Faculty of Engineering & Technology
Electrical & Computer Engineering Department

ADVANCED DIGITAL SYSTEMS DESIGN
-ENCS3310-

First Semester 2021-2022

Course Project

8-Bit Comparator for Signed Number

Prepared by: Eyab Ghifari

ID: 1190999

Instructor's Name: Dr. Abdallatif Abuissa

Section: 2
Date: 21-12-2021

1. Brief introduction and background

The aim of this project to design an 8-bit Comparator for signed 2's complement representation numbers structurally, from basic logic gates with specific delays, and then blocks of the comparator itself assembled from these gates, after that built a complete code for functional verification in order to test out the results of all possible values entering the comparator.

Comparators are very common elements in digital systems. Every comparison in a system needs a comparator. Determining if the counter achieve the requested value, controlling if a sensor value exceed the threshold value are very well known example usages of comparators.

Contents

Table of Figures

2. Design philosophy

- **Gates**

  A logic gate is an idealized model of computation or physical electronic device implementing a Boolean function, a logical operation performed on one or more binary inputs that produces a single binary output. Depending on the context, the term may refer to an ideal logic gate, one that has for instance zero rise time and unlimited fan-out, or it may refer to a non-ideal physical device.

  The first step in the project was to design the gates structurally with the given delays.

GATE	DELAY
INVERTER	**2 NS**
NAND	**5 NS**
NOR	**5 NS**
AND	**7 NS**
OR	**7 NS**
XNOR	**9 NS**
XOR	**12 NS**

- **Carry Look Ahead Adder**

  In ripple carry adders, for each adder block, the two bits that are to be added are available instantly. However, each adder block waits for the carry to arrive from its previous block. Therefore, it is not possible to generate the sum and carry of any block until the input carry is known. The block waits for the block to produce its carry. Therefore, there will be a considerable time delay, which is carry propagation delay, the look ahead structurally will look like figure below.

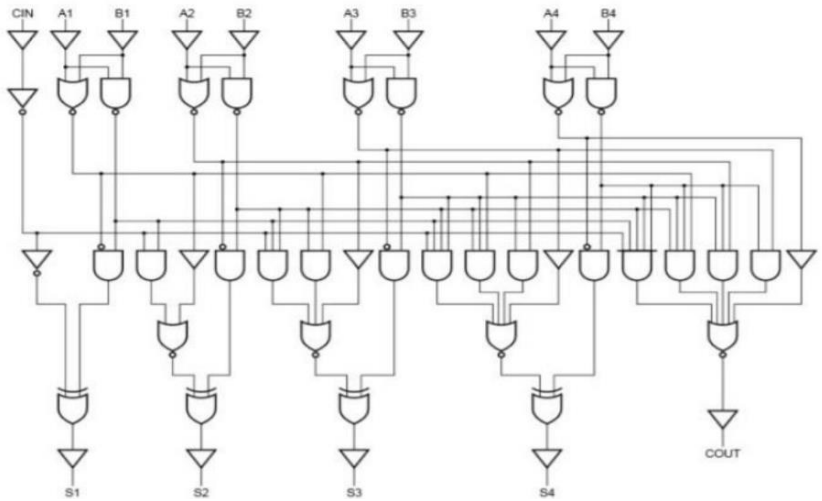


Figure 1 : Logic circuit Carry Look Ahead Adder

In this project, for comparator implementation in two stages of complexity, first by using the Look Ahead Adder and negative, overflow and zero flags, and second by using the magnitude comparator.

2.1. Stage 1

For the first one, a substractor is used. If A and B numbers are compared, B is subtracted from A. If the result is negative, then B is greater than A. If it is positive, it means that A is greater than B. If the result is 0, A is equal to B. For the subtraction a carry look ahead adder is used. For A-B operation B is inverted and C0 is given as 1.

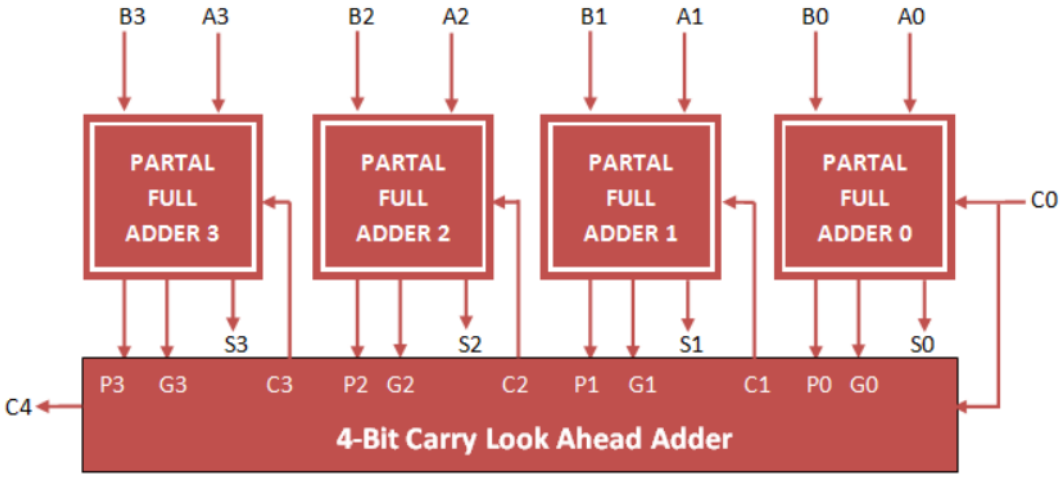


Figure 2 : 4-bit Carry Look Ahead Adder

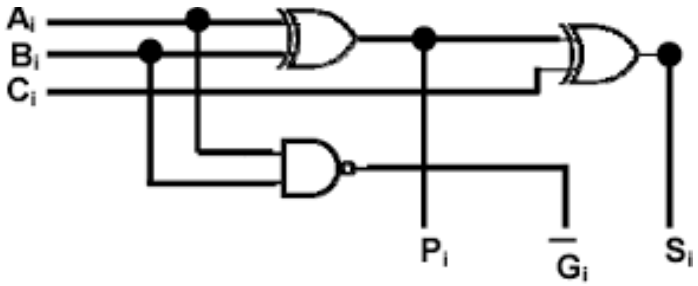


Figure 3 : partial full adder

We can determine the three outputs of the signed comparator circuit using those three equations:

- A equal B = nor(sub0,sub1,sub2,...sub7)
- A less B = (((a(7) and b(7)) or (not a(7) and not b(7))) and sub_out(7)) or (a(7) and b(7)')
- A grater B = (A less B) nor (A equal B)

Latency (longest path) = 96 ns.

2.2. Stage 2

For the second method two cascading 4-bit magnitude comparators are used.

A magnitude digital Comparator is 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. We 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.

Figure 4 : n- bit Comparator

- **1-Bit Magnitude Comparator**

A comparator used to compare two bits is called a single bit comparator. 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.

The truth table for a 1-bit comparator is given below:

A	B	A<B	A=B	A>B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Figure 5 : Truth table of 1 bit comparator

From the previous truth table, we are able to get the Boolean expressions and therefore the digital circuit that represents this comparator, which looks as follows:

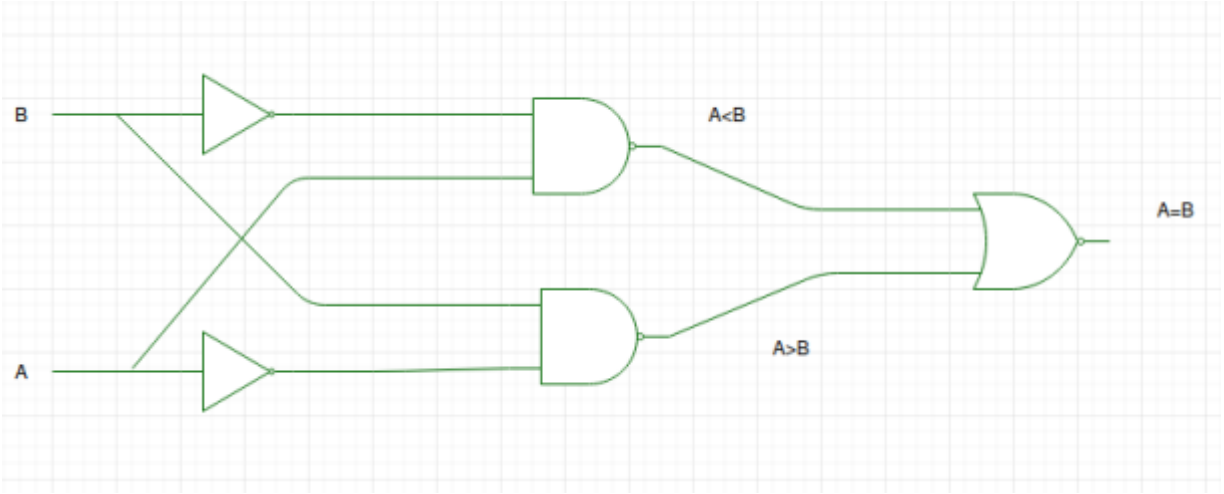


Figure 6 : 2 bit comparator

Keep in mind there are many variations of this circuit, each depends on its' own Boolean expression.

