# Assignment 1

Digital Design Verification

## Contents

# 1 Priority Encoder

## 1.1 1. Design

```
// Bug fix: Compilation error due to ’Y’ not being defined as a reg type      resolved by adding ’reg’ to the Y declaration.
// Bug fix: ’valid’ output was not reset to zero when synchronous reset was asserted      resolved by adding a reset condition for ’valid’ in the always block.
module priority_enc (
  input  clk,
  input  rst,
  input  [3:0] D,
  output reg [1:0] Y,  // Defined as ’reg’ to hold value in always block
  output reg valid     // Ensured proper reset behavior
);

always @(posedge clk) begin
  if (rst) begin
     Y <= 2’b0;         // Reset Y to 0 on reset
     valid <= 1’b0;     // Reset valid to 0 on reset
  end else begin
    casex (D)
      4’b1000: Y <= 0;
      4’bX100: Y <= 1;
      4’bXX10: Y <= 2;
      4’bXXX1: Y <= 3;
    endcase
    valid <= (~|D) ? 1’b0 : 1’b1; // Set valid based on D input
  end
end

endmodule
```

## 1.2 2. Verification Plan

| Label | Design Re-quirement Description | Stimulus Generation | Functional Coverage | Functionality Check |
|---|---|---|---|---|
| RESET_TEST | When reset is asserted, out-put Y should be 00 and valid should be 0 | Directed at the start of the simulation | - | A checker in the testbench verifies Y = 00 and valid = 0 |
| ALL_ZERO | When D is 0000, output Y should be un-defined (xx) and valid should be 0 | Directed test with D = 0000 | - | A checker in the testbench verifies Y = xx and valid = 0 |
| PRIORITY_0 | When D is 1000, Y should be 00 and valid should be 1 | Directed test with D = 1000 | - | A checker in the testbench verifies Y = 00 and valid = 1 |
| PRIORITY_1 | When the high-est priority bit is at 01 (D = 1100 or 0100), Y should be 01 | Directed tests with D = 1100, 0100 | - | A checker in the testbench verifies Y = 01 and valid = 1 |
| PRIORITY_2 | When the high-est priority bit is at 10 (D = 0010, 0110, etc.), Y should be 10 | Directed tests with D = 0010, 0110... | - | A checker in the testbench verifies Y = 10 and valid = 1 |
| PRIORITY_3 | When the high-est priority bit is at 11 (D = 0001, 0011, etc.), Y should be 11 | Directed tests with D = 0001, 0011... | - | A checker in the testbench verifies Y = 11 and valid = 1 |

Table 1: Verification Plan

## 1.3 3. Testbench

```
‘timescale 1ns/1ps
module priority_enc_tb();

    //------------------------------------
    //------- TB signal declaration ------
    //------------------------------------
    logic  clk;
    logic  rst;       // note : active high synchronous reset
    logic  [3:0] D;
    logic  [1:0] Y;
    logic  valid;

    int error_count = 0; // Error count variable

    //------------------------------------
    //--------- instantiate DUT ----------
    //------------------------------------
    priority_enc DUT (
        .clk(clk),
        .rst(rst),
        .D(D),
        .Y(Y),
        .valid(valid)
    );

    //------------------------------------
    //-------- generate clock ------------
    //------------------------------------
    localparam CLOCK_PERIOD = 10;
    always begin
       #(CLOCK_PERIOD/2) clk = ~clk;
    end

    //-----------------------------------
    //----------- reset task ------------
    //-----------------------------------
    task check_rst(input [1:0] expected_output);
    begin
        @(negedge clk);
        if(Y != expected_output || valid != 1’b0) begin    // using -- or not and --- as no one from two checking condition not be violated
            error_count++;
            $display("Reset␣test␣failed:␣rst=%0b,␣D=%0b,␣Y=%0b,␣valid=%0b,␣error␣count=%0d,␣time=%0t", rst, D, Y, valid, error_count, $time);
            $stop;
        end else begin
```

```
            $display("Reset␣test␣passed:␣rst=%0b,␣D=%0b,␣Y=%0b,␣valid=%0b,␣time=%0t", rst, D, Y, valid, $time);
            // wait 5ns then toggle reset
            #(CLOCK_PERIOD/2) rst = 0;
        end
    end
    endtask

    //-------------------------------------
    //----------- all zero task ------------
    //-------------------------------------
    task all_zero(input [1:0] expected_output);
    begin
    @(negedge clk);
        if(Y != expected_output || valid != 1'b0) begin
            error_count++;
            $display("all_zero␣test␣failed:␣rst=%0b,␣D=%0b,␣Y=%0b,␣valid=%0b,␣error␣count=%0d,␣time=%0t", rst, D, Y, valid, error_count,
                $time);
            $stop;
        end else begin
            $display("all_zero␣test␣passed:␣rst=%0b,␣D=%0b,␣Y=%0b,␣valid=%0b,␣time=%0t", rst, D, Y, valid, $time);
        #(CLOCK_PERIOD/2);
        end
    end
    endtask


    //----------------------------------
    //----------- check task ------------
    //----------------------------------
    task check_fun(input [1:0] expected_output);
    begin
    @(negedge clk);
        if(Y != expected_output || valid != 1'b1) begin
            error_count++;
            $display("check␣test␣failed:␣rst=%0b,␣D=%0b,␣Y=%0b,␣valid=%0b,␣error␣count=%0d,␣time=%0t", rst, D, Y, valid, error_count,
                $time);
            $stop;
        end else begin
            $display("check␣test␣passed:␣rst=%0b,␣D=%0b,␣Y=%0b,␣valid=%0b,␣time=%0t", rst, D, Y, valid, $time);
        #(CLOCK_PERIOD/2);
        end
    end
    endtask

    //----------------------------------
    //--------- initial clock -----------
    //----------------------------------
    initial begin
        // initial values
        clk = 1;
        rst = 0;
        D = 4'b1111;

        //------------------------
        // test case 1 : test reset
        //------------------------
        #CLOCK_PERIOD;
        rst = 1 ;
        check_rst(2'b00); // expected output is 00

        //------------------------
        // test case 1 : all zeros
        //------------------------
        D = 4'b0000;
        all_zero(2'bxx);

        //------------------------
        // test case 2 : priority 0
        //------------------------
        D = 4'b1000;
        check_fun(2'b00);

        //------------------------
        // test case 3 : priority 1
        //------------------------
        D = 4'b1100;
        check_fun(2'b01);
        D = 4'b0100;
        check_fun(2'b01);

        //------------------------
        // test case 4 : priority 2
        //------------------------
        D = 4'b0010;
        check_fun(2'b10);
        D = 4'b0110;
        check_fun(2'b10);
        D = 4'b1010;
        check_fun(2'b10);
        D = 4'b1110;
        check_fun(2'b10);

        //------------------------
        // test case 5 : priority 3
        //------------------------
        D = 4'b0001;
        check_fun(2'b11);
        D = 4'b0011;
        check_fun(2'b11);
        D = 4'b0101;
        check_fun(2'b11);
        D = 4'b0111;
        check_fun(2'b11);
        D = 4'b1001;
        check_fun(2'b11);
        D = 4'b1011;
        check_fun(2'b11);
        D = 4'b1101;
        check_fun(2'b11);
        D = 4'b1111;
        check_fun(2'b11);

        //-------------------------
        // finish simulation
        //-------------------------
        #CLOCK_PERIOD;
        $finish;
        end

endmodule
```

