Name: Muhammad Furrukh

Section: C

Repository: https://github.com/Muhammad-Furrukh/DSD_2022-EE-111.git

Reg. No.: 2022-EE-111

EE-272L Digital Systems Design

Marks Obtained: ____________

Lab Manual

DSD Lab Manual Evaluation Rubrics

Assessment	Total Marks	Marks Obtained	0-30%	30-60%	70-100%
Code Organization	3		No Proper Indentation and descriptive naming, no code organization. Zero to Some understanding but not working	Proper Indentation or descriptive naming or code organization. Mild to Complete understanding but not working	Proper Indentation and descriptive naming, code organization. Complete understanding, and proper working
Simulation	5		Simulation not done or incorrect, without any understanding of waveforms	Working simulation with errors, don't cares's(x) and high impedance(z), partial understanding of waveforms	Working simulation without any errors, etc and complete understanding of waveforms
FPGA	2		Not implemented on FPGA and questions related to synthesis and implementation not answered.	Correctly Implemented on FPGA or questions related to synthesis and implementation answered.	Correctly Implemented on FPGA and questions related to synthesis and implementation answered.

Experiment 2

Combinational Circuits: Structural Modeling Simulation

In this lab, we are going to simulate a simple combinational circuit (Fig. 2.1) using QuestaSim and Xcelium as our simulators.

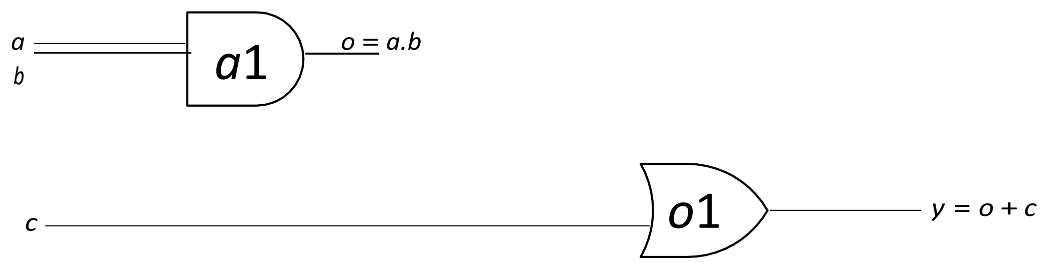


Fig. 2.1: Simple logic circuit implementing $y = a.b + c$

Codes

Make a folder containing System Verilog RTL (Listing 2) and test bench files (Listing 3). Use code editor to write the codes provided.

1. Remember to keep the file name and module name same.
2. Simulation depends upon how time is defined as the simulator needs to know how to interpret the delay provided as *#number*. The '*timescale* directive specifies the timescale and precision for the modules.

```
`timescale <time_unit>/<time_percision>
`timescale 1ns/10ps
// time unit = 1 ns, time precision = 10 ps
```

Listing 1: Timescale

Time unit and time precision can be thought of like the scale along axis in graphs where 1 big square = time_unit and 1 small square = time precision.
It can either be skipped and the simulator will refer to its default or should be added as the first line of both RTL and testbench codes.

```
module lab2(output logic y,
           input  logic a,b,c
           );
  wire o;
  and a1(o, a, b);
  or  o1(y, o, c);
endmodule
```

Listing 2: System Verilog Code.

Overview of QuestaSim

Following steps are used to create a project in QuestaSim.

1. Click on *File* then select *New* and then *Project*. (Fig. 2.2).

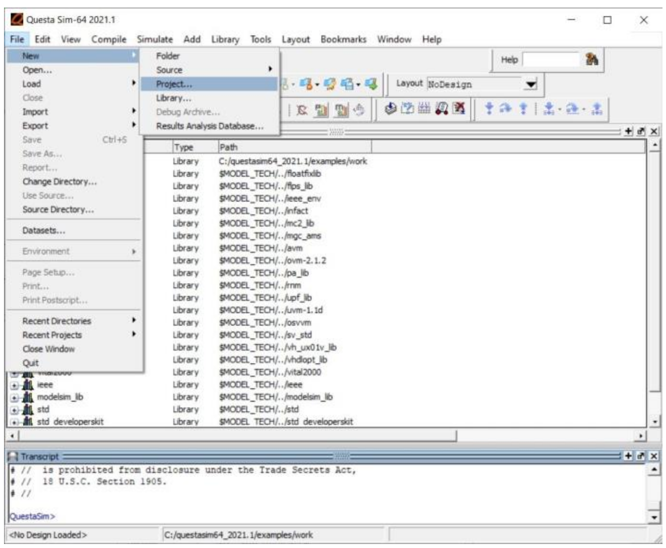


Fig. 2.2: Creating a new project.

2. When prompted, enter the project name and browse the *Project Location* to the folder containing RTL code and the test bench code. Then click on *OK* (Fig. 2.3).