- **2-Bit Magnitude Comparator**

A comparator used to compare two binary numbers each of two bits is called a 2-bit Magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to and greater than between two binary numbers.

It's implemented in the same way of getting the truth table and using the k-map method to get its Boolean expression in order to end up with the following digital circuit.

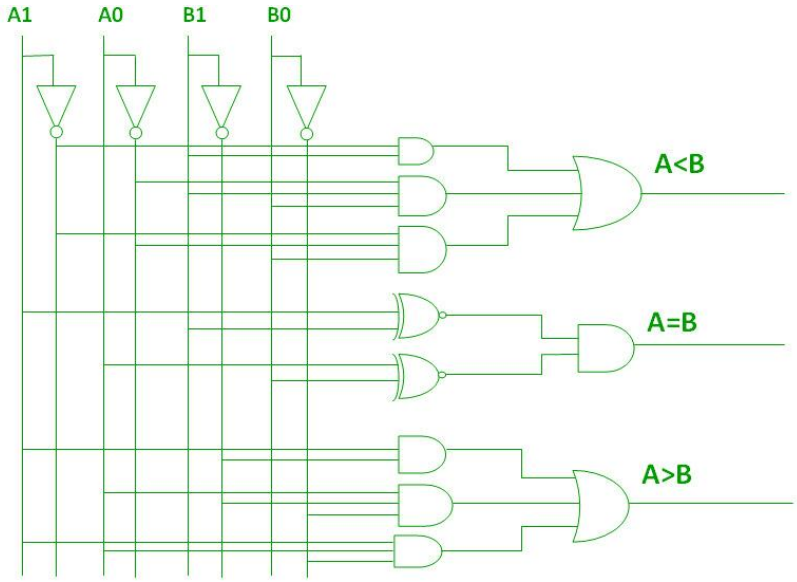


Figure 7 : 2 bit magnitude comparator

➢ 4-Bit Magnitude Comparator

The 4 –bit magnitude comparator is assembled in the same way, but it will be skipped because its implementation goes the same way all over.

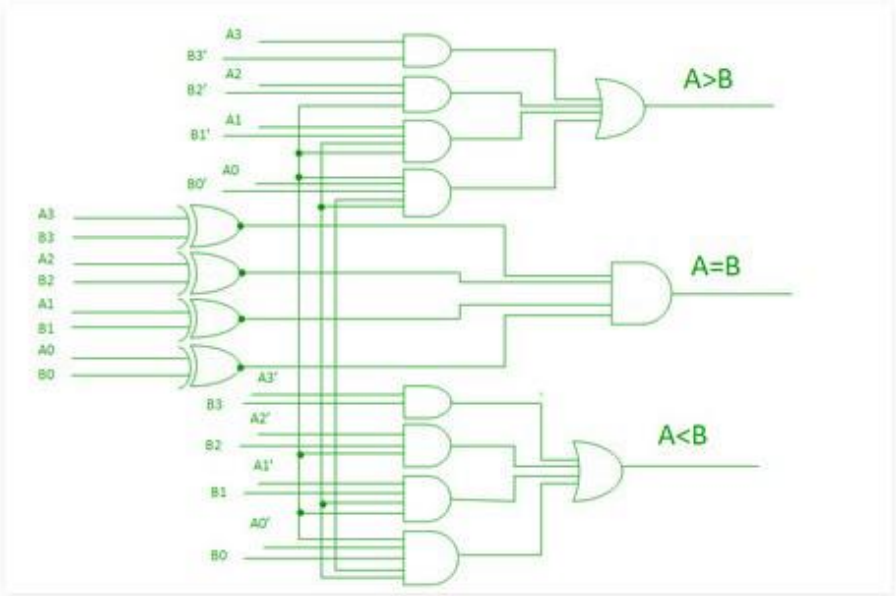


Figure 8 : Logic circuit of a 4-bit magnitude comparator.

➢ Cascading Comparator

A comparator performing the comparison operation to more than four bits by cascading two or more 4-bit comparators is called cascading comparator. When two comparators are to be cascaded, the outputs of the lower-order comparator are connected to corresponding inputs of the higher-order comparator.

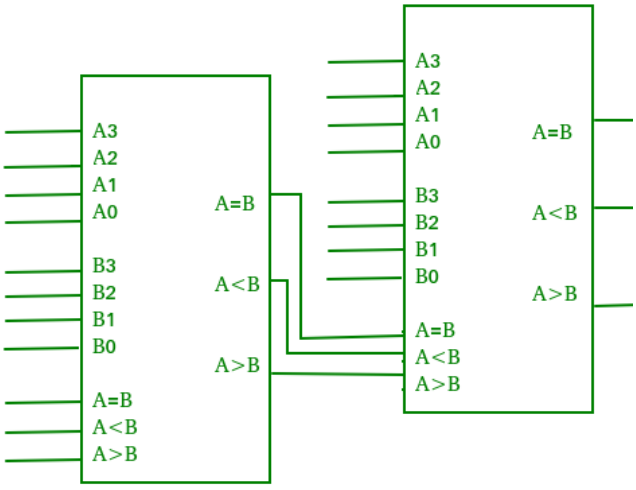
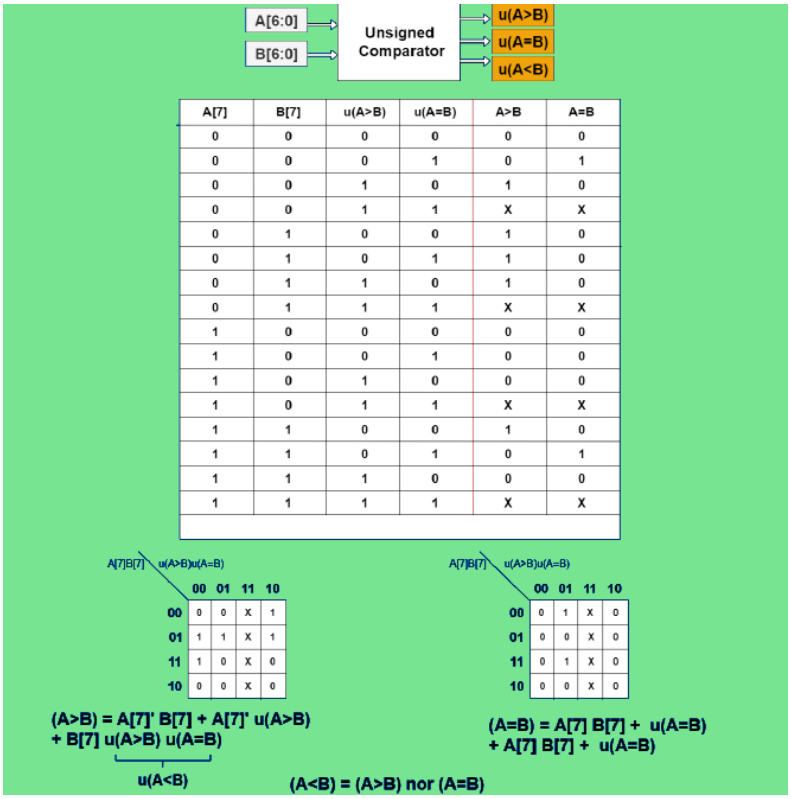


Figure 9 : Block design of the cascading 4-bit magnitude

The magnitude comparator is converted to the signed comparator by the design given in Figure 4. In this design, first, the truth table of the outputs are specified regarding to the most significant bits of the inputs and u(A>B) and u(A=B) outputs of the magnitude comparator. After that, simplified logic statements of the A>B and A=B outputs are determined by K-maps. Finally, the logic statements are implemented by logic gates.

Latency (longest path) = 72 ns.



A[7]	B[7]	u(A>B)	u(A=B)	A>B	A=B
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	X	X
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	X	X
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	X	X
1	1	0	0	1	0
1	1	0	1	0	1
1	1	1	0	0	0
1	1	1	1	X	X

A[7]B[7] \ u(A>B)u(A=B)	00	01	11	10
00	0	0	X	1
01	1	1	X	1
11	1	0	X	0
10	0	0	X	0

A[7]B[7] \ u(A>B)u(A=B)	00	01	11	10
00	0	1	X	0
01	0	0	X	0
11	0	1	X	0
10	0	0	X	0

Figure 10 : K-maps

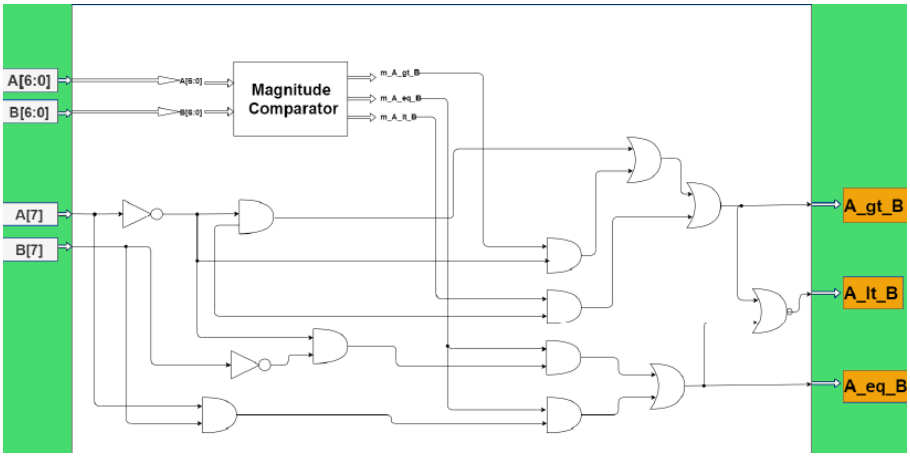


Figure 11 : Signed Comparator design based on magnitude comparator

3. Results

In this section, simulation output of the two comparators and the maximum delay are provided.

