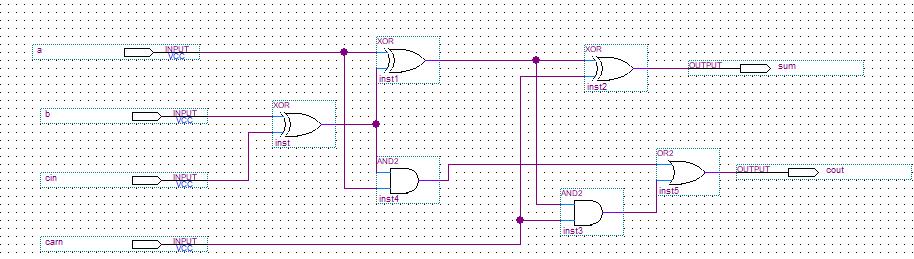
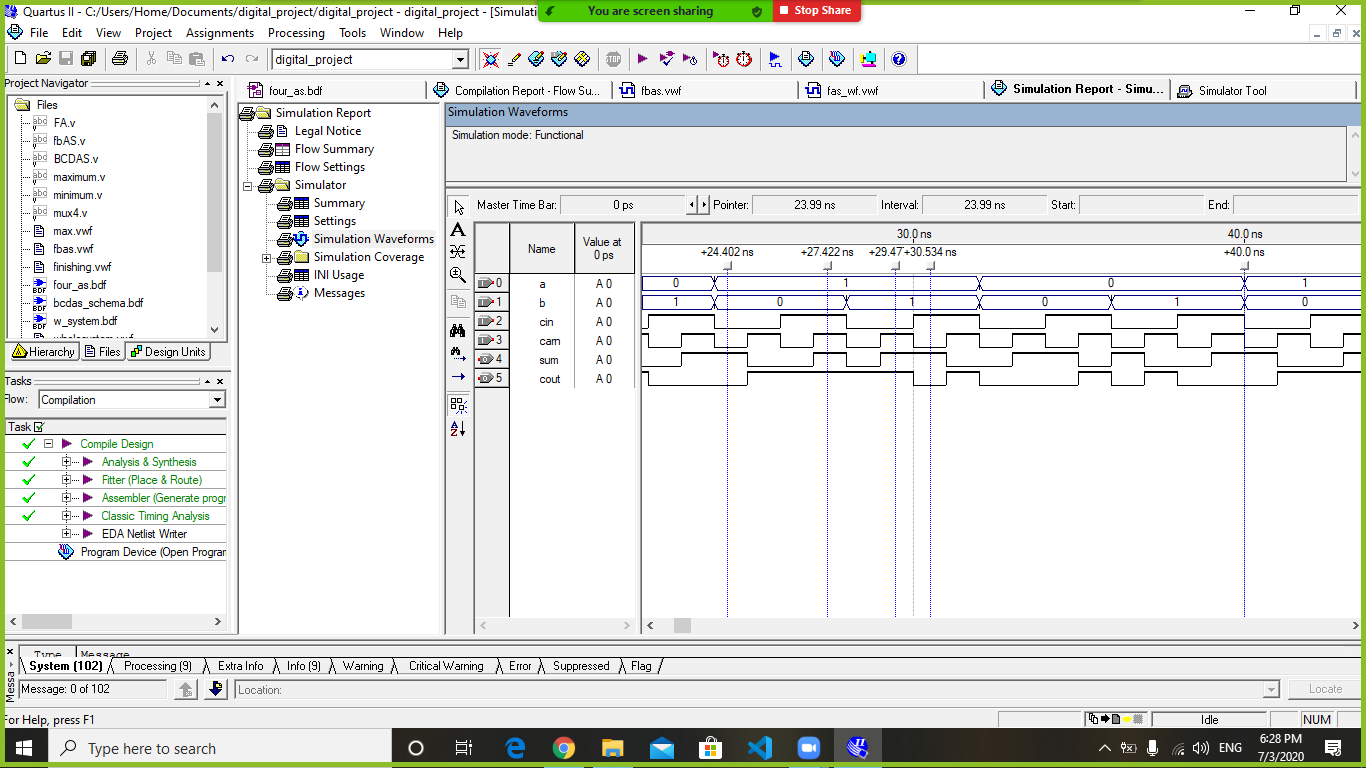
Digital project

