Faculty of Engineering & Technology Electrical & Computer Engineering Department

ADVANCED DIGITAL DESIGN ENCS3310 COURSE PROJECT

Instructor: Dr.Abdallatif Abuissa

Student Name: Rami Majadbeh

Student ID: 1190611

26.12.2021

Abstract

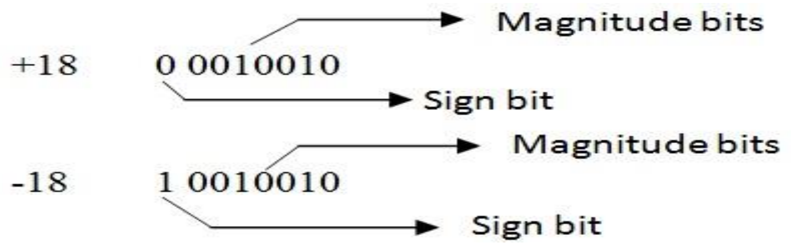
In this project, we learn about 2 methods of comparing signed numbers, the first is with the use of a ripple adder and how could it be modified to suit the need, another is using the conventional magnitude comparator to compare the signed bit with some modification as well, in general comparators are a very important in digital systems, and its efficiency and speed are so important in devices nowadays.

Table of contents

Introduction

Signed comparison

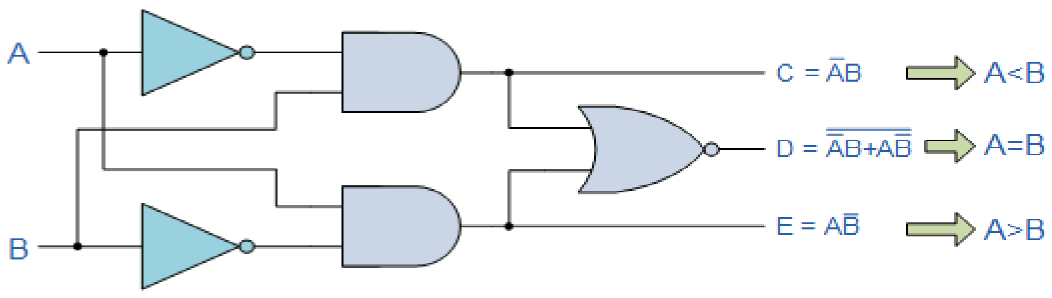
Signed magnitude comparison is comparing two numbers according to the most significant bit of the number, in this project we will be comparing 8-bit vectors in two different ways (adder/subtractor & magnitude comparator) involving the 8th bit as the sign bit as follows:



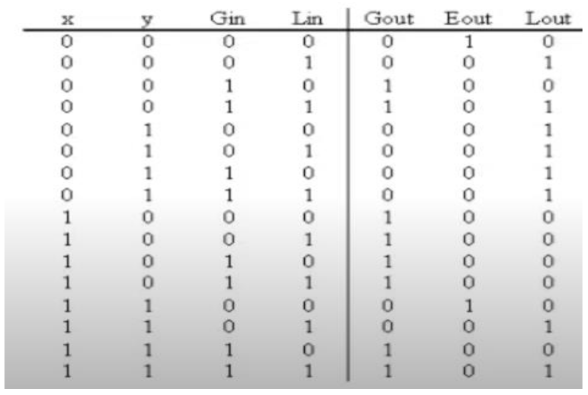
In the past, the magnitude comparator was explained with unsigned numbers, but with signed representation, things are a little bit different. Before diving into the details, let's review some basic logical circuits.

Unsigned magnitude comparator

The famous magnitude comparator used to compare two numbers with each other with no negative values, a comparator for one bit is as follows:



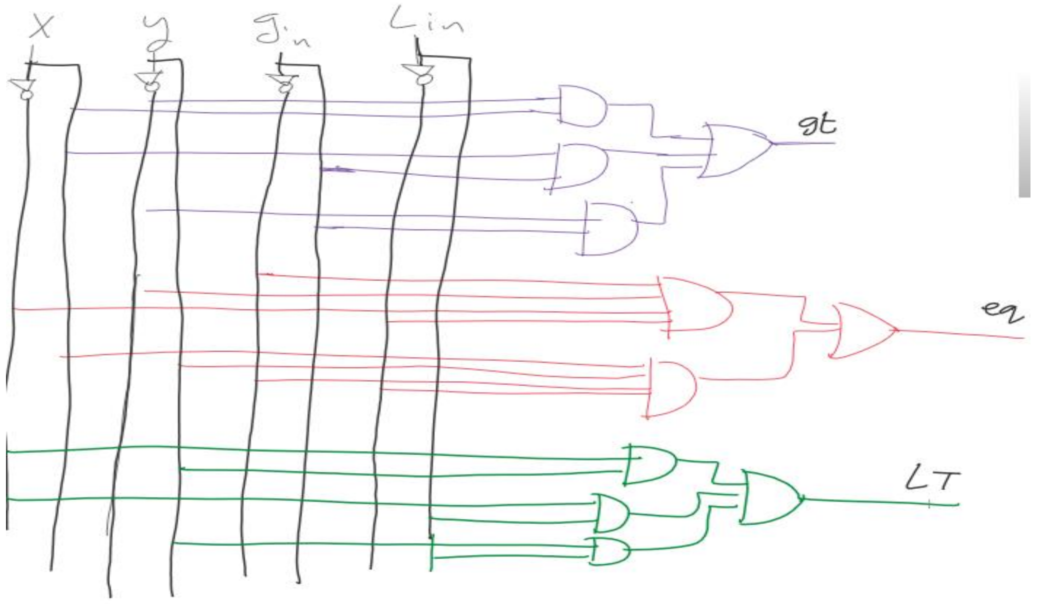
But in this project we use ripple-like comparator so GTin and LTin are required as the shown truth table:



x	y	Gin	Lin	Gout	Eout	Lout
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	1	0	0
0	0	1	1	1	0	1
0	1	0	0	0	0	1
0	1	0	1	0	0	1
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	1	0	0
1	0	1	1	1	0	0
1	1	0	0	0	1	0
1	1	0	1	0	0	1
1	1	1	0	1	0	0
1	1	1	1	1	0	1

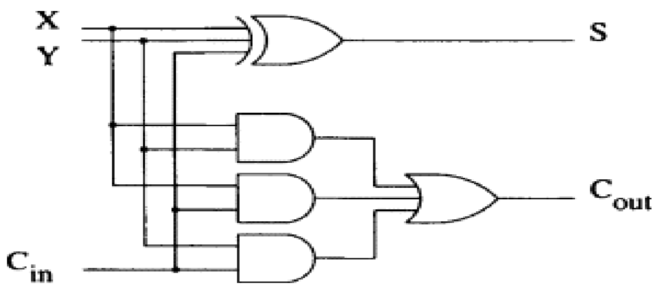
After solving the circuit will look like this

*will be used in stage 2



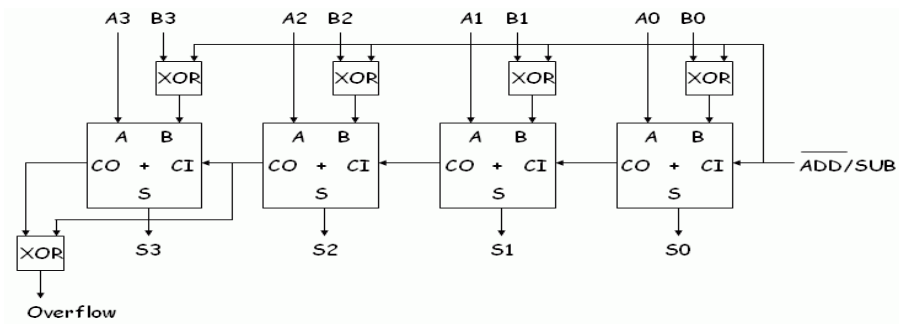
Full adder/subtractor

Full adder is a logical circuit that improves the half adder circuit as by performing addition on two binary numbers plus the carry-in (Cin) and produces two outputs, sum and carry-out (Cout) as there below, S = X XOR Y XOR Cin, Cout = X AND Y OR Y AND Cin OR X AND Cin.



n-bit Adder/Subtractor

Created using n 1-bit adder/subtractor using the model above, Cin = ‘0’ for adder, cin = ‘1’ for subtractor,



Note that the adder used here is a 4-bit adder, as the number of the adders equals the number of bits of the two numbers.

D FlipFlop / Register

Uses a clock to synchronize or move the input to the output on the clock trigger (rising or falling edge). Rising edge in this project, regardless, to create a register we simply create the needed number of flip-flops associated with the number of bits that are passed through.

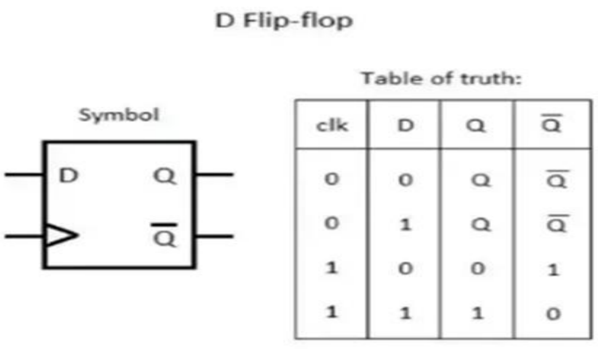
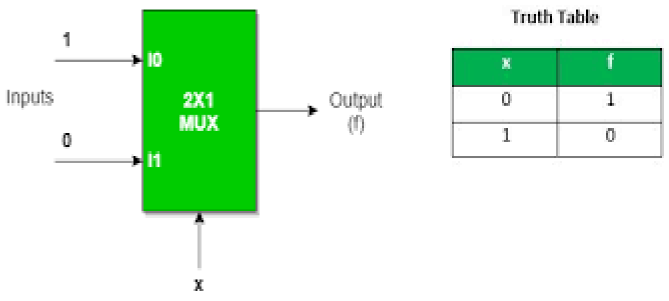


Table of truth:

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

Multiplexer:

A design consists of 2^n inputs and n select lines to pass one of the inputs to the output, in this project, mux is used widely in all of the comparators built.



Truth Table

x	f
0	1
1	0

Design Philosophy

Stage 1

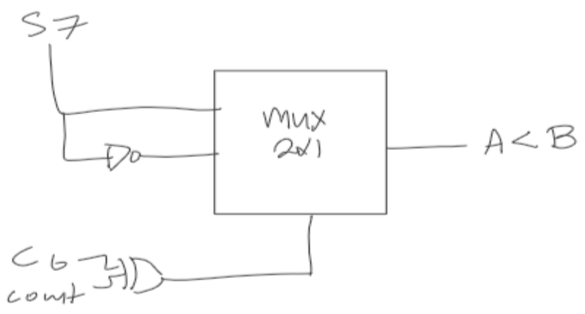
Subtractor Comparator

First, one-bit full adder was created and n-bit adder was constructed from the 1-bit adder, as well as, a D-flip flop to create two registers to feed the inputs and the outputs to, Then the adder comparator entity was created, which takes two inputs as numbered that are being compared: A , B that are 8-bit wide, with the 8th bit as a sign bit, also, the clock to be fed into the flip-flops for the outputs: we produced sum,Cout,V(Overflow), and most importantly: A_Greaterthan_ B as gt, A_Lessthan_ B as lt, A_Equal_B as eq.

To create the design we can observe that when we subtract two numbers as in A-B by performing 2's complement as shown in the background section, if the sign bit is equal to 1, then A is less than B, else, if the Sum is zero then the two numbers are equal, otherwise, the A is greater than. But, in some cases when the overflow is 1, it may cause some problems, and produce wrong answers, as in this example: (we take a 4-bit example for simplification and better understanding)

ex: -3 – 6 = 1101 – 0110 = 1101 + 1001(1's comp.) + 1(to convert to 2's) = 1 0101, but overflow . The sign bit (most significant bit) is 0, so the Comparator computes the output incorrectly as if -3 is greater than 6. In general, when there is overflow, problems with outputs will occur.

In order to solve this problem we do the following



That produces two cases:

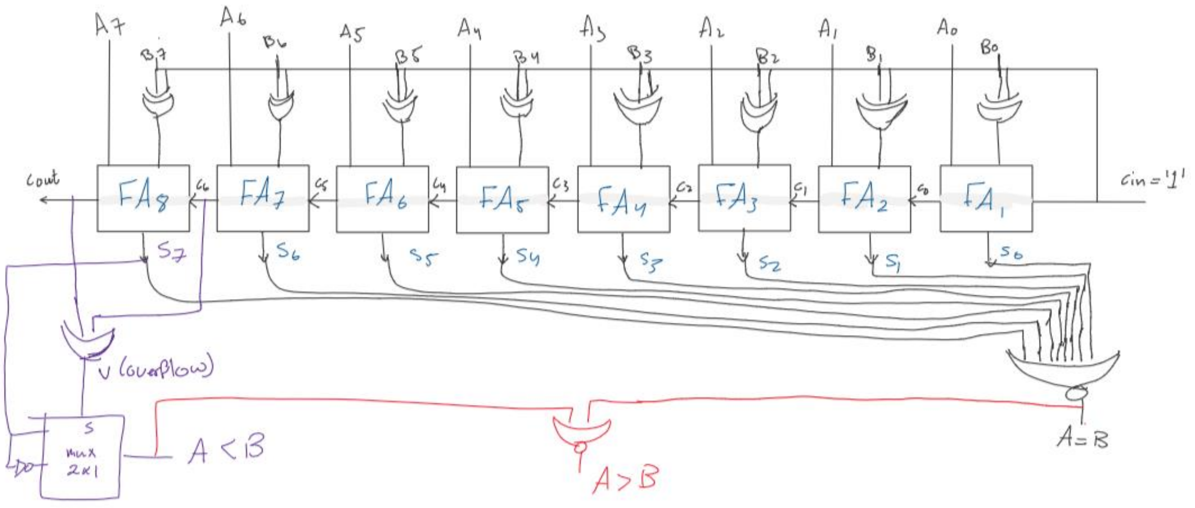
Case 1: when Overflow is '1'

When the overflow is equal to '1', in general the circuit does not execute correctly, as shown from the behaviour when solved by hand and simple testbenches, so in order to fix that now that the MSB is inverted when the Overflow is '1' so in order to fix that we create a 2X1 MUX that takes the MSB and its complement as an input and overflow as a select line to the MUX, so when Overflow occurs we invert the MSB to get the right answer in terms of A<B.

Case 2: when Overflow is ‘0’

This case has the same inputs, outputs and select lines as the first case, but when no overflow occurs the MSB will be passed as it is and no modification will be made.

The circuit below shows the whole process for all outputs:

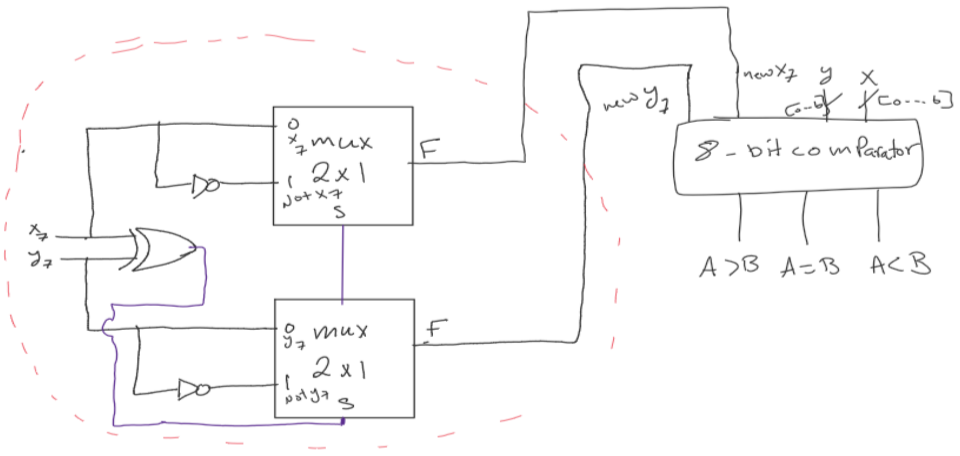


Stage 2

Signed magnitude comparator

First, one-bit comparator was created and n-bit comparator was constructed from the 1-bit comparator, as well as, a D-flip flop to create two registers to feed the inputs and the outputs to, Then the magnitude comparator entity was created, which takes two inputs as numbered that are being compared : A , B that are 8-bit wide, with the 8th bit as a sign bit, also, the clock to be fed into the flip-flops for the outputs: A_Greaterthan_ B as gt, A_Lessthan_ B as lt, A_Equal_B as eq.

The design in this stage is not complicated, as we use the magnitude comparator with a slight modification, and that modification obviously is on the sign bit, or the MSB of both numbers, because if one has sign bit as ‘1’ as the other has ‘0’ then the one with ‘0’ is larger and the one with ‘1’ is less, in order to implement that structurally, we must compare the two most significant bits of both numbers, because originally the magnitude comparator will assign the number with the MSB as ’1’ as larger, but in signed comparator it is the exact opposite , as the number with ‘1’ as MSB is negative, so we do the following for the MSB.



taking a look at the red circle in the figure shown above, because it's where the modification happens, so take the two MSB of each number and pass them through an XOR gate which acts as a select in two muxes one for each which has the bit of the MSB and its complement.

We have two cases:

Case 1: when the MSB’s of both numbers are different

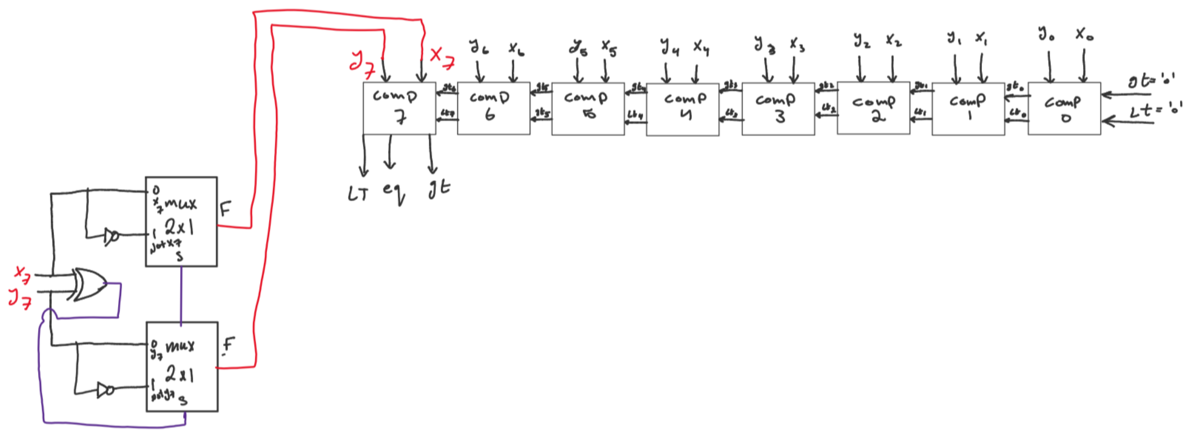
let's say X(7) = ‘1’ and Y(7) = ‘0’. for a normal magnitude comparator X is greater than Y but in a signed comparator it's the opposite, so passing X(7) and Y(7) into the XOR gate which in advance gives the output = ‘1’, and that is the select value to the two muxes, which gives us the complement of each bit, so X(7) is now

‘0’ and Y(7) is now ‘1’, passed into a normal magnitude comparator and it will give Y GREATER THAN x which is the output desired. Same case if (X(7) = ‘1’ and Y(7) = ‘0’)

Case 2: when the MSB’s of both numbers are the same

In this case, let's say X(7) = ‘1’ and Y(7) = ‘1’, both passing through the XOR will give ‘0’ which is the select line for both muxes which will pass the original value and does not invert, then passed through the magnitude comparator which will compare between the remaining bits X [6 down to 0], and Y [6 down to 0]. (same case if X(7) = ‘0’ and Y(7) = ‘0’)

Illustrating the whole circuit and design:



Note that the comparator that was introduced in the introduction is used.

Simulation

Stage 1

In order to build a full correct verification, we need to build 3 blocks, a test generator which generates all the possible inputs and its correct answers, a block of the comparator that was built by the designer with test vectors from the test generator as an input, as well as, a result analyser that compares the correct behaviourally built answers with the comparator that was built by the designer using the ASSERT statement.

The delay shown in the simulation has many factors, first is that inputs and outputs are fed into flipflops, second is that each gate has a specific delay to it.

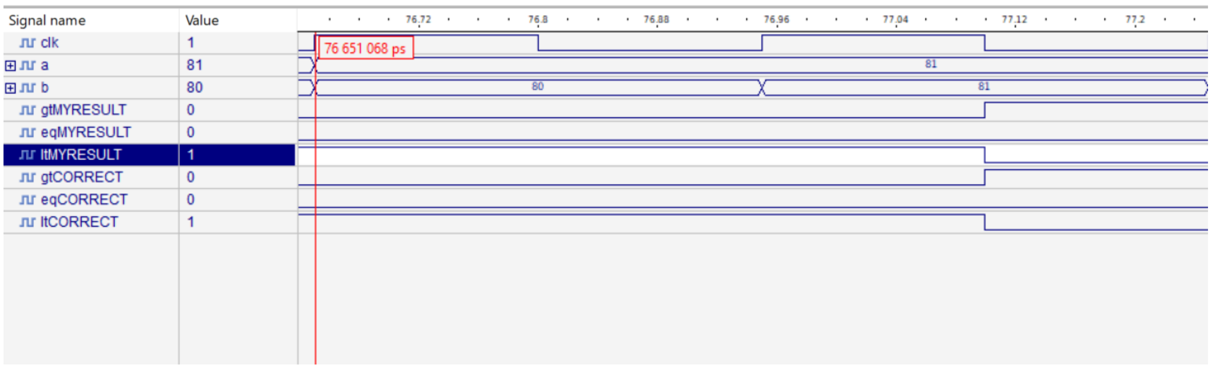
In order to show a correct simulation, many things are taken into consideration, first the worst clock latency can be up to 146ns as shown in the simple testbenches and simulations.

There is number of solutions to this problem, for example:

1- An additional clock could be added to the adder comparator, but since all entities better have the same clock, it wasn't included in the project.

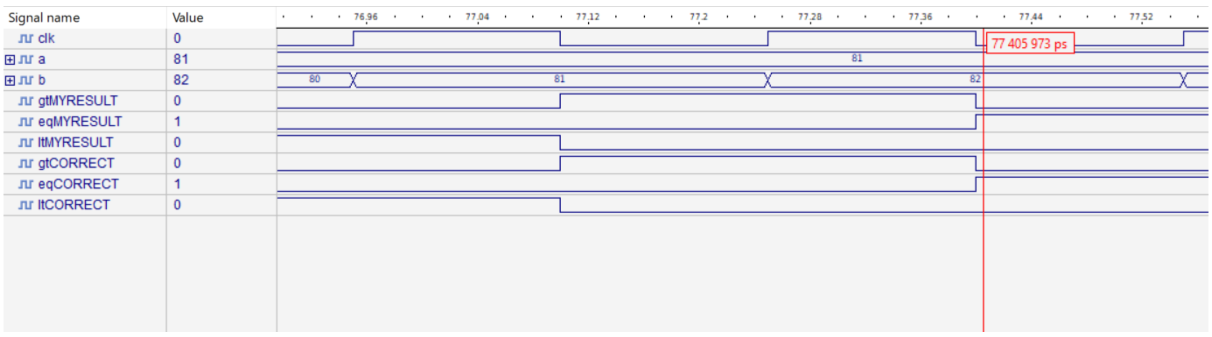
2- The solution that was implemented and is that the outputs of the adder comparator be on the falling edge of the clock to reduce the latency, as if it would be on the rising edge, then an additional clock pulse must be waited for, but on the falling edge we only wait the desired clock time.

So solution number 2 was used and clock was set to 150ns, simulation as follows:



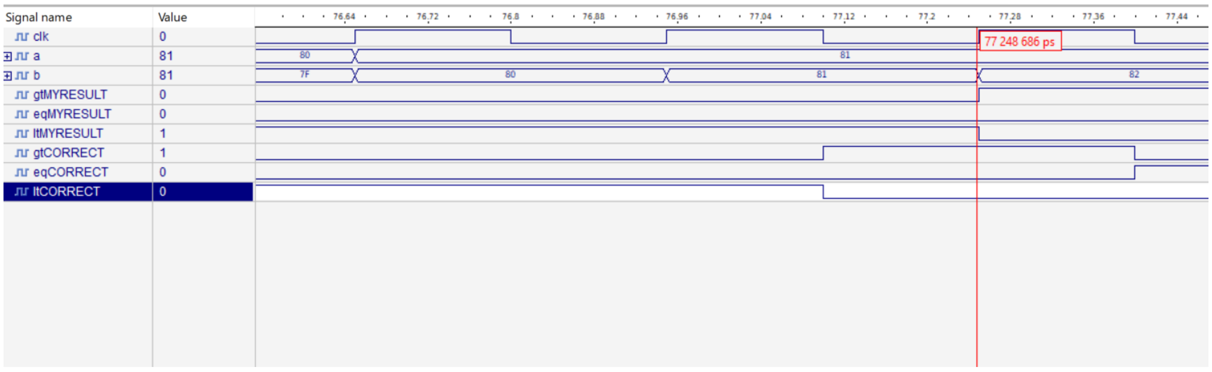
Signal name	Value
clk	1
a	81
b	80
gtMYRESULT	0
eqMYRESULT	0
ltMYRESULT	1
gtCORRECT	0
eqCORRECT	0
ltCORRECT	1

As seen in the figure above, we notice that the outputs change at the falling edge of the clock(one and a half cycle), so when a = '81' and b= '80', then its answer will be given in the middle of the next b number because it follows the falling edge, so that was it for the latency, so for example the a= '81' b='81' answer will be given in the a='80' b = '82' as follows:



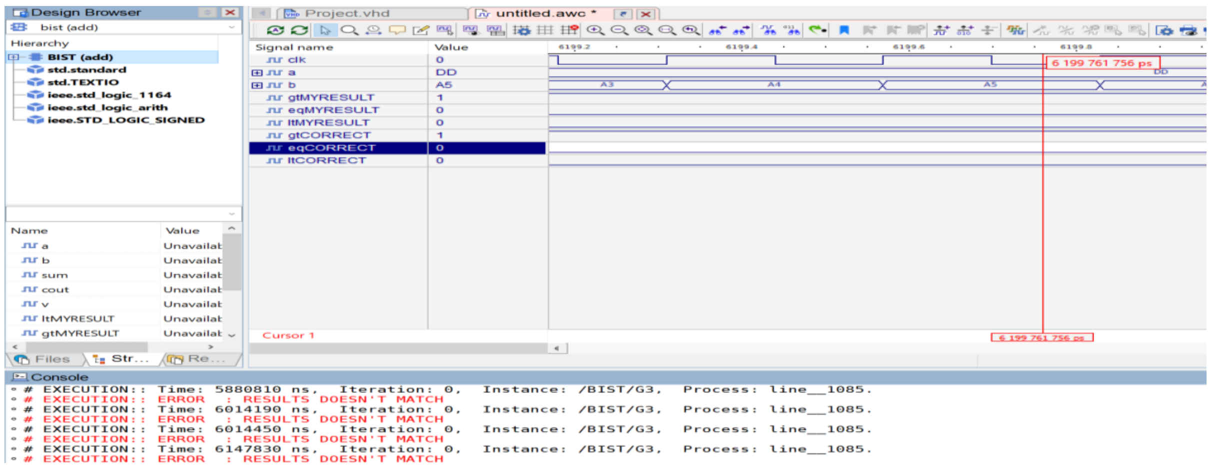
Notice the red marker to see the change, so in my opinion this is the best solution that could have been used for this problem.

To further explain let’s show if the output was put on the rising edge of the clock:



As shown above, the result of the wanted to numbers will be given after two clock cycles, which is not ideal, as well as the behavioural simulation needs to be delayed double the time it has.

Showing an error and warning, as shown below:



```
# EXECUTION:: Time: 5880810 ns,  Iteration: 0,  Instance: /BIST/G3,  Process: line__1085.
# EXECUTION:: ERROR  : RESULTS DOESN'T MATCH
# EXECUTION:: Time: 6014190 ns,  Iteration: 0,  Instance: /BIST/G3,  Process: line__1085.
# EXECUTION:: ERROR  : RESULTS DOESN'T MATCH
# EXECUTION:: Time: 6014450 ns,  Iteration: 0,  Instance: /BIST/G3,  Process: line__1085.
# EXECUTION:: ERROR  : RESULTS DOESN'T MATCH
# EXECUTION:: Time: 6147830 ns,  Iteration: 0,  Instance: /BIST/G3,  Process: line__1085.
# EXECUTION:: ERROR  : RESULTS DOESN'T MATCH
```

➔ done by simply changing the value of the clock

Stage 2

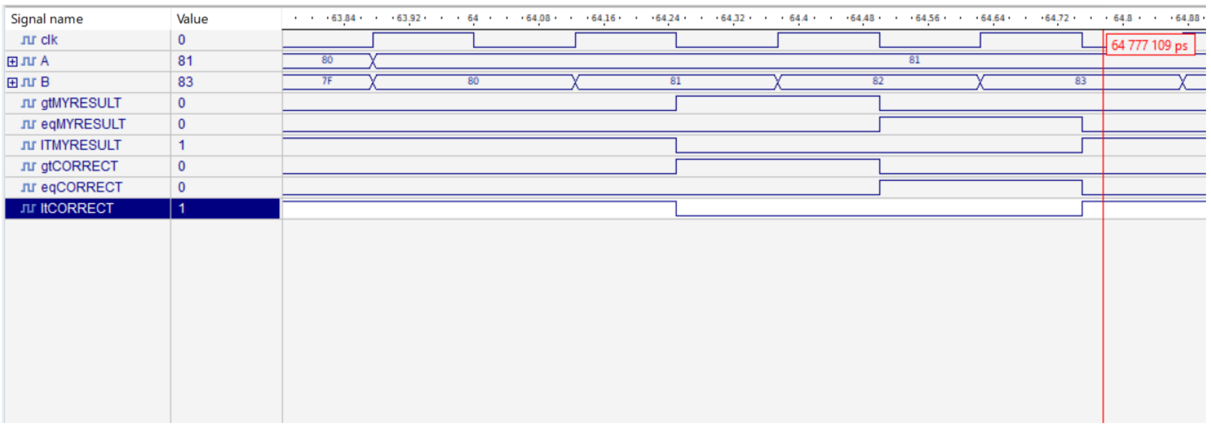
Same verification block was created as stage 1, a test generator which generates all the possible inputs and its correct answers, a block of the comparator that was built by the designer using the magnitude comparator with test vectors from the test generator as an input, as well as, a result analyser that compares the correct behaviourally built answers with the comparator that was built by the designer using the ASSERT statement.

In this section, the same problem happens with the delay, as discussed in the first stage it has multiple factors, first is that inputs and outputs are fed into flipflops, second is that each gate has a specific delay to it, and multiple solutions, to revise the solutions:

1- An additional clock could be added to the adder comparator, but since all entities better have the same clock, it wasn't included in the project.

2- The solution that was implemented and is that the outputs of the adder comparator be on the falling edge of the clock to reduce the latency, as if it would be on the rising edge, then an additional clock pulse must be waited for, but on the falling edge we only wait the desired clock time.

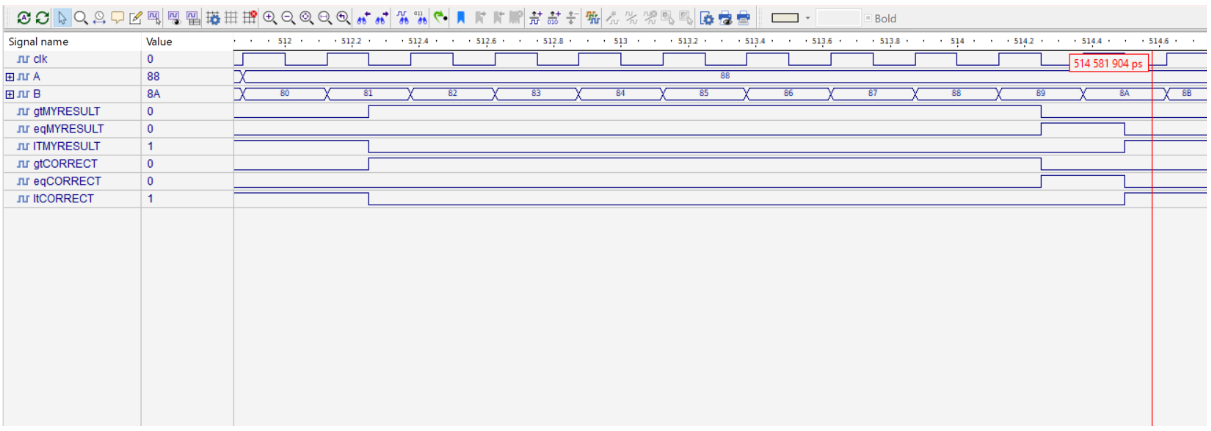
Now, after testing by hand and via testbenches, the maximum frequency is understood ti be approximately 115ns (used 125 in simulation to be more safe), the behavioural clock was delayed the same to match the result, so using the second method of putting the output flip-flops on the falling edge we get the same delay of output as stage 1 (one and a half cycle instead of two when it is set on the rising edge):



Signal name	Value
clk	0
A	81
B	83
gtMYRESULT	0
eqMYRESULT	0
ltMYRESULT	1
gtCORRECT	0
eqCORRECT	0
ltCORRECT	1

As shown in the figure above, the answer of the comparison of two numbers shows in the middle of comparing the next two numbers, and this example is great because it shows three different cases, so when a = '81' and b= '80', then its answer will be given in the middle of the next b number because it follows the falling edge, so that was it for the latency, so for example the a= ‘81’ b=’81’ answer will be given in the a=’80’ b = ‘82’ and so on to the end.

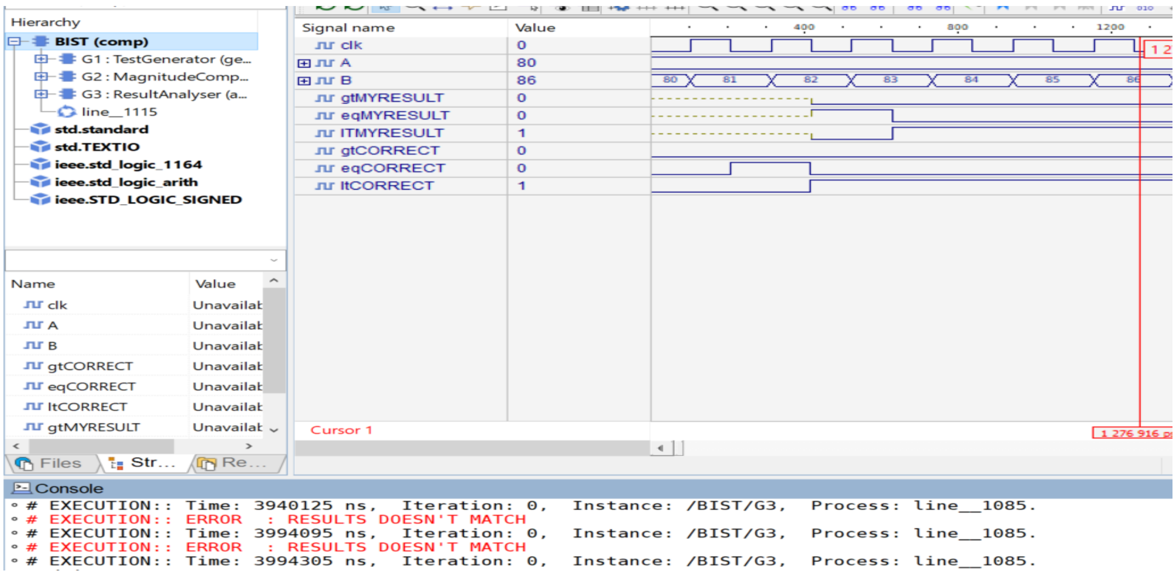
Another screenshot:



Signal name	Value
clk	0
A	88
B	8A
gtMYRESULT	0
eqMYRESULT	0
ltMYRESULT	1
gtCORRECT	0
eqCORRECT	0
ltCORRECT	1

Shows the same results as explained above.

Showing an error:



Signal name	Value
clk	0
A	80
B	86
gtMYRESULT	0
eqMYRESULT	0
ltMYRESULT	1
gtCORRECT	0
eqCORRECT	0
ltCORRECT	1

Name	Value
clk	Unavailab
A	Unavailab
B	Unavailab
gtCORRECT	Unavailab
eqCORRECT	Unavailab
ltCORRECT	Unavailab
gtMYRESULT	Unavailab

```
# EXECUTION:: Time: 3940125 ns,  Iteration: 0,  Instance: /BIST/G3,  Process: line__1085.
# EXECUTION:: ERROR  : RESULTS DOESN'T MATCH
# EXECUTION:: Time: 3994095 ns,  Iteration: 0,  Instance: /BIST/G3,  Process: line__1085.
# EXECUTION:: ERROR  : RESULTS DOESN'T MATCH
# EXECUTION:: Time: 3994305 ns,  Iteration: 0,  Instance: /BIST/G3,  Process: line__1085.
```

➔ As shown in the figure above.

Conclusion

Finally, after implementing the signed comparator in different working properly ways, it working is not more important than how fast, and efficient it works.

The adder comparator is simple to understand, at the end it’s the simple concept of subtracting numbers and the result sign that indicates the new subtracted number, but in digital design overflow can cause some problems that it solution takes new hardware and more time to execute.

The signed magnitude comparator on the other hand is simpler to understand and implement, but in terms of additional hardware it takes 2 MUXes to implement, but in general it takes less time to execute than the adder comparator in some cases, that may be a reason of the original structure of the adder being more redundant.

In the end, I think the two comparators are simple and produce good result in sufficient time taking all the flip-flops and delays into consideration, I think in order to make the designs better less use of hardware may help a lot, with new more innovative designs, as well as using more clock in the verification could have helped slightly.

References

1- https://www.youtube.com/watch?v=3epKbSGbMCA
2- Sarah L. Harris, David Money Harris, in Digital Design and Computer Architecture, 2016
3- https://www.sciencedirect.com/topics/computer-science/comparators
4- https://www.geeksforgeeks.org/magnitude-comparator-in-digital-logic

Appendix

```
--////////////////////Gates/////////////////////////////////////////////////////

--////////////////////and/////////////////////////////////////////////////////
library ieee;
use ieee.std_logic_1164.all;

ENTITY and2 is
        port(a,b: in std_logic;
        f: out std_logic);
end entity;

architecture simple of and2 is

begin

        f <= a and b after 7ns;

end;

 --and3
library ieee;
use ieee.std_logic_1164.all;

ENTITY and3 is
        port(a,b,c: in std_logic;
        f: out std_logic);
end entity;

architecture simple of and3 is

begin

        f <= (a and b and c) after 7ns;

end;

--and4
library ieee;
use ieee.std_logic_1164.all;

ENTITY and4 is
        port(a,b,c,d: in std_logic;
        f: out std_logic);
end entity;

architecture simple of and4 is

begin
```

```
        f <= (a and b and c and d) after 7ns;

end;


--and5
library ieee;
use ieee.std_logic_1164.all;

ENTITY and5 is
        port(a,b,c,d,e: in std_logic;
        f: out std_logic);
end entity;

architecture simple of and5 is

begin

        f <= (a and b and c and d and e) after 7ns;

end;

--and6
library ieee;
use ieee.std_logic_1164.all;

ENTITY and6 is
        port(a,b,c,d,e,g: in std_logic;
        f: out std_logic);
end entity;

architecture simple of and6 is

begin

        f <= (a and b and c and d and e and g) after 7ns;

end;

--and7
library ieee;
use ieee.std_logic_1164.all;

ENTITY and7 is
        port(a,b,c,d,e,g,h: in std_logic;
        f: out std_logic);
end entity;

architecture simple of and7 is

begin
```

```
		f <= (a and b and c and d and e and g and h) after 7ns;

end;

--and8
 library ieee;
use ieee.std_logic_1164.all;

ENTITY and8 is
		port(a,b,c,d,e,g,h,i: in std_logic;
		f: out std_logic);
end entity;

architecture simple of and8 is

begin

		f <= (a and b and c and d and e and g and h and i) after 7ns;

end;

library ieee;
use ieee.std_logic_1164.all;

ENTITY and9 is
		port(a,b,c,d,e,g,h,i,z: in std_logic;
		f: out std_logic);
end entity;

architecture simple of and9 is

begin

		f <= (a and b and c and d and e and g and h and i and z) after 7ns;

end;


--///////////////////////////not/////////////////////////////////////////////////////
 library ieee;
use ieee.std_logic_1164.all;

ENTITY not2 is
		port(a: in std_logic;
		f: out std_logic);
end entity;

architecture simple of not2 is

begin
```

```
            f <= not (a) after 2ns;


end;

   --///////////////////////nand/////////////////////////////////////////////////////////
library ieee;
use ieee.std_logic_1164.all;

ENTITY nand2 is
            port(a,b: in std_logic;
            f: out std_logic);
end entity;

architecture simple of nand2 is

begin

            f <= a nand b after 5ns;

end;


  --/////////////////////////////nor/////////////////////////////////////////////////////
library ieee;
use ieee.std_logic_1164.all;

ENTITY nor2 is
            port(a,b: in std_logic;
            f: out std_logic);
end entity;

architecture simple of nor2 is

begin

            f <= a nor b after 5ns;

end;

  --/////////////////////////////////or2//////////////////////////////////////////////

library ieee;
use ieee.std_logic_1164.all;

ENTITY or2 is
            port(a,b: in std_logic;
            f: out std_logic);
end entity;

architecture simple of or2 is
```

```
begin

            f <= a or b;

end;
--////////////////////////or3/////////////////////////////////////////////
library ieee;
use ieee.std_logic_1164.all;

ENTITY or3 is
            port(a,b,c: in std_logic;
            f: out std_logic);
end entity;

architecture simple of or3 is

begin

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

end;



--//////////////////////////////////////////////////////////////////////
  library ieee;
use ieee.std_logic_1164.all;

ENTITY or8 is
            port(a,b,c,d,e,i,g,h: in std_logic;
            f: out std_logic);
end entity;

architecture simple of or8 is

begin

            f <= (a or b or c or d or e or i or g or h) after 7ns;

end;




--/////////////////////xnor/////////////////////////////////////////////
library ieee;
use ieee.std_logic_1164.all;

ENTITY xnor2 is
            port(a,b: in std_logic;
            f: out std_logic);
```

```
end entity;

architecture simple of xnor2 is

begin

            f <= a xnor b after 9ns;

end;


--////////////////////////xor//////////////////////////////////////////
library ieee;
use ieee.std_logic_1164.all;

ENTITY xor2 is
            port(a,b: in std_logic;
            f: out std_logic);
end entity;

architecture simple of xor2 is

begin

f <= a xor b after 12ns;

end;


--/////////////////////////////////////////////////////////////////////////////
library    ieee;
use ieee.std_logic_1164.all;

entity onebit is
            port(x,y,g,l:in std_logic;
            gt,lt,eq: inout std_logic);
end entity;

architecture simple of onebit is
signal ynot,gnot,lnot,xnot,gt1,gt2,gt3,lt1,lt2,lt3: std_logic:='0';
begin

not1: entity work.not2(simple) port map(y,ynot);
not2d: entity work.not2(simple) port map(g,gnot);
not3: entity work.not2(simple) port map(l,lnot);
not4: entity work.not2(simple) port map(x,xnot);

gte1: entity work.and2(simple) port map(x,ynot,gt1);
gte2: entity work.and2(simple) port map(x,g,gt2);
gte3: entity work.and2(simple) port map(ynot,g,gt3);
gtef: entity work.or3(simple)  port map (gt1,gt2,gt3,gt);
```

```vhdl
lte1: entity work.and2(simple) port map (xnot,y,lt1);
lte2: entity work.and2(simple) port map (xnot,l,lt2);
lte3: entity work.and2(simple) port map (y,l,lt3);
ltef: entity work.or3(simple)  port map (lt1,lt2,lt3,lt);

eq1: entity work.nor2(simple) port map (lt,gt,eq);



--gt <= (x and  not y) or (x and g) or (not y and g);
--eq <= (not x and not y and not g and not l) or (x and y and not g and not l);
--lt <= (not x and y) or (not x and l) or (y and l);



end;

--/////////////////////////////////MUX/////////////////////////////////////////////////////////////////////////


library    ieee;
use ieee.std_logic_1164.all;

entity mux_n is

        port (a,b,s: in std_logic;
        y: out std_logic );
end;

architecture num1 of mux_n is

begin

        y <= a when s <= '0'
        else  b when s<= '1';


end;
--///////////////////////////////////////////////DFF//////////////////////////////////////////////////////////
library    ieee;
use ieee.std_logic_1164.all;

entity DFF is
        port(d,clk:in std_logic;
        f: out std_logic);
end entity;

architecture simple of DFF is
begin
        process(clk)
        begin

                if(rising_edge(clk)) then
                        f<=d ;
                end if;

        end process;
end;

architecture simple1 of DFF is
begin
        process(clk)
        begin

                if(falling_edge(clk)) then
                        f<=d ;
                end if;

        end process;
end;


--////////////////////////////////////////////stage2///////////////////////////////////////////////////


library ieee;
use ieee.std_logic_1164.all;

 entity MagnitudeComparator is
         port(aa,bb: in std_logic_vector(7 downto 0);
         clk : in std_logic;
         gt,eq,lt : inout std_logic);
 end entity;


 architecture simple of MagnitudeComparator is

 signal a,b,anew,bnew,abxor,aband,gtt,ltt,eqq: std_logic_vector(7 downto 0):= (others => '0');
 signal hxor,amux,bmux,ac,bc,aout,bout,gtm,ltm,eqm: std_logic;

 begin

         y: for i in 0 to 7 generate

                 dff3: entity work.DFF(simple) port map(aa(i),clk,a(i));
            dff4: entity work.DFF(simple) port map(bb(i),clk,b(i));

         end generate;


         --hxor <= a(7) xor b(7);
         h: ENTITY work.xor2(simple) PORT MAP (a(7),b(7),hxor);
```

```
		--ac <= not a(7);
		ww: ENTITY work.not2(simple) port map (a(7),ac);
		we: ENTITY work.not2(simple) port map (b(7),bc);
		--bc <= not b(7);

		 r: ENTITY work.mux_n(num1) port map (a(7),ac,hxor,aout);
		 z: ENTITY work.mux_n(num1) port map (b(7),bc,hxor,bout);


		g1: entity work.onebit(simple) port map (a(0),b(0),'0','0',gtt(0),ltt(0),eqq(0));
	g2: entity work.onebit(simple) port map (a(1),b(1),gtt(0),ltt(0),gtt(1),ltt(1),eqq(1));
		g3: entity work.onebit(simple) port map (a(2),b(2),gtt(1),ltt(1),gtt(2),ltt(2),eqq(2));
		g4: entity work.onebit(simple) port map (a(3),b(3),gtt(2),ltt(2),gtt(3),ltt(3),eqq(3));
		g5: entity work.onebit(simple) port map (a(4),b(4),gtt(3),ltt(3),gtt(4),ltt(4),eqq(4));
		g6: entity work.onebit(simple) port map (a(5),b(5),gtt(4),ltt(4),gtt(5),ltt(5),eqq(5));
		g7: entity work.onebit(simple) port map (a(6),b(6),gtt(5),ltt(5),gtt(6),ltt(6),eqq(6));
		g8: entity work.onebit(simple) port map (aout,bout,gtt(6),ltt(6),gtt(7),ltt(7),eqq(7));



	dff1: entity work.DFF(simple1) port map(gtt(7),clk,gtm);
		dff2: entity work.DFF(simple1) port map(eqq(7),clk,eqm);
		dff3: entity work.DFF(simple1) port map(ltt(7),clk,ltm);

gt <= gtm;
eq <= eqm;
lt <= ltm;


 end architecture simple;

--///////////////////////////////////////////////////////////////////////////////////////////////////////////

 library ieee;
use ieee.std_logic_1164.all;

 entity test is
		 end entity;



architecture t of test is

signal a,b: std_logic_vector(7 downto 0);
signal gt,clk,eq,lt,lmt,eqt,gtt: std_logic:= '0';

begin

				g1: entity work.MagnitudeComparator(simple) port map (a,b,clk,gt,eq,lt);
```

```
                        clk <= not clk after 0.25ns;
                        a <= "10000000","00000000" after 200ns,"01111111" after 400ns,"10000100" after
600ns,"10000000" after 800ns,"00000001" after 1000ns,"11111111" after 1200ns;
                        b <=  "00001000","00000001" after 200ns,"11111111" after 400ns,"10000001" after
600ns,"10000000" after 800ns,"00000010" after 1000ns,"11111111" after 1200ns;

end;

--/////////////////////////////////////////////////////stage1/////////////////////////////////////////////////

 library ieee;
use ieee.std_logic_1164.all;

entity onebitadder is
            port (a,b,cin: in std_logic;
            sum,cout:out std_logic);
end entity;


architecture simple of onebitadder is
  signal ab1,n1,n2,n3: std_logic;
begin
            g1: entity work.xor2 port map(a,b,ab1);
    g2: entity work.xor2 port map(cin,ab1,sum);

            g3: entity work.and2 port map (a,b,n1);
            g4: entity work.and2 port map (a,cin,n2);
            g5: entity work.and2 port map (b,cin,n3);
            g6: entity work.or3  port map (n1,n2,n3,cout);

end;

 --/////////////////////////////////////////adder comparartor/////////////////////////////////////////////////////////
 library ieee;
use ieee.std_logic_1164.all;

entity addercomp is
            port(aa,bb : in std_logic_vector(7 downto 0);
            sum: inout std_logic_vector(7 downto 0);
            cout,v,lt,eq,gt:inout std_logic:='0';
            clk: in std_logic);
end entity;


architecture simple of addercomp is
signal b,a,bcomp: std_logic_vector(7 downto 0);
signal carry: std_logic_vector(6 downto 0);
signal notsum7,gtt,ltt,eqq,gtm,ltm,eqm: std_logic;
begin

            df: for i in 0 to 7 generate
                        df1: entity work.dff(simple) port map(aa(i),clk,a(i));
```

```
                    df2: entity work.dff(simple) port map(bb(i),clk,b(i));
           end generate df;


          f1 : for i in 0 to 7 generate
                     g1 : entity work.xor2 port map('1',b(i),bcomp(i));
             end generate;


                    g2: entity work.onebitadder(simple) port map(a(0),bcomp(0),'1',sum(0),carry(0));
                    g3: entity work.onebitadder(simple) port map(a(1),bcomp(1),carry(0),sum(1),carry(1));
                    g4: entity work.onebitadder(simple) port map(a(2),bcomp(2),carry(1),sum(2),carry(2));
                    g5: entity work.onebitadder(simple) port map(a(3),bcomp(3),carry(2),sum(3),carry(3));
                    g6: entity work.onebitadder(simple) port map(a(4),bcomp(4),carry(3),sum(4),carry(4));
                    g7: entity work.onebitadder(simple) port map(a(5),bcomp(5),carry(4),sum(5),carry(5));
                    g8: entity work.onebitadder(simple) port map(a(6),bcomp(6),carry(5),sum(6),carry(6));
                    g9: entity work.onebitadder(simple) port map(a(7),bcomp(7),carry(6),sum(7),cout);

                    hh: entity work.xor2 port map (cout,carry(6),v);
                    not1: entity work.not2(simple) port map(sum(7),notsum7);
                    mux: entity work.mux_n(num1) port map (sum(7),notsum7,v,ltt);
                    no4: entity work.nor7(simple) port map
(sum(0),sum(1),sum(2),sum(3),sum(4),sum(5),sum(6),sum(7),eqq);
                    no5: entity work.nor2(simple) port map (eqq,ltt,gtt);


                    dfo: entity work.dff(simple1) port map (gtt,clk,gtm);
                    df6: entity work.dff(simple1) port map (ltt,clk,ltm);
                    df7: entity work.dff(simple1) port map (eqq,clk,eqm);

gt <= gtm;
eq <= eqm;
lt <= ltm;


end;
--////////////////////////////////simple testbench////////////////////////////////////////////////////////////////////////


 library ieee;
use ieee.std_logic_1164.all;

 entity test1 is
           end entity;



architecture t of test1 is

signal a,b,sum: std_logic_vector(7 downto 0);
signal cout,v,lt,eq,gt,clk:std_logic:='0';
```

```
begin

                g1: entity work.addercomp(simple) port map (a,b,sum,cout,v,lt,eq,gt,clk);

                clk <= not clk after 50 ns;
                a <= "11111111","00001000" after 200ns,"00000000" after 300ns,"00000001" after
400ns,"01111111" after 500ns,"10000100" after 600ns,"10000000" after 800ns,"00000001" after
1000ns,"11111111" after 1200ns;
                b <= "01111111","10000000" after 200ns,"00000001" after 300ns,"00000000" after
400ns,"11111111" after 500ns,"10000001" after 600ns,"10000000" after 800ns,"00000010" after
1000ns,"00000001" after 1200ns;

end;

--//////////////////////////////////////BEHVcomp////////////////////////////////////////////////////////////////////////////
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity behcomp is
        port(a,b: in signed(7 downto 0);
        gt,eq,lt: out std_logic);
end;


architecture simple of behcomp is

begin
        process(a,b)
begin

        gt <= '0';
        eq <= '0';
        lt <= '0';

        if (a>b) then
                gt <= '1';
        elsif (a < b) then
                lt <= '1';
        else
                eq <= '1';

        end if;


        end process;
end;


 --/////////////////////simple testbench//////////////////////////////////////////////////
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity test7 is
end entity;



architecture t3 of test7 is

signal a,b,sum: signed(7 downto 0);
signal cout,v,gt,lt,eq:std_logic;

begin

                g1: entity work.behcomp(simple) port map (a,b,gt,eq,lt);

                --clk <= not clk after 0.25ns;
                a <= "11111111","00001000" after 200ns,"00000000" after 300ns,"00000001" after
400ns,"01111111" after 500ns,"10000100" after 600ns,"10000000" after 800ns,"00000001" after
1000ns,"11111111" after 1200ns;
                b <= "01111111","10000000" after 200ns,"00000001" after 300ns,"00000000" after
400ns,"11111111" after 500ns,"10000001" after 600ns,"10000000" after 800ns,"00000010" after
1000ns,"01111111" after 1200ns;

end;

--/////////////////////////////Verification/////////////////////////////////////////////


  --/////////////////////////////test generator//////////////////////////////////////

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;
  A,B: out STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
  gt,eq,lt: out STD_LOGIC := '0');

end TestGenerator;

architecture generator of TestGenerator is
signal AA,BB: STD_LOGIC_VECTOR(7 downto 0):="00000000";
signal gtt,eqq,ltt: STD_LOGIC:='0';

BEGIN

A<=AA;
B<=BB;
```

```
gt<=gtt;
eq<=eqq;
lt<=ltt;

PROCESS (clk,AA,BB)

BEGIN


if(rising_edge(clk)) then
        if (AA > BB) then
                gtt <= '1'   after 125ns;
          eqq  <= '0'   after 125ns;
          ltt  <= '0'   after 125ns;
        elsif (AA < BB) then
                gtt <= '0' after 125ns;
          eqq  <= '0' after 125ns;
                ltt  <= '1' after 125ns;
        else
                gtt <= '0'  after 125ns;
                eqq  <= '1' after 125ns;
                ltt  <= '0' after 125ns;



        end if;
end if;
END PROCESS;


PROCESS
BEGIN
        FOR i IN -128 TO 127 LOOP
                FOR j IN -128 TO 127 LOOP


AA(7 downto 0) <= CONV_STD_LOGIC_VECTOR(i,8);
BB(7 downto 0) <= CONV_STD_LOGIC_VECTOR(j,8);
wait until rising_edge(CLK);

        END LOOP;
                END LOOP;

WAIT;
END PROCESS;
END;

--///////////////////////////for adder/////////////////////////////////////////////////////////////////

ARCHITECTURE simple of TestGenerator is
signal AA,BB: STD_LOGIC_VECTOR(7 downto 0):="00000000";
signal gtt,eqq,ltt: STD_LOGIC:='0';
```

```
BEGIN

A<=AA;
B<=BB;
gt<=gtt;
eq<=eqq;
lt<=ltt;

process (clk,AA,BB)

BEGIN


if(rising_edge(clk)) then
              if (AA > BB) then
                             gtt <= '1'  after 150NS;
                   eqq  <= '0'  after 150NS;
                   ltt  <= '0'  after 150NS;
              elsif (AA < BB) then
                             gtt <= '0' after 150NS;
                   eqq  <= '0' after 150NS;
                             ltt  <= '1'after 150NS;
              else
                            gtt <= '0'  after 150NS;
                            eqq  <= '1' after 150NS;
                            ltt  <= '0' after 150NS;



              end if;
end if;
END PROCESS;


PROCESS
BEGIN
              FOR i IN -128 TO 127 LOOP
                            FOR j IN -128 TO 127 LOOP



AA(7 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(i,8);
BB(7 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(j,8);
WAIT UNTIL rising_edge(CLK);

              end loop;
               end loop;



wait;
end process;
end;
```

```
--////////////////////////////////////////////////////////////////////////////////////////////////////
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_ARITH.ALL;

ENTITY ResultAnalyser IS
        PORT(CLK: IN STD_LOGIC;
        Cgt,Ceq,Clt,Rgt,Req,Rlt: IN STD_LOGIC);
        END ResultAnalyser;


ARCHITECTURE analyser OF ResultAnalyser IS
BEGIN

PROCESS
        BEGIN
        assert Cgt = Rgt
                and Ceq = Req
                and Clt =  Rlt
        report "RESULTS DOESN'T MATCH"
        severity error;

        WAIT UNTIL rising_edge(CLK);

        END PROCESS;
END;



LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_ARITH.ALL;

entity BIST is
end BIST;

--/////////////////////////////////////bench for comp//////////////////////////////////////////////////////////////
ARCHITECTURE comp OF BIST IS
signal clk: STD_LOGIC:='0';
signal A,B: STD_LOGIC_VECTOR(7 downto 0);
signal gtCORRECT,eqCORRECT,ltCORRECT,gtMYRESULT,lTMYRESULT,eqMYRESULT: std_logic;

BEGIN


clk <= not clk after 125 NS;
G1: entity WORK.TestGenerator(generator) port map(clk, A, B, gtCORRECT,eqCORRECT,ltCORRECT);
G2: entity WORK.MagnitudeComparator(simple) port map(A,
B,clk,gtMYRESULT,eqMYRESULT,ltMYRESULT);
G3: entity WORK.ResultAnalyser(analyser) port
map(clk,gtCORRECT,eqCORRECT,ltCORRECT,gtMYRESULT,eqMYRESULT,ltMYRESULT);
END;
```

```
--////////////////////////////////////////bench for adder/////////////////////////////////////////////////////
architecture add of BIST IS
signal a,b,sum:std_logic_vector(7 downto 0);
signal cout,v,ltMYRESULT,gtMYRESULT,eqMYRESULT,gtCORRECT,eqCORRECT,ltCORRECT:
std_logic;
signal clk: std_logic:='0';

begin

clk <= not clk AFTER 150NS;

G1: entity WORK.TestGenerator(simple) PORT MAP(clk, a, b, gtCORRECT,eqCORRECT,ltCORRECT);
G2: entity work.addercomp(simple) port map
(a,b,sum,cout,v,ltMYRESULT,eqMYRESULT,gtMYRESULT,clk);
G3: ENTITY WORK.ResultAnalyser(analyser) PORT
MAP(clk,gtCORRECT,eqCORRECT,ltCORRECT,gtMYRESULT,eqMYRESULT,ltMYRESULT);

end;
```