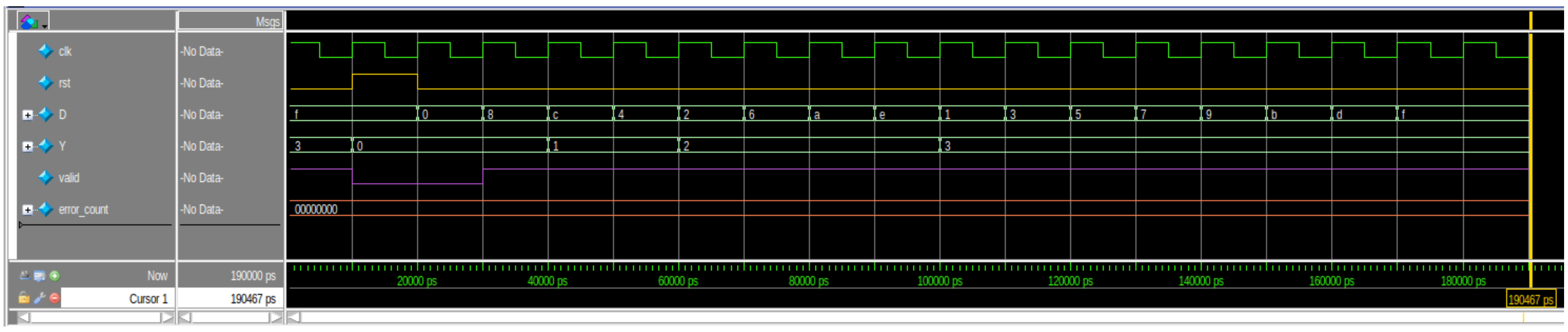
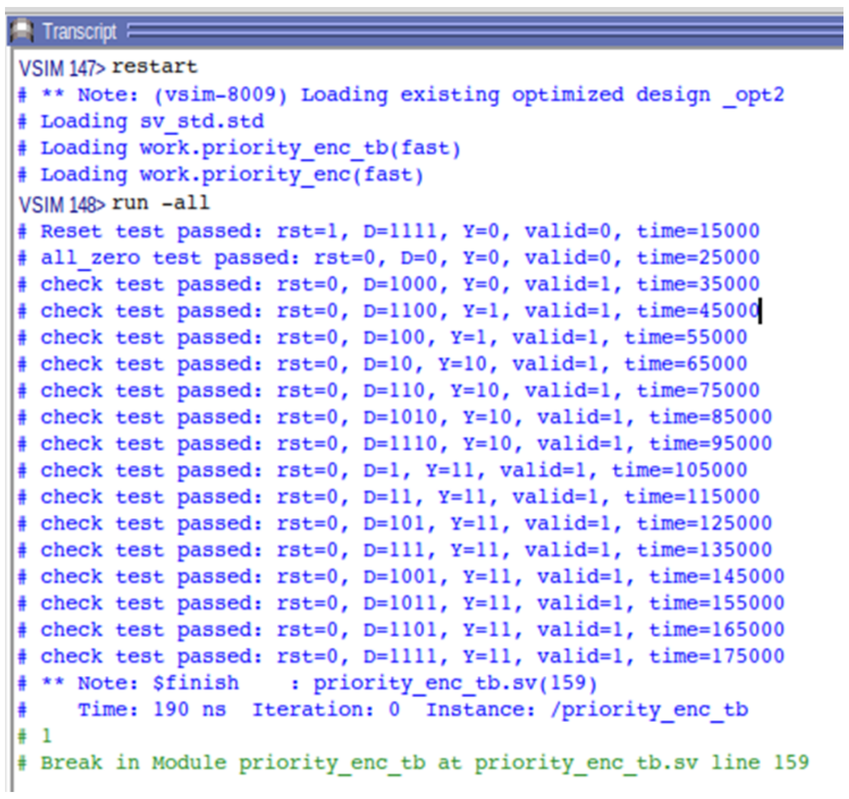


Figure 1: simulation waveform



```
VSIM 147> restart
# ** Note: (vsim-8009) Loading existing optimized design _opt2
# Loading sv_std.std
# Loading work.priority_enc_tb(fast)
# Loading work.priority_enc(fast)
VSIM 148> run -all
# Reset test passed: rst=1, D=1111, Y=0, valid=0, time=15000
# all_zero test passed: rst=0, D=0, Y=0, valid=0, time=25000
# check test passed: rst=0, D=1000, Y=0, valid=1, time=35000
# check test passed: rst=0, D=1100, Y=1, valid=1, time=45000
# check test passed: rst=0, D=100, Y=1, valid=1, time=55000
# check test passed: rst=0, D=10, Y=10, valid=1, time=65000
# check test passed: rst=0, D=110, Y=10, valid=1, time=75000
# check test passed: rst=0, D=1010, Y=10, valid=1, time=85000
# check test passed: rst=0, D=1110, Y=10, valid=1, time=95000
# check test passed: rst=0, D=1, Y=11, valid=1, time=105000
# check test passed: rst=0, D=11, Y=11, valid=1, time=115000
# check test passed: rst=0, D=101, Y=11, valid=1, time=125000
# check test passed: rst=0, D=111, Y=11, valid=1, time=135000
# check test passed: rst=0, D=1001, Y=11, valid=1, time=145000
# check test passed: rst=0, D=1011, Y=11, valid=1, time=155000
# check test passed: rst=0, D=1101, Y=11, valid=1, time=165000
# check test passed: rst=0, D=1111, Y=11, valid=1, time=175000
# ** Note: $finish    : priority_enc_tb.sv(159)
#    Time: 190 ns  Iteration: 0  Instance: /priority_enc_tb
# 1
# Break in Module priority_enc_tb at priority_enc_tb.sv line 159
```

Figure 2: Transcript : all test cases passed

## 1.4 4. Do File

```
vlib work
vlog priority_enc.sv priority_enc_tb.sv +cover -covercells
vsim -voptargs=+acc work.priority_enc_tb -cover
add wave *
coverage save priority_enc_tb.ucdb -du priority_enc -onexit
run -all
```

