System Verilog

Verification Plan & Subroutines

Assignment 1

By: Magdy Ahmed Abbas Abdelhamid

Question 1

ALU

Verification plan & Subroutines

Requirements for the assignment in both questions:

- Create a test plan (same as slide 26).
- Create a self-checking testbench either by calculating the correct result as done in the class or creating a golden reference in the testbench to check the functionality.
- Task for checking the normal operation of the design.
- Task for checking the reset functionality.
- Counters to keep track of the correct and error count. Display them by the end of your testbench.
- 100% design code coverage. Less than 100% must be justified.

1) ALU design will be provided and has the following characteristics.

1. Reset which resets C to 0.
2. 4-bit signed inputs, A and B
3. 5-bit registered signed output C
4. 4 op-codes
   - add
   - sub (A-B)
   - bitwise invert input A
   - reduction OR input B

Assume the following encoding of the opcodes.

Opcode	Encoding
Add	2'b00
Sub	2'b01
bitwise invert input A	2'b10
reduction OR input B	2'b11

VERIFICATION PLAN

Test Plan:

- **Write MAXPOS & MAXNEG Values for Inputs (A, B).**
- **Write Opcode Encoding.**
- **Write two integer counters (error count) & (correct count).**
- **Inputs and Outputs Instantiated ALU.**
- **Clock Generation.**
- **Initialize counters to zero.**
- **Check Reset Values using (assert reset) Task.**
- **Apply All Permutations of Opcode & A & B to the ALU.**
- **Apply All Permutations Using MAXPOS & MAXNEG for A and B.**
- **Set B to non - all 0's and non - all 1's. Apply all 0 and all 1's to bitwise invert input A for Not_A Opcode.**
- **Apply 0's, all 1's, and Walking 1's to B for ReductionOR_B Opcode.**
- **Set Operands and Walk Through the Opcodes.**
- **Apply inputs so that C is all 1's. Reset.**
- **After Reset is Released and a Positive Clock Edge Occurs, Check the Expected Value Again.**
- **In Each case do Check the ALU Output using (check result) Task.**
- **Check Reset Values using (check reset) Task.**
- **Toggling Achieved for All Interfaces (A, B, Opcode, reset, clk, C).**

Technique for Testing Feature:

- **Directed Testing.**
- **Self - Checking.**
- **Verification IP.**
- **Block Level.**