Mazen Batrawi 1190102  
Ibraheem Ajaj 1190939

Code Specification:

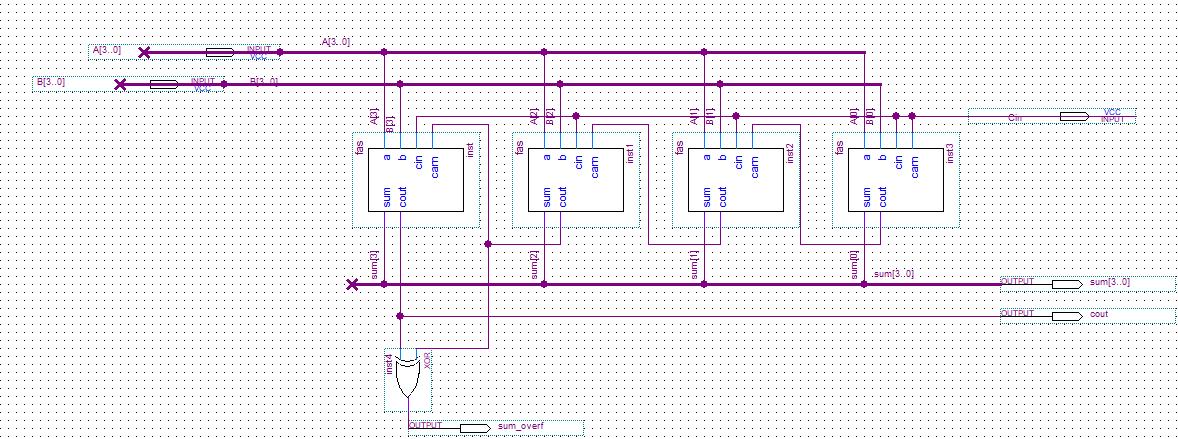
1. 1-bit adder subtractor: If it is a subtraction operation, we XOR b and cin so that we get the complement of b, otherwise, b will be XORed with 0 and it wont change. To get the sum, we XOR a and the result of XORing b and cin, then we XOR the result with the carry (carn). To calculate the carry, we ANDed a with the result of XORing b with cin, then ANDed the carry with XORing a and the result of XORing b and cin, then ORing the two results to get the carry.

