PROJECT 2

CE 6325.001 VLSI DESIGN

PROJECT 2:

CELL REPORT FOR

32-BIT ARITHMETIC LOGIC UNIT (ALU)

TEAM MEMBERS:

1) ALEXANDRA EDWINRAJ (NET ID: AXE210023)

2) HEENISHA REDDY BACHUGUDEM (NET ID: HXR210022)

3)KANUPRIYA SHARMA (NET ID: KXS220016)

Introduction:

In this Project, we have 2 16-bit Inputs which perform functions depending on the selected line. We have found the cell count for the ALU Module using Synopsys Design Vision (CAD Tool). The Design Vision software read our Behavioral Verilog code and then used the library to generate a mapped netlist file. This mapped the netlist file along with the concatenated header.v code was simulated again in ModelSim using the same testbench code. The waveform obtained for the mapped netlist code is compared with the waveforms from the behavioral code.

Block diagram

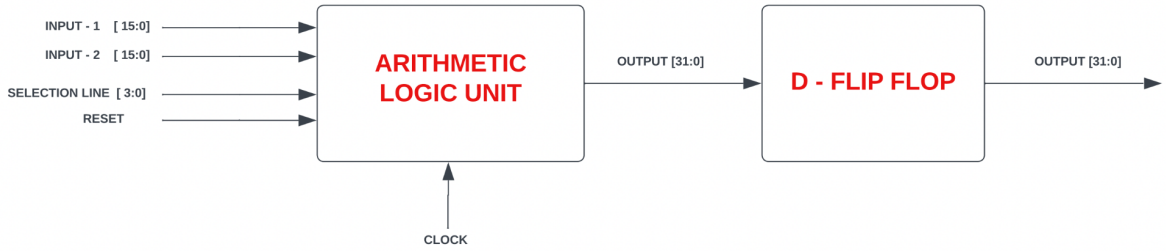


Figure 1 - Block diagram of the ALU with D-flip flop

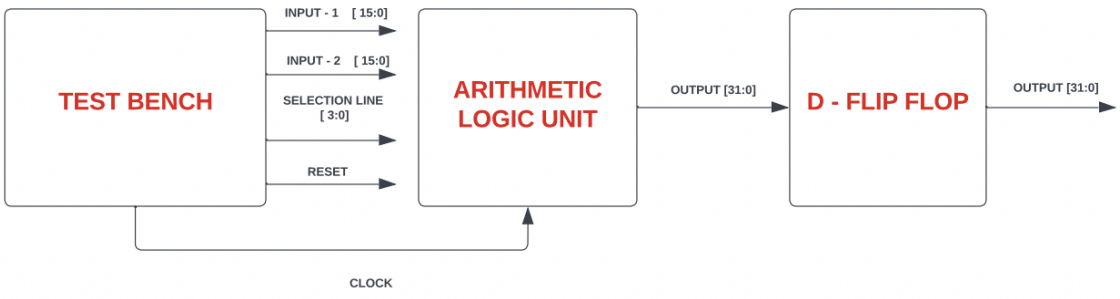


Figure 2 - Block diagram of the test bench connection with ALU

Behavioural Code and Testbench for 32-bit Arithmetic Logic Unit (ALU)

Behavioural Code for 32-bit Arithmetic Logic Unit (ALU):

```
// Arithmetic Logic Unit
 module alu(c, s, r, X, Y, Z);
 input c,r; // inputs- clock and reset
 input [3:0] s; // select line for operations
 input [15:0] X,Y; // 16-bit inputs
 output reg [31:0] Z; // 32-bit output
 reg [31:0] out; //output register
 wire [31:0] w; //output wire
 parameter Add= 4'd0, Substract=4'd1, Multiply= 4'd2, Divide=4'd3, //Arthematic Operations
 Shift_left =4'd4, Shift_right= 4'd5,Rotate_left=4'd6, Rotate_right = 4'd7, //Shift & Rotate
 AND = 4'd8, OR = 4'd9, NOR = 4'd10, NAND = 4'd11, XOR = 4'd12, XNOR = 4'd13, //Logic Operations
 Set_less_than = 4'd14, set_if_equal=4'd15; //less than & equal to
 always @ (negedge c ) // Triggers the block on the negative edge of a clock signal only
 Begin
 if (r==1'b0)
 out<=32'b0; // active low reset
 else
 case (s) //Based on the select line
 Add : out <= X + Y ; // Addition between two inputs
 Substract : out <= X - Y ;  // Subtraction between two inputs
```

```
Multiply : out <= X * Y ;  // Multiplication between two inputs

Divide : out <= X / Y ;  // Division between two inputs

Shift_left : out <= X << 1;  // Input is shifted to the left by 1 bit

Shift_right : out <= X >> 1; // Input is shifted to left by 1 bit

Rotate_left : out <= {X [14:0], X[15]}; // Input is rotated to the left by 1 bit

Rotate_right : out <= {X [0], X[15:1]}; // Input is rotated to the right by 1 bit

AND : out <= X & Y;  // Logical AND function is performed between two inputs

OR : out <= X | Y; // Logical OR function is performed between two inputs

NOR : out <= ~ (X | Y); // Logical NOR function is performed between two inputs

NAND : out <= ~ (X & Y); // Logical NAND function is performed between two inputs

XOR : out <= X ^ Y; // Logical XOR function is performed between two inputs

XNOR : out <=  ~ (X ^ Y) ; // Logical XNOR function is performed between two inputs

Set_less_than : out <= (X < Y)? 32'b1 : 32'b0; // Output is set to 1 if first input is less than second input

set_if_equal : out <= (X == Y)? 32'b1 : 32'b0; // Output is set to 1 if first input is equal to second input

default : out <= 32'd0; // For any other values, the output is reset

Endcase

end

//D Flip-flop

d_ff d1(out,c,r,w); //D Flip-flop inputs and outputs

always@(negedge c) // Triggers the block on the negative edge of a clock signal only

begin

Z<=w;
```

```
end

endmodule

module d_ff(d,c,r,w);

input c,r; // inputs- clock & reset

input [31:0]d; // 32-bit input

output reg[31:0]w; // 32-bit output register

always @( negedge c)

begin

if (r == 1'b0)

w<=64'b0; // active low reset

else

w<=d;

end

endmodule
```

