HW 4: Snake

Due Oct 25, 2017 Preliminary Design Report (one PDF) including *drawing of the top level (hand drawing is ok)* module descriptions including IO descriptions and outline of functional specifications * your outline should also include a test plan

Due Oct 27, 2017 Final Project and Report You must include sufficient testbenches and the results of running them in your PDF report. Your report should explicitly discuss any changes from your Preliminary Design.

You are going to create a snake game much like this one:https://www.coolmath-games.com/0-snake

1 **Game description**

The game should conform to the following specifications:

- The game play area is set up as a 26x38 grid surrounded by an electric fence (blue). The game display should occupy the majority of the screen, don't make a game field that is 26\times 38 pixels on the monitor. Decide an appropriate size of for each grid.
- To start the game, press RESET/BTN_SOUTH. Upon release, a single grid point represents a snake (green) and a single grid point represents food (red).
- The snake starts by moving to the right one grid point roughly every 1/8th of a second (125 ms update interval).
- A player may change the direction of the movement by 90 degree clockwise or counterclockwise by using the rotary switch on the FPGA board. Every position (notch) change of the rotary switch should correspond to a 90 degree angle change.
- If the snake goes onto the fence, the game should freeze.
- If the snake goes onto the food, the length of the snake should increase by one grid segment on the next movement with a trailing body, per the behavior of the game shown in the provided link. The moving head be green, and the grown body should be cyan (green+blue).
- Each time the snake reaches the food, another food should be positioned in a manor seemly random to the user.
- Each subsequent time the snake eats food, the segment length should grow by one, up to a length of 32 (including the head).
- If the head of the snake overlaps a body segment then the game should freeze.
- If the length of the snake reaches 32, the game should instead increase in speed by reducing the update interval by roughly 10 ms..

2 **Design Approach**

The design approach and structure is up to you. Here is one modular approach that breaks up the design into many small pieces. I don't represent that this is the best or easiest. It just seemed like the easiest one to describe. More complex behavioral units would allow for different segmenting of the design.

- Display module: this module would accept the food position and all snake and body segment positions in order to represent them on the grid. It should interface to the "VGA" module used on the first HW/Tutorial. Don't forget to draw the fence in blue as well. If the food (red) overlaps any snake segment (green/cyan), the food should be shown on top. If the snake head overlaps the fence, the snake head should be shown on top.
- Food module: this module would internally choose a pseudo-random position and provide it as an output. It would also accept the snake head position as an input and output a signal if an overlap occurs. Upon overlap, a new position should be chosen.
- Snake module: this should track and update the head position and the body segments. It should accept an input that indicates to grow and four inputs to signal a move (up, down, left, right). It should output all head and body segment positions.
- End game collision: this should accept all snake and body segment positions and detect a game-stopping collision of the head with a fence or the body.
- "Pacemaker": periodically sends a one-clock-cycle "update" signal to other modules. (All modules should be driven by one system clock). The update interval should be stored internally and reduce the update interval based on control signals from other modules.
- other pieces as need...

3 **What to be sure to turn in**

- Submit the whole CLEANED ISE project folder(source files used to generate it) and instead of submitting only Verilog(.v) files (YOUR COMMENTING OF CODE WILL BE GRADED).
- Submit bit files in a separate directory
- Create and hand in one multiple Verilog testbench modules that test your design

- Create a report that briefly explains your design and your testing You must have one testbench for each module.
- Include the output of your Verilog testbench(s) in your report (THIS IS EXPLICITLY GRADED) with additional explanation about each testbench as needed to convince someone that each part of your design works and your simulation-based testing of each module is sufficient.

4 **Bonus**

For bonus points, implement the following features. Do not do these independently of each other; you may only turn in one implementation of the game. You MUST document that you have implemented these so that the grader will know to check for them. Full credit is based on graders' judgments of your design an implementation.

- up to +5 points: implement the apple update such that it does not appear over a body
- up to +5 points: implement a couple portal pairs along the fence, holes in the fence that transport the snake such that it emerges from another portal location along the fence.

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

Provided Files *https://eclipse.umbc.edu/robucci/cmpe415/attachments/pulser.v*
https://eclipse.umbc.edu/robucci/cmpe415/attachments/random2.v
https://eclipse.umbc.edu/robucci/cmpe415/attachments/rotator_oneshot.v
https://eclipse.umbc.edu/robucci/cmpe415/attachments/top.ucf

-