Tribhuvan University

Prithivi Narayan Campus

Pokhara,Kaski

Lab-report of Computer Architecture

Subject code: CSC213

Submitted to:	Submitted by:
Amrita Thapa	Sabin Paudel
Department of CSIT	Roll no: 56
Prithivi Narayan Campus	3rd semester

LAB INDEX

Name: Sabin Paudel　　Roll. No: 56
Level: Bachelors　　Year/Sem: 2nd/3rd
Subject: Computer Architecture　　Faculty: B.Sc. CSIT
Course No: CSC213

Lab No	Exp NO	Experiment name	Date of experiment	Submitted Date	Remark
1		**SIMULATING LOGIC GATES**			
	1.1	Simulate the working of AND logic gate using VHDL	2080/12/05	2080/12/16	
	1.2	Simulate the working of OR logic gate using VHDL	2080/12/06	2080/12/16	
	1.3	Simulate the working of NOT logic gate using VHDL	2080/12/07	2080/12/16	

INTRODUCTION TO VHDL, GHDL, AND GTKWAVE

VHDL (VHSIC Hardware Description Language) serves as a pivotal tool in electronic design automation, offering a means to intricately describe digital and mixed-signal systems. By modeling electronic system behaviors, VHDL empowers designers to synthesize hardware components. Its applications span across digital circuit design, FPGA programming, ASIC design, and beyond.

KEY USES OF VHDL:

1. Digital Circuit Design: VHDL facilitates the creation of digital circuits like processors and controllers.
2. FPGA Programming: It is extensively used for programming FPGAs, enabling diverse digital functions.
3. ASIC Design: VHDL describes the behavior and structure of custom integrated circuits in ASIC design.
4. Verification: Designers leverage VHDL for simulation and testing to validate designs before implementation.

GHDL emerges as an open-source simulator tailored for VHDL, democratizing VHDL development and simulation. It translates VHDL code into executable binaries, offering a cost-effective alternative to proprietary tools.

GHDL APPLICATIONS:

1. VHDL Simulation: GHDL enables designers to simulate VHDL designs, facilitating functionality verification and code debugging.
2. Educational Purposes: It serves as a valuable tool in educational environments for teaching VHDL concepts and digital design principles.
3. Research and Prototyping: GHDL is employed in research and rapid prototyping scenarios, providing a budget-friendly avenue for VHDL development.

GTKWave complements GHDL by offering a user-friendly waveform viewer for analyzing simulation results.

USES OF GTKWAVE:

1. Waveform Visualization: Designers leverage GTKWave to visualize simulation waveforms, aiding in understanding digital design behaviors.
2. Debugging: GTKWave streamlines the debugging process with features like zooming and scrolling, facilitating issue identification and resolution.
3. Signal Analysis: It empowers designers to analyze signals and timing relationships, enhancing performance optimization and verification efforts.

Setting environment to use GHDL and GTKWAVE

- Download the required bin files related to GHDL and GITWAVE from GitHub and other sources.
- Create a folder named VHDL and put both files downloaded in that folder.
- Copy that folder into local disk C.
- Open edit the system environment variable setting from start menu
- On the advance option select the Environment Variables options on the bottom part.
- Copy the path of the the bin folder and paste it in the System Variable by selecting the new option
- Click on save/Ok button and exit.

Command used while performing this experiment

```
ghdl -a filename.vhdl

ghdl -a testbench_filename.vhdl

ghdl -e testbench_filename       (don't include extension of file)

ghdl -r testbench_filename --vcd=newfilename.vcd

gtkwave newfilename.vcd
```

IDE used: Notepad

EXPERIMENT NO: 1.1

SIMULATE THE WORKING OF AND LOGIC GATE USING VHDL

OBJECTIVE:

- To simulate the working of AND logic gate using VHDL.

THEORY

The objective of this experiment is to simulate the behavior of an AND logic gate using VHDL (VHSIC Hardware Description Language). GHDL (GNU Hardware Description Language) will be utilized for simulation, and GTKWAVE for waveform analysis. The experiment aims to comprehend the functioning of an AND logic gate and its application in digital circuit design. The AND gate, a basic digital logic component, yields a true output (logic 1) only when all inputs are true (logic 1). This experiment seeks to replicate the behavior of the AND gate to gain insights into its operation within digital circuits.

Input A	Input B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Demonstration

3.1.Program 1

To create a VHDL file for implementing AND logic.

Source Code:

```
library ieee;
use ieee.std_logic_1164.all;
entity andGate is
   port(
      a,b : in std_logic;
      c: out std_logic
   );
end andGate;
architecture behaviour of andGate is
   begin
   c<= a and b;
end behaviour;
```

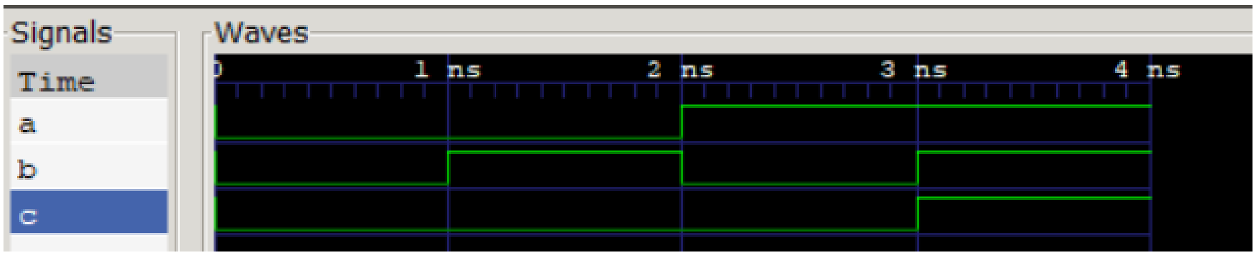
3.2 Program 2

To create a testbench for the AND logic VHDL file.

```
library ieee;
use ieee.std_logic_1164.all;
entity andGate_tb is
        end andGate_tb;
architecture test of andGate_tb is
        component andGate
         port(
        a,b : in std_logic;
        c: out std_logic
         );
```

```
          end component;
signal a,b,c : std_logic;
            begin
           andGate_inst: andGate port map
          (  a => a, b => b, c => c);
            process begin
              a <= '0';
              b <= '0';
              wait for 1 ns;
             a <= '0';
              b <= '1';
              wait for 1 ns;
              a <= '1';
              b <= '0';
              wait for 1 ns;
    a <= '1';
             b <= '1';
   wait for 1 ns;
            assert false report "successfully completed";
             wait;
            end process;
end test;
```

OUTPUT AND DISCUSSION



During simulation, VHDL code for the AND logic gate was executed, with inputs A and B varying between 0 and 1. Using GTKWAVE to analyze the waveform, we verified if the output matched the AND gate's truth table. The results confirmed the gate's functionality: output was true only when both inputs were true. This experiment underscores VHDL's role in simulating logic gates, offering insights into digital circuit design.

CONCLUSION

Through this experiment, we successfully simulated the behavior of an AND logic gate using VHDL along with GHDL and GTKWAVE. This exercise provided insights into digital circuit design and the implementation of basic logic gates using hardware description languages. The simulation results validated the functionality of the AND gate as per its truth table. Further experiments could involve simulating more complex digital circuits and exploring advanced VHDL features.

EXPERIMENT NO: 1.2

SIMULATE THE WORKING OF OR LOGIC GATE USING VHDL

OBJECTIVE:

- To simulate the working of OR logic gate using VHDL.

THEORY

The goal of this experiment is to emulate the functionality of an OR logic gate using VHDL, GHDL for simulation, and GTKWAVE for waveform analysis. By replicating the behavior of the OR gate, we aim to understand its role in digital circuit design. The OR gate, a foundational logic component, yields a true output (logic 1) when any input is true (logic This experiment aims to gain insights into the operation and integration of the OR gate within digital circuits.

Input A	Input B	Output
0	0	0
0	1	1
1	0	1
1	1	1

DEMONSTRATION

3.1.Program 1

To create a VHDL file for implementing OR logic.

Source Code:

```
library ieee;
use ieee.std_logic_1164.all;
entity orGate is
   port(
      a,b : in std_logic;
      c: out std_logic
   );
end orGate;
architecture behaviour of orGate is
   begin
   c<= a or b;
end behaviour;
```

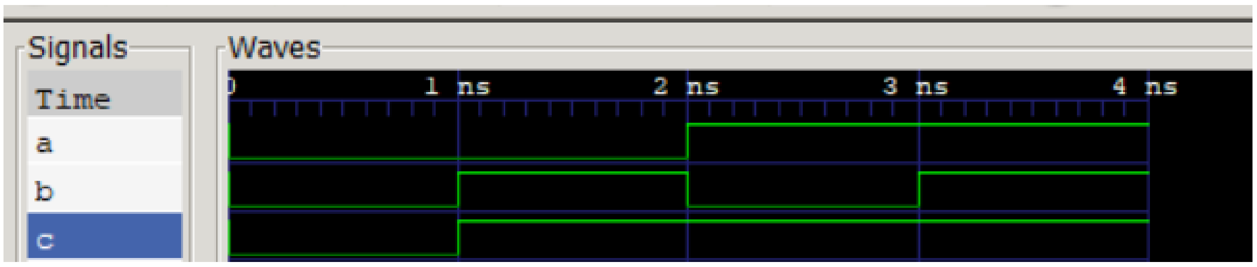
3.2 Program 2

To create a testbench for the OR logic VHDL file.

```
library ieee;
use ieee.std_logic_1164.all;
entity orGate_tb is
                  end orGate_tb;
architecture test of orGate_tb is
                  component orGate
                   port(
         a,b : in std_logic;
          c: out std_logic
                   );
```

```
                    end component;
signal a,b,c : std_logic;
begin                         orGate_inst: orGate port map
                    (  a => a, b => b, c => c);
                      process begin
            a <= '0';
            b <= '0';
          wait for 1 ns;
           a <= '0';
           b <= '1';
           wait for 1 ns;
                      a <= '1';
                      b <= '0';
                     wait for 1 ns;
          a <= '1';
          b <= '1';
          wait for 1 ns;
         assert false report "successfully completed";
         wait;
          end process;
end test;
```

OUTPUT AND DISCUSSION



We implemented an OR logic gate using VHDL, compiling the code with GHDL and simulating it to observe behavior. Various input combinations were applied and analyzed in GTKWAVE. Results confirmed the gate's functionality: output is true when at least one input is true. This experiment illustrates VHDL's role in simulating logic gates, offering insights into digital circuit design.

CONCLUSION

Through this experiment, we successfully simulated the behavior of an OR logic gate using VHDL in conjunction with GHDL and GTKWAVE. The simulation results validated the functionality of the OR gate as per its truth table. This exercise enhanced our understanding of digital circuit design principles and the application of VHDL for describing and simulating digital logic circuits.

EXPERIMENT NO: 1.3

SIMULATE THE WORKING OF NOT LOGIC GATE USING VHDL

OBJECTIVE:

- To simulate the working of NOT logic gate using VHDL.

THEORY

The objective of this experiment is to replicate the operation of an NOT logic gate utilizing VHDL. VHDL will be utilized alongside GHDL for simulating the gate's behavior, while GTKWAVE will be employed for analyzing waveforms.

This experiment seeks to gain insight into the behavior of an NOT logic gate and its integration within digital circuit design. NOT gate is a fundamental digital logic gate that produces an output true (logic 1) when the input is false (logic 0). It can be represented by the following truth table:

Input A	Output
0	1
1	0

3. Demonstration

3.1.Program 1

To create a VHDL file for implementing NOT logic.

Source Code:

```
library ieee;
use ieee.std_logic_1164.all;
entity notGate is
   port(
   a : in std_logic;
   b : out std_logic
);
end notGate;
architecture behaviour of notGate is
   begin
   b <= not a;
end behaviour;
```

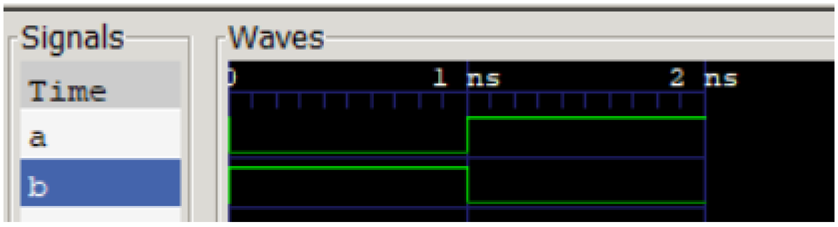
3.2 Program 2

To create a testbench for the NOT logic VHDL file.

```
library ieee;
use ieee.std_logic_1164.all;
entity notGate_tb is
                end notGate_tb;
architecture test of notGate_tb is
                component notGate
                port(
         a: in std_logic;
        b: out std_logic
                );
       end component;
signal a, b : std_logic;
```

```
begin
                    notGate_inst: notGate port map
                    (  a => a, b => b );
                    process begin
                    a <= '0';
                    wait for 1 ns;
                     a <= '1';
                    wait for 1 ns;
                     assert false report "successfully completed";
                    wait;
            end process;
end test;
```

OUTPUT AND DISCUSSION



In this experiment, we implemented a NOT logic gate using VHDL. The code was compiled with GHDL and simulated to observe its behavior. Various input combinations were tested, and the corresponding output was analyzed using GTKWAVE. Results confirmed the gate's functionality: it produces a true output when its input is false. This experiment sheds light on digital circuit design and VHDL's role in simulating logic gates.

CONCLUSION

Through this experiment, we successfully simulated the behavior of an NOT logic gate using VHDL in conjunction with GHDL and GTKWAVE. The simulation results validated the functionality of the OR gate as per its truth table. This exercise enhanced our understanding of digital circuit design principles and the application of VHDL for describing and simulating digital logic circuits.

EXPERIMENT NO: 2.1

SIMULATE THE WORKING OF NAND LOGIC GATE USING VHDL

OBJECTIVE:

- To simulate the working of NAND logic gate using VHDL.

THEORY

A NAND gate is a fundamental building block in digital electronics and is one of the universal gates, meaning that any other logic gate can be constructed using only NAND gates.

Truth Table for NAND gate:

A	B	OUTPUT
0	0	1
0	1	1
1	1	1
1	0	0

Where A and B are the input signals and output is the output signal.

The output Q is high (1) when both inputs A and B are low (0); otherwise, the output is low (0).

NAND gates can be built using various electronic components like transistors, diodes, or even mechanical switches in early implementations. Modern NAND gates are typically constructed using transistors, where combinations of transistors are arranged to perform the logical operations according to the truth table.

And we are simulating the operation using VHDL,GHDL and GTKwave.

Demonstration

Program no.1

To create a VHDL file for implementing NAND logic.

Source code:

```
library ieee;

use ieee.std_logic_1164.all;

entity nandGate is
   port(

        a,b : in std_logic;
        c,d: out std_logic
```

```
    );

end nandGate;

architecture behaviour of nandGate is
    begin

    c<= a and b;
    d<= a nand b;

end behaviour;
```

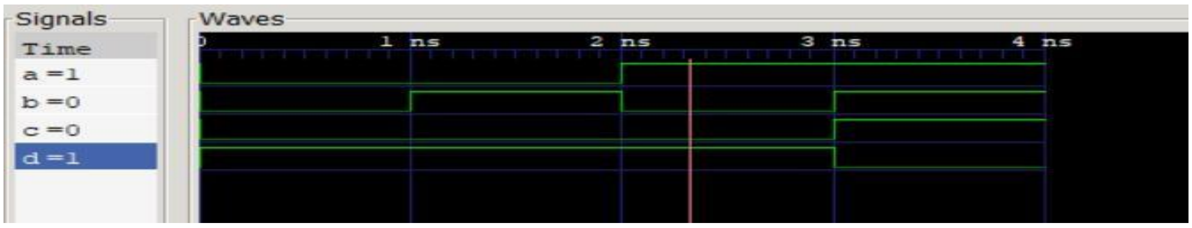
Program no.2

To create a testbench for the NAND logic VHDL file.

Source Code:

```
library ieee;

use ieee.std_logic_1164.all;

entity nandGate_tb is
    end nandGate_tb;

architecture test of nandGate_tb is
    component nandGate

    port(

        a,b : in std_logic;
        c,d : out std_logic

    );

    end component;

signal a,b,c,d : std_logic;
begin

    nandGate_inst: nandGate port map
    (  a => a, b => b, c => c, d => d);

    process begin

        a <= '0';

        b <= '0';

        wait for 1 ns;

        a <= '0';

        b <= '1';

        wait for 1 ns;

        a <= '1';

        b <= '0';

        wait for 1 ns;

        a <= '1';
        b <= '1';

        wait for 1 ns;

        assert false report "successfully completed";
        wait;

    end process; end test
```

OUTPUT AND DISCUSSION



The VHDL code for the NAND gate was simulated, and input combinations were tested to observe the output. Analysis of the waveform in GTKWAVE confirmed that the output matched the expected behavior from the AND gate's truth table. This validates the functionality of the NAND gate and highlights the utility of VHDL for digital circuit design and simulation.

CONCLUSION

Through this experiment, we successfully simulated the behavior of an NAND logic gate using VHDL along with GHDL and GTKWAVE. This exercise provided insights into digital circuit design and the implementation of basic logic gates using hardware description languages. The simulation results validated the functionality of the NAND gate as per its truth table. Further experiments could involve simulating more complex digital circuits and exploring advanced VHDL features

EXPERIMENT NO: 2.2

SIMULATE THE WORKING OF XOR LOGIC GATE USING VHDL

OBJECTIVE:

- To simulate the working of XOR logic gate using VHDL.

THEORY

An XOR gate, short for Exclusive OR gate, is a digital logic gate that performs the exclusive disjunction operation. It produces a true output (logic 1) only when the number of true inputs is odd.

The truth table for an XOR gate is as follows:

A	B	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	0

In other words, the output of an XOR gate is true if the number of true inputs is odd, and false otherwise. The symbol used to represent an XOR gate in circuit diagrams is ⊕ or sometimes the '⊻ ' symbol.And we are implementing this operation using VHDL.GHDL and GTKWAVE.

Demonstration

Program no. 1

To create a VHDL file for implementing XOR Gate.

Source code:

```
library ieee;

use ieee.std_logic_1164.all;

entity xorGate is
   port(

      a,b : in std_logic;

      c: out std_logic

   );

end xorGate;

architecture behaviour of xoGate is
   begin
```

```
                c<= a XOR b;

             end behaviour;
```

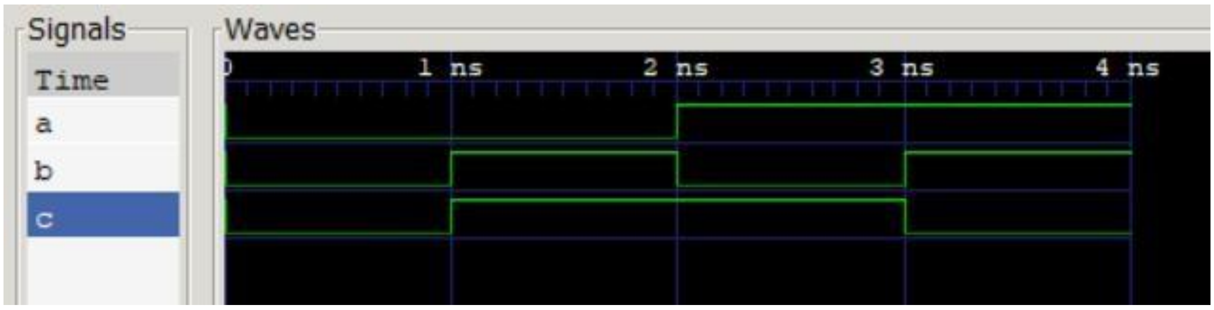
Program No. 2

To create a testbench for the half adder VHDL file.

Source code:

```
library ieee;

use ieee.std_logic_1164.all;

entity xorGate_tb is
    end xorGate_tb;

architecture test of xorGate_tb is
    component xorGate

    port(

        a,b : in std_logic;
        c : out std_logic

    );

    end component;

signal a,b,c : std_logic;
begin

    nandGate_inst: nandGate port map
    (  a => a, b => b, c => c);

    process begin

         a <= '0';

         b <= '0';

        wait for 1 ns;

        a <= '0';

        b <= '1';

        wait for 1 ns;

        a <= '1';

        b <= '0';

wait for 1 ns; a <= '1'b <= '1';wait for 1 ns;

assert false report "successfully completed"; wait;end process; end test;
```

OUTPUT AND DISCUSSION



In this experiment, we implemented an half adder using VHDL. The VHDL code describes the half adder behavior.After compiling the code with GHDL for correctness, we simulated various input combinations. Analyzing the waveform in GTKWAVE confirmed that the output matched the expected behavior from the half adder truth table. This validates the half adder’s functionality and offers insights into digital circuit design and VHDL's role in simulating logic gates.

CONCLUSION

Through this experiment, we successfully simulated the behavior of an XOR logic gate using VHDL along with GHDL and GTKWAVE. This exercise provided insights into digital circuit design and the implementation of basic logic gates using hardware description languages. The simulation results validated the functionality of the XOR gate as per its truth table. Further experiments could involve simulating more complex digital circuits and exploring advanced VHDL features.

EXPERIMENT NO: 2.3

SIMULATE THE WORKING OF HALF ADDER USING VHDL

OBJECTIVE:

- To simulate the working of Half-Adder using VHDL.

THEORY

A half adder is a fundamental digital circuit used in computer engineering and digital electronics. It's used to add two single-bit binary numbers together. The output of a half adder consists of two bits: the sum (S) and the carry (C). The sum represents the least significant bit of the addition, while the carry represents the most significant bit.

Truth Table for OR gate:

A	B	SUM(S)	CARRY(C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the truth table, you can see that the sum output (S) is the XOR of the inputs (A and B), while the carry output (C) is the AND of the inputs (A and B).

The logical expressions for the sum (S) and carry (C) outputs of a half adder can be written as follows:

S = A XOR B C = A AND B

Physically, a half adder can be implemented using basic logic gates such as AND and XOR gates. Typically, it requires two XOR gates and one AND gate.

And we are implementing this operation using VHDL.GHDL and GTKwave.

Demonstration

To create a VHDL file for implementing half adder.

Source code:

```
library ieee;

use ieee.std_logic_1164.all;
```

```
entity halfadder is

                    port(

                        a: in std_ulogic;

                         b: in std_ulogic;

                        cout: out std_ulogic;

                        S:out std_ulogic

                    );


end halfadder;

architecture behave of halfadder is
begin

          S<=(a xor b);
          cout<=(a and b);

                    end behave

;
```

Program No. 2

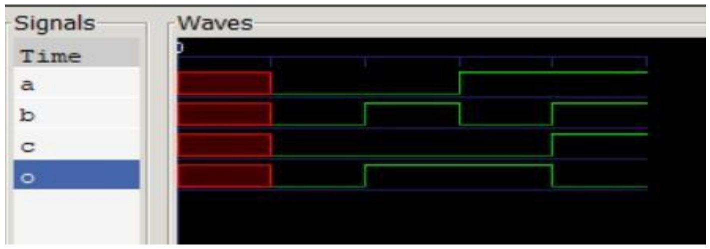
To create a testbench for the half adder VHDL file.

Source code:

```
library ieee;

use ieee.std_logic_1164.all;

entity halfadder_tb is
end ha_tb;

architecture test of halfadder_tb is
          component halfadder

                    port( a: in std_ulogic; b: in std_ulogic;

                              cout: out std_ulogic;
                              s: out std_ulogic);

          end component;

          signal ain, bin, cout, sum : std_logic;
          begin

                    half_adder: halfadder port map (a => ain, b => bin, cout => cout, s

=> sum);process begin
```

```
                    ain <= '0';

                    bin <= '0'; wait
                    for 1 ns;

                    ain <= '0';

                    bin <= '1'; wait
                    for 1 ns;

                    ain <= '1';

                    bin <= '0'; wait
                    for 1 ns;

                    ain <= '1';

                    bin <= '1'; wait
                    for 1 ns;end
                    test;assert
                    false report
                    "Completed";
                    wait;

  end process;
```

OUTPUT AND DISCUSSION



In this experiment, we implemented an half adder using VHDL. The VHDL code describes the half adder behavior.After compiling the code with GHDL for correctness, we simulated various input combinations. Analyzing the waveform in GTKWAVE confirmed that the output matched the expected behavior from the halfadder truth table. This validates the half adder’s functionality and offers insights into digital circuit design and VHDL's role in simulating logic gates.

CONCLUSION

In this experiment, we simulated an OR logic gate using VHDL with GHDL and GTKWAVE. The results confirmed the OR gate's functionality aligned with its truth table. This exercise deepened our grasp of digital circuit design principles and VHDL's role in describing and simulating logic circuits.

EXPERIMENT NO: 2.4

SIMULATE THE WORKING OF FULL ADDER USING VHDL

OBJECTIVE:

- To simulate the working of Full-Adder using VHDL.

THEORY

A full adder is another fundamental digital circuit used in computer engineering and digital electronics. It's used to add three single-bit binary numbers together: two inputs (A and B) and a carry input from the previous stage (C_in). The output of a full adder consists of two bits: the sum (S) and the carry out (C_out).

Here's the truth table for a full adder:

A	**B**	**Cin**	**Sum(S)**	**Cout**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

From the truth table, you can see that the sum output (S) is the XOR of the inputs (A, B, and C_in), while the carry out (C_out) is generated based on the inputs using logical expressions.

The logical expressions for the sum (S) and carry out (C_out) outputs of a full adder can be written as follows:

S = A XOR B XOR C_in

C_out = (A AND B) OR (C_in AND (A XOR B))

Physically, a full adder can be implemented using basic logic gates such as XOR, AND, and OR gates. Typically, it requires two XOR gates, two AND gates, and one OR gate.

And we are simulating this opertaion using VHDL,GHDL and GTKwave.

Demonstration

Program no. 1

To create a VHDL file for implementing full adder.

Source code:

```
library ieee;

use ieee.std_logic_1164.all;

entity fulladder is
port(

        a: in std_ulogic;

        b: in std_ulogic;

         cin: in std_ulogic;

        cout: out std_ulogic; S:out std_ulogic

);

end fulladder;

architecture behave of fulladder is
Begin

S<=(a xor b) xor cin;

cout<= ((a xor b) and cin) or (a and b);
end behave;
```

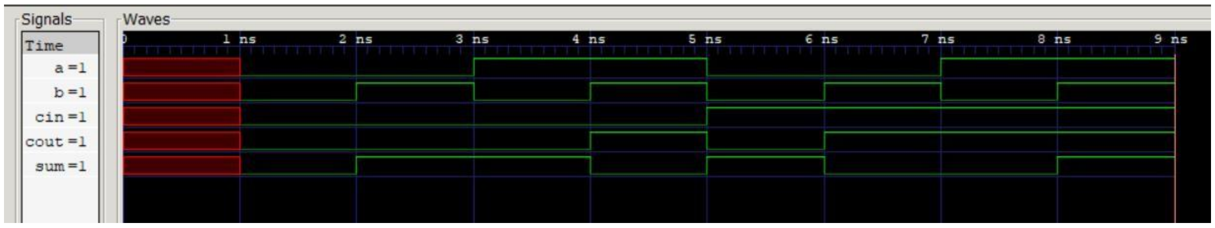
Program no. 2

To create a testbench for the full adder VHDL file.

Source code:

```
library ieee;

use ieee.std_logic_1164.all;
entity fulladder_tb is

end fulladder_tb;

architecture test of fulladder_tb is component fulladdder

                port(
                    a: in std_ulogic; b:
                    in std_ulogic; cin:
                    in std_ulogic;

                    cout: out std_ulogic;
                    s: out std_ulogic

                );

        end component;

        signal ain, bin, cin, cout, sum : std_logic; begin

                full_adder: fa port map (a => ain, b => bin, cin => cin, cout => cout,

    s => sum);

        process begin

                    ain <= '0';

                    bin <= '0';

                    cin <= '0'; wait
                    for 1 ns;

                    ain <= '0';

                    bin <= '0';

                    cin <= '1'; wait
                    for 1 ns;

                    ain <= '0';

                    bin <= '1';

                    cin <= '0'; wait
                    for 1 ns;

                    ain <= '0';

                    bin <= '1';

                    cin <= '1'; wait
                    for 1 ns;

                    ain <= '1';

                    bin <= '0';

                    cin <= '0'; wait
                    for 1 ns;

                    ain <= '1';
                    bin <= '0';

                    cin <= '1'; wait
                    for 1 ns;

                    ain <= '1';

                    bin <= '1';

                    cin <= '0'; wait
                    for 1 ns;

                    ain <= '1';

                    bin <= '1';

                    cin <= '1'; wait
                    for 1 ns;

            assert false report "Reached end of test"; wait;

    end process;

    end test;
```

OUTPUT AND DISCUSSION



In this experiment, we implemented a full adder using VHDL. The VHDL code describes the behavior of the full adder gate.After compiling the code with GHDL, we simulated various input combinations. Analyzing the waveform in GTKWAVE confirmed that the output matched the expected behavior from the full adder's truth table. This validates the full adder's functionality and offers insights into digital circuit design and VHDL's role in simulating logic gates.

CONCLUSION

In this experiment, we simulated a full adder using VHDL with GHDL and GTKWAVE. The results confirmed the gate's functionality aligned with its truth table. This exercise deepened our grasp of digital circuit design principles and VHDL's role in describing and simulating logic circuits.