***Advanced Digital Systems Design***

***ENCS3310***



Computer systems engineering Department.

Project

Student Name: Ibrahim Nobani

Student ID: 1190278

Instructor: Aballatif abuissa

Section: 2

Date 26/12/2021

Abstract:

In this project we design and code an 8-bit Comparator for signed 2’s complement representation numbers in VHDL, first using a ripple adder and then with the use of a magnitude comparator, using Active HDL.

I

Contents

[ **Theory:** 1](#_Toc66625999)

[ **Procedure And Design Philosophy:** 2](#_Toc66626000)

1-[Stage 1 2](#_Toc2594742)

2-Stage 2 [6](#_Toc2594742)

[ **Conclusion:** 10](#_Toc66626000)

[ **References:** 11](#_Toc66626015)

[ **Appendix:** 12](#_Toc66626016)

* [Figures](Report2-2.docx)**:**

[Figure 1.1:Structural one bit adder 2](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 1.2:N-bit ripple subtractor 2](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 1.3:Stage 1 block diagram 3](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 1.4:Comparator using ripple part 1 3](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 1.5: Comparator using ripple part 2 4](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 1.6:Stage 1 normal simulation 4](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 1.7: Test Generator used for both stages 5](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 1.8:Stage 1 TestBench Simulation part 1 5](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 1.9:Stage 1 TestBench Simulation part 2 5](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 2.1:One bit comparator 6](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 2.2:Stage 2 N-bit comparator 6](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 2.3:Stage 2 block diagram 7](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 2.4:Equation Entity 8](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 2.5: Stage 2 normal simulation 8](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 2.6: Stage 2 TestBench Simulation part 1 9](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

[Figure 2.7: Stage 2 TestBench Simulation part 2 9](C:\\Users\\hp\\Desktop\\MicroLab\\Report2-2.docx)

II

Theory:

1. Ripple Adder:[1]

Multiple full adder circuits can be cascaded in parallel to add an N-bit number. For an N- bit parallel adder, there must be N number of full adder circuits. A ripple carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder.[1]

1. Magnitude Comparator:[2]

A magnitude digital Comparator is a combinational circuit that compares two digital or binary numbersin order to find out whether one binary number is equal, less than or greater than the other binary number.[2]

1. Sign Bit:[3]

The sign bit is a signed number representation that indicates the sign of a number, if the sign bit is 0, the number is non-negative (positive or zero), if the sign bit is 1 then the number is negative, the term may be used interchangeably with "most significant bit".[3]

1. D Flip Flop:[4]

The D flip-flop is an edge triggered device which transfers input data to Q on clock rising or falling edge.[4]

1

Procedure and Design Philosophy:

I Started the project by implementing each gate separately with their unique delays.

Stage 1:

In this stage I used the ripple adder/subtractor to compare the two 8-bit values, The ripple adder was implemented using an n-bit adder structured from one-bit adders. Check the 2 figures below.

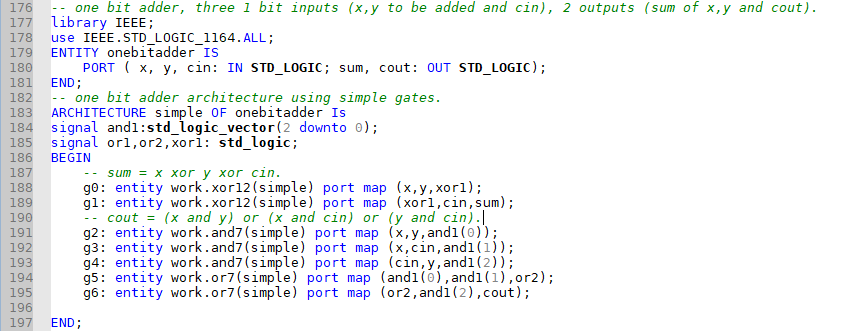


Figure 1-1 (Structural one-bit adder).

Using this one-bit adder code, I then implemented a simple Iterative n-bit adder (8-bit adder).

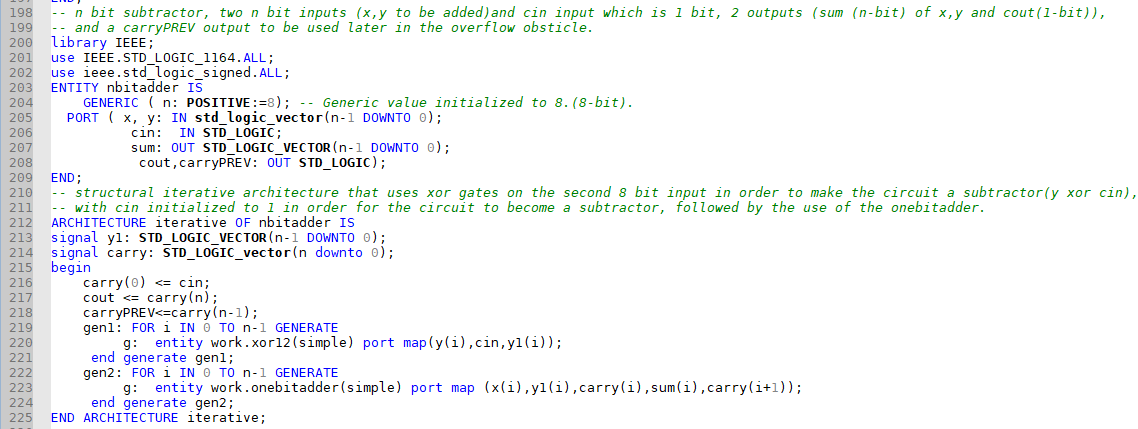


Figure 1-2 (N-bit ripple subtractor).

This adder was turned into a subtractor using xor gates on the second input, the xor gates are attached to the cin which is initialized to 1 in order for it to become a subtractor, then this subtractor using the 1-bit adders to complete its outputs. Refer to the block diagram below for more info about how the xor gates are connected.

2



Figure 1-3 (Stage 1 block diagram).

Notice how each bit is connected to a one-bit adder along with an xor gate attached to each cin that produces the 1’s complement of the second input and then the cin is added which results in turning it into a 2’s complement representation.

Lastly, I started building the comparator using the ripple subtractor, I generated the value of the 3 outputs using logic gates that I got from multiple k-maps then attached them together to get the desired output. If the sum is 0 then the result is equal, while if the last bit of the sum is 1 xor the overflow value and the result is not equal then its less, if neither then greater.

Equations Generated:

Equal =Not( sum(0) or sum(1) or sum(2) or sum(3) or sum(4) or sum(5) or sum(6) or sum(7)).

Less= (Overflow xor sum(7)) and not equal) .when overflow is (carry xor carryPREV).

Greater= Equal Nor Less.

These values can be preserved by looking at the block diagram above.

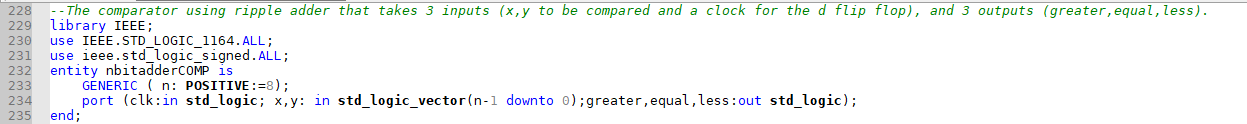


Figure 1-4 comparator using ripple part 1

3

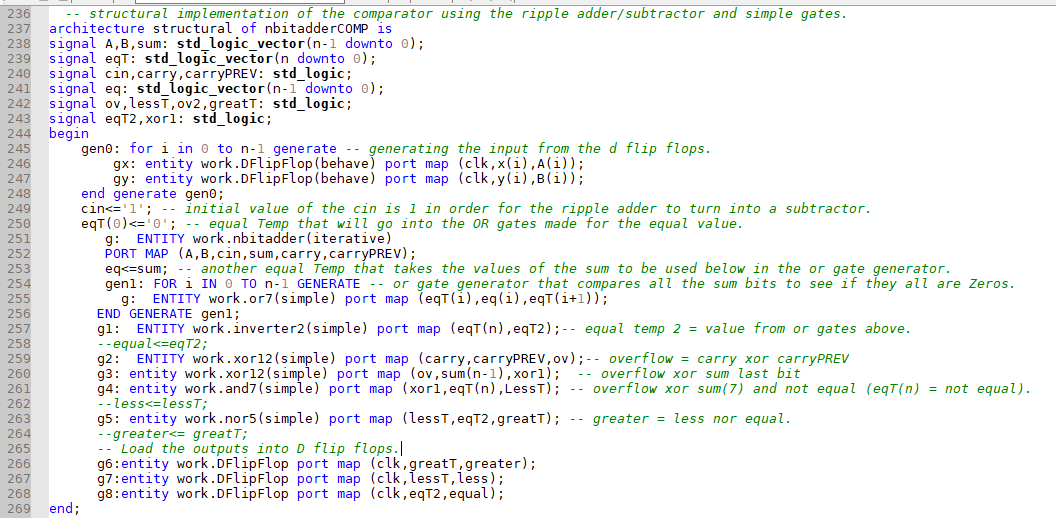


Figure 1-5 comparator using ripple part 2.

You can see in figures (1-4,1-5) how the comparator was implemented using gates in VHDL, and how the inputs/outputs were loaded from and into D-flip flops(lines 245-248/266-268).

Normal Simulation Results:

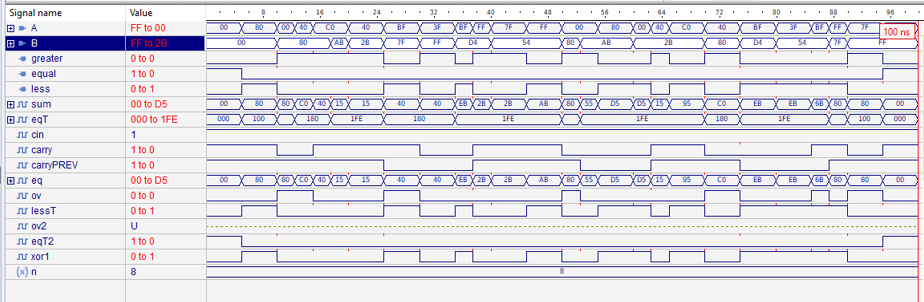


Figure 1-6 Stage 1 normal simulation.

As seen in the figure above the simulation results are correct.

Now I tried the simulation from the test generator and the result analyzer, the gates latency for this was 662 ns all combined. I used the same clock for the test generator, result analyzer and the d flip flop, the delay on the clock I used changes every 220 ns, I also used multiple wait clock statements in the test generator to balance things up a little bit, check the test generator figure below along with the testbench simulation result. Result analyzer and testbench codes are attached with the rest of the code in the appendix.

4

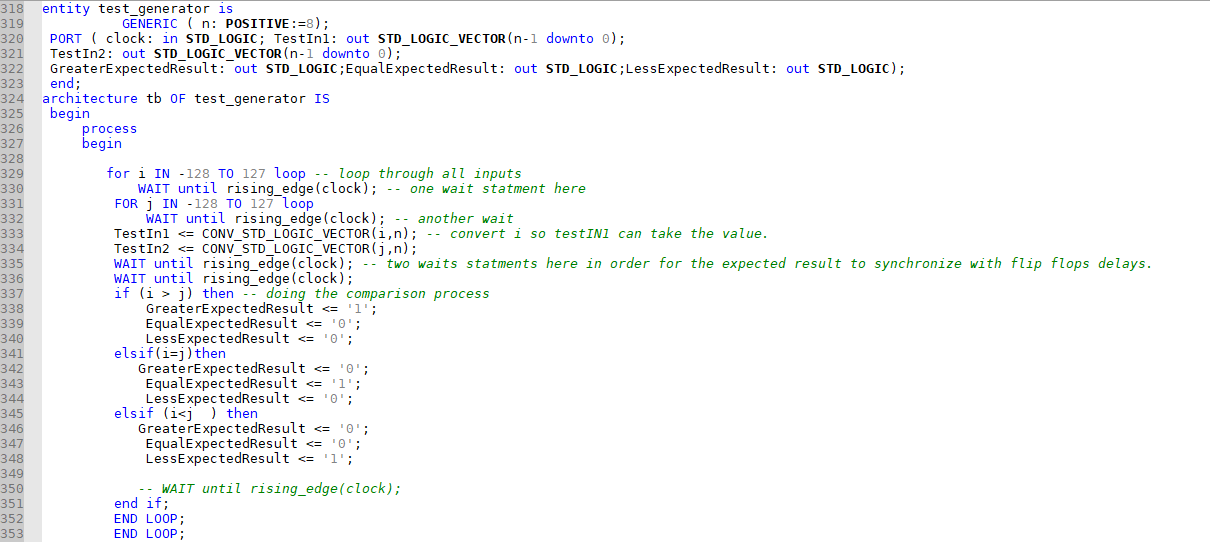


Figure 1-7 (test generator used for both stages).

Check the two wait statements in lines 316 and 317 that synchronize the expected outputs with the ones from the comparator. See Testbench simulation below, there is a 1760 ns delay:

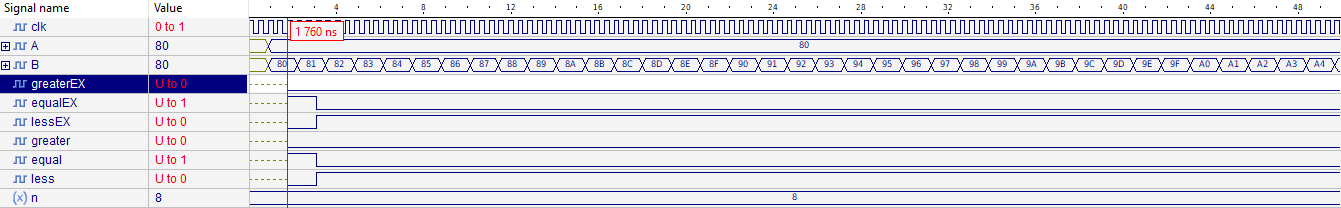
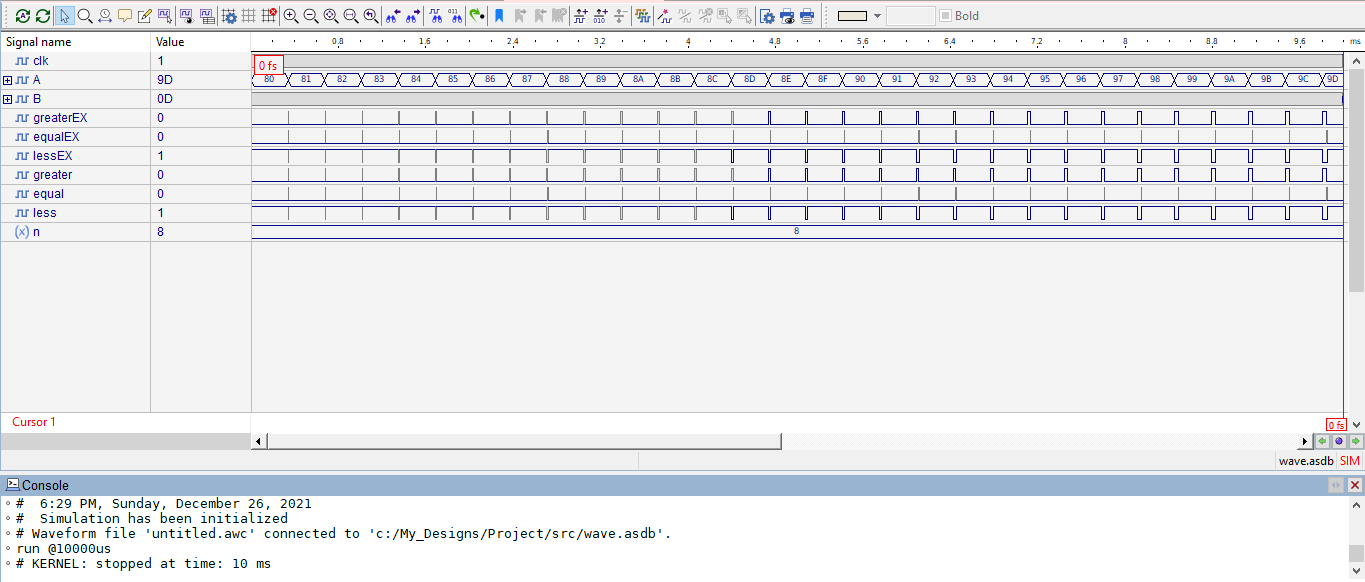


Figure 1-8 Stage 1 TestBench simulation part 1 Figure 1-9 Stage 1 TestBench Simulation part2.

As seen in the figure, no error message has been declared, so the circuit is correct.

5

Stage 2:

In this stage we used the magnitude comparator and the sign bit to implement the required comparator.

My method was using the one bit comparator to implement the n-bit comparator and then using the produced output in simple gates to give the exact solution.

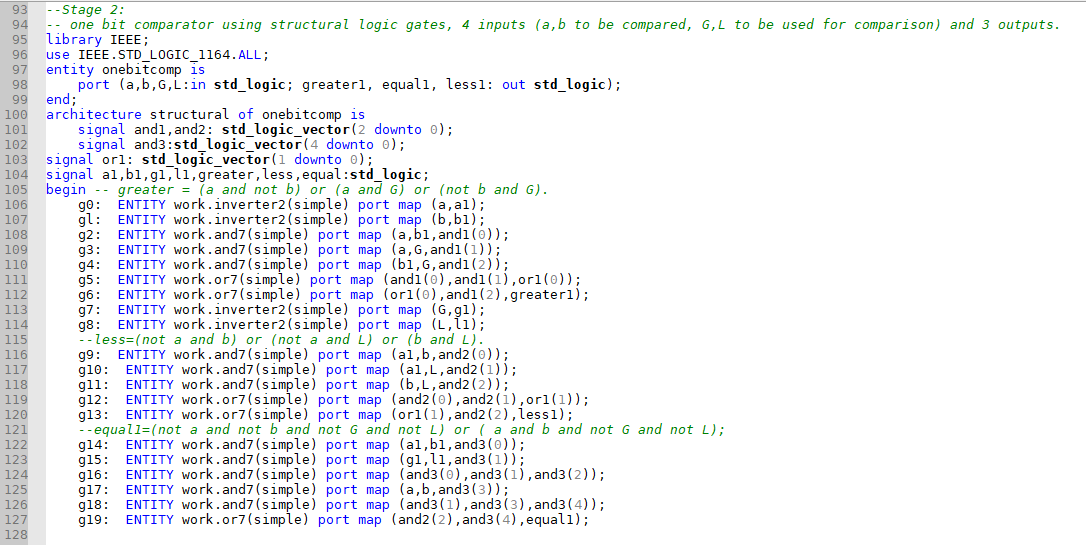


Figure 2-1 one-bit comparator.

