BERZIET UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

COMPUTER DESIGN LABORATORY

ADVANCED DIGITAL DESIGN

ENCS3310

Course Project

Student's name: Sara Issa Student's ID: 1190673

Instructor's name: Dr. Abdellatif Abu-Issa Section No. 1

Date: 19.12.2021

Abstract:

The aim of the task is to design an 8-bit Comparator for signed 2’s complement representation numbers, and then to write a complete code for functional verification. By using the following types of circuits: Ripple Adder/subtractor, and magnitude (unsigned) comparator.

Table of Contents:

List Of Figures:

List of tables:

1. Theory:

1.1 Basic Gates:

All digital systems can be constructed by only three basic logic gates. These basic gates are called the AND gate, the OR gate, and the INVERTER (NOT) gate. Also, include the NAND gate, the NOR gate, the XNOR gate and the XOR gate as the members of the family of basic logic gates [1]. The Comparator is to be built **structurally from a library of gates**, which contains the following devices in Table NO.1-1:

Gate	Delay (in ns)
Inverter (Not)	2 ns
NAND	5 ns
NOR	5 ns
AND	7 ns
OR	7 ns
XNOR	9 ns
XOR	12 ns

NOTE: The code of Basic gates in Appendix 1.

1.2 Full Adder:

A full adder circuit is central to most digital circuits that perform addition or subtraction. it is the adder which adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM.

NOTE: The code of full adder circuit in Appendix 1, and simulation for full adder in Appendix 2.

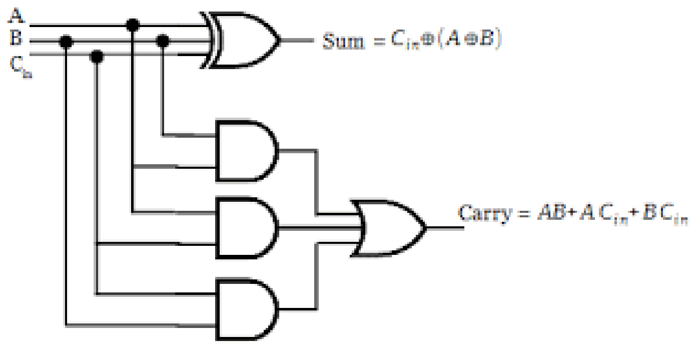

A
B
C_{in}
Sum $= C_{in} \oplus (A \oplus B)$
Carry $= AB + AC_{in} + BC_{in}$

Figure 1-1: full adder circuit [2]

1.3 Ripple Circuit (adder/subtractor):

A type of electronics adder-subtractor used in digital logic. It is capable of both addition and subtraction of binary numbers in one circuit itself. The operation being performed depends the binary value the control signal holds. If input carry = 0, then ripple circuit work as adder. Else if input carry = 1, then ripple circuit work as subtractor. This Circuit used of Xor Gate and Full Adder.

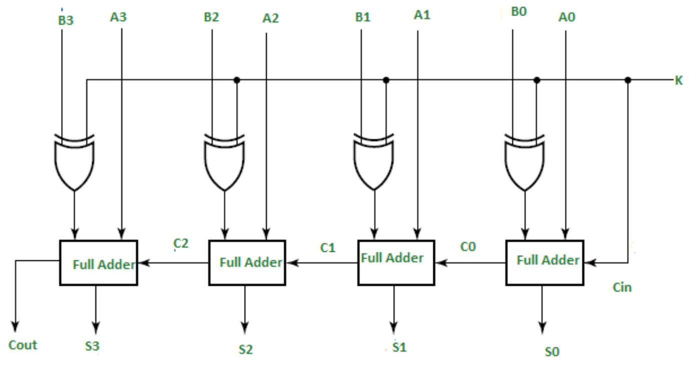


Figure 1-2: 4-Bit ripple Circuit (adder/subtractor) [3]

NOTE:

- in previous figure 1-2: 4-Bit ripple Circuit, but in my project it has been used 8-Bit ripple Circuit (because inputs A,B have 8-bit).
- The code of 8-Bit Carry Look-Ahead Circuit in Appendix 1.

1.4 Carry-lookahead adder (CLA) or fast adder:

A type of electronics adder used in digital logic. A carry-lookahead adder improves speed by reducing the amount of time required to determine carry bits. It can be contrasted with the simpler, but usually slower, ripple-carry adder (RCA).

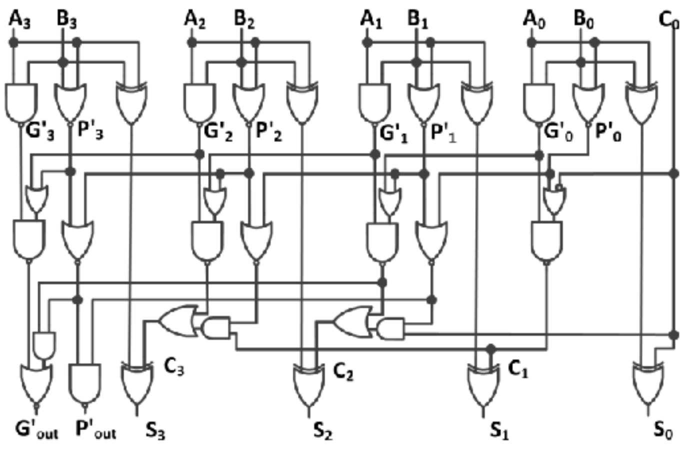


Figure 1-3: 4-Bit Carry-lookahead circuit (fast adder/subtractor) [4]

1.5 7-Bits Magnitude Comparator:

A 7-bit magnitude comparator compares the two 7-bit values and produce a 1-bit flag as result, which indicates that the first value is either greater than or less than or equal to the second value. The block diagram of a comparator is shown in Figure 1-4:

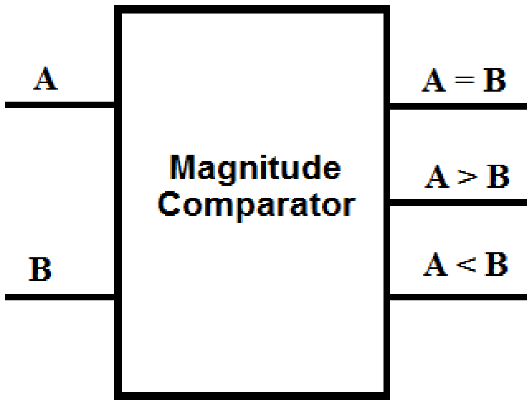


Figure 1-4: Block Diagram Of Magnitude Comparator[5]

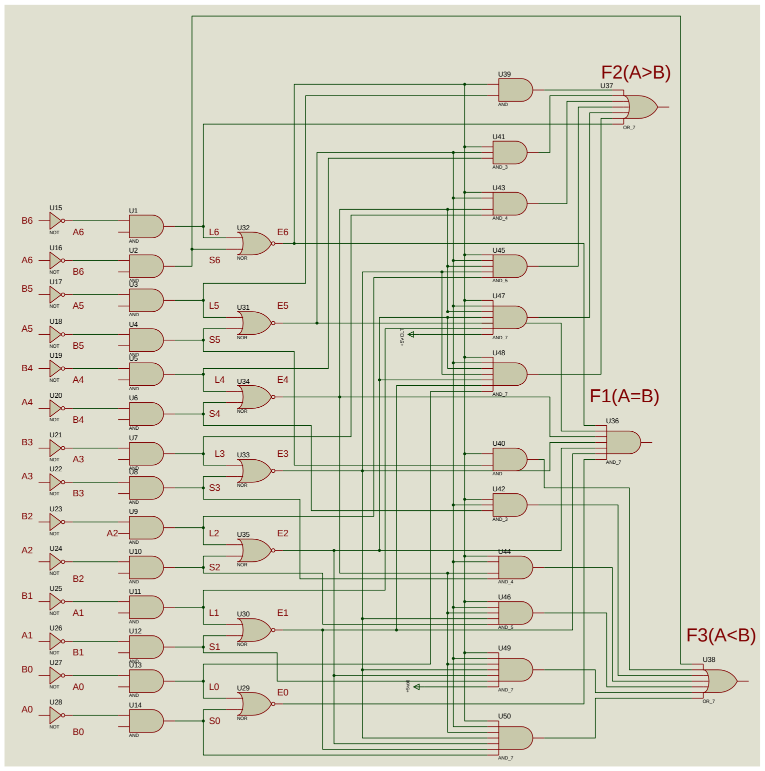


Figure 1-5: 7-Bits Magnitude Comparator circuit

NOTE:

- The code of 7-Bits Magnitude Comparator in Appendix 1.
- There is no picture on the google of 7-Bits Magnitude Comparator circuit so I built it through proteus program, but there is no AND gate with 6 inputs so I used a AND gate with 7 inputs and connected to input number 7 with Vcc, but in the code of 7-Bits Magnitude Comparator circuit was used AND gate with 6 inputs.

2. Procedure and Discussion:

To design an 8-bit Comparator for signed 2’s complement representation 2 numbers (A and B) and to produce 3 outputs F1(A=B), F2(A>B), and F3 (A<B). By using design structurally in different ways: ripple circuit (Adder/subtractor) in stage 1, and magnitude (unsigned) comparator in stage 2.

2.1. Stage1: By using the subtractor (ripple circuit) to implement the Comparison between 2 numbers (A and B) represented in signed 2’s complement representation.

Initially, a circuit of **1-bit full adder** was created by using AndGate2, OrGate3 and XorGate 3, sum = A xor B xor Cin and cout = (A.B) + (A.Cin) + (B.Cin). Then, it has been created a **8-bits ripple circuit** that can work as adder or subtractor depending on the value of Cin, if Cin=0 then it works as adder (A+B) and if Cin=1 works as subtractor (A-B = A+(NOT B)+1), it's built structurally by using 8 circuit from 1-bit full adder, it depends on the number of bits of the input values.

I choose to run the ripple circuit as a subtractor by entering the value Cin = 1, and through the result the two values are compared. If the result = 00000000 and overflow = 0, then F1 (A = B) = 1, F2 (A > B) = 0 and F3 (A < B) = 0 This means that the two numbers are equal in magnitude and sign, then **NorGate9** consisting of 9 inputs (result of ripple circuit and overflow) is used to check if all the inputs are zeros, then A=B, otherwise A! = B.

There are four possibilities for F3(A<B) = 1, when A, B are positive numbers then A(7)=0 & B(7)=0 & overflow=0 & result(7)=1 this is found out from During the experiment on a lot of numbers, and A, B can be negative numbers, then A(7)=1 & B(7)=1 & overflow=0 & result(7)=1, and the probability that A is a negative number but B is A positive number then A(7)=1 & B(7)=0 & overflow=0 & result(7)=1 or A(7)=1 & B(7)=0 & overflow=1 & result(7) =0, as a result, a circuit was created using each of **four from InverterGate**, **four from AndGate4** and **one from OrGate4** so that it gives a result of F3(A<B) = 1 when one of the above possibilities are happened.

As for F2 (A > B), its value = 1 when both F1(A= B) = 0 and F3 (A < B) = 0, while its value = 0 when one of the two values of F1 (A = B) = 1 or F3(A < B) = 1, so to find out the result of F2 (A > B) **NorGate** was used with two inputs (F1 (A = B) and F3 (A < B)).

The code for the Signed Comparator by using ripple circuit (subtractor):

```
LIBRARY IEEE;

USE IEEE.std_logic_1164.ALL;

-- Stage 1: Use the adder (ripple circuit) to implement the Signed Comparator

ENTITY SignedComparatorByRipple IS

        PORT ( A, B: IN STD_LOGIC_VECTOR(7 DOWNTO 0);

            AEqualsB, AGreaterThanB, ALessThanB: OUT  STD_LOGIC );

END ENTITY SignedComparatorByRipple;
```

```
ARCHITECTURE simple OF SignedComparatorByRipple IS
        SIGNAL result: STD_LOGIC_VECTOR (7 DOWNTO 0);
        SIGNAL overflow, NotA, NotB, Notv, NotRes, NotS, E, L, N1, N2, N3, N4: STD_LOGIC;
BEGIN
        AEqualsB <= E;
        ALessThanB <= L;
        g1: ENTITY work. rippleCircuit (simple) PORT MAP (A, B, '1', result, overflow); -- ripple circuit (subtractor)
    g2: ENTITY work.NorGate9(simple) PORT MAP (overflow, result(7), result(6), result(5), result(4), result(3), result(2),
result(1), result(0), E); -- when result="00000000" and overflow='0' , then F1(A=B) = 1
        g3: ENTITY work.InverterGate(simple) PORT MAP (A(7), NotA); -- NotA = NOT A(7)
        g4: ENTITY work.InverterGate(simple) PORT MAP (B(7), NotB);    -- NotB = NOT B(7)
        g5: ENTITY work.InverterGate(simple) PORT MAP (overflow, Notv); -- Notv = NOT overflow
        g6: ENTITY work.InverterGate(simple) PORT MAP (result(7), NotRes); -- NotRes = NOT result(7)
        -- The four probabilities of having A less than B
        g7: ENTITY work.AndGate4(simple) PORT MAP (A(7), NotB, Notv,result(7), N1);
        g8: ENTITY work.AndGate4(simple) PORT MAP (A(7), B(7), Notv, result(7), N2);
        g9: ENTITY work.AndGate4(simple) PORT MAP (NotA, NotB, Notv, result(7), N3);
        g10: ENTITY work.AndGate4(simple) PORT MAP (A(7), NotB, overflow, NotRes, N4);
        -- L = 1 when at least one of the above possibilities = 1
        g11: ENTITY work.OrGate4(simple) PORT MAP (N1, N2, N3, N4, L);
        -- To find value of AGreaterThanB by F2(A>B) = F1(A=B) NOR F3(A<B)
        g12: ENTITY work.NorGate(simple) PORT MAP (E, L, AGreaterThanB);
END ARCHITECTURE simple;
```

2.2. Stage2: By using the 7-bits Magnitude comparator and the sign bit to implement the Comparison between 2 numbers (A and B) represented in signed 2’s complement representation.

Since the last bit on the left is the sign of the number, if 1 is a negative number, but if 0 is a positive number, then **SevenBitComparator** was used with inputs A (7 DOWNTO 0) and B (7 DOWNTO 0) were used to compare the numbers from In the magnitude aspect regardless of the sign of the original 8-bit number, the result will appear on the three outputs of SevenBitComparator, since SevenBitComparator is structurally built using: **14 from InverterGate**, **14 from AndGate2**, and **7 from NorGate** to find the value of F1 (A=B), in addition to using **AndGate7**, **AndGate6**, **AndGate5**, **AndGate4**, **AndGate3**, **AndGate2**, **OrGate7** to find both F2 (A>B) and F3 (A<B).

But the sign bit cannot be ignored, so based on the previous three outputs from SevenBitComparator and the sign bit values of A,B the correct result is obtained to compare the numbers A,B from 8-bit. There are four possibilities for the signal values of A and B, if they are of the same signal A(7)=B(7)=0 or A(7)=B(7)=1 then it can affect the previous three outputs from SevenBitComparator so **XnorGate** is used to give result 1 when A(7)=B(7), then using

AndGate2 with previous three outputs, so the output will remain the same as SevenBitComparator.

While the result of F2 (A>B) = '1' in two cases: first case when A(7)='0' then A is a positive number, while B(7)='1' then B is a negative number, so A>B no matter the value of each magnitude of them (The result depends on the equation of following circuit: (NOT A) AND B, **InverterGate** and **AndGate2** have been used). In the second case, when the sign A and B are the same, then the result of the signed comparator is based on the result of SevenBitComparator (**InverterGate**, **AndGate2**, **SevenBitComparator** and **OrGate** have been used).

Finally, the result of F3 (A<B) = '1' in two cases: first case when A(7)='1' then A is a negative number, while B(7)='0' then B is a positive number, so A<B no matter the value of each magnitude of them (The result depends on the equation of following circuit: A AND (NOT B), **InverterGate** and **AndGate2** have been used). In the second case, when the sign A and B are the same, then the result of the signed comparator is based on the result of SevenBitComparator (**InverterGate**, **AndGate2**, **SevenBitComparator** and **OrGate** have been used).

The code for the Signed Comparator by using 7-bits Magnitude comparator and sign bit:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
--Stage 2: Use the Magnitude comparator and the sign bit to implement the signed comparator
ENTITY SignedComparator IS
        PORT ( A, B: IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
               AEqualsB, AGreaterThanB, ALessThanB : OUT  STD_LOGIC );
END ENTITY SignedComparator;

ARCHITECTURE simple OF SignedComparator IS
  SIGNAL E, G, L, NotA, NotB: STD_LOGIC;
  SIGNAL S: STD_LOGIC_VECTOR (2 DOWNTO 0);
        SIGNAL U: STD_LOGIC_VECTOR (1 DOWNTO 0);
BEGIN
        g1: ENTITY work.InverterGate(simple) PORT MAP (A(7), NotA);
        g2: ENTITY work.InverterGate(simple) PORT MAP (B(7), NotB);
        g3: ENTITY work.XnorGate(simple) PORT MAP (A(7), B(7), S(2)); -- The output = '1' when A(7) = B(7)
        g4: ENTITY work.AndGate2(simple) PORT MAP (NotA, B(7), S(1)); -- The output = '1' when A(7)='0' and B(7)='1'
        g5: ENTITY work.AndGate2(simple) PORT MAP (A(7), NotB, S(0)); -- The output = '1' when A(7)='1' and B(7)='0'
        g6: ENTITY work.SevenBitComparator(simple) PORT MAP (A(6 DOWNTO 0), B(6 DOWNTO 0), E, G, L);
        g7: ENTITY work.AndGate2(simple) PORT MAP (S(2), E, AEqualsB); -- To find the value of F1(A=B)
        g8: ENTITY work.AndGate2(simple) PORT MAP (S(2), G, U(1));
        g9: ENTITY work.AndGate2(simple) PORT MAP (S(2), L, U(0));
        g10: ENTITY work.OrGate(simple) PORT MAP (U(1), S(1), AGreaterThanB); -- To find the value of F2(A>B)
        g11: ENTITY work.OrGate(simple) PORT MAP (U(0), S(0), ALessThanB); -- To find the value of F3(A<B)
END ARCHITECTURE simple;
```

2.3 Test Generator: Test generator has a clock input, A and B are two inputs and two outputs at the same time, as well as three outputs CorrectF1, CorrectF2, CorrectF3. Inside the ARCHITECTURE of each stage contains two process, the first to find out the correct results through behavioral description, and the second to change the values of the inputs A and B with each rising edge for clock to try all the numbers within the period -128 to +127 to make sure that the design is working correctly.

```
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 :='0';
                  A, B: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
                  CorrectF1, CorrectF2, CorrectF3: OUT STD_LOGIC :=  '0');
END ENTITY TestGenerator;
-------------------- ARCHITECTURE to Test generator of stage 1 with delay time = 268 ns --------------------
ARCHITECTURE generater1 OF TestGenerator IS
SIGNAL lA, lB: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
            A <= lA;
            B <= lB;
PROCESS(lA, lB) -- To find the correct result of the comparator using behavioral description
VARIABLE lCorrectF1, lCorrectF2, lCorrectF3: STD_LOGIC:= '0';
BEGIN
            IF (lA = lB) THEN
                        lCorrectF1 := '1';
                        lCorrectF2 := '0';
                        lCorrectF3 := '0';
            ELSIF (lA > lB) THEN
                        lCorrectF1 := '0';
                        lCorrectF2 := '1';
                        lCorrectF3 := '0';
            ELSIF (lA < lB) THEN
                        lCorrectF1 := '0';
                        lCorrectF2 := '0';
                        lCorrectF3 := '1';
   END IF;
            CorrectF1 <= lCorrectF1 AFTER 268 ns; -- delay time = 268 ns
            CorrectF2 <= lCorrectF2 AFTER 268 ns; -- delay time = 268 ns
            CorrectF3 <= lCorrectF3 AFTER 268 ns; -- delay time = 268 ns
```

```
END PROCESS;
-- This process to test all values for A and B from -128 to +127, by change the values with rising edge clock
PROCESS
BEGIN
  FOR I IN -128 TO 127 LOOP
    FOR J IN -128 TO 127 LOOP
      -- Set the inputs to the Comparator
      lA(7 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(i,8);
      lB(7 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(j,8);
      WAIT until rising_edge(CLK);     -- Wait until Comparator output has settled
    END LOOP;
  END LOOP;
  WAIT;
END PROCESS;
END ARCHITECTURE generater1;
-------------------- ARCHITECTURE to Test generator of stage 2 with delay time = 74 ns --------------------
ARCHITECTURE generater2 OF TestGenerator IS
SIGNAL lA, lB: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
		A <= lA;
		B <= lB;
PROCESS(lA, lB) -- To find the correct result of the comparator using behavioral description
VARIABLE lCorrectF1, lCorrectF2, lCorrectF3: STD_LOGIC:= '0';
BEGIN
		IF (lA = lB) THEN
				lCorrectF1 := '1';
				lCorrectF2 := '0';
				lCorrectF3 := '0';
		ELSIF (lA > lB) THEN
				lCorrectF1 := '0';
				lCorrectF2 := '1';
				lCorrectF3 := '0';
		ELSIF (lA < lB) THEN
				lCorrectF1 := '0';
				lCorrectF2 := '0';
				lCorrectF3 := '1';
    END IF;
		CorrectF1 <= lCorrectF1 AFTER 74 ns; -- delay time = 74 ns
		CorrectF2 <= lCorrectF2 AFTER 74 ns; -- delay time = 74 ns
```

```
        CorrectF3 <= lCorrectF3 AFTER 74 ns; -- delay time = 74 ns
END PROCESS;
-- This process to test all values for A and B from -128 to +127, by change the values with rising edge clock
PROCESS
BEGIN
 FOR I IN -128 TO 127 LOOP
  FOR J IN -128 TO 127 LOOP
   -- Set the inputs to the Comparator
   lA(7 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(i,8);
   lB(7 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(j,8);
   WAIT until rising_edge(CLK); -- Wait until Comparator output has settled
  END LOOP;
 END LOOP;
 WAIT;
END PROCESS;
END ARCHITECTURE generater2;
```

2.4 Comparator Test: The design is tested by: Test generator and Result analyzer and they share a clock, its A and B values changes depending on the delay time of the circuit, with each rising edge for a clock, and the difference between the two stages are the delay time. The input values will change and then the correct result will come out from the Test generator, and my design result, then will be compared between them in case there is a flaw in the design, meaning that the two results are not equal, then it will print a warning message on the console.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
ENTITY ComparatorTest IS
END ENTITY ComparatorTest;
ARCHITECTURE stage1 OF ComparatorTest IS
--Declarations of test inputs and outputs
  SIGNAL CLK: STD_LOGIC := '0';
  SIGNAL A, B: STD_LOGIC_VECTOR (7 DOWNTO 0);
  SIGNAL CorrectF1, CorrectF2, CorrectF3, MyResult1, MyResult2, MyResult3: STD_LOGIC := '0';
BEGIN
        CLK <= NOT CLK AFTER 268 ns;
 -- Place one instance of test generator
  TG:  ENTITY work.TestGenerator(generater1) PORT MAP (CLK, A, B, CorrectF1, CorrectF2, CorrectF3);
 -- Place one instance of the Unit Under Test
  SignedComparator1:  ENTITY work.SignedComparatorByRipple(simple) PORT MAP (A, B, MyResult1, MyResult2,
MyResult3);
 -- Place one instance of result analyzer
```

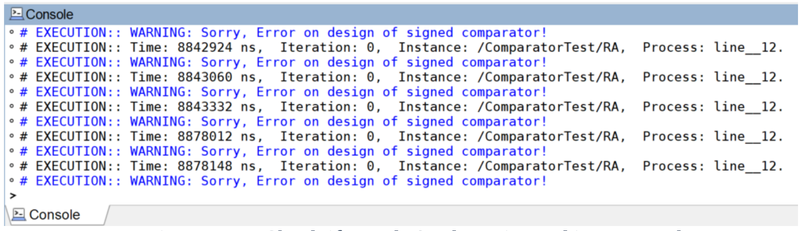
```
   RA:  ENTITY work.ResultAnalyzer(analyzer) PORT MAP (CLK, CorrectF1, CorrectF2, CorrectF3, MyResult1, MyResult2,
MyResult3);

END ARCHITECTURE stage1;

ARCHITECTURE stage2 OF ComparatorTest IS

--Declarations of test inputs and outputs

   SIGNAL CLK: STD_LOGIC := '0';

   SIGNAL A, B: STD_LOGIC_VECTOR (7 DOWNTO 0);

   SIGNAL CorrectF1, CorrectF2, CorrectF3, MyResult1, MyResult2, MyResult3: STD_LOGIC := '0';

BEGIN

            CLK <= NOT CLK AFTER 74 ns;

 -- Place one instance of test generator

   TG:  ENTITY work.TestGenerator(generater2) PORT MAP (CLK, A, B, CorrectF1, CorrectF2, CorrectF3);

 -- Place one instance of the Unit Under Test

   SignedComparator2:  ENTITY work.SignedComparator(simple) PORT MAP (A, B, MyResult1, MyResult2, MyResult3);

 -- Place one instance of result analyzer

   RA:  ENTITY work.ResultAnalyzer(analyzer) PORT MAP (CLK, CorrectF1, CorrectF2, CorrectF3, MyResult1, MyResult2,
MyResult3);

END ARCHITECTURE stage2;
```

2.5 Result Analyzer: when rising edge for clock then check if exception result equal the result from my comparator, if not then print warning message in console.

```
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

USE IEEE.STD_LOGIC_ARITH.ALL;

-- Result analyzer "print warning if difference found"

ENTITY ResultAnalyzer IS

   PORT ( CLK: IN STD_LOGIC := '0';

                  CorrectF1, CorrectF2, CorrectF3, MyResult1, MyResult2, MyResult3: IN STD_LOGIC := '0');

END ENTITY ResultAnalyzer;

ARCHITECTURE analyzer OF ResultAnalyzer IS

BEGIN

  PROCESS

  BEGIN

     -- Check whether Comparator outputs matches expectation

     ASSERT   (CorrectF1 =  MyResult1) AND (CorrectF2 =  MyResult2) AND (CorrectF3 =  MyResult3)

     REPORT   "Sorry, Error on design of signed comparator!"

     SEVERITY WARNING; -- Report error message to console and continue

   WAIT UNTIL rising_edge(CLK);

  END PROCESS;

END ARCHITECTURE analyzer;
```

3. Results:

3.1. Check if Result Analyzer is working properly:
XNOR has been replaced by XOR in Signed Comparator, so the results will be showing an error, Or by making the delay time less than necessary, so the warning message was printed on the console as shown in the Figure 3-1:



```
Console
# EXECUTION:: WARNING: Sorry, Error on design of signed comparator!
# EXECUTION:: Time: 8842924 ns,  Iteration: 0,  Instance: /ComparatorTest/RA,  Process: line__12.
# EXECUTION:: WARNING: Sorry, Error on design of signed comparator!
# EXECUTION:: Time: 8843060 ns,  Iteration: 0,  Instance: /ComparatorTest/RA,  Process: line__12.
# EXECUTION:: WARNING: Sorry, Error on design of signed comparator!
# EXECUTION:: Time: 8843332 ns,  Iteration: 0,  Instance: /ComparatorTest/RA,  Process: line__12.
# EXECUTION:: WARNING: Sorry, Error on design of signed comparator!
# EXECUTION:: Time: 8878012 ns,  Iteration: 0,  Instance: /ComparatorTest/RA,  Process: line__12.
# EXECUTION:: WARNING: Sorry, Error on design of signed comparator!
# EXECUTION:: Time: 8878148 ns,  Iteration: 0,  Instance: /ComparatorTest/RA,  Process: line__12.
# EXECUTION:: WARNING: Sorry, Error on design of signed comparator!
>
Console
```

Figure 3-1: Check if Result Analyzer is working properly or not

3.2. Stage 1:
By using the adder (ripple) to implement the Signed Comparator:

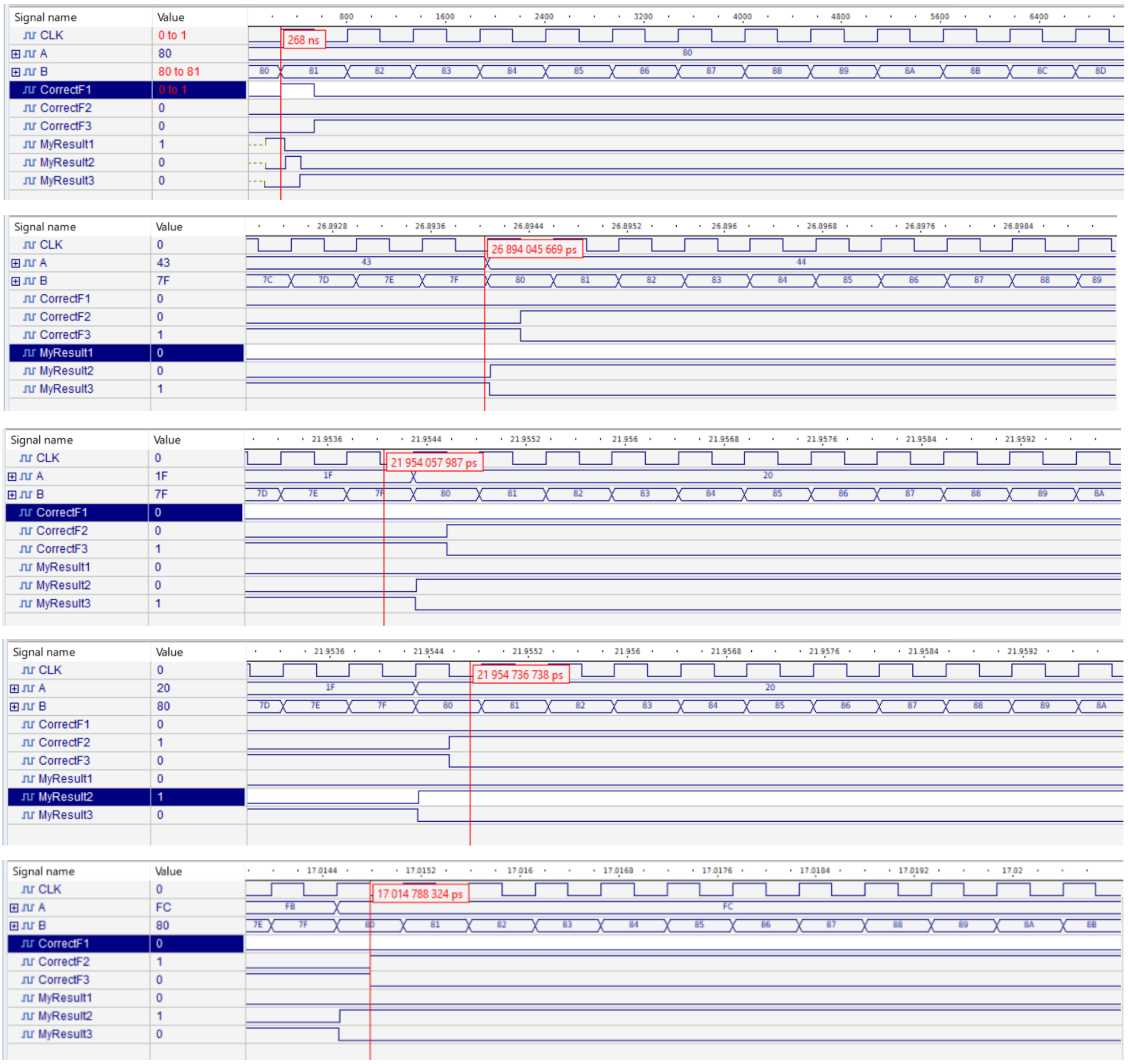
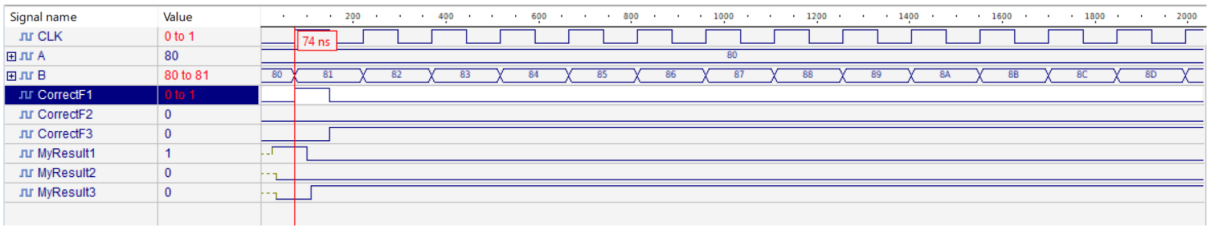
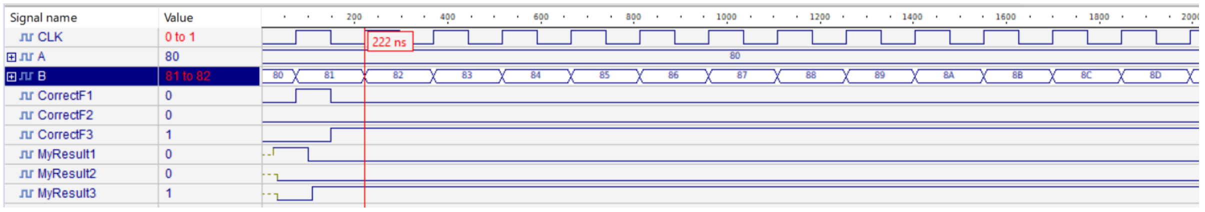


Figure 3-2: Stage 1: By using the adder (ripple) to implement the Signed Comparator

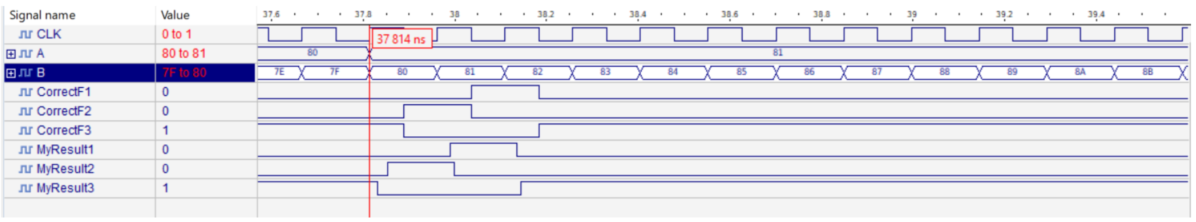
3.3. Stage 2: By using the Magnitude comparator and the sign bit to implement the signed comparator:



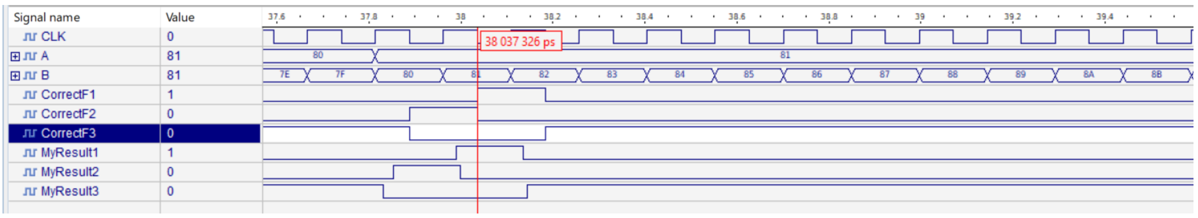
Signal name	Value
CLK	0 to 1
A	80
B	80 to 81
CorrectF1	0 to 1
CorrectF2	0
CorrectF3	0
MyResult1	1
MyResult2	0
MyResult3	0



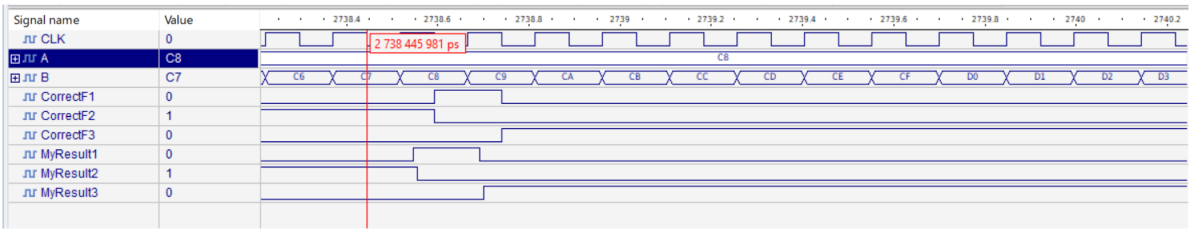
Signal name	Value
CLK	0 to 1
A	80
B	81 to 82
CorrectF1	0
CorrectF2	0
CorrectF3	1
MyResult1	0
MyResult2	0
MyResult3	1



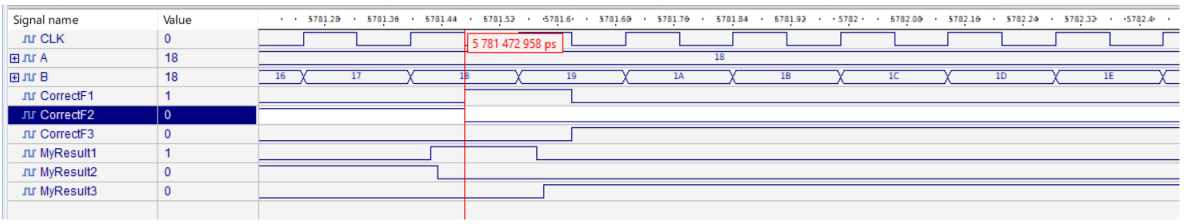
Signal name	Value
CLK	0 to 1
A	80 to 81
B	7F to 80
CorrectF1	0
CorrectF2	0
CorrectF3	1
MyResult1	0
MyResult2	0
MyResult3	1



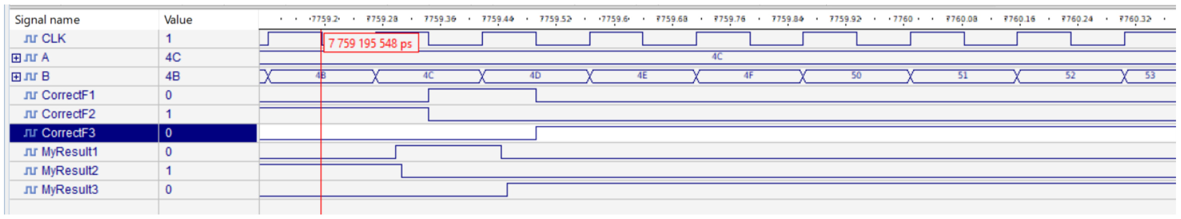
Signal name	Value
CLK	0
A	81
B	81
CorrectF1	1
CorrectF2	0
CorrectF3	0
MyResult1	1
MyResult2	0
MyResult3	0



Signal name	Value
CLK	0
A	C8
B	C7
CorrectF1	0
CorrectF2	1
CorrectF3	0
MyResult1	0
MyResult2	1
MyResult3	0



Signal name	Value
CLK	0
A	18
B	18
CorrectF1	1
CorrectF2	0
CorrectF3	0
MyResult1	1
MyResult2	0
MyResult3	0



Signal name	Value
CLK	1
A	4C
B	4B
CorrectF1	0
CorrectF2	1
CorrectF3	0
MyResult1	0
MyResult2	1
MyResult3	0

Figure 3-3: Stage 2: By using the Magnitude comparator and the sign bit to implement the signed comparator

Note: Initially the values for Correct F1= Correct F2= Correct F3 = '0' while the values for MyResult1 = MyResult2 = MyResult3 = "U", so the warning message will be printed only once.

4. Conclusion, Judgement and Creativity:

Through this project we learned about: Full Adder, Ripple Circuit (adder/subtractor), Carry-lookahead adder (CLA) and 7-Bits Magnitude Comparator and how they work. In addition, the Carry-lookahead adder is faster than the Ripple Circuit because the Full Adder inside it works in parallel all together, unlike the Ripple Circuit, each Cin depends on the Cout result of the previous Full Adder, so it is rather slow.

Also, the focus was on what was learned in the Advanced Digital Systems Design course about VHSIC Hardware Description Language, and we were able to understand both: behavioral and structural descriptions and design verification (by using test generator, test bench and result analyzer to print warning in console when the design has error).

Mistakes and creativity:

- I thought that only gates have only two inputs, so I used, for example, instead of **AND** gate with 8 inputs, 7 from **AND** gate with two inputs, so in this case the delay time will become very large because each gate depends on the result of the previous gate.
- In stage 1: when F2 (A > B) was found, **NorGate** was used, knowing that I could have used **XnorGate** because the possible states for the values of F1 (A = B) and F3 (A < B) are only three:

  1. When F1(A= B) = 0 and F3 (A < B) = 0.
  2. When F1(A= B) = 0 and F3 (A < B) = 1.
  3. When F1(A= B) = 1 and F3 (A < B) = 0.

  Since it cannot be F1(A= B) = 1 and F3(A < B) = 1, so it was possible to use **NorGate** or **XnorGate**, but **NorGate** was used because its delay time = 5 ns less than **XnorGate** with delay time = 9 ns.

Ideas to improve the design: Since my design works as required, this is good, but it does not mean that it is the perfect design, because there are flaws in every design, so it can improve its work by replacing ripple subtractor with look ahead subtractor because it is faster, therefore it will reduce the delay time and the design will be better, the project can also be worked through generic hardware design, which leads to make it work correctly no matter the number of bits for inputs.

Design Verification Overview:

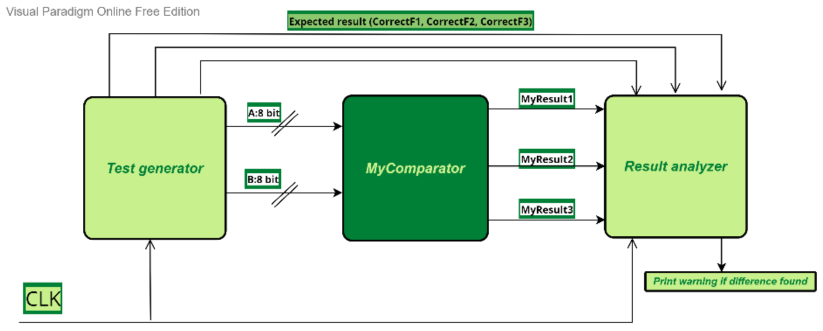

Visual Paradigm Online Free Edition
Expected result (CorrectF1, CorrectF2, CorrectF3)
Test generator
A:8 bit
B:8 bit
MyComparator
MyResult1
MyResult2
MyResult3
Result analyzer
Print warning if difference found
CLK

Figure 4-1: Design Verification Overview [6]

5. Appendix 1:

5.1 The code of Basic gates:

I. Inverter Gate (Not Gate):

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY InverterGate IS
PORT(a: IN STD_LOGIC;
b: OUT STD_LOGIC);
END ENTITY InverterGate;
ARCHITECTURE simple OF InverterGate IS
BEGIN
b <= NOT a AFTER 2 NS; -- delay = 2 ns
END ARCHITECTURE simple;
```

II. Nand Gate:

```
 LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY NandGate IS
            PORT(a,b: IN STD_LOGIC;
            c: OUT STD_LOGIC);
END ENTITY NandGate;
ARCHITECTURE simple OF NandGate IS
BEGIN
   c <= a NAND b AFTER 5 NS; -- delay = 5 ns
END ARCHITECTURE simple;
```

III. Nor Gate:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-- Two inputs for NOR gate
ENTITY NorGate IS
            PORT(a1,a0: IN STD_LOGIC;
            b: OUT STD_LOGIC);
END ENTITY NorGate;
ARCHITECTURE simple OF NorGate IS
BEGIN
   b <= a1 NOR a0 AFTER 5 NS; -- delay = 5 ns
END ARCHITECTURE simple;
-- Nine inputs for NOR gate
ENTITY NorGate9 IS
            PORT (a8,a7,a6,a5,a4,a3,a2,a1,a0: IN STD_LOGIC;
            b: OUT STD_LOGIC);
END ENTITY NorGate9;
ARCHITECTURE simple OF NorGate9 IS
BEGIN
   b <= NOT (a8 OR a7 OR a6 OR a5 OR a4 OR a3 OR a2 OR a1 OR a0) AFTER 5 NS; -- delay = 5 ns
END ARCHITECTURE simple;
```

IV. And Gate:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-- Two inputs for AND gate

ENTITY AndGate2 IS

            PORT (a1,a0: IN STD_LOGIC;

            b: OUT STD_LOGIC );
```

```
END ENTITY AndGate2;

ARCHITECTURE simple OF AndGate2 IS
BEGIN
   b <= a1 AND a0 AFTER 7 NS; -- delay = 7 ns
END ARCHITECTURE simple;
-- Three inputs for AND gate
ENTITY AndGate3 IS
            PORT (a2,a1,a0: IN STD_LOGIC;
            b: OUT STD_LOGIC );
END ENTITY AndGate3;

ARCHITECTURE simple OF AndGate3 IS
BEGIN
   b <= a2 AND a1 AND a0 AFTER 7 NS; -- delay = 7 ns
END ARCHITECTURE simple;
-- Four inputs for AND gate
ENTITY AndGate4 IS
            PORT (a3,a2,a1,a0: IN STD_LOGIC;
            b: OUT STD_LOGIC );
END ENTITY AndGate4;

ARCHITECTURE simple OF AndGate4 IS
BEGIN
   b <= a3 AND a2 AND a1 AND a0 AFTER 7 NS; -- delay = 7 ns
END ARCHITECTURE simple;
-- Five inputs for AND gate
ENTITY AndGate5 IS
            PORT (a4,a3,a2,a1,a0: IN STD_LOGIC;
            b: OUT STD_LOGIC );
END ENTITY AndGate5;

ARCHITECTURE simple OF AndGate5 IS
BEGIN
   b <= a4 AND a3 AND a2 AND a1 AND a0 AFTER 7 NS; -- delay = 7 ns
END ARCHITECTURE simple;
-- Six inputs for AND gate
ENTITY AndGate6 IS
            PORT (a5,a4,a3,a2,a1,a0: IN STD_LOGIC;
            b: OUT STD_LOGIC );
END ENTITY AndGate6;
```

```
ARCHITECTURE simple OF AndGate6 IS
BEGIN
   b <= a5 AND a4 AND a3 AND a2 AND a1 AND a0 AFTER 7 NS; -- delay = 7 ns
END ARCHITECTURE simple;
-- Seven inputs for AND gate
ENTITY AndGate7 IS
            PORT (a6,a5,a4,a3,a2,a1,a0: IN STD_LOGIC;
            b: OUT STD_LOGIC );
END ENTITY AndGate7;

ARCHITECTURE simple OF AndGate7 IS
BEGIN
   b <= a6 AND a5 AND a4 AND a3 AND a2 AND a1 AND a0 AFTER 7 NS; -- delay = 7 ns
END ARCHITECTURE simple;
```

V. Or Gate:

```
 LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-- Two inputs for OR gate
ENTITY OrGate IS
            PORT (a,b: IN STD_LOGIC;
            c: OUT STD_LOGIC);
END ENTITY OrGate;

ARCHITECTURE simple OF OrGate IS
BEGIN
   c <= a OR b AFTER 7 NS; -- delay = 7 ns
END ARCHITECTURE simple;
-- Three inputs for OR gate
ENTITY OrGate3 IS
            PORT(a2,a1,a0: IN STD_LOGIC;
            b: OUT STD_LOGIC);
END ENTITY OrGate3;

ARCHITECTURE simple OF OrGate3 IS
BEGIN
   b <= a2 OR a1 OR a0 AFTER 7 NS; -- delay = 7 ns
END ARCHITECTURE simple;
-- Four inputs for OR gate
ENTITY OrGate4 IS
            PORT (a3,a2,a1,a0: IN STD_LOGIC;
            b: OUT STD_LOGIC);
```

```
END ENTITY OrGate4;

ARCHITECTURE simple OF OrGate4 IS
BEGIN
   b <= a3 OR a2 OR a1 OR a0 AFTER 7 NS; -- delay = 7 ns
END ARCHITECTURE simple;
-- Seven inputs for OR gate
ENTITY OrGate7 IS
            PORT(a6,a5,a4,a3,a2,a1,a0: IN STD_LOGIC;
            b: OUT STD_LOGIC);
END ENTITY OrGate7;

ARCHITECTURE simple OF OrGate7 IS
BEGIN
   b <= a6 OR a5 OR a4 OR a3 OR a2 OR a1 OR a0 AFTER 7 NS; -- delay = 7 ns
END ARCHITECTURE simple;
```

VI. Xnor Gate:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY XnorGate IS
            PORT (a,b: IN STD_LOGIC;
            c: OUT STD_LOGIC);
END ENTITY XnorGate;
ARCHITECTURE simple OF XnorGate IS
BEGIN
   c <= a XNOR b AFTER 9 NS; -- delay = 9 ns
END ARCHITECTURE simple;
```

VII. Xor Gate:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-- Two inputs for XOR gate
ENTITY XorGate IS
            PORT(a,b: IN STD_LOGIC;
            c: OUT STD_LOGIC);
END ENTITY XorGate;

ARCHITECTURE simple OF XorGate IS
BEGIN
   c <= a XOR b AFTER 12 NS; -- delay = 12 ns
END ARCHITECTURE simple;
-- Three inputs for XOR gate
ENTITY XorGate3 IS
            PORT (a,b,c: IN STD_LOGIC;
            d: OUT STD_LOGIC);
```

```
END ENTITY XorGate3;

ARCHITECTURE simple OF XorGate3 IS
BEGIN
  d <= a XOR b XOR c AFTER 12 NS; -- delay = 12 ns
END ARCHITECTURE simple;
```

5.2 The code of full adder circuit:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
-- One Bit Full Adder Circuit
ENTITY FullAdder IS
  PORT ( x, y, cin: IN STD_LOGIC;   -- Three inputs
              sum, cout: OUT STD_LOGIC);     -- Two outputs (Sum, Carry)
END ENTITY FullAdder;

ARCHITECTURE simple OF FullAdder IS
  SIGNAL n: STD_LOGIC_VECTOR (2 DOWNTO 0);
BEGIN
        g1: ENTITY work.XorGate3(simple) PORT MAP (x, y, cin, sum); -- sum = x XOR y XOR cin
        g2: ENTITY work.AndGate2(simple) PORT MAP (x, y, n(2));
        g3: ENTITY work.AndGate2(simple) PORT MAP (x, cin, n(1));
        g4: ENTITY work.AndGate2(simple) PORT MAP (y, cin, n(0));
        g5: ENTITY work.OrGate3(simple) PORT MAP (n(2), n(1), n(0), cout); -- cout = (x.y) + (x.cin) + (y.cin)
END ARCHITECTURE simple;
```

NOTE: a test bench for the Full Adder Circuit:

```
ENTITY MyTestBenchFullAdder IS
END ENTITY MyTestBenchFullAdder;

ARCHITECTURE testFullAdder OF MyTestBenchFullAdder IS
 SIGNAL A, B, Cin, Sum, Cout: STD_LOGIC;
BEGIN
 G1: ENTITY work.FullAdder(simple) PORT MAP (A, B, Cin, Sum, Cout);
PROCESS
         BEGIN
            A <= '0';
            B <= '1';
             Cin <= '0';
            wait for 28 ns;
            A <= '1';
            B <= '1';
```

```
                    Cin <= '0';
                     wait for 28 ns;
                      A <= '0';
                     B <= '0';
                    Cin <= '1';
                     wait for 28 ns;
                    A <= '1';
                    B <= '0';
                    Cin <= '1';
                     wait for 28 ns;
                     A <= '0';
                      B <= '1';
                    Cin <= '1';
                     wait for 28 ns;
                  END PROCESS;
             END ARCHITECTURE testFullAdder;
```

5.3 The code of 8-Bit Ripple Circuit:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
 -- ripple circuit (Adder/subtractor)
ENTITY rippleCircuit IS
     PORT (A, B: IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- Two inputs from 8 bits (A, B)
       cin: IN STD_LOGIC; -- Cin input from 1 bit (0/1)
       sum: OUT STD_LOGIC_VECTOR(7 DOWNTO 0); -- result output from 8 bits
           v: OUT STD_LOGIC); -- Overflow output from 1 bit
END ENTITY rippleCircuit;

ARCHITECTURE simple OF rippleCircuit IS
  SIGNAL carry: STD_LOGIC_VECTOR(8 DOWNTO 0);
 SIGNAL XorB: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
  carry(0) <= cin;
     gen1: FOR i IN 0 TO 7 GENERATE
       g1: ENTITY work.XorGate(simple) PORT MAP (B(i), cin, XorB(i));
   END GENERATE gen1;
  gen2: FOR i IN 0 TO 7 GENERATE
       g2:  ENTITY work.FullAdder(simple) PORT MAP (A(i),XorB(i),carry(i),sum(i),carry(i+1));
   END GENERATE gen2;
      g3: ENTITY work.XorGate(simple) PORT MAP (carry(8), carry(7), v);
END ARCHITECTURE simple;
```

5.4 The code of 7-Bits Magnitude comparator:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
-- 7-Bits Magnitude comparator
ENTITY SevenBitComparator IS
        PORT ( A, B: IN   STD_LOGIC_VECTOR(6 DOWNTO 0);
            AEqualsB, AGreaterThanB, ALessThanB : OUT  STD_LOGIC );
END ENTITY SevenBitComparator;

ARCHITECTURE simple OF SevenBitComparator IS
  SIGNAL SignalOfInverterA, SignalOfInverterB, E, S, L: STD_LOGIC_VECTOR (6 DOWNTO 0);
        SIGNAL n, m : STD_LOGIC_VECTOR (5 DOWNTO 0);
BEGIN
  gen1: FOR i IN 6 DOWNTO 0 GENERATE
          g1:  ENTITY work.InverterGate(simple) PORT MAP (A(i), SignalOfInverterA(i));
   END GENERATE gen1;
         gen2: FOR i IN 6 DOWNTO 0 GENERATE
           g2:  ENTITY work.InverterGate(simple) PORT MAP (B(i), SignalOfInverterB(i));
   END GENERATE gen2;
         gen3: FOR i IN 6 DOWNTO 0 GENERATE
           g3:  ENTITY work.AndGate2(simple) PORT MAP (SignalOfInverterA(i), B(i), S(i));
   END GENERATE gen3;
         gen4: FOR i IN 6 DOWNTO 0 GENERATE
           g4:  ENTITY work.AndGate2(simple) PORT MAP (A(i), SignalOfInverterB(i), L(i));
   END GENERATE gen4;
         gen5: FOR i IN 6 DOWNTO 0 GENERATE
           g5:  ENTITY work.NorGate(simple) PORT MAP (L(i), S(i), E(i));
   END GENERATE gen5;
         -- To find the value of F1 (A=B)
         g6: ENTITY work.AndGate7(simple) PORT MAP (E(6), E(5), E(4), E(3), E(2), E(1), E(0), AEqualsB);
        -- To find the value of F2 (A>B)
         g7: ENTITY work.AndGate2(simple) PORT MAP (E(6), L(5), n(5));
         g8: ENTITY work.AndGate3(simple) PORT MAP (E(6), E(5), L(4), n(4));
         g9: ENTITY work.AndGate4(simple) PORT MAP (E(6), E(5), E(4), L(3), n(3));
         g10: ENTITY work.AndGate5(simple) PORT MAP (E(6), E(5), E(4), E(3), L(2), n(2));
        g11: ENTITY work.AndGate6(simple) PORT MAP (E(6), E(5), E(4), E(3), E(2), L(1), n(1));
         g12: ENTITY work.AndGate7(simple) PORT MAP (E(6), E(5), E(4), E(3), E(2), E(1), L(0), n(0));
         g13: ENTITY work.OrGate7(simple) PORT MAP (L(6), n(5), n(4), n(3), n(2), n(1), n(0), AGreaterThanB);
        -- To find the value of F3 (A<B)
        g14: ENTITY work.AndGate2(simple) PORT MAP (E(6), S(5), m(5));
        g15: ENTITY work.AndGate3(simple) PORT MAP (E(6), E(5), S(4), m(4));
        g16: ENTITY work.AndGate4(simple) PORT MAP (E(6), E(5), E(4), S(3), m(3));
        g17: ENTITY work.AndGate5(simple) PORT MAP (E(6), E(5), E(4), E(3), S(2), m(2));
       g18: ENTITY work.AndGate6(simple) PORT MAP (E(6), E(5), E(4), E(3), E(2), S(1), m(1));
        g19: ENTITY work.AndGate7(simple) PORT MAP (E(6), E(5), E(4), E(3), E(2), E(1), S(0), m(0));
        g20: ENTITY work.OrGate7(simple) PORT MAP (S(6), m(5), m(4), m(3), m(2), m(1), m(0), ALessThanB);
END ARCHITECTURE simple;
```

__NOTE:__ **a test bench for the 7-Bits Magnitude comparator:**

```
ENTITY MyTestSevenComp IS
END ENTITY MyTestSevenComp;

ARCHITECTURE test OF MyTestSevenComp IS
  SIGNAL A, B: STD_LOGIC_VECTOR (6 DOWNTO 0);
  SIGNAL E, G, L: STD_LOGIC;
BEGIN
 G1: ENTITY work.SevenBitComparator(simple) PORT MAP (A, B, E, G, L);
PROCESS
        BEGIN
                A <= "1111111";
                 B <= "1010101";
                wait for 30 ns;
                A <= "0011001";
               B <= "1100110";
                wait for 30 ns;
               A <= "1000100";
               B <= "0000000";
               wait for 30 ns;
                A <= "0101010";
               B <= "0011000";
                wait for 30 ns;
        END PROCESS;
END ARCHITECTURE test;
```

6. Appendix 2:

6.1 Simulation for full adder circuit: We notice in the full adder circuit that the sum result appears 12 nanoseconds after one or more of the input values has changed, because the XOR gate was used, which has a delay of 12 ns, while the result of the carry needed 14 nanoseconds because it depends on the output of the AND gate, which has a delay of 7 ns and OR gate that has a delay of 7 ns as well. Therefore, in the beginning, there will be no values for the outputs until after 12 ns have passed, the sum value appears, and after 14 ns, the value of the carry appears, as we note that there are glitches in outputs, as shown in Figure 5-1:

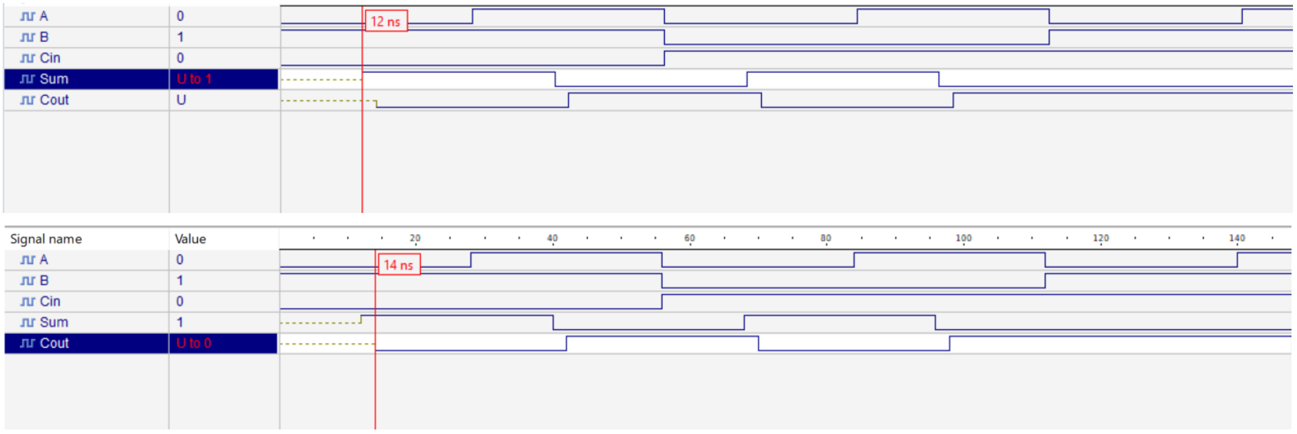


Figure 5-1: Simulation for full adder circuit

6.2 Simulation for seven bits magnitude comparator: As mentioned previously, the result does not appear directly because it depends on the delay time of each gate used in the magnitude comparator, as shown in Figure 5-2:

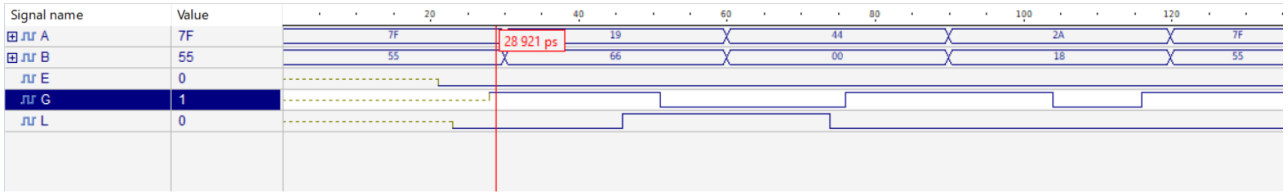


Figure 5-2: Simulation for 7-Bits magnitude comparator

6.3 Clock for stage 1: To find the value of delay time of the clock in the first stage through the experiment A="FF" and B="FF", then we noticed that it took 134 ns for the result to appear correctly, so it was chosen to change the value of the clock every 268 ns, the following Figure 5-3 shows the result of the simulation:

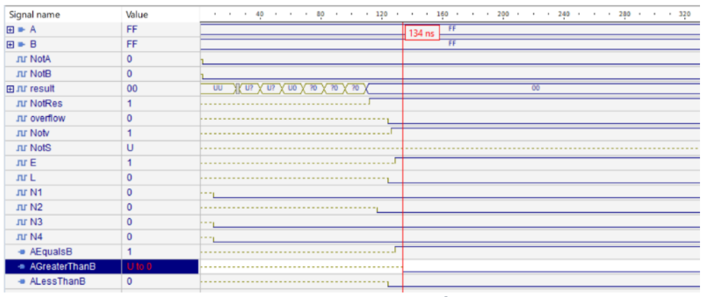
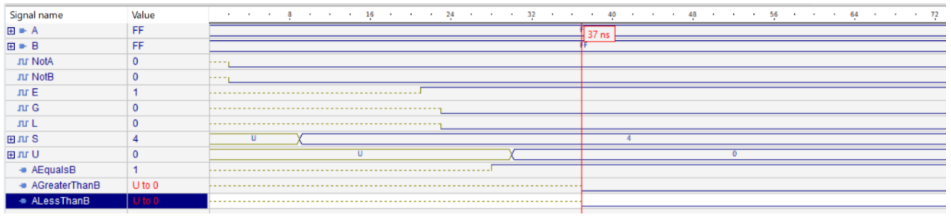


Figure 5-3: Clock for stage 1

6.4 Clock for stage 2: *To find the value of delay time of the clock in the second stage through the experiment A="FF" and B="FF", then we noticed that it took 37 ns for the result to appear correctly, so it was chosen to change the value of the clock every 74 ns, the following Figure 5-4 shows the result of the simulation:*



Signal name	Value
A	FF
B	FF
NotA	0
NotB	0
E	1
G	0
L	0
S	4
U	0
AEqualsB	1
AGreaterThanB	U to 0
ALessThanB	U to 0

Figure 5-4: Clock for stage 2

7. References:

1. "Basic Logic Gates," 12 Dec 2021. [Online]. Available: https://www.eduhk.hk/has/phys/de/lg/basic.htm.
2. "Full Adder," 15 Dec 2021. [Online]. Available: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.electrically4u.com%2Fhalf-adder-and-full-adder%2F&psig=AOvVaw0h5Tg9kVt2Yc7bOFD1uEDV&ust=1640629950450000&source=images&cd=vfe&ved=0CAsQjRxqFwoTCLC944uNgvUCFQAAAAAdAAAAABAD.
3. "4-bit binary Adder-Subtractor," 12 Dec 2021. [Online]. Available: https://www.geeksforgeeks.org/4-bit-binary-adder-subtractor/.
4. "carry look ahead adder," 14 Dec 2021. [Online]. Available: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.semanticscholar.org%2Fpaper%2FA-novel-implementation-of-4-bit-carry-look-ahead-Miao-Li%2Ff2e00bed8b4a1b6700b83459a9c27ef1ccd37d87&psig=AOvVaw3ufXDmzJ6wFPdFWGHEwTYP&ust=1640629617405000&source=images&cd.
5. "Magnitude Comparator," 16 Dec 2021. [Online]. Available: https://esrd2014.blogspot.com/p/8-bit-magnitude-comparator.html.
6. 23 Dec 2021. [Online]. Available: https://online.visual-paradigm.com/diagrams/templates/logic-diagram/products-of-sums/?fbclid=IwAR0ybve_hlh1YPAqzGQ6fhyqRvtgjLKXYP1j6s3M8z3LgHYumyTvesvj9kM
7. "Carry-lookahead adder," 14 Dec 2021. [Online]. Available: https://en.wikipedia.org/wiki/Carry-lookahead_adder.
8. "zoom meeting to explain project," 7 Dec 2021. [Online].