Faculty of Engineering & Technology Electrical & Computer Engineering Department

ADVANCED DIGITAL SYSTEMS DESIGN ENCS3310 Course Project

Prepared by:

Hasan Hamed 1190496

Instructor: Dr. Abdallatif Abuissa

Section: 1

Date: 24/12/2021

1. Introduction & Background

The main goal of the project is to design an 8-bit signed comparator VHDL code which compares two numbers represented in two's complement representation structurally in two stages of complexity, first by using the ripple adder and negative, overflow and zero flags, and second by using the magnitude comparator and the sign bit of the two numbers, and finally, to test both circuits using effective test benches that reports if there is an error in the designs.

Table of Contents

Table of Figures

2. Design philosophy

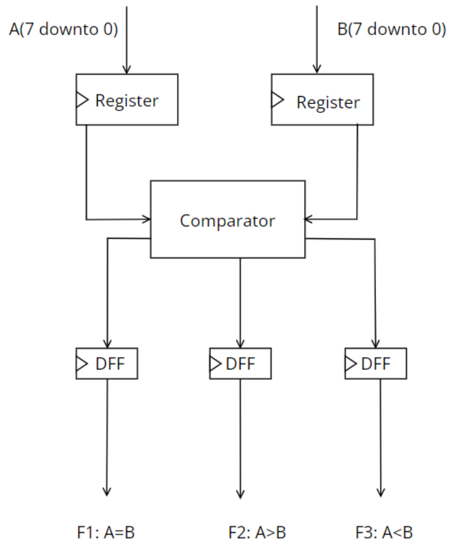


Figure 2.1: Block diagram of the 8-bit comparator.

As shown above in Figure 2.1, the comparator has two 8-bit inputs that are connected to the comparator from registers, and the outputs of the comparator is connected to output registers that give the output of the whole circuit.

2.1. Stage 1

In this stage, the signed comparator concept is by subtracting both numbers and determining the overflow, zero and negative flags as shown in Figure 2.2.

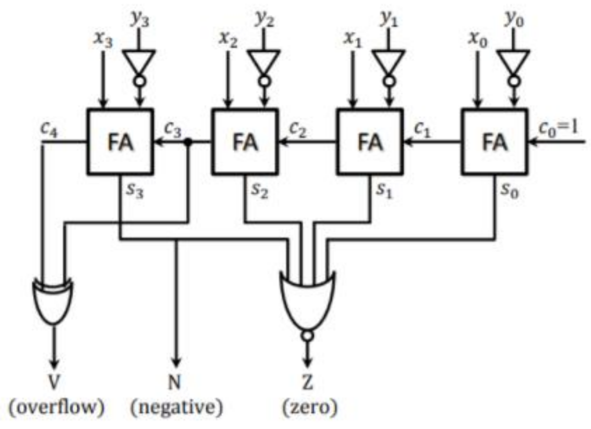


Figure 2.2: 4-bit Signed comparator with flags.

The 8-bit comparator will be designed by cascading this 4-bit comparator with another 4-bit comparator and then, the flags will be given by:

V= C (7) XOR C (6)

N= S7

Z= NOR (S0 to S7)

Using the three flags the three outputs can be easily determined by using those important rules about comparing in computer organization:

A=B when Z=1

A>B when Z=0 and N=V

A<B when N! =V

Using that information, the outputs are given by:

F1=Z

F2= (N XNOR V) AND Z'

F3= N XNOR V

Latency = longest path = 2 inverter+ (14*8) FA + 12 XOR + 9 XNOR + 2 inverters+7 AND = 144 ns.

2.2. Stage 2

In this stage, the signed comparator was designed by comparing the magnitude of the 7 least significant bits of both numbers using magnitude comparator, and then using the sign bit and the output of the magnitude comparator to determine the output of the signed comparator.

F1 is generated by the equality circuit which is:

F1= AND e (0 to 7)

And e is given by:

e (0 to 7) = A (0 to 7) XNOR B (0 to 7)

Which means that both numbers are equal if every bit of each number is the same.

And having the equality output, the circuit design will become as shown in Figure 2.3.

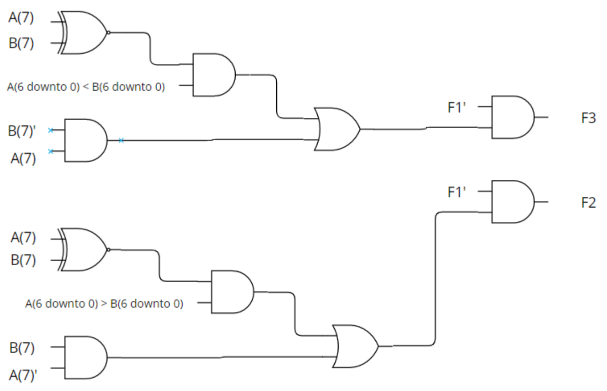


Figure 2.3: 8-bit signed comparator using magnitude comparator and sign bit.

A will be greater than B if both have the same sign and the magnitude of the least significant bits of A are greater than B and they are not equal or if the sign of A was positive and B is negative and they are not equal.