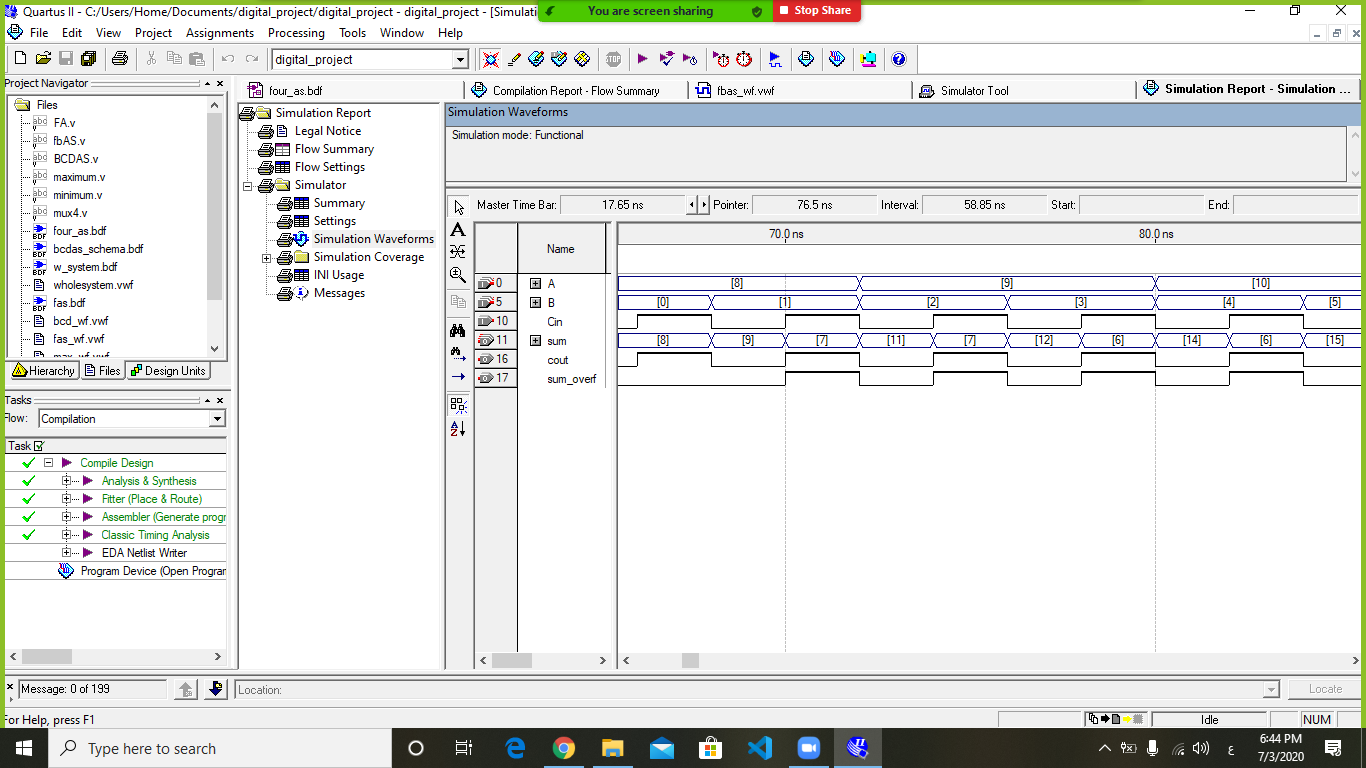
module FA(a,b,cin,carn,sum,car);  
input a,b,cin,carn;  
output sum,car;  
wire w1,w2,w3,w4;  
xor(w1,b,cin);  
xor(w2,a,w1);  
xor(sum,carn,w2);  
and(w3,a,w1);  
and(w4,carn,w2);  
or(car,w3,w4);  
endmodule



1. 4-bit adder subtractor: we called the function of the 1-bit adder subtractor and gave it it’s parameters, so the sum and the carry are calculated, then we calculated the overflow by XORing the last two carries.

module fbAS(a,b,cin,sum,car,v);  
input [3:0] a,b;  
output [3:0] sum;  
input cin;  
output car,v;  
wire car1,car2,car3;  
FA firstadder(a[0],b[0],cin,cin,sum[0],car1);  
FA secondadder(a[1],b[1],cin,car1,sum[1],car2);  
FA thirdadder(a[2],b[2],cin,car2,sum[2],car3);  
FA fourthadder(a[3],b[3],cin,car3,sum[3],car);  
xor (v,car3,car);  
endmodule





1. BCD adder subtractor: We XORed the cin with it self to generate two zeros on the first and third bit, then ANDed the cin with itself so we can get ones on the fourth and second bits if cin is 1, otherwise, the number will be 0000, then we called a 4-bit adder subtractor to make b in it’s 10’s complement form if cin =1, then the answer will be added to a, by calling another 4-bit adder subtractor, and cin is placed to be 1 so it will always be an adding operation. We ANDed the fourth and the third bits, and ANDed the fourth with the second bit, then ORed the previous two results with the carry of the addition, so if the answer is 1, the sum will be added to 6(0110), otherwise, the sum will remain the same.

module BCDAS(a,b,cin,result,cout);

input [3:0] a,b;

input cin;

output cout;

output [3:0] result;

wire w1,w2,car1,car2;

wire [3:0] err,temp\_sum,temp\_sum2,ten\_comp;

and (ten\_comp[3],cin,cin);

and (ten\_comp[1],cin,cin);

xor (ten\_comp[2],cin,cin);

xor (ten\_comp[0],cin,cin);

fbAS (ten\_comp,b,cin,temp\_sum,car1);

fbAS (a,temp\_sum,1'b0,temp\_sum2,car2);

and(w1,temp\_sum2[3],temp\_sum2[2]);

and (w2,temp\_sum2[3],temp\_sum2[1]);

or(err[2],car2,w1,w2);

