جامعة بيرزيت

BIRZEIT UNIVERSITY

ENCS2340

Project Report

1st semester 22/23

Instructor: Dr. Bilal Karaki

Prepared by: Diaa Badaha

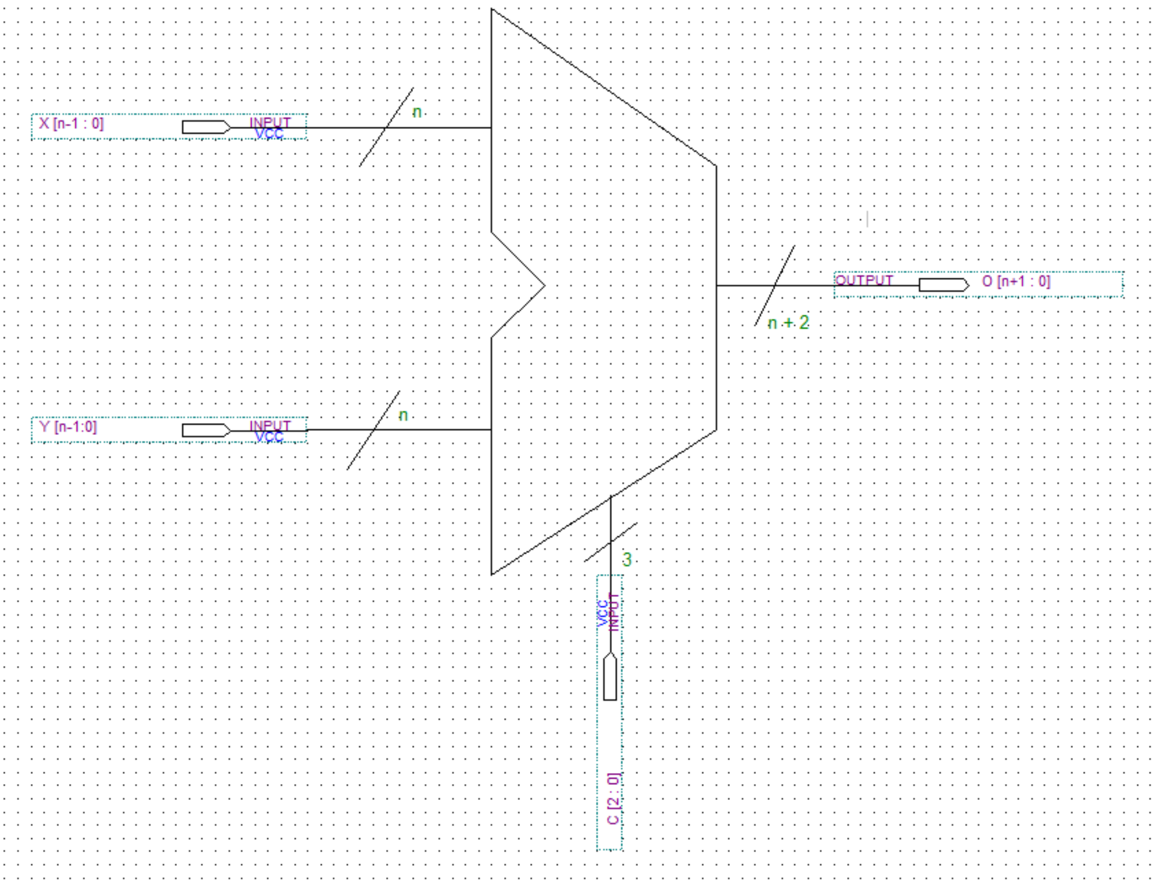
ID: 1210478

Table of contents

Introduction

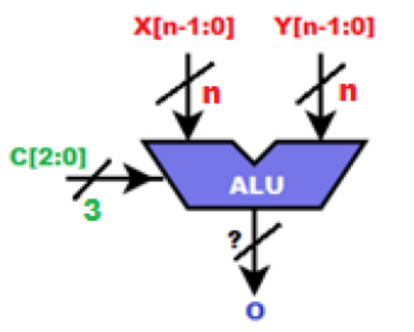
ALU:

An arithmetic logic unit (ALU) is a combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers. This is in contrast to a floating-point unit (FPU), which operates on floating point numbers. An ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs.



Implementation:

This project is designed to do multiple tasks on n-bits binary numbers as shown in the table below.

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**	

Later, answers to the following questions will be mentioned

a) Specify the size of the output (**O**) in bits so the overflow can never occur.
b) Show the ALU implementation using medium-scale integration (MSI) components and minimum number of gates (i.e. in blocks with their sizes). Note that, you might use some kind of extension (sign- or zero-extension).
c) Write behavioral Verilog modules for your elements you defined in Part (b). Be noted that the size of every element you define should be **parameterized**, so that you can vary the design during the testing phase.
d) Write a structural Verilog model for your ALU designed in Part (b) using the elements you defined in Part (c).

e) Generate the waveforms of the ALU defined in Part (d), assumes that X and Y are 4-bits and their values based on your student ID should be set as follows: **2** | P a g e

The general representation of the student ID is **1C₂Y₂X₂C₁Y₁X₁**, so, if your student ID is 1220520, then X, Y, and C values for the three test cases as follows:

Test	X	Y	C	O
1	$X_1 = 0$	$Y_1 = 2$	$C_1 = 5$	NOT(0)
2	$X_2 = 0$	$Y_2 = 2$	$C_2 = 2$	((0)/2)+(2)
3	$X_3 = -X_1$	$Y_3 = -Y_1$	$C_3 = C_2$	((0)/2)+(-2)

Answers

a) Specify the size of the output (**O)** in bits so the overflow can never occur.

Main explanation :

- ❖ The output of an addition operation of 2 numbers with n bits for each one requires (n+1) bits → (Sum & Carry[overflow])

- ❖ The output of a subtraction operation of two numbers with n bits for each one needs the same number of bits as an addition operation → (n+1) bits, since the subtraction is converted to addition of 2's complement before doing the operation.

- ❖ Multiplying an n-bits number by 2 is the same is as shifting each bit to the left by one bit, and the most left bit is shifted into the carry flag, which means that the output of this operation needs n+1 bits.

- ❖ However, dividing an n-bits number by 2 is the same is as shifting each bit to the right by one bit, and the most right bit is shifted into the carry flag, but since we add zero on the left of the number(zero's on the left don't affect the value) the output of this operation needs the same number of bits in the input (n) bits.

Specific explanation :

- **X NAND Y, NOT(X), X NOR Y, X XOR Y:**
  These are logical operations in which the number of bits in the output is the same as the number of input's bits.

- **(X + Y) / 2:**
  As mentioned before, the operation of adding X & Y requires (n+1) bits to represent it. And the division of the summation by 2 doesn't affect on the number of input's bits. Which means the result of this operation needs (n+1) bits.

- **(X / 2) + Y, X – (Y / 2):**
  As the previous point, these operations consist of (addition or subtraction) and division operations, division does not affect the number of bits in the input, addition or subtraction requires one more bit than bits in the input. As a result, these operations needs (n+1) bits to represent them.

