Faculty of Engineering & Technology – Electrical & Computer Engineering Department

First Semester 2021 – 2022

ADVANCED DIGITAL SYSTEMS DESIGN

ENCS3310

COURSE PROJECT

Name: Ahmaide Al-Awawdah

ID: 1190823

Section: 1

Instructor: Dr. Abdeallatif Abuissa

Date: 13th – 25th December 2021

1. Abstract

This project is using VHDL language to construct a signed 8-bit comparator that compares between two signed 8-bit inputs, and it gives three outputs, F1 which will be given the value of 1 if both inputs are equal, F2 that will give 1 if the first input's value is larger, and F3 that will give 1 if the first input has a smaller value than the second one, and they all give a 0 in other cases.

There are two stages for this project, the first one is by using a signed adder for both inputs where it takes the 2's complement of the second input and sums it to the first one so basically it is a subtractor where it depends on whether the output is either a positive or negative or zero, the second stage is a magnitude comparator where it compares the two inputs by their magnitude and judges which has a larger value or if they are equal.

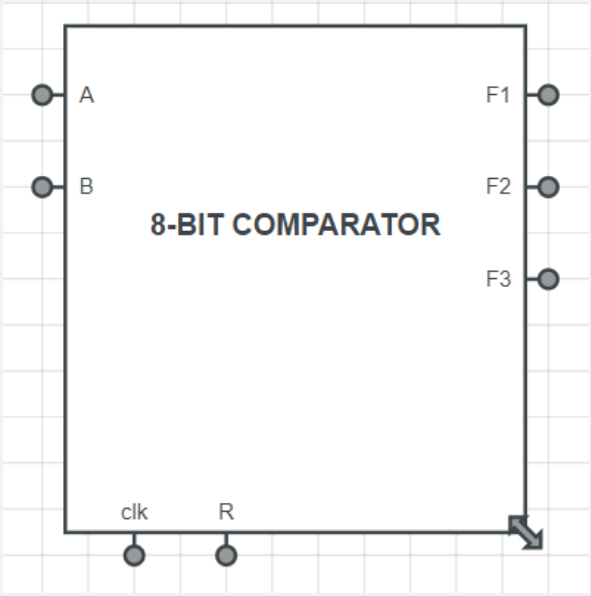

Figure 1: 8-bit comparator IC

This project will also include a testing part for each stage where all the possible inputs are testing by comparing them to the curtained output of the circuit as an error will be risen if the output was wrong.

All the drawn diagrams are done using proteus and https://www.circuit-diagram.org/
And the project is done using Aldec Active-HDL, the code can be found in the **Appendix.**

❖Table of Content

2.Theory

2.1. Stage 1: Comparator using ripple adder

In this stage the two signed inputs are compared by using a 9-bit signed adder that takes the 9-bit representation of the first input and the 9-bit representation of the 2's complements of the second input (so it is basically a subtractor), and judging by the 9-bit output the system can decide whether if the first input is greater than or equal to or smaller than the second input.

The reason why the adder converts both inputs into 9-bit representation is to not have overflow issues as if the adder is taking up to 8-bit signed inputs and the output of the two input was higher than 127 or less than -127 then an overflow will accrue, however in the 9-bit signed adder the overflow will only accrue if the output was either higher than 255 or less than -256 which is impossible considering that the adder inputs are between -128 and 127 as the output will be between -256 and 254 so there will not be any overflow issues and the 9th bit of the output will always have the right bit of the sign.

The less than output will be as the sign bit and it doesn't depend on any other bits where it is active when the sign bit is equal to 1 which means that the first input is less than the second input as the subtraction output is negative, both of the other two outputs (equal and greater than) need the sign bit of the subtraction to be zero, where the equal output needs all the bits of the subtractor output to be zero however the greater than output needs at least one the first seven bits to be equal to 1, as shown in figure 2.

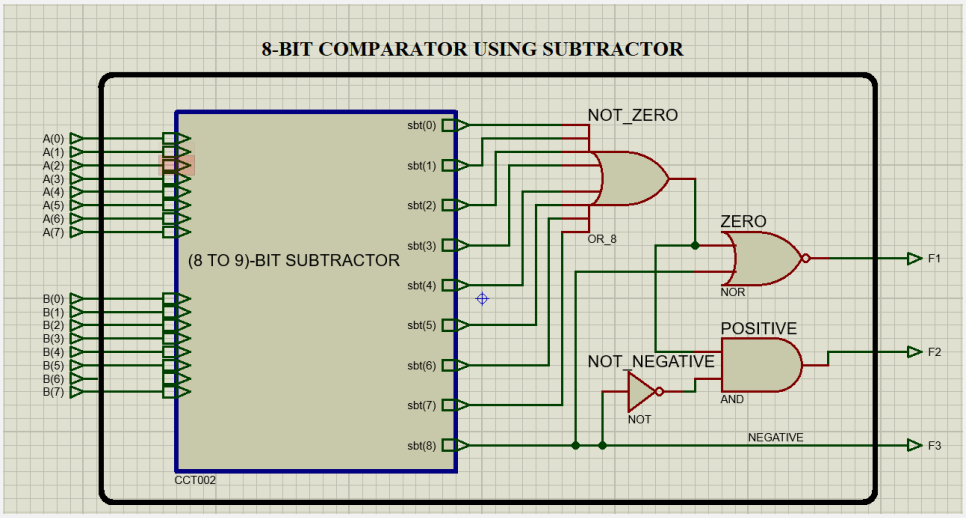

Figure 2: stage 1 comparator

2.2. Stage 2: Magnitude Comparator and sign bit

In this stage the sign bit and the other seven bits go into separate paths, the signs on the two inputs are compared to see weather if the both inputs have the same sign or if the first is positive and the second is negative or if the first is negative and the second is positive, the other seven bits are comparted using by a 1-bit comparator gate that compares each bit with the one on its level from the second input starting off from the highest to the lowest bit.

This 1-bit comparator depends on the previous comparator that compared between the previous bit as if a prior difference in the previous compared bits is found, in case it was found then it doesn't matter which of the current bits is higher because the higher bits have already found which of the inputs is larger by its first seven bits, so each 1-bit comparator works as a comparator if no difference is found and it works as a register if a previous difference is already found, figure 3 shows how this 1-bit comparator is made.

