

Modeling of Multifunction ALU

Verilog Project

ID: 1212326

2/6/2023



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

Bits of output (O) = $N + 2$

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).

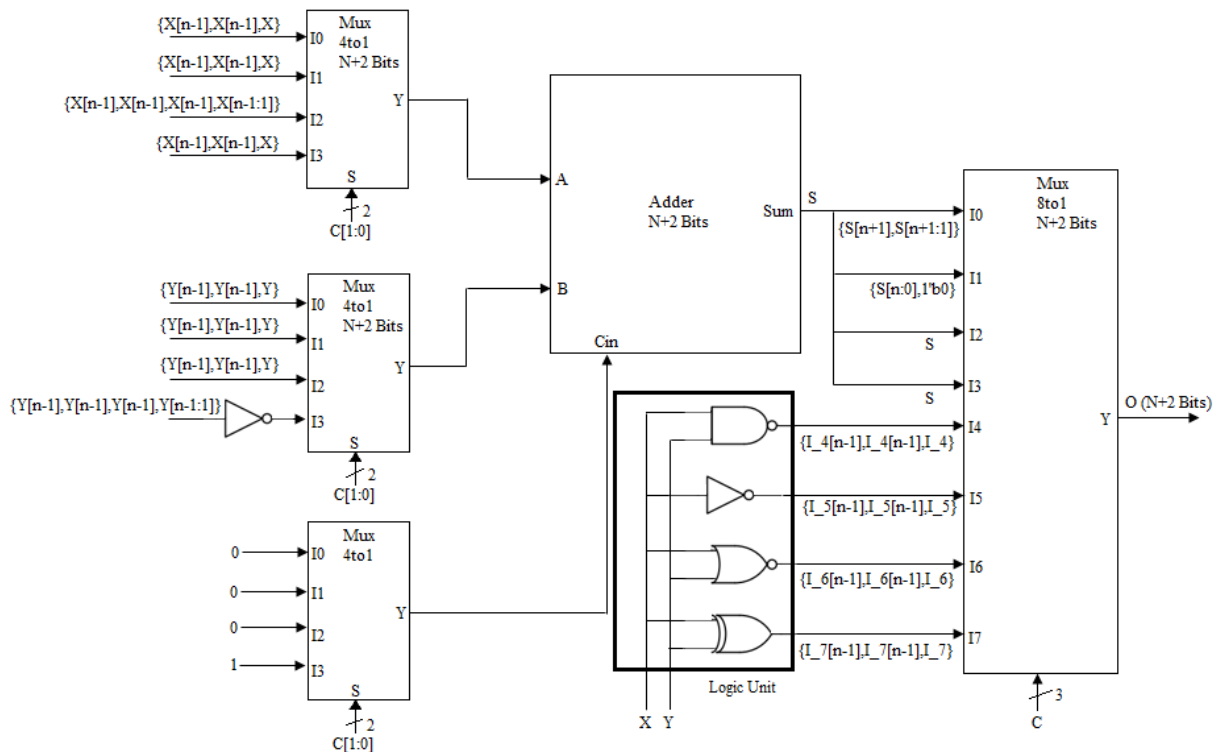


Figure 1. ALU Diagram

There are three 4-to-1 multiplexer are used which control the inputs of the N+2 Bit adder based on the first four functions. In this logic unit last four functions are perform which are bitwise NAND, NOT, NOR and XOR gate on the input values. After the logic unit each function output bit is extended from the MSB side so that actual bits will be equal to the output bits. At the output N+2 Bit 8-to-1 multiplexer is connected which select the one operation based on the selection lines and gives the O result.

- 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.

Adder Code

```
module Adder_1212326#(parameter N=4)(A,B,Cin,Sum); //define inputs and outputs, N by default is 4
input [N+1:0] A,B; //define N+2 A and B inputs
input Cin; //define carry input of Adder
output [N+1:0] Sum; //define N+2 addition output
assign Sum = A + B + {{N{1'b0}},Cin}; //addition on A+B+Cin
endmodule
```

