Verilog _HDL_Report

Faculty of Engineering and Technology
Electrical and Computer Engineering Department

Digital Systems ENCS2340

Name: shaimaa dar taha	**ID number:1222440**
Section :1	**Date:25/1/2024**

Instructor: Ismail Khater

اللهم ارحم شهداءنا وانصرنا على القوم الظالمين ❤

Project Description:

In this project, I will design a simple Arithmetic Logic Unit (ALU) using Verilog Hardware Description Language (HDL). The ALU capable of performing four basic arithmetic and logic operations: addition, subtraction, bitwise AND, and bitwise OR.

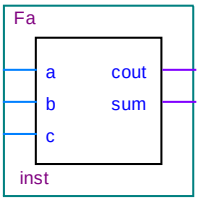
The Alu take two 4-bit inputs (A and B).

Include a 3-bit control input (OpCode) to select the operation:

- **3'b000: Addition**
- **3'b001: Subtraction**
- **3'b010: Bitwise AND**
- **3'b011: Bitwise OR**

Contents:

Full adder :(Fa)

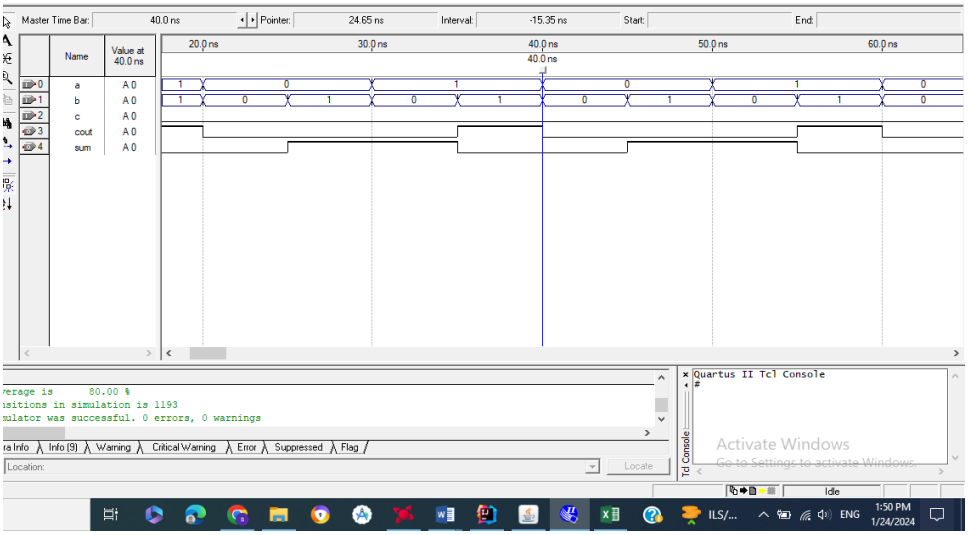


A full adder is a fundamental building block in digital electronics used to perform binary addition of three input bits: two single-bit binary numbers (A and B) and a carry-in (C) from a previous stage of addition. It produces two outputs: the sum (Sum) of the three input bits and a carry-out (Cout) that represents whether there is a carry

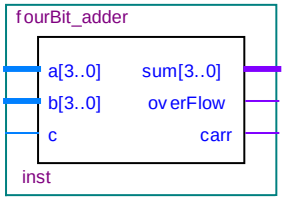
Verilog code

```
1   module Fa(input a, b, c, output cout, sum);
2   wire w1, w2, w3;
3   // w1: reseve a and b
4   // w2:reseve a xor b
5   //w3: reseve w2 and c
6   // sum=(a xor b)xor c
7   // cout= (a and b)+(a xor b)and c
8   and (w1, a, b);
9   xor (w2, a, b);
10  and (w3, w2, c);
11  xor (sum, w2, c);
12  or (cout, w1, w3);
13  endmodule
14  //shaimaa dar taha
```

Simulation report:(a=0,b=1 ,sum =1,cout =0) so its true



4_bit_Adder :