Testbench for 32-bit ALU:

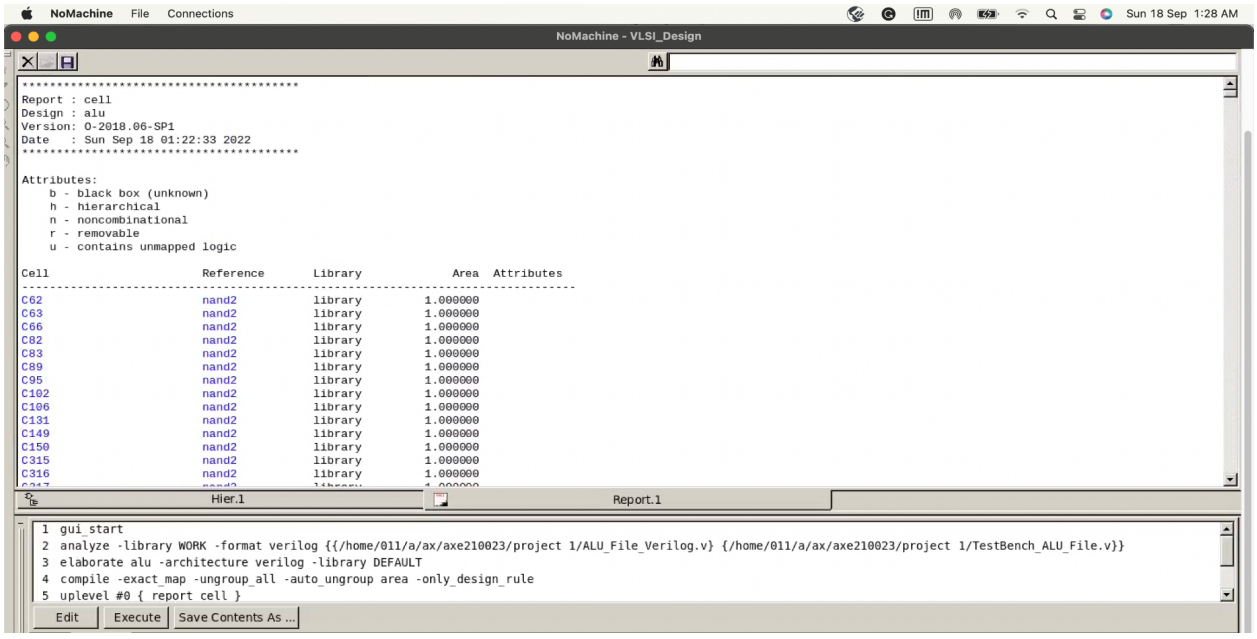
```
//Test Bench Module for Arithmetic Logic Unit
module alu_tb;
reg c; //Clock
reg r; //Reset
reg [3:0] s; //Select
reg [15:0] X; //Input
reg [15:0] Y; //Input
wire [31:0] Z; //Output

// Instantiating the unit under test
alu uut (
.c(c),
.r(r),
.s(s),
```