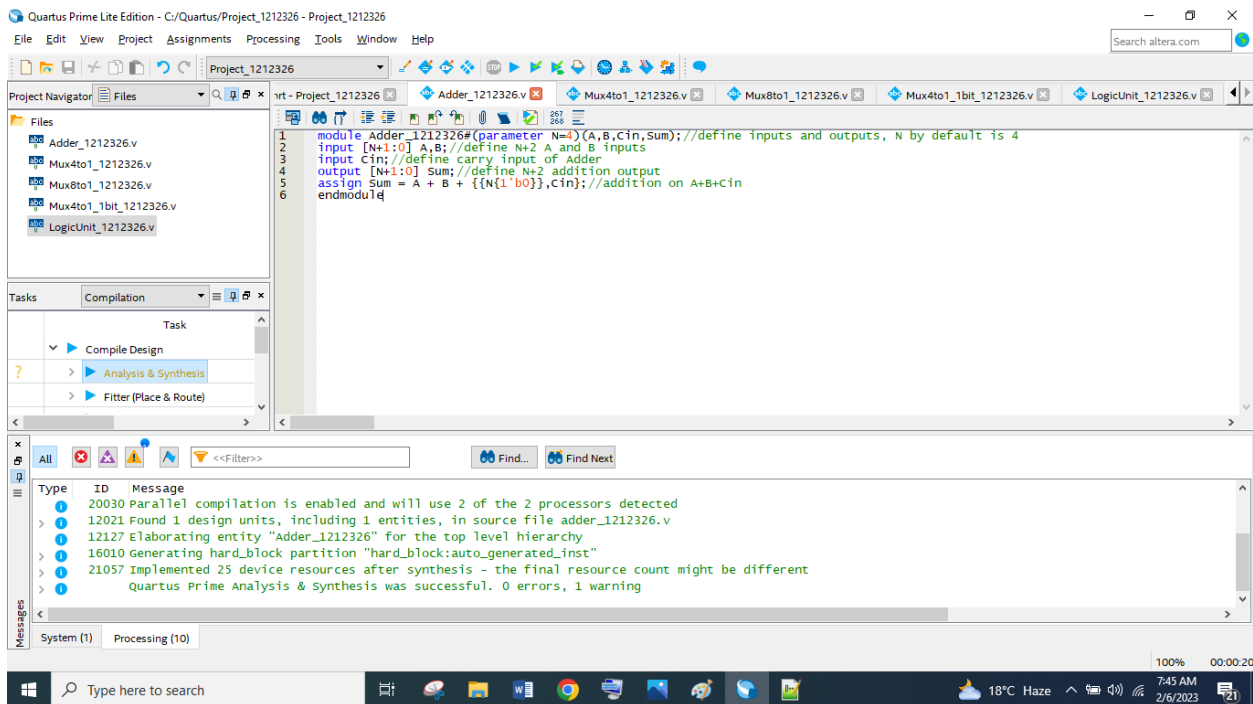


Figure 2. N+2 Bits Adder Module Verilog Code

This module takes the N+2 bit X and Y inputs and 1-bit Cin input, based on the first four ALU function this module perform addition and gives result in the form of N+2 bit S. Which will further classifies for the operation of the ALU.

Mux 4-to-1 Code

```
module Mux4to1_1212326 # (parameter N=4) (I0,I1,I2,I3,S,Y); //define input and output, N by default is 4
input [N+1:0] I0,I1,I2,I3; //N+2 bits data inputs of mux
input [1:0] S; //2 bit selection input of mux
output [N+1:0] Y; //N+2 mux output
assign Y = S[1] ? (S[0] ? I3:I2) : (S[0] ? I1:I0); //mux implementation using conditional statement
endmodule
```

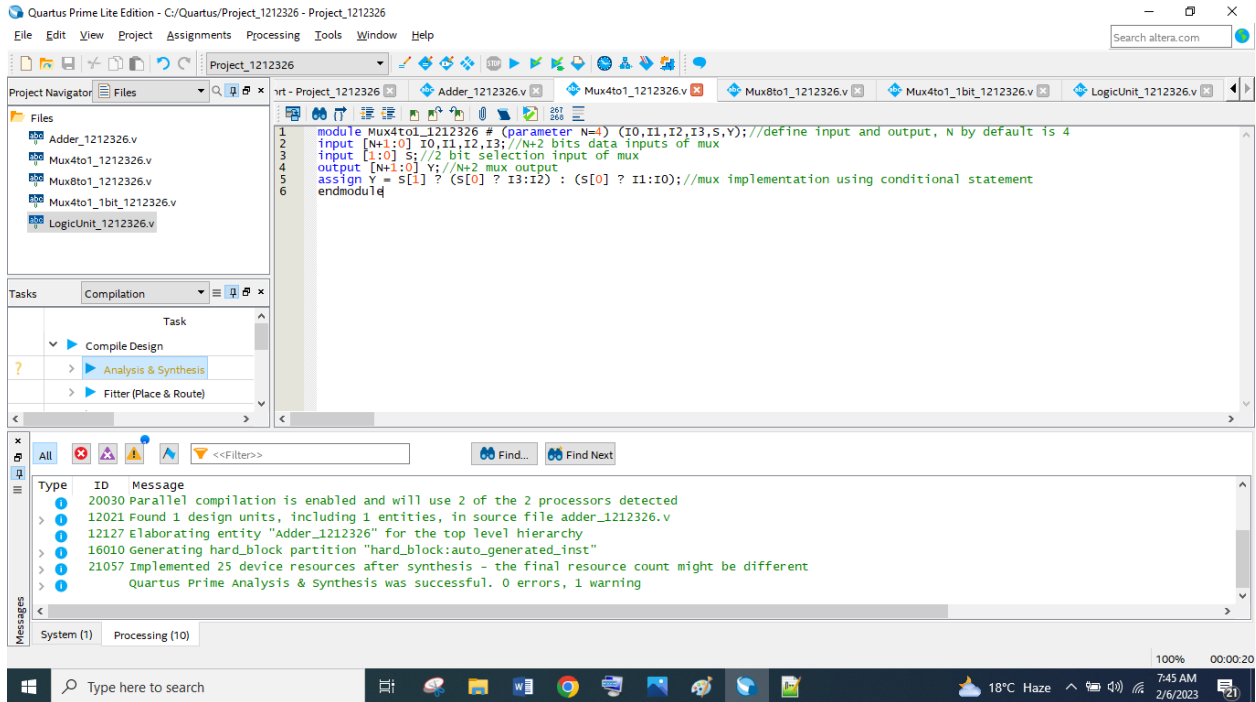


Figure 3. N+2 Bit 4-to-1 Multiplexer Module Verilog Code

This module takes 4 data inputs I0, I1, I2, I3 of N+2 bits and one 2-bit select line input and based on the select line input N+2 bit output is connected with respective input data line.

Mux 8-to-1 Code

```
module Mux8to1_1212326 # (parameter N=4) (I0,I1,I2,I3,I4,I5,I6,I7,S,Y); //define input and output, N by default is 4
input [N+1:0] I0,I1,I2,I3,I4,I5,I6,I7; //N+2 bits data inputs of mux
input [2:0] S; //3 bit selection input of mux
output [N+1:0] Y; //N+2 mux output
assign Y = S[2] ? (S[1] ? (S[0] ? I7:I6) : (S[0] ? I5:I4)) : (S[1] ? (S[0] ? I3:I2) : (S[0] ? I1:I0)); //mux implementation using
conditional statement
endmodule
```

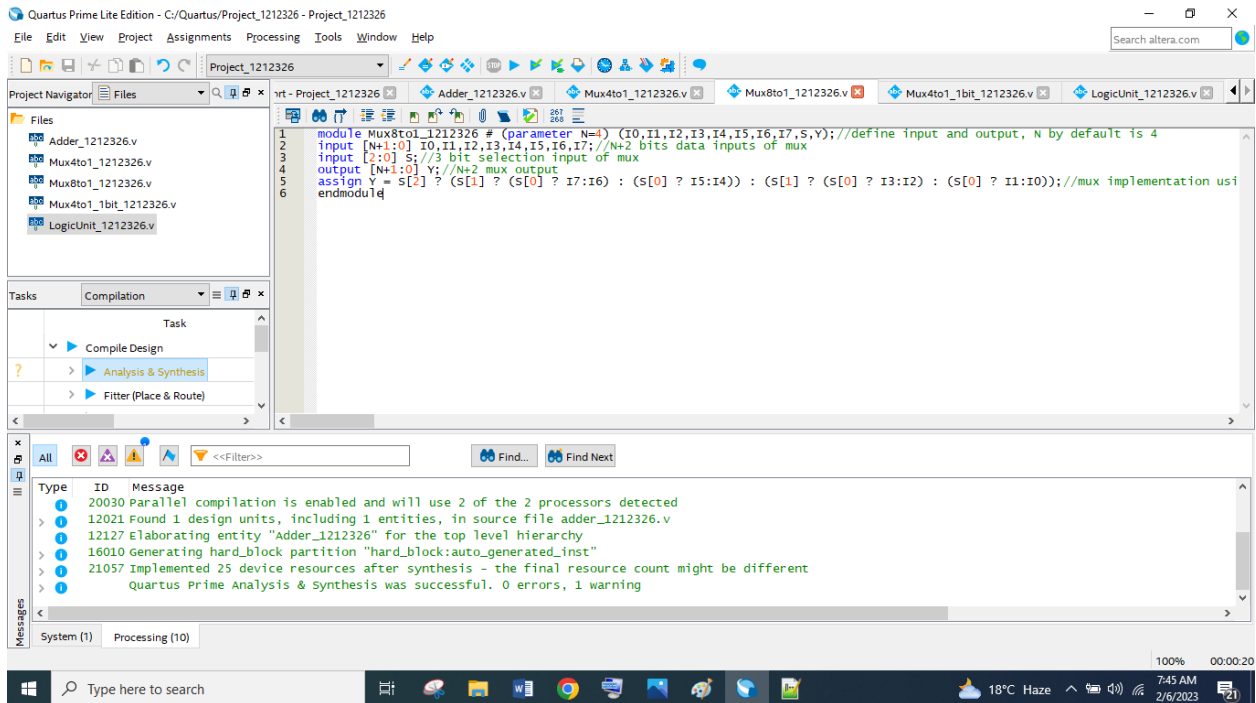


Figure 4. N+2 Bit 8-to-1 Multiplexer Module Verilog Code

This module takes 8 data inputs I0, I1, I2, I3, I4, I5, I6, I7 of N+2 bits and one 3-bit select line input and based on the select line input N+2 bit output is connected with respective input data line.

Mux 4-to-1 1-Bit Code

```
module Mux4to1_1bit_1212326 (I0,I1,I2,I3,S,Y); //define input and output, N by default is 4
input I0,I1,I2,I3; //N+2 bits data inputs of mux
input [1:0] S; //2 bit selection input of mux
output Y; //N+2 mux output
assign Y = S[1] ? (S[0] ? I3:I2) : (S[0] ? I1:I0); //mux implementation using conditional statement
endmodule
```

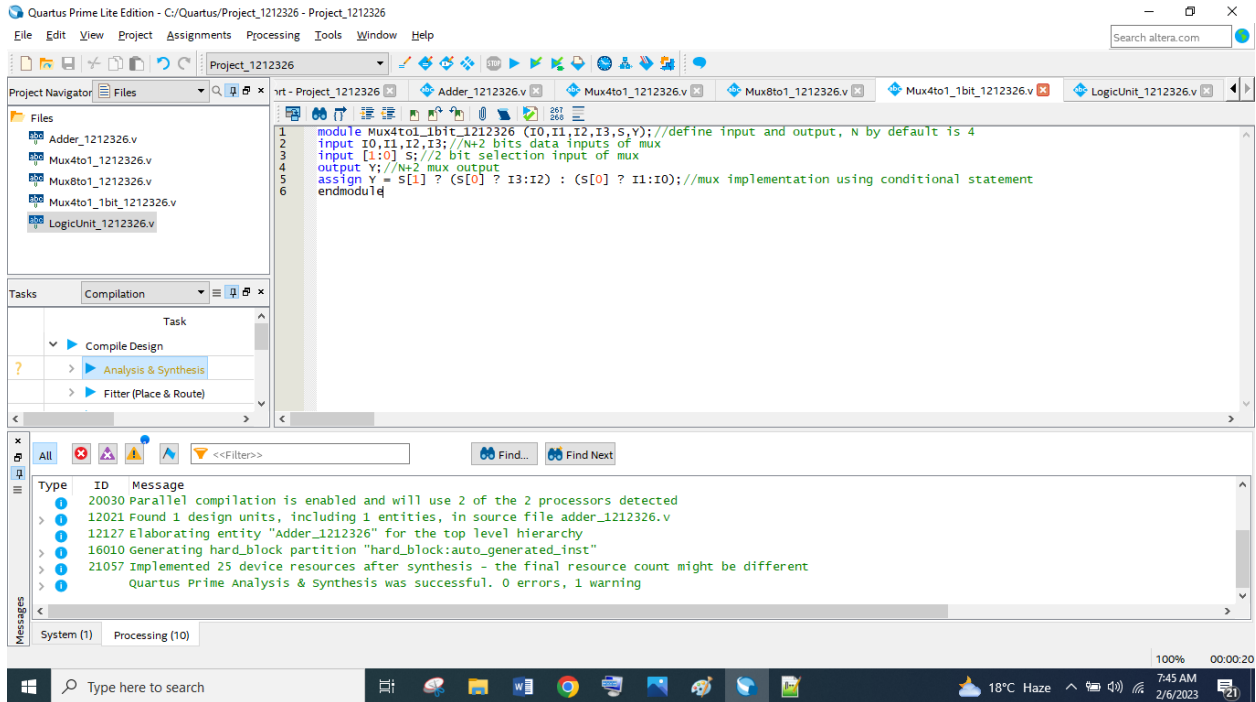


Figure 5. 1-Bit 4-to-1 Multiplexer Module Verilog Code

This module takes 4 data inputs I0, I1, I2, I3 of 1-bits and one 2-bit select line input and based on the select line input 1-bit output is connected with respective input data line.

Logic Unit Code

```
module LogicUnit_1212326 # (parameter N=4) (X,Y,I_4,I_5,I_6,I_7); //define input and output, N by default is 4
input [N-1:0] X,Y; //N bits inputs
output [N-1:0] I_4,I_5,I_6,I_7; //N bits outputs of logic Unit
assign I_4 = ~(X&Y); //Bitwise NAND Gate
assign I_5 = ~(X); //Bitwise NOT Gate
assign I_6 = ~(X|Y); //Bitwise NOR Gate
assign I_7 = X^Y; //Bitwise XOR Gate
endmodule
```