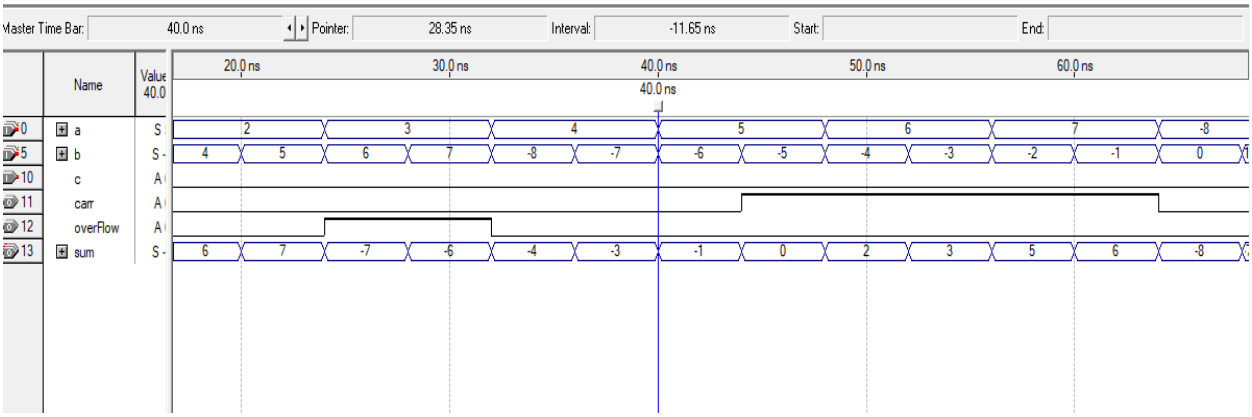


can add two four-bit binary numbers together. It's constructed using four full adders and some additional logic to handle the carry between each stage.

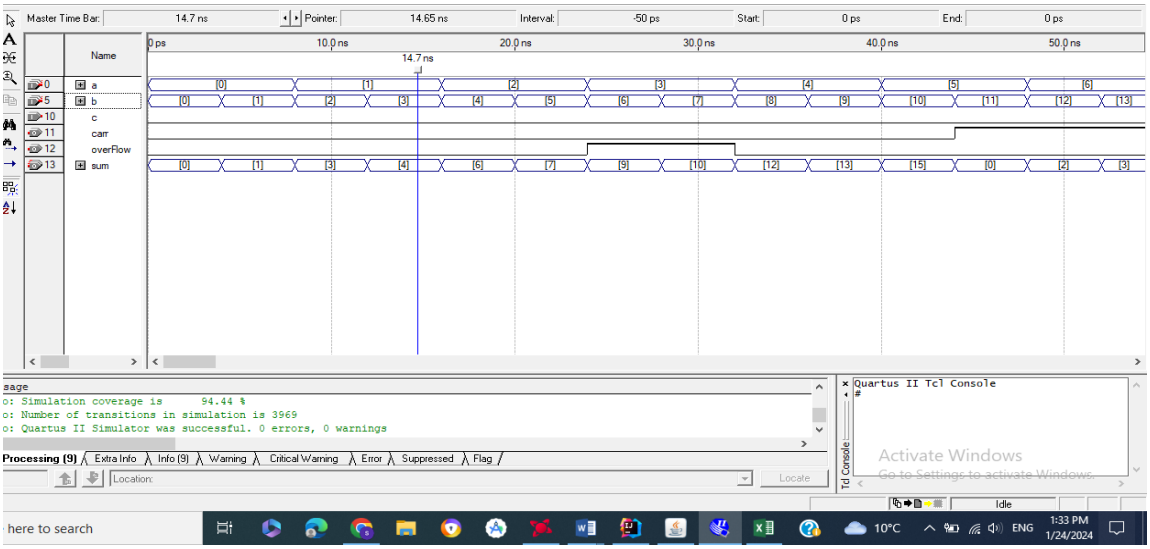
Verilog code :

```
1   module fourBit_adder(input[3:0]a,b,input c,output [3:0]sum,output overFlow,carr);
2   wire [2:0]w;
3   Fa f1( a[0],b[0],c,w[0],sum[0]);  // call the module Fa for bit[0]
4
5   Fa f2( a[1],b[1],w[0],w[1] ,sum[1]);  // for bit [1]
6
7   Fa f3( a[2],b[2],w[1] ,w[2],sum[2]);   // for bit[2]
8
9   Fa f4( a[3],b[3],w[2],carr, sum[3]);  // for bit[3]
10
11  xor (overFlow,carr,w[2]);  // to find over flow
12
13   //shaimaa dar taha
14  endmodule
15
```

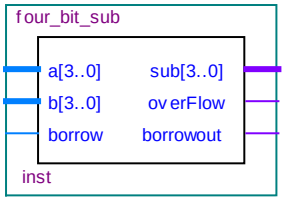
***Simulation report*: (*signed number*):**(3+6= -7 but over flow=1) ,(2+5=7)so the adder work correct



ASCII code =the same result at unsigned number :(2+4=6,carry=0)



4_bit _Subtractor



four-bit subtractor is a digital circuit that computes the subtraction of two four-bit binary numbers. It's essentially a combination of adders I call the module (Fa)and inverters to perform the subtraction operation. One common method to implement a four-bit subtractor is by using a two's complement approach.

