Lab 6: Whack-A-Mole

Objective

1.   Getting familiar with finite state machines with Verilog.

2.   Getting familiar with the keyboard control on the FPGA board.

Description

Whack-A-Mole is a casual and fun game in which players use a mallet to hit toy moles, which appear at random, back into their holes. In this lab, we want you to design the Whack-A-Mole game for us and implement it on the FPGA board with a keyboard. The game starts after pressing the start button, hitting the moles by keyboard, and showing the score by the seven-segment. It stops after one minute, and you can press the start button to start over again.

I.Initial state

•   Enter this state after the reset.

a.   **7-segment**

The 7-segment shows — — — —.

b.   **LED**

All the LED lights are **off**.

c.   **start** (connected to **BTNU**)

Press the start button to enter the Game state.

II.Game state

•   Enter this state after pressing **start** in Initial/Final state.

•   We used LEDs to represent the moles and holes. There are nine holes in total, represented by LD15 to LD7, respectively. The lit-up LED indicates that the

mole is out of the hole, so you can press the corresponding button to hit it and get one point.

• The game will end and enter the final state in the following two cases: (1) 30 seconds past; (2) Players get 10 points (You can implement a higher score as long as you are sure you can get that many points in the demo).

• **start** shouldn't function in this state.

a. **LED** (LD15 - LD7)

LEDs represent the moles. They are lit up as soon as entering this state and then updated every 1 second. The number and location of the moles will be determined by the pseudo-random pattern (refer to the hints later).
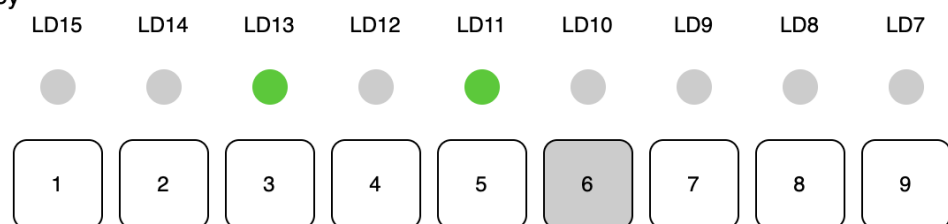
b. **Keyboard**

Numeric keys (1 to 9) represent whacks and correspond to the LD15 to LD7 holes. The mole can be hit by pressing the key, and only one key can be pressed at a time. When a key is pressed, any subsequent key presses are invalid.
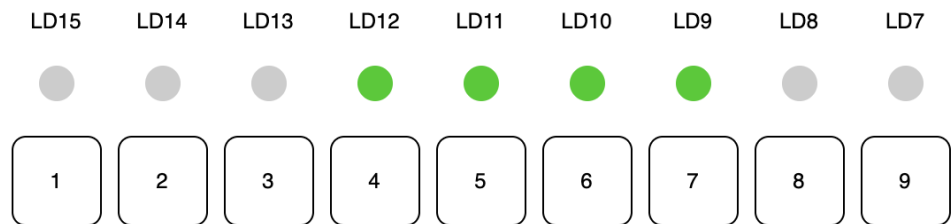
Show a pattern
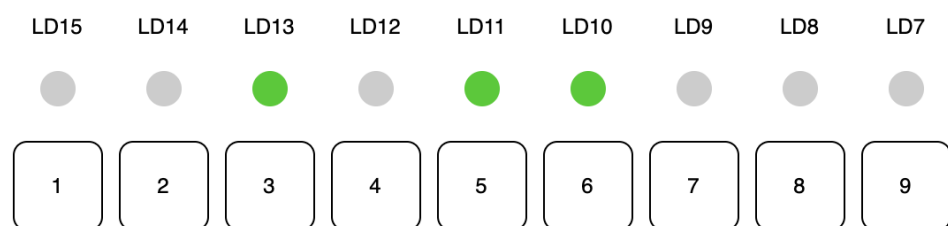
Turn off the LED light after
pressed corresponding key

Figure 1: lab6 Game state example

Display the next pattern 1
seconds after the first
pattern is displayed

LD15	LD14	LD13	LD12	LD11	LD10	LD9	LD8	LD7

1	2	3	4	5	6	7	8	9

Figure 1: lab6 Game state example

d. **7-segment**

The 7-segment display is split into two parts. The leftmost two digits count down from 30 seconds, and the right ones are used to record the score. Therefore, the 7-segment should look like this in the beginning: 3 0 0 0. And it turns into 2 9 0 0 after one second; 2 9 0 1 after hitting a mole.

III. Final state

a. **7-segment**

If the player enters this state by reaching the target score, the 7-segment display shows — W I N. Otherwise, you should set the two digits on the left to **0** and show the score on the two rightmost digits. WIN in the 7-segment display can be shown as

b. **LED**

All the LED lights are **on**.

c. **start**

Press the start button to enter the Game state.

I/O Signal Specification

✓ Input: **clk**, **rst**, **start**

✓ Output: **digit[3:0]**, **display[6:0]**

✓ Inout: **PS2_CLK, PS2_DATA**

rst: The asynchronous active-high reset (connected to **BTNC**).

start: To start the game (connected to **BTNU**).

LED[15:0]: LD15~LD7 represent the holes. All the lights turn on in the final state (connected to **LD15 ~ LD0**).

DIGIT[3:0]: the signals to enable one of the 7-segment digits.

DISPLAY[6:0]: the signals to control the digits on the 7-segment display.

Note

1. For the keyboard control, you may refer to KeyboardController and KeyboardDecoder discussed in class.

2. PS2_CLK and PS2_DATA are inout signals, that is, bidirectional signals. Don't change them to input or output.

Hint:

1. Use LFSR (Linear Feedback Shift Register) discussed in the class to generate pseudo-random numbers. You can either use the outputs of the LFSR as the patterns of moles directly or use the random numbers to look up predefined patterns of moles. The number of patterns should be more than 10. Here are examples of two different patterns, each with 9 bits.

Figure 2: lab6 pattern 0

Figure 2: lab6 pattern 0

2. Use the following template for your design (you should determine the proper data type of each IO):

```verilog
module lab6 (
    input wire clk,
    input wire rst,
    input wire start,
    inout wire PS2_DATA;
    inout wire PS2_CLK;
    output reg [15:0] LED,
    output reg [3:0] DIGIT,
    output reg [6:0] DISPLAY
);

    /* Note that output ports can be either reg or wire.
     * It depends on how you design your module. */
    // add you design here
endmodule
```

DEMO: https://reurl.cc/aGQYY4

Questions and Discussion

Please answer the following questions in your report.
- A. What are the pros and cons of using the LFSR as a pseudo random number generator?
- B. What can we do to improve the randomness of our LFSR pseudo number generator?

Attention

✓ DO NOT copy-and-paste code segments from the PDF materials. It may also paste invisible non-ASCII characters, leading to hard-to-debug syntax errors.

✓ You should hand in only one source file, lab6.v. If you have several modules in your design, integrate them in lab6.v. You don't need to include the debounce, one-pulse, and keyboard controller modules in the file you hand in. Please do not integrate them into lab6.v

✓ You should also hand in your report as **lab6_report_StudentID.pdf** (i.e., lab6_report_103456789.pdf).

✓ Please do not hand in any compressed files, which will be considered an incorrect format.

✓ You should be able to answer questions about this lab from TA during the demo.

✓ You need to prepare the bitstream files before the lab demo to make the demo process smooth.

✓ Feel free to ask questions about the specification on the EECLASS forum.