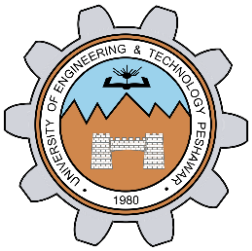
MULTIPLEXER IN VERILOG

LAB # 07



Fall 2023

CSE-304L Computer Organization and Architecture Lab

Submitted by: **Ali Asghar**

Registration No.: **21PWCSE2059**

Class Section: **C**

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Submitted to:

Dr. Bilal Habib

Date:

7th December 2023

Department of Computer Systems Engineering

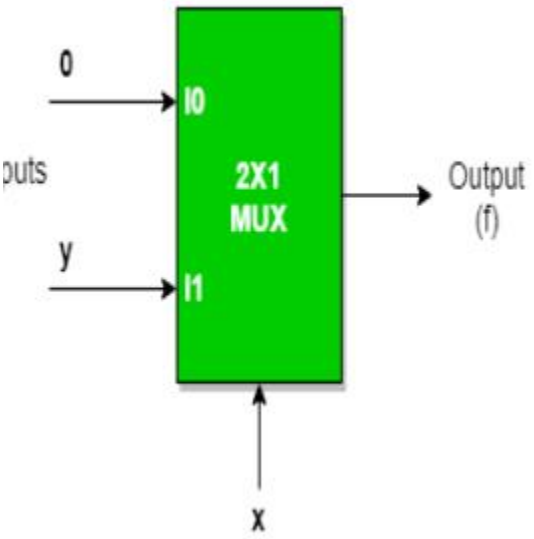
University of Engineering and Technology, Peshawar

ASSESSMENT RUBRICS COA LABS

LAB REPORT ASSESSMENT				
Criteria	**Excellent**	**Average**	**Nill**	**Marks Obtained**
1. Objectives of Lab	All objectives of lab are properly covered [Marks 10]	Objectives of lab are partially covered [Marks 5]	Objectives of lab are not shown [Marks 0]	
2. MIPS instructions with Comments and proper indentations.	All the instructions are well written with comments explaining the code and properly indented [Marks 20]	Some instructions are missing are poorly commented code [Marks 10]	The instructions are not properly written [Marks 0]	
3. Simulation run without error and warnings	The code is running in the simulator without any error and warnings [Marks 10]	The code is running but with some warnings or errors. [Marks 5]	The code is written but not running due to errors [Marks 0]	
4. Procedure	All the instructions are written with proper procedure [Marks 20]	Some steps are missing [Marks 10]	steps are totally missing [Marks 0]	
5. OUTPUT	Proper output of the code written in assembly [Marks 20]	Some of the outputs are missing [Marks 10]	No or wrong output [Marks 0]	
6. Conclusion	Conclusion about the lab is shown and written [Marks 20]	Conclusion about the lab is partially shown [Marks 10]	Conclusion about the lab is not shown[Marks0]	
7. Cheating			Any kind of cheating will lead to 0 Marks	
Total Marks Obtained:_____ Instructor Signature: _____				

Task 1:

Implement 2X1 MUX in Verilog.



Truth Table

x	y	f	f→y
0	0	0	f = 0
0	1	0	
1	0	0	f = y
1	1	1	

Code:

DUT Code:

Task4.asm | mux2x1.v | tb_mux2x1.v | tb_demux.v | tb_mux4x1.v | mux_4x1.v | task2.v | tb_task2.v | task3.v | tb_task3.v

D:\Uni\COA Lab\Lab4\Task4.asm

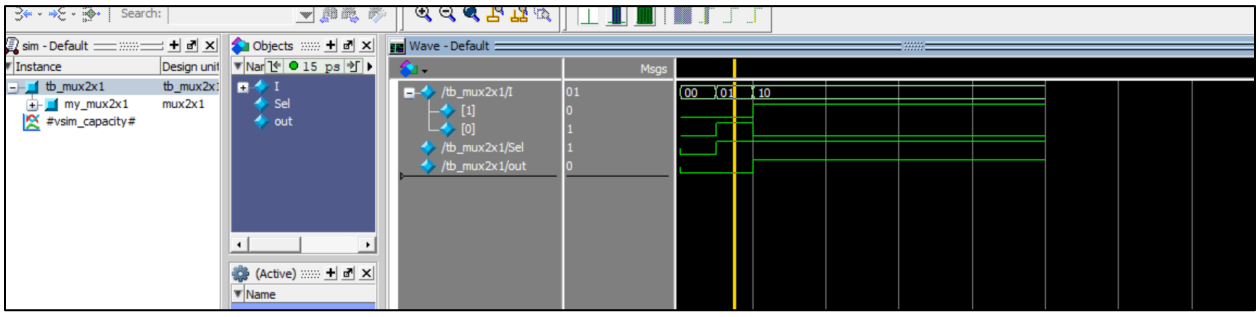
```
1   module mux2x1(I, Sel, Out);
2       input [1:0] I;
3       input Sel;
4
5       output Out;
6       reg Out;
7
8       always@(I, Sel)
9           case(Sel)
10          1'b0 : Out = I[0];
11          1'b1 : Out = I[1];
12          endcase
13
14  endmodule
```

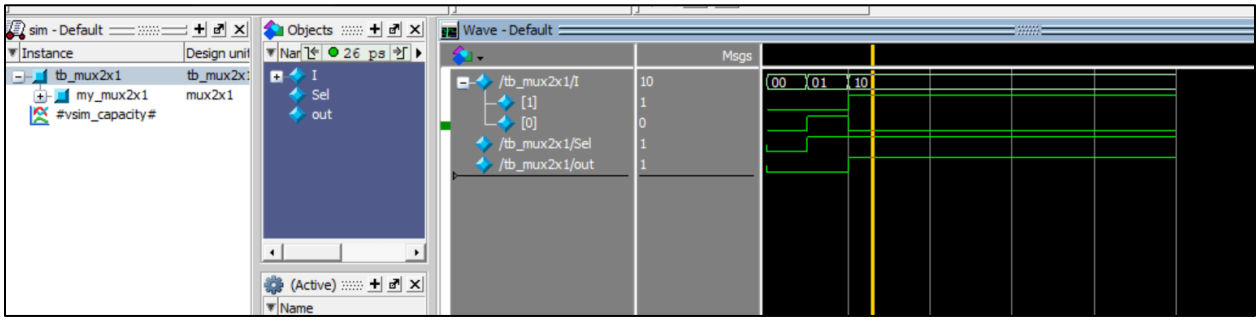
Test bench Code:

Task4.asm | mux2x1.v | tb_mux2x1.v | tb_demux.v | tb_mux4x1.v | mux_4x1.v | task2.v | tb_task2.v | task3.v | tb_task3.v

```
1  module tb_mux2x1();
2      reg [1:0] I;
3      reg Sel;
4      wire out;
5
6      mux2x1 my_mux2x1(I, Sel, out);
7
8      initial begin
9      I = 2'b00;
10     Sel = 0;
11     #10;
12
13     I = 2'b01;
14     Sel = 1;
15     #10;
16
17     I = 2'b10;
18     Sel = 1;
19     #10;
20     end
21
22 endmodule
```

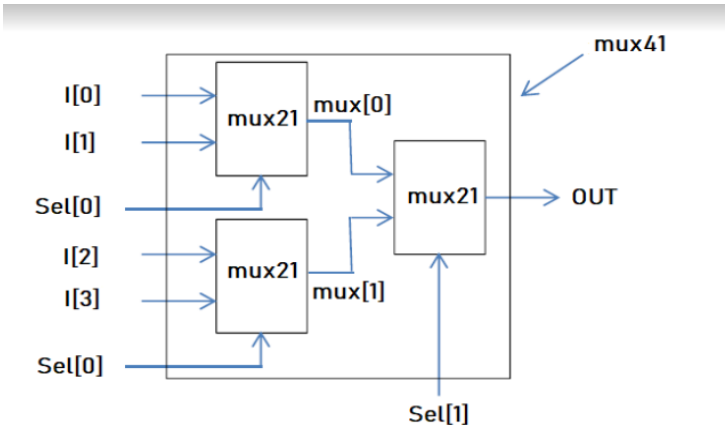
Output:





Task 2:

Implement 4X1 MUX using 2X1 MUX.



Code:

DUT Code:

Task4.asm | tb_mux2x1.v | tb_demux.v | tb_mux4x1.v | mux_4x1.v | mux2x1.v | task2.v | tb_task2.v | task3.v | tb_task3.v

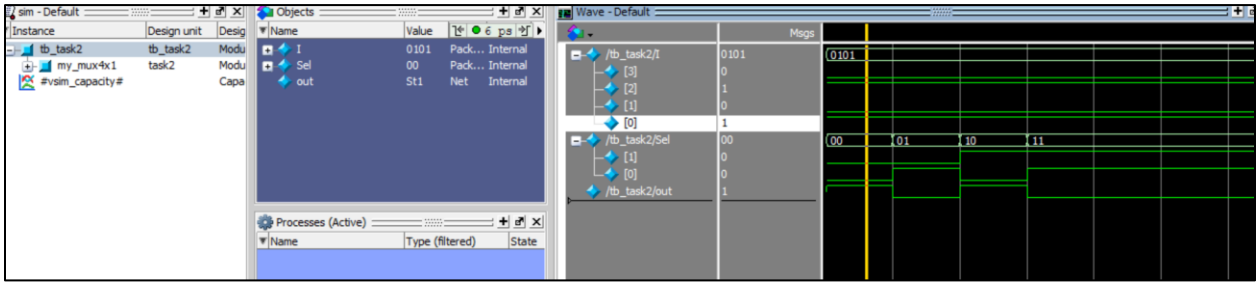
```
1   module task2(I, Sel, Out);
2       input [3:0] I;
3       input [1:0] Sel;
4       wire O1, O2;
5       output Out;
6
7       mux2x1 mux1({I[1],I[0]}, Sel[0], O1);
8       mux2x1 mux2({I[3],I[2]}, Sel[0], O2);
9       mux2x1 mux3({O2,O1}, Sel[1], Out);
10
11  endmodule
12
13
```

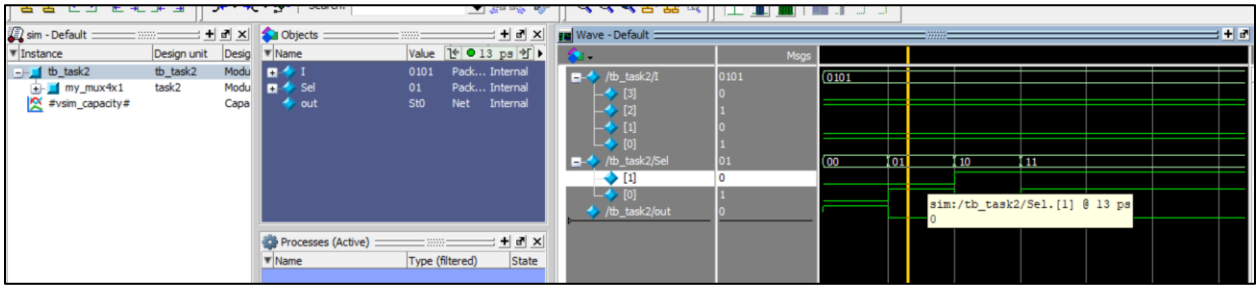
Test bench Code:

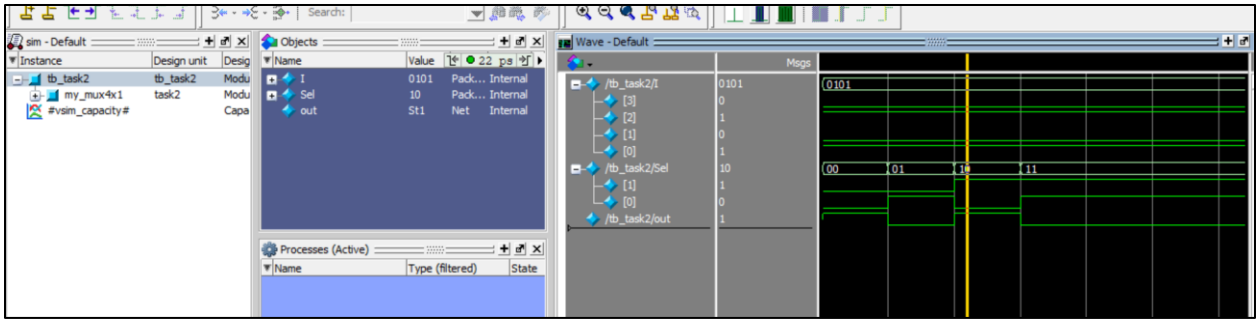
Task4.asm | tb_mux2x1.v | tb_demux.v | tb_mux4x1.v | mux_4x1.v | mux2x1.v | task2.v | tb_task2.v | task3.v | tb_task3.v

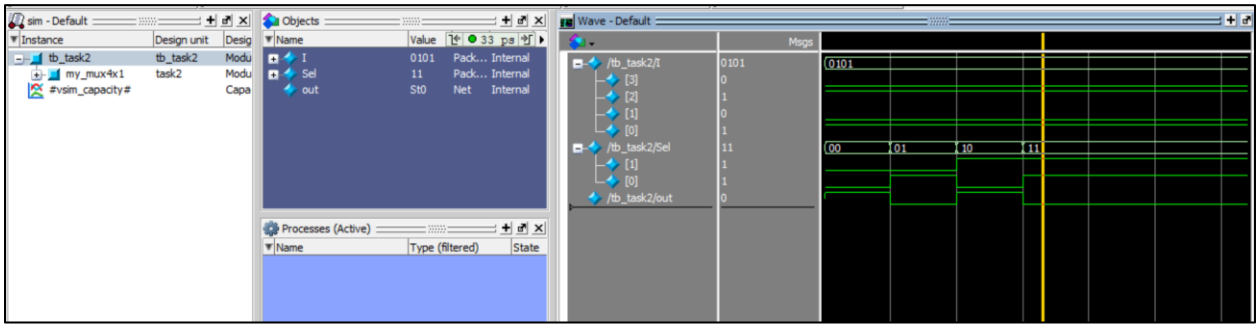
```
1  module tb_task2();
2      reg [3:0] I;
3      reg [1:0] Sel;
4      wire out;
5      task2 my_mux4x1(I, Sel, out);
6
7      initial begin
8      I = 4'b0101;
9      Sel = 2'b00;
10     #10;
11
12     I = 4'b0101;
13     Sel = 2'b01;
14     #10;
15
16     I = 4'b0101;
17     Sel = 2'b10;
18     #10;
19
20     I = 4'b0101;
21     Sel = 2'b11;
22     #10;
23     end
```

Output:









Task 3:

Design MUX where if the select to a MUX is 00 it will output A , 01 the output will be B , for 10 the output will be A+B and for 11 the output will be A-B.

Code:

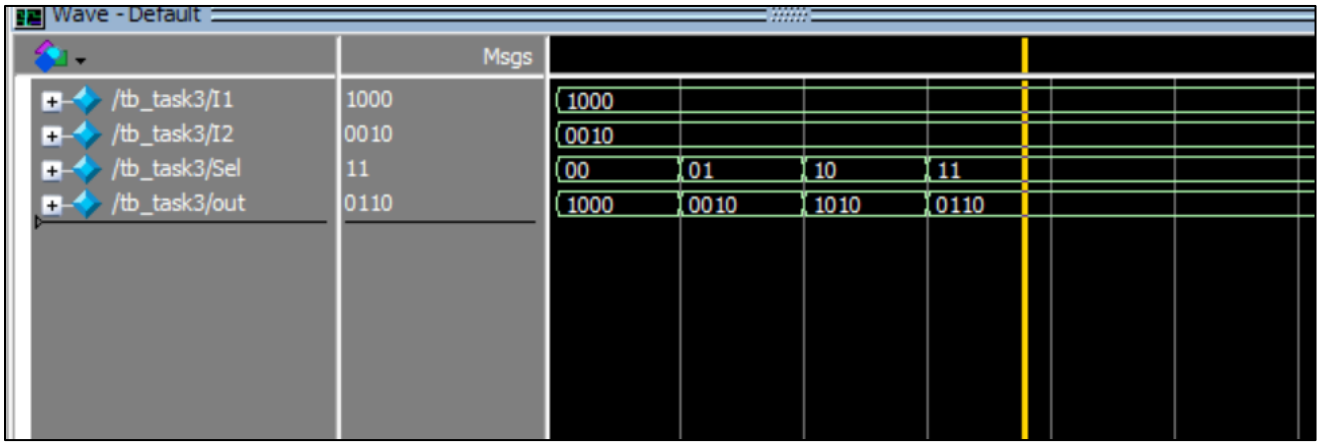
DUT Code:

```
module task3(A, B, Sel, Out);
    input [3:0] A;
    input [3:0] B;
    input [1:0] Sel;

    output [3:0] Out;
    reg  [3:0] Out;

    always@(A, B, Sel)
        case(Sel)
            2'b00 : Out = A;
            2'b01 : Out = B;
            2'b10 : Out = A + B;
            2'b11 : Out = A - B;
        endcase
endmodule
```

Test bench Code:

```
module tb_task3();
    reg [3:0] I1;
    reg [3:0] I2;
    reg [1:0] Sel;
    wire [3:0] out;
    task3 my_task3(I1, I2, Sel, out);

    initial begin
    I1 = 4'b1000;
    I2 = 4'b0010;
    Sel = 2'b00;
    #10;

    I1 = 4'b1000;
    I2 = 4'b0010;
    Sel = 2'b01;
    #10;

    I1 = 4'b1000;
    I2 = 4'b0010;
    Sel = 2'b10;
    #10;

    I1 = 4'b1000;
    I2 = 4'b0010;
    Sel = 2'b10;
    #10;

    I1 = 4'b1000;
    I2 = 4'b0010;
    Sel = 2'b11;
    #10;
    end
endmodule
```

Output:



Conclusion:

In this lab, I learned how to implement Multiplexers using always statement (behavior level modelling) in Verilog.