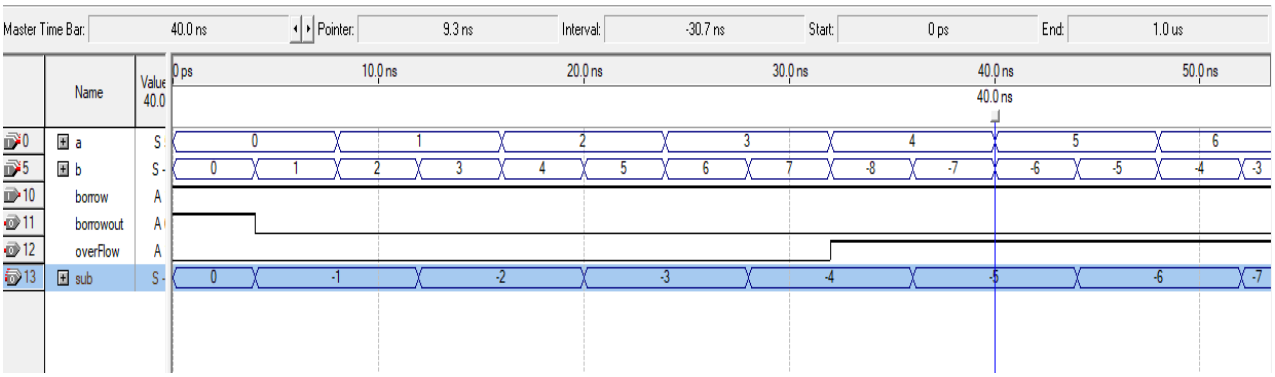
Verilog code :

```
module four_bit_sub(input[3:0]a,b,input borrow,output[3:0]sub ,output overFlow,borrowout);
wire [2:0]m;
Fa s1( a[0],~b[0],borrow,m[0], sub[0]); // for subtraction  bit [0]
Fa s2( a[1],~b[1],m[0],m[1],sub[1]);  // for subtraction  bit [1]
Fa s3( a[2],~b[2],m[1],m[2],sub[2]);  // for subtraction  bit [2]
Fa s4( a[3],~b[3],m[2],borrowout,sub[3]);  // for subtraction  bit [3]

xor (overFlow,borrowout,m[2]);   // to fined over flow its importent when subtract signed number

// shaimaa Dar taha
endmodule
```

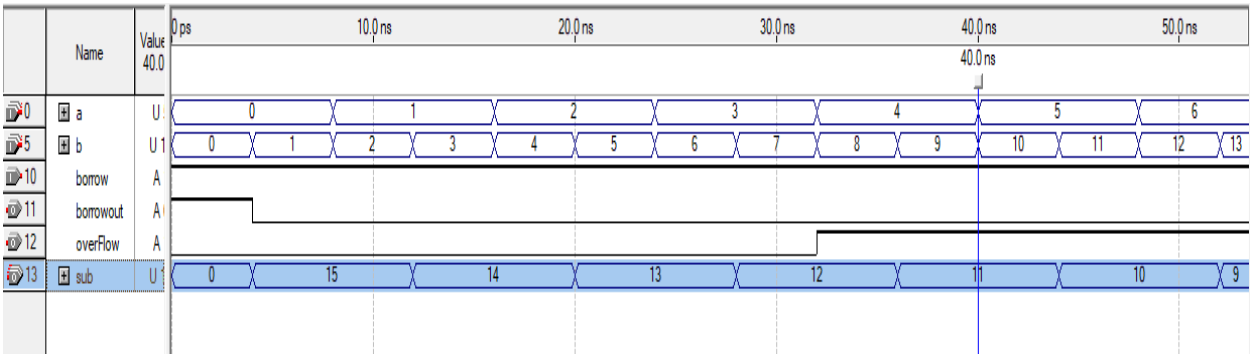
Another code :

```
module four_bit_sub(input[3:0]a,b,input borrow,output[3:0]sub ,output overFlow,borrowout);
wire [2:0]m;
wire[3:0]nb;
not g1(nb[0],b[0]);
not g2(nb[1],b[1]);
not g3(nb[2],b[2]);
not g4(nb[3],b[3]);


Fa s1( a[0],nb[0],borrow,m[0], sub[0]); // for subtraction  bit [0]
Fa s2( a[1],nb[1],m[0],m[1],sub[1]);  // for subtraction  bit [1]
Fa s3( a[2],nb[2],m[1],m[2],sub[2]);  // for subtraction  bit [2]
Fa s4( a[3],nb[3],m[2],borrowout,sub[3]);  // for subtraction  bit [3]

xor (overFlow,borrowout,m[2]);   // to fined over flow its importent when subtract signed number

// shaimaa Dar taha
endmodule