B will be greater than A if both have the same sign and the magnitude of the least significant bits of B are greater than A and they are not equal or if the sign of B was positive and A is negative and they are not equal.

Total latency = 2+14+7+7+7=37 ns.

3.Results

3.1. Stage 1

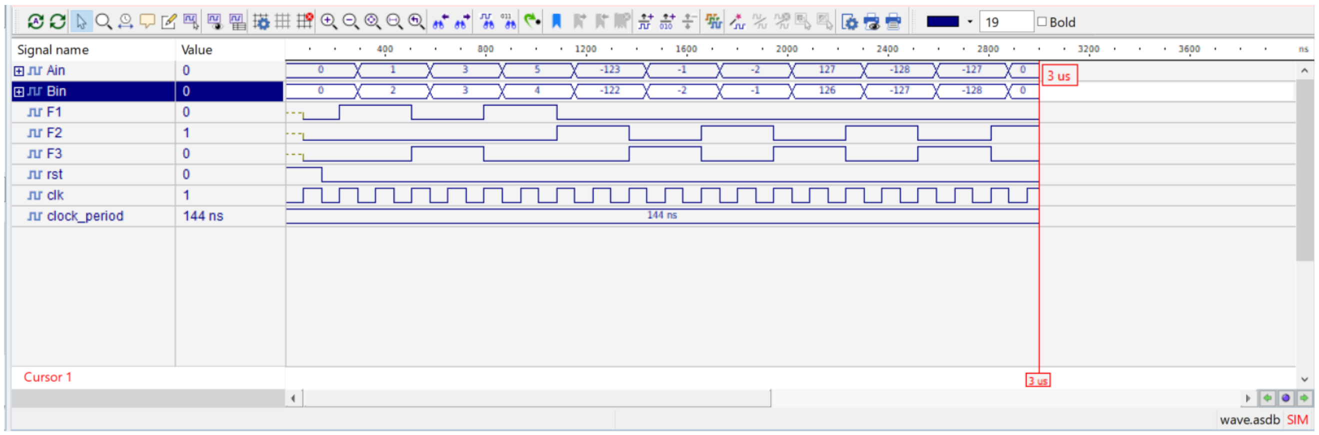


Figure 3.1: Stage 1 simulation.

As shown in Figure 3.1, results are correct for comparing both negative and positive numbers and large and small numbers, F1 when A=B and F2 when A>B and F3 when A<B. There is a 1 clock delay because inputs and outputs are connected to registers.

The design is also capable of detecting any error in the circuit using asserts as shown in Figure 3.2 and 3.3.

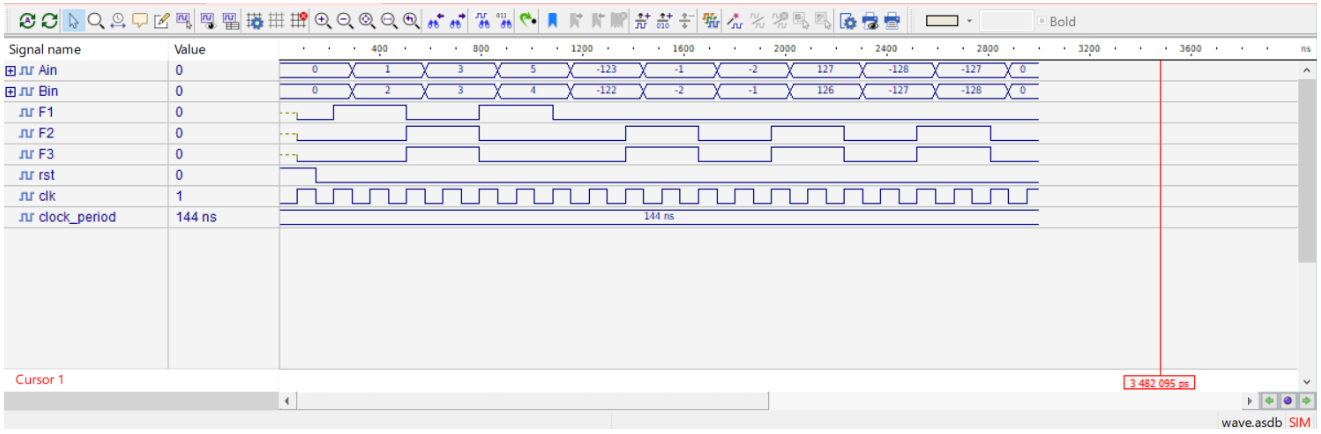
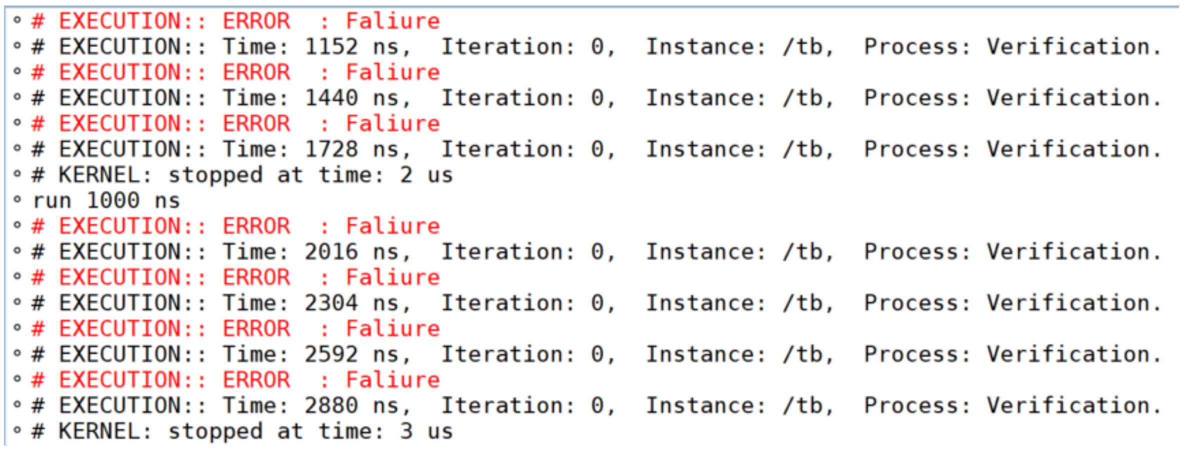


Figure 3.2: Stage 1 wrong circuit simulation.



```
# EXECUTION:: ERROR   : Faliure
# EXECUTION:: Time: 1152 ns,  Iteration: 0,  Instance: /tb,  Process: Verification.
# EXECUTION:: ERROR   : Faliure
# EXECUTION:: Time: 1440 ns,  Iteration: 0,  Instance: /tb,  Process: Verification.
# EXECUTION:: ERROR   : Faliure
# EXECUTION:: Time: 1728 ns,  Iteration: 0,  Instance: /tb,  Process: Verification.
# KERNEL: stopped at time: 2 us
run 1000 ns
# EXECUTION:: ERROR   : Faliure
# EXECUTION:: Time: 2016 ns,  Iteration: 0,  Instance: /tb,  Process: Verification.
# EXECUTION:: ERROR   : Faliure
# EXECUTION:: Time: 2304 ns,  Iteration: 0,  Instance: /tb,  Process: Verification.
# EXECUTION:: ERROR   : Faliure
# EXECUTION:: Time: 2592 ns,  Iteration: 0,  Instance: /tb,  Process: Verification.
# EXECUTION:: ERROR   : Faliure
# EXECUTION:: Time: 2880 ns,  Iteration: 0,  Instance: /tb,  Process: Verification.
# KERNEL: stopped at time: 3 us
```

Figure 3.3: Reporting error for wrong circuit of stage 1.

3.2. Stage 2

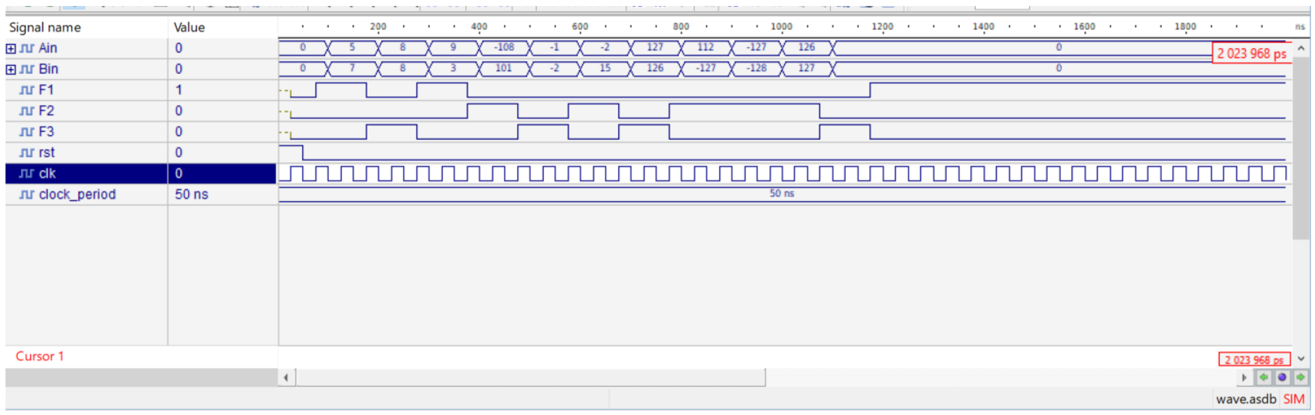


Figure 3.4: Stage 2 simulation.

As shown in Figure 3.4, results are correct for comparing both negative and positive numbers and large and small numbers, F1 when A=B and F2 when A>B and F3 when A<B. There is a 1 clock delay because inputs and outputs are connected to registers.

The design is also capable of detecting any error in the circuit using asserts as shown in Figure 3.5 and 3.6.

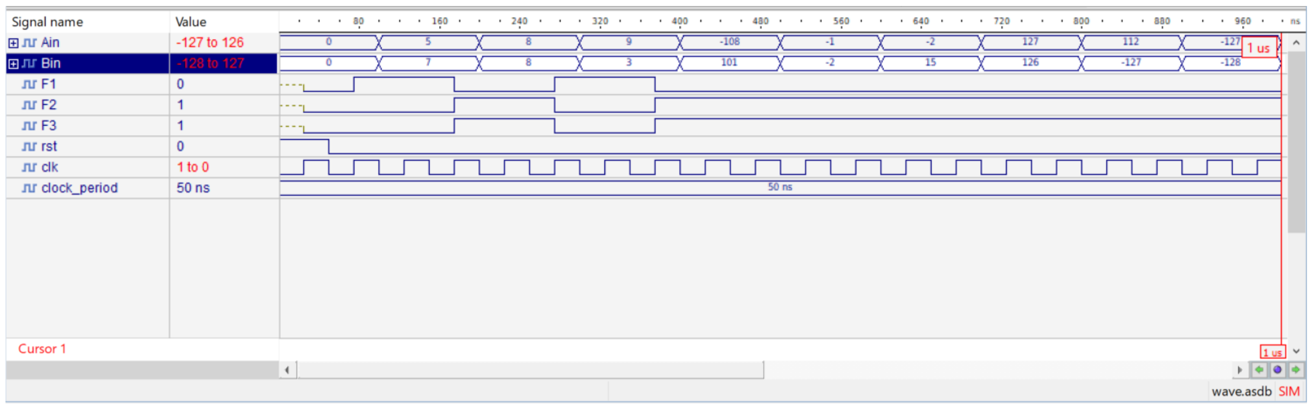
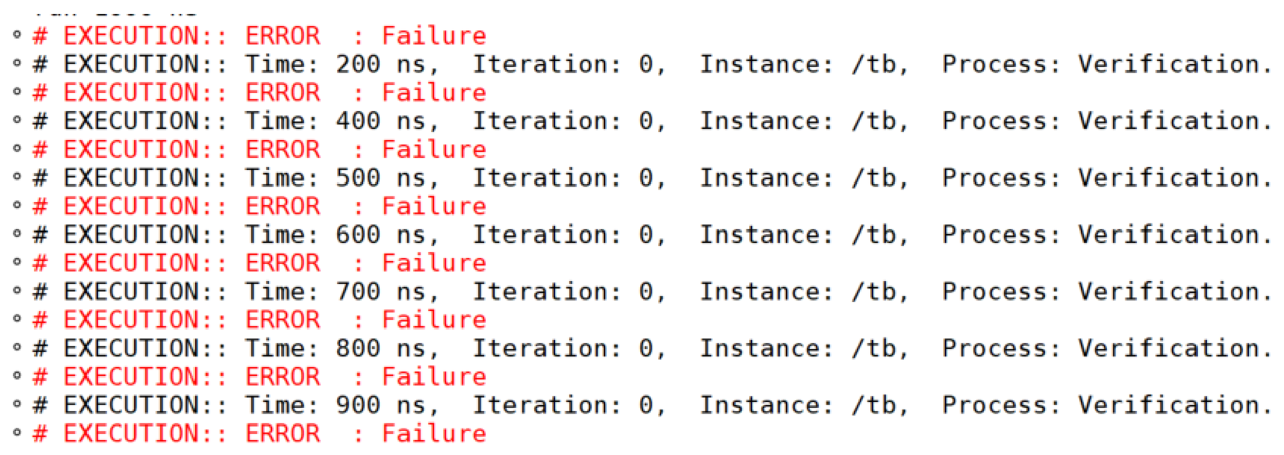


Figure 3.5: Stage 2 wrong circuit simulation



```
◦ # EXECUTION:: ERROR   : Failure
◦ # EXECUTION:: Time: 200 ns,  Iteration: 0,  Instance: /tb,  Process: Verification.
◦ # EXECUTION:: ERROR   : Failure
◦ # EXECUTION:: Time: 400 ns,  Iteration: 0,  Instance: /tb,  Process: Verification.
◦ # EXECUTION:: ERROR   : Failure
◦ # EXECUTION:: Time: 500 ns,  Iteration: 0,  Instance: /tb,  Process: Verification.
◦ # EXECUTION:: ERROR   : Failure
◦ # EXECUTION:: Time: 600 ns,  Iteration: 0,  Instance: /tb,  Process: Verification.
◦ # EXECUTION:: ERROR   : Failure
◦ # EXECUTION:: Time: 700 ns,  Iteration: 0,  Instance: /tb,  Process: Verification.
◦ # EXECUTION:: ERROR   : Failure
◦ # EXECUTION:: Time: 800 ns,  Iteration: 0,  Instance: /tb,  Process: Verification.
◦ # EXECUTION:: ERROR   : Failure
◦ # EXECUTION:: Time: 900 ns,  Iteration: 0,  Instance: /tb,  Process: Verification.
◦ # EXECUTION:: ERROR   : Failure
```

Figure 3.6: Stage 2 wrong circuit simulation.

4. Conclusion and Future works

In conclusion, both stages of the 8-bit signed comparator worked perfectly and results were correct after testing it using test benches with asserts that reports if there exist an error in the circuit as was shown. Stage 1 was designed in a generic form which means it can be updated to compare between any n-bits numbers but stage 2 wasn’t designed in a generic form which it can be updated to work on any number of bits on the future.

The signed 8-bit comparator can be used on security purposes such as opening doors of banks with passwords using microcontrollers, the comparator purpose is to compare between the number put by the person who wish to enter and the password that was defined before and thus the door will open by the signal sent when both numbers are equal (F1).

F2 and F3 can be used in evil ways, such as if the input was more or less than the password a signal will be sent to speakers that will alarm the user that someone is trying to open the door and security dogs’ cells will be opened to attack the thieve.

5. Appendix

5.1. Flip Flop Code

```
--D-Flip Flop Design For Outputs
library ieee;
use ieee.std_logic_1164.all;

entity flipflop is
port(
        clock   : in std_logic;
        enable  : in std_logic;
        reset : in std_logic;
        data_in: in std_logic;
        data_out        : out std_logic);
end flipflop;

architecture behavior of flipflop is
begin

reg:    process(clock)
                begin

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

                end process;

end behavior;
```

5.2. Registers Code

```
--generic register for inputs
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity regn is
generic(N : integer:=8 );
port(
        clk     : in std_logic;
        ena     : in std_logic;
        rst : in std_logic;
        din     : in STD_LOGIC_VECTOR(N-1 downto 0);
        dout    : out STD_LOGIC_VECTOR(N-1 downto 0));
end regn;

architecture regn of regn is
begin

reg:    process(clk)
                begin

                        if rising_edge(clk) then
                                if (rst = '1' ) then
                                        dout<= (OTHERS=>'0');
                                elsif (ena ='1') then
                                        dout <= din;
                                end if;
                        end if;

                end process;

end regn;
```

5.3. Full Adder Code

```
--Designing FA to creat Ripple Adder
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;


ENTITY FA IS
   PORT ( A : IN  STD_LOGIC;
          B : IN  STD_LOGIC;
          Cin : IN  STD_LOGIC;
          S : OUT  STD_LOGIC;
          Cout : OUT  STD_LOGIC);
END FA;

ARCHITECTURE Behavioral OF FA IS



SIGNAL st,ct:std_logic;
BEGIN
          ct<= ((B AND Cin) OR (A AND Cin) OR (A AND B)) AFTER 14 NS;
          st<= (A XOR B XOR Cin) AFTER 12 NS;
          S<= st ;
          Cout<= ct ;



END Behavioral;
```

5.4. Stage 1 Comparator Code

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY comparator IS
GENERIC(N: INTEGER := 8);
PORT(
        a: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        b: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        f1, f2, f3: OUT STD_LOGIC
        );
END comparator;
ARCHITECTURE synth OF comparator IS
SIGNAL S: STD_LOGIC_VECTOR(N-1 DOWNTO 0);              --Sum Signal
SIGNAL notB: STD_LOGIC_VECTOR(N-1 DOWNTO 0);                      --Inverting B to
calculate 2's Complement
SIGNAL t: STD_LOGIC_VECTOR(N DOWNTO 0);                                   -- Carry out of
every adder
SIGNAL z_flag, v_flag, n_flag: STD_LOGIC;

COMPONENT FA IS
   PORT ( A : IN  STD_LOGIC;
        B : IN  STD_LOGIC;
        Cin : IN  STD_LOGIC;
        S : OUT  STD_LOGIC;
        Cout : OUT  STD_LOGIC);
END COMPONENT;

BEGIN

        notB<= NOT(b) AFTER 2 NS;
          --Inverting B
        t(0) <= '1';
          -- Adding 1 after taking 1's complement of B to calculate 2's complement

        subtractor: FOR i IN 0 TO N-1 GENERATE-- Ripple Carry Adder
        BEGIN
                FA_X: FA PORT MAP( A => a(i), B=> notB(i), Cin =>t(i), S=>S(i), Cout=>t(i+1) );
          --Connecting Adders
  END GENERATE;
  z_flag <=  NOT (S(0) OR S(1) OR S(2) OR S(3) OR S(4) OR S(5) OR S(6) OR S(7)) AFTER 9 NS;
--Zero falg generation
  v_flag <= t(8) XOR t(7) AFTER 12 NS;                        --Overflow flag generation
  n_flag <= S(7);                          -- Negative flag generation
  f3 <= n_flag XOR v_flag AFTER 12 NS;                                              --A<B
using above flags
  f2 <= (n_flag XNOR v_flag) and (NOT z_flag) AFTER 18 NS;               -- A>B using same flags
  f1 <= z_flag ;
        -- A=B using zero flag

END synth;
```

5.5. Stage 1 Top Level View Code

```
--Top View Design to make testbench creation and simulating easier
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity top is
        generic (
                        N       : integer := 8
                        );
   Port ( Ain : in  STD_LOGIC_VECTOR(N-1 downto 0);
        Bin : in  STD_LOGIC_VECTOR(N-1 downto 0);

                         F1_reg, F2_reg, F3_reg : out STD_LOGIC;
                         rst: in STD_LOGIC;
                         clk: in STD_LOGIC
        );
end top;

architecture Behavioral of top is
signal Aout, Bout: STD_LOGIC_VECTOR(N-1 downto 0);
signal en_A, en_B, en_out: STD_LOGIC;
signal F1, F2, F3: STD_LOGIC;

component regn is
generic(N : integer:=8 );
port(
        clk     : in std_logic;
        ena     : in std_logic;
        rst : in std_logic;
        din     : in STD_LOGIC_VECTOR(N-1 downto 0);
        dout    : out STD_LOGIC_VECTOR(N-1 downto 0));
end component;

component flipflop is
port(
        clk     : in std_logic;
        ena     : in std_logic;
```

```
            rst : in std_logic;
            din       : in std_logic;
            dout     : out std_logic);
end component;

component comparator is
generic(N: integer := 8);
port(
            a: in STD_LOGIC_VECTOR(N-1 downto 0);
            b: in STD_LOGIC_VECTOR(N-1 downto 0);
            f1, f2, f3: out STD_LOGIC
            );
end component;
begin

en_A<= '1';
en_B<= '1';
en_out<= '1';

Reg_A: regn
                              generic map (N=> N)
                              port map(
                               clk=> clk,
                               ena=> en_A,
                               rst => rst,
                               din=> Ain,
                               dout=> Aout
                               );

Reg_B: regn
                              generic map (N=> N)
                              port map(
                               clk=> clk,
                               ena=> en_B,
                               rst => rst,
                               din=> Bin,
                               dout=> Bout
                               );


FF_F1: flipflop
                              port map(
                               clk=> clk,
                               ena=> en_out,
                               rst => rst,
                               din=> F1,
                               dout=> F1_reg
                               );
FF_F2: flipflop
                              port map(
                               clk=> clk,
                               ena=> en_out,
                               rst => rst,
                               din=> F2,
                               dout=> F2_reg
                               );
FF_F3: flipflop
                              port map(
                               clk=> clk,
                               ena=> en_out,
                               rst => rst,
                               din=> F3,
                               dout=> F3_reg
                               );
Comp_1: comparator
                   generic map( N => N)
                   port map(
                   a=> Aout,
                   b => Bout,
                   f1 => F1,
                   f2 => F2,
                   f3 => F3
                   );
end Behavioral;
```

5.6. Stage 1 Test Bench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.all;
ENTITY tb IS
END tb;

ARCHITECTURE behavioral OF tb IS
--Component Declaration for the Unit Under Test (UUT)
component  top is
        generic (
                        N       : integer := 8
                        );
   Port ( Ain : in  STD_LOGIC_VECTOR(N-1 downto 0);
        Bin : in  STD_LOGIC_VECTOR(N-1 downto 0);

                         F1_reg, F2_reg, F3_reg : out STD_LOGIC;
                         rst: in STD_LOGIC;
                         clk: in STD_LOGIC
        );
end component;

constant N: integer := 8;



SIGNAL Ain : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
SIGNAL Bin : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
SIGNAL F1, F2, F3 : STD_LOGIC;
SIGNAL rst: STD_LOGIC;
--Clock generation-------
SIGNAL clk:STD_LOGIC;
SIGNAL clock_period:time :=144ns; --period of clock

BEGIN
        -- Instantiate the Unit Under Test (UUT)
        UUT: top
        generic map (N=> N)
        PORT MAP(
        clk => clk,
        rst => rst,
  Ain=> Ain,
        Bin=> Bin,
        F1_reg=> F1,
        F2_reg=> F2,
        F3_reg=>F3
        );
        --Making Clock
        clock_process :process
                begin
                clk <= '0';
                wait for clock_period/2;
                clk <= '1';
                wait for clock_period/2;
        end process;
        -- Test casses
        Testing: process
        begin
        Ain<= (others => '0');
        Bin<= (others => '0');
        rst<= '1';

        wait for clock_period;
rst<= '0';

        wait for clock_period;
        Ain<= x"01";
        Bin<= x"02";
        wait for 2*clock_period;
        Ain<= x"03";
        Bin<= x"03";
        wait for 2*clock_period;
        Ain<= x"05";
        Bin<= x"04";
        wait for 2*clock_period;
        Ain<= x"85";
        Bin<= x"86";
        wait for 2*clock_period;
        Ain<= x"FF";
        Bin<= x"FE";
        wait for 2*clock_period;
        Ain<= x"FE";
        Bin<= x"FF";
        wait for 2*clock_period;
        Ain<= x"7F";
        Bin<= x"7E";
        wait for 2*clock_period;
        Ain<= x"80";
        Bin<= x"81";
        wait for 2*clock_period;
        Ain<= x"81";
        Bin<= x"80";
        wait for 2*clock_period;
        Ain<= x"00";
        Bin<= x"00";
        wait for 2*clock_period;

                wait;
        end process;
---------------------------------
        -- verifier
Verification: process
        variable  error_status :  boolean := true;
begin
        wait  on  Ain;
        wait  for   2*clock_period;
        if  ( (signed(Ain) =  signed(Bin) and  F1 = '1' and F2 = '0' and F3 = '0')  or
                (signed(Ain) > signed(Bin) and  F1 = '0' and F2 = '1' and F3 = '0')    or
                (signed(Ain) < signed(Bin) and  F1 = '0' and F2 = '0' and F3 = '1'))
        then
                error_status  :=  false;
        else
                error_status  :=  true ;
        end if;

-- error   reporting
        assert  not  error_status
        report  "Faliure"
        severity  error ;
end  process;



        END;
```