3.1. Stage 1

Figures below shows simulation output of stage 1 comparator.

The maximum delay of the comparator is 96 ns.
Therefore maximum clock Frequency that can be applied to the circuit is 10.42 MHz

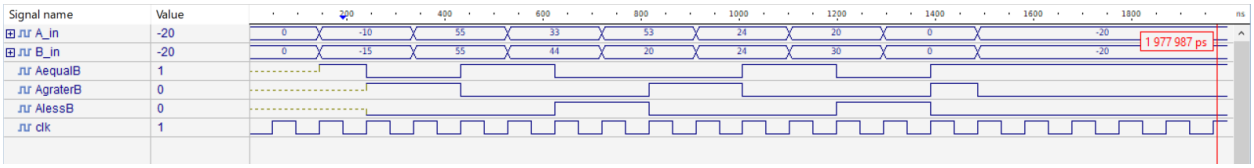


Figure 12: Stage 1 simulation.

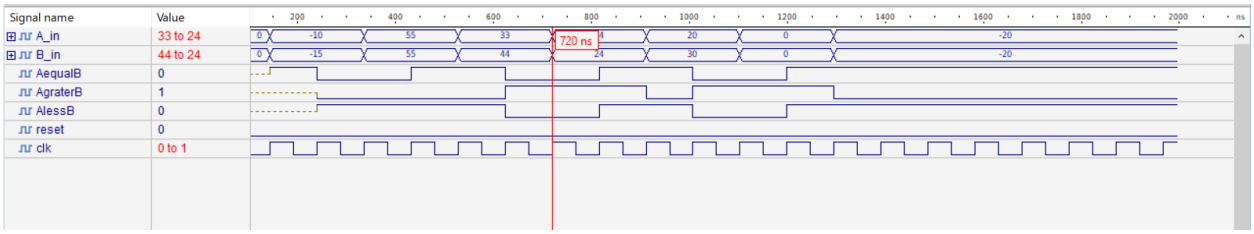


Figure 13: Stage 1 wrong circuit simulation

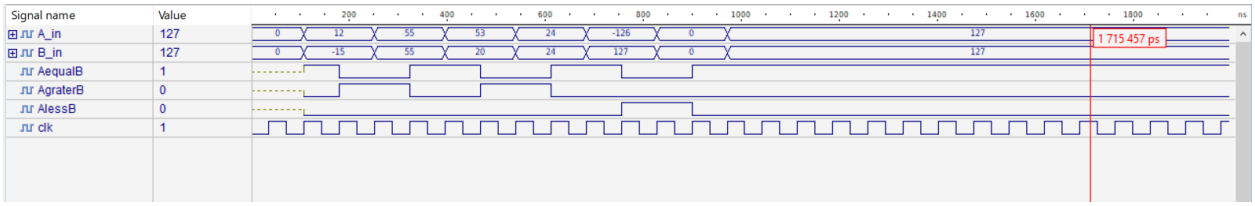
```
# EXECUTION:: ERROR  : error!! not matched
# EXECUTION:: Time: 720 ns,  Iteration: 0,  Instance: /testingStage1,  Process: testStage.
# KERNEL: stopped at time: 1 us
run 1000 ns
# EXECUTION:: ERROR  : error!! not matched
# EXECUTION:: Time: 1104 ns,  Iteration: 0,  Instance: /testingStage1,  Process: testStage.
# KERNEL: stopped at time: 2 us
>
```

Figure 14 : Reporting error for wrong circuit of stage 1.

3.2. Stage 2

Figures below shows the simulation output of stage 2 comparator.

The maximum delay of the comparator is 72 ns.



Therefore maximum clock Frequency that can be applied to the circuit is 13.89 MHz

Figure 15: Stage 2 simulation.

Detecting any error in the circuit using asserts as shown in Figure below:

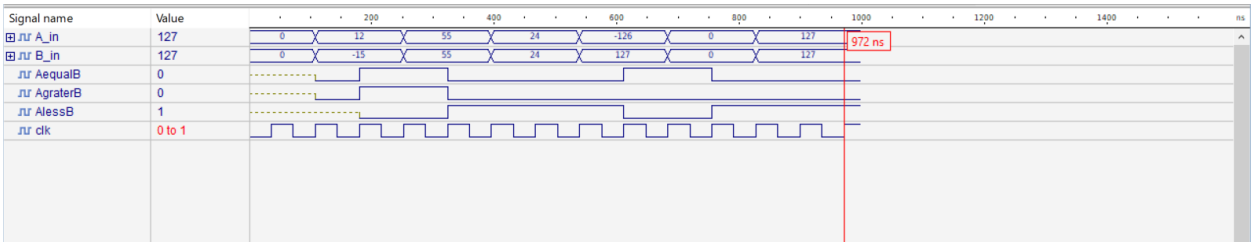


Figure 16 : Stage 2 wrong circuit simulation

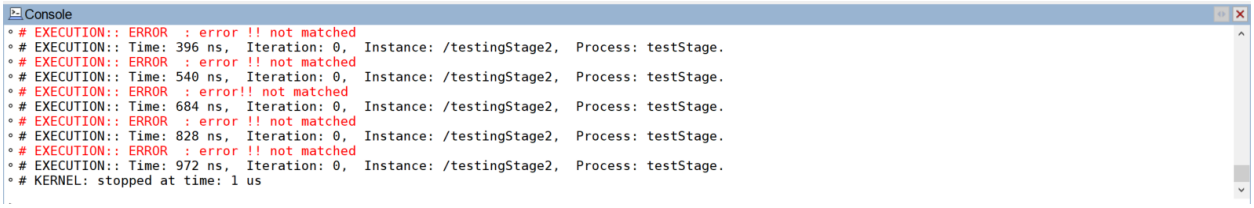


Figure 17 : Reporting error for wrong circuit of stage 2.

4. Conclusion and Future works

In this project, signed number comparator is implemented using two different method, Substractor and magnitude comparator based. The design is verified with a test bench Automatically checking all possible values.
The comparator design have the following potential applications;

- ✓ Microprocessors and microcontrollers have the comparators
- ✓ Threshold problems regarding to sensor values needs a comparator.
- ✓ Cryptocurrency mining hardware are searching a hash results less than a target value.

Therefore, a well-designed comparator is also needed for such an application.

References

[1]. Advanced Digital System Design slides of Dr. Abdallatif Abuissa

[2]. Magnitude Comparator- Birzeit University-Dr. Mohammed Hussein (https://www.youtube.com/watch?v=sTx2dc0VldM&ab_channel=MohammedHussein) [Accessed On 24/12/2021]

[3]. https://allaboutfpga.com/carry-look-ahead-adder-vhdl-code/ [Accessed On 21/12/2021]

[4]. https://www.geeksforgeeks.org/magnitude-comparator-in-digital-logic/ [Accessed On 21/12/2021]

[5]. https://www.electronics-tutorials.ws/combination/comb_8.html [Accessed On 22/12/2021]