```

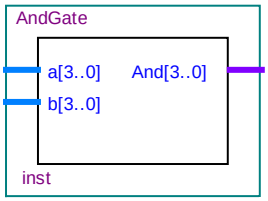
Simulation report: (*signed number*): (3-6=-3 ,over flow =0)so the module is correct



Simulation report: (*unsigned number*):(5-10=11 but over flow is 1)



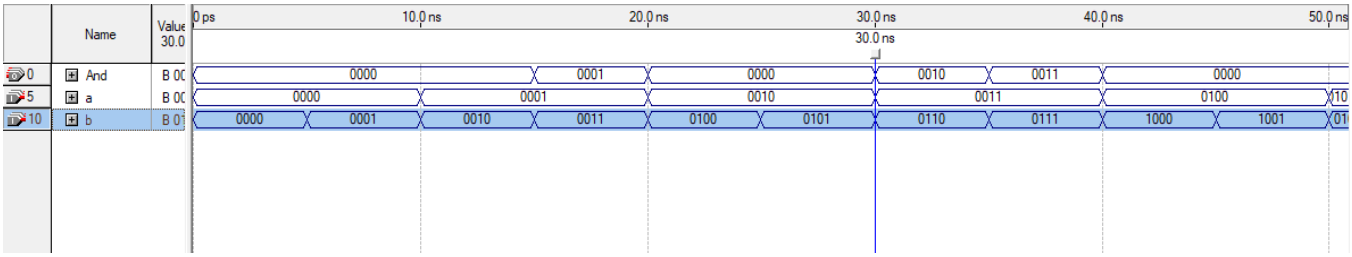
Bitwise AND:



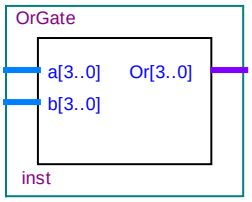
Verilog code : **this module all 4 bit(a)anding with 4 bit (b)**

```
1  module AndGate(input [3:0]a,b,output [3:0]And);
2  //and gate for every bit in a with every bit in b
3
4  and(And[0],a[0],b[0]);
5  and(And[1],a[1],b[1]);
6  and(And[2],a[2],b[2]);
7  and(And[3],a[3],b[3]);
8  //shaimaa Dar taha
9  endmodule
```

Simulation report: (0000 and 0000 give 0000)so the simulation is true

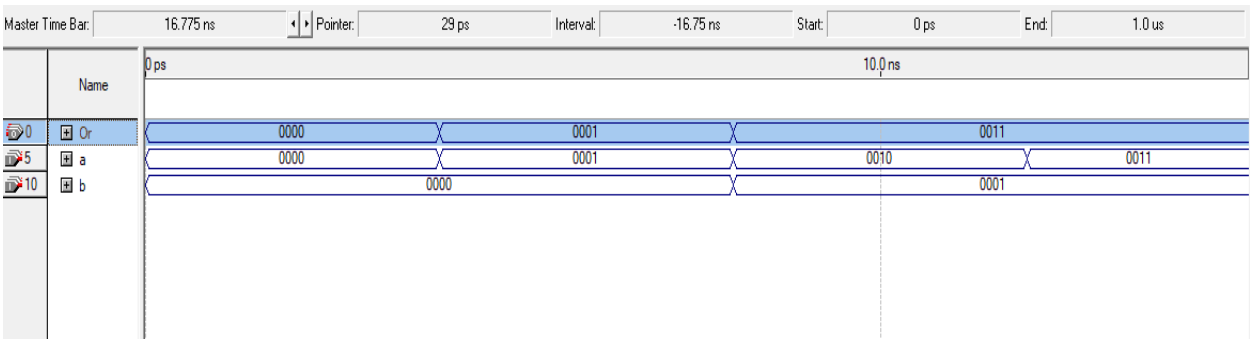


Bitwise OR: Verilog code : this module all 4 bit(a) oring with 4 bit (b)

