Department Of electrical and computer Engineering

ADVANCED DIGITAL SYSTEMS DESIGN

ENCS3310

Project report

Instructor: Dr. Abdallatif AbuIssa

Made By: Islam Jihad

ID: 1191375

Section 2

Date: 25/Nov/2021

Table of Contents

Table of figures

Brief Introduction and Background

We'll build a signed 8-bit comparator in two parts in this project. This system will be made up of little entities that will also be made up of a small number of entities. Table 1 shows how we'll employ the fundamental gates with various delays. Regardless of the number of inputs, the fundamental gates have the same delay.

Gate	Delay
Inverter	2 ns
NAND	5 ns
NOR	5 ns
AND	7 ns
OR	7 ns
XNOR	9 ns
XOR	12 ns

We will create a 1-bit full adder using the basic gates listed above, and then an 8-bit full adder/subtractor, which may be used to implement the adder comparator approach.

And we will create a 1-bit magnitude comparator and 2- bit too, it will be used in the second stage.

The two stages that we will build the system: first, the full adder subtraction as a comparator, is using an adder with (ripple and/or look ahead) full adder 8 times. And the second stage is the magnitude comparator, we had learned them in Digital course.

We will calculate the duration of delay for each stage to use in testing the outputs.

For each step, there will be a Built-In Self-Test (BIST) with two registers: The first register is the test generator, which sends the inputs to our system and sends the outputs to the second register, which has a clock input. The second register is the Result analyzer, which receives the behavioral output from the test generator and the system's output, and ensures that the two outputs are correct.

We will simulate our system, test it, and ensure that the outputs are valid using Aldec Active-HDL Student Edition.

Design philosophy

Basic Gates

First, we created the basic gates as shown in Figure 1. There was an idea to create a single entity for each basic gate and make it generic with a N variable that determines the number of inputs, but it is difficult to construct and call the gate entity in other entities. On the other hand, we made the needed gates that need more than 2 inputs with the same delay to reduce delay.

```
library ieee;
USE ieee.std_logic_1164.ALL;

entity Inverter is
    port(a: in std_logic;
    b: out std_logic);
end entity Inverter;

architecture strct of Inverter is
begin
    b<= not a after 2 ns;

end architecture strct;

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

library ieee;
USE ieee.std_logic_1164.ALL;

entity NANDG is
    port(a,b: in std_logic;
    c: out std_logic);
end entity NANDG;

architecture strct of NANDG is
begin
    c<=  a nand b after 5 ns;

end architecture strct;

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

library ieee;
USE ieee.std_logic_1164.ALL;

entity NORG is
    port(a,b: in std_logic;
    c: out std_logic);
end entity NORG;

architecture strct of NORG is
begin
    c<=  a nor b after 5 ns;

end architecture strct;

```

```

library ieee;
USE ieee.std_logic_1164.ALL;

entity ANDG is
    port(a,b: in std_logic;
    c: out std_logic);
end entity ANDG;

architecture strct of ANDG is
begin
    c<=  a and b after 7 ns;

end architecture strct;

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

library ieee;
USE ieee.std_logic_1164.ALL;

entity ORG is
    port(a,b: in std_logic;
    c: out std_logic);
end entity ORG;

architecture strct of ORG is
begin
    c<=  a or b after 7 ns;

end architecture strct;

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

library ieee;
USE ieee.std_logic_1164.ALL;

entity XNORG is
    port(a,b: in std_logic;
    c: out std_logic);
end entity XNORG;

architecture strct of XNORG is
begin
    c<=  a xnor b after 9 ns;

end architecture strct;
```

```
96
97  library ieee;
98  USE ieee.std_logic_1164.ALL;
99
100 entity XORG is
101     port(a,b: in std_logic;
102     c: out std_logic);
103 end entity XORG;
104
105 architecture strct of XORG is
106 begin
107     c<=  a xor b after 12 ns;
108
109 end architecture strct;
110
111 --xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
112
113 library ieee;
114 USE ieee.std_logic_1164.ALL;
115
116 entity AND3G is
117     port(a,b,d: in std_logic;
118     c: out std_logic);
119 end entity AND3G;
120
121 architecture strct of AND3G is
122 begin
123     c<=  a and b and d after 7 ns;
124
125 end architecture strct;
126
127 --xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
129 library ieee;
130 USE ieee.std_logic_1164.ALL;
131 entity OR3G is
132     port(a,b,d: in std_logic;
133     c: out std_logic);
134 end entity OR3G;
135
136 architecture strct of OR3G is
137 begin
138     c<=  a or b or d after 7 ns;
139
140 end architecture strct;
141 --xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
142
143 library ieee;
144 USE ieee.std_logic_1164.ALL;
145
146 entity XOR_subG is
147     port(a: in std_logic_vector(7 downto 0);
148     c: out std_logic_vector(7 downto 0));
149 end entity XOR_subG;
150
151 architecture strct of XOR_subG is
152 begin
153     c<=  a xor "11111111" after 12 ns;
154
155 end architecture strct;
156 --xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
157
158 library ieee;
159 USE ieee.std_logic_1164.ALL;
160
161 entity NOR8G is
162     port(a: in std_logic_vector(7 downto 0);
163     c: out std_logic);
164 end entity NOR8G;
165
166 architecture strct of NOR8G is
167
168 signal s: std_logic;
169 begin
170
171     s<=  a(0) or a(1) or a(2) or a(3) or a(4) or a(5) or a(6) or a(7);
172     c<= not s after 5 ns;
173 end architecture strct;
```

FIGURE 1:USED GATES WITH DELAY

1-bit Full Adder:

A full adder is a combinational circuit that forms the arithmetic sum of three input bits. It has two inputs: X and Y, that represent the two significant bits to be added, and a Z input that is a carry-in from the previous significant position. It has two outputs: S which is the sum of the two input bits which can be 0-3 and Z to carry the value in case the output from S is 2 or 3 because the binary forms of these require two digits for their representation.[2]

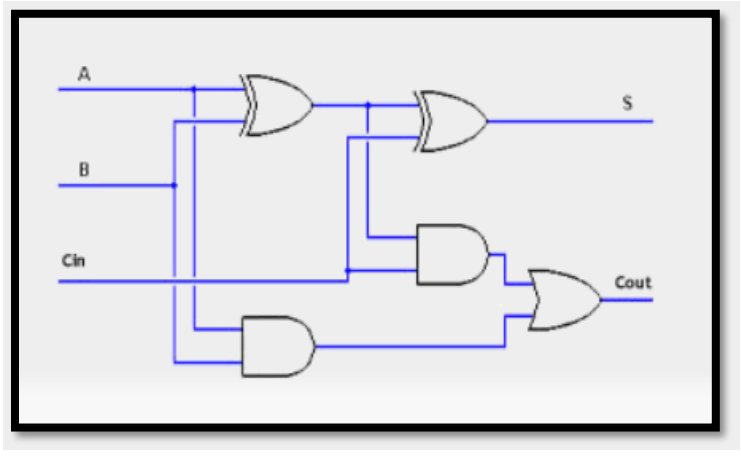

FIGURE 2: 1BIT FULL ADDER IS IMPLEMENTED USING AND,OR AND XOR GATES

```
-- one bit full adder circuit
library ieee;
USE ieee.std_logic_1164.ALL;

entity FA is
    port(A,B,Cin:in std_logic;
    s,Cout:out std_logic);

end entity FA;


architecture one_bit_adder of FA is
signal s1,s2,s3,s4,s5: std_logic;
begin
    g1: entity work.XORG(strct) port map(A,B,s1);
    g2: entity work.ANDG(strct) port map(A,B,s2);
    g3: entity work.ANDG(strct) port map(Cin,s1,s3);
    g4: entity work.XORG(strct) port map(s1,Cin,s4);
    g5: entity work.ORG(strct) port map(s2,s3,s5);

    s<=s4;
    Cout<=s5;

    --24 ns needed to give correct answer
end architecture one_bit_adder;
```

FIGURE 3: FULL ADDER CODE

This full adder is constructed as shown in Figure 3 to be utilized later in the construction of a 8-bit full adder. The maximum delay time was 24 ns.