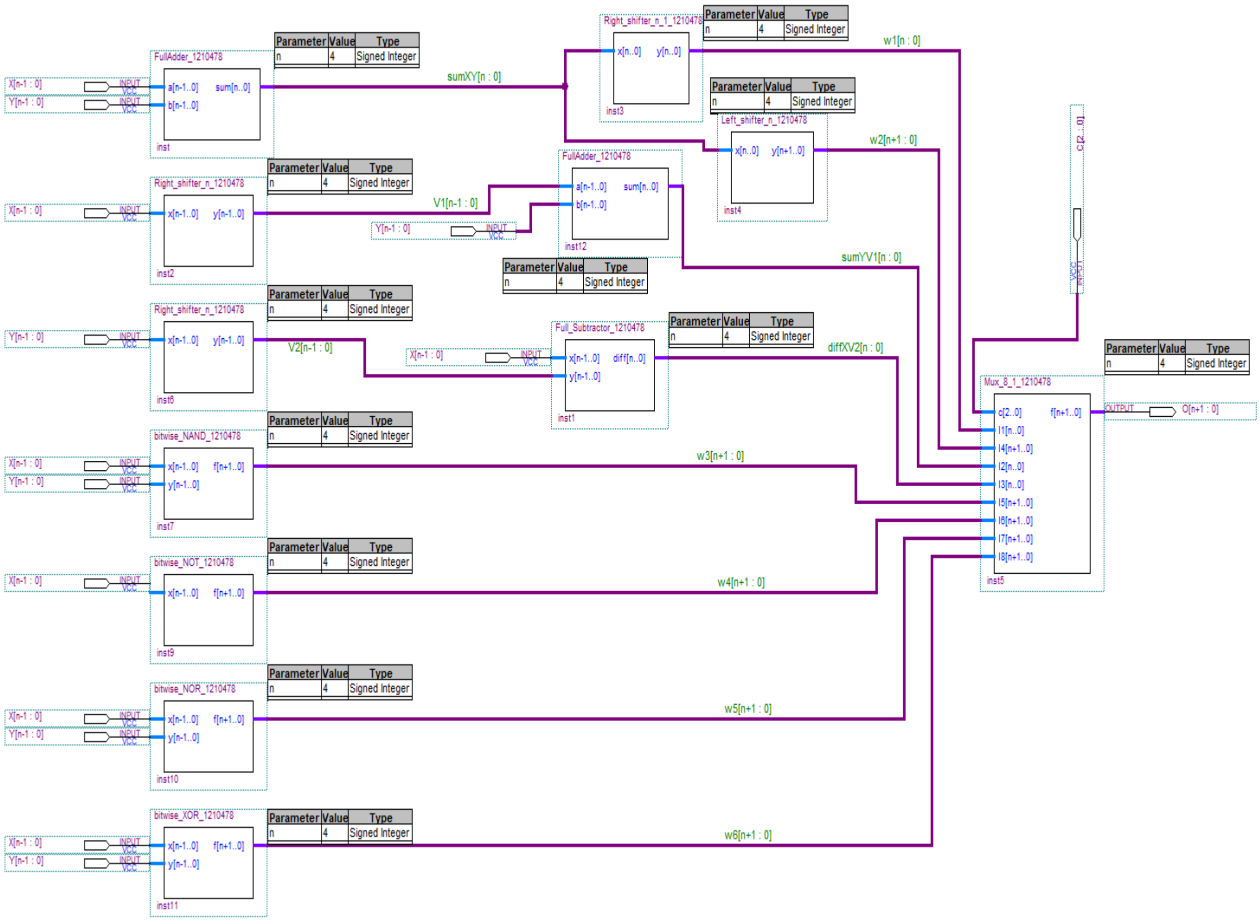
- **2 * (X + Y):**
  Let X + Y = Z, which needs (n+1) bits to represent, the operation of multiplication Z by 2 requires ((no. of bits in Z) + 1), that equals (n+2) bits.

Summary :

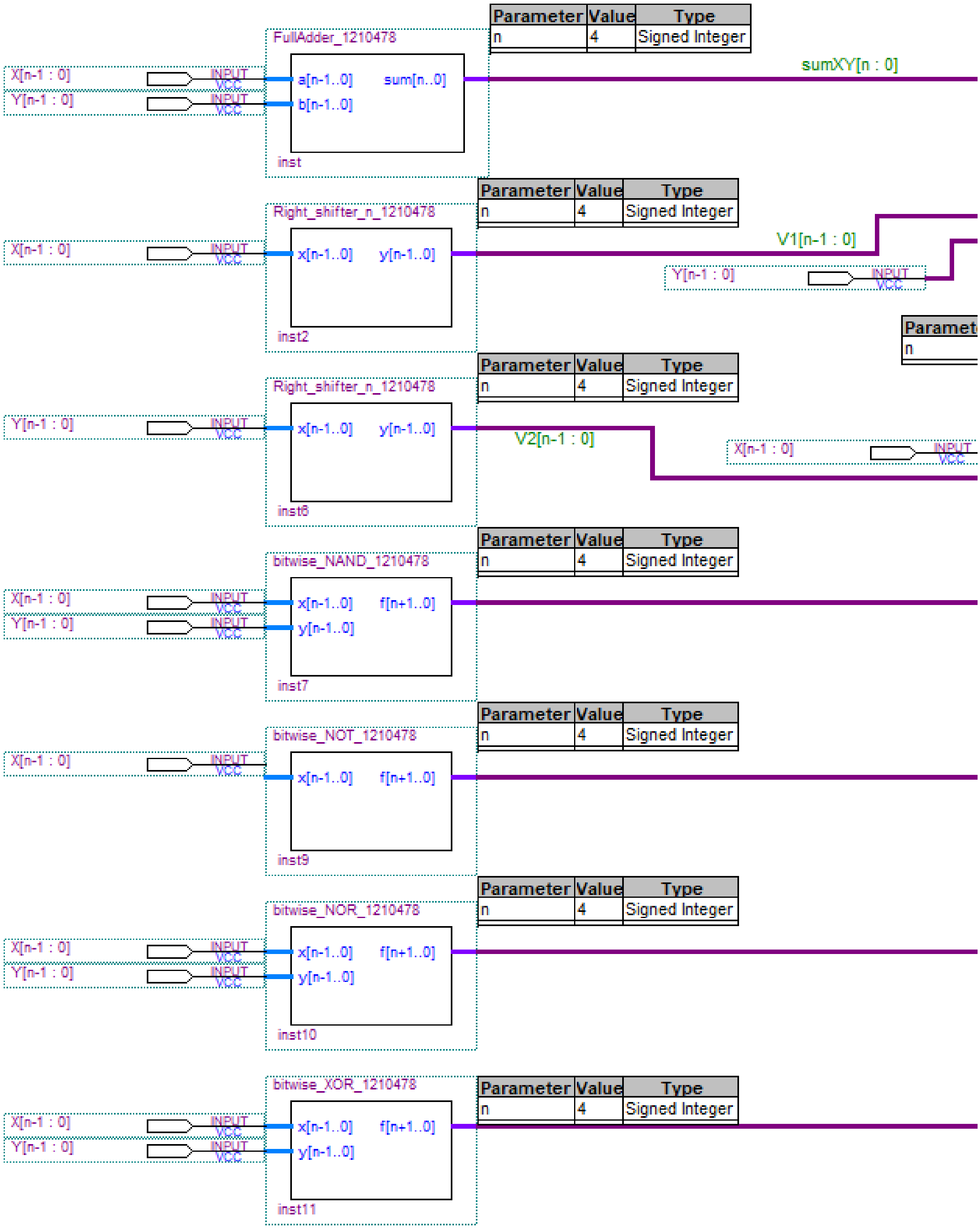
Operation	Arithmetic	Logic	No. of bits
(X + Y) / 2	✓		n+1
2 * (X + Y)	✓		n+2
(X / 2) + Y	✓		n+1
X – (Y / 2)	✓		n+1
X NAND Y		✓	n
NOT(X)		✓	n
X NOR Y		✓	n
X XOR Y		✓	n

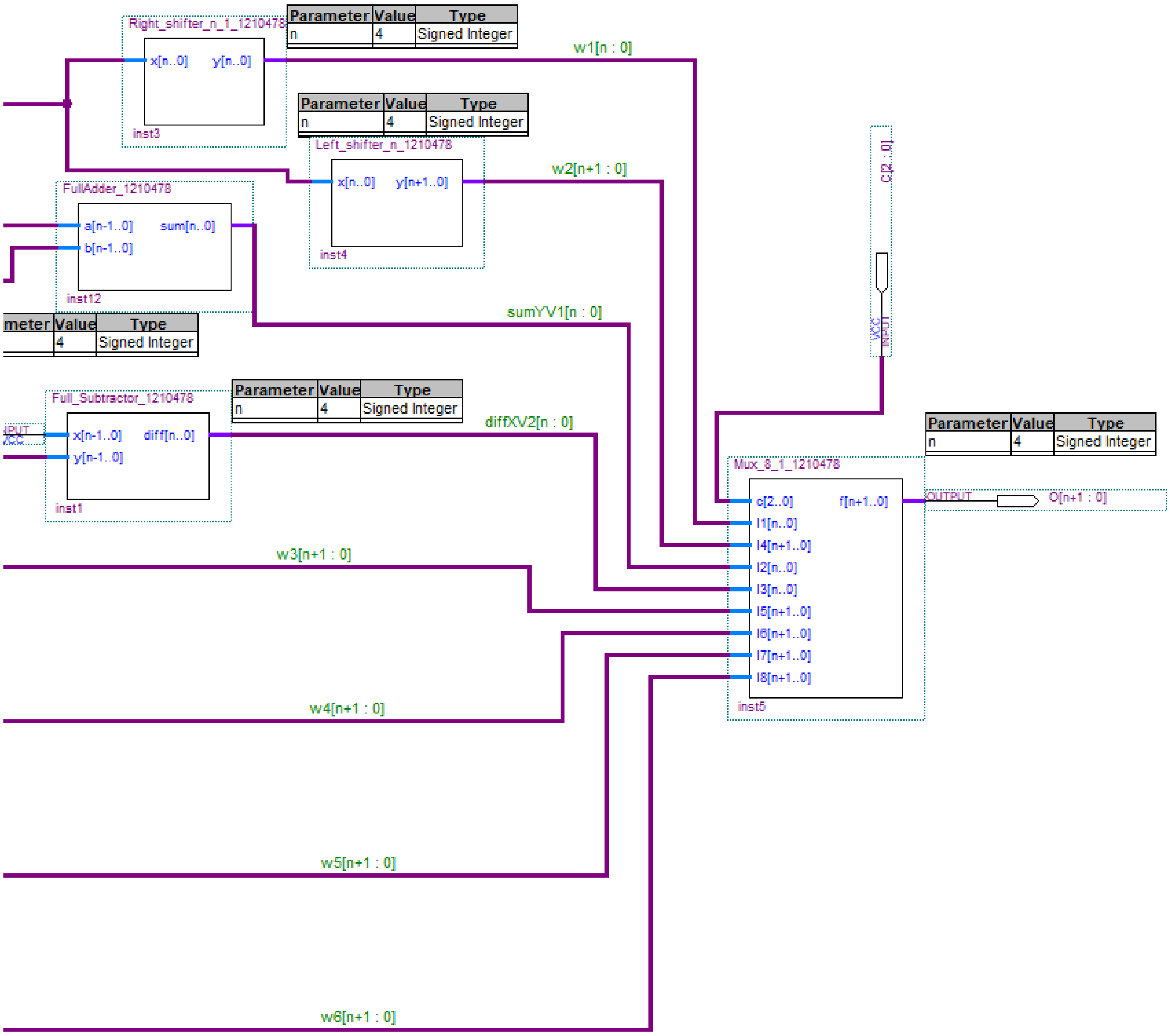
b) Show the ALU implementation using medium-scale integration (MSI) components and minimum number of gates (i.e. in blocks with their sizes). Note that, you might use some kind of extension (sign- or zero-extension).