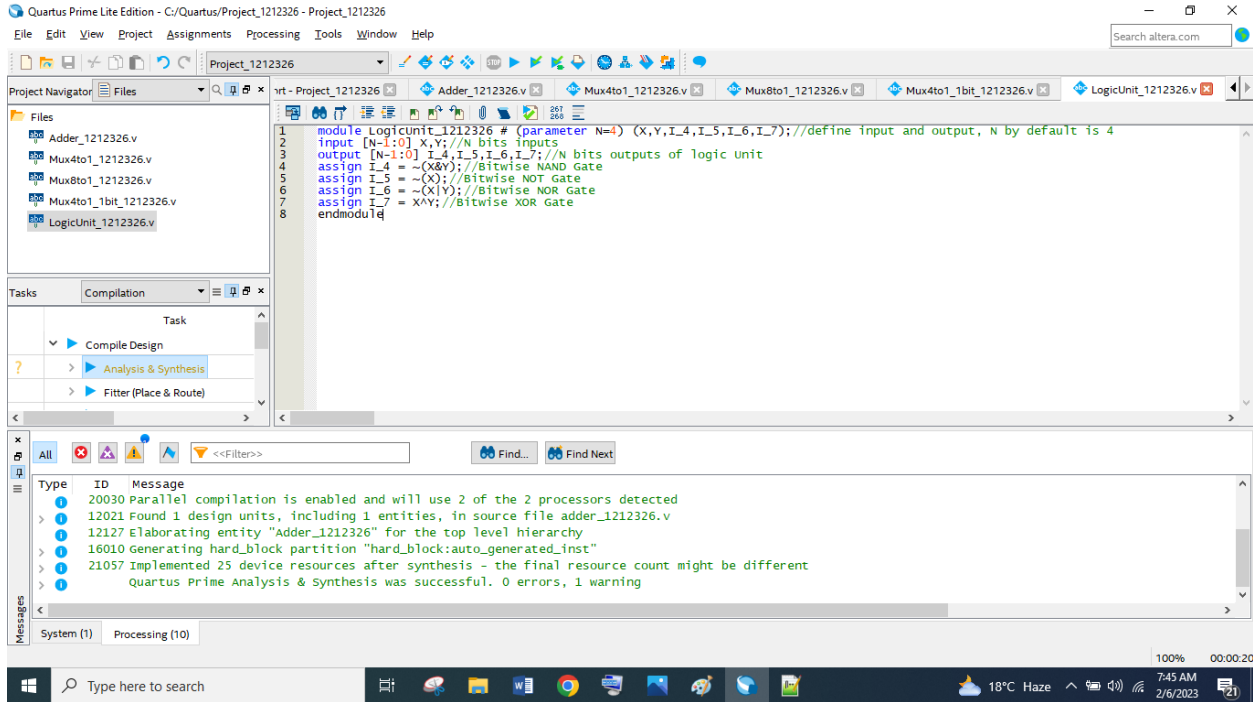


Figure 6. N-Bit Logic Unit Module Verilog Code

This module takes the N bit X and Y inputs and gives the result of bitwise NAND, NOT, NOR and XOR gate at the N bit output I_4, I_5, I_6, I_7 respectively.

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

ALU Code

```
module ALU_1212326 # (parameter N=4) (X,Y,C,O); //define input and output, N by default is 4
input [N-1:0] X,Y; //N bits inputs
input [2:0] C; //3 bit C input for different operation selection
output [N+1:0] O; //N+2 bits output
wire [N+1:0] A,B,S; //N+2 bits internal variable
wire Cin; //internal variable
wire [N-1:0] I_4,I_5,I_6,I_7; //N bits internal variable
```

```

//multiplexer module instantiation for Adder input A
Mux4to1_1212326 #(.N(N)) Mux4to1_X(.IO({X[N-1],X[N-1],X}),
.I1({X[N-1],X[N-1],X}),.I2({X[N-
1],X[N-1],X[N-1],X[N-1:1]}),
.I3({X[N-1],X[N-
1],X}),.S(C[1:0]),Y(A));
//multiplexer module instantiation for Adder input B
Mux4to1_1212326 #(.N(N)) Mux4to1_Y(.IO({Y[N-1],Y[N-1],Y}),
.I1({Y[N-1],Y[N-1],Y}),.I2({Y[N-
1],Y[N-1],Y}),
.I3(~({Y[N-1],Y[N-1],Y[N-1],Y[N-
1:1]})),.S(C[1:0]),Y(B));
//multiplexer module instantiation for Adder input Cin
Mux4to1_1bit_1212326 Mux4to1_Cin (.IO(1'b0),.I1(1'b0),.I2(1'b0),.I3(1'b1),.S(C[1:0]),Y(Cin));
//Adder module instantiation
Adder_1212326 #(.N(N)) Adder(.A(A),.B(B),.Cin(Cin),.Sum(S));
//Logic Unit Module instantiations
LogicUnit_1212326 #(.N(N)) logicUnit(.X(X),.Y(Y),.I_4(I_4),.I_5(I_5),.I_6(I_6),.I_7(I_7));
//Multiplexer module instantiation for output O
Mux8to1_1212326 #(.N(N)) MUX8to1_O(.IO({S[N+1],S[N+1:1]}),.I1({S[N:0],1'b0}),
.I2(S),.I3(S),
.I4({I_4[N-1],I_4[N-
1],I_4}),.I5({I_5[N-1],I_5[N-1],I_5}),
.I6({I_6[N-1],I_6[N-
1],I_6}),.I7({I_7[N-1],I_7[N-1],I_7}),.S(C),.Y(O));

