Department of Electrical and Computer Engineering

Digital Systems – ENCS2340

HDL Homework

~~~~~~

Student's Name: **Tariq Odeh**

Student's No: **1190699**

Sec: **2**

Date: 27 -12-2020

Instructor: Dr.Mohammed Hussein

~~~~~~

INDEX:

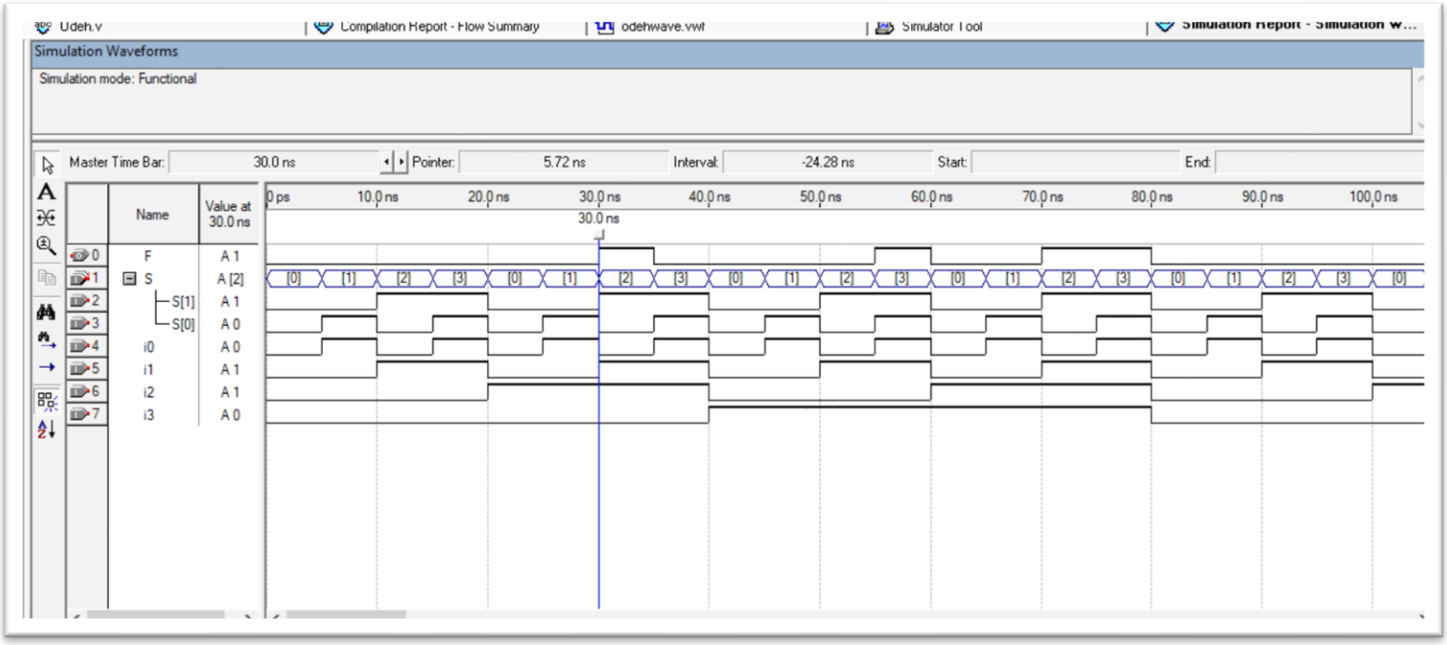
Question 1:

"mux4*1"

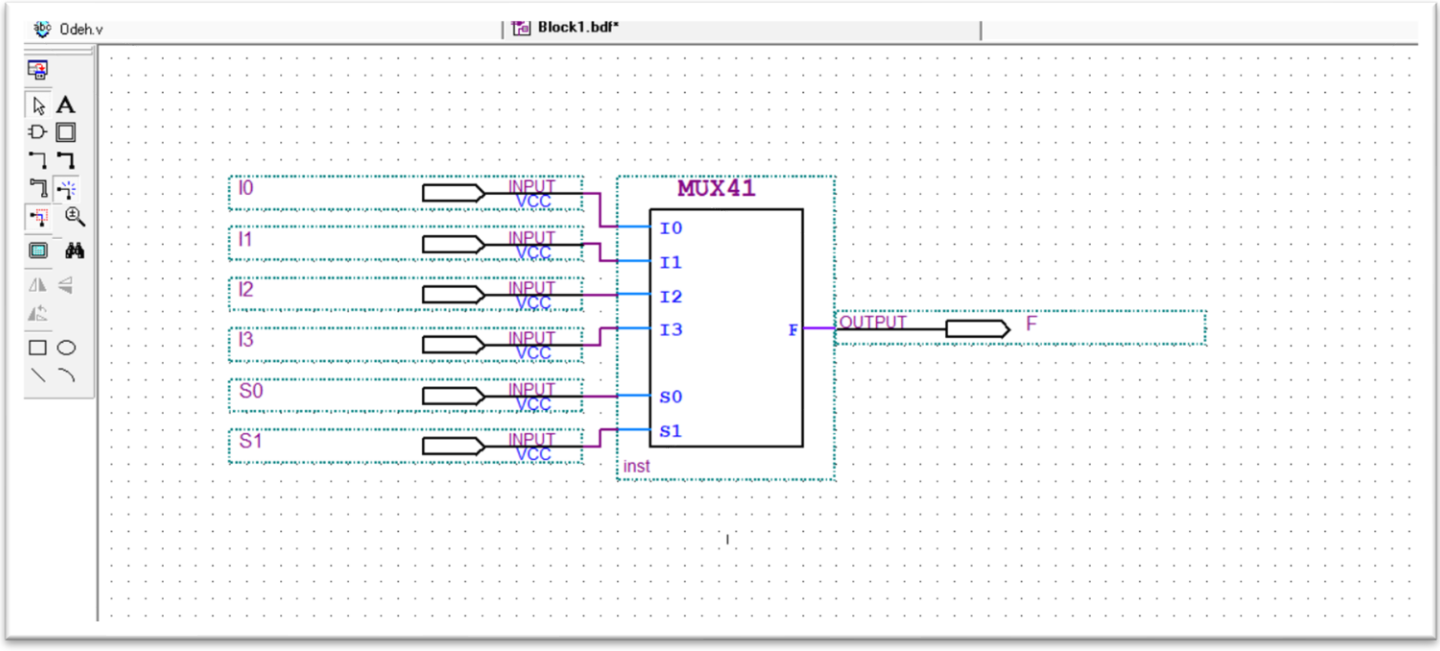
1- (mux4*1 Verilog Code):

```
module Odeh ( i0 , i1 , i2 , i3 , S , F );
 input i0 , i1 , i2 , i3;
 input [1:0] S;
 output F;
 reg    F;

 always @( i0 or i1 or i2 or i3 or S )
 begin
                case(S)
                        2'b00: F = i0;
                        2'b01: F = i1;
                        2'b10: F = i2;
                        2'b11: F = i3;
                        default: F = 1'b0;
                endcase
        end
endmodule
```

2- (mux4*1 Simulation):



3- (mux4*1 Block diagram):



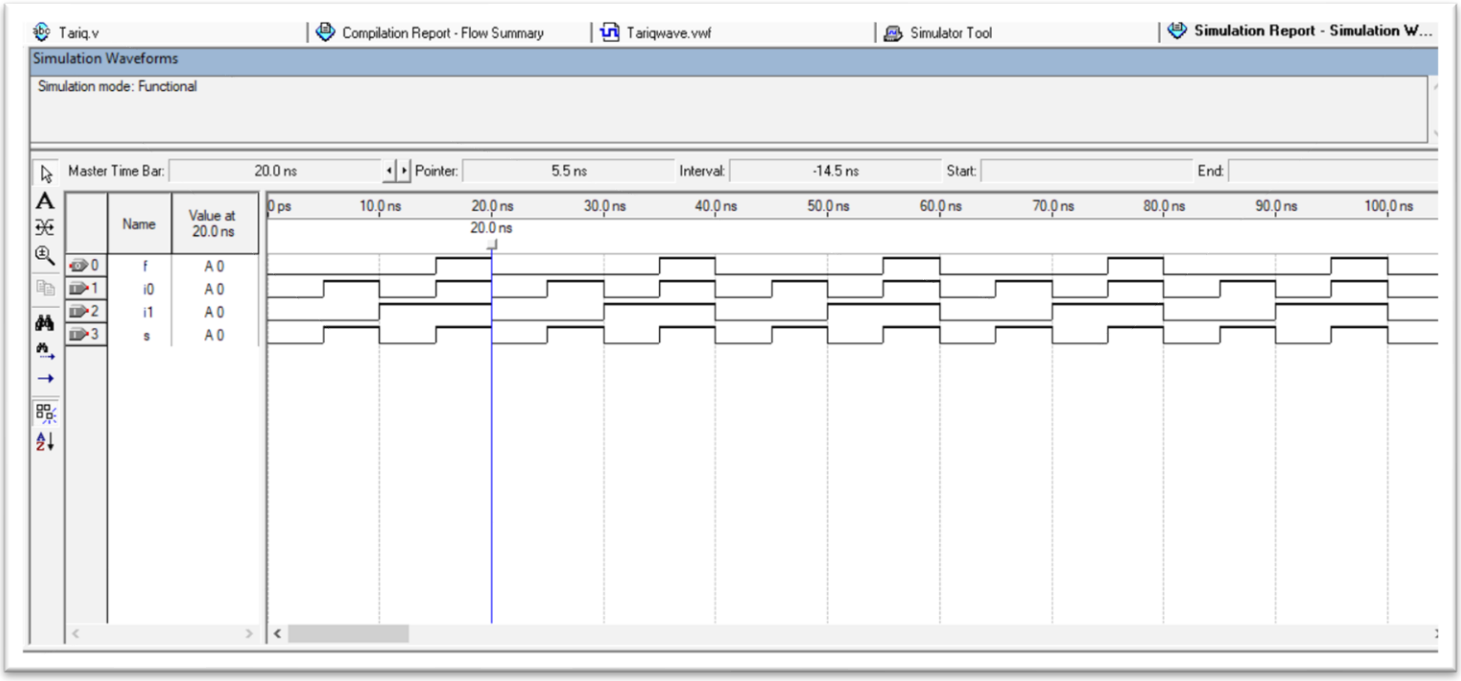
Remark: I used the behavioral (always) language to write the MUX 4*1 code

"mux2*1"