## 1.5 5. Coverage Report

```
Coverage Report by instance with details

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

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

Branch Coverage for instance /\priority_enc_tb#DUT

    Line         Item                      Count     Source
    ----          ----                            -----      ------
  File priority_enc.sv
------------------------------------IF Branch------------------------------------
    10                                     18      Count coming in to IF
    10              1                       1        if (rst) begin
    13              1                      17        end else begin


Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------CASE Branch------------------------------------
    14                                     17      Count coming in to CASE
    15              1                       1                4'b1000: Y <= 0;
    16              1                       2                4'bX100: Y <= 1;
    17              1                       4                4'bXX10: Y <= 2;
    18              1                       9                4'bXXX1: Y <= 3;

                                            1      All False Count
Branch totals: 5 hits of 5 branches = 100.00%


Statement Coverage:
    Enabled Coverage              Bins        Hits      Misses   Coverage
    ----------------              ----        ----      ------   --------
    Statements                       8           8           0    100.00%

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

Statement Coverage for instance /\priority_enc_tb#DUT --

    Line          Item                        Count      Source
    ----          ----                        -----      ------
  File priority_enc.sv
    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               1                          18        always @(posedge clk) begin
    10                                                     if (rst) begin
    11              1                           1             Y <= 2'b0;
    12              1                           1            valid <= 1'b0;
    13                                                     end else begin
    14                                                      casex (D)
    15              1                           1                4'b1000: Y <= 0;
    16              1                           2                4'bX100: Y <= 1;
    17              1                           4                4'bXX10: Y <= 2;
    18              1                           9                4'bXXX1: Y <= 3;
    19                                                      endcase
    20              1                          17           valid <= (~|D)? 1'b0: 1'b1;



Toggle Coverage:
    Enabled Coverage              Bins        Hits      Misses   Coverage
    ----------------              ----        ----      ------   --------
    Toggles                         18          18           0    100.00%

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

Toggle Coverage for instance /\priority_enc_tb#DUT --

                                               Node       1H->0L       0L->1H   "Coverage"
                                           ---------------------------------------------
                                             D[0-3]            1            1       100.00
                                             Y[1-0]            1            1       100.00
                                                clk            1            1       100.00
                                                rst            1            1       100.00
                                              valid            1            1       100.00

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

Toggle Coverage      =     100.00% (18 of 18 bins)


Total Coverage By Instance (filtered view): 100.00%
```

## 2 ALU 4-bit

### 2.1 1. Design

```
// no bugs
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
```

## 2.2 2. Verification Plan

| **Label** | **Design Requirement Description** | **Stimulus Generation** | **Functional Coverage** | **Functionality Check** |
|---|---|---|---|---|
| RESET_TEST | When the reset is asserted, the ALU's output (C) must initialize to 0 | Drive reset high for one clock cycle (with defined A, B, Opcode) then deassert it; check output immediately after clock edge | Reset behavior; Initialization | Compare C to expected value (5'b0) after reset is applied |
| ADDITION | With Opcode = 00, the ALU shall perform signed addition (A + B) with proper handling of negative values and potential overflow | Set Opcode to 2'b00 and apply multiple input vectors (e.g., 0+0, -1+(-1), 3+1, -5+5) as given in the test cases | Signed arithmetic addition | Verify that output C exactly matches the expected 5-bit result from the golden model |
| SUBTRACTION | With Opcode = 01, the ALU shall perform signed subtraction (A - B) with correct arithmetic results even for negative operands | Set Opcode to 2'b01 and provide test vectors (0-0, -1-(-1), 4-1) to exercise both zero and nonzero differences | Signed arithmetic subtraction | Check that C equals the computed difference (using two's complement math) for each test vector |
| INVERT_A | With Opcode = 10, the ALU shall output the bitwise inversion of A with proper sign extension to produce a 5-bit result | Set Opcode to 2'b10 and apply vectors such as A = 4'b0000, 4'b0011, and 4'b1111; B is provided but not used for the inversion operation | Bitwise inversion and sign extension | Confirm that the output C matches the expected sign-extended inversion |
| REDUCTION_OR_B | With Opcode = 11, the ALU shall perform a reduction OR on B (output 1 if any bit of B is high, otherwise 0), with the result extended to 5 bits | Set Opcode to 2'b11 and stimulate with various values of B (e.g., B = 4'b0000, 4'b1111, 4'b0011) while A can be arbitrary | Reduction logic on B; handling of bitwise reduction operations | Compare the output C with the expected value |
| DEFAULT_CASE | For undefined or don't care conditions (e.g. Opcode = x or inputs are unknown), the ALU shall output 0, ensuring safe behavior | Drive Opcode and input signals (A, B) to 'x' (unknown) values to check that the default case is properly handled, and that C is driven to 0 | Handling of undefined input conditions; safe default outputs | Verify that C is set to 5'b0 when invalid or unknown values are provided |

