

Digital Systems ENCS2340

Verilog HDL Project

Dr. Ismail Khater

Section (2)

Yaqen Nouman Hamouda (1211168)

Contents:

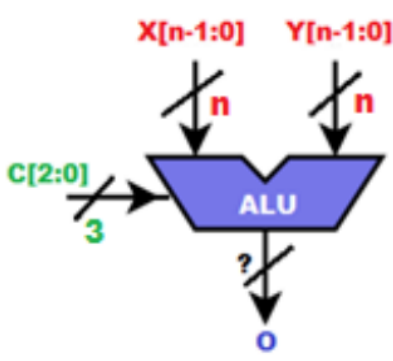
- >> Definition for N-bit ALU
- >> Description for ALU we want to design
- >> A) Specify the size of the output (O)
- >> B) ALU components & implementation
 - >> The components use to design this ALU
 - >> Implementation for each component
 - >> Implementation for whole system
- >> C) Behavioral Verilog modules
- >> D) Structural Verilog module for ALU
- >> E) Waveform of the Structural ALU
- >> F) Behavioral Verilog module for ALU
- >> G) Waveform of the Behavioral ALU



Definition for N-bit ALU:

(Arithmetic Logic Unit) ALU is a digital function that implements the microoperation on the information stored in registers, ALU is a fundamental building block of the many varieties of computing circuits, including the central processing unit (CPU) of computers and graphics processing units (GPUs). one CPU or GPU may contain multiple ALUs.

Description for ALU we want to design:

| ALU Function Code (C) | ALU Output (O) | ALU Symbol |
|-----------------------|-----------------|--|
| 000 | $(X+Y)/2$ |  |
| 001 | $2*(X+Y)$ | |
| 010 | $(X/2)+Y$ | |
| 011 | $X-(Y/2)$ | |
| 100 | X NAND Y | |
| 101 | NOT(X) | |
| 110 | X NOR Y | |
| 111 | X XOR Y | |

In this project, we need to design ALU that execute the above operations. Such that:

1. X and Y are the inputs of the unit and they are N-bit signed numbers represented by 2's complement.
2. C is a 3-bit unsigned number and used to select the operation of the unit (i.e. arithmetic or logical operation).
3. O is the signed ALU output and represented in 2's complement and The size of the O is shown below.

A) Specify the size of the output (O):

The size of the output (O) in bits should be two bit larger than the size of the inputs (X and Y) in order for overflow to never occur.

Since X and Y are N-bit signed numbers represented in 2's complement, the size of O should be N+2 bits.

The first extra bit because the 2's complement representation allows for the detection of overflow by checking the sign bit of the result and having an extra bit in the output allows for the representation of the extra bit in the result that would occur in the case of overflow.

The second extra bit because the second operation in this ALU ($2*(X+Y)$), (X+Y) are represented in N+1 bits and we need extra bit to Multiplication by two then the size of O should be N+2 bits.

Example:

Suppose N=3 ;

Then the maximum value for X and Y equal 3 (011) ;

Execute the second operation (001) ($2*(X+Y)$) ;

The value for (X+Y) equal 6 and need 4 bits to represented it in 2's complement (0110) ;

The value for ($2*(X+Y)$) equal 12 and need 5 bits to represented it in 2's complement (01100) ;

Then we need 5 bits to represented the output without occur overflow.

B) ALU components & implementation:

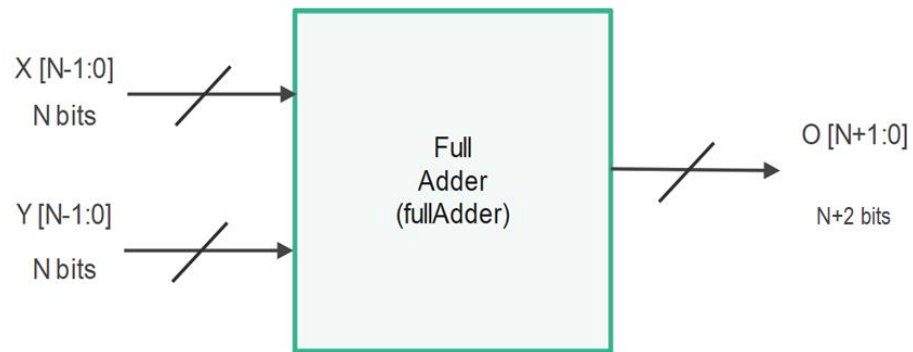
>> ALU implementation using all the following:

1. A adder is used for addition and subtraction operations.
2. A right shift register for N+2 is used for division by 2.
3. A right shift register N is used for division by 2
4. A left shift register is used for multiplication by 2.
5. A NOT module is used for NOT operation.

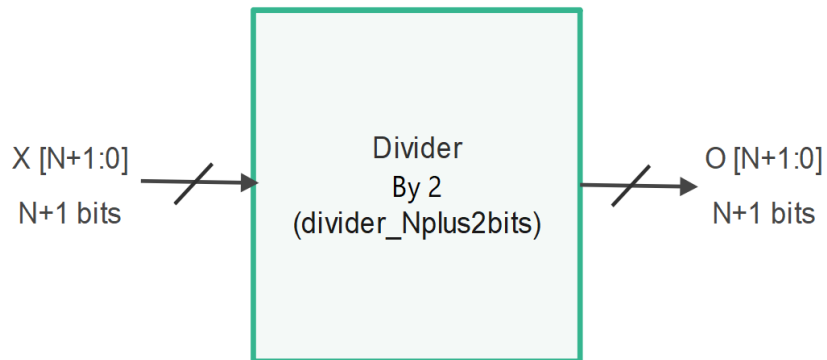
6. A NAND module is used for NAND operation.
7. A NOR module is used for NOR operation.
8. A XOR module is used for XOR operation.
9. A multiplexer(8to1) is used to select the output of the operation.

>> I will use this component to design the ALU, and the implementation for each component will be as shown below:

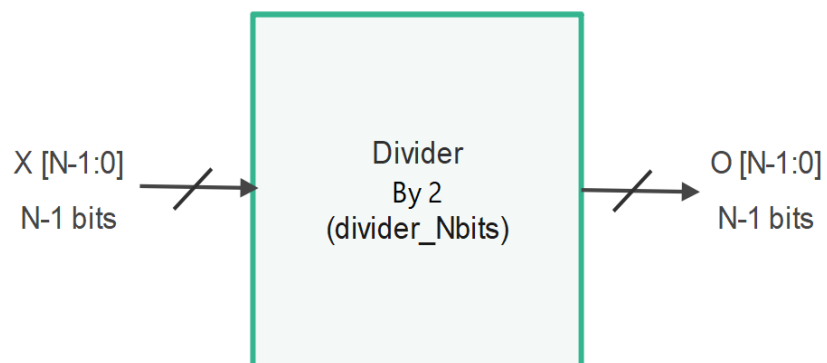
1. implementation for Adder:



2. implementation for divider N+2 bits

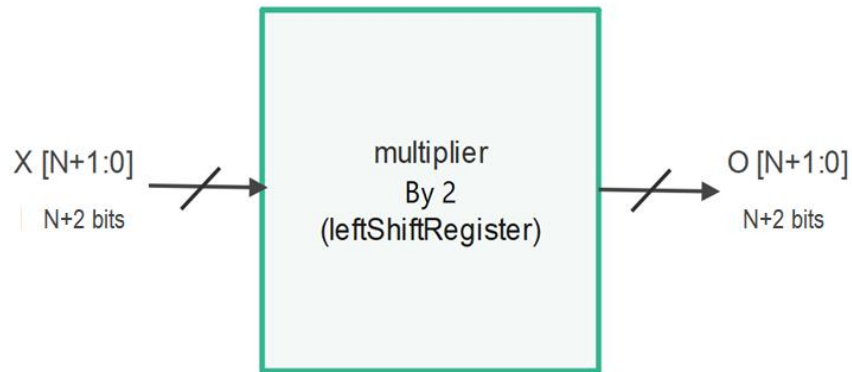


3. implementation for divider N bits

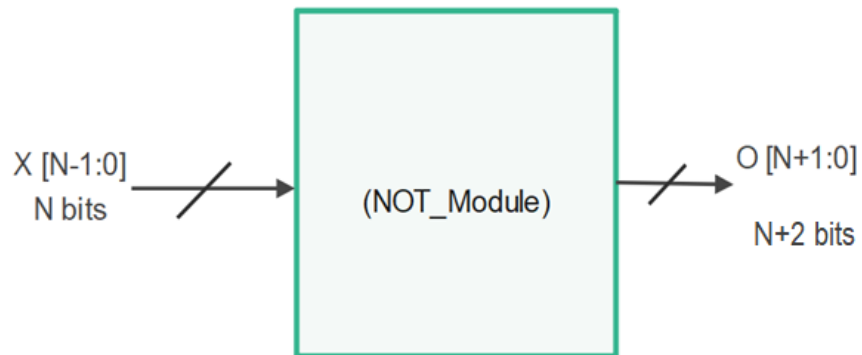


Two modules were created to divide by two the first module has input with size N and the second module has input with size N+2, The reason for the creation of these two because it will become easier to put in the extra bits (sign- or zero-extension).

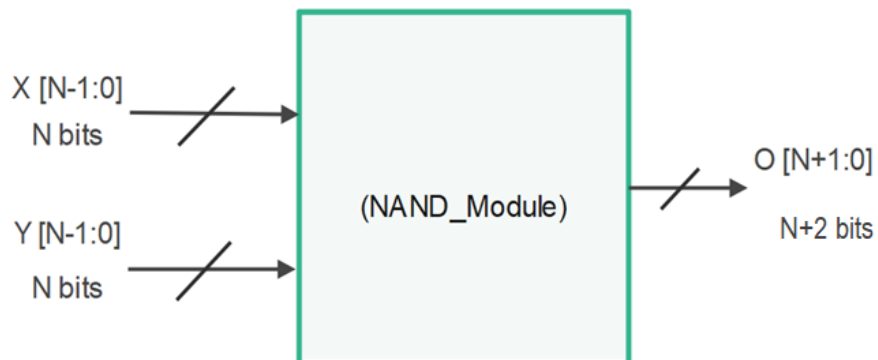
4. implementation for multiplier N+2 bits



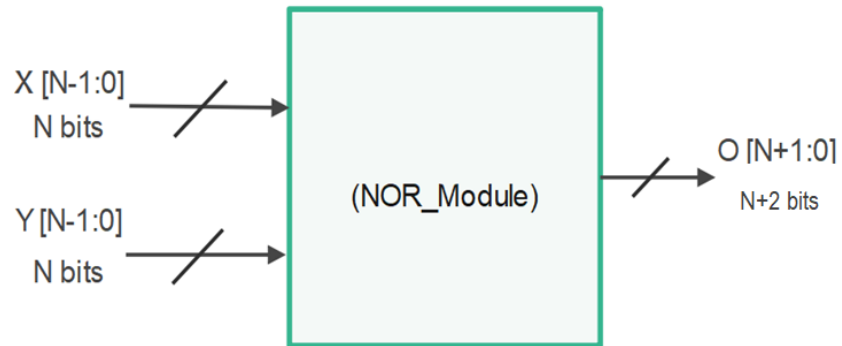
5. implementation for NOT module:



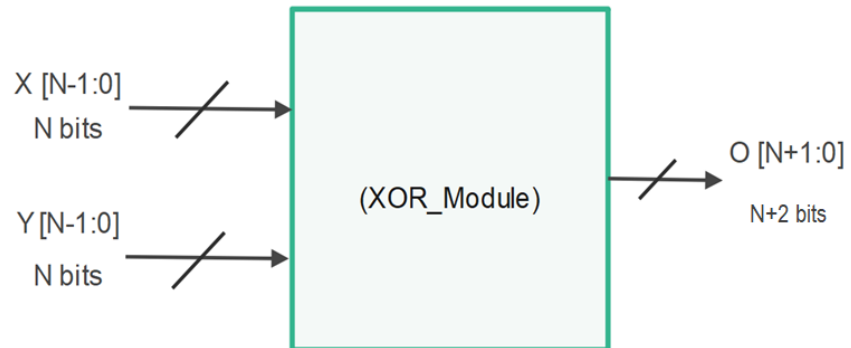
6. implementation for NAND module:



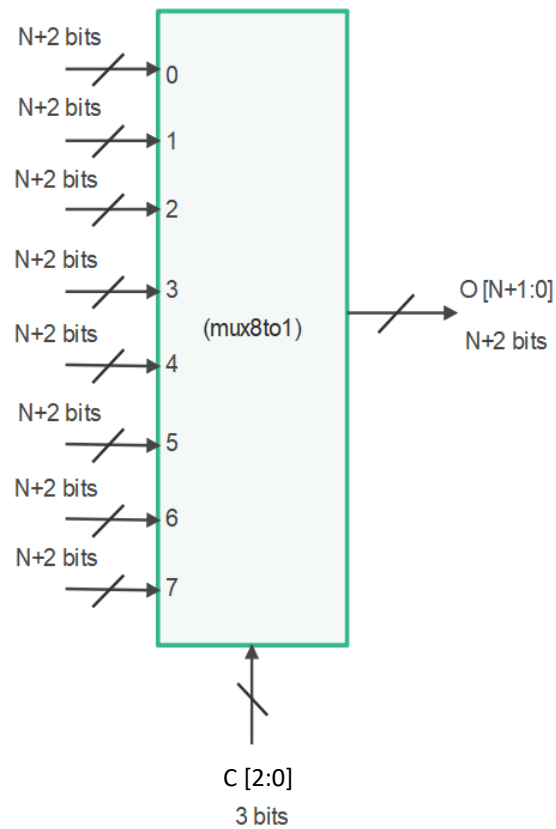
7. implementation for NOR module:



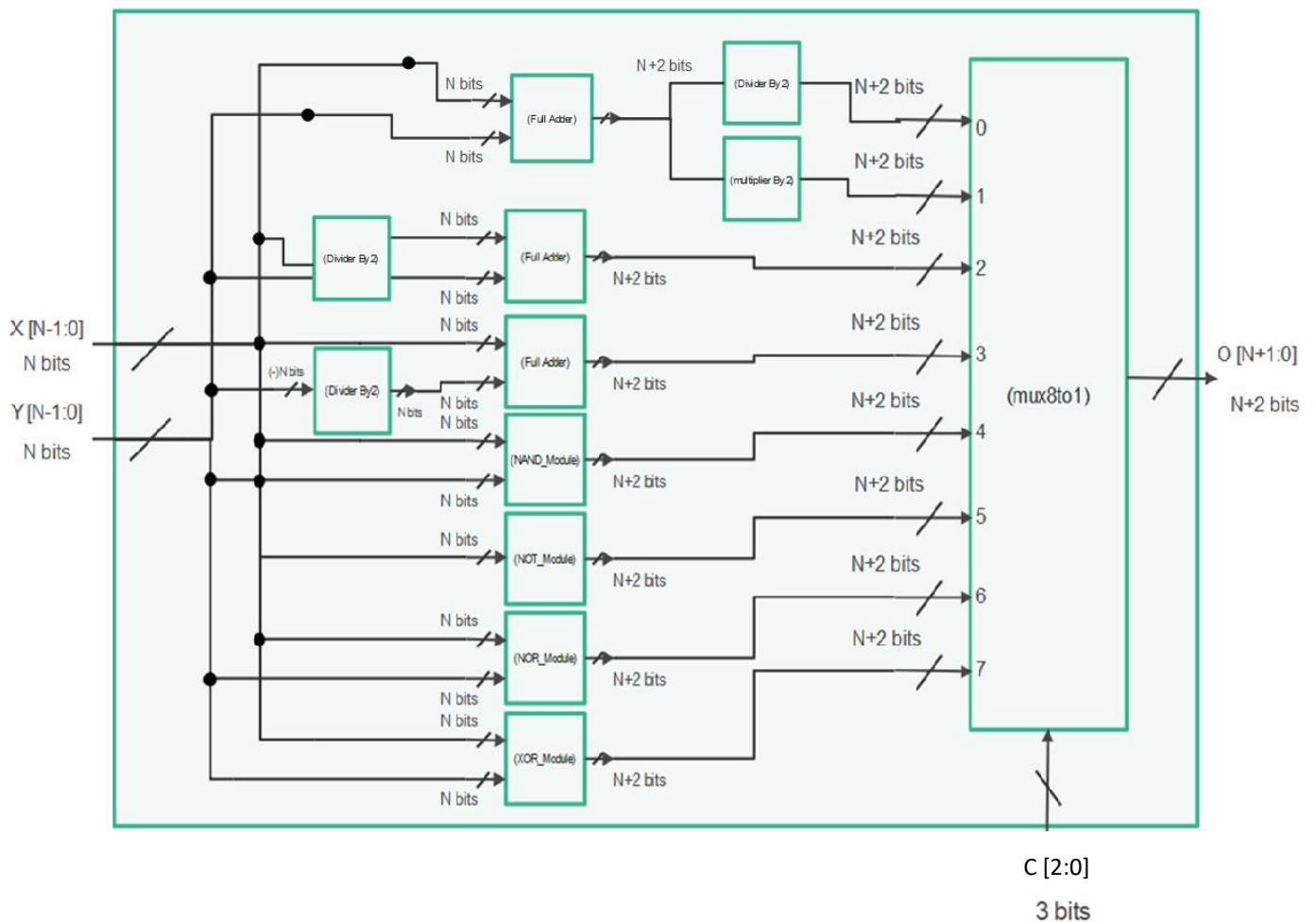
8. implementation for XOR module:



9. implementation for NOR module:



>> Implementation for whole system:



C) Behavioral Verilog modules

Behavioral Verilog code for each component:

1. fullAdder_1211168

```
1 module fullAdder_1211168 #(parameter N = 4) (X, Y, Out);
2     // Define input and output
3     input [N-1:0] X, Y;
4     output reg [N+1:0] Out;
5     reg [N-1:0] s, c;
6
7     always @(*) begin
8         // calculate the result
9         s = X + Y;
10        // calculate the N+1 bit,
11        //by check the value of carry and the value of overFlow
12        c = (X[N-1] & Y[N-1]) | (X[N-1] & s[N-1]) | (Y[N-1] & s[N-1]);
13        // the final result with number bits N+1
14        Out = {c, s};
15        // make sign-extension for the last bit
16        Out = {Out[N], Out[N:0]};
17    end
18 endmodule
```


2. divider_Nplus2bits_1211168

```
1 module divider_Nplus2bits_1211168 #(parameter N = 4)
2   (input signed [N+1:0] X, output reg signed[N+1:0] O);
3   // Define input and output as signed
4   //because the output will represented in 2's complement
5   = always @* begin
6     // calculate the result
7     O = X / 2 ;
8   end
9 endmodule
```

3. divider_Nplus2bits_1211168

```
1 module divider_Nplus2bits_1211168 #(parameter N = 4)
2   (input signed [N+1:0] X, output reg signed[N+1:0] O);
3   // Define input and output as signed
4   //because the output will represented in 2's complement
5   = always @* begin
6     // calculate the result
7     O = X / 2 ;
8   end
9 endmodule
```

4. multiplier_1211168

```
1 module multiplier_1211168 #(parameter N = 4)
2   (input signed [N+1:0] X, output reg signed [N+1:0] O);
3   // Define input and output as signed
4   // because the output will represented in 2's complement
5   = always @* begin
6     // calculate the result
7     O <= X * 2 ;
8   end
9 endmodule
```

5. NAND_Module_1211168

```
1 module NAND_Module_1211168 #(parameter N = 4)
2   (input [N-1:0] X, input [N-1:0] Y, output reg [N+1:0] O);
3   // Define input and output
4   = always @* begin
5     // calculate the result
6     O = ~(X & Y);
7   end
8 endmodule
```

6. XOR_Module_1211168

```
1 module XOR_Module_1211168 #(parameter N = 4)
2     (input [N-1:0] X, input [N-1:0] Y, output reg [N+1:0] O);
3     // Define input and output
4     always @* begin
5         // calculate the result
6         O = (X ^ Y);
7     end
8 endmodule
```

7. NOR_Module_1211168

```
1 module NOR_Module_1211168 #(parameter N = 4)
2     (input [N-1:0] X, input [N-1:0] Y, output reg [N+1:0] O);
3     // Define input and output
4     always @* begin
5         // calculate the result
6         O = ~(X | Y);
7     end
8 endmodule
```

8. NOT_Module_1211168

```
1 module NOT_Module_1211168 #(parameter N = 4)
2     (input [N-1:0] X, output reg [N+1:0] O);
3     // Define input and output
4     always @* begin
5         // calculate the result
6         O = ~ (X);
7     end
8 endmodule
```

9. mux8to1_1211168

```
1 module mux8to1_1211168 #(parameter N = 4) ( input [N+1:0]
2     result0,result1,result2,result3,result4,result5,result6,result7,
3     input [2:0] sel, output reg [N+1:0] O);
4     // Define input and output for the mux
5     always @* begin
6         // by the selection concatenate between the input and output
7         case(sel)
8             3'b000: O = result0;
9             3'b001: O = result1;
10            3'b010: O = result2;
11            3'b011: O = result3;
12            3'b100: O = result4;
13            3'b101: O = result5;
14            3'b110: O = result6;
15            3'b111: O = result7;
16        endcase
17    end
18 endmodule
```

D) structural Verilog model for ALU

```
1 module structuralALU_1211168 #(parameter N = 4) (  
2     input [N-1:0] X,  
3     input [N-1:0] Y,  
4     input [2:0] C,  
5     output [N+1:0] O );  
6     // Define input and output for this ALU  
7  
8     // Define wires to save the result for each operation  
9     wire [N+1:0] result0, result1, result2, result3, result4, result5, result6, result7;  
10  
11     // Define wires to save temp res  
12     wire [N+1:0] op1;  
13     wire [N-1:0] op2, op3;  
14  
15     // operation (000)  
16     fullAdder_1211168 #(N) FA_0 (.X(X), .Y(Y), .Out(op1));  
17     divider_Nplus2bits_1211168 #(N) RS_0 (.X(op1), .O(result0));  
18  
19     // operation (001)  
20     multiplier_1211168 #(N) M_0 (.X(op1), .O(result1));  
21  
22     // operation (010)  
23     divider_Nbits_1211168 #(N) RS_1 (.X(X), .O(op2));  
24     fullAdder_1211168 #(N) FA_1 (.X(op2), .Y(Y), .Out(result2));  
25  
26     // operation (011)  
27     divider_Nbits_1211168 #(N) RS_2 (.X(Y), .O(op3));  
28     fullAdder_1211168 #(N) FA_2 (.X(X), .Y(-op3), .Out(result3));  
29  
30     // operation (100)  
31     NAND_Module_1211168 #(N) NAND_0 (.X(X), .Y(Y), .O(result4));  
32  
33     // operation (101)  
34     NOT_Module_1211168 #(N) NOT_0 (.X(X), .O(result5));  
35  
36     // operation (110)  
37     NOR_Module_1211168 #(N) NOR_0 (.X(X), .Y(Y), .O(result6));  
38  
39     // operation (111)  
40     XOR_Module_1211168 #(N) XOR_0 (.X(X), .Y(Y), .O(result7));  
41  
42     // send all result with selection to Mux8to1 to define the final Output  
43     mux8to1_1211168 #(N) MUX_0 (.result0(result0),.result1(result1),  
44         .result2(result2), .result3(result3),.result4(result4),  
45         .result5(result5),.result6(result6),.result7(result7),  
46         .sel(C),.O(O)  
47     );  
48 endmodule
```

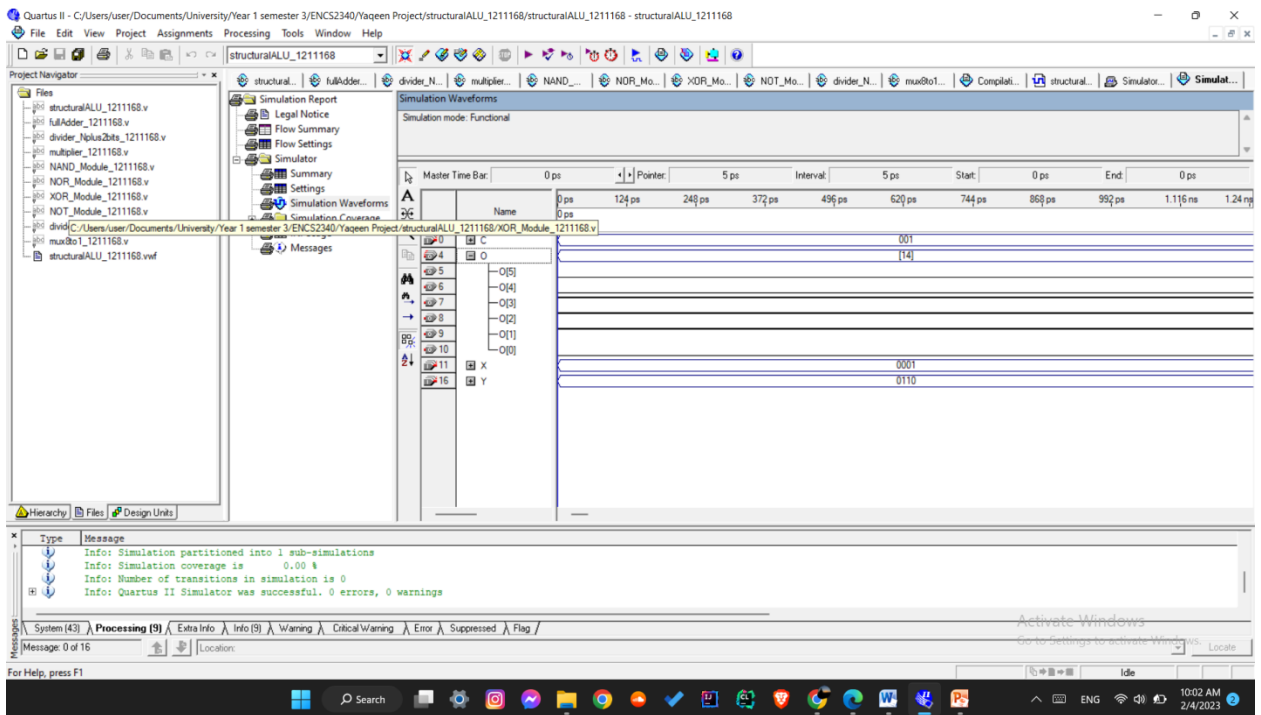
E) Waveform of the Structural ALU

My number 1211168 ; The Form: $1 C_2 Y_2 X_2 C_1 Y_1 X_1$

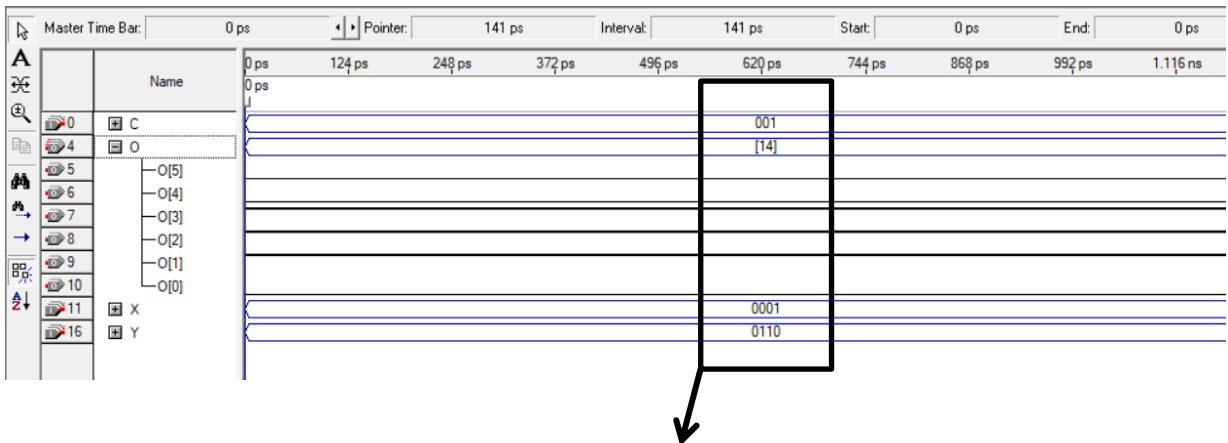
Then $C_2 = 2 Y_2 = 1 X_2 = 1$ $C_1 = 1 Y_1 = 6 X_1 = 1$

| <i>Test</i> | <i>X</i> | <i>Y</i> | <i>C</i> | <i>Operation</i> |
|-------------|------------|------------|-----------|------------------|
| 1 | $X_1 = 1$ | $Y_1 = 6$ | $C_1 = 1$ | $2*(1+6)$ |
| 2 | $X_2 = 1$ | $Y_2 = 1$ | $C_2 = 2$ | $(1/2)+(1)$ |
| 3 | $X_3 = -1$ | $Y_3 = -6$ | $C_3 = 2$ | $(-1/2)+(-6)$ |

Waveform for test 1:



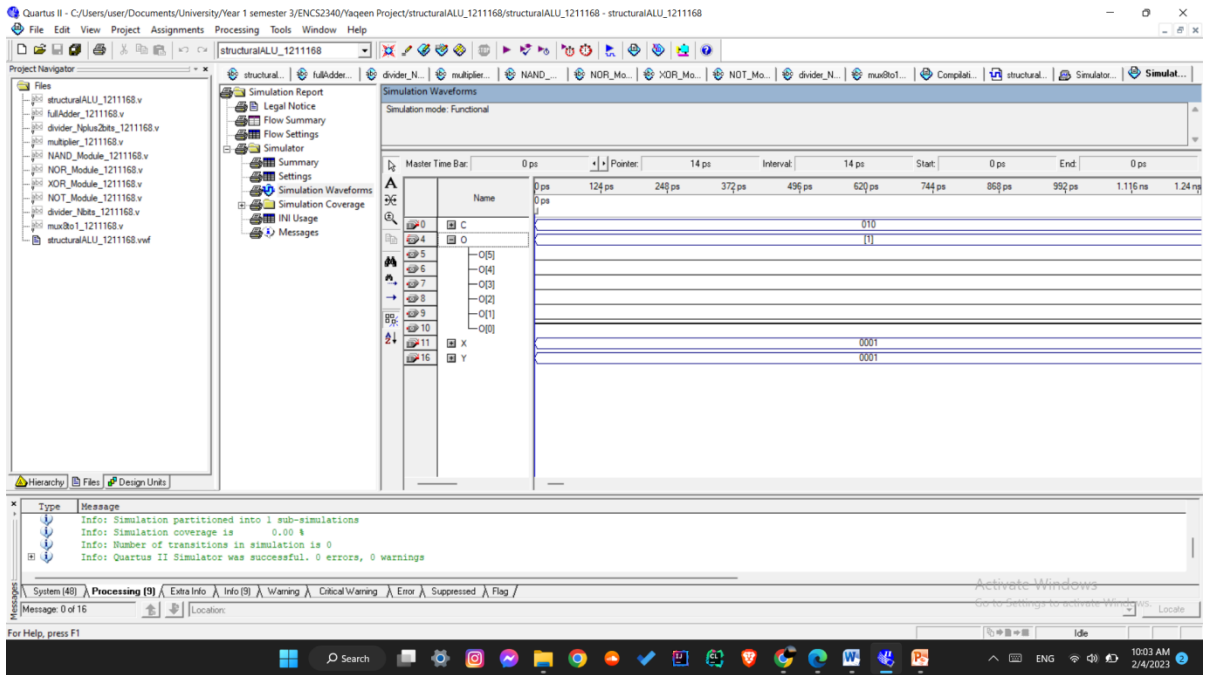
Zoom:



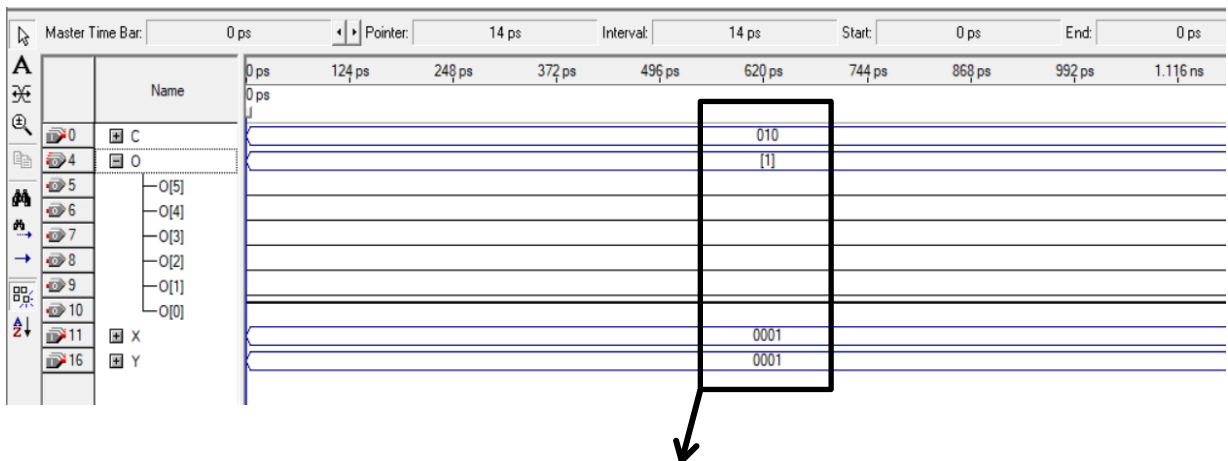
The result for the first test equal (001110)

$$2*(1+6) = 14$$

Waveform for test 2:



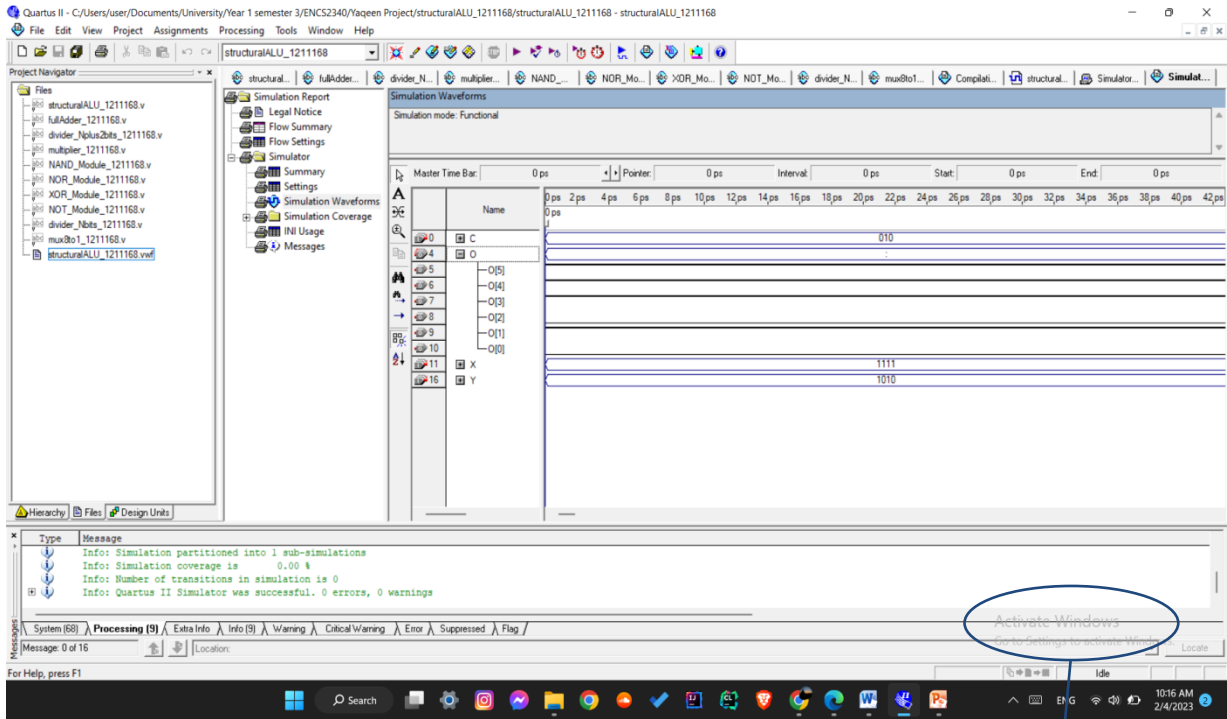
Zoom:



The result for the first test equal (000001)

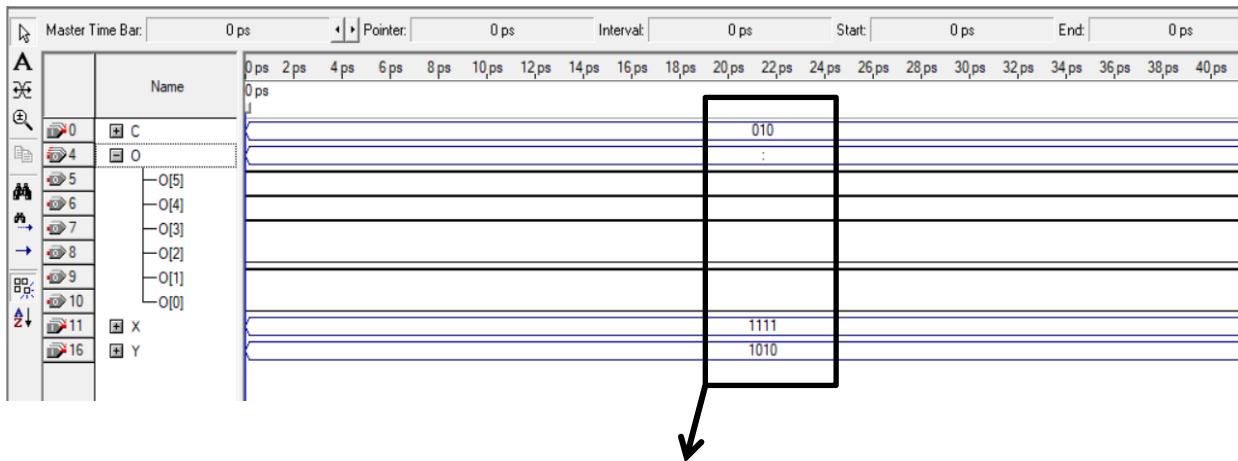
$$(1/2)+(1) = 1$$

Waveform for test 3:



I know I need to activate windows

Zoom:



The result for the first test equal (111010)

$$(-1/2)+(-6) = -6$$

F) Behavioral Verilog module for ALU

In the previous part we designed Structural ALU with his component but in this part we designed a behavioral ALU as we shown.

The behavioral code in Verilog for this ALU:

```
1  module behavioralALU_1211168 #(parameter N = 4)
2    (input signed [N-1:0] X, Y, input signed [2:0] C, output reg signed[N+1:0] O);
3    // define input & output for ALU
4
5    // define a zero-extension for logic operation
6    wire ex = 2'b00;
7
8    always @* begin
9        // check the selection and calculate result
10       case (C)
11           3'b000: O = (X + Y)/2;
12           3'b001: O = 2*(X + Y);
13           3'b010: O = (X/2) + Y;
14           3'b011: O = X - (Y/2);
15           3'b100: O = {ex, ~(X & Y)};
16           3'b101: O = {ex, ~X};
17           3'b110: O = {ex, ~(X | Y)};
18           3'b111: O = {ex, X ^ Y};
19       endcase
20     end
21 endmodule
22
```

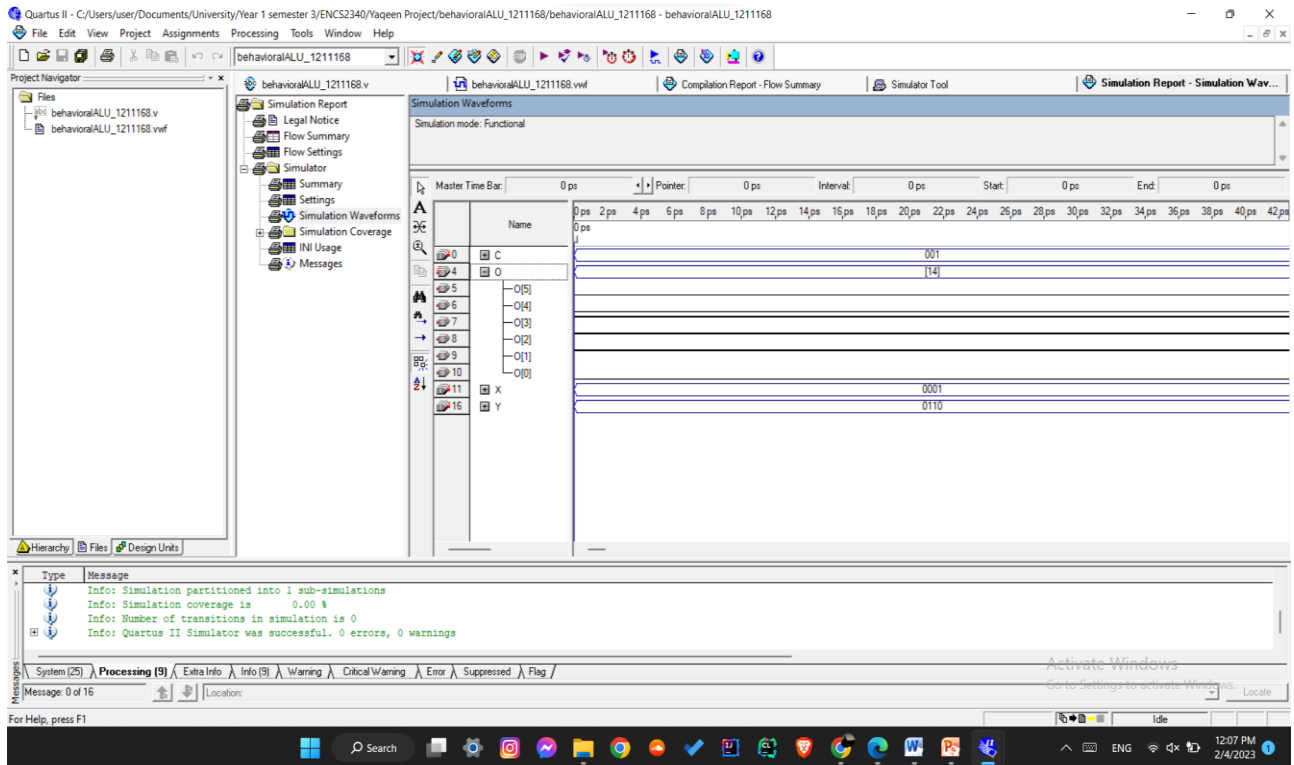
G) Waveform of the Behavioral ALU

My number 1211168 ; The Form: $1 C_2 Y_2 X_2 C_1 Y_1 X_1$

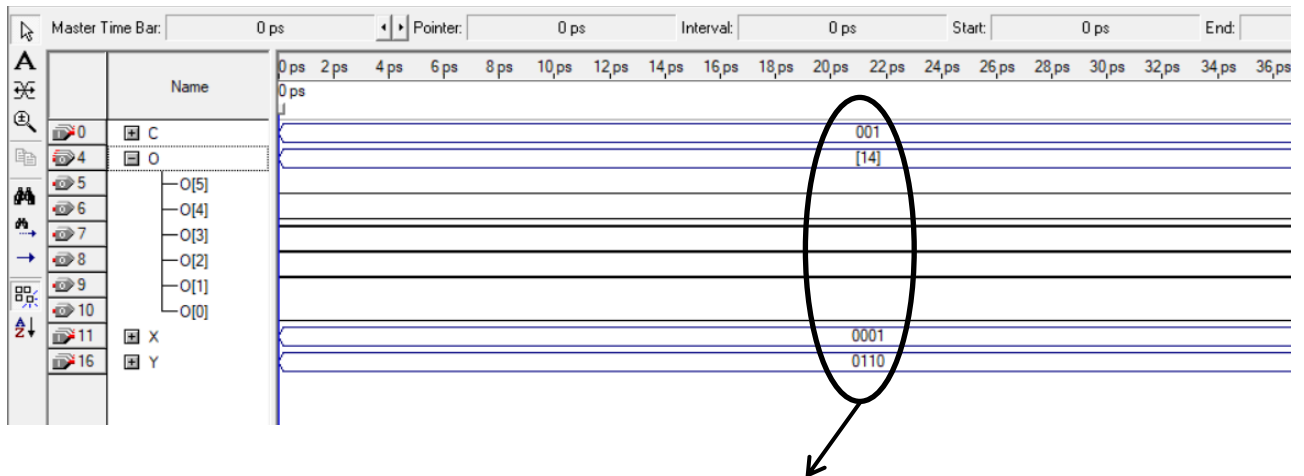
Then $C_2 = 2 Y_2 = 1 X_2 = 1$ $C_1 = 1 Y_1 = 6 X_1 = 1$

| Test | X | Y | C | Operation |
|------|------------|------------|-----------|---------------|
| 1 | $X_1 = 1$ | $Y_1 = 6$ | $C_1 = 1$ | $2*(1+6)$ |
| 2 | $X_2 = 1$ | $Y_2 = 1$ | $C_2 = 2$ | $(1/2)+(1)$ |
| 3 | $X_3 = -1$ | $Y_3 = -6$ | $C_3 = 2$ | $(-1/2)+(-6)$ |

Waveform for test 1:



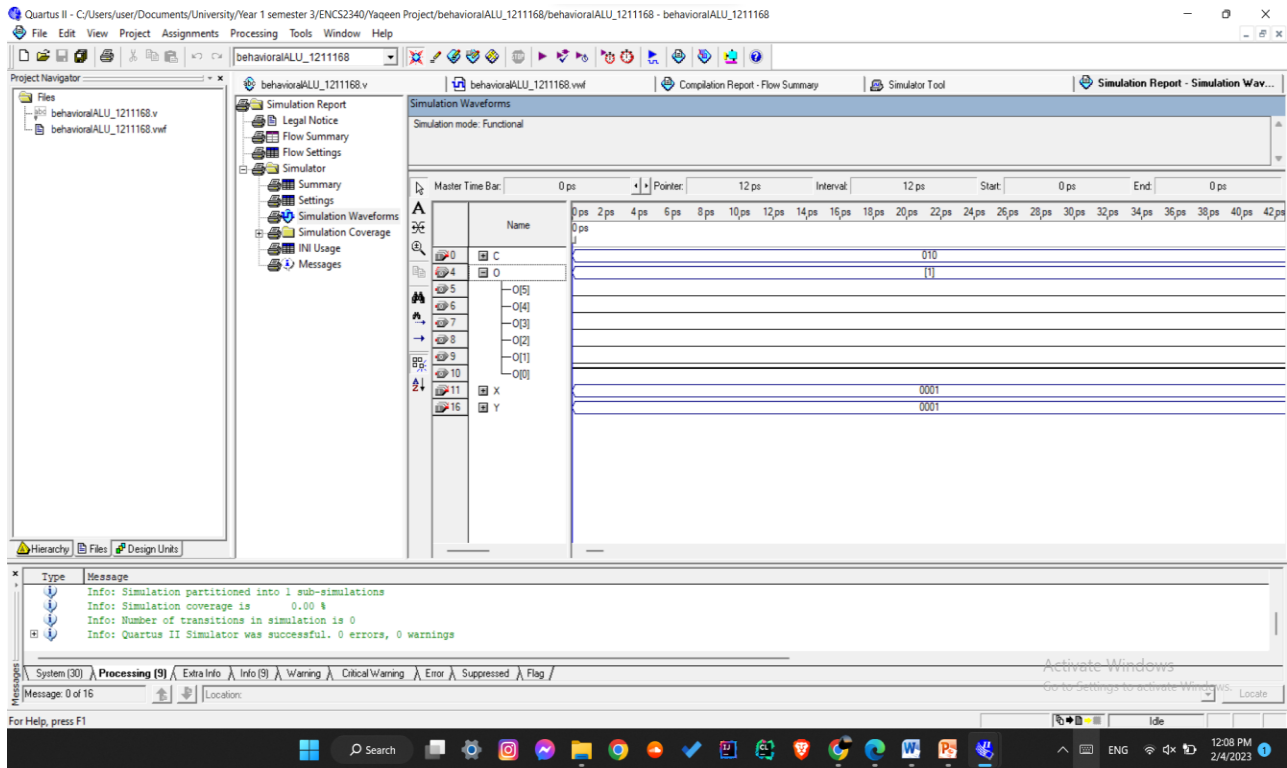
Zoom:



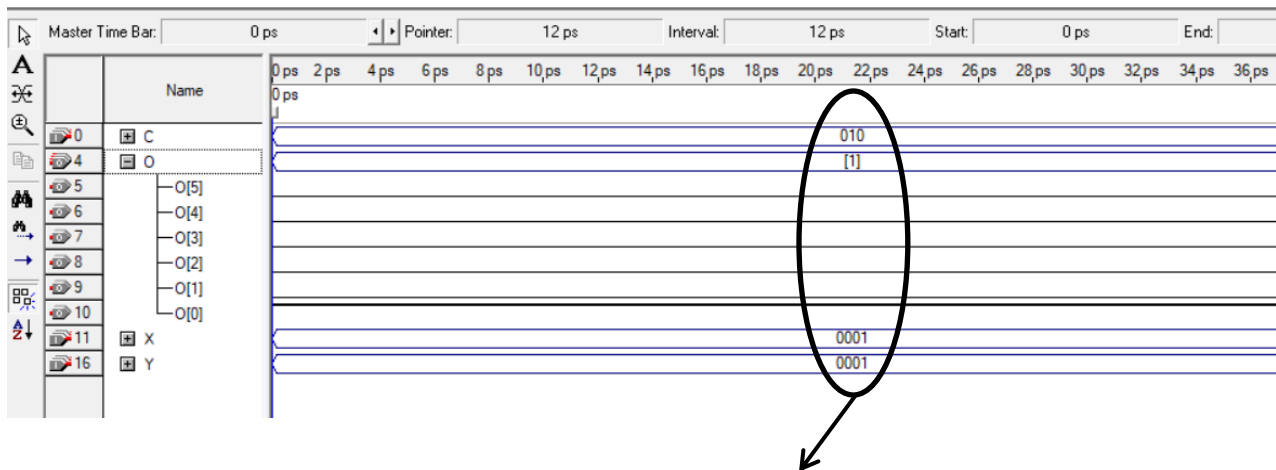
The result for the first test equal (001110)

$$2*(1+6) = 14$$

Waveform for test 2:



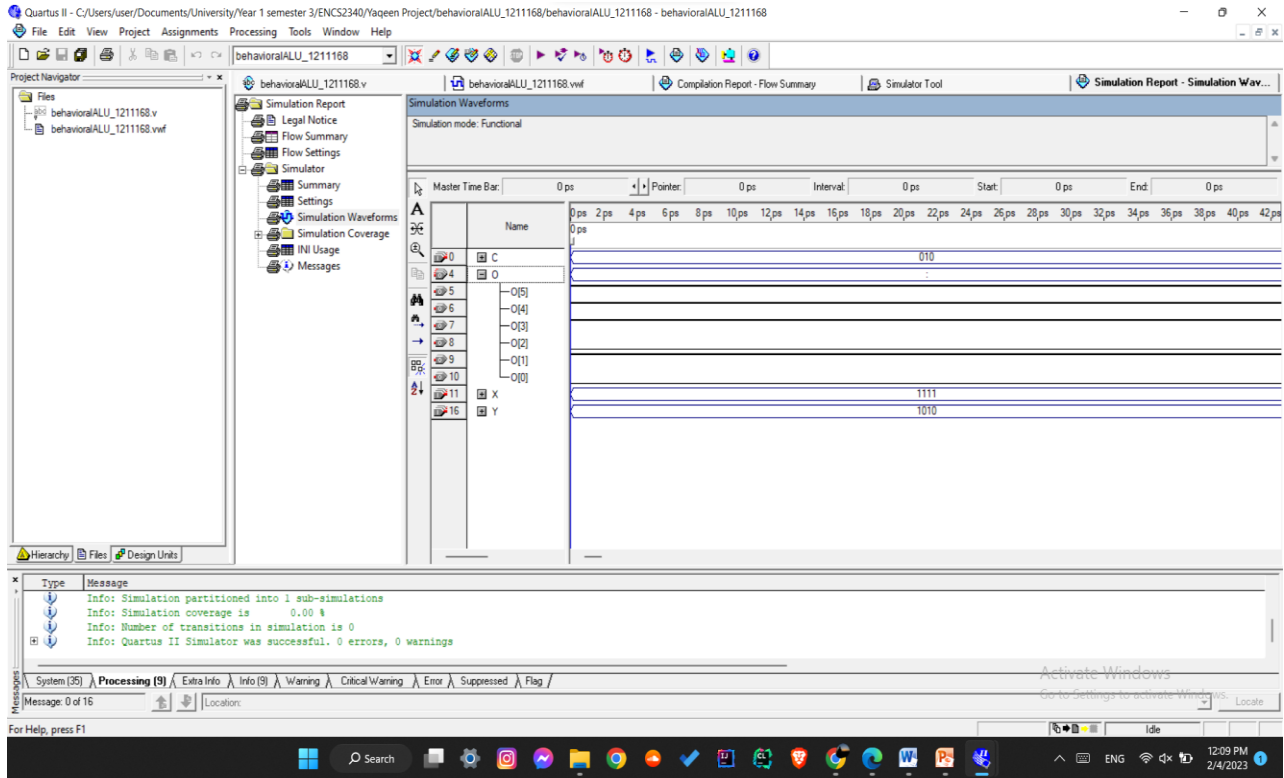
Zoom:



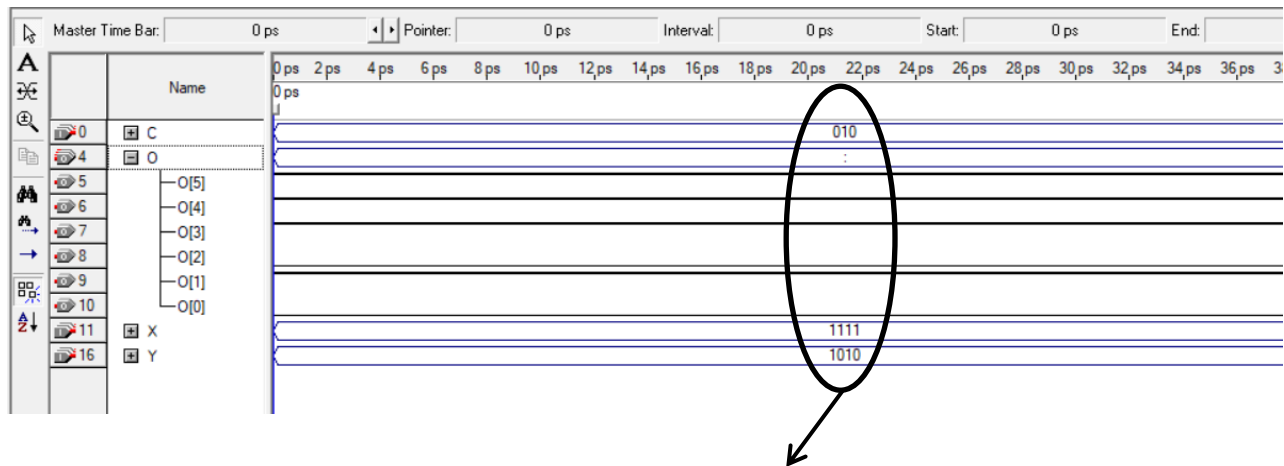
The result for the first test equal (000001)

$$(1/2)+(1) = 1$$

Waveform for test 3:



Zoom:



The result for the first test equal (111010)

$$(-1/2)+(-6) = -6$$