

## ECC3301 Digital System Design

### Tutorial 3 – Hardware Description Language

#### 1 Why HDL?

Hardware Description Language (HDL) is a Computer-Aided Design (CAD) tool for the modern design and synthesis of digital systems. The recent, steady advances in semiconductor technology continue to increase the power and complexity of digital systems. Due to their complexity, such system cannot be realize using high-density, programmable chips, such as Application-Specific Integrated Circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs), and require sophisticated CAD tools. HDL is an integral part of such tools. HDL offers the designer a very efficient tool for implementing and synthesizing design on chip.

The designer uses HDL to describe the system in a computer language that is similar to several commonly used software language such as C. Debugging the design is easy, since HDL packages implement simulator and test benches.

HDL's allows the design to be simulated earlier in the design cycle in order to correct errors or experiments with different architecture. Designs described in HDL are technology-independent, easy to design as well as debug and are usually more readable than schematics, particularly for large circuits.

There are two types of code in most HDLs:

*Structural* - this is a verbal wiring diagram without storage.

```
assign a=b & c | d;          /* "|" is a OR */
assign d = e & (~c);
```

Here the order of the statements does not matter. Changing e will change a.

*Procedural* - is used for circuit with storage, or as a convenient way to write conditional logic.

```
always @(posedge clk)      // Execute the next statement on every
rising clock edge.
count <= count+1;
```

## **2 What is “Verilog”?**

Verilog is a hardware description language (HDL), similar to VHDL that was originally written by Phil Moorby in 1984 who needed a simple, intuitive and effective way of describing digital circuit for modeling, simulation and analysis purpose. Sometimes the language is also called Verilog HDL and it supports the design verification as well as implementation of analog, digital and mixed-signal circuits at various levels of abstraction. If you are familiar with C programming language, Verilog is a fairly language to learn. Verilog is a case-sensitive language.

Verilog can be used to describe design at four levels of abstraction:

- i. Switch level (the switches are MOS transistor inside gates)
- ii. Gate level (interconnected AND, NOR etc)
- iii. Register transfer level (RTL uses registers connected by Boolean equations)
- iv. Algorithmic level (much like code c with case and loop statements)

## **3 Verilog vs VHDL**

The language syntax is the main factor – since VHDL is based on ADA and Verilog is based on C.

- Verilog is easier to learn since C is a far simpler language. It also produces more compact code; easier both to write and read.
- VHDL is very strongly typed, and allow programmer to define their own types although, in practice, the main types used are either the basic types of the language itself or those define by the IEEE. The benefit is that type checking is performed by the compiler which can reduce errors; the disadvantage is that changing types must be done explicitly.

VHDL has two clear advantages over Verilog:

- It provides a simple mechanism (the *configure* statement) that allows the designer to switch painless between different description of a particular module.
- It allows the conditional instancing of modules (if/for ... generate). This is one of those features that you do not miss until you have used it once – and then you need it all the time. Many Verilog users recognize this lack and create personal pre-processing routine it implement it (which negates some of the advantages of a language standard).

Verilog has two clear advantages over VHDL:

- It ensures that all signals are initialized to “unknown” which ensures that all designers will produce the necessary logic to initialize their design – the base type in VHDL initialize to zero and the “hasty” designer may omit the global reset.
- It allows switch-level modeling – which some designers find useful for exploring new circuits.

#### **4 Lexical tokens**

Verilog source text files consist of the following lexical tokens:

- White space
- Comments
- Numbers
- Identifier
- Operators
- Verilog keywords

#### **5 Structure of the Verilog module**

The Verilog module has a declaration and a body. In the declaration, name, inputs and outputs of the module are listed. The body shows the relationship between the inputs and the outputs. Example of Verilog module:

```
module half_adder (IP1, IP2, OP1, OP2);
input IP1, IP2;
output OP1, OP2;
// Blank line

assign OP1 = IP1 ^ IP2;           //statement 1
assign OP2 = IP1 & IP2;         //statement 2
endmodule
```

#### **6 Verilog port**

Verilog port can be one of the following three modes:

**input:** In any assignment statement, the port should appear only on the right-hand side of the statement.

**output:** The output port can appear on either side of the assignment statement.

**inout:** the port can be used as both an input and output. These represent port through which one gets entry into the module or exits the module; these are terminals through which signal are input to the module sometimes; at some other times signal are output from the module through these.

## **7 Operators**

Operators perform a wide variety of functions. These functions can be classified as:

- Logical
- Relational
- Arithmetic
- Shift

## **8 Data types**

Verilog supports several data types, including: `nets`, `register`, `vectors`, `integer`, `real`, `parameters` and `arrays`.

## **9 Assignment**

1. Discuss all the data types listed in section 5.8 and for each of them, provide an example on how they are declared in Verilog.