Table 2: Verification Plan

## 2.3 3. Testbench

```
module ALU_4_bit_tb();

    //-----------------------------------
    //------- TB signal declaration ------
    //-----------------------------------
    logic clk;
    logic reset;
    logic [1:0] Opcode;
    logic signed [3:0] A, B;
    logic signed [4:0] C;

    int error_count = 0; // Error count variable

    //-----------------------------------
    //--------- instantiate DUT ----------
    //-----------------------------------
    ALU_4_bit DUT (
        .clk(clk),
        .reset(reset),
        .Opcode(Opcode),
        .A(A),
        .B(B),
        .C(C)
    );

    //-----------------------------------
    //-------- generate clock ------------
    //-----------------------------------
    localparam CLOCK_PERIOD = 10;
    always begin
       #(CLOCK_PERIOD/2) clk = ~clk;
    end
```

```verilog

    //-----------------------------------
    //----------- reset task ------------
    //-----------------------------------
    task reset_ALU(input [4:0] expected_output);
    begin
        @(posedge clk);
        if(C != expected_output) begin
            error_count++;
            $display("Reset␣test␣failed:␣reset=%0b,␣A=%0b,␣B=%0b,␣C=%0b,␣Opcode=%0b,␣error␣count=%0d,␣Expected=%0d,␣time=%0t", reset, A, B, C, Opcode, error_count, expected_output, $time);
            $stop;
        end else begin
            $display("Reset␣test␣passed:␣reset=%0b,␣A=%0b,␣B=%0b,␣C=%0b,␣Opcode=%0b,␣time=%0t", reset, A, B, C, Opcode, $time);
            #(CLOCK_PERIOD/2) reset = 0;
        end
    end
    endtask

    //-----------------------------------
    //----------- test task ------------
    //-----------------------------------
    task test_ALU(
        input [1:0] op,
        input signed [3:0] a,
        input signed [3:0] b,
        input signed [4:0] expected_output,
        input string operation
    );
    begin
        Opcode = op;
        A = a;
        B = b;
        @(posedge clk);
        #1;
        if (C !== expected_output) begin
            error_count++;
            $display("%s␣Test␣failed:␣reset=%0b,␣A=%0b,␣B=%0b,␣C=%0b,␣Opcode=%0b,␣error␣count=%0d,␣Expected=%0d,␣Time=%0t",
                    operation, reset, A, B, C, Opcode, error_count, expected_output, $time);
            $stop;
        end else begin
            $display("%s␣Test␣passed:␣reset=%0b,␣A=%0b,␣B=%0b,␣C=%0b,␣Opcode=%0b,␣Time=%0t",
                    operation, reset, A, B, C, Opcode, $time);
            #((CLOCK_PERIOD/2)-1);
        end
    end
    endtask

    //-----------------------------------
    //--------- initial block -----------
    //-----------------------------------
    initial begin
        // Initial values
        clk = 0;
        reset = 0;
        Opcode = 2'b01;
        A = 4'b0100;
        B = 4'b0011;

        //-------------------------------------
        // test case 1 : test reset
        //-------------------------------------
        #CLOCK_PERIOD;
        reset = 1 ;
        reset_ALU(5'b0);

        //-------------------------------------
        // test case 2 : test addition
        //-------------------------------------
        test_ALU(2'b00, 4'b0000, 4'b0000, 5'b00000, "Addition"); // Add: 0 + 0 = 0
        test_ALU(2'b00, 4'b1111, 4'b1111, 5'b11110, "Addition"); // Add: -1 + -1 = -2
        test_ALU(2'b00, 4'b0011, 4'b0001, 5'b00100, "Addition"); // Add: 3 + 1

        //-------------------------------------
        // test case 3 : substraction addition
        //-------------------------------------
        test_ALU(2'b01, 4'b0000, 4'b0000, 5'b00000, "Subtraction"); // Sub: 0 - 0 = 0
        test_ALU(2'b01, 4'b1111, 4'b1111, 5'b00000, "Subtraction"); // Sub: -1 - -1 = 0
        test_ALU(2'b01, 4'b0100, 4'b0001, 5'b00011, "Subtraction"); // Sub: 4 - 1 = 3

        //--------------------------------------
        // test case 4 : invert A "sign extend"
        //--------------------------------------
        test_ALU(2'b10, 4'b0000, 4'b1100, 5'b11111, "Invert␣A"); // Not A: ~0 --> 1_1111
        test_ALU(2'b10, 4'b0011, 4'b0000, 5'b11100, "Invert␣A"); // Not A: ~3 --> 1_1100
        test_ALU(2'b10, 4'b1111, 4'b1111, 5'b00000, "Invert␣A"); // Not A: ~-1 --> 0_0000

        //--------------------------------------
        // test case 5 : Reduction OR B
        //--------------------------------------
        test_ALU(2'b11, 4'b1111, 4'b0000, 5'b00000, "Reduction␣OR␣B"); // Reduction OR B: |0
        test_ALU(2'b11, 4'b1001, 4'b1111, 5'b00001, "Reduction␣OR␣B"); // Reduction OR B: |-1
        test_ALU(2'b11, 4'b0000, 4'b0011, 5'b00001, "Reduction␣OR␣B"); // Reduction OR B: |3

        // add this bit only fot toggling op code from 11 to 00
        test_ALU(2'b00, 4'b1011, 4'b0101, 5'b00000, "Addition"); // Add: -5 + 5 = 0

        //------------------------------------------
        // test case 6 : don't cares "default case"
        //------------------------------------------
        test_ALU(2'bxx, 4'bxxxx, 4'bxxxx, 5'b00000, "Default␣Case");  // Add: x + x = 0

        // Finish simulation
        #CLOCK_PERIOD;
        $finish;
    end

endmodule
```

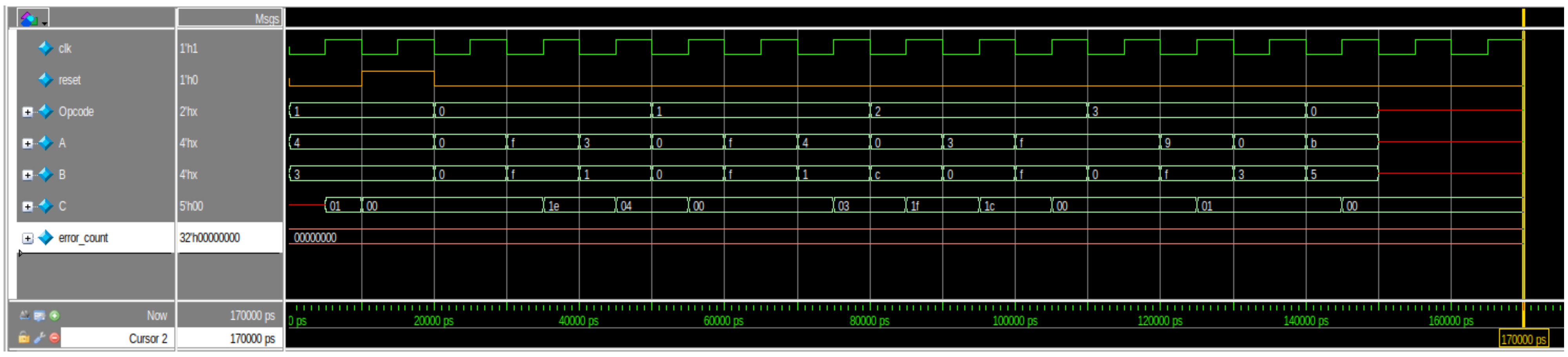
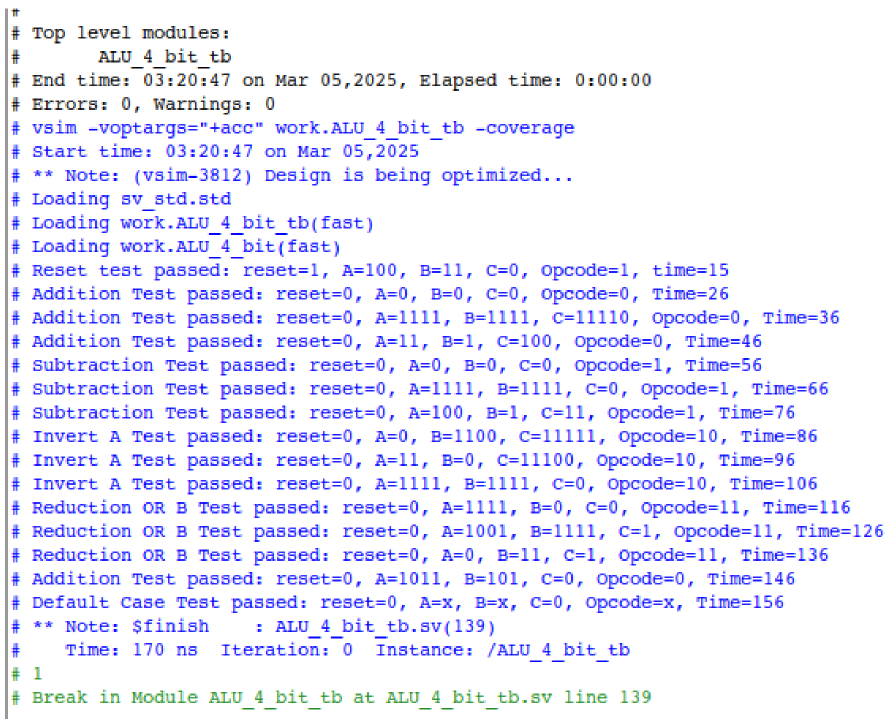