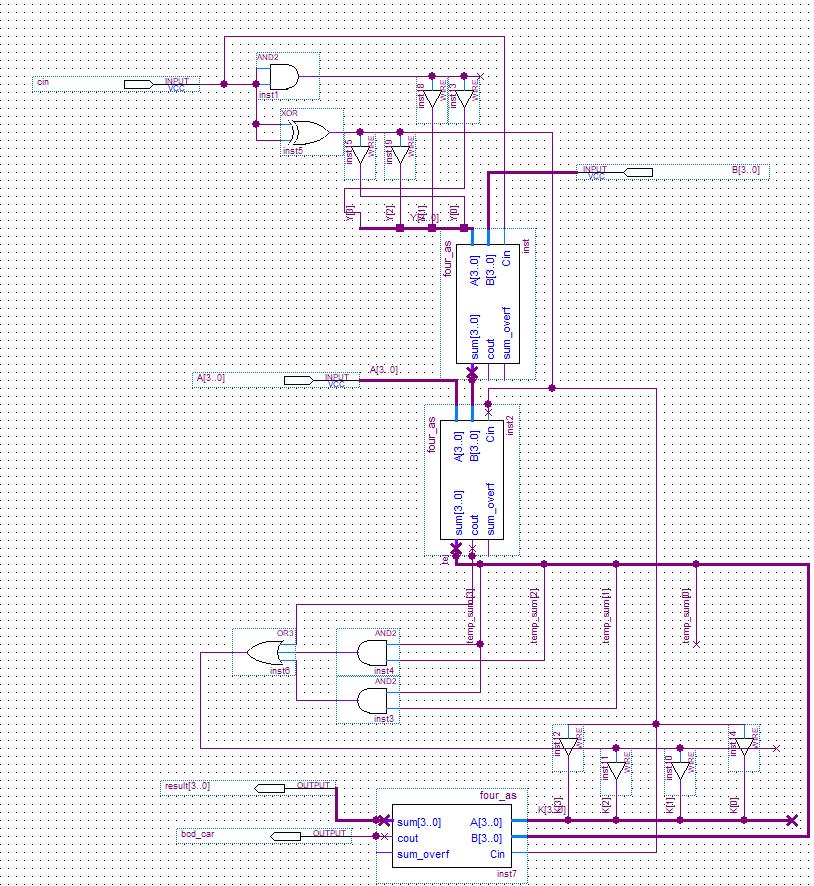
or(err[1],car2,w1,w2);