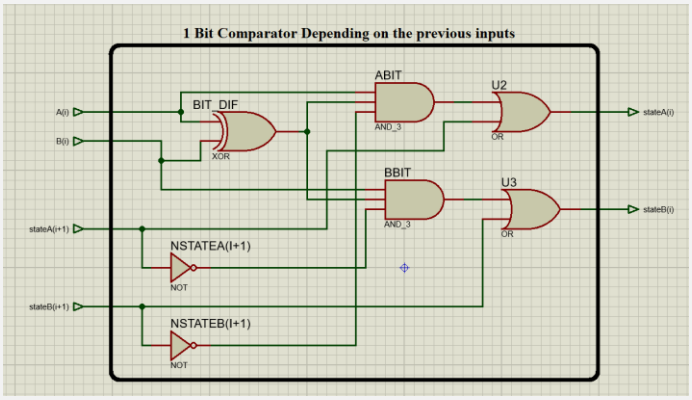

Figure 3: 1-bit comparator depending on previous componence

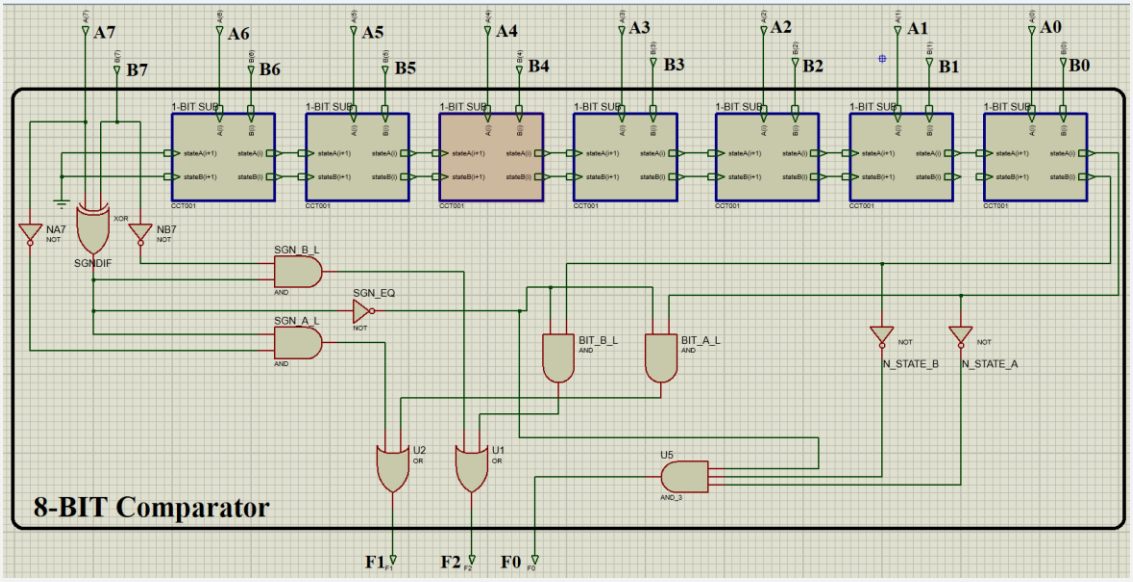

Figure 4: Stage 2 Comparator

3. Procedure

The code is in the **Appendix**.

3.1. Library of Gates and Flip-Flops

The table below shows all the used logical gates and flip-flops that will be used in this project.

Gate Entity Name	TYPE OF THE GATE
inverter9	9-bit inverter
inverter	1-bit inverter
NOR9	NOR gate for 9 1-bit inputs
AND2	AND gate for 2 1-bit inputs
AND3	AND gate for 3 1-bit inputs
OR2	OR gate for 2 1-bit inputs
OR8	OR gate for 8 1-bit inputs
XOR2	XOR gate for 2 1-bit inputs
XOR8	XOR gate for 2 8-bit inputs
DFF	D flip-flop with 3 inputs and a clock a reset input

Table 2

3.2. Stage 1: Comparator using Adder

In this stage the comparator that has two 8-bit inputs (A, B) and three 1-bit outputs (F1 for when A=B, F2 for when A>B, and F3 for when A<B) is made out of an 9-bit adder as it makes a subtractor by inverting the second input and setting the Cin of the adder to 1 in order to take the 2's complement of the second input B so now the adder is a subtractor that gets the subtraction of B from A in 9-bit representation.

The 9-bit adder is made out of nine full adders and takes two 9-bit inputs and 1-bit cin input. This adder uses the full adder entity in a loop that goes on all the bits of the inputs where each i^{th} bit from A is summed with the i^{th} bit from B and with the $(i-1)^{th}$ cout as i^{th} cin as it gives a 9-bit summation with a carry out bit.

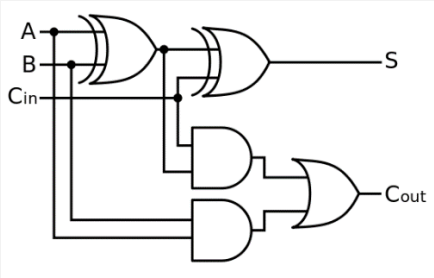

Figure 5: Full Adder Diagram

```
--                FULL ADDER
-- ------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ADDER IS
    PORT (a, b, cin: IN std_logic;
    sum, cout: OUT std_logic);
END;

ARCHITECTURE simple OF ADDER IS
SIGNAL s1, s2, s3: std_logic;
BEGIN
    g1: ENTITY work.XOR2(XOR_2) PORT MAP (a,b,s1);
    g2: ENTITY work.XOR2(XOR_2) PORT MAP (s1,cin,sum);
    g3: ENTITY work.AND2(AND_2) PORT MAP (s1,cin,s2);
    g4: ENTITY work.AND2(AND_2) PORT MAP (a, b,s3);
    g5: ENTITY work.OR2(OR_2) PORT MAP (s2, s3, cout);
END;
-------------------------------------------------------------
```

Figure 6: Full Adder Code

```
--                9-BIT ADDER
-- ------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY Bit9Adder IS
    PORT (a, b: IN std_logic_vector(8 downto 0);
    cin: IN std_logic;
    s: OUT std_logic_vector(8 downto 0);
    cout: OUT std_logic);
END;

ARCHITECTURE structAdd OF Bit9Adder IS
SIGNAL carrySignal: STD_logic_vector( 9 downto 0);
CONSTANT n: INTEGER := 9;
SIGNAL sig: STD_logic_vector( 8 downto 0);
BEGIN
    carrySignal(0) <= cin;
    gen0:
    FOR i IN 0 TO n-1 GENERATE -- Loop to go on all the Bits of both inputs
        g1: ENTITY work.ADDER(simple) PORT MAP (a(i), b(i), carrySignal(i), sig(i), carrySignal(i+1));
    END GENERATE gen0;
    s <= sig;
    cout <= carrySignal(9);
END;
-------------------------------------------------------------------------------------------------------
```

Figure 7: 9-BIT Adder Code

The subtractor takes two 8-bit inputs and converts them into 9 bit representation since they can be represented in signed 8-bit so definitely the ninth bit is the same as the eight as its zero when the input is positive and one if it is negative, so after converting the two inputs into 9-bit representation, then it inverts the second input (B) and then puts the 9-bit representation of A and the inverted 9-bit representation of B with a cin that equals to 1 so that the output of the subtractor can be (A - B) in 9-bit representation with on overflow as its been explained why it converts to 9-bit in the theory, with a cout that wont be needed in the comparator.

Since the two inputs can be represented in signed 8-bit so the maximum difference between the two inputs can be (127 - -128) which is equal to 255 which can be represented in signed 9-bit with the 9th bit has the right sign into it.

```
--               A subtractor that takes 8-bit inputs and gives a 9-bit subtraction output
-- ============================================================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY Bit8Subtractor IS
    PORT(a, b: IN std_logic_vector(7 downto 0);
    sub: OUT std_logic_vector(8 downto 0));
END;

ARCHITECTURE structSub OF Bit8Subtractor IS
SIGNAL onesComp: std_logic_vector( 8 downto 0);-- B Inverted
SIGNAL nun: std_logic;
SIGNAL extraA, extraB: std_logic; -- A and B in 9-bit representation
SIGNAL a9, b9: std_logic_vector( 8 downto 0); -- 9th Bit of A and B
BEGIN
    extraA <= a(7);
    extraB <= b(7);
    a9 <= extraA&a;
    b9 <= extraB&b;
    g0: ENTITY work.inverter9(invert9) PORT MAP (b9, onesComp);
    g1: ENTITY work.Bit9Adder(structAdd) PORT MAP (a9, onesComp, '1', sub, nun);
END;
--------------------------------------------------------------------------------
```

Figure 8: The Subtractor Code

The Comparator takes these two 8-bit signed inputs and send them into the subtractor where it can give a signed 9-bit subtraction, this subtraction decides which of the two inputs is greater or if they are equal.

The first output (F1 as both inputs are equal) needs all nine bits of the subtraction to be zero, so it can be done by a 9-input NOR gate that only turns to 1 if all the 9-bits are zero, The second output (F1 as in A>B) means that the output of the subtraction is positive (sign equals zero) and it also needs to check that the magnitude is not also zero so it takes the sign of the subtraction (9th bit) to an inverter not gate that gives 1 if the sign bit is 0 and it checks that the 8 other bits have at least one of them equals to 1 to make sure that the output is a positive number and not zero so it takes these first 8 bit into an 8-input OR gate and to make sure that both the output of this OR gate and the not of the 9th bit are equal to 1, they are connected to an AND gate that determines the value of F1, for the third and final output (F2 as in A<B) only needs to check if the subtraction output is negative so it can basically be connected to the 9th output pin if the subtractor and be determined by it.

These outputs should only get their value by a clock that can rise and fall and they need a reset that resets them to zeros, for that all three outputs are connected to a 3 input/output D flip-flop that gives out the final output only when its clock has a rising edge, and all three outputs are zeros when reset is active.

```
--               STAGE 1: STAGE 1 USING RIPPLE ADDER
-- ================================================================================================
ARCHITECTURE adderComparator OF comparator IS
SIGNAL sgnPos, notZero, E1, E2, E3: std_logic;
SIGNAL sbt: std_logic_vector( 8 downto 0 );
BEGIN
    g0: ENTITY work.Bit8Subtractor(structSub) PORT MAP (a, b, sbt);
    g1: ENTITY  work.OR8(OR_8) PORT MAP (sbt(0), sbt(1), sbt(2), sbt(3), sbt(4), sbt(5), sbt(6), sbt(7), notZero);
    g2: ENTITY work.inverter(invert) PORT MAP (sbt(8), sgnPos);
    g3: ENTITY work.NOR9(NOR_9) PORT MAP (sbt(0), sbt(1), sbt(2), sbt(3), sbt(4), sbt(5), sbt(6), sbt(7), sbt(8), E1);
    g4: ENTITY work.AND2(AND_2) PORT MAP (notZero, sgnPos, E2);
    E3 <= sbt(8);
    g5: ENTITY work.DFF(DFF3) PORT MAP (E1, E2, E3, clk, reset, F1, F2, F3);
END;
--------------------------------------------------------------------------------
```

Figure 9: Stage 1 Comparator Code

3.3. Stage 2: Magnitude Comparator and Sign Bit

In this stage the comparator that has two 8-bit signed inputs (A, B) and three outputs (F1, F2, F3) compares between the magnitude of the two input (bit 0 to bit 7) and gives which input has a larger magnitude, and it also compares between the signs of the two inputs (8^{th} bit).

For the sign bit, there are three cases which can be determined by an XOR gate between the two sign bits , the first case is that the both inputs have the same sign (same 8^{th} bit) either positive or negative by having a not after the XOR gate, the second case is that A is positive and B is negative by having an AND between the NOT of the sign of A and the XOR gate, and the third case is that A is negative and B is positive by having an AND between the NOT of the sign of B and the XOR gate.

The magnitude comparator uses a structural 1-bit comparator between each bit of the two inputs and it goes from the highest to the lowest bit searching for a difference, if there is a difference found in any of the bits the greater one will have its 1-bit comparator give output of 1 for that input and it will keep going on the rest of the 1-bit comparators as if it is a normal shifting register where it will give 0 before the difference is found and it will also give a zero shifting for the smaller other input.

The 1-bit comparator depends on previous higher bits compare, and as shown in the figures below that comparator takes the i^{th} bit from A and B and the output of the comparator for the $(i+1)^{th}$ bit so if any of the previous outputs is 1 then the state will take the same value of the previous state, however if no difference is found in the previous compared bits then the comparator will work as the state of the input will work on two conditions other than the previous state for the other input is 0, and the are that the current bit for this input is 1 and that there is difference between it and the bit for the other input, so this will need an XOR gate between the two bits and a 3 input AND gate for each one that takes its bit value and the XOR between both of them and the NOT of the previous state for the other input so that no previous difference is found, and the output of each AND gate will go to an OR gate with the previous state for each one so it can take the 1 value if it is already found, and that happens for each and every bit in all 7 magnitude bits and this comparator is shown in figures 10, 11.

```
255  --                One bit comparator that depends on previous higher bits
256  -- ============================================================================
257  Library ieee;
258  USE ieee.std_logic_1164.all;
259  USE ieee.std_logic_unsigned.all;
260
261  ENTITY BitCompare IS
262      PORT (a, b, pstateA, pstateB: IN std_logic;
263      stateA, stateB: OUT std_logic);
264  END;
265
266  ARCHITECTURE chip OF BitCompare IS
267  SIGNAL bitDif, Abit, Bbit, As, Bs, nstateA, nstateB: std_logic;
268  BEGIN
269      g1: ENTITY work.inverter(invert) PORT MAP (pstateA, nstateA);
270      g2: ENTITY work.inverter(invert) PORT MAP (pstateB, nstateB);
271      g3: ENTITY work.XOR2(XOR_2) PORT MAP (a, b, bitDif);
272      g4: ENTITY work.AND3(AND_3) PORT MAP (a, bitDif, nstateB, Abit);
273      g5: ENTITY work.AND3(AND_3) PORT MAP (b, bitDif, nstateA, Bbit);
274      g6: ENTITY work.OR2(OR_2) PORT MAP (pstateA, Abit, As);
275      g7: ENTITY work.OR2(OR_2) PORT MAP (pstateB, Bbit, Bs);
276      stateA <= As;
277      stateB <= Bs;
278  END;
279  ------------------------------------------------------------------------------
```

Figure 10: 1-Bit Comparator code

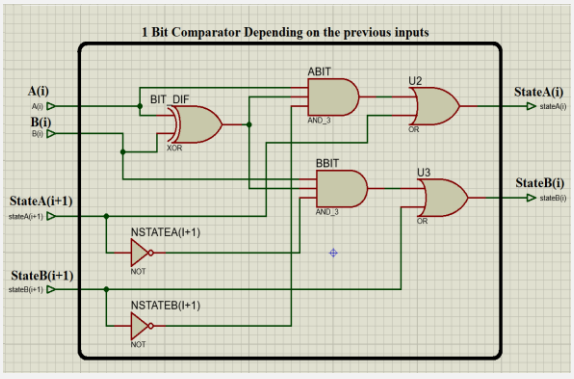

Figure 11: 1-Bit Comparator Diagram

When all magnitude bits are compared down to the lest significant bit the outputs of the last 1-bit comparator gives which input has a greater magnitude or if they are both the same so if state A(0) is equal to 1 that means that the magnitude of A is larger and if the state B(0) is equal to 1 then the magnitude of B is larger, and if both states of(0) are 0 then the magnitude is the same yet it's impossible for both states to equal 1 at once.

For the three outputs in this stage (F0, F1, F2) they can go as this: F0 needs both the sign and the magnitude for both inputs to be the same which means when the signs are equal and the state for both A(0) and B(0) equals 0 so that can be done using a three input and gate for the first case of the sign compare and the NOT of state A(0) and the NOT of state B(0). F1 can go on in the second case of the sign compare (A is positive and B is negative) or the sign can be the same but the magnitude of A is larger, so it will need an AND gate between the equality of the signs and the state A(0) and the output of that AND gate goes to an OR gate with the second case of the sign bit. F3 can go as the opposite of the F2 where there there is an OR gate that takes the third case of the sign compare (A is negative and B is positive) and the output of an AND gate that takes the equality of the sign bits and the state B(0).

And such as stage 1 these outputs should get their value the clock reaches a rising edge and should be connected to a reset input that resets their values to zeros and that can be done by a 3 input/output D flip-flop that takes the outputs and only displays them when the clock reaches a rising edge, the code is displayed in figure 12.

```
295  --                STAGE 2: USING MAGNITUDE COMPARATOR AND SIGN BIT
296  -- ================================================================================================
297  ARCHITECTURE magnitudeComparator OF comparator IS
298  CONSTANT n: INTEGER := 8;
299  SIGNAL sgnDif, nA7, nB7, sgnAl, sgnBl, sgnEQ, nstateA, nstateB, bitAl, bitBl, E1, E2, E3: std_logic;
300  SIGNAL stateA, stateB: std_logic_vector(7 downto 0) ;
301  BEGIN
302      stateA(7) <= '0';
303      stateB(7) <= '0';
304      g0: ENTITY work.inverter(invert) PORT MAP (a(n-1), nA7);
305      g1: ENTITY work.inverter(invert) PORT MAP (b(n-1), nB7);
306      g2: ENTITY work.XOR2(XOR_2) PORT MAP (a(n-1), b(n-1), sgnDif);
307      g3: ENTITY work.AND2(AND_2) PORT MAP (nA7, sgnDif, sgnAl);
308      g4: ENTITY work.AND2(AND_2) PORT MAP (nB7, sgnDif, sgnBl);
309      g5: ENTITY work.inverter(invert) PORT MAP (sgnDif, sgnEQ);
310      gen: FOR i IN 2 TO n GENERATE
311           g6: ENTITY work.BitCompare(chip)
312                PORT MAP (a(n-i), b(n-i), stateA(n+1-i), stateB(n+1-i),
313                stateA(n-i), stateB(n-i));
314      END GENERATE gen;
315      g7: ENTITY work.inverter(invert) PORT MAP (stateA(0), nstateA);
316      g8: ENTITY work.inverter(invert) PORT MAP (stateB(0), nstateB);
317      g9: ENTITY work.AND2(AND_2) PORT MAP (sgnEQ, stateA(0), bitAl);
318      g10: ENTITY work.AND2(AND_2) PORT MAP (sgnEQ, stateB(0), bitBl);
319      g11: ENTITY work.AND3(AND_3) PORT MAP (nstateA, nstateB, sgnEQ, E1);
320      g12: ENTITY work.OR2(OR_2) PORT MAP (bitAl, sgnAl, E2);
321      g13: ENTITY work.OR2(OR_2) PORT MAP (bitBl, sgnBl, E3);
322      g14: ENTITY work.DFF(DFF3) PORT MAP (E1, E2, E3, clk, reset, F1, F2, F3);
323  END;
324  ----------------------------------------------------------------------------------------
```

Figure 12: Stage 2 comparator code

3.4. Testing

Testing any circuit need a test generator that will go on all the possible input values which are (-128 to 127) for the signed 8-bit values in this project and see for each case the exact true output that is which one of the inputs is greater or if they are both equal.

Then there is the result analyzer that gets two inputs from the same type and the same number of bits and compared between them and rises an error “The output is incorrect” if both inputs aren’t the same.

To test the two comparator a testbench is made for each comparator that takes all the possible inputs from the generators with the expected true values of each output and sends these inputs to the comparator and then the expected true result from the generator and the output of the built comparator are sent to the analyzer that checks if they are the same so it can rise an error if they weren’t, the clock of the generator and analyzer is 2 times the one for the Comparator, so that the analyzer checks when the comparator already got its values when its clock is risen.

```
--                RESULT ANALYSER
-- ================================================================================
Library IEEE;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY analyser IS
    PORT ( expF1, expF2, expF3, actF1, actF2, actF3 : IN std_logic;
    clock: IN std_logic);
END;

ARCHITECTURE resultAnalyser OF analyser IS
BEGIN
    PROCESS(clock)
    BEGIN
        IF rising_edge(clock) THEN
            ASSERT (expF1 = actF1 AND expF2 = actF2 AND expF3 = actF3)
            REPORT "The output is incorrect"
            SEVERITY ERROR;
        END IF;
    END PROCESS;
END;
--------------------------------------------------------------------------------
```

Figure 13: Result Analyzer Code

The best clock for the stage 1 is 165ns and the best clock for stage 2 is 92ns

```
--                TEST GENERATOR
-- ================================================================================
Library IEEE;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY generator IS
    PORT (clock : IN std_logic;
    testA, testB: OUT std_logic_vector(7 downto 0);
    expF1, expF2, expF3: OUT std_logic);
END;

ARCHITECTURE tester OF generator IS
BEGIN
    PROCESS
    BEGIN
        FOR i IN -128 TO 127 LOOP
            FOR j IN -128 TO 127 LOOP
                testA <= CONV_STD_LOGIC_VECTOR(i, 8);
                testB <= CONV_STD_LOGIC_VECTOR(j, 8);
                IF ( i > j ) THEN
                    expF1 <= '0';
                    expF2 <= '1';
                    expF3 <= '0';
                ELSIF ( j > i ) THEN
                    expF1 <= '0';
                    expF2 <= '0';
                    expF3 <= '1';
                ELSE
                    expF1 <= '1';
                    expF2 <= '0';
                    expF3 <= '0';
                END IF;
                WAIT UNTIL rising_edge(clock);
            END LOOP;
        END LOOP;
        WAIT;
    END PROCESS;
END;
--------------------------------------------------------------------------------
```

. Figure 14: Test Generator Code

```
--                TESTING FOR BOTH STAGES
-- ================================================================================
-- ================================================================================
Library IEEE;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY testBinch IS
END;

--                TESTING FOR STAGE 1
-- ================================================================================
ARCHITECTURE adderCompTest OF testBinch IS
SIGNAL clock, clock2, expF1, expF2, expF3, actF1, actF2, actF3: std_logic:='0';
SIGNAL A, B: std_logic_vector( 7 downto 0 ):="00000000";
BEGIN
    clock <= not clock after 165 ns;
    clock2 <= not clock2 after 330 ns;
    g1: ENTITY work.generator(tester) PORT MAP (clock2, A, B, expF1, expF2, expF3);
    g2: ENTITY work.comparator(adderComparator) PORT MAP (A, B, clock, '0', actF1, actF2, actF3);
    g3: ENTITY work.analyser(resultAnalyser) PORT MAP (expF1, expF2, expF3, actF1, actF2, actF3, clock2);
END;
--------------------------------------------------------------------------------

--                TESTING FOR STAGE 2
-- ================================================================================
ARCHITECTURE magCompTest OF testBinch IS
SIGNAL clock, clock2, expF1, expF2, expF3, actF1, actF2, actF3: std_logic:='0';
SIGNAL A, B: std_logic_vector( 7 downto 0 ):="00000000";
BEGIN
    clock <= not clock after 92 ns;
    clock2 <= not clock2 after 184 ns;
    g1: ENTITY work.generator(tester) PORT MAP (clock2, A, B, expF1, expF2, expF3);
    g2: ENTITY work.comparator(magnitudeComparator) PORT MAP (A, B, clock, '0', actF1, actF2, actF3);
    g3: ENTITY work.analyser(resultAnalyser) PORT MAP (expF1, expF2, expF3, actF1, actF2, actF3, clock2);
END;
--------------------------------------------------------------------------------
```

Figure 15: Testbench Code

Rising an error

For stage 1 in order to rise an error the AND gate in the entity g4 in line 336 is replaced with an OR gate as shown in Figure 16.

```
326 --                STAGE 1: STAGE 1 USING RIPPLE ADDER
327 -- ============================================================================================
328 ARCHITECTURE adderComparator OF comparator IS
329 SIGNAL sgnPos, notZero, E1, E2, E3: std_logic;
330 SIGNAL sbt: std_logic_vector( 8 downto 0 );
331 BEGIN
332     g0: ENTITY work.Bit8Subtractor(structSub) PORT MAP (a, b, sbt);
333     g1: ENTITY  work.OR8(OR_8) PORT MAP (sbt(0), sbt(1), sbt(2), sbt(3), sbt(4), sbt(5), sbt(6), sbt(7), notZero);
334     g2: ENTITY work.inverter(invert) PORT MAP (sbt(8), sgnPos);
335     g3: ENTITY work.NOR9(NOR_9) PORT MAP (sbt(0), sbt(1), sbt(2), sbt(3), sbt(4), sbt(5), sbt(6), sbt(7), sbt(8), E1);
336     g4: ENTITY work.OR2(OR_2) PORT MAP (notZero, sgnPos, E2);
337     E3 <= sbt(8);
338     g5: ENTITY work.DFF(DFF3) PORT MAP (E1, E2, E3, clk, reset, F1, F2, F3);
339 END;
340 ---------------------------------------------------------------------------------------------
```

Figure 16: Error Spot in Stage 1

Now the simulation will rise an error as F1 wont work properly as seen in Figure 17.

```
Console
◦ # EXECUTION:: Time: 42914190 ns,  Iteration: 0,  Instance: /testBinch/g3,  Process:
  line__399.
◦ # EXECUTION:: ERROR  : The output is incorrect
◦ # EXECUTION:: Time: 42914850 ns,  Iteration: 0,  Instance: /testBinch/g3,  Process:
  line__399.
◦ # EXECUTION:: ERROR  : The output is incorrect
◦ # EXECUTION:: Time: 43083810 ns,  Iteration: 0,  Instance: /testBinch/g3,  Process:
  line__399.
◦ # EXECUTION:: ERROR  : The output is incorrect
◦ # EXECUTION:: Time: 43084470 ns,  Iteration: 0,  Instance: /testBinch/g3,  Process:
  line__399.
◦ # EXECUTION:: ERROR  : The output is incorrect
◦ # EXECUTION:: Time: 43253430 ns,  Iteration: 0,  Instance: /testBinch/g3,  Process:
  line__399.
>
```

Figure 17: Stage 1 Error in Console

For stage 2 in order to rise an error the OR gate in the entity g13 in line 321 is replaced with an AND gate as shown in Figure 18, and then the simulation will rise an error as F2 wont work properly as seen in Figure 19.

```
295 --              STAGE 2: USING MAGNITUDE COMPARATOR AND SIGN BIT
296 -- ============================================================================================
297 ARCHITECTURE magnitudeComparator OF comparator IS
298 CONSTANT n: INTEGER := 8;
299 SIGNAL sgnDif, nA7, nB7, sgnAl, sgnBl, sgnEQ, nstateA, nstateB, bitAl, bitBl, E1, E2, E3: std_logic;
300 SIGNAL stateA, stateB: std_logic_vector(7 downto 0) ;
301 BEGIN
302     stateA(7) <= '0';
303     stateB(7) <= '0';
304     g0: ENTITY work.inverter(invert) PORT MAP (a(n-1), nA7);
305     g1: ENTITY work.inverter(invert) PORT MAP (b(n-1), nB7);
306     g2: ENTITY work.XOR2(XOR_2) PORT MAP (a(n-1), b(n-1), sgnDif);
307     g3: ENTITY work.AND2(AND_2) PORT MAP (nA7, sgnDif, sgnAl);
308     g4: ENTITY work.AND2(AND_2) PORT MAP (nB7, sgnDif, sgnBl);
309     g5: ENTITY work.inverter(invert) PORT MAP (sgnDif, sgnEQ);
310     gen: FOR i IN 2 TO n GENERATE
311         g6: ENTITY work.BitCompare(chip)
312             PORT MAP (a(n-i), b(n-i), stateA(n+1-i), stateB(n+1-i),
313             stateA(n-i), stateB(n-i));
314     END GENERATE gen;
315     g7: ENTITY work.inverter(invert) PORT MAP (stateA(0), nstateA);
316     g8: ENTITY work.inverter(invert) PORT MAP (stateB(0), nstateB);
317     g9: ENTITY work.AND2(AND_2) PORT MAP (sgnEQ, stateA(0), bitAl);
318     g10: ENTITY work.AND2(AND_2) PORT MAP (sgnEQ, stateB(0), bitBl);
319     g11: ENTITY work.AND3(AND_3) PORT MAP (nstateA, nstateB, sgnEQ, E1);
320     g12: ENTITY work.OR2(OR_2) PORT MAP (bitAl, sgnAl, E2);
321     g13: ENTITY work.AND2(AND_2) PORT MAP (bitBl, sgnBl, E3);
322     g14: ENTITY work.DFF(DFF3) PORT MAP (E1, E2, E3, clk, reset, F1, F2, F3);
323 END;
324 ---------------------------------------------------------------------------
```

Figure 18: Error Spot in Stage 2

```
Console
◦ # EXECUTION:: ERROR  : The output is incorrect
◦ # EXECUTION:: Time: 23644552 ns,  Iteration: 0,  Instance:
  /testBinch/g3,  Process: line__399.
◦ # EXECUTION:: ERROR  : The output is incorrect
◦ # EXECUTION:: Time: 23739128 ns,  Iteration: 0,  Instance:
  /testBinch/g3,  Process: line__399.
◦ # EXECUTION:: ERROR  : The output is incorrect
◦ # EXECUTION:: Time: 23833704 ns,  Iteration: 0,  Instance:
  /testBinch/g3,  Process: line__399.
◦ # EXECUTION:: ERROR  : The output is incorrect
◦ # EXECUTION:: Time: 23928280 ns,  Iteration: 0,  Instance:
```

Figure 19: Stage 2 Error in Console

3.5. Simulation Results

The figure below shows 8 different clips from the simulation of the testbench for Stage 1.

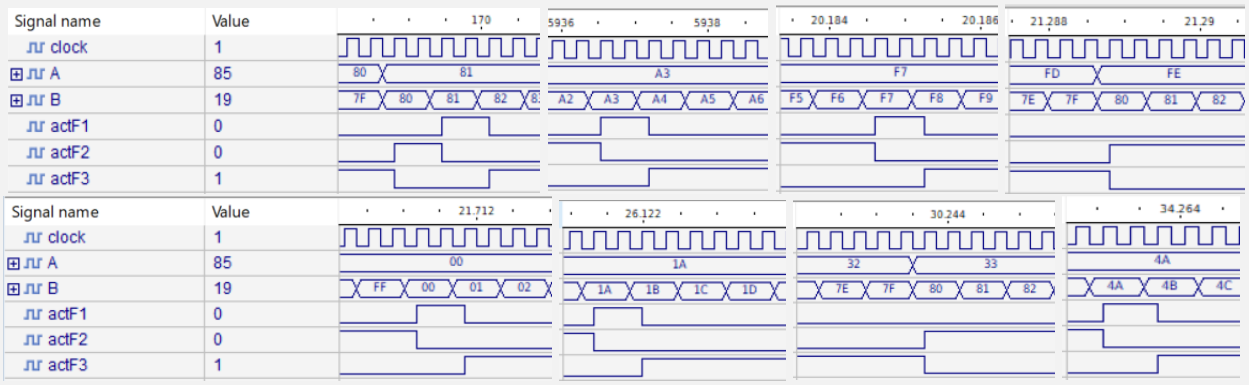

Figure 20: Stage 1 Simulation

And also, the figure below shows 6 different clips from the simulation of the test bench is stage 2.

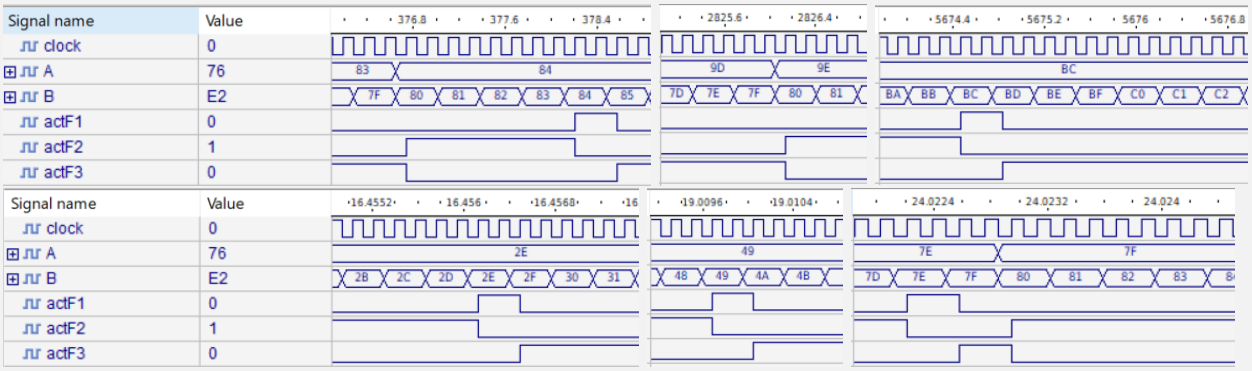

Figure 21: Stage 2 Simulation

4. Conclusion and Future Work

In conclusion we learned how to structurally make a comparator that compares between signed inputs and how to check and verify if our work is done right.

The two stages showed us that there is always more than a way to do something, yet each way has it its own advantages and disadvantages, in time or the number of needed gates for it, and some can be considered better than others judging on their results.

This project helped and developed our skills in VHDL Coding and into making more complex circuits with more gates and jobs in future time, as it also can give out thoughts about good ideas too.

5. References

- https://www.ece.uvic.ca/~fayez/courses/ceng465/lab_465/project1/adders.pdf

  On December 18th 2021 9:55 PM

- https://www.watelectronics.com/what-is-full-adder-circuit-using-basic-gates/

  On December 25th 11:35 PM

6. Appendix

```
-- Ahmaide Al-Awawdah
-- 1190823



--              Inverters
-- ============================================================================================================
Library ieee;
USE ieee.std_logic_1164.all;

ENTITY inverter9 IS
            PORT (x : IN std_logic_vector(8 downto 0);
            y: OUT std_logic_vector(8 downto 0));
            END;

ARCHITECTURE invert9 OF inverter9 IS
BEGIN
            y <= NOT x AFTER 2 ns;
END;
-------------------------------------------------

Library ieee;
USE ieee.std_logic_1164.all;

ENTITY inverter IS
            PORT (x : IN std_logic;
            y: OUT std_logic);
            END;

ARCHITECTURE invert OF inverter IS
BEGIN
```

```
        y <= NOT x AFTER 2 ns;
END;
-----------------------------------------------


--          NOR Gates
-- ============================================================================================================
Library ieee;
USE ieee.std_logic_1164.all;

ENTITY NOR9 IS
        PORT (x, y, w, v, u, t, s, r, q: IN std_logic;
        z: OUT std_logic);
        END;

ARCHITECTURE NOR_9 OF NOR9 IS
BEGIN
        z <= NOT (x OR y OR w OR v OR u OR t OR s OR r OR q) AFTER 5 ns;
END;
-----------------------------------------------------------------


--          AND GATES
-- ============================================================================================================
Library ieee;
USE ieee.std_logic_1164.all;

ENTITY AND2 IS
        PORT (x, y : IN std_logic;
        z: OUT std_logic);
        END;

ARCHITECTURE AND_2 OF AND2 IS
```

```
BEGIN
        z <= x AND y AFTER 7 ns;
END;
-----------------------------------------

Library ieee;
USE ieee.std_logic_1164.all;

ENTITY AND3 IS
        PORT (x, y, w : IN std_logic;
        z: OUT std_logic);
        END;

ARCHITECTURE AND_3 OF AND3 IS
BEGIN
        z <= x AND y AND w AFTER 7 ns;
END;
-----------------------------------------


--          OR GATES
-- ============================================================================================================
Library ieee;
USE ieee.std_logic_1164.all;

ENTITY OR2 IS
        PORT (x ,y : IN std_logic;
        z: OUT std_logic);
        END;

ARCHITECTURE OR_2 OF OR2 IS
BEGIN
        z <= x OR y AFTER 7 ns;
```

```
END;
-------------------------------------------------------------


Library ieee;
USE ieee.std_logic_1164.all;

ENTITY OR8 IS
        PORT (x, y, w, v, u, t, s, r : IN std_logic;
        z: OUT std_logic);
        END;

ARCHITECTURE OR_8 OF OR8 IS
BEGIN
        z <= (x OR y OR w OR v OR u OR t OR s OR r) AFTER 7 ns;
END;
-------------------------------------------------------------



--          XOR GATES
-- ============================================================================================================
Library ieee;
USE ieee.std_logic_1164.all;

ENTITY XOR2 IS
        PORT (x ,y : IN std_logic;
        z: OUT std_logic);
        END;

ARCHITECTURE XOR_2 OF XOR2 IS
BEGIN
        z <= x XOR y AFTER 12 ns;
END;
-------------------------------------------------
```

```
Library ieee;
USE ieee.std_logic_1164.all;

ENTITY XOR8 IS
        PORT (x ,y : IN std_logic_vector( 7 downto 0);
        z: OUT std_logic_vector( 7 downto 0));
        END;

ARCHITECTURE XOR_8 OF XOR8 IS
BEGIN
        z <= x XOR y AFTER 12 ns;
END;
-------------------------------------------------------


--              D FLIP-FLOP
-- ============================================================================================================
Library ieee;
USE ieee.std_logic_1164.all;

ENTITY DFF IS
        PORT (a,b,c, clock, reset: IN std_logic;
        qa, qb, qc: OUT std_logic);
END;

ARCHITECTURE DFF3 OF DFF IS
BEGIN
        PROCESS (clock)
        BEGIN
                IF rising_edge(clock) THEN
                        IF (reset='1') THEN
                                qa <= '0';
```

```
                                        qb <= '0';
                                        qc <= '0';
                                    ELSE
                                        qa <= a;
                                        qb <= b;
                                        qc <= c;
                                    END IF;
                                END IF;
            END PROCESS;
END;
-----------------------------------------------


--              FULL ADDER
-- ============================================================================================================
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ADDER IS
            PORT (a, b, cin: IN std_logic;
            sum, cout: OUT std_logic);
END;

ARCHITECTURE simple OF ADDER IS
SIGNAL s1, s2, s3: std_logic;
BEGIN
            g1: ENTITY work.XOR2(XOR_2) PORT MAP (a,b,s1);
            g2: ENTITY work.XOR2(XOR_2) PORT MAP (s1,cin,sum);
            g3: ENTITY work.AND2(AND_2) PORT MAP (s1,cin,s2);
            g4: ENTITY work.AND2(AND_2) PORT MAP (a, b,s3);
            g5: ENTITY work.OR2(OR_2) PORT MAP (s2, s3, cout);
END;
-------------------------------------------------------
```

```
--              9-BIT ADDER
-- ============================================================================================================
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY Bit9Adder IS
        PORT (a, b: IN std_logic_vector(8 downto 0);
        cin: IN std_logic;
        s: OUT std_logic_vector(8 downto 0);
        cout: OUT std_logic);
END;

ARCHITECTURE structAdd OF Bit9Adder IS
SIGNAL carrySignal: STD_logic_vector( 9 downto 0);
CONSTANT n: INTEGER := 9;
SIGNAL sig: STD_logic_vector( 8 downto 0);
BEGIN
        carrySignal(0) <= cin;
        gen0:
        FOR i IN 0 TO n-1 GENERATE -- Loop to go on all the Bits of both inputs
                g1: ENTITY work.ADDER(simple) PORT MAP (a(i), b(i), carrySignal(i), sig(i), carrySignal(i+1));
        END GENERATE gen0;
        s <= sig;
        cout <= carrySignal(9);
END;
------------------------------------------------------------------------------------------


--              A subtractor that takes 8-bit inputs and gives a 9-bit subtraction output
-- ============================================================================================================
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY Bit8Subtractor IS
        PORT(a, b: IN std_logic_vector(7 downto 0);
        sub: OUT std_logic_vector(8 downto 0));
END;

ARCHITECTURE structSub OF Bit8Subtractor IS
SIGNAL onesComp: std_logic_vector( 8 downto 0);-- B Inverted
SIGNAL nun: std_logic;
SIGNAL extraA, extraB: std_logic; -- A and B in 9-bit representation
SIGNAL a9, b9: std_logic_vector( 8 downto 0); -- 9th Bit of A and B
BEGIN
        extraA <= a(7);
        extraB <= b(7);
        a9 <= extraA&a;
        b9 <= extraB&b;
        g0: ENTITY work.inverter9(invert9) PORT MAP (b9, onesComp);
        g1: ENTITY work.Bit9Adder(structAdd) PORT MAP (a9, onesComp, '1', sub, nun);
END;
--------------------------------------------------------------------------------


--          One bit comparator that depends on previous higher bits
-- ============================================================================================================
Library ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY BitCompare IS
```

```
        PORT (a, b, pstateA, pstateB: IN std_logic;
        stateA, stateB: OUT std_logic);
END;

ARCHITECTURE chip OF BitCompare IS
SIGNAL bitDif, Abit, Bbit, As, Bs, nstateA, nstateB: std_logic;
BEGIN
        g1: ENTITY work.inverter(invert) PORT MAP (pstateA, nstateA);
        g2: ENTITY work.inverter(invert) PORT MAP (pstateB, nstateB);
        g3: ENTITY work.XOR2(XOR_2) PORT MAP (a, b, bitDif);
        g4: ENTITY work.AND3(AND_3) PORT MAP (a, bitDif, nstateB, Abit);
        g5: ENTITY work.AND3(AND_3) PORT MAP (b, bitDif, nstateA, Bbit);
        g6: ENTITY work.OR2(OR_2) PORT MAP (pstateA, Abit, As);
        g7: ENTITY work.OR2(OR_2) PORT MAP (pstateB, Bbit, Bs);
        stateA <= As;
        stateB <= Bs;
END;
--------------------------------------------------------------------------------


--                  THE COMPARATORS
-- ============================================================================================================
-- ============================================================================================================
Library ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY comparator IS
        PORT (a,b: IN std_logic_vector(7 downto 0);
        clk,reset: IN std_logic;
        F1, F2, F3: OUT std_logic);
END;
```

```
--              STAGE 2: USING MAGNITUDE COMPARATOR AND SIGN BIT
-- ============================================================================================================
ARCHITECTURE magnitudeComparator OF comparator IS
CONSTANT n: INTEGER := 8;
SIGNAL sgnDif, nA7, nB7, sgnAl, sgnBl, sgnEQ, nstateA, nstateB, bitAl, bitBl, E1, E2, E3: std_logic;
SIGNAL stateA, stateB: std_logic_vector(7 downto 0) ;
BEGIN
        stateA(7) <= '0';
        stateB(7) <= '0';
        g0: ENTITY work.inverter(invert) PORT MAP (a(n-1), nA7);
        g1: ENTITY work.inverter(invert) PORT MAP (b(n-1), nB7);
        g2: ENTITY work.XOR2(XOR_2) PORT MAP (a(n-1), b(n-1), sgnDif);
        g3: ENTITY work.AND2(AND_2) PORT MAP (nA7, sgnDif, sgnAl);
        g4: ENTITY work.AND2(AND_2) PORT MAP (nB7, sgnDif, sgnBl);
        g5: ENTITY work.inverter(invert) PORT MAP (sgnDif, sgnEQ);
        gen: FOR i IN 2 TO n GENERATE
                g6: ENTITY work.BitCompare(chip)
                        PORT MAP (a(n-i), b(n-i), stateA(n+1-i), stateB(n+1-i),
                        stateA(n-i), stateB(n-i));
        END GENERATE gen;
        g7: ENTITY work.inverter(invert) PORT MAP (stateA(0), nstateA);
        g8: ENTITY work.inverter(invert) PORT MAP (stateB(0), nstateB);
        g9: ENTITY work.AND2(AND_2) PORT MAP (sgnEQ, stateA(0), bitAl);
        g10: ENTITY work.AND2(AND_2) PORT MAP (sgnEQ, stateB(0), bitBl);
        g11: ENTITY work.AND3(AND_3) PORT MAP (nstateA, nstateB, sgnEQ, E1);
        g12: ENTITY work.OR2(OR_2) PORT MAP (bitAl, sgnAl, E2);
        g13: ENTITY work.OR2(OR_2) PORT MAP (bitBl, sgnBl, E3);
        g14: ENTITY work.DFF(DFF3) PORT MAP (E1, E2, E3, clk, reset, F1, F2, F3);
END;
-------------------------------------------------------------------------------


--              STAGE 1: STAGE 1 USING RIPPLE ADDER
-- ============================================================================================================
```

```
ARCHITECTURE adderComparator OF comparator IS
SIGNAL sgnPos, notZero, E1, E2, E3: std_logic;
SIGNAL sbt: std_logic_vector( 8 downto 0 );
BEGIN
	g0: ENTITY work.Bit8Subtractor(structSub) PORT MAP (a, b, sbt);
	g1: ENTITY  work.OR8(OR_8) PORT MAP (sbt(0), sbt(1), sbt(2), sbt(3), sbt(4), sbt(5), sbt(6), sbt(7), notZero);
	g2: ENTITY work.inverter(invert) PORT MAP (sbt(8), sgnPos);
	g3: ENTITY work.NOR9(NOR_9) PORT MAP (sbt(0), sbt(1), sbt(2), sbt(3), sbt(4), sbt(5), sbt(6), sbt(7), sbt(8), E1);
	g4: ENTITY work.AND2(AND_2) PORT MAP (notZero, sgnPos, E2);
	E3 <= sbt(8);
	g5: ENTITY work.DFF(DFF3) PORT MAP (E1, E2, E3, clk, reset, F1, F2, F3);
END;
--------------------------------------------------------------------------------


--		TEST GENERATOR
-- ============================================================================================================
Library IEEE;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY generator IS
	PORT (clock : IN std_logic;
	testA, testB: OUT std_logic_vector(7 downto 0);
	expF1, expF2, expF3: OUT std_logic);
END;

ARCHITECTURE tester OF generator IS
BEGIN
	PROCESS
	BEGIN
		FOR i IN -128 TO 127 LOOP
			FOR j IN -128 TO 127 LOOP
```

```
                                        testA <= CONV_STD_LOGIC_VECTOR(i, 8);
                                        testB <= CONV_STD_LOGIC_VECTOR(j, 8);
                                        IF ( i > j ) THEN
                                                expF1 <= '0';
                                                expF2 <= '1';
                                                expF3 <= '0';
                                        ELSIF ( j > i ) THEN
                                                expF1 <= '0';
                                                expF2 <= '0';
                                                expF3 <= '1';
                                        ELSE
                                                expF1 <= '1';
                                                expF2 <= '0';
                                                expF3 <= '0';
                                        END IF;
                                        WAIT UNTIL rising_edge(clock);
                                END LOOP;
                        END LOOP;
                        WAIT;
                END PROCESS;
END;
-------------------------------------------------------------



--              RESULT ANALYSER
-- ================================================================================================
Library IEEE;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY analyser IS
                PORT ( expF1, expF2, expF3, actF1, actF2, actF3 : IN std_logic;
```

```
        clock: IN std_logic);
END;

ARCHITECTURE resultAnalyser OF analyser IS
BEGIN
        PROCESS(clock)
        BEGIN
                IF rising_edge(clock) THEN
                        ASSERT (expF1 = actF1 AND expF2 = actF2 AND expF3 = actF3)
                        REPORT "The output is incorrect"
                        SEVERITY ERROR;
                END IF;
        END PROCESS;
END;
--------------------------------------------------------------------------------



--              TESTING FOR BOTH STAGES
-- ============================================================================================================
-- ============================================================================================================
Library IEEE;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY testBinch IS
END;

--          TESTING FOR STAGE 1
-- ============================================================================================================
ARCHITECTURE adderCompTest OF testBinch IS
SIGNAL clock, clock2, expF1, expF2, expF3, actF1, actF2, actF3: std_logic:='0';
SIGNAL A, B: std_logic_vector( 7 downto 0 ):="00000000";
```

```
BEGIN
        clock <= not clock after 165 ns;
        clock2 <= not clock2 after 330 ns;
        g1: ENTITY work.generator(tester) PORT MAP (clock2, A, B, expF1, expF2, expF3);
        g2: ENTITY work.comparator(adderComparator) PORT MAP (A, B, clock, '0', actF1, actF2, actF3);
        g3: ENTITY work.analyser(resultAnalyser) PORT MAP (expF1, expF2, expF3, actF1, actF2, actF3, clock2);
END;
-------------------------------------------------------------------------------


--          TESTING FOR STAGE 2
-- ===============================================================================================================
ARCHITECTURE magCompTest OF testBinch IS
SIGNAL clock, clock2, expF1, expF2, expF3, actF1, actF2, actF3: std_logic:='0';
SIGNAL A, B: std_logic_vector( 7 downto 0 ):="00000000";
BEGIN
        clock <= not clock after 92 ns;
        clock2 <= not clock2 after 184 ns;
        g1: ENTITY work.generator(tester) PORT MAP (clock2, A, B, expF1, expF2, expF3);
        g2: ENTITY work.comparator(magnitudeComparator) PORT MAP (A, B, clock, '0', actF1, actF2, actF3);
        g3: ENTITY work.analyser(resultAnalyser) PORT MAP (expF1, expF2, expF3, actF1, actF2, actF3, clock2);
END;
---------------------------------------------------------------------------------------------------------------
```