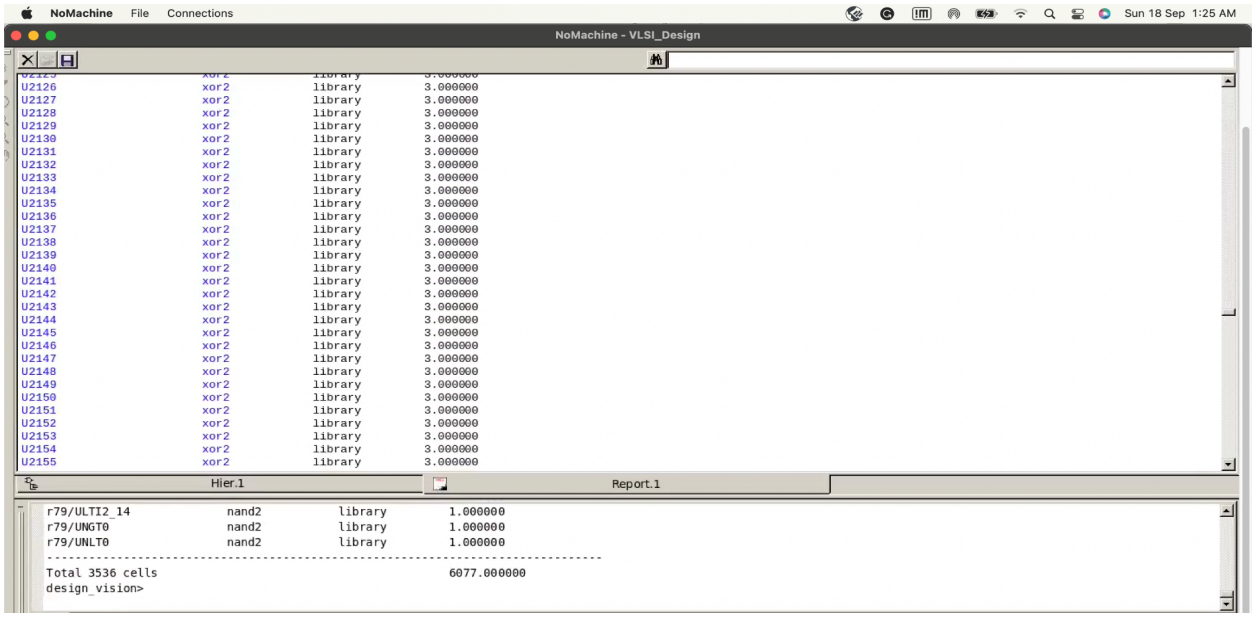
```
.X(X),
.Y(Y),
.Z(Z)
);

initial begin
 c=0; r=0; s=4'h0; X=16'h0; Y=16'h0; #100; // Reset and wait for 100ns
 r=1; s=4'h0; X=16'h28; Y=16'h14; #100; // Performing addition and waiting for 100ns
 r=1; s=4'h1; X=16'h28; Y=16'h14; #100; // Performing subtraction and waiting for 100ns
 r=1; s=4'h2; X=16'h28; Y=16'h14; #100; // Performing multiplication and waiting for 100ns
 r=1; s=4'h3; X=16'h28; Y=16'h14; #100; // Performing division and waiting for 100ns
 r=1; s=4'h4; X=16'h28; Y=16'h14; #100; // Shifting to the left by one bit and waiting for 100ns
 r=1; s=4'h5; X=16'h28; Y=16'h14; #100; // Shifting to the right by one bit and waiting for 100ns
 r=1; s=4'h6; X=16'h28; Y=16'h14; #100; // Rotating to the left by one bit and waiting for 100ns
 r=1; s=4'h7; X=16'h28; Y=16'h14; #100; // Rotating to the right by one bit and waiting for 100ns
 r=1; s=4'h8; X=16'h28; Y=16'h14; #100; // Performing logical AND operation and waiting for
100ns
 r=1; s=4'h9; X=16'h28; Y=16'h14; #100; // Performing logical OR operation and waiting for
100ns
 r=1; s=4'ha; X=16'h28; Y=16'h14; #100; // Performing logical NOR operation and waiting for
100ns
 r=1; s=4'hb; X=16'h28; Y=16'h14; #100; // Performing logical NAND operation and waiting for
100ns
 r=1; s=4'hc; X=16'h28; Y=16'h14; #100; // Performing logical XOR operation and waiting for
100ns
 r=1; s=4'hd; X=16'h28; Y=16'h14; #100; // Performing logical XNOR operation and waiting for
100ns
 r=1; s=4'he; X=16'h28; Y=16'h14; #100; // Setting the output to 1 if "X" is less than "Y", or
setting the output to 0 and waiting for 100ns
 r=1; s=4'hf; X=16'h28; Y=16'h14; #100; // Setting the output to 1 if "X" is equal to "Y", or
setting the output to 0 and waiting for 100ns
 #100; // delay
 $finish; // end of simulation
 end
 always #50 c = ~c; // generating the clock signal every 50ns by toggling
endmodule
```

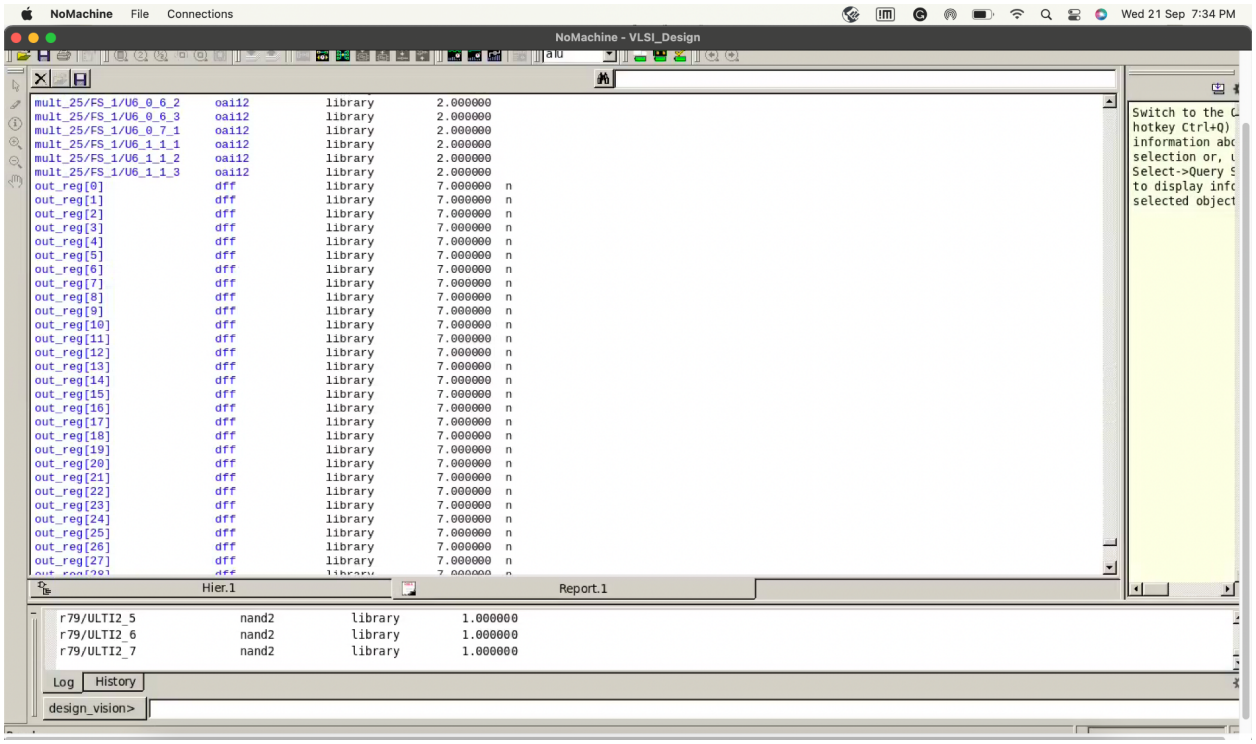
Cell Report by Synopsys Design Vision:

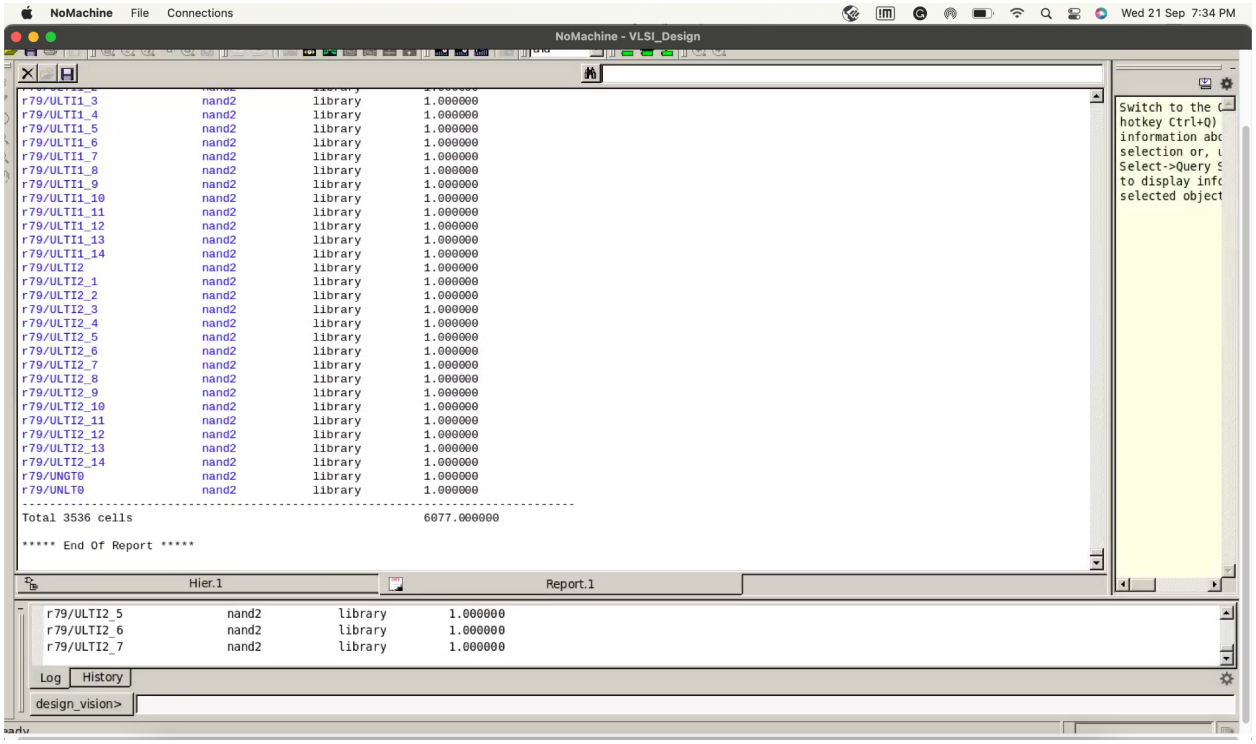
The Cell Report for ALU has two 16-bit Inputs and 32-bit output; We have found the cell count for the ALU Module using Synopsys Design Vision - CAD tool and the report was generated with zero errors.The output pictures show the cell reports from a compiled result by checking the **ungroup** option.

