**MULTIPLEXERS IN**

**VERILOG**

**LAB # 0****7**

**Fall 2023**

**CSE-304L**

**Computer Organization & Architecture Lab**

Submitted by: **AIMAL KHAN**

Registration No.: **21PWCSE1996**

Class Section: **A**

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



Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted to:

**Dr. Bilal Habib**

Friday, December 8, 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] |  |
| 1. **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] |  |
| 1. **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] |  |
| 1. **Procedure** | All the instructions are written with proper procedure  [Marks 20] | Some steps are missing  [Marks 10] | steps are totally missing  [Marks 0] |  |
| 1. **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] |  |
| 1. **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]  [Marks 0] |  |
| 1. **Cheating** |  |  | Any kind of cheating will lead to 0 Marks |  |
| Total Marks Obtained: \_\_\_\_\_\_\_\_\_\_  Instructor Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | |

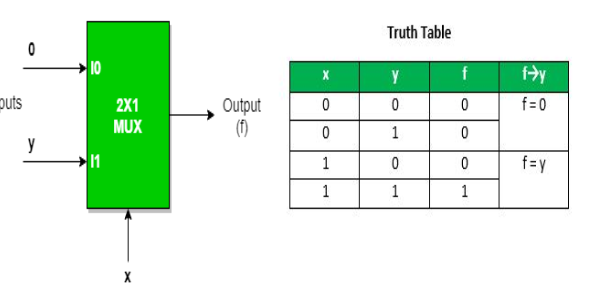
**Multiplexers in Verilog**

Objectives:

* Designing multiplexers in Verilog.

Tasks:

**Task 1**: Implement 2X1 MUX in Verilog.



**DUT Code:**

module Mux\_2x1 (I, Sel, Out);

input [1:0] I; // 2-bit input bus

input Sel; // 1-bit select input

output Out; // Output

reg Out; // Output register

always @(I, Sel)

case(Sel)

1'b0: Out = I[0]; // If Sel is 0, Out is I[0]

1'b1: Out = I[1]; // If Sel is 1, Out is I[1]

endcase

endmodule;

**Test Code:**

module test\_Mux\_2x1 ();

reg [1:0] I;

reg Sel;

wire Out;

Mux\_2x1 m2x1(I, Sel, Out);

initial

begin

#10

I[0] = 1;

I[1] = 1;

Sel = 0;

#10

I[0] = 1;

I[1] = 0;

Sel = 1;

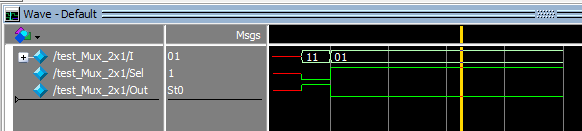
end

initial

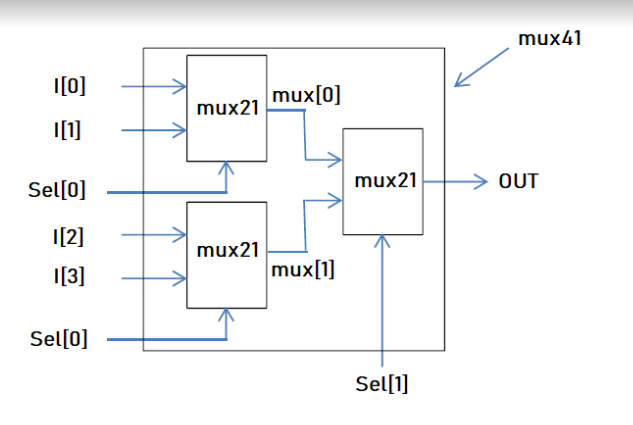
$monitor("%d %b %b %b %b", $time, I[0], I[1], Sel, Out);

endmodule;

**Output:**

****

**Task 2**: Implement 4X1 MUX using 2X1 MUX.



**DUT Code:**

module Mux\_4to1\_using\_Mux2x1 (I, Sel, Out);

input [3:0] I; // 4-bit input bus

input [1:0] Sel; // 2-bit select input

output Out; // Output

wire [1:0] MuxOut; // Output from each 2-to-1 MUX

Mux\_2x1 mux0 (

.I({I[0], I[1]}), // Select inputs {I[0], I[1]}

.Sel(Sel[1]), // Select bit Sel[1]

.Out(MuxOut[0]) // Output of the first MUX

);

Mux\_2x1 mux1 (

.I({I[2], I[3]}), // Select inputs {I[2], I[3]}

.Sel(Sel[1]), // Select bit Sel[1]