ALU implementation: -This is the main answer for question (b)-



ALU implementation(divided into two photos):



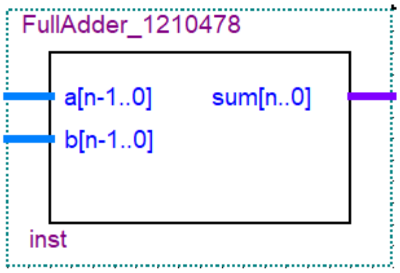

Right_shifter_n_1_1210478
Parameter Value Type
n 4 Signed Integer
x[n..0] y[n..0]
inst3
w1[n : 0]
Left_shifter_n_1210478
x[n..0] y[n+1..0]
inst4
w2[n+1 : 0]
FullAdder_1210478
a[n-1..0] sum[n..0]
b[n-1..0]
inst12
sumYV1[n : 0]
C[2 : 0]
Full_Subtractor_1210478
x[n-1..0] diff[n..0]
y[n-1..0]
inst1
diffXV2[n : 0]
Mux_8_1_1210478
c[2..0] f[n+1..0]
I1[n..0]
I4[n+1..0]
I2[n..0]
I3[n..0]
I5[n+1..0]
I6[n+1..0]
I7[n+1..0]
I8[n+1..0]
inst5
OUTPUT
O[n+1 : 0]
w3[n+1 : 0]
w4[n+1 : 0]
w5[n+1 : 0]
w6[n+1 : 0]

C) Write behavioral Verilog modules for your elements you defined in Part (b). Be noted that the size of every element you define should be **parameterized**, so that you can vary the design during the testing phase.

Used components

Full Adder:

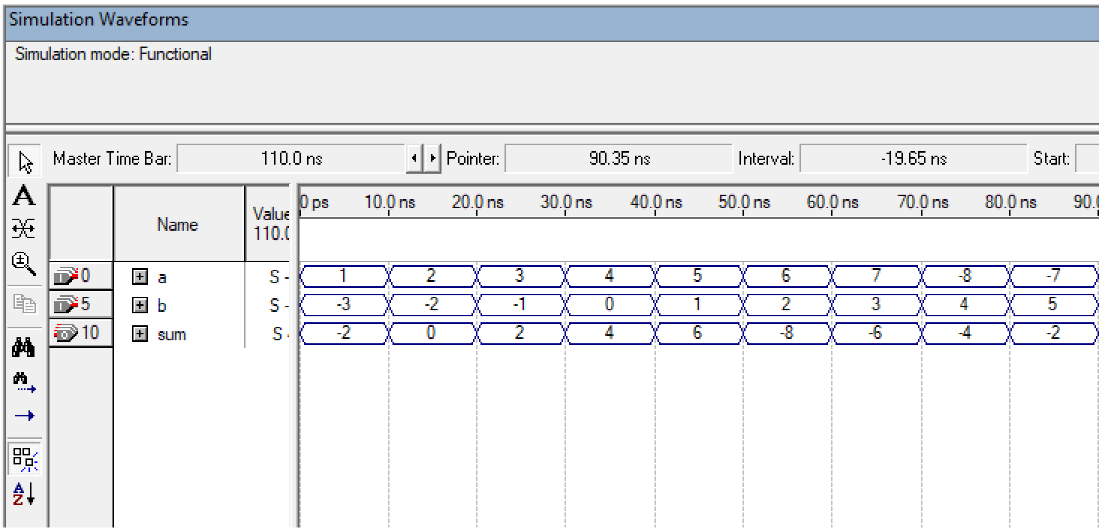
Full_adder circuit has two inputs: X and Y it calculates the summation of them and set it as output (sum).



Code: -from the answer of question (c)-

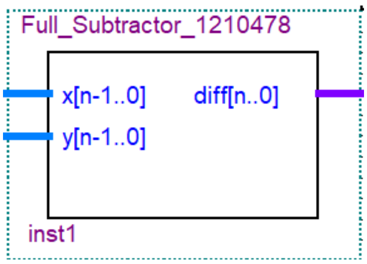
```
module FullAdder_1210478 #(parameter n = 4)/*parameterization*/ (a, b, sum);

input signed [n-1 : 0]a, b;      //declaring inputs
output reg signed [n : 0]sum;    //declaring outputs

always @ (*)
begin
     sum = a + b;  //addition operation
end

endmodule
```

Simulation:


Simulation Waveforms
Simulation mode: Functional
Master Time Bar: 110.0 ns
Pointer: 90.35 ns
Interval: -19.65 ns
Start:
Name
Value 110.0
0 ps
10.0 ns
20.0 ns
30.0 ns
40.0 ns
50.0 ns
60.0 ns
70.0 ns
80.0 ns
0
a
1 2 3 4 5 6 7 -8 -7
5
b
-3 -2 -1 0 1 2 3 4 5
10
sum
-2 0 2 4 6 -8 -6 -4 -2

Full Subtractor:

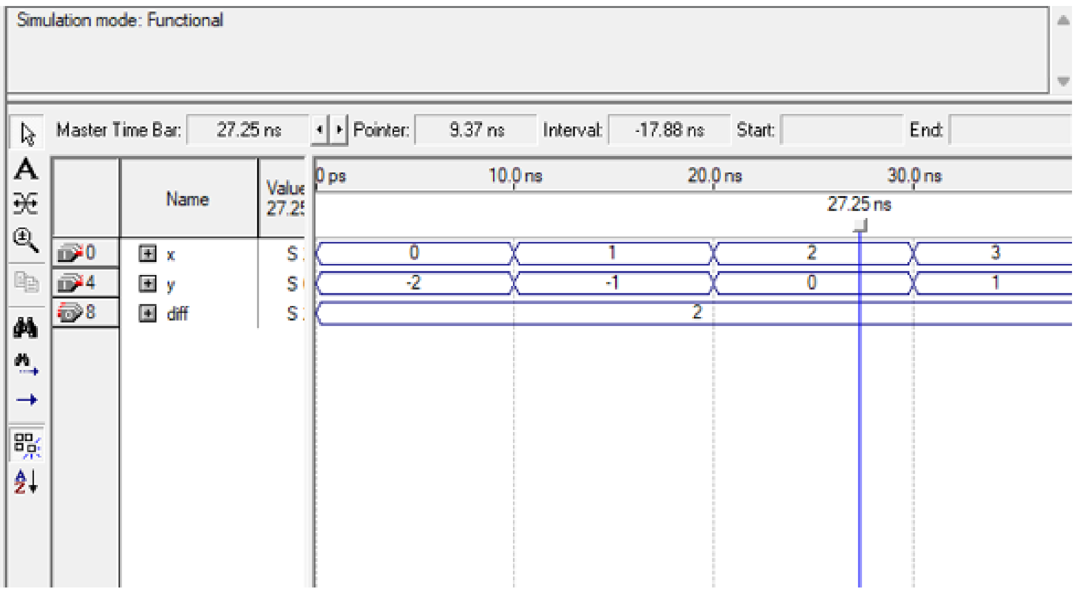
Full_Subtractor_1210478 circuit has two inputs: X & Y it calculates the difference between them as (X – Y) and sets it as output (diff).



