HW 3: Light the Candles

Due Oct 13, 2017

For this assignment you are going to create a game played on the dev board. In this game the player's goal is to light all the candles, which are represented by the 8 LEDs on the board just above the switches.

The game starts by the player pressing the RESET button (a.k.a. BTN_SOUTH) just below the rotatory knob.

The player is placed initially at a position 0.

The player may light the candles by jumping on them. This is done using the BTN_NORTH button on the board just above the rotary knob. A player may jump at the current position by setting the switches to zero and pressing BTN_NORTH. Any desired horizontal movement in the jump to another position is provide by the four switches which indicate a value from 7 to -8, in binary two's compliment.

Once a candle is lit, the player may jump again jump on the same location (using 0) or to another location using the delta provide by the switches.

Here is the catch. The candles you light will continually go out at random and must be relit. To be specific, a candle may at random go out at the same time you make jump. Thus the player does wants to make as few jumps as possible. Secondly, note that there is no direct indicator of where the player is so the player must keep track of For this assignment you are going to create a game. In this game the player's goal is to light all the candles, which are represented by the 8 LEDs on the board just above the switches.

Modules to Write and Test

Module 1: igniter

- igniter (player) module:
  - The declaration should be as follows: module igniter(position, delta, enable_jump, sys_clk,clr_n);
    - position: output that indicates the position of the igniter, this value should always be in the range [0,7]
    - delta[3:0], enable_jump: when the enable_jump signal is high and a positive clock edge arives, the position should change by the value provided from the signal delta
      - The new position should be calculated obeying modular arithmetic (mod 8), e.g. 6+2 becomes 0,6+3 becomes 1,2-4 becomes 6,4-2 becomes 2
    - sys_clk is the system clock signal
    - clr_n is an asynchronous clear signal, which sets all the position to 0
- Create a testbench to demonstrate proper behaviour
- Along with the module source and the testbench, provide a waveform or text output from the testbench simulation convincingly demonstrating proper functionality

Module 2: extinguisher

- extinguisher module (will allow a candle to be extinguished at random):
  - module declaration should be as follows: module extinguisher(active, position, enable, clk, clr_n);
  - within this module, implement an internal counter which counts from 0 to 15 repeatedly, it should increment with every cycle of the system clock (50Mhz) (this is so fast that it will effectively appear as a random value to a human that might sample its value repeatedly)
  - active is a 1-bit output signal, it should be high if the the enable signal is high AND the internal counter is at a value in the range [0 to 7] (thus only half the time indicating a candle to go off if not off already)
  - the position should be a value from 0 to 7 taken from the lower bits of the internal counter
  - enable is a 1-bit input signal, its use is described above (note that is does not affect the internal counter or pos). In the final game it will recieve a signal derived from the jump signal so that a candle might be extinguished every time the play makes a jump.
  - sys_clk is the system clock signal
  - clr_n is an asynchronous clear signal, which should set the position to 0
- Create a testbench to demonstrate proper behaviour
- Along with the module source and the testbench, provide a waveform or text output from the testbench simulation convincingly demonstrating proper functionality

Module 3: candle_controller

- Create a module to store and control the state of the candles.
- The declaration should be as follows: module candle_controller(candle_state, pos_to_set, set_enable, pos_to_clear,clear_enable, sys_clk,clr_async);
  - candle_state[7:0]: an output which indicates the state of the 8 candles (a high designates a lit candle)
  - whenever the set_enable signal is high, the pos_to_set[2:0] provided in the same clock cyle is used to set the corresponding candle bit high
  - whenever the clear_enable signal is high, the pos_to_clear[2:0] provided in the same clock cyle is used to clear the corresponding candle bit
  - a set should have precedence over a clear if both are commanded to the same position in the same clock cycle
  - sys_clk: is the system clock signal
  - clr_async: an asynchronous which whenever it is high the value of 0 should be loaded into the state register
- Create a testbench to demonstrate proper behaviour
- Along with the module source and the testbench, provide a waveform or text output from the testbench simulation convincingly demonstrating proper functionality

Module 4: Top Module to implement the game

At this point you can assemble the game from the parts you've created, any required glue logic, and an appropriate UCF. The results should be demonstrated on the FPGA board. I have provided a one-shot module that creates a one-cylce pulse every time the button is pressed and held to prevent multiple presses being registered due to bouncing (I don't know if it is required on this board or not but I thought it would be safer to do this). You should include this in your design. You should provide this and all source code including the ucf used for your project. (I do not mandate a testbench to be submitted for this module, but I recommend you create one or more and simulate as much of your design as possible before testing on the board.)

Special Note

When you implement your physical design, you may note that one of the switches is "trimmed" from the design. Give this some thought and investigate what it says about the arithmetic performed and the hardware synthesis process. Also consider how it relates to a players strategy in using the interface. You may include a discussion in your report if you like but I am not specifically asking for it.

Report

Create a report that briefly explains your completion for each part as well as the top module, and provide a description of the testing results. Your discussion should include a description of the tests performed and text or waveform outputs from stimulations. Provide pictures and text outputs directly in the report. If a text output is too long to include in a report, include the output as a file with your submission, then make reference to it and included only key output lines in the report itself. Also, make references to source files used where appropriate. Your report should be able to convince someone that your design works and your simulation-based testing is sufficient. Your final physical testing for the last module should be discussed as well.

Submission and Grading Guidelines

You must organize your submission into folders

1. Source Code and Xilinx Projects (cleaned)
2. Report folder, and extra test-benches
3. Bit and Log files (since these are removed by cleaning)

We expect the whole Xilinx Project directory but cleaned

1. Verilog Files (.v,.vhd)
2. Xise Project file (.xise)
3. UCF files (.ucf)
4. .prj files and

After final synthesis and before cleaning you should save your log files and file bit file. Include these in a separate folder

1. Bit file (.bit)
2. Log files (.log)

Grading Guidelines

Correctness (40 pts) e.g.

- Completeness of homework and matching specifications

Design (20 pts) e.g.

- Usage of Blocking and Non-blocking according to lectures
- Choice for Default case
- Unnecessary use of extra clocks in design

Test-bench and thoroughness of testing and reporting(15 pts) e.g.

- Use of $finish letting TA/Grader know that simulation is complete

Style (10 pts) e.g.

- Well commented code
- Meaningful variable names
- Meaningful file names

Documentation (15 pts) e.g

- Student should clearly state to what degree that the code is complete and works
- Block diagram of the design
- Students should clearly state Usage Instruction: if its Board assignment
- Readable waveforms

Late Policy: One or Two day late: 20/100 points off

-