JAYPEE UNIVERSITY OF ENGINEERING & TECHNOLOGY, GUNA
DEPARMENT OF COMPUTER SCIENCE & ENGINEERING
Course: Computer Organization & Architecture Lab
Course Code: 14B17CI673
B. Tech. (CSE VI Sem.)

Experiment # 1

Aim: Introduction to VHDL and Xilinx ISE Software

VHDL: It is a description language that describes the digital circuits by their functions, data flow behaviour and/or structures. The **VHDL** stands for ***VHSIC (Very High Speed Integrated Circuits) Hardware Description Language*.** This means VHDL can be used to accelerate the design process of digital circuits. It is very important to note that **VHDL is NOT a programming language** but it is an **HDL (Hardware Description Language)** which describes asynchronous, synchronous, combinational and sequential digital circuits. This language is used to build a prototype to discover complexity, problems and faults in the design before actually implementing it in hardware. This hardware description is used to configure a Programmable Logic Device (PLD), such as a Field Programmable Gate Array (FPGA), with a custom logic design. FPGA is a semiconductor devices built around a matrix of Configurable Logic Blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after its manufacturing. After the design is created, it is simulated and synthesized to check its logical operation. SIMULATION is the test to see if the basic logic works according to design and concept. SYNTHESIS allows timing factors and other influences of actual FPGA devices to effect the simulation thereby doing a more thorough type of check before the design is committed to the FPGA or similar device.

Basic VHDL Programming Structure

- **Library**: Contains the list of libraries used in the project. To include the library two line code is needed. First line contains the **library name** and the second line includes the **use** clause. The IEEE library and the following three packets appear by default in any source VHDL code with Xilinx ISE tool.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

The first two lines import the IEEE standard logic library which contains the predefined logic functions and data types. The **use** statement determines which portion of a library file to use. For example, all the items are selected in the 1164 library by including second line in the program.

- **Entity:** The basic design element in VHDL is called 'Entity'. It represents a template for a hardware block and describes the signals, called **port**, which connects this hardware to the outside. The ports of an entity can be input ports (**in**), output ports (**out**), bidirectional ports (**inout**) and buffered output port (**buffer)** that can be read by the architecture body. An entity may include a set of **generic** values that declare properties and constant of the circuits. It is used to define delays in signals and clock cycles. **Generics declaration are optional and used only when there is need to pass the specific information into an entity.** The syntax for entity declaration is as following:-

```
entity Entity Name is
generic (Generic Name : Generic Type := Value);
port (Port Name : Mode type);
end Entity Name;
```

- **Architecture:** An architecture, always associated with an entity, is a specification of the design's internal implementation and defines how the entity operates by including a set of inner signals, functions and procedures. Multiple architectures can be associated with an particular entity each optimized with design goals i.e. performance, area, power consumption and ease of simulation. Architecture can be modelled in four styles:
  - **Dataflow-** Provides logical and arithmetic operators. In this style, program statements remain concurrent, order independent and executed in parallel. This is useful for modelling of small combinational logic circuits.
  - **Behavioral-** Allows modelling with at least one **process** statement using ***if-then-elsif***, ***case*** etc. In this style, program statements are order dependent and executed sequentially inside a process. This modeling is used for both combinatorial and sequential circuits.
  - **Structural-** This modeling is used when complete hardware of the system being modeled is known and consists of the **components** and their inter connection. Structural modeling is used to model the combinatorial circuits
  - **Mixed-**Any combination of the above three styles of the modeling

  The syntax for architecture declaration is as following:-

```
architecture Architecture Name of Entity Name is
architecture/component declarations;
signal declaration;
begin
concurrent statements defining design operation;
process(sensitivity list*)                          }
begin                                               }  This requires only in
conditional statements for design operation;        }  behavioral modeling
end process;                                        }
end Architecture Name;
```

*Sensitivity list includes **all input variables** for modelling of combinational logic and **only the clock** for sequential logic modelling.

- **Component:** An entity-architecture pair actually describes a component type which is **used in the structural style of modeling**. In a design there may be several instances of the same component type each distinguished by a unique name. The syntax for component declaration is as following:-

  component Component Name
  generic (list);
  port (list);
  end Component Name;

- **Configuration:** A configuration describes linkage between component types and entity-architecture pairs. It selects which component should be used in architecture and exact set of entities and architectures used in a particular simulation or synthesis. However, has limited or no support for synthesis.
- **Package:** A package declaration includes all globally used declarations of data types, components, procedures and functions. Once these objects are declared and defined in a package, they can be used by different VHDL design units.

Other Important Features of VHDL

- VHDL is **case** and **space insensitive** language.
- It supports whole design process i.e. system level, register transfer level (RTL), logic level and upto some extent circuit level.
- VHDL statements are terminated with **semicolon (;).**
- A VHDL statement converts into comment by applying two consecutive **dashes (--)** before it.
- VHDL is used for both synthesis and simulation of digital circuits.
- VHDL identifiers are **A-Z, a-z, 0-9** and **underscore (_)**. Must start with alphabet and last not with an underscore. No two successive underscores.
- Reserved words cannot be used as identifiers.
- VHDL objects include:-
  - **Constant**: a constant declaration must assign a fixed, never be modified during synthesis or the operation of the circuit, value to it before the **begin** of an **architecture** and/or **process**.
    Declaration Syntax: constant identifier : **type** := value;
  - **Variable**: used for local storage within process, must be declared before the **begin** of an **architecture** and/or **process,** may or may not assign value to it, can change during simulation/execution, usually used as indexes, mainly in loops, or to take values that allow to model other components but do not represent physical connections or memory element.
    Declaration Syntax: variable identifier : **type** [:= value];
  - **Signal**: represents interconnections that connects components and ports, unlike to constant and variable, signals can be synthesized i.e. can be physically translated into a memory element in the final circuit, must be declared before the **begin** of an **architecture**.
    Declaration Syntax: signal identifier : **type**;

- The object type could be a scalar or an array.
- VHDL operators are used to build variety of expressions that allow to calculate data and/or to assign them signals or variable. Various VHDL operators are:-
  - Arithmetic: +, -, *, /, mod, rem
  - Concatenation: &
  - Logical: and, or, nand, nor, not, xor
  - Assignment operator for constants and variables: :=
  - Assignment operator for signals: <=
- The most common predefined types in VHDL are the followings:
  - **bit**: takes only the values 0 and 1 written between single quotes ('0' or '1').
  - **bit_vector (range)**: takes an array of 0's and 1's. **range** always written between brackets and indicates the number of bits. For n-bit bit_vector, its range is written as **N-1 downto 0**.
  - **boolean**: takes only the values **true** or **false**.
  - **character**: takes any ASCII value.
  - **string**: takes any chain of ASCII characters.
  - **std_logic**: Type defined in the IEEE 1164 standard and represents a multivalued logic comprising 9 different possible values. Most commonly used are '0', '1' 'Z' (for high impedance), 'X' (for uninitialized) and 'U' (for undefined).
  - **std_logic_vector (range)**: represents a vector of elements of type std_logic.
  - **natural/integer/real/positive range**: any number within range. **range** is optional.
- VHDL features the **wait** statement which stops the simulation of the code until condition is met. There are three types of **wait** statements:
  - **wait on list_of_signals**: simulation stops until any signal in the **list_of_signals** is modified.
  - **wait for time:** simulation stops for the time specified in the **time** variable.
  - **Wait until condition:** simulation stops until the **condition** is met.
- For sequential logic designs **'event** is used on the clock signal as follows:

  ```
  if ( clk 'event and clk='1') then ...
  ```

  The **'event** attribute on a signal returns true if that signal has just been modified, returns false otherwise. In above **if-then** statement checks if there has been any modification on the **clk** signal, and if its new value is '1'. Thus, it recognizes a rising edge in clock signal.
- **Concurrent Statements:** kind of assignment statements to signals whose operation depends on a set of conditions. Two kinds of concurrent statement exists:
  - **when-else:** modifies the value of a given signal depending on a set of conditions, being the assigned values and conditions independent among each other. For example:

    ```
    C <=  "00" when A=B else
          "01" when A<B else
          "10";
    ```
  - **with-select-when:** modifies the value of a signal, depending on the values the signal_condition may have. For example:

    ```
    with input select
    output <=  "00" when "0001",
               "01" when "0010",
               "10" when "0100",
               "11" when others ;
    ```

- **Conditional Statements:** assignment statements to variable that may or may not be based on a condition must be placed inside a **process.** Following conditional statements exist in VHDL:
  - **If-then-else:** Example-

```
process (control, A, B,)
begin
    if control = "00" then
            output <= A+B;
    elsif control = "11" then
            output <= A-B;
    else
            output <= A;
    end if;
end process;
```

Note: **elsif** statement differs from **else if.** Each **else if** statement ends with **end** statement while **elsif** statement requires **end** only once. The **elsif** is used when different nested conditions have to be checked.

  - **case-when:** Example-

```
process (control, A, B,)
begin
    case control is
            when "00" => output <= A+B;
            when "11" => output <= A-B;
            when others => output <= A;
    end case;
end process;
```

  - **for-loop:** Example-

```
process (A)
begin
    for i in 0 to 7 loop
        B (i+1) <= A(i);
    end loop;
end process;
```

  - **while-loop:** Example-

```
process (A)
variable i : natural := 0;
begin
    while i < 7 loop
        B (i+1) <= A(i);
        i := i+1;
    end loop;
end process;
```

Example#1: VHDL coding for full adder: A combinational type circuit.

Boolean Expressions

$s = x \oplus y \oplus z$

$c = xy + yz + yx$

Truth Table

x	y	z	sum (s)	carry (c)
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

Logic Diagrams

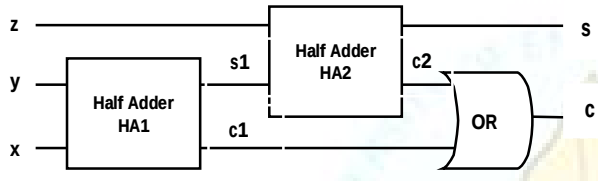

Fig.1: Full adder using half adders

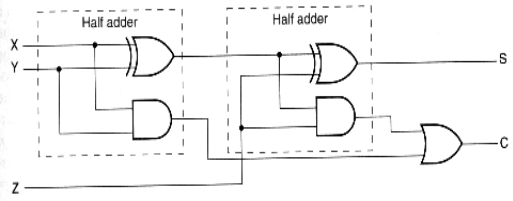

Fig.2: Full adder using gates

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

Library declaration
(Appears by default in source file)
Same for all styles of modeling.
Can be included other libraries also
as per requirement.

 # VHDL codes in Data flow modeling

```
entity FA_DataFlow  is
port (x, y, z : in std_logic;
        s, c : out std_logic);
end FA_DataFlow ;
```

Entity declaration.
Remains same for all styles of modeling.
However, Entity name can be changed as
FA_Behavioral and FA_Structural.

```
architecture DataFlow of FA_DataFlow is
begin
s <= x xor y xor z;
c <= (x and y) or (y and z) or (z and x);
end DataFlow;
```

Architecture declaration in terms of Boolean
expressions for data flow style of modeling.
Architecture name can also be changed.

 # VHDL codes in Behavioral modeling

```
architecture Behavioral of FA_Behavioral is
begin
process (x, y, z)
```

Sensitivity list

Using **if-then** statement	Using **case-when** statement

begin
 	
if (x='0' and y='0' and z='0') **then**
s <= '0';
c <= '0';
elsif (x='0' and y='0' and z='1') **then**
s <= '1';
c <= '0';
elsif (x='0' and y='1' and z='0') **then**
s <= '1';
c <= '0';
elsif (x='0' and y='1' and z='1') **then**
s <= '0';
c <= '1';
elsif (x='1' and y='0' and z='0') **then**
s <= '1';
c <= '0';
elsif (x='1' and y='0' and z='1') **then**
s <= '0';
c <= '1';
elsif (x='1' and y='1' and z='0') **then**
s <= '0';
c <= '1';
else
s <= '1';
c <= '1';
end if;
end process;
end Behavioral;
	variable i : std_logic_vector(2 **downto** 0);
	begin
	i := x & y & z;
	case i **is**
	when "000" =>
	s <= '0';
	c <= '0';
	when "001" =>
	s <= '1';
	c <= '0';
	when "010" =>
	s <= '1';
	c <= '0';
	when "011" =>
	s <= '0';
	c <= '1';
	when "100" =>
	s <= '1';
	c <= '0';
	when "101" =>
	s <= '0';
	c <= '1';
	when "110" =>
	s <= '0';
	c <= '1';
	when others =>
	s <= '1';
	c <= '1';
	end case;
	end process;
	end Behavioral;

Architecture declaration in terms of truth table for behavioral style of modeling

 # VHDL codes in Structural modeling

architecture Structural **of** FA_Structural **is**

signal s1, c1, c2 : std_logic; ← Signal (interconnections) declaration

component HalfAdder
port (x, y : **in** std_logic;
s, c : **out** std_logic);
end component ;

Component declaration

begin

HA1: HalfAdder **port map** (x, y, s1, c1);
HA2: HalfAdder **port map** (z, s1, s, c2);
c <= c1 **or** c2;

Configuration declaration

end Structural ;

Architecture declaration in terms of logic diagram for structural style of modeling. Here component **HalfAdder** is being called in the main program which has been already created by the user.

Example#2: VHDL coding for counter: A sequential type circuit.

This program describes a generic counter that, once it reaches a maximum value, is re-initialized and starts over again. Programming has been done in behavioral modeling.

```
entity counter is
  generic (maximum : natural := max ; N: natural := 8 ) ;
 port(reset , clk  : in std_logic; n : out std_logic_vector (N-1 downto 0));
end counter;
architecture arch of counter is
signal state : std_logic_vector(N-1 downto 0) ;
begin
process(reset, clk, state)
begin
   if(reset = '1' )
     state <= "000";
   elsif clk 'event and clk =  '1' then
   if state < max
        state <= state + 1;
   else
        state <= (others => '0') ;
     end if;
     end if;
end process;
 n <= state;
end arch ;
```

In this lab, circuit synthesis and design will be performed using **Xilinx ISE 12.1i** software while the **ISim** logic simulator is used for system-level testing. **Xilinx ISE** (**I**ntegrated **S**ynthesis **E**nvironment) is a powerful software tool produced by Xilinx (*Xilinx Inc. is an American technology company, primarily a supplier of programmable logic devices. It is the first semiconductor company and invented the FPGA* that is used to design, synthesis, simulate, test and verify digital circuit designs using VHDL. Each experiment will be performed in two parts i.e. **designing (creation)** of digital circuits and then **operational verification** of designed circuits by writing **design** and **test bench codes** respectively**.**
Following steps to be followed in each lab:-

- VHDL coding for the design
- Syntax checking of the design
- Generating Register Transfer Logic (RTL) schematic of the design
- Test bench coding with possible inputs
- Simulation of test bench code to verify the design
- Synthesis of the design to see logic resources and timing detail

Following section presents tutorial on the **Xilinx ISE Simulator** which explains all the steps to design and verify the digital circuits.