Faculty of Engineering and Technology

Electrical and Computer Engineering Department

ENCS3310

ADVANCED DIGITAL SYSTEMS DESIGN

Project #1

Prepared by: Mohammad Ammar AbuJaber

ID: 1190298

Instructor: Dr. Abdallatif Abuissa

SEC.2

26th Dec. 2021

Abstract:

The aim of this project is understanding, analyzing and testing comparators using two different structural ways and comparing them according to their time delay. Finally, we are going to deal with glitches and wrong outputs.

Table of Contents

Table of Figures

1. Introduction

In this project, we will implement a comparator using two different structures (using ripple adder/subtractor or look-ahead adder/subtractor, and magnitude comparator) and then to write a full code for verification the two stages. I will build the two stages structurally using my own library of logical gates with specific delays as given. The comparator has two 8-bit input and three outputs (F1, F2, F3) when (A=B, A>B, A<B) respectively.

At the beginning, I made several basic logical gates and entities. Then, I used them for building structural circuits using them. Then I made an n-bit register (generic) using D-flip flop to make a synchronized circuit.

The first stage consists of a ripple adder/subtractor and other logical operations. It should calculate the difference between the two numbers, and if they were equal, it will give 0. If A was larger than B, then the value should be positive. And if A was less than B the value should be negative. And our job is to determine -using some algorithms- when the value is negative or positive. I tested the worst case in it and it took around 40 ns to give the correct output. So, I took 2*40 ns for the clock delay and the delay for comparing expected result with the outcome of my circuit.

The second stage is based on comparing the sign bit at first to determine the logical way we will compare the other bits at. And the other 7 bits will go to a 7-bit comparator. This stage had a large delay at worst cases 165 ns. So, I took 2*165 ns to be sure the correct output has been arrived to the output register.

After building the comparator using the two stages, I had to add registers for input and outputs connected with a clock based on the time that avoids the glitches.

Finally, verification with test generator, error message and built-in self-test was made to check the functionality of the comparators and see if there is any error or incorrect output.

2. Theoretical overview

In digital systems, signed comparators can be implemented using many ways to find which input is bigger than the other. In our project we have two stages:

2.1. Stage1

Comparator can be done by subtracting the two numbers and checking whether the answer is zero, positive or negative. It can be determined whether the value is negative or positive by using special flags like: carry flag, zero flag, negative flag and overflow flag. Over flow flag is the (XOR) operation between the last two carries (Carry (7) and Carry (8)).

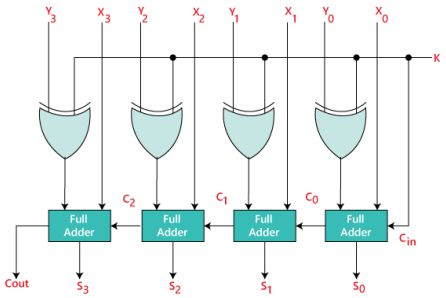


Figure 1: 4-bit adder/subtractor

2.2. Stage2

Another way of comparing two numbers is the magnitude comparator and the sign bit comparator. It will compare the first (sign) bit of the two numbers. Then, it will compare the rest of bits. The first bit (most significant bit) will determine the sign of subtraction to see if the value was '0' then we have to see other bits. But if it was '1', this means that A is less than B.

e.g.:

A:	1	0	1	1	1	1	0	1
B:	0	0	0	1	0	1	0	1

In this case the comparator will check the sign bit at first. As A (7) = 1 and B (7) = 0, then A is less than B and there is no need to compare the other bits.

3. Design Philosophy

Both of stages were made structurally using the delayed gates as given. So, at first the following gates was built:

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

And some gates of more than 2 inputs were made with the same delay as the gates with only two inputs.

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity invertor is
    port (input : in std_logic;
    output : out std_logic);
end entity;

architecture invertor of invertor is
begin
    output <= not input after 2ns;
end architecture;

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

library ieee;
use ieee.std_logic_1164.all;

entity delayed_NAND is
    port (input1,input2 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_NAND of delayed_NAND is
begin
    output <= input1 NAND input2 after 5ns;
end architecture;

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

library ieee;
use ieee.std_logic_1164.all;

entity delayed_NOR is
    port (input1,input2 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_NOR of delayed_NOR is
begin
    output <= input1 NOR input2 after 5ns;
end architecture;
```

```vhdl
entity delayed_AND is
    port (input1,input2 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_AND of delayed_AND is
begin
    output <= input1 AND input2 after 7ns;
end architecture;
--**************************************************************
library ieee;
use ieee.std_logic_1164.all;

entity delayed_AND_3 is
    port (input1,input2,input3 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_AND_3 of delayed_AND_3 is
begin
    output <= input1 AND input2 AND input3 after 7ns;
end architecture;
--**************************************************************
library ieee;
use ieee.std_logic_1164.all;

entity delayed_AND_4 is
    port (input1,input2,input3,input4 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_AND_4 of delayed_AND_4 is
begin
    output <= input1 AND input2 AND input3 AND input4 after 7ns;
end architecture;
--**************************************************************
library ieee;
use ieee.std_logic_1164.all;

entity delayed_AND_5 is
    port (input1,input2,input3,input4,input5 : in std_logic;
    output : out std_logic);
end entity;
```

```vhdl
architecture delayed_AND_5 of delayed_AND_5 is
begin
    output <= input1 AND input2 AND input3 AND input4 AND input5 after 7ns;
end architecture;

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

library ieee;
use ieee.std_logic_1164.all;

entity delayed_OR is
    port (input1,input2 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_OR of delayed_OR is
begin
    output <= input1 OR input2 after 7ns;
end architecture;
--**************************************************************
library ieee;
use ieee.std_logic_1164.all;

entity delayed_OR_3 is
    port (input1,input2,input3 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_OR_3 of delayed_OR_3 is
begin
    output <= input1 OR input2 OR input3 after 7ns;
end architecture;
--**************************************************************
library ieee;
use ieee.std_logic_1164.all;

entity delayed_OR_4 is
    port (input1,input2,input3,input4 : in std_logic;
    output : out std_logic);
end entity;
```

```vhdl
architecture delayed_OR_4 of delayed_OR_4 is
begin
    output <= input1 OR input2 OR input3 OR input4 after 7ns;
end architecture;
--**************************************************************
library ieee;
use ieee.std_logic_1164.all;

entity delayed_OR_5 is
    port (input1,input2,input3,input4,input5 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_OR_5 of delayed_OR_5 is
begin
    output <= input1 OR input2 OR input3 OR input4 OR input5 after 7ns;
end architecture;

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

library ieee;
use ieee.std_logic_1164.all;

entity delayed_XNOR is
    port (input1,input2 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_XNOR of delayed_XNOR is
begin
    output <= input1 XNOR input2 after 9ns;
end architecture;

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

library ieee;
use ieee.std_logic_1164.all;

entity delayed_XOR is
    port (input1,input2 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_XOR of delayed_XOR is
begin
    output <= input1 XOR input2 after 12ns;
end architecture;
```

Figure 2: Basic logic gates with delays

3.1. Registers

To make my circuit synchronized, I added registers for the inputs and outputs connected with the same clock with the circuit as shown below:

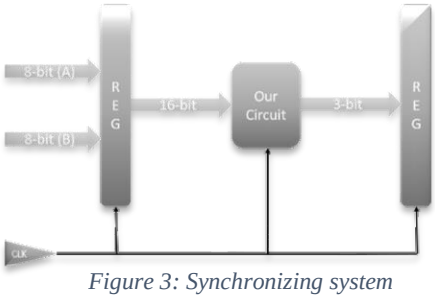


Figure 3: Synchronizing system

3.1.1. One-bit register

I built a D-flip flop to use it in building n-bit register.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity dff is
port (D,clk ,reset: in std_logic;
 Q: out std_logic);
end entity;

architecture dff of dff is
begin
process(clk, reset)
begin
    if (reset = '1') then
        Q<='0';
    elsif (rising_edge(clk)) then
        if (D='1') then
            Q<= '1';
        elsif(D='0') then
            Q<= '0';
        end if;
    end if;
end process;
end architecture;
```