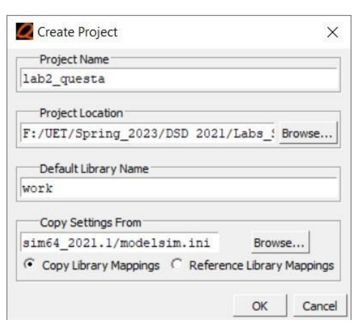


Fig. 2.3: Create Project window.

3. A new window will open (Fig. 2.4) select *Add Existing File*.

```
module lab2_tb;

logic  a1;
logic  b1;
logic  c1;
logic  y1;

localparam period = 10;

lab2 UUT(
.a(a1),
.b(b1),
.c(c1),
.y(y1)
);

initial //initial block executes only once
begin
        // Provide different combinations of the inputs to check validity of code
        a1 = 0; b1 = 0; c1 = 0;
        #period;
        a1 = 0; b1 = 0; c1 = 1;
        #period;
        a1 = 0; b1 = 1; c1 = 0;
        #period;
        a1 = 0; b1 = 1; c1 = 1;
        #period;
        a1 = 1; b1 = 0; c1 = 0;
        #period;
        a1 = 1; b1 = 0; c1 = 1;
        #period;
        a1 = 1; b1 = 1; c1 = 0;
        #period;
        a1 = 1; b1 = 1; c1 = 1;
        #period;
        $stop;
end
initial
begin
        /*the following system task will print out the signal values
        every time they change on the Transcript Window */
        $monitor("y=%b, a=%b, b=%b, c=%b", y1,a1,b1,c1);
end
endmodule
```

Listing 3: Test bench simulation Code

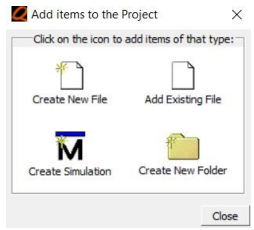


Fig. 2.4: Add items to the Project window.

4. Next click on *Browse* (Fig. 2.5) and select the System Verilog RTL and test bench code file (Fig. 2.6), click *Open* and then click *OK*.

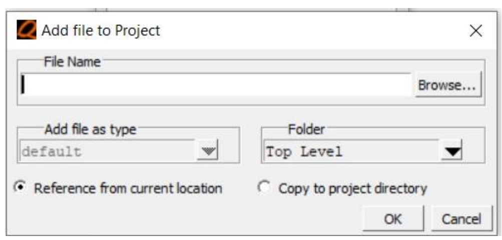


Fig. 2.5: Add File to the Project window.

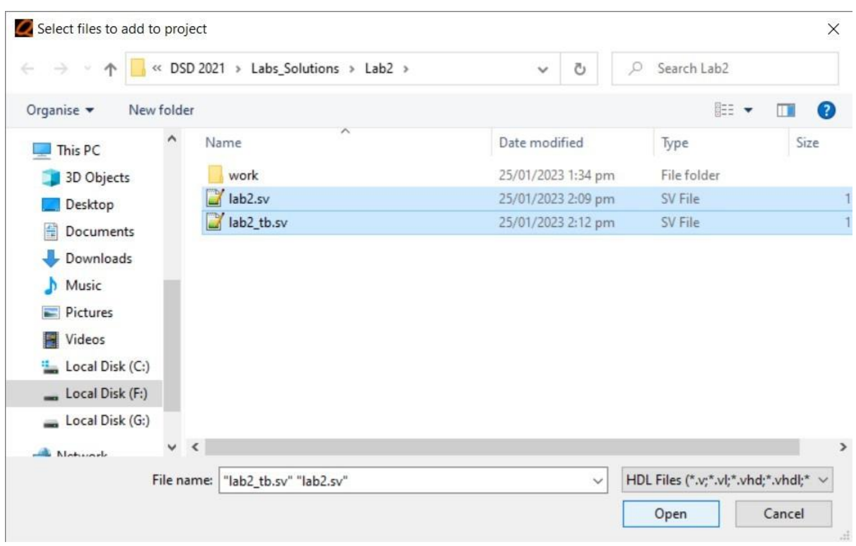


Fig. 2.6: Adding RTL and test bench files to project.

5. The two files will be added to QuestaSim project (Fig. 2.7).

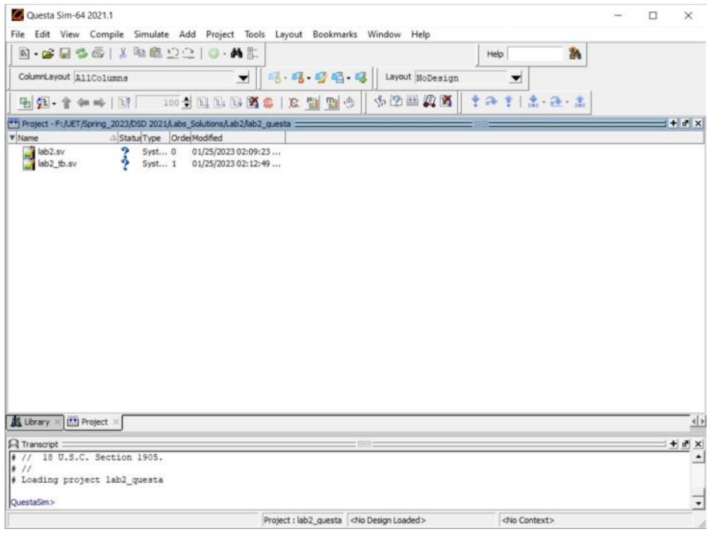


Fig. 2.7: QuestaSim project created.

6. Click on *compile all* (Fig. 2.8). If there are no errors the transcript will show no errors (Fig. 2.9).Otherwise the Transcript window will show the number of errors, and can be viewed by clicking on the line in the transcript which provides information about the number of errors, written in red.

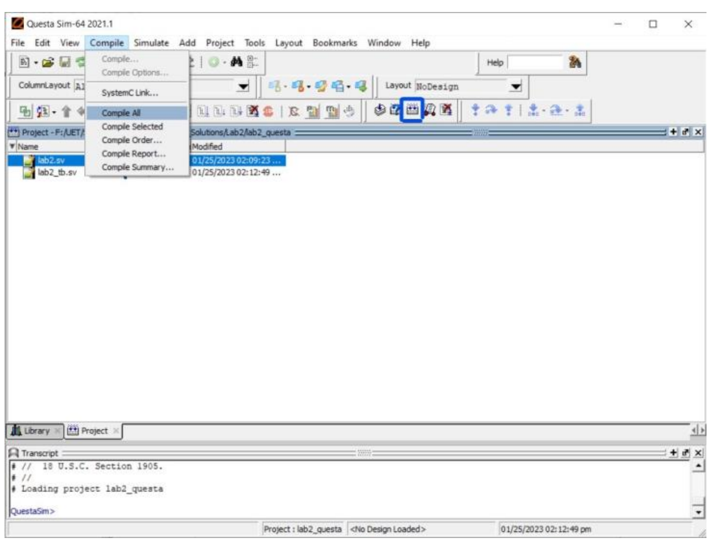


Fig. 2.8: Compiling on QuestaSim.

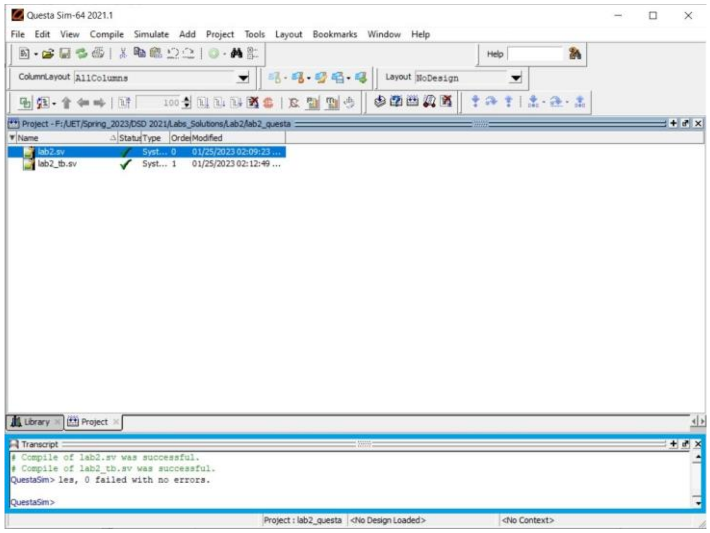


Fig. 2.9: QuestaSim project created.

7. Then go to *Library* tab. Expand the *work* tab and click the test bench file (in our case lab2 tb). And select *Simulate*. (Fig. 2.10).

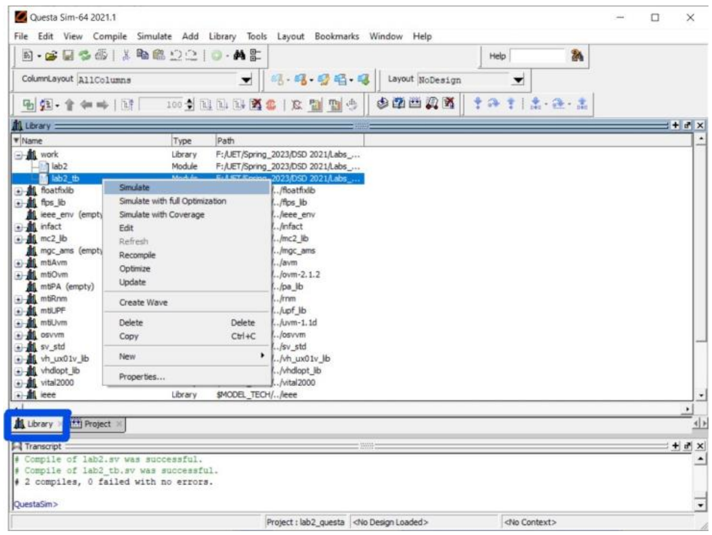


Fig. 2.10: Start Simulation.

8. QuestaSim will load the simulation on the window.

9. Check if the *Objects* dark blue box is appearing in the window. If this box does not appear click on *View* and select *Objects*. Select any of the objects that were inputs and outputs of the code written and then click on *Add to* then select *Wave* and select *Signals in Region*. (Fig. 2.11).

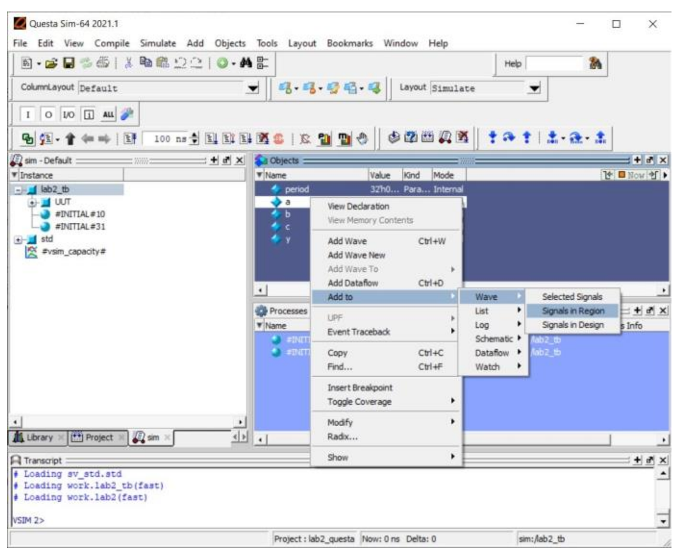


Fig. 2.11: Adding waves to the Signals in Objects box (dark blue box) in the upper right corner.

10. For viewing local signals of your design along-with inputs and outputs, instead of the previous step, click on the instance name of your design module (in this case *UUT*). Now the object window will show all the signals present in your design file. Select any of the objects and then click on *Add to* then select *Wave* and select *Signals in Region*.

11. The *Wave* window will appear. Select the run length (80 ns) as shown in Fig. 2.12.

Fig. 2.12: Selecting Run Length (80ns).

12. Click on *Run*. The simulation waveform will appear on the screen. Go to the *View* tab, click *Zoom* and select *Zoom Full* (Fig. 2.13) (or you can press *F* key on your keyboard), to view the output of y for all the various inputs (Fig. 2.14).

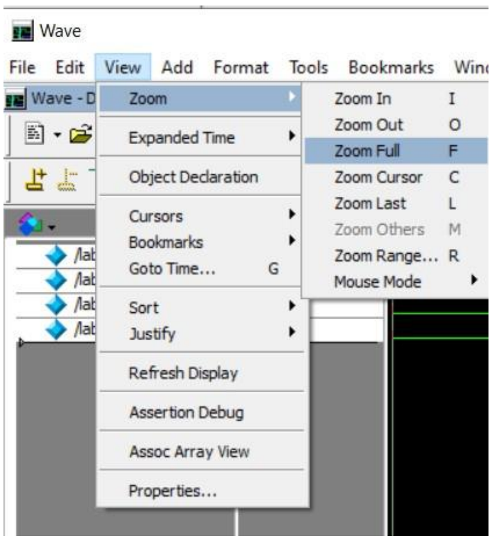


Fig. 2.13: Zoom to view entire waveform.

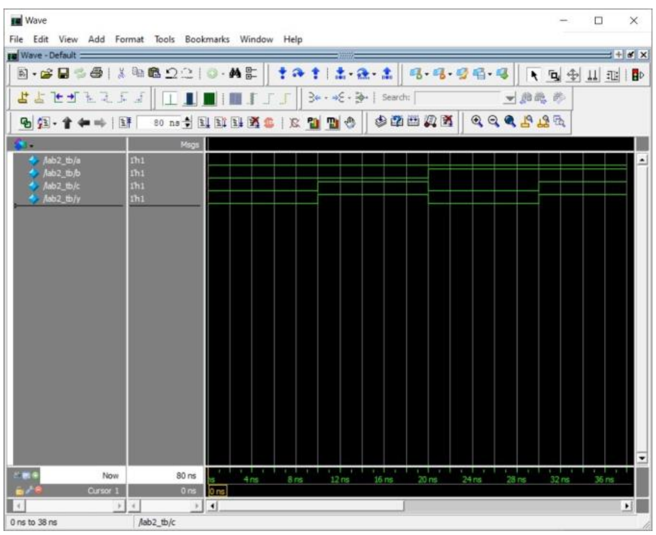


Fig. 2.14: Output wave-forms of the logic circuit.

13. If you want to restart your simulation, then click on *Restart* icon (Fig. 2.12).

Overview of Xcelium

1. For running simulation on Xcelium, Open the Linux Terminal and enter the following command:

```
xrun filename.sv testbench.sv access +rwc
```

For this lab,

```
xrun lab2.sv lab2_tb.sv access +rwc
```

The result will be displayed on the terminal.

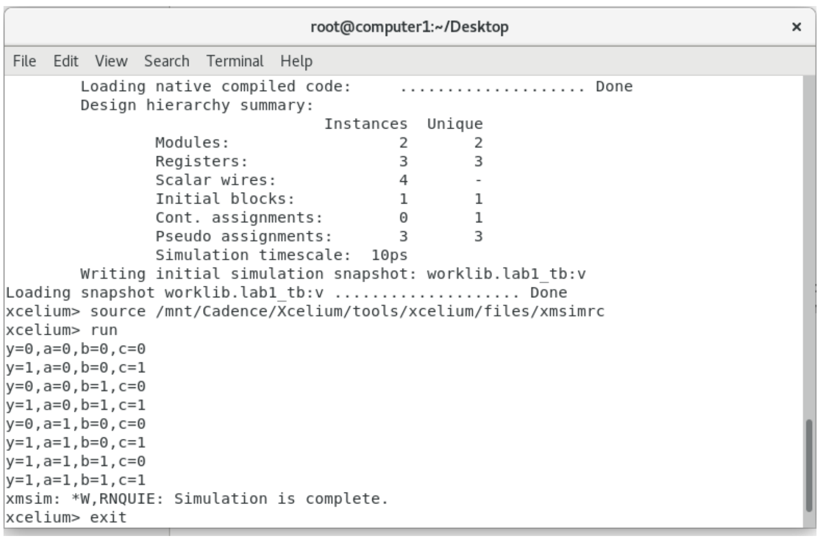


Fig. 2.15: Simulation Results displayed on the terminal

2. For viewing the waveforms, enter the following command:

```
xrun filename.sv testbench.sv access +rwc gui
```

For this lab,

```
xrun lab2.v lab2 tb.v access +rwc gui
```

SimVision GUI will open.

3. Click on the module name and select *Add to Waveform Window*.

4. Click *run* to view the simulation results.

Fig. 2.16: Simulation Waveforms on SimVision

Tasks

1. Implement the circuit shown in Fig. 2.17 on QuestaSim and verify it's truth table.

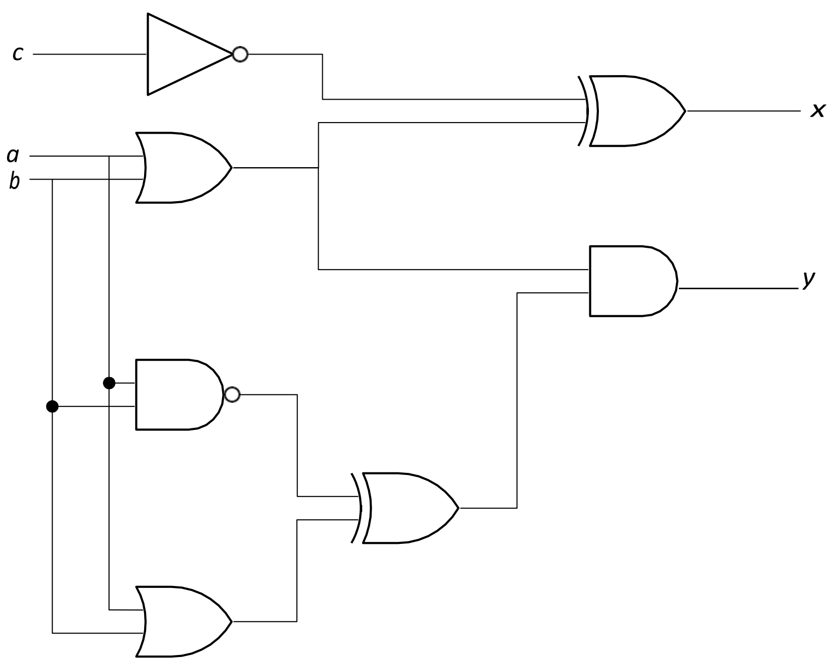


Fig. 2.17: Circuit to be implemented.

2. RTL and testbench codes of a full adder circuit are provided in Listings 4 and 5. Compile and simulate these in Questasim and correct the errors. The Equations for Sum and Carry are:

$$Sum = (A \otimes B) \otimes C$$
$$Carry = A.B + C(A \otimes B)$$

```
module full_adder(
input  logic a,
input  logic b,
input  logic c,
output logic sum,
output logic carry,
);

sum = (a ^ b) ^ c;
assign carry = (a & b) | (c(a ^ b));

endmodule
```

Listing 4: System Verilog Code.

```
module full_adder_tb();
logic  a1;
logic b1;
logic c1;
logic sum1;

full_adder (
.a(a1),
.b(b1),
.c(c1),
.sum(sum1),
.carry(carry1)
);

initial
begin
// Provide different combinations of the inputs to check validity of code
a = 0; b = 0; c = 0;
#10;
 a1 = 0; b1 = 0; c = 1;
#10;
 a1 = 0; b1 = 1; c1 = 0;
#10;
 a1 = 0; b = 1; c1 = 1;
#10;
 a1 = 1; b1 = 0; c1 = 0;
#10;
 a1 = 1; b1 = 0; c1 = 1;
#10;
a = 1; b1 = 1; c1 = 0;
#10
a1 = 1; b1 = 1; c1 = 1;
#10;
$stop;


endmodule
```

Listing 5: System Verilog Code.

Deliverables

1. A report including:
   (a) Truthtable of the circuit shown in Fig. 2.17.
   (b) Errors found in the codes (Listing 4 and 5).
   (c) Corrected codes.
2. System Verilog code of the given circuit in Fig. 2.17 using structural coding (using primitives such as AND, OR, etc., gates).
3. Testbench code of the same.
4. Perform the simulation of the circuit Task1 and Task2 on QuestaSim. And show the results of that simulation to the Instructor.

The collaboration between students is encouraged, but blind code sharing/copying is not allowed. If you are unable to explain anything in your code, it will be assumed you have copied it. So make sure you know every thing you have written in your code. We are least concerned about how you have learnt something as long as you have learnt it well. Copied assignments will get ZERO marks.

Acknowledgments

The manual has been written by Mr. Ali Imran and Ms. Shehzeen Malik.

1.

Report

a) Truth table circuit in Fig. 2.17.

a	**b**	**c**	**o1**	**o2**	**o3**	**o4**	**x**	**y**
0	0	0	1	0	1	1	1	0
0	0	1	0	0	1	1	0	0
0	1	0	1	1	1	0	0	0
0	1	1	0	1	1	0	1	0
1	0	0	1	1	1	0	0	0
1	0	1	0	1	1	0	1	0
1	1	0	1	1	0	1	0	1
1	1	1	0	1	0	1	1	1

b) Errors in listings 4 and 5:

i) There should be no ‘,’ after carry in line 6 of listing 4

ii) No ‘assign’ used with sum in line 9 of listing 4

iii) ‘&’ missing between ‘c’ and ‘(a ^ b)’ in line 10 of listing 4

iv) ‘logic carry1’ missing in listing 5

v) ‘UUT’ missing in line 7 of listing 5

vi) ‘a’, ‘b’ and ‘c’ written instead of ‘a1’, ‘b1’ and ‘c1’, respectively in line 18 of listing 5

vii) ‘end’ missing after ‘begin’ in listing 5

c) Corrected codes:

RTL code:

D: > Documents > UET Courses > 4th Semester > Digital Design > La

```
module full_adder(
input   logic a,
input   logic b,
input   logic c,
output  logic sum,
output  logic carry
);
assign sum = (a ^ b) ^ c;
assign carry = (a & b) | (c & (a ^ b));
endmodule
```

Test bench code:

D: > Documents > UET Courses > 4th Semester > Digital Design > Lab > Lab

```
module full_adder_tb();
logic a1;
logic b1;
logic c1;
logic sum1;
logic carry1;

full_adder UUT(
    .a(a1),
    .b(b1),
    .c(c1),
    .sum(sum1),
    .carry(carry1)
);
initial
begin
    a1 = 0; b1 = 0; c1 = 0;
    #10;
    a1 = 0; b1 = 0; c1 = 1;
    #10;
    a1 = 0; b1 = 1; c1 = 0;
    #10;
    a1 = 0; b1 = 1; c1 = 1;
    #10;
```

```
    a1 = 1; b1 = 0; c1 = 0;
    #10;
    a1 = 1; b1 = 0; c1 = 1;
    #10;
    a1 = 1; b1 = 1; c1 = 0;
    #10;
    a1 = 1; b1 = 1; c1 = 1;
    #10;
    $stop;
end
endmodule
```

2. Code for Fig. 2.17.

D: > Documents > UET Courses > 4th Semester > Digital

```
module lab2(output logic x, y,
            input logic a, b, c
            );
            wire o1, o2, o3, o4;
            not n1(o1, c);
            or r1(o2, a, b);
            nand na1(o3, a, b);
            xor x1(x, o1, o2);
            xor x2(o4, o3, o2);
            and a1(y, o2, o4);
endmodule
```