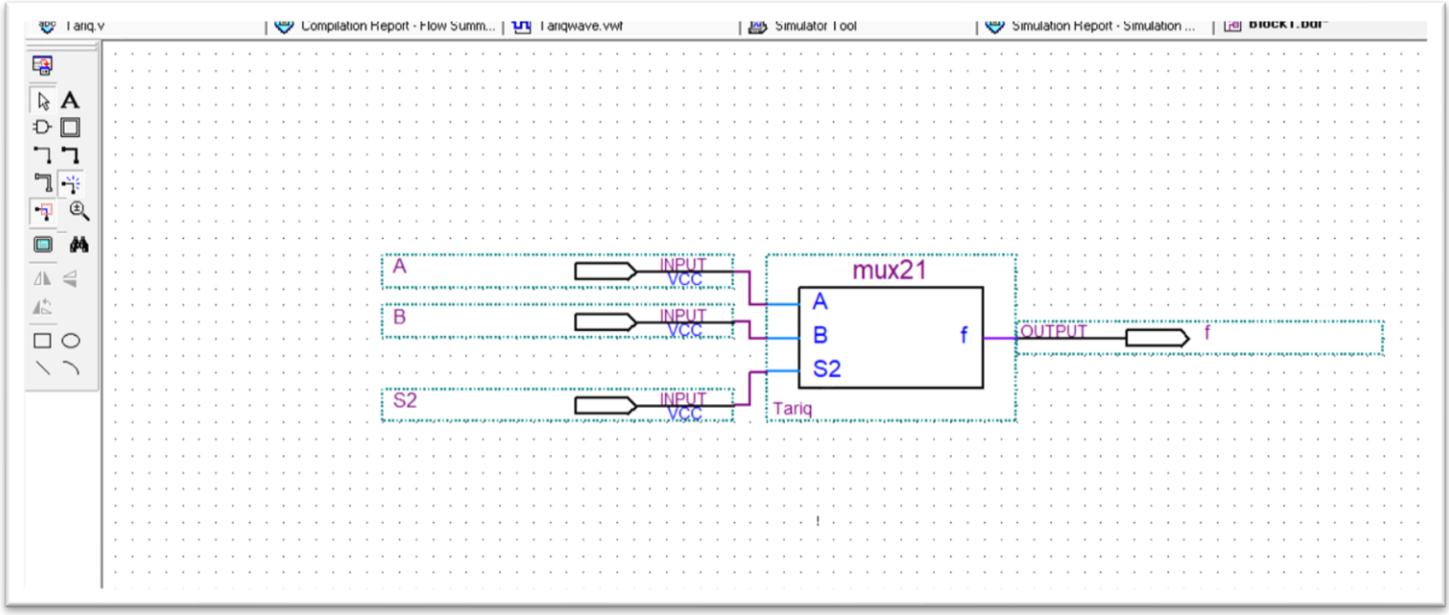
1- (mux2*1 Verilog Code):

```
module Tariq(i0 , i1 , s , f);

input i0 , i1 , s;

output f;

reg f;

always @(i0 or i1 or s)
  begin
   if(s == 0)
     f = i0;
   else
     f = i1;
  end

endmodule
```

2- (mux2*1 Simulation):



3- (mux2*1 Block diagram):



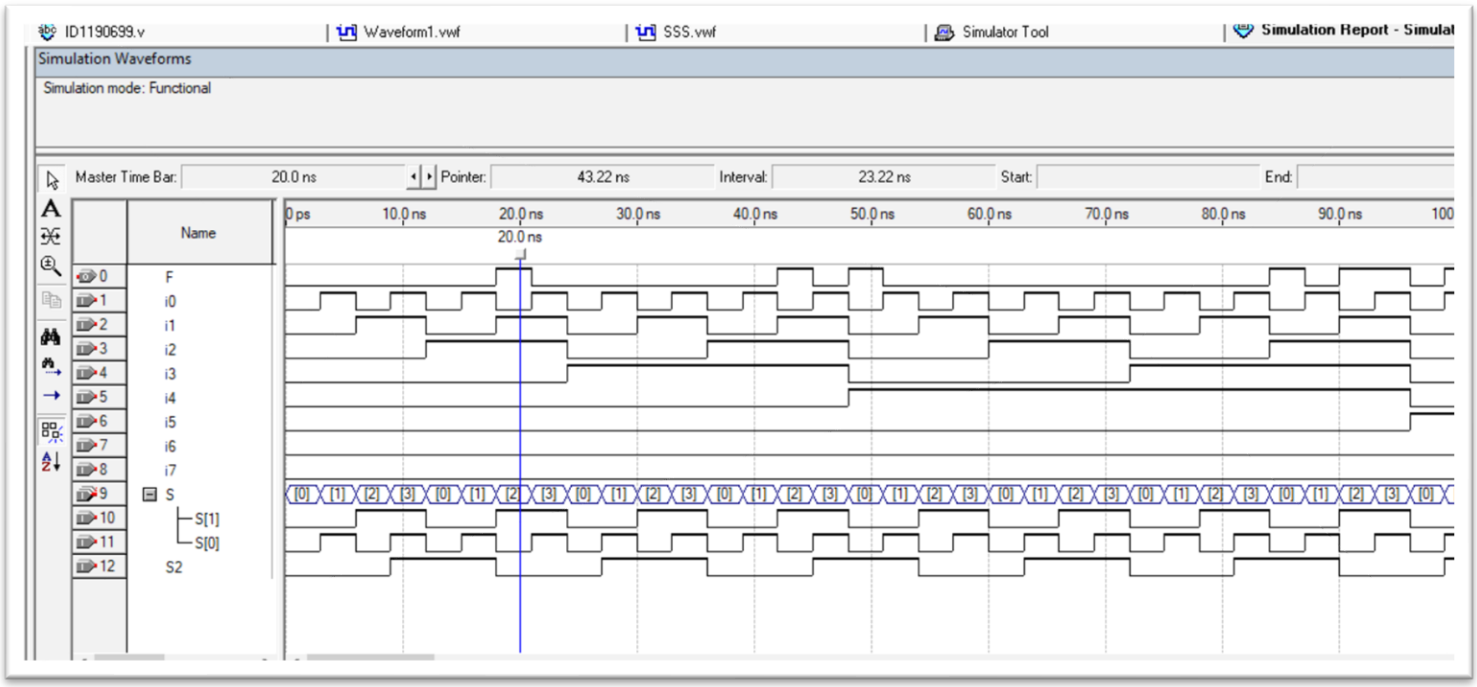
Remark: I used the behavioral (always) language to write the MUX 2*1 code

"THE SYSTRM"

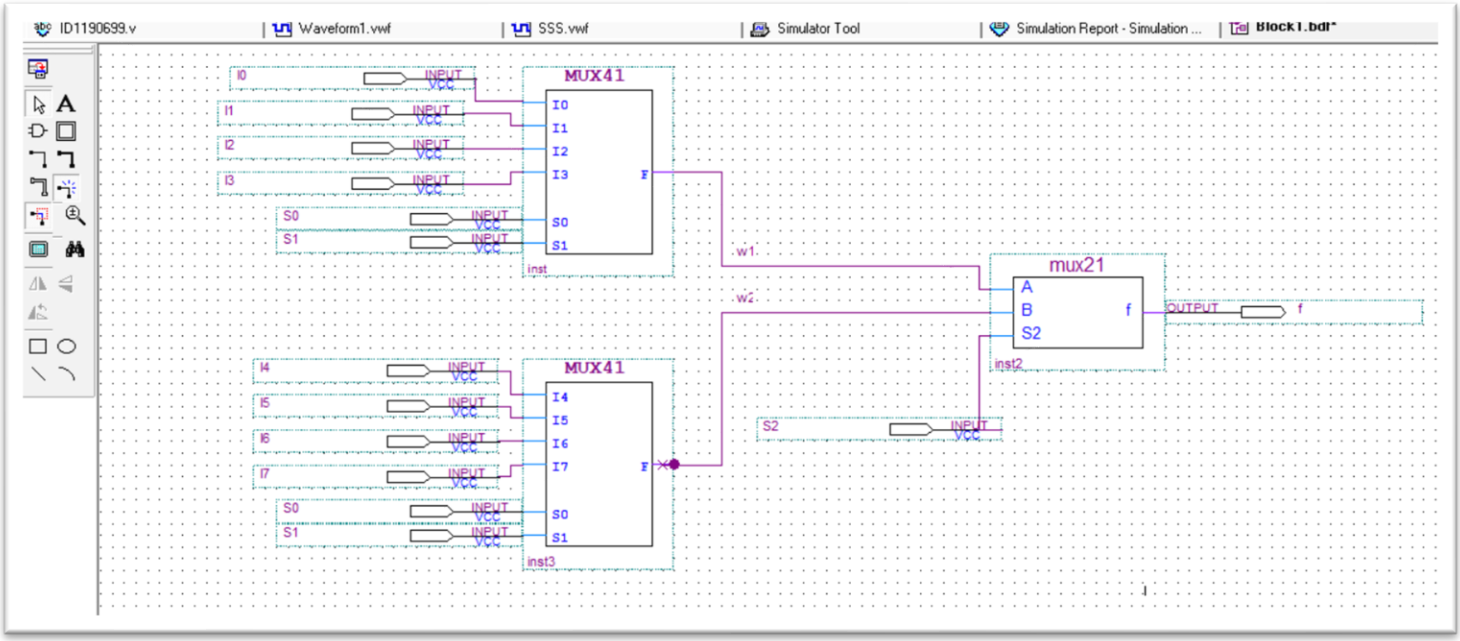
1- (The System Verilog Code):

```
module ID1190699 (i0,i1,i2,i3,i4,i5,i6,i7,S,S2,F);
input i0,i1,i2,i3,i4,i5,i6,i7,S2;
input [1:0]S;
output F;
wire w1,w2;

Odeh odeh1(i0,i1,i2,i3,S,w1),odeh2(i4,i5,i6,i7,S,w2);

Tariq tariq1(w1,w2,S2,F);

endmodule
```

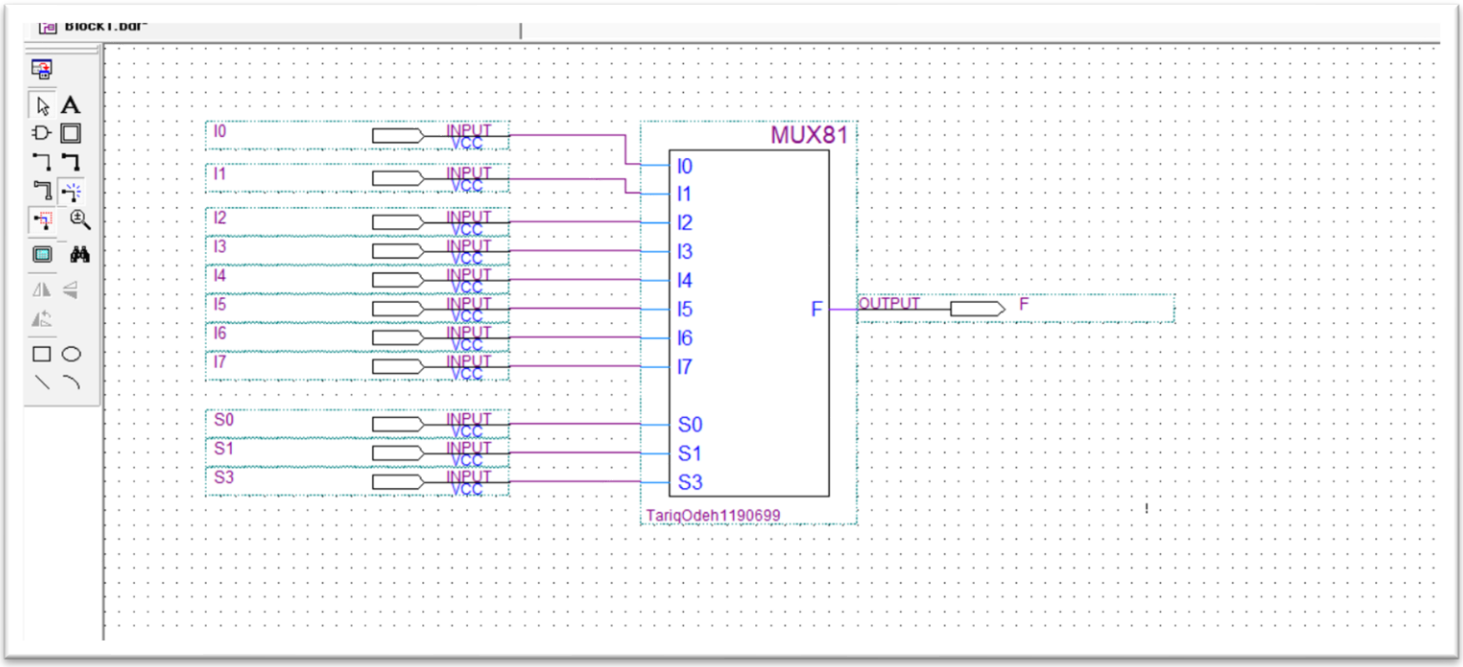
2- (The System Simulation):

