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

The objective is to turn this rope-based game into an electronic version!.

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. It is crucial that you put together a block diagram of the whole system early in the design process to visualize how the different modules will be connected together. 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. **To avoid random behavior, you must deal with metastability in your code.**

  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 listen to the Q output of the DFF that receives that button as the D input).

- **Overall**

  Build each of the pieces and test them independently in ModelSim before combining them together. **You must write a testbench for every module and test it in ModelSim before combining all modules. You must test the combined designed in DE1-SoC in ModelSim before downloading the project to the FPGA. 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.**
  Note that during testing 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 the Tug of War on the DE1_SoC board in a video that shows the functionality of your project.
- Submit a short lab report that should include 3 main sections, detailed below.

  Procedure
  - Describe how you approached the problem and include state diagrams for all FSMs.
  - Include a top-level block diagram for your entire design, showing the major modules and how they are interconnected.

  Results
  - Include screenshots of ModelSim simulation for **all** modules. You must have a testbench for every module in the project.
  - Turn in a screen shot of the "Resource Utilization by Entity" page. Write the computed size for your design.
  - Describe what you tested in the simulation, and what the results in the screenshot show
  - Give a brief overview of the finished project, compared to what was asked
- Submit the SystemVerilog files (files with extension .sv). Make sure to follow the commenting guide provided. **A significant amount of grade will depend on the commenting style.**
- Submit your report, video and programs to Canvas.
- Export the code of any .sv file you have to a pdf version. Submit the Pdf version and the original .sv files to canvas.