Digital Systems ENCS234

2022/2033

ALU Verilog Project

Mujahed Abuali #1211047

Dr.Ismail Khater sec.1

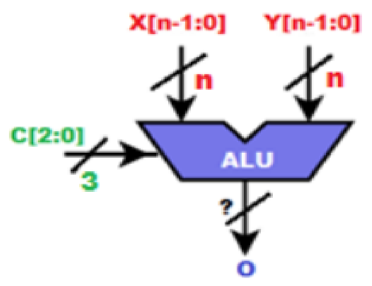
Problem: Modeling a Multifunction ALU

Design and implement a multifunction arithmetic and logic unit (ALU)

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.[1]

Implementation:

ALU Function Code (C)	ALU Output (O)	ALU Symbol
000	(**X**+**Y**)/2	 X[n-1:0] Y[n-1:0] n n C[2:0] 3 ALU ? O
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**	

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

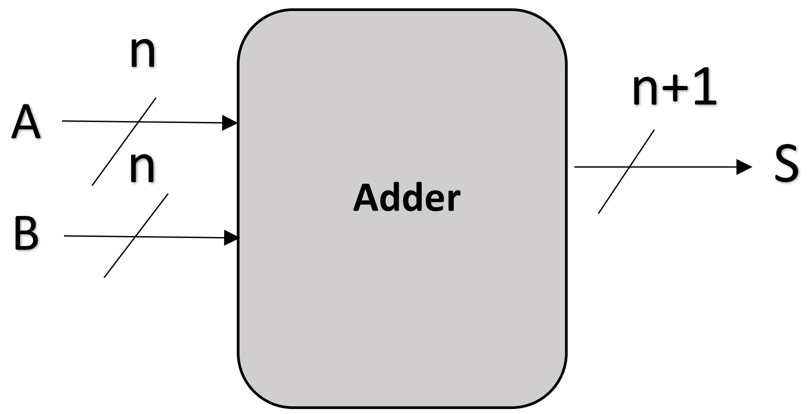
- The size in the first option **(X+Y)/2,** it's had sum process, so You have to increase bits by one bit. However, the division process does not lead to an increase in digits. so, it's needed **n+1 bits.**
- The size in the Second option **2*(X+Y),** it's had sum and Multiplication process, so You have to increase bits by one bit to sum process and more one for Multiplication process.so, it's needed **n+2 bits.**
- The size in the third option **(X/2) +Y,** it's had sum process, so You have to increase bits by one bit. However, the division process does not lead to an increase in digits. so, it's needed **n+1 bits.**
- The size in the first option **X-(Y/2),** it's had subtracted process, so You have to increase bits by one bit. However, the division process does not lead to an increase in digits. so, it's needed **n+1 bits.**
- The size for **(X NAND Y), (NOT(x)), (X NOR Y), (X XOR Y),** No change occurs to it. **n** it's still **n.**

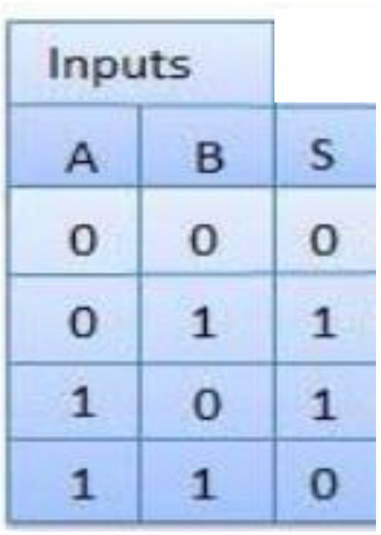
  So, the overflow can never occur it's (n+2) bits.

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).

- **Adder** for sum number

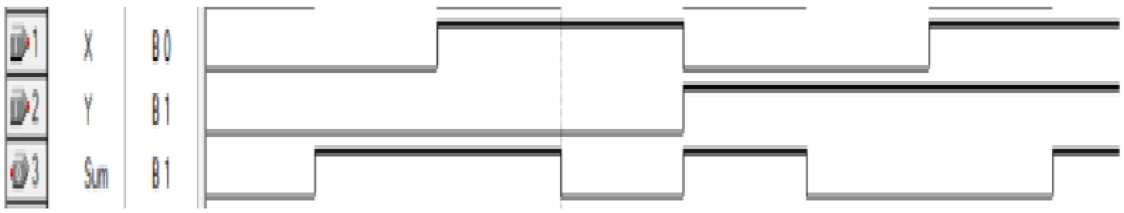
  Adding two numbers using adders. This function outputs one n+1-bit output(sum).





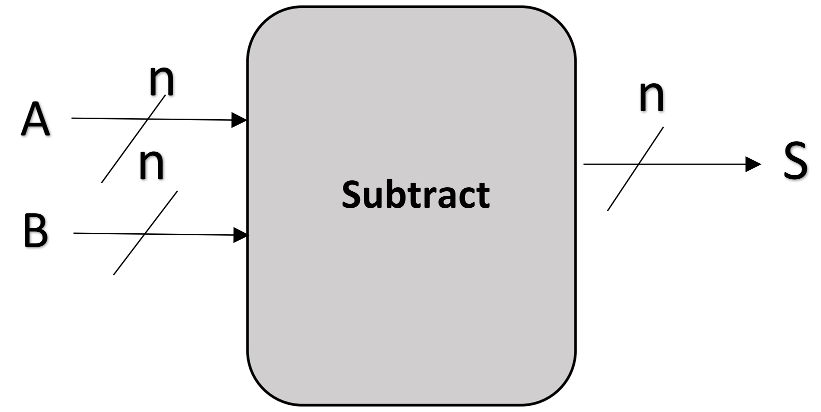
Inputs		
A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

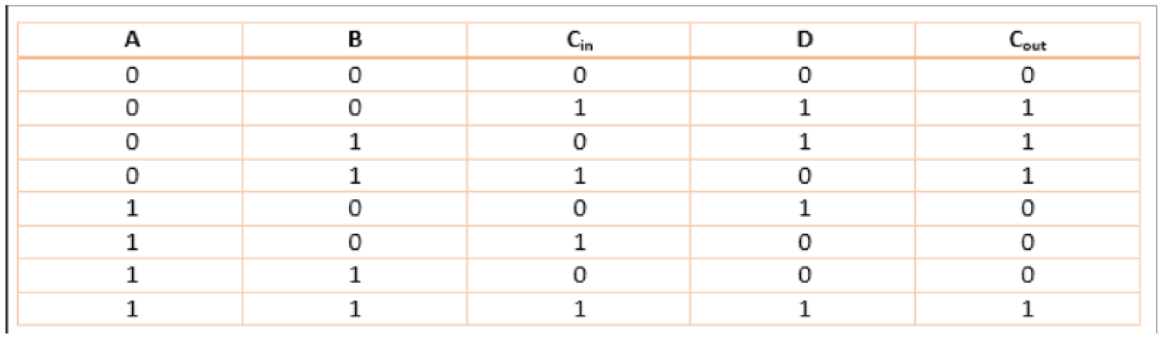
Simulation:



- **<u>Subtract</u>** for subtract number

  Subtracting two binary numbers using adders and xor gates that change the number to its 2's complement. This function outputs one n-bit output (sum) and one outputs.

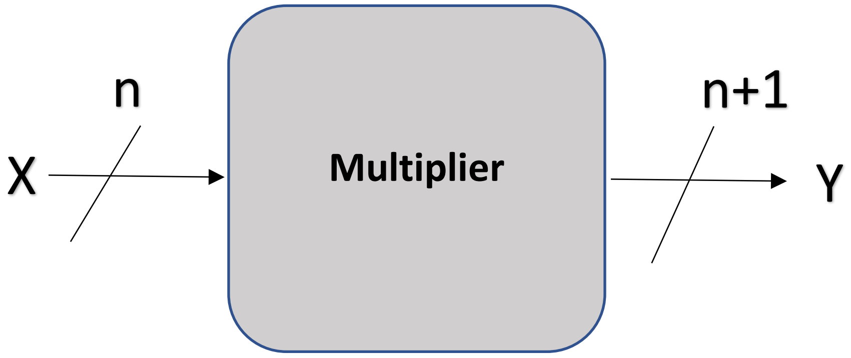


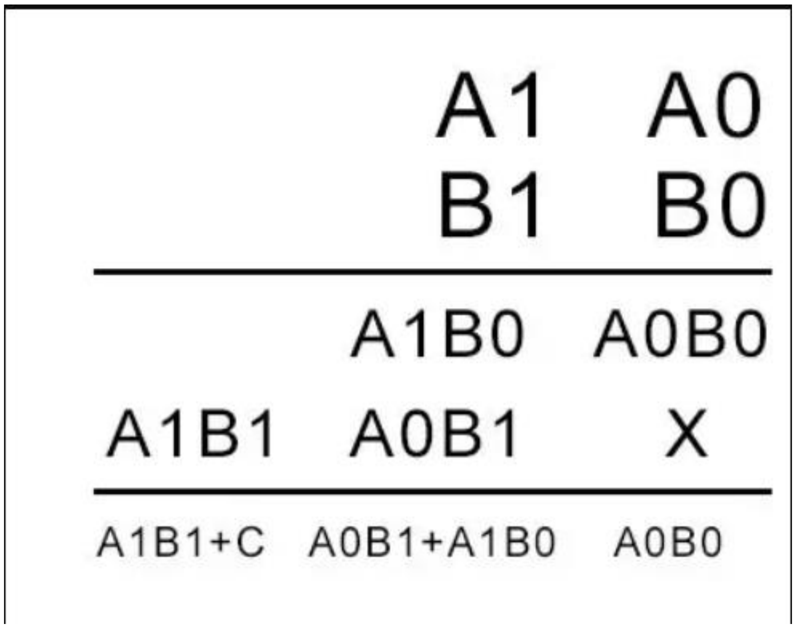


A	B	C_{in}	D	C_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

- **<u>Multiplier</u>** number by 2

A multiplier is a combinational logic circuit that we use to multiply binary digits. Just like the adder and the subtractor, a multiplier is an arithmetic combinational logic circuit. It is also known as a binary multiplier or a digital multiplier.

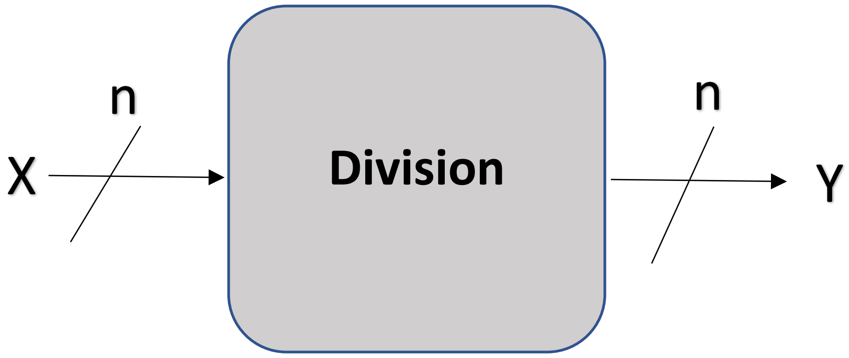




	A1	A0
	B1	B0
	A1B0	A0B0
A1B1	A0B1	X
A1B1+C	A0B1+A1B0	A0B0

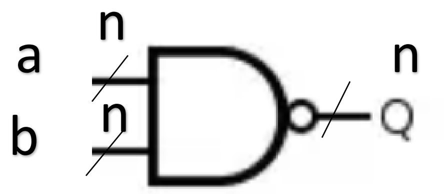
- **<u>Division</u>** number by 2

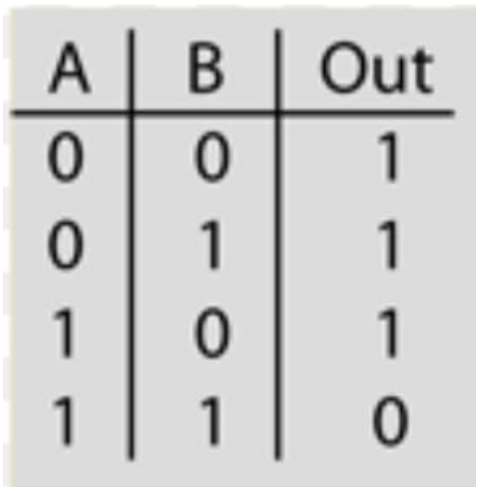
circuit to divide a digital signal by an even integer multiple is a **Johnson counter**. This is a type of shift register network that is clocked by the input signal. The last register's complemented output is fed back to the first register's input.



- NAND

A **NAND gate** ("not AND gate") is a logic gate that produces a low output (0) only if all its inputs are true, and high output (1) otherwise. Hence the NAND gate is the inverse of an AND gate, and its circuit is produced by connecting an AND gate to a NOT gate. Just like an AND gate, a NAND gate may have any number of input probes but only one output probe.

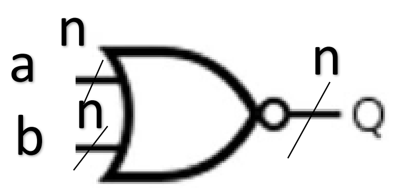


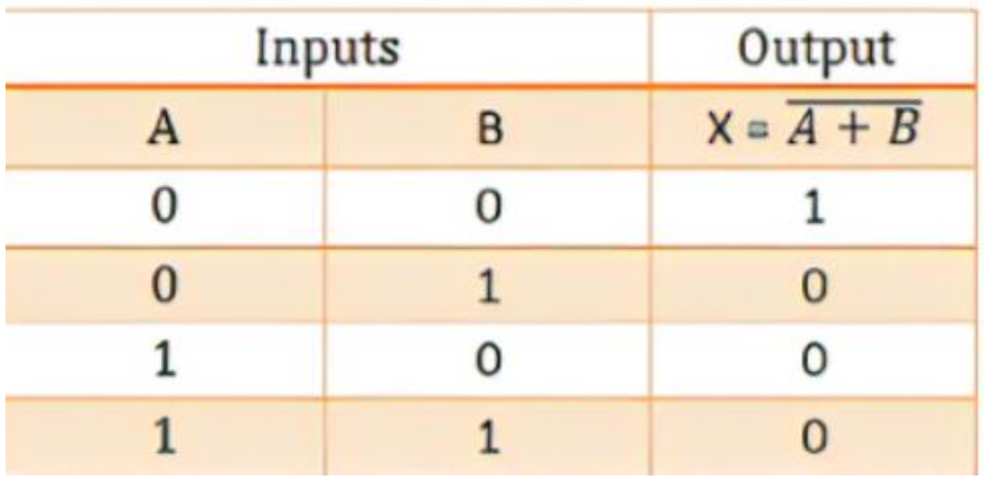


A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

- <u>NOR</u>

A NOR gate ("not OR gate") is a logic gate that produces a high output (1) only if all its inputs are false, and low output (0) otherwise. Hence the NOR gate is the inverse of an <u>OR gate</u>, and its circuit is produced by connecting an OR gate to a NOT gate. Just like an OR gate, a NOR gate may have any number of input probes but only one output probe. **A NOR gate** ("not OR gate") is a logic gate that produces a high output (1) only if all its inputs are false, and low output (0) otherwise. Hence the NOR gate is the inverse of an <u>OR gate</u>, and its circuit is produced by connecting an OR gate to a NOT gate. Just like an OR gate, a NOR gate may have any number of input probes but only one output probe.

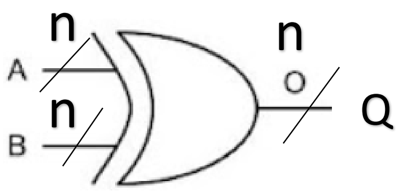


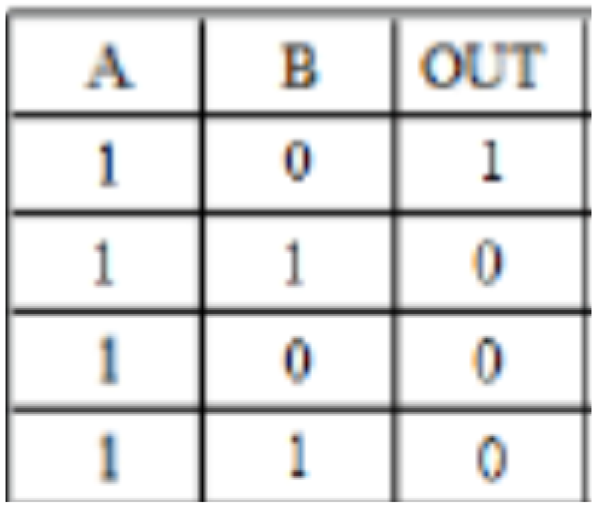


Inputs		Output
A	B	$X = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

- <u>XOR</u>

The **XOR gate** stands for the Exclusive-OR gate. This gate is a special type of gate used in different types of computational **circuits.** The **XOR gate** stands for the Exclusive-OR gate. This gate is a special type of gate used in different types of computational **circuits**.

