Intro to Digital Logic, Lab 4
Design with multiple FSMs

Lab Objectives

In the last lab we developed a simple system using only a single finite state machine (FSM). Now we want to build a more complex system containing multiple FSMs. Careful design and testing of each individual module, will be key to getting this working well. **Please note that this lab will take significantly more time than the previous labs you have done. As such, you should start early, be methodical, and thoroughly simulate and test each individual FSM before connecting them together.**

Design Problem – Tug of War

Sweat pouring from their brow, body straining, muscles pulsing back and forth, we have the epic conflict which is: Tug of War! It's time to update this rope-based team sport into an electronic analog of finger-pounding power!

We're going to build a 2-player game of tug of war using the keys and LEDs on the DE1 board. Player 1 will "pull" by pressing KEY[0], while player 2 will use KEY[3]. The LEDs from LEDR9 to LEDR1, skipping LEDR0, will represent the playfield.

When the game starts, only the centermost LED is lit (LEDR5). Each time the first player presses the KEY[0] button, the light moves one LED to the right. Each time the second player presses the KEY[3] button, the light moves one LED to the left. If the light ever goes off the end of the playfield, the player that moved it off the end wins, and the HEX0 7-segment display will show "1" if the first player won, or "2" if the second player won. SW9 should be used as the reset signal, resetting the game completely, ready to be played again. You should use the 50MHz clock (pin CLOCK_50) directly to control the whole design.

If you try to design this as one big state machine, you will never get anything working. Instead, think about breaking it down into smaller pieces. We will help you with some ideas, but we STRONGLY advise putting together a block diagram of the system early in the design process.

User Input

Since we are using a fast clock, each time the user presses a button, the button will be ON for many clock cycles. However, for tug of war, you don't want players to just hold the button down - you want the input to be TRUE for one clock cycle for each button press. Therefore, you want to design a simple FSM that detects the moment the button is pressed – its output is TRUE for only 1 cycle for every button press. This will handle all user input.

Playfield

There are 9 lights, which is too big to do as a single huge FSM. However, what about an FSM for each location? A given playfield light needs to know the following:

- Does it start as TRUE (the center LED) or FALSE (all of the other LEDs)?
- During play, it needs to know which button(s) are pressed, whether its light is currently lit, and whether its right and left neighbors are currently lit.

Given that information, plus the reset signal, you can figure out whether any particular light should be lit during the next clock cycle.

Victory

You can tell when someone wins by watching the ends of the playfield – when the leftmost LED is lit and only the left button is pressed, player 2 wins. Player 1 wins similarly, but for the rightmost LED. So, you should build a module that controls the HEX0 display based on these victory conditions.

Suggested Modules

Your playfield can be controlled by implementing FSMs in the following SystemVerilog modules. These are only suggestions; you are free to implement your design using any number of different FSMs.

```
module centerLight (Clock, Reset, L, R, NL, NR, lightOn);
   input logic Clock, Reset;

   // L is true when left key is pressed, R is true when the right key
   // is pressed, NL is true when the light on the left is on, and NR
   // is true when the light on the right is on.
   input logic L, R, NL, NR;

   // when lightOn is true, the center light should be on.
   output logic lightOn;

   // Your code goes here!

endmodule
```

```
module normalLight (Clock, Reset, L, R, NL, NR, lightOn);
   input logic Clock, Reset;
   // L is true when left key is pressed, R is true when the right key
   // is pressed, NL is true when the light on the left is on, and NR
   // is true when the light on the right is on.
   input logic L, R, NL, NR;

   // when lightOn is true, the normal light should be on.
   output logic lightOn;

   // Your code goes here!

endmodule
```

If you were to use the above modules to create your FSMs, you'd need to instantiate 1 centerLight and 8 normalLight modules. In addition you'd need two instantiations of your userInput FSM, and logic to determine when someone has won the competitions. None of the FSMs should require more than four states.

Metastability

This lab has user input going into a somewhat high-speed circuit. That means there's a pretty good chance you can get metastability – the input to a DFF changing at about the same time as the clock edge occurs. **If you do not deal with this problem, you may encounter random, unpredictable problems with your circuit.**

To deal with metastability, make sure you send the user input (KEY[3] and KEY[0]) to a pair of D-flipflops BEFORE you use it in your logic (i.e. the rest of your circuit won't use KEY[3] nor KEY[0] directly, but instead will use the Q outputs of DFFs that receive the keys as inputs).

Advice

Build each of your FSMs and test them independently in ModelSim before combining them together. **SIMULATE EACH MODULE IN MODELSIM BEFORE TRYING TO CONNECT THEM TOGETHER. SIMULATE YOUR TOP-LEVEL DE1 MODULE IN MODELSIM BEFORE DOWNLOADING IT TO THE FPGA.** If you try to do everything by just downloading it to the FPGA and seeing if it works, you will have LOTS of trouble getting this lab working, and subsequent labs will be MUCH harder. Simulation and good, complete testbenches are your friend, and they will SIGNIFICANTLY speed up your debugging. Only once you have all of your submodules and the entire system working in Modelsim should you download the design to the FPGA and test the working game (the fun part!).

Note that during testing and debugging you may want a slower clock – you can always use the clock divider from lab #3 to help you in this process.

Lab Demonstration/Turn-In Requirements

- Demonstrate your tug of war design to the TA.
- Submit a short lab report that includes the following three sections:
    - Procedure
        - Describe how you approached the problem and any relevant truth tables, circuit schematics, or state machine diagrams.
        - Include the top-level block diagram for your entire design, showing the major modules and how they are interconnected.
    - Results
        - Include screenshots of your ModelSim simulations.
        - Describe what you tested in your simulations, and what the results show.
        - Include a screenshot of your "Resource Utilization by Entity" page.
        - Include the computed size of your design.
        - Give a brief overview of the finished project, referencing what was asked of you.
    - Problems Faced and Feedback
        - Describe any issues you had while completing the lab, and how/if you were able to overcome them.
        - Include tips and tricks that you found out about that may be useful to you in future labs.
- Submit the SystemVerilog files (files with extension .sv) for your tug of war design. Make sure to follow the commenting guide provided.
    - All of your modules, including the top level module, should include testbenches.
- Submit your report and files to Canvas.