Figure 3: simulation waveform



```
#
# Top level modules:
#       ALU_4_bit_tb
# End time: 03:20:47 on Mar 05,2025, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# vsim -voptargs="+acc" work.ALU_4_bit_tb -coverage
# Start time: 03:20:47 on Mar 05,2025
# ** Note: (vsim-3812) Design is being optimized...
# Loading sv_std.std
# Loading work.ALU_4_bit_tb(fast)
# Loading work.ALU_4_bit(fast)
# Reset test passed: reset=1, A=100, B=11, C=0, Opcode=1, time=15
# Addition Test passed: reset=0, A=0, B=0, C=0, Opcode=0, Time=26
# Addition Test passed: reset=0, A=1111, B=1111, C=11110, Opcode=0, Time=36
# Addition Test passed: reset=0, A=11, B=1, C=100, Opcode=0, Time=46
# Subtraction Test passed: reset=0, A=0, B=0, C=0, Opcode=1, Time=56
# Subtraction Test passed: reset=0, A=1111, B=1111, C=0, Opcode=1, Time=66
# Subtraction Test passed: reset=0, A=100, B=1, C=11, Opcode=1, Time=76
# Invert A Test passed: reset=0, A=0, B=1100, C=11111, Opcode=10, Time=86
# Invert A Test passed: reset=0, A=11, B=0, C=11100, Opcode=10, Time=96
# Invert A Test passed: reset=0, A=1111, B=1111, C=0, Opcode=10, Time=106
# Reduction OR B Test passed: reset=0, A=1111, B=0, C=0, Opcode=11, Time=116
# Reduction OR B Test passed: reset=0, A=1001, B=1111, C=1, Opcode=11, Time=126
# Reduction OR B Test passed: reset=0, A=0, B=11, C=1, Opcode=11, Time=136
# Addition Test passed: reset=0, A=1011, B=101, C=0, Opcode=0, Time=146
# Default Case Test passed: reset=0, A=x, B=x, C=0, Opcode=x, Time=156
# ** Note: $finish    : ALU_4_bit_tb.sv(139)
#    Time: 170 ns  Iteration: 0  Instance: /ALU_4_bit_tb
# 1
# Break in Module ALU_4_bit_tb at ALU_4_bit_tb.sv line 139
```

Figure 4: Transcript : all test cases passed

## 2.4 4. Do File

```
vlib work
vlog ALU_4_bit.sv ALU_4_bit_tb.sv +cover -covercells
vsim -voptargs=+acc work.ALU_4_bit_tb -cover
add wave *
coverage save ALU_4_bit_tb.ucdb -du ALU_4_bit -onexit
run -all
```

## 2.5 5. Coverage Report

```
Coverage Report by instance with details

=================================================================================
=== Instance: /\ALU_4_bit_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_4_bit_tb#DUT

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File ALU_4_bit.sv
------------------------------------CASE Branch------------------------------------
    21                                        15     Count coming in to CASE
    22              1                          4               Add:            Alu_out = A + B;
    23              1                          4               Sub:            Alu_out = A - B;
    24              1                          3               Not_A:          Alu_out = ~A;
    25              1                          3               ReductionOR_B:  Alu_out = |B;
    26              1                          1            default:   Alu_out = 5'b0;
Branch totals: 5 hits of 5 branches = 100.00%

------------------------------------IF Branch------------------------------------
```

```
    32                                        17      Count coming in to IF
    32              1                          2            if (reset)
    34              1                         15            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_4_bit_tb#DUT --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File ALU_4_bit.sv
    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
    12                                                  reg signed [4:0]            Alu_out; // ALU output in 2's complement
    13
    14                                                  localparam                  Add             = 2'b00; // A + B
    15                                                  localparam                  Sub             = 2'b01; // A - B
    16                                                  localparam                  Not_A               = 2'b10; // ~A
    17                                                  localparam                  ReductionOR_B   = 2'b11; // |B
    18
    19                                                  // Do the operation
    20              1                         15        always @* begin
    21                                                     case (Opcode)
    22              1                          4               Add:            Alu_out = A + B;
    23              1                          4               Sub:            Alu_out = A - B;
    24              1                          3               Not_A:          Alu_out = ~A;
    25              1                          3               ReductionOR_B:  Alu_out = |B;
    26              1                          1             default:
    27                                                     endcase
    28                                                  end // always @ *
    29
    30                                                  // Register output C
    31              1                         17        always @(posedge clk or posedge reset) begin
    32                                                     if (reset)
    33              1                          2             C <= 5'b0;
    34                                                     else
    35              1                         15             C<= Alu_out;

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

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

Toggle Coverage for instance /\ALU_4_bit_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%
```

## 3 DSP

### 3.1 1 Design

```
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;
reg  [18:0] adder_out_stg1, adder_out_stg2;
reg  [47:0] C_reg;
reg  [36:0] mult_out;

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;
        adder_out_stg2 <= adder_out_stg1;
        if (OPERATION == "ADD") begin
            adder_out_stg1 <= D_reg + B_reg;
            P <= mult_out + C_reg;
        end
        else if (OPERATION == "SUBTRACT") begin
            adder_out_stg1 <= D_reg - B_reg;
            P <= {{11{mult_out[36]}},mult_out} - C_reg;
        end
        mult_out <= A_reg_stg2 * adder_out_stg2;
    end
end

endmodule
```

