

BIRZEIT UNIVERSITY

ENCS2340

Project Report

1st semester 22/23

Instructor: Dr. Bilal Karaki Prepared by: Diaa Badaha ID: 1210478

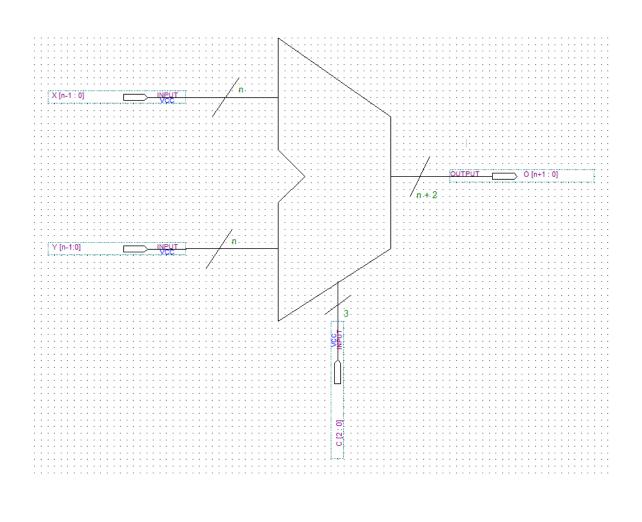
Table of contents

Introduction	2
Specifying output's bits	4
ALU Implementation	6
Used components	9
Full Adder	.9
Full Subtractor	.11
Shifters (Divider & Multiplier)	.12
Bitwise NAND	.15
Bitwise NOT	.16
Bitwise NOR	.17
Bitwise XOR	.18
MUX 8-1	19
ALU (Structural)	.21
Code	.21
Tests	.22
ALU (Behavioral)	.24
Code	.24
Tests	25

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	X[n-1:0] Y[n-1:0]
001	2*(X+Y)	
010	(X/2)+Y	<u>√</u> n <u>√</u> n
011	X-(Y/2)	C[2:0]
100	X NAND Y	ALU
101	NOT(X)	3/
110	X NOR Y	¥
111	X XOR Y	0

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	Χ	Y	С	0
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	$X_3 = -X_1$	$Y_3 = -Y_1$	$\mathbf{C}_3 = \mathbf{C}_2$	((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.

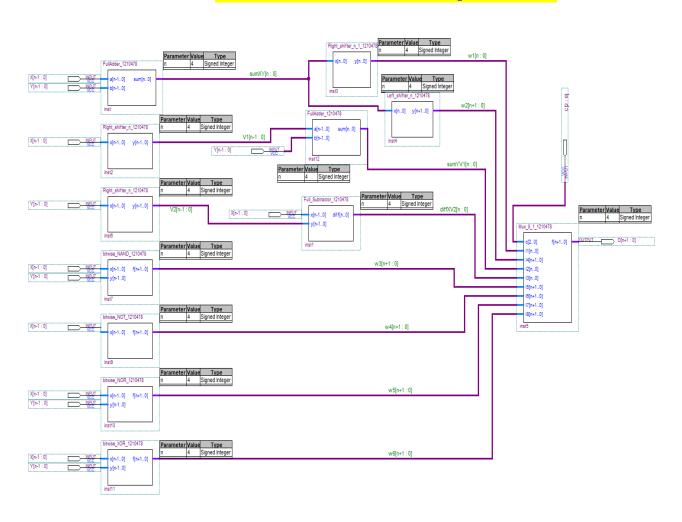
• 2 * (X + Y):

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.