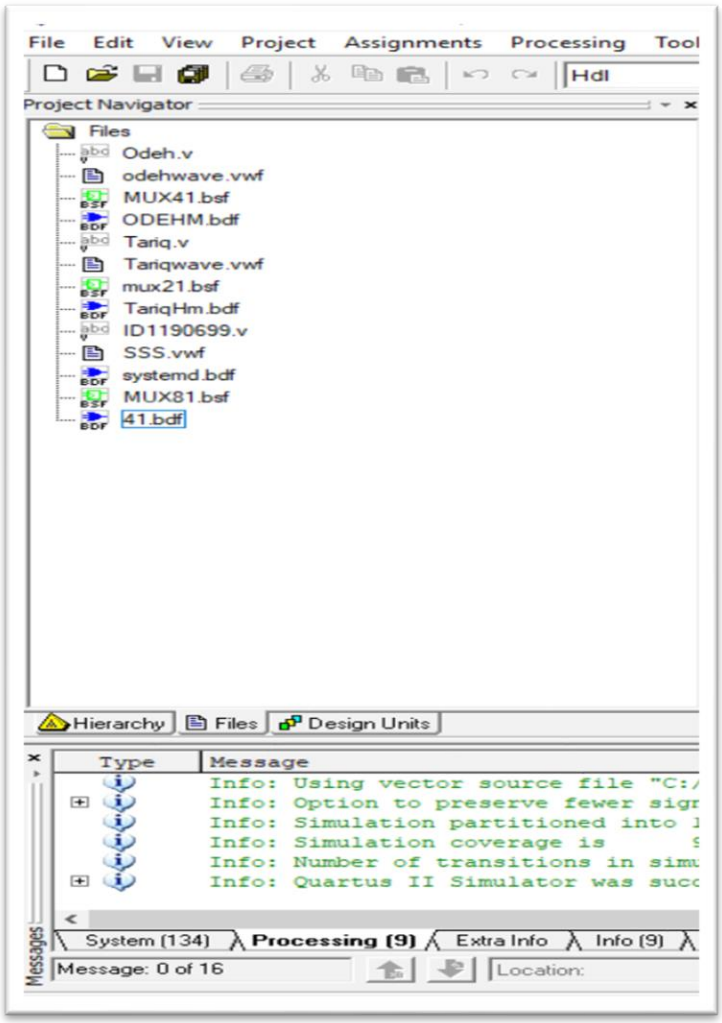


3- (The System Block diagram):



Remark: I used the structurally to write the system code


I0
I1
I2
I3
I4
I5
I6
I7
S0
S1
S3
INPUT
VCC
MUX81
F
OUTPUT
TariqOdeh1190699

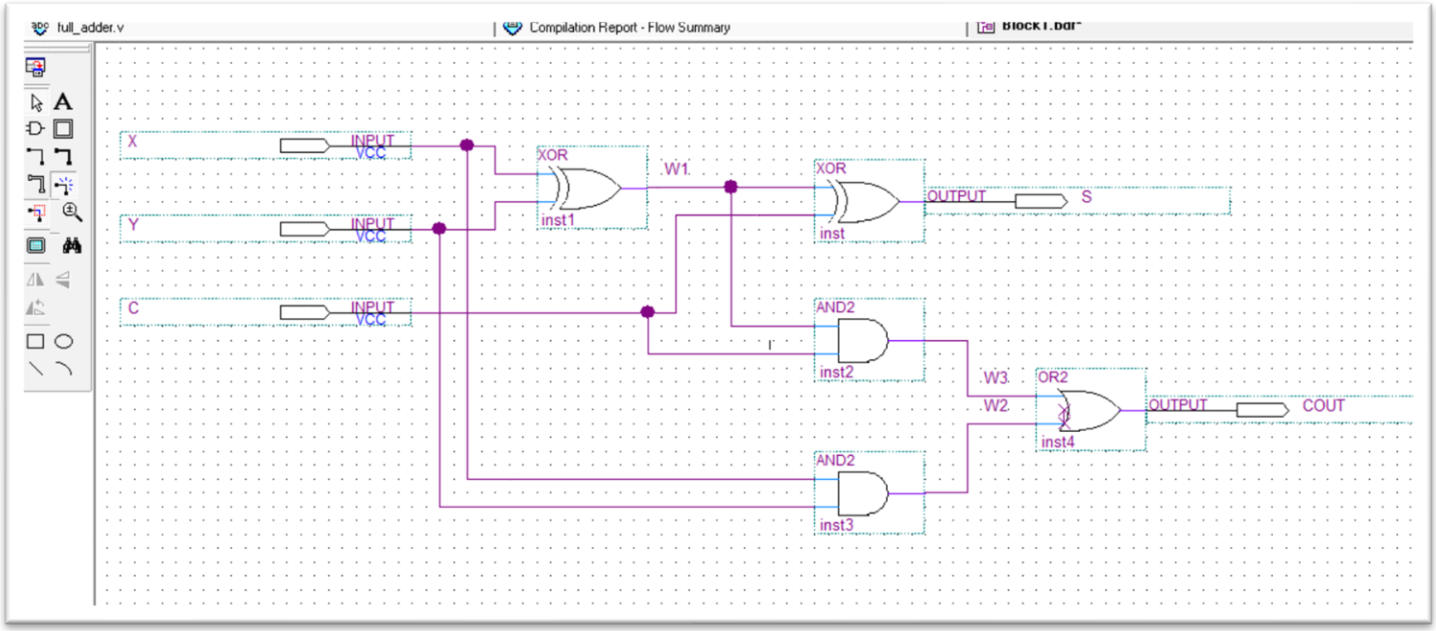

File Edit View Project Assignments Processing Tool
Project Navigator
Files
Odeh.v
odehwave.vwf
MUX41.bsf
ODEHM.bdf
Tariq.v
Tariqwave.vwf
mux21.bsf
TariqHm.bdf
ID1190699.v
SSS.vwf
systemd.bdf
MUX81.bsf
41.bdf
Hierarchy
Files
Design Units
Type
Message
Info: Using vector source file "C:/
Info: Option to preserve fewer sign
Info: Simulation partitioned into
Info: Simulation coverage is
Info: Number of transitions in simu
Info: Quartus II Simulator was succ
System (134)
Processing (9)
Extra Info
Info (9)
Message: 0 of 16
Location:
Messages

Question 2:

"FULL ADDER CODE"

```
module full_adder(sum , cout ,   X , Y , C);


        input X,Y,C;
        output sum , cout;
        wire w1,w2,w3;


        xor x1(w1 , X , Y0);
        xor x2(sum , C , w1);
        and a1(w2 , X , Y);
        and a2(w3 , w1 , C);
        or  o1(cout , w2 , w3);


endmodule
```

(FULL ADDER Block diagram):



#1 FIRST CODE: (000 // Addition: Z=X+Y)

THE VERILOG CODE:

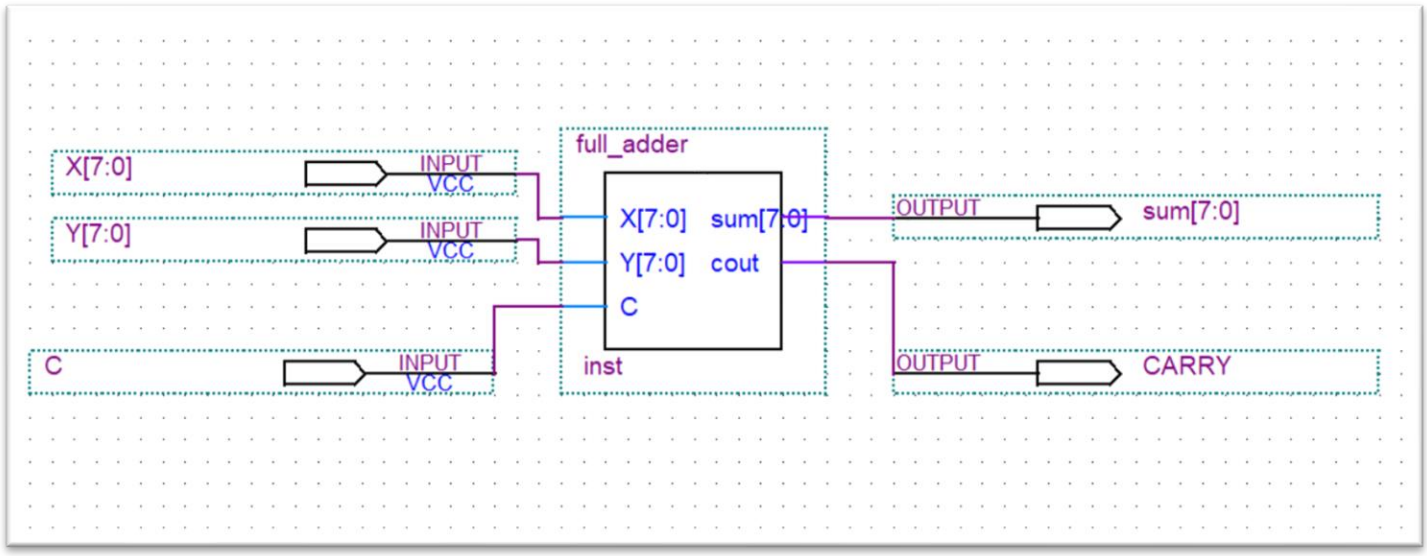
```
module full_adder8bit(Sum , Cout , X , Y , C);
        input C;
        input [7:0] X,Y;
        output Cout;
        output [7:0] Sum;

        wire w0,w1,w2,w3,w4,w5,w6;

        full_adder fad0(Sum[0] , w0 , X[0] , Y[0] , C);
        full_adder fad1(Sum[1] , w1 , X[1] , Y[1] , w0);
        full_adder fad2(Sum[2] , w2 , X[2] , Y[5] , w1);
        full_adder fad3(Sum[3] , w3 , X[3] , Y[3] , w2);
        full_adder fad4(Sum[4] , w4 , X[4] , Y[4] , w3);
        full_adder fad5(Sum[5] , w5 , X[5] , Y[5] , w4);
        full_adder fad6(Sum[6] , w6 , X[6] , Y[6] , w5);
        full_adder fad7(Sum[7] , Cout , X[7] , Y[7] , w6);

endmodule
```

EXPLAINING THE CODE

In this code we needed 3 inputs (X , Y , Cin) and 2 outputs (Sum , Cout) this code make the 8 bit addition for 3 inputs by using the full adder as we see in first the code adding X to Y and of course we have Cin , so the sum of X and Y will be shown in Sum and the carry will be shown in Cout.

