Lab 8

Final Project

Connor Aksama – 1778028

How does someone use/operate/play your design?

Instructions:

- To start the game, flip SW[9] to the on position. When you are ready to play, flip SW[9] back to the off position – the snake of green lights will start moving!

- The goal of the game is to move your snake, represented by a string of green LED lights, around the grid to eat as many apples, represented by red LED lights, as you can! Every time your snake eats an apple, the length of the snake increases.

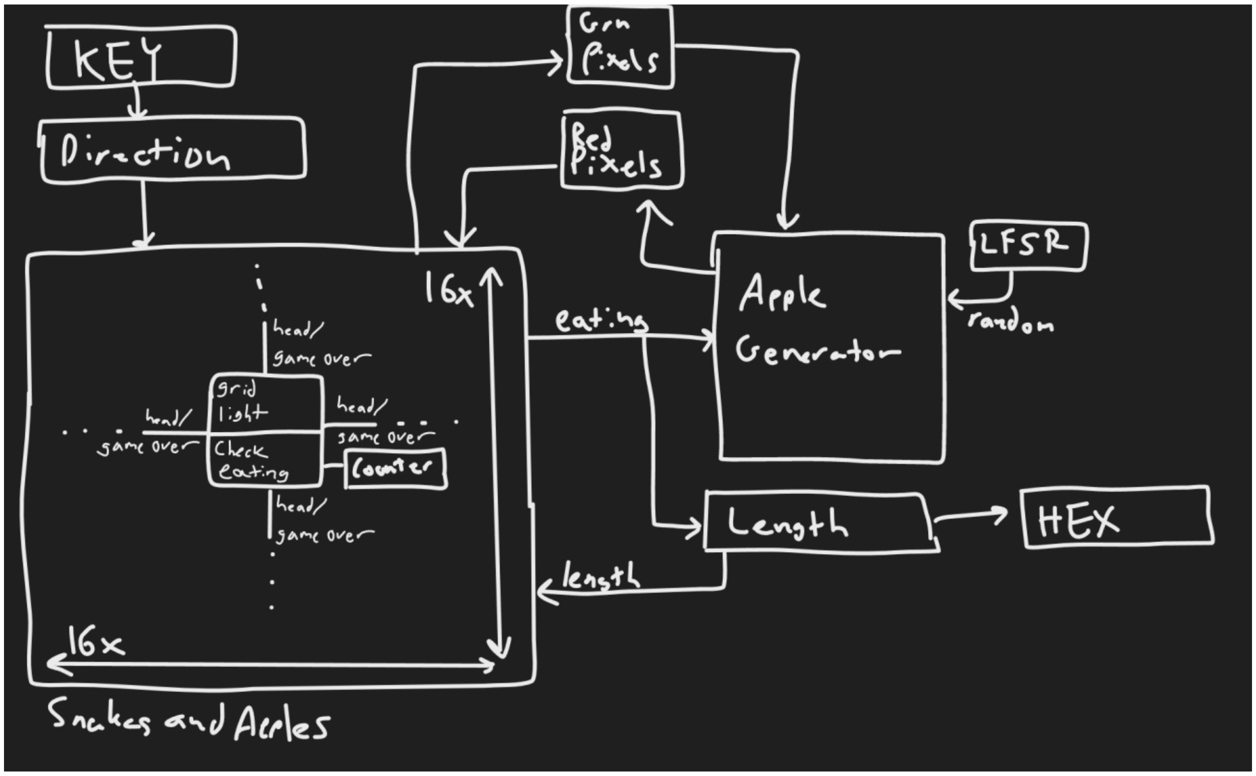
- To move the snake around the grid, use the four KEY buttons. Pressing KEY[0] will toggle your snake to move in the right direction, KEY[1] will toggle to the upwards direction, KEY[2] will toggle down, and KEY[3] will toggle left. Note: Your snake can only change direction perpendicular to where it is currently moving – i.e. If your snake is currently moving downwards, you can only toggle its direction to the left or to the right.