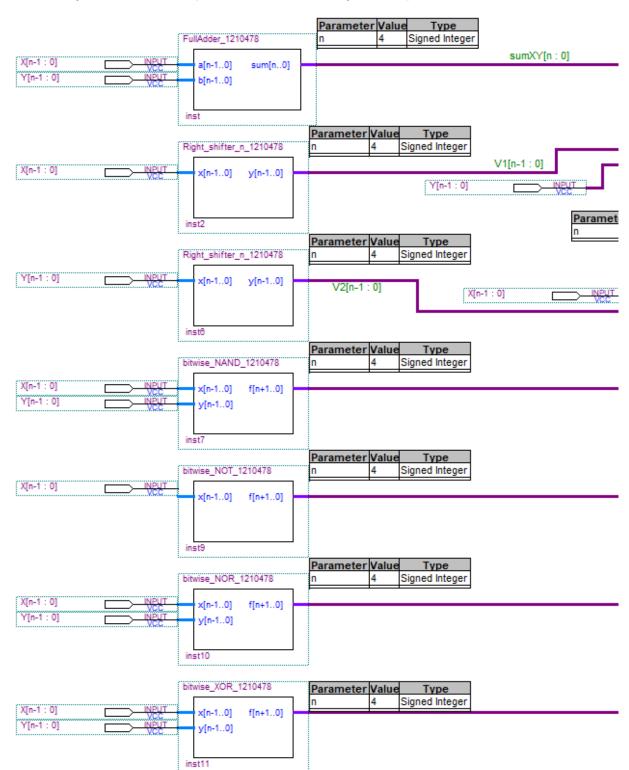
Summary :

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		\checkmark	n
NOT(X)		\checkmark	n
X NOR Y		\checkmark	n
X XOR Y		\checkmark	n

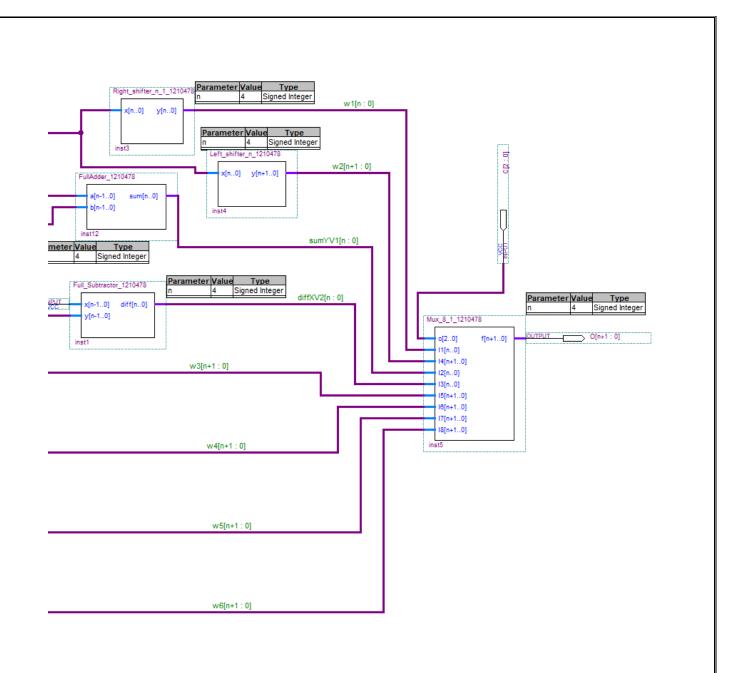
b) Show the ALU implementation using medium-scale integration (MSI) components and minimum number of gates (i.e. in blocks with their sizes). Note that, you might use some kind of extension (sign- or zero-extension).



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



ALU implementation(divided into two photos):



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).