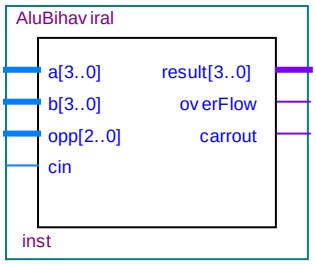


```
1  module OrGate(input[3:0]a,b,output [3:0] Or);
2  // take 2 number with 4 bit and oring them
3  or(Or[0],a[0],b[0]);
4  or(Or[1],a[1],b[1]);
5  or(Or[2],a[2],b[2]);
6  or(Or[3],a[3],b[3]);
7  // shaimaa dar taha
8  endmodule
9
```

Simulation report:(0000 or0001=0001) so the simulation is true



Data flow ALU:



Module for ALU that capable of performing four basic arithmetic and logic operations: addition, subtraction, bitwise AND, and bitwise OR.in data flow

Verilog code:

```
1  module AluDataflow(output [3:0]Result,output overflow,Carry,input[3:0]a,b,input cin,input[2:0]opp);
2   wire [3:0]add,sub,andBitwize,orBitwize ;  //for the result
3   wire carr1,carr2,overflow1,overflow2;   // for carry and over flow
4
5
6   fourBit_adder  k1(a,b,cin,add,overflow1,carr1);
7
8   four_bit_sub k2(a,b,cin,sub,overflow2,carr2);
9
10  OrGate  k3(a,b, orBitwize);
11
12  AndGate  k4(a,b,andBitwize);
13
14  assign Result =(opp==3'b000)?add:   // to assign the result for the ALU depends on opp code
15  (opp==3'b001)?sub:
16  (opp==3'b010)?andBitwize:
17  (opp==3'b011)?orBitwize:
18   3'bz;   // dont care if op =4,5,6,7
19
20  assign overflow =(opp==3'b000)?overflow1:   // to assign the over flow for the subetraction and addition only
21  (opp==3'b001)?overflow2:
22  3'bz;// dont care if opp 2,3,4,5,6,7
23
24  assign Carry =(opp==3'b000)?carr1:// to assign the carry for the subetraction and addition only
25  (opp==3'b001)?carr2:
26  3'bz; // dont care
27  // shaimaa dar taha
28  endmodule
29
```

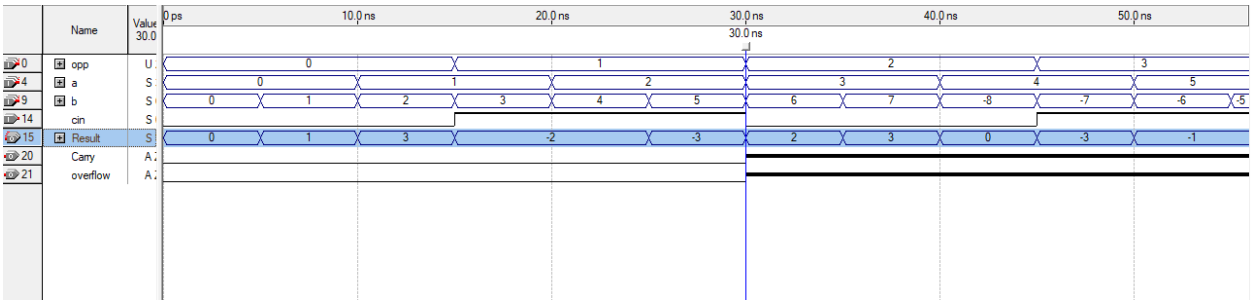
Simulation report: (signed number)

