UM TAGUM COLLEGE

DEPARTMENT OF ENGINEERING EDUCATION

COMPUTER ENGINEERING PROGRAM

User-Defined Primitives

DRILL 4

NAME: Jobenilita R. Cuñado

STUDENT NUMBER: 2015390026

TERMINAL NUMBER:

DATE OF PERFORMANCE: June 19, 2016

DATE OF SUBMISSION: June 19, 2016

\_\_\_Engr. Jocelyn Villaverde\_\_\_

PROFESSOR

I. DISCUSSION

The logic gates used in HDL descriptions with keywords **and**, **or**, etc. are defined by the system and are referred to as system primitives. The user can create additional primitives by defining them in a tabular form. These types of circuits are referred to as *user-defined primitives* (UDPs). One way of specifying a digital circuit in tabular from is by means of a truth table.

General rules for defining a UDP:

* Starts with the keyword **primitive**, and ends with the keyword **endprimitive**
* The keyword **primitive** should be followed by an identifier, a list of ports, and followed by a semicolon (;)
* Order in which they are listed in the input declaration must conform to the order in which they are given values in the table that follows.
* The table inside your primitive should be enclosed within the keywords **table** and **endtable**.
* The values of inputs are listed in order, ending with a colon (:). The output is always the last entry in a row and is followed by a semicolon (;)
* Port lists should contain any number of inputs but only a single output.

There are two types of behaviour that can be defined in UDP.

1. Combinational UDP

* This table specifies the various input combinations and their corresponding output values. Any combination that is not specified is an x for the output.

1. Sequential UDP
   * The internal state is described using a 1-bit register.
   * It uses the current value of the register and the input values to determine the next value of the register.
   * Its model requires that its output be declared as **reg** data type, and that column be added to the truth table to describe the next state. So the columns are organized as:

*inputs: state: next state*

The state of a sequential UDP can be initialized by using an initial statement that has one procedural assignment statement. This is of the form:

**initial**  reg\_name = 0, 1 or x ;

The following is the summary of table entries.

|  |  |
| --- | --- |
| Symbol | Meaning |
| 0 | logic 0 |
| 1 | logic 1 |
| x | unknown |
| ? | any of 0, 1, or x |
| b | any of 0 or 1 |
| - | no change |
| (AB) | value change from A to B |
| \* | same as (??) |
| r | same as (01) |
| f | same as (10) |
| p | any of (01), (0x), (x1) |
| n | any of (10), (1x), (x0) |

II. Drill Exercises

A. Create a UDP description of a 3-bit majority circuit.

We can consider that the output of this circuit is 1 if the input vector has two or more 1’s. The program below shows a synthesis of a majority circuit with the corresponding test bench. Note that the declaration *input A, B, C* matters because this is how Verilog will determine which input belongs to which column. Copy the code then save as drill4\_1.vl.

*primitive Majority3 ( Z, A, B, C ) ;*

*input A, B, C ;*

*output Z ;*

*table*

*// A B C Z*

*0 0 ? : 0 ;*

*0 ? 0 : 0 ;*

*? 0 0 : 0 ;*

*1 1 ? : 1 ;*

*1 ? 1 : 1 ;*

*? 1 1 : 1 ;*

*endtable*

*endprimitive*

*module testprimitive1;*

*reg A, B, C;*

*wire X;*

*Majority3 (X,A,B,C);*

*initial begin*

*A=1'b0;B=1'b0;C=1'b0;*

*$display(" A B C X");*

*$monitor(" %b %b %b %b",A,B,C,X);*

*#2 A=1'b0; B=1'b0; C=1'b0;*

*#2 A=1'b0; B=1'b0; C=1'b1;*

*#2 A=1'b0; B=1'b1; C=1'b0;*

*#2 A=1'b0; B=1'b1; C=1'b1;*

*#2 A=1'b1; B=1'b0; C=1'b0;*

*#2 A=1'b1; B=1'b0; C=1'b1;*

*#2 A=1'b1; B=1'b1; C=1'b0;*

*#2 A=1'b1; B=1'b1; C=1'b1;*

*#2 $finish;*

*end*

*endmodule*

B. Create a UDP description of a positive-edge triggered T flip-flop.

The table below shows an example of sequential UDP. Simulate the program; save it as drill4\_2.vl

*primitive T\_FF(F, clk, T);*

*input clk, T;*

*output F;*

*reg F;*

*initial F=0;*

*table*

*// CLK T : F(STATE) : F(NEXT)*

*(01) ? : x : 0 ;*

*(01) 0 : 0 : 0 ;*

*(01) 0 : 1 : 1 ;*

*(01) 1 : 0 : 1 ;*

*(01) 1 : 1 : 0 ;*

*(10) ? : x : 0 ;*

*(10) 0 : 0 : 0 ;*

*(10) 0 : 1 : 1 ;*

*(10) 1 : 0 : 0 ;*

*(10) 1 : 1 : 1 ;*

*endtable*

*endprimitive*

*module pri;*

*reg x, clk;*

*wire y;*

*T\_FF TF(y, clk, x);*

*initial begin*

*x=0; clk=0;*

*end*

*always #1 clk=!clk;*

*initial begin*

*x=0;*

*#4 x=1;*

*repeat(8)*

*#8 x=~x;*

*end*

*initial begin*

*$display(" TIME clk T Q");*

*$monitor($time,,," %b %b %b ",clk, x, y);*

*#16 $finish;*

*end*

*endmodule*

III. Programming Exercise

1. Design a combinational UDP of 8x1 multiplexer. Save as exercise4\_1.vl.
2. Design a combinational UDP of a binary subtractor.

IV. Review Questions

1. Can you implement UDP for circuits with multiple outputs? If yes, how?

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

1. Why do you need to declare the output of sequential UDPs as **reg**?

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

1. What type of table is being placed inside the program to create a sequential UDP? Can you display the other tables using the created UDP? If yes, how?

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

1. Explain the entry below:

*//Clk Clr Data : Q(state) : Q(next)*

*(?0) 1 b : 0 : - ;*

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