Y(Y),
19      .valid(valid)
20  );
21
22
23  always #20 clk = ~clk ;
24
25  initial
26      begin
27          clk = 0 ;
28          D  = 0 ;
29          rest ();
30
31          D  = 4'b1000 ;
32          check_result(0 , 1);
33
34          D  = 4'b0100 ;
35          check_result(1 , 1);
36
37
38          D  = 4'b1010 ;
39          check_result(2 , 1);
40
41          D  = 4'b0101 ;
42          check_result(3 , 1);
43
44          D  = 4'b1000 ;
45          check_result(0 , 1);
46
47          D  = 4'b0000 ;
48          check_result(2'b0 , 0);
49
50          $display ("error_counter = %0d " ,error_counter );
51          $display ("correct_counter = %0d " ,correct_counter );
52
53          $stop ;
54
55  end
56
57
58  task check_result(input [1:0] expected_result , input expected_valid );
59      @(negedge clk );
60      if(expected_result != Y && expected_valid != valid )
61          begin
62              error_counter = error_counter +1 ;
63              $display (":error " );
64          end
65      else
66          correct_counter = correct_counter + 1 ;
67      endtask
68
69
70  task rest ();
71      rst = 0 ;
72      #1
73      rst = 1 ;
74      check_result(2'b0 , 0 );
75      rst = 0 ;
76  endtask
77
78
79
80
81
82  endmodule

```

#### IV. Do file

```

1 vlib work
2 vlog priority_enc.v priority_enc_tb.svh +cover -covercells
3 vsim -voptargs=+acc work.priority_enc_tb -cover
4 add wave *
5 coverage save priority_enc_tb.ucdb -onexit
6 run -all
7

```

#### V. Waveform



#### VI. Coverage

##### 1- Branch

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

Statement Coverage:

Enabled Coverage

Active

Hits

Misses % Covered

-----

-----

-----

-----

Stmts

9

9

0

100.0

=====Statement Details=====

Statement Coverage for file priority\_enc.v --

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

module priority\_enc (

input clk,

input rst,

input [3:0] D,

output reg [1:0] Y,

output reg valid

);

always @(posedge clk) begin

if (rst)

begin

Y <= 2'b0;

valid <= 1'b0 ;

end

else

begin

casex (D)

4'b1000: Y <= 0;

4'bX100: Y <= 1;

4'bXX10: Y <= 2;

4'bXXX1: Y <= 3;

default: Y <= 2'b0;

endcase

valid <= (~[D])? 1'b0: 1'b1;

## 2- Statement

```
Branch Coverage:
  Enabled Coverage      Active      Hits      Misses % Covered
-----
  Branches              7          7          0    100.0

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

Branch Coverage for file priority_enc.v --

-----IF Branch-----
  10                      7      Count coming in to IF
  10          1          1      if (rst)
  15          1          6      else
Branch totals: 2 hits of 2 branches = 100.0%

-----CASE Branch-----
  17                      6      Count coming in to CASE
  18          1          2      4'b1000: Y <= 0;
  19          1          1      4'bX100: Y <= 1;
  20          1          1      4'bXX10: Y <= 2;
  21          1          1      4'bXXX1: Y <= 3;
  22          1          1      default: Y <= 2'b0;
Branch totals: 5 hits of 5 branches = 100.0%
```

## 3- Toggle coverage

```
Toggle Coverage:
  Enabled Coverage      Active      Hits      Misses % Covered
-----
  Toggle Bins          18          18          0    100.0

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

Toggle Coverage for File priority_enc.v --

  Line      Node      1H->0L      0L->1H      "Coverage"
-----
  2          clk          1          1      100.00
  3          rst          1          1      100.00
  4          D[3]          1          1      100.00
  4          D[2]          1          1      100.00
  4          D[1]          1          1      100.00
  4          D[0]          1          1      100.00
  5          Y[1]          1          1      100.00
  5          Y[0]          1          1      100.00
  6          valid          1          1      100.00

Total Node Count      =      9
Toggled Node Count    =      9
Untoggled Node Count  =      0

Toggle Coverage      =    100.0% (18 of 18 bins)
```



### 3- Alu

#### I. Verification plan

|   | LABEL  | Description   | Stimulus Generation                     | Function check  |
|---|--------|---|---|---|
| 1 |        |   |   |   |
| 2 | case_1 | when the reset is asserted . The output C value must be low                             | directed at the start of the simulation | A checker in the testbench to make sure the output is correct |
| 3 | case_2 | verifying maximum negative value on A and maximum negative value on B && Opcode equal 0 | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 4 | case_3 | verifying maximum negative value on A and ZERO value on B && Opcode equal 1             | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 5 | case_4 | verifying maximum negative value on A and maximum positive value on B && Opcode equal 2 | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 6 | case_5 | verifying ZERO value on A and maximum negative value on B && Opcode equal 3             | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 7 | case_6 | verifying maximum positive value on A and ZERO value on B && Opcode equal 0             | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 8 | case_7 | verifying ZERO value on A and ZERO value on B && Opcode equal 1                         | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 9 | case_8 | verifying maximum positive value on A and ZERO value on B && Opcode equal 2             | directed during the simulation          | A checker in the testbench to make sure the output is correct |

#### II. Testbench

|    |  |    |  |                                      |
|----|--|----|--|--------------------------------------|
| 1  | <code>module ALU_4_bit_tb ();</code>       | 37 |  | <code>reset = 0 ;</code>             |
| 2  |  | 38 |  | <code>A = MAXNEG ;</code>            |
| 3  | <code>localparam MAXPOS = 7 ;</code>       | 39 |  | <code>B = MAXNEG ;</code>            |
| 4  | <code>localparam MAXNEG = -8 ;</code>      | 40 |  | <code>Opcode = 0 ;</code>            |
| 5  |  | 41 |  | <code>no_case = no_case + 1 ;</code> |
| 6  | <code>logic clk ;</code>                   | 42 |  | <code>check_result(-16);</code>      |
| 7  | <code>logic reset ;</code>                 | 43 |  |                                      |
| 8  | <code>logic [1:0] Opcode;</code>           | 44 |  | <code>A = MAXNEG ;</code>            |
| 9  | <code>logic signed [3:0] A ;</code>        | 45 |  | <code>B = 0 ;</code>                 |
| 10 | <code>logic signed [3:0] B ;</code>        | 46 |  | <code>Opcode = 1 ;</code>            |
| 11 | <code>logic signed [4:0] C ;</code>        | 47 |  | <code>no_case = no_case + 1 ;</code> |
| 12 |  | 48 |  | <code>check_result(-8);</code>       |
| 13 | <code>logic [4:0] no_case ;</code>         | 49 |  |                                      |
| 14 | <code>integer correct_counter = 0 ;</code> | 50 |  | <code>A = MAXNEG ;</code>            |
| 15 | <code>integer error_counter = 0 ;</code>   | 51 |  | <code>B = MAXPOS ;</code>            |
| 16 |  | 52 |  | <code>Opcode = 2 ;</code>            |
| 17 | <code>ALU_4_bit dut (</code>               | 53 |  | <code>no_case = no_case + 1 ;</code> |
| 18 | <code>  .clk(clk),</code>                  | 54 |  | <code>check_result(7);</code>        |
| 19 | <code>  .reset(reset),</code>              | 55 |  |                                      |
| 20 | <code>  .Opcode(Opcode),</code>            | 56 |  | <code>A = 0 ;</code>                 |
| 21 | <code>  .A(A),</code>                      | 57 |  | <code>B = MAXNEG ;</code>            |
| 22 | <code>  .B(B),</code>                      | 58 |  | <code>Opcode = 3 ;</code>            |
| 23 | <code>  .C(C)</code>                       | 59 |  | <code>no_case = no_case + 1 ;</code> |
| 24 | <code>);</code>                            | 60 |  | <code>check_result(1);</code>        |
| 25 |  | 61 |  |                                      |
| 26 | <code>always #20 clk = ~clk ;</code>       | 62 |  |                                      |
| 27 |  | 63 |  |                                      |
| 28 | <code>initial</code>                       |    |  |                                      |
| 29 | <code>  begin</code>                       |    |  |                                      |
| 30 | <code>    clk = 0 ;</code>                 |    |  |                                      |
| 31 | <code>    A = 0 ;</code>                   |    |  |                                      |
| 32 | <code>    B = 0 ;</code>                   |    |  |                                      |
| 33 | <code>    Opcode = 0 ;</code>              |    |  |                                      |
| 34 | <code>    no_case = 0 ;</code>             |    |  |                                      |
| 35 | <code>    rest();</code>                   |    |  |                                      |

