Intro to Digital Logic, Lab 6

Communicating Sequential Logic

Lab Objectives

In the last lab, we developed a single FSM to accomplish our task. Now we want to build a more complex system with multiple, interconnected FSMs. Careful creation of a block diagram, along with 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 do proper unit testing (thorough simulation and check) of each FSM you design 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 that 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[3]` and `KEY[0]` buttons and inputs and `LEDR[9]` to `LEDR[1]` as a 9-light playfield. A single LED on the playfield will be lit to indicate the current position of the "ribbon/knot." A player wins by moving the light off his or her end of the playfield.

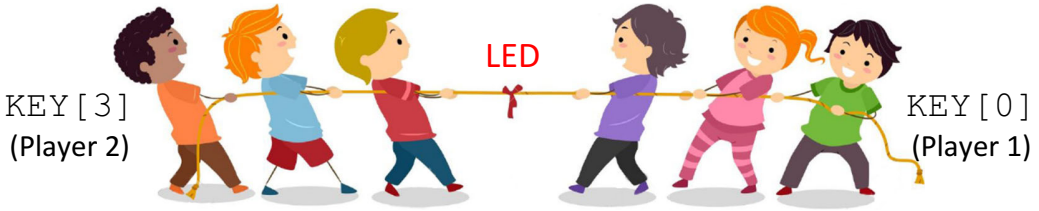

Figure 1: *Tug of war. For this lab, `KEY[0]` will tug the "ribbon" (an LED signal) right for Player 1 and `KEY[3]` will tug the "ribbon" left for Player 2.*

Design Rules:

- The "ribbon/knot" starts at the centermost LED (`LEDR[5]`).
- Player 1 presses `KEY[0]` to move the light one LED to the right and Player 2 presses `KEY[3]` to move the light one LED to the left.
- When a player wins, display the number (1 or 2) of the victor on the 7-segment display `HEX0`.
- Use `SW[9]` as the reset signal.
- You should use the 50 MHz 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…

If you tried to design the entire game as one big state machine, it would get pretty complex and testing would be incredibly difficult. Instead, we are asking you to break it down into smaller pieces for easier testing and for practice making interconnections between different parts of a system.

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 press of a button will span many cycles. Design a simple FSM that detects the moment the button is pressed, *i.e.* its output is TRUE for only 1 cycle for every button press. This will be used for all user input.

Metastability warning:

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 D flip-flop (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 the user input to a DFF *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).

Playfield

It is certainly possible to design a single FSM for all 9 lights, however, we want you to design an FSM for each location (LED). A given playfield light needs to know the following:

- Does it start as True or False? (you may want to create a separate module for the center light)
- Which button(s) were just pressed?
- Am I currently lit? Are my right and left neighbors currently lit?

With this information plus the reset signal, it's now possible to figure out whether or not this light should be lit during the *next* clock cycle.

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

   //  L - True when  left key (KEY[3]) is pressed
   //  R - True when right key (KEY[0]) is pressed
   // NL - True when the light to the  left of this one is ON
   // NR - True when the light on the right of this one is ON
   input logic L, R, NL, NR;

   // lightOn - True when this normal light should be ON/lit
   output logic lightOn;

   // YOUR CODE GOES HERE

endmodule
```

Figure 2: *Suggested Verilog starter code for a light. You may want to create a separate, but similar, module for the center light.*

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. Build a unit that controls the `HEX0` display based on these victory conditions.

Suggested Interconnections

The modules you design will read user input from KEYs and control the playfield (LEDR) and victory output lights (HEX). The modules described above are only suggestions. You are free to create the design using any number of different FSMs.


None of the suggested FSMs should require more than four states!

Build each of the pieces and test them independently in ModelSim before combining them together. Make sure that you can reset your system. 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 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!).


During testing you may want a slower clock. You can use the clock divider from Lab 5.

Lab Grading

Working Design: 100 points for correctness, style, and testing.

Bonus: Up to 10 points for developing the smallest circuit possible – measure this the same way as 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

Lab Report (*before* Wednesday section, submit as PDF on Canvas)

- The top-level block diagram, showing the major modules and how they are interconnected.
- For each of the major modules, include a state diagram (if applicable) and screenshot of the ModelSim simulation.
- A screenshot of the "Resource Utilization by Entity" page, showing your design's computed size.
- How many hours (estimated) it took to complete this lab in total, including reading, planning, design, coding, debugging, and testing.
- As *separate* Canvas file uploads, the Verilog code for all of the elements of your design, including the testbenches. You MUST have a testbench for the basic elements *and* the overall design.

In-Person Demo (*during* your demo slot)

- Demonstrate the simulation of your overall design.
- Demonstrate your tug of war game working on the DE1 board.

Lab 6 Rubric

Grading Criteria	Points
Q1: Top-level block diagram of your system	5 pts
▪ Explanation of modules and interconnections	4 pts
Q2: ModelSim screenshot (& state diagram) of input module	6 pts
▪ Explanation of waveforms (& FSM)	4 pts
Q2: ModelSim screenshot & state diagram of lights module(s)	6 pts
▪ Explanation of waveforms & FSM(s)	4 pts
Q2: ModelSim screenshot (& state diagram) of victory module	4 pts
▪ Explanation of waveforms (& FSM)	4 pts
Q2: ModelSim screenshot of top-level module	4 pts
▪ Explanation of waveforms	4 pts
Q3: Screenshot of Resource Utilization	8 pts
▪ BONUS for small resource utilization	(10 pts)
Time spent	2 pts
Verilog code uploaded	5 pts
LAB DEMO	40 pts
	100 pts