## 3.2 Bug Fixes

- **Bug 1: Adder Output Width Mismatch**
  - Issue: The adder output was stored in 18-bit registers Since the sum or difference of two 18-bit inputs (D_reg + B_reg or D_reg - B_reg) can produce a 19-bit result, this caused overflow and data truncation.
  - Fix: Increased the adder output width to 19 bits
- **Bug 2: Multiplier Output Width Mismatch**
  - Issue: The multiplier output was defined as 48 bits However, multiplying two 18-bit numbers (A_reg_stg2 * adder_out_stg2) produces a maximum of a 36-bit result, leading to an over-provisioned width and potential misalignment in subsequent operations
  - Fix: Adjusted to 37 bits
- **Bug 3: Sign Extension Error**
  - Issue: The subtraction operation did not properly account for the width mismatch between the 37-bit multiplier output (mult_out) and the 48-bit accumulation register (P). This could cause incorrect behavior due to the absence of sign extension
  - Fix: Sign extension was added to mult_out before the subtraction, ensuring proper 48-bit alignment
- **Bug 4: Uninitialized Registers**
  - Issue: The following registers did not have an initial reset value, leading to potential X (unknown) states during simulation or invalid values during hardware power-up: [ adder_out_stg2 & C_reg ]
  - Fix: Added reset logic for all registers

## 3.3 2. Verification Plan

| Label | Design Requirement Description | Stimulus Generation | Functional Coverage | Functionality Check |
|---|---|---|---|---|
| RESET_TEST | When reset is asserted, the output P should be zero | Directed at the start of the simulation | - | A checker verifies P == 0 during reset |
| ADD_TEST | Perform ADD operation: P = (A * (D + B)) + C | Randomized inputs | - | Compare P to golden model result |
| SUB_TEST | Perform SUBTRACT operation: P = (A * (D - B)) - C | Randomized inputs | - | Compare P to golden model result |
| RAND_TEST | Randomize A, B, C, D inputs and verify correct ADD/SUBTRACT results | Randomization with constraints | - | Verify against golden model |

Table 3: Verification Plan

## 3.4 3 Testbench

```
module DSP_tb;
    //-------------------------------------------
    // Declare signals
    //-------------------------------------------
    reg [17:0] A, B, D;
    reg [47:0] C;
    reg clk, rst_n;
    wire [47:0] P;

    //-------------------------------------------
    // Instantiate the DUT (Device Under Test)
    //-------------------------------------------
    DSP #(.OPERATION("ADD")) dut (
        .A(A),
        .B(B),
        .C(C),
        .D(D),
        .clk(clk),
        .rst_n(rst_n),
        .P(P)
    );

    //-----------------------------------------
    // Golden Model
    //-----------------------------------------
    reg [47:0] golden_P;

    //-----------------------------------------
    // Clock generation
    //-----------------------------------------
    parameter CLOCK_PERIOD = 10;
    always begin
        #(CLOCK_PERIOD/2) clk = ~clk;
    end

    //-----------------------------------------
    // Reset task
    //-----------------------------------------
    task reset_dut;
        begin
            // Assert reset
            rst_n = 0;
            @(posedge clk);

            // Check output reset to zero
            if (P != 48'b0) begin
                $display("Reset␣test␣failed:␣rst_n=%0b␣,␣A=%0d␣,␣B=%0d␣,␣D=%0d␣,␣C=%0d␣,␣P=%0d", rst_n, A, B, D, C, P);
                $stop;
            end else begin
                $display("Reset␣test␣passed:␣rst_n=%0b␣,␣A=%0d␣,␣B=%0d␣,␣D=%0d␣,␣C=%0d␣,␣P=%0d", rst_n, A, B, D, C, P);
            end

            // Deassert reset
            #(CLOCK_PERIOD/2);
            rst_n = 1;
        end
    endtask

    //-----------------------------------------
    // Check "ADD" result task
    //-----------------------------------------
    task check_ADD_result;
        begin
            // Golden model calculation
            golden_P = (A * (D + B)) + C;

            // Wait for 4 clock cycles
            #50;

            // Compare the output with the golden model
            if (P !== golden_P) begin
                $display("Error:␣in␣ADD␣TEST␣Mismatch␣found!␣Expected:␣%h,␣Got:␣%h␣,rst_n=%0b␣,␣A=%0h␣,␣B=%0h␣,␣D=%0h␣,␣C=%0h", golden_P,
                    P , rst_n, A, B, D, C);
                $stop;
            end else begin
                $display("Success:␣in␣ADD␣TEST␣Output␣matches␣the␣golden␣model.␣P␣=␣%d␣,rst_n=%0b␣,␣A=%0h␣,␣B=%0h␣,␣D=%0h␣,␣C=%0h", P ,
                    rst_n, A, B, D, C);
            end
        end
    endtask

    //-----------------------------------------
    // Check "SUBSTRACT" result task
    //-----------------------------------------
    task check_sub_result;
        begin
            // Golden model calculation
            golden_P = (A[17:0] * (D [17:0] - B[17:0])) - C;

            // Wait for 4 clock cycles
            #50;

            // Compare the output with the golden model
            if (P !== golden_P) begin
                $display("Error:␣in␣sub␣TEST␣Mismatch␣found!␣Expected:␣%h,␣Got:␣%h␣,rst_n=%0b␣,␣A=%0h␣,␣B=%0h␣,␣D=%0h␣,␣C=%0h", golden_P,
                    P , rst_n, A, B, D, C);
                $stop;
            end else begin
                $display("Success:␣in␣sub␣TEST␣Output␣matches␣the␣golden␣model.␣P␣=␣%d␣,rst_n=%0b␣,␣A=%0h␣,␣B=%0h␣,␣D=%0h␣,␣C=%0h", P,
                    rst_n, A, B, D, C);
            end
        end
    endtask

    //-----------------------------------------
    // Randomization with constraints
    //-----------------------------------------
    class InputValues;
    rand reg [17:0] A, B, D;
    rand reg [47:0] C;

    // Balanced distribution for extreme and mid-range values
    constraint balanced_values {
           A dist { 18'h00000 := 10, 18'h3FFFF := 40, [18'h00001:18'h3FFFE] :/ 50 };
           B dist { 18'h00000 := 10, 18'h3FFFF := 40, [18'h00001:18'h3FFFE] :/ 50 };
```

```
       D dist { 18'h00000 := 10, 18'h3FFFF := 40, [18'h00001:18'h3FFFE] :/ 50 };
           C dist { 48'h000000000000 := 10, 48'hFFFFFFFFFFFF := 40, [48'h000000000001:48'hFFFFFFFFFFFE] :/ 50 };
    }
    endclass


    InputValues inputs;

    //------------------------------------------
    // Randomization and simulation procedure.
    //------------------------------------------
    initial begin
        //-------------------------------------
        // Initialize signals
        //-------------------------------------
        clk = 1'b0;
        rst_n = 1'b1;
        A = 1;
    B = 4;
    C = 2;
    D = 16;

        #(CLOCK_PERIOD);

        //----------------------------------
        // Check reset functionality
        //----------------------------------
        reset_dut;

        //---------------------------------
        // Start the test
        //---------------------------------
        inputs = new();
        for (int i = 0; i < 20; i = i + 1) begin
            if (inputs.randomize()) begin
                A = inputs.A;
                B = inputs.B;
                C = inputs.C;
                D = inputs.D;
            end else begin
                $display("Randomization␣failed.");
                $stop;
            end

            // Call the check ADD result task
            // operation = "ADD";
            check_ADD_result;

            // Call the check substract result task
            // operation = "SUBTRACT";
            // check_sub_result;
        end

        // Finish the simulation
        $finish;
    end
endmodule
```

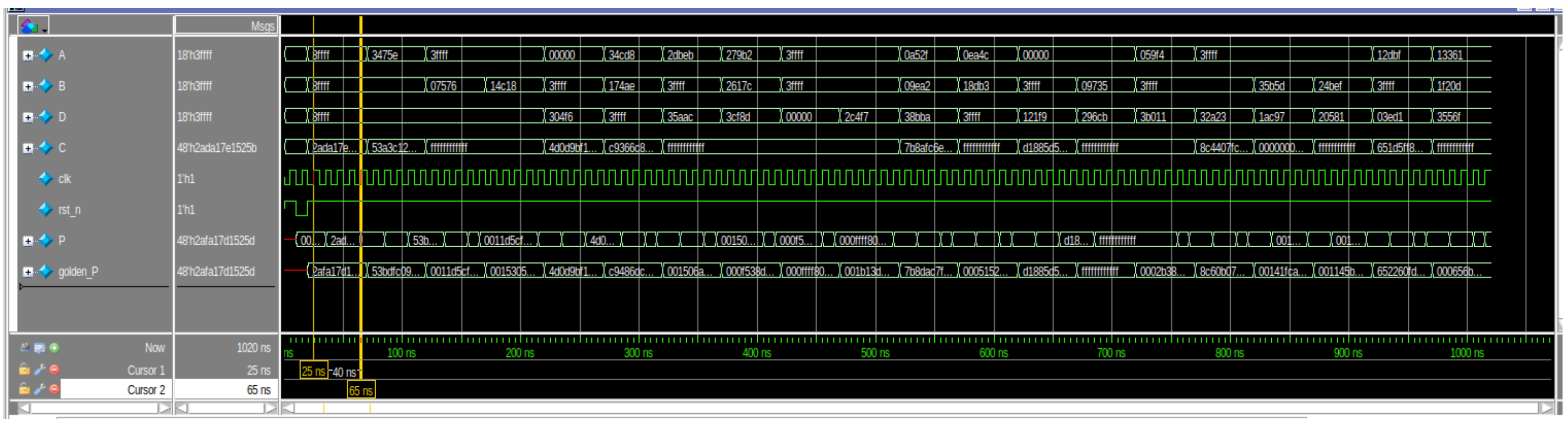
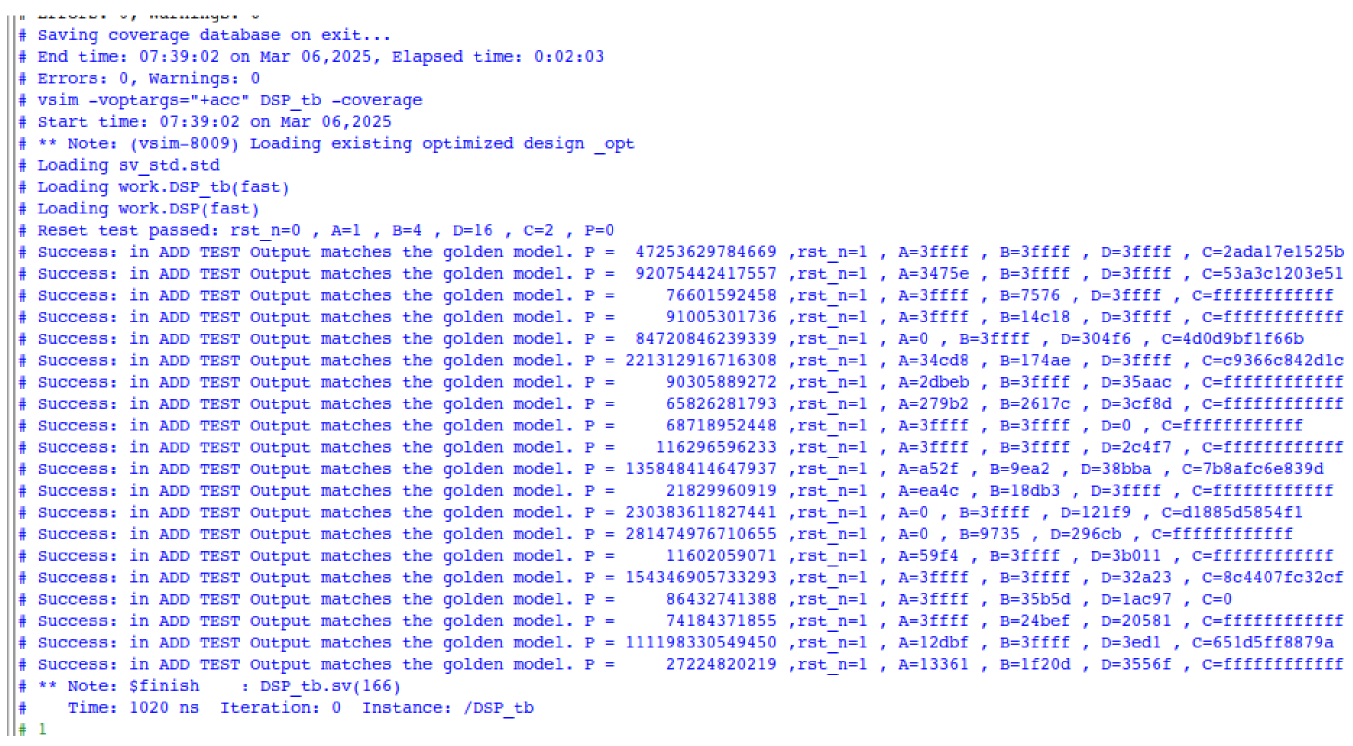