```

endmodule

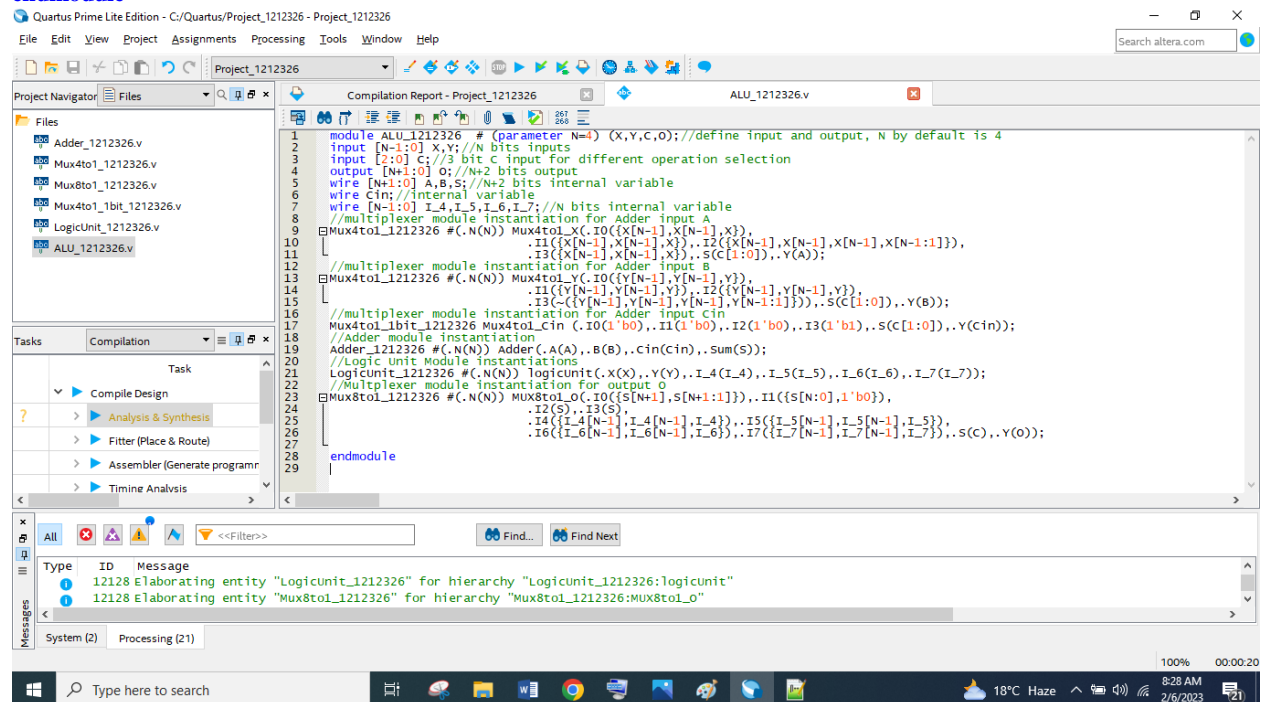


Figure 7. Structural ALU Module Verilog Code

In this module all sub module are instantiate and connected as according to the diagram that are mentioned in the part b. This is structural main module code of ALU. In this module N value is by default 4 and it can be change to any value.

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, student ID is 1212326, then X, Y, and C values for the three test cases as follows:

Table 1. Structural Test Case Table

Test	X	Y	C	O
1	$X1 = 6$	$Y1 = 2$	$C1 = 3$	$6 - (2/2) = 5$
2	$X2 = 2$	$Y2 = 1$	$C2 = 2$	$(2/2) + 1 = 2$
3	$X3 = -6$	$Y3 = -2$	$C3 = 2$	$(-6/2) + (-2) = -5$

