Writing Code for Synthesis. Electronic Dice Game

PURPOSE - In this lab you will learn about how to write synthesizable Verliog code for finite state machines, and how to drive the 7-segment LEDs of the BASYS3 board. As an application you will design an electronic dice game.

---

1. Designing an electronic dice game

The idea of the dice game is that two counters are used to simulate the roll of the dice. Each counter counts in the sequence 1, 2, 3, 4, 5, 6, 1, 2, …. After the "roll" of the dice, the sum of the values in the two counters will be in the range 2 through 12. The rules are as follows:

1> After the first roll of the dice, the player wins if the sum is 7 or 11. The player loses if the sum is 2, 3 or 12. Otherwise, the player obtained on the first roll is referred to as **a point**, and he or she must roll the dice again.

2> On the second or subsequent roll of the dice, the player wins if the sum **equals the point**, and he or she loses if the sum is 7. Otherwise, the player must roll again until he or she finally wins or loses.

The inputs to the dice game come from two push buttons, Rb (Roll button) and Reset. Reset is used to initiate a new game. When the roll button is pushed, the dice counters count at a high speed, so the values cannot be read on the display. When the roll button is released, the values in the two counters are displayed, and the game can proceed.

The control logic for this game is best implemented as **a finite state machine.** The Verliog files (from the textbook) include a top-level module that instantiates the counter, clock_divider (same as used in the previous lab but you will want to set a different time constant), and a partial state machine module which will implement the dice game control. The top-level module also implements logic to control the 7-segment LEDs, which you will need to correct (it has don't cares to begin with). Complete the verilog modules, and implement the design. Then, you will download the bitstream file to the lab boards and verify the game.

The class textbook contains a diagram of the state machine that was implemented in the as the control module. You can implement this state machine as is, or you may be able to simplify it if you understand the game requirements sufficiently. To test and verify your design, you should set a fairly large time constant (50 or 60) for the clock divider so that the dice roll very slowly, so that you can see what roll is active and stop at the interesting numbers. To play the game, you should set a smaller time constant (say 15).

You will have to draw your SM chart of the dice game and its implementation design with F/Fs and basic gates including all the internal interconnection between components. **You will have to attach this diagram to the lab note-**book you will turn in at the end of the semester!

The output of the dice game will have to drive at least one of the 7-segment LEDs of the BASYS3 board. The LED will indicate "U" in case that the player wins, "L" in case that the player loses, or "A" in case that the player has to play/roll again. When the game is reset the 7-segment LED displays "0". Figure out (using the manual for the BASYS-3 board manual, available from the class website) what pin assignments you have to use to properly activate the 7 segments of each 7-segment LED. Also, assign the proper pins such that to use **one push button** as **the Reset** of the game entity, **another push button** as **Rb**. For the **CLK_IN** input in your game entity assign pin number **W5** in the Constraints File (.xdc) of your project, which will have to be used for implementation.

To verify and demonstrate proper operation, **you must display the two final numbers of the two dice after a roll**. To do that you will have to modify the above Verliog files accordingly in order to drive two more 7-segment LEDs properly. Alternatively, you could display the binary value of the two dice using 6 of the green LEDs, but you will need to drive all four 7-segment LEDs for future labs so if you do so now you can reuse that code later. The best way to do this is to use another instance of clock_divider, and set its time constant parameter to a fast value such as 10 or 15. Use this fast clock to run a 2-bit counter which cycles 0-3, then write a combinatorial 2-to-4 decoder that takes this 2-bit value and creates the 4 LED select output signals, and chooses which value to put on the 7-segment outputs (which are common to all 4 LEDs) based on which LED is current. You should have an internal 7-bit value for each of the 4 LEDs, and choose which of these to assign to the output pins based on the decoder value. By cycling the LED select fast enough, it appears to the human eye that all four LEDs are always on. So you end up with one clock_divider running the game logic, and another running the LED selection logic. You can run the game logic slower, but keep the LED selection fast this way.

If you are having trouble getting your game to function properly, you may want to bring the state machine current-state value out to the green LEDs as a binary value, so you can watch your game progress as you push the Rb button. If you show the sum of the two dice instead of the individual values on green LEDs, you should have room for both the roll total and the current state value. If you have the dice values on 7-segment LEDs then you have all 16 green LEDs to show internal values, another advantage of doing the 7-segment logic now.

SUMMARY -- During this lab you learned general rules about how to write synthesizable code in Verliog and you implemented and verified an electronic dice game with Foundation Series.