Figure 5: simulation wave form



```
# Errors: 0, Warnings: 0
# Saving coverage database on exit...
# End time: 07:39:02 on Mar 06,2025, Elapsed time: 0:02:03
# Errors: 0, Warnings: 0
# vsim -voptargs="+acc" DSP_tb -coverage
# Start time: 07:39:02 on Mar 06,2025
# ** Note: (vsim-8009) Loading existing optimized design _opt
# Loading sv_std.std
# Loading work.DSP_tb(fast)
# Loading work.DSP(fast)
# Reset test passed: rst_n=0 , A=1 , B=4 , D=16 , C=2 , P=0
# Success: in ADD TEST Output matches the golden model. P =  47253629784669 ,rst_n=1 , A=3ffff , B=3ffff , D=3ffff , C=2ada17e1525b
# Success: in ADD TEST Output matches the golden model. P =  92075442417557 ,rst_n=1 , A=3475e , B=3ffff , D=3ffff , C=53a3c1203e51
# Success: in ADD TEST Output matches the golden model. P =     76601592458 ,rst_n=1 , A=3ffff , B=7576 , D=3ffff , C=ffffffffffff
# Success: in ADD TEST Output matches the golden model. P =     91005301736 ,rst_n=1 , A=3ffff , B=14c18 , D=3ffff , C=ffffffffffff
# Success: in ADD TEST Output matches the golden model. P =  84720846239339 ,rst_n=1 , A=0 , B=3ffff , D=304f6 , C=4d0d9bf1f66b
# Success: in ADD TEST Output matches the golden model. P = 221312916716308 ,rst_n=1 , A=34cd8 , B=174ae , D=3ffff , C=c9366c842d1c
# Success: in ADD TEST Output matches the golden model. P =     90305889272 ,rst_n=1 , A=2dbeb , B=3ffff , D=35aac , C=ffffffffffff
# Success: in ADD TEST Output matches the golden model. P =     65826281793 ,rst_n=1 , A=279b2 , B=2617c , D=3cf8d , C=ffffffffffff
# Success: in ADD TEST Output matches the golden model. P =     68718952448 ,rst_n=1 , A=3ffff , B=3ffff , D=0 , C=ffffffffffff
# Success: in ADD TEST Output matches the golden model. P =    116296596233 ,rst_n=1 , A=3ffff , B=3ffff , D=2c4f7 , C=ffffffffffff
# Success: in ADD TEST Output matches the golden model. P = 135848414647937 ,rst_n=1 , A=a52f , B=9ea2 , D=38bba , C=7b8afc6e839d
# Success: in ADD TEST Output matches the golden model. P =     21829960919 ,rst_n=1 , A=ea4c , B=18db3 , D=3ffff , C=ffffffffffff
# Success: in ADD TEST Output matches the golden model. P = 230383611827441 ,rst_n=1 , A=0 , B=3ffff , D=121f9 , C=d1885d5854f1
# Success: in ADD TEST Output matches the golden model. P = 281474976710655 ,rst_n=1 , A=0 , B=9735 , D=296cb , C=ffffffffffff
# Success: in ADD TEST Output matches the golden model. P =     11602059071 ,rst_n=1 , A=59f4 , B=3ffff , D=3b011 , C=ffffffffffff
# Success: in ADD TEST Output matches the golden model. P = 154346905733293 ,rst_n=1 , A=3ffff , B=3ffff , D=32a23 , C=8c4407fc32cf
# Success: in ADD TEST Output matches the golden model. P =     86432741388 ,rst_n=1 , A=3ffff , B=35b5d , D=1ac97 , C=0
# Success: in ADD TEST Output matches the golden model. P =     74184371855 ,rst_n=1 , A=3ffff , B=24bef , D=20581 , C=ffffffffffff
# Success: in ADD TEST Output matches the golden model. P = 111198330549450 ,rst_n=1 , A=12dbf , B=3ffff , D=3ed1 , C=651d5ff8879a
# Success: in ADD TEST Output matches the golden model. P =     27224820219 ,rst_n=1 , A=13361 , B=1f20d , D=3556f , C=ffffffffffff
# ** Note: $finish    : DSP_tb.sv(166)
#    Time: 1020 ns  Iteration: 0  Instance: /DSP_tb
# 1
```

Figure 6: transcript : all tests passed

### 3.5 4 Do File

```
vlib work
vlog DSP.v DSP_tb.sv +cover -covercells
vsim -voptargs=+acc DSP_tb -cover
add wave *
coverage save DSP_tb.ucdb -du DSP -onexit
run -all
```

### 3.6 5 Coverage Report

```
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                                       102     Count coming in to IF
    14              1                          2        if (!rst_n) begin
    26              1                        100        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                                                parameter OPERATION = "ADD";
    3                                                input  [17:0] A, B, D;
    4                                                input  [47:0] C;
    5                                                input clk, rst_n;
    6                                                output reg  [47:0] P;
    7
    8                                                reg  [17:0] A_reg_stg1, A_reg_stg2, B_reg, D_reg;
    9                                                reg  [18:0] adder_out_stg1, adder_out_stg2;
    10                                               reg  [47:0] C_reg;
    11                                               reg  [36:0] mult_out;
    12
    13              1                        102     always @(posedge clk or negedge rst_n) begin
    14                                                  if (!rst_n) begin
    15                                                          // reset
    16              1                          2                A_reg_stg1 <= 0;
    17              1                          2                A_reg_stg2 <= 0;
    18              1                          2                B_reg <= 0;
    19              1                          2                D_reg <= 0;
    20              1                          2                C_reg <= 0;
    21              1                          2                adder_out_stg1 <= 0;
    22              1                          2                adder_out_stg2 <= 0;
    23              1                          2                mult_out <= 0;
    24              1                          2                P <= 0;
    25                                                      end
    26                                                      else begin
    27              1                        100                A_reg_stg1 <= A;
    28              1                        100                A_reg_stg2 <= A_reg_stg1;
    29              1                        100                B_reg <= B;
    30              1                        100                C_reg <= C;
    31              1                        100                D_reg <= D;
    32              1                        100                adder_out_stg2 <= adder_out_stg1;
    33                                                          if (OPERATION == "ADD") begin
    34              1                        100                        adder_out_stg1 <= D_reg + B_reg;
    35              1                        100                        P <= mult_out + C_reg;
    36                                                          end
    37                                                          else if (OPERATION == "SUBTRACT") begin
    38                                                                  adder_out_stg1 <= D_reg - B_reg;
    39                                                                  P <= {{11{mult_out[36]}},mult_out} - C_reg;
    40                                                          end
    41              1                        100                mult_out <= A_reg_stg2 * adder_out_stg2;

Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                        694       694         0   100.00%

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

Toggle Coverage for instance /\DSP_tb#dut --

                                              Node      1H->0L      0L->1H  "Coverage"
                                              ---------------------------------------
                                           A[0-17]           1           1      100.00
                                  A_reg_stg1[17-0]           1           1      100.00
                                  A_reg_stg2[17-0]           1           1      100.00
                                           B[0-17]           1           1      100.00
                                       B_reg[17-0]           1           1      100.00
                                           C[0-47]           1           1      100.00
                                       C_reg[47-0]           1           1      100.00
                                           D[0-17]           1           1      100.00
                                       D_reg[17-0]           1           1      100.00
                                           P[47-0]           1           1      100.00
                              adder_out_stg1[18-0]           1           1      100.00
                              adder_out_stg2[18-0]           1           1      100.00
                                               clk           1           1      100.00
                                    mult_out[36-0]           1           1      100.00
                                             rst_n           1           1      100.00

Total Node Count     =        347
Toggled Node Count   =        347
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (694 of 694 bins)


Total Coverage By Instance (filtered view): 100.00%
```