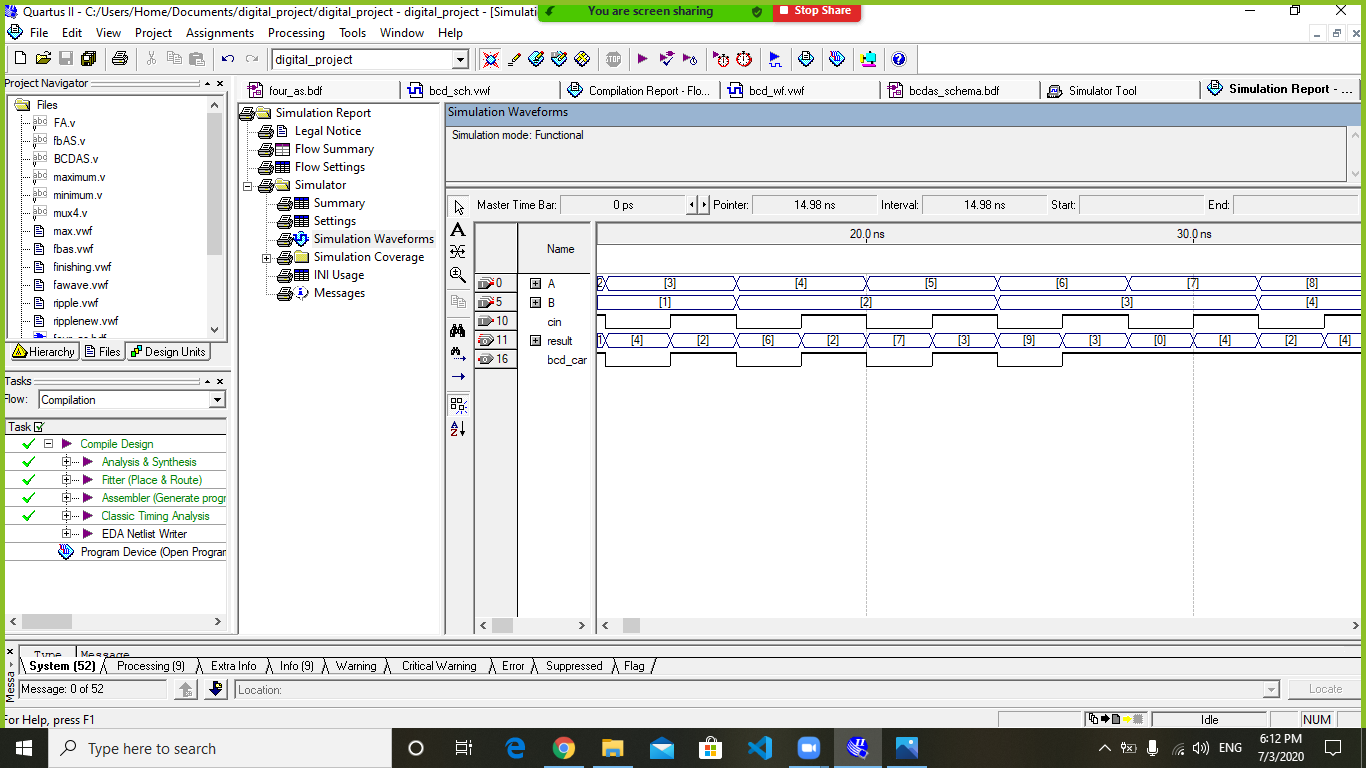
xor(err[3],car2,car2);

xor(err[0],car2,car2);

fbAS finish(temp\_sum2,err,1'b0,result,cout);

endmodule





1. Minimum and maximum: We used the behavioral way, so if a is bigger than b, the output will be a, the minimum code is the same, just changing the variable and the sign.

module maximum(a,b,max);

input [3:0] a,b;

output reg [3:0] max;

always @(a,b)

begin

if (a>b)

begin

max = a;

