جامعة بيرزيت

BIRZEIT UNIVERSITY

Faculty of Engineering & Technology

Electrical & Computer Engineering Department

ADVANCED DIGITAL SYSTEMS DESIGN
ENCS 3310

Project Report

Student's Name: Tariq Odeh

Student's No: 1190699

Sec: 2

Instructor: Dr. Abdallatif Abuissa

Date: 20th December 2021

Table of Contents

List Of Figures

1. Introduction

In this project, we will implement an 8-bit Comparator for signed 2's complement representation numbers structurally by using several basic gates and entities with certain delays, in two stages. And then we want verification the two stages by creating a test generator that generates all correct possibilities for the device and compared them with the resulting values from stages. The first stage It will implement by using ripple adder-subtractor or look ahead and in second stage by using Magnitude comparator and the sign bit (we will talk about these circuits in general), We will calculate the duration of delay for each stage to use in testing the outputs.

Using the basic gates, we will build the important circuits that we will use in several stages, such as full adder, n-bit full adder. And we will focus on implementing these circuits in generic which take n-bit, in addition to focusing on building circuits with the least possible delay time through the use of NAND gates, and we will build circuits in several ways structurally, which makes this project more flexible and faster and the user can modify the design. Finally, we'll simulate the project and test all of the findings and outputs in both stages using (Aldec Active-HDL Student Edition).

2. Theoretical overview

2.1. Ripple Adder/Subtractor

A ripple adder-subtractor is a special type of circuit that is used to perform both operations addition and subtraction. The process to be used depends on the value of the control signal M. The ripple adder-subtractor is one of the most important components of the arithmetic logical unit, and it is called ripple carry because each carry bit is rippled to the next stage. The circuit consists of n full adders since we are performing operation on n-bit numbers.

As shown in the figure below, the first full adder has three input cin, A0, and B0 XOR M. and has two outputs S0 and carry Cout. If the value of control line is M = 1, the output of B0 XOR 1= B0', and the output of full adder is S0 = A0+B0' and carry Cout, which is the 2's complement for two numbers A and B. This suggests that when M=1, the operation being performed on the four-bit numbers is subtraction.

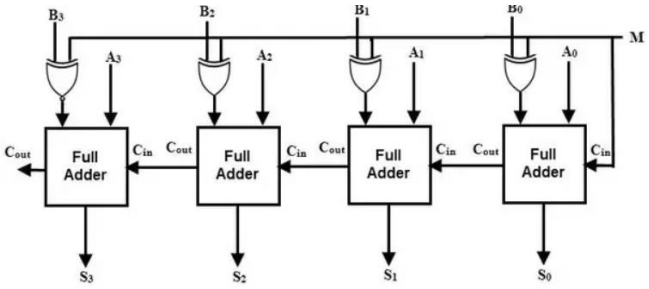


Figure 1: Ripple Carry Adder/Subtractor

2.2. Look-ahead Adder/Subtractor

Each adder block in a ripple adder-subtractor waits for the carry from the preceding block. As a result, unless the input carry is known, it is impossible to produce the sum and carry of any block. As a result, there will be a significant time delay due to carry propagation delay. The look-ahead adder/subtractor minimizes the propagation latency by using more complicated hardware, it is a faster parallel adder, but it is also more expensive. It takes use of the fact that each bit location in the addition may be used to determine whether a carry will be created at that bit or propagated via that bit.

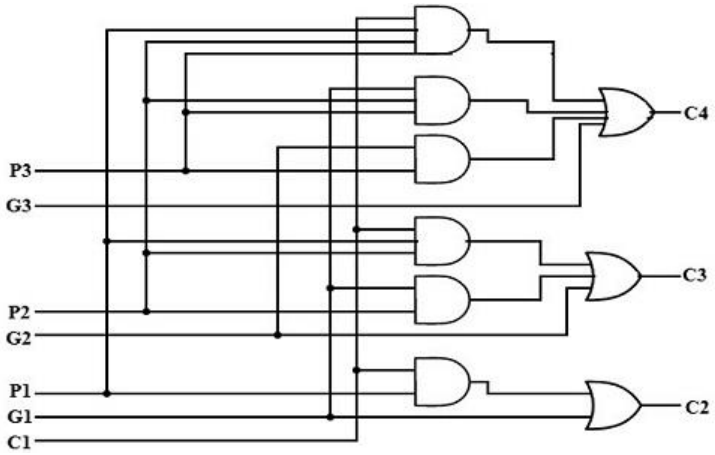


Figure 2: Carry Look-ahead adder

2.3. Magnitude Comparator

A magnitude digital Comparator is a combinational circuit that compares two digital or binary numbers (say, A and B) to determine whether one is equal to, less than, or greater than the other. Digital or binary comparators are made up of standard AND, NOR, and NOT gates that compare digital signals at their input terminals and output a result dependent on the state of those inputs. Three binary variables show the outcome of the comparison: A=B, A>B, or A<B. Below is a block diagram of an n-bit comparator, which compares two n-bit integers and generates their relation between themselves.

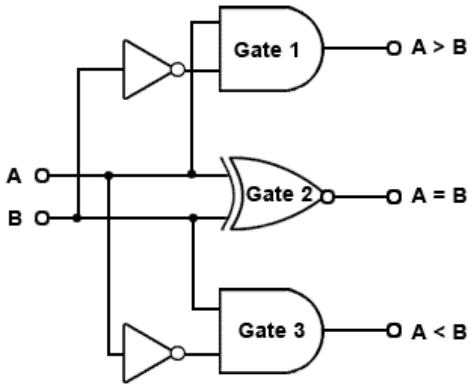


Figure 3: 1-Bit Magnitude Comparator

3. Design philosophy

3.1. Basic Gates

To start building the stages, we need to implement the basic gates. As shown in appendix (Code of basic gates), we built the gates in generic (N-bit) with one PORT, and (2-bit) in two PORTS, which will make it easier to implement the stages and make the design more flexible.

3.2. Stage 1

- **Half and full adder**

We implemented half adder and the full adder to use later in building ripple full adder. As shown in appendix (Code of half and full adder), the full adder can be built in three ways: first by using AND gates, XOR gates and OR gates, second by using half adder and third by using NAND gate. We used NAND gates to decrease the time of delay.

- **N-bit Ripple Adder/Subtractor**

We used the full adder library to implement the code for n-bit Ripple Adder /Subtractor as shown in appendix (Code of N-bit Ripple Adder/Subtractor). In this design we can use it as adder by making Cin = 0 and we can use it as subtractor by making Cin = 1. In this stage we want to use the subtractor to find 2's complement of B, so we making Cin = 1. We tried to build Look-ahead Adder/Subtractor, as it is faster in performance, but we had difficulty building it properly structurally.

- **N-bit Comparator**

By using the n-bit ripple adder/subtractor library and the basic gates, we built n-Bit Comparator for signed 2's complement representation numbers, that use the output of n-bit ripple adder/subtractor (overflow, Cout, SUM, and the last bit of SUM) to determine the comparison output, as shown in appendix (Code of N-bit Comparator). In the figure below show the circuit which takes the output of n-bit ripple adder/subtractor and determines the result of the comparison.

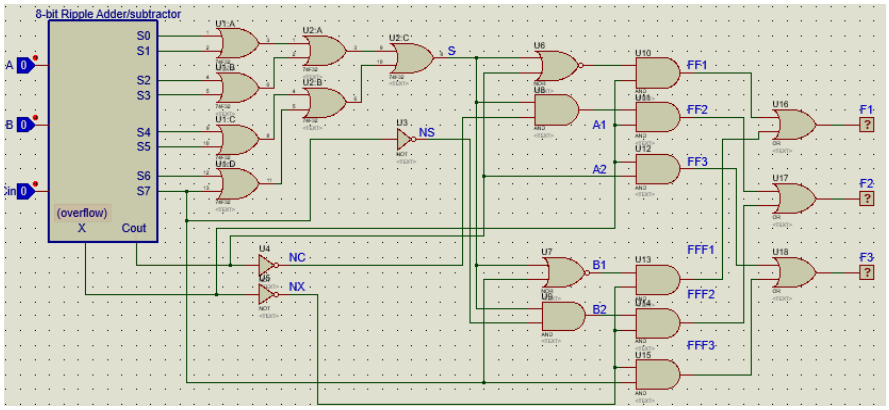


Figure 4: 8-Bit Comparator circuit

- **Stage 1 with ripple adder/subtractor**

Here we built the stage 1 by using n-Bit comparator library and making it 8-bit. From here we can easily modify the design for any number of bits we need, as shown in appendix (Code of Stage 1 with ripple adder/subtractor).

3.3. Stage 2

- **1-Bit Magnitude comparator**

We implemented 1-bit magnitude comparator by using basic gates to use later in building 4-bit magnitude comparator. As shown in appendix (Code of 1-bit magnitude comparator) the 1-bit magnitude comparator by using AND gates, XNOR gates and INVERTER gates.

- **4-Bit Magnitude comparator**

We implemented 4-bit magnitude comparator by using 1-bit magnitude comparator library and basic gates, to use later in building 8-bit magnitude comparator. As shown in appendix (Code of 4-bit magnitude comparator) we find the possibilities for greater, less and equal output using OR/AND gates, we can see the circuit in appendix (7.9).

- **8-Bit Magnitude comparator**

We implemented 8-bit magnitude comparator by using two of 4-bit magnitude comparator library and basic gates, to use later in building stage 2. As shown in appendix (Code of 8-bit magnitude comparator) we find the possibilities for greater, less and equal output using OR/AND gates.

- **Stage 2 with magnitude comparator**

We implemented stage 2 by using 8-bit magnitude comparator library and basic gates. As shown in appendix (Code of Stage 2 with Magnitude comparator), we used XOR gate to deal with sign bit and to find to output.

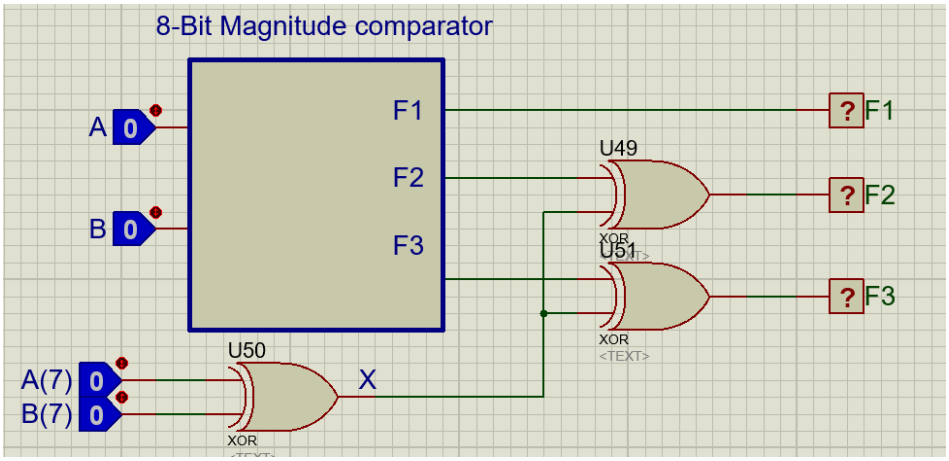


Figure 5: Stage 2 with Magnitude comparator

3.4. Test Generator

In this generator we made two nested loop to reach all possible values of A and B and passed to output when the clock input rises, and at each value of A and B will resulting the correct output in behavioral logic as shown in appendix (**Code of Test Generator**).

3.5. Result Analyze

The result analyzer makes sure that the output from our system equals the correct which we calculated in Test Generator when the clock input rises, and if not equals print an error by using assert component and report to typing the error message. As shown in appendix (**Code of Result Analyze**).

3.6. Verification Stage 1 & Stage 2

In verification we want to automatically decide if the design works by checks the answers for correctness and notifies the user of any error, so in verification we built the whole system with a test generator, result analyzer and the stage we want. The generator and analyzer will have the same clock signal, after a specified amount of time, the clock signal reverses, and the test generator alters the A and B signals, sending the right output to the result analyzer. The outputs A and B are sent to the system, which generates an output. This output is then sent to the result analyzer, which determines whether or not the output is valid based on the test generator's proper result. The figure below shown the block diagram for design and we can see the code in appendix (**Code of Verification Stage 1 & Stage 2**).

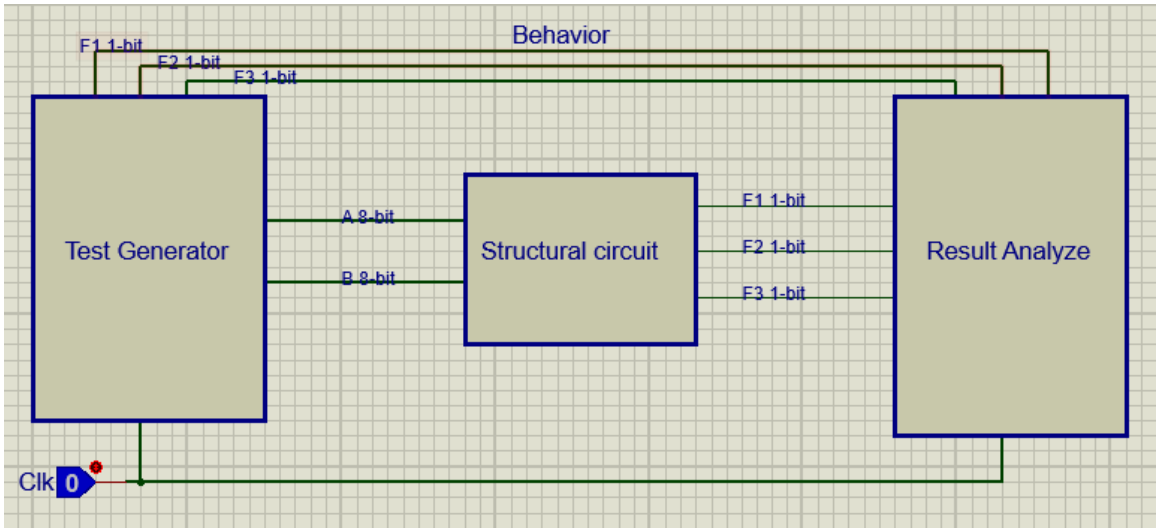


Figure 6: block diagram of the design

4. Simulation and results

The full Code with comments shown in appendix (**Full Code with comments**).

4.1. Stage 1 with ripple adder/subtractor

From the screenshots below for the simulation shows the results of the structurally circuit that we built and the results of test generator (behavioral), and shows the difference between the results. We can see that the results of the structurally circuit that we built have some glitches which we want to show to see the difference between the results, and through it, we can set the clock.

And based on simulation and the glitches that have appeared, we found that when the value of the **Clk = 133 ns**, we will get matched results without any errors, since if we take a lower value from it, an error will happen.

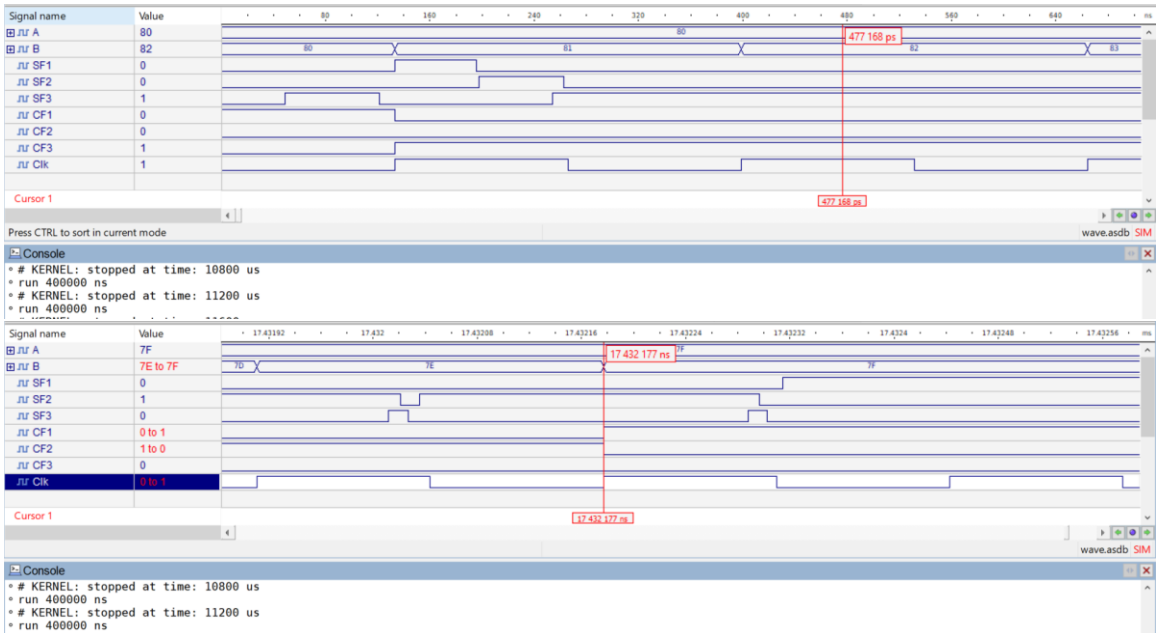


Figure 7: Stage 1 simulation when Clk = 133 ns

We notice in Figure 8, that we made Clk less than 133 ns, which showed some errors.

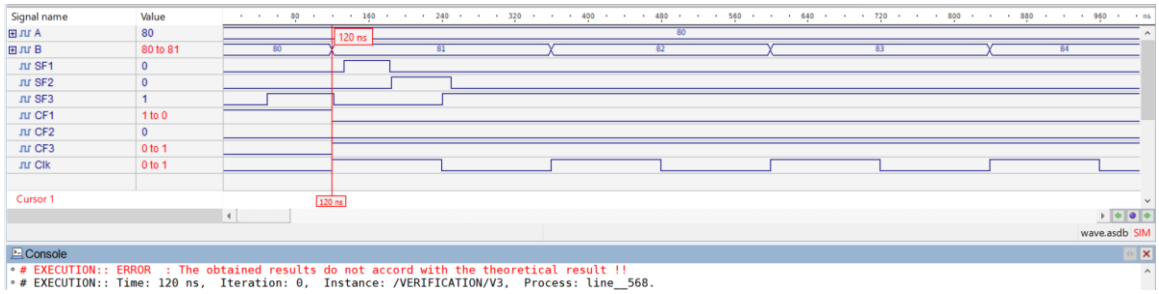


Figure 8: Stage 1 simulation when Clk less than 133 ns

4.2. Stage 2 with magnitude comparator

From the screenshots below for the simulation shows the results of the structurally circuit that we built and the results of test generator (behavioral), and shows the difference between the results. We can see that the results of the structurally circuit that we built have some glitches which we want to show to see the difference between the results, and through it, we can set the clock.

And based on simulation and the glitches that have appeared, we found that when the value of the **Clk = 25 ns**, we will get matched results without any errors, since if we take a lower value from it, an error will happen.

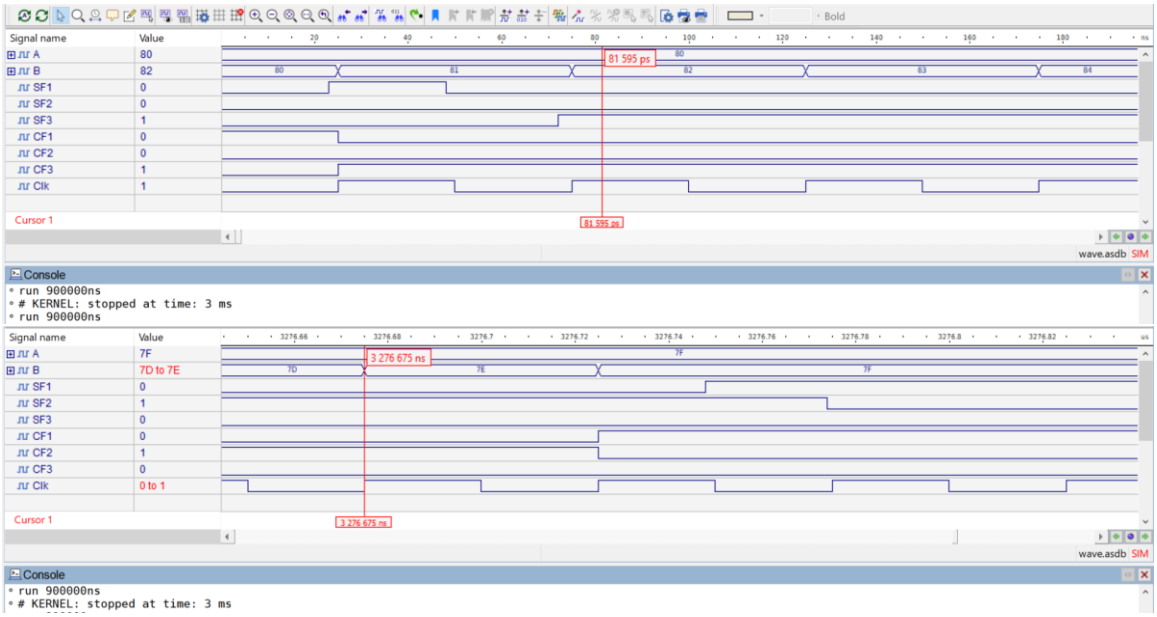


Figure 9: Stage 2 simulation when Clk = 25 ns

We notice in Figure 10, that we made Clk less than 25 ns, which showed some errors.

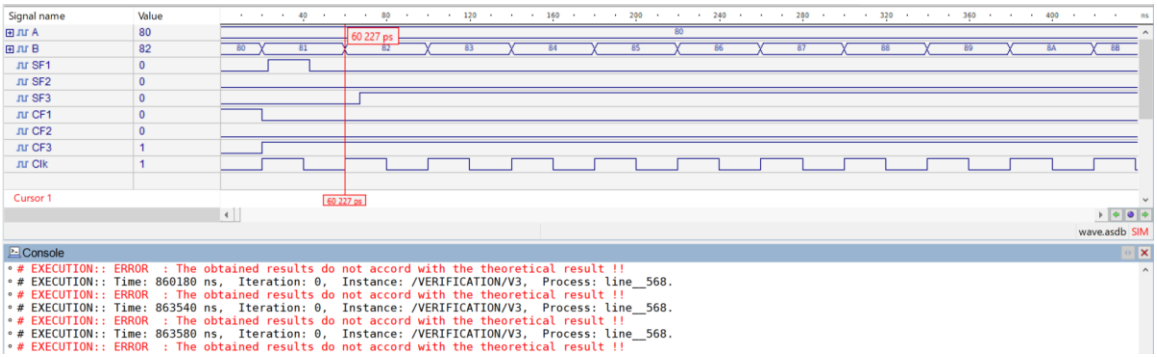


Figure 10: Stage 2 simulation when Clk less than 25 ns

5. Conclusion and future works

In conclusion, we built two stages structurally in this project and we successfully design an 8-bit Comparator for signed 2's complement and then write a functioning verification method. The results of the preceding methods agree with the theoretical outcomes, and we found that the built-in test is helpful in ensuring that the outputs are accurate. Furthermore, we infer that we can build large systems with smaller ones. We learned a lot about types of circuits like Ripple Adder/subtractor, Look-ahead Adder/subtractor, and magnitude (unsigned) comparator. And by creating entities in behavioral and structural logics, making a delay, testing the systems, and learning how to create commands like how to print an error, we grew more accustomed with VHDL.

During the project implementation process, our focus was on reaching the best possible design and the best performance. It was necessary to expand our knowledge and search for several topics and ways to improve performance and to facilitate our work and ensure access to a better design. On the design side, we focused on the clarity of the code, the clear structure, and the ability to modify the design, as most of the blocks were built in a general way, which facilitates the modification and development of the design, and in terms of performance, we tried to use the gates with the least delay time, such as NAND and NOR.

We faced many challenges and problems during the implementation of this project. We faced is how to build circuits in the general way, and also how to reach a correct design through which a correct output can be obtained. We had problems with how to build the lookahead carry structurally, so we tried over and over and finally had to use behavioral in the build so we worked on ripple adder/subtractor. Most of the challenges and problems were solved through research and experience.

In conclusion, it is necessary to look to the future of the project in order to be able to solve any defect that appears in the design and to ensure continuous development. We are looking forward to improving the delay time of the system by using the Look-ahead Adder/subtractor as it is faster in performance in addition to using gates with less delay time, and we will convert all the design to the general format to reach the high flexibility in the design. We will show our project to many experts to try and and we will take feedback and build on it to improve the design.

6. References

[1] 4-bit binary adder-subtractor. GeeksforGeeks. (2021, October 21). Retrieved December 19, 2021, from https://www.geeksforgeeks.org/4-bit-binary-adder-subtractor/

[2] Dynamic 16-bit carry- lookaheadadder/subtractor. (2020, April 45). Retrieved December 19, 2021, from http://people.ee.duke.edu/~jmorizio/ece261/F07/projects/CarryLookahead.pdf

[3] kishorekashyap. (2017, November 21). *Magnitude comparator 1 bit, 2 bit, 3 bit, 4 bit*. YouTube. Retrieved December 19, 2021, from https://www.youtube.com/watch?v=7-MbS-wKWuw

[4] Performance analysis of different bit carry look ahead ... (2020, June 4). Retrieved December 19, 2021, from https://www.researchgate.net/publication/319236867_Performance_Analysis_of_Different_Bit_Carry_Look_Ahead_Adder_Using_VHDL_Environment

[5] Sas, W. (2021, August 4). *Two's complement calculator*. Omni Calculator. Retrieved December 19, 2021, from https://www.omnicalculator.com/math/twos-complement

[6] uxnzdwsiuxnzdwsi 122 bronze badges, Giorgi TsiklauriGiorgi Tsiklauri 7, John AspinallJohn Aspinall 933 bronze badges, mhawkemhawke 78.1k99 gold badges103103 silver badges133133 bronze badges, & WJSWJS 26.2k44 gold badges1919 silver badges3434 bronze badges. (1969, April 1). *Two's complement binary range for 8-bit*. Stack Overflow. Retrieved December 19, 2021, from https://stackoverflow.com/questions/66538718/twos-complement-binary-range-for-8-bit

[7] YouTube. (2018, January 24). *Carry ripple adder and subtractor circuits*. YouTube. Retrieved December 19, 2021, from https://www.youtube.com/watch?v=-21zUqEKeqk

7. Appendix

7.1. Code of Basic Gates

```
12  ----------------------------------------------------
13  --              INVERTER GATE (1 PORT)             --
14  ----------------------------------------------------
15  LIBRARY IEEE;
16  USE IEEE.STD_LOGIC_1164.ALL;
17
18  ENTITY NOT_1P IS
19      PORT(A: IN STD_LOGIC:= '0';      --1 bit Input
20           F: OUT STD_LOGIC:= '0');    --1 bit Output
21  END ENTITY NOT_1P;
22
23  ARCHITECTURE STRUCT_NOT_1P OF NOT_1P IS
24      BEGIN
25          F <= NOT A AFTER 2 NS;       --2 ns delay
26  END ARCHITECTURE STRUCT_NOT_1P;
27
28
29  ----------------------------------------------------
30  --               NAND GATE (N-BIT)                --
31  ----------------------------------------------------
32  LIBRARY IEEE;
33  USE IEEE.STD_LOGIC_1164.ALL;
34
35  ENTITY NAND_NBIT IS
36      GENERIC ( N : POSITIVE := 2);          --N bit Input
37      PORT(A: IN STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";
38           F: OUT STD_LOGIC:= '0');        --1 bit Output
39  END ENTITY NAND_NBIT;
40
41  ARCHITECTURE STRUCT_NAND_NBIT OF NAND_NBIT IS
42      SIGNAL TEMP : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
43      BEGIN
44          TEMP(0) <= A(0);
45          FLOOP: FOR i IN 1 TO (N-1) GENERATE
46                  TEMP(i) <= TEMP(i-1) AND A(i);
47          END GENERATE FLOOP;
48          F <= NOT(TEMP(N-1)) AFTER 5 NS; --NAND = NOT(AND)
49  END ARCHITECTURE STRUCT_NAND_NBIT;       --5 ns delay
50
51
```

```
52  ----------------------------------------------------
53  --              NAND GATE (2 PORTS)               --
54  ----------------------------------------------------
55  LIBRARY IEEE;
56  USE IEEE.STD_LOGIC_1164.ALL;
57
58  ENTITY NAND_2P IS
59      PORT(A,B: IN STD_LOGIC:= '0';  --2 bit Input
60           F: OUT STD_LOGIC:= '0'); --1 bit Output
61  END ENTITY NAND_2P;
62
63  ARCHITECTURE STRUCT_NAND_2P OF NAND_2P IS
64      BEGIN
65          F <= A NAND B AFTER 5 NS;        --5 ns delay
66       -- F <= NOT(A AND B) AFTER 5 NS;
67  END ARCHITECTURE STRUCT_NAND_2P;
68
69
70  ----------------------------------------------------
71  --               NOR GATE (N-BIT)                 --
72  ----------------------------------------------------
73  LIBRARY IEEE;
74  USE IEEE.STD_LOGIC_1164.ALL;
75
76  ENTITY NOR_NBIT IS
77      GENERIC ( N : POSITIVE := 2);  --N bit Input
78      PORT(A: IN STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";
79           F: OUT STD_LOGIC:='0');             --1 bit Output
80  END ENTITY NOR_NBIT;
81
82  ARCHITECTURE STRUCT_NOR_NBIT OF NOR_NBIT IS
83      SIGNAL TEMP : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
84      BEGIN
85          TEMP(0) <= A(0);
86          FLOOP: FOR i IN 1 TO (N-1) GENERATE
87                  TEMP(i) <= TEMP(i-1) OR A(i);
88          END GENERATE FLOOP;
89          F <= NOT(TEMP(N-1)) AFTER 5 NS; --NOR = NOT(OR)
90  END ARCHITECTURE STRUCT_NOR_NBIT;   --5 ns delay
91
92
```

```vhdl
----------------------------------------------------
--              NOR GATE (2 PORTS)                --
----------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY NOR_2P IS
    PORT(A,B: IN STD_LOGIC:= '0';    --2 bit Input
           F: OUT STD_LOGIC:= '0'); --1 bit Output
END ENTITY NOR_2P;

ARCHITECTURE STRUCT_NOR_2P OF NOR_2P IS
    BEGIN
        F <= A NOR B AFTER 5 NS;     --5 ns delay
     -- F <= NOT(A OR B) AFTER 5 NS;
END ARCHITECTURE STRUCT_NOR_2P;


----------------------------------------------------
--               AND GATE (N-BIT)                 --
----------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY AND_NBIT IS
    GENERIC ( N : POSITIVE := 2);    --N bit Input
    PORT(A: IN STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";
         F: OUT STD_LOGIC:= '0');        --1 bit Output
END ENTITY AND_NBIT;

ARCHITECTURE STRUCT_AND_NBIT OF AND_NBIT IS
    SIGNAL TEMP : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
    BEGIN
        TEMP(0) <= A(0);
        FLOOP: FOR i IN 1 TO (N-1) GENERATE
                TEMP(i) <= TEMP(i-1) AND A(i);
        END GENERATE FLOOP;
        F <= TEMP(N-1) AFTER 7 NS;   --7 ns delay
END ARCHITECTURE STRUCT_AND_NBIT;

----------------------------------------------------
--              AND GATE (2 PORTS)                --
----------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY AND_2P IS
    PORT(A,B: IN STD_LOGIC:= '0';    --2 bit Input
           F: OUT STD_LOGIC:= '0'); --1 bit Output
END ENTITY AND_2P;

ARCHITECTURE STRUCT_AND_2P OF AND_2P IS
    BEGIN
        F <= A AND B AFTER 7 NS;     --7 ns delay
END ARCHITECTURE STRUCT_AND_2P;


----------------------------------------------------
--                OR GATE (N-BIT)                 --
----------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY OR_NBIT IS
    GENERIC ( N : POSITIVE := 2); --N bit Input
    PORT(A: IN STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";
         F: OUT STD_LOGIC:= '0');        --1 bit Output
END ENTITY OR_NBIT;

ARCHITECTURE STRUCT_OR_NBIT OF OR_NBIT IS
    SIGNAL TEMP : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
    BEGIN
        TEMP(0) <= A(0);
        FLOOP: FOR i IN 1 TO (N-1) GENERATE
                TEMP(i) <= TEMP(i-1) OR A(i);
        END GENERATE FLOOP;
        F <= TEMP(N-1) AFTER 7 NS;   --7 ns delay
END ARCHITECTURE STRUCT_OR_NBIT;
```

```
172  ----------------------------------------------------
173  --               OR GATE (2 PORTS)                --
174  ----------------------------------------------------
175  LIBRARY IEEE;
176  USE IEEE.STD_LOGIC_1164.ALL;
177
178  ENTITY OR_2P IS
179      PORT(A,B: IN STD_LOGIC:= '0';   --2 bit Input
180               F: OUT STD_LOGIC:= '0');     --1 bit Output
181  END ENTITY OR_2P;
182
183  ARCHITECTURE STRUCT_OR_2P OF OR_2P IS
184      BEGIN
185           F <= A OR B AFTER 7 NS;         --7 ns delay
186  END ARCHITECTURE STRUCT_OR_2P;
187
188
189  ----------------------------------------------------
190  --               XNOR GATE (2 PORTS)              --
191  ----------------------------------------------------
192  LIBRARY IEEE;
193  USE IEEE.STD_LOGIC_1164.ALL;
194
195  ENTITY XNOR_2P IS
196      PORT(A,B: IN STD_LOGIC:= '0';   --2 bit Input
197               F: OUT STD_LOGIC:= '0');     --1 bit Output
198  END ENTITY XNOR_2P;
199
200  ARCHITECTURE STRUCT_XNOR_2P OF XNOR_2P IS
201      BEGIN
202           F <= A XNOR B AFTER 9 NS;   --9 ns delay
203  END ARCHITECTURE STRUCT_XNOR_2P;
204
205
206  ----------------------------------------------------
207  --               XOR GATE (2 PORTS)               --
208  ----------------------------------------------------
209  LIBRARY IEEE;
210  USE IEEE.STD_LOGIC_1164.ALL;
211
212  ENTITY XOR_2P IS
213      PORT(A,B: IN STD_LOGIC:= '0'; --2 bit Input
214               F: OUT STD_LOGIC:= '0');     --1 bit Output
215  END ENTITY XOR_2P;
216
217  ARCHITECTURE STRUCT_XOR_2P OF XOR_2P IS
218      BEGIN
219           F <= A XOR B AFTER 12 NS;   --12 ns delay
220  END ARCHITECTURE STRUCT_XOR_2P;
221
222
```

7.2. Code of half and full adder

```vhdl
----------------------------------------------------
--                  Half Adder                    --
----------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY HA IS
    PORT(A,B: IN STD_LOGIC:= '0';            --Input
         Sum,Cout: OUT STD_LOGIC:= '0');     --Output
END ENTITY HA;

ARCHITECTURE STRUCT_HA OF HA IS
    SIGNAL X : STD_LOGIC_VECTOR (1 DOWNTO 0):= "00";
    BEGIN
        X <= A & B;
        H1: ENTITY WORK.XOR_2P(STRUCT_XOR_2P) PORT MAP (A,B,Sum);       --Sum = A XOR B
        H2: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(2) PORT MAP (X,Cout);     --Cout = A AND B
END ARCHITECTURE STRUCT_HA;
```

```vhdl
----------------------------------------------------
--                  Full Adder                    --
----------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY FAD IS
    PORT(A,B,Cin: IN STD_LOGIC:= '0';
         Sum,Cout: OUT STD_LOGIC:= '0');
END ENTITY FAD;

ARCHITECTURE STRUCT_FAD OF FAD IS
    SIGNAL N1,N2,N3,N4,N5,N6,N7: STD_LOGIC := '0';
    BEGIN
                    --First way to implement Full Adder Using XOR,AND,OR gates
        --F1: ENTITY WORK.XOR_2P(STRUCT_XOR_2P) PORT MAP(A,B,N1);           --SUM = A XOR B XOR Cin
        --F2: ENTITY WORK.XOR_2P(STRUCT_XOR_2P) PORT MAP(N1,Cin,Sum);       --Cout = Cin.(A XOR B) + A.B
        --F3: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP(A,B,N2);
        --F4: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP(N1,Cin,N3);
        --F5: ENTITY WORK.OR_2P(STRUCT_OR_2P) PORT MAP(N2,N3,Cout);

                    --Second way to implement Full Adder Using Half Adder
        --F1: ENTITY WORK.HA(STRUCT_HA) PORT MAP(A,B,N1,N2);
        --F2: ENTITY WORK.HA(STRUCT_HA) PORT MAP(N1,Cin,Sum,N3);
        --F3: ENTITY WORK.OR_2P(STRUCT_OR_2P) PORT MAP(N2,N3,Cout);

                    --Third way to implement Full Adder Using NAND gate (faster)
        F1: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(A,B,N1);
        F2: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N1,A,N2);
        F3: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N1,B,N3);
        F4: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N2,N3,N4);
        F5: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N4,Cin,N5);
        F6: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N5,N1,Cout);
        F7: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N5,Cin,N6);
        F8: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N5,N4,N7);
        F9: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N6,N7,Sum);
END ARCHITECTURE STRUCT_FAD;
```