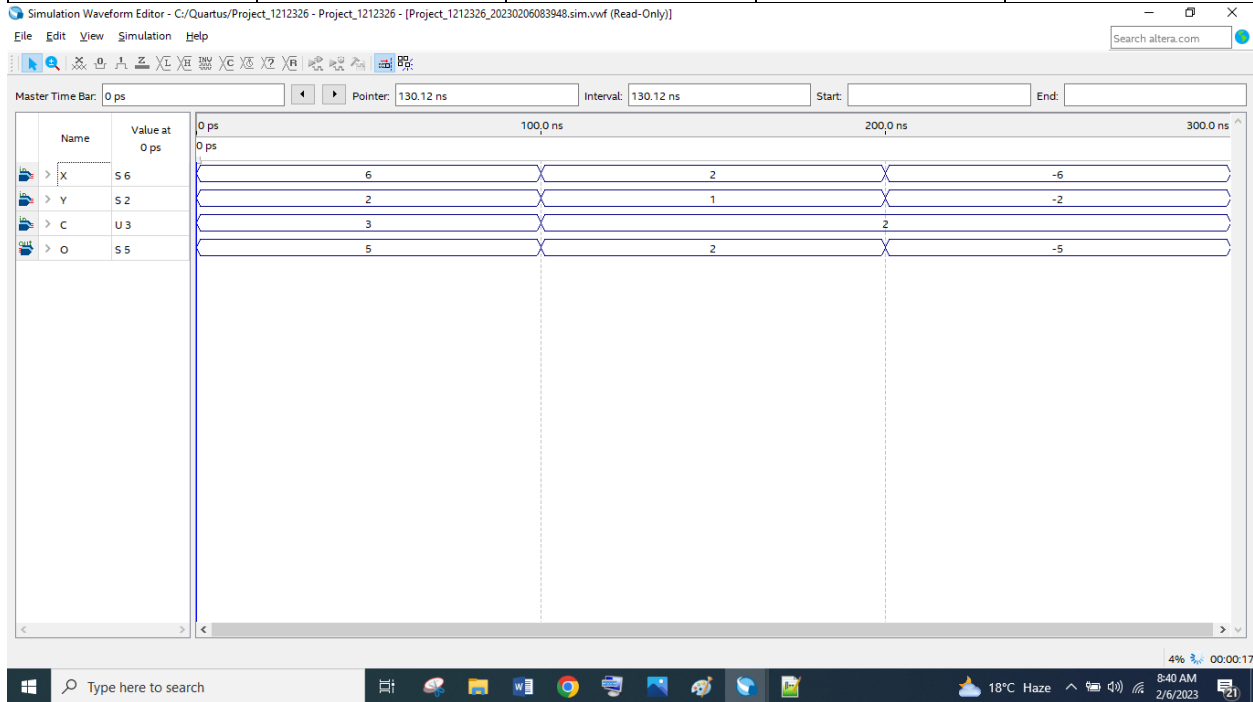


Figure 8. Structural ALU Module Simulation

This is the simulation of structural ALU code, in this simulation first 100 ns simulation is for X-(Y/2), from 100 ns to 300 ns simulation is for (X/2)+Y. Result of this simulation is same as the table mentioned above.

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

ALU Behavioral Code

```

module alu_behavioral_1212326 # (parameter N=4) (X,Y,C,O);//define input and output, N by default is 4
input [N-1:0] X,Y;//N bits inputs
input [2:0] C;//3 bit C input for different operation selection
output reg [N+1:0] O;//N+2 bits output
reg [N:0] S0;//internal variable for addition
always @(*)
begin
case (C)
3'b000:
    begin
        S0 = X+Y; //X+Y
        O = {S0[N],S0[N],S0[N+1:1]}; //(X+Y)/2
    end
3'b001:
    O = {X+Y,1'b0}; //2*(X+Y)
3'b010:
    O = {X[N-1],X[N-1],X[N-1:1]} + {Y[N-1],Y}; //(X/2)+Y
3'b011:
    O = {X[N-1],X[N-1],X} - {Y[N-1],Y[N-1],Y[N-1:1]}; //X-(Y/2)
3'b100:
    O = ~({X[N-1],X[N-1],X} & {Y[N-1],Y[N-1],Y}); //X NAND Y
3'b101:
    O = ~({X[N-1],X[N-1],X}); //NOT(X)
3'b110:
    O = ~({X[N-1],X[N-1],X} | {Y[N-1],Y[N-1],Y}); //X NOR Y
3'b111:
    O = {X[N-1],X[N-1],X} ^ {Y[N-1],Y[N-1],Y}; //X XOR Y
endcase
end

endmodule