end

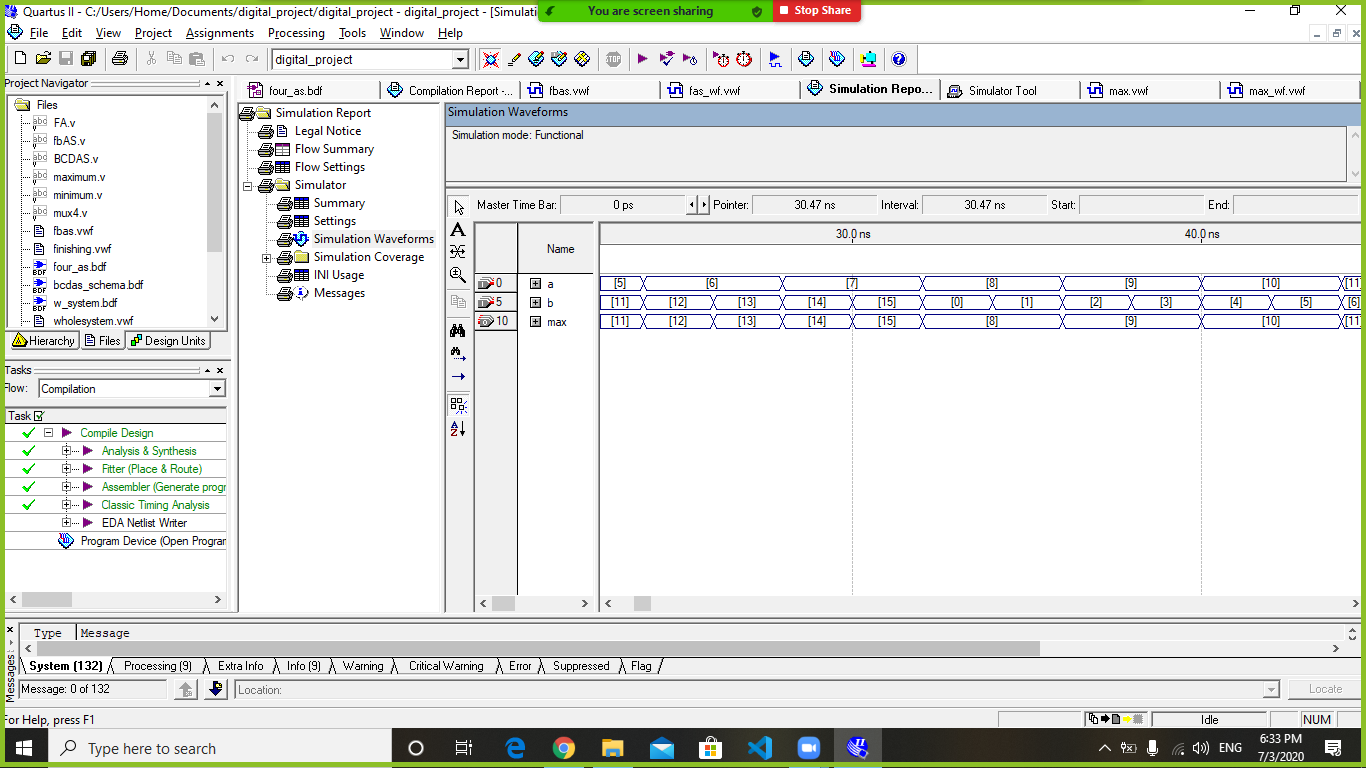
else

begin

max = b;

end

end  
endmodule



module minimum(a,b,min);

input [3:0] a,b;

output reg [3:0] min;

always @(a,b)

begin

if (a<b)

begin

min = a;

end

else

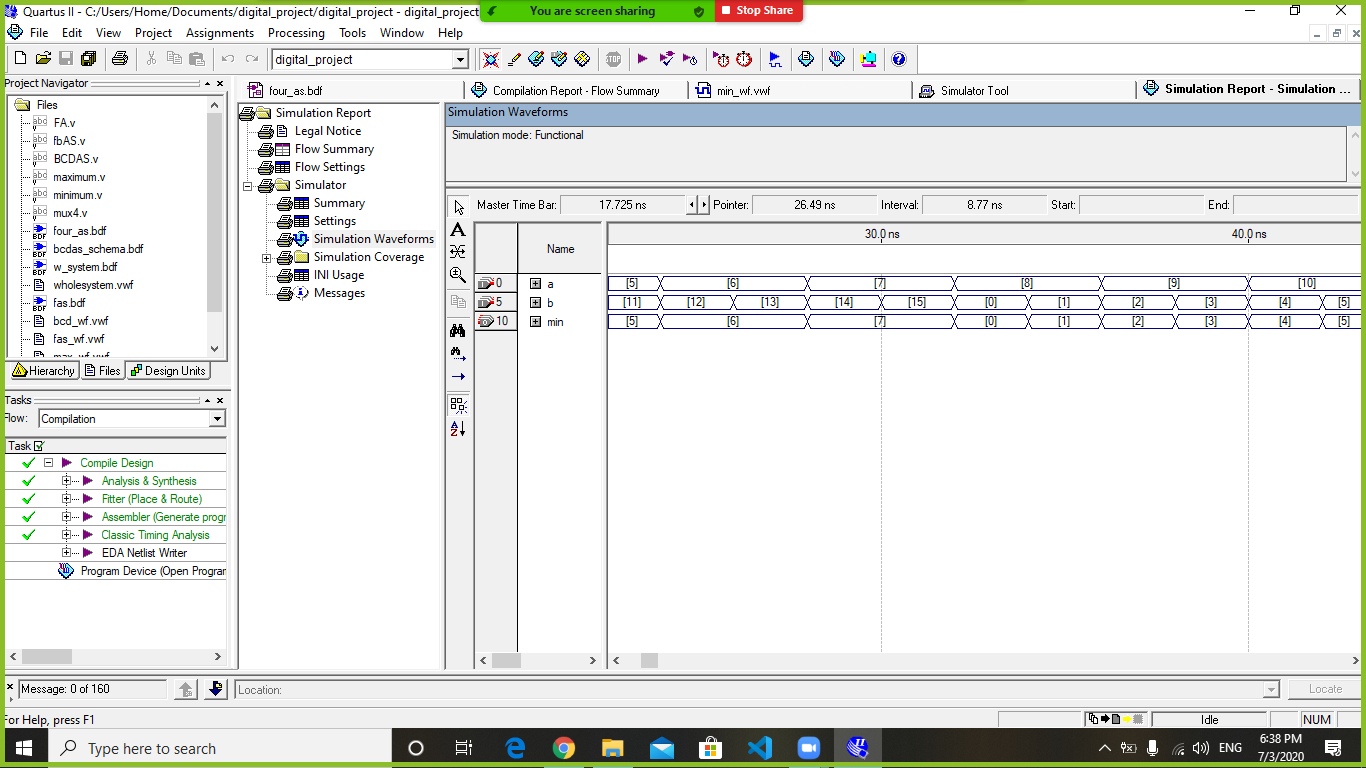
begin

min = b;

end

end

endmodule



1. Multiplexer: We used the behavioral way, we called the previous 4 functions, then showed the s1s0 cases in order to show the correct output by using if statements.

module mux4(d1,d2,d3,d4,s1,s0,outp);

input [3:0] d1,d2,d3,d4;

input s1,s0;

output reg [3:0]outp;

always @ (s1,s0)

begin

if (s1==0)

begin

if (s0==0)

outp=d1;

else

outp = d2;

end

else

begin

if (s0==0)

outp=d3;