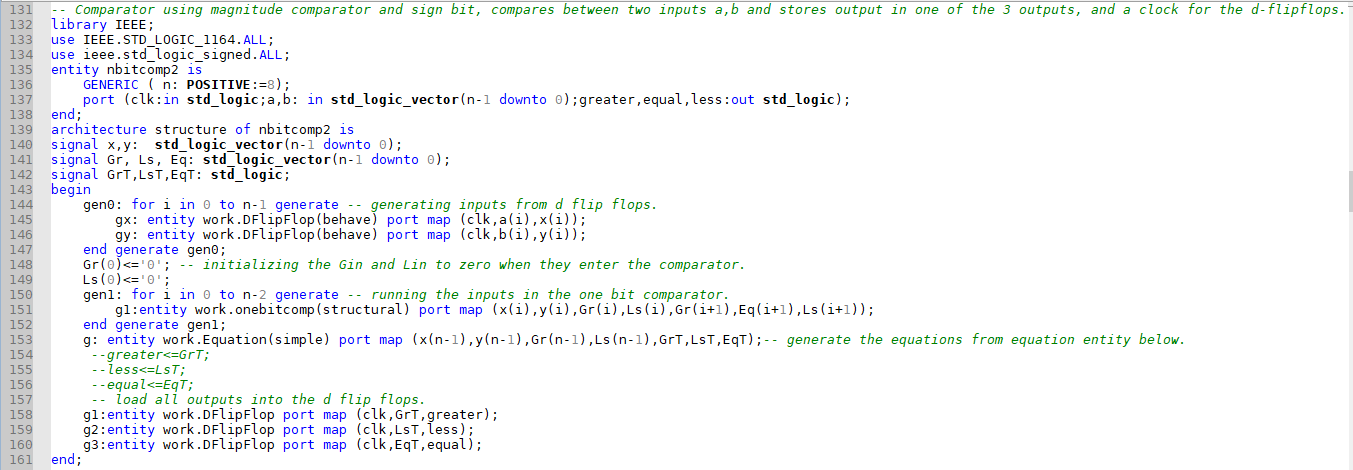
The one bit comparator works this way and was implemented upon:

The variable Greater1 is 1 if a>b or if a=b and G= 1.

The variable Less1 is 1 if a<b or if a=b and L= 1.

The variable Equal1 is 1 if a=b or and G= and L=0.

Then I coded the n-bit comparator structurally using this, see the image below:

 Figure 2-2 Stage 2 N-bit comparator

6

You can notice in lines (144-147) how the inputs were loaded from D flip flops into signals x,y and in lines 158-160 how outputs were loaded into the D flip flop.

Go back to figure 2-2 and check line 153, you will notice that I used an entity called equation to solve the comparator.

The equations generated to produce the exact output are:

Greater = (not x(7) and y(7)) or (not x(7) and not y(7) and gr) or (x(7) and y(7) and gr).

Less = (x(7) and not y(7)) or (not x(7) and not y(7) and ls) or (x(7) and y(7) and ls).

Equal = greater nor less.

When Gr,Lr are the outputs we got from the n-bit comparator, refer to line 151 in figure 2-2.

This block diagram will show how the comparator was done using this entity equation logic gates, followed by another image that shows how these equations were coded:



Figure 2-3 Stage 2 Block Diagram.

7

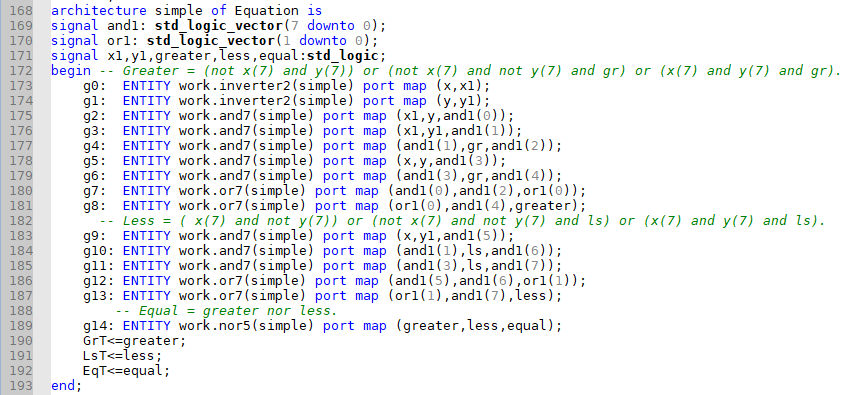


Figure 2-4 Equation entity.

Normal Simulation Results:

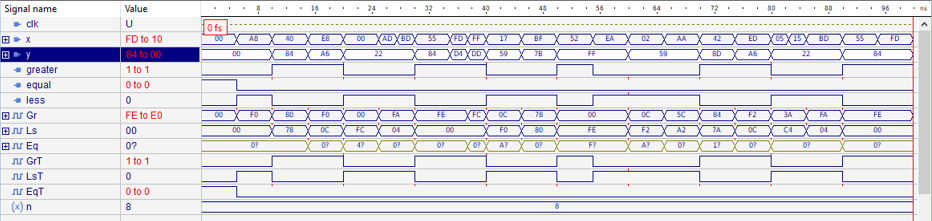


Figure 2-5 Stage2 Normal simulation.

After that I tried with the testbench including test generator and the result analyzer, the gates latency (delay) is 933 ns for all used gates combined, The clock delay I used is 400ns, Used the same generator in Stage 1, check figure 1-7 for the test generator code, Also same way was used to balance the expected output and the comparator output in order for them synchronize together.

A small delay caused form the D flip flops has occurred so I delayed the inputs for a bit so all of them can be generated and compared ( same way with stage 1).

The outputs were delayed by 3200ns as shown below:

8

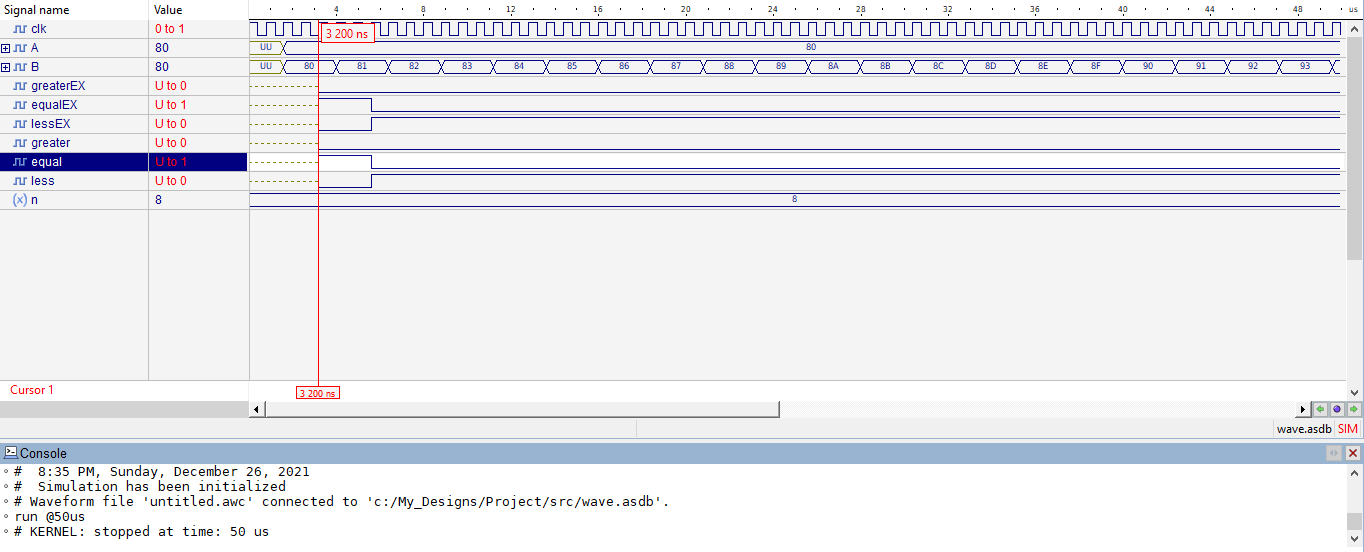


Figure 2-6 Stage 2 Testbench simulation part 1

All values were tested and no error has been shown:

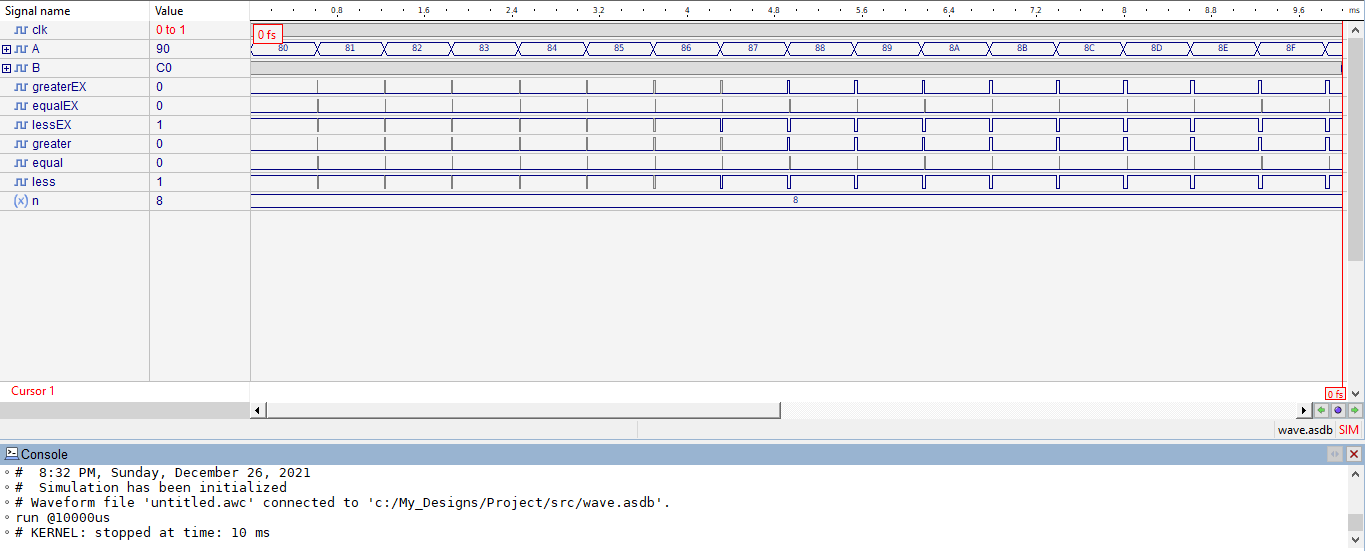


Figure 2-7 Stage 2 TestBench Simulation part 2.

9

Conclusion:

In this project we implemented a comparator that compares two 8-bit values in more than one way, first was using ripple subtractor and the other was using magnitude comparator, all with the use of the aldec active hdl program and coding in VHDL.

In stage 1, I turned a ripple adder into a subtractor using xor gates on the second input and cin initialized to one, then comparing all the sum bits together using or gates, if they all are zero then equal is set to one, the less value was a bit trickier, as the overflow obstacle caused me some problems and took me sometime to solve, but eventually I solved it by using xor gate between the overflow (carry xor carryPREV) and the last sum bit while making sure its not equal, if the answer is neither less or equal, then its greater.

Stage 2 was simpler, I used the magnitude comparator and the sign bit, the comparator was built from 8 one bit comparators that was built structurally, then I used the output produced from the comparator in some equations to produce the needed outputs, for the greater I made sure that if the sign bits were 0 and 1 the answer should be generated immediately, same thing goes for the less output if the sign bit is 1 and 0, if the sign bit is 0 and 0 or 1 and 1, then we take the output generated from the comparator, if the output is neither less nor greater, then its equal.

In Last, I Implemented the test bench and inside was the result analyzer and the test generator, In stage 1 there was an 1760 ns delay until the outputs started getting values, while in stage 2 it took 3200 ns which is a bit longer, I used the same clock for the D flip flops and the test generator, also the same test generator for both which synchronized the expected output with the comparator outputs.

10

References:

[1] <https://www.circuitstoday.com/ripple-carry-adder>

[2] <https://www.geeksforgeeks.org/magnitude-comparator-in-digital-logic/>

[3] <https://en.wikipedia.org/wiki/Sign_bit>

[4] <https://www.electronics-tutorials.ws/sequential/seq_4.html>

All codes were written using VHDL language, tested and simulated using ALDEC ACTIVE HDL.

11

APPENDIX: (CODE)

-- Ibrahim Nobani || 1190278 || section 2.

-- A simple implementation of all gates with their desired Delays.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity inverter2 is

port(x:in std\_logic; y:out std\_logic);

end;

architecture simple of inverter2 is

begin

y<= not x after 2 ns;

end;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity nand5 is

port(inNand,inNand2:in std\_logic; outNand:out std\_logic);

end;

architecture simple of nand5 is

begin

outNand<= inNand nand inNand2 after 5 ns;

end;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity nor5 is

port(inNor,inNor2:in std\_logic; outNor:out std\_logic);

end;

architecture simple of nor5 is

begin

outNor<= inNor nor inNor2 after 5 ns;

end;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity and7 is

port(inAnd,inAnd2:in std\_logic; outAnd:out std\_logic);

end;

architecture simple of and7 is

begin

outAnd<= inAnd and inAnd2 after 7 ns ;

end;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity or7 is

port(inOr,inOr2:in std\_logic; outOr:out std\_logic);

end;

architecture simple of or7 is

begin

outOr<= inOr or inOr2 after 7 ns ;

end;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity xnor9 is

port(inXNor,inXNor2:in std\_logic; outXNor:out std\_logic);

end;

architecture simple of xnor9 is

begin

outXNor<= inXNor xnor inXNor2 after 9 ns ;

end;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity xor12 is

port(inXor,inXor2:in std\_logic; outXor:out std\_logic);

end;

architecture simple of xor12 is

begin

outXor<= inXor xor inXor2 after 12 ns ;

end;

-- d flip flop, takes clock in and D, gives Q value of D when the clock rises.

Library Ieee;

use Ieee.std\_logic\_1164.all;

entity DFlipFlop is

port(clk:in std\_logic; D :in std\_logic; Q:out std\_logic);

end ;

architecture behave of DFlipFlop is

begin

process(clk)

begin

if(rising\_edge(clk)) then

Q <= D;

end if;

end process;

end;

--Stage 2:

-- one bit comparator using structural logic gates, 4 inputs (a,b to be compared, G,L to be used for comparison) and 3 outputs.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity onebitcomp is

port (a,b,G,L:in std\_logic; greater1, equal1, less1: out std\_logic);

end;

architecture structural of onebitcomp is

-- and signals to be used as temporary holders.

signal and1,and2: std\_logic\_vector(2 downto 0);

signal and3:std\_logic\_vector(4 downto 0);

signal or1: std\_logic\_vector(1 downto 0);

signal a1,b1,g1,l1,greater,less,equal:std\_logic;

begin -- greater = (a and not b) or (a and G) or (not b and G).

g0: ENTITY work.inverter2(simple) port map (a,a1);

gl: ENTITY work.inverter2(simple) port map (b,b1);

g2: ENTITY work.and7(simple) port map (a,b1,and1(0));

g3: ENTITY work.and7(simple) port map (a,G,and1(1));

g4: ENTITY work.and7(simple) port map (b1,G,and1(2));

g5: ENTITY work.or7(simple) port map (and1(0),and1(1),or1(0));

g6: ENTITY work.or7(simple) port map (or1(0),and1(2),greater1);

g7: ENTITY work.inverter2(simple) port map (G,g1);

g8: ENTITY work.inverter2(simple) port map (L,l1);

--less=(not a and b) or (not a and L) or (b and L).

g9: ENTITY work.and7(simple) port map (a1,b,and2(0));

g10: ENTITY work.and7(simple) port map (a1,L,and2(1));

g11: ENTITY work.and7(simple) port map (b,L,and2(2));

g12: ENTITY work.or7(simple) port map (and2(0),and2(1),or1(1));

g13: ENTITY work.or7(simple) port map (or1(1),and2(2),less1);

--equal1=(not a and not b and not G and not L) or ( a and b and not G and not L);

g14: ENTITY work.and7(simple) port map (a1,b1,and3(0));

g15: ENTITY work.and7(simple) port map (g1,l1,and3(1));

g16: ENTITY work.and7(simple) port map (and3(0),and3(1),and3(2));

g17: ENTITY work.and7(simple) port map (a,b,and3(3));

g18: ENTITY work.and7(simple) port map (and3(1),and3(3),and3(4));

g19: ENTITY work.or7(simple) port map (and2(2),and3(4),equal1);

end;

-- Comparator using magnitude comparator and sign bit, compares between two inputs a,b and stores output in one of the 3 outputs, and a clock for the d-flipflops.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_signed.ALL;

entity nbitcomp2 is

GENERIC ( n: POSITIVE:=8);

port (clk:in std\_logic;a,b: in std\_logic\_vector(n-1 downto 0);greater,equal,less:out std\_logic);

end;

architecture structure of nbitcomp2 is

signal x,y: std\_logic\_vector(n-1 downto 0);

signal Gr, Ls, Eq: std\_logic\_vector(n-1 downto 0);

signal GrT,LsT,EqT: std\_logic;

begin

gen0: for i in 0 to n-1 generate -- generating inputs from d flip flops.

gx: entity work.DFlipFlop(behave) port map (clk,a(i),x(i));

gy: entity work.DFlipFlop(behave) port map (clk,b(i),y(i));

end generate gen0;

Gr(0)<='0'; -- initializing the Gin and Lin to zero when they enter the comparator.

Ls(0)<='0';

gen1: for i in 0 to n-2 generate -- running the inputs in the one bit comparator.

g1:entity work.onebitcomp(structural) port map (x(i),y(i),Gr(i),Ls(i),Gr(i+1),Eq(i+1),Ls(i+1));

end generate gen1;

g: entity work.Equation(simple) port map (x(n-1),y(n-1),Gr(n-1),Ls(n-1),GrT,LsT,EqT);-- generate the equations from equation entity below.

--greater<=GrT;

--less<=LsT;

--equal<=EqT;

-- load all outputs into the d flip flops.

g1:entity work.DFlipFlop port map (clk,GrT,greater);

g2:entity work.DFlipFlop port map (clk,LsT,less);

g3:entity work.DFlipFlop port map (clk,EqT,equal);

end;

-- Equation entity that includes all the gates needed to generate the output from the n bit comparator

-- this entity takes 4 inputs needed to be used in the equation and produces the 3 last outputs to be put in D flip flops.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Equation is

port(x,y,gr,ls:in std\_logic;GrT,LsT,EqT:out std\_logic);

end;

architecture simple of Equation is

signal and1: std\_logic\_vector(7 downto 0);

signal or1: std\_logic\_vector(1 downto 0);

signal x1,y1,greater,less,equal:std\_logic;

begin -- Greater = (not x(7) and y(7)) or (not x(7) and not y(7) and gr) or (x(7) and y(7) and gr).

g0: ENTITY work.inverter2(simple) port map (x,x1);

g1: ENTITY work.inverter2(simple) port map (y,y1);

g2: ENTITY work.and7(simple) port map (x1,y,and1(0));

g3: ENTITY work.and7(simple) port map (x1,y1,and1(1));

g4: ENTITY work.and7(simple) port map (and1(1),gr,and1(2));

g5: ENTITY work.and7(simple) port map (x,y,and1(3));

g6: ENTITY work.and7(simple) port map (and1(3),gr,and1(4));

g7: ENTITY work.or7(simple) port map (and1(0),and1(2),or1(0));

g8: ENTITY work.or7(simple) port map (or1(0),and1(4),greater);

-- Less = ( x(7) and not y(7)) or (not x(7) and not y(7) and ls) or (x(7) and y(7) and ls).

g9: ENTITY work.and7(simple) port map (x,y1,and1(5));

g10: ENTITY work.and7(simple) port map (and1(1),ls,and1(6));

g11: ENTITY work.and7(simple) port map (and1(3),ls,and1(7));

g12: ENTITY work.or7(simple) port map (and1(5),and1(6),or1(1));

g13: ENTITY work.or7(simple) port map (or1(1),and1(7),less);

-- Equal = greater nor less.

g14: ENTITY work.nor5(simple) port map (greater,less,equal);

GrT<=greater;

LsT<=less;

EqT<=equal;

end;

-- STAGE 1:

-- one bit adder, three 1 bit inputs (x,y to be added and cin), 2 outputs (sum of x,y and cout).

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

ENTITY onebitadder IS

PORT ( x, y, cin: IN STD\_LOGIC; sum, cout: OUT STD\_LOGIC);

END;

-- one bit adder architecture using simple gates.

ARCHITECTURE simple OF onebitadder Is

signal and1:std\_logic\_vector(2 downto 0);

signal or1,or2,xor1: std\_logic;

BEGIN

-- sum = x xor y xor cin.

g0: entity work.xor12(simple) port map (x,y,xor1);

g1: entity work.xor12(simple) port map (xor1,cin,sum);

-- cout = (x and y) or (x and cin) or (y and cin).

g2: entity work.and7(simple) port map (x,y,and1(0));

g3: entity work.and7(simple) port map (x,cin,and1(1));

g4: entity work.and7(simple) port map (cin,y,and1(2));

g5: entity work.or7(simple) port map (and1(0),and1(1),or2);

g6: entity work.or7(simple) port map (or2,and1(2),cout);

END;

-- n bit subtractor, two n bit inputs (x,y to be added)and cin input which is 1 bit, 2 outputs (sum (n-bit) of x,y and cout(1-bit)),

-- and a carryPREV output to be used later in the overflow obsticle.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_signed.ALL;

ENTITY nbitadder IS

GENERIC ( n: POSITIVE:=8); -- Generic value initialized to 8.(8-bit).

PORT ( x, y: IN std\_logic\_vector(n-1 DOWNTO 0);

cin: IN STD\_LOGIC;

sum: OUT STD\_LOGIC\_VECTOR(n-1 DOWNTO 0);

cout,carryPREV: OUT STD\_LOGIC);

END;

-- structural iterative architecture that uses xor gates on the second 8 bit input in order to make the circuit a subtractor(y xor cin),

-- with cin initialized to 1 in order for the circuit to become a subtractor, followed by the use of the onebitadder.

ARCHITECTURE iterative OF nbitadder IS

signal y1: STD\_LOGIC\_VECTOR(n-1 DOWNTO 0);

signal carry: STD\_LOGIC\_vector(n downto 0);

begin

carry(0) <= cin;

cout <= carry(n);

carryPREV<=carry(n-1);

gen1: FOR i IN 0 TO n-1 GENERATE

g: entity work.xor12(simple) port map(y(i),cin,y1(i));

end generate gen1;

gen2: FOR i IN 0 TO n-1 GENERATE

g: entity work.onebitadder(simple) port map (x(i),y1(i),carry(i),sum(i),carry(i+1));

end generate gen2;

END ARCHITECTURE iterative;

--The comparator using ripple adder that takes 3 inputs (x,y to be compared and a clock for the d flip flop), and 3 outputs (greater,equal,less).

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_signed.ALL;

entity nbitadderCOMP is

GENERIC ( n: POSITIVE:=8);

port (clk:in std\_logic; x,y: in std\_logic\_vector(n-1 downto 0);greater,equal,less:out std\_logic);

end;

-- structural implementation of the comparator using the ripple adder/subtractor and simple gates.

architecture structural of nbitadderCOMP is

signal A,B,sum: std\_logic\_vector(n-1 downto 0);

signal eqT: std\_logic\_vector(n downto 0);

signal cin,carry,carryPREV: std\_logic;

signal eq: std\_logic\_vector(n-1 downto 0);

signal ov,lessT,ov2,greatT: std\_logic;

signal eqT2,xor1: std\_logic;

begin

gen0: for i in 0 to n-1 generate -- generating the input from the d flip flops.

gx: entity work.DFlipFlop(behave) port map (clk,x(i),A(i));

gy: entity work.DFlipFlop(behave) port map (clk,y(i),B(i));

end generate gen0;

cin<='1'; -- initial value of the cin is 1 in order for the ripple adder to turn into a subtractor.

eqT(0)<='0'; -- equal Temp that will go into the OR gates made for the equal value.

g: ENTITY work.nbitadder(iterative)

PORT MAP (A,B,cin,sum,carry,carryPREV);

eq<=sum; -- another equal Temp that takes the values of the sum to be used below in the or gate generator.

gen1: FOR i IN 0 TO n-1 GENERATE -- or gate generator that compares all the sum bits to see if they all are Zeros.

g: ENTITY work.or7(simple) port map (eqT(i),eq(i),eqT(i+1));

END GENERATE gen1;

g1: ENTITY work.inverter2(simple) port map (eqT(n),eqT2);-- equal temp 2 = value from or gates above.

--equal<=eqT2;

g2: ENTITY work.xor12(simple) port map (carry,carryPREV,ov);-- overflow = carry xor carryPREV

g3: entity work.xor12(simple) port map (ov,sum(n-1),xor1); -- overflow xor sum last bit

g4: entity work.and7(simple) port map (xor1,eqT(n),LessT); -- overflow xor sum(7) and not equal (eqT(n) = not equal).

--less<=lessT;

g5: entity work.nor5(simple) port map (lessT,eqT2,greatT); -- greater = less nor equal.

--greater<= greatT;

-- Load the outputs into D flip flops.

g6:entity work.DFlipFlop port map (clk,greatT,greater);

g7:entity work.DFlipFlop port map (clk,lessT,less);

g8:entity work.DFlipFlop port map (clk,eqT2,equal);

end;

-- STAGE 2 TESTBENCH.

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.STD\_LOGIC\_ARITH.ALL;

use ieee.std\_logic\_signed.ALL;

entity testBench is

GENERIC ( n: POSITIVE:=8);

end entity;

architecture test of testBench is

-- Signals to be used in the structural maps below.

signal clk : std\_logic := '1'; -- clock for edge rising.

signal A,B : std\_logic\_vector (n-1 downto 0);

signal greaterEX,equalEX,lessEX : std\_logic;

signal greater,equal,less : std\_logic;

begin

clk <= NOT clk AFTER 400 NS; -- all share the same clock, synchronization is closer this way.

g: entity work.test\_generator(tb) port map (clk,A,B,greaterEX,equalEX,lessEX); -- test generator structural.

g1: entity work.nbitcomp2(structure) port map (clk,A,B,greater,equal,less); -- n bit comp

g2: entity work.resultAnalyzer(tb) port map (clk,A,B,greaterEX,equalEX,lessEX,greater,equal,less); -- result analyzer

end;

-- STAGE 1 TESTBENCH

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.STD\_LOGIC\_ARITH.ALL;

use ieee.std\_logic\_signed.ALL;

entity testBench2 is

GENERIC ( n: POSITIVE:=8);

end entity;

architecture test of testBench2 is

-- Signals to be used in the structural maps below.

signal clk : std\_logic := '1';

signal A,B : std\_logic\_vector (n-1 downto 0);

signal greaterEX,equalEX,lessEX : std\_logic;

signal greater,equal,less : std\_logic;

begin

clk <= NOT clk AFTER 220 NS; -- all share the same clock, synchronization is closer this way.

g: entity work.test\_generator(tb) port map (clk,A,B,greaterEX,equalEX,lessEX);

g1: entity work.nbitaddercomp(structural) port map (clk,A,B,greater,equal,less);

g2: entity work.resultAnalyzer(tb) port map (clk,A,B,greaterEX,equalEX,lessEX,greater,equal,less);

end;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.math\_real.all;

use ieee.std\_logic\_signed.ALL;

-- test generator that produces all expected results (true results).

entity test\_generator is

GENERIC ( n: POSITIVE:=8);

PORT ( clock: in STD\_LOGIC; TestIn1: out STD\_LOGIC\_VECTOR(n-1 downto 0);

TestIn2: out STD\_LOGIC\_VECTOR(n-1 downto 0);

GreaterExpectedResult: out STD\_LOGIC;EqualExpectedResult: out STD\_LOGIC;LessExpectedResult: out STD\_LOGIC);

end;

architecture tb OF test\_generator IS

begin

process

begin

for i IN -128 TO 127 loop -- loop through all inputs

WAIT until rising\_edge(clock); -- one wait statment here

FOR j IN -128 TO 127 loop

WAIT until rising\_edge(clock); -- another wait

TestIn1 <= CONV\_STD\_LOGIC\_VECTOR(i,n); -- convert i so testIN1 can take the value.

TestIn2 <= CONV\_STD\_LOGIC\_VECTOR(j,n);

WAIT until rising\_edge(clock); -- two waits statments here in order for the expected result to synchronize with flip flops delays.

WAIT until rising\_edge(clock);

if (i > j) then -- doing the comparison process

GreaterExpectedResult <= '1';

EqualExpectedResult <= '0';

LessExpectedResult <= '0';

elsif(i=j)then

GreaterExpectedResult <= '0';

EqualExpectedResult <= '1';

LessExpectedResult <= '0';

elsif (i<j ) then

GreaterExpectedResult <= '0';

EqualExpectedResult <= '0';

LessExpectedResult <= '1';

-- WAIT until rising\_edge(clock);

end if;

END LOOP;

END LOOP;

WAIT;

END PROCESS;

END ARCHITECTURE tb;

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.STD\_LOGIC\_ARITH.ALL;

-- a result analyzer that takes the output from the test generater along with the ones from the implemented comparator, and compares if they are the same

-- if not, an error message is shown.

ENTITY resultAnalyzer IS

GENERIC ( n: POSITIVE:=8);

PORT ( clock: IN STD\_LOGIC; TestIn1: IN STD\_LOGIC\_VECTOR(n-1 DOWNTO 0); TestIn2: IN STD\_LOGIC\_VECTOR(n-1 DOWNTO 0);

GreaterExpectedResult: in STD\_LOGIC;EqualExpectedResult: in STD\_LOGIC;LessExpectedResult: in STD\_LOGIC;

ActualGreater: in STD\_LOGIC;ActualEqual: in STD\_LOGIC;ActualLess: in STD\_LOGIC);

end;

ARCHITECTURE tb OF resultAnalyzer IS

BEGIN

PROCESS(clock) -- if the clock changes (rising edge)

BEGIN

IF rising\_edge(clock) THEN

-- Check whether adder output matches expectation

ASSERT GreaterExpectedResult = ActualGreater

and LessExpectedResult = ActualLess and EqualExpectedResult= ActualEqual

REPORT "Something is wrong with the Results" -- error message

SEVERITY WARNING;

END IF;

END PROCESS;

END ARCHITECTURE tb;