Fu	llAdder_121	0478	
	a[n-10]	sum[n0]	
	b[n-10]		
	-4]
in	st		

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
```

		laveforms de: Functional														
L3	Master T	ime Bar:	110.0	ns	•	 Pointe 	r:	90.3	15 ns		Interva	l:	-1	9.65 ns		Start:
A ⊛		Name	Value 110.() ps	10.0 ns	20.0 r	ns 30).0 ns	40.0	ns (50.0 ns	60.0) ns	70.0 ns	80.0 n	s 90
€ ¶a	₽ 0 ₽5	∃∎a ≣b	S - S -	1 -3		2 X	3 -1	X 4 X 0	=X	5 1	X C		7	<u>X -8</u> X 4	_X	-7 5
ÅÅ.	a 10	🛨 sum	S	-2	X	<u>0 X</u>	2	X 4	≡x	6		<u> </u>	-6	_X -4		-2
<u>₩</u> , →																
院 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).

Fu	II_Subtracto	or_1210478	
	x[n-10] y[n-10]	diff[n0]	
in	st1		1

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

Simulation:

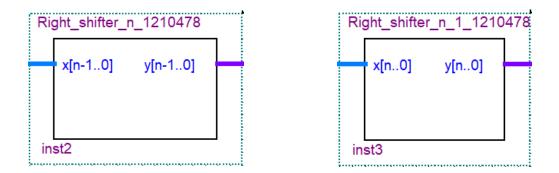
Master	Time Bar: 27	.25 ns	Pointer:	9.37 ns Interva	t -17.88 ns S	tart:	End:
		Value	0 ps	10.0 ns	20.0 ns		30.0 ns
	Name	27.25				27.25	ns
i ₽0	Ξx	S	0	X	1 X	2	<u>Х</u> 3
i ≱4	⊡ y	S	-2	X	-1 X	0	χ 1
@ 8	🗉 diff	S:			2		

11 | Page

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

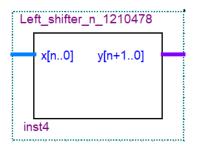
Simulation Waveforms Simulation mode: Functional

ß	Master T	ime Bar:	11	0.0 ns		↓ ►	ointer:		128.99 ns		Inte	erval:		1	8.99 ns		S	tart:			
A ⊛		Name	Va 11	0 ps	10.0 ns	20).0 ns	30.0 ns	40. <u>0</u>	ns	50.0 n	s	60.0 ns		70.0 ns		80.0 ns		90.0 ns	100,0) ns 110 110
Ð	₩0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Ξ× Ξy	:	-3 -2	\neq	-2	χ <u>-1</u> -1	X	0 X 0	1	X	2		3	X	4	2	5	X	6 X	7
ф М																					
→ 閉::																					
ĝ↓																					

13 | Page

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).



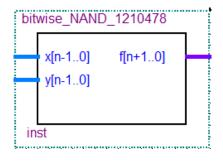
Code: -from the answer of question (c)-

```
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
```

			forms Functional	1																											
4	Master	Time	Bar:	210	0.0 n:	s		• •	Poin	er:		70	8.12	ns		lr	nterva	al:		-131	.88 ns		s	tart:					End	:	_
A ⊛			Name	Value 210.0)S	10.0 r	าร	20.) ns	30.	.0 ns		40.0 r	ıs	5(0.0 ns	•	60.0 n	IS	70.0	ns	80.0	ns	90.0 n	IS	100 _, 0	ns	110 _, 0 n	s	120
	₽ 0 ₽ 4	Ŧ		S 2 S 4		-3 -6	X	-2 -4		(1 (2		Æ	0	₹	1		X	2	¥	3	X	-4 -8		-3 -6		-2 -4		-1 -2	¥	0	
# <u></u> , →																															
踪 ≹↓																															

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);
```

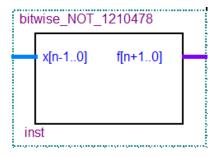
```
input [n-1 : 0]x, y; //declaring inputs
output reg [n+1 : 0]f; //declaring output
always @ (x, y)
    begin
    f = ~(x & y); //bitwise nand operation
```

```
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
```

Simu	ulation W	/aveforms												
Simu	ulation mo	de: Functional												
ß	Master T	ime Bar:	27.3	95 ns	Pointer:	158.4	15 ns	Interval:	131.1	ns	Start:			End:
A		Name	Value	110.0 ns	130 _, 0 ns	3	150 _, 0 ns	17	0, <mark>0 ns</mark>	190	0 ns	210	0 ns	230
€ €		Name	27.35											
<u>`</u>	₽ 0	Ξ×	B OC	(1011)	<u>1100 X</u>	1101 X 111	Ο χ 1111	X 0000	X 0001)	(0010)	0011	<u>(</u> 0100)	0101	<u>(</u> 0110
Ēð	₫\$5	Шy	B 11	(0101)	0110	0111 / 100	0 1001	1010	<u>× 1011</u>	(1100)	(1101)	(1110)	(1111)	0000
桷	ii) 10	⊞ f	B 111	(111110)	<u>111011 X 1</u>	11010 X 0001	<u>11 X 000110</u>) <u>X 111111</u>	<u>× 111110</u>	(111111))	(111110)	(<u>111011</u>)	(111010	<u>(111111)</u>
<u>8</u>														
咒														
₽↓														
- '		I												

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.



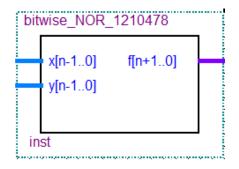
Code: -from the answer of question (c)-

```
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
```

		aveforms de: Functional												
4	Master T	ime Bar:	27.2	25 ns	 Image: Point 	ter:	120.81 ns		Interval:	93.56 n	\$	Start:		
A ⊛		Name	Value 27.25	110.0 ns	130,	0 ns	150	0 ns	170),0 ns	190,	0 ns	210	0 ns
€ ħ	₽ 0 ₽ 5	+ x + f	B 01 B 111	(<u>1110</u>) (<u>000001</u>)	1111) 000000)	0000	(0001) (111110)	X 0010 X 111101	X 0011 X 111100	X <u>0100</u> X X 111011	0101 >	0110	(0111) (111000)	(1000 (000111
#4 **_														
→ 5														
₿↓														

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.



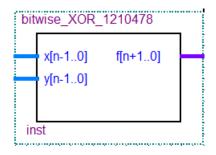
Code: -from the answer of question (c)-

```
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
```

		aveforms										
Sim	ulation mod	de: Functional										
R	Master T	ime Bar:	130	0 ns	I ► Pointer:	4.41 ns	Inte	rval:	-125.59 ns	Start:		End:
Α			Value	0 ps 10.0) ns 20.0 ns	30.0 ns 40.	0 ns 50.0 n	s 60.0 ns	70.0 ns 8	0.0 ns 90.0) ns 100,0 ns 1	110,0 ns 120,
€		Name	130.0				· · ·					
Ð,	₽ 0	±х	B 11	0000	<u>0001 X 00</u>	10 X 0011 X	0100	0101 <u>X</u> 01	10 <u>×</u> 0111	<u>x 1000 x</u>	1001 <u>× 1010</u>	<u>(1011</u>)
B	₫\$5	±γ	B 01	(1001)	(1010 / 10		(1101)	<u>1110 X 11</u>		<u>x 0001 x</u>	0010 X 0011	<u> (0100)</u>
<i>6</i> 4	ig 10	⊞ f	B 000	000110	000100	<u> </u>	<u>000010 X</u>	000000	X 111000	<u>X 000110 X</u>	000100	<u> 000000</u>
M.,												
-												
暻												
₽↓												

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.



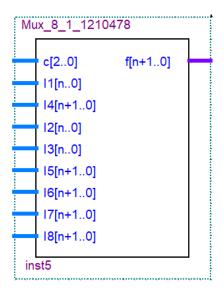
Code: -from the answer of question (c)-

```
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
```

		/aveforms de: Functional												
4	Master T	ime Bar:	440.	Ons 🕠	Pointer:	355.74 ns		nterval:	-84.26	ns	Start:			End:
A ⊛		Name	Value 440.(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 90	.0 ns 100	0 ns 1	10 _, 0 ns 120
e Pa	₽ 0 ₽ 5	±× ±y	B 11 B 00	(<u>0000 X 000</u> (<u>0101 X 011</u>		X 0011	0100	0101	0110 1011	0111 X	1000 1101	X 1001 X 1110	(1010)	<u>× 1011</u> × 0000 >
纳	ing 10	∎f	B 111	000101 0001	11 X 00010	<u> </u>	(111101)	(111111)	<u>111101</u>	111011 X	000101	<u>x 000111</u>	000101	<u>× 111011</u>
^ <u>∧</u>														
₽ <u>;</u> 2↓														

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 = 15;
        else if (c == 'b101)
            f = I6;
        else if (c == 'b110)
           f = I7;
        else if (c == 'b111)
            f = 18;
    end
endmodule
```

Simulation:

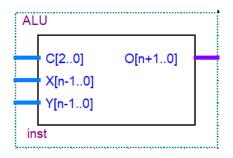
Simulation Waveforms

Simulation mode: Functional

ß	Master Time Bar:		90.0 ns		• • Pointer	: 11.5	ns Interv	Interval:	
A ⊛ €		Name	Va 9(0 ps	20.0 ns	40.0 ns	60.0 ns	80.	
~	₽ 0	∃ 1	B	(00000)	00001 (00010)	00011 00100	<u>X 00101 X 00110</u>	00111	
Þ	€	± 12	B	00000	00001 00010	00011 00100	<u>X 00101 X 00110</u>	00111	
Ņ	12	± 13	B	00000	00001 (00010)	00011 00100	<u>X 00101 X 00110</u>	00111	
	18	± 14	BC	(000000)	000001 000010	(000011 <u>X</u> 000100	<u>X 000101 X 000110</u>	(000111)	
*	₽ 25	± 15	BC	000000	000001 000010	000011 000100	<u> x 000101 x 000110 x</u>	000111	
•	i]¥32	± 16	BC	000000	000001 000010	000011 000100	<u>× 000101 × 000110</u>	(000111)	
账	3 9	± 17	BC	(000000)	000001 000010	000011 000100	<u>X 000101 X 000110</u>	000111	
	∰ 46	± 18	BC	(000000)	000001 000010	000011 000100	<u>X 000101 X 000110</u>	000111	
ļ	5 3	± c	E	000	001 (010)	011 100	<u>× 101 × 110</u>	(111)	
	57	± f	BC	(000000)	000001 000010)	000011 000100	X 000101 X 000110	000111	

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]0;
                                                 //declaring output: 0;
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_l_1210478 (sumXY, wl);//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 1C₂Y₂X₂C₁Y₁X₁, so, if your student ID is 1220520, then X, Y, and C values for the three test cases as follows:

Test	Χ	Y	С	0
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 {C₂, Y₂, X₂, C₁, Y₁, X₁} is 8 or 9, you need to replace it by 1

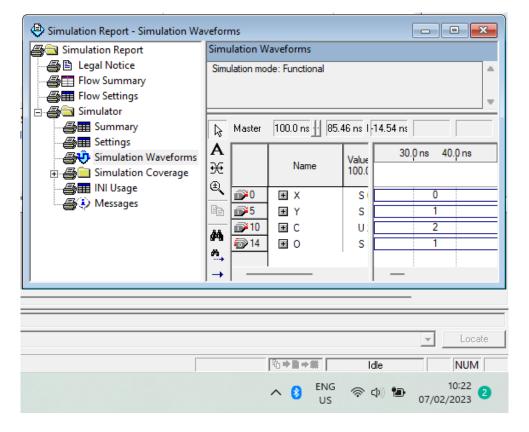
Simulation:-This is the answer for question (e), test by test (depending on my ID: 1210478)-

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

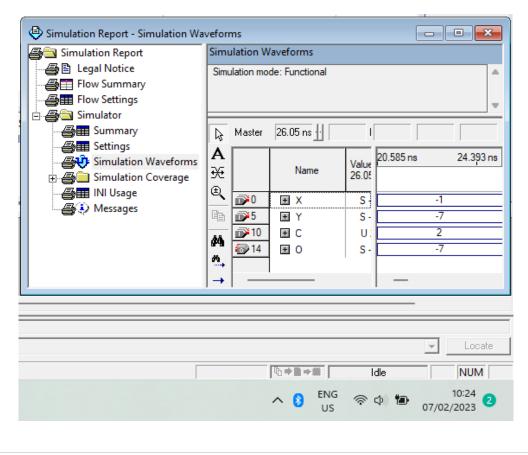
First test

Simulation Report - Simulation Wa	veforms
Simulation Report Legal Notice Flow Summary Simulator Simulator Summary Settings Summary Settings Settings Simulation Waveforms	Simulation Waveforms Simulation mode: Functional Master 90.0 ns 1 605 ps 1 -89.4 ns A Value 0 ps 10.0 ns 20.0 ns
Simulation Coverage	Mame Value 90.0 ● ● ● ● ● ● ● ● Name 90.0 ● ● ● Name 90.0 ● ● Name 90.0 ● ● ● Name 90.0 ● <td< td=""></td<>
	Locate Locate
	22 P

Second test



Third test



23 | Page

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]0; //declaring output: 0;
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
             0 = 0;
    end
endmodule
```

g) Generate the waveforms of the behavioral ALU defined in Part (f), assumes that X and Y are 4bits and their values based on your student ID should be set as follows:

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:

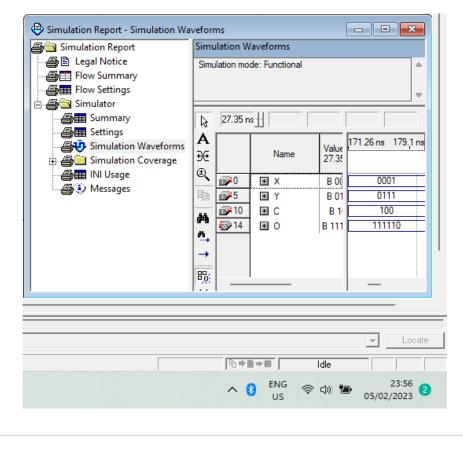
Test	X	Y	С	0
1	$X_1 = 0$	$Y_1 = 2$	$C_1 = 5$	NOT(0)
2	$X_2 = 0$	$Y_2 = 2$	$C_2 = 2$	((0)/2)+(2)
3	$X_3 = -X_1$	$Y_3 = -Y_1$	$C_3 = C_2$	((0) /2)+(-2)

Note: If any value from the set {C₂, Y₂, X₂, C₁, Y₁, X₁} is 8 or 9, you need to replace it by 1

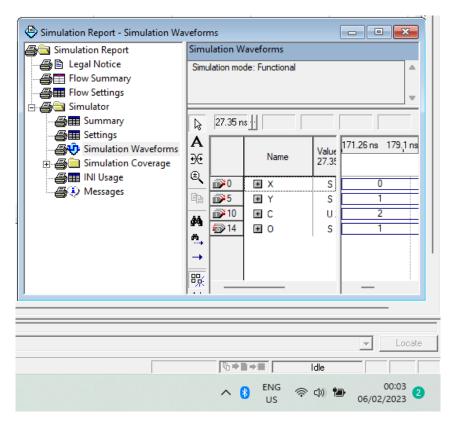
Simulation:-This is the answer for question (g), test by test (depending on my ID: 1210478)-

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

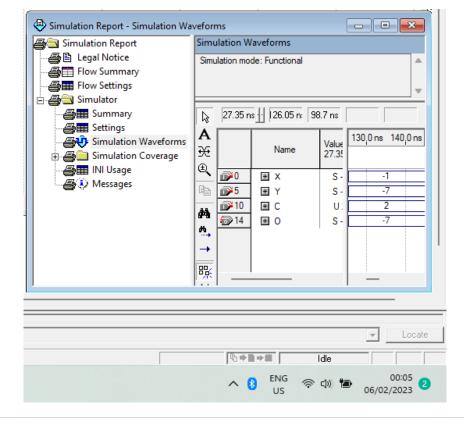
First test



Second test



Third test



Thank you ...