ALU RTL Design

System 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

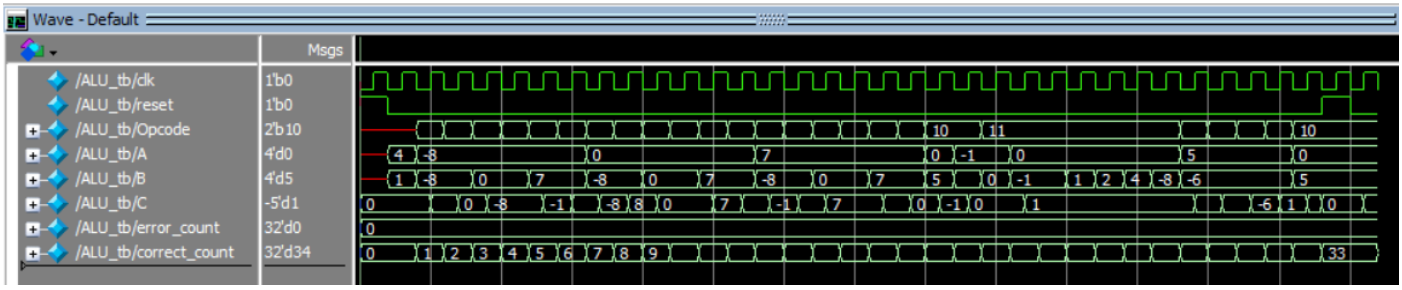
```
module ALU_tb ();

localparam  INPUT_WIDTH  =  4,
        MAXPOS  =  (2 ** (INPUT_WIDTH − 1)) − 1,
        MAXNEG  =  −(2 ** (INPUT_WIDTH − 1)),
        ZERO  =  0;

localparam  Add  =  2'b00,
        Sub  =  2'b01,
        Not_A  =  2'b10,
        ReductionOR_B  =  2'b11;

// Inputs to Instantiated ALU
reg clk, reset;              // Input clk, reset
reg [1: 0] Opcode;           // Input Opcode
reg signed [INPUT_WIDTH − 1: 0] A;  // Input data A: 2's Complement
reg signed [INPUT_WIDTH − 1: 0] B;  // Input data B: 2's Complement
wire signed [INPUT_WIDTH: 0] C;   // AlU Output: 2's Complement

integer error_count;   // 32  −  Bit Signed
integer correct_count; // 32  −  Bit Signed

ALU_4_bit DUT (.*);

// Clock Generation
initial begin
   clk  =  0;
   forever
      #1 clk  =  ~clk;
end

initial begin
   error_count  =  0;
   correct_count  =  0;

   assert_reset;

   // Check the Default Operation Where Not Opcode
   A  =  $random; B  =  $random;
   check_result(Opcode, A, B, 0);
```

```
// Apply All Permutations of MAXPOS, MAXNEG, and ZERO to each Opcode

// Opcode Add, Sub;
A = MAXNEG; B = MAXNEG;
Opcode = Add;
check_result(Opcode, A, B, A + B);
Opcode = Sub;
check_result(Opcode, A, B, A − B);

A = MAXNEG; B = ZERO;
Opcode = Add;
check_result(Opcode, A, B, A + B);
Opcode = Sub;
check_result(Opcode, A, B, A − B);

A = MAXNEG; B = MAXPOS;
Opcode = Add;
check_result(Opcode, A, B, A + B);
Opcode = Sub;
check_result(Opcode, A, B, A − B);

A = ZERO; B = MAXNEG;
Opcode = Add;
check_result(Opcode, A, B, A + B);
Opcode = Sub;
check_result(Opcode, A, B, A − B);

A = ZERO; B = ZERO;
Opcode = Add;
check_result(Opcode, A, B, A + B);
Opcode = Sub;
check_result(Opcode, A, B, A − B);

A = ZERO; B = MAXPOS;
Opcode = Add;
check_result(Opcode, A, B, A + B);
Opcode = Sub;
check_result(Opcode, A, B, A − B);

A = MAXPOS; B = MAXNEG;
Opcode = Add;
check_result(Opcode, A, B, A + B);
Opcode = Sub;
check_result(Opcode, A, B, A − B);
```

```
A = MAXPOS; B = ZERO;
Opcode = Add;
check_result(Opcode, A, B, A + B);
Opcode = Sub;
check_result(Opcode, A, B, A − B);
A = MAXPOS; B = MAXPOS;
Opcode = Add;
check_result(Opcode, A, B, A + B);
Opcode = Sub;
check_result(Opcode, A, B, A − B);
// Opcode Not_A;
// Set B to non − all 0′s and non
        − all 1′s. Apply all 0 and all 1′s to bitwise invert input A for Not_A Opcode
Opcode = Not_A;
A = 4′b0000; B = 4′b0101;
check_result(Opcode, A, B, 5′b11111);
A = 4′b1111; B = 4′b1010;
check_result(Opcode, A, B, 5′b00000);
// Apply 0′s, all 1′s, and Walking 1′s to B for ReductionOR_B Opcode.
Opcode = ReductionOR_B;
A = 4′b1111; B = 4′b0000;
check_result(Opcode, A, B, 0);
A = 4′b0000; B = 4′b1111;
check_result(Opcode, A, B, 1);
A = 4′b0000; B = 4′b1111;
check_result(Opcode, A, B, 1);
A = 4′b0000; B = 4′b0001;
check_result(Opcode, A, B, 1);
A = 4′b0000; B = 4′b0010;
check_result(Opcode, A, B, 1);
A = 4′b0000; B = 4′b0100;
check_result(Opcode, A, B, 1);
A = 4′b0000; B = 4′b1000;
check_result(Opcode, A, B, 1);
// Set Operands and Walk Through the Opcodes
A = 4′b0101; // A = 5
B = 4′b1010; // B = −6
Opcode = Add;
check_result(Opcode, A, B, A + B);
Opcode = Sub;
check_result(Opcode, A, B, A − B);
Opcode = Not_A;
check_result(Opcode, A, B, 5′b11010);
```

```
    Opcode = ReductionOR_B;
    check_result(Opcode, A, B, 1);
    // Apply inputs so that C is all 1's. Reset
    Opcode = Not_A;
    A = 4'b0000; B = 4'b0101;
    check_result(Opcode, A, B, 5'b11111);
    assert_reset;
    // After Reset is Released and a Positive Clock Edge Occurs, Check the Expected Value Again.
    check_result(Opcode, A, B, 5'b11111);

    $display("%0t: At End of test error counter = %0d and
                                correct counter = %0d", $time, error_count, correct_count);
    $stop;
end
task check_result (input [1: 0] Opcode, input signed [3: 0] A, B,
                                                    input signed [4: 0] expected_result);
    @(negedge clk);
    if (expected_result !== C) begin
        error_count = error_count + 1;
        $display("%0t: Error: For Opcode = %0b A = %0d, and
                                B = %0d and C should be equal to %0d but it is equal %0d",
                                                    $time, Opcode, A, B, expected_result, C);
    end
    else
        correct_count = correct_count + 1;
endtask
task checkC (input reset, input signed [4: 0] expected_C);
    @(negedge clk);
    if (expected_C !== C) begin
        error_count = error_count + 1;
        $display("%0t: Error:
                reset = %0d, C should be equal to %0d but the calculated result is %0d",
                                                    $time, reset, expected_C, C);
    end
endtask
task assert_reset();
    // Assert reset
    reset = 1;
    // Check Reset Value of C
    checkC(reset, 5'b0);
    // desert reset
    reset = 0;
endtask
endmodule
```

