Write program to implement some basic logic gates and combinational circuits using VHDL.

VHDL

Description

- V: VHSIC (Very High Speed Integrated Circuit)
- H: Hardware
- D: Description
- L: Language
- VHDL (VHSIC Hardware Description Language) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits.
- VHDL can also be used as a general purpose parallel programming language.
- VHDL is an industry standard hardware description language that is widely used for specifying, modeling, designing, and simulating digital systems.

VHDL Code

- Every piece of VHDL code is composed of at least three fundamental sections
- Library declarations: contains a list of all libraries to be used in the design.
- Entity: specifies the I/O pins of the circuit

- Architecture: Contains the VHDL code which describes how the circuit should behave (function)

Library

- A library is a collection of commonly used pieces of code.
- Placing such pieces inside a library allows them to be reused or shared by other designs.
- To declare a library (that is to make it visible to the design) two lines of code are needed, one containing the name of the library, and the other a use clause.
- LIBRARY library_name;
- Use Library_name.package_name.package_parts;

Entity

- An ENTITY is a list with specifications of all input and output pins (ports) of the circuit with the following Syntax

```
ENTITY entity_name IS
PORT (port_name : signal_mode signal_type;
port_name : signal_mode signal_type;
...);
END entity_name;
```

- The mode of the signal can be IN, OUT, INOUT, or BUFFER
- IN and OUT are truly unidirectional pins, while INOUT is bidirectional
- BUFFER is employed when the output signal must be used (read) internally
- The name of the entity can be basically any name, except

VHDL reserved words

Input-Output

- Let my-ckt consists of following input & outputs
- Inputs: A, B, C
- Outputs: X, Y
- VHDL description is:

```
entity my-ckt is
port (
A in bit;
B in bit;
C in bit;
X: out bit;
Y: out bit;

End my-ckt;
```

- Similarly for a two input NAND gate:

```
ENTITY nand_gate IS
PORT (a,b IN BIT;
      x: OUT BIT);
END nand_gate;
```

Architecture

- The ARCHITECTURE denotes the description of how the circuit should behave or function.
- The syntax is as:

```
ARCHITECTURE architecture_name OF entity_name IS
                                    [declarations]
BEGIN
 (code)
END architecture_name;
```

- An architecture has two parts: a declarative part (optional), where signals and constants (among others) are declared, and the code part (from BEGIN down)

Inverter in VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164 ALL;
entity invert_top is
      Port (PB: in STD_LOGIC;
            LED: out STD_LOGIC);
end invert_top;
```

```
architecture Behavioral of invert_top is
begin
    -- invert the signal from the push button switch and
       route it to the LED
   LED <= not PB;
end Behavioral;
```

AND gate

```
library ieee;
use ieee.std_logic_1164.all;

entity and_gate is

port (a: in std_logic; -- AND gate input
 b: in std_logic; -- AND gate input
 y: out std_logic); -- AND gate output
end entity;

architecture behavioral of and_gate is
 begin

y <= a and b; -- two input AND gate

end behavioral;
```

OR gate

```
library ieee;
use ieee.std_logic_1164.all;

entity or_gate is

port (x1: in std_logic; -- OR gate input
x2: in std_logic; --OR gate input
y0: out std_logic); -- OR gate output

end or_gate;

architecture behavioral of or_gate is
begin

y0<= x1 or x2; -- two input OR gate

end behavioral;
```

Multiplexer

```
library ieee;
use ieee.std_logic_1164.all;

entity mux is

port (a,b,c,d,s0,s1: in std_logic;
y: out std_logic);

end entity;

architecture pure_logic of mux is

begin

y <= (a and not s1 and not s0) or
     (b and not s1 and s0) or
     (c and s1 and not s0) or
     (d and s1 and s0);

end pure_logic;
```

Full Adder

```
Library ieee;
use ieee.std_logic_1164.all;

entity full_adder is
port (a,b,c: in bit; sum, carry: out bit);
end entity;

architecture out_data of full_adder is
begin
sum <= a xor b xor c;
carry <= ((a and b) or (b and c) or (a and c));
end out_data;
```

Half Subtractor

```
Library ieee;
use ieee.std_logic_1164.all;

entity half_sub is
port (a, c : in bit;
d, b : out bit);
end entity;

architecture data of half_sub is
begin
d <= a xor c;
b <= (a and (not c));
end data;
```