Homework 5: Snake Levels and Obstacles Report
November 18, 2017
Sabbir Ahmed

1 Background

For this assignment, the previous Snake Game from HW04 was reimplemented with additional features such as levels and obstacles.

The game was to conform to the following specifications:

- The obstacles should be drawn in magenta, but otherwise are like the fence and the game should stop once the snake (head) overlays an obstacle.
- The game field should be initiated with no obstacles in the field of play for Level 0, and proceed just as in HW4 until the 5th apple is eaten.
- Every time the player eats 5 apples within a level, a new level should be generated with 10 more obstacles than the previous level and play should restart on that level.
  - Level 0 has no obstacles, Level 1 has ten obstacles, Level 2 has twenty obstacles, and so on...
  - The snake size is reinitialized to 1 to start each level
- The level score, and the level should be displayed on the LCD display. The display should be as follows:

  ```
  Lyy
  ```

  with (yy) denoting the current level.
- When the game ends, append an E as follows:

  ```
  LyyE
  ```

2 Design Approach

Several discrete modules from the previous implementation were used in this version. The `direction`, `food_pos` and `pacemaker` modules were left unmodified. The `snake_pos` module was modified to truncate the size of the snake from 32 to 5. The `vga_layout` module was modified to include the new coordinate pairs of the obstacles. Additional submodules: `game_state` and `lcd_driver` were integrated into the design.

These submodules were connected using a top level module that may be visualized with the schematic diagram configured as a block diagram in Figure 1. All the modules implicitly accept clock cycles as inputs.

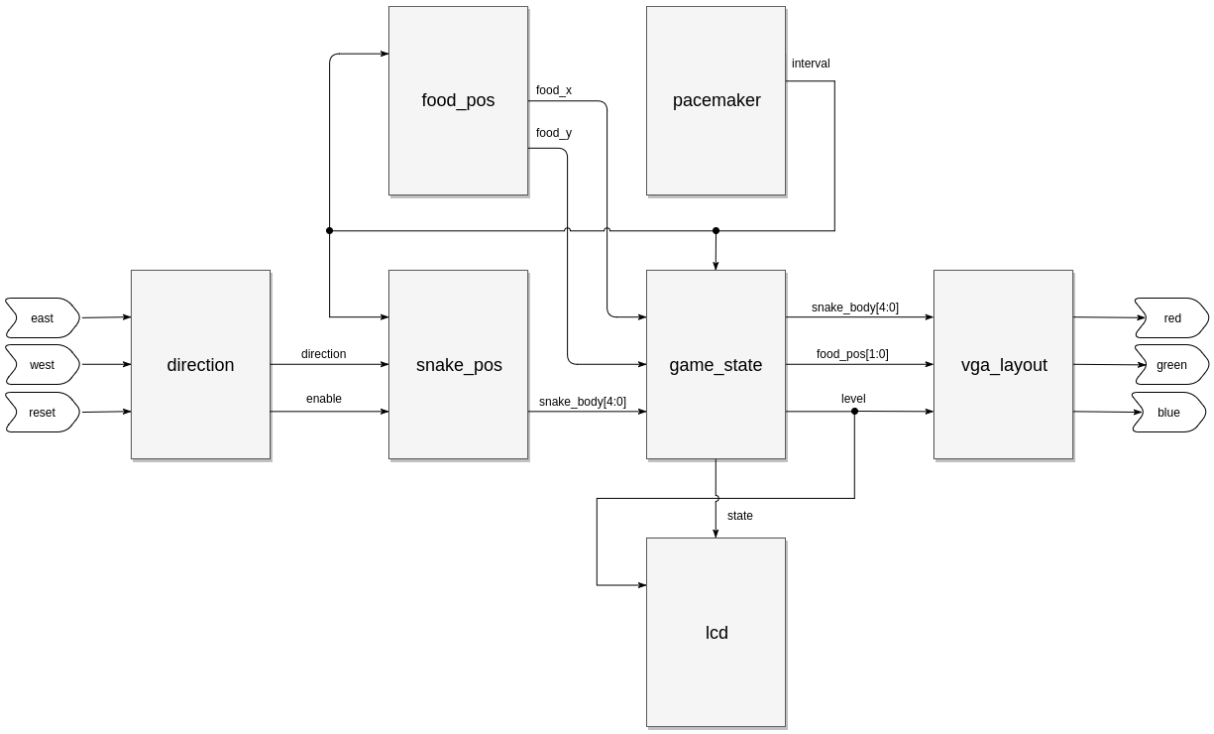

Figure 1: Block Diagram of the Implementation of the Game

2.1 direction

The `direction` module is used to control the user inputs. The inputs are one-shotted, debounced and fed into the internal state machine to determine the direction the user intended. This module sets an enable to the `snake_pos` module to notify a change in direction.

The sample output demonstrates the `dir` signal incrementing when `east` is high and decrementing when `west` is high.

2.2 food_pos

This module generates the `food_x` and `food_y` coordinates of the food when enabled by the `collision` module. The module combinedly utilizes an internal counter and a linear feedback shift register to generate the pseudo- random coordinates.

The sample output demonstrates the seemingly random coordinates generated for `food_x` and `food_y`.

2.3 snake_pos

The `snake_pos` module generates the coordinates for the 32 segments of the snake body, including its head. The module takes in the 2 bit direction from `direction` and 2 enable control signals from `collision`. The control signals, `grow` and `dead` are used to indicate the state of the snake body. If `grow` is enabled, the module utilizes `dir` to shift the body segments. If `dead` is enabled, the body segments freeze to indicate end of the game. Its clock is timed by `pacemaker` to control the speed of the moving snake body.

The signals were reorganized to highlight the relevant waveforms. The sample output demonstrates the movement of the individual body segments by utilizing the internal shift register. Only the initial segments change due to the snake body's growth being restricted to a length of 2 in the test bench.

2.4 collision

`collision` accepts the coordinates of the food and the snake segments and determines if a collision has been detected. If a collision has not been detected, it sends out an enable signal to the `snake_pos` module. If a collision with the snake body, specifically the snake head, with the food is detected, the module sends a signal to `pacemaker` to determine the interval at which the snake should move. This module has an internal counter that speeds up when the snake head had made 32 collisions with the food. If a collision between the snake head and the fence is detected, the game is frozen.

The signals were reorganized to highlight the relevant waveforms. The sample output indicates the conditions to which a collision is classified as `grow` and `dead`.

2.5 vga_layout

This module draws the fence of the game, and the snake and the randomly placed food on the VGA display.

2.6 Other Modules

Other minor modules have been utilized for the implementation. `pacemaker` is used to send out control signals to the other modules such that they update at reasonable rates. The module consists of an internal counter that speeds up the update rate once the game has registered over 31 bites of the food. `vga_sync` is used to synchronize the outputs to the VGA display.