5.7. Stage 2 Comparator Code

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY comparator IS
GENERIC(N: INTEGER := 8);
PORT(
        a: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        b: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        f1, f2, f3: OUT STD_LOGIC
        );
END comparator;
ARCHITECTURE synth OF comparator IS
SIGNAL e: STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL agreat, aless: STD_LOGIC;                              -- Signals shown in Circuit in
Report
SIGNAL great, less: STD_LOGIC_VECTOR (6 DOWNTO 0);      --Magnitude Comparator(long circuit)
Signals
SIGNAL not_a,not_b : STD_LOGIC_VECTOR (7 DOWNTO 0); --Inverters
SIGNAL f1_temp: STD_LOGIC;
SIGNAL n1,n2,n3,n4,n5,n6,n7,n8,n9: STD_LOGIC;          --Circuit in Report Signals
BEGIN

        e_gen: FOR i IN N-1 DOWNTO 0 GENERATE
                e(i)<= a(i) XNOR b(i) AFTER 12 ns;
        END GENERATE;

                f1_temp <= e(0) AND e(1) AND e(2) AND e(3) AND e(4) AND e(5) AND e(6) AND e(7)
AFTER 7 ns;                          --A=B
                f1<= f1_temp;

not_a <= NOT a AFTER 2 ns;
not_b <= NOT b AFTER 2 ns;

great(6) <= a(6) and not_b(6) after 7 ns;
great(5) <= e(6) and a(5) and not_b(5) after 7 ns;
great(4) <= e(6) and e(5) and a(4) and not_b(4) after 7 ns;
great(3) <= e(6) and e(5) and e(4) and a(3) and not_b(3) after 7 ns;
great(2) <= e(6) and e(5) and e(4) and e(3) and a(2) and not_b(2) after 7 ns;
great(1) <= e(6) and e(5) and e(4) and e(3) and e(2) and a(1) and not_b(1) after 7 ns;
great(0) <= e(6) and e(5) and e(4) and e(3) and e(2) and e(1) and a(0) and not_b(0) after 7 ns;
agreat <= great(6) or great(5) or great(4) or great(3) or great(2) or great(1) or great(0) after 7 ns;   --A>B

less(6) <= not_a(6) and b(6) after 7 ns;
less(5) <= e(6) and not_a(5) and b(5) after 7 ns;
less(4) <= e(6) and e(5) and not_a(4) and b(4) after 7 ns;
less(3) <= e(6) and e(5) and e(4) and not_a(3) and b(3) after 7 ns;
less(2) <= e(6) and e(5) and e(4) and e(3) and not_a(2) and b(2) after 7 ns;
less(1) <= e(6) and e(5) and e(4) and e(3) and e(2) and not_a(1) and b(1) after 7 ns;
less(0) <= e(6) and e(5) and e(4) and e(3) and e(2) and e(1) and not_a(0) and b(0) after 7 ns;
aless <= less(6) or less(5) or less(4) or less(3) or less(2) or less(1) or less(0) after 7 ns;        --A<B


n1 <= a(7) XNOR b(7) AFTER 9 ns;
n2 <= not_a(7) AND b(7) AFTER 7 ns;
n3 <= not_b(7) AND a(7) AFTER 7 ns;
n4 <= a(7) XNOR b(7) AFTER 9 ns;
n5 <= n4 AND aless AFTER 7 ns;                                                                        --Signed
comparator signals as shown in Report
n6 <= n1 AND agreat AFTER 7 ns;
n7 <= n6 OR n2 AFTER 7 ns;
n8 <= n3 OR n5 AFTER 7 ns;
n9 <= not f1_temp after 2 ns;


f2 <= n9 AND n7 AFTER 7 ns;
f3 <= n8 AND n9 AFTER 7 ns;                         --Outputs




end synth;
```

5.8. Stage 2 Top Level View Code

```
--This file has all the components connected using port map so that the simulation of the testbench will be
much easier
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY top IS
        GENERIC (
                        N       : integer := 8
                        );
   PORT ( Ain : IN  STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        Bin : IN  STD_LOGIC_VECTOR(N-1 DOWNTO 0);

                         F1_reg, F2_reg, F3_reg : out STD_LOGIC;
                         rst: in STD_LOGIC;
                         clk: in STD_LOGIC
        );
end top;

architecture Behavioral of top is
signal Aout, Bout: STD_LOGIC_VECTOR(N-1 downto 0);
signal en_A, en_B, en_out: std_logic;
signal F1, F2, F3: std_logic;

component regn is
generic(N : integer:=8 );
port(
        clk     : in std_logic;
        ena     : in std_logic;
        rst : in std_logic;
        din     : in STD_LOGIC_VECTOR(N-1 downto 0);
        dout    : out STD_LOGIC_VECTOR(N-1 downto 0));
end component;

component flipflop is
port(
        clock   : in std_logic;
        enable  : in std_logic;
        reset : in std_logic;
        data_in: in std_logic;
        data_out        : out std_logic);
end component;

component comparator is
generic(N: integer := 8);
```

```
port(
        a: in STD_LOGIC_VECTOR(N-1 downto 0);
        b: in STD_LOGIC_VECTOR(N-1 downto 0);
        f1, f2, f3: out STD_LOGIC
        );
end component;

begin

en_A<= '1';
en_B<= '1';
en_out<= '1';

Register_A: regn
                        generic map (N=> N)
                        port map(
                         clk=> clk,
                         ena=> en_A,
                         rst => rst,
                         din=> Ain,
                         dout=> Aout
                         );

Register_B: regn
                        generic map (N=> N)
                        port map(
                         clk=> clk,
                         ena=> en_B,
                         rst => rst,
                         din=> Bin,
                         dout=> Bout
                         );

Comp_1: comparator
                generic map( N => N)
                port map(
                a=> Aout,
                b => Bout,

                f1 => F1,
                f2 => F2,
                f3 => F3
                );

Flip_Flop_F1: flipflop
                        port map(
                         clock=> clk,
                         enable=> en_out,
                         reset => rst,
                         data_in=> F1,
                         data_out=> F1_reg
                         );
Flip_Flop_F2: flipflop
                        port map(
                         clock=> clk,
                         enable=> en_out,
                         reset => rst,
                         data_in=> F2,
                         data_out=> F2_reg
                         );
Flip_Flop_F3: flipflop
                        port map(
                         clock=> clk,
                         enable=> en_out,
                         reset => rst,
                         data_in=> F3,
                         data_out=> F3_reg
                         );



end Behavioral;
```

5.9. Stage 2 Test Bench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;
ENTITY tb IS
END tb;

ARCHITECTURE behavioral OF tb IS
--Component Declaration for the Unit Under Test (UUT)
component  top is
        generic (
                        N       : integer := 8
                        );
   Port ( Ain : in  STD_LOGIC_VECTOR(N-1 downto 0);
        Bin : in  STD_LOGIC_VECTOR(N-1 downto 0);

                         F1_reg, F2_reg, F3_reg : out STD_LOGIC;
                         rst: in STD_LOGIC;
                         clk: in STD_LOGIC
        );
end component;

constant N: integer := 8;


SIGNAL Ain : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
SIGNAL Bin : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
SIGNAL F1, F2, F3 : STD_LOGIC;
SIGNAL rst: STD_LOGIC;
--Clock generation----------
SIGNAL clk:STD_LOGIC;
SIGNAL clock_period:time :=50ns;

BEGIN
        -- Instantiate the Unit Under Test (UUT)
        UUT: top
        generic map (N=> N)
        PORT MAP(
        clk => clk,
        rst => rst,
  Ain=> Ain,
        Bin=> Bin,
        F1_reg=> F1,
        F2_reg=> F2,
        F3_reg=>F3
        );

-- Clock process definitions
        clock_process :process
                begin
                clk <= '0';
                wait for clock_period/2;
                clk <= '1';
                wait for clock_period/2;
        end process;
        -- Test cases
        Testing: process
        begin
        Ain<= (others => '0');
        Bin<= (others => '0');
        rst<= '1';

        wait for clock_period;
  rst<= '0';

        wait for clock_period;
        Ain<= x"05";
        Bin<= x"07";
        wait for 2*clock_period;
        Ain<= x"08";
        Bin<= x"08";
        wait for 2*clock_period;
        Ain<= x"09";
        Bin<= x"03";
        wait for 2*clock_period;
        Ain<= x"94";
        Bin<= x"65";
        wait for 2*clock_period;
        Ain<= x"FF";
        Bin<= x"FE";
        wait for 2*clock_period;
        Ain<= x"FE";
        Bin<= x"0F";
        wait for 2*clock_period;
        Ain<= x"7F";
        Bin<= x"7E";
        wait for 2*clock_period;
        Ain<= x"70";
        Bin<= x"81";
        wait for 2*clock_period;
        Ain<= x"81";
        Bin<= x"80";
        wait for 2*clock_period;
        Ain<= x"7E";
        Bin<= x"7F";
        wait for 2*clock_period;
        Ain<= x"00";
        Bin<= x"00";
        wait for 2*clock_period;

                wait;
        end process;
----------------------------------
        -- verifier
Verification: process
        variable  error_status :  boolean := true;
begin
        wait  on  Ain;
        wait  for   2*clock_period;
        if  ( (SIGNED(Ain) =  SIGNED(Bin) and  F1 = '1' and F2 = '0' and F3 = '0')  or
                (SIGNED(Ain) > SIGNED(Bin) and  F1 = '0' and F2 = '1' and F3 = '0')        or
                (SIGNED(Ain) < SIGNED(Bin) and  F1 = '0' and F2 = '0' and F3 = '1'))
        then
                error_status  :=  false;
        else
                error_status  :=  true ;
        end if;

-- error   reporting
        assert  not  error_status
        report  "Failure"
        severity  error ;
end  process;




        END;
```