Number of Cells: 3536



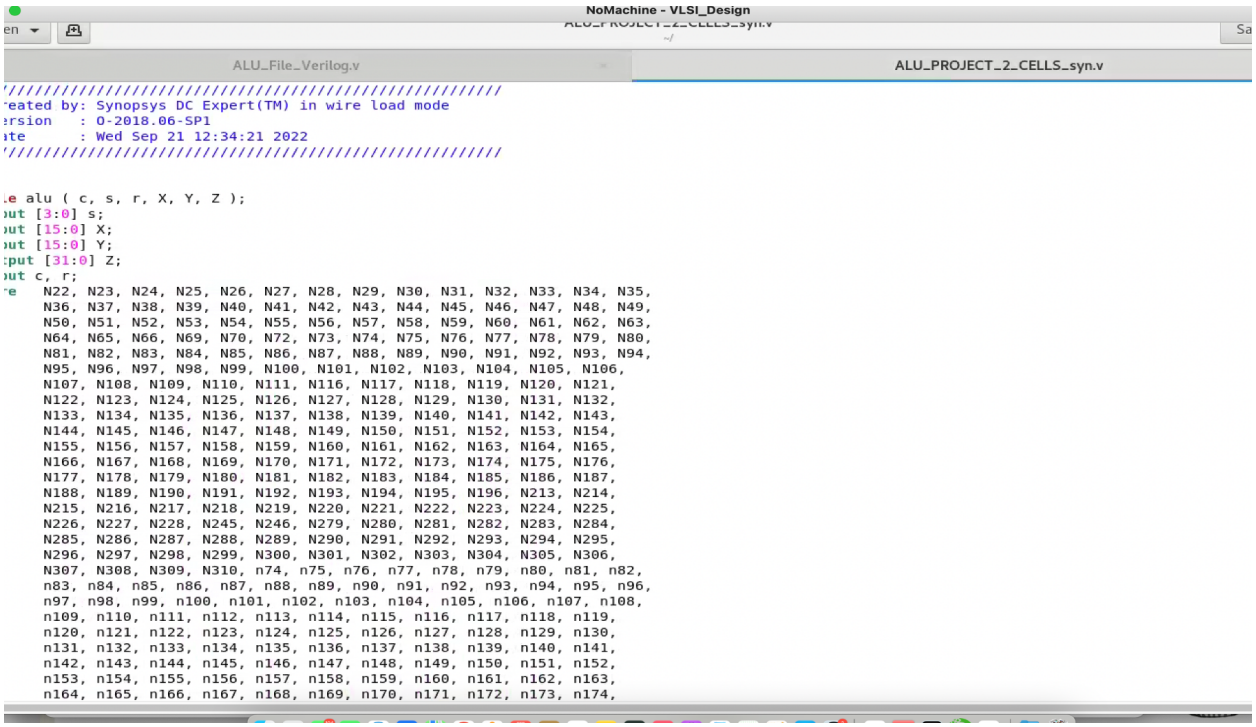


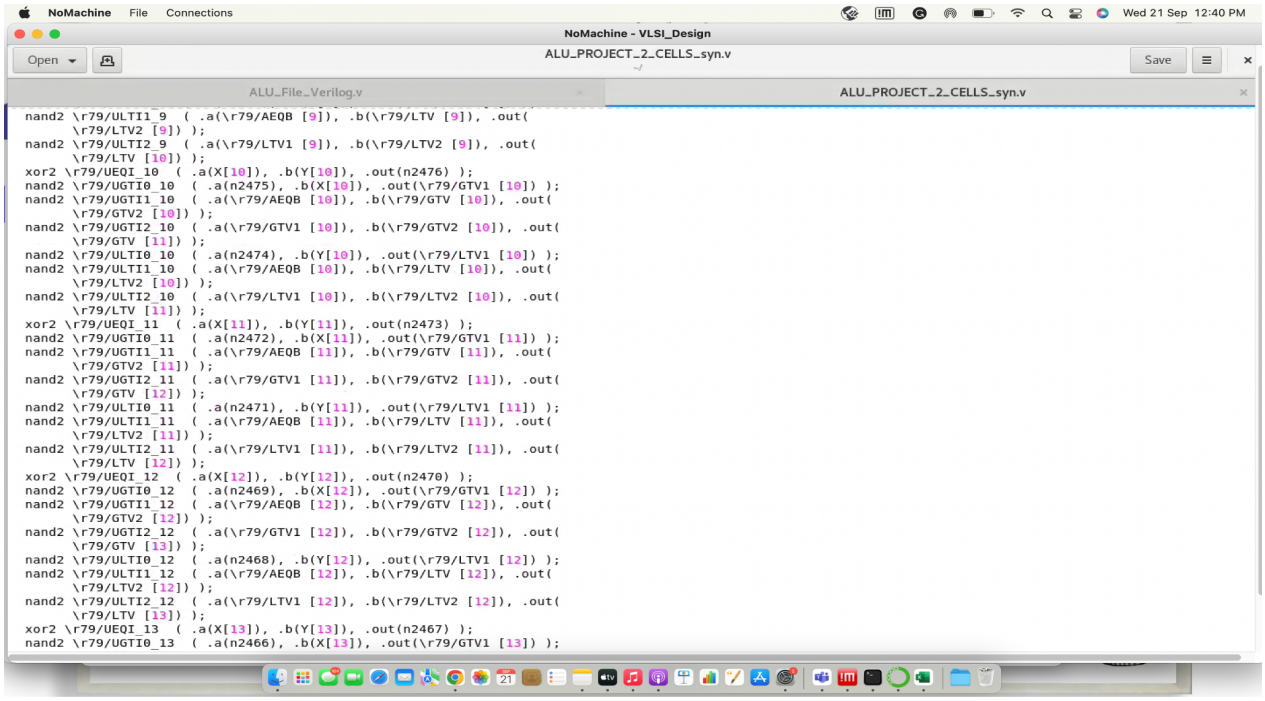


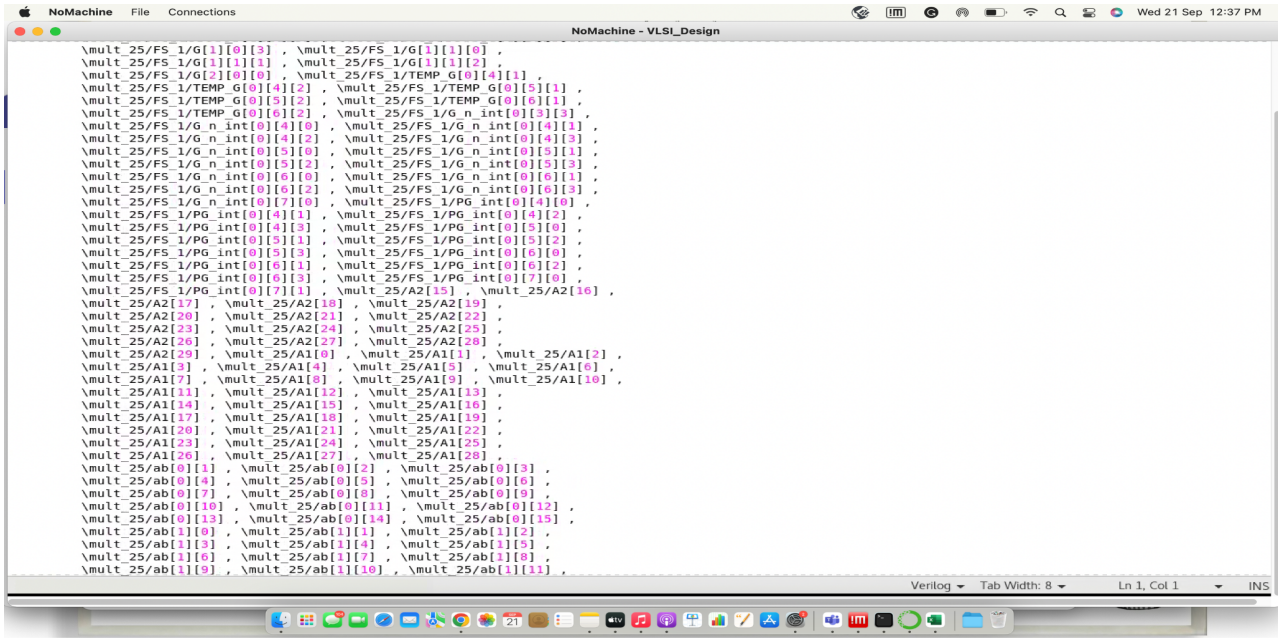


The Cell report has the D - Flip flop and the Logic Gates

Mapped Net List:






NoMachine File Connections
Wed 21 Sep 12:37 PM
NoMachine - VLSI_Design
\mult_25/FS_1/G[1][0][3] , \mult_25/FS_1/G[1][1][0] ,
\mult_25/FS_1/G[1][1][1] , \mult_25/FS_1/G[1][1][2] ,
\mult_25/FS_1/G[2][0][0] , \mult_25/FS_1/TEMP_G[0][4][1] ,
\mult_25/FS_1/TEMP_G[0][4][2] , \mult_25/FS_1/TEMP_G[0][5][1] ,
\mult_25/FS_1/TEMP_G[0][5][2] , \mult_25/FS_1/TEMP_G[0][6][1] ,
\mult_25/FS_1/TEMP_G[0][6][2] , \mult_25/FS_1/G_n_int[0][3][3] ,
\mult_25/FS_1/G_n_int[0][4][0] , \mult_25/FS_1/G_n_int[0][4][1] ,
\mult_25/FS_1/G_n_int[0][4][2] , \mult_25/FS_1/G_n_int[0][4][3] ,
\mult_25/FS_1/G_n_int[0][5][0] , \mult_25/FS_1/G_n_int[0][5][1] ,
\mult_25/FS_1/G_n_int[0][5][2] , \mult_25/FS_1/G_n_int[0][5][3] ,
\mult_25/FS_1/G_n_int[0][6][0] , \mult_25/FS_1/G_n_int[0][6][1] ,
\mult_25/FS_1/G_n_int[0][6][2] , \mult_25/FS_1/G_n_int[0][6][3] ,
\mult_25/FS_1/G_n_int[0][7][0] , \mult_25/FS_1/PG_int[0][4][0] ,
\mult_25/FS_1/PG_int[0][4][1] , \mult_25/FS_1/PG_int[0][4][2] ,
\mult_25/FS_1/PG_int[0][4][3] , \mult_25/FS_1/PG_int[0][5][0] ,
\mult_25/FS_1/PG_int[0][5][1] , \mult_25/FS_1/PG_int[0][5][2] ,
\mult_25/FS_1/PG_int[0][5][3] , \mult_25/FS_1/PG_int[0][6][0] ,
\mult_25/FS_1/PG_int[0][6][1] , \mult_25/FS_1/PG_int[0][6][2] ,