Waveform



Do File for (ALU) Question 1

```
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.txt
```

```
# 72: At End of test error counter = 0 and correct counter = 34
```

Code Coverage

ALU_Coverage_Rpt.txt

```
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-----------------------------------
    21                                       32     Count coming in to CASE
    22             1                         10             Add:            Alu_out = A + B;

    23             1                         10             Sub:            Alu_out = A - B;

    24             1                          4             Not_A:          Alu_out = ~A;

    25             1                          7             ReductionOR_B:  Alu_out = |B;

    26             1                          1            default:  Alu_out = 5'b0;

Branch totals: 5 hits of 5 branches = 100.00%
```

```
33  ------------------------------------IF Branch------------------------------------
34      32                                      34     Count coming in to IF
35      32              1                        4           if (reset)
36
37      34              1                       30           else
38
39  Branch totals: 2 hits of 2 branches = 100.00%
40
41
42  Statement Coverage:
43      Enabled Coverage              Bins      Hits    Misses  Coverage
44      ----------------              ----      ----    ------  --------
45      Statements                       9         9         0   100.00%
46
47  ================================Statement Details================================
48  -
49  Statement Coverage for instance /\ALU_tb#DUT  --
50
51      Line         Item                     Count     Source
52      ----         ----                     -----     ------
53    File ALU.v
54      1                                              module ALU_4_bit (
55
56      2                                                  input  clk,
57
58      3                                                  input  reset,
59
60      4                                                  input  [1:0] Opcode,   // The opcode
61
62      5                                                  input  signed [3:0] A, // Input data A in 2's complement
63
64      6                                                  input  signed [3:0] B, // Input data B in 2's complement
65
```

```
65
66      7
67
68      8                                                output reg signed [4:0] C // ALU output in 2's complement
69
70      9
71
72      10                                                 );
73
74      11
75
76      12                                               reg signed [4:0] Alu_out; // ALU output in 2's complement
77
78      13
79
80      14                                               localparam Add = 2'b00;           // A + B
81
82      15                                               localparam Sub = 2'b01;           // A - B
83
84      16                                               localparam Not_A = 2'b10;         // ~A
85
86      17                                               localparam ReductionOR_B = 2'b11; // |B
87
88      18
```

```
 89
 90      19                                          // Do the operation
 91
 92      20              1                   32      always @(*) begin
 93
 94      21                                             case (Opcode)
 95
 96      22              1                   10                Add:           Alu_out = A + B;
 97
 98      23              1                   10                Sub:           Alu_out = A - B;
 99
100      24              1                    4                Not_A:         Alu_out = ~A;
101
102      25              1                    7                ReductionOR_B: Alu_out = |B;
103
104      26              1                    1               default:  Alu_out = 5'b0;
105
106      27                                             endcase
107
108      28                                          end // always @ (*)
109
110      29
111
112      30                                          // Register output C
113
114      31              1                   34      always @(posedge clk or posedge reset) begin
115
116      32                                             if (reset)
117
118      33              1                    4            C <= 5'b0;
119
120      34                                             else
121
122      35              1                   30            C<= Alu_out;
123
124
125   Toggle Coverage:
126      Enabled Coverage             Bins      Hits    Misses  Coverage
127      ----------------             ----      ----    ------  --------
128      Toggles                        44        44         0   100.00%
129
```