7.3. Code of N-bit Ripple Adder/Subtractor

```
----------------------------------------------------
--           n-Bit Ripple Adder/subtractor        --
----------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY NBIT_ADDER IS
    GENERIC ( N : POSITIVE :=8);
    PORT(A,B: IN STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";
         Cin: IN STD_LOGIC :='1';                --Set Cin = 1 to find 2's complemant for B
         Sum: OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0):= "00000000";
        Cout, OV: OUT STD_LOGIC:= '0');    --OV (overflow)
END ENTITY NBIT_ADDER;

ARCHITECTURE STRUCT_RIPPLE OF NBIT_ADDER IS
    SIGNAL CARRY : STD_LOGIC_VECTOR (N DOWNTO 0):= "000000000";
    SIGNAL X :STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";  --to save 2's complemant of B
    BEGIN
        CARRY(0) <= Cin;
        Cout <= CARRY(N);
        R1: ENTITY WORK.XOR_2P(STRUCT_XOR_2P) PORT MAP (CARRY(N),CARRY(N-1),OV);
        FLOOP: FOR i IN 0 TO (N-1) GENERATE      --Loop to generate ripple Adder/subtractor
               R2: ENTITY WORK.XOR_2P(STRUCT_XOR_2P) PORT MAP (B(i),Cin,X(i));  --if Cin = 0 then ripple Adder work, if Cin = 1 ripple subtractor work
               R3: ENTITY WORK.FAD(STRUCT_FAD) PORT MAP (A(i),X(i),CARRY(i),Sum(i),CARRY(i+1));
        END GENERATE FLOOP;
END ARCHITECTURE STRUCT_RIPPLE;

```

7.4. Code of N-bit Comparator

```
----------------------------------------------------
----                n-Bit Comparator              --
----------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY NBIT_COMPARATOR IS
    GENERIC ( N : POSITIVE :=8);
    PORT(A,B: IN STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";  --Input
         F1,F2,F3: OUT STD_LOGIC:= '0');               --Output: F1(A=B), F2(A>B), and F3 (A<B).
END ENTITY NBIT_COMPARATOR;

ARCHITECTURE STRUCT_NBIT_COMPARATOR  OF NBIT_COMPARATOR IS
    SIGNAL SUM : STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";
    SIGNAL Cout,X,S : STD_LOGIC:= '0';       --X to check overflow, S to check if sum equal zero
    SIGNAL NC,NS,NX: STD_LOGIC:= '0';  --NC (not Cout) ,NS (not S(N-1)),NX (not Overflow)
    SIGNAL FF1,FF2,FF3: STD_LOGIC:= '0';  --to save output of F1,F2,F3 if there is overflow
    SIGNAL FFF1,FFF2,FFF3: STD_LOGIC:= '0';--to save output of F1,F2,F3 if there is not  overflow
    SIGNAL A1,A2,B1,B2: STD_LOGIC:= '0';

    BEGIN
        P1: ENTITY WORK.NBIT_ADDER(STRUCT_RIPPLE) GENERIC MAP(N) PORT MAP(A,B,'1',SUM,Cout,X);      --ripple subtractor (the output of overflow save in X)
        P3: ENTITY WORK.OR_NBIT(STRUCT_OR_NBIT) GENERIC MAP(N) PORT MAP(SUM,S);              --check if sum equal zero
        P4: ENTITY WORK.NOT_1P(STRUCT_NOT_1P)PORT MAP(Cout,NC);                              --NC (not Cout)
        P5: ENTITY WORK.NOT_1P(STRUCT_NOT_1P)PORT MAP(SUM(N-1),NS);                          --NS (not S(N-1))
        P6: ENTITY WORK.NOT_1P(STRUCT_NOT_1P)PORT MAP(X,NX);                                 --NX (not Overflow)

        X1: ENTITY WORK.NOR_2P(STRUCT_NOR_2P) PORT MAP (Cout,S,A1);      --find the value of F1,F2,F3 if there is overflow
        X2: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (NC,S,A2);
        X3: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (X,A1,FF1);
        X4: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (X,A2,FF2);
        X5: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (X,Cout,FF3);

        Y1: ENTITY WORK.NOR_2P(STRUCT_NOR_2P) PORT MAP (SUM(N-1),S,B1);  --find the value of F1,F2,F3 if there is not overflow
        Y2: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (NS,S,B2);
        Y3: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (NX,B1,FFF1);
        Y4: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (NX,B2,FFF2);
        Y5: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (NX,SUM(N-1),FFF3);

        Z1: ENTITY WORK.OR_2P(STRUCT_OR_2P) PORT MAP(FF1,FFF1,F1);       --final value of F1,F2,F3
        Z2: ENTITY WORK.OR_2P(STRUCT_OR_2P) PORT MAP(FF2,FFF2,F2);
        Z3: ENTITY WORK.OR_2P(STRUCT_OR_2P) PORT MAP(FF3,FFF3,F3);

END ARCHITECTURE STRUCT_NBIT_COMPARATOR;
```

7.5. Code of Stage 1 with ripple adder/subtractor

```
354
355  ----------------------------------------------------
356  ----      Stage_1 with ripple adder/subtractor      --
357  ----------------------------------------------------
358  LIBRARY IEEE;
359  USE IEEE.STD_LOGIC_1164.ALL;
360
361  ENTITY STAGE_1 IS
362      PORT(A,B: IN STD_LOGIC_VECTOR (7 DOWNTO 0):= "00000000";     --8-bit input
363           F1,F2,F3: OUT STD_LOGIC:='0');                          --Output: F1(A=B), F2(A>B), and F3 (A<B).
364  END ENTITY STAGE_1;
365
366  ARCHITECTURE STRUCT_STAGE_1  OF STAGE_1 IS
367      BEGIN
368          S1: ENTITY WORK.NBIT_COMPARATOR(STRUCT_NBIT_COMPARATOR) GENERIC MAP(8) PORT MAP(A,B,F1,F2,F3);
369  END ARCHITECTURE STRUCT_STAGE_1;
```

7.6. Code of 1-Bit Magnitude comparator

```
376  --           1-Bit Magnitude comparator          --
377  ----------------------------------------------------
378
379  LIBRARY IEEE;
380  USE IEEE.STD_LOGIC_1164.ALL;
381
382  ENTITY MAGNCOMP_1BIT IS
383      PORT(A,B: IN STD_LOGIC:= '0';          --1-Bit input
384           F1,F2,F3: OUT STD_LOGIC:= '0');   --Output: F1(A=B), F2(A>B), and F3 (A<B)
385  END ENTITY MAGNCOMP_1BIT;
386
387  ARCHITECTURE STRUCT_MAGNCOMP_1BIT OF MAGNCOMP_1BIT IS
388      SIGNAL NA,NB : STD_LOGIC:= '0';    --NA(not A) , NB(not B)
389      BEGIN
390          M1: ENTITY WORK.XNOR_2P(STRUCT_XNOR_2P)PORT MAP(A,B,F1);
391          M2: ENTITY WORK.NOT_1P(STRUCT_NOT_1P)PORT MAP(B,NB);
392          M3: ENTITY WORK.AND_2P(STRUCT_AND_2P)PORT MAP(A,NB,F2);
393          M4: ENTITY WORK.NOT_1P(STRUCT_NOT_1P)PORT MAP(A,NA);
394          M5: ENTITY WORK.AND_2P(STRUCT_AND_2P)PORT MAP(NA,B,F3);
395  END ARCHITECTURE STRUCT_MAGNCOMP_1BIT;
```

7.7. Code of 4-Bit Magnitude comparator

```
398  ----------------------------------------------------
399  --          4-Bit Magnitude comparator          --
400  ----------------------------------------------------
401  LIBRARY IEEE;
402  USE IEEE.STD_LOGIC_1164.ALL;
403
404  ENTITY MAGNCOMP_4BIT IS
405      PORT(A,B: IN STD_LOGIC_VECTOR (3 DOWNTO 0):= "0000";     --4-Bit input
406           F1,F2,F3: OUT STD_LOGIC:= '0');                  --Output: F1(A=B), F2(A>B), and F3 (A<B).
407  END ENTITY MAGNCOMP_4BIT;
408
409  ARCHITECTURE STRUCT_MAGNCOMP_4BIT OF MAGNCOMP_4BIT IS
410      SIGNAL E,G,L :STD_LOGIC_VECTOR (3 DOWNTO 0):= "0000";  --E(equal)A=B, G(greater)A>B, L(less)A<B
411      SIGNAL GT,LT :STD_LOGIC_VECTOR (3 DOWNTO 0):= "0000";
412      SIGNAL LL1,GG1 :STD_LOGIC_VECTOR (1 DOWNTO 0):= "00";
413      SIGNAL LL2,GG2 :STD_LOGIC_VECTOR (2 DOWNTO 0):= "000";
414      SIGNAL LL3,GG3 :STD_LOGIC_VECTOR (3 DOWNTO 0):= "0000";
415      SIGNAL X1,X2,X3,X4,X5,X6 :STD_LOGIC := '0';
416      BEGIN
417          FLOOP1: FOR i IN 0 TO (3) GENERATE     --loop to compare 4 bit by using 1-Bit Magnitude comparator
418              M1: ENTITY WORK.MAGNCOMP_1BIT(STRUCT_MAGNCOMP_1BIT) PORT MAP (A(i),B(i),E(i),G(i),L(i));
419          END GENERATE FLOOP1;
420
421          LL1 <= L(2)&E(3);  --To find out the possibilities for greater or less
422          GG1 <= G(2)&E(3);
423          LL2 <= L(1)&E(3)&E(2);
424          GG2 <= G(1)&E(3)&E(2);
425          LL3 <= L(0)&E(3)&E(2)&E(1);
426          GG3 <= G(0)&E(3)&E(2)&E(1);
427
428          M2: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(2) PORT MAP(LL1,X1);
429          M3: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(2) PORT MAP(GG1,X2);
430          M4: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(3) PORT MAP(LL2,X3);
431          M5: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(3) PORT MAP(GG2,X4);
432          M6: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(4) PORT MAP(LL3,X5);
433          M7: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(4) PORT MAP(GG3,X6);
434
435          GT <= G(3)&X2&X4&X6;  --to check if greater or less
436          LT <= L(3)&X1&X3&X5;
437
438          M8: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(4) PORT MAP(E,F1);
439          M9: ENTITY WORK.OR_NBIT(STRUCT_OR_NBIT) GENERIC MAP(4) PORT MAP(GT,F2);
440          M10: ENTITY WORK.OR_NBIT(STRUCT_OR_NBIT) GENERIC MAP(4) PORT MAP(LT,F3);
441  END ARCHITECTURE STRUCT_MAGNCOMP_4BIT;
```

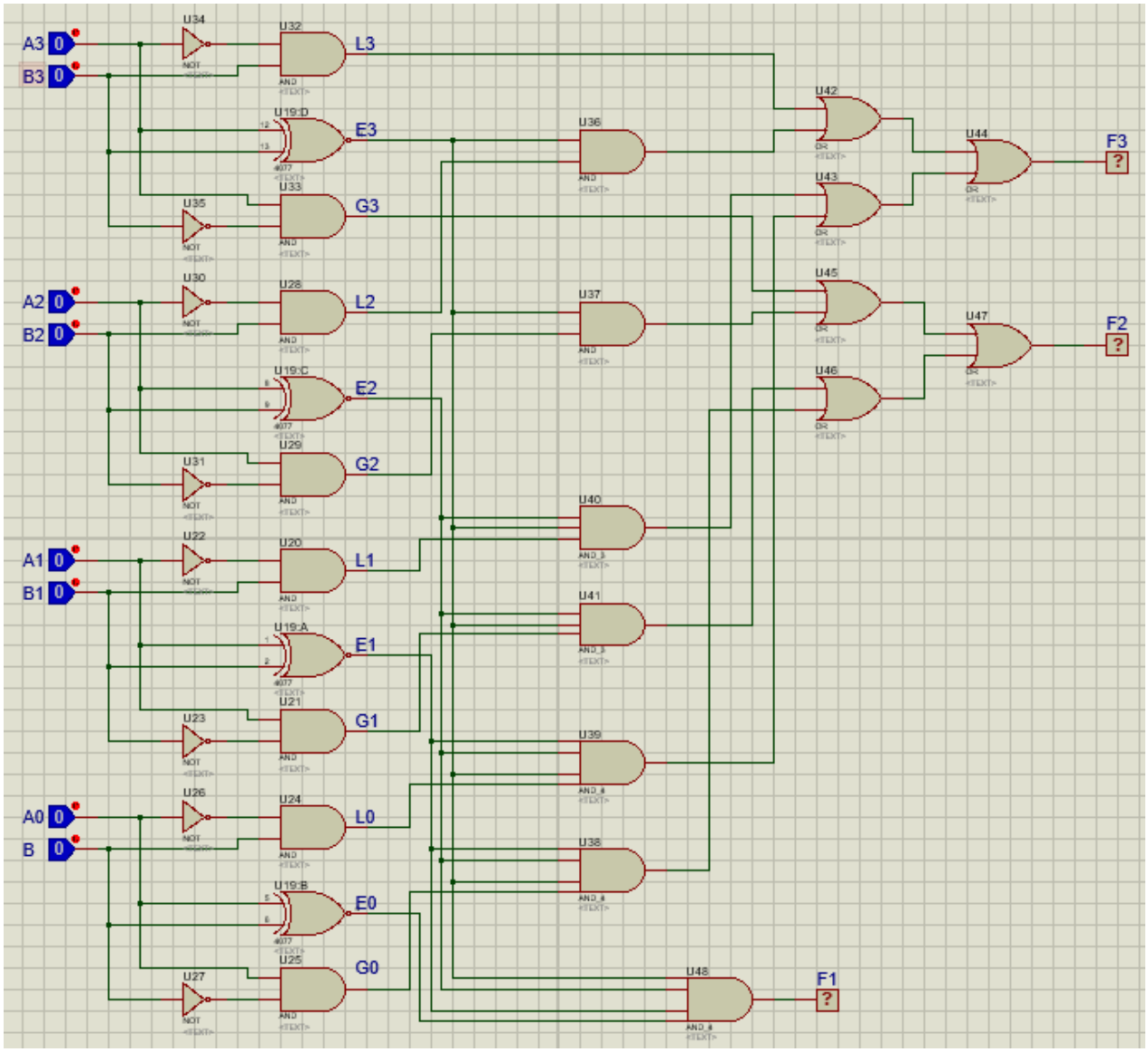
7.8. Code of 8-Bit Magnitude comparator

```
444  ----------------------------------------------------
445  --          8-Bit Magnitude comparator            --
446  ----------------------------------------------------
447  LIBRARY IEEE;
448  USE IEEE.STD_LOGIC_1164.ALL;
449
450  ENTITY MAGNCOMP_8BIT IS
451      PORT(A,B: IN STD_LOGIC_VECTOR (7 DOWNTO 0):= "00000000";    --8-Bit input
452           F1,F2,F3: OUT STD_LOGIC:= '0');              --Output: F1(A=B), F2(A>B), and F3 (A<B).
453  END ENTITY MAGNCOMP_8BIT;
454
455  ARCHITECTURE STRUCT_MAGNCOMP_8BIT OF MAGNCOMP_8BIT IS
456      SIGNAL T1,T2,T3,T4 :STD_LOGIC_VECTOR (3 DOWNTO 0):= "0000"; --To save A and B
457      SIGNAL E1,G1,L1 :STD_LOGIC:= '0';    --To save the output of the first 4-Bit Magnitude comparator
458      SIGNAL E2,G2,L2 :STD_LOGIC:= '0';    --To save the output of the second 4-Bit Magnitude comparator
459      SIGNAL GG,LL :STD_LOGIC:= '0';
460
461      BEGIN
462          T1 <= A(3)&A(2)&A(1)&A(0);
463          T2 <= B(3)&B(2)&B(1)&B(0);
464          T3 <= A(7)&A(6)&A(5)&A(4);
465          T4 <= B(7)&B(6)&B(5)&B(4);
466
467          M1: ENTITY WORK.MAGNCOMP_4BIT(STRUCT_MAGNCOMP_4BIT) PORT MAP (T1,T2,E1,G1,L1);
468          M2: ENTITY WORK.MAGNCOMP_4BIT(STRUCT_MAGNCOMP_4BIT) PORT MAP (T3,T4,E2,G2,L2);
469          M3: ENTITY WORK.AND_2P(STRUCT_AND_2P)PORT MAP(E2,G1,GG); --To find out the possibilities for greater or less
470          M4: ENTITY WORK.AND_2P(STRUCT_AND_2P)PORT MAP(E2,L1,LL);
471
472          M5: ENTITY WORK.AND_2P(STRUCT_AND_2P)PORT MAP(E1,E2,F1);
473          M6: ENTITY WORK.OR_2P(STRUCT_OR_2P)PORT MAP(G2,GG,F2);
474          M7: ENTITY WORK.OR_2P(STRUCT_OR_2P)PORT MAP(L2,LL,F3);
475  END ARCHITECTURE STRUCT_MAGNCOMP_8BIT;
476
```

7.8. Code of Stage 2 with Magnitude comparator

```
478  ----------------------------------------------------
479  ----      Stage_2 with Magnitude comparator         --
480  ----------------------------------------------------
481  LIBRARY IEEE;
482  USE IEEE.STD_LOGIC_1164.ALL;
483
484  ENTITY STAGE_2 IS
485      PORT(A,B: IN STD_LOGIC_VECTOR (7 DOWNTO 0):= "00000000";    --8-bit input
486           F1,F2,F3: OUT STD_LOGIC:='0');                         --Output: F1(A=B), F2(A>B), and F3 (A<B).
487  END ENTITY STAGE_2;
488
489  ARCHITECTURE STRUCT_STAGE_2  OF STAGE_2 IS
490      SIGNAL X, FF2,FF3 : STD_LOGIC:= '0';
491      BEGIN
492
493          S1: ENTITY WORK.MAGNCOMP_8BIT(STRUCT_MAGNCOMP_8BIT) PORT MAP(A,B,F1,FF2,FF3);
494          S2: ENTITY WORK.XOR_2P(STRUCT_XOR_2P)PORT MAP(A(7),B(7),X);  --To check sign bit
495          S3: ENTITY WORK.XOR_2P(STRUCT_XOR_2P)PORT MAP(X,FF2,F2);     --To get a correct answer
496          S4: ENTITY WORK.XOR_2P(STRUCT_XOR_2P)PORT MAP(X,FF3,F3);
497
498  END ARCHITECTURE STRUCT_STAGE_2;
499
```

7.9. Circuit of 4-bit magnitude comparator



7.10. Code of Test Generator

```
503  ----------------------------------------------------
504  ----                Test Generator                 --
505  ----------------------------------------------------
506  LIBRARY IEEE;
507  USE IEEE.STD_LOGIC_1164.ALL;
508  USE IEEE.STD_LOGIC_SIGNED.ALL;
509  USE IEEE.STD_LOGIC_ARITH.ALL;
510
511  ENTITY TEST_GENERATOR IS
512       PORT(Clk: IN STD_LOGIC:= '0';                                    --clock input
513            A,B: OUT STD_LOGIC_VECTOR(7 DOWNTO 0):= "00000000";     --8 bit output:
514            F1,F2,F3: OUT STD_LOGIC:= '0');                          --Output: F1(A=B), F2(A>B), and F3 (A<B)
515  END ENTITY TEST_GENERATOR;
516
517  ARCHITECTURE BEH_TEST_GENERATOR OF TEST_GENERATOR IS
518       SIGNAL IN1,IN2: STD_LOGIC_VECTOR(7 DOWNTO 0):= "00000000";
519       BEGIN
520            A <= IN1;      --set the value of IN1 in A to send it to comparator
521            B <= IN2;      --set the value of IN2 in B to send it to comparator
522            PROCESS         --changes the values of IN1 and IN2 also (A and B) when the clock positive edge
523            BEGIN
524                 FOR i IN -128 TO 127 LOOP    --to go through all possible values
525                      FOR j IN -128 TO 127 LOOP
526                                IN1(7 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(i,8); --conv from integer to STD_LOGIC
527                                IN2(7 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR(j,8); --conv from integer to STD_LOGIC
528                                WAIT UNTIL rising_edge(Clk);
529                      END LOOP;
530                 END LOOP;
531                 WAIT;
532            END PROCESS;
533
534            PROCESS (IN1,IN2)    --implement the system using behavioural logic
535                 BEGIN
536                      IF(IN1 > IN2) THEN
537                           F1 <= '0' ;
538                           F2 <= '1' ;
539                           F3 <= '0' ;
540                      ELSIF (IN1 < IN2) THEN
541                           F1 <= '0' ;
542                           F2 <= '0' ;
543                           F3 <= '1' ;
544                      ELSE
545                           F1 <= '1' ;
546                           F2 <= '0' ;
547                           F3 <= '0' ;
548                      END IF;
549            END PROCESS;
550  END ARCHITECTURE BEH_TEST_GENERATOR;
```

7.11. Code of Result Analyze

```
553  ----------------------------------------------------
554  --                  Analyze                   --
555  ----------------------------------------------------
556  LIBRARY IEEE;
557  USE IEEE.STD_LOGIC_1164.ALL;
558  USE IEEE.STD_LOGIC_ARITH.ALL;
559
560  ENTITY ANALYZE_RESULT IS
561       PORT(SF1,SF2,SF3: IN STD_LOGIC:='0';   --input the result of system
562            CF1,CF2,CF3: IN STD_LOGIC:='0';   --input correct result
563            Clk: IN STD_LOGIC:='0');          --clock input
564  END ENTITY ANALYZE_RESULT;
565
566  ARCHITECTURE ANALYZE OF ANALYZE_RESULT IS
567       BEGIN
568            PROCESS               --to ensure that the system output is equal to the theoretical output
569                 BEGIN
570                      assert ((SF1 = CF1) AND (SF2 = CF2) AND (SF3 = CF3))      --if the system output is not equal to the theoretical output print an error
571                      report "The obtained results do not accord with the theoretical result !!"
572                      severity ERROR;
573                      WAIT UNTIL rising_edge(Clk);
574            END PROCESS;
575  END ARCHITECTURE  ANALYZE;
576
```

7.12. Code of Verification Stage 1 & Stage 2

```
578  ------------------------------------------------
579  --              Verification circuit            --
580  ------------------------------------------------
581  LIBRARY IEEE;
582  USE IEEE.STD_LOGIC_1164.ALL;
583  USE IEEE.STD_LOGIC_ARITH.ALL;
584
585  ENTITY VERIFICATION IS
586  END ENTITY VERIFICATION;
587
588  ------------------------------------------------
589  --              Verification Stage_1            --
590  ------------------------------------------------
591  ARCHITECTURE VERIF_STAGE_1 OF VERIFICATION IS
592      SIGNAL A,B: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
593      SIGNAL SF1,SF2,SF3: STD_LOGIC:='0';    --input the result of system
594      SIGNAL CF1,CF2,CF3: STD_LOGIC:='0';    --input correct result
595      SIGNAL Clk: STD_LOGIC:='0';
596
597
598      BEGIN
599          Clk <= NOT CLK AFTER 133 NS;
600          V1: ENTITY WORK.TEST_GENERATOR(BEH_TEST_GENERATOR) PORT MAP(Clk , A , B , CF1 , CF2 , CF3);
601          V2: ENTITY WORK.STAGE_1(STRUCT_STAGE_1) PORT MAP(A , B , SF1 , SF2 , SF3);
602          V3: ENTITY WORK.ANALYZE_RESULT(ANALYZE) PORT MAP(SF1 , SF2 , SF3 , CF1 , CF2 , CF3 , Clk);
603  END ARCHITECTURE VERIF_STAGE_1;
604
605
606  ------------------------------------------------
607  --              Verification Stage_2            --
608  ------------------------------------------------
609  ARCHITECTURE VERIF_STAGE_2 OF VERIFICATION IS
610      SIGNAL A,B: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
611      SIGNAL SF1,SF2,SF3: STD_LOGIC:='0';    --input the result of system
612      SIGNAL CF1,CF2,CF3: STD_LOGIC:='0';    --input correct result
613      SIGNAL Clk: STD_LOGIC:='0';
614
615
616      BEGIN
617          Clk <= NOT CLK AFTER 25 NS;
618          V1: ENTITY WORK.TEST_GENERATOR(BEH_TEST_GENERATOR) PORT MAP(Clk , A , B , CF1 , CF2 , CF3);
619          V2: ENTITY WORK.STAGE_2(STRUCT_STAGE_2) PORT MAP(A , B , SF1 , SF2 , SF3);
620          V3: ENTITY WORK.ANALYZE_RESULT(ANALYZE) PORT MAP(SF1 , SF2 , SF3 , CF1 , CF2 , CF3 , Clk);
621  END ARCHITECTURE VERIF_STAGE_2;
```

7.13. Full Code with comments

```
------------------------------------------------

--              COURSE PROJECT                          --

------------------------------------------------

--                                      Tariq Odeh                              --

--                                       1190699                                --

--                                       sec: 2                                 --

------------------------------------------------
```

----------Basic gates with delay----------

```
----------------------------------------------------
--           INVERTER GATE (1 PORT)           --
----------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY NOT_1P IS
        PORT(A: IN STD_LOGIC:= '0';     --1 bit Input
                F: OUT STD_LOGIC:= '0');          --1 bit Output
END ENTITY NOT_1P;

ARCHITECTURE STRUCT_NOT_1P OF NOT_1P IS
  BEGIN
                F <= NOT A AFTER 2 NS;            --2 ns delay
END ARCHITECTURE STRUCT_NOT_1P;



----------------------------------------------------
--           NAND GATE (N-BIT)             --
----------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY NAND_NBIT IS
        GENERIC ( N : POSITIVE := 2);                --N bit Input
        PORT(A: IN STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";
                F: OUT STD_LOGIC:= '0');                --1 bit Output
END ENTITY NAND_NBIT;

ARCHITECTURE STRUCT_NAND_NBIT OF NAND_NBIT IS
        SIGNAL TEMP : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
```

```
   BEGIN
                TEMP(0) <= A(0);
          FLOOP: FOR i IN 1 TO (N-1) GENERATE
                TEMP(i) <= TEMP(i-1) AND A(i);
          END GENERATE FLOOP;
          F <= NOT(TEMP(N-1)) AFTER 5 NS;       --NAND = NOT(AND)
END ARCHITECTURE STRUCT_NAND_NBIT;                  --5 ns delay



--------------------------------------------------
--          NAND GATE (2 PORTS)             --
--------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY NAND_2P IS
        PORT(A,B: IN STD_LOGIC:= '0';  --2 bit Input
                  F: OUT STD_LOGIC:= '0');         --1 bit Output
END ENTITY NAND_2P;

ARCHITECTURE STRUCT_NAND_2P OF NAND_2P IS
   BEGIN
                F <= A NAND B AFTER 5 NS;                   --5 ns delay
        -- F <= NOT(A AND B) AFTER 5 NS;
END ARCHITECTURE STRUCT_NAND_2P;



--------------------------------------------------
--          NOR GATE (N-BIT)              --
--------------------------------------------------
LIBRARY IEEE;
```

```
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY NOR_NBIT IS
        GENERIC ( N : POSITIVE := 2);  --N bit Input
        PORT(A: IN STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";
                F: OUT STD_LOGIC:='0');                         --1 bit Output
END ENTITY NOR_NBIT;

ARCHITECTURE STRUCT_NOR_NBIT OF NOR_NBIT IS
        SIGNAL TEMP : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
  BEGIN
                TEMP(0) <= A(0);
           FLOOP: FOR i IN 1 TO (N-1) GENERATE
                TEMP(i) <= TEMP(i-1) OR A(i);
           END GENERATE FLOOP;
           F <= NOT(TEMP(N-1)) AFTER 5 NS;      --NOR = NOT(OR)
END ARCHITECTURE STRUCT_NOR_NBIT;           --5 ns delay


--------------------------------------------------
--          NOR GATE (2 PORTS)              --
--------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY NOR_2P IS
        PORT(A,B: IN STD_LOGIC:= '0';   --2 bit Input
                F: OUT STD_LOGIC:= '0');        --1 bit Output
END ENTITY NOR_2P;

ARCHITECTURE STRUCT_NOR_2P OF NOR_2P IS
```

```
   BEGIN
                F <= A NOR B AFTER 5 NS;          --5 ns delay
          -- F <= NOT(A OR B) AFTER 5 NS;
END ARCHITECTURE STRUCT_NOR_2P;



--------------------------------------------------
--          AND GATE (N-BIT)              --
--------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY AND_NBIT IS
          GENERIC ( N : POSITIVE := 2);     --N bit Input
          PORT(A: IN STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";
                F: OUT STD_LOGIC:= '0');                    --1 bit Output
END ENTITY AND_NBIT;

ARCHITECTURE STRUCT_AND_NBIT OF AND_NBIT IS
          SIGNAL TEMP : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
   BEGIN
                TEMP(0) <= A(0);
             FLOOP: FOR i IN 1 TO (N-1) GENERATE
                TEMP(i) <= TEMP(i-1) AND A(i);
             END GENERATE FLOOP;
             F <= TEMP(N-1) AFTER 7 NS;   --7 ns delay
END ARCHITECTURE STRUCT_AND_NBIT;



--------------------------------------------------
--          AND GATE (2 PORTS)            --
```

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

ENTITY AND_2P IS
        PORT(A,B: IN STD_LOGIC:= '0';   --2 bit Input
                F: OUT STD_LOGIC:= '0');        --1 bit Output
END ENTITY AND_2P;

ARCHITECTURE STRUCT_AND_2P OF AND_2P IS
  BEGIN
                F <= A AND B AFTER 7 NS;        --7 ns delay
END ARCHITECTURE STRUCT_AND_2P;



----------------------------------------------------
--          OR GATE (N-BIT)             --
----------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY OR_NBIT IS
        GENERIC ( N : POSITIVE := 2); --N bit Input
        PORT(A: IN STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";
                F: OUT STD_LOGIC:= '0');                --1 bit Output
END ENTITY OR_NBIT;

ARCHITECTURE STRUCT_OR_NBIT OF OR_NBIT IS
        SIGNAL TEMP : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
  BEGIN
                TEMP(0) <= A(0);
```

```
            FLOOP: FOR i IN 1 TO (N-1) GENERATE
                  TEMP(i) <= TEMP(i-1) OR A(i);
            END GENERATE FLOOP;
            F <= TEMP(N-1) AFTER 7 NS;   --7 ns delay
END ARCHITECTURE STRUCT_OR_NBIT;
--------------------------------------------------
--          OR GATE (2 PORTS)            --
--------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY OR_2P IS
        PORT(A,B: IN STD_LOGIC:= '0';   --2 bit Input
                  F: OUT STD_LOGIC:= '0');        --1 bit Output
END ENTITY OR_2P;

ARCHITECTURE STRUCT_OR_2P OF OR_2P IS
  BEGIN
                  F <= A OR B AFTER 7 NS;          --7 ns delay
END ARCHITECTURE STRUCT_OR_2P;



--------------------------------------------------
--          XNOR GATE (2 PORTS)          --
--------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY XNOR_2P IS
        PORT(A,B: IN STD_LOGIC:= '0';   --2 bit Input
                  F: OUT STD_LOGIC:= '0');        --1 bit Output
```

```
END ENTITY XNOR_2P;

ARCHITECTURE STRUCT_XNOR_2P OF XNOR_2P IS
   BEGIN
                F <= A XNOR B AFTER 9 NS;        --9 ns delay
END ARCHITECTURE STRUCT_XNOR_2P;



--------------------------------------------------
--          XOR GATE (2 PORTS)              --
--------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY XOR_2P IS
        PORT(A,B: IN STD_LOGIC:= '0'; --2 bit Input
                F: OUT STD_LOGIC:= '0');        --1 bit Output
END ENTITY XOR_2P;

ARCHITECTURE STRUCT_XOR_2P OF XOR_2P IS
   BEGIN
                F <= A XOR B AFTER 12 NS;        --12 ns delay
END ARCHITECTURE STRUCT_XOR_2P;



--------------------------------------------------
--              Half Adder                  --
--------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
```

```
ENTITY HA IS
        PORT(A,B: IN STD_LOGIC:= '0';                   --Input
                Sum,Cout: OUT STD_LOGIC:= '0');         --Output
END ENTITY HA;

ARCHITECTURE STRUCT_HA OF HA IS
        SIGNAL X : STD_LOGIC_VECTOR (1 DOWNTO 0):= "00";
  BEGIN
                X <= A & B;
                H1:     ENTITY WORK.XOR_2P(STRUCT_XOR_2P) PORT MAP (A,B,Sum);
        --Sum = A XOR B
                H2:     ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(2) PORT
MAP (X,Cout);   --Cout = A AND B
END ARCHITECTURE STRUCT_HA;



--------------------------------------------------
--              Full Adder              --
--------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY FAD IS
        PORT(A,B,Cin: IN STD_LOGIC:= '0';
                Sum,Cout: OUT STD_LOGIC:= '0');
END ENTITY FAD;

ARCHITECTURE STRUCT_FAD OF FAD IS
        SIGNAL N1,N2,N3,N4,N5,N6,N7: STD_LOGIC := '0';
        BEGIN
                                        --First way to implement Full Adder Using XOR,AND,OR
gates
```

```
                --F1: ENTITY WORK.XOR_2P(STRUCT_XOR_2P) PORT MAP(A,B,N1);
        --SUM = A XOR B XOR Cin

                --F2: ENTITY WORK.XOR_2P(STRUCT_XOR_2P) PORT MAP(N1,Cin,Sum);     --
Cout = Cin.(A XOR B) + A.B

                --F3: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP(A,B,N2);

                --F4: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP(N1,Cin,N3);

                --F5: ENTITY WORK.OR_2P(STRUCT_OR_2P) PORT MAP(N2,N3,Cout);


                                --Second way to implement Full Adder Using Half Adder

                --F1: ENTITY WORK.HA(STRUCT_HA) PORT MAP(A,B,N1,N2);

                --F2: ENTITY WORK.HA(STRUCT_HA) PORT MAP(N1,Cin,Sum,N3);

                --F3: ENTITY WORK.OR_2P(STRUCT_OR_2P) PORT MAP(N2,N3,Cout);


                                --Third way to implement Full Adder Using NAND gate
(faster)

                F1: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(A,B,N1);

                F2: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N1,A,N2);

                F3: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N1,B,N3);

                F4: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N2,N3,N4);

                F5: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N4,Cin,N5);

                F6: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N5,N1,Cout);

                F7: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N5,Cin,N6);

                F8: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N5,N4,N7);

                F9: ENTITY WORK.NAND_2P(STRUCT_NAND_2P) PORT MAP(N6,N7,Sum);

END ARCHITECTURE STRUCT_FAD;



--------------------------------------------------

--          n-Bit Ripple Adder/subtractor       --

--------------------------------------------------

LIBRARY IEEE;
```

```
USE IEEE.STD_LOGIC_1164.ALL;


ENTITY NBIT_ADDER IS
        GENERIC ( N : POSITIVE :=8);
        PORT(A,B: IN STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";
                Cin: IN STD_LOGIC :='1';                        --Set Cin = 1 to find 2's complemant for B
                Sum: OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0):= "00000000";
                Cout, OV: OUT STD_LOGIC:= '0');          --OV (overflow)
END ENTITY NBIT_ADDER;


ARCHITECTURE STRUCT_RIPPLE OF NBIT_ADDER IS
        SIGNAL CARRY : STD_LOGIC_VECTOR (N DOWNTO 0):= "000000000";
        SIGNAL X :STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";  --to save 2's complemant of B
   BEGIN
                CARRY(0) <= Cin;
            Cout <= CARRY(N);
                R1: ENTITY WORK.XOR_2P(STRUCT_XOR_2P) PORT MAP (CARRY(N),CARRY(N-1),OV);
            FLOOP: FOR i IN 0 TO (N-1) GENERATE          --Loop to generate ripple Adder/subtractor
                        R2: ENTITY WORK.XOR_2P(STRUCT_XOR_2P) PORT MAP (B(i),Cin,X(i)); --if Cin = 0 then ripple Adder work, if Cin = 1 ripple subtractor work
                        R3: ENTITY WORK.FAD(STRUCT_FAD) PORT MAP (A(i),X(i),CARRY(i),Sum(i),CARRY(i+1));
            END GENERATE FLOOP;
END ARCHITECTURE STRUCT_RIPPLE;



-------------------------------------------------------
----            n-Bit Comparator            --
-------------------------------------------------------
```

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

ENTITY NBIT_COMPARATOR IS
        GENERIC ( N : POSITIVE :=8);
        PORT(A,B: IN STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";        --Input
                F1,F2,F3: OUT STD_LOGIC:= '0');                    --Output: F1(A=B), F2(A>B), and F3 (A<B).
END ENTITY NBIT_COMPARATOR;

ARCHITECTURE STRUCT_NBIT_COMPARATOR  OF NBIT_COMPARATOR IS
        SIGNAL SUM : STD_LOGIC_VECTOR (N-1 DOWNTO 0):= "00000000";
        SIGNAL Cout,X,S : STD_LOGIC:= '0';                    --X to check overflow, S to check if sum equal zero
        SIGNAL NC,NS,NX: STD_LOGIC:= '0';  --NC (not Cout) ,NS (not S(N-1)),NX (not Overflow)
        SIGNAL FF1,FF2,FF3: STD_LOGIC:= '0';  --to save output of F1,F2,F3 if there is overflow
        SIGNAL FFF1,FFF2,FFF3: STD_LOGIC:= '0';--to save output of F1,F2,F3 if there is not overflow
        SIGNAL A1,A2,B1,B2: STD_LOGIC:= '0';

        BEGIN
                P1: ENTITY WORK.NBIT_ADDER(STRUCT_RIPPLE) GENERIC MAP(N) PORT MAP(A,B,'1',SUM,Cout,X);        --ripple subtractor        (the output of overflow save in X)
                P3: ENTITY WORK.OR_NBIT(STRUCT_OR_NBIT) GENERIC MAP(N) PORT MAP(SUM,S);                --check if sum equal zero
                P4: ENTITY WORK.NOT_1P(STRUCT_NOT_1P)PORT MAP(Cout,NC);
                                                --NC (not Cout)
                P5: ENTITY WORK.NOT_1P(STRUCT_NOT_1P)PORT MAP(SUM(N-1),NS);
                                                --NS (not S(N-1))
                P6: ENTITY WORK.NOT_1P(STRUCT_NOT_1P)PORT MAP(X,NX);
                                                --NX (not Overflow)

                X1: ENTITY WORK.NOR_2P(STRUCT_NOR_2P) PORT MAP (Cout,S,A1);        --find the value of F1,F2,F3 if there is overflow
```

```
                X2: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (NC,S,A2);

                X3: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (X,A1,FF1);

                X4: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (X,A2,FF2);

                X5: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (X,Cout,FF3);


                Y1: ENTITY WORK.NOR_2P(STRUCT_NOR_2P) PORT MAP (SUM(N-1),S,B1);
--find the value of F1,F2,F3 if there is not overflow

                Y2: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (NS,S,B2);

                Y3: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (NX,B1,FFF1);

                Y4: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (NX,B2,FFF2);

                Y5: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT MAP (NX,SUM(N-1),FFF3);


                Z1: ENTITY WORK.OR_2P(STRUCT_OR_2P) PORT MAP(FF1,FFF1,F1);          --
final value of F1,F2,F3

                Z2: ENTITY WORK.OR_2P(STRUCT_OR_2P) PORT MAP(FF2,FFF2,F2);

                Z3: ENTITY WORK.OR_2P(STRUCT_OR_2P) PORT MAP(FF3,FFF3,F3);


END ARCHITECTURE STRUCT_NBIT_COMPARATOR;



-------------------------------------------------------

----    Stage_1 with ripple adder/subtractor    --

-------------------------------------------------------

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;


ENTITY STAGE_1 IS

        PORT(A,B: IN STD_LOGIC_VECTOR (7 DOWNTO 0):= "00000000";          --8-bit input

                F1,F2,F3: OUT STD_LOGIC:='0');
        --Output: F1(A=B), F2(A>B), and F3 (A<B).

END ENTITY STAGE_1;
```

```
ARCHITECTURE STRUCT_STAGE_1  OF STAGE_1 IS
        BEGIN
                S1: ENTITY WORK.NBIT_COMPARATOR(STRUCT_NBIT_COMPARATOR) GENERIC
MAP(8) PORT MAP(A,B,F1,F2,F3);
END ARCHITECTURE STRUCT_STAGE_1;


--
================================================================================
================================================

--
================================================================================
================================================


------------------------------------------------
--        1-Bit Magnitude comparator        --
------------------------------------------------


LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MAGNCOMP_1BIT IS
        PORT(A,B: IN STD_LOGIC:= '0';                --1-Bit input
           F1,F2,F3: OUT STD_LOGIC:= '0');  --Output: F1(A=B), F2(A>B), and F3 (A<B)
END ENTITY MAGNCOMP_1BIT;

ARCHITECTURE STRUCT_MAGNCOMP_1BIT OF MAGNCOMP_1BIT IS
        SIGNAL NA,NB : STD_LOGIC:= '0';   --NA(not A) , NB(not B)
        BEGIN
                M1: ENTITY WORK.XNOR_2P(STRUCT_XNOR_2P)PORT MAP(A,B,F1);
                M2: ENTITY WORK.NOT_1P(STRUCT_NOT_1P)PORT MAP(B,NB);
                M3: ENTITY WORK.AND_2P(STRUCT_AND_2P)PORT MAP(A,NB,F2);
                M4: ENTITY WORK.NOT_1P(STRUCT_NOT_1P)PORT MAP(A,NA);
                M5: ENTITY WORK.AND_2P(STRUCT_AND_2P)PORT MAP(NA,B,F3);
```

```
END ARCHITECTURE STRUCT_MAGNCOMP_1BIT;


--------------------------------------------------
--          4-Bit Magnitude comparator        --
--------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MAGNCOMP_4BIT IS
          PORT(A,B: IN STD_LOGIC_VECTOR (3 DOWNTO 0):= "0000";       --4-Bit input
                    F1,F2,F3: OUT STD_LOGIC:= '0');                    --Output: F1(A=B), F2(A>B), and
F3 (A<B).
END ENTITY MAGNCOMP_4BIT;

ARCHITECTURE STRUCT_MAGNCOMP_4BIT OF MAGNCOMP_4BIT IS
          SIGNAL E,G,L :STD_LOGIC_VECTOR (3 DOWNTO 0):= "0000";  --E(equal)A=B,
G(greater)A>B, L(less)A<B
          SIGNAL GT,LT :STD_LOGIC_VECTOR (3 DOWNTO 0):= "0000";
          SIGNAL LL1,GG1 :STD_LOGIC_VECTOR (1 DOWNTO 0):= "00";
          SIGNAL LL2,GG2 :STD_LOGIC_VECTOR (2 DOWNTO 0):= "000";
          SIGNAL LL3,GG3 :STD_LOGIC_VECTOR (3 DOWNTO 0):= "0000";
          SIGNAL X1,X2,X3,X4,X5,X6 :STD_LOGIC := '0';
          BEGIN
            FLOOP1: FOR i IN 0 TO (3) GENERATE     --loop to compare 4 bit by using 1-Bit
Magnitude comparator
                              M1: ENTITY WORK.MAGNCOMP_1BIT(STRUCT_MAGNCOMP_1BIT)
PORT MAP (A(i),B(i),E(i),G(i),L(i));
          END GENERATE FLOOP1;


                    LL1 <= L(2)&E(3);  --To find out the possibilities for greater or less
                    GG1 <= G(2)&E(3);
```

```
                LL2 <= L(1)&E(3)&E(2);
                GG2 <= G(1)&E(3)&E(2);
                LL3 <= L(0)&E(3)&E(2)&E(1);
                GG3 <= G(0)&E(3)&E(2)&E(1);

                M2: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(2) PORT
MAP(LL1,X1);
                M3: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(2) PORT
MAP(GG1,X2);
                M4: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(3) PORT
MAP(LL2,X3);
                M5: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(3) PORT
MAP(GG2,X4);
                M6: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(4) PORT
MAP(LL3,X5);
                M7: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(4) PORT
MAP(GG3,X6);

                GT <= G(3)&X2&X4&X6;  --to check if greater or less
                LT <= L(3)&X1&X3&X5;

                M8: ENTITY WORK.AND_NBIT(STRUCT_AND_NBIT) GENERIC MAP(4) PORT
MAP(E,F1);
                M9: ENTITY WORK.OR_NBIT(STRUCT_OR_NBIT) GENERIC MAP(4) PORT
MAP(GT,F2);
                M10: ENTITY WORK.OR_NBIT(STRUCT_OR_NBIT) GENERIC MAP(4) PORT
MAP(LT,F3);
END ARCHITECTURE STRUCT_MAGNCOMP_4BIT;



-------------------------------------------------
--        8-Bit Magnitude comparator        --
-------------------------------------------------
LIBRARY IEEE;
```

```
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MAGNCOMP_8BIT IS
        PORT(A,B: IN STD_LOGIC_VECTOR (7 DOWNTO 0):= "00000000";          --8-Bit input
                F1,F2,F3: OUT STD_LOGIC:= '0');                    --Output: F1(A=B), F2(A>B), and F3 (A<B).
END ENTITY MAGNCOMP_8BIT;

ARCHITECTURE STRUCT_MAGNCOMP_8BIT OF MAGNCOMP_8BIT IS
        SIGNAL T1,T2,T3,T4 :STD_LOGIC_VECTOR (3 DOWNTO 0):= "0000"; --To save A and B
        SIGNAL E1,G1,L1 :STD_LOGIC:= '0';    --To save the output of the first 4-Bit Magnitude comparator
        SIGNAL E2,G2,L2 :STD_LOGIC:= '0';    --To save the output of the second 4-Bit Magnitude comparator
        SIGNAL GG,LL :STD_LOGIC:= '0';

        BEGIN
                T1 <= A(3)&A(2)&A(1)&A(0);
                T2 <= B(3)&B(2)&B(1)&B(0);
                T3 <= A(7)&A(6)&A(5)&A(4);
                T4 <= B(7)&B(6)&B(5)&B(4);

                M1: ENTITY WORK.MAGNCOMP_4BIT(STRUCT_MAGNCOMP_4BIT) PORT MAP (T1,T2,E1,G1,L1);
                M2: ENTITY WORK.MAGNCOMP_4BIT(STRUCT_MAGNCOMP_4BIT) PORT MAP (T3,T4,E2,G2,L2);
                M3: ENTITY WORK.AND_2P(STRUCT_AND_2P)PORT MAP(E2,G1,GG); --To find out the possibilities for greater or less
            M4: ENTITY WORK.AND_2P(STRUCT_AND_2P)PORT MAP(E2,L1,LL);

                M5: ENTITY WORK.AND_2P(STRUCT_AND_2P)PORT MAP(E1,E2,F1);
                M6: ENTITY WORK.OR_2P(STRUCT_OR_2P)PORT MAP(G2,GG,F2);
                M7: ENTITY WORK.OR_2P(STRUCT_OR_2P)PORT MAP(L2,LL,F3);
```

```
END ARCHITECTURE STRUCT_MAGNCOMP_8BIT;



--------------------------------------------------

----	Stage_2 with Magnitude comparator	--

--------------------------------------------------

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;


ENTITY STAGE_2 IS

	PORT(A,B: IN STD_LOGIC_VECTOR (7 DOWNTO 0):= "00000000";		--8-bit input

		F1,F2,F3: OUT STD_LOGIC:='0');
	--Output: F1(A=B), F2(A>B), and F3 (A<B).

END ENTITY STAGE_2;


ARCHITECTURE STRUCT_STAGE_2  OF STAGE_2 IS

	SIGNAL X, FF2,FF3 : STD_LOGIC:= '0';

	BEGIN


		S1: ENTITY WORK.MAGNCOMP_8BIT(STRUCT_MAGNCOMP_8BIT) PORT
MAP(A,B,F1,FF2,FF3);

		S2: ENTITY WORK.XOR_2P(STRUCT_XOR_2P)PORT MAP(A(7),B(7),X);	--To
check sign bit

		S3: ENTITY WORK.XOR_2P(STRUCT_XOR_2P)PORT MAP(X,FF2,F2);	--To get a
correct answer

		S4: ENTITY WORK.XOR_2P(STRUCT_XOR_2P)PORT MAP(X,FF3,F3);


END ARCHITECTURE STRUCT_STAGE_2;


--
================================================================================
================================================
```

```
--
================================================================================
================================================

----------------------------------------------------

----            Test Generator              --

----------------------------------------------------

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

USE IEEE.STD_LOGIC_SIGNED.ALL;

USE IEEE.STD_LOGIC_ARITH.ALL;


ENTITY TEST_GENERATOR IS

        PORT(Clk: IN STD_LOGIC:= '0';
        --clock input

                A,B: OUT STD_LOGIC_VECTOR(7 DOWNTO 0):= "00000000";     --8 bit output:

           F1,F2,F3: OUT STD_LOGIC:= '0');                                          --
Output: F1(A=B), F2(A>B), and F3 (A<B)

END ENTITY TEST_GENERATOR;


ARCHITECTURE BEH_TEST_GENERATOR OF TEST_GENERATOR IS

        SIGNAL IN1,IN2: STD_LOGIC_VECTOR(7 DOWNTO 0):= "00000000";

        BEGIN

                A <= IN1;          --set the value of IN1 in A to send it to comparator

                B <= IN2;          --set the value of IN2 in B to send it to comparator

                PROCESS          --changes the values of IN1 and IN2 also (A and B) when the
clock positive edge

                BEGIN

                        FOR i IN -128 TO 127 LOOP        --to go through all possible values

                                FOR j IN -128 TO 127 LOOP

                                                        IN1(7 DOWNTO 0) <=
CONV_STD_LOGIC_VECTOR(i,8);          --conv from integer to STD_LOGIC
```

```
                                                IN2(7 DOWNTO 0) <=
CONV_STD_LOGIC_VECTOR(j,8);          --conv from integer to STD_LOGIC
                                                WAIT UNTIL rising_edge(Clk);
                        END LOOP;
                END LOOP;
                WAIT;
        END PROCESS;

        PROCESS (IN1,IN2)          --implement the system using behavioural logic
                BEGIN
                        IF(IN1 > IN2) THEN
                                F1 <= '0' ;
                                F2 <= '1' ;
                                F3 <= '0' ;
                        ELSIF (IN1 < IN2) THEN
                                F1 <= '0' ;
                                F2 <= '0' ;
                                F3 <= '1' ;
                        ELSE
                                F1 <= '1' ;
                                F2 <= '0' ;
                                F3 <= '0' ;
                        END IF;
        END PROCESS;
END ARCHITECTURE BEH_TEST_GENERATOR;



-------------------------------------------------
--              Analyze             --
-------------------------------------------------
LIBRARY IEEE;
```

```
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

ENTITY ANALYZE_RESULT IS
        PORT(SF1,SF2,SF3: IN STD_LOGIC:='0';   --input the result of system
                CF1,CF2,CF3: IN STD_LOGIC:='0';   --input correct result
                Clk: IN STD_LOGIC:='0');                --clock input
END ENTITY ANALYZE_RESULT;

ARCHITECTURE ANALYZE OF ANALYZE_RESULT IS
        BEGIN
                PROCESS                         --to ensure that the system output is equal to
the theoretical output
                        BEGIN
                                assert ((SF1 = CF1) AND (SF2 = CF2) AND (SF3 = CF3))        --if the
system output is not equal to the theoretical output print an error
                                report "The obtained results do not accord with the theoretical
result !!"
                                severity ERROR;
                                WAIT UNTIL rising_edge(Clk);
                END PROCESS;
END ARCHITECTURE  ANALYZE;



-------------------------------------------------
--          Verification circuit          --
-------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

ENTITY VERIFICATION IS
```

```
END ENTITY VERIFICATION;


-------------------------------------------------
--          Verification Stage_1          --
-------------------------------------------------
ARCHITECTURE VERIF_STAGE_1 OF VERIFICATION IS
        SIGNAL A,B: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
        SIGNAL SF1,SF2,SF3: STD_LOGIC:='0';   --input the result of system
        SIGNAL CF1,CF2,CF3: STD_LOGIC:='0';   --input correct result
        SIGNAL Clk: STD_LOGIC:='0';



        BEGIN
                Clk <= NOT CLK AFTER 133 NS;
                V1: ENTITY WORK.TEST_GENERATOR(BEH_TEST_GENERATOR) PORT MAP(Clk , A
, B , CF1 , CF2 , CF3);
                V2: ENTITY WORK.STAGE_1(STRUCT_STAGE_1) PORT MAP(A , B , SF1 , SF2 ,
SF3);
                V3: ENTITY WORK.ANALYZE_RESULT(ANALYZE) PORT MAP(SF1 , SF2 , SF3 , CF1 ,
CF2 , CF3 , Clk);
END ARCHITECTURE VERIF_STAGE_1;



-------------------------------------------------
--          Verification Stage_2          --
-------------------------------------------------
ARCHITECTURE VERIF_STAGE_2 OF VERIFICATION IS
        SIGNAL A,B: STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
        SIGNAL SF1,SF2,SF3: STD_LOGIC:='0';   --input the result of system
        SIGNAL CF1,CF2,CF3: STD_LOGIC:='0';   --input correct result
        SIGNAL Clk: STD_LOGIC:='0';
```

```
        BEGIN

                Clk <= NOT CLK AFTER 25 NS;

                V1: ENTITY WORK.TEST_GENERATOR(BEH_TEST_GENERATOR) PORT MAP(Clk , A
, B , CF1 , CF2 , CF3);

                V2: ENTITY WORK.STAGE_2(STRUCT_STAGE_2) PORT MAP(A , B , SF1 , SF2 ,
SF3);

                V3: ENTITY WORK.ANALYZE_RESULT(ANALYZE) PORT MAP(SF1 , SF2 , SF3 , CF1 ,
CF2 , CF3 , Clk);

END ARCHITECTURE VERIF_STAGE_2;


--
|=============================================================================
==================================|

--|======      D-flip-flop (1-bit)   //   D-flip-flop (n-bit)   //   n-Bit Carry Lookahead
          ======|

--
|=============================================================================
==================================|


-------------------------------------------------

--            D-flip-flop (1-bit)            --

-------------------------------------------------

--LIBRARY IEEE;

--USE IEEE.STD_LOGIC_1164.ALL;

--

--ENTITY DFF_1B IS

--         PORT(D,Clk,Reset: IN STD_LOGIC:= '0';               --Input

--                        Q,NQ: OUT STD_LOGIC:= '0');                    --Output

--END ENTITY DFF_1B;

--

--ARCHITECTURE STRUCT_DFF_1B OF DFF_1B IS

--   BEGIN
```

```
--                  PROCESS(Reset,Clk)
--                  BEGIN
--                          IF (Reset = '1') THEN                                   --Reset the dff when
reset is 1
--                                  Q <= '0';
--                                  NQ <= '0';
--                          ELSE
--                                  IF (rising_edge(Clk)) THEN     --On positive edge
--                                          Q <= D ;
--                                          NQ <= NOT D ;
--                                  END IF;
--                          END IF;
--                  END PROCESS;
--END ARCHITECTURE STRUCT_DFF_1B;



--------------------------------------------------
--        n-Bit (flip-flops/registers)       --
--------------------------------------------------
--LIBRARY IEEE;
--USE IEEE.STD_LOGIC_1164.ALL;
--
--ENTITY NBIT_REGISTERS IS
--        GENERIC (N : POSITIVE := 16);
--        PORT (D: IN STD_LOGIC_vector(N-1 DOWNTO 0):= "00000000";              --Input
--            Clk,Reset: IN STD_LOGIC:= '0';
          --Input
--                  Q,NQ : OUT STD_LOGIC_vector(N-1 DOWNTO 0):= "00000000");          --
Output
--END ENTITY NBIT_REGISTERS;
--
--
```

```
--ARCHITECTURE STRUCT_NBIT_REGISTERS OF NBIT_REGISTERS IS
--          BEGIN
--                    FLOOP : FOR i IN 0 TO (N-1) GENERATE                    --Loop to generate n-bit dff from 1-bit dff
--                              G_DFF : ENTITY WORK.DFF_1B(STRUCT_DFF_1B) PORT MAP(D(i),Clk,Reset,Q(i),NQ(i));
--                    END GENERATE FLOOP;
--END ARCHITECTURE STRUCT_NBIT_REGISTERS;



-------------------------------------------------------
----          n-Bit Carry Lookahead          --
-------------------------------------------------------
--LIBRARY IEEE;
--USE IEEE.STD_LOGIC_1164.ALL;
--
--ENTITY LOOK_AHEAD IS          --look ahead adder/subtractor faster than ripple adder/subtractor
--          GENERIC (N : POSITIVE := 8);
--    PORT (A,B: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0):= "00000000";
--         Cin: IN   STD_LOGIC:='0';
--         Sum: OUT  STD_LOGIC_VECTOR(N-1 DOWNTO 0):= "00000000";
--        Cout: OUT  STD_LOGIC:= '0');
--END ENTITY LOOK_AHEAD;
--
--ARCHITECTURE STRUCT_CARRY_LOOK_AHEAD OF LOOK_AHEAD IS
--          SIGNAL S,G,P: STD_LOGIC_VECTOR(N-1 DOWNTO 0):= "00000000";
--          SIGNAL C: STD_LOGIC_VECTOR(N-1 DOWNTO 1):= "00000000";
--          BEGIN
--             PROCESS (G,P,C)
--             BEGIN
--                    C(1) <= G(0) OR (P(0) AND Cin);
```

```
--              inst: FOR i IN 1 TO N-2 LOOP
--                  C(i+1) <= G(i) OR (P(i) AND C(i));
--                  END LOOP;
--          Cout <= G(N-1) OR (P(N-1) AND C(N-1));
--          END PROCESS;
--
--                  FLOOP1: FOR i IN 0 TO (N-1) GENERATE
--                                  G1: ENTITY WORK.XOR_2P(STRUCT_XOR_2P) PORT
MAP(A(i),B(i),S(i));
--                  END GENERATE FLOOP1;
--
--                  FLOOP2: FOR i IN 0 TO (N-1) GENERATE
--                                  G2: ENTITY WORK.AND_2P(STRUCT_AND_2P) PORT
MAP(A(i),B(i),G(i));
--                  END GENERATE FLOOP2;
--
--                  FLOOP3: FOR i IN 0 TO (N-1) GENERATE
--                                  G3: ENTITY WORK.OR_2P(STRUCT_OR_2P) PORT
MAP(A(i),B(i),P(i));
--                  END GENERATE FLOOP3;
--
--                  G4: ENTITY WORK.XOR_2P(STRUCT_XOR_2P) PORT MAP(S(0),Cin,Sum(0));
--          FLOOP4: FOR i IN 1 TO (N-1) GENERATE
--                                  G5: ENTITY WORK.XOR_2P(STRUCT_XOR_2P) PORT
MAP(S(i),C(i),Sum(i));
--                  END GENERATE FLOOP4;
--
--END ARCHITECTURE STRUCT_CARRY_LOOK_AHEAD;
```