\mult_25/FS_1/PG_int[0][6][3] , \mult_25/FS_1/PG_int[0][7][0] ,
\mult_25/FS_1/PG_int[0][7][1] , \mult_25/A2[15] , \mult_25/A2[16] ,
\mult_25/A2[17] , \mult_25/A2[18] , \mult_25/A2[19] ,
\mult_25/A2[20] , \mult_25/A2[21] , \mult_25/A2[22] ,
\mult_25/A2[23] , \mult_25/A2[24] , \mult_25/A2[25] ,
\mult_25/A2[26] , \mult_25/A2[27] , \mult_25/A2[28] ,
\mult_25/A2[29] , \mult_25/A1[0] , \mult_25/A1[1] , \mult_25/A1[2] ,
\mult_25/A1[3] , \mult_25/A1[4] , \mult_25/A1[5] , \mult_25/A1[6] ,
\mult_25/A1[7] , \mult_25/A1[8] , \mult_25/A1[9] , \mult_25/A1[10] ,
\mult_25/A1[11] , \mult_25/A1[12] , \mult_25/A1[13] ,
\mult_25/A1[14] , \mult_25/A1[15] , \mult_25/A1[16] ,
\mult_25/A1[17] , \mult_25/A1[18] , \mult_25/A1[19] ,
\mult_25/A1[20] , \mult_25/A1[21] , \mult_25/A1[22] ,
\mult_25/A1[23] , \mult_25/A1[24] , \mult_25/A1[25] ,
\mult_25/A1[26] , \mult_25/A1[27] , \mult_25/A1[28] ,
\mult_25/ab[0][1] , \mult_25/ab[0][2] , \mult_25/ab[0][3] ,
\mult_25/ab[0][4] , \mult_25/ab[0][5] , \mult_25/ab[0][6] ,
\mult_25/ab[0][7] , \mult_25/ab[0][8] , \mult_25/ab[0][9] ,
\mult_25/ab[0][10] , \mult_25/ab[0][11] , \mult_25/ab[0][12] ,
\mult_25/ab[0][13] , \mult_25/ab[0][14] , \mult_25/ab[0][15] ,
\mult_25/ab[1][0] , \mult_25/ab[1][1] , \mult_25/ab[1][2] ,
\mult_25/ab[1][3] , \mult_25/ab[1][4] , \mult_25/ab[1][5] ,
\mult_25/ab[1][6] , \mult_25/ab[1][7] , \mult_25/ab[1][8] ,
\mult_25/ab[1][9] , \mult_25/ab[1][10] , \mult_25/ab[1][11] ,
Verilog Tab Width: 8 Ln 1, Col 1 INS