SIMULATING OF THE CODE:

Block diagram:



#2 SECOND CODE: (001 // SUBTRACTION: Z=X-Y)

THE VERILOG CODE:

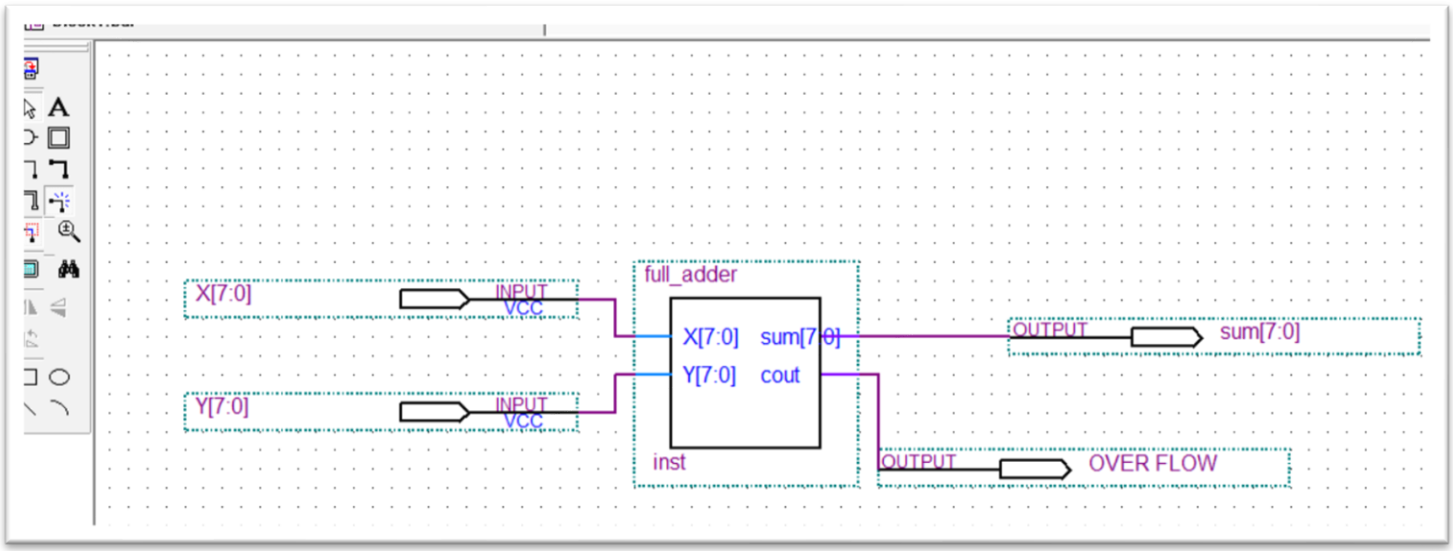
```
module substractor (Sum , Cout , X , Y);
        input [7:0] X,Y;
        output Cout;
        output [7:0]Sum;

        wire [7:0] w;

        xor x1(w[0] , 1 , Y[0]);
        xor x2(w[1] , 1 , Y[1]);
        xor x3(w[2] , 1 , Y[2]);
        xor x4(w[3] , 1 , Y[3]);
        xor x5(w[4] , 1 , Y[4]);
        xor x6(w[5] , 1 , Y[5]);
        xor x7(w[6] , 1 , Y[6]);
        xor x8(w[7] , 1 , Y[7]);

        full_adder8bit(Sum , Cout , X , w ,1);

endmodule
```

EXPLAINING THE CODE

In this code the inputs are (X , Y) and both they are 8 bit inputs this code do the subtraction of two numbers. Also here we need 8 xor gates they can calculate the 2'complement of Y and calculating the Cout.

SIMULATING OF THE CODE:

Block diagram:



#3 THIRD CODE: (010 // REMINDER: Z=X%Y)

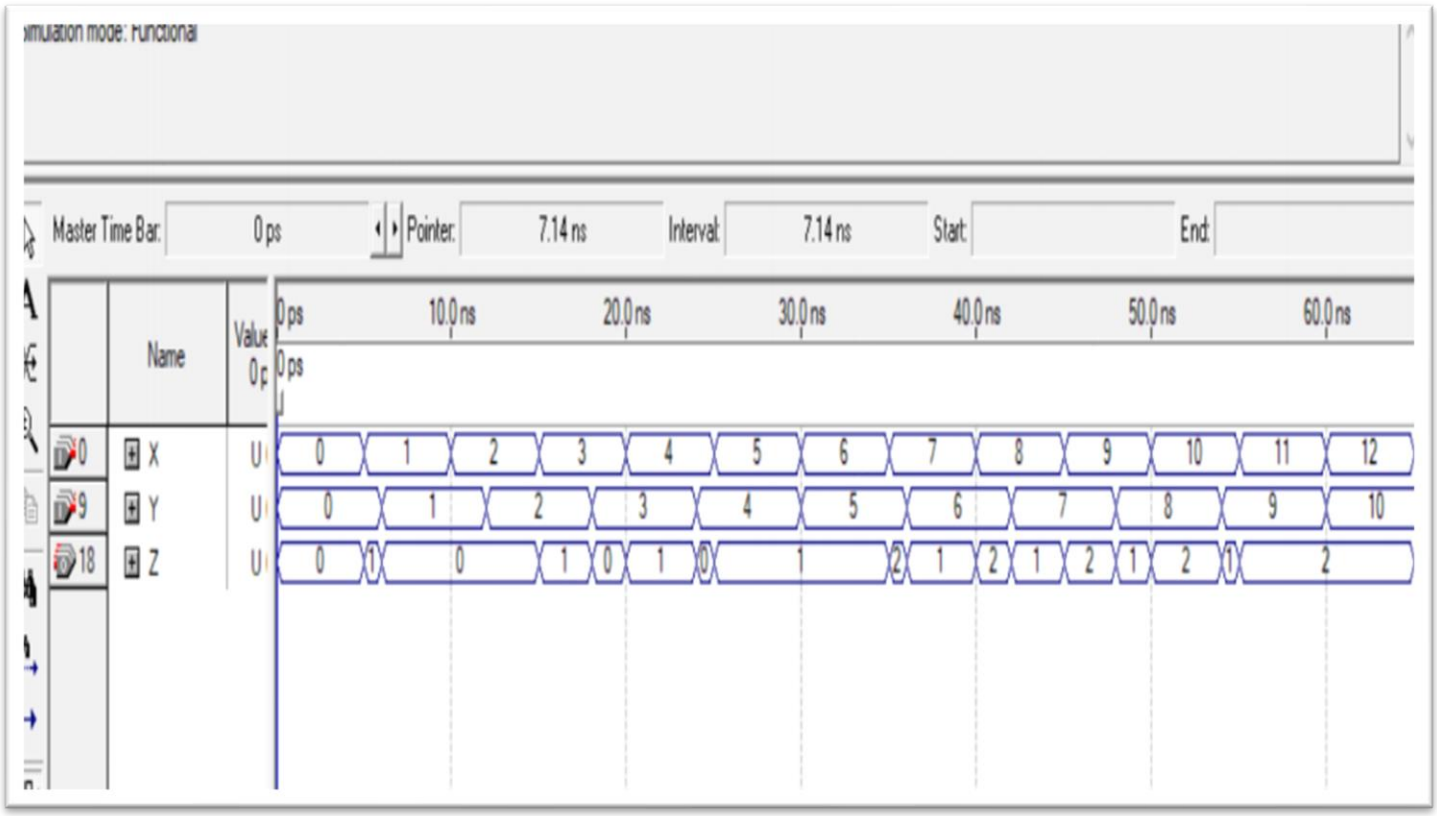
THE VERILOG CODE:

```
module reminder(Z , X , Y);
        input [7:0] X,Y;
        output [7:0] Z;

        reg [7:0]Z;

        always @(X , Y)
        begin

        {Z}= X % Y;

        end

endmodule
```

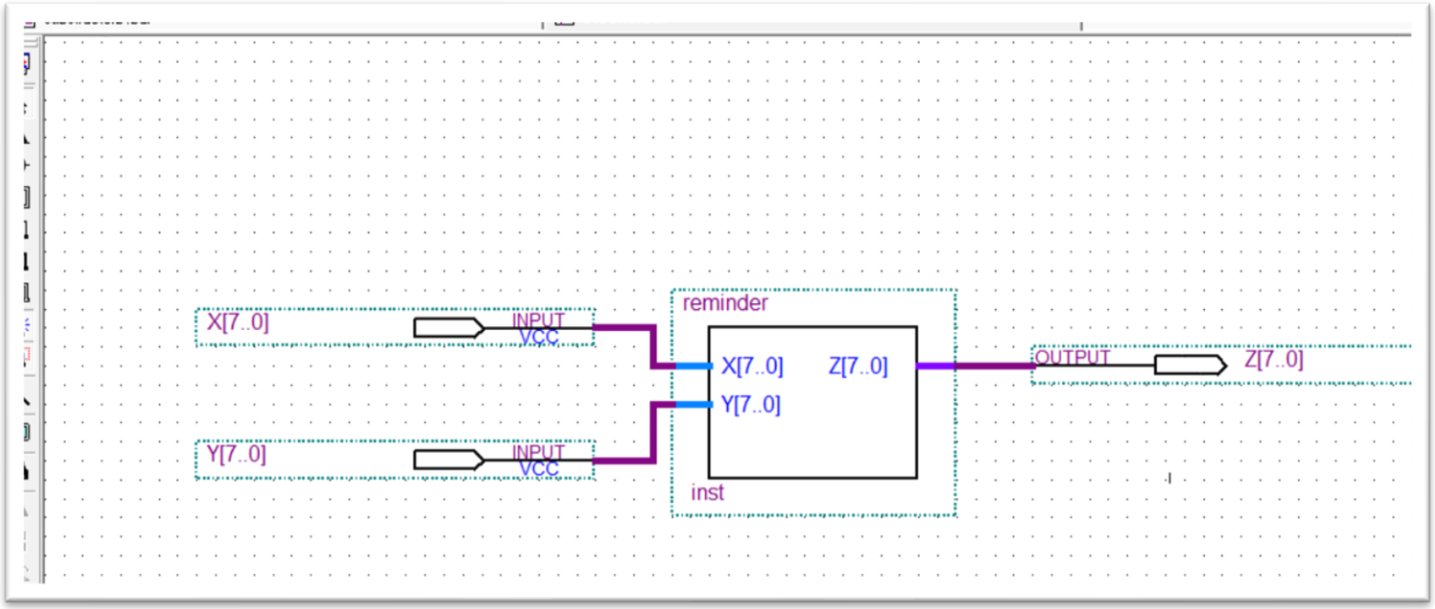
EXPLAINING THE CODE

This code is used to find the reminder for two numbers when we divided them so we have two inputs (X , Y) and the reminder will be shown in the output (Z) for example X= 7 , Y=5 so Z= 2 and that it is the answer of the reminder.

SIMULATING OF THE CODE:



Block diagram:



#4 FORTH CODE: (011//Bitwise AND: Z=X&Y)

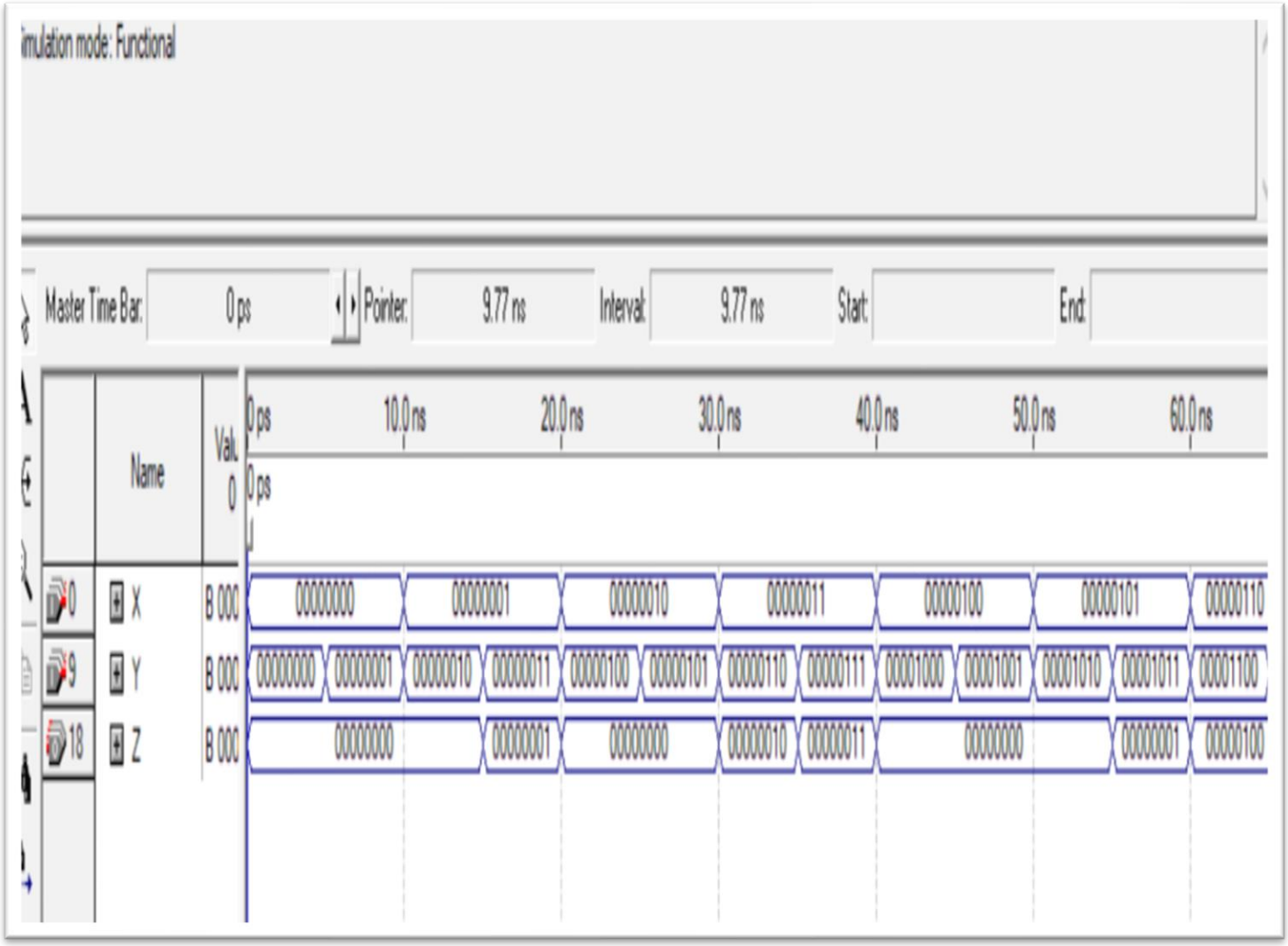
THE VERILOG CODE:

```
module Bitwise_AND(Z , X , Y);
        input [7:0] X,Y;
        output [7:0] Z;

        reg [7:0]Z;

        always @(X , Y)
        begin

        {Z}= X & Y;

        end

endmodule
```

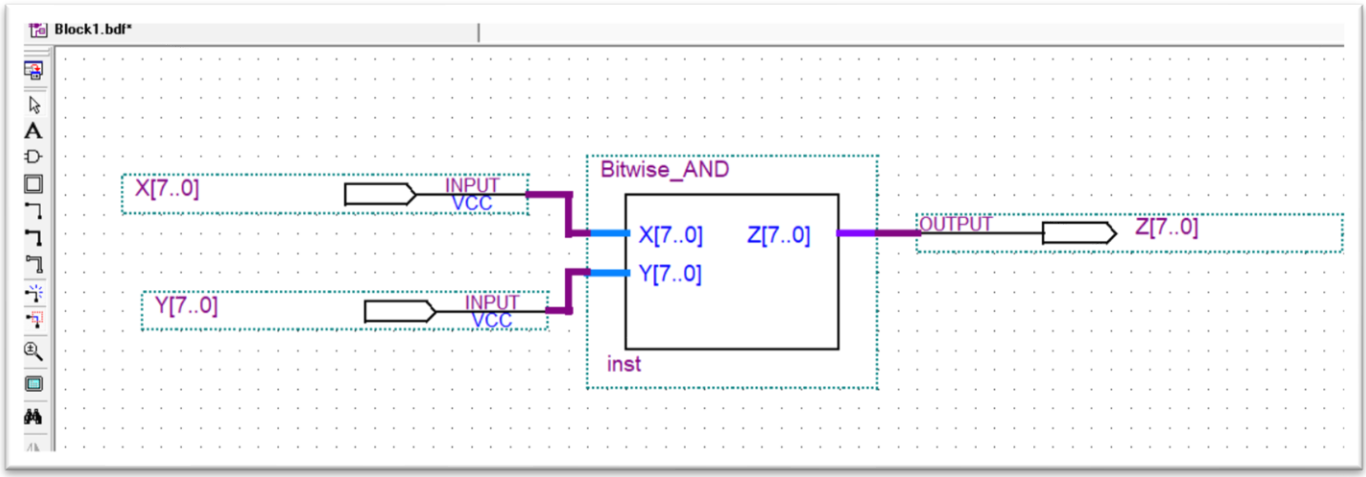
EXPLAINING THE CODE

This code is used like an AND gate this gate gives output (1) just when all inputs are (1) else the output is ZERO for example (010101) & (001100) = 000100. First input (x) = 010101 , (y) = 001100 >> so the output(z) = 000100 .

SIMULATING OF THE CODE:



Block diagram:



#5 FIFTH CODE: (100//Bitwise OR: Z=X|Y)

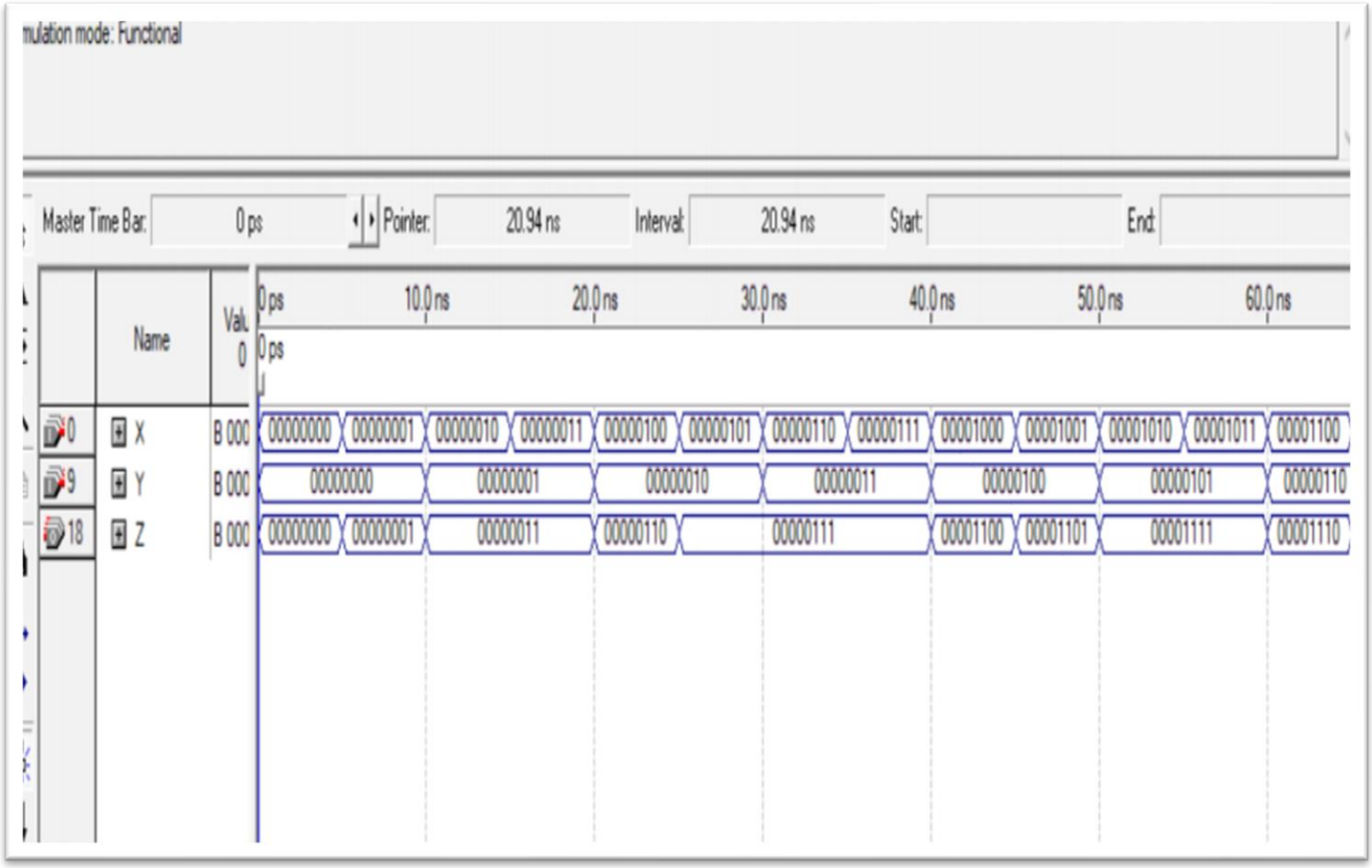
THE VERILOG CODE:

```
module Bitwise_OR(Z , X , Y);
        input [7:0] X,Y;
        output [7:0] Z;

        reg [7:0]Z;

        always @(X , Y)
        begin

        {Z}= X | Y;

        end

endmodule
```

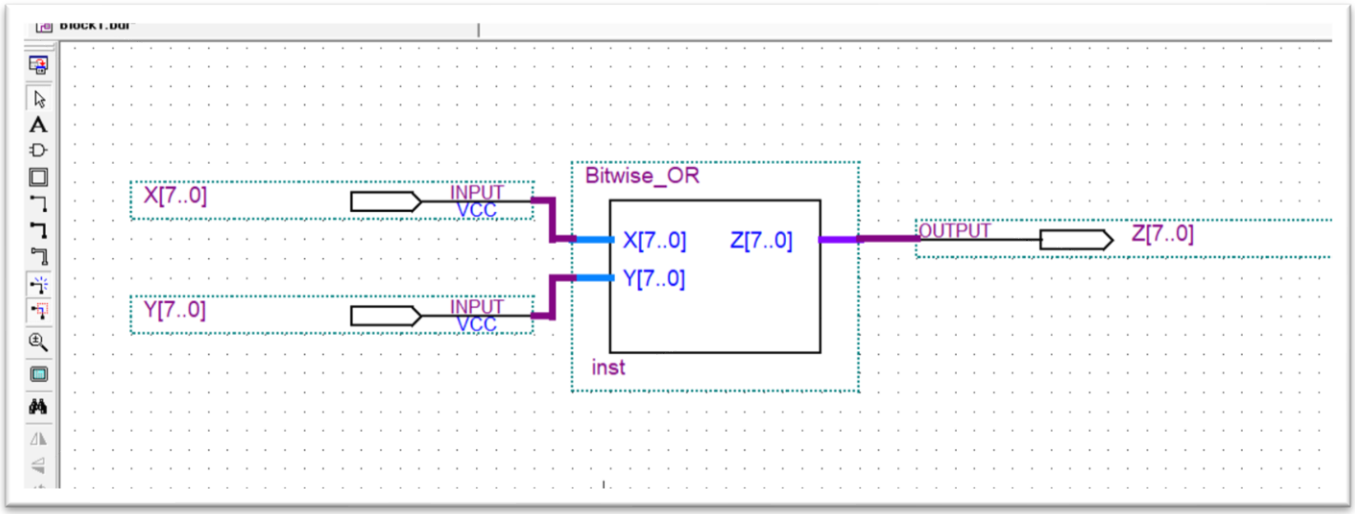
EXPLAINING THE CODE

This code is used like an OR gate this gate gives output (1) when at least one input is (1) else the output is ZERO for example (010101) | (001100) = 000100. First input (x) = 010101 , (y) = 001100 >> so the output(z) = 011101 .

SIMULATING OF THE CODE:



Block diagram:

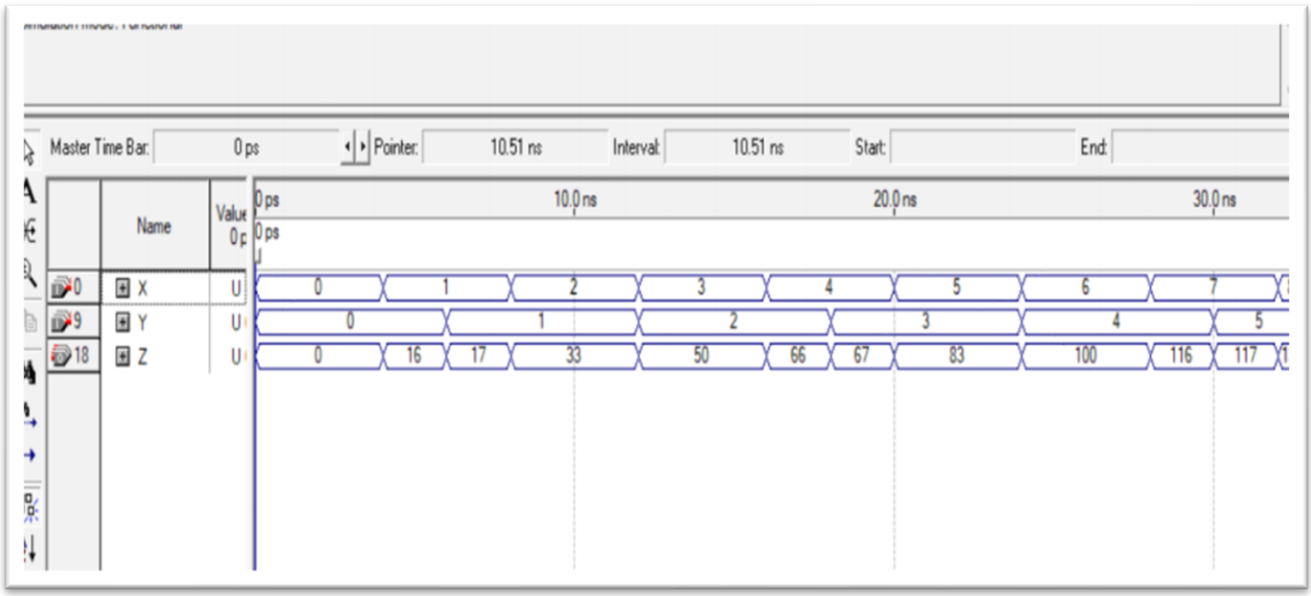


#6 SIXTH CODE: (101// Concatenate: Z={X[3:0] , Y[3:0]})

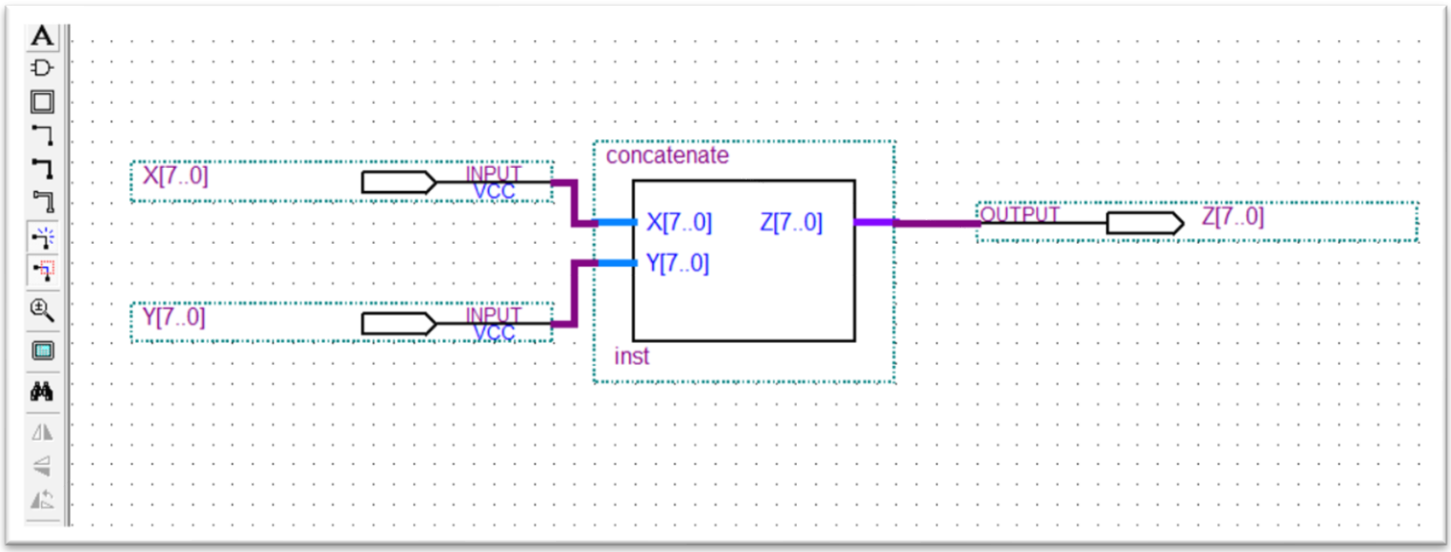
THE VERILOG CODE:

```
module concatenate(Z , X , Y);
        input [7:0] X,Y;
        output [7:0] Z;

        reg [7:0]Z;

        always @(X , Y)
        begin

        {Z}= {X[3:0] , Y[3:0]};

        end

endmodule
```

SIMULATING OF THE CODE:



Block diagram:



#7 SEVENTH CODE: (110// Equality: Zero=X==Y)

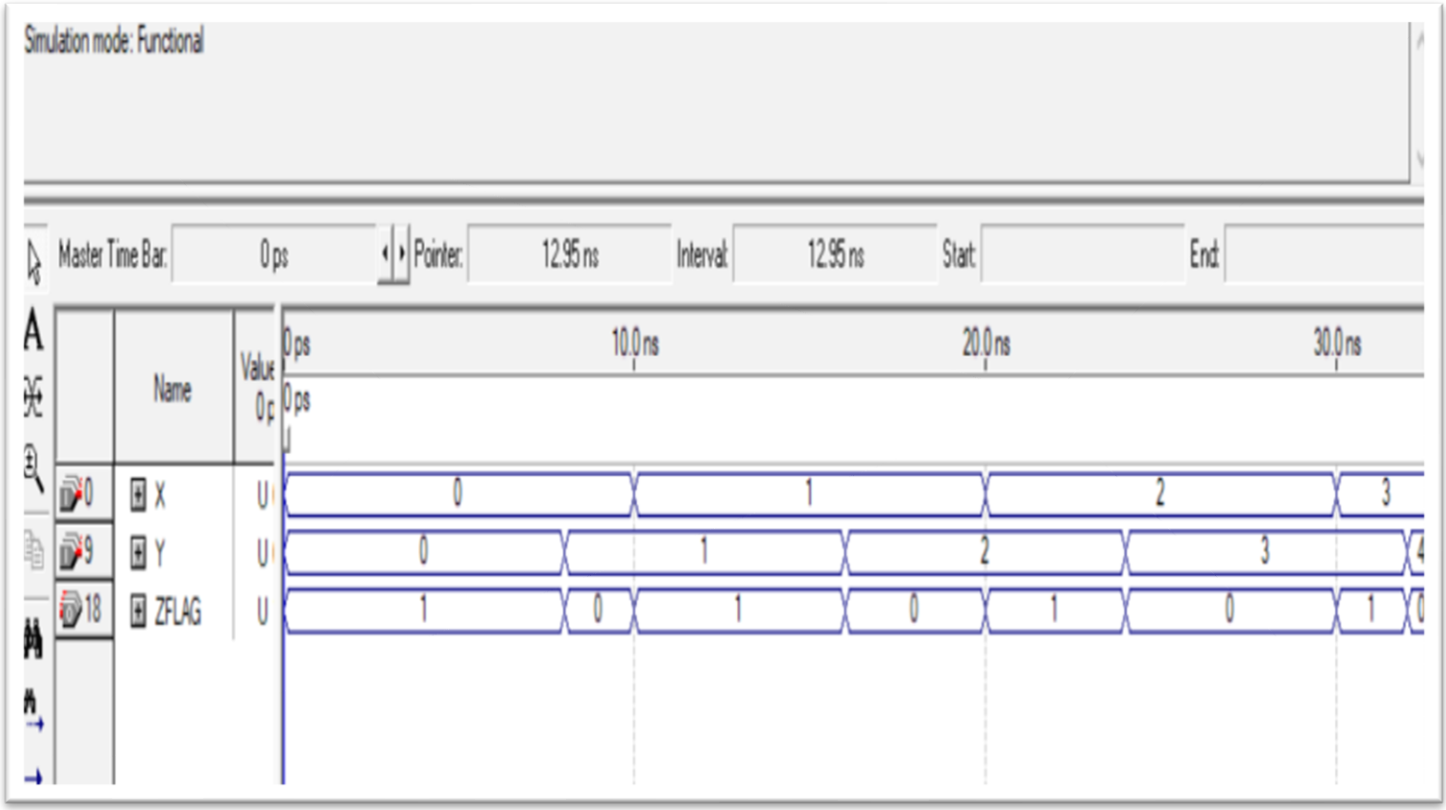
THE VERILOG CODE:

```
module equality(ZFLAG, X , Y);
        input [7:0] X,Y;
        output [7:0] ZFLAG;

        reg [7:0] ZFLAG;

        always @(X , Y)
        begin

                if(X == Y)
                { ZFLAG } = 1;
                else
                {FL ZFLAG }= 0;

        end

endmodule
```