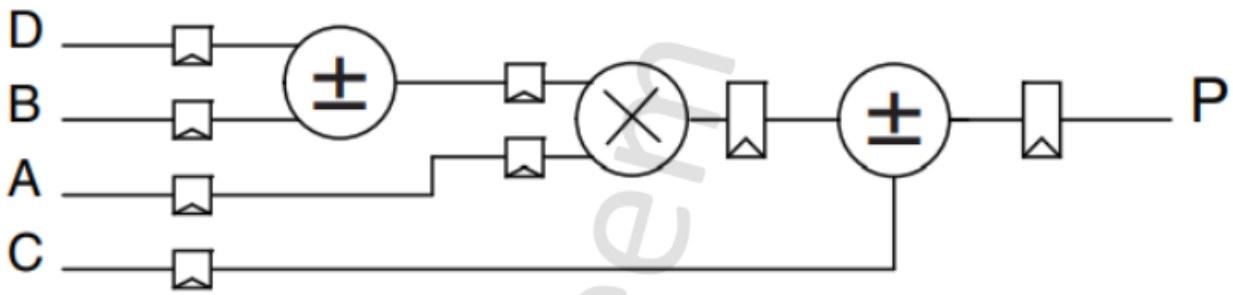
```
130   ================================Toggle Details================================
131
132   Toggle Coverage for instance /\ALU_tb#DUT  --
133
134                                              Node      1H->0L      0L->1H  "Coverage"
135                                              ----------------------------------------
136                                          A[0-3]           1           1      100.00
137                                     Alu_out[4-0]          1           1      100.00
138                                          B[0-3]           1           1      100.00
139                                          C[4-0]           1           1      100.00
140                                       Opcode[0-1]         1           1      100.00
141                                             clk           1           1      100.00
142                                           reset           1           1      100.00
143
144   Total Node Count     =         22
145   Toggled Node Count   =         22
146   Untoggled Node Count =          0
147
148   Toggle Coverage      =     100.00% (44 of 44 bins)
149
150
151   Total Coverage By Instance (filtered view): 100.00%
152
153
```

Question 2

DSP

2) Verify the functionality of the following simplified version of the DSP block DSP48A1. All the registers are positive-edge triggered with the clock and have an async active low reset. Assume that we will only use the DSP in the addition mode only. For simplicity, you can drive the DSP inputs every 4 clock cycles to check the P output.

Note: it is an unsigned DSP block.



Port	Type	Width
A	Input	18
B	Input	18
C	Input	48
D	Input	18
Clk	Input	1
rst_n (async active low)	Input	1
P	Output	48

Parameters:

1. OPERATION: take 2 values either “ADD” or “SUBTRACT”, Default value “ADD”
    - When subtracting use “D - B” and “multiplier_out – C”. multiplier_out is an internal signal

VERIFICATION PLAN

Test Plan:

- **Write for A, B, C, D Constant Large Numbers To act as a Max. Limit.**
- **Write two integer counters (error count) & (correct count).**
- **Clock Generation.**
- **Initialize counters to zero.**
- **Check Reset Values using (check reset) Task.**
- **Test All Inputs Are Zeros.**
- **Test A Bus is ones and B, C, D are Zeros.**
- **Test B Bus is ones and A, C, D are Zeros.**
- **Test C Bus is ones and A, B, D are Zeros.**
- **Test D Bus is ones and A, B, C are Zeros.**
- **Test Combinations between A, B, C, D Max. Limit and ZERO for Inputs.**
- **Randomize for 100 Iteration for different input values ranged between ZERO and Max. Limit**
- **In Each case do Check the ALU Output using (check result) Task.**
- **Check Reset Values using (check reset) Task.**
- **Toggling Achieved for All Interfaces (A, B, C, D, P, clk, rst_n).**