Appendix

- **All gates**

```
-- all gates --


-- and gate --
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;


entity and_gate is

        port ( I_0 : in  std_logic;
               I_1 : in  std_logic;
               O   : out std_logic );

end entity;

architecture arch of and_gate is
begin

        O <= I_0 and I_1 after 7ns;

end architecture;

-- and generic --
library IEEE;
```

```
library work;
use work.gate_package.all;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity and_generic is
  generic (
    WIDTH : integer := 4
  );
  port (
    I_0 : in std_logic;
    I_1 : in std_logic;
    I_2 : in std_logic;
    I_3 : in std_logic;
    I_4 : in std_logic;
    I_5 : in std_logic;
    I_6 : in std_logic;
    I_7 : in std_logic;
    O   : out std_logic);

end entity;

architecture arch of and_generic is
  signal wire : std_logic_vector(5 downto 0);
begin
  GEN2 : if WIDTH = 2 generate
    AND0: and_gate port map(I_0 => I_0, I_1 => I_1, O => O);
  end generate;
```

```
GEN3 : if WIDTH = 3 generate
  AND0: and_gate port map(I_0 => I_0, I_1 => I_1, O => wire(0));
  AND1: and_gate port map(I_0 => wire(0), I_1 => I_2, O => O);
end generate;

GEN4 : if WIDTH = 4 generate
  AND0: and_gate port map(I_0 => I_0, I_1 => I_1, O => wire(0));
  AND1: and_gate port map(I_0 => wire(0), I_1 => I_2, O => wire(1));
  AND2: and_gate port map(I_0 => wire(1), I_1 => I_3, O => O);
end generate;

GEN5 : if WIDTH = 5 generate
  AND0: and_gate port map(I_0 => I_0, I_1 => I_1, O => wire(0));
  AND1: and_gate port map(I_0 => wire(0), I_1 => I_2, O => wire(1));
  AND2: and_gate port map(I_0 => wire(1), I_1 => I_3, O => wire(2));
  AND3: and_gate port map(I_0 => wire(2), I_1 => I_4, O => O);
end generate;

GEN6 : if WIDTH = 6 generate
  AND0:and_gate port map(I_0 => I_0, I_1 => I_1, O => wire(0));
  AND1:and_gate port map(I_0 => wire(0), I_1 => I_2, O => wire(1));
  AND2:and_gate port map(I_0 => wire(1), I_1 => I_3, O => wire(2));
  AND3:and_gate port map(I_0 => wire(2), I_1 => I_4, O => wire(3));
  AND4:and_gate port map(I_0 => wire(3), I_1 => I_5, O => O);
end generate;

GEN7 : if WIDTH = 7 generate
  AND0: and_gate port map(I_0 => I_0, I_1 => I_1, O => wire(0));
```

```
   AND1: and_gate port map(I_0 => wire(0), I_1 => I_2, O => wire(1));
   AND2: and_gate port map(I_0 => wire(1), I_1 => I_3, O => wire(2));
   AND3: and_gate port map(I_0 => wire(2), I_1 => I_4, O => wire(3));
   AND4: and_gate port map(I_0 => wire(3), I_1 => I_5, O => wire(4));
   AND5: and_gate port map(I_0 => wire(4), I_1 => I_6, O => O);
  end generate;

  GEN8 : if WIDTH = 8 generate
   AND0:and_gate port map(I_0 => I_0, I_1 => I_1, O => wire(0));
   AND1:and_gate port map(I_0 => wire(0), I_1 => I_2, O => wire(1));
   AND2:and_gate port map(I_0 => wire(1), I_1 => I_3, O => wire(2));
   AND3:and_gate port map(I_0 => wire(2), I_1 => I_4, O => wire(3));
   AND4:and_gate port map(I_0 => wire(3), I_1 => I_5, O => wire(4));
   AND5:and_gate port map(I_0 => wire(4), I_1 => I_6, O => wire(5));
   AND6:and_gate port map(I_0 => wire(5), I_1 => I_7, O => O);
  end generate;
end architecture;

-- nand gate --
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity nand_gate is

        port ( I_0 : in  std_logic;
                I_1 : in  std_logic;
                O   : out std_logic );
```

```
end entity;

architecture arch of nand_gate is
begin

        O <= I_0 nand I_1 after 5ns;

end architecture;
-- nor gate --
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity nor_gate is

        port ( I_0 : in  std_logic;
                I_1 : in  std_logic;
                O   : out std_logic );

end entity;

architecture arch of nor_gate is
begin

        O <= I_0 nor I_1 after 5ns;

end architecture;
```

```
-- or gate --
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity or_gate is

        port ( I_0 : in  std_logic;
               I_1 : in  std_logic;
               O   : out std_logic );

end entity;

architecture arch of or_gate is
begin

        O <= I_0 or I_1 after 7ns;

end architecture;

-- or generic --
library IEEE;
library work;
use work.gate_package.all;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
```

```
entity or_generic is
  generic (
    WIDTH : integer := 4
  );
  port (
    I_0 : in std_logic;
    I_1 : in std_logic;
    I_2 : in std_logic;
    I_3 : in std_logic;
    I_4 : in std_logic;
    I_5 : in std_logic;
    I_6 : in std_logic;
    I_7 : in std_logic;
    O   : out std_logic);

end entity;

architecture arch of or_generic is
  signal wire : std_logic_vector(5 downto 0);
begin
  GEN2 : if WIDTH = 2 generate
    or0: or_gate port map(I_0 => I_0, I_1 => I_1, O => O);
  end generate;

  GEN3 : if WIDTH = 3 generate
    or0: or_gate port map(I_0 => I_0, I_1 => I_1, O => wire(0));
    or1: or_gate port map(I_0 => wire(0), I_1 => I_2, O => O);
  end generate;
```

```
GEN4 : if WIDTH = 4 generate
  or0: or_gate port map(I_0 => I_0, I_1 => I_1, O => wire(0));
  or1: or_gate port map(I_0 => wire(0), I_1 => I_2, O => wire(1));
  or2: or_gate port map(I_0 => wire(1), I_1 => I_3, O => O);
end generate;

GEN5 : if WIDTH = 5 generate
  or0: or_gate port map(I_0 => I_0, I_1 => I_1, O => wire(0));
  or1: or_gate port map(I_0 => wire(0), I_1 => I_2, O => wire(1));
  or2: or_gate port map(I_0 => wire(1), I_1 => I_3, O => wire(2));
  or3: or_gate port map(I_0 => wire(2), I_1 => I_4, O => O);
end generate;

GEN6 : if WIDTH = 6 generate
  or0:or_gate port map(I_0 => I_0, I_1 => I_1, O => wire(0));
  or1:or_gate port map(I_0 => wire(0), I_1 => I_2, O => wire(1));
  or2:or_gate port map(I_0 => wire(1), I_1 => I_3, O => wire(2));
  or3:or_gate port map(I_0 => wire(2), I_1 => I_4, O => wire(3));
  or4:or_gate port map(I_0 => wire(3), I_1 => I_5, O => O);
end generate;

GEN7 : if WIDTH = 7 generate
  or0: or_gate port map(I_0 => I_0, I_1 => I_1, O => wire(0));
  or1: or_gate port map(I_0 => wire(0), I_1 => I_2, O => wire(1));
  or2: or_gate port map(I_0 => wire(1), I_1 => I_3, O => wire(2));
  or3: or_gate port map(I_0 => wire(2), I_1 => I_4, O => wire(3));
  or4: or_gate port map(I_0 => wire(3), I_1 => I_5, O => wire(4));
  or5: or_gate port map(I_0 => wire(4), I_1 => I_6, O => O);
end generate;
```

```
  GEN8 : if WIDTH = 8 generate
    or0:or_gate port map(I_0 => I_0, I_1 => I_1, O => wire(0));
    or1:or_gate port map(I_0 => wire(0), I_1 => I_2, O => wire(1));
    or2:or_gate port map(I_0 => wire(1), I_1 => I_3, O => wire(2));
    or3:or_gate port map(I_0 => wire(2), I_1 => I_4, O => wire(3));
    or4:or_gate port map(I_0 => wire(3), I_1 => I_5, O => wire(4));
    or5:or_gate port map(I_0 => wire(4), I_1 => I_6, O => wire(5));
    or6:or_gate port map(I_0 => wire(5), I_1 => I_7, O => O);
  end generate;
end architecture;

-- inverter --
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity inverter is

        port ( I : in  std_logic;
                    O : out std_logic );

end entity;

architecture arch of inverter is
begin

        O <= not I after 2ns;
```

```
end architecture;
```

- **Gates package**

```
library IEEE;
use IEEE.std_logic_1164.all;
package gate_package is
 component inverter is

   port (
    I : in std_logic;
    O : out std_logic);

 end component;

 component nand_gate is

   port (
    I_0 : in std_logic;
    I_1 : in std_logic;
    O   : out std_logic);

 end component;

 component nor_gate is

   port (
    I_0 : in std_logic;
```

```
    I_1 : in std_logic;
    O   : out std_logic);

end component;

component and_gate is

  port (
    I_0 : in std_logic;
    I_1 : in std_logic;
    O   : out std_logic);

end component;

component or_gate is

  port (
    I_0 : in std_logic;
    I_1 : in std_logic;
    O   : out std_logic);

end component;

component xnor_gate is

  port (
    I_0 : in std_logic;
    I_1 : in std_logic;
    O   : out std_logic);
```

```
end component;

component xor_gate is

  port (
    I_0 : in std_logic;
    I_1 : in std_logic;
    O   : out std_logic);

end component;

component AND_GENERIC is
  generic (
    WIDTH : integer := 4
  );
  port (
                I_0 : in std_logic;
                I_1 : in std_logic;
                I_2 : in std_logic;
                I_3 : in std_logic;
                I_4 : in std_logic;
                I_5 : in std_logic;
                I_6 : in std_logic;
                I_7 : in std_logic;
                O   : out std_logic);
end component;

component OR_GENERIC is
```

```
  generic (
    WIDTH : integer := 4
  );
  port (
                I_0 : in std_logic;
                I_1 : in std_logic;
                I_2 : in std_logic;
                I_3 : in std_logic;
                I_4 : in std_logic;
                I_5 : in std_logic;
                I_6 : in std_logic;
                I_7 : in std_logic;
                O   : out std_logic);
 end component;
end package;
```

- Carry lock ahead adder

```
library IEEE;
library work;
use work.gate_package.all;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
entity carry_look_ahead is
 port (
  A     : in std_logic_vector (7 downto 0);
  B     : in std_logic_vector (7 downto 0);
  C_in  : in std_logic;
  S     : out std_logic_vector (7 downto 0);
  C_out : out std_logic);
```

```
end entity;

architecture arch of carry_look_ahead is

  component partial_full_adder
    port (
      A : in std_logic;
      B : in std_logic;
      C : in std_logic;
      P : out std_logic;
      S : out std_logic;
      G : out std_logic);
  end component;

  signal C1, C2, C3, C4, C5, C6, C7 : std_logic;
  signal P, G                       : std_logic_vector(7 downto 0);
  signal wire                       : std_logic_vector(37 downto 0);
begin
  --Port Mapping

  --for 1's complement
  PFA0 : partial_full_adder port map(A => A(0), B => B(0), C => C_in, P => P(0), S => S(0), G => G(0));
  PFA1 : partial_full_adder port map(A => A(1), B => B(1), C => C1, P => P(1), S => S(1), G => G(1));
  PFA2 : partial_full_adder port map(A => A(2), B => B(2), C => C2, P => P(2), S => S(2), G => G(2));
  PFA3 : partial_full_adder port map(A => A(3), B => B(3), C => C3, P => P(3), S => S(3), G => G(3));
  PFA4 : partial_full_adder port map(A => A(4), B => B(4), C => C4, P => P(4), S => S(4), G => G(4));
  PFA5 : partial_full_adder port map(A => A(5), B => B(5), C => C5, P => P(5), S => S(5), G => G(5));
  PFA6 : partial_full_adder port map(A => A(6), B => B(6), C => C6, P => P(6), S => S(6), G => G(6));
  PFA7 : partial_full_adder port map(A => A(7), B => B(7), C => C7, P => P(7), S => S(7), G => G(7));
```

```
--CLA Circuit Functions
-- C1 <= G(0) OR (P(0) AND C_in);
AND0 : and_gate port map(P(0), C_in, wire(0));
OR0  : or_gate port map(wire(0), G(0), C1);

-- C2 <= G(1) OR (P(1) AND G(0)) OR (P(1) AND P(0) AND C_in);
AND1    : and_gate port map(P(1), G(0), wire(1));
AND_GEN0 : and_generic generic map(3) port map(P(1), P(0), C_in, '0', '0', '0', '0', '0', wire(2));
OR_GEN0  : or_generic generic map(3) port map(wire(1), wire(2), G(1), '0', '0', '0', '0', '0', C2);

-- C3 <= G(2) OR (P(2) AND G(1)) OR (P(2) AND P(1) AND G(0)) OR (P(2) AND P(1) AND P(0) AND C_in);
AND2    : and_gate port map(P(2), G(1), wire(3));
AND_GEN1 : and_generic generic map(3) port map(P(2), P(1), G(0), '0', '0', '0', '0', '0', wire(4));
AND_GEN2 : and_generic generic map(4) port map(P(2), P(1), P(0), C_in, '0', '0', '0', '0', wire(5));
OR_GEN1  : or_generic generic map(4) port map(wire(3), wire(4), wire(5), G(2), '0', '0', '0', '0', C3);
-- C4 <= G(3) OR (P(3) AND G(2)) OR (P(3) AND P(2) AND G(1)) OR (P(3) AND P(2) AND P(1) AND G(0)) OR
(P(3) AND P(2) AND P(1) AND P(0) AND C_in);
AND3    : and_gate port map(P(3), G(2), wire(6));
AND_GEN3 : and_generic generic map(3) port map(P(3), P(2), G(1), '0', '0', '0', '0', '0', wire(7));
AND_GEN4 : and_generic generic map(4) port map(P(3), P(2), P(1), G(0), '0', '0', '0', '0', wire(8));
AND_GEN5 : and_generic generic map(5) port map(P(3), P(2), P(1), P(0), C_in, '0', '0', '0', wire(9));
OR_GEN2  : or_generic generic map(5) port map(wire(6), wire(7), wire(8), wire(9), G(3), '0', '0', '0', C4);
-- C5 <= G(4) OR (P(4) AND G(3)) OR (P(4) AND P(3) AND G(2)) OR (P(4) AND P(3) AND P(2) AND G(1)) OR
(P(4) AND P(3) AND P(2) AND P(1) AND G(0)) OR (P(4) AND P(3) AND P(2) AND P(1) AND P(0) AND C_in);
AND4    : and_gate port map(P(4), G(3), wire(10));
AND_GEN6 : and_generic generic map(3) port map(P(4), P(3), G(2), '0', '0', '0', '0', '0', wire(11));
AND_GEN7 : and_generic generic map(4) port map(P(4), P(3), P(2), G(1), '0', '0', '0', '0', wire(12));
AND_GEN8 : and_generic generic map(5) port map(P(4), P(3), P(2), P(1), G(0), '0', '0', '0', wire(13));
```

```
  AND_GEN9 : and_generic generic map(6) port map(P(4), P(3), P(2), P(1), P(0), C_in, '0', '0', wire(14));

  OR_GEN3  : or_generic generic map(6) port map(wire(10), wire(11), wire(12), wire(13), wire(14), G(4),
'0', '0', C5);

  -- C6 <= G(5) OR (P(5) AND G(4)) OR (P(5) AND P(4) AND G(3)) OR (P(5) AND P(4) AND P(3) AND G(2)) OR
(P(5) AND P(4) AND P(3) AND P(2) AND G(1)) OR (P(5) AND P(4) AND P(3) AND G(2)) OR (P(5) AND P(4)
AND P(3) AND P(2) AND G(1)) OR (P(5) AND P(4) AND P(3) AND P(2) AND P(1) AND G(0)) OR (P(5) AND
P(4) AND P(3) AND P(2) AND P(1) AND P(0) AND C_in);

  AND5     : and_gate port map(P(5), G(4), wire(15));

  AND_GEN10 : and_generic generic map(3) port map(P(5), P(4), G(3), '0', '0', '0', '0', '0', wire(16));

  AND_GEN11 : and_generic generic map(4) port map(P(5), P(4), P(3), G(2), '0', '0', '0', '0', wire(17));

  AND_GEN12 : and_generic generic map(5) port map(P(5), P(4), P(3), P(2), G(1), '0', '0', '0', wire(18));

  AND_GEN13 : and_generic generic map(6) port map(P(5), P(4), P(3), P(2), P(1), G(0), '0', '0', wire(19));

  AND_GEN14 : and_generic generic map(7) port map(P(5), P(4), P(3), P(2), P(1), P(0), C_in, '0', wire(20));

  OR_GEN4  : or_generic generic map(7) port map(wire(15), wire(16), wire(17), wire(18), wire(19),
wire(20), G(5), '0', C6);

  -- C7 <= G(6) OR (P(6) AND G(5)) OR (P(6) AND P(5) AND G(4)) OR (P(6) AND P(5) AND P(4) AND G(3)) OR
(P(6) AND P(5) AND P(4) AND P(3) AND G(2)) OR (P(6) AND P(5) AND P(4) AND G(3)) OR (P(6) AND P(5)
AND P(4) AND P(3) AND G(2)) OR (P(6) AND P(5) AND P(4) AND P(3) AND P(2) AND G(1)) OR (P(6) AND
P(5) AND P(4) AND P(3) AND P(2) AND P(1) AND G(0)) OR (P(6) AND P(5) AND P(4) AND P(3) AND P(2)
AND P(1) AND P(0) AND C_in) ;

  AND6     : and_gate port map(P(6), G(5), wire(21));

  AND_GEN15 : and_generic generic map(3) port map(P(6), P(5), G(4), '0', '0', '0', '0', '0', wire(22));

  AND_GEN16 : and_generic generic map(4) port map(P(6), P(5), P(4), G(3), '0', '0', '0', '0', wire(23));

  AND_GEN17 : and_generic generic map(5) port map(P(6), P(5), P(4), P(3), G(2), '0', '0', '0', wire(24));

  AND_GEN18 : and_generic generic map(6) port map(P(6), P(5), P(4), P(3), P(2), G(1), '0', '0', wire(25));

  AND_GEN19 : and_generic generic map(7) port map(P(6), P(5), P(4), P(3), P(2), P(1), G(0), '0', wire(26));

  AND_GEN20 : and_generic generic map(8) port map(P(6), P(5), P(4), P(3), P(2), P(1), P(0), C_in,
wire(27));

  OR_GEN5  : or_generic generic map(8) port map(wire(21), wire(22), wire(23), wire(24), wire(25),
wire(26), wire(27), G(6), C7);

  --

  AND7     : and_gate port map(P(7), G(6), wire(28));

  AND_GEN21 : and_generic generic map(3) port map(P(7), P(6), G(5), '0', '0', '0', '0', '0', wire(29));
```

```
  AND_GEN22 : and_generic generic map(4) port map(P(7), P(6), P(5), G(4), '0', '0', '0', '0', wire(30));

  AND_GEN23 : and_generic generic map(5) port map(P(7), P(6), P(5), P(4), G(3), '0', '0', '0', wire(31));

  AND_GEN24 : and_generic generic map(6) port map(P(7), P(6), P(5), P(4), P(3), G(2), '0', '0', wire(32));

  AND_GEN25 : and_generic generic map(7) port map(P(7), P(6), P(5), P(4), P(3), P(2), G(1), '0', wire(33));

  AND_GEN26 : and_generic generic map(8) port map(P(7), P(6), P(5), P(4), P(3), P(2), P(1), G(0),
wire(34));

  AND_GEN27 : and_generic generic map(8) port map(P(7), P(6), P(5), P(4), P(3), P(2), P(1), P(0),
wire(35));

  AND8      : and_gate port map(wire(35), C_in, wire(36));

  OR_GEN6   : or_generic generic map(8) port map(wire(28), wire(29), wire(30), wire(31), wire(32),
wire(33), wire(34), wire(36), wire(37));

  OR_0      : or_gate port map(wire(37), G(7), C_out);


end architecture;
```

