Homework 5: Snake Levels and Obstacles Report
November 19, 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. The speed-up of the snake feature after a certain condition was removed. 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.

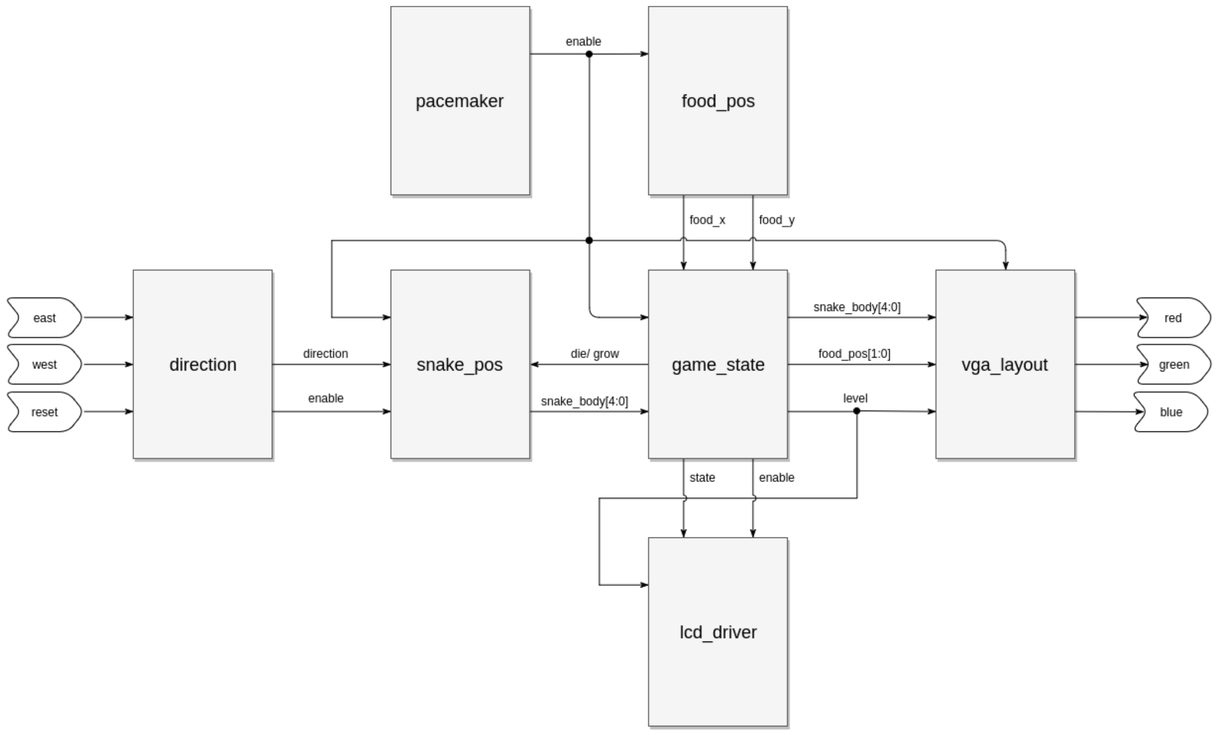

Figure 1: Block Diagram of the Implementation of the Game

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.

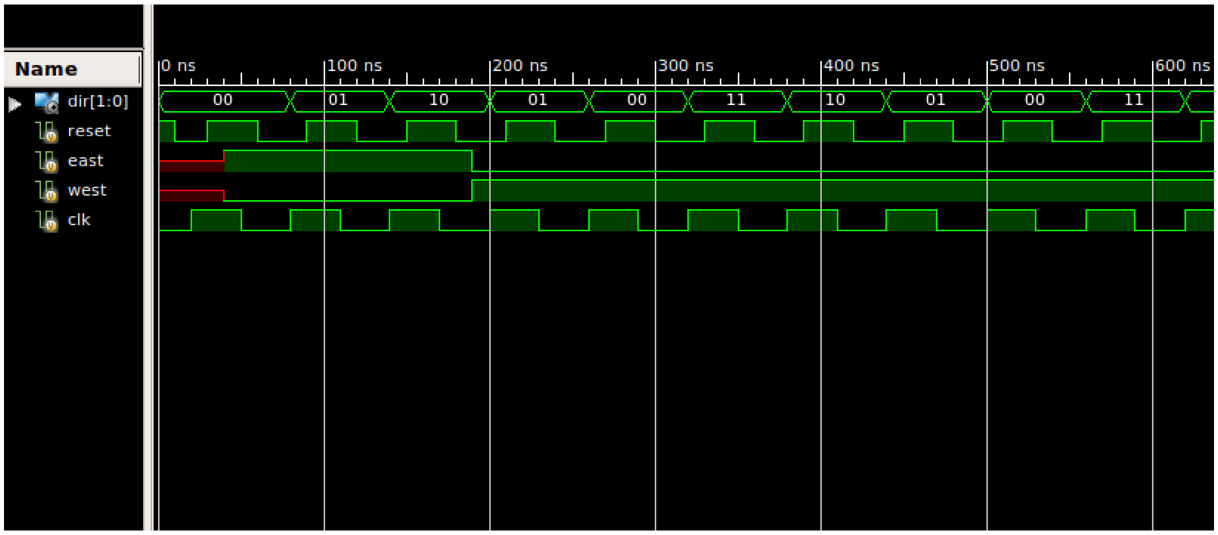

Figure 2: Waveform of the Testbench of `direction`

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

2.2 Food

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.

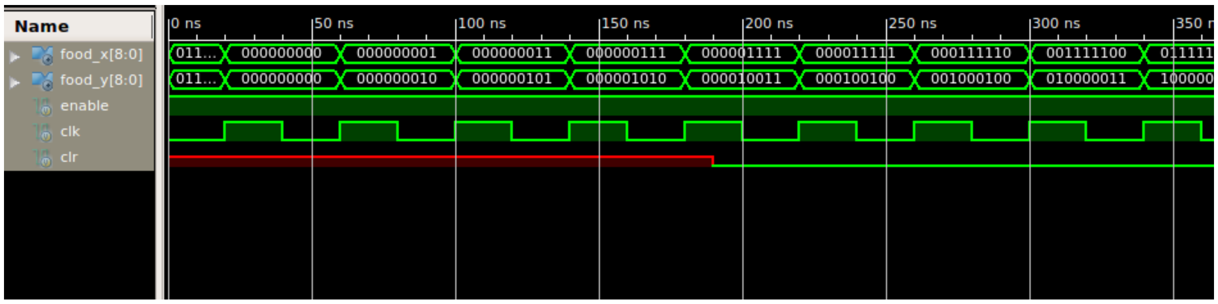

Figure 3: Waveform of the Testbench of `food_pos`

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

2.3 Snake Segments

The `snake_pos` module generates the coordinates for the 5 segments of the snake body, including its head. The module takes in the 2 bit `dir` input from `direction` and 2 1-bit control signals from `game_state`.

The control signals, `grow` and `die` are used to indicate the state of the snake body.

- If both `grow` and `die` are disabled, the module utilizes `dir` to shift the body segments.
- If `grow` is enabled and `die` is disabled, the module decrements its internal masking register to enable an extra snake segment. The snake segments do not move when they grow.
- If `die` is enabled, the body segments freeze to indicate end of the game.

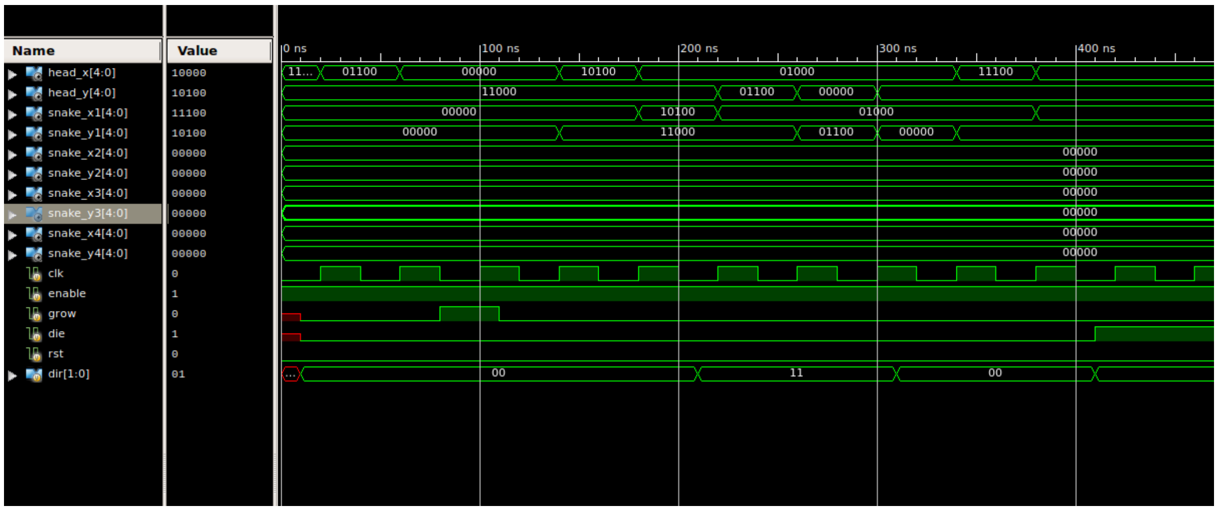

Figure 4: Waveform of the Testbench of `snake_pos`

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.

2.4 Collisions and Levels

The previous `collision` module was integrated into `game_state`. The module checks for collisions in addition to implementing several of new features. The current level of the game is determined by the internal state machine of the module. The states proceed to their corresponding next states based on serialized checks on collisions with the snake's head with its body segments, the fence, and the obstacles per level. Only the third and fourth segments are used to check for collisions between the snake's head with its body since checking the other segments are redundant.

The module 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 with the food is detected, the internal register `bite` is incremented. If a collision between the snake head and the fence or any of the obstacles is detected, the game is frozen.

The state machine consists of states indicating individual levels of the game. Since the game is composed with 3 levels including the zeroth level (00, 01, 10, 11), the state machine has 4 states. Each of these states generate their corresponding obstacles. The states also communicate with the LCD driver to send characters containing the level information.

2.5 vga_layout

This module draws the fence of the game, and the snake and the randomly placed food on the VGA display. The module also uses its input signal `level` to determine which obstacles to display.

2.6 Other Modules

Other modules have been provided to be utilized for the implementation. `pacemaker` is used to send out enable signals to the other modules such that they update at reasonable rates. `vga_sync` is used to synchronize the outputs to the VGA display. `lcd_driver` has been provided to communicate with the LCD of the board.