Technique for Testing Feature:

- **Directed and Random Testing.**
- **Self - Checking.**
- **Verification IP.**
- **Block Level.**

System Verilog Design Code:

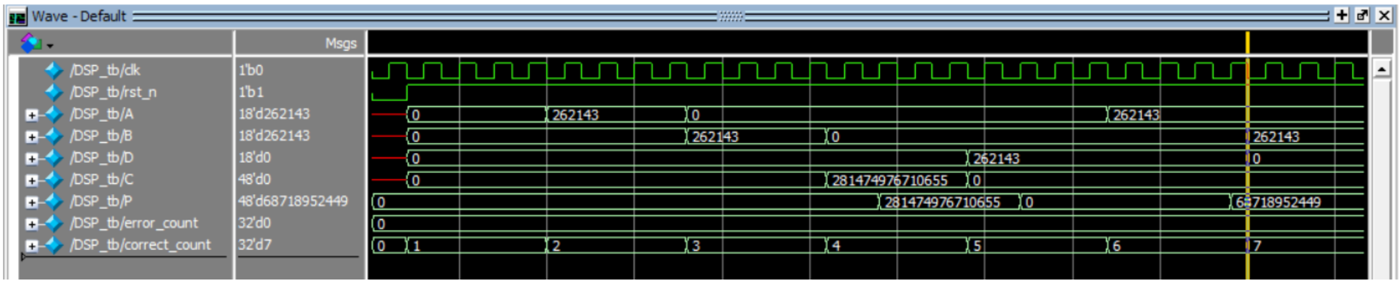
```
module DSP(A, B, C, D, clk, rst_n, P);

parameter OPERATION = "ADD";
input [17:0] A, B, D;
input [47:0] C;
input clk, rst_n;
output reg [47:0] P;
reg [17:0] A_reg_stg1, A_reg_stg2, B_reg, D_reg, adder_out_stg1 ;
reg [47:0] C_reg, mult_out, adder_out_stg2;

always @(posedge clk or negedge rst_n) begin
  if (!rst_n) begin
    // reset
    A_reg_stg1 <= 0 ;
    A_reg_stg2 <= 0 ;
    B_reg <= 0 ;
    D_reg <= 0 ;
    C_reg <= 0 ;
    adder_out_stg1 <= 0 ;
    adder_out_stg2 <= 0 ;
    mult_out <= 0 ;
    P <= 0 ;
  end
  else begin
    A_reg_stg1 <= A;
    A_reg_stg2 <= A_reg_stg1;
    B_reg <= B;
    C_reg <= C;
    D_reg <= D;
    if (OPERATION == "ADD") begin
      adder_out_stg1 <= D_reg + B_reg ;
      adder_out_stg2 <= mult_out + C_reg ;
      P <= mult_out + C_reg ;
    end
    else if (OPERATION == "SUBTRACT") begin
      adder_out_stg1 <= D_reg - B_reg ;
      adder_out_stg2 <= mult_out - C_reg ;
      P <= mult_out - C_reg ;
    end
    mult_out <= A_reg_stg2 * adder_out_stg1 ;
  end
end
endmodule
```

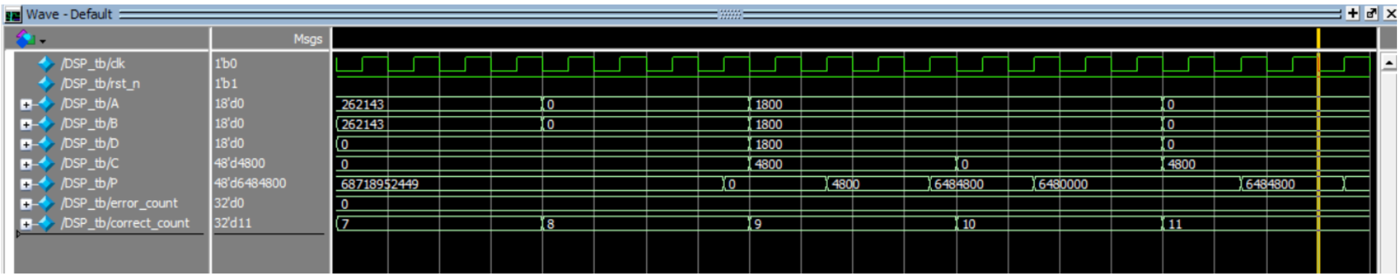
DSP Testbench

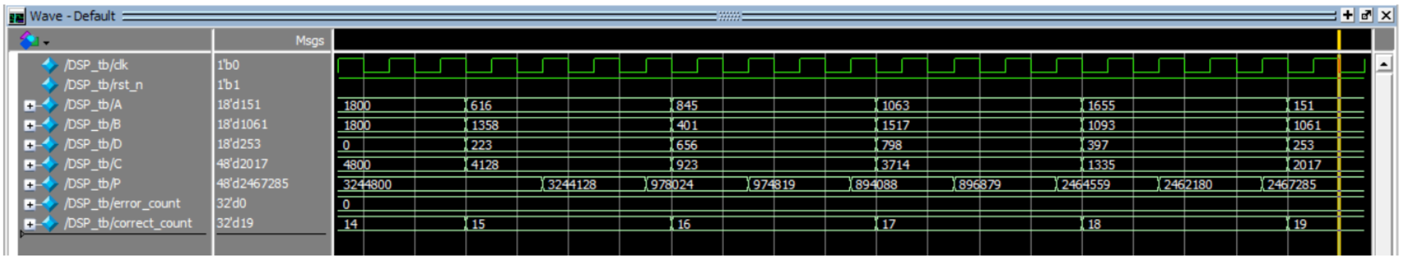
```
module DSP_tb ();

parameter OPERATION = "ADD";

localparam ABD_INPUT_WIDTH = 18;
localparam ABD_Const_No = 100 * ABD_INPUT_WIDTH;
localparam C_INPUT_WIDTH = 48;
localparam C_Const_No = 100 * C_INPUT_WIDTH;
localparam ZERO = 0;

reg [ABD_INPUT_WIDTH - 1:0] A, B, D;  // Input A, B, D
reg [C_INPUT_WIDTH - 1:0] C;        // Input C
reg clk, rst_n;                 // Input clk, reset
wire [47:0] P;                   // Output P

// Clock Generation
initial begin
  clk = 0;
  forever
    #1 clk = ~clk;
end

DSP #(.OPERATION(OPERATION)) DUT (A, B, C, D, clk, rst_n, P);

integer error_count;   // 32 - Bit Signed
integer correct_count; // 32 - Bit Signed

initial begin
  error_count = 0;
  correct_count = 0;

  check_reset;

  A = ZERO; B = ZERO; C = ZERO; D = ZERO; check_result(((D + B) * A) + C);
  A = {18{1'b1}}; B = ZERO; C = ZERO; D = ZERO; check_result(((D + B) * A) + C);
  A = ZERO; B = {18{1'b1}}; C = ZERO; D = ZERO; check_result(((D + B) * A) + C);
  A = ZERO; B = ZERO; C = {48{1'b1}}; D = ZERO; check_result(((D + B) * A) + C);
  A = ZERO; B = ZERO; C = ZERO; D = {18{1'b1}}; check_result(((D + B) * A) + C);
  A = {18{1'b1}}; B = ZERO; C = ZERO; D = {18{1'b1}}; check_result(((D + B) * A) + C);
  A = {18{1'b1}}; B = {18{1'b1}}; C = ZERO; D = ZERO; check_result(((D + B) * A) + C);
  A = ZERO; B = ZERO; C = ZERO; D = ZERO; check_result(((D + B) * A) + C);
// Toggle Acheived for all Input Bits
```

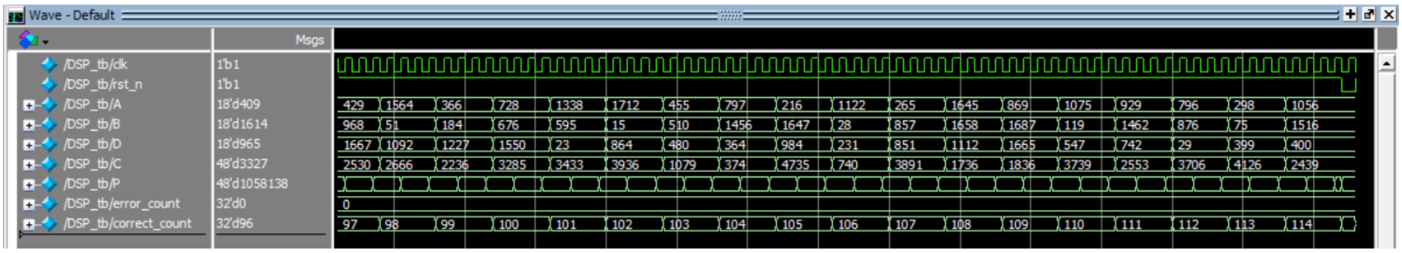
```
  // To Acheive Toggle for all Output Bits Do Alot of Test Cases
  A = ABD_Const_No; B = ABD_Const_No; C = C_Const_No; D
            = ABD_Const_No; check_result(((D + B) * A) + C);
  A = ABD_Const_No; B = ABD_Const_No; C = ZERO; D
            = ABD_Const_No; check_result(((D + B) * A) + C);
  A = ZERO; B = ZERO; C = C_Const_No; D = ZERO; check_result(((D + B) * A) + C);
  A = ZERO; B = ABD_Const_No; C = C_Const_No; D
            = ABD_Const_No; check_result(((D + B) * A) + C);
  A = ABD_Const_No; B = ZERO; C = C_Const_No; D
            = ABD_Const_No; check_result(((D + B) * A) + C);
  A = ABD_Const_No; B = ABD_Const_No; C = C_Const_No; D
            = ZERO; check_result(((D + B) * A) + C);
  for (int i = 0 ; i < 100 ; i + +) begin
    A = $urandom_range(ZERO, ABD_Const_No);
    B = $urandom_range(ZERO, ABD_Const_No);
    C = $urandom_range(ZERO, C_Const_No);
    D = $urandom_range(ZERO, ABD_Const_No);
    check_result(((D + B) * A) + C);
  end

  // Toggle Closure for Reset, Reset Again?  maybe repeated code?
  check_reset;

  $display("%0t: At End of test error counter = %0d and correct counter
            = %0d", $time, error_count, correct_count);
  $stop;
End
task check_result (input [47: 0] expected_result);
  repeat(4) @(negedge clk);
  if (P ! == expected_result) begin
    error_count = error_count + 1;
    $display("%0t: Error: For A = %0d, B = %0d, C = %0d,
   D = %0d and P should be equal to %0d but it is equal %0d", $time, A, B, C, D, expected_result, P);
  end
  else
    correct_count = correct_count + 1;
endtask
task check_reset();
  // Assert reset
  rst_n = 0;
  @(negedge clk);
  if (P ! = 0) begin
    error_count = error_count + 1;
    $display("%0t: Error: Reset Value is Asserted and the Output is Not Tied to Low", $time);
  end
  else
    correct_count = correct_count + 1;
  // desert reset
  rst_n = 1;
endtask
endmodule
```

Waveform









Do File for (DSP) Question 2

```
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.txt
```

```
# 916: At End of test error counter = 0 and correct counter = 116
```