3. Test bench code for Fig. 2.17.

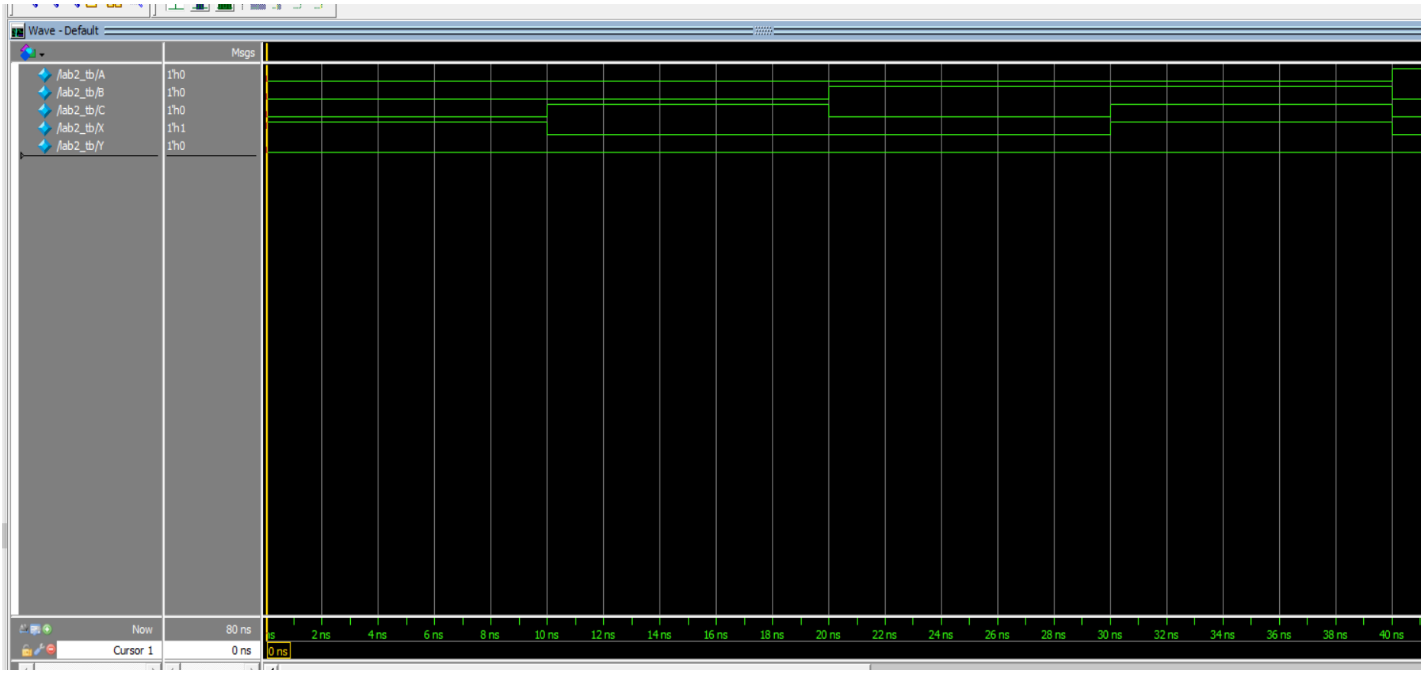
D: > Documents > UET Courses > 4th Semester > Digital Design > Lab > Lab 2 > lab2_tb.sv

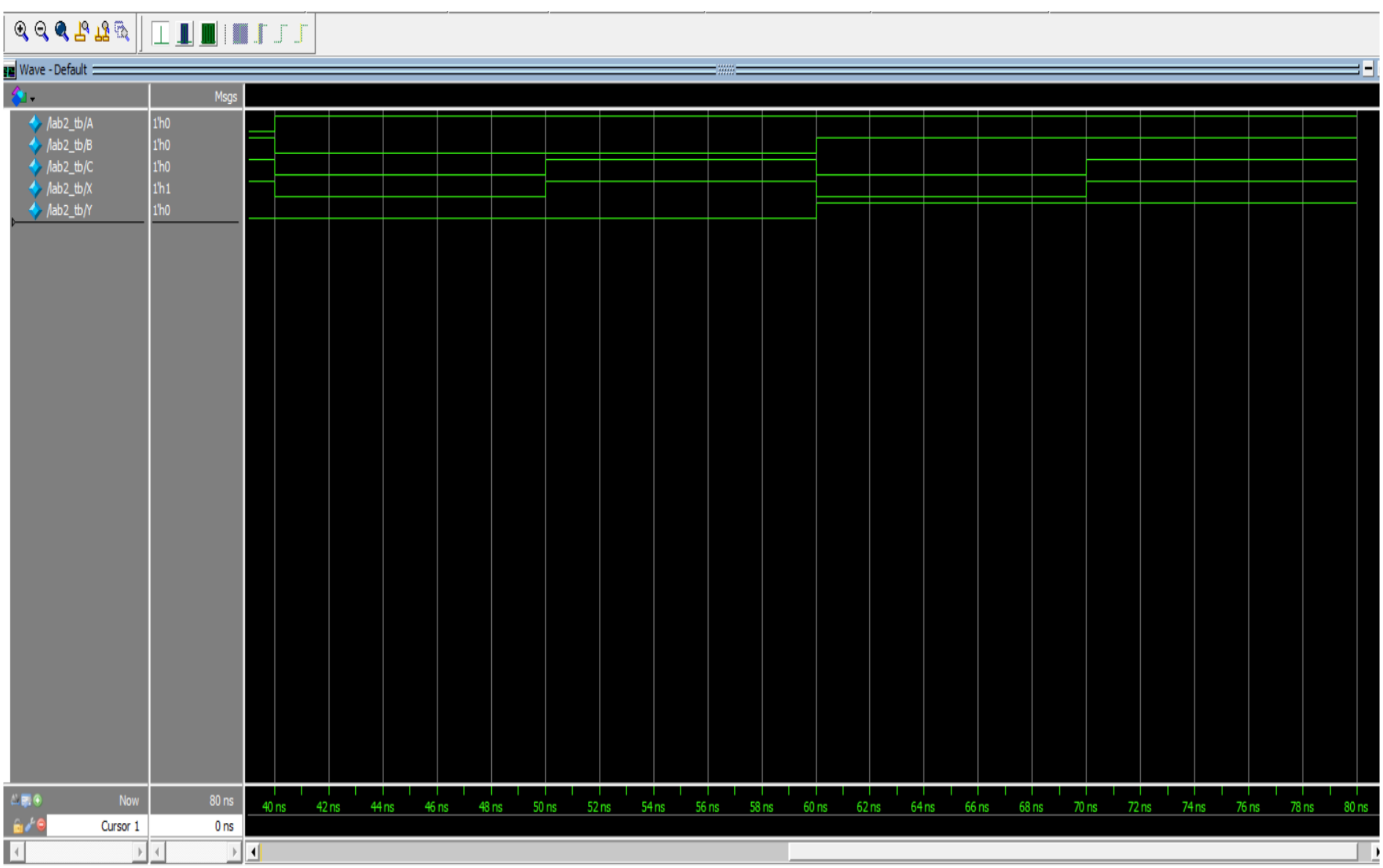
```
module lab2_tb;
logic A;
logic B;
logic C;
logic X;
logic Y;

lab2 UUT(
    .a(A),
    .b(B),
    .c(C),
    .x(X),
    .y(Y)
);
initial
begin
    A = 0; B = 0; C = 0;
    #10;
    A = 0; B = 0; C = 1;
    #10;
    A = 0; B = 1; C = 0;
    #10;
    A = 0; B = 1; C = 1;
    #10;
    A = 1; B = 0; C = 0;
    #10;
    A = 1; B = 0; C = 1;
    #10;
    A = 1; B = 1; C = 0;
    #10;
    A = 1; B = 1; C = 1;
    #10;
    $stop;
end
initial
begin
$monitor("x = %b, y = %b, a = %b, b = %b, c = %b", X, Y, A, B, C);
end
endmodule
```