- Moving the snake beyond the top row of lights will wrap the snake around to the bottom row, and vice versa (e.g. PacMan). Same thing for the left and right edges of the grid.

- If the head of your snake ever collides with any other part of your snake, the game ends! Your score is viewable on the HEX display. Similarly use SW[9] to reset the game and play again!

A block diagram of your entire system with a brief description of the interconnections

Top-Level Block Diagram



The top-level block diagram for this project.

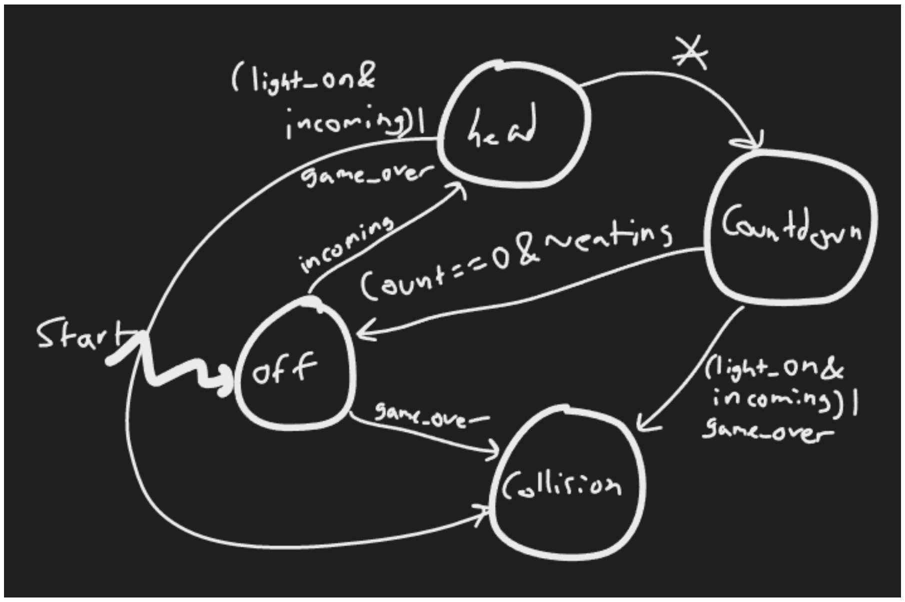
- The "Direction" module takes in the input from the KEY buttons and handles the allowable direction changes, which it feeds into the "Snakes and Apples" module.
- The "Snakes and Apples" module instantiates the 16x16 grid of "Grid Light" modules and "Check Eating" modules, which handle how long each snake light should remain on, as well as checking for collisions and apple-eating events (which uses information from the red pixel grid). Each "Grid Light" module communicates with its neighbors about which cell is the "head" of the snake, as well as whether the game is over or not. "Snakes and Apples" ultimately drives the green pixels output.
- The "Apple Generator" module handles creating new apples on the playfield whenever an apple is eaten by the snake (this is determined by an output signal from the "Snakes and Apples" module). This generator uses a random number from an LFSR to pick a new grid position. This module ultimately drives the red pixels output.
- The "Length" module keeps track of the length of the snake. This increments a counter when the "Snakes and Apples" module outputs an "eating" signal, and informs each of the grid lights for how long it should stay on. This module also drives the HEX display, which displays the player's score.

A brief description of each module you created and its purpose. For modules that include finite state machines, include a state diagram for each one. Include ModelSim simulation results that reasonably prove each module's correct behavior.