```

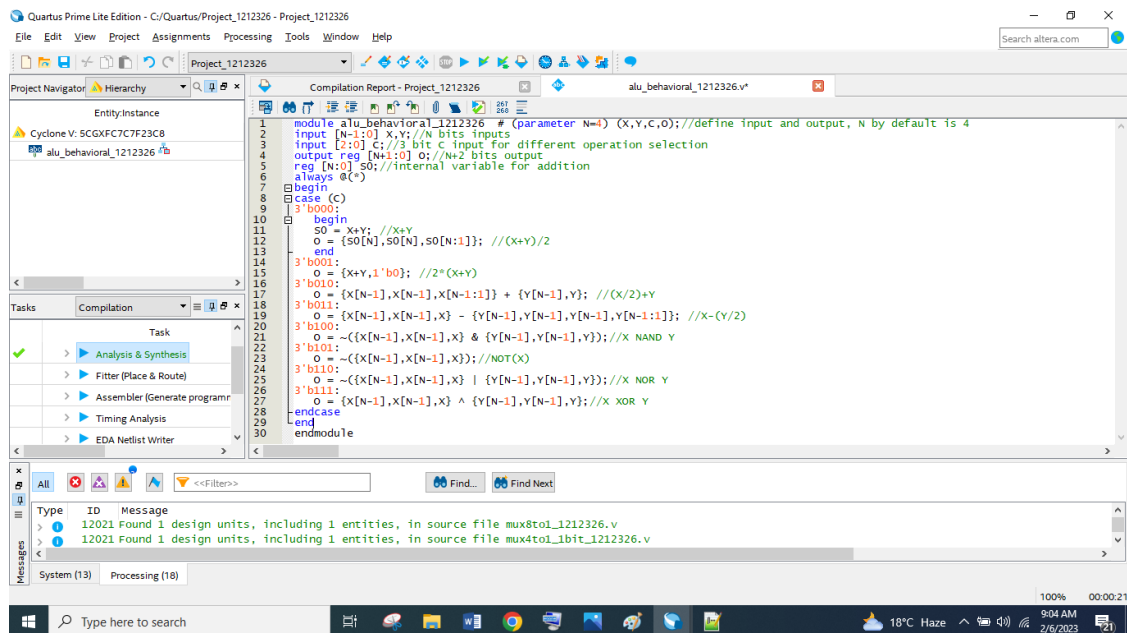


Figure 9. Behavioral ALU Module Verilog Code

This is behavioral code for the ALU, in this code a case statement is implemented which is performing the different operation of the ALU based on the value of C selection input. In each case different operation of ALU is defined and it will give output based on the changing of any input signal.

- 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 1212326, then X, Y, and C values for the three test cases as follows:**

Table 2. Behavioral Test Case Table

Test	X	Y	C	O
1	X1 = 6	Y1 = 2	C1 = 3	$6 - (2/2) = 5$
2	X2 = 2	Y2 = 1	C2 = 2	$(2/2) + 1 = 2$
3	X3 = -6	Y3 = -2	C3 = 2	$(-6/2) + (-2) = -5$

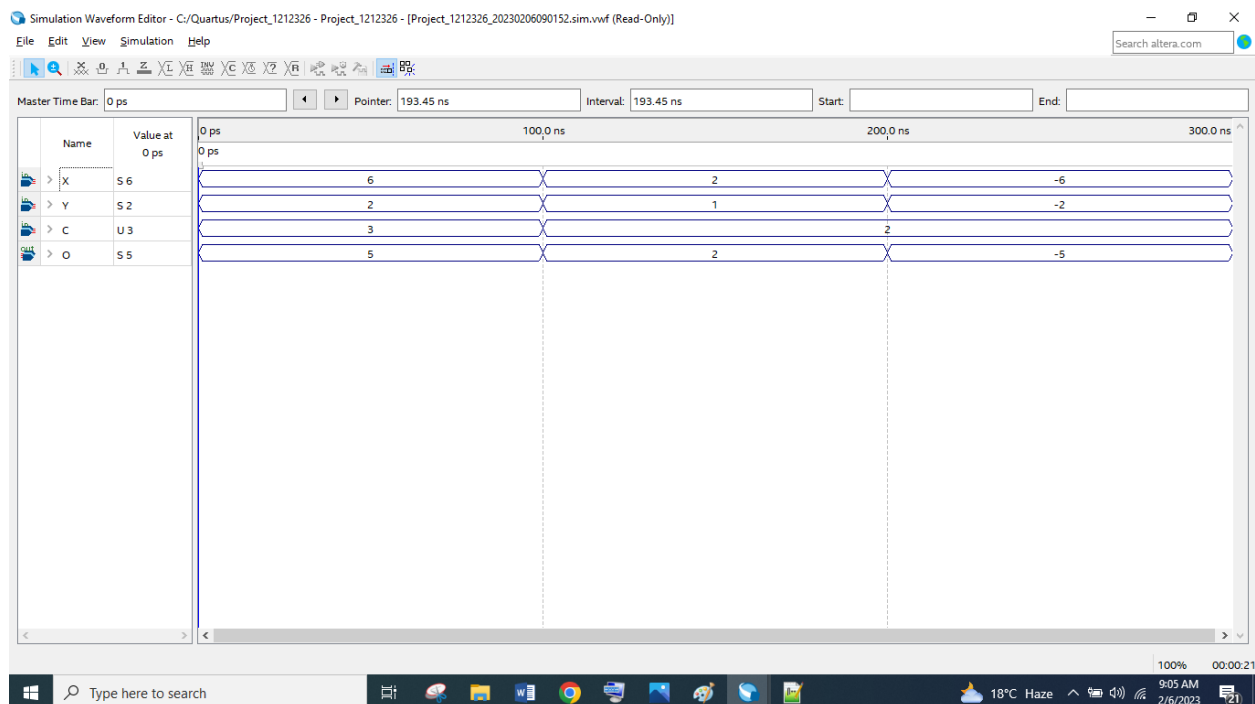


Figure 10. Behavioral ALU Module Simulation

This is the simulation of behavioral ALU code, in this simulation first 100 ns simulation is for $X - (Y/2)$, from 100 ns to 300 ns simulation is for $(X/2) + Y$. Result of this simulation is same as the table mentioned above.