Code Coverage

DSP_Coverage_Rpt.txt

```
Coverage Report by instance with details

================================================================================
=== Instance: /\DSP_tb#DUT
=== Design Unit: work.DSP
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         2         2         0   100.00%

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

Branch Coverage for instance /\DSP_tb#DUT

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File DSP.v
------------------------------------IF Branch------------------------------------
    14                                       447     Count coming in to IF
    14             1                           4        if (!rst_n) begin

    26             1                         443        else begin

Branch totals: 2 hits of 2 branches = 100.00%


Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      19        19         0   100.00%

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

Statement Coverage for instance /\DSP_tb#DUT  --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File DSP.v
    1                                                module DSP(A, B, C, D, clk, rst_n, P);

    2

    3                                                parameter OPERATION = "ADD";

    4

    5                                                input  [17:0] A, B, D;

    6                                                input  [47:0] C;

    7                                                input clk, rst_n;

    8                                                output reg  [47:0] P;

    9

    10                                               reg  [17:0] A_reg_stg1, A_reg_stg2, B_reg, D_reg, adder_out_stg1 ;

    11                                               reg  [47:0] C_reg, mult_out, adder_out_stg2;
```

```
63
64        13              1                     447     always @(posedge clk or negedge rst_n) begin
65
66        14                                              if (!rst_n) begin
67
68        15                                                  // reset
69
70        16              1                       4           A_reg_stg1 <= 0 ;
71
72        17              1                       4           A_reg_stg2 <= 0 ;
73
74        18              1                       4           B_reg <= 0 ;
75
76        19              1                       4           D_reg <= 0 ;
77
78        20              1                       4           C_reg <= 0 ;
79
80        21              1                       4           adder_out_stg1 <= 0 ;
81
82        22              1                       4           adder_out_stg2 <= 0 ;
83
84        23              1                       4           mult_out <= 0 ;
85
86        24              1                       4           P <= 0 ;
87
88        25                                              end
89
90        26                                              else begin
91
92        27              1                     443           A_reg_stg1 <= A;
93
94        28              1                     443           A_reg_stg2 <= A_reg_stg1;
95
96        29              1                     443           B_reg <= B;
```

```
 98        30              1                     443           C_reg <= C;
 99
100        31              1                     443           D_reg <= D;
101
102        32                                                  if (OPERATION == "ADD") begin
103
104        33              1                     443               adder_out_stg1 <= D_reg + B_reg ;
105
106        34              1                     443               adder_out_stg2 <= mult_out + C_reg ;
107
108        35              1                     443               P <= mult_out + C_reg ;
109
110        36                                                  end
111
112        37                                                  else if (OPERATION == "SUBTRACT") begin
113
114        38                                                      adder_out_stg1 <= D_reg - B_reg ;
115
116        39                                                      adder_out_stg2 <= mult_out - C_reg ;
117
118        40                                                      P <= mult_out - C_reg ;
119
120        41                                                  end
121
122        42              1                     443           mult_out <= A_reg_stg2 * adder_out_stg1 ;
123
124
125    Toggle Coverage:
126        Enabled Coverage              Bins      Hits    Misses  Coverage
127        ----------------              ----      ----    ------  --------
128        Toggles                        772       748        24    96.89%
```

```
130   ================================Toggle Details================================
131
132   Toggle Coverage for instance /\DSP_tb#DUT  --
133
134                                              Node      1H->0L      0L->1H  "Coverage"
135                                              ----------------------------------------
136                                          A[0-17]           1           1      100.00
137                                 A_reg_stg1[17-0]           1           1      100.00
138                                 A_reg_stg2[17-0]           1           1      100.00
139                                         B[0-17]           1           1      100.00
140                                      B_reg[17-0]           1           1      100.00
141                                         C[0-47]           1           1      100.00
142                                      C_reg[47-0]           1           1      100.00
143                                         D[0-17]           1           1      100.00
144                                      D_reg[17-0]           1           1      100.00
145                                         P[47-0]           1           1      100.00
146                               adder_out_stg1[17-0]           1           1      100.00
147                               adder_out_stg2[47-0]           1           1      100.00
148                                             clk           1           1      100.00
149                                   mult_out[47-36]           0           0        0.00
150                                    mult_out[35-0]           1           1      100.00
151                                             rst_n           1           1      100.00
152
153   Total Node Count     =        386
154   Toggled Node Count   =        374
155   Untoggled Node Count =         12
156
157   Toggle Coverage      =      96.89% (748 of 772 bins)
158
159
160   Total Coverage By Instance (filtered view): 98.96%
```

Toggle Coverage Can't Reach 100% for Mult_out[47 : 36]

Justification:

As mult_out $= (D + B) * A$, From this Equation to fill out all 48 bits mult_out we need (D, B, A) driven with Maximum Limit Value and that is Impossible as From Multiplication Rule any factors multiply each other the result of multiplication equal the sum of their size here:

$(D + B) = 18$ bits & $A = 18$ bits.

So, Mult_out $= 18 + 18 = 36$ bits.

Therefore Mult_out[47:0] fill only Mult_out[35:0] and Mult_out[47:36] will be undriven so it is impossible to toggle.