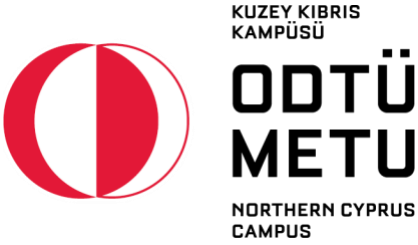

Lab 1 Report

Name: Mert Can Bilgin

Student ID: 2453025

PRELIMINARY WORK AND EXPERIMENTAL RESULTS

2.2.2 GATES

QUESTION 2.2.2.1

ii. Derive the truth table and Boolean expression for each logic gate.

X	Y	X'	Y'	X.Y	X+Y	(X.Y)'	(X+Y)'	X⊕Y	(X⊕Y)'
0	0	1	1	0	0	1	1	0	1
1	0	0	1	0	1	1	0	1	0
0	1	1	0	0	1	1	0	1	0
1	1	0	0	1	1	0	0	0	1

ii. Draw the corresponding logic schematics for all gates.

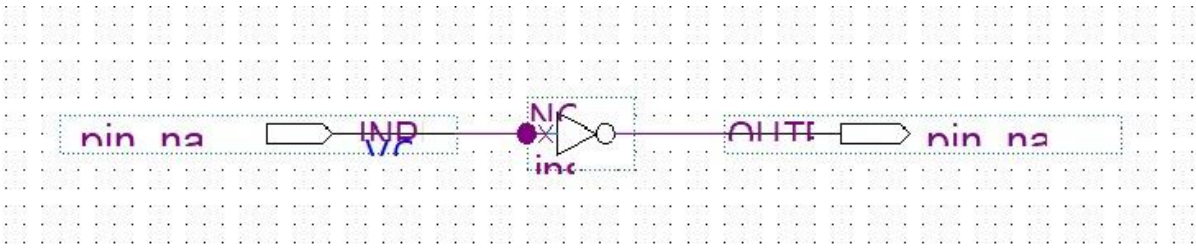

Figure 1: NOT Gate

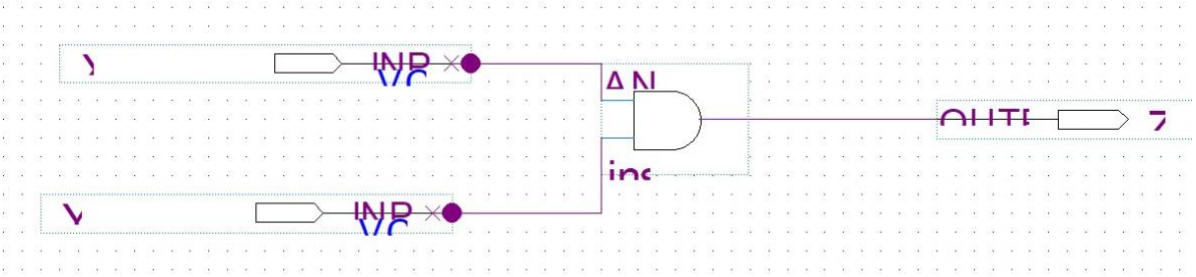

Figure 2: AND Gate

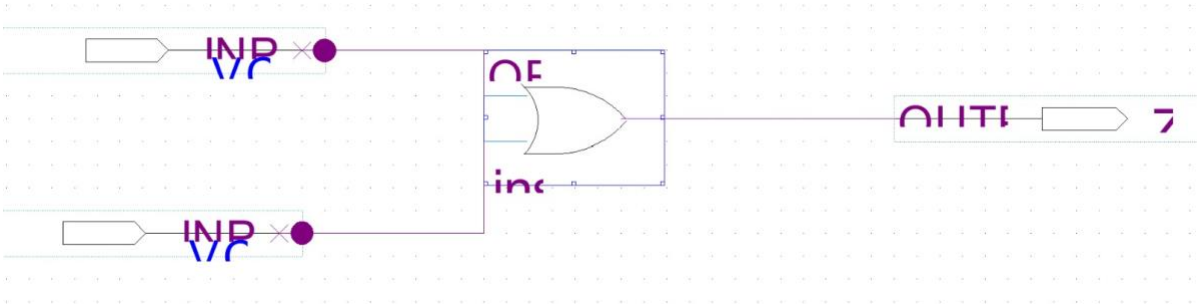

Figure 3: OR Gate

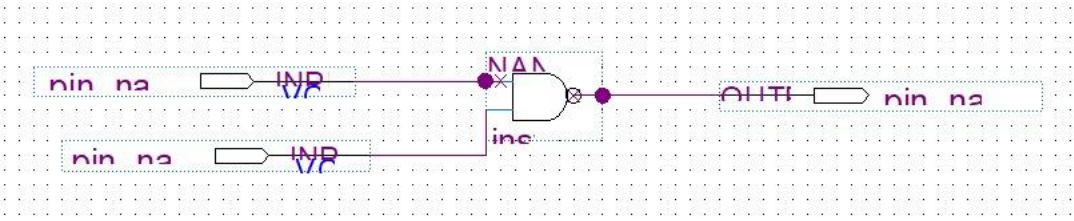

Figure 4: NAND Gate

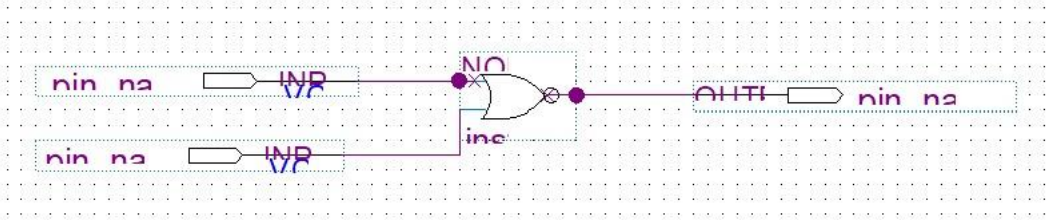

Figure 5: NOR Gate

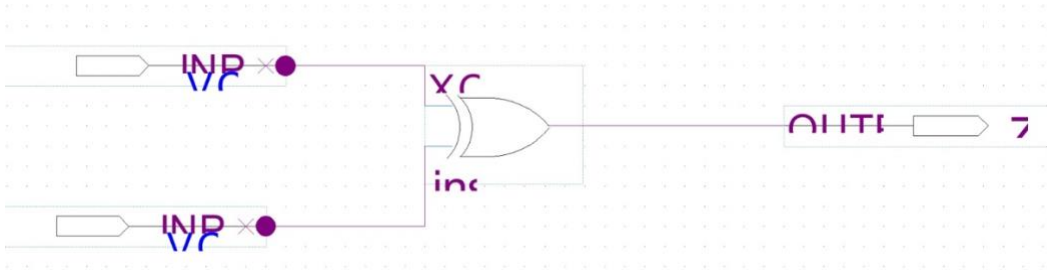

Figure 6: XOR Gate

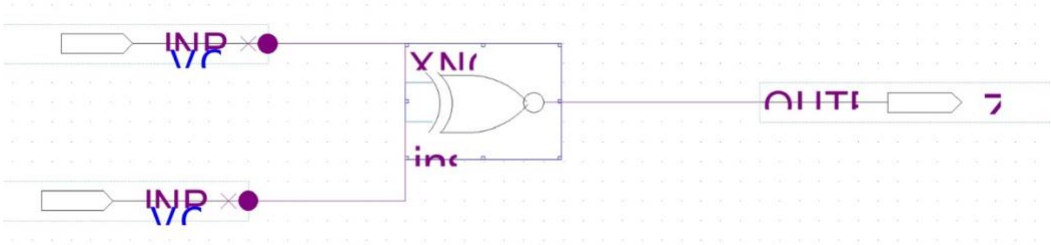

Figure 7: XNOR Gate

Question: 2.2.2.2

i. Write a structural Verilog code for each logic gate and simulate using Modelsim.

Verilog Code for NOT gate

```
1. module NOTGate(X, Y);
2. input X;
3. output Y;
4. not G1(Y, X);
5. endmodule
```

Verilog Code for AND gate

```
1. module ANDGate(X, Y, Z);
2. input X, Y;
3. output Z;
4. and G1(Z, X, Y);
5. endmodule
```

Verilog Code for OR gate

```
1. module ORGate(X, Y, Z);
2. input X, Y;
3. output Z;
4. or G1(Z, X, Y);
5. endmodule
```

Verilog Code for NAND gate

```
1. module NANDGate(X, Y, Z);
2. input X, Y;
3. output Z;
4. nand G1(Z, X, Y);
5. endmodule
```

Verilog Code for NOR gate

```
1. module NORGate(X, Y, Z);
2. input X, Y;
3. output Z;
4. nor G1(Z, X, Y);
5. endmodule
```

Verilog Code for XOR gate

```
1. module XORGate(X, Y, Z);
2. input X, Y;
3. output Z;
4. xor G1(Z, X, Y);
5. endmodule
```

Verilog Code for XNOR gate

```
1. module XNORGate(X, Y, Z);
2. input X, Y;
3. output Z;
4. xnor G1(Z, X, Y);
5. endmodule
```

TestBenches and Modelsim Simulations:

TestBench of NOT

```
1. module Not_tb();
2. reg A;
3. wire B;
4. Not DUT(A,B);
5. initial
6. begin
7. A = 1'b0; #100;
8. A = 1'b1; #100;
9. end
10.     endmodule
```

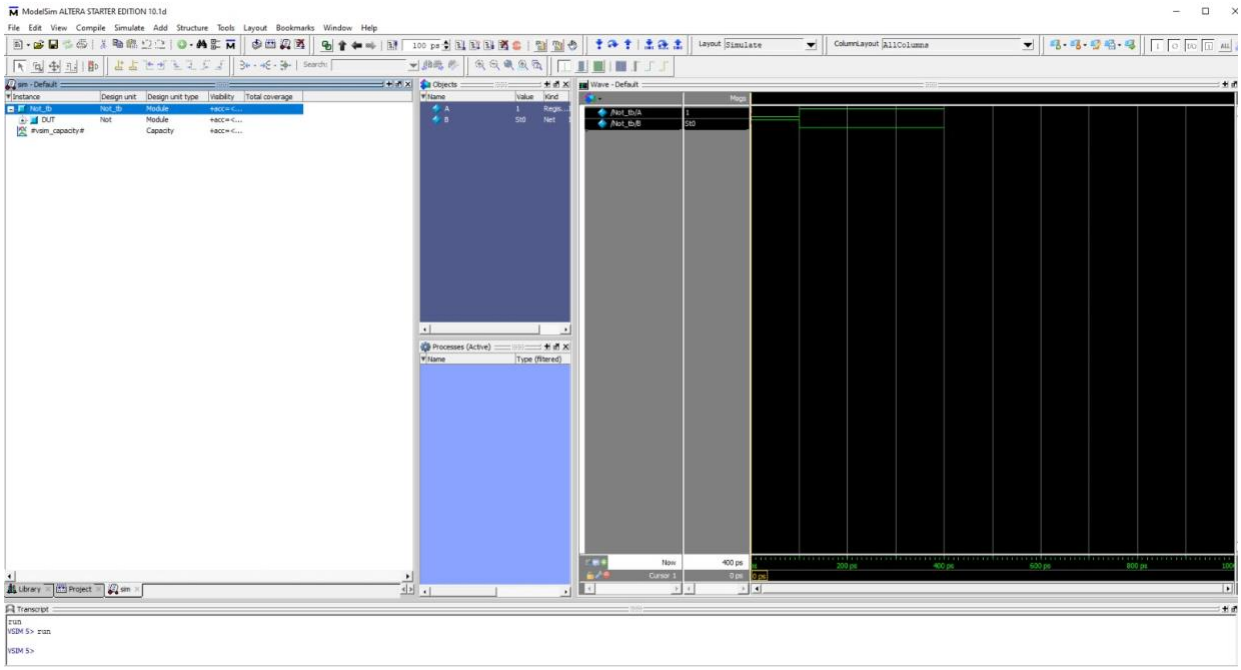

Figure 8: Simulation of NOT gate

Comments:

- By NOT gate, the input reversed. I can be sure that the simulation results are true by comparing with truth table.

TestBench of AND

```
1. module And_tb();
2. reg A, B;
3. wire C;
4. And DUT(A,B,C);
5. initial
6. begin
7. A = 1'b0; B = 1'b0; #100;
8. A = 1'b0; B = 1'b1; #100;
9. A = 1'b1; B = 1'b0; #100;
10.      A = 1'b1; B = 1'b1; #100;
11.      end
12.      endmodule
```

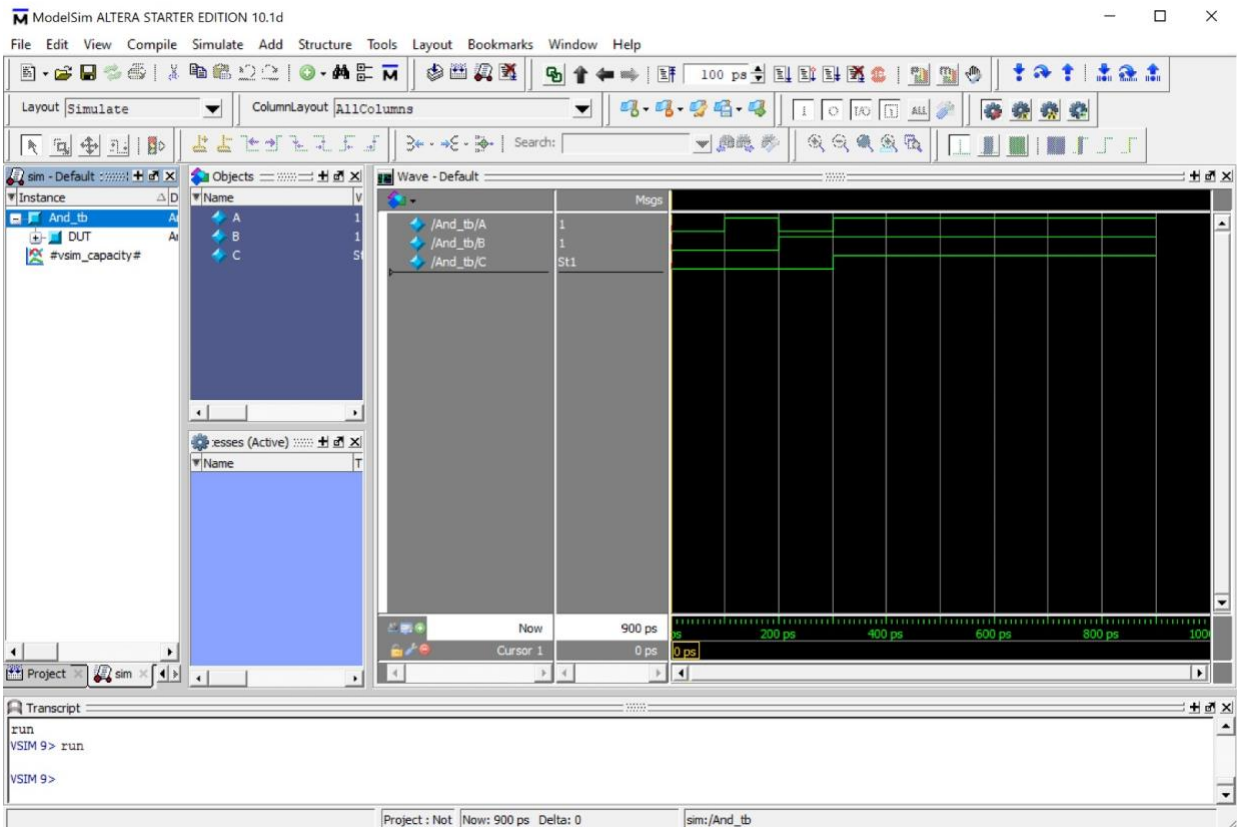

Figure 9: Simulation of AND gate

Comments:

- We only get the result as 1 when we have (1,1) as inputs. I can be sure that the simulation results are true by comparing with truth table.

TestBench of OR

```
1. module Or_tb();
2. reg A, B;
3. wire C;
4. Or DUT(A,B,C);
5. initial
6. begin
7. A = 1'b0; B = 1'b0; #100;
8. A = 1'b0; B = 1'b1; #100;
9. A = 1'b1; B = 1'b0; #100;
10.      A = 1'b1; B = 1'b1; #100;
11.      end
12.      endmodule
```

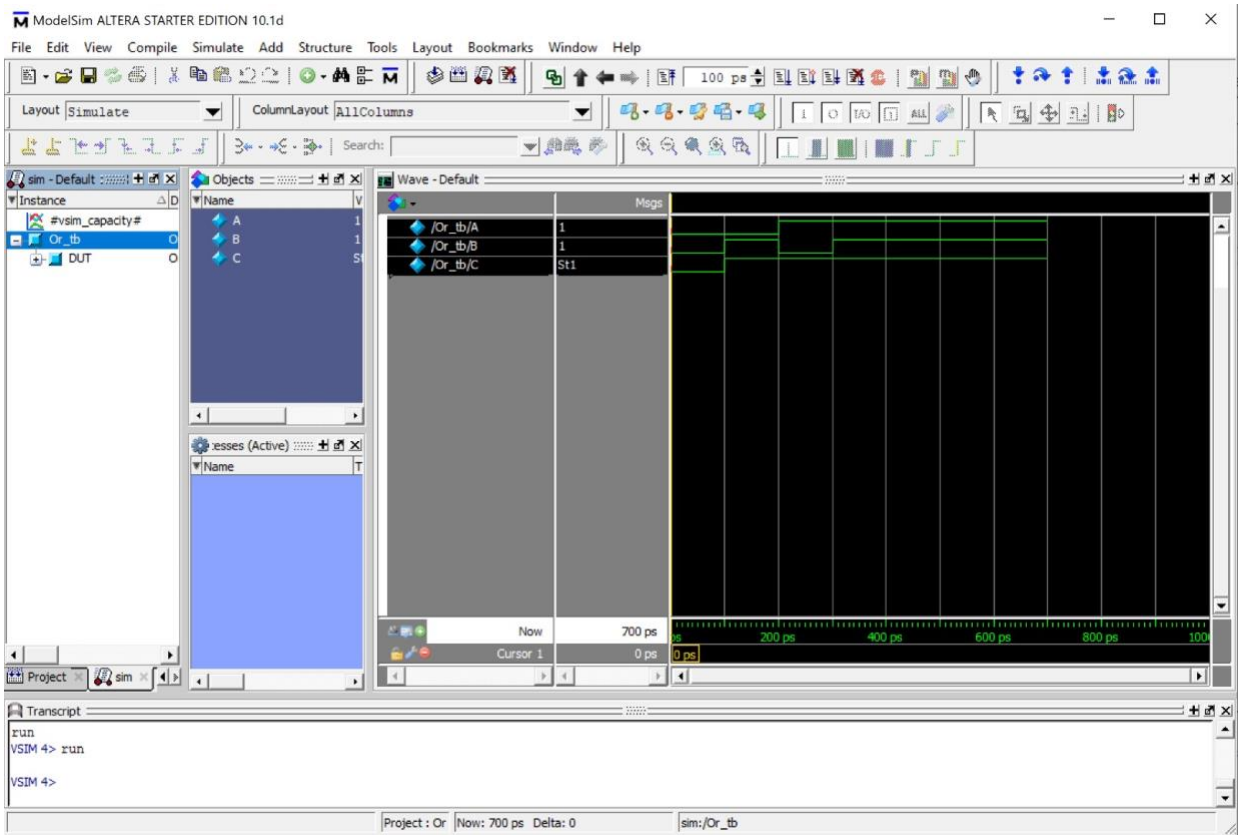

Figure 10: Simulation of OR

Comment:

- When we have 1 in the inputs, the result becomes 1. I can be sure that the simulation results are true by comparing with truth table.

TestBench of NAND

```
1. module Nand_tb();
2. reg A, B;
3. wire C;
4. Nand DUT(A,B,C);
5. initial
6. begin
7. A = 1'b0; B = 1'b0; #100;
8. A = 1'b0; B = 1'b1; #100;
9. A = 1'b1; B = 1'b0; #100;
10.      A = 1'b1; B = 1'b1; #100;
11.      end
12.      endmodule
```

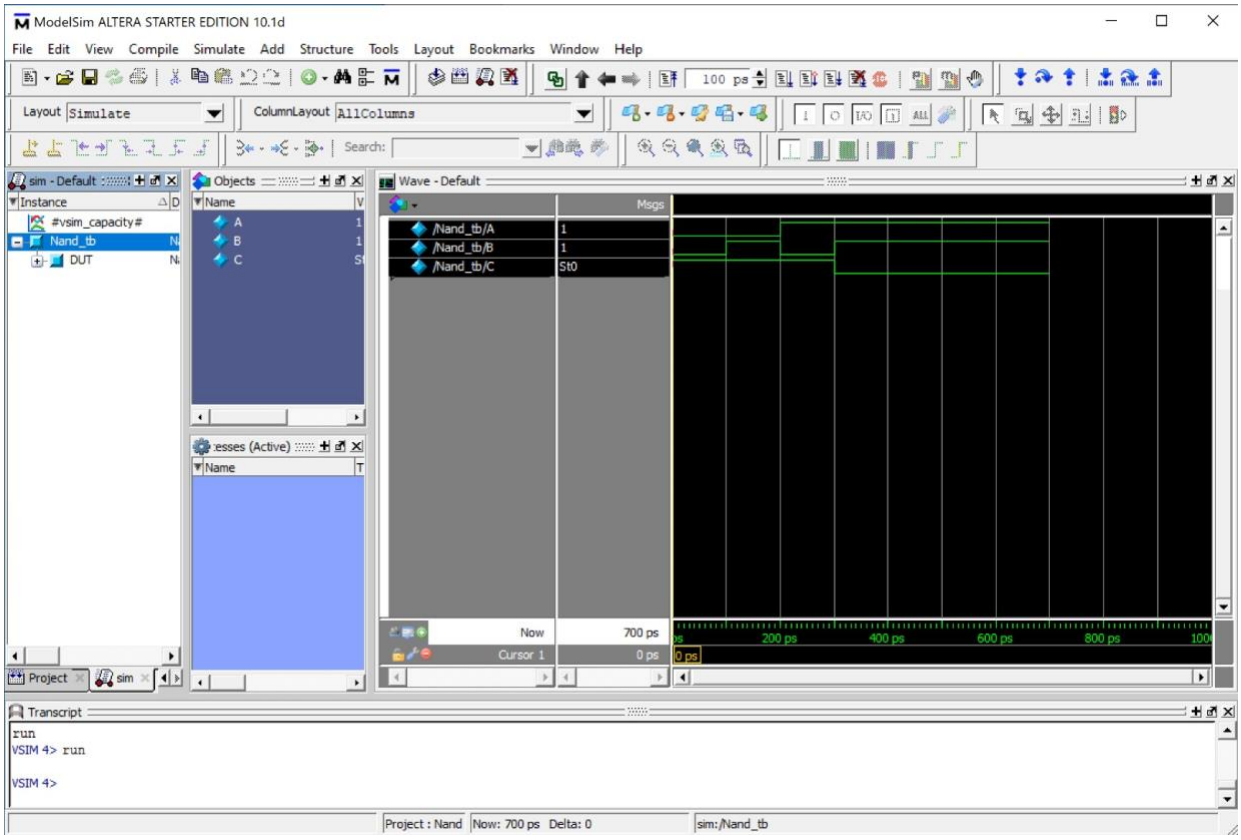

Figure 11: Simulation of NAND

Comments:

- NAND is the opposite type of AND gate. We only get the result as 1 when the inputs are (1,1). I can be sure that the simulation results are true by comparing with truth table.

TestBench of NOR

```
1. module Nor_tb();
2. reg A, B;
3. wire C;
4. Nor DUT(A,B,C);
5. initial
6. begin
7. A = 1'b0; B = 1'b0; #100;
8. A = 1'b0; B = 1'b1; #100;
9. A = 1'b1; B = 1'b0; #100;
10.      A = 1'b1; B = 1'b1; #100;
11.      end
12.      endmodule
```

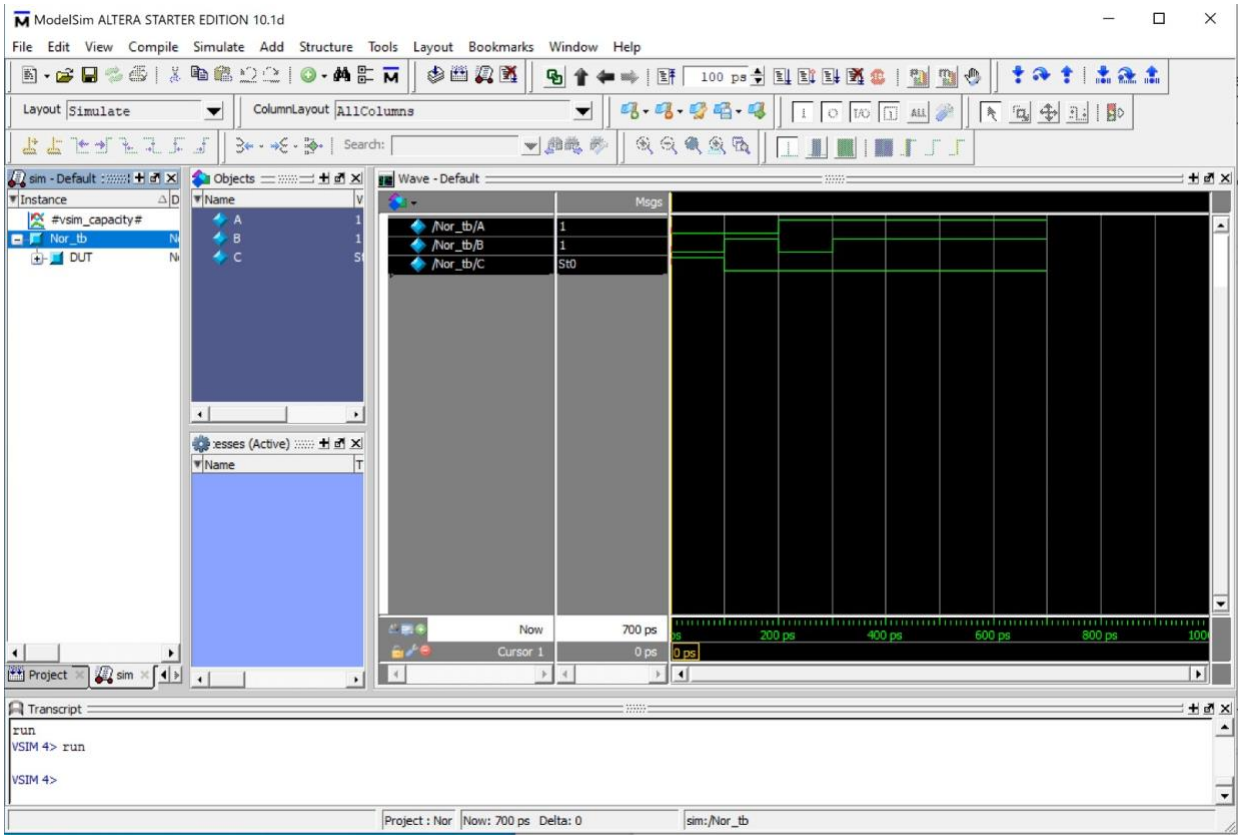

Figure 12: Simulation of NOR

Comments:

- NOR is the opposite type of OR gate. We only get the result as 1 when the inputs are (0,0). I can be sure that the simulation results are true by comparing with truth table.

TestBench of XOR

```
1. module Xor_tb();
2. reg A, B;
3. wire C;
4. Xor DUT(A,B,C);
5. initial
6. begin
7. A = 1'b0; B = 1'b0; #100;
8. A = 1'b0; B = 1'b1; #100;
9. A = 1'b1; B = 1'b0; #100;
10.      A = 1'b1; B = 1'b1; #100;
11.      end
12.      endmodule
```

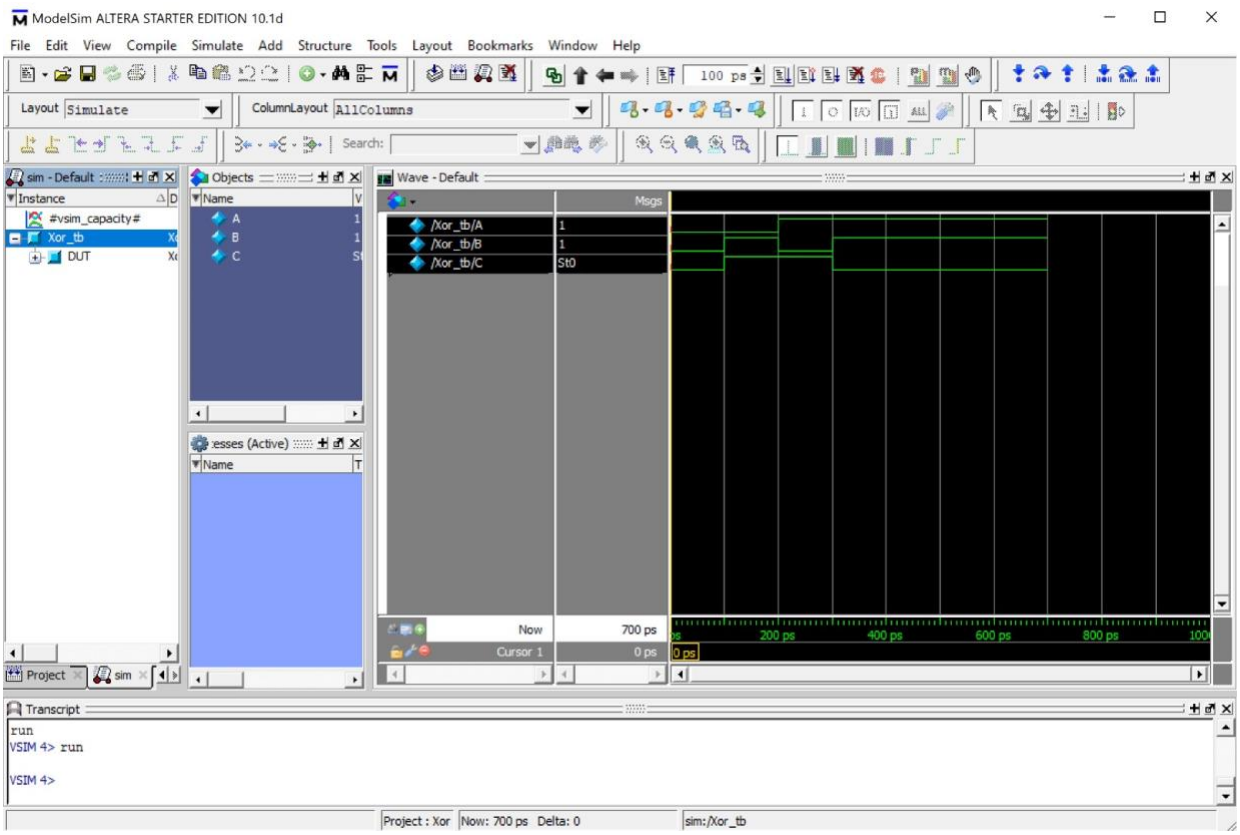

Figure 13: Simulation of XOR

Comments:

- By XOR gate, we can detect opposite inputs. That means, when one of the inputs 0 and the other one is 1, we get 1 as a result. I can be sure that the simulation results are true by comparing with truth table.

TestBench of XNOR

```
1. module Xnor_tb();
2. reg A, B;
3. wire C;
4. Xnor DUT(A,B,C);
5. initial
6. begin
7. A = 1'b0; B = 1'b0; #100;
8. A = 1'b0; B = 1'b1; #100;
9. A = 1'b1; B = 1'b0; #100;
10.      A = 1'b1; B = 1'b1; #100;
11.      end
12.      endmodule
```

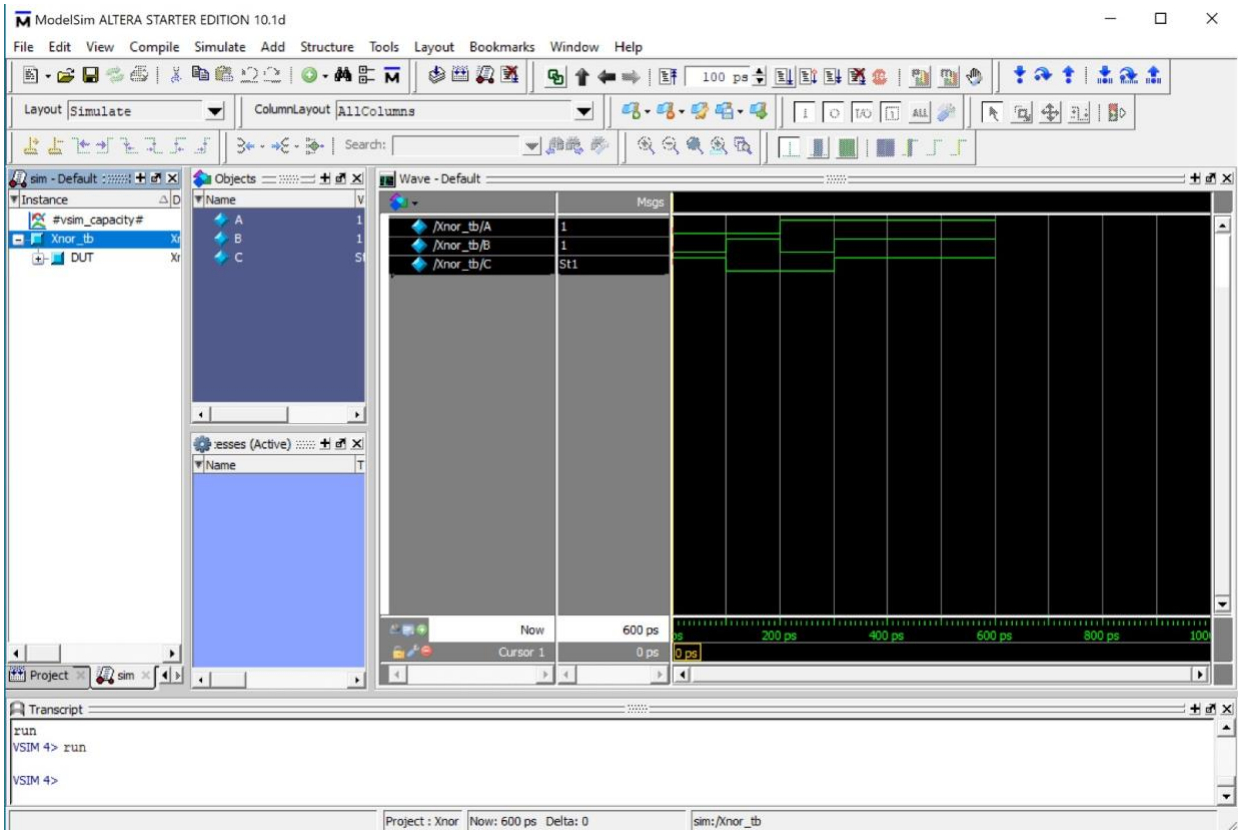

Figure 14: Simulation of XNOR

Comments:

- XNOR is a gate that the opposite of XOR gate. It can take only two inputs. When the inputs are the same the result becomes 1. I can be sure that the simulation results are true by comparing with truth table.

Question: 2.2.2.3

i. Write a structural Verilog code and implement all logic gates together as depicted in figure, simulate and verify the functionality using Modelsim. Compare your results with 1.i and 1.ii.

Verilog Code of Circuit 1

```
1. module Figure1(A,B,C,D,E,F,G,H,I,J);
2. input A, B;
3. output C,D,E,F,G,H,I,J;
4. and G1(C,A,B);
5. or G2(D, A, B);
6. nand G3(E,A,B);
7. nor G4(F,A,B);
8. xor G5(G,A,B);
9. xnor G6(H,A,B);
10.     not G7(I,A);
11.     not G8(J, B);
12.     endmodule
```

TestBench of Circuit 1

```
1. module Figure1_tb();
2. reg A, B;
3. wire C,D,E,F,G,H,I,J;
4. Figure1 DUT(A,B,C,D,E,F,G,H,I,J);
5. initial
6. begin
7. A = 1'b0; B = 1'b0; #100;
8. A = 1'b0; B = 1'b1; #100;
9. A = 1'b1; B = 1'b0; #100;
10.     A = 1'b1; B = 1'b1; #100;
11.     end
12.     endmodule
```

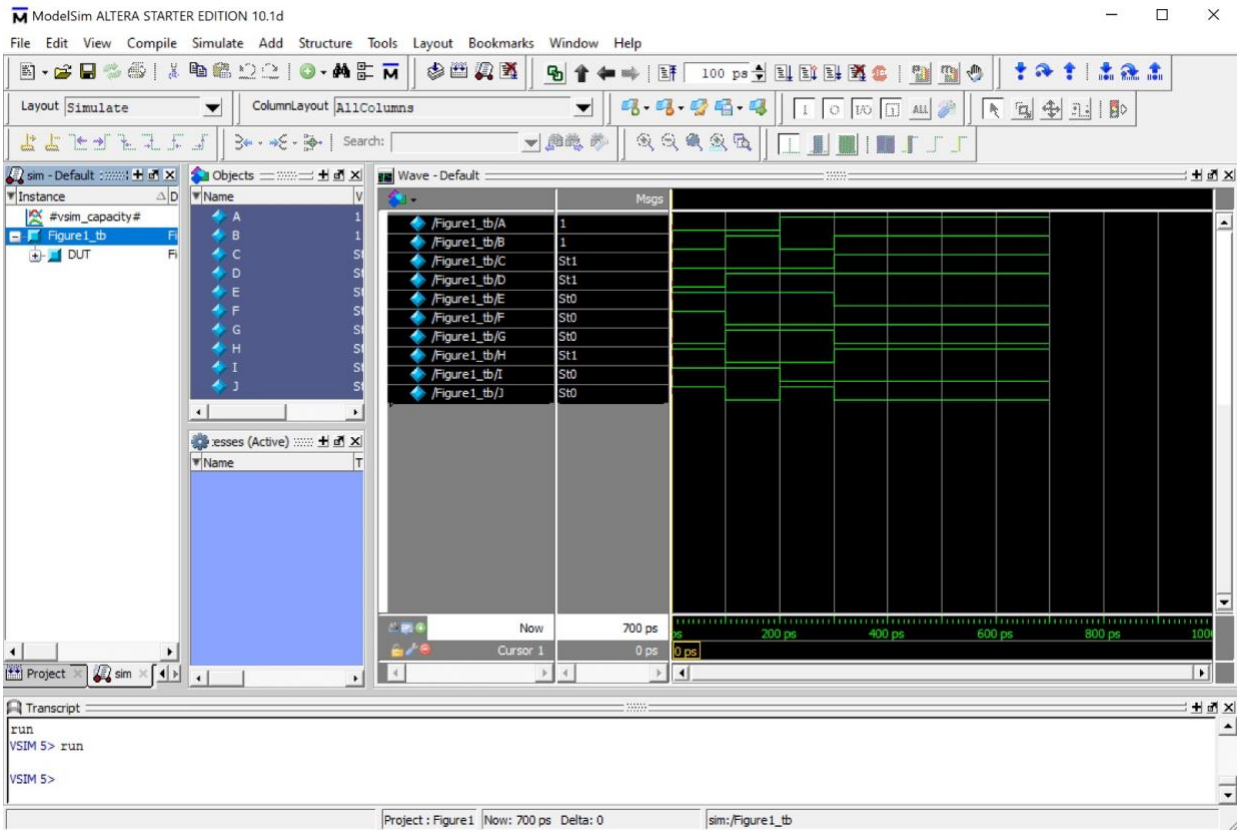

Figure 15: Simulation of Circuit 1

Comments:

- The results of the simulations are the same with 2.2.2.1 simulation results. Circuit 1 is a complete circuit which has all the gates in 2.2.2.1. Therefore, the waves behaved as expected.

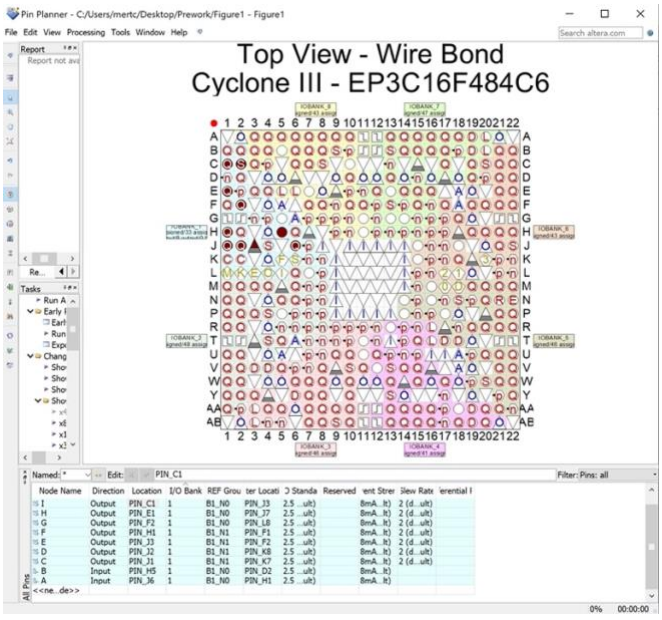

Figure 16: Pin Planner of Circuit 1

2.2.3 Complex Gates

QUESTION 2.2.3.1

Briefly explain in words the function of each logic block represented by the truth tables (a) and (b) i.e., describe the condition for which the output becomes '1' for each function.

For Minority Gate, the condition for which the output becomes '1' is that sum of the inputs must be less than half of the number of inputs. For example, sum of the inputs at 2nd row in truth table of minority is 1, and half of the number of inputs is 1.5. Therefore, 1 is less than 1.5, and the result is 1. However, the sum in the last row of the table is 3, and 3 is bigger than 1.5. As a result, the output becomes 0.

For Even Detector, the condition for which the output becomes '1' is that the sum of the inputs must be even. For instance, sum of the inputs in the 1st row of the table is 0, and 0 is even; therefore, the output becomes 1. On the other hand, sum of the inputs in the 2nd row is 1, and 1 is odd. Therefore, the output is 0.

QUESTION 2.2.3.2

Using AND, OR, and inverter operators, derive a Boolean expression for each of the two outputs as a function of the inputs:

i. MN-out (a, b, c) = ?

- ⇨ a'b'c' + a'b'c + a'bc' + ab'c'
- ⇨ a'(b'c' + b'c + bc') + a'b'c'
- ⇨ a'[b'(c' + c)] + ab'c'
- ⇨ a'(b' + c') + ab'c'
- ⇨ a'b' + a'c' + ab'c'
- ⇨ b'(a' + ac') + a'c'
- ⇨ b'[(a' + a)(a'+ c')] + a'c'
- ⇨ b'(a' + c')
- ⇨ a'b' + b'c' + a'c'

ii. EVEN-out (a,b,c) = ?

- ⇨ a'b'c' + a'bc + ab'c + abc'

QUESTION 2.2.3.3

Sketch a logic schematic implementation for each of your answers in 1.2.3,2

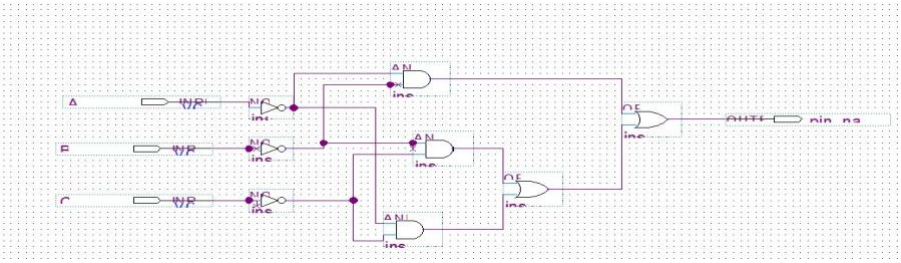

Figure 17: Schematic of MN-out

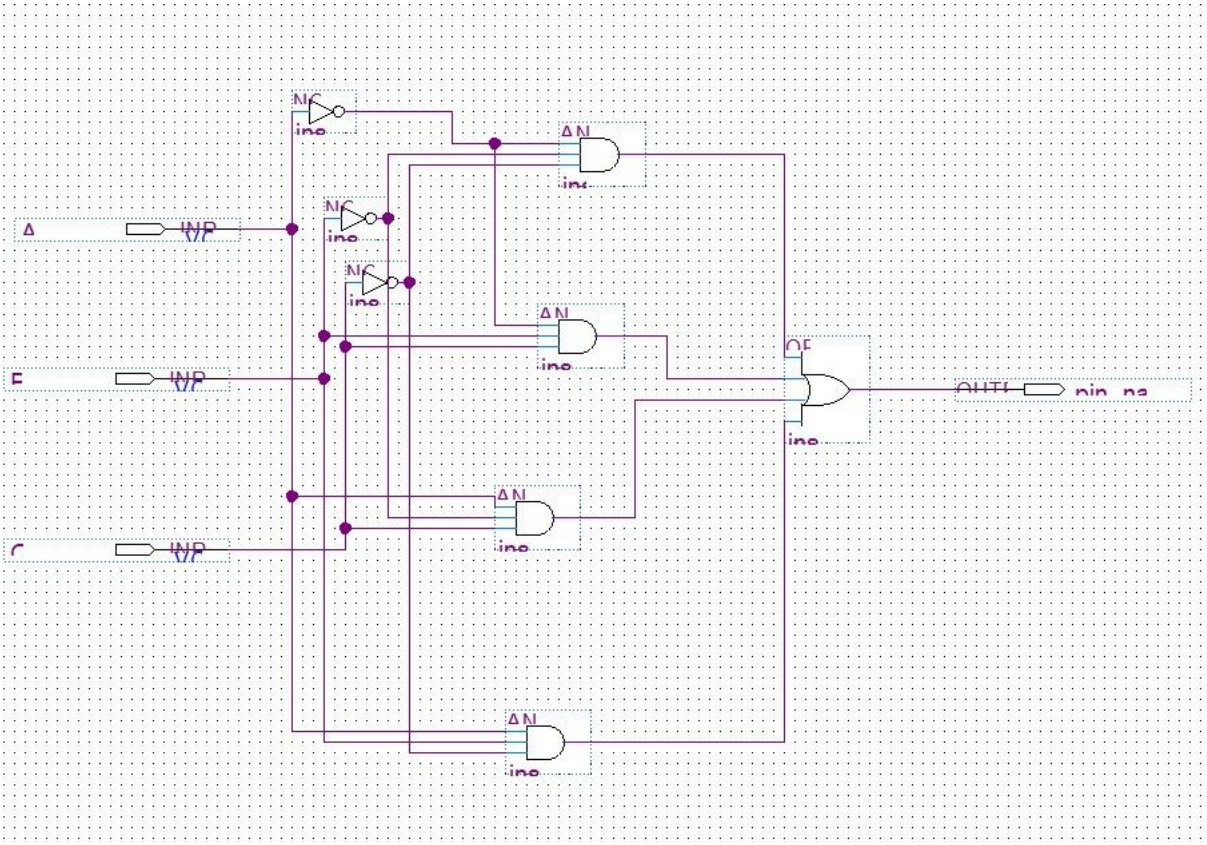

Figure 18: Schematic of EVEN-out

QUESTION 2.2.3.4

Given the logic XNOR function in 1.2.2, redesign the 'EVEN-out' output using XNOR gates only i.e., first express EVEN-out as an expression with one or more XNOR operations, and then sketch a schematic.

EVEN-out function can be written as (A $\odot$ B) $\odot$ C'

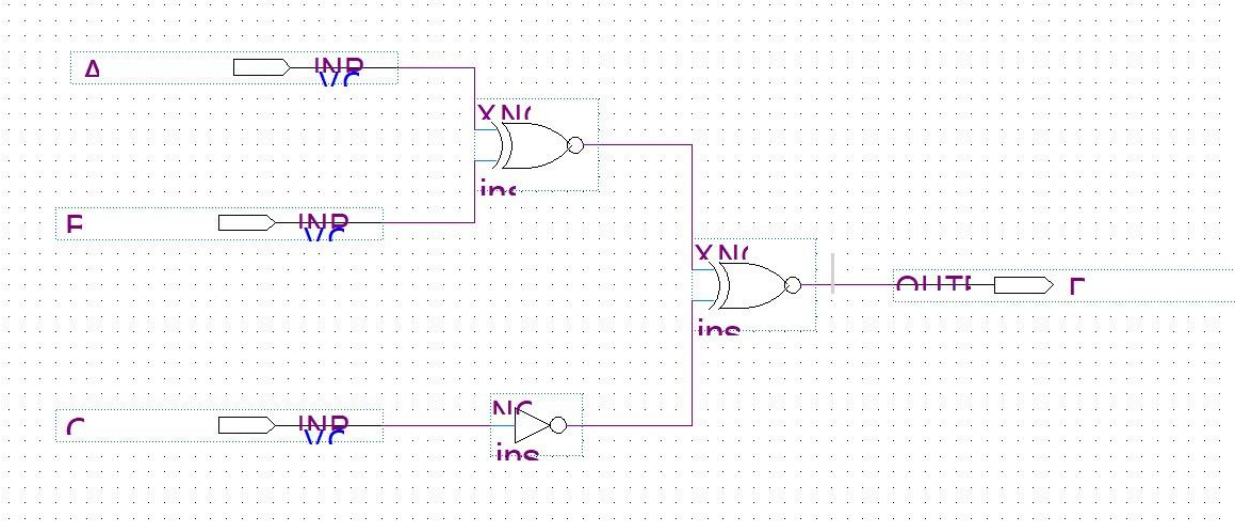

Figure 19: Schematic of EVEN-out using XNOR

```
1. module complex1(A,B,C,D);
2. output D;
3. input A,B,C;
4. wire w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11;
5. not N1(w1,A);
6. not N2(w2,B);
7. not N3(w3,C);
8. and A1(w4,w1,w2);
9. and A2(w5,w2,w3);
10.      and A3(w6,w1,w3);
11.      or O1(w7,w5,w6);
12.      or O2(D,w7,w4);
13.      endmodule
```

Figure 20: Verilog Code of MN-out

```
1. module complex2(A,B,C,D);
2. output D;
3. input A,B,C;
4. wire w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13;
5. not N1(w1,A);
6. not N2(w2,B);
7. not N3(w3,C);
8. and A1(w4,w1,w2);
9. and A2(w5,w4,w3);
10.      and A3(w6,w1,B);
11.      and A4(w7,w6,C);
12.      and A5(w8,A,w2);
13.      and A6(w9,w8,C);
14.      and A7(w10,A,B);
15.      and A8(w11,w10,w3);
16.      or O1(w12,w5,w7);
17.      or O2(w13,w9,w11);
18.      or O3(D,w12,w13);
19.      endmodule
```

Figure 21: Verilog Code of EVEN-out

```
1. module complex1_tb();
2. reg A,B,C;
3. wire D;
4. complex1 DUT(A,B,C,D);
5. initial
6. begin
7. A = 1'b0; B = 1'b0; C = 1'b0; #100;
8. A = 1'b0; B = 1'b0; C = 1'b1; #100;
9. A = 1'b0; B = 1'b1; C = 1'b0; #100;
10.   A = 1'b0; B = 1'b1; C = 1'b1; #100;
11.   A = 1'b1; B = 1'b0; C = 1'b0; #100;
12.   A = 1'b1; B = 1'b0; C = 1'b1; #100;
13.   A = 1'b1; B = 1'b1; C = 1'b0; #100;
14.   A = 1'b1; B = 1'b1; C = 1'b1; #100;
15.   end
16.   endmodule
```

Figure 22: TestBench of MN-out

```
1. module complex2_tb();
2. reg A,B,C;
3. wire D;
4. complex2 DUT(A,B,C,D);
5. initial
6. begin
7. A = 1'b0; B = 1'b0; C = 1'b0; #100;
8. A = 1'b0; B = 1'b0; C = 1'b1; #100;
9. A = 1'b0; B = 1'b1; C = 1'b0; #100;
10.      A = 1'b0; B = 1'b1; C = 1'b1; #100;
11.      A = 1'b1; B = 1'b0; C = 1'b0; #100;
12.      A = 1'b1; B = 1'b0; C = 1'b1; #100;
13.      A = 1'b1; B = 1'b1; C = 1'b0; #100;
14.      A = 1'b1; B = 1'b1; C = 1'b1; #100;
15.      end
16.      endmodule
```

Figure 23: TestBench of EVEN-out

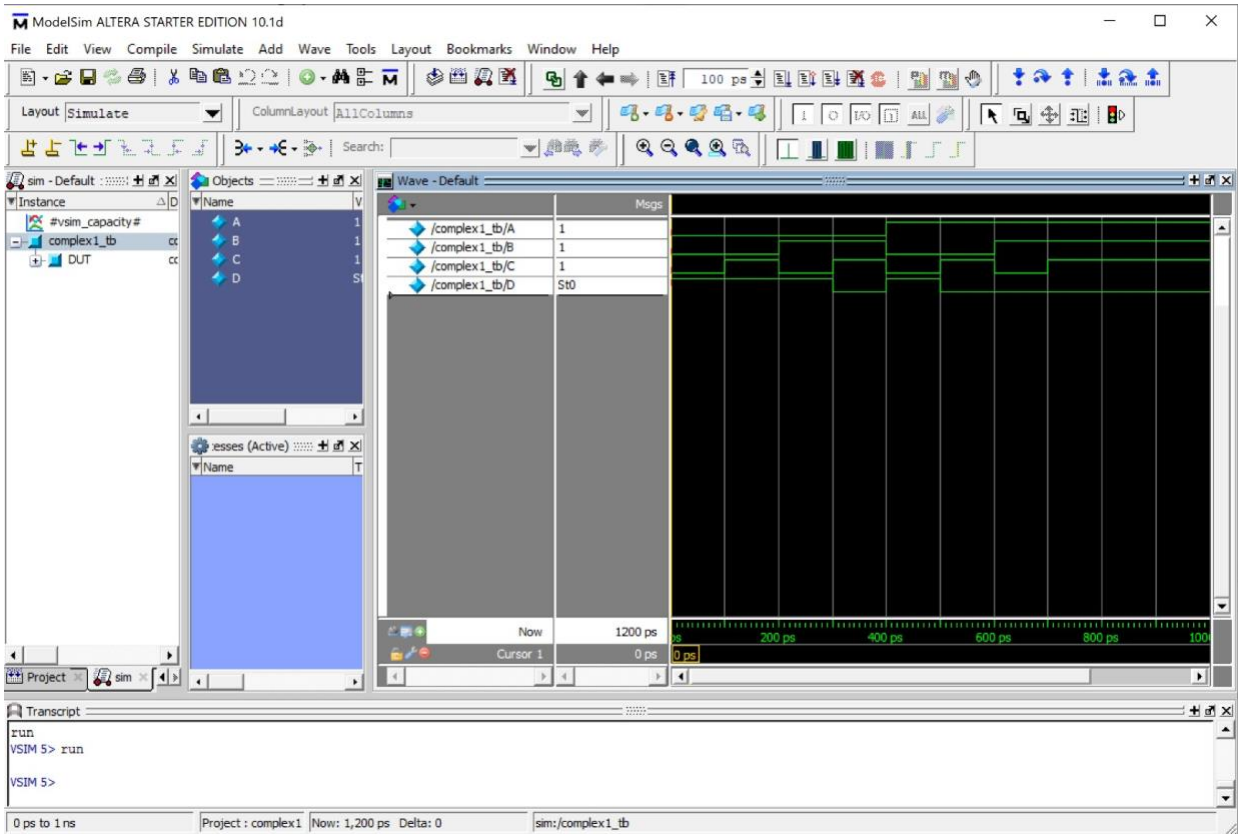

Figure 24: Simulation of MN-out

Comments:

- If inputs that represent 1 are minority, then MN-out becomes 1. It means that when the half of number of the inputs greater than the sum of the inputs, the output becomes 1. The result are as I expected and I can be sure that the result of the simulation is correct by comparing with truth table.

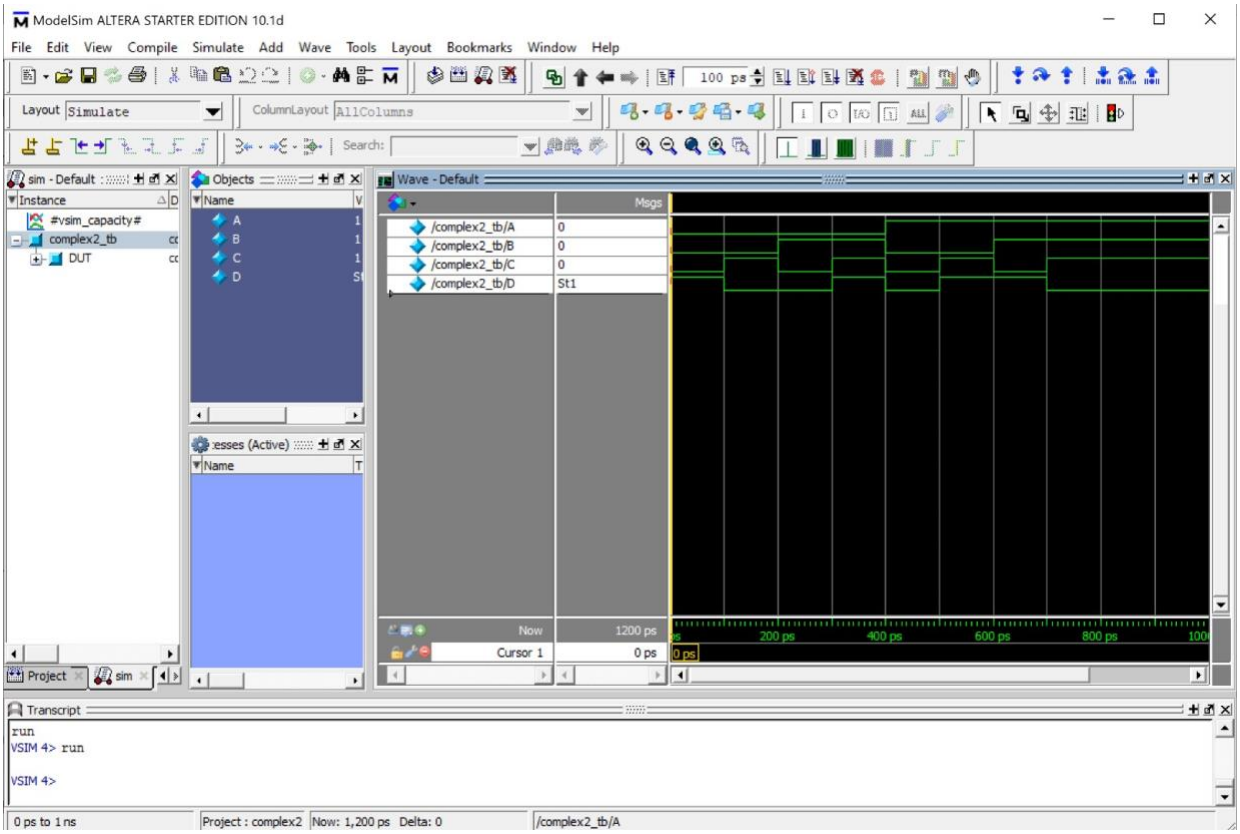

Figure 25: Simulation of EVEN-out

Comments:

- If the sum of the inputs is even, the output becomes 1. Otherwise, it is 0. When I compared the truth table with the result of the simulation, they are the same as it is expected. Hence, I can be sure that the result of the simulation is correct.

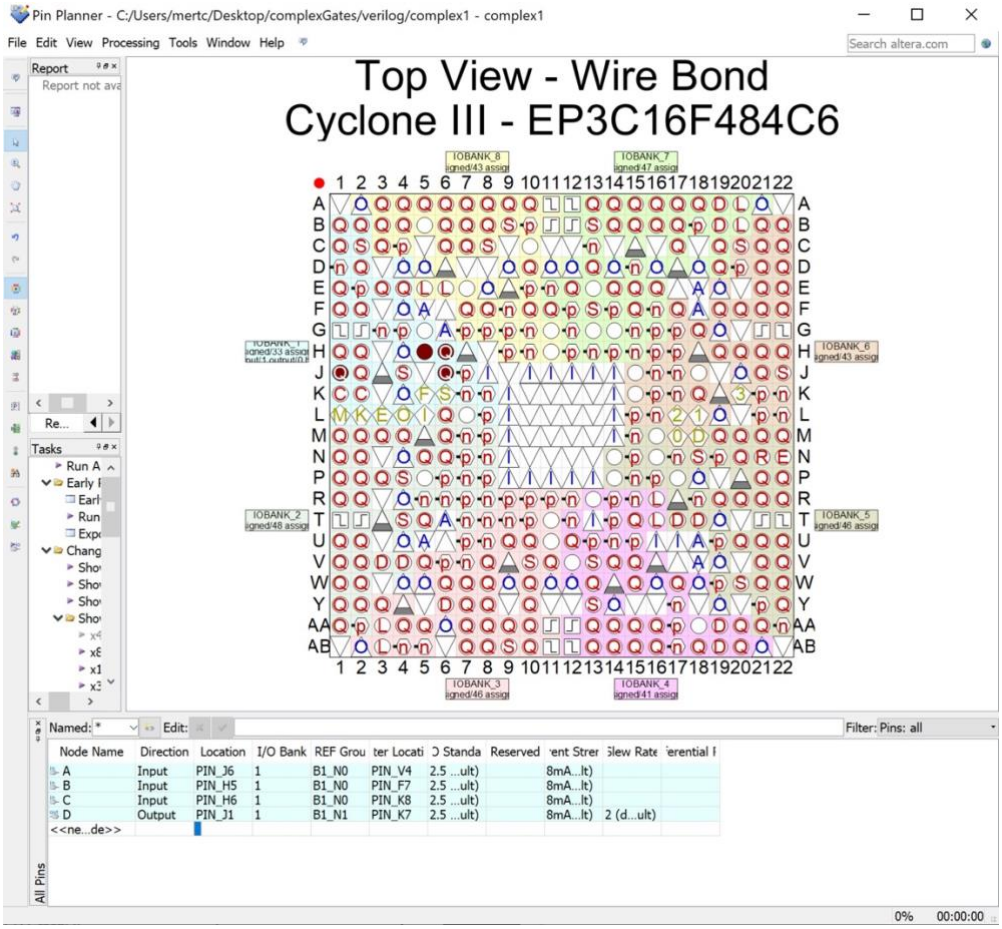

Figure 26: Pin Planner of MN-out

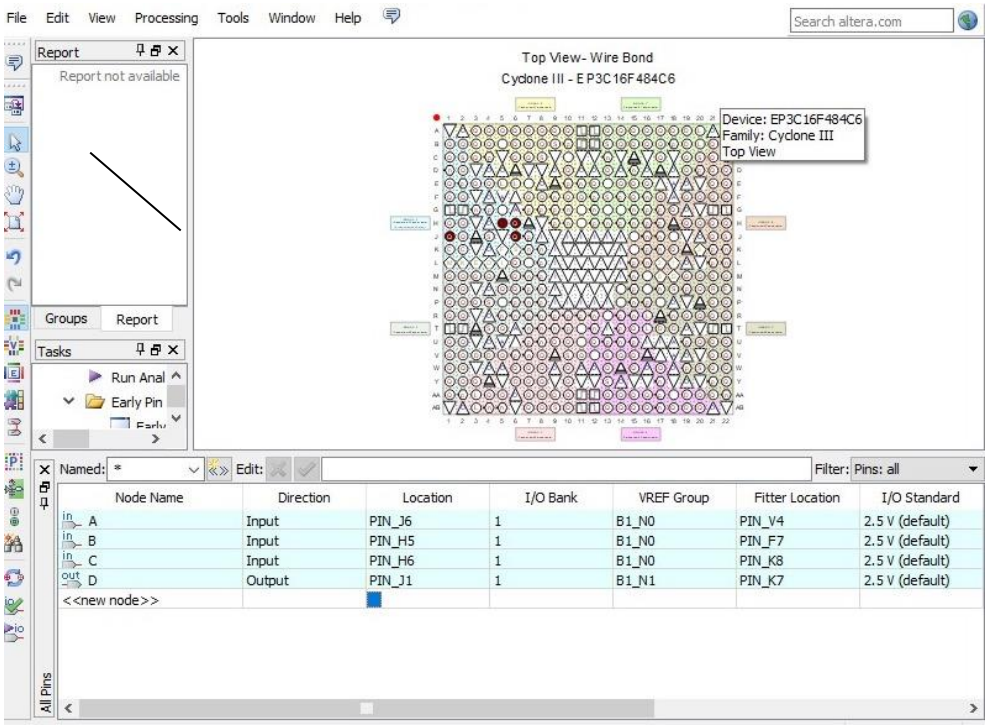

Figure 27: Pin Planner of EVEN-out

QUESTION 2.2.4 WORD PROBLEM: BAKERY PROFIT CALCULATOR

In this design, you will implement a simple digital circuit to compute total profits made by a bakery based on the kinds of deserts that they are baking.

Truth Table

Croissant	Éclair	Donuts	Brownie	P2	P1	P0
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	1	0	0
0	0	1	1	1	1	0
0	1	0	0	0	1	1
0	1	0	1	1	0	1
0	1	1	0	1	1	1
0	1	1	1	X	X	X
1	0	0	0	0	0	1
1	0	0	1	0	1	0
1	0	1	0	0	1	1
1	0	1	1	X	X	X
1	1	0	0	1	0	1
1	1	0	1	X	X	X
1	1	1	0	X	X	X
1	1	1	1	X	X	X

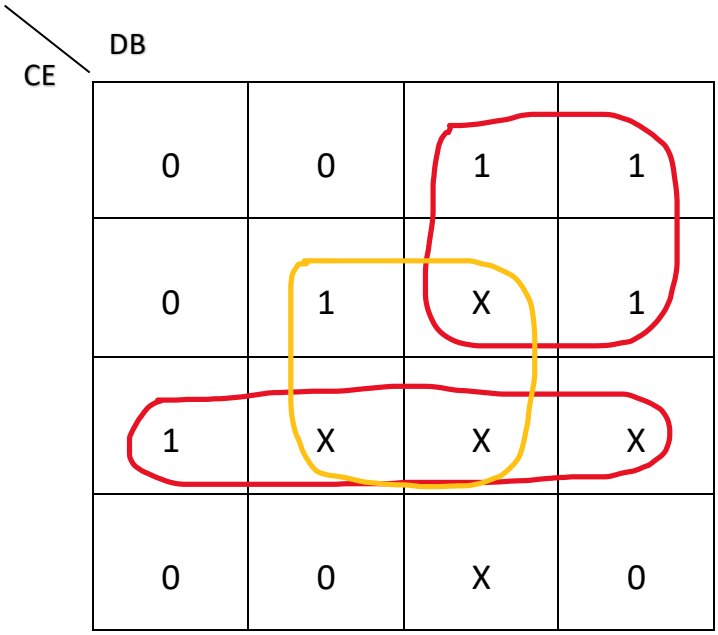

Figure 28: K-Map of P2

Boolean Expression of P2: C’D + EB + CE

CE \ DB				
	0	1	1	0
	1	0	X	1
	0	X	X	X
	0	1	X	1

Figure 29: K-Map of P1

Boolean Expression of P1: C’EB’ + E’B + CD

CE \ DB				
	0	0	0	0
	1	1	X	1
	1	X	X	X
	1	0	X	1

Figure 30: K-Map of P0

Boolean Expression of P0: C’E + CB’

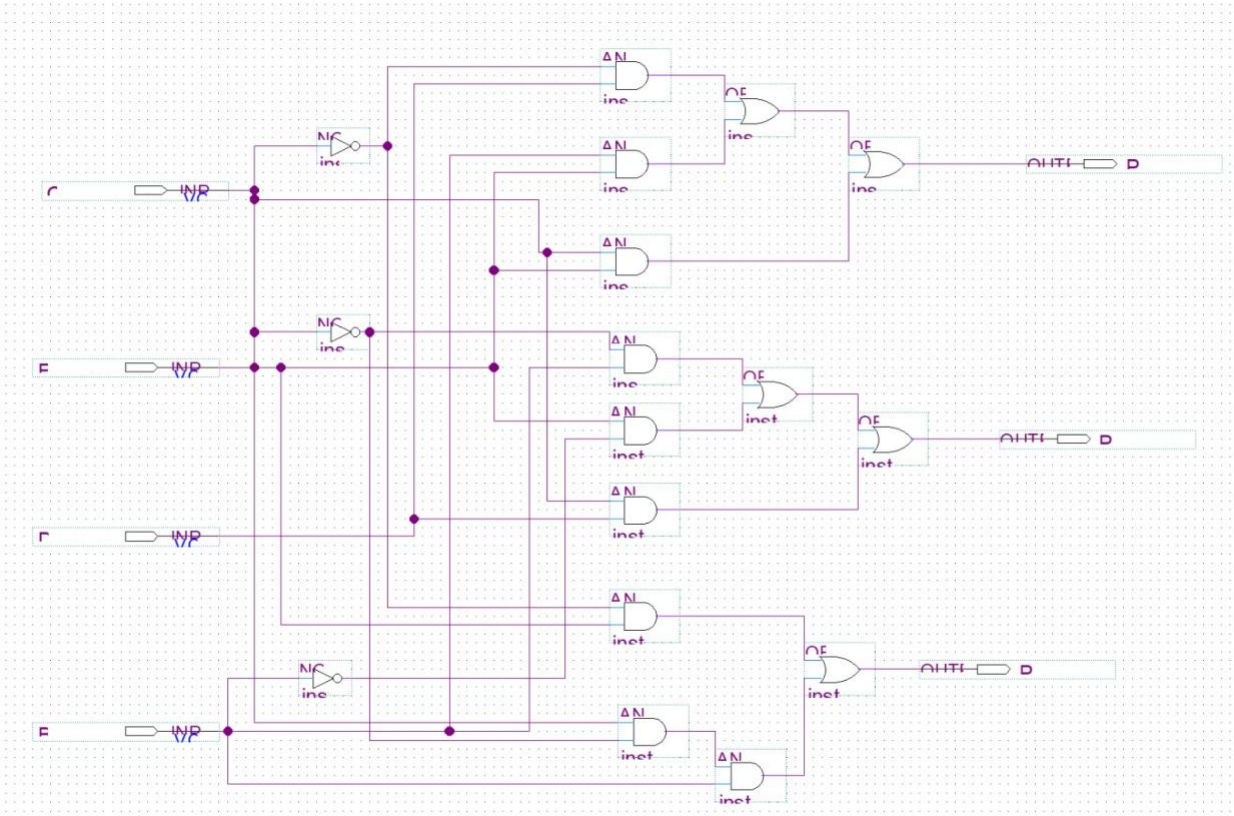

Figure 31: Schematic of Bakery Profit Calculator

```
1. module Bakery (C, E, D, B, P2, P1, PO) ;
2. output P2, P1, PO;
3. input C, E, D, B;
4. wire w1, w2, w3, w4, W5, W7, w8, w9, w10, w11, w12, w14, w15, W16;
5. //p2 part
6. not N1 (w1, C);
7. not N2 (w7, E);
8. not N3 (w8,B);
9. and A1 (w2,w1,D);
10.      and A2(w3,E,B);
11.      and A3(w4,C,E);
12.      or O1(w5, w2, w3);
13.      or O2(P2, w5, w4);
14.      //p1 part
15.      and A4(w9,w7,B);
16.      and A5(w10,E,w8);
17.      and A6(w11,C,D);
18.      or O3(w12,w9,w10);
19.      or O4(P1,w12,w11);
20.      //p0 part
21.      and A7(w14,w1,E);
22.      and A8(w15,C,w7);
23.      and A9(w16,w15,w8);
24.      or O5(P0, w14, w16);
25.      endmodule
```

Figure 32: Verilog Code of Bakery Profit Calculator

```
1. module Bakery_tb();
2. reg C,E,D,B;
3. wire P2,P1,P0;
4. Bakery DUT(C,E,D,B,P2,P1,P0);
5. Initial
6. Begin
7. C = 1'b0; E = 1'b0; D = 1'b0; B = 1'b0; #100;
8. C = 1'b0; E = 1'b0; D = 1'b0; B = 1'b1; #100;
9. C = 1'b0; E = 1'b0; D = 1'b1; B = 1'b0; #100;
10.      C = 1'b0; E = 1'b0; D = 1'b1; B = 1'b1; #100;
11.      C = 1'b0; E = 1'b1; D = 1'b0; B = 1'b0; #100;
12.      C = 1'b0; E = 1'b1; D = 1'b0; B = 1'b1; #100;
13.      C = 1'b0; E = 1'b1; D = 1'b1; B = 1'b0; #100;
14.      C = 1'b0; E = 1'b1; D = 1'b1; B = 1'b1; #100;
15.      C = 1'b1; E = 1'b0; D = 1'b0; B = 1'b0; #100;
16.      C = 1'b1; E = 1'b0; D = 1'b0; B = 1'b1; #100;
17.      C = 1'b1; E = 1'b0; D = 1'b1; B = 1'b0; #100;
18.      C = 1'b1; E = 1'b0; D = 1'b1; B = 1'b1; #100;
19.      C = 1'b1; E = 1'b1; D = 1'b0; B = 1'b0; #100;
20.      C = 1'b1; E = 1'b1; D = 1'b0; B = 1'b1; #100;
21.      C = 1'b1; E = 1'b1; D = 1'b1; B = 1'b0; #100;
22.      C = 1'b1; E = 1'b1; D = 1'b1; B = 1'b1; #100;
23.      end
24.      endmodule
```

Figure 33: Testbench of Bakery Profit Calculator

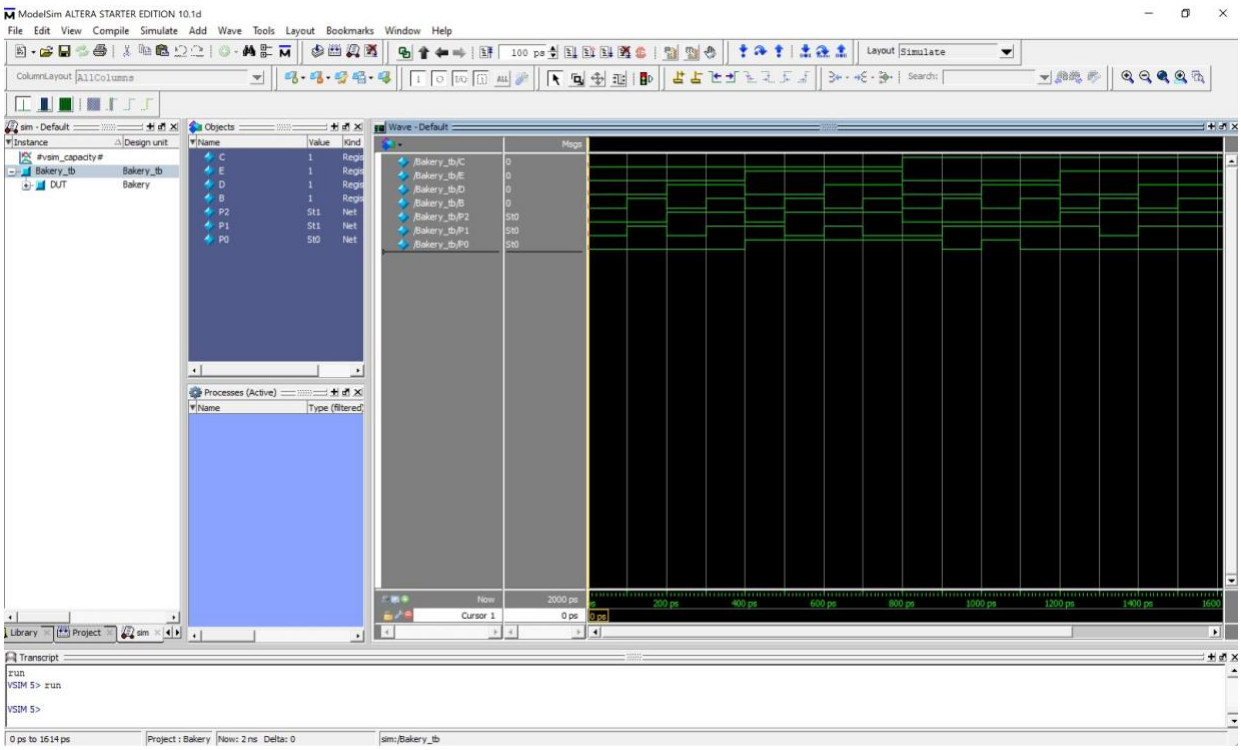

Figure 34: Simulation of Bakery Profit Calculator

Comments:

- When E is 1, the output P0 is 1 too because E has an odd value, and E is the only odd value in the profits. The only exception is when bakery prepare E and C, but when E and C are prepared at the same time profit increases by 1, so the output P0 is ‘1’ again.

- We can see the restrictions of the bakery in the simulation result, more than 2 products cannot be prepared at the same time, so after 1100ps all the outputs always ‘0’.

- The profit of the each product can be seen on the simulation result, P2 is the most significant bit. Therefore, it is clear that Donuts, Brownie, Euclier and Croissant have 4, 2, 3 and 1 profits, respectively.

- Although the profit of the each product can bee seen, the restrictions can be realized. When Brownie and Croissant are prepared together at 900ps, the profit reduces by 1. Also at 1000ps when Croissant and Donuts are prepared together, the profit go down by 1 unit. Moreover, at 1200ps when Croissant and Euclier are prepared together, the profit go up by 1 unit.

- By considering the restrictions of the Bakery, we can manipulate the Boolean expression of the truth table. By doing that, we can simplify the circuit and get an efficient circuit.

- By comparing the truth table and simulation result, I can be sure that the results are correct.

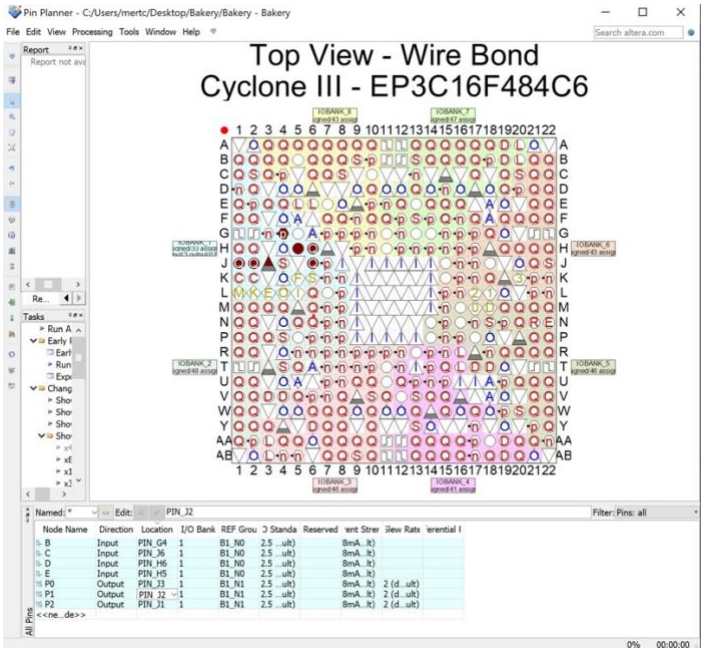

Figure 35: Pin Planner of Bakery Profit Calculator