```

63
64     A = MAXPOS ;
65     B = 0 ;
66     Opcode = 0 ;
67     no_case = no_case + 1 ;
68     check_result(7);
69
70
71
72     A = 0 ;
73     B = 0 ;
74     Opcode = 1 ;
75     no_case = no_case + 1 ;
76     check_result(0);
77
78
79     A = MAXPOS ;
80     B = 0 ;
81     Opcode = 2 ;
82     no_case = no_case + 1 ;
83     check_result(-8);
84
85
86
87     $display ("error_counter = %0d " ,error_counter );
88     $display ("correct_counter = %0d " ,correct_counter );
89     $stop ;
90
91 end
92
93
94 task check_result(input signed [4:0] expected_result );
95     @(negedge clk );
96     if(expected_result != C )
97         begin
98             error_counter = error_counter +1 ;
99             $display ("%0t:error no_case = %0d ; ==> A = %0d , B = %0d ==> output equal C = %0d should be %0d " , $time , no_case , A , B , expected_result , C );
100         end
101     else
102         correct_counter = correct_counter + 1 ;
103
104
105 endtask
106
107 task rest ();
108     reset = 0 ;
109     #1
110     reset = 1 ;
111     check_result(0);
112     reset = 0 ;
113
114 endtask
115
116
117 endmodule

```

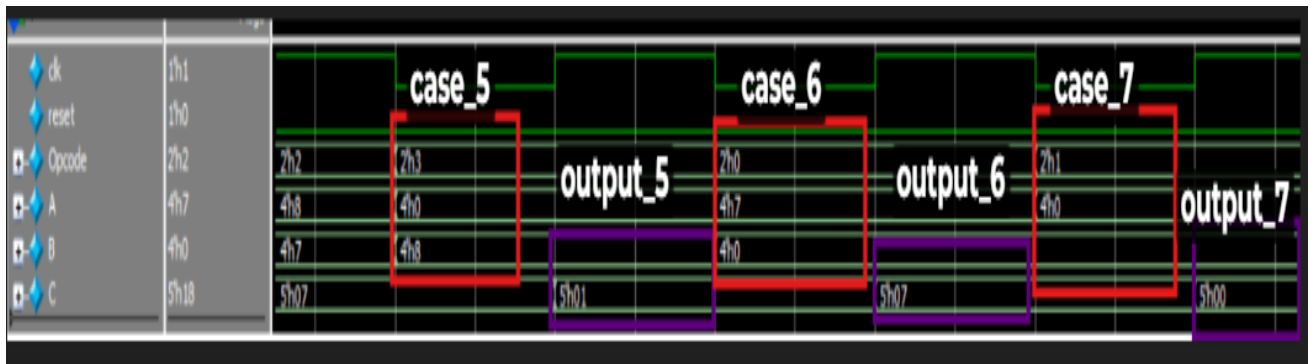
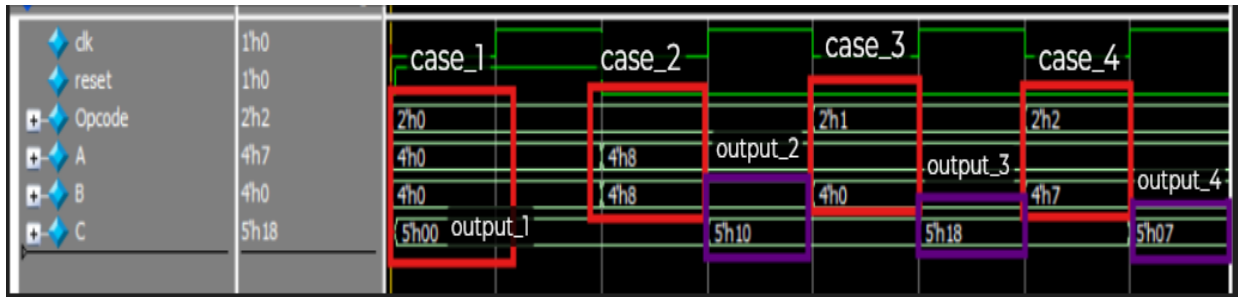
### III. Do file

```

1  vlib work
2  vlog ALU.v alu_tb.sv +cover -covercells
3  vsim -voptargs=+acc work.ALU_4_bit_tb -cover
4  add wave *
5  coverage save alu_tb.ucdb -onexit
6  run -all
7  coverage exclude -du ALU_4_bit_tb -togglenode {C[0]}
8  coverage exclude -du ALU_4_bit -togglenode {Alu_out[0]}
9  coverage exclude -src ALU.v -line 26 -code s
10 coverage exclude -src ALU.v -line 26 -code b
11 coverage exclude -clear -du ALU_4_bit_tb -togglenode {C[0]}

```

#### IV. Waveform



#### V. Coverage

##### 1- Branch

