UM TAGUM COLLEGE

DEPARTMENT OF ENGINEERING EDUCATION

COMPUTER ENGINEERING PROGRAM

MODULE INSTANTIATION AND

TEST BENCHES

DRILL 1

NAME:

STUDENT NUMBER:

TERMINAL NUMBER:

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PROFESSOR

I. DISCUSSION

A hardware description language (HDL) is a computer-based language that describes the hardware of digital systems in a textual form. It resembles an ordinary computer programming language, such as C, but is specifically oriented to describing hardware structures and behaviour of logic circuits.

HDLs are used in several major steps in the design flow of an integrated circuit:

* design entry,
* functional simulation or verification,
* logic synthesis,
* timing verification,
* fault simulation.

Companies that design integrated circuits use proprietary and public HDLs. In the public domain, there are two standard HDLs that are supported by the IEEE: VHDL and Verilog.

VHDL

* stands for VHSIC (very high speed integrated circuit) HDL
* a Department of Defense mandated language

Verilog

* began as a proprietary of companies and universities known as Open Verilog International (OVI) as a step leading to its adoption as an IEEE standard.
* It was initially approved as a standard HDL in 1995; revised and enhanced versions of the language were approved in 2001 and 2005.

Throughout this course, the Verilog HDL descriptions will be listed to introduce a design methodology based on the concept of computer-aided modelling of digital systems.

**Module Declaration**

In particular, a Verilog model is composed of text using keywords, of which there are about 100. Keywords are predefined lowercase identifiers that define the language constructs. Any text following the two forward slashes is interpreted as a comment and will have no effect on a simulation using the model. Multiline comments begin with /\* and terminate with \*/. Blank spaces are ignored, but they may not appear within the text of a keyword, an identifier, an operator, or the representation of a number. Verilog is case-sensitive, which means that the uppercase and lowercase letters are distinguishable.

A module is the fundamental descriptive unit in the Verilog language. It is declared by the keyword **module** and must always be terminated by the keyword **endmodule**.

*module module\_name (port list);*

*//Verilog statements*

*endmodule*

The port list of a module is the interface between the module and its environment. This list is enclosed in parentheses, and commas are used to separate elements of the list. The statement is terminated with a semicolon (;). The keywords **input** and **output** specify which of the ports are inputs and which are outputs. Internal connections are declared as wires. This connection is declared with the keyword **wire**.

*input in1;*

*output out1, out2;*

*wire x, y, z;*

**Test Benches**

In order to simulate a circuit with an HDL, it is necessary to apply inputs to the circuit so that the simulator will generate an output response. An HDL description that provides the stimulus to a design is called a *test bench*. In its simplest form, a test bench is a module containing a single generator and an instantiation of the model that is to be verified. Note that it has no input or output ports, because it does not interact with its environment. Within the test bench, the inputs to the circuit are declared with keyword **reg** and the outputs are declared with the keyword **wire**. Note that using a test bench is similar to testing actual hardware by attaching signal generators to the inputs of a circuit and attaching probes (wires) to the outputs of the circuit).

r*eg a, b, c;*

*wire w1, w2;*

The **initial** keyword is used with a set of statements that begin executing when the simulation is initialized, and terminates execution when the last statement has finished executing. The set of statements to be executed is called a block statement and consists of several statements enclosed by keywords **begin** and **end.** The action specified by the statements begins when the simulation is launched, and the statements are executed in sequence.

The response to the stimulus generated by the **initial** and **always** blocks will appear in text format as standard output and as waveforms (timing diagrams) in simulators having graphical output capability. Numerical outputs are displayed by using Verilog *system tasks*. These are built-in system functions that are recognized by keywords that begin with the symbol **$**. Some of the system tasks that are useful for display are:

**$display** – display a one-time value of variables or strings with an end-of-line return,

**$write** – same as **$display**, but without going to next line,

**$monitor** – display variables whenever a value changes during a simulation run,

**$time** – display the simulation time,

**$finish** – terminate the simulation.

The syntax is of the form:

*Task\_name(format specification, argument list);*

The format specification uses the symbol % to specify radix of numbers that are to be displayed and may have a string enclosed in quotes. The base may be binary (%b), decimal (%d), and hexadecimal (%h).

II. Drill Exercises

A. Module Declaration

The HDL description of the circuit of the Fig 1.1 is shown in the example below. Open your Notepad and place the code inside. Save the file as Drill1\_1.v in the directory where your Icarus Verilog compiler is currently residing (common location is at C:\iverilog\bin).

Fig 1.1

*//Verilog model of circuit of Fig 1.1*

*module circuit1\_1(A, B, C, X);*

*input A, B, C;*

*output X;*

*wire wire1, wire2, wire3;*

*not NOT(wire2, A);*

*xor EOR2(wire1, B, C);*

*and AND2(wire3, wire1, A);*

*nor NOR2(X, wire3, wire2);*

*endmodule*

B. Test Bench

Edit the saved file Drill1\_1 by placing the following code below the previous code.

*//Test bench for the Verilog model of Fig 1.1*

*module testbench1\_1;*

*reg A, B, C;*

*wire Z;*

*circuit1\_1 tb1(A, B, C, Z);*

*initial*

*begin*

*A=1’b0; B=1’b0; C=1’b0;*

*$display(“Simulating output for circuit1\_1”);*

*$monitor($time,,,”A=%b B=%b C=%b Z=%b”,A,B,C,Z);*

*#2 A=1’b0; B=1’b0; C=1’b1;*

*#1 A=1’b0; B=1’b1; C=1’b0;*

*#1 A=1’b0; B=1’b1; C=1’b1;*

*#1 A=1’b1; B=1’b0; C=1’b0;*

*#1 A=1’b1; B=1’b0; C=1’b1;*

*#1 A=1’b1; B=1’b1; C=1’b0;*

*#1 A=1’b1; B=1’b1; C=1’b1;*

*#2 $finish;*

*end*

*endmodule*

C. Code the next example and save it under the filename Drill1\_2.vl.

*//behavioural Verilog code for a simple 1-bit full subtracter.*

*module full\_subtract(diff, borrowOut, a, b, borrowIn);*

*output diff;*

*output borrowOut;*

*input a, b, borrowIn;*

*assign {borrowOut, diff} = a – b – borrowIn;*

*//result of subtraction is two bits; the MSB is borrowOut and the LSB*

*//is diff.*

*endmodule*

*module testingFS();*

*reg a, b,borrowIn;*

*wire diff, borrowOut;*

*full\_subtract fs(diff, borrowOut, a, b, borrowIn);*

*initial begin*

*a=1’b1; b=1’b1; borrowIn=1’b0;*

*end*

*initial begin*

*#10 a=1’b1;*

*#10 a=1’b0; b=1’b1;*

*#10 a=1’b1; b=1’b0;*

*#10 borrowIn=1’b1;*

*end*

*initial begin*

*$display(“ a b borrowIn difference borrowOut time”);*

*$monitor(“ %b %b %b %b %b %b ”, a, b, borrowIn, diff, borrowOut, $time);*

*#10 $finish;*

*end*

*endmodule*

III. Programming Exercise

1. Copy the code below, and then create a test bench for it to determine what combinational circuit is being simulated. Save as exercise1\_1.vl file.

*module exercise1\_1(W, X, Y, Z);*

*output [0:3] W;*

*input X, Y;*

*input Z;*

*wire X1, Y1, Z1;*

*not G1(X1, X), G2(Y1, Y),G3(Z1, Z);*

*nand G4(W[0], X1, Y1, Z1),G5(W[1], X1, Y, Z1),*

*G6(W[2], X, Y1, Z1),G7(W[3], X, Y, Z1);*

*endmodule*

1. Edit the test bench from Drill1\_1 in such a way that the same output will be obtained without using the **$monitor** task. Terminate your simulation after 80 ns. Save the new file as exercise1\_2.vl.
2. Describe the circuit that is being simulated by the program below. Verify your answer by displaying the truth table using an appropriate test bench. Save the new file as exercise1\_3.vl

*module exercise1\_3(*

*output var1, x\_4,*

*input x\_1, x\_2, x\_3);*

*xor EOR1(x\_4, x\_1, x\_2, x\_3);*

*xor EOR2(var1, x\_1, x\_2, x\_3, x\_4);*

*endmodule*

IV. Review Questions

1. Based from Drill1\_1, notice that *wire1*, *wire2*, and *wire3* are outputs of gates *EOR2*, *NOT* and *AND2* respectively. Why are they declared as **wire** instead of **output**?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. How do you instantiate a module within a test bench?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Can we instantiate a test bench from another test bench? Why or why not?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_