Code: -from the answer of question (c)-

```
module Full_Subtractor_1210478 #(parameter n = 4)/*parameterization*/ (x, y, diff);

input signed [n-1 : 0]x, y;       //declaring inputs
output reg signed [n : 0]diff;  //declaring outputs

always @(*)
    begin
            diff = x - y;        //subtraction operation
    end
endmodule
```

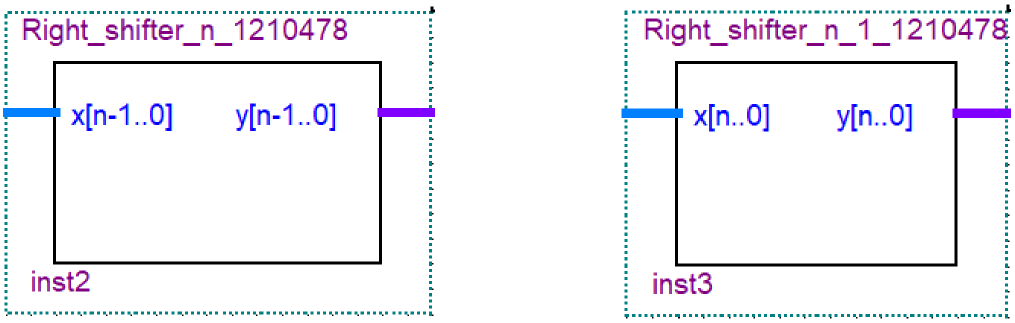
Simulation:



Shifters (Dividers & Multipliers)

Right shifters (Dividers):

I made tow dividers circuits, in order to get the same number of bits in inputs & outputs in ALU circuit to get an accuracy results. In general, R_shifter_n_1210478 (which has inputs and outputs of n bits) and R_shifter_n_1_1210478 (which has inputs and outputs of n+1 bits) circuit take one number as input (x) and find the division by 2 operation and set it as output (y).

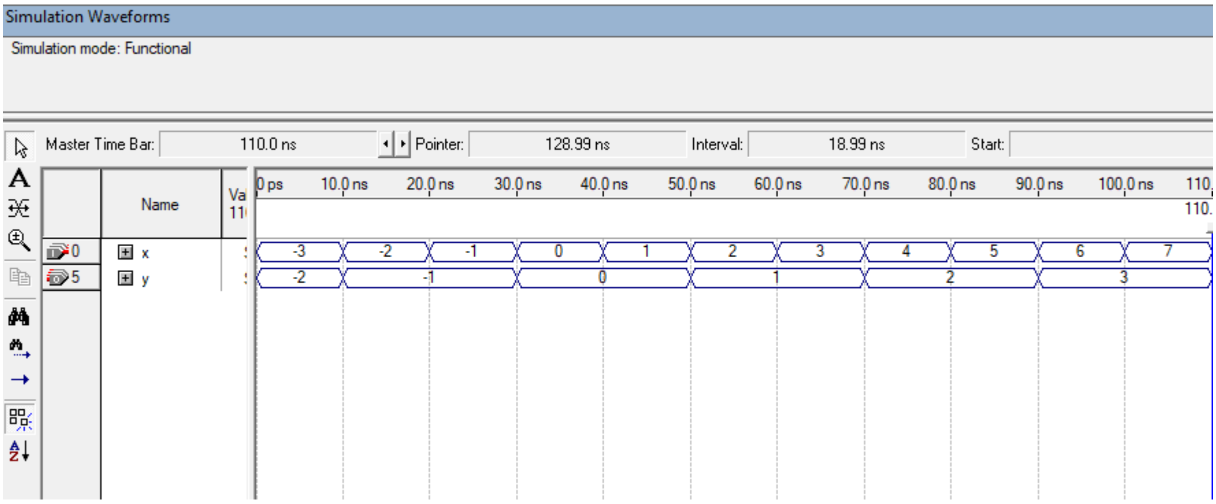


Codes: -from the answer of question (c)-

```
module Right_shifter_n_1210478 #(parameter n = 4)/*parameterization*/ (x, y);

input signed [n-1 : 0]x;        //declaring input
output reg signed [n-1 : 0]y;   //declaring output

always @(x)
    begin
         y = x / 2;      //dividing operation
    end

endmodule
```

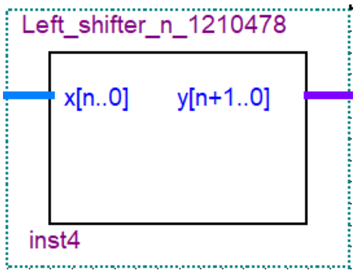
```
module Right_shifter_n_1_1210478 #(parameter n = 4)/*parameterization*/ (x, y);

input signed [n : 0]x;        //declaring input
output reg signed [n : 0]y;   //declaring output

always @(x)
    begin
         y = x / 2;         //dividing operation
    end

endmodule
```

Simulation: *Simulation for both circuits is the same



Left shifters (Multiplier):

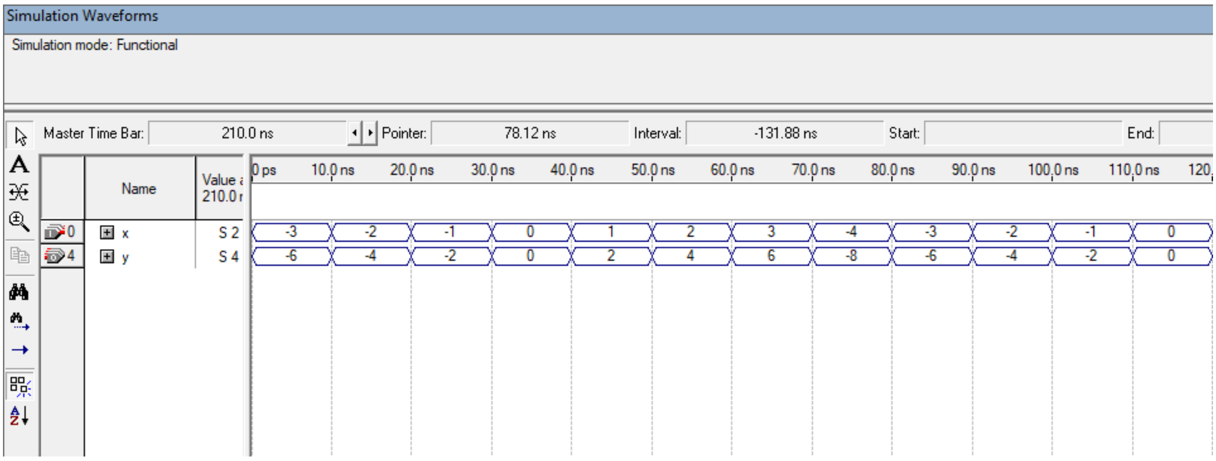
L_shifter_n_1210478 circuit take one number as input (x) and find the multiplication by 2 operation and set it as output (y).



Code: -from the answer of question (c)-

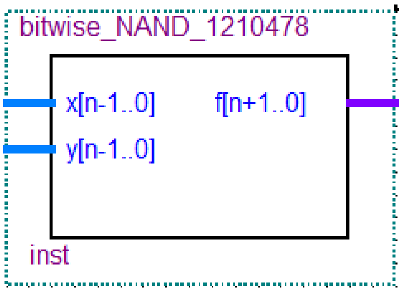
```
module Left_shifter_n_1210478 #(parameter n = 4)/*parameterization*/ (x, y);

input signed [n : 0]x;        //declaring input
output reg signed [n+1 : 0]y;  //declaring output

always @(x)
    begin
        y = x * 2;
    end
endmodule
```

Simulation:



Bitwise NAND:

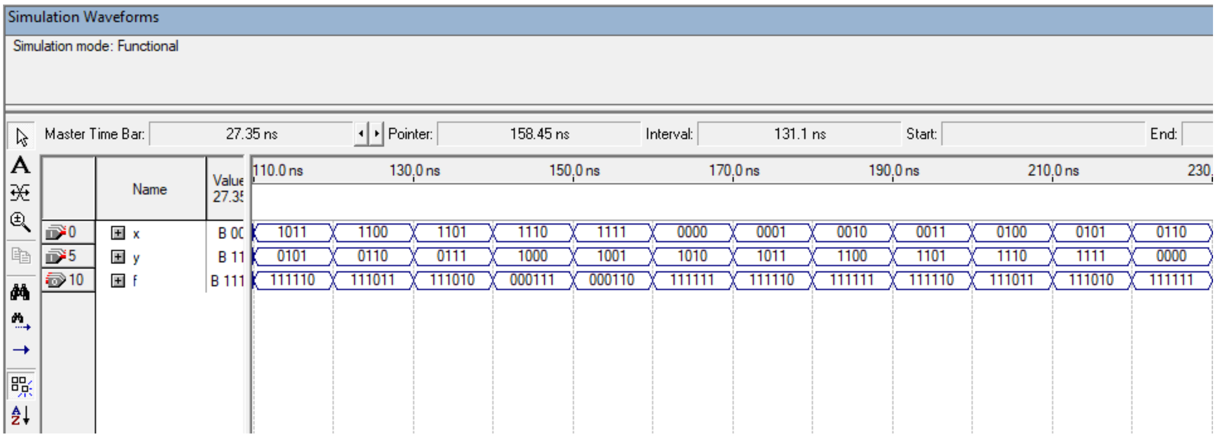
This block has two n-bit inputs (X and Y), and one output (f), The bitwise_nand circuit check X and Y bit by bit, if they are both one, the result of this bit in the f output will be zero, otherwise, it will be one.



Code: -from the answer of question (c)-

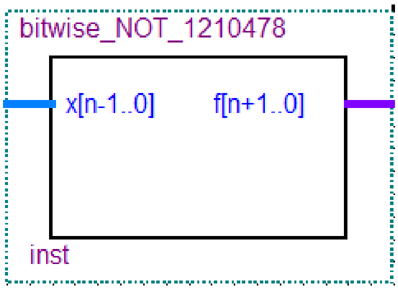
```
module bitwise_NAND_1210478 #(parameter n = 4)/*parameterization*/ (x, y, f);

input [n-1 : 0]x, y;     //declaring inputs
output reg [n+1 : 0]f;   //declaring output

always @ (x, y)
    begin
        f = ~(x & y);        //bitwise nand operation
        f[n+1] = f[n-1];     //(setting extra bits as the n's bit to make
        f[n] = f[n-1];       //the number of bits in the output equals n+2)
    end
endmodule
```

Simulation:



Bitwise NOT:

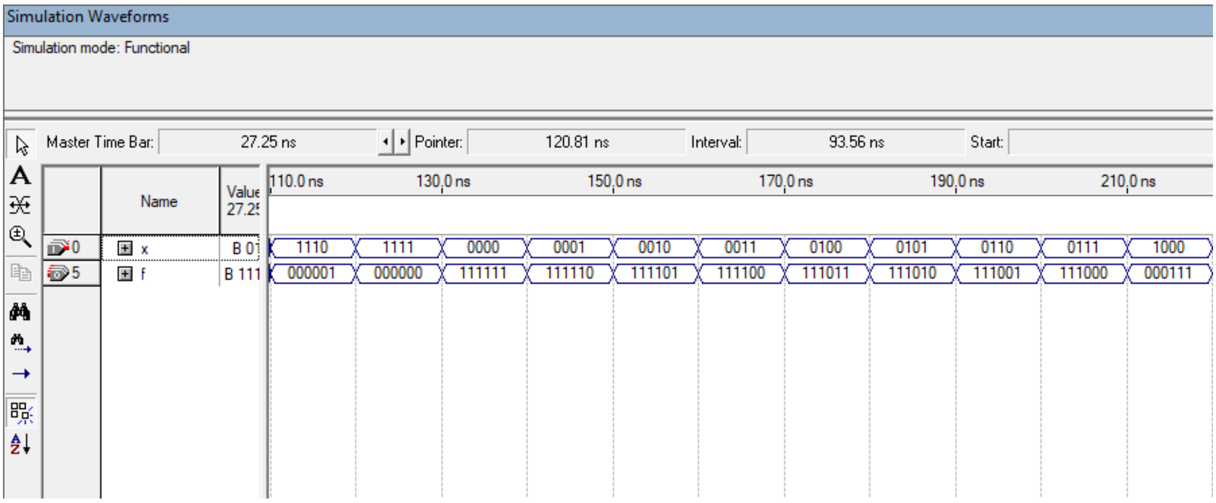
This block has one n-bit input (X), and one output (f). The bitwise_nor circuit check X bit by bit, if the bit is one, the result of this bit in the f output will be zero, otherwise, it will be one.



Code: -from the answer of question (c)-

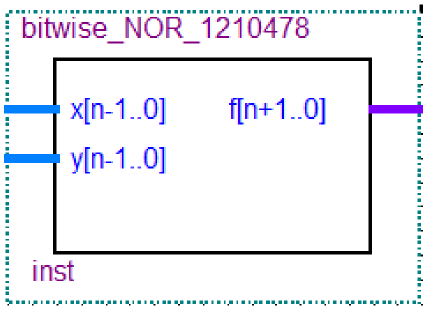
```
module bitwise_NOT_1210478 #(parameter n = 4)/*parameterization*/ (x, f);

input [n-1 : 0]x;          //declaring input
output reg [n+1 : 0]f;     //declaring output

always @ (x)
    begin
        f = ~(x);              //bitwise not operation
        f[n+1] = f[n-1];       //(setting extra bits as the n's bit to make
        f[n] = f[n-1];         //the number of bits in the output equals n+2)
    end
endmodule
```

Simulation:



Bitwise NOR:

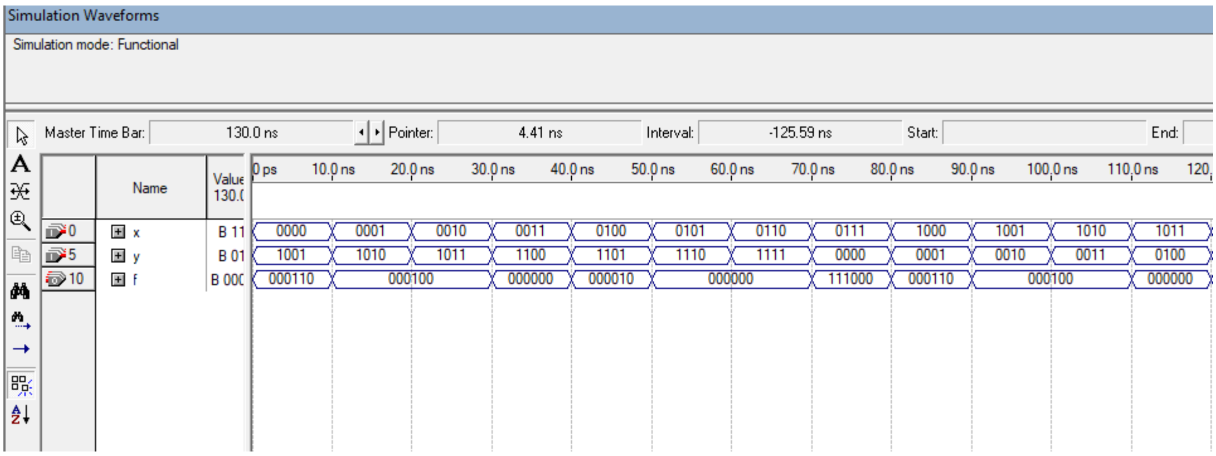
This block has two n-bit inputs (X and Y), and one output (f), The bitwise_nor circuit check X and Y bit by bit, if they are both zero, the result of this bit in the f output will be one, otherwise, it will be zero.



Code: -from the answer of question (c)-

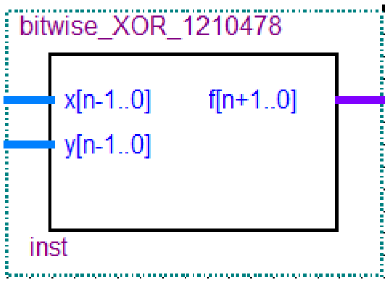
```
module bitwise_NOR_1210478 #(parameter n = 4)/*parameterization*/ (x, y, f);

input [n-1 : 0]x, y;     //declaring inputs
output reg [n+1 : 0]f;   //declaring output

always @ (x, y)
    begin
        f = ~(x | y);        //bitwise nor operation
        f[n+1] = f[n-1];     //(setting extra bits as the n's bit to make
        f[n] = f[n-1];       //the number of bits in the output equals n+2)
    end
endmodule
```

Simulation:



Bitwise XOR:

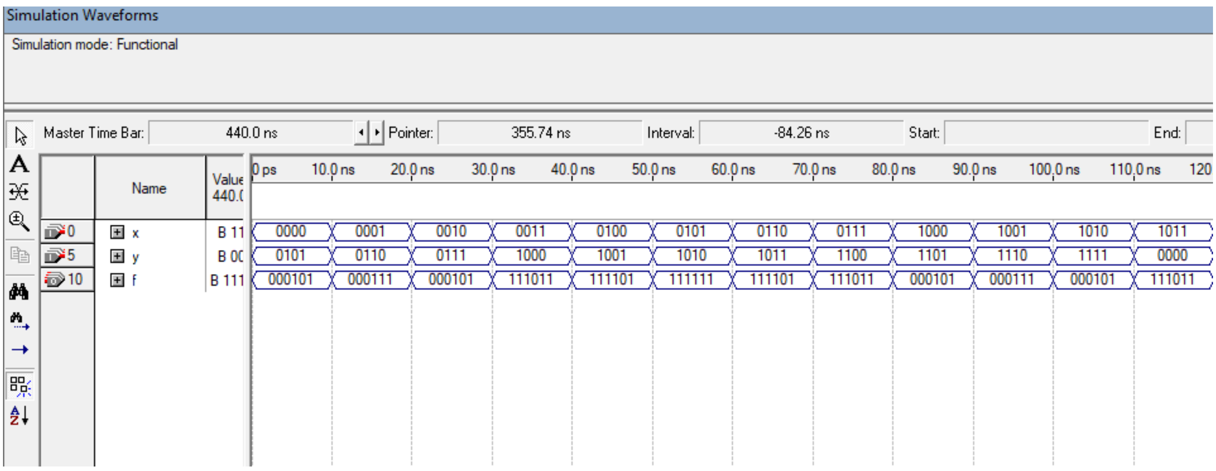
This block has two n-bit inputs (X and Y), and one output (f), The bitwise_nor circuit check X and Y bit by bit, if they are both zero or both one, the result of this bit in th f output will be zero, otherwise, it will be one.



Code: -from the answer of question (c)-

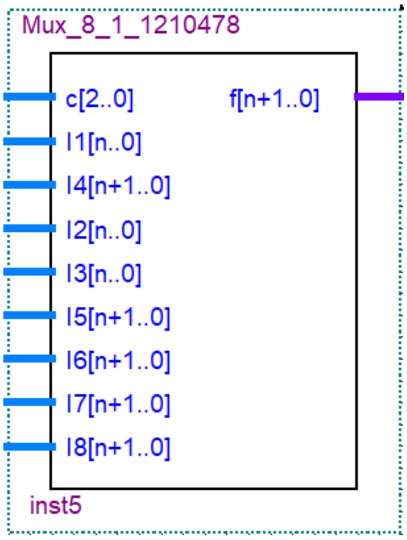
```
module bitwise_XOR_1210478 #(parameter n = 4)/*parameterization*/ (x, y, f);

input [n-1 : 0]x, y;     //declaring inputs
output reg [n+1 : 0]f;   //declaring output

always @ (x, y)
    begin
        f = x ^ y;          //bitwise xor operation
        f[n+1] = f[n-1];    //(setting extra bits as the n's bit to make
        f[n] = f[n-1];      //the number of bits in the output equals n+2)
    end
endmodule
```

Simulation:



8-1 Multiplexer:

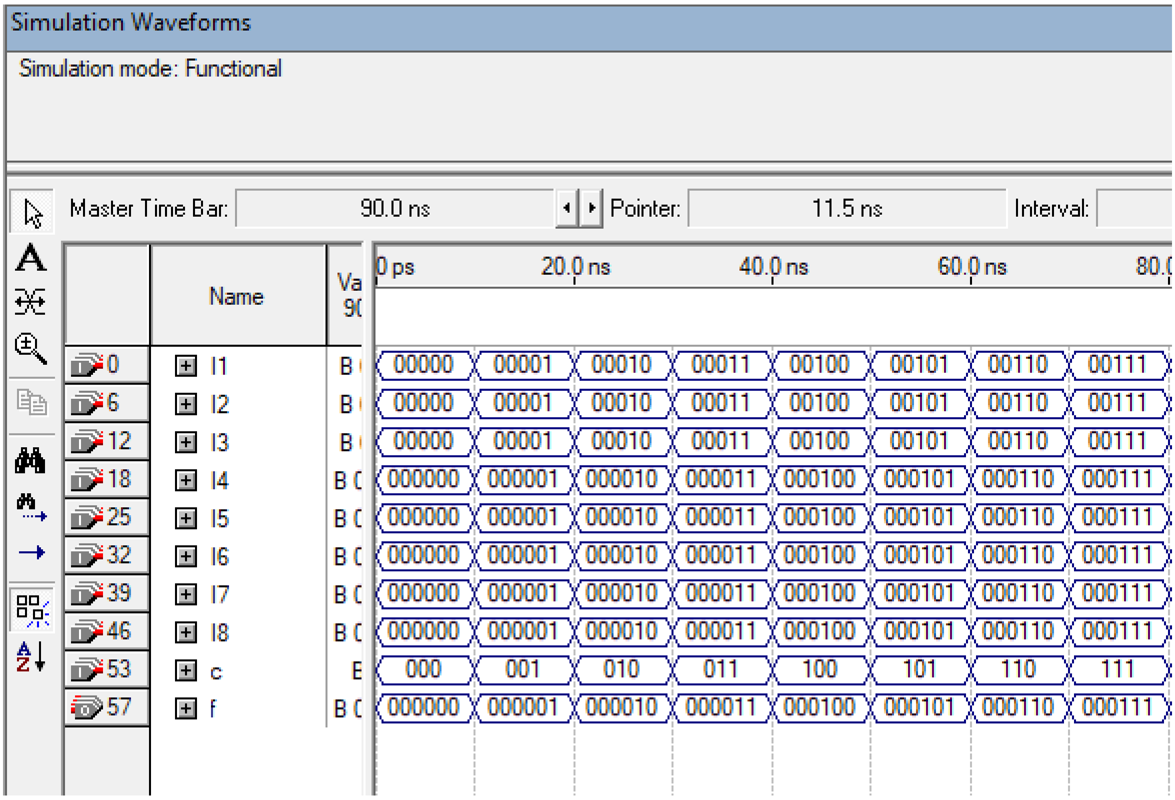
This multiplexer has **five (n+2) bits inputs**, **three (n+1) bits inputs**, **selection ((3) bits input)** and **one (n+2) bits output**. The output has value same as one of the inputs depending on the selection input (the 3-bit input).



Code: -from the answer of question (c)-

```
module Mux_8_1_1210478 # (parameter n = 4) (c, I1, I4, I2, I3, I5, I6, I7, I8, f);
input signed [n : 0] I1, I2, I3;
input signed [n+1 : 0]I4, I5, I6, I7, I8; //declaring inputs
input [2 : 0]c;               //declaring selection as input
output reg signed [n+1 : 0]f;   //declaring output

always @(c, I1, I4, I2, I3, I5, I6, I7, I8)
    begin
    //setting values of the output depending on selection
        if (c == 'b000)
            f = I1;
        else if (c == 'b001)
            f = I4;
        else if (c == 'b010)
            f = I2;
        else if (c == 'b011)
            f = I3;
        else if (c == 'b100)
            f = I5;
        else if (c == 'b101)
            f = I6;
        else if (c == 'b110)
            f = I7;
        else if (c == 'b111)
            f = I8;
    end

endmodule
```

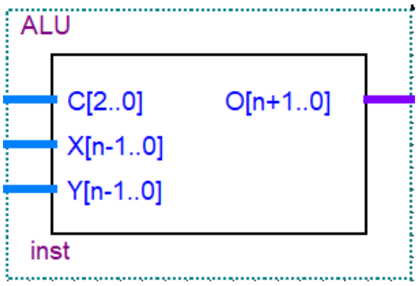
Simulation:



Simulation Waveforms
Simulation mode: Functional
Master Time Bar: 90.0 ns
Pointer: 11.5 ns
Interval:
0 ps
20.0 ns
40.0 ns
60.0 ns
Name
I1
I2
I3
I4
I5
I6
I7
I8
c
f
00000 00001 00010 00011 00100 00101 00110 00111
000000 000001 000010 000011 000100 000101 000110 000111
000 001 010 011 100 101 110 111

ALU:

ALU is the block that collects all of this project components, it has two n-bit inputs that the operations depend on, and one 3-bit input (selection) which determines the operation, Also it has one n-bit output.



D) Write a structural Verilog model for your ALU designed in Part (b) using the elements you defined in Part (c).

ALU (structural) code: -This is the answer for question (d)-

```
module ALU_structural_1210478 #(parameter n = 4) (C, X, Y, O);

input [2:0]C;                              //declaring selection as input
input signed [n-1 : 0]X, Y;                //declaring inputs: X & Y;
output signed [n+1 : 0]O;                  //declaring output: O;
wire signed [n-1 : 0]V1, V2;
wire signed [n : 0]sumXY, sumYV1, diffXV2, w1;
wire signed [n+1 : 0]w2, w3, w4, w5, w6; //declaring wires to carry values

//invoking made modules and setting the result in wires:
FullAdder_1210478 (X, Y, sumXY);
Right_shifter_n_1_1210478 (sumXY, w1);//The result of first operation: (X+Y)/2

Left_shifter_n_1210478 (sumXY, w2);//The result of second operation: 2*(X+Y)

Right_shifter_n_1210478 (X, V1);
FullAdder_1210478 (V1, Y, sumYV1);//The result of third operation: (X/2)+Y

Right_shifter_n_1210478 (Y, V2);
Full_Subtractor_1210478 (X, V2, diffXV2);//The result of 4th operation: X-(Y/2)

//invoking made bitwise-gates and setting the result in wires::
bitwise_NAND_1210478 (X, Y, w3);//The result of 5th operation: X NAND Y
bitwise_NOT_1210478 (X, w4);//The result of 6th operation: NOT X
bitwise_NOR_1210478 (X, Y, w7);//The result of 7th operation: X NOR Y
bitwise_XOR_1210478 (X, Y, w6);//The result of 8th operation: X XOR Y

//invoking the multiplixer module to get the final output:
Mux_8_1_1210478 (C, w1, w2, sumYV1, diffXV2, w3, w4, w5, w6, O);


endmodule
```

E) Generate the waveforms of the ALU defined in Part (d), assumes that X and Y are 4-bits and their values based on your student ID should be set as follows:
The general representation of the student ID is **$1C_2Y_2X_2C_1Y_1X_1$**, so, if your student ID is 1220520, then X, Y, and C values for the three test cases as follows:

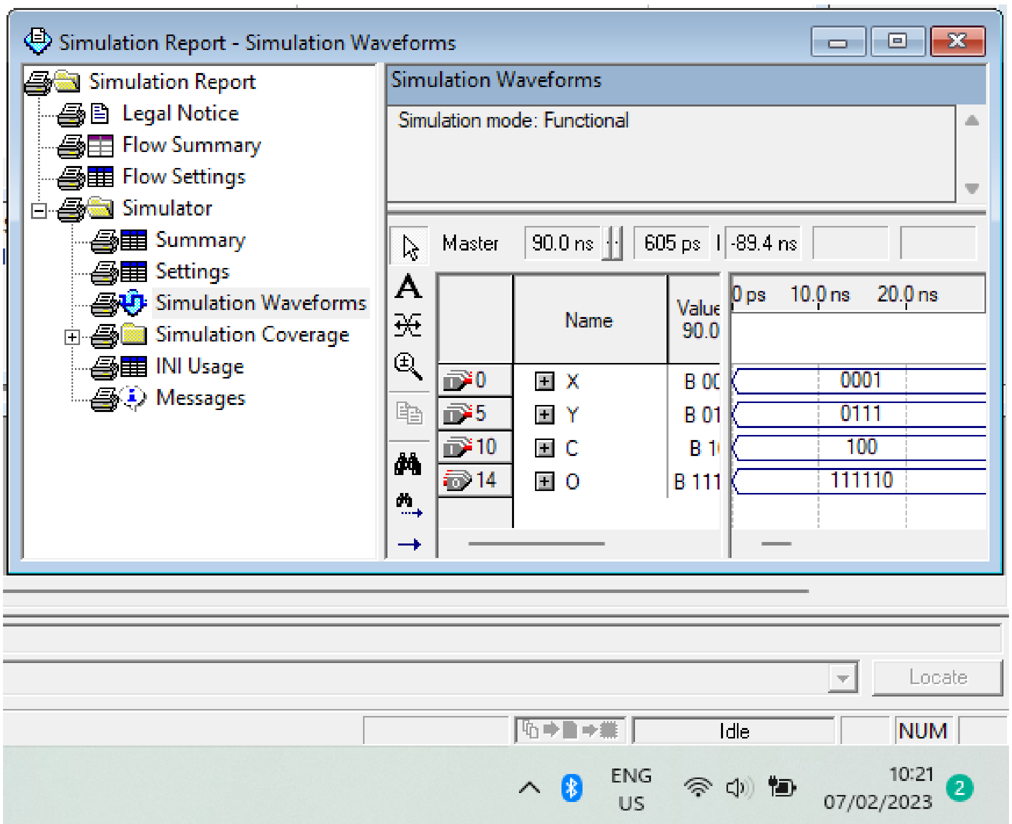
Test	X	Y	C	O
1	$X_1 = 0$	$Y_1 = 2$	$C_1 = 5$	NOT(0)
2	$X_2 = 0$	$Y_2 = 2$	$C_2 = 2$	((0)/2)+(2)
3	$X_3 = -X_1$	$Y_3 = -Y_1$	$C_3 = C_2$	((0)/2)+(-2)

Note: If any value from the set {C_2, Y_2, X_2, C_1, Y_1, X_1} is 8 or 9, you need to replace it by 1

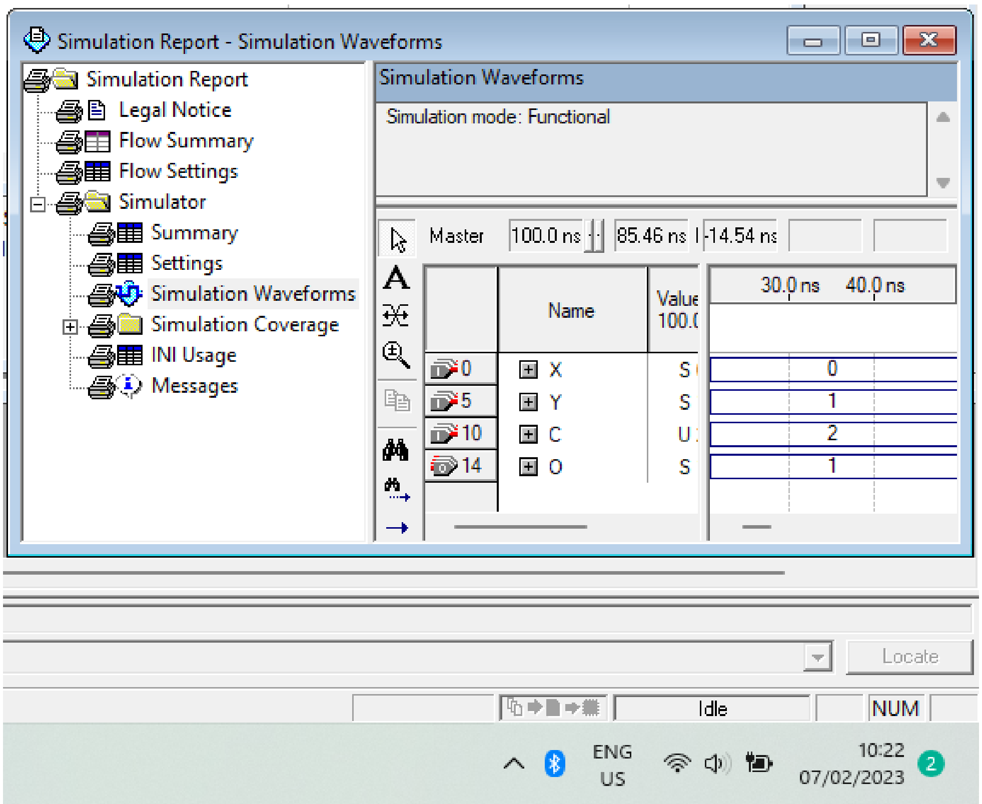
Simulation:-This is the answer for question (e), test by test (depending on my ID: 1210478)-

Test	X	Y	C	O
1	X1 = 1	Y1 = 7	C1 = 4	X NAND Y
2	X2 = 0	Y2 = 1	C2 = 2	((0)/2)+(1) = 1
3	X3 = -1	Y3 = -7	C3 = 2	((-1)/2)+(-7) = -7

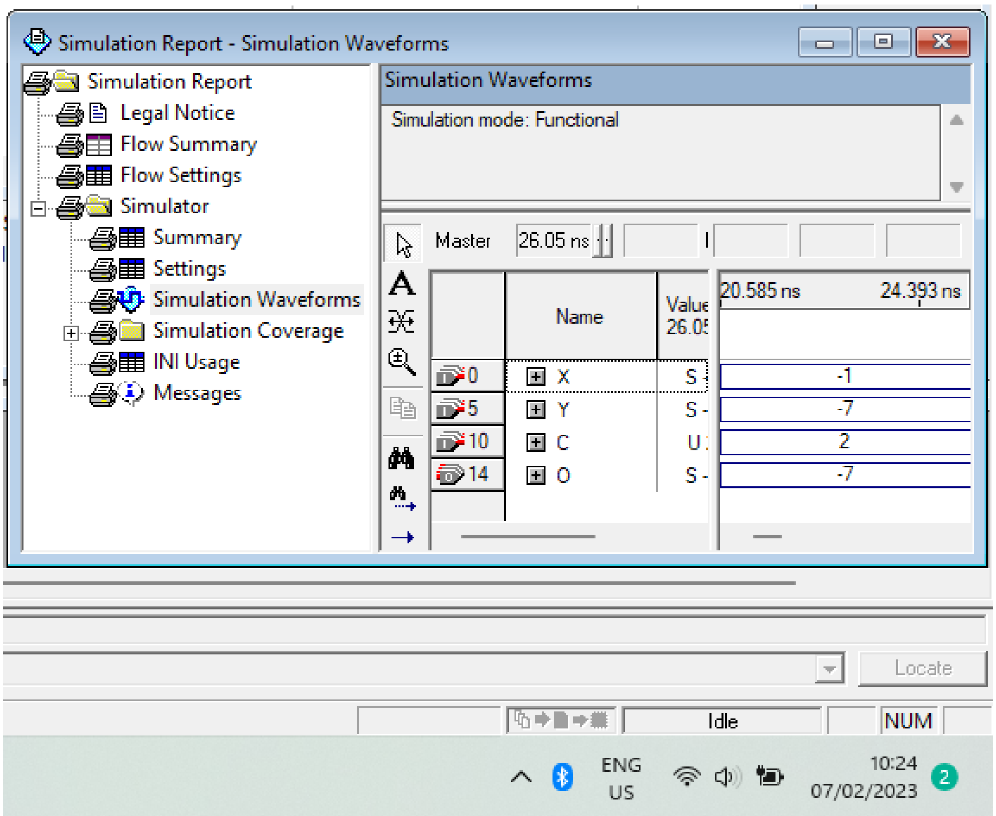
First test



Second test



Third test



f) Write a single behavioral Verilog module that models the designed ALU.

ALU (behavioral) code: -This is the answer for question (f)-

```
module ALU_behavioral_1210478 #(parameter n = 4)  (X, Y, C, O);

input [2:0]C;                       //declaring selection as input
input signed [n-1 : 0]X, Y;         //declaring inputs: X & Y;
output reg signed [n+1 : 0]O;       //declaring output: O;

always @ (X, Y, C)
    begin
    //setting values of the final output depending on selection
        if (C == 'b000)
            O = (X + Y) / 2;
        else if (C == 'b001)
            O = 2 * (X + Y);
        else if (C == 'b010)
            O = (X / 2) + Y;
        else if (C == 'b011)
            O = X - (Y / 2);
        else if (C == 'b100)
            O = ~(X & Y);
        else if (C == 'b101)
            O = ~(X);
        else if (C == 'b110)
            O = ~(X | Y);
        else if (C == 'b111)
            O = X ^ Y;
        else
            O = 0;
    end
endmodule
```

g) Generate the waveforms of the behavioral ALU defined in Part (f), assumes that X and Y are 4-bits and their values based on your student ID should be set as follows:
The general representation of the student ID is **1$C_2Y_2X_2C_1Y_1X_1$**, so, if your student ID is 1220520, then X, Y, and C values for the three test cases as follows:

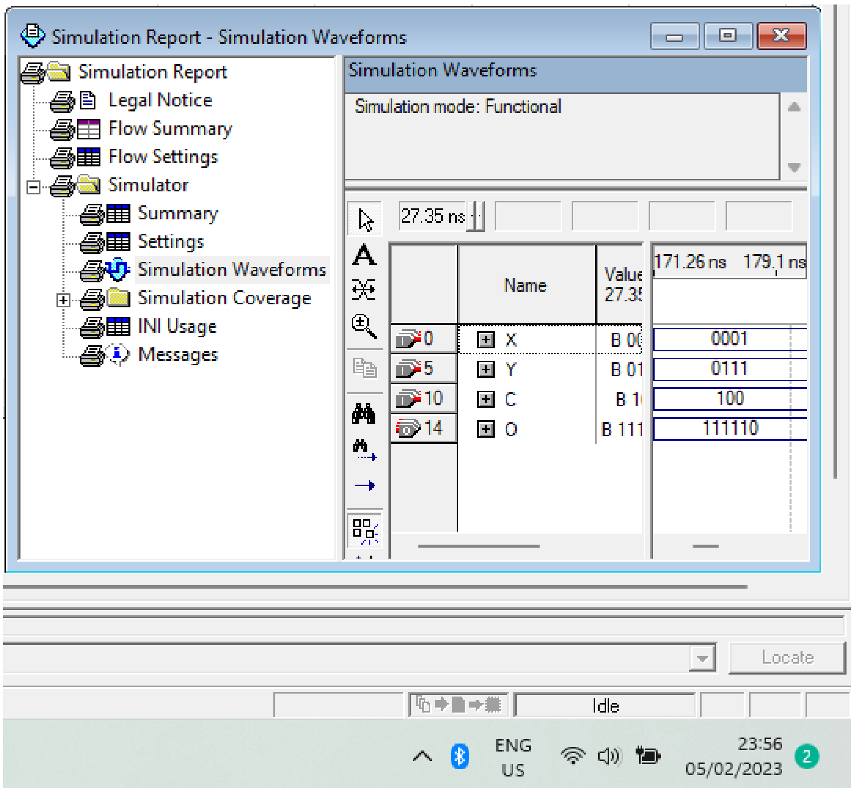
Test	X	Y	C	O
1	$X_1 = 0$	$Y_1 = 2$	$C_1 = 5$	NOT(0)
2	$X_2 = 0$	$Y_2 = 2$	$C_2 = 2$	((0)/2)+(2)
3	$X_3 = -X_1$	$Y_3 = -Y_1$	$C_3 = C_2$	((0)/2)+(-2)

Note: If any value from the set {C_2, Y_2, X_2, C_1, Y_1, X_1} is 8 or 9, you need to replace it by 1

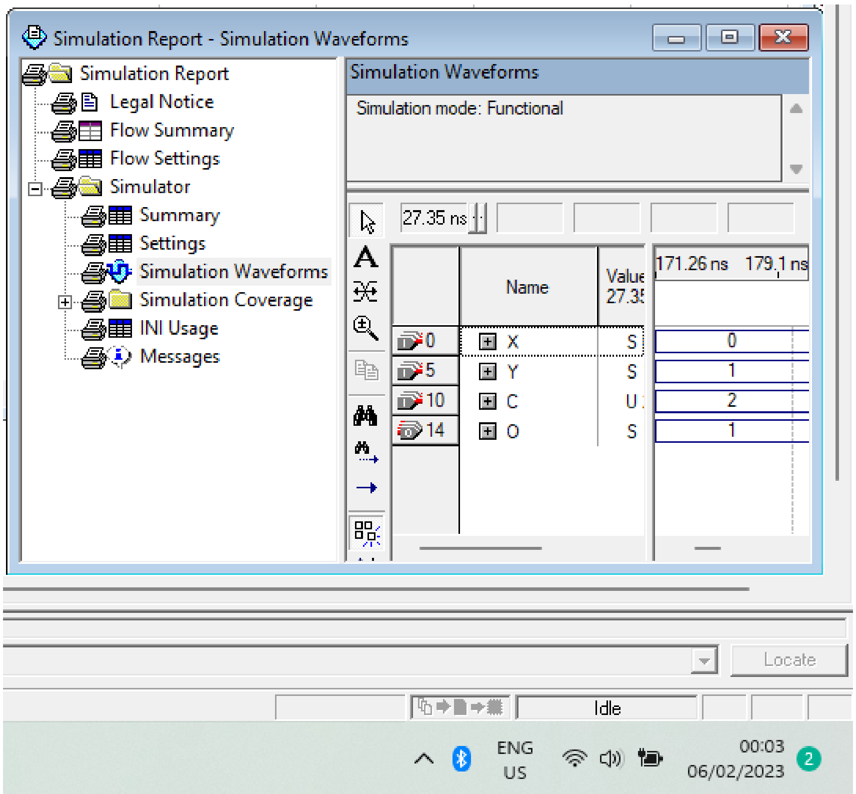
Simulation:-This is the answer for question (g), test by test (depending on my ID: 1210478)-

Test	X	Y	C	O
1	X1 = 1	Y1 = 7	C1 = 4	X NAND Y
2	X2 = 0	Y2 = 1	C2 = 2	((0)/2)+(1) = 1
3	X3 = -1	Y3 = -7	C3 = 2	((-1)/2)+(-7) = -7

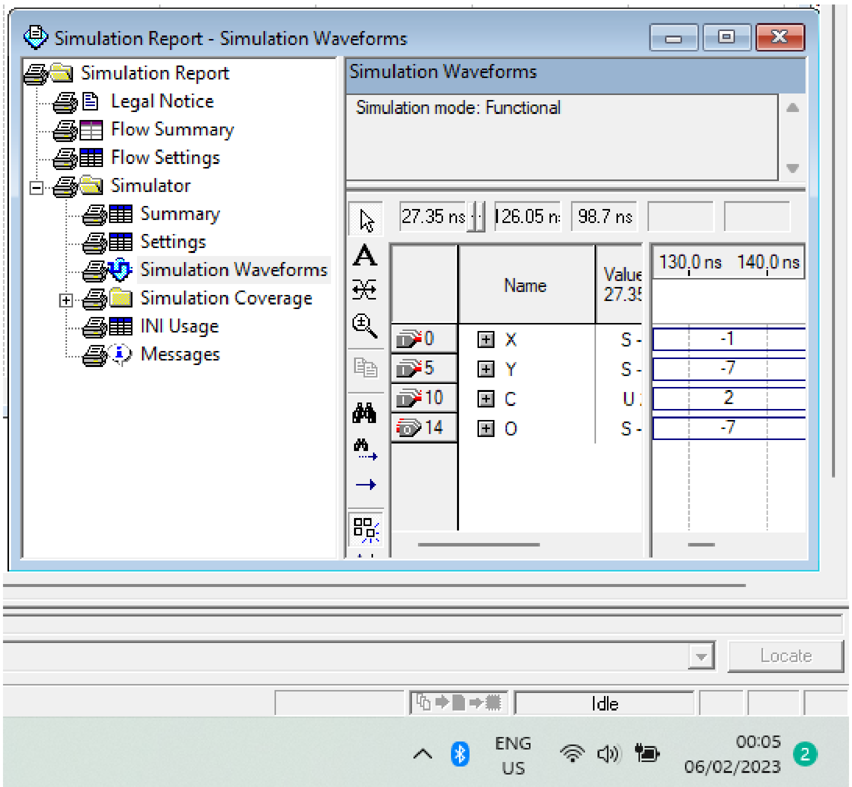
First test



Second test



Third test



Thank you ...