Behavioural Code Output:

Behavioural code result output for all operations of the ALU

S.NO	FUNCTION	INPUT - X	INPUT - Y	OUTPUT - Z
1.	Reset	0000	0000	00000000
2.	Addition	28	14	0000003c
3.	Subtraction	28	14	00000014
4.	Multiplication	28	14	00000320
5.	Division	28	14	00000002
6.	Logical shift left	28	14	00000050
7.	Logical shift right	28	14	00000014
8.	Rotate left	28	14	00000050
9.	Rotate right	28	14	00000014
10.	AND	28	14	00000000
11.	OR	28	14	0000003c
12.	NOR	28	14	ffffffc3
13.	NAND	28	14	ffffffff
14.	XOR	28	14	0000003c
15.	XNOR	28	14	ffffffc3
16.	Less than	28	14	0000

Comparison of Mapped code and Behavioural code Waveform:

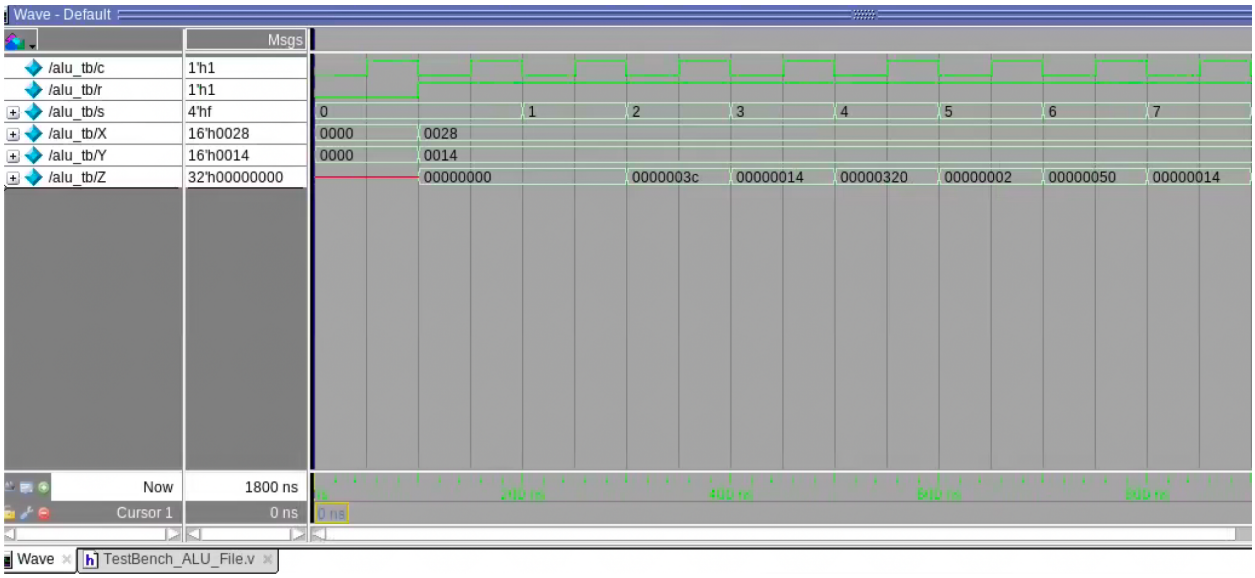
The following images consist of the resultant Mapped code waveforms and Behavioural code waveforms that have been simulated in ModelSim. The waveform shows the outputs for 16 different ALU operations. The first two set of waveforms consists of 7 ALU operations and the second set of waveforms consists of 9 ALU operations.

Waveform - Output for the following operations:

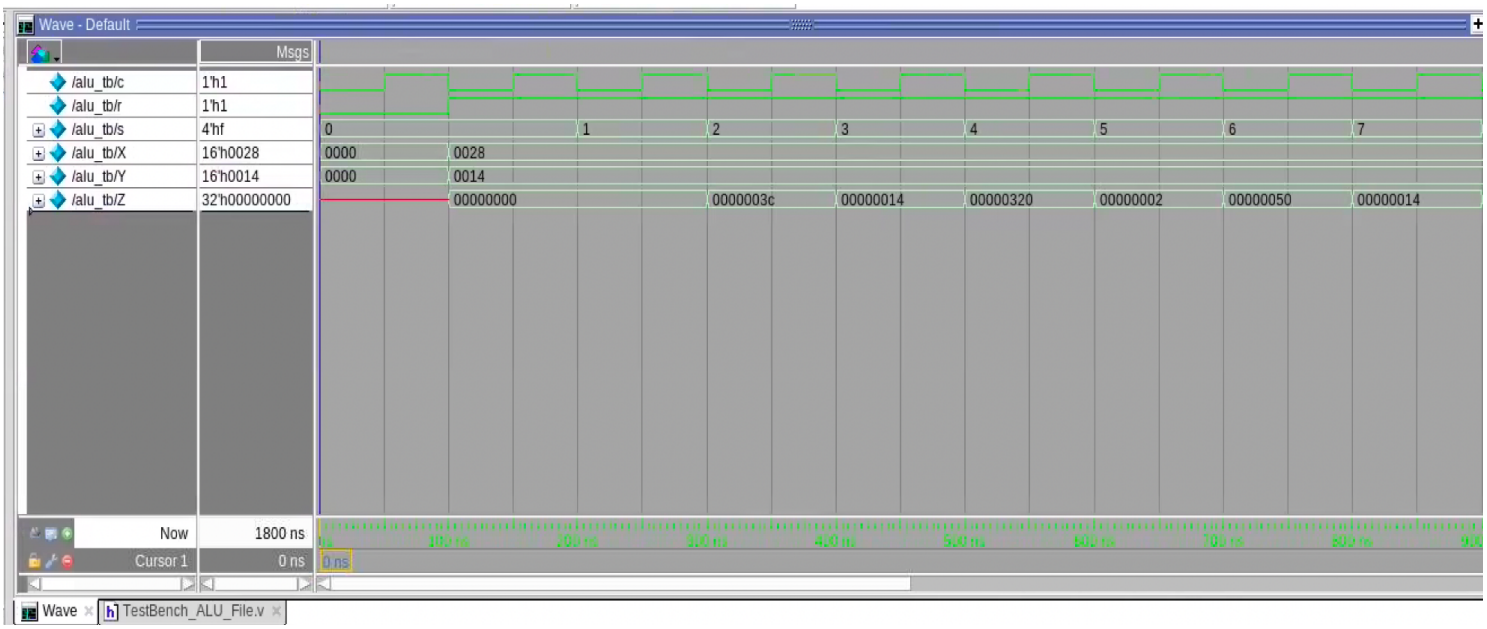
- **Operations:**

1. ALU - RESET
2. ALU - ADDITION OPERATION
3. ALU - SUBTRACTION OPERATION
4. ALU - MULTIPLICATION OPERATION
5. ALU - DIVISION OPERATION
6. ALU - Logical Shift Left OPERATION
7. ALU - Logical Shift Right OPERATION

Mapped Code Waveform:



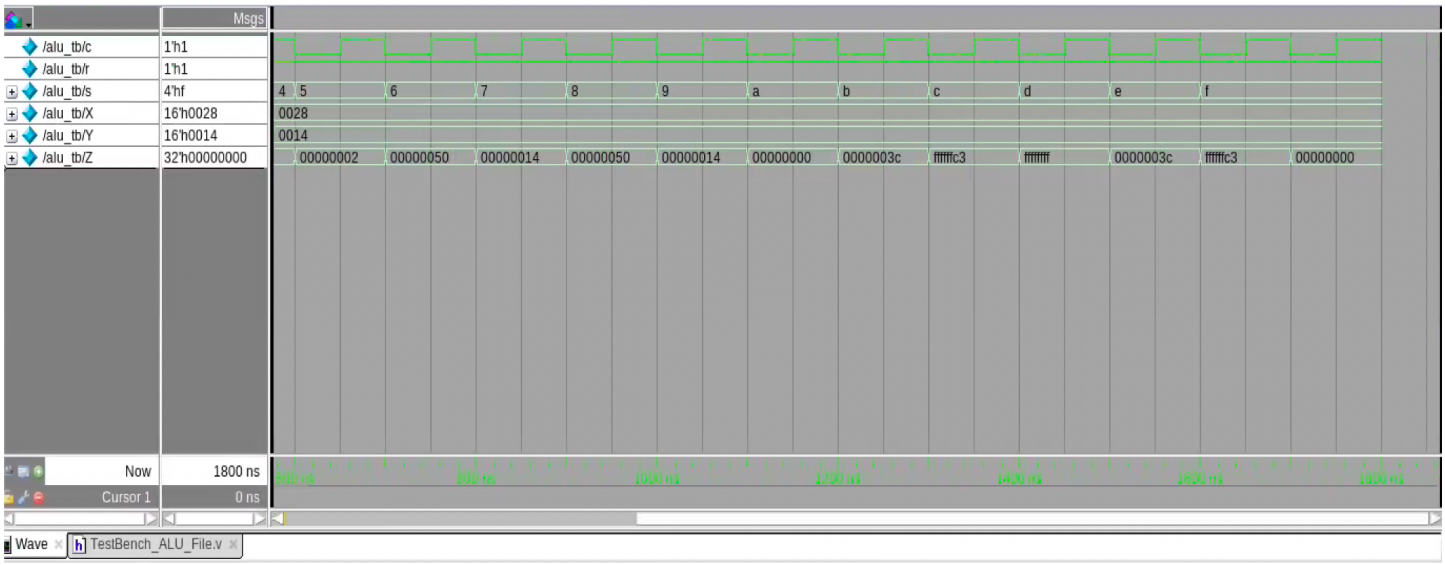
Behavioral Code Waveform



- **Operations:**

8. ALU - Rotate left OPERATION
9. ALU - Rotate right OPERATION
10. ALU - AND OPERATION
11. ALU - OR OPERATION
12. ALU - NOR OPERATION
13. ALU - NAND OPERATION
14. ALU - XOR OPERATION
15. ALU - XNOR OPERATION
16. ALU - Less than OPERATION

Mapped Code Waveform:



Behavioral Code Waveform: