Madan Bhandari Memorial College

Experiment 3: Introduction to VHDL

Objective:

1. To implement the Basic gates and universal gates using VHDL

Theory:

VHDL is one of the types of hardware description language which describes the behavior of an integrated circuit or system which is used to implement physical circuit or system. VHDL is an abbreviation for VHSIC which stands for Very High Speed Integrated Circuit Hardware Description Language. It is a standard, portable, reusable and vendor independent language and used primarily in designing hardware and describing its behavior, structure, and timing. VHDL program code can be used to implement the circuit in programmable device like Complex Programmable Logic Device (CPLD) , Field Programmable Gate Array (FPGA) or used for fabrication of an ASIC chips. It is a hardware description language that can be used to model a digital system at many levels of abstraction.

An entity can be described using:

i. Entity declaration

ii. Architecture

iii. Configuration

iv. Package declaration

v. Package body

Entity Declaration:

It defines the name, input/output signals and modes of hardware module. An entity is the most basic building block in the design.

The Syntax of an Entity is:

```
entity ENTITY_NAME is
port declaration;
end ENTITY_NAME
```

Architecture:

Architecture describes the behavior of the entity. An entity can have multiple architectures. Architecture assigned to an entity describes internal relationship between input and output of the entity. It can be described using structural, data flow, behavior or mixed type.

Syntax:

```
architecture architecture_name of entity_name is
architecture architecture_declarative part;
begin
statements;
end architecture_name;
logic operation: NAND gate
```

Architecture Program:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity And_gate is
Port ( A : in STD_LOGIC;
B : in STD_LOGIC;
y : out STD_LOGIC);
end And_gate;
architecture Behavioral of And_gate is
begin
y <= A and B;
end Behavioral;
```

Configuration - A configuration statement is used for binding a component instance to an entity architecture pair, when there are multiple architectures for a single entity. It is used during design instantiation (putting together the pieces) to map component instances to specific entity-architecture pairs.
Package - A package used to build the design using data types and sub programs which are commonly used. Packages are used for increasing code reusability, organization, and better management of design elements.
Library - Library is used to store entities, architectures and packages used in VHDL program .

Experiment i: AND Gate

Objective:

1. To implement the AND gate using VHDL

Theory:

AND gate is used to perform logical Multiplication of binary input. The Output state of the AND gate will be high(1) if both the input are high(1) ,else the output state will be low(0) if any of the input is low(0). The AND gate is one of the fundamental building blocks in digital electronics.

Truth Table for AND Gate:

Input A	Input B	Output (A AND B)
0	0	0
0	1	0
1	0	0
1	1	1

Source code: For AND gate

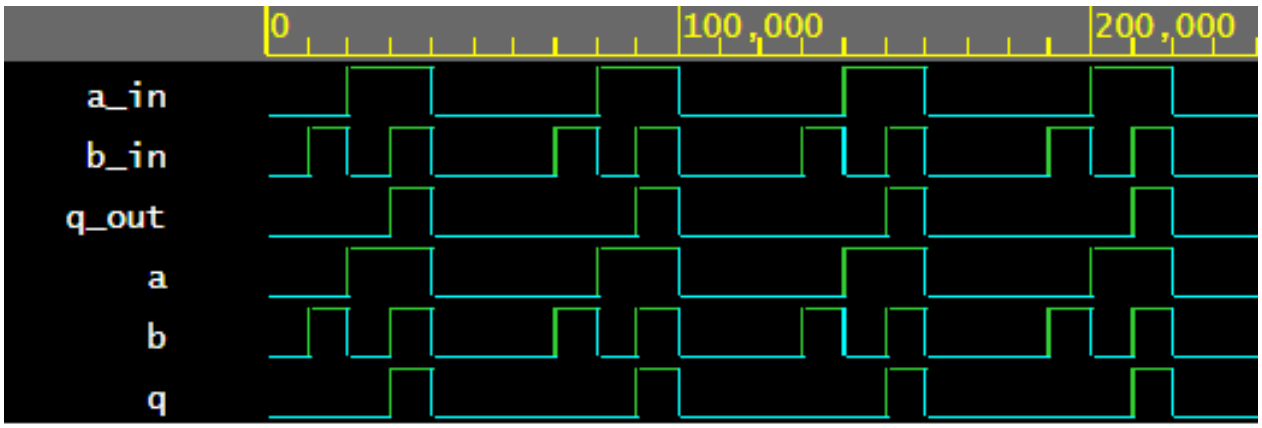
Testbench:

```
-- Testbench for AND gate
library IEEE;
use IEEE.std_logic_1164.all;

entity testbench is
-- empty
end testbench;

architecture tb of testbench is
-- DUT component
component and_gate is
port(
        a: in std_logic;
        b: in std_logic;
        q: out std_logic);
end component;
signal a_in, b_in, q_out: std_logic;
begin
  -- Connect DUT
  DUT: and_gate port map(a_in, b_in, q_out);
  process
  begin
    a_in <= '0';
    b_in <= '0';
    wait for 10 ns;

    a_in <= '0';
    b_in <= '1';
    wait for 10 ns;

    a_in <= '1';
    b_in <= '0';
    wait for 10 ns;

    a_in <= '1';
    b_in <= '1';
    wait for 10 ns;

    -- Clear inputs
    a_in <= '0';
    b_in <= '0';
    wait for 20 ns;
   end process;
  end tb;
```

Architecture:

```
-- Simple AND gate design
library IEEE;
use IEEE.std_logic_1164.all;
entity and_gate is
port(
  a: in std_logic;
  b: in std_logic;
  q: out std_logic);
end and_gate;
architecture rtl of and_gate is
begin
  process(a, b) is
  begin
    q <= a and b;
  end process;
end rtl;
```

Output:



Experiment ii: Half adder

Objective:

1. To implement the Half adder using VHDL

Theory:

A half adder is a basic digital circuit used to add two single-bit binary numbers. It has two inputs, typically labeled A and B, and two outputs: the sum (S) and the carry (C).

- **Sum (S)**: This is the result of the bitwise addition of A and B, without considering any carry from a previous operation. It can be found using the XOR (exclusive OR) operation.
- **Carry (C)**: This is the carry-out bit, which is generated when both A and B are 1, meaning there is an overflow into the next higher bit. It can be found using the AND operation.

Truth table:

A	B	Sum(S)	Carry(C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Source code:

Testbench:

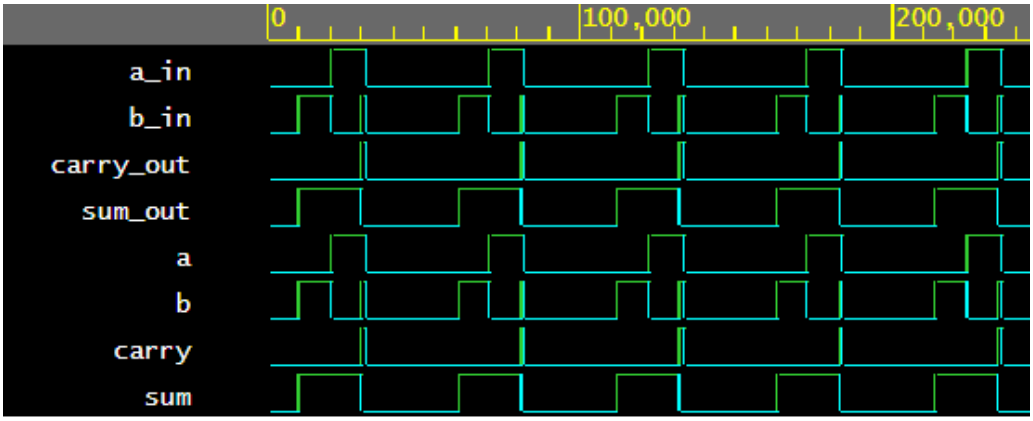
```
-- Testbench for halfadder gate
library IEEE;
use IEEE.std_logic_1164.all;

entity testbench is
-- empty
end testbench;

architecture tb of testbench is
-- DUT component
component halfadder_gate is
port(
  a: in std_logic;
  b: in std_logic;
  sum: out std_logic;
  carry:out std_logic);
end component;
signal a_in, b_in, sum_out, carry_out: std_logic;

begin

  -- Connect DUT
  DUT: halfadder_gate port map(a_in, b_in, sum_out, carry_out);

  process
  begin
    a_in <= '0';
    b_in <= '0';
    wait for 10 ns;

    a_in <= '0';
    b_in <= '1';
    wait for 10 ns;
    a_in <= '1';
    b_in <= '0';
    wait for 10 ns;
```

```
    a_in <= '1';
    b_in <= '1';
    wait for 10 ns;
    -- Clear inputs
    a_in <= '0';
    b_in <= '0';
    wait for 20 ns;
  end process;
end tb;
```

Architecture:

```
-- Simple halfadder gate design
library IEEE;
use IEEE.std_logic_1164.all;

entity halfadder_gate is
port(
  a: in std_logic;
  b: in std_logic;
  sum: out std_logic;
  carry:out std_logic);
end halfadder_gate;

architecture rtl of halfadder_gate is
begin
  process(a, b) is
  begin
    sum <= a xor b;
    carry <= a and b;
  end process;
end rtl;
```

Output:



Experiment ii: Full adder

Objective:

1. To implement the Full adder using VHDL

Theory:

A Full Adder is a digital circuit that computes the sum of three binary inputs: two operands and a carry-in bit. It produces a sum and a carry-out bit.

The Full Adder is an extension of the Half Adder, which only adds two binary bits. In the Full Adder, we also consider a carry from the previous addition, making it more suitable for multi-bit binary addition, such as adding numbers in computer arithmetic. The output of full adder are:

1. **Sum (S)** (The result of adding A, B, and Carry-in)
2. **Carry-out (Cout)** (Carry to the next bit position)

Truth table:

A	B	Cin	Sum(S)	Carry(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

Source code:

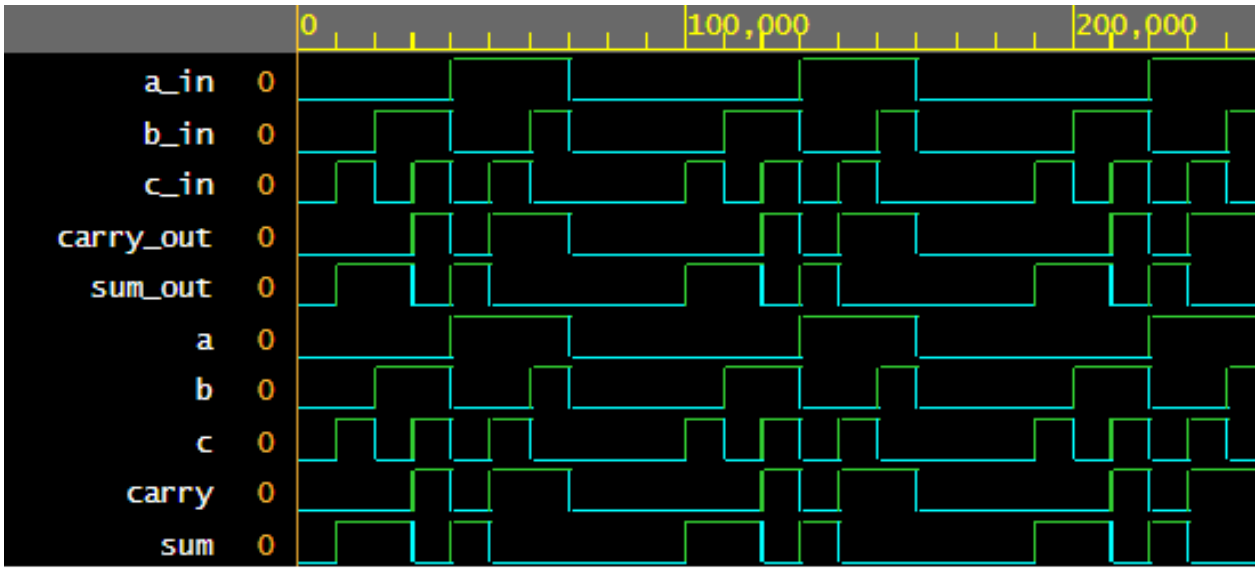
```
-- Testbench for fulladder gate
library IEEE;
use IEEE.std_logic_1164.all;

entity testbench is
-- empty
end testbench;
architecture tb of testbench is
-- DUT component
component fulleladder_gate is
port(
  a: in std_logic_vector(3 downto 0);
  b: in std_logic_vector(3 downto 0);
```

```
  c:in std_logic;
  sum: out std_logic_vector(3 downto 0);
  carry:out std_logic);
end component;

signal a_in,b_in:std_logic_vector, c_in:std_logic;
signal sum_out:std_logic_vector(3 downto 0),carry_out:std_logic;

begin
  -- Connect DUT
  DUT: fulladder_gate port map(a_in, b_in, c_in, sum_out, carry_out);
  process
  begin
    a_in <= '0';
    b_in <= '0';
    c_in <= '0';
    wait for 10 ns;

    a_in <= '0';
    b_in <= '0';
    c_in <= '1';
    wait for 10 ns;

    a_in <= '0';
    b_in <= '1';
    c_in <= '0';
    wait for 10 ns;

    a_in <= '0';
    b_in <= '1';
    c_in <= '1';
    wait for 10 ns;
    a_in <= '1';
    b_in <= '0';
    c_in <= '0';
    wait for 10 ns;

    a_in <= '1';
    b_in <= '0';
    c_in <= '1';
    wait for 10 ns;
    a_in <= '1';
    b_in <= '1';
    c_in <= '0';
    wait for 10 ns;
    a_in <= '1';
    b_in <= '1';
    c_in <= '1';
     wait for 10 ns;
    -- Clear inputs
    a_in <= '0';
    b_in <= '0';
          c_in <= '0';
    wait for 20 ns;
  end process;
end tb;
```

Architecture:

```
-- Simple fulladder gate design
library IEEE;
use IEEE.std_logic_1164.all;

entity fulladder_gate is
port(
  a: in std_logic;
  b: in std_logic;
  c: in std_logic;
  sum: out std_logic;
  carry:out std_logic);
end fulladder_gate;

architecture rtl of fulladder_gate is
begin
  process(a, b, c) is
  begin
    sum <= a xor b xor c;
    carry <= (a and b) or (a and c) or (b and c);
  end process;
end rtl;
```

Output:



Experiment iv: 4-bit parallel adder

Objective:

1. To implement the 4-bit parallel adder using VHDL

Theory:

A 4-bit parallel adder is a digital circuit that adds two 4-bit binary numbers simultaneously, producing a 4-bit sum and a carry-out bit. It is made up of multiple full adders connected in parallel. Each full adder in the circuit adds corresponding bits of the two 4-bit inputs, along with any carry from the previous stage.

Source code:

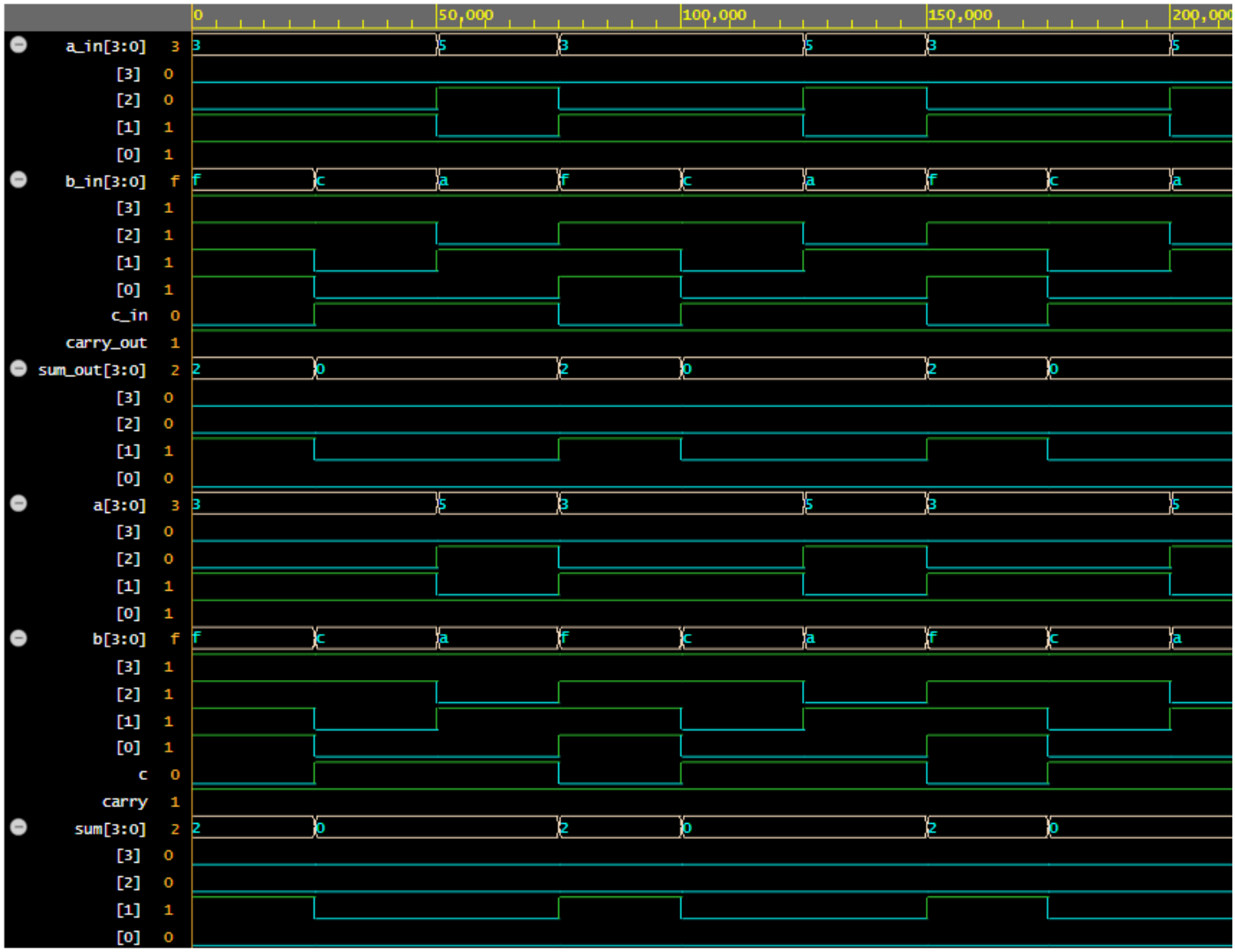
```
-- Testbench for paralleladder gate
library IEEE;
use IEEE.std_logic_1164.all;

entity testbench is
-- empty
end testbench;

architecture tb of testbench is

-- DUT component
component paralleladder_gate is
port(
  a: in std_logic_vector(3 downto 0);
  b: in std_logic_vector(3 downto 0);
  c:in std_logic;
  sum: out std_logic_vector(3 downto 0);
  carry:out std_logic);
end component;

signal a_in,b_in:std_logic_vector(3 downto 0):=(others=>'0'); signal c_in:std_logic;
signal sum_out:std_logic_vector(3 downto 0):= (others=>'0');
signal carry_out:std_logic;
begin
  -- Connect DUT
  DUT: paralleladder_gate port map(a_in, b_in, c_in, sum_out, carry_out);
process
  begin
  --test 1
    a_in <="0011";
    b_in <="1111";
    c_in <='0';
    wait for 10 ns;
    --test 2
    a_in <="0011";
    b_in <="1100";
    c_in <='1';
    wait for 10 ns;
    --test 3
    a_in<="0101";
    b_in <="1010";
    c_in <='1';
    wait for 20 ns;
  end process;
end tb;
```

Architecture:

```
-- Simple parallel adder gate design
library IEEE;
use IEEE.std_logic_1164.all;

entity paralleladder_gate is
```

```
port(
  a: in std_logic_vector(3 downto 0);
  b: in std_logic_vector(3 downto 0);
  c: in std_logic;
  sum: out std_logic_vector(3 downto 0);
  carry:out std_logic);
end paralleladder_gate;

architecture rtl of paralleladder_gate is
begin
  process(a, b, c) is
  variable temp: std_logic;
  begin
  temp:= c;
  for i in 0 to 3 loop
    sum(i) <= a(i) xor b(i) xor temp;
    temp:=(a(i) and B(i)) or (temp and a(i)) or (temp and b(i));
    end loop;
    carry<=temp;
  end process;
end rtl;
```

Output:



Experiment v: Encoder

Objective:
1. To implement encoder using VHDL

Theory:
An encoder is a digital circuit that converts data from one format or code to another, typically from a larger set of inputs to a smaller set of outputs. In the context of binary systems, an encoder is a device that converts an active input line into a binary code corresponding to that input.

For example, in a binary encoder**,** the number of output bits is fewer than the number of input lines. The encoder "encodes" the input into a binary number representing the position of the active input line.

Source code:

Encoder:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
ENTITY Encoder_test IS
END Encoder_test;
ARCHITECTURE behavior OF Encoder_test IS
 COMPONENT Encoder4_2
 PORT (a: IN std_logic_vector(3 downto 0);
 b: OUT std_logic_vector(1 downto 0)
 );
 END COMPONENT;
signal a: std_logic_vector(3 downto 0) := (others => '0');
signal b: std_logic_vector(1 downto 0);
BEGIN
uut: Encoder4_2 PORT MAP (a => a,b => b);
process
begin
 wait for 100 ns;
 a <= "0000";
 wait for 100 ns;
 a <= "0001";
 wait for 100 ns;
a <= "0010";
 wait for 100 ns;
 a <= "0100";
 wait for 100 ns;
 a <= "1000";
 wait;
 end process;
```

end;

Architecture:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Encoder4_2 is
 Port ( a: in STD_LOGIC_VECTOR (3 downto 0);
 b: out STD_LOGIC_VECTOR (1 downto 0));
end Encoder4_2;
architecture rtl of Encoder4_2 is
begin
 process(a)
 begin
 if (a="0001") then
 b <= "00";
 elsif (a="0010") then
 b <= "01";
 elsif (a="0100") then
 b <= "10";
 elsif (a="1000") then
 b <= "11";
 else
 b <= "ZZ";
 end if;
 end process;
end rtl;
```

Output: