**Experiment / Assignment / Tutorial No. \_\_\_7\_\_\_**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Batch: B3 Roll No.: 121 Experiment / assignment / tutorial No.: 7** |

|  |
| --- |
| **Title:** VHDL Programming for Gates |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Objective:** Implement a simpleOR, AND, XOR, NOR, NAND, XNOR gate in VHDL.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO4:** Implement digital networks using VHDL.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

* ModelSim Software Link:

https://www.mentor.com/company/higher\_ed/modelsim-student-edition

* J. Bhasker, “VHDL Primer”, Pearson Education
* Douglas L. Perry, “VHDL Programming by Example”, Tata McGraw Hill
* http://esd.cs.ucr.edu/labs/tutorial/

**Pre Lab/ Prior Concepts:**

VHDL is an acronym for VHSIC Hardware Description Language (VHSIC is an acronym for Very High Speed Integrated Circuits). It is a hardware description language that can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the gate level. The complexity of the digital system being modeled could vary from that of a simple gate to a complete digital electronic system, or anything in between. The digital system can also be described hierarchically. Timing can also be explicitly modeled in the same description.

**VHDL Programming Structure**

Entity and Architecture are the two main basic programming structures in VHDL.

Entity: Entity can be seen as the black box view of the system. The inputs and outputs of the system are defined which need to be interfaced. It is used to declare the I/O ports of the circuit.

E.g.

Entity myand is

Port

(

a: in std\_logic;

b: in std\_logic;

c: out std\_logic

);

End myand;

Entity name “myand” is given by the programmer. Each entity must have a name.

Architecture: Architecture defines what is in the black box that was described using Entity. The description code resides within architecture portion. Either behavioral or structural models can be used to describe our system in the architecture. In architecture we will have interconnections, components, processes, etc.

E.g.,

architecture myand\_a of myand is

-- declarations

begin

-- statements

c <= a and b;

end architecture myand\_a;

Entity name or architecture name is user defined. Identifiers can have uppercase alphabets, lowercase alphabets, numbers and underscore. The first letter of identifier must be an alphabet and an identifier cannot end with an underscore. In VHDL, keywords and identifiers are case insensitive.

VHDL is a strongly typed language i.e., every object must be declared. Standardized design libraries are typically used and are included prior to the entity declaration. This is accomplished by including the code “library ieee;” and “use ieee.std\_logic\_1164.all;”.

**Implementation Details**

1. Half Adder

Entity

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity myhalfadder is

Port

(

a, b: in std\_logic;

s: out std\_logic;

c: out std\_logic

);

end myhalfadder;

architecture myhalfadder\_a of myhalfadder is

begin

s <= a xor b;

c <= a and b;

end myhalfadder\_a;

Test bench

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity myhalfadder\_tb is

end myhalfadder\_tb;

architecture myhalfadder\_tb\_a of myhalfadder\_tb is

component myhalfadder is

Port

(

a, b: in std\_logic;

s: out std\_logic;

c: out std\_logic

);

end component;

signal a: std\_logic:= '0';

signal b: std\_logic:= '0';

signal s: std\_logic;

signal c: std\_logic;

begin

uut: myhalfadder port map(a => a, b => b, s => s, c => c);

stim\_proc: process

begin

wait for 10 ns;

a <= '1';

b <= '0';

wait for 10 ns;

a <= '0';

b <= '1';

wait for 10 ns;

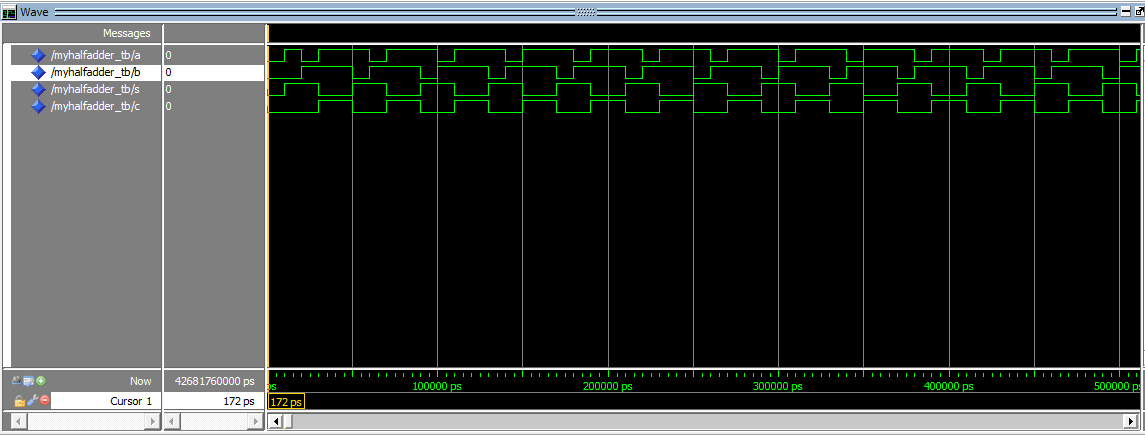
a <= '1';

b <= '1';

wait for 10 ns;

end process;

end myhalfadder\_tb\_a;

Output:

1. Full Adder

Entity

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity myfulladder is

Port

(

a, b, cin: in std\_logic;

s: out std\_logic;

cout: out std\_logic

);

end myfulladder;

architecture myfulladder\_a of myfulladder is

begin

s <= a xor b xor cin;

cout <= (a and b) or (cin and a) or (cin and b);

end myfulladder\_a;

Test bench

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity myfulladder\_tb is

end myfulladder\_tb;

architecture myfulladder\_tb\_a of myfulladder\_tb is

component myfulladder is

Port

(

a, b, cin: in std\_logic;

s: out std\_logic;

cout: out std\_logic

);

end component;

signal a: std\_logic:= '0';

signal b: std\_logic:= '0';

signal cin: std\_logic:= '0';

signal s: std\_logic;

signal cout: std\_logic;

begin

uut: myfulladder port map(a => a, b => b, s => s, cin => cin, cout => cout);

stim\_proc: process

begin

a <= '1';

b <= '0';

cin <= '0';

wait for 10 ns;

a <= '0';

b <= '1';

cin <= '0';

wait for 10 ns;

a <= '1';

b <= '1';

cin <= '0';

wait for 10 ns;

a <= '0';

b <= '0';

cin <= '1';

wait for 10 ns;

a <= '1';

b <= '0';

cin <= '1';

wait for 10 ns;

a <= '0';

b <= '1';

cin <= '1';

wait for 10 ns;

a <= '1';

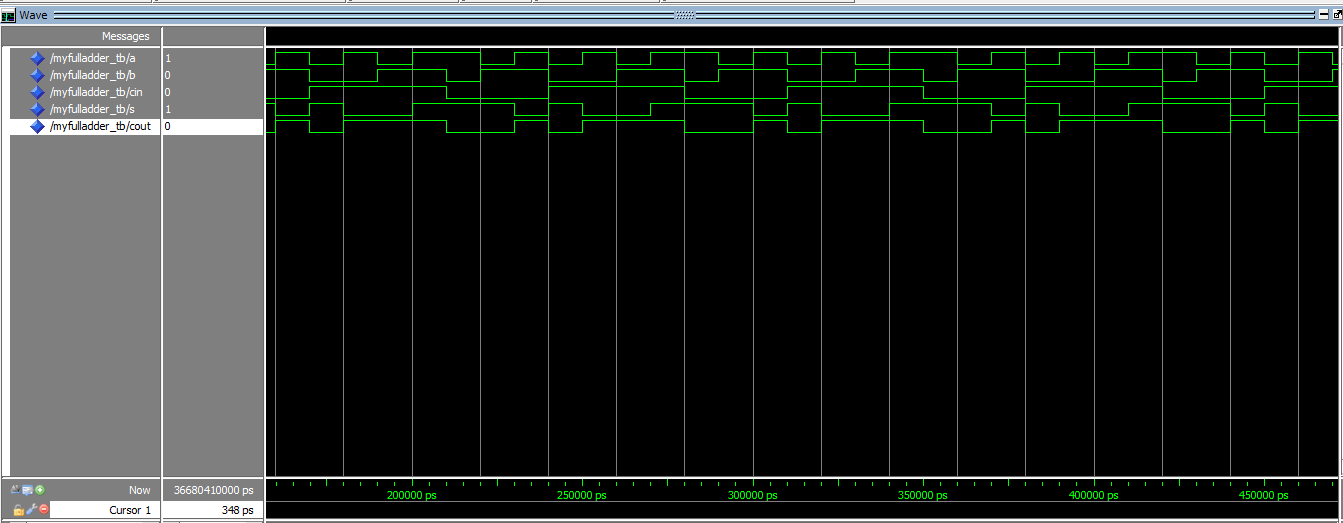
b <= '1';

cin <= '1';

wait for 10 ns;

end process;

end myfulladder\_tb\_a;

Output:

**Conclusion:**

Thus, in this experiment, the concept of VHDL programming for the implementation of basic and derived gates was learnt and implemented on Quartus II and Model-Sim Altera software.

**Post Lab Descriptive Questions**

1. What are two types of HDL?

Ans. The two types of Hardware Description Language are:

* 1. VHDL – Very High Speed Integrated Circuit HDL. It is strongly typed language, easier to understand, more natural in use, wordy and has a non-C like syntax.
  2. Verilog – It is weakly typed, has lesser code to write, is more of a hardware modelling language, succinct and similar to C language.