```

5  =====
6  Statement Coverage:
7      Enabled Coverage      Active      Hits      Misses % Covered
8      -----
9      Stmts                8          8          0    100.0
10
11  =====Statement Details=====
12
13  Statement Coverage for file ALU.v --
14
15      1      module ALU_4 bit (
16      2          input clk,
17      3          input reset,
18      4          input [1:0] Opcode, // The opcode
19      5          input signed [3:0] A, // Input data A in 2's complement
20      6          input signed [3:0] B, // Input data B in 2's complement
21      7
22      8          output reg signed [4:0] C // ALU output in 2's complement
23      9
24      10      );
25      11
26      12      reg signed [4:0] Alu_out; // ALU output in 2's complement
27      13
28      14      localparam Add = 3'b00; // A + B
29      15      localparam Sub = 3'b01; // A - B
30      16      localparam Not_A = 3'b10; // ~A
31      17      localparam ReductionOR_B = 3'b11; // |B
32      18
33      19      // Do the operation
34      20          1      8      always @* begin
35      21          21      case (Opcode)
36      22          22          1      3      Add:      Alu_out = A + B;

```

## 2- Statement

```
53
54 Branch Coverage:
55   Enabled Coverage      Active      Hits      Misses % Covered
56   -----
57   Branches              6         6         0      100.0
58
59 =====Branch Details=====
60
61 Branch Coverage for file ALU.v --
62
63 -----CASE Branch-----
64   21                      8      Count coming in to CASE
65   22                      3      Add:      Alu_out = A + B;
66   23                      2      Sub:      Alu_out = A - B;
67   24                      2      Not_A:    Alu_out = ~A;
68   25                      1      ReductionOR_B: Alu_out = |B;
69   26                      E      default: Alu_out = 5'b0;
70 Branch totals: 4 hits of 4 branches = 100.0%
71
72 -----IF Branch-----
73   32                      9      Count coming in to IF
74   32                      2      if (reset)
75   34                      7      else
76 Branch totals: 2 hits of 2 branches = 100.0%
77
```

## 3- Toggle coverage

```
101
102   Line      Node      1H->0L      0L->1H      "Coverage"
103   -----
104   2         clk        1          1          100.00
105   3         reset      1          1          100.00
106   4         Opcode[1]  1          1          100.00
107   4         Opcode[0]  1          1          100.00
108   5         A[3]       1          1          100.00
109   5         A[2]       1          1          100.00
110   5         A[1]       1          1          100.00
111   5         A[0]       1          1          100.00
112   6         B[3]       1          1          100.00
113   6         B[2]       1          1          100.00
114   6         B[1]       1          1          100.00
115   6         B[0]       1          1          100.00
116   8         C[4]       1          1          100.00
117   8         C[3]       1          1          100.00
118   8         C[2]       1          1          100.00
119   8         C[1]       1          1          100.00
120   8         C[0]       1          1          100.00
121   12        Alu_out[4]  1          1          100.00
122   12        Alu_out[3]  1          1          100.00
123   12        Alu_out[2]  1          1          100.00
124   12        Alu_out[1]  1          1          100.00
125   12        Alu_out[0]  E-hit      E-hit      100.00
126
127 Total Node Count      =      21
128 Toggled Node Count   =      21
129 Untoggled Node Count =      0
130
131 Toggle Coverage      =    100.0% (42 of 42 bins)
132
```

#### 4- DSP

##### I. Design

```

1  module DSP(A, B, C, D, clk, rst_n, P);
2  parameter OPERATION = "ADD";
3  input  [18:0] A ;
4  input  [17:0] B, D;
5  input  [37:0] C;
6  input clk, rst_n;
7  output reg [38:0] P;
8
9  reg [17:0] B_reg, D_reg ;
10 reg [18:0] A_reg_stg1, A_reg_stg2, adder_out_stg1 , adder_out_stg2 ;
11 reg [37:0] C_reg ;
12 reg [38:0] mult_out ;
13
14 always @(posedge clk or negedge rst_n) begin
15     if (!rst_n) begin
16         // reset
17         A_reg_stg1 <= 0;
18         A_reg_stg2 <= 0;
19         B_reg <= 0;
20         D_reg <= 0;
21         C_reg <= 0;
22         adder_out_stg1 <= 0;
23         adder_out_stg2 <= 0;
24         mult_out <= 0;
25         P <= 0;
26     end
27     else begin
28         A_reg_stg1 <= A;
29         A_reg_stg2 <= A_reg_stg1;
30         B_reg <= B;
31         C_reg <= C;
32         D_reg <= D;
33         adder_out_stg2 <= adder_out_stg1;
34         if (OPERATION == "ADD") begin

```

Change the width of A , P , A\_reg\_stg1, A\_reg\_stg2, adder\_out\_stg1 , adder\_out\_stg2 and mult\_out

##### II. Verification plan

|   | LABEL  | Description  | Stimulus Generation                     | Function check  |
|---|--------|--|---|---|
| 1 |        |  |   |   |
| 2 | case_1 | when the reset is asserted . The output C value must be low                                | directed at the start of the simulation | A checker in the testbench to make sure the output is correct |
| 3 | case_2 | The test runs for 1000 iterations. For each iteration, four random values to A, B, C and D | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 4 |        |  |   |   |

### III. Testbench

```
1  module DSP_tb ();
2
3  logic [18:0] A ;
4  logic [17:0] B, D;
5  logic [37:0] C;
6  logic clk , rst_n;
7  logic [38:0] P;
8
9  integer i;
10
11  // logic [4:0] no_case ;
12  integer correct_counter = 0 ;
13  integer error_counter = 0 ;
14  logic [38:0] output_ex;
15
16  DSP dut (
17      .clk(clk),
18      .rst_n(rst_n),
19      .D(D),
20      .A(A),
21      .B(B),
22      .C(C),
23      .P(P)
24  );
25
26  always #10 clk = ~clk ;
27
28
29  initial
30  begin
31      clk = 0 ;
32      A = 0;
33      B = 0 ;
34      C = 0 ;
35      D = 0 ;
36      output_ex = 0 ;
37      // no_case = 0 ;
38      rest();
39
40      for (i=0 ; i < 1000 ; i = i+1)
41      begin
42
43          A = $urandom_range(10, 524288) ;
44          B = $urandom_range(10, 262144) ;
45          C = $urandom_range(10, 52428800) ;
46          D = $urandom_range(10, 262144) ;
47          output_ex = ((D+B)*A)+C ;
48          @(negedge clk );
49          @(negedge clk );
50          @(negedge clk );
51          @(negedge clk );
52          check_result(output_ex);
53      end
54
55
56      $display ("error_counter = %0d " ,error_counter );
57      $display ("correct_counter = %0d " ,correct_counter );
58      $stop ;
59
60      end
61
62
63  task check_result(input [38:0] expected_result );
64      @(negedge clk );
65
66      if(expected_result != P )
67      begin
68          error_counter = error_counter +1 ;
69          $display (":error");
70      end
71      else
72          correct_counter = correct_counter + 1 ;
73
74
75  endtask
76
77  task rest ();
78      rst_n = 1 ;
79      #1
80      rst_n = 0 ;
81      check_result(0);
82      rst_n = 1 ;
83
84  endtask
85
86
87  endmodule
```

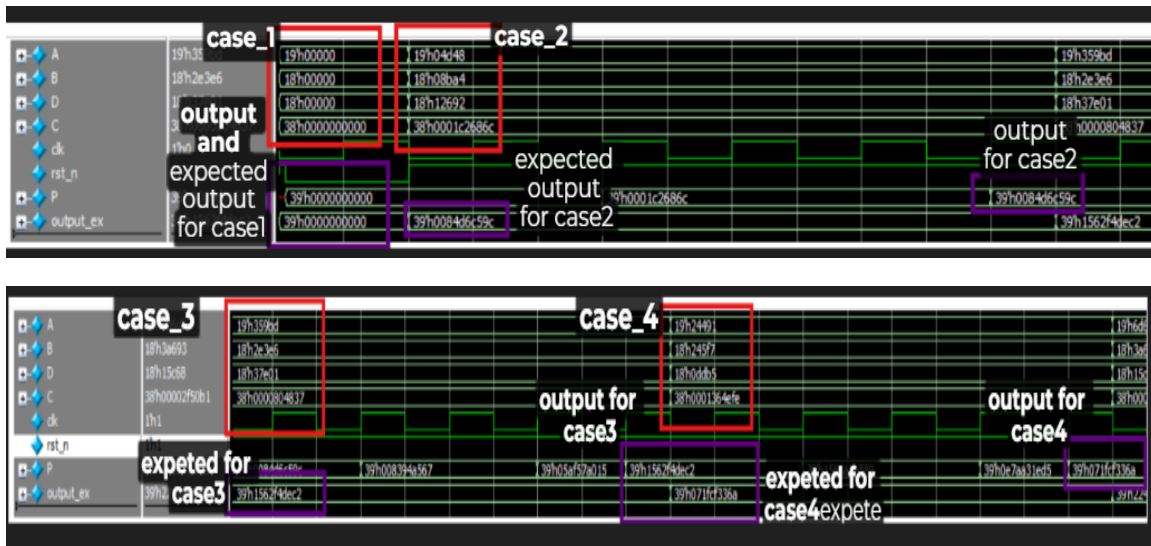
#### IV. Do file

```

1  vlib work
2  vlog DSP.v DSP_tb.sv +cover -covercells
3  vsim -voptargs=+acc work.DSP_tb -cover
4  add wave *
5  coverage save DSP_tb.ucdb -onexit
6  run -all
7  coverage exclude -du DSP -togglenode C
8  coverage exclude -du DSP -togglenode C_reg
9  coverage exclude -du DSP -togglenode mult_out
10 coverage exclude -du DSP -togglenode P

```

#### V. Waveform



## VI. Coverage

### 1- Branch

```
5 =====
6 Statement Coverage:
7   Enabled Coverage      Active      Hits      Misses % Covered
8   -----
9   Stmts                19         19         0      100.0
10
11 =====Statement Details=====
12
13 Statement Coverage for file DSP.v --
14
15     1      module DSP(A, B, C, D, clk, rst_n, P);
16     2      parameter OPERATION = "ADD";
17     3      input  [18:0] A ;
18     4      input  [17:0] B, D;
19     5      input  [37:0] C;
20     6      input  clk, rst_n;
21     7      output reg [38:0] P;
22
23     8
24     9      reg [17:0] B_reg, D_reg ;
25    10      reg [18:0] A_reg_stg1, A_reg_stg2, adder_out_stg1 , adder_out_stg2 ;
26    11      reg [37:0] C_reg ;
27    12      reg [38:0] mult_out ;
28
29    13
30    14      always @(posedge clk or negedge rst_n) begin
31    15      if (!rst_n) begin
32    16      // reset
33    17      A_reg_stg1 <= 0;
34    18      A_reg_stg2 <= 0;
35    19      B_reg <= 0;
36    20      D_reg <= 0;
37    21      C_reg <= 0;
38    22      adder_out_stg1 <= 0;
39    23      adder_out_stg2 <= 0;
40    24      mult_out <= 0;
41    25      end
42    26      end
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
```

### 2- Statement

```
62 Branch Coverage:
63   Enabled Coverage      Active      Hits      Misses % Covered
64   -----
65   Branches                2         2         0      100.0
66
67 =====Branch Details=====
68
69 Branch Coverage for file DSP.v --
70
71 -----IF Branch-----
72     15      5007      Count coming in to IF
73     15      1         2      if (!rst_n) begin
74     27      1      5005      else begin
75 Branch totals: 2 hits of 2 branches = 100.0%
76
77
```

### 3- Toggle coverage

```
91 Transitions      0      0      0      100.0
92 Toggle Coverage:
93   Enabled Coverage      Active      Hits      Misses % Covered
94   -----
95   Toggle Bins          338      338         0      100.0
96
97 =====Toggle Details=====
98
99 Toggle Coverage for File DSP.v --
100
101   Line      Node      1H->0L      0L->1H      "Coverage"
102   -----
103   3      A[9]      1      1      100.00
104   3      A[8]      1      1      100.00
105   3      A[7]      1      1      100.00
106   3      A[6]      1      1      100.00
107   3      A[5]      1      1      100.00
108   3      A[4]      1      1      100.00
109   3      A[3]      1      1      100.00
110   3      A[2]      1      1      100.00
111   3      A[1]      1      1      100.00
112   3      A[18]      1      1      100.00
113   3      A[17]      1      1      100.00
114   3      A[16]      1      1      100.00
115   3      A[15]      1      1      100.00
116   3      A[14]      1      1      100.00
117   3      A[13]      1      1      100.00
```



## 5- D\_FF

### I. Design

```
C: > Users > CS > Downloads > dff > ≡ dff.v
1  module dff#(parameter USE_EN = 1)(clk, rst, d, q, en);
2  input clk, rst, d, en;
3  output reg q;
4
5  always @(posedge clk) begin
6      if (rst)
7          q <= 0;
8      else
9          begin
10             if(USE_EN)
11                 begin
12                     if (en)
13                         q <= d;
14                     else
15                         q <= q;
16                 end
17             else
18                 q <= d;
19         end
20     end
21
22 endmodule
```

### II. Verification plan for the two testbench

|   | LABEL  | Description  | Stimulus Generation                     | Function check  |
|---|--------|--|---|---|
| 1 |        |  |   |   |
| 2 | case_1 | when the reset is asserted . The output C value must be low && valid must be equal 0 | directed at the start of the simulation | A checker in the testbench to make sure the output is correct |
| 3 | case_2 | Verifying when the d= 1 && en = 0  | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 4 | case_3 | Verifying when the d= 1 && en = 1  | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 5 | case_4 | Verifying when the d= 0 && en = 0  | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 6 | case_5 | Verifying when the d= 0 && en = 1  | directed during the simulation          | A checker in the testbench to make sure the output is correct |
| 7 |        |  |   |   |

### III. Testbench1

```
C: > Users > CS > Downloads > dff > dff_t1.sv
1  module dff_t1 #(parameter USE_EN = 1)();
2
3      logic clk ;
4      logic rst ;
5      logic d ;
6      logic q ;
7      logic en ;
8
9
10     integer correct_counter = 0 ;
11     integer error_counter = 0 ;
12
13     dff #(.USE_EN(USE_EN)) du1 (
14         .clk(clk),
15         .rst(rst),
16         .d(d),
17         .q(q),
18         .en(en)
19     );
20
21     always #20 clk = ~clk ;
22
23     initial
24     begin
25         clk = 0 ;
26         en = 0 ;
27         d = 0 ;
28         rest();
29
30         en = 0 ;
31         d = 1 ;
32         check_result(0);
33
34
35         en = 1 ;
36         d = 1 ;
37         check_result(1);
38
39
40         en = 0 ;
41         d = 0 ;
42         check_result(1);
43
44
45         en = 1 ;
46         d = 0 ;
47         check_result(0);
48
49
50         $display (" testbench 1 " );
51         $display ("error_counter = %0d ",error_counter );
52         $display ("correct_counter = %0d ",correct_counter );
53         $stop ;
54
55     end
56
57
```

```
58
59     task check_result(input expected_result );
60         @(negedge clk );
61         if(expected_result != q )
62             begin
63                 $display (":error");
64                 error_counter = error_counter + 1 ;
65             end
66         else
67             correct_counter = correct_counter + 1 ;
68
69     endtask
70
71
72     task rest ();
73         rst = 0 ;
74         #1
75         rst = 1 ;
76         check_result(0);
77         rst = 0 ;
78
79     endtask
80
81
82 endmodule
```

#### IV. Testbench2

```

1  module dff_t2 #(parameter USE_EN = 0)();
2
3      logic clk ;
4      logic rst ;
5      logic d ;
6      logic q ;
7      logic en ;
8
9
10 integer correct_counter = 0 ;
11 integer error_counter = 0 ;
12
13 dff #(.USE_EN(USE_EN)) du2(
14     .clk(clk),
15     .rst(rst),
16     .d(d),
17     .q(q),
18     .en(en)
19 );
20
21
22 always #20 clk = ~clk ;
23
24 initial
25     begin
26         clk = 0 ;
27         en = 0 ;
28         d = 0 ;
29         rest();
30
31         en = 0 ;
32         d = 1 ;
33         check_result(1);
34
35
36         en = 1 ;
37         d = 1 ;
38         check_result(1);
39
40
41         en = 0 ;
42         d = 0 ;
43         check_result(0);
44
45
46         en = 1 ;
47         d = 0 ;
48         check_result(0);
49
50
51         $display (" testbench 2 " );
52         $display ("error_counter = %0d ",error_counter );
53         $display ("correct_counter = %0d ",correct_counter );
54         $stop ;
55
56
57
58     end

```

```

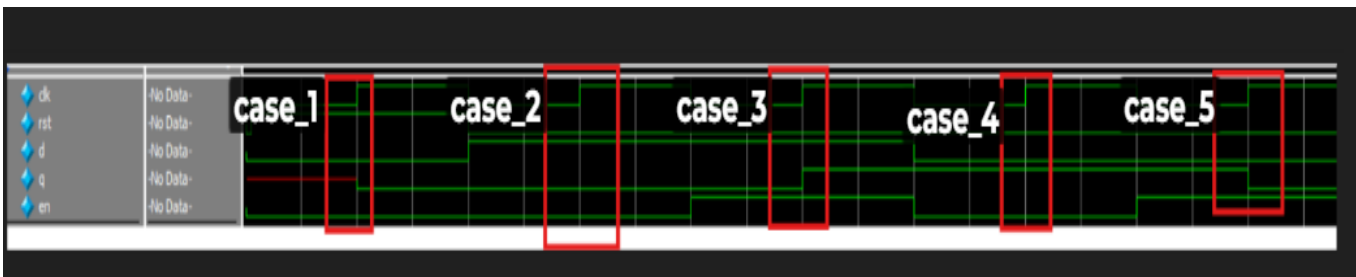
60
61     task check_result(input expected_result );
62         @(negedge clk );
63         if(expected_result != q )
64             begin
65                 $display (":error");
66                 error_counter = error_counter +1 ;
67             end
68         else
69             correct_counter = correct_counter + 1 ;
70
71     endtask
72
73
74     task rest ();
75         rst = 0 ;
76         #1
77         rst = 1 ;
78         check_result(0);
79         rst = 0 ;
80
81     endtask
82
83
84 endmodule

```

V. Do file

```
1 vlib work
2 vlog dff.v dff_t1.sv +cover -covercells
3 vsim -voptargs=+acc work.dff_t1 -cover
4 add wave *
5 coverage save dff_t1.ucdb -onexit
6 run -all
7
8 quit -sim
9
10 vlib work
11 vlog dff.v dff_t2.sv +cover -covercells
12 vsim -voptargs=+acc work.dff_t2 -cover
13 add wave *
14 coverage save dff_t2.ucdb -onexit
15 run -all
16
17 quit -sim
18
19 vccover merge dff_merged.ucdb dff_t1.ucdb dff_t2.ucdb -du dff
20 vccover report dff_merged.ucdb -details -all -output coverage.txt
21
22
```

VI. Waveform for testbench1



VII. Waveform for testbench2



## VIII. Total Coverage

### 1- Branch

```
5 =====
6 Statement Coverage:
7   Enabled Coverage      Active      Hits      Misses % Covered
8   -----
9   Stmts                5          5          0    100.0
10
11 =====Statement Details=====
12
13 Statement Coverage for file dff.v --
14
15   1      module dff#(parameter USE_EN = 1)(clk, rst, d, q, en
16   2      input clk, rst, d, en;
17   3      output reg q;
18   4
19   5          1      10      always @(posedge clk) begin
20   6              if (rst)
21   7                  1      2      q <= 0;
22   8              else
23   9                  begin
24   10                     if(USE_EN)
25   11                        begin
26   12                           if (en)
27   13                              1      2      q <= d;
28   14                              else
29   15                                 1      2      q <= q;
30   16                          end
31   17                      else
32   18                         1      4      q <= d;
33   19                      end
34   20                  end
35   21
36   22      endmodule
37
```

### 2- Statement

```
37
38 Branch Coverage:
39   Enabled Coverage      Active      Hits      Misses % Covered
40   -----
41   Branches                3          3          0    100.0
42
43 =====Branch Details=====
44
45 Branch Coverage for file dff.v --
46
47 -----IF Branch-----
48   6      10      Count coming in to IF
49   6          1      2      if (rst)
50   12         1      2      if (en)
51   14         1      6      else
52 Branch totals: 3 hits of 3 branches = 100.0%
53
54
55 Condition Coverage:
```

### 3- Toggle coverage

```
69 Toggle Coverage:
70   Enabled Coverage      Active      Hits      Misses % Covered
71   -----
72   Toggle Bins           10         10         0      100.0
73
74 =====Toggle Details=====
75
76 Toggle Coverage for File dff.v --
77
78   | Line                | Node  | 1H->0L  | 0L->1H  | "Coverage"
79   |-----|
80   | 2                    | rst   | 2        | 2        | 100.00
81   | 2                    | en    | 2        | 2        | 100.00
82   | 2                    | d     | 2        | 2        | 100.00
83   | 2                    | clk   | 2        | 2        | 100.00
84   | 3                    | q     | 2        | 2        | 100.00
85
86 Total Node Count      =      5
87 Toggled Node Count    =      5
88 Untoggled Node Count =      0
89
90 Toggle Coverage      =    100.0% (10 of 10 bins)
91
92
93 Total Coverage By File (code coverage only, filtered view): 100.0%
94
```