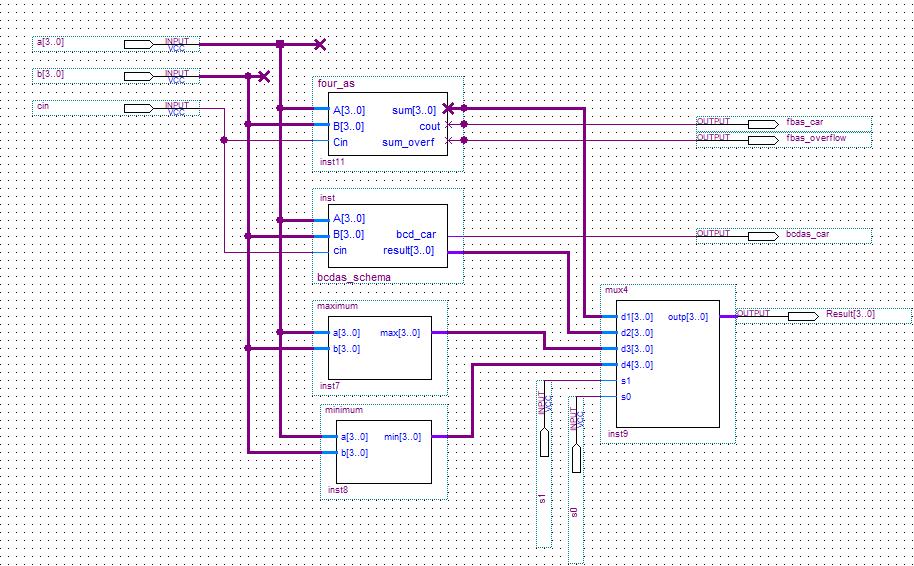
else

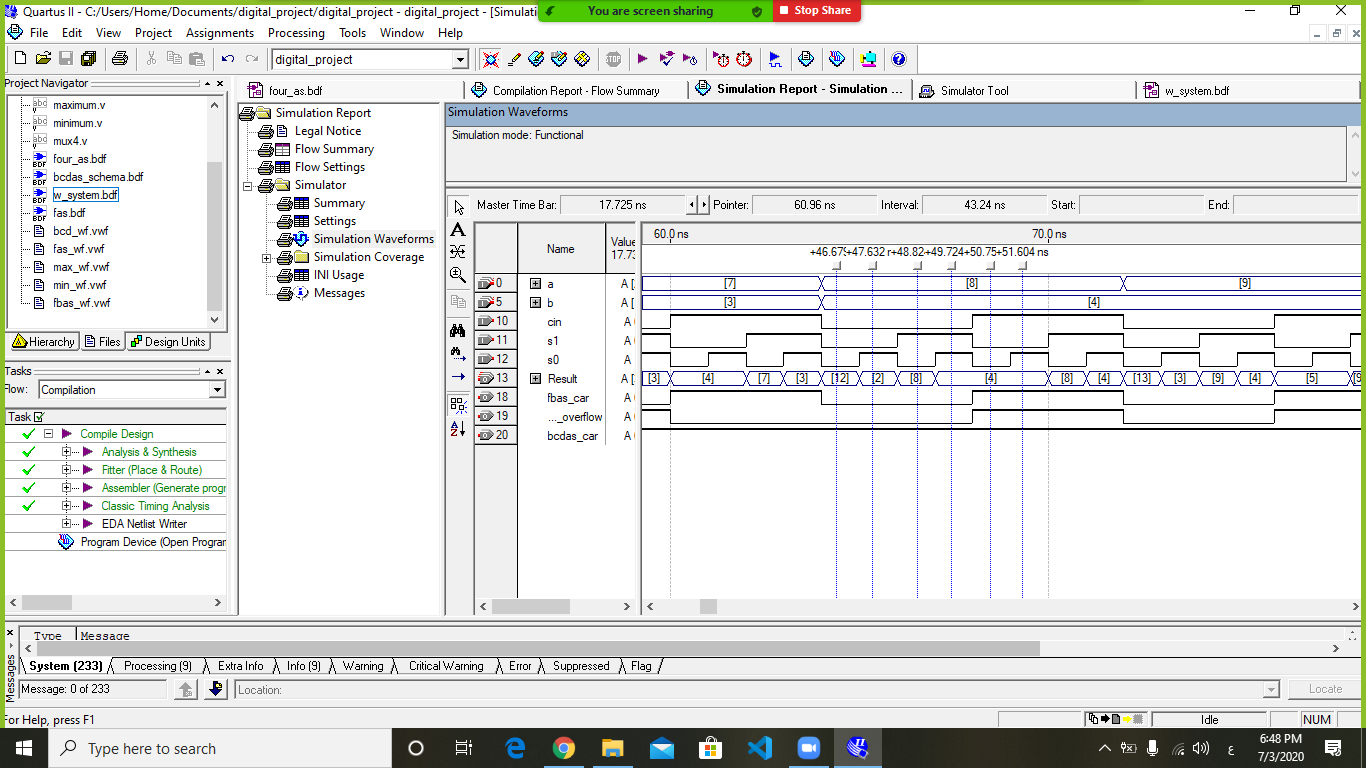
outp=d4;

end

end

endmodule





Teamwork

Ibraheem Ajaj 1190939:

* Coding 1-digit BCD adder/subtractor.
* Coding 4-bit Multiplexer.
* Designing the previous coded modules.
* Connected the whole system.
* Simulated the previous modules to check for the correctness.

Mazen Batrawi 1190102:

* Coding full adder/subtractor.
* Coding 4-bit adder/subtractor.
* Coding maximum and minimum.
* Designing the previous coded modules.
* Simulated the previous modules to check for the correctness.

At the end, we opened a zoom meeting together and tested the whole system and checked different cases.