Intro to Digital Logic, Lab 4
Communicating Sequential Logic

Lab Objectives

In the last lab we developed a single, simple FSM. Now we want to build a more complex system with multiple FSMs. Careful creation of a block diagram, and design and testing of each individual piece, 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. So please start early, be methodical, and thoroughly simulate and check each individual FSM before connecting the FSMs 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 using the KEY[0] and KEY[3] buttons, and the leds from LEDR9 to LEDR1, skipping LEDR0, as 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 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 shows 1 for first player, 2 for second player. You can use SW9 as the reset signal.

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.

You should use the 50MHz clock directly (pin CLOCK_50) to control the whole design – we'll assume no player can press the button faster than 25 million times a second…

User Input

Since we are using a fast clock, each time the user presses a button the button will be ON for many cycles, and OFF for many cycles. However, 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? This could be an input to the module.
- During play, it needs to know which button(s) were just pressed, whether its light is currently lit, and whether its right and left neighbors are currently lit.
- Given all that data, plus the reset signal, it's now easy to figure out whether you should be lit during the next clock cycle.

Victory

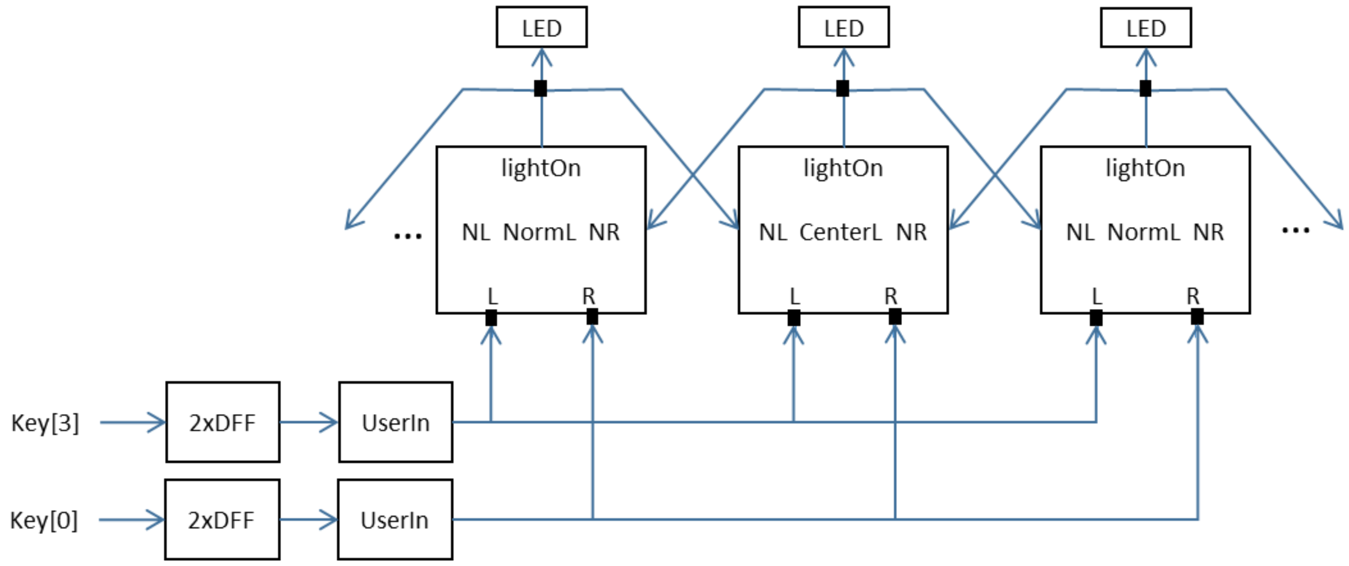
You can tell when someone wins by watching the ends of the playfield – when the leftmost LED is lit and only the left button is pressed, the left player wins. Similar logic can be found for the right player. So, build a unit that controls the HEX0 display, based on these victory conditions.

Suggested FSMs

Your playfield and victory lights could be controlled by the following FSMs. Note that these are only suggestions. You are free to create the 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
```

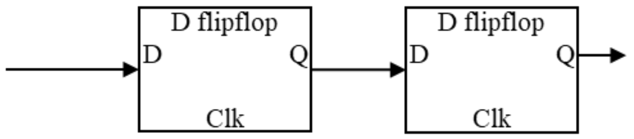
If you were to use the above FSMs in your design, you'd need 1 centerLight and 8 normalLight FSMs. In addition you'd need two instantiations of a userInput FSM, and logic to determine when someone has won the competitions. None of the FSMs should require more than four states.



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, your circuit may randomly screw up.**

To deal with metastability, make sure you send each of the user inputs (KEY[3] and KEY[0]) to a pair of D-flipflops in series 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 the input will go in as the D of the first DFF, and the circuit will listen to the Q output of the second DFF).



Overall

Build each of the pieces and test them independently in ModelSim before combining them together. TEST EACH ELEMENT IN MODELSIM BEFORE TRYING TO HOOK IT ALL UP. TEST THE WHOLE THING IN MODELSIM BEFORE DOWNLOADING TO THE FPGA. If you try to do everything by just downloading it to the FPGA 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, will SIGNIFICANTLY speed up your debugging. Only once you have all the pieces, and then 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 you may want a slower clock – you can always use the clock divider from lab #5 to help you in this process.

You will be graded 100 points on correctness, style, testing, etc. Your bonus goal is developing the smallest circuit possible – measure this the same way you did in lab #5. Note that the “Resource Utilization by Entity” report will give you the sizes of each of the modules in your design, so you can focus your sizing improvement efforts accordingly.

Lab Demonstration/Turn-In Requirements

A TA needs to "Check You Off" for each of the tasks listed below.

- Turn in the top-level block diagram for your entire design, showing the major modules and how they are interconnected.
- Turn in the Verilog for all of the elements of your design, including your testbenches. **You MUST have a testbench for the basic elements AND the ENTIRE design.**
- Demonstrate your working design to your TA.
- Demonstrate the simulation of your entire design to your TA.
- Turn in a printout of the “Resource Utilization by Entity” page. Write on this the computed size for your design.
- Tell the TA how many hours (estimated) it took to complete this lab, including reading, planning, design, coding, debugging, testing, etc. Everything related to the lab (in total).