Grid Light Module

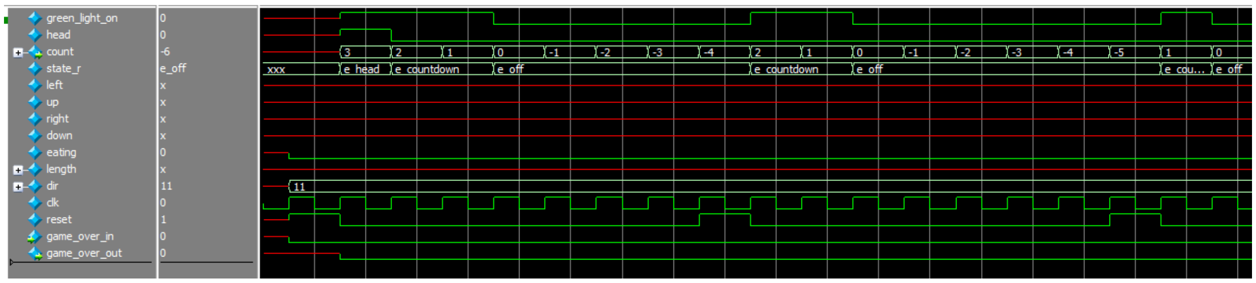
The purpose of this module is to conceptually control each grid light on the LED board. This module takes in information about the "Head" state of adjacent grid lights, the length of the snake, and outputs information about its "Head" state, whether its corresponding light should be turned on, and whether the game is over or not.

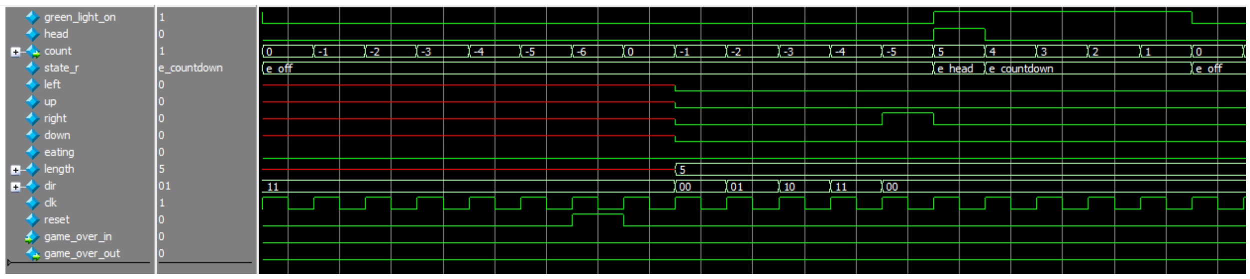
Grid Light Finite State Machine

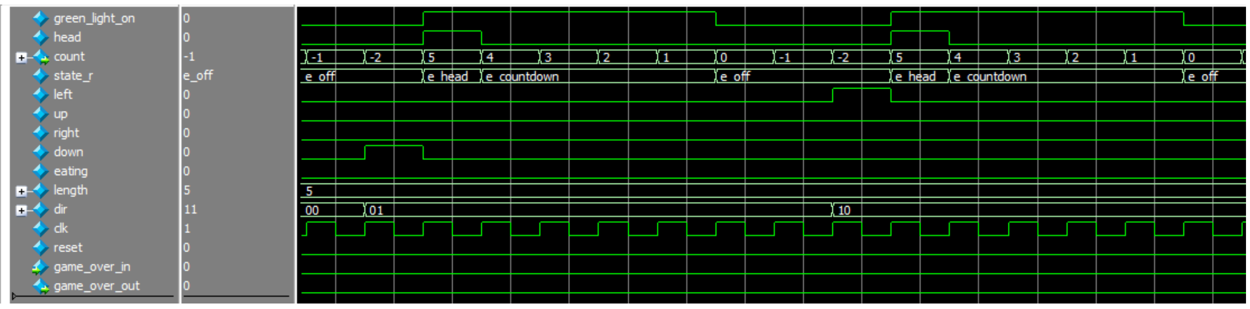


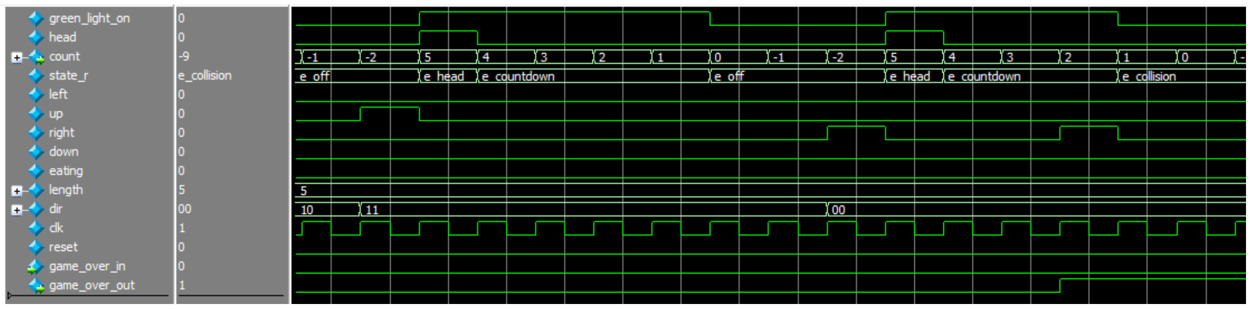
The finite state machine for the "Grid Light" module. The input "incoming" in the diagram indicates when there is an adjacent grid light in the "Head" state that is incoming into the current grid position (i.e. the grid light to the left is in the head state, and the current direction is to the right). In any case the light is on ("Head"/"Countdown"), and there is an "incoming" input, or the game is over, this puts the FSM in a "Collision" state. The FSM is in the "Head" state for 1 cycle, and stays in the "Countdown" state until its internal counter hits 0, and the eating process has finished.

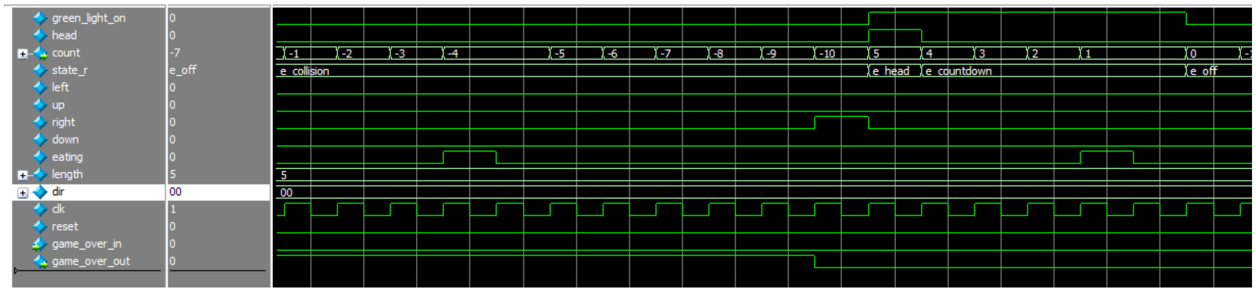
Grid Light Module Simulation









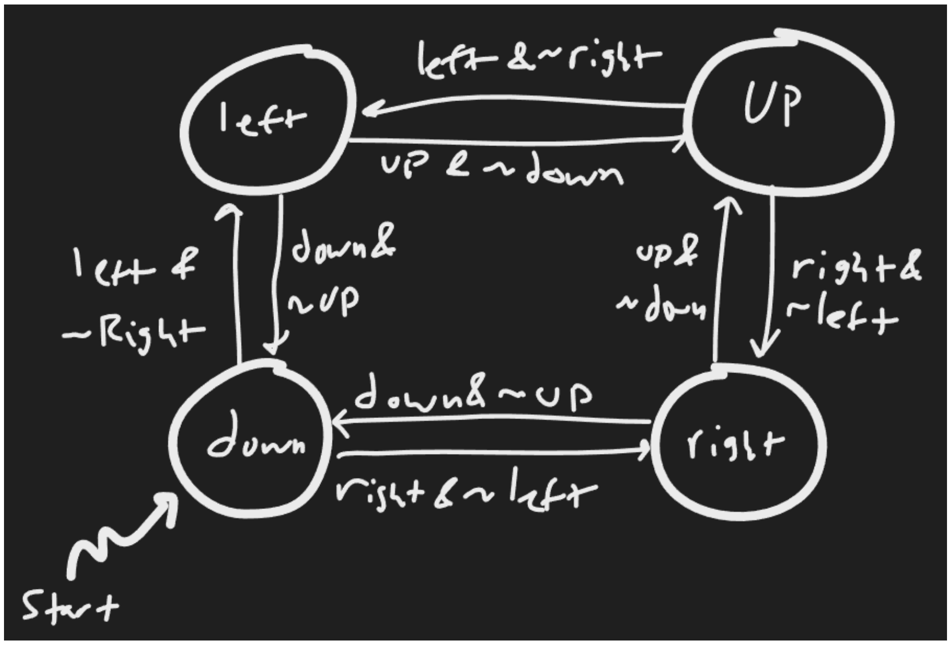


This simulation first shows how the current light stays off if there is no incoming head, then shows that the light does turn on for the appropriate conditions for an incoming head. The simulation also shows that when the light is turned on, it outputs a on signal for the corresponding pixel for the appropriate number of clock signals in different cases. Additionally, we can see that when the snake eats an apple, the duration of the countdown state is appropriately changed. In the event of a collision, we can see that the module stays in a "game over" state until a reset is received.

Direction Module

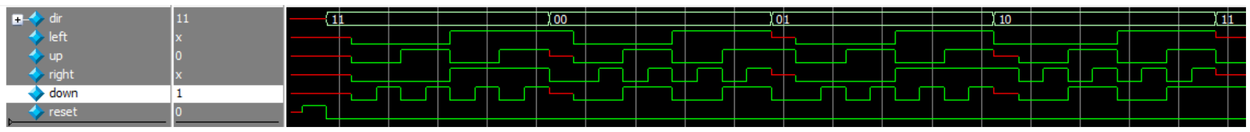
The purpose of this module is to properly update the movement direction of the snake given inputs from the user from the KEY buttons. This module needs to control the direction changes as they come from the user, in order to make valid direction changes.

Direction Module Finite State Machine



The finite state machine for the "Direction" module. In this FSM, the direction of the snake can only change perpendicular to the current direction, and when exactly one of the corresponding direction is enabled. For example, transitioning from right to up requires an on right signal and off left signal.

Direction Module Simulation



The simulation for the "Direction" module. This simulation shows that for any "invalid" inputs for the given state (attempting to toggle down while moving up), the direction does not change. The simulation also demonstrates that for each valid direction change, we only care about the two relevant inputs, the other two inputs can be in the X state.

Snakes and Apples Module

The purpose of this module is to instantiate the array of “Grid Light” modules and make the necessary connections between these instantiations. This module also handles much of the logic that detects certain events, including when the snake is eating an apple, when the game is over. The module also handles updating the top-level length of the snake.

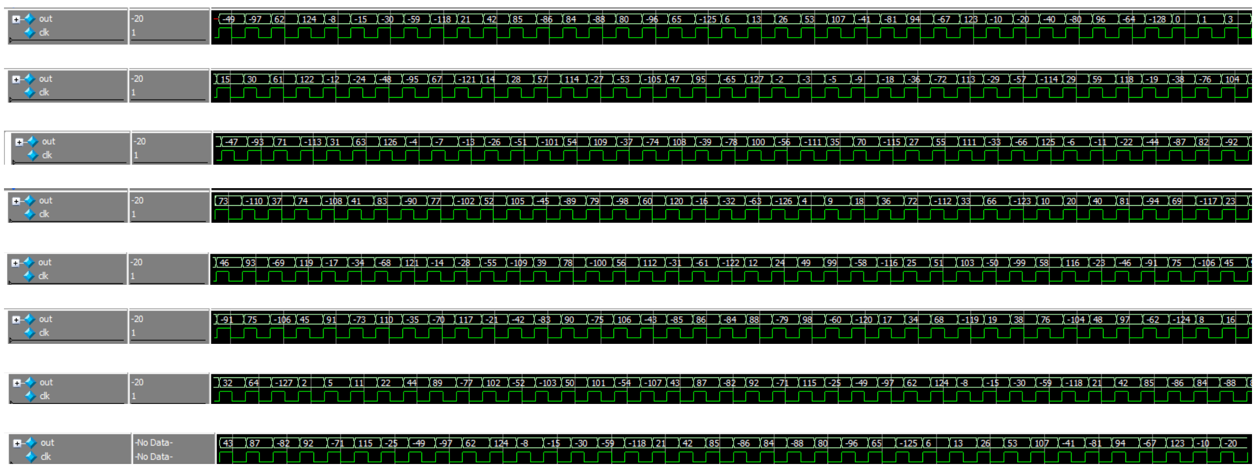
Triple Seven Segment Module

The purpose of this module is to handle taking in an 8-bit binary number and outputting a 3-digit decimal number to the HEX display.

8-Bit LFSR

The purpose of this module is to generate a pseudo-random 8-bit number, to be used for apple generation.

8-Bit LFSR Simulation

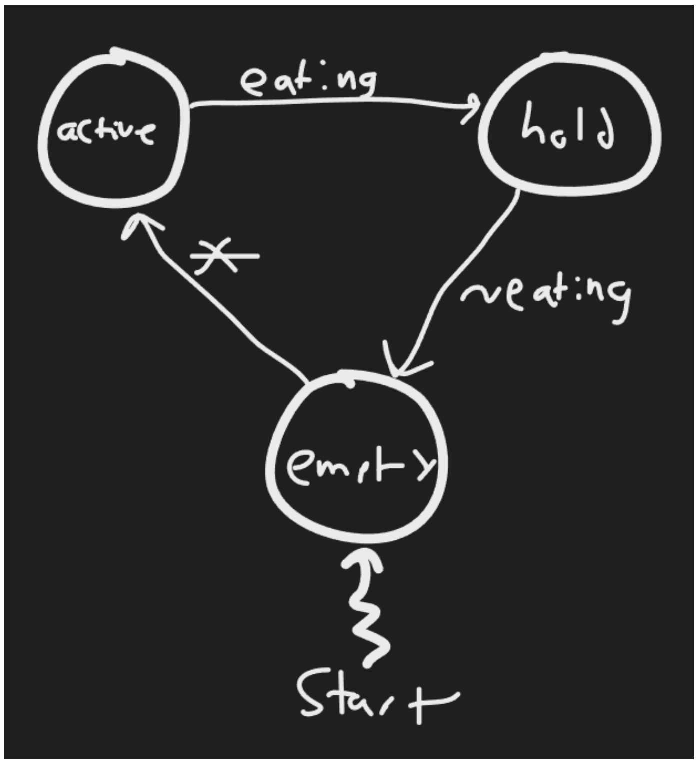


This simulation shows all the different states that the LFSR visits. This register shifts bits to the left on each clock edge, and shifts in the XNOR of bits 8, 6, 5, and 4 (1-indexed), as given by the LFSR table from Lab 7.

Apple Generator Module

The purpose of this module is to handle generating a new random apple on the playfield. The module detects when the snake is eating an apple, and when the grid is empty of an apple. The module then repeatedly tries to pick a position for an apple. If there is a conflict between the generated position and an already on light, it will try to pick a different position.

Apple Generator Module Finite State Machine

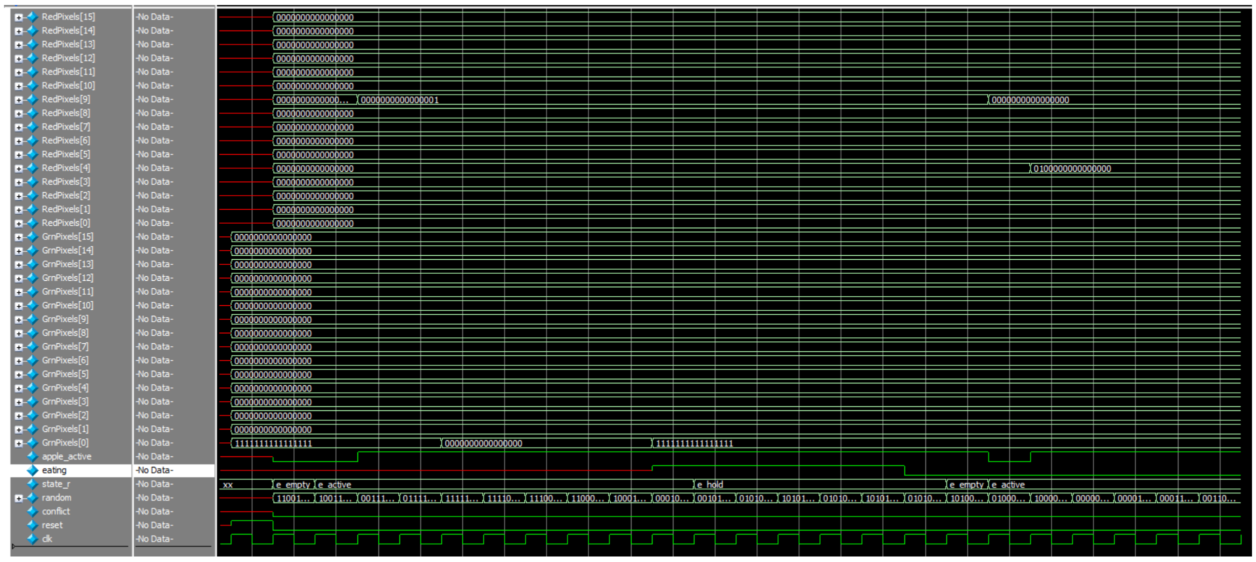


The finite state machine for the "Apple Generator" module. After exiting the "Empty" state, the module resets the red pixels in the grid and enters the "Active" state where it tries to generate a random apple in the grid. Once the snake begins eating the apple, the state machine enters the "Hold" state where it awaits the eating signal to toggle down, at which point it will enter the "Empty" state again.

Check Eating Light Module

The purpose of this module is to determine whether the snake is currently eating an apple or not. One instance of this module is created for each grid light in the playfield and takes in information about whether its corresponding light is an apple, and whether the head of the snake is incoming into that grid position.

Apple Generator Module Simulation

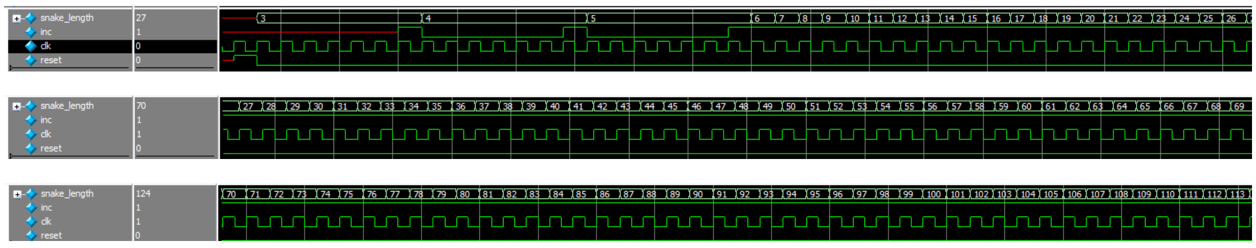


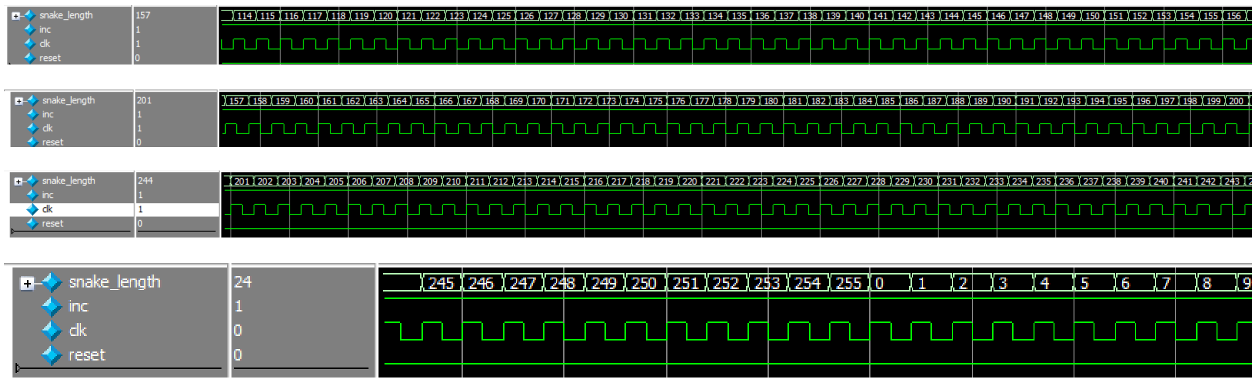
The simulation for the "Apple Generator" module. The simulation starts in the "Empty" state, then generates an apple in the playfield, then entering the "Active" state. Then, the snake eating the apple is simulated, and we can see that while in the "Hold" state, the red pixel grid remains unchanged while the snake is eating the apple. Once the eating signal toggles down, the red pixel grid is reset, and a new apple position is chosen.

Length Module

The purpose of this module is to keep track of the overall length of the snake. When the snake eats an apple, this module will increment its count by one. The individual grid light modules use the instance of this module to independently keep track of how long it should stay on, once enabled.

Length Module Simulation

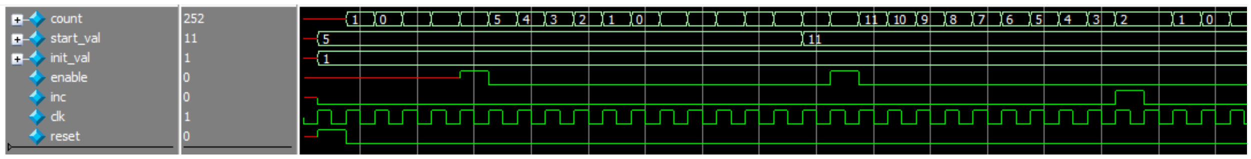




This simulation demonstrates how the module properly starts at length 3 (the initial length of the snake), then increments the count by one for every clock cycle where the "inc" signal is high, and properly increments by 1 up to 255.

Counter 256 Module

The purpose of this module is to provide each individual grid light module the ability to keep track of how long it should stay on. This counter module is given some starting number, which it registers when given an enable signal, then decrements that count by one for each clock cycle. The module also handles incrementing its internal count when the snake eats an apple.



This simulation shows that the module correctly takes on the value given by the "init_val" bus after receiving a reset signal, then correctly counts down to 0. When the enable signal is raised, the count value takes on the value on the "start_val" bus, and again correctly counts down to 0. Finally, notice that when the "inc" signal is raised, the count is successfully increased by 1 by effectively not decrementing the count on the next clock edge.

Column Select Module

The purpose of this module is to select a particular column in the playfield where an apple should be generated. The module takes in an integer n, and outputs a 16-bit string of 0s with a 1 in the nth position. The apple generator module uses this module to select a random grid position for an apple.

Check Conflict Module

The purpose of this module is to check for a conflict between a randomly generated apple and the current position of the snake in the playfield. The module also checks to see if there is an apple currently in the playfield or not.

Lab 8 Module

The purpose of this module is to act as the top-level module for the project, instantiating the major modules, connecting these instances together, and wiring in the physical input and output ports.

Briefly describe how you went about testing your system.

My methodology for testing this system was to do it iteratively after completing a major portion of each module. As with other labs, at this point I would create the testbench for the module, run the simulation in ModelSim, and then identify any errors. Once this particular portion of the module was working as intended, then development would continue with the rest of the module, or to the next module that needed to be created.

After a group of necessary modules were created, a separate higher level module would be created to connect the necessary parts together, then this module would itself be tested in a similar manner to detect any timing/synchronization issues when the modules were working concurrently.

Time Estimation

This lab took approximately 15 hours, in total, to complete.