Figure 4: DFF

3.1.2. N-bit Register

I built the n-bit register (generic), to use it for inputs and outputs of my circuits.

```
entity gen_register is
GENERIC (N : positive := 8);
PORT (clk, reset : in STD_LOGIC;
        D: in std_logic_vector(n-1 downto 0);
        Q: out std_logic_vector(n-1 downto 0));
end entity gen_register;
architecture gen_register of gen_register is
begin
    gen : for i in 0 to n - 1 generate
    gen1 : entity work.dff(dff) port map (D(i),clk,reset,Q(i));
    end generate;
end architecture;
```

Figure 5: N-bit register

3.2. Stage1

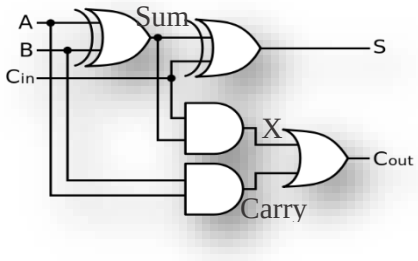
In this stage I implemented an eight-bit ripple adder/subtractor to calculate (A-B). I have done this with several steps:

3.2.1. 1-bit full adder

The adder was made using the known equations:

- $S = A \oplus B \oplus Cin$
- $Cout = (A \cdot B) + (Cin \cdot (A \oplus B))$.

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

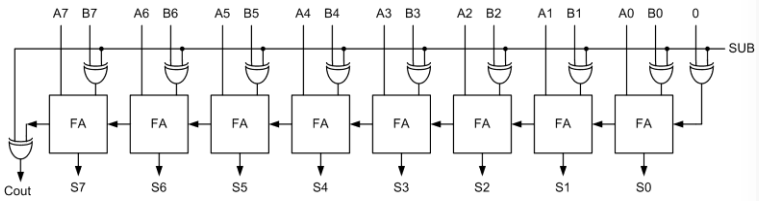



```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.std_logic_signed.ALL;

entity F_Adder is
    port (a,b,c_in: in std_logic;
    sum,carry: out std_logic);
end entity;

architecture F_Adder of F_Adder is
signal sum0,carry0,x: std_logic;
begin
    g0: entity work.delayed_XOR(delayed_XOR) port map (a,b,sum0);
    g1: entity work.delayed_XOR(delayed_XOR) port map (sum0,c_in,sum);
    g2: entity work.delayed_AND(delayed_AND) port map (a,b,carry0);
    g3: entity work.delayed_AND(delayed_AND) port map (sum0,c_in,x);
    g4: entity work.delayed_OR(delayed_OR) port map (carry0,x,carry);
end architecture;
```

Figure 6: 1-bit full adder

3.2.2. 8-bit full adder/subtractor

The 1-bit adder was used to build the 8-bit adder/subtractor. XOR gates were added to calculate the 1's complement of B.




```
entity eight_bit_adder is
    port (a,b: in std_logic_vector(7 downto 0);
    c_in: std_logic;
    sum: out std_logic_vector (7 downto 0);
    carry,overflow: out std_logic
    );
end entity;

architecture eight_bit_adder of eight_bit_adder is
signal ones_complement: std_logic_vector (7 downto 0);
signal c: std_logic_vector (8 downto 0);
begin
    c(0) <= c_in;
    gen1: FOR i IN 0 TO 7 GENERATE
          BEGIN
              g: entity work.delayed_XOR(delayed_XOR) port map (b(i),c_in,ones_complement(i));
              f: entity work.F_Adder(F_Adder) port map (a(i),ones_complement(i),c(i),sum(i),c(i+1));
          END GENERATE;
    carry <= c(8);
    h: entity work.delayed_XOR(delayed_XOR) port map (c(7),c(8),overflow);
end architecture;
```

Figure 7: 8-bit full adder/subtractor

I added an (XOR) operation between the last two carries <#7 and #8> for the adder to set overflow flag that will help for determine whether the answer is negative or not.

3.2.3. The full comparator

As we learned in digital systems and computer design laboratory courses, "greater than" flag will be set if the number isn't zero and negative flag equals to overflow flag. While the "less than" flag will be set if the negative flag isn't equal to overflow flag.

LT	N!=V	<
GT	Z==0&&N==V	>

Figure 8: Flags

```
entity comparator is
    port (a,b: in std_logic_vector (7 downto 0);
    f1,f2,f3: inout std_logic;
    outReg: out std_logic_vector(2 downto 0);
    clk: in std_logic);
end entity;

architecture comparator of comparator is
signal value,a_temp,b_temp: std_logic_vector(7 downto 0);
signal V_flag,Carry_flag,Not_Z_flag,test: std_logic;
signal reg: std_logic_vector(2 downto 0);

begin
Not_Z_flag <= '0' when value = "00000000" else '1';
test <= '1' when V_flag = value(7) else '0';
    g0: entity work.eight_bit_adder(eight_bit_adder) port map (a_temp,b_temp,'1',value,Carry_flag,V_flag);
    g1: entity work.delayed_XOR(delayed_XOR) port map (value(7),V_flag,f3);
    g2: entity work.delayed_AND(delayed_AND) port map (Not_Z_flag,test,f2);

    f1 <= '1' when value = "00000000" else '0';
```

Figure 9: Full comparator

To make the comparator synchronized with the register and test generator, I added a clock and I moved the outputs into a generic register connected with the same clock.

```
    reg <= f1 & f2 & f3;

    reg0: entity work.gen_register(gen_register) generic map(8) port map(clk,'0',a,a_temp);
    reg1: entity work.gen_register(gen_register) generic map(8) port map(clk,'0',b,b_temp);
    reg2: entity work.gen_register(gen_register) generic map(3) port map(clk,'0',reg,outReg);

end architecture;
```

Figure 10: Connecting comparator with registers

3.3. Stage2

In this stage, I built a four-bit comparator and used it five times to compare all the bits starting from the sign-bit. The output of sign-bit comparator is inverted (if A (7) > B (7) then A is negative and less than B). The other comparators work normally.

3.3.1. Block Diagram

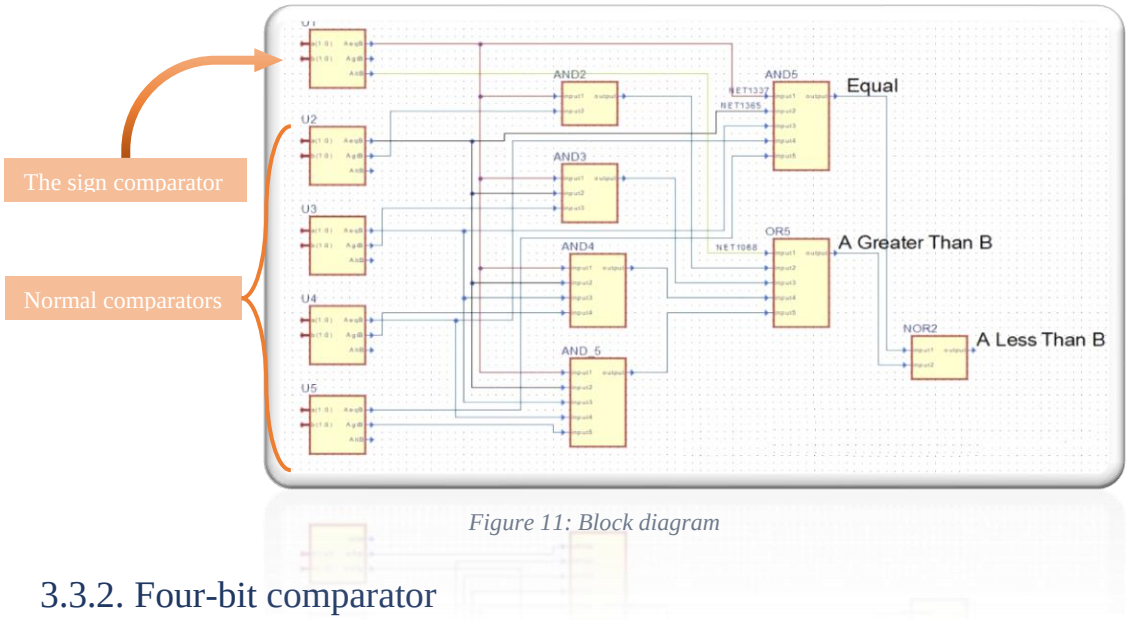


Figure 11: Block diagram

3.3.2. Four-bit comparator

I built a truth table for a four-bit comparator and made a k-map simplification to find the Boolean equation for (A=B, A>B, A<B) as followed:

	A1	*A0*	*B1*	*B0*	***A=B***	***A>B***	***A<B***
0	0	0	0	0	**1**	**0**	**0**
1	0	0	0	1	**0**	**0**	**1**
2	0	0	1	0	**0**	**0**	**1**
3	0	0	1	1	**0**	**0**	**1**
4	0	1	0	0	**0**	**1**	**0**
5	0	1	0	1	**1**	**0**	**0**
6	0	1	1	0	**0**	**0**	**1**
7	0	1	1	1	**0**	**0**	**1**
8	1	0	0	0	**0**	**1**	**0**
9	1	0	0	1	**0**	**1**	**0**
10	1	0	1	0	**1**	**0**	**0**
11	1	0	1	1	**0**	**0**	**1**
12	1	1	0	0	**0**	**1**	**0**
13	1	1	0	1	**0**	**1**	**0**
14	1	1	1	0	**0**	**1**	**0**
15	1	1	1	1	**1**	**0**	**0**

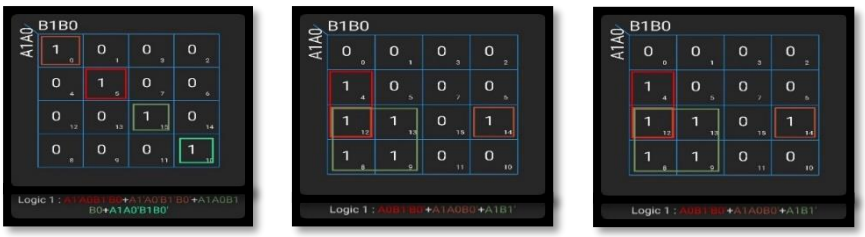


Figure 12: K-map simplification

For the equality:

```
eq0: entity work.delayed_AND_4(delayed_AND_4) port map (notA(1),notA(0),notB(1),notB(0),operation(0));
eq1: entity work.delayed_AND_4(delayed_AND_4) port map (notA(1),a(0),notB(1),b(0),operation(1));
eq2: entity work.delayed_AND_4(delayed_AND_4) port map (a(1),notA(0),b(1),notB(0),operation(2));
eq3: entity work.delayed_AND_4(delayed_AND_4) port map (a(1),a(0),b(1),b(0),operation(3));
    eq4: entity work.delayed_OR_4(delayed_OR_4) port map (operation(0),operation(1),operation(2),operation(3),AeqB);
```

Figure 13: Equality

For A greater than B:

```
agtb0: entity work.delayed_AND_3(delayed_AND_3) port map (a(0),notB(1),notB(0),operation(4));
agtb1: entity work.delayed_AND_3(delayed_AND_3) port map (a(1),a(0),notB(0),operation(5));
agtb2: entity work.delayed_AND(delayed_AND) port map (a(1),notB(1),operation(6));
    agtb3: entity work.delayed_OR_3(delayed_OR_3) port map (operation(4),operation(5),operation(6),AgtB);
```

Figure 14: Greater than

For A less than B

```
altb0: entity work.delayed_AND(delayed_AND) port map (notA(1),b(1),operation(7));
altb1: entity work.delayed_AND_3(delayed_AND_3) port map (notA(1),notA(0),b(0),operation(8));
altb2: entity work.delayed_AND_3(delayed_AND_3) port map (notA(0),b(1),b(0),operation(9));
    altb3: entity work.delayed_OR_3(delayed_OR_3) port map (operation(7),operation(8),operation(9),AltB);
```

Figure 16: Less than

The full code:

```
entity four_bit_comparator is
    port (a,b: in std_logic_vector(1 downto 0);
    AeqB,AgtB,AltB: out std_logic);
end entity;

architecture four_bit_comparator of four_bit_comparator is
signal notA,notB:std_logic_vector (2 downto 0);
signal operation: std_logic_vector (9 downto 0);
begin
    nA0: entity work.invertor(invertor) port map (a(0),notA(0));
    nA1: entity work.invertor(invertor) port map (a(1),notA(1));
    nB0: entity work.invertor(invertor) port map (b(0),notB(0));
    nB1: entity work.invertor(invertor) port map (b(1),notB(1));

    eq0: entity work.delayed_AND_4(delayed_AND_4) port map (notA(1),notA(0),notB(1),notB(0),operation(0));
    eq1: entity work.delayed_AND_4(delayed_AND_4) port map (notA(1),a(0),notB(1),b(0),operation(1));
    eq2: entity work.delayed_AND_4(delayed_AND_4) port map (a(1),notA(0),b(1),notB(0),operation(2));
    eq3: entity work.delayed_AND_4(delayed_AND_4) port map (a(1),a(0),b(1),b(0),operation(3));
        eq4: entity work.delayed_OR_4(delayed_OR_4) port map (operation(0),operation(1),operation(2),operation(3),AeqB);

    agtb0: entity work.delayed_AND_3(delayed_AND_3) port map (a(0),notB(1),notB(0),operation(4));
    agtb1: entity work.delayed_AND_3(delayed_AND_3) port map (a(1),a(0),notB(0),operation(5));
    agtb2: entity work.delayed_AND(delayed_AND) port map (a(1),notB(1),operation(6));
        agtb3: entity work.delayed_OR_3(delayed_OR_3) port map (operation(4),operation(5),operation(6),AgtB);

    altb0: entity work.delayed_AND(delayed_AND) port map (notA(1),b(1),operation(7));
    altb1: entity work.delayed_AND_3(delayed_AND_3) port map (notA(1),notA(0),b(0),operation(8));
    altb2: entity work.delayed_AND_3(delayed_AND_3) port map (notA(0),b(1),b(0),operation(9));
        altb3: entity work.delayed_OR_3(delayed_OR_3) port map (operation(7),operation(8),operation(9),AltB);
end architecture;
```

Figure 17: Full code

3.3.3. The full comparator

For the sign-bit, I entered it to a normal 4-bit comparator with zeros for the rest of bits. E.g.: 0A (7) and 0B (7) are the inputs of sign-bit comparator.

```
concatenation0 <= '0' & a_temp(7);
concatenation1 <= '0' & b_temp(7);
concatenation2 <= '0' & a_temp(6);
concatenation3 <= '0' & b_temp(6);
```

I built the full comparator structurally for (A = B, A > B). And for (A < B) I used NOR operation between (A = B, A > B) outputs which will give high output only when the others are low.

<if the numbers aren't equal, and A isn't greater than B, then A is less than B>.

```
g0: entity work.four_bit_comparator(four_bit_comparator) port map (concatenation0,concatenation1,a_eq_b(0),a_gt_b(0),a_lt_b);
g1: entity work.four_bit_comparator(four_bit_comparator) port map (concatenation2,concatenation3,a_eq_b(1),a_gt_b(1),open);
g2: entity work.four_bit_comparator(four_bit_comparator) port map (concatenation4,concatenation5,a_eq_b(2),a_gt_b(2),open);
g3: entity work.four_bit_comparator(four_bit_comparator) port map (concatenation6,concatenation7,a_eq_b(3),a_gt_b(3),open);
g4: entity work.four_bit_comparator(four_bit_comparator) port map (concatenation8,concatenation9,a_eq_b(4),a_gt_b(4),open);

f1: entity work.delayed_AND(delayed_AND) port map (a_eq_b(0),a_gt_b(1),temp(1));
f2: entity work.delayed_AND_3(delayed_AND_3) port map (a_eq_b(0),a_eq_b(1),a_gt_b(2),temp(2));
f3: entity work.delayed_AND_4(delayed_AND_4) port map (a_eq_b(0),a_eq_b(1),a_eq_b(2),a_gt_b(3),temp(3));
f4: entity work.delayed_AND_5(delayed_AND_5) port map (a_eq_b(0),a_eq_b(1),a_eq_b(2),a_eq_b(3),a_gt_b(4),temp(4));
f5: entity work.delayed_AND_5(delayed_AND_5) port map (a_eq_b(0),a_eq_b(1),a_eq_b(2),a_eq_b(3),a_eq_b(4),temp(5));
    AeqB <= temp(5);
f6: entity work.delayed_OR_5(delayed_OR_5) port map (a_lt_b,temp(1),temp(2),temp(3),temp(4),temp(6));
    AgtB <= temp(6);
f7: entity work.delayed_NOR(delayed_NOR) port map (temp(5),temp(6),AltB);
```

Figure 18

Finally, I connected the circuit to the register and the clock to make the system synchronized.

```
reg <= AeqB & AgtB & AltB;

reg0: entity work.gen_register(gen_register) generic map(8) port map(clk,'0',a,a_temp);
reg1: entity work.gen_register(gen_register) generic map(8) port map(clk,'0',b,b_temp);
reg2: entity work.gen_register(gen_register) generic map(3) port map(clk,'0',reg,outReg);
```

Figure 19

The full code:

```
entity eight_bit_comparator is
    port (a,b: in std_logic_vector(7 downto 0);
    AeqB,AgtB,AltB: inout std_logic;
    outReg: out std_logic_vector(2 downto 0);
    clk: in std_logic);
end entity;

architecture eight_bit_comparator of eight_bit_comparator is
signal temp: std_logic_vector(6 downto 0);
signal a_temp,b_temp: std_logic_vector(7 downto 0);
signal a_eq_b,a_gt_b: std_logic_vector (4 downto 0);
signal a_lt_b: std_logic;
signal reg: std_logic_vector(2 downto 0);
signal concatenation0,concatenation1,concatenation2,concatenation3,
       concatenation4,concatenation5,concatenation6,concatenation7,
       concatenation8,concatenation9: std_logic_vector (1 downto 0);
begin
    concatenation0 <= '0' & a_temp(7);
    concatenation1 <= '0' & b_temp(7);
    concatenation2 <= '0' & a_temp(6);
    concatenation3 <= '0' & b_temp(6);
    concatenation4 <= a_temp(5) & a_temp(4);   -- I could also do this by using a_temp(5 downto 4) inside the port map
    concatenation5 <= b_temp(5) & b_temp(4);
    concatenation6 <= a_temp(3) & a_temp(2);
    concatenation7 <= b_temp(3) & b_temp(2);
    concatenation8 <= a_temp(1) & a_temp(0);
    concatenation9 <= b_temp(1) & b_temp(0);

    g0: entity work.four_bit_comparator(four_bit_comparator) port map (concatenation0,concatenation1,a_eq_b(0),a_gt_b(0),a_lt_b);
    g1: entity work.four_bit_comparator(four_bit_comparator) port map (concatenation2,concatenation3,a_eq_b(1),a_gt_b(1),open);
    g2: entity work.four_bit_comparator(four_bit_comparator) port map (concatenation4,concatenation5,a_eq_b(2),a_gt_b(2),open);
    g3: entity work.four_bit_comparator(four_bit_comparator) port map (concatenation6,concatenation7,a_eq_b(3),a_gt_b(3),open);
    g4: entity work.four_bit_comparator(four_bit_comparator) port map (concatenation8,concatenation9,a_eq_b(4),a_gt_b(4),open);

    f1: entity work.delayed_AND(delayed_AND) port map (a_eq_b(0),a_gt_b(1),temp(1));
    f2: entity work.delayed_AND_3(delayed_AND_3) port map (a_eq_b(0),a_eq_b(1),a_gt_b(2),temp(2));
    f3: entity work.delayed_AND_4(delayed_AND_4) port map (a_eq_b(0),a_eq_b(1),a_eq_b(2),a_gt_b(3),temp(3));
    f4: entity work.delayed_AND_5(delayed_AND_5) port map (a_eq_b(0),a_eq_b(1),a_eq_b(2),a_eq_b(3),a_gt_b(4),temp(4));
    f5: entity work.delayed_AND_5(delayed_AND_5) port map (a_eq_b(0),a_eq_b(1),a_eq_b(2),a_eq_b(3),a_eq_b(4),temp(5));
        AeqB <= temp(5);
    f6: entity work.delayed_OR_5(delayed_OR_5) port map (a_lt_b,temp(1),temp(2),temp(3),temp(4),temp(6));
        AgtB <= temp(6);
    f7: entity work.delayed_NOR(delayed_NOR) port map (temp(5),temp(6),AltB);

    reg <= AeqB & AgtB & AltB;

    reg0: entity work.gen_register(gen_register) generic map(8) port map(clk,'0',a,a_temp);
    reg1: entity work.gen_register(gen_register) generic map(8) port map(clk,'0',b,b_temp);
    reg2: entity work.gen_register(gen_register) generic map(3) port map(clk,'0',reg,outReg);

end architecture;
```

Figure 20: Full code Stage2

3.4. Verification

In this step we have to check our system whether it works fine or not. So, I built test generator, analyzer and a test bench for each stage.

3.4.1. Test generator

To be sure that our circuits run correctly, I used simple “if” statement to calculate the expected (correct) answer.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity TestGenerator is
PORT(
clk: in std_logic:='0';
A,B: out std_logic_vector(7 downto 0):="00000000";
correctAnswer: out std_logic_vector(2 downto 0):="000");
end entity;

architecture generator of TestGenerator is
signal possible_A,possible_B: std_logic_vector(7 downto 0):="00000000";
begin
A<=possible_A;
B<=possible_B;
-- this is a process to calculate the expected value for the comparator
process (possible_A,possible_B)
variable answer: std_logic_vector(2 downto 0):="000";
begin
if (possible_A = possible_B) then
    answer:="100";
    elsif (possible_A > possible_B) then
        answer:="010";
    else
        answer:="001";
end if;
correctAnswer <= answer;
end process;
```

Figure 21: Test generator

I also built a process work as test generator that generate values for the inputs to find the answers that we are supposed to get corresponding to them.

```
-- it will change the value using loops over all possible input values
process
begin
for i in 0 to 9 loop
    for j in 0 to 9 loop
        for K in 0 to 9 loop
            for L in 0 to 9 loop
                possible_A(7 downto 4) <= conv_std_logic_vector(i,4);
                possible_A(3 downto 0) <= conv_std_logic_vector(j,4);
                possible_B(7 downto 4) <= conv_std_logic_vector(K,4);
                possible_B(3 downto 0) <= conv_std_logic_vector(L,4);
wait until rising_edge(clk);
            end loop;
        end loop;
    end loop;
end loop;
wait;
end process;
```

Figure 22: All possible values

3.4.2. Result analyzer

This part aims to print an error message on the terminal to let the user know when wrong output appears. I used “assert” statement to compare circuit outcome with the correct (expected) answer. Then I used severity of type ERROR with a report statement. Finally, I added a temp value that will take the correct answer after the delay each circuit needs to get the right answer.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity ResultAnalyser is
port(clk: in std_logic:='0';
correctAnswer,circuitOutcome: in std_logic_vector(2 downto 0):="000");
end entity;

architecture analyser1 of ResultAnalyser is
signal tmp:std_logic_vector(2 downto 0);
begin
-- it will check if the calculated answer is same as expected answer
process (clk)
begin
    if rising_edge(clk) then
            tmp <= correctAnswer after 80 ns;
            assert (circuitOutcome = tmp)
            report "The result you has is not as expected ==> there's an ERROR"
            severity ERROR;
    end if;
end process;
end architecture;
--------------------------------------------------------------------------------
architecture analyser2 of ResultAnalyser is
signal tmp:std_logic_vector(2 downto 0);
begin
-- it will check if the calculated answer is same as expected answer
process (clk)
begin
    if rising_edge(clk) then
            tmp <= correctAnswer after 330 ns;
            assert (circuitOutcome = tmp)
            report "The result you has is not as expected ==> there's an ERROR"
            severity ERROR;
    end if;
end process;
end architecture;
```

Figure 23: Result analyzer

3.4.3. Built-in self-test

In this entity, I combined the test generator with result analyzer connected with the clock which will take 2*(time needed to get the right answer). The test generator will generate all possible values for A and B and take the correct result for each one of them. Then, the tested values of A and B were sent to the designed circuit. Finally, the output of the two former levels were sent to result analyzer to compare them and print error message when needed.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity BIST is
end BIST;
-------------------------------------------------
----- testing for stage1
architecture stage1 of BIST is
signal clk: std_logic:='0';
signal A,B: std_logic_vector(7 downto 0):="00000000";
signal correctAnswer,circuitOutcome: std_logic_vector(2 downto 0):="000";
begin
-- 40 * 2
clk <= not clk after 80 ns;
    g0: entity work.TestGenerator(generator) port map(clk, A, B, correctAnswer);
    g1: entity work.eight_bit_comparator(eight_bit_comparator) port map(A, B,circuitOutcome(2),circuitOutcome(1),circuitOutcome(0),clk=>clk);
    g2: entity work.ResultAnalyser(analyser1) port map(clk, correctAnswer, circuitOutcome);
end architecture;

-------------------------------------------------
--------- testing for stage2
architecture stage2 of BIST is
signal clk: std_logic:='0';
signal A,B: std_logic_vector(7 downto 0):="00000000";
signal correctAnswer,circuitOutcome: std_logic_vector(2 downto 0):="000";
begin
-- 165 * 2
clk <= not clk after 330 ns;
    g0: entity work.TestGenerator(generator) port map(clk, A, B, correctAnswer);
    g1: entity work.comparator(comparator) port map(A, B,circuitOutcome(2),circuitOutcome(1),circuitOutcome(0),clk=>clk);
    g2: entity work.ResultAnalyser(analyser2) port map(clk, correctAnswer, circuitOutcome);
end architecture;
```

Figure 24: BIST

4. Results

4.1. Stage1

With the minimum time needed to avoid delay problems and errors:

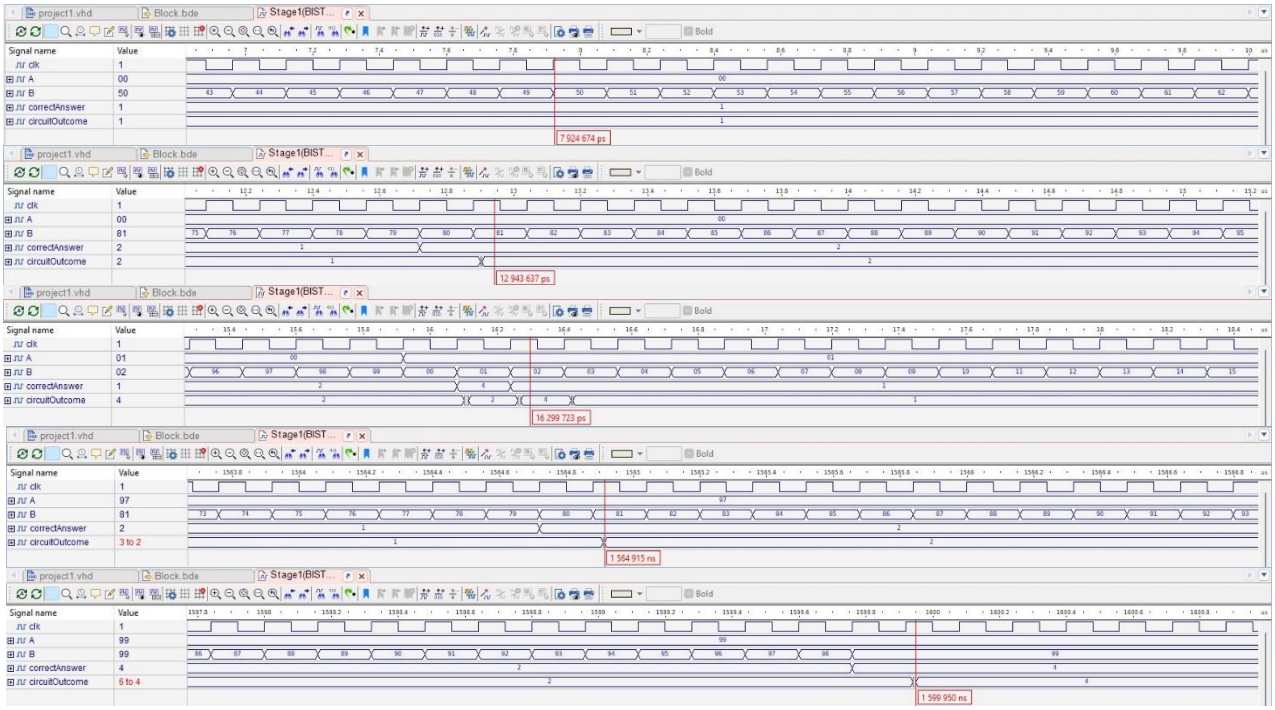


Figure 25: Results of stage 1 with correct clock

With clock less than needed:

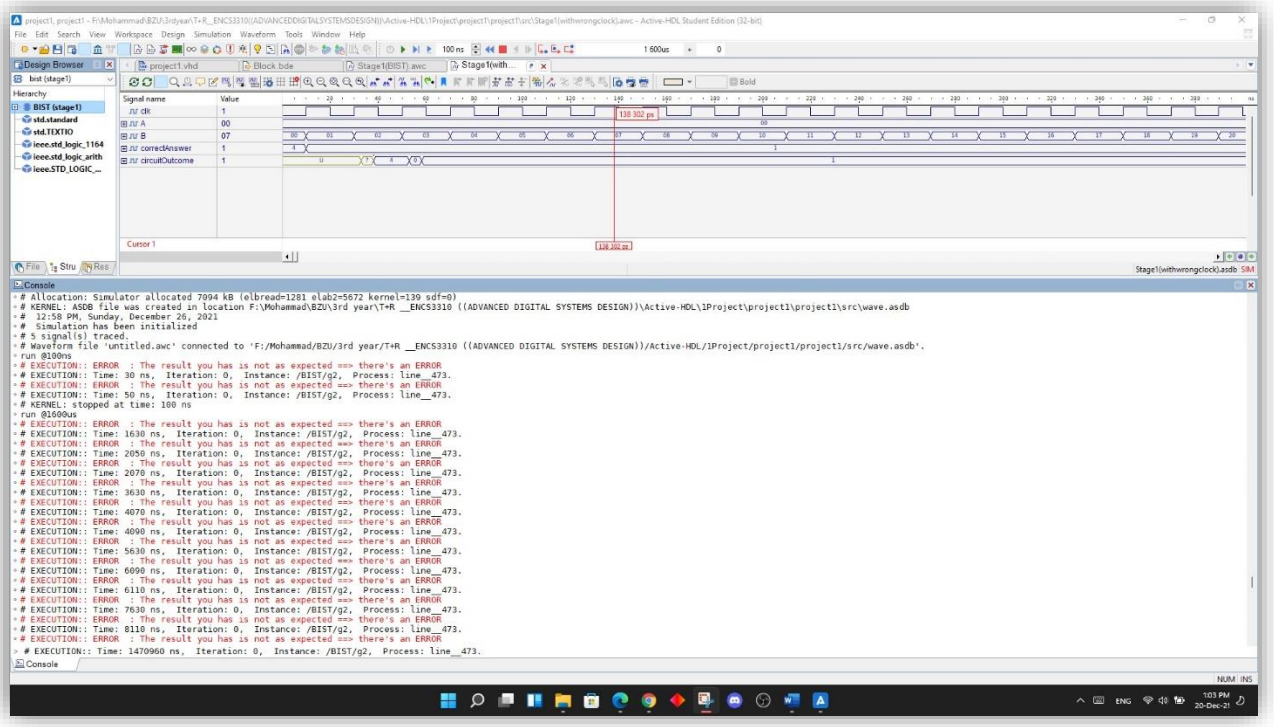


Figure 26: Results of stage 1 with wrong clock

4.2. Stage2

With the minimum time needed to avoid delay problems and errors:

Figure 27: Results of stage 2 with correct clock

With clock less than needed:

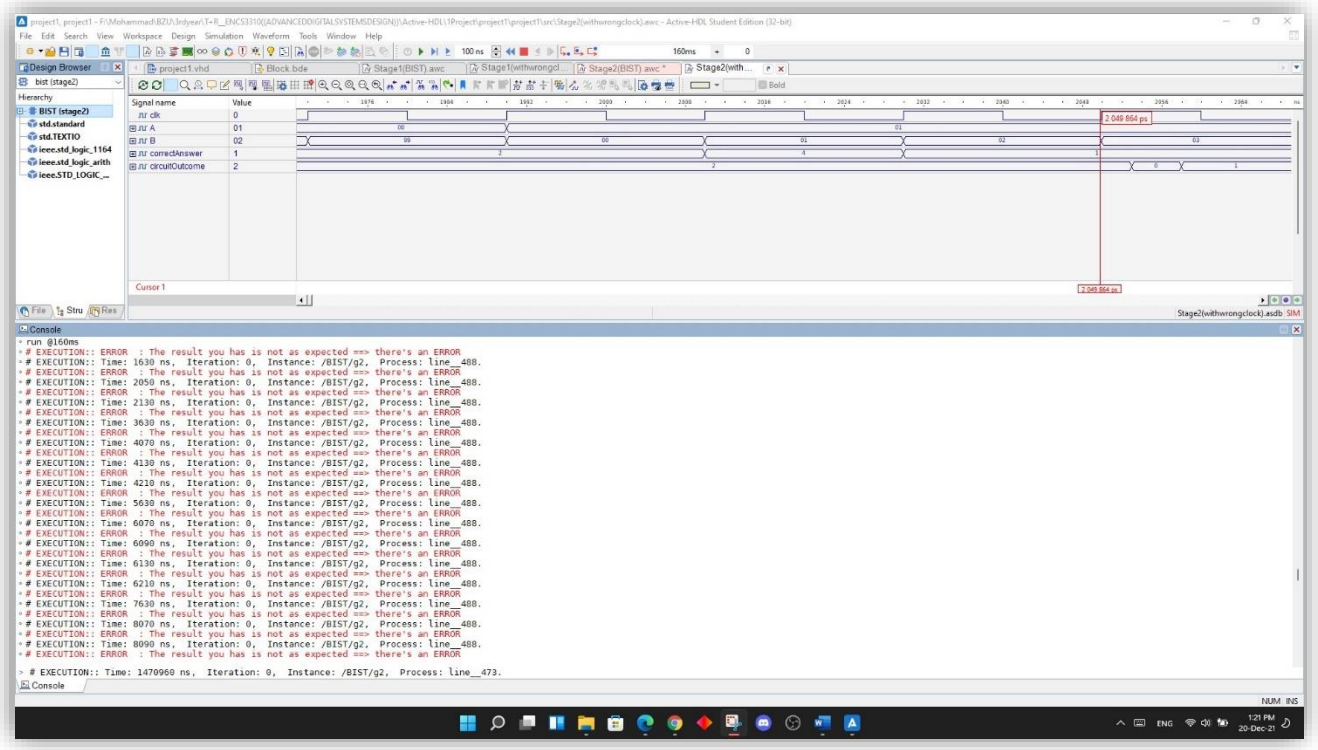


Figure 28: Results of stage 2 with wrong clock

5. Conclusion and future works

In this project, I learned how to design comparator with VHDL language and simulate my design using (Aldec Active-HDL) software. I designed the comparator structurally in two different ways: using a full adder/subtractor and using magnitude comparator built by using truth table and k-map simplification. Both designs were built using basic logic gates with special delays.

I wrote both concurrent and sequential codes. I also dealt with generic parameters, processes and other skills I learned. I learned how to make a system synchronized by using registers connected with the same clock as our circuits. I also faced some problems in determine the correct delay needed.

I think I could enhance my project in several parts like building a library for the basic logic gates and include their components into my main project. I also think that I could decrease the delay if I simplified my code more. Stage2 took more time than Stage1 to give the right output.

Finally, I am satisfied with what I learned in VHDL language. And I think that I have done a good job in this project. I think that this course in general helps a lot in hardware branch which is my passion. I hope that I can learn more about VHDL language and digital systems.

6. Appendix

```
library ieee;
use ieee.std_logic_1164.all;

entity invertor is
    port (input : in std_logic;
    output : out std_logic);
end entity;

architecture invertor of invertor is
begin
    output <= not input after 2ns;
end architecture;

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

library ieee;
use ieee.std_logic_1164.all;

entity delayed_NAND is
    port (input1,input2 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_NAND of delayed_NAND is
begin
    output <= input1 NAND input2 after 5ns;
end architecture;

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

library ieee;
use ieee.std_logic_1164.all;

entity delayed_NOR is
    port (input1,input2 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_NOR of delayed_NOR is
begin
    output <= input1 NOR input2 after 5ns;
end architecture;

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

library ieee;
use ieee.std_logic_1164.all;

entity delayed_AND is
    port (input1,input2 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_AND of delayed_AND is
begin
    output <= input1 AND input2 after 7ns;
end architecture;
--************************************************************
library ieee;
```

```
use ieee.std_logic_1164.all;

entity delayed_AND_3 is
    port (input1,input2,input3 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_AND_3 of delayed_AND_3 is
begin
    output <= input1 AND input2 AND input3 after 7ns;
end architecture;
--************************************************************
library ieee;
use ieee.std_logic_1164.all;

entity delayed_AND_4 is
    port (input1,input2,input3,input4 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_AND_4 of delayed_AND_4 is
begin
    output <= input1 AND input2 AND input3 AND input4 after 7ns;
end architecture;
--************************************************************
library ieee;
use ieee.std_logic_1164.all;

entity delayed_AND_5 is
    port (input1,input2,input3,input4,input5 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_AND_5 of delayed_AND_5 is
begin
    output <= input1 AND input2 AND input3 AND input4 AND input5 after 7ns;
end architecture;

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

library ieee;
use ieee.std_logic_1164.all;

entity delayed_OR is
    port (input1,input2 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_OR of delayed_OR is
begin
    output <= input1 OR input2 after 7ns;
end architecture;
--************************************************************
library ieee;
use ieee.std_logic_1164.all;

entity delayed_OR_3 is
    port (input1,input2,input3 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_OR_3 of delayed_OR_3 is
```

```
begin
    output <= input1 OR input2 OR input3 after 7ns;
end architecture;
--**************************************************************
library ieee;
use ieee.std_logic_1164.all;

entity delayed_OR_4 is
    port (input1,input2,input3,input4 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_OR_4 of delayed_OR_4 is
begin
    output <= input1 OR input2 OR input3 OR input4 after 7ns;
end architecture;
--**************************************************************
library ieee;
use ieee.std_logic_1164.all;

entity delayed_OR_5 is
    port (input1,input2,input3,input4,input5 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_OR_5 of delayed_OR_5 is
begin
    output <= input1 OR input2 OR input3 OR input4 OR input5 after 7ns;
end architecture;

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

library ieee;
use ieee.std_logic_1164.all;

entity delayed_XNOR is
    port (input1,input2 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_XNOR of delayed_XNOR is
begin
    output <= input1 XNOR input2 after 9ns;
end architecture;

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

library ieee;
use ieee.std_logic_1164.all;

entity delayed_XOR is
    port (input1,input2 : in std_logic;
    output : out std_logic);
end entity;

architecture delayed_XOR of delayed_XOR is
begin
    output <= input1 XOR input2 after 12ns;
end architecture;

------------------------------------------------------------
-- 1-bir register
```

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
port (D,clk ,reset: in std_logic;
 Q: out std_logic);
end entity;

architecture dff of dff is
begin
process(clk, reset)
begin
    if (reset = '1') then
        Q<='0';
    elsif (rising_edge(clk)) then
        if (D='1') then
            Q<= '1';
        elsif(D='0') then
            Q<= '0';
        end if;
    end if;
end process;
end architecture;

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

-- n-bir register ==> multiple DFFs

library ieee;
use ieee.std_logic_1164.all;
entity gen_register is
GENERIC (N : positive := 8);
PORT (clk, reset : in STD_LOGIC;
        D: in std_logic_vector(n-1 downto 0);
        Q: out std_logic_vector(n-1 downto 0));
end entity gen_register;
architecture gen_register of gen_register is
begin
    gen : for i in 0 to n - 1 generate
    gen1 : entity work.dff(dff) port map (D(i),clk,reset,Q(i));
    end generate;
end architecture;

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

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity F_Adder is
    port (a,b,c_in: in std_logic;
    sum,carry: out std_logic);
end entity;

architecture F_Adder of F_Adder is
signal sum0,carry0,x: std_logic;
begin
    g0: entity work.delayed_XOR(delayed_XOR) port map (a,b,sum0);
    g1: entity work.delayed_XOR(delayed_XOR) port map (sum0,c_in,sum);
    g2: entity work.delayed_AND(delayed_AND) port map (a,b,carry0);
    g3: entity work.delayed_AND(delayed_AND) port map (sum0,c_in,x);
    g4: entity work.delayed_OR(delayed_OR) port map (carry0,x,carry);
end architecture;
```

```
----------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity eight_bit_adder is
    port (a,b: in std_logic_vector(7 downto 0);
    c_in: std_logic;
    sum: out std_logic_vector (7 downto 0);
    carry,overflow: out std_logic
    );
end entity;

architecture eight_bit_adder of eight_bit_adder is
signal ones_complement: std_logic_vector (7 downto 0);
signal c: std_logic_vector (8 downto 0);
begin
    c(0) <= c_in;
    gen1: for i in 0 to 7 generate
          begin
              g: entity work.delayed_XOR(delayed_XOR) port map
(b(i),c_in,ones_complement(i));
              f: entity work.F_Adder(F_Adder) port map
(a(i),ones_complement(i),c(i),sum(i),c(i+1));
            end generate;
    carry <= c(8);
    h: entity work.delayed_XOR(delayed_XOR) port map (c(7),c(8),overflow);
end architecture;

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

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;


entity comparator is
    port (a,b: in std_logic_vector (7 downto 0);
    f1,f2,f3: inout std_logic:='0';
    outReg: out std_logic_vector(2 downto 0);
    clk: in std_logic);
end entity;

architecture comparator of comparator is
signal value,a_temp,b_temp: std_logic_vector(7 downto 0);
signal V_flag,Carry_flag,Not_Z_flag,test: std_logic;
signal reg: std_logic_vector(2 downto 0);

begin
Not_Z_flag <= '0' when value = "00000000" else '1';
test <= '1' when V_flag = value(7) else '0';
    g0: entity work.eight_bit_adder(eight_bit_adder) port map
(a_temp,b_temp,'1',value,Carry_flag,V_flag);
    g1: entity work.delayed_XOR(delayed_XOR) port map (value(7),V_flag,f3);
    g2: entity work.delayed_AND(delayed_AND) port map (Not_Z_flag,test,f2);

    f1 <= '1' when value = "00000000" else '0';

    reg <= f1 & f2 & f3;
```

```
    reg0: entity work.gen_register(gen_register) generic map(8) port
map(clk,'0',a,a_temp);
    reg1: entity work.gen_register(gen_register) generic map(8) port
map(clk,'0',b,b_temp);
    reg2: entity work.gen_register(gen_register) generic map(3) port
map(clk,'0',reg,outReg);

end architecture;

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

library ieee;
use ieee.std_logic_1164.all;

entity four_bit_comparator is
    port (a,b: in std_logic_vector(1 downto 0);
    AeqB,AgtB,AltB: out std_logic);
end entity;

architecture four_bit_comparator of four_bit_comparator is
signal notA,notB:std_logic_vector (2 downto 0);
signal operation: std_logic_vector (9 downto 0);
begin
    nA0: entity work.invertor(invertor) port map (a(0),notA(0));
    nA1: entity work.invertor(invertor) port map (a(1),notA(1));
    nB0: entity work.invertor(invertor) port map (b(0),notB(0));
    nB1: entity work.invertor(invertor) port map (b(1),notB(1));

    eq0: entity work.delayed_AND_4(delayed_AND_4) port map
(notA(1),notA(0),notB(1),notB(0),operation(0));
    eq1: entity work.delayed_AND_4(delayed_AND_4) port map
(notA(1),a(0),notB(1),b(0),operation(1));
    eq2: entity work.delayed_AND_4(delayed_AND_4) port map
(a(1),notA(0),b(1),notB(0),operation(2));
    eq3: entity work.delayed_AND_4(delayed_AND_4) port map
(a(1),a(0),b(1),b(0),operation(3));
        eq4: entity work.delayed_OR_4(delayed_OR_4) port map
(operation(0),operation(1),operation(2),operation(3),AeqB);

    agtb0: entity work.delayed_AND_3(delayed_AND_3) port map
(a(0),notB(1),notB(0),operation(4));
    agtb1: entity work.delayed_AND_3(delayed_AND_3) port map
(a(1),a(0),notB(0),operation(5));
    agtb2: entity work.delayed_AND(delayed_AND) port map
(a(1),notB(1),operation(6));
        agtb3: entity work.delayed_OR_3(delayed_OR_3) port map
(operation(4),operation(5),operation(6),AgtB);

    altb0: entity work.delayed_AND(delayed_AND) port map
(notA(1),b(1),operation(7));
    altb1: entity work.delayed_AND_3(delayed_AND_3) port map
(notA(1),notA(0),b(0),operation(8));
    altb2: entity work.delayed_AND_3(delayed_AND_3) port map
(notA(0),b(1),b(0),operation(9));
        altb3: entity work.delayed_OR_3(delayed_OR_3) port map
(operation(7),operation(8),operation(9),AltB);
end architecture;

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

library ieee;
use ieee.std_logic_1164.all;
```

```
entity eight_bit_comparator is
    port (a,b: in std_logic_vector(7 downto 0);
    AeqB,AgtB,AltB: inout std_logic;
    outReg: out std_logic_vector(2 downto 0);
    clk: in std_logic);
end entity;

architecture eight_bit_comparator of eight_bit_comparator is
signal temp: std_logic_vector(6 downto 0);
signal a_temp,b_temp: std_logic_vector(7 downto 0);
signal a_eq_b,a_gt_b: std_logic_vector (4 downto 0);
signal a_lt_b: std_logic;
signal reg: std_logic_vector(2 downto 0);
signal concatenation0,concatenation1,concatenation2,concatenation3,
       concatenation4,concatenation5,concatenation6,concatenation7,
       concatenation8,concatenation9: std_logic_vector (1 downto 0);
begin
    concatenation0 <= '0' & a_temp(7);
    concatenation1 <= '0' & b_temp(7);
    concatenation2 <= '0' & a_temp(6);
    concatenation3 <= '0' & b_temp(6);
    concatenation4 <= a_temp(5) & a_temp(4);   -- I could also do this by
using a_temp(5 downto 4) inside the port map
    concatenation5 <= b_temp(5) & b_temp(4);
    concatenation6 <= a_temp(3) & a_temp(2);
    concatenation7 <= b_temp(3) & b_temp(2);
    concatenation8 <= a_temp(1) & a_temp(0);
    concatenation9 <= b_temp(1) & b_temp(0);

    g0: entity work.four_bit_comparator(four_bit_comparator) port map
(concatenation0,concatenation1,a_eq_b(0),a_gt_b(0),a_lt_b);
    g1: entity work.four_bit_comparator(four_bit_comparator) port map
(concatenation2,concatenation3,a_eq_b(1),a_gt_b(1),open);
    g2: entity work.four_bit_comparator(four_bit_comparator) port map
(concatenation4,concatenation5,a_eq_b(2),a_gt_b(2),open);
    g3: entity work.four_bit_comparator(four_bit_comparator) port map
(concatenation6,concatenation7,a_eq_b(3),a_gt_b(3),open);
    g4: entity work.four_bit_comparator(four_bit_comparator) port map
(concatenation8,concatenation9,a_eq_b(4),a_gt_b(4),open);

    f1: entity work.delayed_AND(delayed_AND) port map
(a_eq_b(0),a_gt_b(1),temp(1));
    f2: entity work.delayed_AND_3(delayed_AND_3) port map
(a_eq_b(0),a_eq_b(1),a_gt_b(2),temp(2));
    f3: entity work.delayed_AND_4(delayed_AND_4) port map
(a_eq_b(0),a_eq_b(1),a_eq_b(2),a_gt_b(3),temp(3));
    f4: entity work.delayed_AND_5(delayed_AND_5) port map
(a_eq_b(0),a_eq_b(1),a_eq_b(2),a_eq_b(3),a_gt_b(4),temp(4));
    f5: entity work.delayed_AND_5(delayed_AND_5) port map
(a_eq_b(0),a_eq_b(1),a_eq_b(2),a_eq_b(3),a_eq_b(4),temp(5));
        AeqB <= temp(5);
    f6: entity work.delayed_OR_5(delayed_OR_5) port map
(a_lt_b,temp(1),temp(2),temp(3),temp(4),temp(6));
        AgtB <= temp(6);
    f7: entity work.delayed_NOR(delayed_NOR) port map (temp(5),temp(6),AltB);

    reg <= AeqB & AgtB & AltB;

    reg0: entity work.gen_register(gen_register) generic map(8) port
map(clk,'0',a,a_temp);
```

```
    reg1: entity work.gen_register(gen_register) generic map(8) port
map(clk,'0',b,b_temp);
    reg2: entity work.gen_register(gen_register) generic map(3) port
map(clk,'0',reg,outReg);

end architecture;

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


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity TestGenerator is
PORT(
clk: in std_logic:='0';
A,B: out std_logic_vector(7 downto 0):="00000000";
correctAnswer: out std_logic_vector(2 downto 0):="000");
end entity;

architecture generator of TestGenerator is
signal possible_A,possible_B: std_logic_vector(7 downto 0):="00000000";
begin
A<=possible_A;
B<=possible_B;
-- this is a process to calculate the expected value for the comparator
process (possible_A,possible_B)
variable answer: std_logic_vector(2 downto 0):="000";
begin
if (possible_A = possible_B) then
    answer:="100";
    elsif (possible_A > possible_B) then
        answer:="010";
    else
        answer:="001";
end if;
correctAnswer <= answer;
end process;

-- it will change the value using loops over all possible input values
process
begin
for i in 0 to 9 loop
    for j in 0 to 9 loop
        for K in 0 to 9 loop
            for L in 0 to 9 loop
                possible_A(7 downto 4) <= conv_std_logic_vector(i,4);
                possible_A(3 downto 0) <= conv_std_logic_vector(j,4);
                possible_B(7 downto 4) <= conv_std_logic_vector(K,4);
                possible_B(3 downto 0) <= conv_std_logic_vector(L,4);
wait until rising_edge(clk);
            end loop;
        end loop;
    end loop;
end loop;
wait;
end process;
end architecture;
-------------------------------------------------------------------------------
-
```

```
-- analyzing result
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity ResultAnalyser is
port(clk: in std_logic:='0';
correctAnswer,circuitOutcome: in std_logic_vector(2 downto 0):="000");
end entity;

architecture analyser1 of ResultAnalyser is
signal tmp:std_logic_vector(2 downto 0);
begin
-- it will check if the calculated answer is same as expected answer
process (clk)
begin
    if rising_edge(clk) then
            tmp <= correctAnswer after 80 ns;
            assert (circuitOutcome = tmp)
            report "The result you has is not as expected ==> there's an
ERROR"
            severity ERROR;
    end if;
end process;
end architecture;
--------------------------------------------------------------------------
-
architecture analyser2 of ResultAnalyser is
signal tmp:std_logic_vector(2 downto 0);
begin
-- it will check if the calculated answer is same as expected answer
process (clk)
begin
    if rising_edge(clk) then
            tmp <= correctAnswer after 330 ns;
            assert (circuitOutcome = tmp)
            report "The result you has is not as expected ==> there's an
ERROR"
            severity ERROR;
    end if;
end process;
end architecture;
-------- BIST
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity BIST is
end BIST;
------------------------------------------------
----- testing for stage1
architecture stage1 of BIST is
signal clk: std_logic:='0';
signal A,B: std_logic_vector(7 downto 0):="00000000";
signal correctAnswer,circuitOutcome: std_logic_vector(2 downto 0):="000";
begin
-- 40 * 2
clk <= not clk after 80 ns;
    g0: entity work.TestGenerator(generator) port map(clk, A, B,
correctAnswer);
    g1: entity work.eight_bit_comparator(eight_bit_comparator) port map(A,
B,circuitOutcome(2),circuitOutcome(1),circuitOutcome(0),clk=>clk);
```

```
    g2: entity work.ResultAnalyser(analyser1) port map(clk, correctAnswer,
circuitOutcome);
end architecture;

-----------------------------------------------
--------- testing for stage2
architecture stage2 of BIST is
signal clk: std_logic:='0';
signal A,B: std_logic_vector(7 downto 0):="00000000";
signal correctAnswer,circuitOutcome: std_logic_vector(2 downto 0):="000";
begin
-- 165 * 2
clk <= not clk after 330 ns;
    g0: entity work.TestGenerator(generator) port map(clk, A, B,
correctAnswer);
    g1: entity work.comparator(comparator) port map(A,
B,circuitOutcome(2),circuitOutcome(1),circuitOutcome(0),clk=>clk);
    g2: entity work.ResultAnalyser(analyser2) port map(clk, correctAnswer,
circuitOutcome);
end architecture;
```

7. References

https://www.researchgate.net/publication/283037309_Design_implementation_and_performance_comparison_of_multiplier_topologies_in_power-delay_space [Accessed on December 16, 2021 at 2:10 PM]

Digital Comparator and Magnitude Comparator (electronicshub.org) [Accessed on December 16, 2021 at 2:10 PM]

https://www.ics.uci.edu/~jmoorkan/vhdlref/waits.html [Accessed on December 24, 2021 at 12:46 AM]