8-bit Full-Adder:

I take 8 of these full adders(1-bit full adder), and combine them to create an 8-bit Adder. In an 8 bit adder the full adders are connected in a cascade with a 1 carry cascading from a least significant bit to the most significant bit.[2]

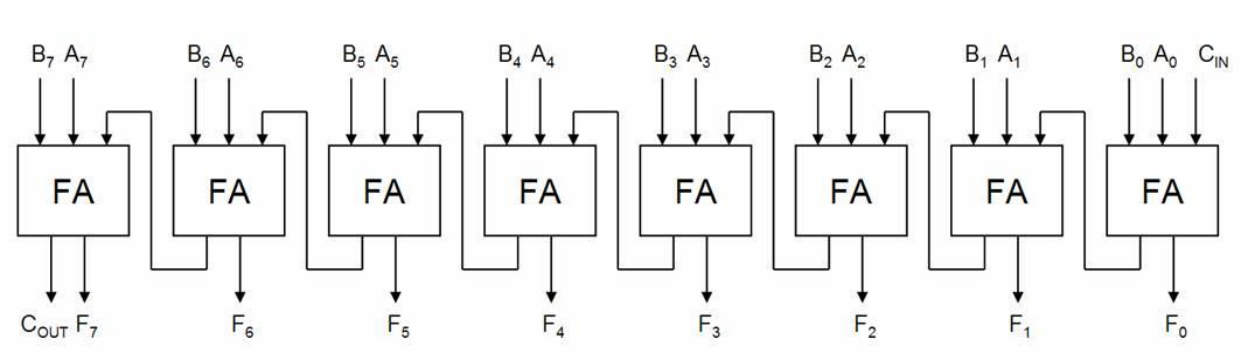

FIGURE 4:COMPLETE 8-BIT ADDER

And the carry out of the last bit and the one before to get the overflow from them later, the delay that needed in this circuit was 24 ns only! It looks like it worked as a lookahead.

The code of the circuit is shown below.

```
--8 bits full adder

library ieee;
USE ieee.std_logic_1164.ALL;

entity bit8_adder is
    port(A,B:in std_logic_vector(7 downto 0);
    Cin:in std_logic;
    sum:out std_logic_vector(7 downto 0);
    Cout6, Cout:out std_logic);

end entity bit8_adder;



architecture strct of bit8_adder is

signal s,c:std_logic_vector(7 downto 0);
begin

    g1: entity work.FA(one_bit_adder) port map(A(0),B(0),Cin,s(0),c(0));
    g2: entity work.FA(one_bit_adder) port map(A(1),B(1),c(0),s(1),c(1));
    g3: entity work.FA(one_bit_adder) port map(A(2),B(2),c(1),s(2),c(2));
    g4: entity work.FA(one_bit_adder) port map(A(3),B(3),c(2),s(3),c(3));
    g5: entity work.FA(one_bit_adder) port map(A(4),B(4),c(3),s(4),c(4));
    g6: entity work.FA(one_bit_adder) port map(A(5),B(5),c(4),s(5),c(5));
    g7: entity work.FA(one_bit_adder) port map(A(6),B(6),c(5),s(6),c(6));
    g8: entity work.FA(one_bit_adder) port map(A(7),B(7),c(6),s(7),c(7));

    Cout6<=c(6);
    Cout<=c(7);
    sum<=s;
    -- 24 ns needed for delay correction
end architecture strct;
```

FIGURE 5: 8-BIT FULL ADDER CODE

Magnitude Comparator in digital logic:

It's a combinational circuit that compares two digital or binary numbers in order to find out whether one binary number is **equal**, **less than** or **greater than** the other binary number. I logically design a circuit for which we will have two inputs one for A and other for B and have three output terminals, one for A > B condition, one for A = B condition and one for A < B condition. [1]

1-Bit Magnitude Comparator

A single bit comparator used to compare two bits. It consists of two inputs each for two single bit numbers and three outputs to generate less than, equal to and greater than between two binary numbers. [1]

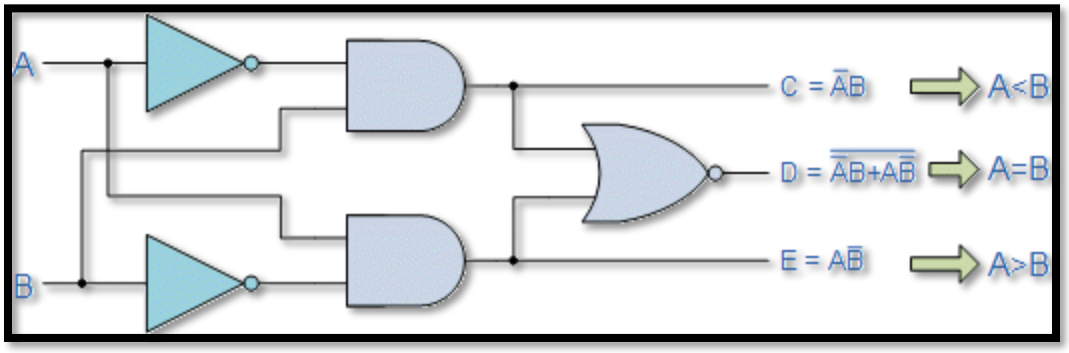

FIGURE 6:1 BIT MAGNITUDE COMPARATOR

The following figure show the code. The delay was 14 ns which is good.

```
5
6   library ieee;
7   USE ieee.std_logic_1164.ALL;
8
9   entity one_bit is
10      port(a,b: in std_logic;
11      F:out std_logic_vector(2 downto 0));
12  end entity one_bit;
13
14
15
16  architecture strct of one_bit is
17
18  signal s1,s2,s3,s4,s5:std_logic;
19
20  begin
21
22      g1: entity work.Inverter(strct) port map(A,s1);
23      g2: entity work.Inverter(strct) port map(B,s2);
24      g3: entity work.ANDG(strct) port map(B,s1,s3);  --A<B
25      g4: entity work.ANDG(strct) port map(A,s2,s4);  --A>B
26      g5: entity work.NORG(strct) port map(s3,s4,s5); --A=B
27
28      F<=(s5 & s3 & s4);
29      -- 14 ns delay
30
31  end architecture strct;
```

FIGURE 7:1-BIT MAGNITUDE COMPARATOR CODE

2-Bit Magnitude Comparator

This comparator used to compare two binary numbers each of two bits. It consists of four inputs and three outputs to generate less than, equal to and greater than between two binary numbers. [1]

I used the gates that we made with their delay to compare it with physical and real life.

