**Experiment / Assignment / Tutorial No. 9**

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

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

| Batch: B1 | Roll No.: 1711072 | Experiment / assignment / tutorial No.: 9 |
|---|---|---|

**Title:** VHDL programming for gates

---

**Objective:** Implements a simple OR, AND, XOR, NOR, NAND, XNOR gate in VHDL

---

**Expected Outcome of Experiment:**

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

---

**Books/ Journals/ Websites referred:**


- 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 VHSlC 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. We define the inputs and outputs of the system which we need to interface. It is used to declare the I/O ports of the circuit.
Eg:

```
Entity ANDGATE is
        Port (A: in std_logic;
```

```
                B: in std_logic;
                Y: out std_logic);
End entity ANDGATE;
```

Entity name ANDGATE is given by the programmer, each entity must have a name.

**Architecture:** Architecture defines what is in our black box that we 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, processes, components, etc.

Eg:

```
Architecture AND1 of ANDGATE is
        --declarations
Begin
        --statements
        Y <= A AND B;
End architecture AND1;
```

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

VHDL is 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 (in VHDL):**

**VHDL program code and simulation output**

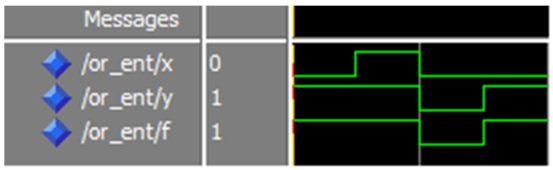
**OR**

**Code:**

```
library ieee;
use ieee.std_logic_1164.all;
entity OR_ent is
port( x: in std_logic;
      y: in std_logic;
      F: out std_logic
);
```

```
end OR_ent;

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

architecture OR_arch of OR_ent is
begin

    process(x, y)
    begin
        -- compare to truth table
        if ((x='0') and (y='0')) then
          F <= '0';
      else
          F <= '1';
      end if;
    end process;

end OR_arch;
```

**Output:**



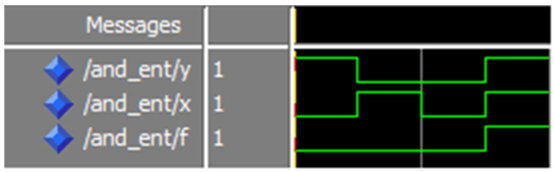

**AND**

**Code:**

```
library ieee;
use ieee.std_logic_1164.all;
entity AND_ent is
port( x: in std_logic;
      y: in std_logic;
      F: out std_logic
);
end AND_ent;


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

```
architecture AND_arch of AND_ent is
begin

    process(x, y)
    begin
        -- compare to truth table
        if ((x='1') and (y='1')) then
          F <= '1';
      else
          F <= '0';
      end if;
    end process;

end AND_arch;
```

**Output:**



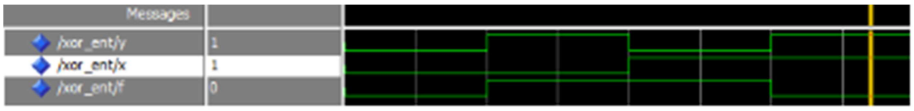

**XOR**

**Code:**

```
library ieee;
use ieee.std_logic_1164.all;
entity XOR_ent is
port( x: in std_logic;
      y: in std_logic;
      F: out std_logic
);
end XOR_ent;

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

architecture XOR_arch of XOR_ent is
begin

    process(x, y)
    begin
```

```
        -- compare to truth table
        if (x=y) then
          F <= '0';
      else
          F <= '1';
      end if;
    end process;

end XOR_arch;
```

**Output:**




**NOT**

**Code:**

```
library ieee;
use ieee.std_logic_1164.all;
entity NOT_ent is
port( x: in std_logic;
      F: out std_logic
);
end NOT_ent;
---------------------------------------
architecture NOT_arch of NOT_ent is
begin

    process(x)
    begin
        -- compare to truth table
        if (x='1') then
          F <= '0';
      else
          F <= '1';
      end if;
    end process;

end NOT_arch;
```

**Output:**


**NOR**

**Code:**

```
library ieee;
use ieee.std_logic_1164.all;
entity NOR_ent is
port( x: in std_logic;
      y: in std_logic;
      F: out std_logic
);
end NOR_ent;

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

architecture NOR_arch of NOR_ent is
begin

    process(x, y)
    begin
        -- compare to truth table
        if ((x='0') and (y='0')) then
          F <= '1';
      else
          F <= '0';
      end if;
    end process;

end NOR_arch;
```

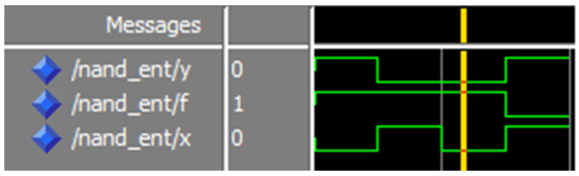
**Output:**

**NAND**

**Code:**

```
library ieee;
use ieee.std_logic_1164.all;
entity NAND_ent is
port( x: in std_logic;
      y: in std_logic;
      F: out std_logic
);
end NAND_ent;

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

architecture NAND_arch of NAND_ent is
begin

    process(x, y)
    begin
        -- compare to truth table
        if ((x='1') and (y='1')) then
          F <= '0';
      else
          F <= '1';
      end if;
    end process;

end NAND_arch;
```

**Output:**

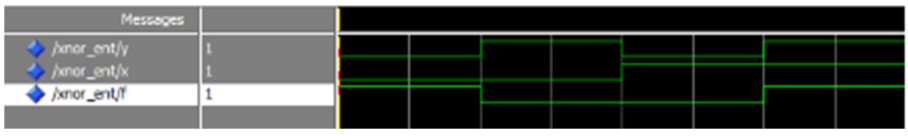


**XNOR**

**Code:**

```
library ieee;
use ieee.std_logic_1164.all;
entity XNOR_ent is
port( x: in std_logic;
      y: in std_logic;
      F: out std_logic
);
end XNOR_ent;

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

architecture XNOR_arch of XNOR_ent is
begin

    process(x, y)
    begin
        -- compare to truth table
        if (x=y) then
          F <= '1';
      else
          F <= '0';
      end if;
    end process;

end XNOR_arch;
```

**Output:**




**Conclusion: The programs were executed successfully as we were able to simulate all the seven gates.**

**Post Lab Descriptive Questions**

1. What are two types of HDL?

**Ans.** Hardware description language (HDL) is a specialized computer language used to program electronic and digital logic circuits. The structure, operation and design of the circuits are programmable using HDL. HDL includes a textual description consisting of operators, expressions, statements, inputs and outputs. Instead of generating a computer executable file, the HDL compilers provide a gate map. The gate map obtained is then downloaded to the programming device to check the operations of the desired circuit. The language helps to describe any digital circuit in the form of structural, behavioral and gate level and it is found to be an excellent programming language for FPGAs and CPLDs.

The two types of HDL are:

- VHDL
- Verilog