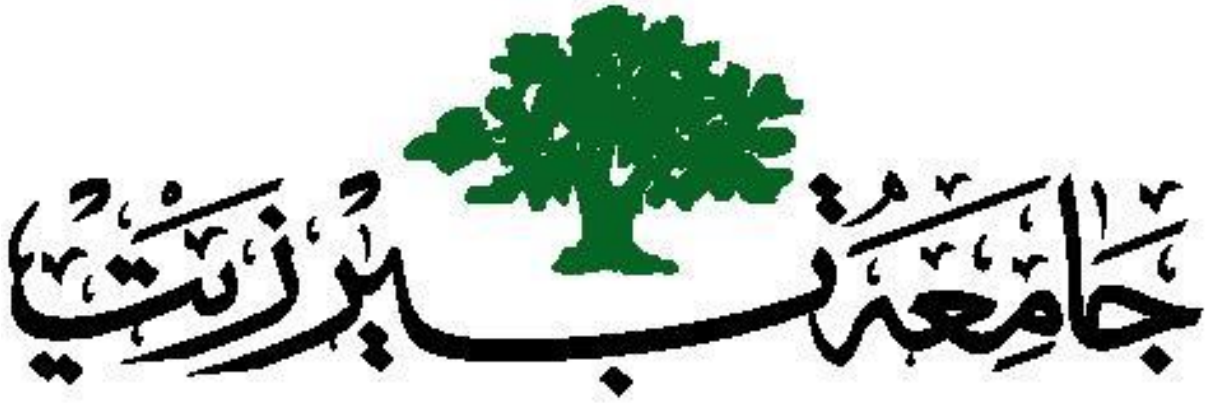

Faculty of Engineering and Technology

Electrical and Computer Engineering Department

DIGITAL ELECTRONICS AND COMPUTER ORGANIZATION LABORATORY

ENCS2110

Report#3

EXP.No.8.Introduction to QUARTUSII Software

.....

Student name: Saja Asfour

Student ID: 1210737

Group members name and ID:

-Saja Asfour 1210737

-Baraa Nassar 1210880

-Reda Srour 1201656

Instructor name: Bilal Karaki

Teacher name: katy sadi

Section number:10

Date: 27/1/2023

Place: Masri107

8.1 Abstract

The aim of the experiment:

- To learn how to use QUARTUS II and write code using Verilog HDL language.
- To learn how test code and make symbol from code.

Equipment used in the experiment:

-QUARTUS II program.

Contents :

List of figure:

8.2 Theory

8.2.1 QUARTUS II Software

-Is a software that provide user with ability to design and simulate different programmable chip designs .

-To make Verilog HDL file :
File->new ->Verilog HDL file.

-To test the code :
New->Vector Waveform File->right click on the left most side of the window ->click insert ->click on Node Finder ->click all->select all input and output->value->countvalue->change the start value and the ends value and the radix to ASCII->save the file->processing-> Simulator Tools->generated waveform is inserted as simulation input-> press on Generate Functional Simulation Netlist ->start ->Report.

-To make symbol from code , make diagram and Schematic File :
New -> Block Diagram / Schematic File.

8.2.2 Verilog HDL

Hardware description language , it describes the hardware of digital system in a textual form , it is also can represent logic diagrams, expression and complex circuits. And a module can be described in any one (or a combination) of the following modeling techniques :

- *Gate-level Modeling:* using instantiation of primitive gate and user defined modules(may have wire).
- *Date-Flow Modeling:* using continues assignment statement with keyword assign.
- *Behavioral Modeling:* using procedural assignment statements with keyword always.

8.2.3 Logic simulations and synthesis

Logic simulation mainly produce timing diagrams that predicts how the hardware will behave before it is fabricated and allows the detection of functional errors in a design before physically implementing the circuit .

8.3 procedure

8.3.1 Task 1

First we implement the full adder in Verilog HDL:

```
1 module FullAdder(input a,b,cin,output sum,cout);
2  assign {cout,sum}=a+b+cin;
3 endmodule
```

Figure 1:full adder Verilog HDL code

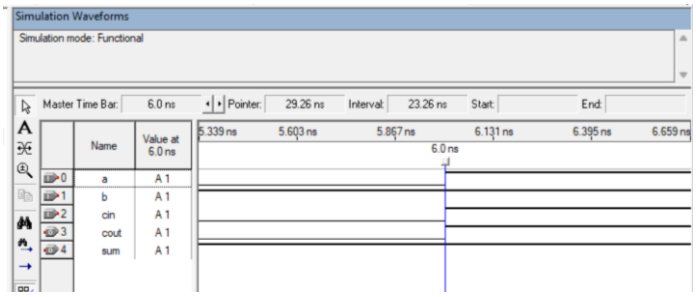

Figure 2: full adder wave form

Second Create Symbol files from the FullAdder module to use it in the final block
Third connect the circuit as follow:

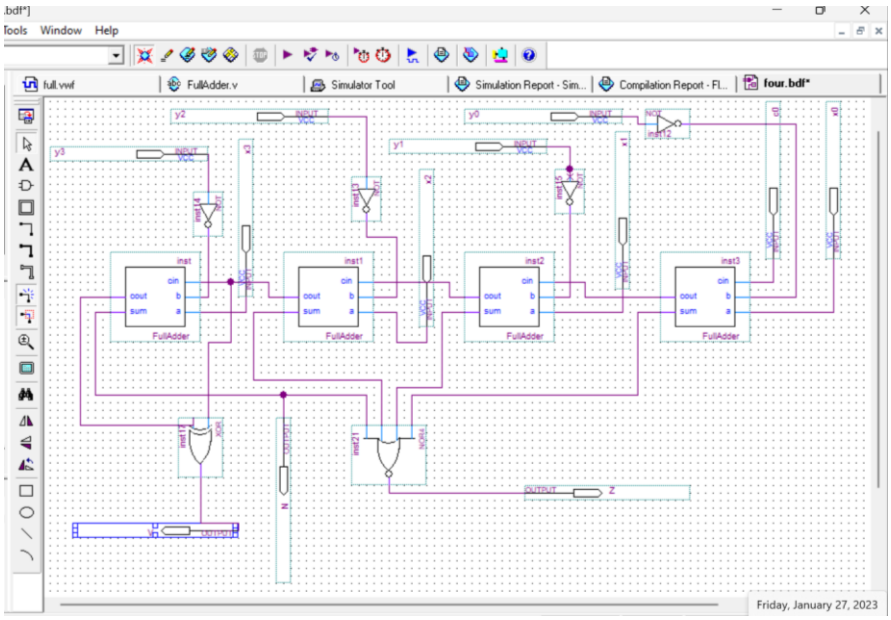

Figure 3: 4-bit-adder block diagram

Then run the meaningful simulation for the above circuit and make waveform for this circuit to make simulation report.

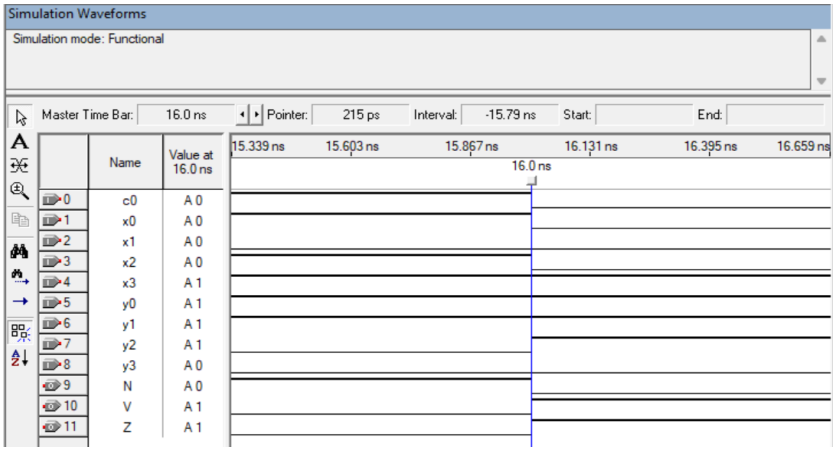

Figure 4: 4-bit-adder waveform

In the above circuit we have 3 output :

-z : zero flag , zero flag be 1 if all input to nor gate are 0 , it one of output is 1 then zero flag be 0.

-N: negative flag : this is the sum output of the fourth full adder , when it is 1 then the number is negative and when it is zero then the number is positive number.

-V: overflow flag, this output be 1 if we add positive number to positive number and the answer was negative number , or when we add negative number to negative number and the answer was positive number .

8.3.2 Task 2

First Use a Verilog HDL to implement a 2 to 1 mux :

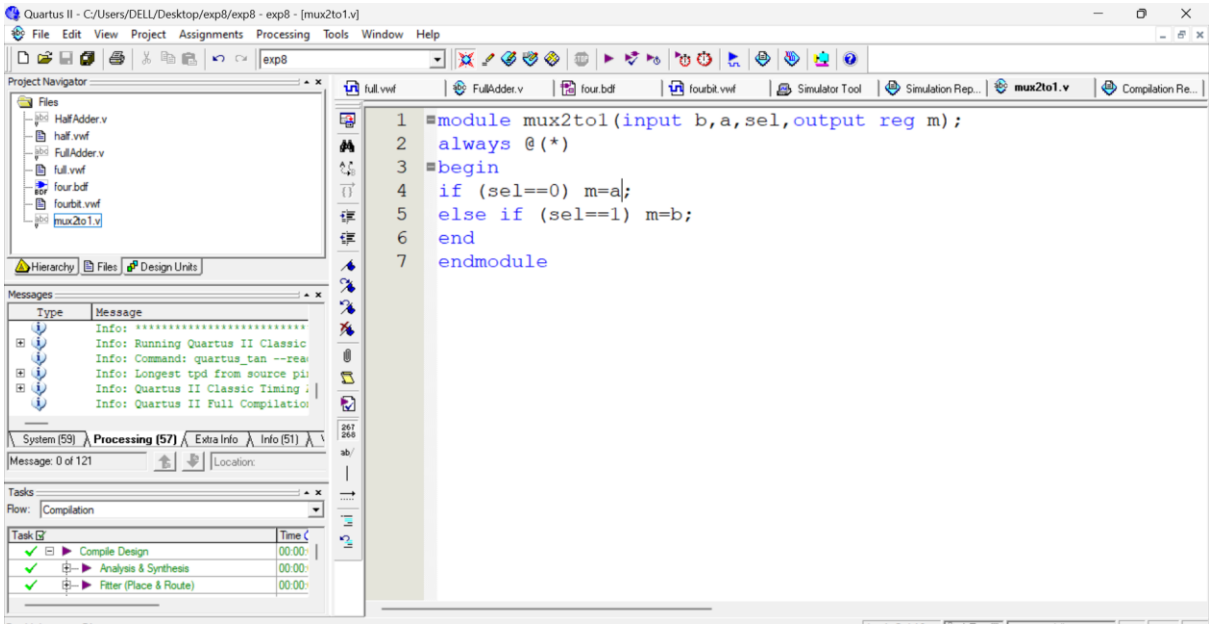

Figure 5: mux 2 to 1 Verilog HDL code

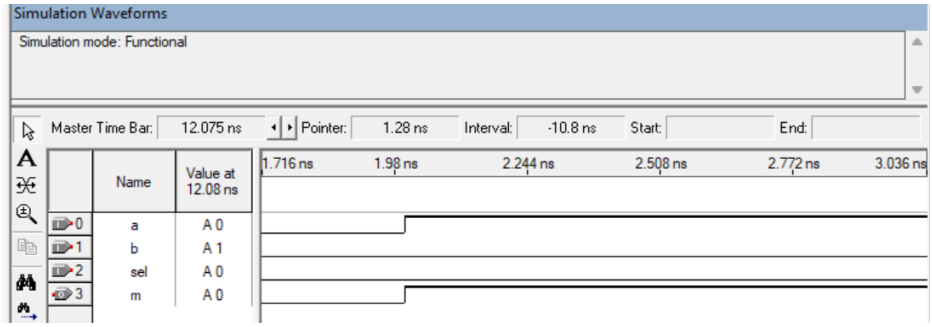

Figure 6: mux 2 to 1 waveform

Second create Schematic symbols for both the MUX and Full Adder ,then connect them as follow:

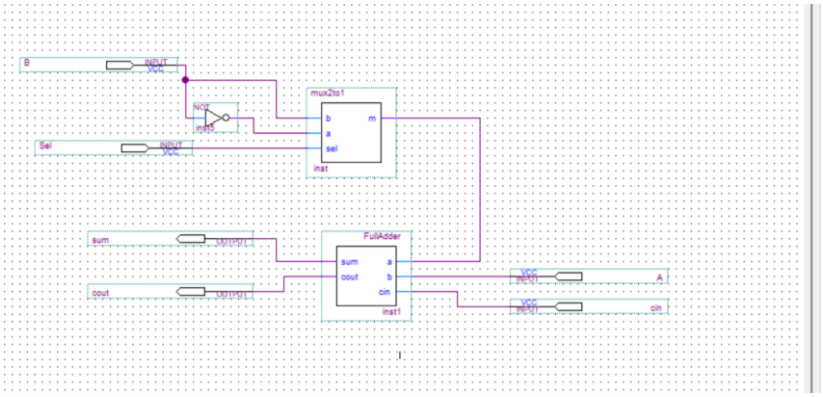

Figure 7: Adder-sub block diagram

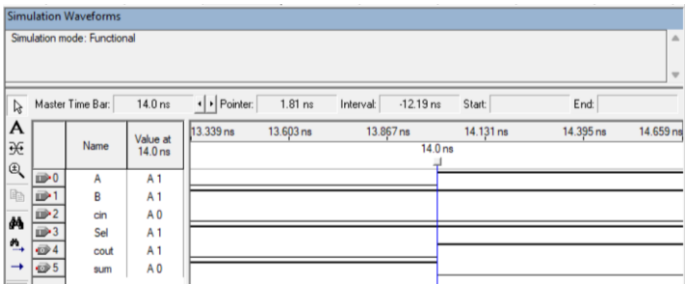

Figure 8: Adder-sub waveform

The circuit above form a sub-adder , when selection equal 0 then we have addition but when the selection equal 1 then we have subtraction.

8.3.3 Task3

First Use a Verilog HDL to implement a 2-bit counter with direct reset input(RESET).

```
1  module counter2bit(clk,rst,count);
2  input clk,rst;
3  output reg [1:0] count;
4  always@ (posedge clk)
5  begin
6  if(rst)//rst==1'b1;
7  begin
8  count <=0;
9  end
10 else
11 begin
12 count <=count+1;
13 end
14 end
15 endmodule
16
```

Figure 9: 2-bit counter Verilog HDL code

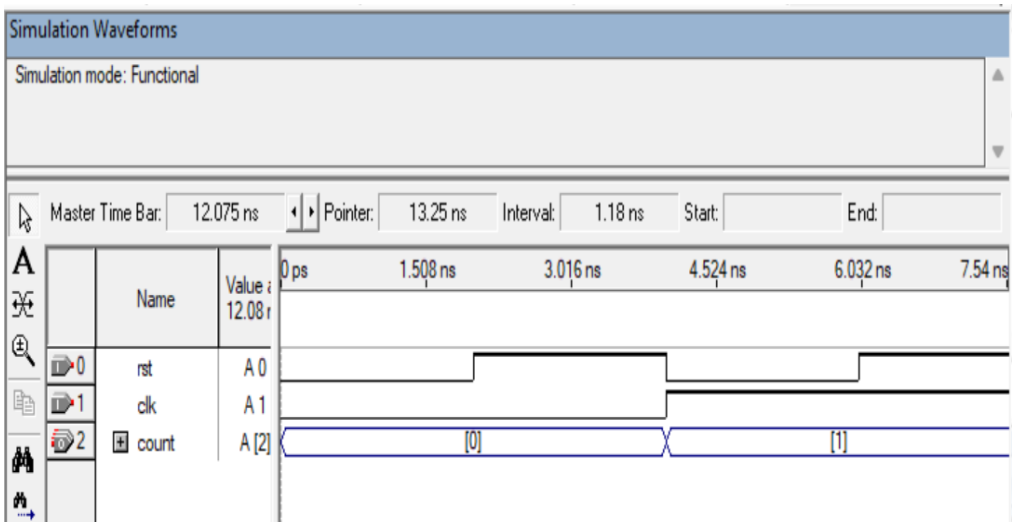

Figure 10: 2-bit counter waveform

Second: Use Verilog HDL to implement a 2-to-4 Decoder:

```
1  module dec2(a,T0,T1,T2,T3);
2   input [1:0]a;
3   output reg T0,T1,T2,T3;
4   always@(a)
5  begin
6  if(a==0) begin T0=1;
7   T1=0;
8   T2=0;
9   T3=0;
10  end
11 else if(a==1)begin  T0=0;
12  T1=1;
13  T2=0;
14  T3=0;
15  end
16 else if(a==2)begin T0=0;
17  T1=0;
18  T2=1;
19  T3=0;
20  end
21 else begin T0=0;
22  T1=0;
23  T2=0;
24  T3=1;
25  end
26  end
27 endmodule
```

Figure 11: 2 to 4 decoder Verilog HDL code

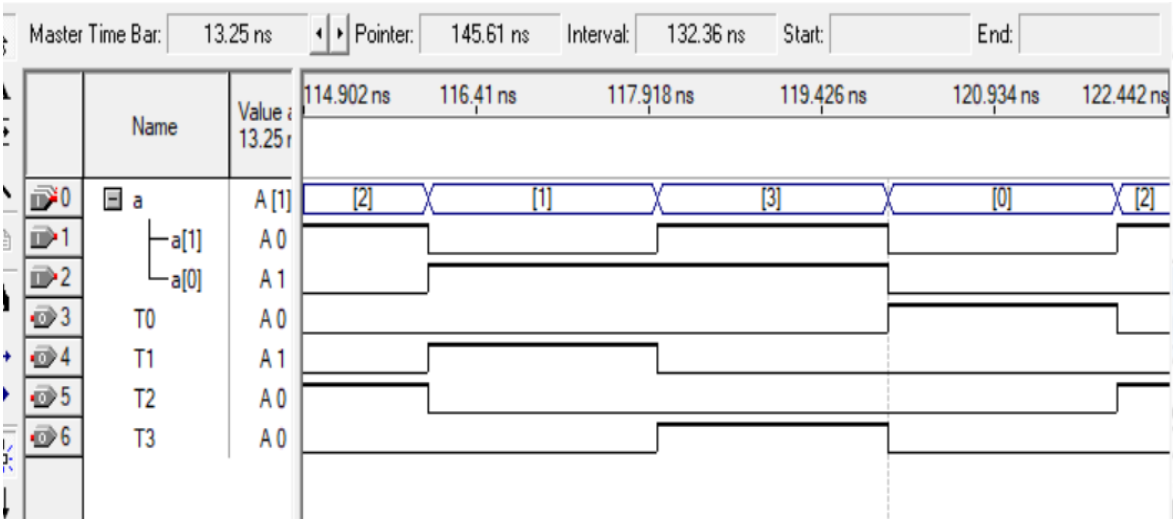

Figure 12: 2 to 4 decoder waveform

Third :Create schematic symbols for both the counter and decoder , then connect them as shown in figure below:

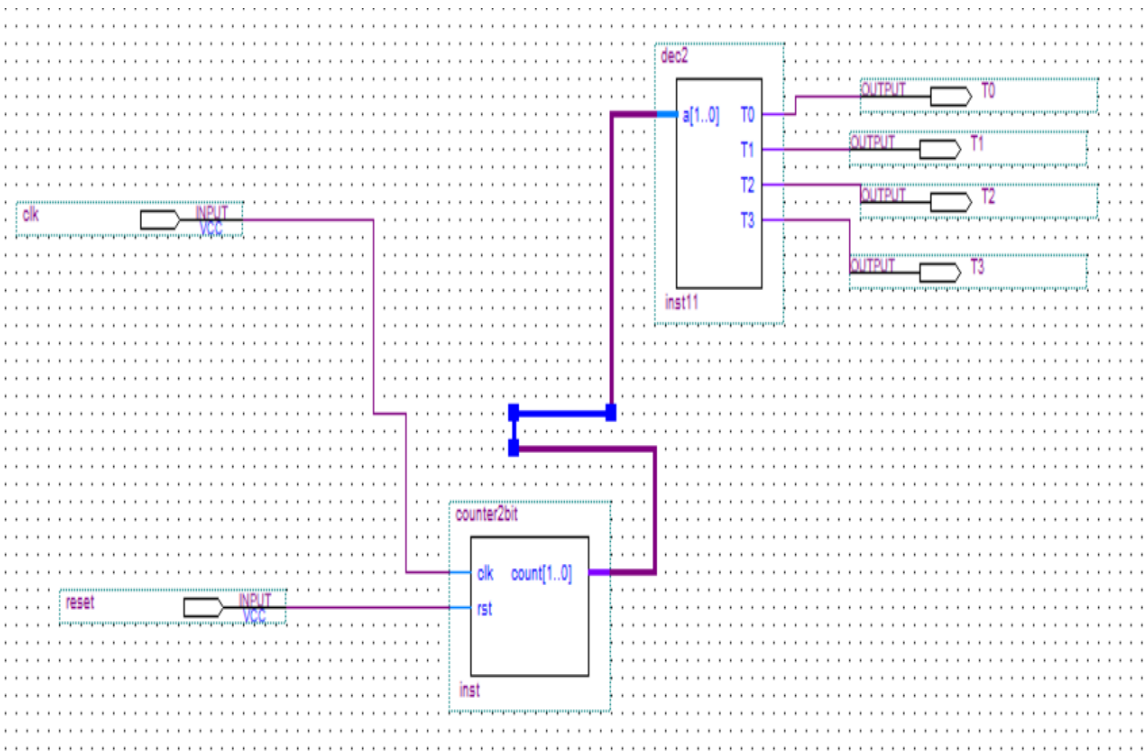

Figure 13: task 3 final block diagram

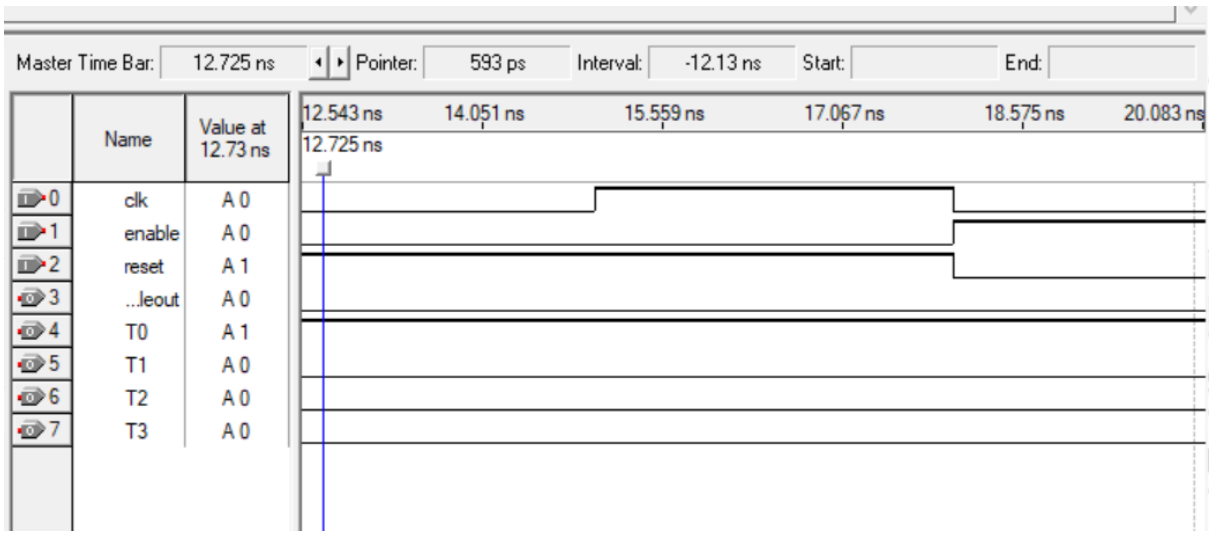

Figure 14: task 3 final block waveform

8.4 Conclusion :

In this experiment every circuit mentioned before was tested and simulated , all of them worked successfully and give result as expected , so in conclusion this experiment helped us to build a digital system using different and separated modules.

8.5 References

- https://www.youtube.com/playlist?list=PLnyw1IVZpaTukmt80aNs7gT74U3vboDYr
- https://www.youtube.com/watch?v=uG1GTRelG3I
- https://www.youtube.com/watch?v=TdLqbgrVREQ
- digital lab manual