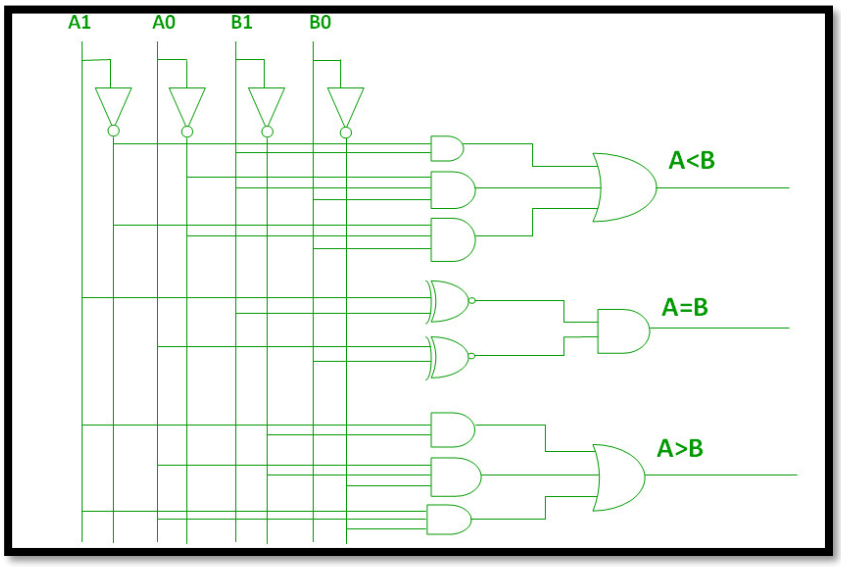

FIGURE 8:: 2 BIT MAGNITUDE COMPARATOR

Here is the code picture, the delay was 16 ns which is good

```
41
42  library ieee;
43  USE ieee.std_logic_1164.ALL;
44
45  entity two_bit is
46      port(Fin: in std_logic_vector(2 downto 0);
47      a1,a0,b1,b0: in std_logic;
48      Fout:out std_logic_vector(2 downto 0));
49  end entity two_bit;
50
51
52
53  architecture strct of two_bit is
54
55  signal na1,na0,nb1,nb0:std_logic;
56  signal a:std_logic_vector(5 downto 0);
57  signal n:std_logic_vector(1 downto 0);
58  signal smaller, equall, greater: std_logic;
59  signal f: std_logic_vector(2 downto 0);
60
61  begin
62
63  -- compare if the previous bits are greater to pass the answer or not to start calculating
64      Fout<="010" when Fin="010"
65  else
66      "001" when Fin="001"
67  else
68      f when Fin="100"
69  else
70      f when Fin="100";
71
```

```
71
72  --=======================================================
73  --the design of the 2 bit cercuit to compare
74  --=======================================================
75      nota1:  entity work.Inverter(strct) port map(A1,na1);
76      nota0:  entity work.Inverter(strct) port map(A0,na0);
77      notb1:  entity work.Inverter(strct) port map(B1,nb1);
78      notb0:  entity work.Inverter(strct) port map(B0,nb0);
79
80
81      -- smaller gates to connect
82      g1: entity work.ANDG(strct) port map(na1,B1,a(0));
83      g2: entity work.AND3G(strct) port map(na0,B1,B0,a(1));
84      g3: entity work.AND3G(strct) port map(na1,na0,B0,a(2));
85      g4: entity work.OR3G(strct) port map(a(0),a(1),a(2),smaller);
86
87      -- equall gates to connect
88      g5: entity work.XNORG(strct) port map(A1,B1,n(0));
89      g6: entity work.XNORG(strct) port map(A0,B0,n(1));
90      g7: entity work.ANDG(strct) port map(n(0),n(1),equall);
91
92      -- greater gates to connect
93      g8: entity work.ANDG(strct) port map(A1,nb1,a(3));
94      g9: entity work.AND3G(strct) port map(A0,nb1,nb0,a(4));
95      g10:entity work.AND3G(strct) port map(A1,A0,nb0,a(5));
96      g11:entity work.OR3G(strct) port map(a(3),a(4),a(5),greater);
97
98
99
100     f<=(equall & smaller & greater);
101
102     --16 ns delay
103 end architecture strct;
```

FIGURE 9:2BIT MAGNITUDE COMPARATOR CODE

➢ Stage 1

In this stage we were required to make a comparator between A and B using the full adders, so what I did was to get the 2's complement of B by XOR it with 1 and log it into full adder with A, so we will get this formula: A + (-B) = A - B

As a result, the answer if was 0 that means that A and B are equals, on the other hand to know if A>B or A<B I had to solve it and find a function that give me a relation and found this:

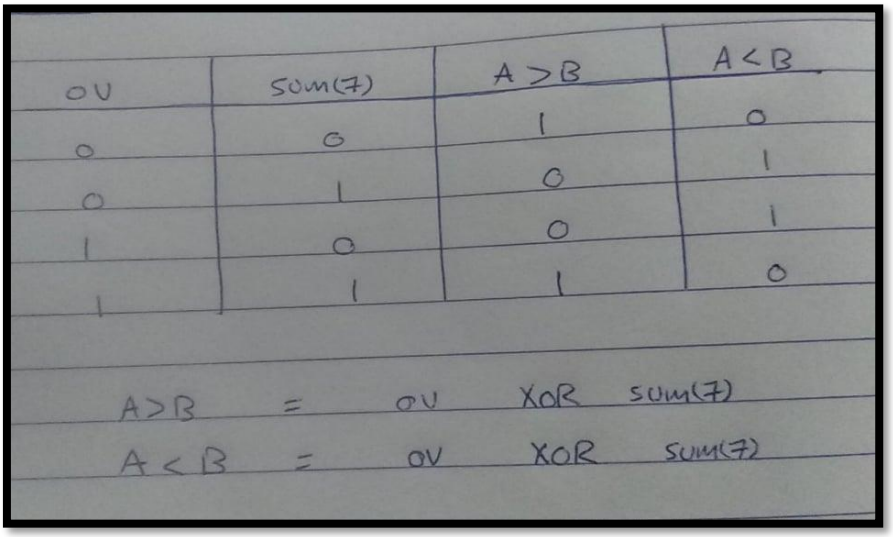

FIGURE 10:COMPARATOR TRUTH TABLE

If the XOR between overflow and the 8th bit of the answer of the summation was 1 this means A<B and if 0 then A>B and applied this as a code.

The next figure shows the code:

```
30
31  library ieee;
32  USE ieee.std_logic_1164.ALL;
33
34  entity comparator is
35      port(clk: in std_logic;
36      A,B: in std_logic_vector(7 downto 0);
37      Fq,Fg,Fs:out std_logic);
38  end entity comparator;
39
```

```
42  -- the comparator using full adder
43  architecture adder_comp of comparator is
44  signal Bxored, sum: std_logic_vector(7 downto 0);
45  signal cout6, cout, Ov, res,equall : std_logic;
46  signal Fout: std_logic_vector(2 downto 0);
47  signal fqr, fgr,fsr: std_logic;
48
49  begin
50      XORB:  entity work.XOR_subG(strct) port map(B, Bxored); -- to nigative all B digits  to be subtracted
51
52      FA8:   entity work.bit8_adder(strct) port map(A, Bxored, '1', sum, cout6, cout); --  8 bit adder subtractor work
53          --like subtractor as it has the B is xor with 1 and have a Cin =1 to work as subtract
54
55      g1: entity work.XORG(strct) port map(cout6, cout, Ov); -- to get the over flow by making xor between the Cout and the previous cout
56
57      g2: entity work.XORG(strct) port map(Ov, sum(7), res); -- to check ether it's grater or smaller (A & B)
58
59      g3: entity work.NOR8G(strct) port map(sum, equall); -- a nor gate for all summation result index so to know ether the sum =0 or not
60
61
62      Fout<= "100" when equall='1'
63      else
64          "001" when  res='1'
65      else
66          "010" when  res='0';
67
68      Fqr<= Fout(2);
69      Fgr<= Fout(1);
70      Fsr<= Fout(0);
71
72      g4: entity work.dfflop(rise_dff) port map(clk,Fqr,Fq);
73      g5: entity work.dfflop(rise_dff) port map(clk,Fgr,Fg);
74      g6: entity work.dfflop(rise_dff) port map(clk,Fsr,Fs);
75  -- delay needed is 127 ns for this circuit
76  -- so th clk will be 127 ns
77  --clk<= not clk after 127 ns;
78  end architecture adder_comp;
```

FIGURE 11:COMPARATOR STAGE 1 CODE

- I made XOR gate with “11111111” to negative all B digits to be subtracted
- Then I insert it with A into the 8 bit adder subtractor to work like subtractor as it has the B is xor with 1 and have a Cin =1 to work as subtractor
- Then I XOR the Carry out with the carry out of the previous one to get the overflow
- Then I made XOR between overflow and the last bit of the summation answer to check ether it's greater or smaller (A & B)
- And added a nor gate for all summation result index so to know ether the sum =0 or not
- The delay needed was 127 ns and that's strange a bit.

The simulation of the results was good but with some glitches because O didn't use a register flip flop till now, here is some results

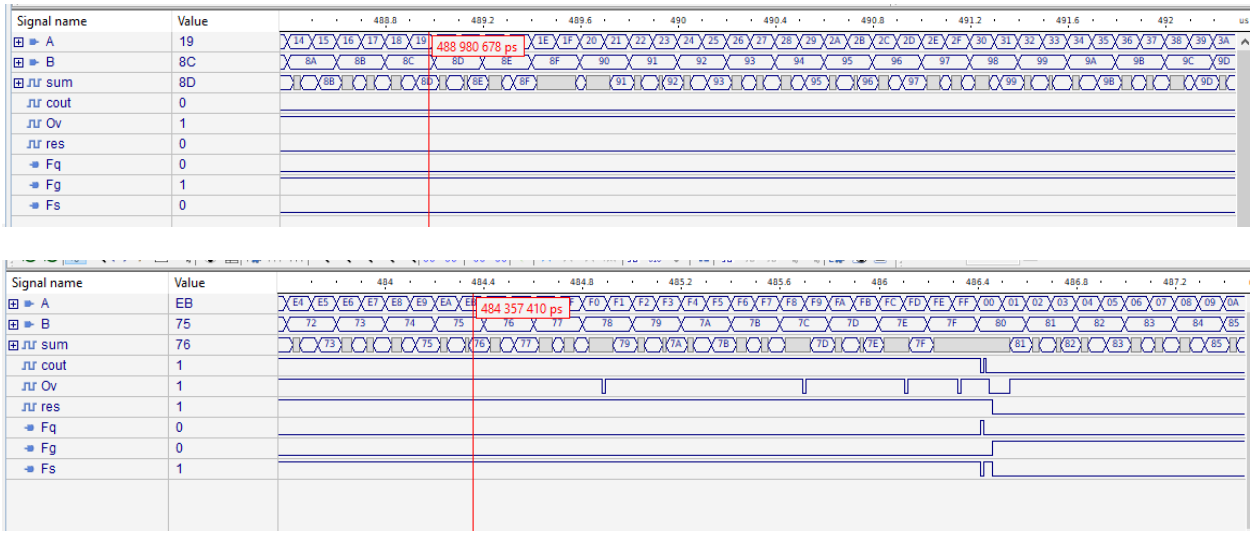

FIGURE 12: STAGE1 OUTPUT

➢ Stage 2

Here we were asked to make the comparator using the magnitude comparator method, so what I did was creating 7-bit magnitude comparator by using the 1-bit and used the 2-bits 3 times and the last bit was the sign bit so there is no need to add it, it's a comparison between 2 digits, if one of them was a negative and the other was positive then the answer will pop up fast but if they both were positive/negative then I have to use the magnitude comparator.

The delay of this circuit was 16 ns and that's good, here is a screenshot of the code:

```
96
97  architecture mag_comp of comparator is
98  signal result, s1, s2, s3, Fout: std_logic_vector(2 downto 0);
99  signal Fqr, Fgr, Fsr: std_logic;
100 begin
101
102     Fout<="010" when A(7)='1' and B(7)='0'
103 else
104     "001" when A(7)='0' and B(7)='1'
105 else
106     result when A(7)='1' and B(7)='1'
107 else
108     result when A(7)='0' and B(7)='0';
109
110     g1: entity work.one_bit(strct) port map(A(6), B(6), s1);
111     g2: entity work.two_bit(strct) port map(s1, A(5),A(4), B(5),B(4), s2);
112     g3: entity work.two_bit(strct) port map(s2, A(3),A(2), B(3),B(2), s3);
113     g4: entity work.two_bit(strct) port map(s3, A(1),A(0), B(1),B(0), result);
114
115     Fqr<= Fout(2);
116     Fsr<= Fout(1);
117     Fgr<= Fout(0);
118
119     g5: entity work.dfflop(rise_dff) port map(clk,Fqr,Fq);
120     g6: entity work.dfflop(rise_dff) port map(clk,Fgr,Fg);
121     g7: entity work.dfflop(rise_dff) port map(clk,Fsr,Fs);
122
123 -- delay needed is 16 ns for this circuit
124 -- so th clk will be 16 ns
125 --clk<= not clk after 16 ns;
126
127 end architecture mag_comp;
```

FIGURE 13:COMPARATORE STAGE2 CODE

The simulation of the results was very good and here is some results

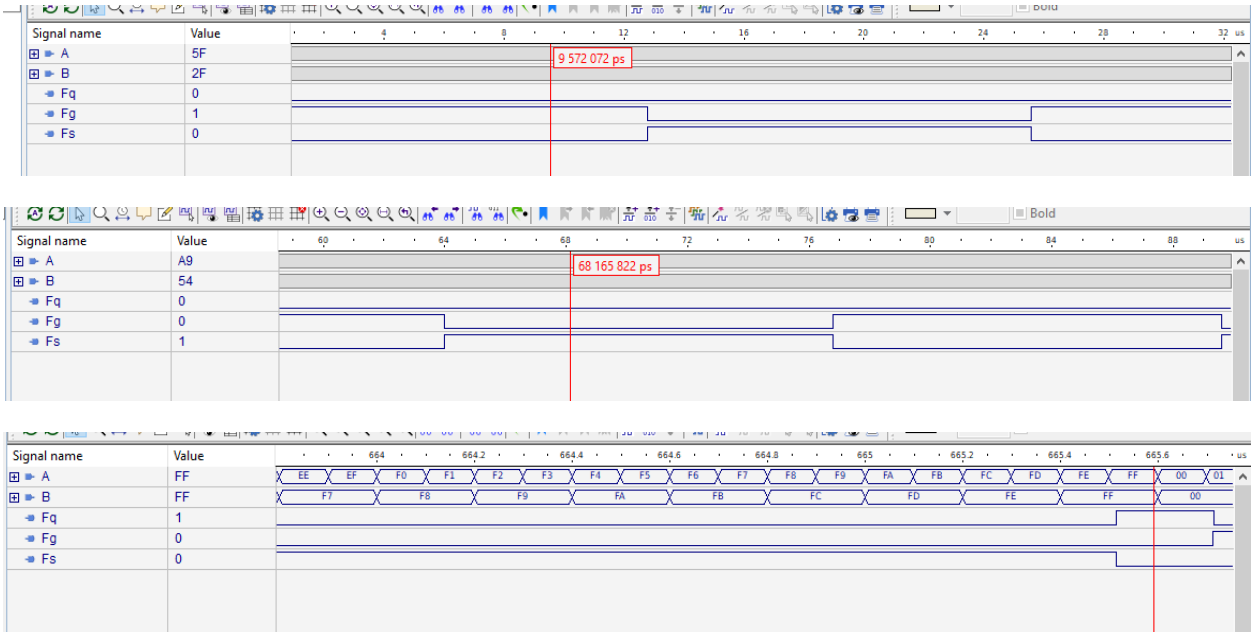

FIGURE 14:STAGE2 OUTPUT

D-Flip Flop

This register works as shown below:

Symbol

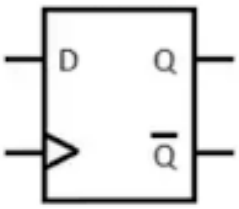

Table of truth:

clk	D	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	Q	$\overline{Q}$
1	0	0	1
1	1	1	0

FIGURE 15:D-FLIP FLOP TRUTH TABLE

It was used to get rid of glitches that appears on the comparator to get clear results; the code of DFF is shown below:

```
2
3
4   -- D Flip Flop to get off delays
5   library ieee;
6   use ieee.std_logic_1164.all;
7
8   entity dfflop is
9      port(clk, d:in std_logic;
10     o : out std_logic );
11  end entity dfflop;
12
13
14  architecture rise_dff of dfflop is
15  begin
16   process(clk)
17   begin
18      if(rising_edge(clk)) then
19      o <= d;
20      end if;
21   end process;
22  end rise_dff;
23
```

FIGURE 16: D FLIP-FLOP CODE

It solved the glitches, and therefore it was added in every stage with a controlled clock by the Built In Self-Test.

Test Generator

This generator contains a clock input, A, B inputs, and the proper output. Its design comprises two processes: the first produces the correct output in behavioral logic, and the second modifies (increments) the values of A and B as the clock input increases to reach all conceivable inputs.

```
82  ----------------------------------------------------
83  ----------------- Test Generator -----------------
84  ----------------------------------------------------
85  LIBRARY ieee;
86  USE ieee.std_logic_1164.ALL;
87  USE ieee.std_logic_ARITH.ALL;
88  USE ieee.std_logic_UNSIGNED.ALL;
89
90
91  ENTITY TestGenerator IS
92  PORT(clk: IN STD_LOGIC:='0';
93  A,B: OUT STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
94  FqCorr, FgCorr, FsCorr: OUT STD_LOGIC:='0');
95  END TestGenerator;
96
97
98  ARCHITECTURE generator OF TestGenerator IS
99  SIGNAL AA,BB: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
100 SIGNAL x: STD_LOGIC_VECTOR(2 DOWNTO 0):="000";
101 BEGIN
102         A<=AA;
103         B<=BB;
104
105         FqCorr<=X(2);
106         FgCorr<=X(1);
107         FsCorr<=X(0);
108
109     -- The Process Below calculate the behavioural results
110
111     PROCESS (clk)
112     BEGIN
113
114             if (AA = BB) then
115                 x<="100";
110
111     PROCESS (clk)
112     BEGIN
113
114             if (AA = BB) then
115                 x<="100";
116             elsif (AA > BB) then
117                 x<="010";
118             elsif (AA < BB) then
119                 x<="001";
120           end if;
121       END PROCESS;
122
123
124         -- this  2 loops to make sure to check all the possible results between A and B
125       PROCESS
126           BEGIN
127               FOR i IN 0 TO 255 LOOP
128                   FOR j IN 0 TO 255 LOOP
129
130                               AA(7 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(i,8);
131                               BB(7 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(j,8);
132                               WAIT UNTIL rising_edge(CLK);
133
134                   END LOOP;
135               END LOOP;
136           WAIT;
137       END PROCESS;
138 END;
139
```

FIGURE 17:TEST CODE

And the outputs of random A and B will go to the circuit I made to be calculated and the result will be compared with the correct ones that had been solved, this will happen in the next level which is called result analyzer.

Result Analyzer

When the clock input increases as illustrated, this analyzer confirms that the outputs are proper. If not, it will give an error message. Here is the code:

```
164  ----------------------------------------------------
165  ---------------- Result Analyser ----------------
166  ----------------------------------------------------
167  LIBRARY ieee;
168  USE ieee.std_logic_1164.ALL;
169  USE ieee.std_logic_ARITH.ALL;
170
171
172  ENTITY ResultAnalyser IS
173  PORT(CLK: IN STD_LOGIC:='0';
174      Fqc, Fgc, Fsc, Fq, Fg, Fs: IN STD_LOGIC:='0');
175  END ResultAnalyser;
176
177
178  ARCHITECTURE analyser OF ResultAnalyser IS
179  BEGIN
180  -- The code below is to make sure that the result from my system equals to the correct one or not
181  -- if not it will print an error when the outputs are not equal to each other
182  PROCESS
183  BEGIN
184  assert (Fqc = Fq and Fgc = Fg and Fsc = Fs)
185  report "The results that were obtained from your design don't agree with the correct results"
186  severity ERROR;
187  WAIT UNTIL rising_edge(CLK);
188  END PROCESS;
189  END;
190
```

FIGURE 18:RESULT ANALYZER CODE

Built in Self-Test

This entity has the whole system with a test generator and result analyzer, as illustrated in Figure - in two phases. The clock signal inverses after a set period, and the test generator changes A and B signals and sends the right output to the result analyzer. The outputs A and B are sent to the system, which generates an output. This output is then sent to the result analyzer, which determines whether or not the output is valid based on the test generator's proper result. The clock signal for the generator and analyzer will be the same. The delay is the difference between the two phases.

```
224  ------------------------------------------------
225  -------------- Built In Self Test --------------
226  ------------------------------------------------
227  LIBRARY ieee;
228  USE ieee.std_logic_1164.ALL;
229  USE ieee.std_logic_ARITH.ALL;
230  ENTITY BIST IS
231  END ENTITY BIST;
232  ------------------------------------------------
233  -------- Test For The adder comparator ---------
234  ARCHITECTURE adder_comp OF BIST IS
235
236  SIGNAL clk: STD_LOGIC:='0';
237  SIGNAL A,B: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
238  SIGNAL Fq, Fg, Fs,Fqc, Fgc, Fsc: STD_LOGIC:='0';
239  BEGIN
240
241  -- 127 ns is the minimum delay we should have to have a correct output
242  -- so I'll increse it 3 ns to make sure of the of the correct answer
243
244  CLK <= NOT CLK AFTER 130 NS;
245
246  G1: ENTITY WORK.TestGenerator(generator) PORT MAP(clk, A, B, Fqc, Fgc, Fsc);
247
248  G2: ENTITY WORK.comparator(adder_comp) PORT MAP(clk, A, B, Fq, Fg, Fs);
249
250  G3: ENTITY WORK.ResultAnalyser(analyser) PORT MAP(clk, Fqc, Fgc, Fsc, Fq, Fg, Fs);
251
252  END ARCHITECTURE adder_comp;
253
```

FIGURE 19:BUILT IN SELF TEST CODE FOR ADDER COMPARATOR

```
265  ------------------------------------------------
266  ------ Test For The magnitude comparator -------
267  ARCHITECTURE mag_comp OF BIST IS
268
269  SIGNAL clk: STD_LOGIC:='0';
270  SIGNAL A,B: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
271  SIGNAL Fq, Fg, Fs,Fqc, Fgc, Fsc: STD_LOGIC:='0';
272  BEGIN
273
274  -- 16 ns is the minimum delay we should have to have a correct output
275  -- so I'll increse it 3 ns to make sure of the of the correct answer
276
277  CLK <= NOT CLK AFTER 19 NS;
278
279  G1: ENTITY WORK.TestGenerator(generator) PORT MAP(clk, A, B, Fqc, Fgc, Fsc);
280
281  G2: ENTITY WORK.comparator(mag_comp) PORT MAP(clk, A, B, Fq, Fg, Fs);
282
283  G3: ENTITY WORK.ResultAnalyser(analyser) PORT MAP(clk, Fqc, Fgc, Fsc, Fq, Fg, Fs);
284
285  END ARCHITECTURE mag_comp;
286
287
```

FIGURE 20:BUILT IN SELF-TEST CODE FOR MAGNITUDE COMPARATOR

Results

❖ Stage 1

We can see that the shortest period to avoid delay issues is 127 nanoseconds, and it will not display an error. Therefore, I increase it by 3 ns to make sure.

The outcomes of the simulation, as well as the discrepancy between the behavioral output and the actual output, are shown in these images.

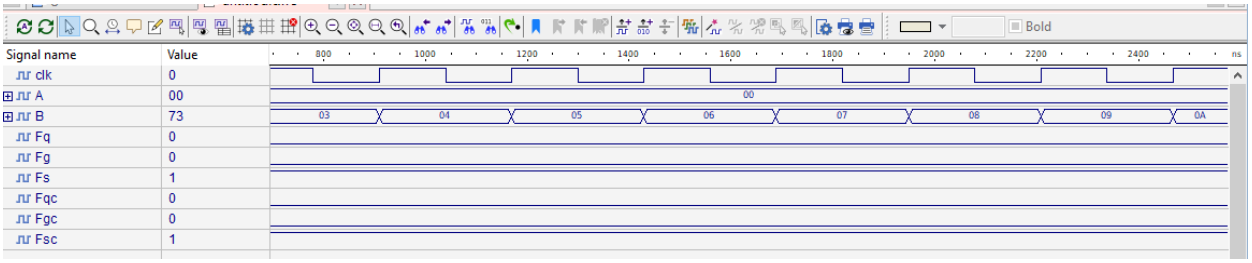

FIGURE 21:SIMULATION OUTCOMES

❖ Stage 2

We can see that the shortest period to avoid delay issues is 16 nanoseconds, and it will not display an error. Therefore, I increase it by 3 ns to make sure.

The outcomes of the simulation, as well as the discrepancy between the behavioral output and the actual output, are shown in these images.

FIGURE 22:OUTCOME OF THE SIMULATION

And I tried to create an error on purpose to check if this is working well or not

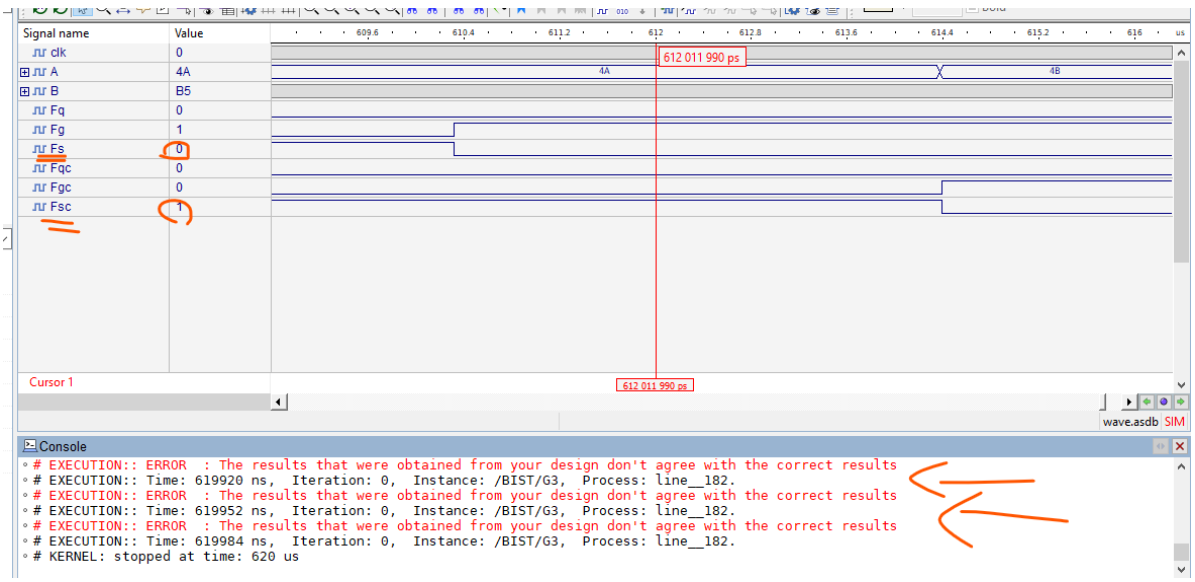

FIGURE 23: TEST THE OUTPUT BY CREATION ERROR

Conclusion and Future works

The outcomes of the preceding operations are consistent with the theoretical results. Furthermore, we infer that we can build large systems using smaller ones.

We successfully construct an 8-bit signed comparator and then write a functioning verification method. We discovered that the built-in test is helpful in ensuring that the results are accurate.

In our system, we learnt about two kinds of adders and saw the difference between the ripple full adder and the carry lookahead adder since it cuts latency significantly. Because the carry of each 1-bit complete adder is independent of the preceding carries save the first, the lookahead adder is quicker than the ripple adder, so we knew what the differences are between the 2 stages.

And how the full adder can be implemented to work in much things.

And how useful it is to use the small blocks instead of creating one huge entity.

We learned more about VHDL and how to create commands such as printing an error, delaying a signal, testing systems, and creating entities in behavioral and structural logics. We also used Aldec HDL to simulate our project and observe the signals of the entity on which we worked.

References

[1] "GeeksForGeeks," 19 Feb 2021. [Online]. Available: https://www.geeksforgeeks.org/magnitude-comparator-in-digital-logic/. [Accessed 24 12 2021].

[2] "UIC Computer Science," [Online]. Available: https://www.cs.uic.edu/~i266/hwk6/42.pdf.

Appendix

```
library ieee;
USE ieee.std_logic_1164.ALL;

entity Inverter is
        port(a: in std_logic;
        b: out std_logic);
end entity Inverter;

architecture strct of Inverter is
begin
        b<= not a after 2 ns;

end architecture strct;

--********************************

library ieee;
USE ieee.std_logic_1164.ALL;

entity NANDG is
        port(a,b: in std_logic;
        c: out std_logic);
end entity NANDG;
```

```
architecture strct of NANDG is
begin
        c<=  a nand b after 5 ns;

end architecture strct;

--********************************

library ieee;
USE ieee.std_logic_1164.ALL;

entity NORG is
        port(a,b: in std_logic;
        c: out std_logic);
end entity NORG;

architecture strct of NORG is
begin
        c<=  a nor b after 5 ns;

end architecture strct;

--********************************

library ieee;
```

```
USE ieee.std_logic_1164.ALL;

entity ANDG is
        port(a,b: in std_logic;
        c: out std_logic);
end entity ANDG;

architecture strct of ANDG is
begin
        c<=  a and b after 7 ns;

end architecture strct;

--*******************************

library ieee;
USE ieee.std_logic_1164.ALL;

entity ORG is
        port(a,b: in std_logic;
        c: out std_logic);
end entity ORG;

architecture strct of ORG is
begin
```

```
        c<=  a or b after 7 ns;

end architecture strct;

--*******************************

library ieee;
USE ieee.std_logic_1164.ALL;

entity XNORG is
        port(a,b: in std_logic;
        c: out std_logic);
end entity XNORG;

architecture strct of XNORG is
begin
        c<=  a xnor b after 9 ns;

end architecture strct;

--*******************************

library ieee;
USE ieee.std_logic_1164.ALL;
```

```
entity XORG is
        port(a,b: in std_logic;
        c: out std_logic);
end entity XORG;

architecture strct of XORG is
begin
        c<=  a xor b after 12 ns;

end architecture strct;

--********************************

library ieee;
USE ieee.std_logic_1164.ALL;

entity AND3G is
        port(a,b,d: in std_logic;
        c: out std_logic);
end entity AND3G;

architecture strct of AND3G is
begin
        c<=  a and b and d after 7 ns;
```

```
end architecture strct;

--*******************************

library ieee;
USE ieee.std_logic_1164.ALL;
entity OR3G is
        port(a,b,d: in std_logic;
        c: out std_logic);
end entity OR3G;

architecture strct of OR3G is
begin
        c<=  a or b or d after 7 ns;

end architecture strct;
--*******************************

library ieee;
USE ieee.std_logic_1164.ALL;

entity XOR_subG is
        port(a: in std_logic_vector(7 downto 0);
        c: out std_logic_vector(7 downto 0));
end entity XOR_subG;
```

```
architecture strct of XOR_subG is
begin
        c<=  a xor "11111111" after 12 ns;

end architecture strct;
--*******************************

library ieee;
USE ieee.std_logic_1164.ALL;

entity NOR8G is
        port(a: in std_logic_vector(7 downto 0);
        c: out std_logic);
end entity NOR8G;

architecture strct of NOR8G is

signal s: std_logic;
begin

        s<=  a(0) or a(1) or a(2) or a(3) or a(4) or a(5) or a(6) or a(7);
        c<= not s after 5 ns;
end architecture strct;
```

```
-- one bit full adder circuit
library ieee;
USE ieee.std_logic_1164.ALL;

entity FA is
        port(A,B,Cin:in std_logic;
        s,Cout:out std_logic);

end entity FA;

architecture one_bit_adder of FA is
signal s1,s2,s3,s4,s5: std_logic;
begin
        g1:     entity work.XORG(strct) port map(A,B,s1);
        g2:     entity work.ANDG(strct) port map(A,B,s2);
        g3:     entity work.ANDG(strct) port map(Cin,s1,s3);
        g4:     entity work.XORG(strct) port map(s1,Cin,s4);
        g5:     entity work.ORG(strct) port map(s2,s3,s5);

        s<=s4;
        Cout<=s5;

        --24 ns needed to give correct answer
end architecture one_bit_adder;
```

```
--8 bits full adder

library ieee;
USE ieee.std_logic_1164.ALL;

entity bit8_adder is
        port(A,B:in std_logic_vector(7 downto 0);
        Cin:in std_logic;
        sum:out std_logic_vector(7 downto 0);
        Cout6, Cout:out std_logic);

end entity bit8_adder;

architecture strct of bit8_adder is

signal s,c:std_logic_vector(7 downto 0);
begin

        g1: entity work.FA(one_bit_adder) port map(A(0),B(0),Cin,s(0),c(0));
        g2: entity work.FA(one_bit_adder) port map(A(1),B(1),c(0),s(1),c(1));
        g3: entity work.FA(one_bit_adder) port map(A(2),B(2),c(1),s(2),c(2));
        g4: entity work.FA(one_bit_adder) port map(A(3),B(3),c(2),s(3),c(3));
        g5: entity work.FA(one_bit_adder) port map(A(4),B(4),c(3),s(4),c(4));
        g6: entity work.FA(one_bit_adder) port map(A(5),B(5),c(4),s(5),c(5));
        g7: entity work.FA(one_bit_adder) port map(A(6),B(6),c(5),s(6),c(6));
```

```
        g8: entity work.FA(one_bit_adder) port map(A(7),B(7),c(6),s(7),c(7));


        Cout6<=c(6);
        Cout<=c(7);
        sum<=s;
        -- 24 ns needed for delay correction
end architecture strct;

--entities  to make the magnitude comparator
--I'll make a 1 bit comparator and 2X3 bit comparators************************************


library ieee;
USE ieee.std_logic_1164.ALL;

entity one_bit is
        port(a,b: in std_logic;
        F:out std_logic_vector(2 downto 0));
end entity one_bit;


architecture strct of one_bit is

signal s1,s2,s3,s4,s5:std_logic;

begin
```

```
        g1:     entity work.Inverter(strct) port map(A,s1);
        g2:     entity work.Inverter(strct) port map(B,s2);
        g3:     entity work.ANDG(strct) port map(B,s1,s3);  --A<B
        g4:     entity work.ANDG(strct) port map(A,s2,s4);      --A>B
        g5:     entity work.NORG(strct) port map(s3,s4,s5);     --A=B

        F<=(s5 & s3 & s4);
        -- 14 ns delay

end architecture strct;




library ieee;
USE ieee.std_logic_1164.ALL;

entity two_bit is
        port(Fin: in std_logic_vector(2 downto 0);
        a1,a0,b1,b0: in std_logic;
        Fout:out std_logic_vector(2 downto 0));
end entity two_bit;
```

```
architecture strct of two_bit is

signal na1,na0,nb1,nb0:std_logic;
signal a:std_logic_vector(5 downto 0);
signal n:std_logic_vector(1 downto 0);
signal smaller, equall, greater: std_logic;
signal f: std_logic_vector(2 downto 0);

begin

-- compare if the previous bits are greater to pass the answer or not to start calculating
        Fout<="010" when Fin="010"
else
        "001" when Fin="001"
else
        f when Fin="100"
else
        f when Fin="100";
```

```
--=====================================================
--the design of the 2 bit cercuit to compare
--=====================================================
        nota1: entity work.Inverter(strct) port map(A1,na1);
        nota0: entity work.Inverter(strct) port map(A0,na0);
        notb1: entity work.Inverter(strct) port map(B1,nb1);
        notb0: entity work.Inverter(strct) port map(B0,nb0);



        -- smaller gates to connect
        g1:     entity work.ANDG(strct) port map(na1,B1,a(0));
        g2:     entity work.AND3G(strct) port map(na0,B1,B0,a(1));
        g3:     entity work.AND3G(strct) port map(na1,na0,B0,a(2));
        g4:     entity work.OR3G(strct) port map(a(0),a(1),a(2),smaller);

        -- equall gates to connect
        g5:     entity work.XNORG(strct) port map(A1,B1,n(0));
        g6:     entity work.XNORG(strct) port map(A0,B0,n(1));
        g7:     entity work.ANDG(strct) port map(n(0),n(1),equall);

        -- greater gates to connect
        g8:     entity work.ANDG(strct) port map(A1,nb1,a(3));
        g9:     entity work.AND3G(strct) port map(A0,nb1,nb0,a(4));
        g10:entity work.AND3G(strct) port map(A1,A0,nb0,a(5));
        g11:entity work.OR3G(strct) port map(a(3),a(4),a(5),greater);
```

```
        f<=(equall & smaller & greater);


        --16 ns delay
end architecture strct;


-- D Flip Flop to get off delays
library ieee;
use ieee.std_logic_1164.all;

entity dfflop is
   port(clk, d:in std_logic;
   o : out std_logic );
end entity dfflop;

architecture rise_dff of dfflop is
begin
 process(clk)
 begin
    if(rising_edge(clk)) then
    o <= d;
    end if;
 end process;
end rise_dff;

--********************************************************************************************
```

```
library ieee;
USE ieee.std_logic_1164.ALL;

entity comparator is
        port(clk: in std_logic;
        A,B: in std_logic_vector(7 downto 0);
        Fq,Fg,Fs:out std_logic);
end entity comparator;

-- the comparator using full adder
architecture adder_comp of comparator is
signal Bxored, sum: std_logic_vector(7 downto 0);
signal cout6, cout, Ov, res,equall : std_logic;
signal Fout: std_logic_vector(2 downto 0);
signal fqr, fgr,fsr: std_logic;

begin
        XORB:  entity work.XOR_subG(strct) port map(B, Bxored);    -- to nigative all B
digits  to be subtracted

        FA8:   entity work.bit8_adder(strct) port map(A, Bxored, '1', sum, cout6, cout); -
-  8 bit adder subtractor work
                --like subtractor as it has the B is xor with 1 and have a Cin =1 to work as
subtract
```

g1: entity work.XORG(strct) port map(cout6, cout, Ov); -- to get the over flow by making xor between the Cout and the previous cout

g2: entity work.XORG(strct) port map(Ov, sum(7), res); -- to check ether it's grater or smaller (A & B)

g3: entity work.NOR8G(strct) port map(sum, equall); -- a nor gate for all summation result index so to know ether the sum =0 or not

```
Fout<= "100" when equall='1'
else
        "001" when  res='1'
else
        "010" when  res='0';

Fqr<= Fout(2);
Fgr<= Fout(1);
Fsr<= Fout(0);

g4:     entity work.dfflop(rise_dff) port map(clk,Fqr,Fq);
g5:     entity work.dfflop(rise_dff) port map(clk,Fgr,Fg);
g6:     entity work.dfflop(rise_dff) port map(clk,Fsr,Fs);
-- delay needed is 127 ns for this circuit
-- so th clk will be 127 ns
--clk<= not clk after 127 ns;
```

```
end architecture adder_comp;


--********************************************************************************

-- the comparator using magnitude comparator

architecture mag_comp of comparator is
signal result, s1, s2, s3, Fout: std_logic_vector(2 downto 0);
signal Fqr, Fgr, Fsr: std_logic;
begin

        Fout<="010" when A(7)='1' and B(7)='0'
else
        "001" when A(7)='0' and B(7)='1'
else
        result when A(7)='1' and B(7)='1'
else
        result when A(7)='0' and B(7)='0';

        g1:     entity work.one_bit(strct) port map(A(6), B(6), s1);
        g2:     entity work.two_bit(strct) port map(s1, A(5),A(4), B(5),B(4), s2);
    g3:  entity work.two_bit(strct) port map(s2, A(3),A(2), B(3),B(2), s3);
        g4:     entity work.two_bit(strct) port map(s3, A(1),A(0), B(1),B(0), result);

        Fqr<= Fout(2);
```

```
        Fsr<= Fout(1);
        Fgr<= Fout(0);

        g5:     entity work.dfflop(rise_dff) port map(clk,Fqr,Fq);
        g6:     entity work.dfflop(rise_dff) port map(clk,Fgr,Fg);
        g7:     entity work.dfflop(rise_dff) port map(clk,Fsr,Fs);

-- delay needed is 16 ns for this circuit
-- so th clk will be 16 ns
--clk<= not clk after 16 ns;

end architecture mag_comp;


--islam jihad 1191375
library ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity testbanch is
end;

architecture test of testbanch is
```

```
signal testa,testb:std_logic_vector(7 downto 0):="00000000";
signal sum: std_logic_vector(7 downto 0);
signal cin:std_logic:='0';
signal cout: std_logic;
begin

        g1:     entity work.bit8_adder(strct) port map(testa, testb,cin,sum,cout);



        testa<=testa + 1 after 200 ns;
        testb<=testb + 1 after 400 ns;
        cin<= not cin after 800 ns;
end;

--**************************************

library ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity testbanch1 is
end;

architecture test of testbanch1 is
```

```
signal testa,testb:std_logic_vector(7 downto 0):="00000000";
signal ans: std_logic_vector(2 downto 0);
signal Fq, Fs, Fg,clk: std_logic;
begin

        g1:     entity work.mag_comp(strct) port map(clk, testa, testb,Fq, Fs, Fg);



        testa<=testa + 1 after 200 ns;
        testb<=testb + 1 after 400 ns;
end;
```

```
------------------------------------------------
---------- Test Generator ----------
------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_ARITH.ALL;
USE ieee.std_logic_UNSIGNED.ALL;

ENTITY TestGenerator IS
PORT(clk: IN STD_LOGIC:='0';
A,B: OUT STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
FqCorr, FgCorr, FsCorr: OUT STD_LOGIC:='0');
END TestGenerator;

ARCHITECTURE generator OF TestGenerator IS
SIGNAL AA,BB: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
SIGNAL x: STD_LOGIC_VECTOR(2 DOWNTO 0):="000";
BEGIN
        A<=AA;
        B<=BB;

        FqCorr<=X(2);
        FgCorr<=X(1);
        FsCorr<=X(0);
```

```
-- The Process Below calculate the behavioural results

PROCESS (clk)
BEGIN

        if (AA = BB) then
                x<="100";
        elsif (AA > BB) then
                x<="010";
        elsif (AA < BB) then
                x<="001";
    end if;
    END PROCESS;


     -- this  2 loops to make sure to check all the possible results between A
and B
    PROCESS
        BEGIN
            FOR i IN 0 TO 255 LOOP
                FOR j IN 0 TO 255 LOOP

                                                AA(7 DOWNTO 0) <=
CONV_STD_LOGIC_VECTOR(i,8);
                                                BB(7 DOWNTO 0) <=
CONV_STD_LOGIC_VECTOR(j,8);
```

```
                                                    WAIT UNTIL rising_edge(CLK);


                                            END LOOP;

                                    END LOOP;

                            WAIT;

                    END PROCESS;

END;


-------------------------------------------------

---------- Result Analyser ----------

-------------------------------------------------

LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

USE ieee.std_logic_ARITH.ALL;


ENTITY ResultAnalyser IS

PORT(CLK: IN STD_LOGIC:='0';

        Fqc, Fgc, Fsc, Fq, Fg, Fs: IN STD_LOGIC:='0');

END ResultAnalyser;


ARCHITECTURE analyser OF ResultAnalyser IS

BEGIN

-- The code below is to make sure that the result from my system equals to the correct one or not

-- if not it will print an error when the outputs are not equal to each other

PROCESS
```

```
BEGIN

assert (Fqc = Fq and Fgc = Fg and Fsc = Fs)

report "The results that were obtained from your design don't agree with the correct
results"

severity ERROR;

WAIT UNTIL rising_edge(CLK);

END PROCESS;

END;



-------------------------------------------------

---------- Built In Self Test ----------

-------------------------------------------------

LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

USE ieee.std_logic_ARITH.ALL;

ENTITY BIST IS

END ENTITY BIST;

-----------------------------------------------

-------- Test For The adder comparator ---------

ARCHITECTURE adder_comp OF BIST IS


SIGNAL clk: STD_LOGIC:='0';

SIGNAL A,B: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";

SIGNAL Fq, Fg, Fs,Fqc, Fgc, Fsc: STD_LOGIC:='0';

BEGIN
```

```
-- 127 ns is the minimum delay we should have to have a correct output

-- so I'll increse it 3 ns to make sure of the of the correct answer


CLK <= NOT CLK AFTER 130 NS;


G1: ENTITY WORK.TestGenerator(generator) PORT MAP(clk, A, B, Fqc, Fgc, Fsc);


G2: ENTITY WORK.comparator(adder_comp) PORT MAP(clk, A, B, Fq, Fg, Fs);


G3: ENTITY WORK.ResultAnalyser(analyser) PORT MAP(clk, Fqc, Fgc, Fsc, Fq, Fg,
Fs);


END ARCHITECTURE adder_comp;


-----------------------------------------------

------ Test For The magnitude comparator -------

ARCHITECTURE mag_comp OF BIST IS


SIGNAL clk: STD_LOGIC:='0';

SIGNAL A,B: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";

SIGNAL Fq, Fg, Fs,Fqc, Fgc, Fsc: STD_LOGIC:='0';

BEGIN


-- 16 ns is the minimum delay we should have to have a correct output
```

```
-- so I'll increse it 3 ns to make sure of the of the correct answer


CLK <= NOT CLK AFTER 16 NS;


G1: ENTITY WORK.TestGenerator(generator) PORT MAP(clk, A, B, Fqc, Fgc, Fsc);


G2: ENTITY WORK.comparator(mag_comp) PORT MAP(clk, A, B, Fq, Fg, Fs);


G3: ENTITY WORK.ResultAnalyser(analyser) PORT MAP(clk, Fqc, Fgc, Fsc, Fq, Fg,
Fs);


END ARCHITECTURE mag_comp;
```