CMPE 415 - Homework #3

Rescue POW

Start assignment as soon as possible. Work individually, but you can ask your classmates for help when you get stuck, with consideration to the course collaboration policy (please read it in the course website). Please send me email if something isn't clear and I will update the assignment. Changes are logged at the bottom of this page.

Before getting started, you should go through the verilog notes located under Course Readings on the course home page.

A paper copy of everything and electronic copies of all your code and testing files (all in one zipped file) are due at the beginning of class on the due date.

Notes:

- [15% of points] Clearly state whether your design is fully functional, and state the failing sections if any exist.
- Make sure your design and code are easily readable and understandable (clear and well commented).
  - Up to 5% extra credit will be given for especially thorough, well-documented, or insightful solutions.
- *** Where three '*'s appear in the homework, perform the required test(s) and turn in a printout of either:
  1. a table printed by your verilog testbench module listing all inputs and corresponding outputs,
  2. an Isim waveform plot which clearly shows (labeled and highlighted) corresponding inputs and outputs, or
  3. a section of testing code which clearly compares the designed circuit and a simple reference circuit, and two short cut & paste sections of text from your simulation (one for pass, and one for fail where you purposely make a slight change to your reference code to make it fail) that look something like this:
    Error: input=0101, out_module=11110000, out_ref=11110001

In all three options, each test case must be marked whether the output is correct or not.

Keep "hardware" modules separate from testing code. Instantiate a copy of your processing module(s) in your testing module (the highest level module) and drive the inputs and check the outputs from there.

You are going to create a simple video game using the leds and the rotary pushbutton simulating saving POWs (relax, you can be a civilian in the right place at the right time to safely release the POWs in this game).
The games will go as follows:

- To start the game, press RESET/BTN_SOUTH. One or two random LEDs should be lit indicating the positions of the POWs. However, there is also an XOR overlay of a blinking LED at position 0 representing your position.

- As you turn the rotary switch, the your position moves (and the blinking cursor moves).

- Move towards one of the POWs. Once you have reached them, press the rotary button. This rescues the POW and the corresponding the LED no longer stays lit after the cursor is moved.

- When all the POWs are saved (1 or 2 depending on the initial randomization), the game is over. You must create some observable action to indicate the end of game, such as making all the LEDs blink or outputting an image to a monitor or using the LCD display.

**Your Part**

- To store the POW position(s), create an 8-bit flip flop module.
  - The declaration should be as follows: pow_position (pos,clk,rst,init,d);
    - pos is 8 bit output
    - init and d are 8 bit inputs
  - One every clock, load _init if rst is pressed and load d otherwise
    - init should be provided from random2
  - **YOU MUST USE THE ALWAYS BLOCK WITH NO ASSIGN STATEMENTS OR INSTANTIATIONS**

- To store your position, create an 8-bit shift register module.
  - The declaration should be as follows: position_shifter(pos,clk,rst,sr,sl);
  - On every clock, it should initialize to POS[7:0]=8'b00000001 when rst is high and otherwise it should right shift when shift_right is high and left when shift_left is high
  - **YOU MUST USE THE ALWAYS BLOCK WITH NO ASSIGN STATEMENTS OR INSTANTIATIONS**

- To create the LED display:
  - AND each of the 8 bits from the position register with the output of the pulser to create a blinking indicator. Then XOR this result with the output of the POW position flipflop.
  - The pulser will have a duty cycle less then 50% so that the cursor is shown as short blinks of light when not over a POW and the inverse when over a POW.
  - **YOU CAN USE WHATEVER YOU LIKE TO ACHIEVE THIS**

- You need to figure out what to provide to the d input of the pow position flip flop
- You need to figure out to create the effect of clearing some bits if the cursor position is the same as a POW position
• Use a NOR to figure out when the game is over
• Remember to create some indication that the game is over

**Turn in**

• Compiled bit file and all source files used to generate the program
• Create and submit one or multiple Verilog testbench modules that tests your design.
• Create a report that explains your design and your testing (Pleas follow the rules at the beginning)
  - Include the output of your Verilog testbench(s), with additional explanation as needed to convince someone that your design works and your simulation-based testing is sufficient
Turn in ***, option 1.