.Out(MuxOut[1]) // Output of the second MUX

);

Mux\_2x1 mux2 (

.I({MuxOut[0], MuxOut[1]}), // Select inputs {MuxOut[0], MuxOut[1]}

.Sel(Sel[0]), // Select bit Sel[0]

.Out(Out) // Final output

);

endmodule;

**Test Code:**

module test\_Mux\_4to1\_using\_Mux2x1();

reg [3:0] I;

reg [1:0] Sel;

wire Out;

Mux\_4to1\_using\_Mux2x1 uut (I, Sel, Out);

initial begin

// Initialize signals

I = 4'b0000;

Sel = 2'b00;

// Apply test case 1

Sel = 2'b00;

#10; // Wait for some time to stabilize

$display("Test case 1: Sel = %b, I = %b, Out = %b", Sel, I, Out);

// Apply test case 2

Sel = 2'b01;

#10; // Wait for some time to stabilize

$display("Test case 2: Sel = %b, I = %b, Out = %b", Sel, I, Out);

// Apply test case 3

Sel = 2'b10;

#10; // Wait for some time to stabilize

$display("Test case 3: Sel = %b, I = %b, Out = %b", Sel, I, Out);

// Apply test case 4

Sel = 2'b11;

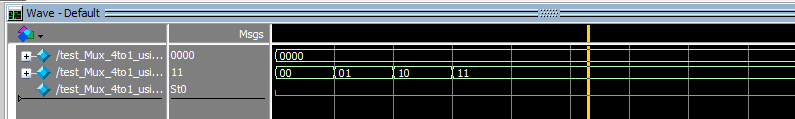
#10; // Wait for some time to stabilize

$display("Test case 4: Sel = %b, I = %b, Out = %b", Sel, I, Out);

end

endmodule;

**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.

**DUT Code:**

module Custom\_Mux (

input A,

input B,

input [1:0] Sel,

output Y

);

wire Y\_AB, Y\_A\_minus\_B;

// MUX for A and B selection

assign Y\_AB = (Sel == 2'b00) ? A : B;

// Calculate A - B

assign Y\_A\_minus\_B = A - B;

// Output selection based on Sel

assign Y = (Sel == 2'b00) ? A :

(Sel == 2'b01) ? B :

(Sel == 2'b10) ? Y\_AB :

(Sel == 2'b11) ? Y\_A\_minus\_B : 0; // Default to 0 if Sel is invalid

endmodule;

**Test Code:**

module test\_Custom\_Mux();

reg A;

reg B;

reg [1:0] Sel;

wire Y;

Custom\_Mux uut (

.A(A),

.B(B),

.Sel(Sel),

.Y(Y)

);

initial begin

// Initialize signals

A = 1;

B = 2;

Sel = 2'b00;

// Apply test case 1: Sel = '00' (output A)

#10; // Wait for some time to stabilize

$display("Test case 1: Sel = %b, A = %b, B = %b, Y = %b", Sel, A, B, Y);

// Apply test case 2: Sel = '01' (output B)

Sel = 2'b01;

#10; // Wait for some time to stabilize

$display("Test case 2: Sel = %b, A = %b, B = %b, Y = %b", Sel, A, B, Y);

// Apply test case 3: Sel = '10' (output A + B)

Sel = 2'b10;

#10; // Wait for some time to stabilize

$display("Test case 3: Sel = %b, A = %b, B = %b, Y = %b", Sel, A, B, Y);

// Apply test case 4: Sel = '11' (output A - B)

Sel = 2'b11;

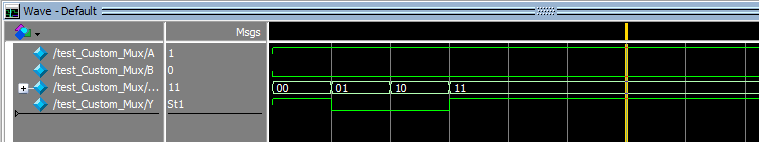
#10; // Wait for some time to stabilize

$display("Test case 4: Sel = %b, A = %b, B = %b, Y = %b", Sel, A, B, Y);

end

endmodule;

**Output:**

****

Reference:

To view my codes, please refer to my [GitHub Account.](https://github.com/aimalexe/DCSE/tree/main/semester_5_(fall-23)/computer_organization_and_architechure_lab/lab_reports/)

Conclusion:

In conclusion I am now able to design multiplexers and test my design too.

The End.