when opp=000 (add)(0+1=1) carry is 0 ,over flow is 0Its true

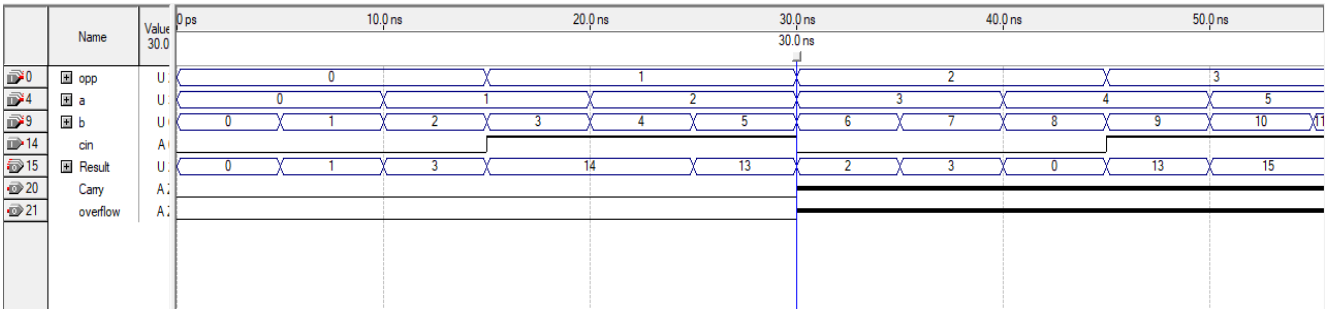
when opp=001(sub)(2-4= -2)so its true

When opp=010(and)(0011&0110=0010) its true

When opp =011(or)(0100|1001=1101=-3)so its true



Simulation report: (unsigned number):



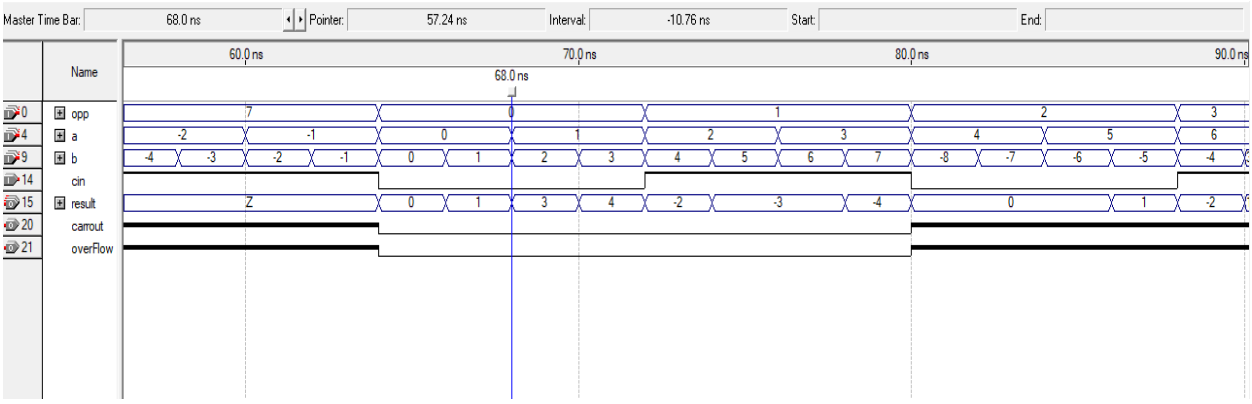
Behavioral_ALU:

Module for ALU that capable of performing four basic arithmetic and logic operations: addition, subtraction, bitwise AND, and bitwise OR.in behavioral

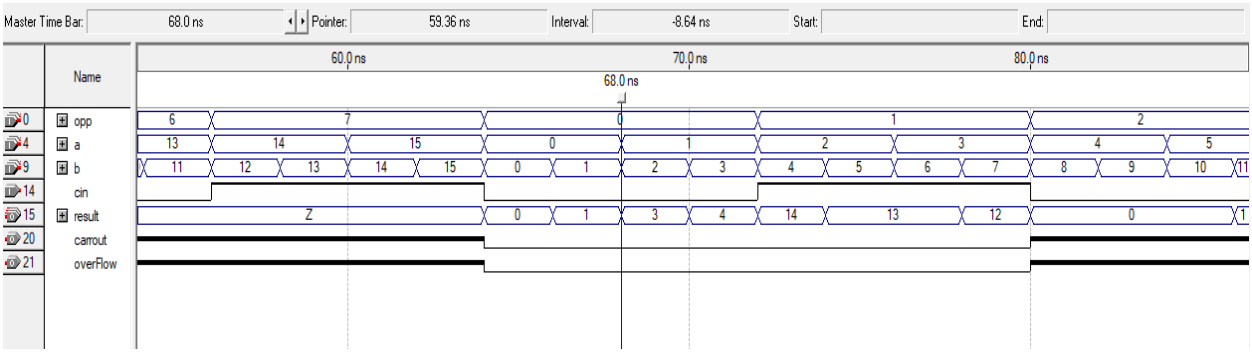
Verilog code:

```
1  module AluBihaviral(input [3:0]a,b,input [2:0]opp,input cin ,output reg [3:0]result ,output reg overFlow,carrout);
2
3    wire [3:0]add,sub,andBitwize,orBitwize ;  //for the result
4    wire carr1,carr2,overflow1,overflow2;   // for carry and over flow
5
6
7    fourBit_adder  k1(a,b,cin,add,overflow1,carr1);
8    four_bit_sub k2(a,b,cin,sub,overflow2,carr2);
9    OrGate  k3(a,b, orBitwize);
10   AndGate  k4(a,b,andBitwize);
11
12   // always @(a,b,opp,cin)
13   always @(*)
14 begin
15   if (opp==3'b000)  result =add;
16   else if (opp==3'b001)  result=sub ;
17   else if (opp==3'b010) result= andBitwize;
18   else if (opp==3'b011)  result=orBitwize ;
19   else  result =4'bz;
20   // end of result
21   if (opp==3'b000) carrout=carr1;
22   else if (opp==3'b001) carrout=carr2;
23   else  carrout=1'bz;
24   //end of carry
25   if (opp==3'b000) overFlow=overflow1;
26   else if (opp==3'b001) overFlow=overflow2;
27   else  overFlow=1'bz;
28   end
29   endmodule
30
```

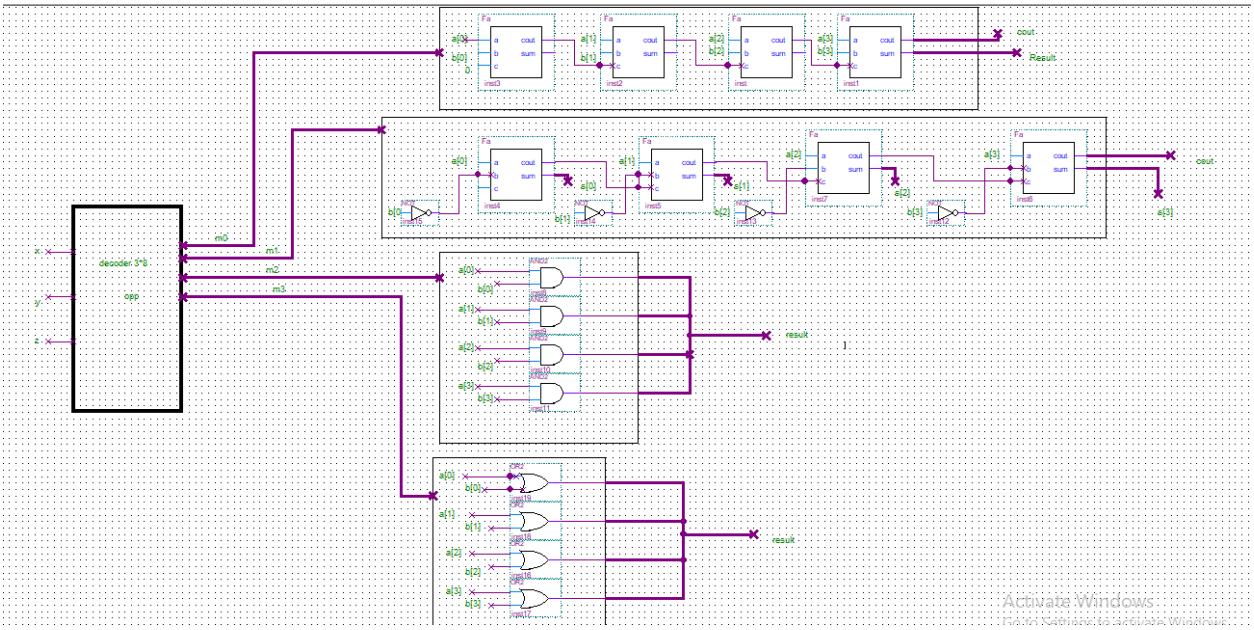
Simulation report: *(signe d number)* **(true 3-6=-3 , 0+1=1)**



Simulation report: *(unsigned number):*



Sympole for project:



Another sympole: