Casino-type Game

PURPOSE - In this lab you will design a casino-type game using the random number generator from the previous lab. Also, you will learn about how to write synthesizable Verilog code for finite state machines.

1. Writing the Verilog description

You will have to design a game using the random number generator from the previous lab. This game will function as follows: **On pressing the Start button (eg. reset button from Lab 4)** the random number generator will start generating numbers. When the Roll button is pressed you will use these two hex digits to play the game.

In order to reuse the most code possible you will have the two random hex numbers displayed in real time on the seven segment display (on either the left two or the right two digits, just like in lab 4), and then display the win "UI" or lose "LO" notification on the remaining two digits. **Read the following example to understand the proper operation of the game:**

A sample play of the game is as follows:

1) The player starts to generate the random numbers **The player presses the Start/Reset button so the random numbers are generated** on two of the digits just as in lab 4.

2) The player then **taps** the Roll button. Sum up the current two hex numbers, if the sum is greater than 25, then "UI" is displayed and the player wins. If the sum is less than 5, then "LO" is displayed (on the other remaining two digits) and the player loses. In both cases the game is over. If neither of the above cases are hit store the sum obtained and proceed to the next step. (here hex digits are represented by their 4 bit number i.e. A=10, F=15)

3) If the sum was between 5 and 25 (5 and 25 included) store the sum as “target”, the player has to roll again and ‘Ao’ is displayed. On pressing the roll button take the sum, if the new sum is less than target roll again. If the new sum is greater that or equal to the target but lesser than or equal to 25 the player wins, and if it is greater than 25 the player loses.

4) The "UI" or "LO" is still displayed until the next tap of the start button.

For simplicity "LO" can be displayed before the first tap of the roll button.

Of course, the player can always see the value of the hex digits, and thus tap the Roll button at the right time in order to win or lose the game, but this is done for debugging purposes.

2. Verification

Synthesize and implement the game to verify the correctness. You will have to write an **.xdc** file to correctly map the pins (you only need to add two more lines for the Roll button.) Download the bit-stream file to the board. Show the correct functioning to your TA for full credit.

Notes

The control logic for this game is best implemented as **a finite state machine.** The Verliog files (from the textbook) show a similar design where they include a top-level module that instantiates the counter(our case uses a LFSR), 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 a dice game control (ours implements a similar game). 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 sample state machine for a similar game that was implemented in the as the control module. You can implement this state machine in a similar manner, 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 which you will turn in!

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 "UI" in case that the player wins, "LO" in case that the player loses, or "Ao" 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 7segment 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