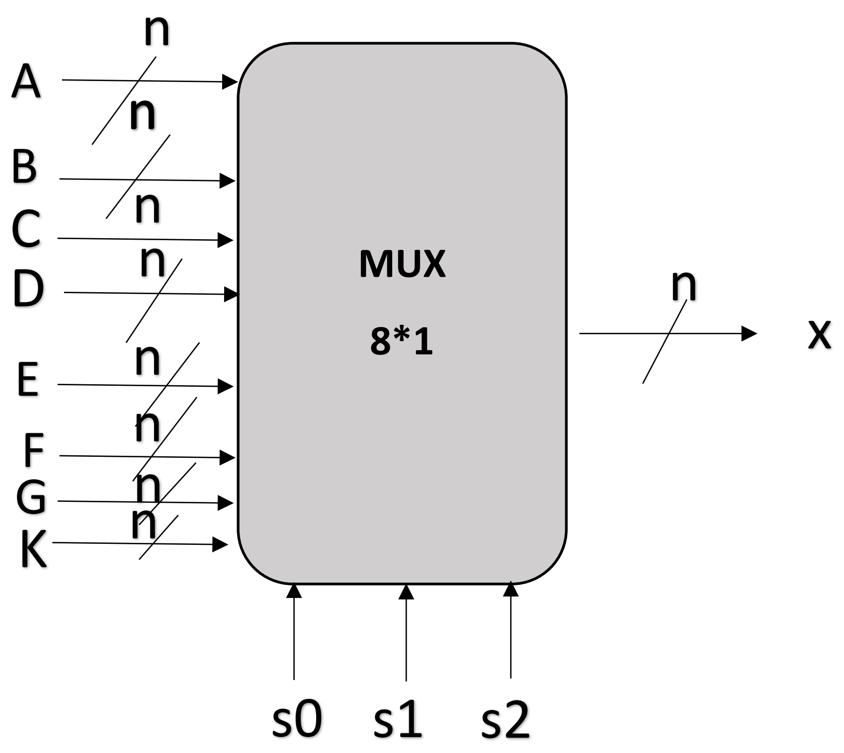




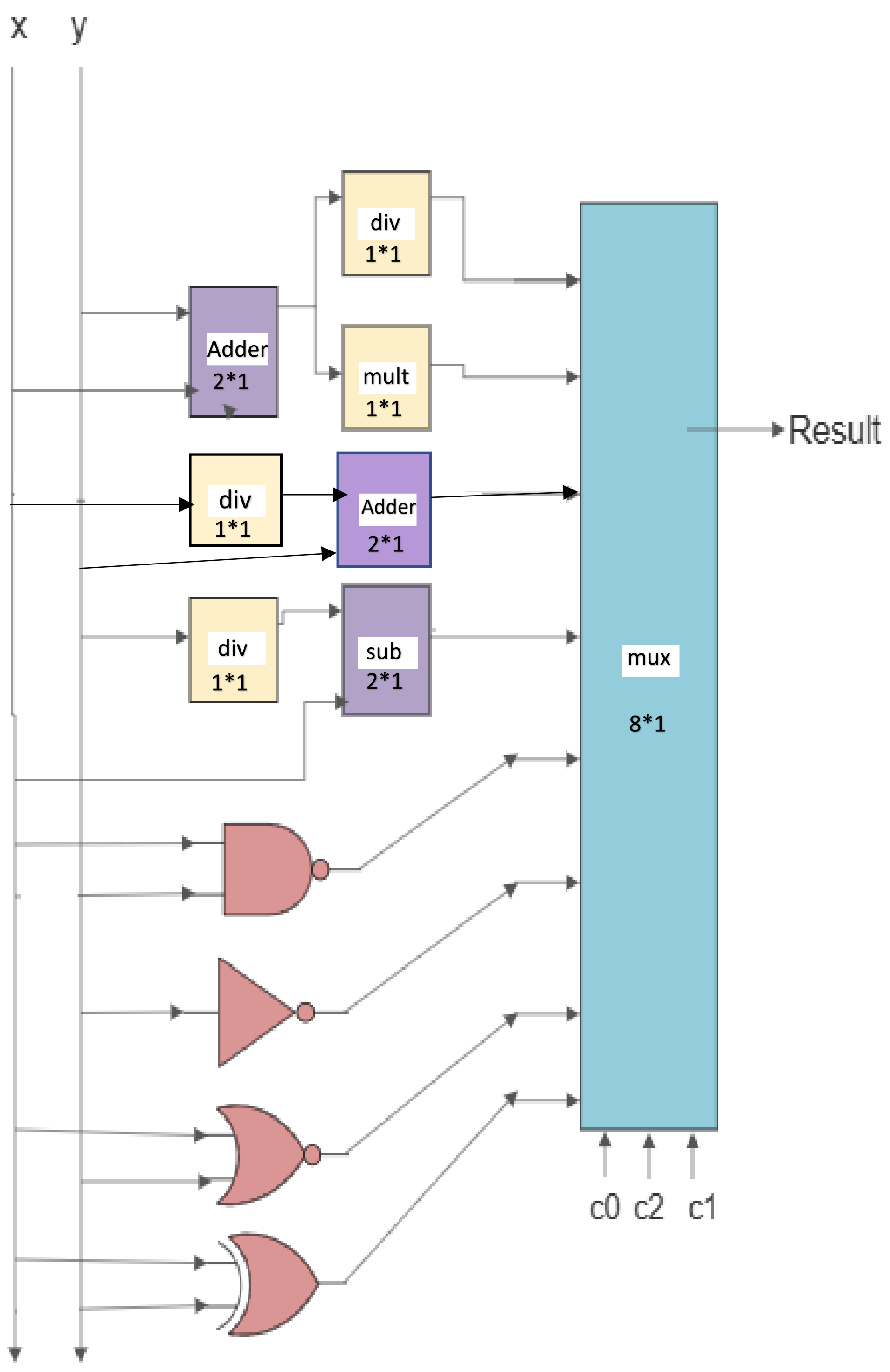
A	B	OUT
1	0	1
1	1	0
1	0	0
1	1	0

- **<u>MUX</u>**

The multiplexer is a combinational logic circuit designed to switch one of several input lines to a single common output line. Has six 8-bit inputs, one 3-bit input, and two 1-bit inputs, and has one 8-bit output. The output has value same as one of the 8-bit or 1-bit inputs depending on the selection input (the 3-bit input).

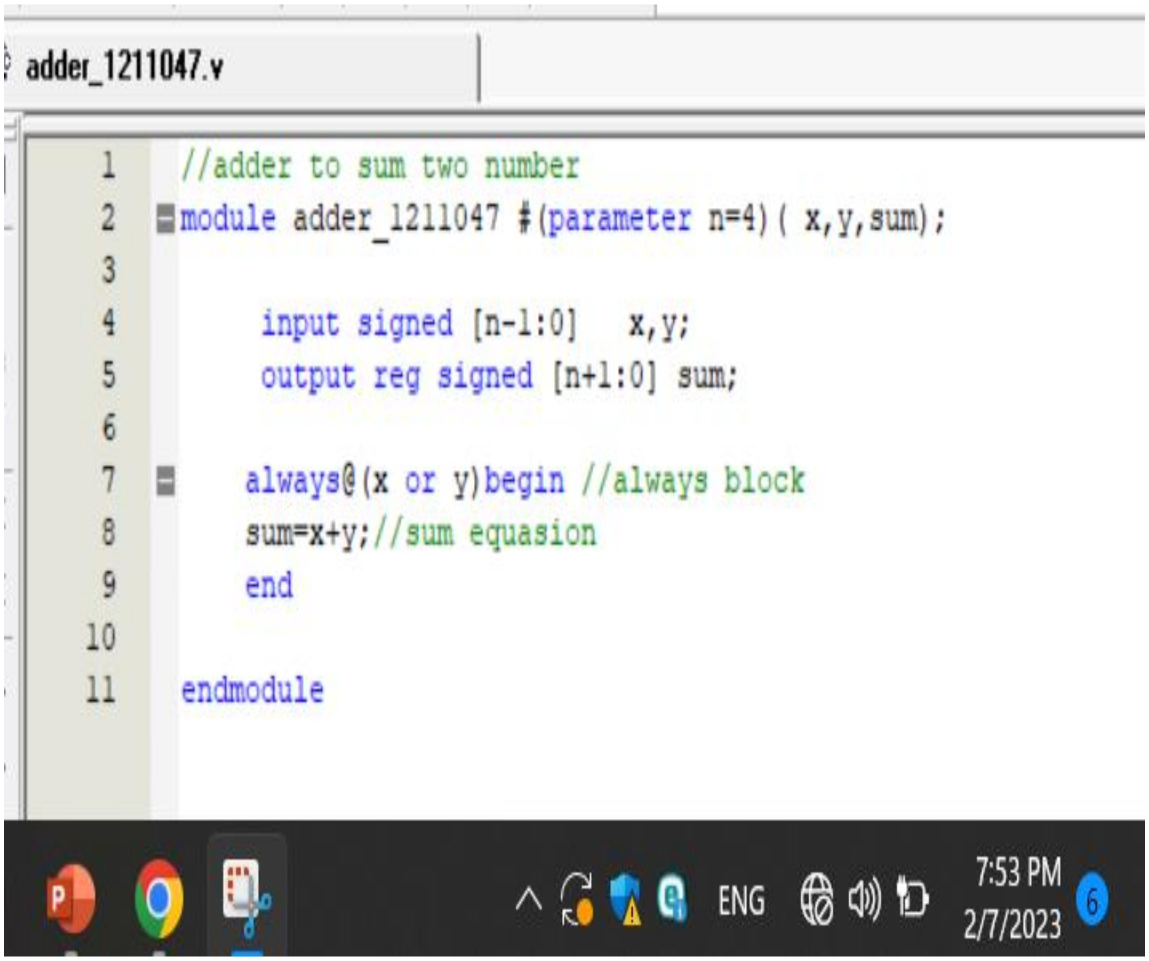


- **Design**


x
y
div
1*1
Adder
2*1
mult
1*1
div
1*1
Adder
2*1
div
1*1
sub
2*1
mux
8*1
Result
c0 c2 c1

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.

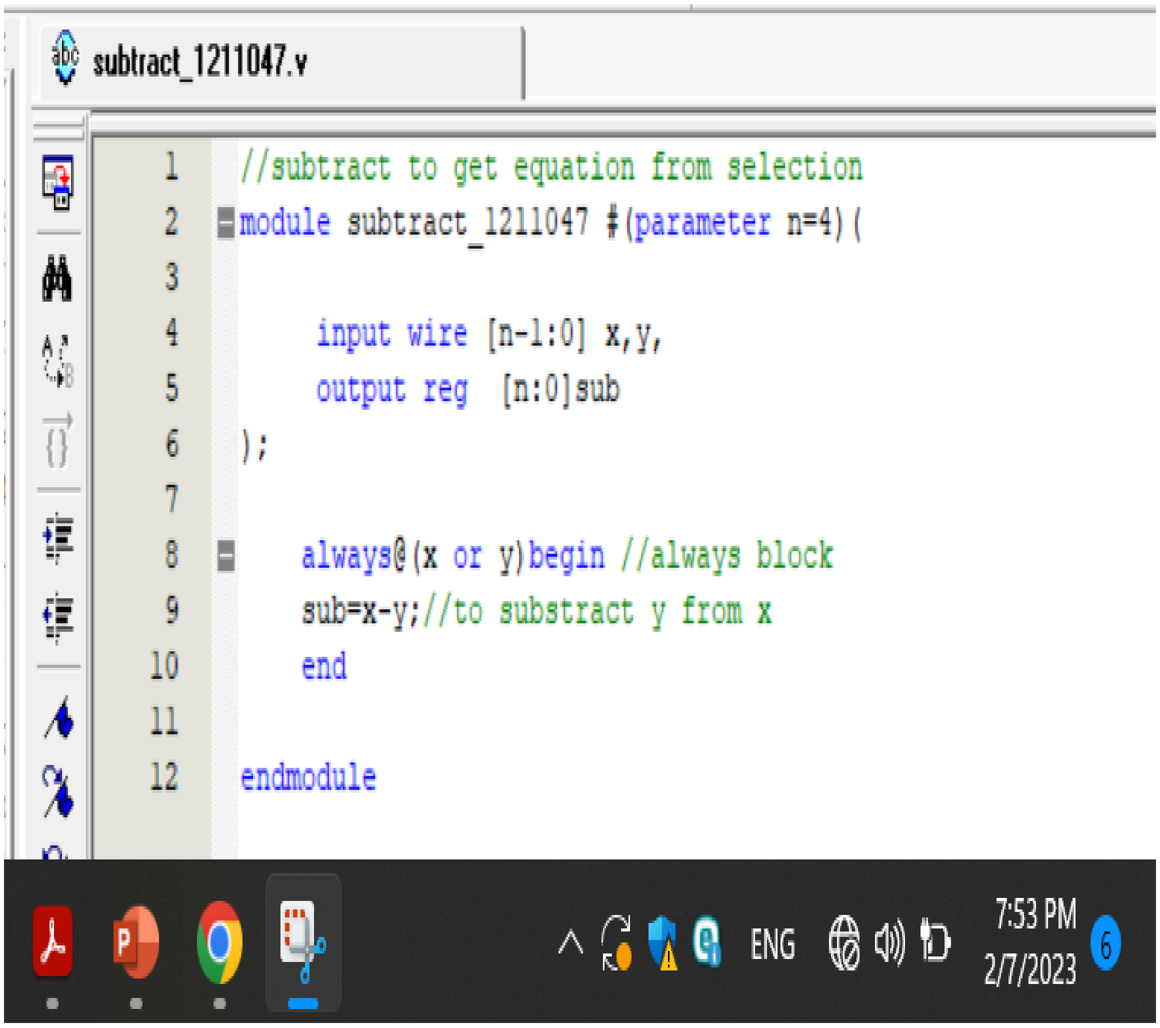
- **<u>Adder</u>**



adder_1211047.v

```
//adder to sum two number
module adder_1211047 #(parameter n=4) ( x,y,sum);

     input signed [n-1:0]   x,y;
     output reg signed [n+1:0] sum;

    always@(x or y)begin //always block
    sum=x+y;//sum equasion
    end

endmodule
```

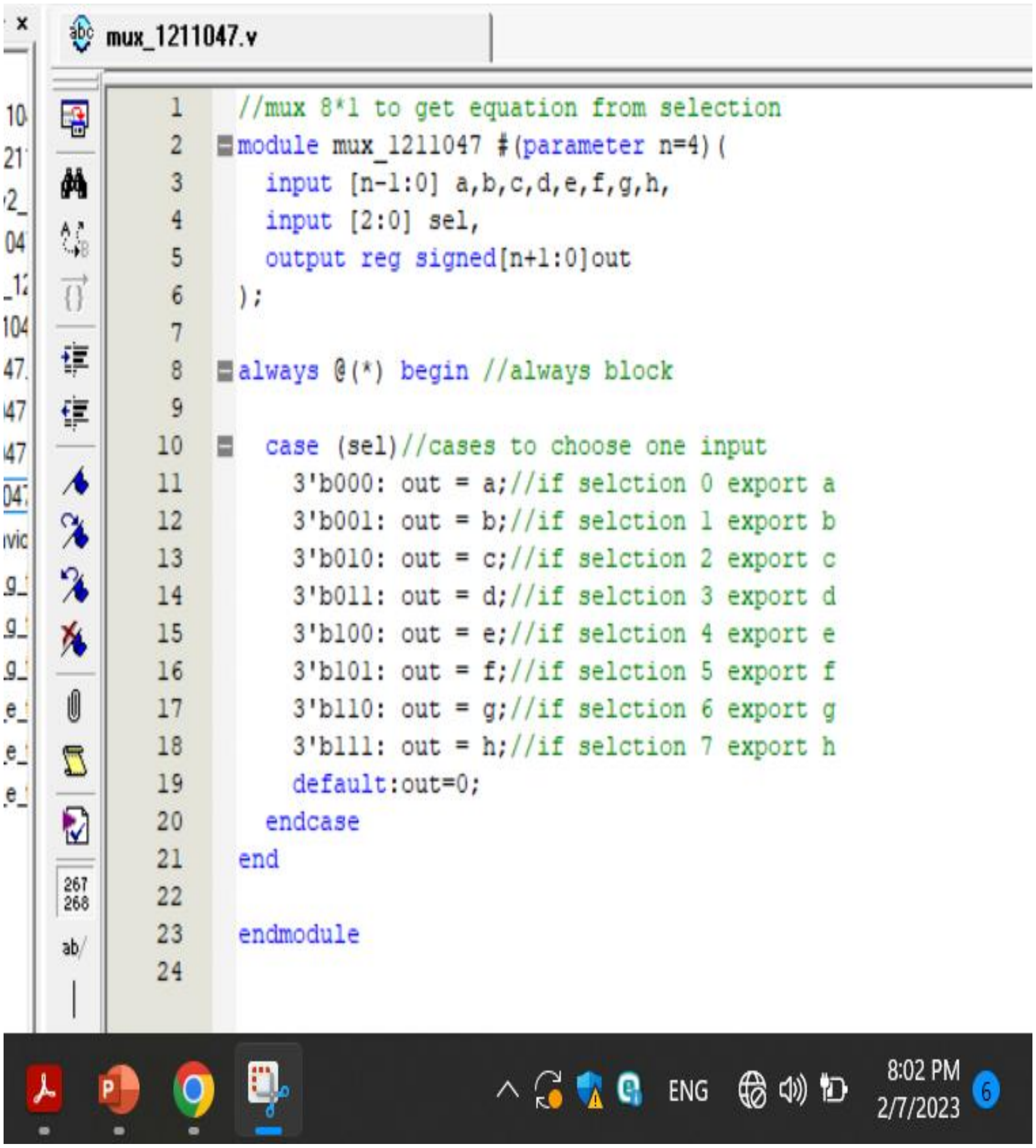
- **Subtract**



subtract_1211047.v

```
//subtract to get equation from selection
module subtract_1211047 #(parameter n=4)(

     input wire [n-1:0] x,y,
     output reg  [n:0]sub
);

    always@(x or y)begin //always block
    sub=x-y;//to substract y from x
    end

endmodule
```

- **Mux**

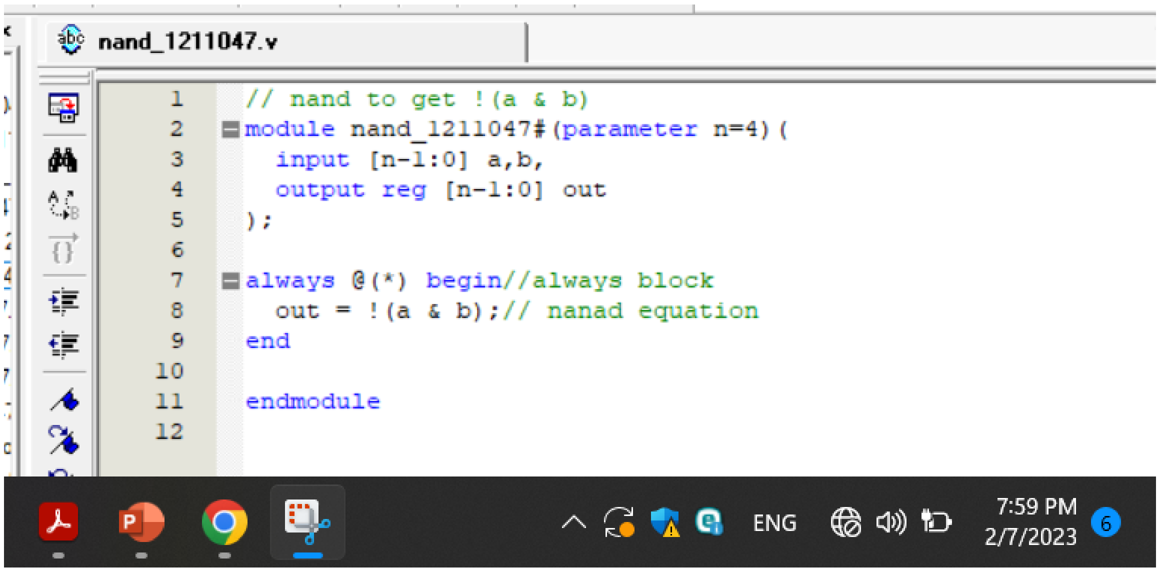


mux_1211047.v

```
//mux 8*1 to get equation from selection
module mux_1211047 #(parameter n=4)(
  input [n-1:0] a,b,c,d,e,f,g,h,
  input [2:0] sel,
  output reg signed[n+1:0]out
);

always @(*) begin //always block

  case (sel)//cases to choose one input
    3'b000: out = a;//if selction 0 export a
    3'b001: out = b;//if selction 1 export b
    3'b010: out = c;//if selction 2 export c
    3'b011: out = d;//if selction 3 export d
    3'b100: out = e;//if selction 4 export e
    3'b101: out = f;//if selction 5 export f
    3'b110: out = g;//if selction 6 export g
    3'b111: out = h;//if selction 7 export h
    default:out=0;
  endcase
end

endmodule
```

-

- **NAND**

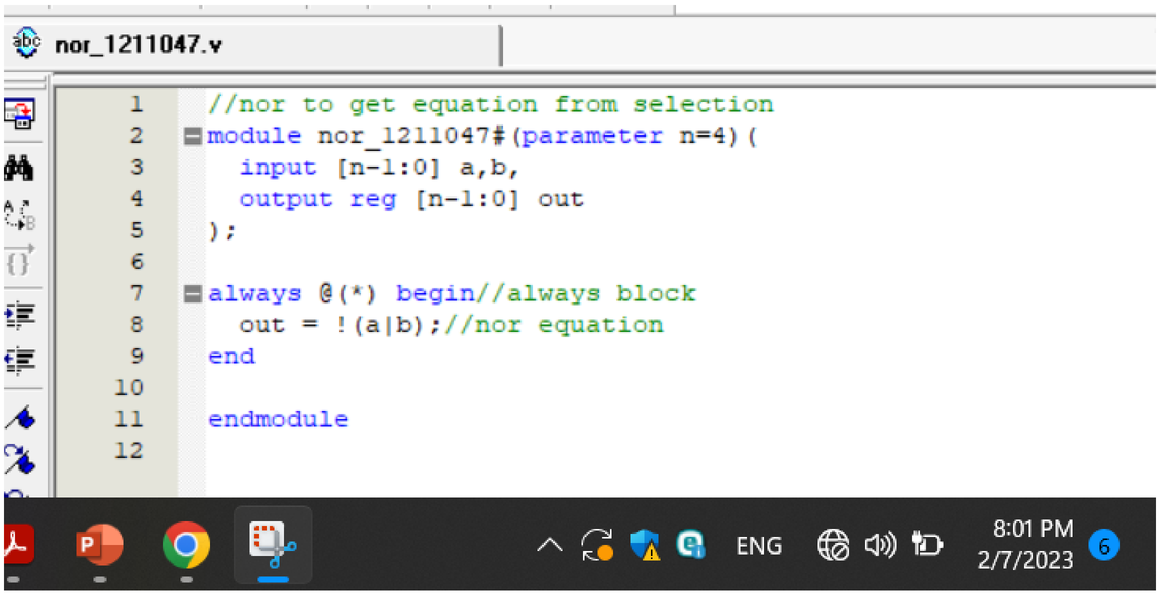


nand_1211047.v

```
// nand to get !(a & b)
module nand_1211047#(parameter n=4)(
  input [n-1:0] a,b,
  output reg [n-1:0] out
);

always @(*) begin//always block
  out = !(a & b);// nanad equation
end

endmodule
```

7:59 PM
2/7/2023

- <u>**NOR**</u>

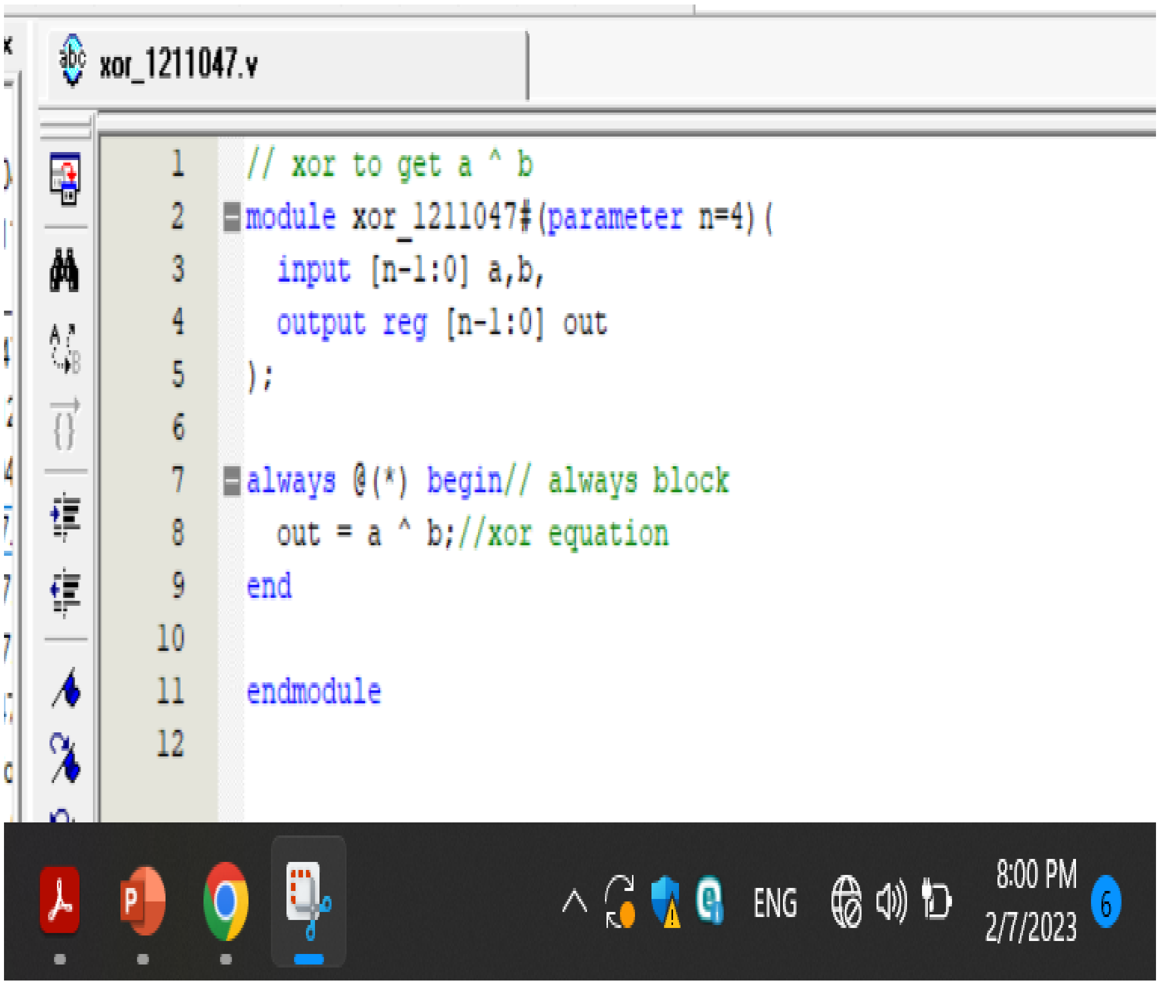


nor_1211047.v

```
//nor to get equation from selection
module nor_1211047#(parameter n=4)(
  input [n-1:0] a,b,
  output reg [n-1:0] out
);

always @(*) begin//always block
  out = !(a|b);//nor equation
end

endmodule
```

8:01 PM
2/7/2023

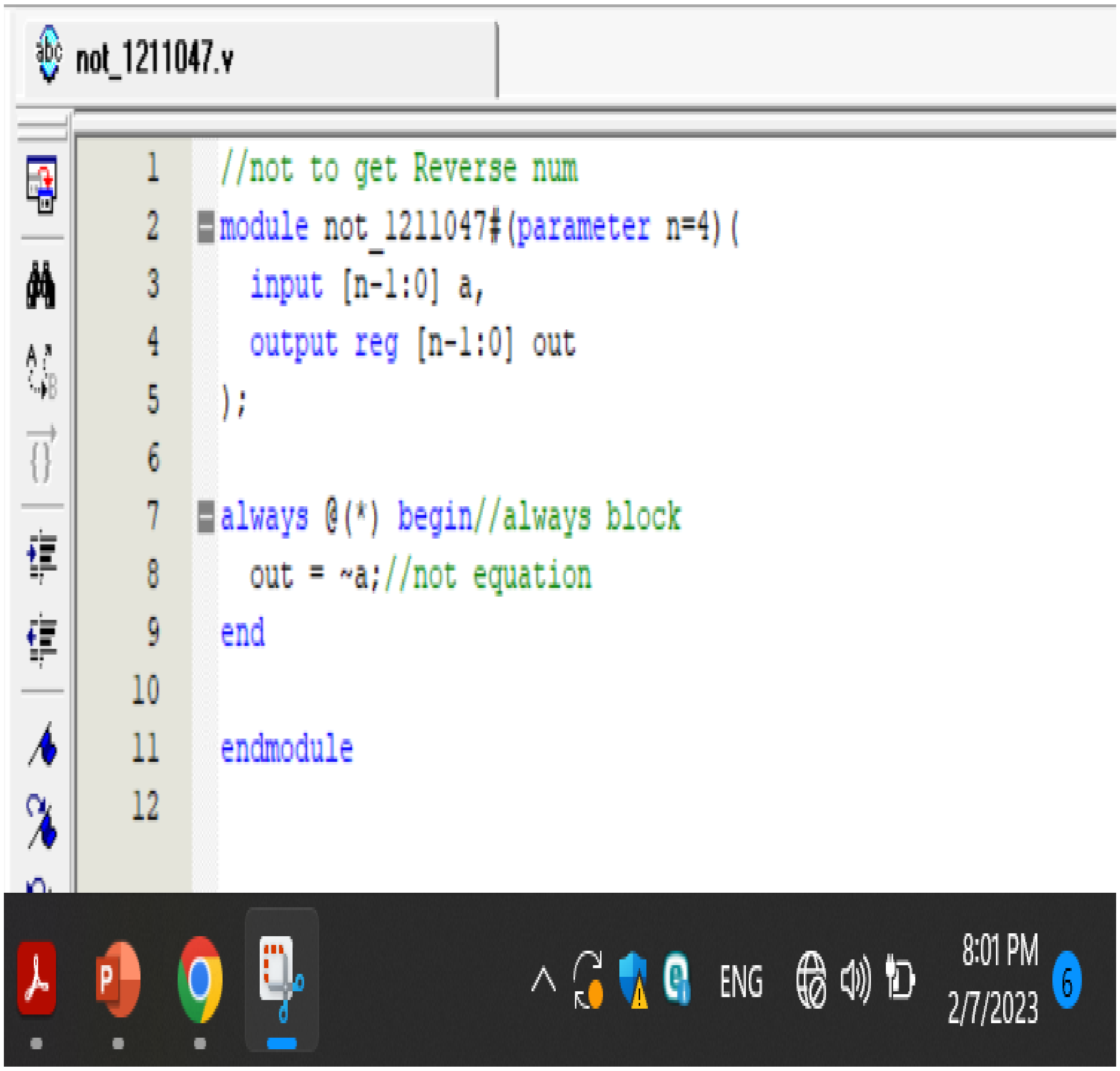
- **XOR**



xor_1211047.v

```
// xor to get a ^ b
module xor_1211047#(parameter n=4)(
  input [n-1:0] a,b,
  output reg [n-1:0] out
);

always @(*) begin// always block
  out = a ^ b;//xor equation
end

endmodule
```

- **NOT**

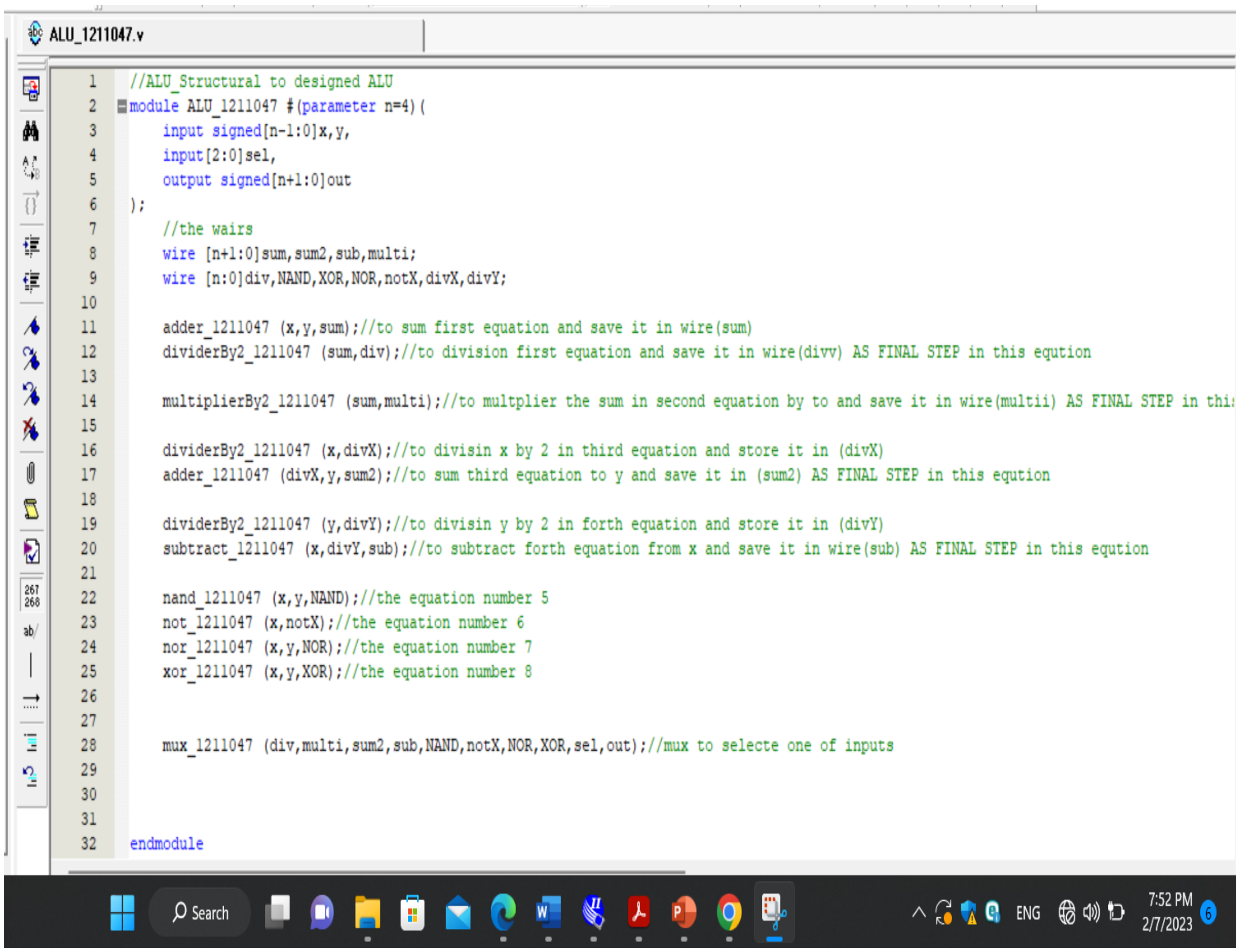


not_1211047.v

```
//not to get Reverse num
module not_1211047#(parameter n=4)(
  input [n-1:0] a,
  output reg [n-1:0] out
);

always @(*) begin//always block
  out = ~a;//not equation
end

endmodule
```

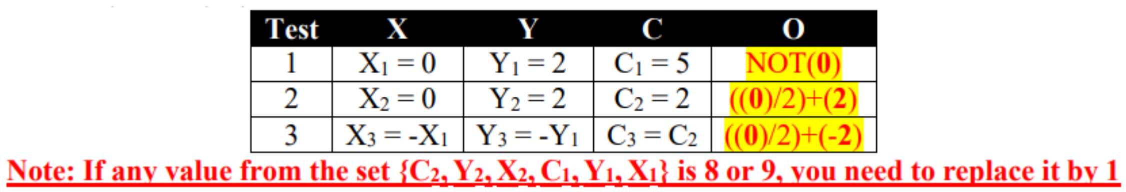
d) Write a structural Verilog model for your ALU designed in Part (b) using the elements you defined in Part (c)

ALU is the block that collects all of our project components, it has two n-bit inputs that the operations depend on, and one 3-bit input which determines the operation, also it has one n+2-bit output.


ALU_1211047.v

```verilog
//ALU_Structural to designed ALU
module ALU_1211047 #(parameter n=4)(
    input signed[n-1:0]x,y,
    input[2:0]sel,
    output signed[n+1:0]out
);
    //the wairs
    wire [n+1:0]sum,sum2,sub,multi;
    wire [n:0]div,NAND,XOR,NOR,notX,divX,divY;

    adder_1211047 (x,y,sum);//to sum first equation and save it in wire(sum)
    dividerBy2_1211047 (sum,div);//to division first equation and save it in wire(divv) AS FINAL STEP in this eqution

    multiplierBy2_1211047 (sum,multi);//to multplier the sum in second equation by to and save it in wire(multii) AS FINAL STEP in thi

    dividerBy2_1211047 (x,divX);//to divisin x by 2 in third equation and store it in (divX)
    adder_1211047 (divX,y,sum2);//to sum third equation to y and save it in (sum2) AS FINAL STEP in this eqution

    dividerBy2_1211047 (y,divY);//to divisin y by 2 in forth equation and store it in (divY)
    subtract_1211047 (x,divY,sub);//to subtract forth equation from x and save it in wire(sub) AS FINAL STEP in this eqution

    nand_1211047 (x,y,NAND);//the equation number 5
    not_1211047 (x,notX);//the equation number 6
    nor_1211047 (x,y,NOR);//the equation number 7
    xor_1211047 (x,y,XOR);//the equation number 8


    mux_1211047 (div,multi,sum2,sub,NAND,notX,NOR,XOR,sel,out);//mux to selecte one of inputs



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	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

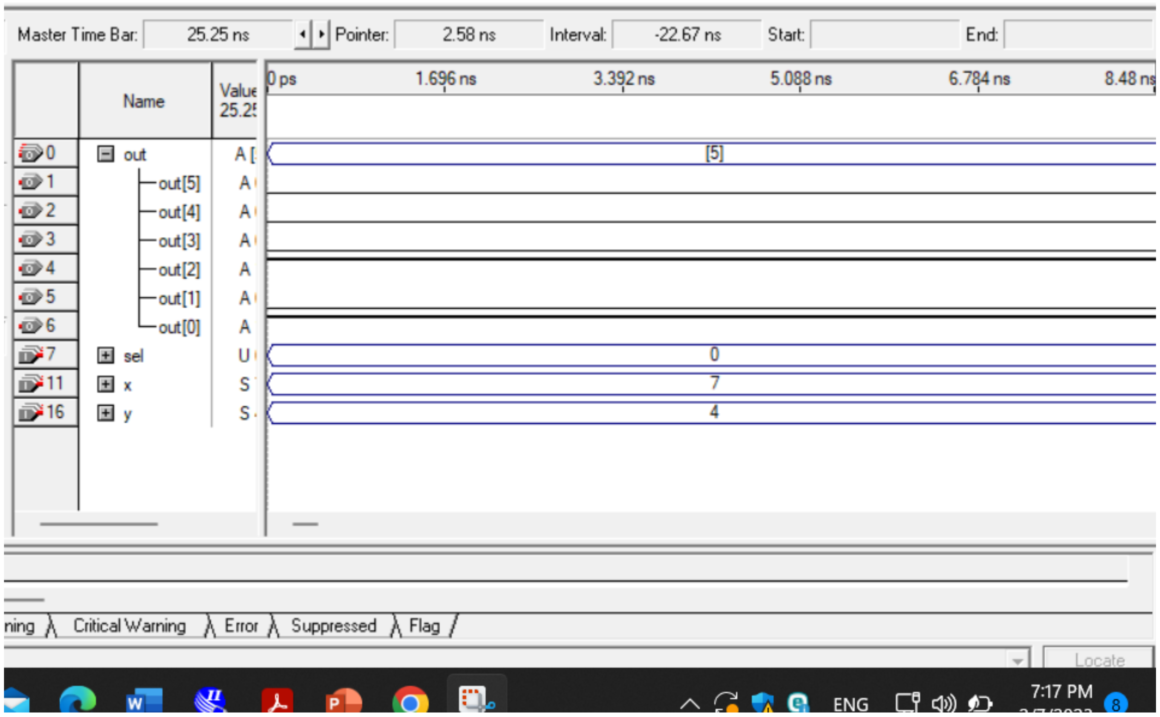
My ID number is **1211047** so,

Test 1 X1 =7, Y1 =4, C1 =0 out: (7+4)/2

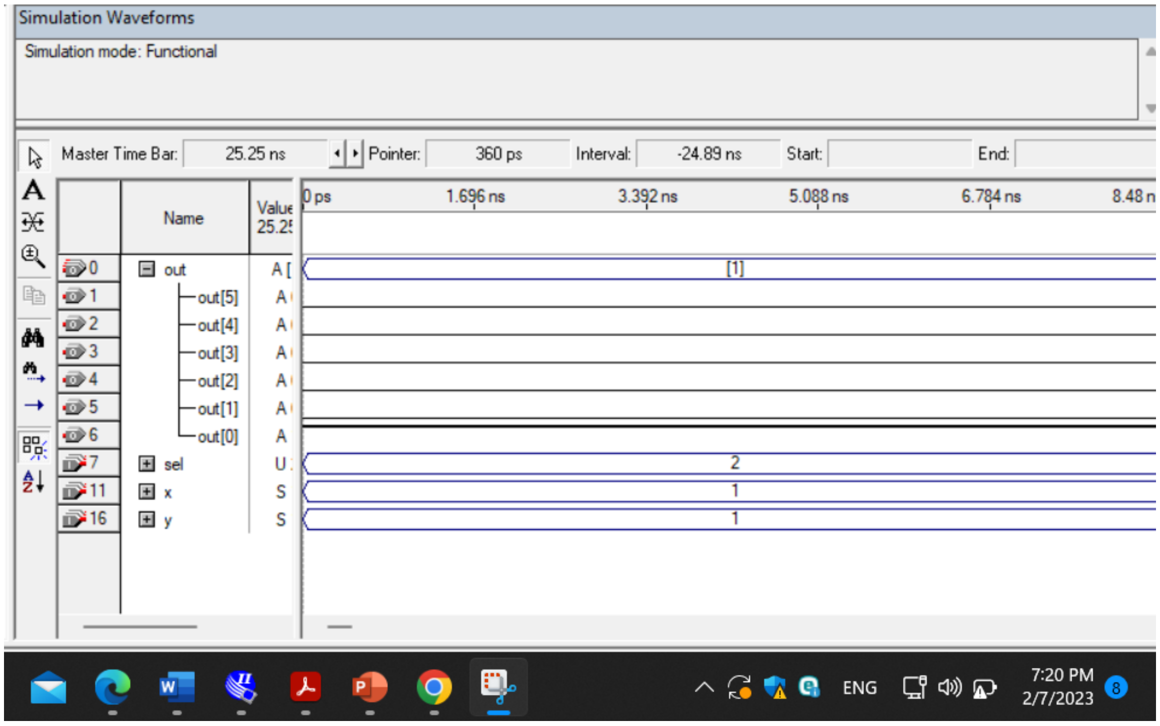
Test 2 X2 =1, Y2 =1, C2 =2 out: (1/2) +1

Test 3 X3 =-7, Y3 =-4, C3 =2 out: (-7/2) +-4

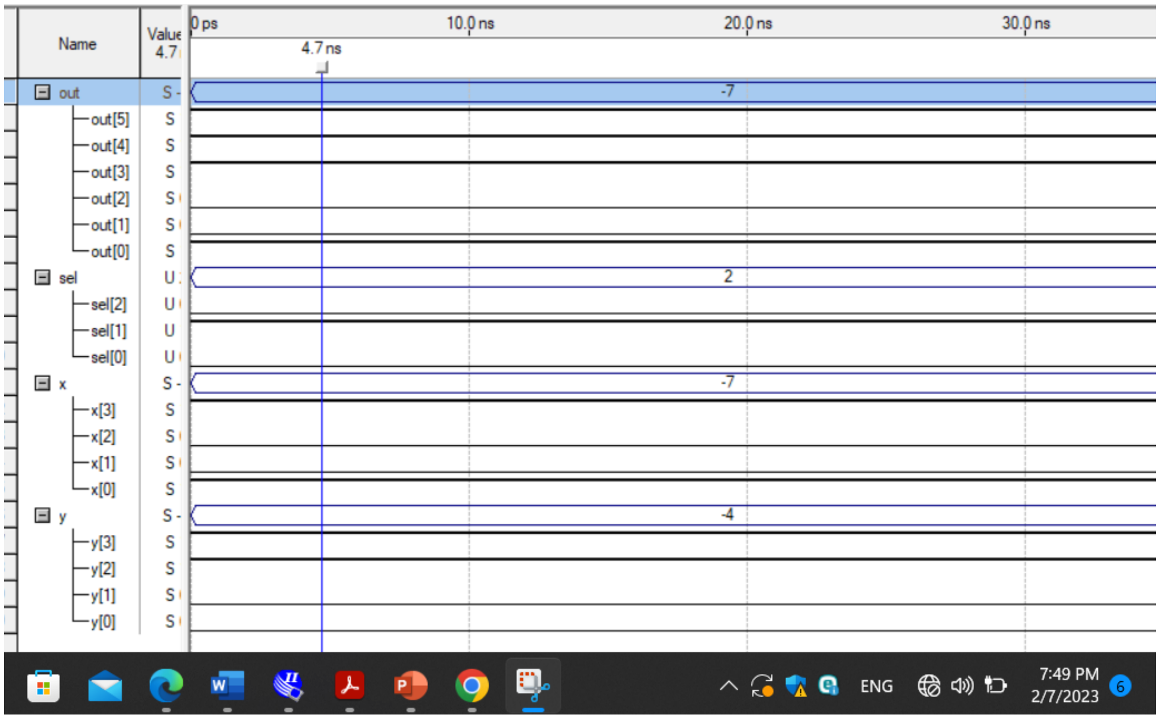
The waveforms:


Master Time Bar: 25.25 ns
Pointer: 2.58 ns
Interval: -22.67 ns
Start:
End:
Name
Value 25.25
0 ps
1.696 ns
3.392 ns
5.088 ns
6.784 ns
8.48 ns
out
out[5]
out[4]
out[3]
out[2]
out[1]
out[0]
sel
x
y
[5]
0
7
4
Critical Warning
Error
Suppressed
Flag
Locate
ENG
7:17 PM

Test1


Simulation Waveforms
Simulation mode: Functional
Master Time Bar: 25.25 ns
Pointer: 360 ps
Interval: -24.89 ns
Start:
End:
Name
Value 25.25
0 ps
1.696 ns
3.392 ns
5.088 ns
6.784 ns
8.48 n
out
out[5]
out[4]
out[3]
out[2]
out[1]
out[0]
sel
x
y
[1]
2
1
1
ENG
7:20 PM
2/7/2023

Test2

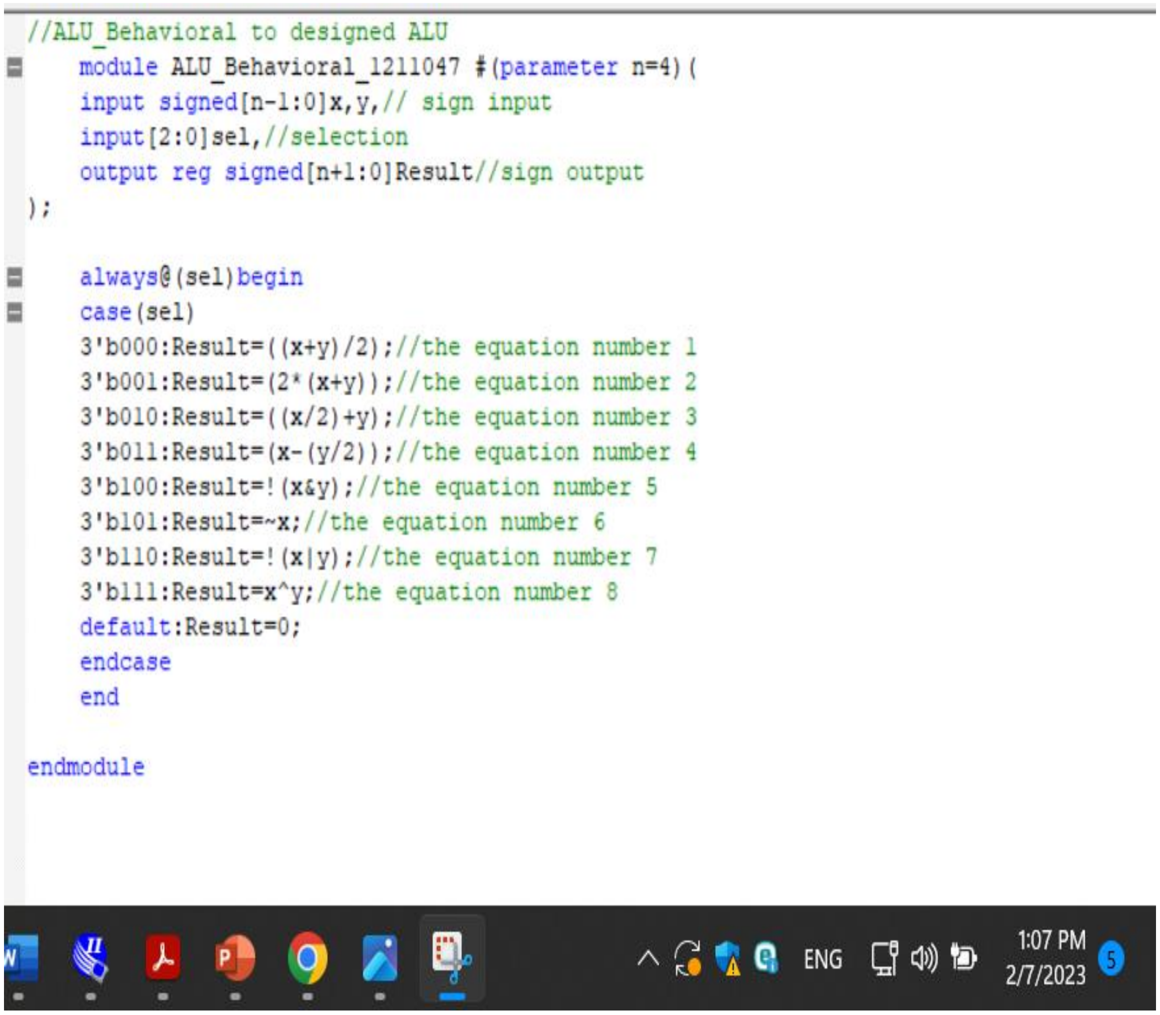

Name
Value 4.7
0 ps
10.0 ns
20.0 ns
30.0 ns
4.7 ns
out
out[5]
out[4]
out[3]
out[2]
out[1]
out[0]
sel
sel[2]
sel[1]
sel[0]
x
x[3]
x[2]
x[1]
x[0]
y
y[3]
y[2]
y[1]
y[0]
-7
2
-7
-4
ENG
7:49 PM
2/7/2023

Test3

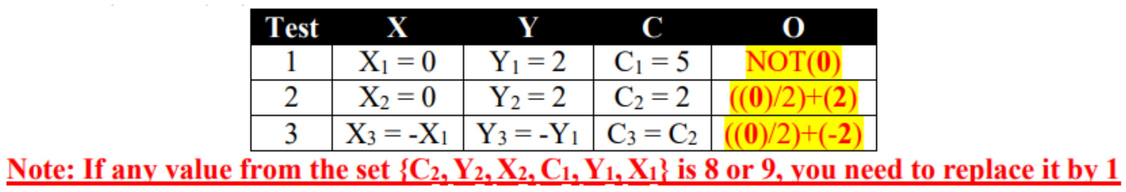
=================

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

ALU is the block that collects all of our project components, it has two n-bit inputs that the operations depend on, and one 3-bit input which determines the operation, also it has one n+2-bit output.


```
//ALU_Behavioral to designed ALU
    module ALU_Behavioral_1211047 #(parameter n=4)(
    input signed[n-1:0]x,y,// sign input
    input[2:0]sel,//selection
    output reg signed[n+1:0]Result//sign output
);

    always@(sel)begin
    case(sel)
    3'b000:Result=((x+y)/2);//the equation number 1
    3'b001:Result=(2*(x+y));//the equation number 2
    3'b010:Result=((x/2)+y);//the equation number 3
    3'b011:Result=(x-(y/2));//the equation number 4
    3'b100:Result=!(x&y);//the equation number 5
    3'b101:Result=~x;//the equation number 6
    3'b110:Result=!(x|y);//the equation number 7
    3'b111:Result=x^y;//the equation number 8
    default:Result=0;
    endcase
    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 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)

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

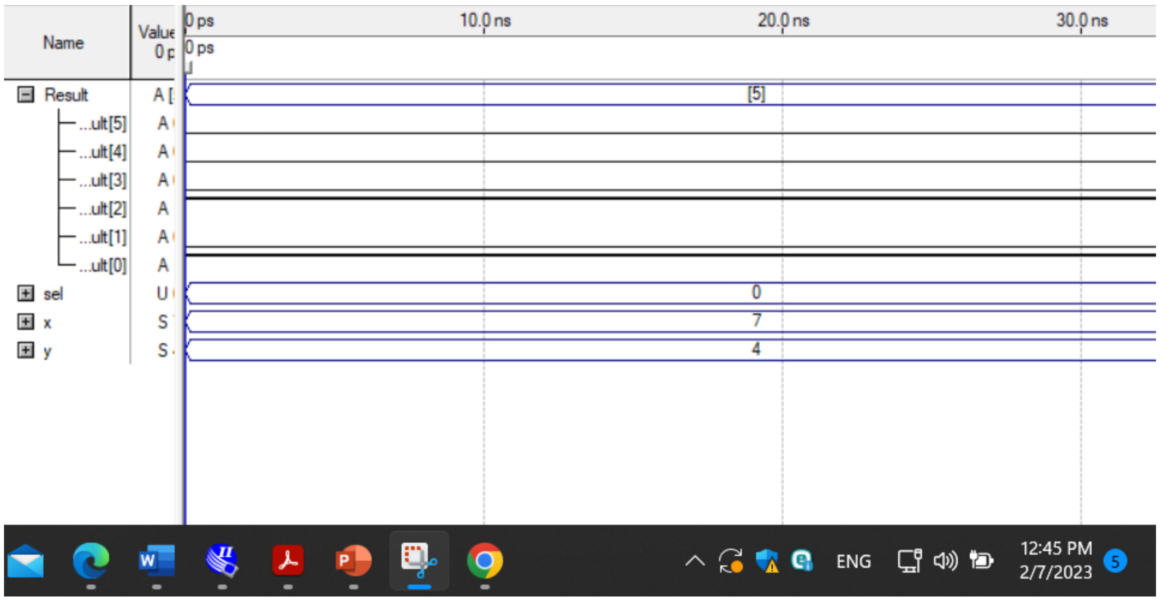
My ID number is **1211047** so,

Test 1 X1 =7, Y1 =4, C1 =0 out: (7+4)/2

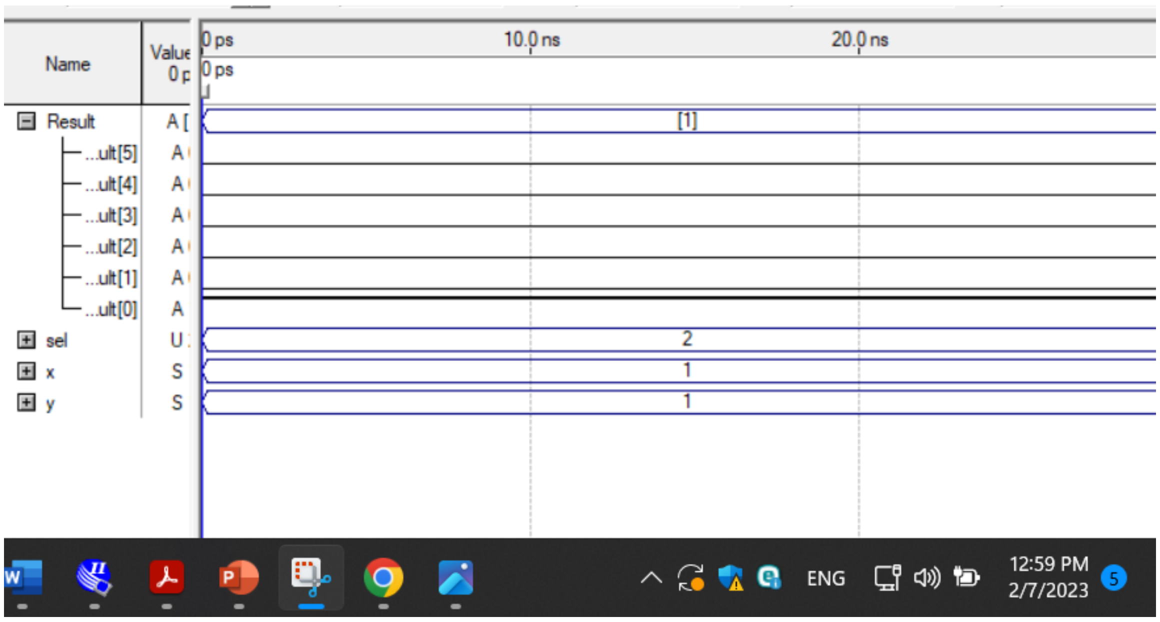
Test 2 X2 =1, Y2 =1, C2 =2 out: (1/2) +1

Test 3 X3 =-7, Y3 =-4, C3 =2 out: (-7/2) +-4

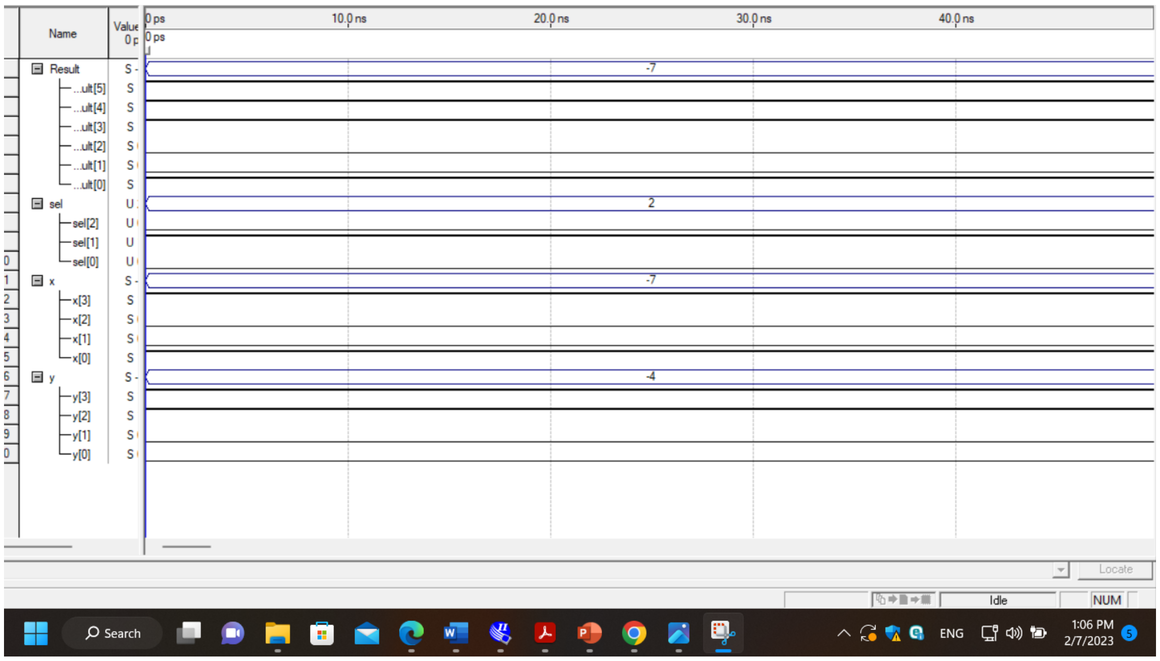
The waveforms:



Test1



Test2


Name
Value
0 ps
10.0 ns
20.0 ns
30.0 ns
40.0 ns
Result
...ult[5]
...ult[4]
...ult[3]
...ult[2]
...ult[1]
...ult[0]
sel
sel[2]
sel[1]
sel[0]
x
x[3]
x[2]
x[1]
x[0]
y
y[3]
y[2]
y[1]
y[0]
-7
2
-7
-4
Locate
Idle
NUM
Search
ENG
1:06 PM
2/7/2023

Test3

END
ROAD WORK