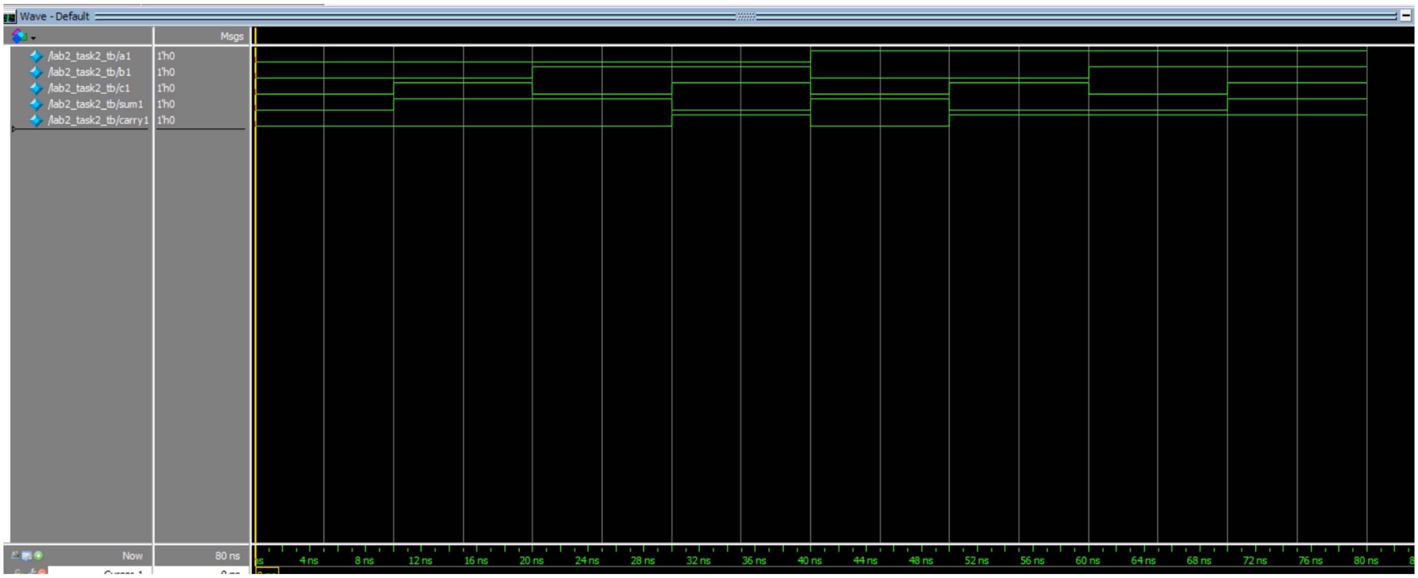
4. Simulation Results

Task 1:





Task 2:


Wave - Default
Msgs
/lab2_task2_tb/a1
/lab2_task2_tb/b1
/lab2_task2_tb/c1
/lab2_task2_tb/sum1
/lab2_task2_tb/carry1
1'h0
1'h0
1'h0
1'h0
1'h0
Now
80 ns
4 ns
8 ns
12 ns
16 ns
20 ns
24 ns
28 ns
32 ns
36 ns
40 ns
44 ns
48 ns
52 ns
56 ns
60 ns
64 ns
68 ns
72 ns
76 ns
80 ns