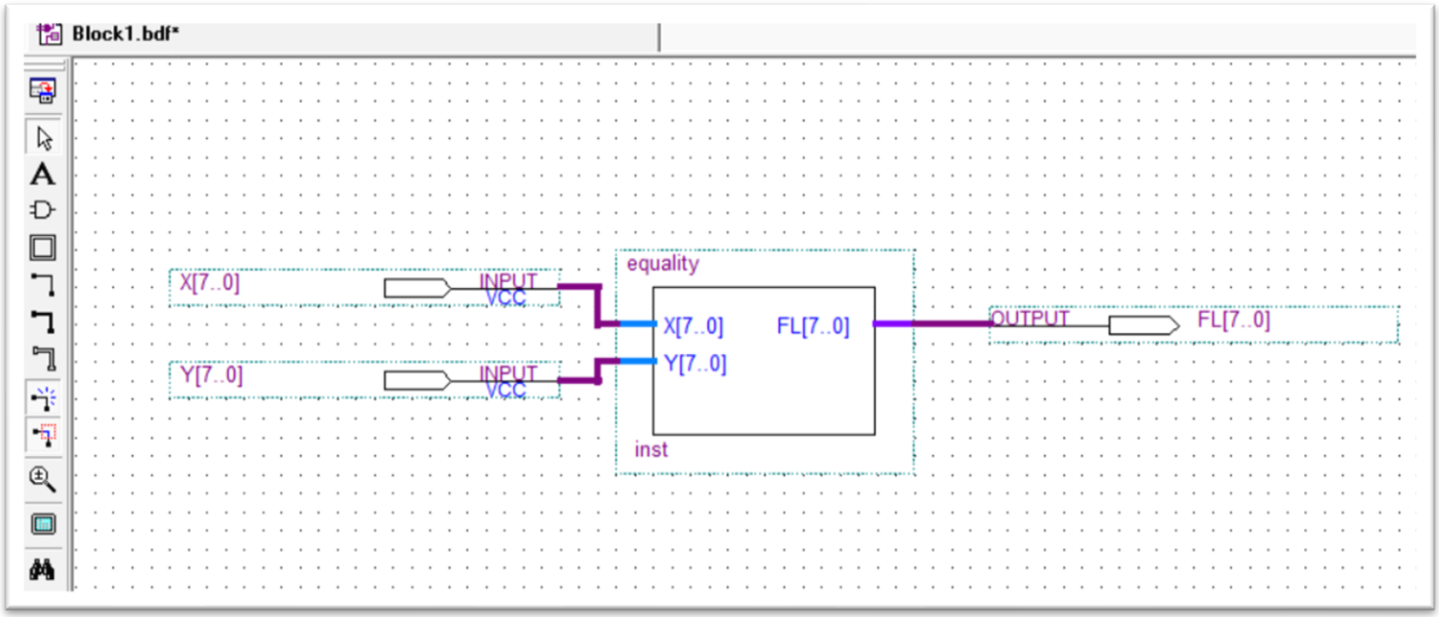
EXPLAINING THE CODE

Here we did a code for the equality operation we needed two inputs (X , Y) and one output it's called the ZFLAG this output will give a result 1 or 0 , so if the two numbers are equal it will give 1 else it gives 0 ……..for example : X = 5 , Y = 4 >> ZFLAG = 0 \\\\\\ X=3 , Y=3 >> ZFLAG=1 .

SIMULATING OF THE CODE:



Block diagram:



#8 EIGHTH CODE: (111>> Less than: Cout=X<Y)

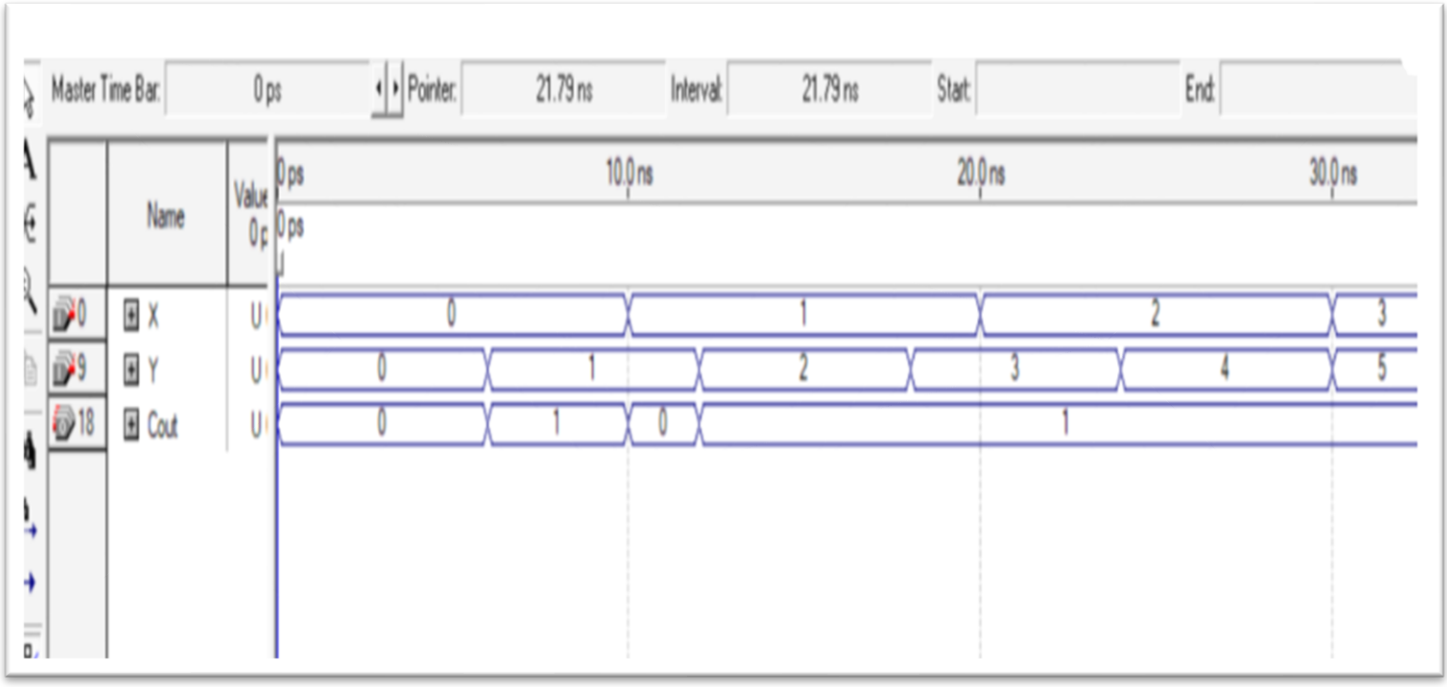
THE VERILOG CODE:

```
module less(Cout , X , Y);
        input [7:0] X,Y;
        output [7:0] Cout;

        reg [7:0]Cout;

        always @(X , Y)
        begin

                if(X < Y)
                        {Cout} = 1;
                else
                        {Cout} = 0;

        end

endmodule
```

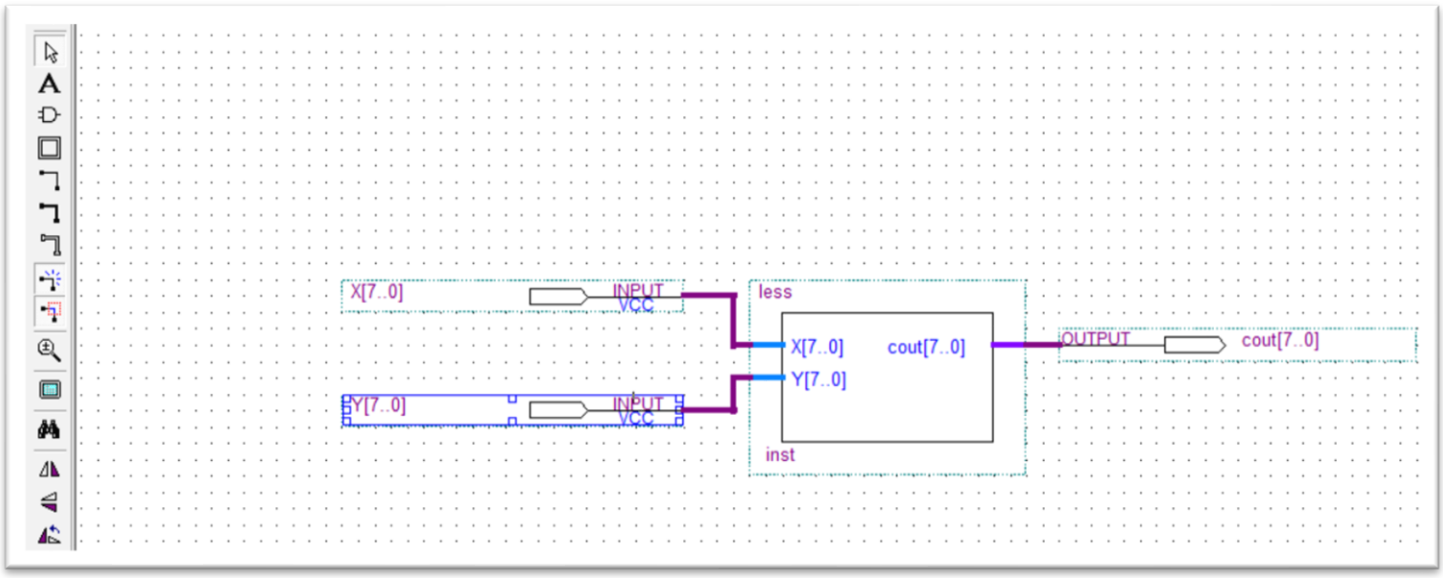
EXPLAINING THE CODE

Here we did a code for the less than operation we needed two inputs (X , Y) and one output it's called Cout this output will give a result 1 or 0 , so if X is less than Y it will give us 1 else it gives 0 …….. for example : X = 5 , Y = 4 >> Cout = 0 \\\\\\ X=2 , Y=4 >> Cout=1 .

SIMULATING OF THE CODE:



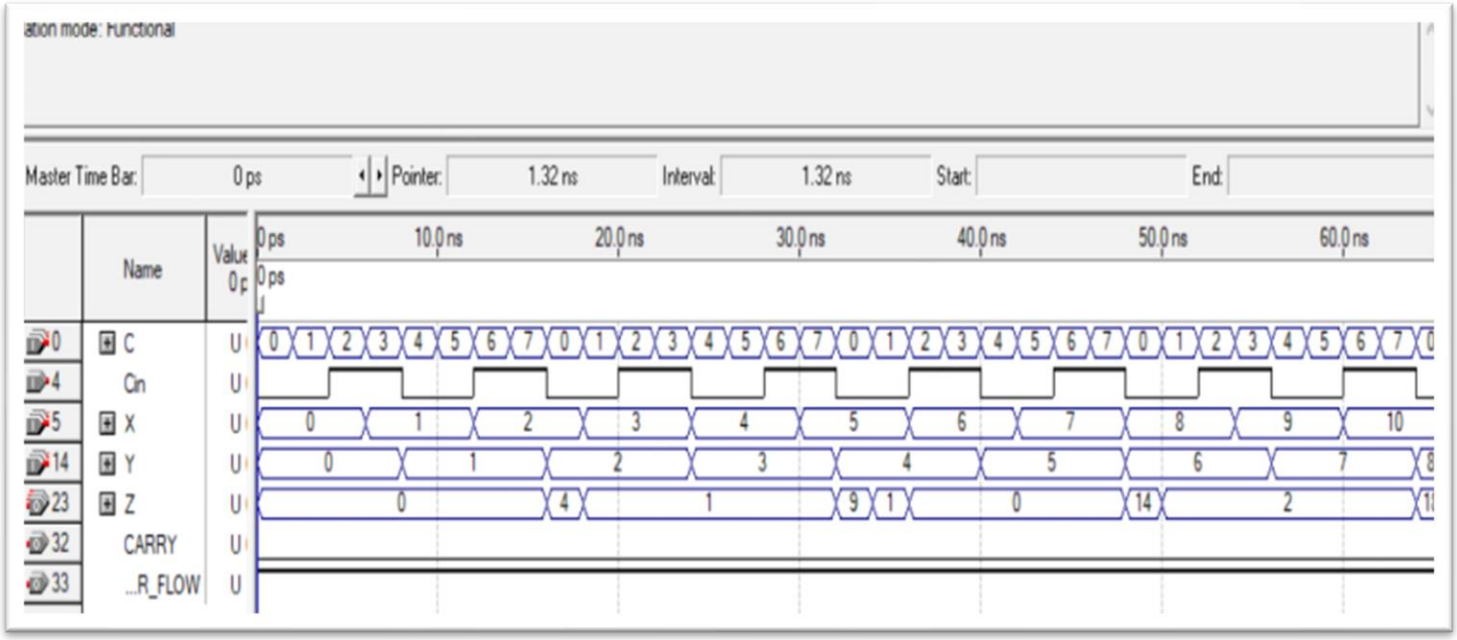
Block diagram:

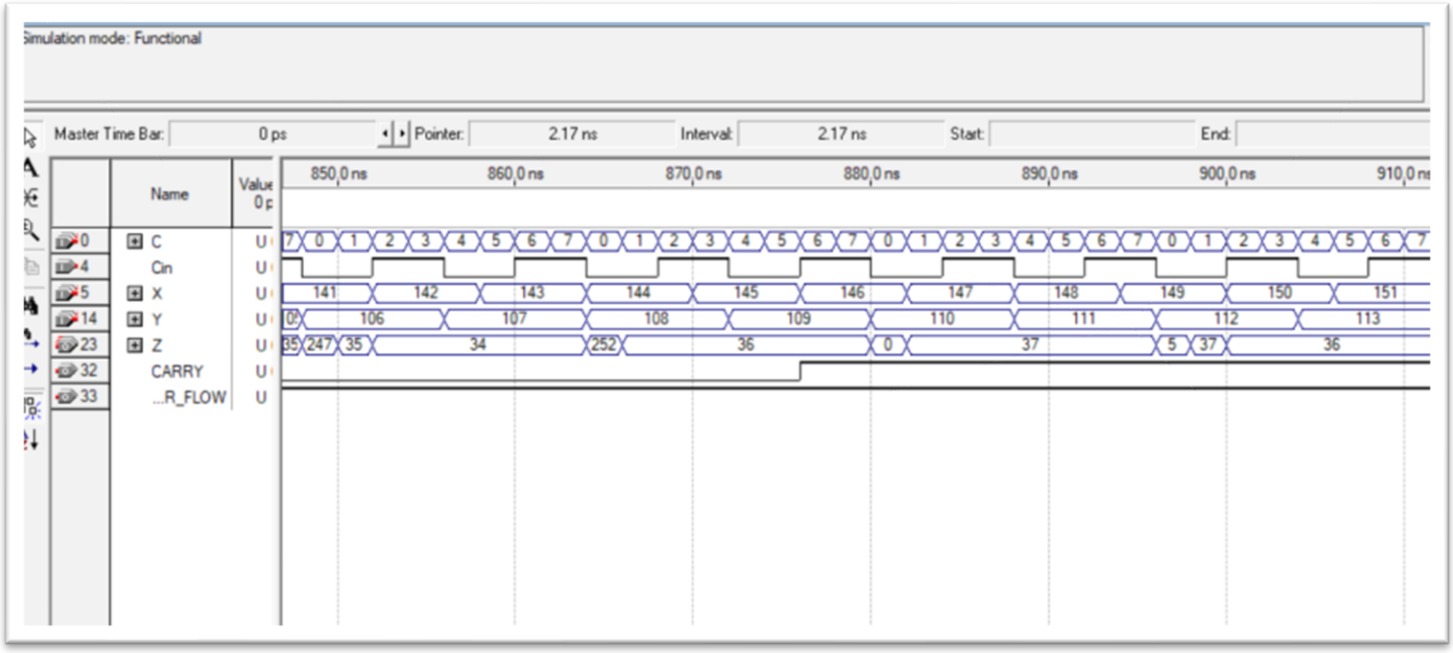


MULTIPEXOR CODE

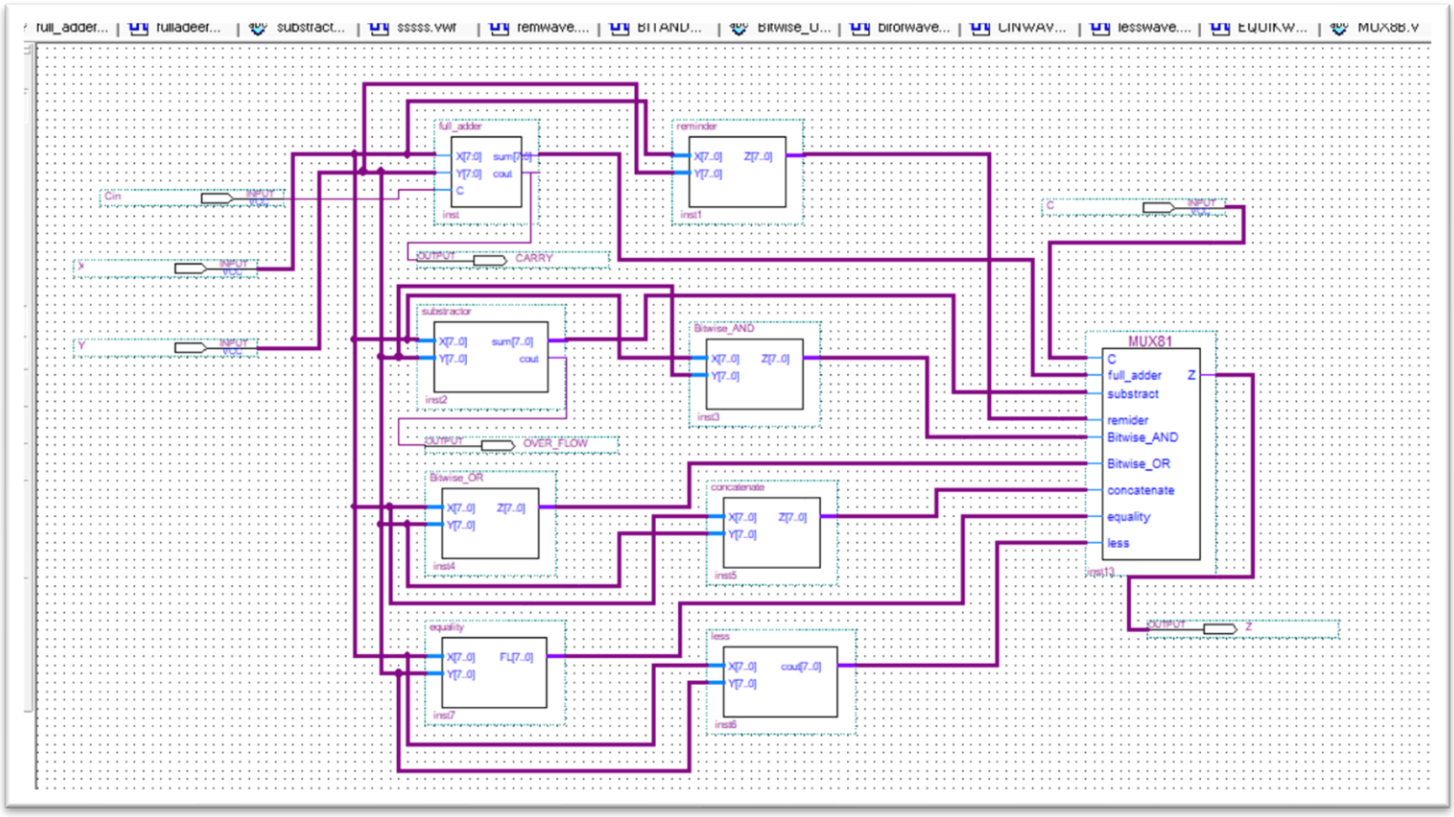
```
module MUX8B (Z , C , fulladder8bit , substractor , remider , Bitwise_And , Bitwise_OR , concatenate , equality , less);
   output [7:0] Z;
   input [7:0] fulladder8bit , substractor , remider , Bitwise_And , Bitwise_OR , concatenate , equality , less;
   input [2:0] C;

   reg [7:0] Z;

   always @(C or fulladder8bit , substractor , remider , Bitwise_And , Bitwise_OR , concatenate , equality , less)
   begin

   case(C)
   000: Z = fulladder8bit;
   001: Z = substractor;
   010: Z = remider;
   011: Z = Bitwise_And;
   100: Z = Bitwise_OR;
   101: Z = concatenate;
   110: Z = equality;
   111: Z = less;

   endcase
   end
endmodule
```

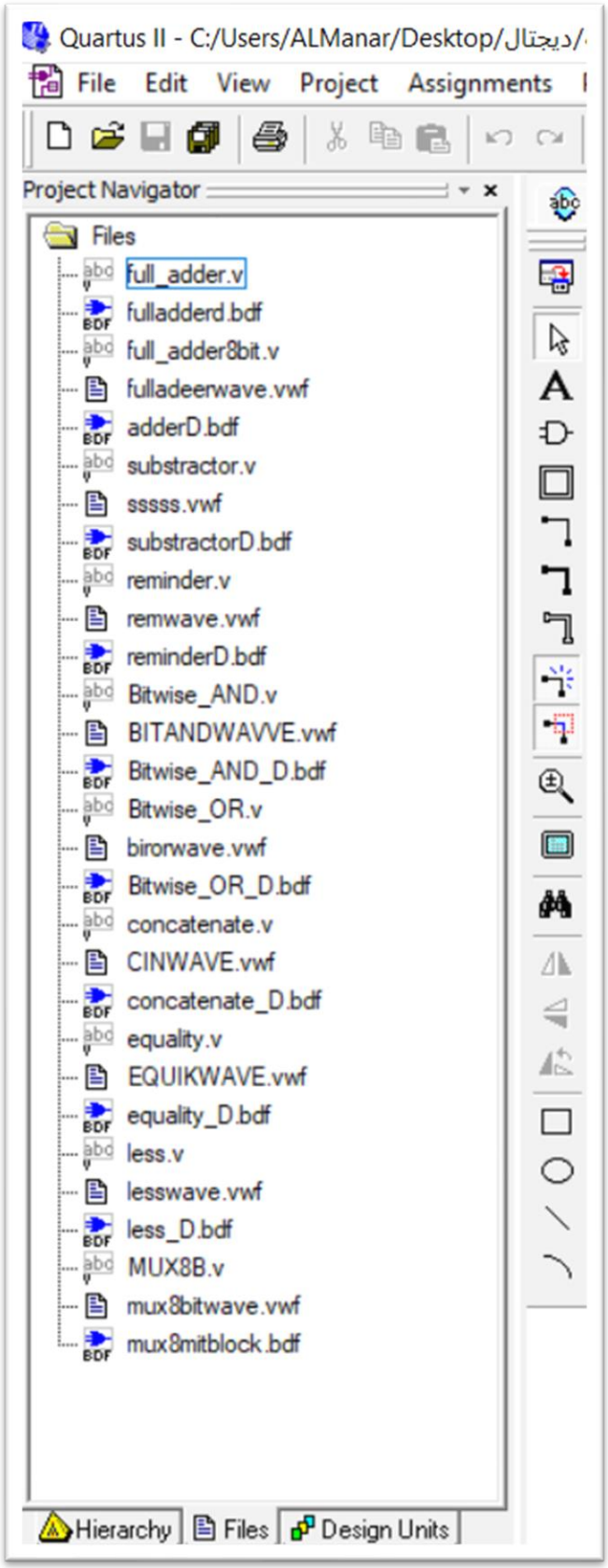
SIMULATING OF THE CODE:





THE BLOCK DIAGRAM:




Quartus II - C:/Users/ALManar/Desktop/ديجتال/
File Edit View Project Assignments
Project Navigator
Files
full_adder.v
fulladderd.bdf
full_adder8bit.v
fulladeerwave.vwf
adderD.bdf
substractor.v
sssss.vwf
substractorD.bdf
reminder.v
remwave.vwf
reminderD.bdf
Bitwise_AND.v
BITANDWAVVE.vwf
Bitwise_AND_D.bdf
Bitwise_OR.v
birorwave.vwf
Bitwise_OR_D.bdf
concatenate.v
CINWAVE.vwf
concatenate_D.bdf
equality.v
EQUIKWAVE.vwf
equality_D.bdf
less.v
lesswave.vwf
less_D.bdf
MUX8B.v
mux8bitwave.vwf
mux8mitblock.bdf
Hierarchy
Files
Design Units