- 2's complement

```
library IEEE;

library work;

use work.gate_package.all;

use IEEE.std_logic_1164.all;

use IEEE.numeric_std.all;

use IEEE.std_logic_unsigned.all;


entity twos_complement is

  port (

  A     : in std_logic_vector (7 downto 0);

  neg_A : out std_logic_vector(7 downto 0));

end entity;


architecture arch of twos_complement is
```

```
 signal not_A : std_logic_vector(7 downto 0);

begin

 GEN_NOT : for jj in 0 to 7 generate
  not_gen : inverter port map(A(jj), not_A(jj));
 end generate GEN_NOT;
 inc_1 : entity work.carry_look_ahead
 port map(
  A    => not_A,
  B    => "00000001",
  C_in  => '0',
  S    => neg_A,
  C_out => open
 );

end architecture;
```

- **4bit Magnitude comparator**

```
library IEEE;
library work;
use work.gate_package.all;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
entity magnitude_comparator_4bit is

 port (
  A : in std_logic_vector(3 downto 0);
```

```
    B : in std_logic_vector(3 downto 0);

    i_a_gt_b : in std_logic;
    i_a_lt_b : in std_logic;
    i_a_eq_b : in std_logic;
    a_gt_B   : out std_logic;
    a_eq_B   : out std_logic;
    a_lt_B   : out std_logic);
end entity;

architecture arch of magnitude_comparator_4bit is

  signal s, not_B, not_A        : std_logic_vector(3 downto 0);
  signal o_and0, o_and1, a7, b7  : std_logic;
  signal gate_O_and, gate_O_and1 : std_logic_vector(8 downto 0);
  signal gate_O_or, gate_O_or1   : std_logic_vector(1 downto 0);

  signal m_A_gt_B : std_logic;
  signal m_A_lt_B : std_logic;
  signal m_A_eq_B : std_logic;

  signal wire : std_logic_vector(1 downto 0);
begin
```

```
-- A=B --

XNOR0 : xnor_gate port map(I_0 => A(0), I_1 => B(0), O => s(0));
XNOR1 : xnor_gate port map(I_0 => A(1), I_1 => B(1), O => s(1));
XNOR2 : xnor_gate port map(I_0 => A(2), I_1 => B(2), O => s(2));
XNOR3 : xnor_gate port map(I_0 => A(3), I_1 => B(3), O => s(3));


AND0 : and_gate port map(I_0 => s(3), I_1 => s(2), O => o_and0);
AND1 : and_gate port map(I_0 => o_and0, I_1 => s(1), O => o_and1);
AND2 : and_gate port map(I_0 => o_and1, I_1 => s(0), O => m_A_eq_B);


-- A>B --

INV0 : inverter port map(I => B(0), O => not_B(0));
INV1 : inverter port map(I => B(1), O => not_B(1));
INV2 : inverter port map(I => B(2), O => not_B(2));
INV3 : inverter port map(I => B(3), O => not_B(3));

AND3  : and_gate port map(I_0 => A(3), I_1 => not_B(3), O => gate_O_and(0));
AND4  : and_gate port map(I_0 => s(3), I_1 => A(2), O => gate_O_and(1));
AND5  : and_gate port map(I_0 => gate_O_and(1), I_1 => not_B(2), O => gate_O_and(2));
AND6  : and_gate port map(I_0 => gate_O_and(8), I_1 => A(1), O => gate_O_and(3));
AND7  : and_gate port map(I_0 => gate_O_and(3), I_1 => not_B(1), O => gate_O_and(4));
AND8  : and_gate port map(I_0 => gate_O_and(8), I_1 => s(1), O => gate_O_and(5));
AND9  : and_gate port map(I_0 => gate_O_and(5), I_1 => A(0), O => gate_O_and(6));
AND10 : and_gate port map(I_0 => gate_O_and(6), I_1 => not_B(0), O => gate_O_and(7));
```

```
AND22 : and_gate port map(I_0 => s(3), I_1 => s(2), O => gate_O_and(8));

OR0 : or_gate port map(I_0 => gate_O_and(0), I_1 => gate_O_and(2), O => gate_O_or(0));
OR1 : or_gate port map(I_0 => gate_O_and(4), I_1 => gate_O_and(7), O => gate_O_or(1));
OR2 : or_gate port map(I_0 => gate_O_or(0), I_1 => gate_O_or(1), O => m_A_gt_B);
-- A<B --


INV4 : inverter port map(I => A(0), O => not_A(0));
INV5 : inverter port map(I => A(1), O => not_A(1));
INV6 : inverter port map(I => A(2), O => not_A(2));
INV7 : inverter port map(I => A(3), O => not_A(3));

AND11 : and_gate port map(I_0 => B(3), I_1 => not_A(3), O => gate_O_and1(0));
AND12 : and_gate port map(I_0 => s(3), I_1 => B(2), O => gate_O_and1(1));
AND13 : and_gate port map(I_0 => gate_O_and1(1), I_1 => not_A(2), O => gate_O_and1(2));

AND14 : and_gate port map(I_0 => gate_O_and1(8), I_1 => B(1), O => gate_O_and1(3));
AND15 : and_gate port map(I_0 => gate_O_and1(3), I_1 => not_A(1), O => gate_O_and1(4));

AND16 : and_gate port map(I_0 => gate_O_and1(8), I_1 => s(1), O => gate_O_and1(5));
AND17 : and_gate port map(I_0 => gate_O_and1(5), I_1 => B(0), O => gate_O_and1(6));
AND18 : and_gate port map(I_0 => gate_O_and1(6), I_1 => not_A(0), O => gate_O_and1(7));

AND23 : and_gate port map(I_0 => s(3), I_1 => s(2), O => gate_O_and1(8));

OR3 : or_gate port map(I_0 => gate_O_and1(0), I_1 => gate_O_and1(2), O => gate_O_or1(0));
OR4 : or_gate port map(I_0 => gate_O_and1(4), I_1 => gate_O_and1(7), O => gate_O_or1(1));
OR5 : or_gate port map(I_0 => gate_O_or1(0), I_1 => gate_O_or1(1), O => m_A_lt_B);
```

```
--a_gt_b <= (m_a_eq_b and i_a_gt_b) or m_A_gt_B;
AND19 : and_gate port map(I_0 => m_a_eq_b, I_1 => i_a_gt_b, O => wire(0));
OR6   : or_gate port map(I_0 => wire(0), I_1 => m_A_gt_B, O => a_gt_B);

--a_lt_b <= (m_a_eq_b and i_a_lt_b) or m_A_lt_B;
AND20 : and_gate port map(I_0 => m_a_eq_b, I_1 => i_a_lt_b, O => wire(1));
OR7   : or_gate port map(I_0 => wire(1), I_1 => m_A_lt_B, O => a_lt_B);

--a_eq_b <= m_a_eq_b and i_A_eq_B;
AND21 : and_gate port map(I_0 => m_a_eq_b, I_1 => i_a_eq_b, O => a_eq_b);

end architecture;
```

- # Partial full adder

```
library IEEE;
library work;
use work.gate_package.all;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity partial_full_adder is
  port (
    A : in std_logic;
    B : in std_logic;
    C : in std_logic;
    P : out std_logic;
    S : out std_logic;
    G : out std_logic
  );

end entity;

architecture arch of partial_full_adder is
  --Signals
  signal A_in, B_in, xor1_out, C_xor2, xor2_out, and1_out : std_logic;
```

```
begin
  --Port Mapping
  xor1 : xor_gate port map(I_0 => A, I_1 => B, O => xor1_out);
  xor2 : xor_gate port map(I_0 => xor1_out, I_1 => C, O => S);
  and1 : and_gate port map(I_0 => A, I_1 => B, O => G);
  or1  : or_gate port map(I_0 => A, I_1 => B, O => P);


end architecture;
```

- Stage 1

```
library IEEE;
library work;
use work.gate_package.all;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
entity stage_1_comp is
  port (
    A      : in std_logic_vector (7 downto 0);
    B      : in std_logic_vector (7 downto 0);
    a_gt_b : out std_logic;
    a_lt_b : out std_logic;
    a_eq_b : out std_logic);
end entity;

architecture arch of stage_1_comp is

  component carry_look_ahead
    port (
      A     : in std_logic_vector (7 downto 0);
      B     : in std_logic_vector (7 downto 0);
      C_in  : in std_logic;
      S     : out std_logic_vector (7 downto 0);
      C_out : out std_logic
    );
  end component;

  signal not_b   : std_logic_vector(7 downto 0);
```

```
  signal not_a   : std_logic;
  signal sub_out : std_logic_vector(8 downto 0);
  signal P, G    : std_logic_vector(7 downto 0);
  signal wire    : std_logic_vector(37 downto 0);
begin

  GEN_NOT : for jj in 0 to 7 generate
    not_gen : inverter port map(B(jj), not_b(jj));
  end generate GEN_NOT;
  substractor : carry_look_ahead
  port map(
    A     => A,
    B     => not_b,
    C_in  => '1',
    S     => sub_out(7 downto 0),
    C_out => sub_out(8)
  );

  not_0 : inverter port map(A(7), not_a);
  and_0 : and_gate port map(not_a, B(7), wire(0));

  -- a_eq_b = nor(sub0,sub1,sub2,...sub7)
  or_gen0 : or_generic generic map(8) port map(sub_out(0), sub_out(1), sub_out(2),
sub_out(3), sub_out(4), sub_out(5), sub_out(6), sub_out(7), wire(1));
  not_1   : inverter port map(wire(1), wire(2));

  a_eq_b <= wire(2);
  -- handling the overflow issues by the following line
  -- a_lt_b = (((a(7) and b(7)) or (not a(7) and not b(7))) and sub_out(7))  or (a(7) and b(7)')
  and_1 : and_gate port map(A(7), B(7), wire(3)); -- a(7) and b(7)
  and_2 : and_gate port map(not_a, not_b(7), wire(4)); --a(7)' and b(7)'
  or_0  : or_gate port map(wire(3), wire(4), wire(5)); --((a(7) and b(7)) or (not a(7) and not
b(7)))
  and_3 : and_gate port map(wire(5), sub_out(7), wire(6)); --(a(7) and b(7)) or (not a(7) and
not b(7))) and sub_out(7))
  and_4 : and_gate port map(a(7), not_b(7), wire(7)); --(a(7) and b(7)')
  or_1  : or_gate port map(wire(6), wire(7), wire(8)); -- a_lt_b

  a_lt_b <= wire(8);
```

```
  -- a_gt_b = a_lt_b nor a_eq_b
  nor_0 : nor_gate port map(wire(2), wire(8), a_gt_b);
end architecture;
```

- Stage 2

```
library IEEE;
library work;
use work.gate_package.all;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
entity stage_2_comp is
  port (
    A       : in std_logic_vector (7 downto 0);
    B       : in std_logic_vector (7 downto 0);
    A_gt_B_o : out std_logic;
    A_lt_B_o : out std_logic;
    A_eq_B_o : out std_logic);
end entity;

architecture arch of stage_2_comp is

  component magnitude_comparator_4bit
    port (
      A       : in std_logic_vector(3 downto 0);
      B       : in std_logic_vector(3 downto 0);
      i_a_gt_b : in std_logic;
      i_a_lt_b : in std_logic;
      i_a_eq_b : in std_logic;
      a_gt_B   : out std_logic;
      a_eq_B   : out std_logic;
      a_lt_B   : out std_logic
    );
  end component;

  component twos_complement
    port (
    A : in std_logic_vector (7 downto 0);
    neg_A : out std_logic_vector(7 downto 0)
  );
end component;
```

```
  signal m_A_gt_B : std_logic;
  signal m_A_lt_B : std_logic;
  signal m_A_eq_B : std_logic;

  signal A_gt_B : std_logic;
  signal A_lt_B : std_logic;
  signal A_eq_B : std_logic;

  signal A_high   : std_logic_vector(3 downto 0);
  signal B_high   : std_logic_vector(3 downto 0);

  signal A_gt_B_o_wire : std_logic;
  signal A_eq_B_o_wire : std_logic;
  signal not_A_eq_B : std_logic;
  signal not_A_gt_B : std_logic;
  signal not_a        : std_logic;
  signal not_b        : std_logic;
  signal wire         : std_logic_vector(5 downto 0);


  signal neg_A, neg_B : std_logic_vector(7 downto 0);

  signal comparator_in_A : std_logic_vector(7 downto 0);
  signal comparator_in_B : std_logic_vector(7 downto 0);

begin



  magnitude_comparator_4bit_low : magnitude_comparator_4bit
  port map(
    A       => comparator_in_A(3 downto 0),
    B       => comparator_in_B(3 downto 0),
    i_a_gt_b => '0',
    i_a_lt_b => '0',
    i_a_eq_b => '1',
    a_gt_B   => m_A_gt_B,
    a_eq_B   => m_A_eq_B,
```

```
  a_lt_B   => m_A_lt_B
 );

 A_high <=  comparator_in_A(7 downto 4);
 B_high <=  comparator_in_B(7 downto 4);
 magnitude_comparator_4bit_high : magnitude_comparator_4bit
 port map(
  A       => A_high,
  B       => B_high,
  i_a_gt_b => m_A_gt_B,
  i_a_lt_b => m_A_lt_B,
  i_a_eq_b => m_A_eq_B,
  a_gt_B   => A_gt_B,
  a_eq_B   => A_eq_B,
  a_lt_B   => A_lt_B
 );

 twos_complement_inst0 : twos_complement
 port map (
  A => A,
  neg_A => neg_A
 );

 twos_complement_inst1 : twos_complement
 port map (
  A => B,
  neg_A => neg_B
 );

 comparator_in_A <= ((A(7) & A(7) & A(7) & A(7) & A(7) & A(7) & A(7) & A(7)) and
neg_A) or ((not_A & not_A & not_A & not_A & not_A & not_A & not_A & not_A) and
A);
 comparator_in_B <= ((B(7) & B(7) & B(7) & B(7) & B(7) & B(7) & B(7) & B(7)) and
neg_B) or ((not_B & not_B & not_B & not_B & not_B & not_B & not_B & not_B) and B);

 -- A > B

 INV0    : inverter port map(I => A(7), O => not_A);
 INV1    : inverter port map(I => A_eq_B, O => not_A_eq_B);
 INV2    : inverter port map(I => A_gt_B, O => not_A_gt_B);
```

```
  AND0     : and_gate port map(I_0 => not_A, I_1 => B(7), O => wire(0));
  AND1     : and_gate port map(I_0 => not_A, I_1 => A_gt_B, O => wire(1));
  AND_GEN0 : and_generic generic map(3) port map(B(7), not_A_gt_B, not_A_eq_B, '0',
'0', '0', '0', '0', wire(2));
  OR_GEN0  : or_generic generic map(3) port map(wire(0), wire(1), wire(2), '0', '0', '0', '0',
'0', A_gt_B_o_wire);

  A_gt_B_o <= A_gt_B_o_wire;

  -- A = B
  INV3 : inverter port map(I => B(7), O => not_B);
  AND2 : and_gate port map(I_0 => not_A, I_1 => not_B, O => wire(3));
  AND3 : and_gate port map(I_0 => A(7), I_1 => B(7), O => wire(4));
  OR0  : or_gate port map(I_0 => wire(3), I_1 => wire(4), O => wire(5));
  AND4 : and_gate port map(I_0 => wire(5), I_1 => A_eq_B, O => A_eq_B_o_wire);

  A_eq_B_o <= A_eq_B_o_wire;

  -- A < B

  NOR0 : nor_gate port map(I_0 => A_gt_B_o_wire, I_1 => A_eq_B_o_wire, O =>
A_lt_B_o);

end architecture;
```

- System 1 and 2

```
LIBRARY IEEE;
LIBRARY work;
USE work.gate_package.ALL;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE IEEE.std_logic_unsigned.ALL;
ENTITY SystemOfStage1 IS
   PORT (A,B : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
                  AequalB, AgraterB, AlessB : OUT STD_LOGIC;
                  reset,clk: IN STD_LOGIC);
END SystemOfStage1;

ARCHITECTURE arch OF SystemOfStage1 IS
SIGNAL reg_A, reg_B: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL enable_Of_A, enable_Of_B, enable_Of_OUT: STD_LOGIC;
```

```
SIGNAL Reg_AequalB, Reg_AgraterB, Reg_AlessB: STD_LOGIC;

--make component of 8 bit Register
COMPONENT Register8bit IS
PORT(
   clock,enable,reset       : IN std_logic;
   d        : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
   q        : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;

--make component of FlipFlop
COMPONENT FlipFlop IS
PORT(
   clock,enable,reset,d: IN std_logic;
   q        : OUT std_logic);
END COMPONENT;

--make component of stage 1 comparator
COMPONENT stage_1_comp IS
PORT(
   A,B : IN std_logic_vector (7 DOWNTO 0);
   a_gt_b,a_lt_b,a_eq_b : OUT std_logic);

END COMPONENT;
BEGIN

Reg1_In:Register8bit  PORT MAP( clk,enable_Of_A,reset,A,reg_A);
Reg2_In: Register8bit PORT MAP( clk,enable_Of_B,reset,B,reg_B);
Comp: stage_1_comp PORT MAP( reg_A,reg_B, Reg_AgraterB,Reg_AlessB,
Reg_AequalB);
AequalB_F: FlipFlop PORT MAP( clk,enable_Of_OUT,reset,Reg_AequalB,AequalB);
AgraterB_F: FlipFlop PORT MAP( clk,enable_Of_OUT,reset,Reg_AgraterB,AgraterB);

AlessB_F: FlipFlop PORT MAP(clk,enable_Of_OUT,reset,Reg_AlessB,AlessB);


enable_Of_A<= '1';
enable_Of_B<= '1';
enable_Of_OUT<= '1';
```

```
END arch;
```

```
LIBRARY IEEE;
LIBRARY work;
USE work.gate_package.ALL;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE IEEE.std_logic_unsigned.ALL;
ENTITY SystemOfStage2 IS
   PORT (A,B : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
                  AequalB, AgraterB, AlessB : OUT STD_LOGIC;
                  reset,clk: IN STD_LOGIC);
END SystemOfStage2;

ARCHITECTURE arch OF SystemOfStage2 IS
SIGNAL reg_A, reg_B: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL enable_Of_A, enable_Of_B, enable_Of_OUT: STD_LOGIC;
SIGNAL Reg_AequalB, Reg_AgraterB, Reg_AlessB: STD_LOGIC;

--make component of 8 bit Register
COMPONENT Register8bit IS
PORT(
   clock,enable,reset      : IN std_logic;
   d        : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
   q        : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;

--make component of FlipFlop
COMPONENT FlipFlop IS
PORT(
   clock,enable,reset,d: IN std_logic;
   q        : OUT std_logic);
END COMPONENT;

--make component of stage 2 comparator
COMPONENT stage_2_comp IS
PORT(
   A,B : IN std_logic_vector (7 DOWNTO 0);
   a_gt_b_o,a_lt_b_o,a_eq_b_o : OUT std_logic);

END COMPONENT;
```

```
BEGIN

Reg1_In:Register8bit  PORT MAP( clk,enable_Of_A,reset,A,reg_A);
Reg2_In: Register8bit PORT MAP( clk,enable_Of_B,reset,B,reg_B);
Comp: stage_2_comp PORT MAP( reg_A,reg_B, Reg_AgraterB,Reg_AlessB,
Reg_AequalB);
AequalB_F: FlipFlop PORT MAP( clk,enable_Of_OUT,reset,Reg_AequalB,AequalB);
AgraterB_F: FlipFlop PORT MAP( clk,enable_Of_OUT,reset,Reg_AgraterB,AgraterB);

AlessB_F: FlipFlop PORT MAP(clk,enable_Of_OUT,reset,Reg_AlessB,AlessB);


enable_Of_A<= '1';
enable_Of_B<= '1';
enable_Of_OUT<= '1';

END arch;
```

- Testing

```
library IEEE;
library work;
use work.gate_package.all;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;


ENTITY testingStage1 IS
END testingStage1;

ARCHITECTURE arch OF testingStage1 IS

component  SystemOfStage1 is

   Port ( A,B : in  STD_LOGIC_VECTOR(7 downto 0);
               AequalB, AgraterB, AlessB : out STD_LOGIC;
               reset,clk: in STD_LOGIC
         );
end component;

SIGNAL A_in,B_in : STD_LOGIC_VECTOR (7 DOWNTO 0);
```

```
SIGNAL AequalB, AgraterB, AlessB : STD_LOGIC;
SIGNAL reset: STD_LOGIC;
SIGNAL clk:STD_LOGIC;

BEGIN
   UUT: SystemOfStage1
   PORT MAP(A_in,B_in,AequalB,AgraterB,AlessB,reset,clk);
   clock_process :process
            begin
            clk <= '0';
            wait for 48 ns;
            clk <= '1';
            wait for 48 ns;
   end process;
   Testing: process
   begin
   A_in<= ("00000000");
   B_in<= ("00000000");
   reset<= '1';

   wait for 48 ns;
  reset<= '0';

   wait for 96 ns;
   A_in<= "11110110";  -- -10
   B_in<= "11110001";  -- -15
   wait for 192 ns;
   A_in<= "00110111"; --55
   B_in<= "00110111"; --55
   wait for 192 ns;
   A_in<= "00100001"; -- 33
   B_in<= "00101100";  -- 44
   wait for 192 ns;
   A_in<= "00011000";  -- 24
   B_in<= "00011000";  -- 24
   wait for 192 ns;
   A_in<= "00010100"; -- 20
   B_in<= "00011110"; -- 30
   wait for 192 ns;
   A_in<= "00000000"; -- 0
```

```
    B_in<= "00000000";-- 0
    wait for 192 ns;
    A_in<= "11101100"; -- -20
    B_in<= "11101100";-- -20
            wait;
    end process;
---------------------------------
-- verifier


testStage: process

begin
    wait  on  A_in;
    wait  for   192 ns;
    if  (signed(A_in) =  signed(B_in) and AequalB /= '1') then
            assert false
            report "error !! not matched "
        severity error;
    elsif(signed(A_in) > signed(B_in)and AgraterB /= '1') then
            assert false
            report "error !! not matched "
        severity error;

    elsif(signed(A_in) < signed(B_in) and AlessB /= '1') then
            assert false
            report "error!! not matched "
             severity error;

    end if;


end  process;

END arch;
```

```
library IEEE;
library work;
use work.gate_package.all;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
```

```
ENTITY testingStage2 IS
END testingStage2;

ARCHITECTURE arch OF testingStage2 IS
component  SystemOfStage2 is

   Port ( A,B : in  STD_LOGIC_VECTOR(7 downto 0);
               AequalB, AgraterB, AlessB : out STD_LOGIC;
               reset,clk: in STD_LOGIC
         );
end component;

SIGNAL A_in,B_in : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL AequalB, AgraterB, AlessB : STD_LOGIC;
SIGNAL reset: STD_LOGIC;
SIGNAL clk:STD_LOGIC;

BEGIN
   UUT: SystemOfStage2
   PORT MAP(A_in,B_in,AequalB,AgraterB,AlessB,reset,clk);
   clock_process :process
            begin
            clk <= '0';
            wait for 36 ns;
            clk <= '1';
            wait for 36 ns;
   end process;

   Testing: process
   begin
   A_in<= ("00000000");
   B_in<= ("00000000");
   reset<= '1';

   wait for 36 ns;
   reset<= '0';

   wait for 72 ns;
```

```
    A_in<= "00001100";  -- 12
    B_in<= "11110001";  -- -15
    wait for 144 ns;
    A_in<= "00110111"; --55
    B_in<= "00110111"; --55
    wait for 144 ns;
    A_in<= "00011000";  -- 24
    B_in<= "00011000";  -- 24
    wait for 144 ns;
    A_in<= "10000010"; -- -126
    B_in<= "01111111"; -- 127
    wait for 144 ns;
    A_in<= "00000000"; -- 0
    B_in<= "00000000";-- 0
    wait for 144 ns;
    A_in<= "01111111"; -- 127
    B_in<= "01111111";-- 127
            wait;
    end process;
---------------------------------
-- verifier



testStage: process

begin
    wait  on  A_in;
    wait  for   144 ns;
    if  (signed(A_in) =  signed(B_in) and AequalB /= '1') then
            assert false
            report "error !! not matched "
        severity error;
    elsif(signed(A_in) > signed(B_in)and AgraterB /= '1') then
            assert false
            report "error !! not matched "
        severity error;

    elsif(signed(A_in) < signed(B_in) and AlessB /= '1') then
            assert false
```

```
            report "error!! not matched "
             severity error;

    end if;


end  process;

END arch;
```

- Register and filp flop

```
library IEEE;
library work;
use work.gate_package.all;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
entity Register8bit is
port(
    clock   : in std_logic;
    enable  : in std_logic;
    reset : in std_logic;
    d       : in STD_LOGIC_VECTOR(7 downto 0);
    q       : out STD_LOGIC_VECTOR(7 downto 0));
end Register8bit;

architecture simple of Register8bit is
begin

Register8:          process(clock)
            begin

                    if rising_edge(clock) then
                            if (reset = '1' ) then
                                    q<= (OTHERS=>'0');
                            elsif (enable ='1') then
                                    q <= d;
                            end if;
                    end if;

            end process;
```

```
end simple;
```

```
library IEEE;
library work;
use work.gate_package.all;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
entity FlipFlop is
port(
    clock   : in std_logic;
    enable  : in std_logic;
    reset : in std_logic;
    d       : in std_logic;
    q       : out std_logic);
end FlipFlop;

architecture behavioral of FlipFlop is
begin

FlipFlop:  process(clock)
           begin

                  if rising_edge(clock) then
                        if (reset = '1' ) then
                              q<= '0';
                        elsif (enable ='1') then
                              q <= d;
                        end if;
                  end if;

           end process;

end behavioral;
```