FPGA experiment report

Part 1 - Status update

Module name/Step	status	notes/comments
Ps2_Interface	done	This module is from past experiments
VGA_interface	done	This module is from past experiments
Seg_7_Display	done	This module is from past experiments
Lim_Inc	done	This module is from past experiments
CSA	done	This module is from past experiments
FA	done	This module is from past experiments
Snake_game	done	This is the TOP module
Snake_Movement	done	
tb_SnakeMovement	done	
ScoreCounter	done	
tb_ScoreCounter	done	
SnakePixelRenderer	done	
tb_SnakePixelRenderer	done	
SnakeGameLogic	done	
tb_SnakeGameLogic	done	

Issues:

In this section please describe the current issues you are facing.

- Per each module and simulation, if you receive any errors at the time of submission, please describe which errors and what they mean.

Fill here:

1. Ps2_Interface:
2. VGA_interface:
3. Seg_7_Display:
4. Lim_Inc:
5. CSA:
6. FA:
7. Snake_game:
8. Snake_Movement:
9. tb_SnakeMovement:
10. ScoreCounter:
11. tb_ScoreCounter:
12. SnakePixelRenderer:
13. tb_SnakePixelRenderer:
14. SnakeGameLogic:
15. tb_SnakeGameLogic:
16. Elaborated Design:
17. Synthesis:
18. Setting up constraints:
19. Implementing the design:
20. FPGA programming and debugging:

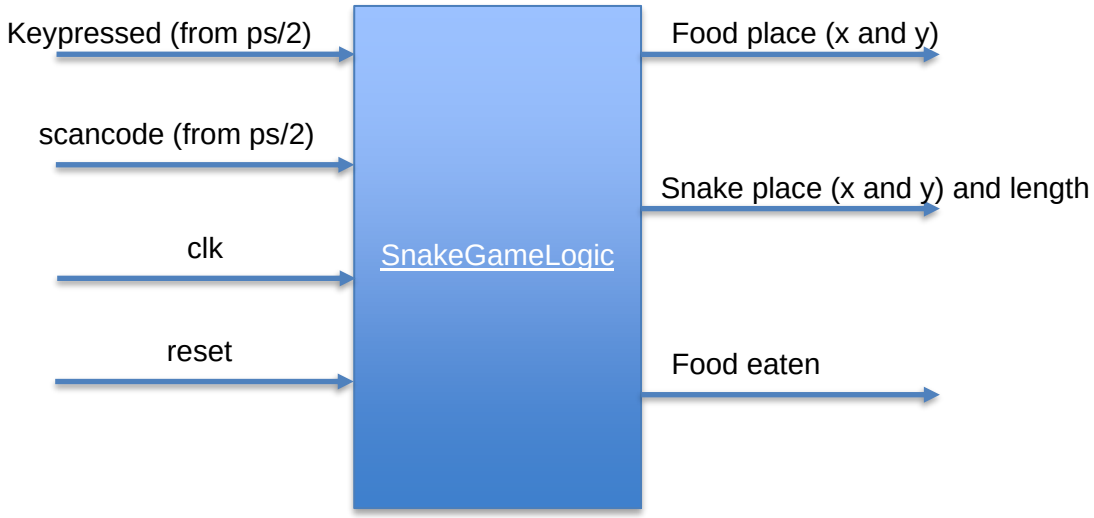
Block diagram:

The Ps2_Interface & VGA_interface & Seg_7_Display & Lim_Inc & CSA & FA

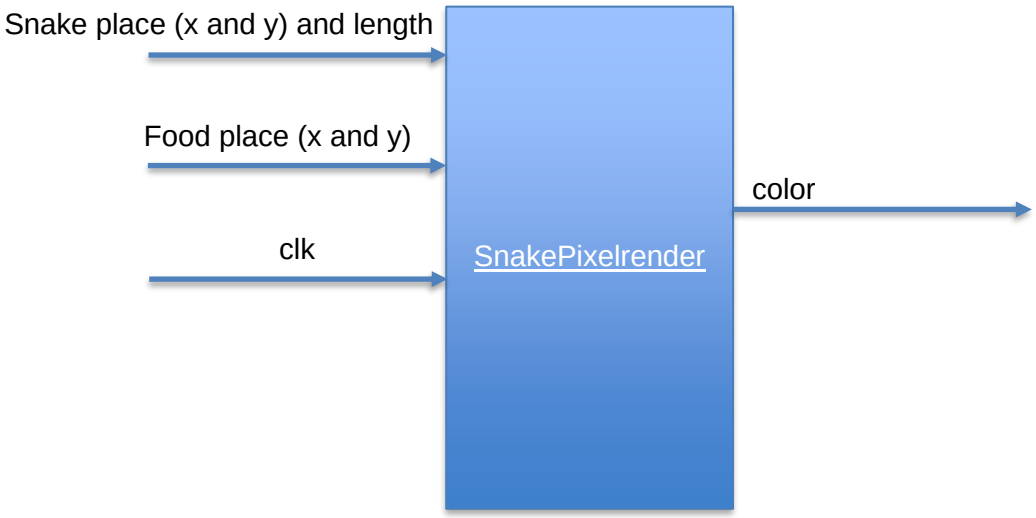
Are the same from past experiments

The ps/2 gives snake game the inputs

SnakeGameLogic: this block is responsiple for the movement logic of the snake and the food random generator.

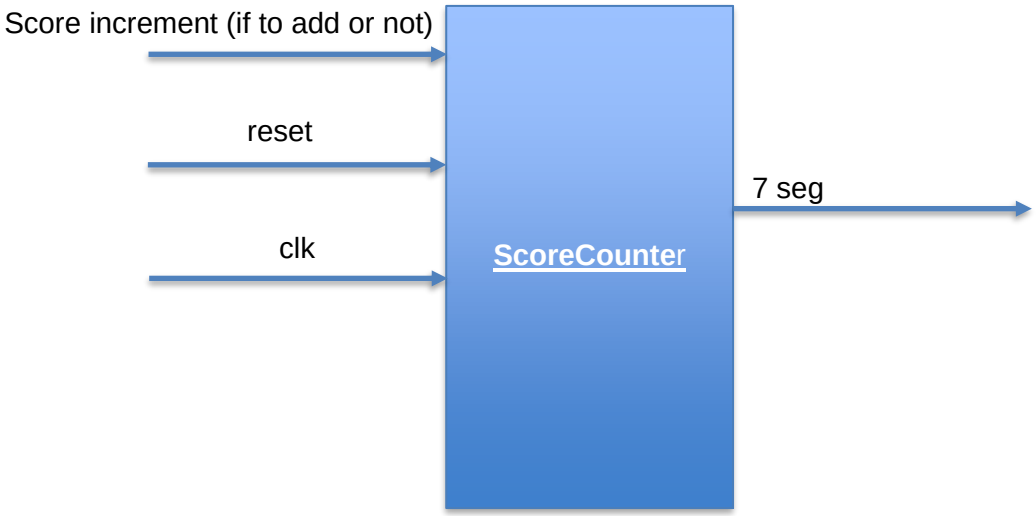


renderer:

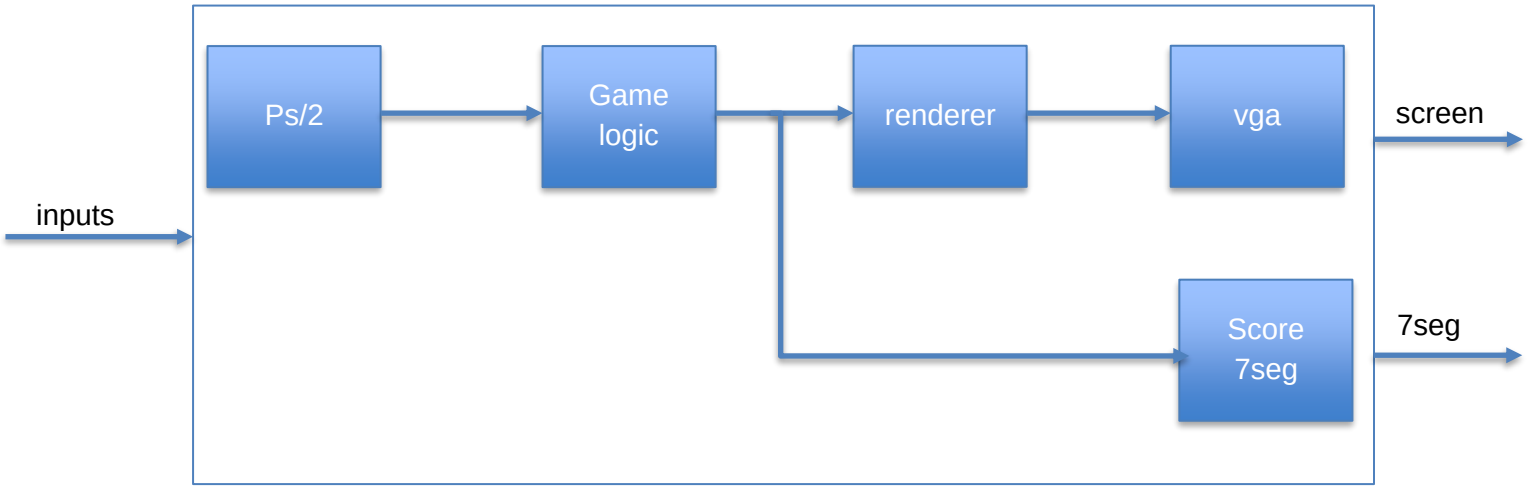


The outputs goes to the VGA and to the screen

ScoreCounter:



The output goes to the 7 seg on the fpga board



Part 2 - Simulations and answers

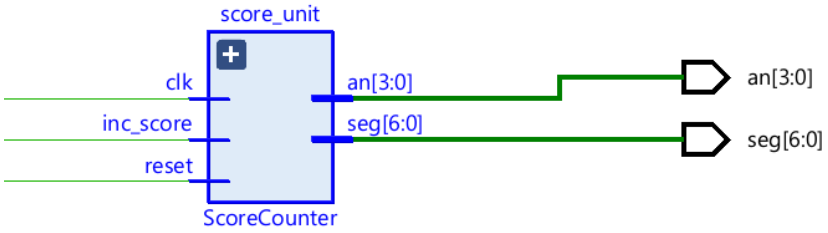
Fill here:

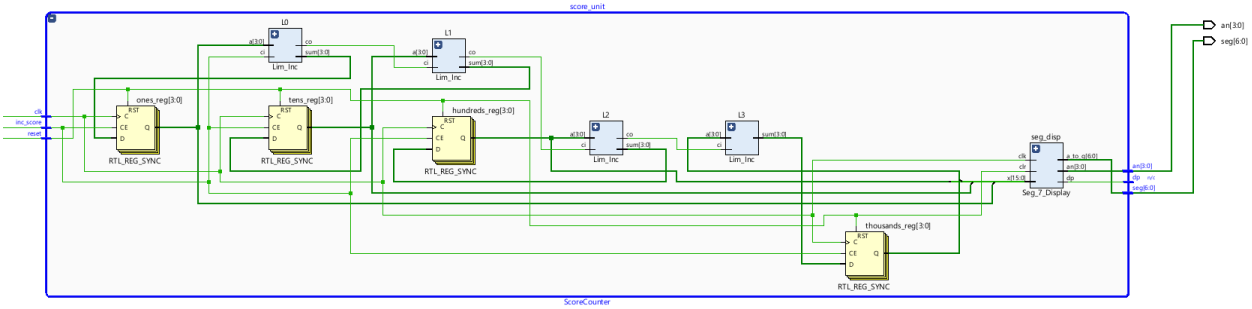
1. Ps2_Interface & VGA_interface & Seg_7_Display & Lim_Inc & CSA & FA

   Are the same from past experiments

2.ScoreCounter:

The ScoreCounter module is designed to track and display the player's score during the snake game. It implements a 4-digit BCD (Binary-Coded Decimal) counter using a chain of four Lim_Inc modules, each responsible for incrementing a single decimal digit: ones, tens, hundreds, and thousands. The score increases by one each time the inc_score signal receives a rising-edge pulse, which occurs when the snake eats an apple.
The module receives three inputs: clk ,reset, and inc_score (the pulse that triggers an increment). It outputs the current score to the 7-segment display. The display logic is handled by the module Seg_7_Display.



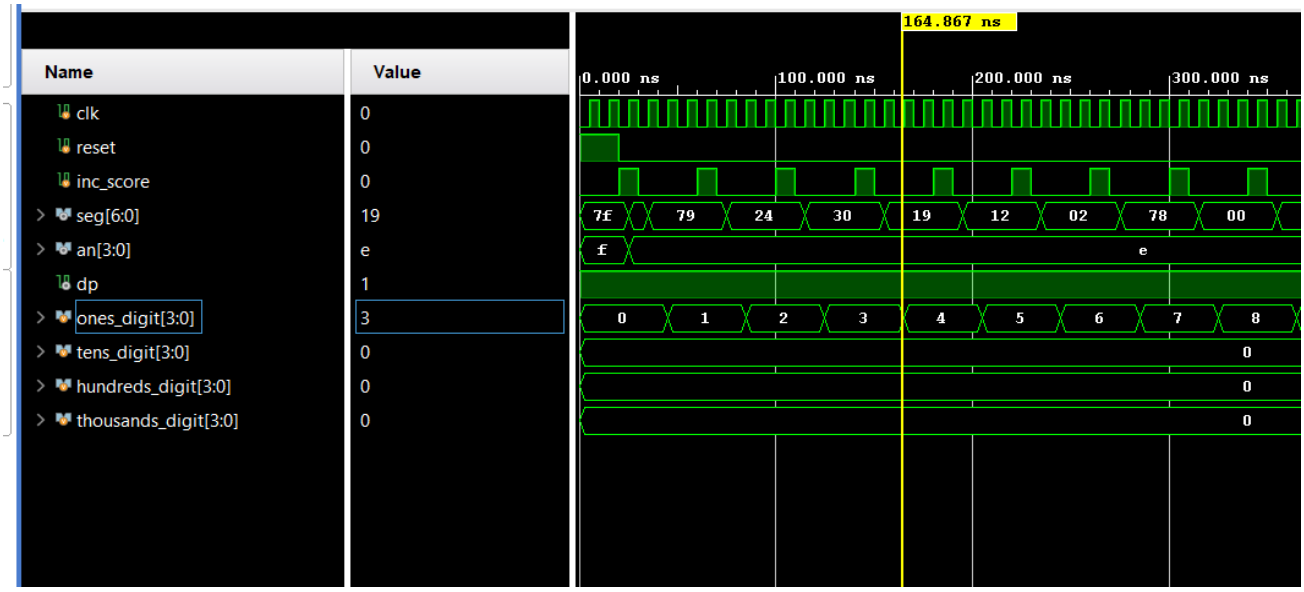


3. tb_ScoreCounter:

This testbench verifies the core functionality of the ScoreCounter module by applying a series of inc_score pulses and observing the resulting output on the 7-segment display. The segment output (seg) is decoded into a numerical digit using a common cathode mapping function, and only the active digit (an = 4'b1110, representing the ones place) is sampled.

The test confirms that the counter correctly resets, increments, and displays the expected values. By focusing on the least significant digit, we ensure that the basic counting and display logic function correctly without needing to track carry propagation across multiple digits.

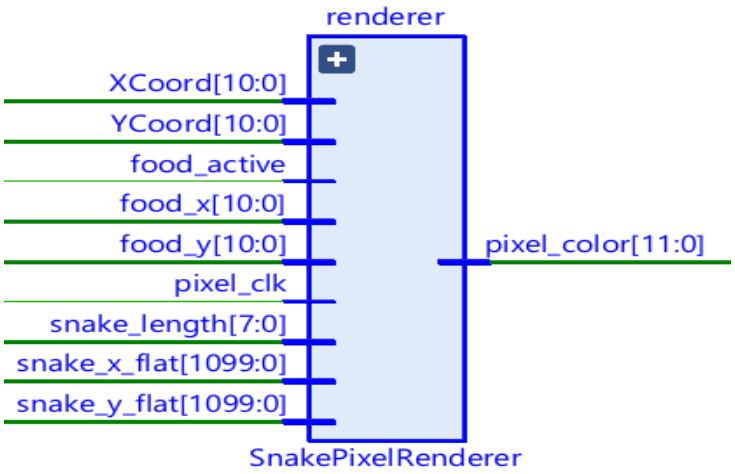
This provides a simple yet effective way to validate that the scoring mechanism responds accurately to scoring events in the game.



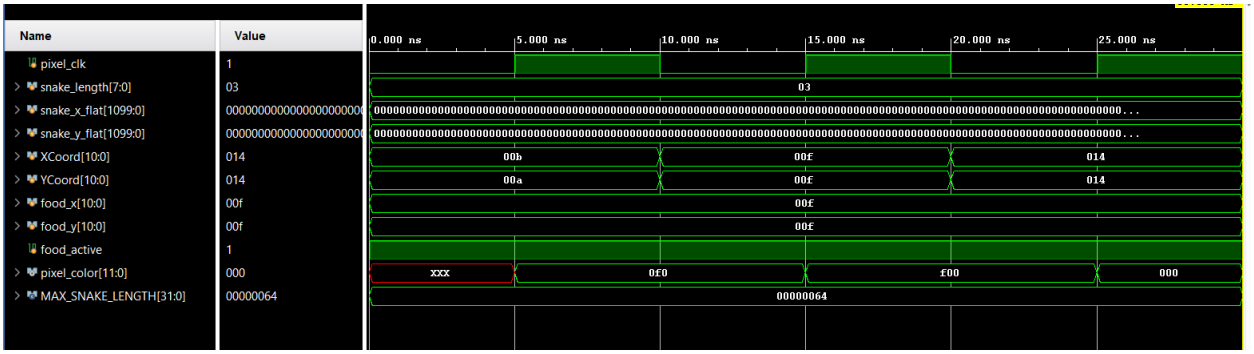
4.SnakePixelRenderer

The SnakePixelRenderer module is responsible for determining the color of each pixel on the VGA screen based on the current game state. It compares the active pixel coordinates (XCoord, YCoord) against the positions of the snake body and the food. The snake's coordinates are stored in flattened vectors (snake_x_flat, snake_y_flat) and unpacked dynamically using a loop that checks each segment up to the current snake_length. If a match is found, the pixel is colored green (12'h0F0). If the current pixel matches the food location and the food is active, it is colored red (12'hF00). All other pixels default to black (12'h000). This module runs on the pixel clock and updates the pixel_color output in real time, enabling accurate visual rendering of the snake and food on screen.

Inputs and Outputs:
The module receives the current pixel clock (pixel_clk), snake length, snake body coordinates, food position, and screen coordinates. Its single output is pixel_color, a 12-bit RGB value that determines the color of the current pixel. The module's logic ensures that each VGA pixel is correctly colored according to the presence of the snake or food at that location.



5. tp_ SnakePixelRenderer



This testbench validates the pixel rendering logic of the SnakePixelRenderer module by simulating different screen coordinates and comparing them against known snake and food positions. The snake is initialized with three segments at positions (10,10), (11,10), and (12,10), and the food is placed at (15,15). The test then checks three scenarios: when the current pixel matches a snake segment, matches the food, and when it is empty. For each case, the pixel_color output is monitored to ensure it reflects the correct color: green for snake, red for food, and black for background. This confirms that the module accurately determines pixel color based on game state.

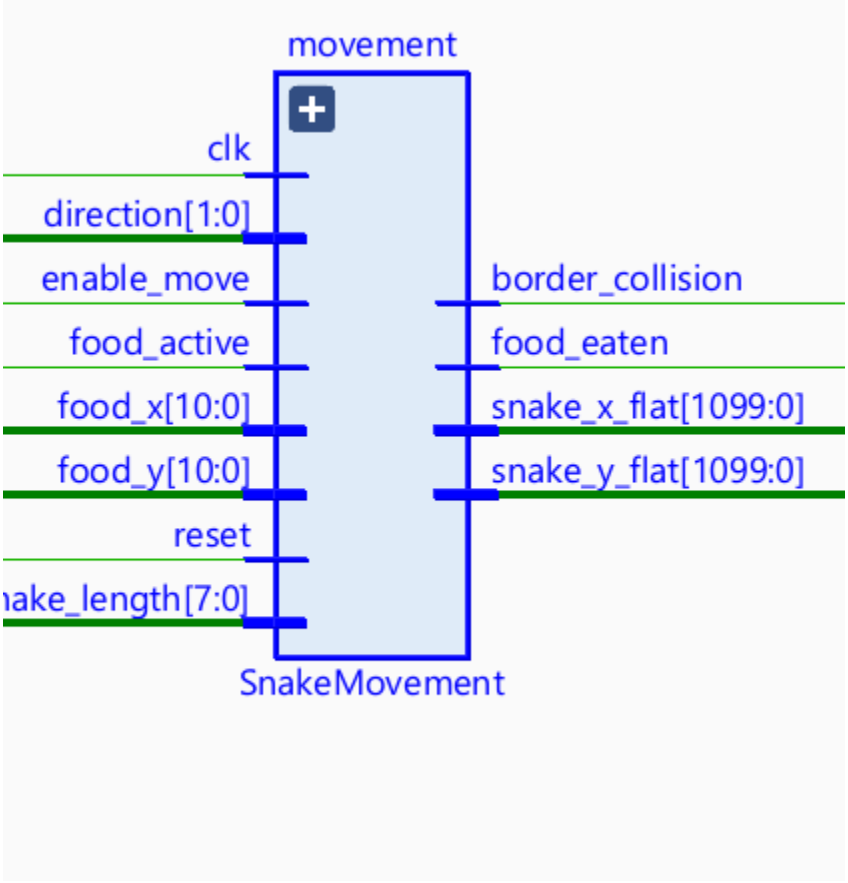
6. Snake_Movement

The SnakeMovement module controls the snake's position update logic during gameplay. At every positive clock edge and when enable_move is active, it updates the snake's body based on the current direction, checks for border collisions, and detects whether the snake's head has reached an apple.

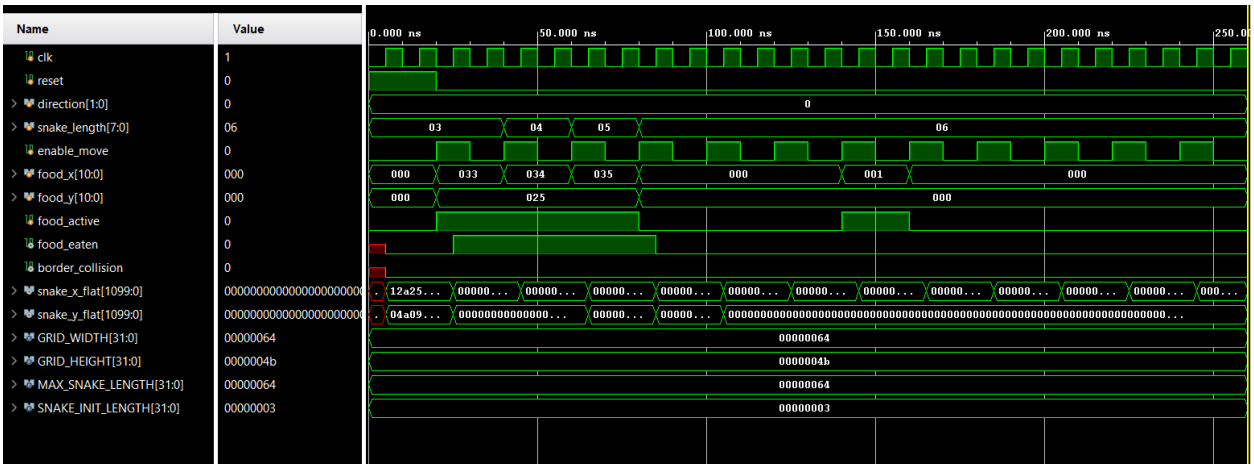
The module stores the snake's body positions in flattened vectors (snake_x_flat, snake_y_flat), with the head always at index 0. On movement, the snake's body is shifted forward, and a new head position is calculated. If this position matches the current food coordinates (food_x, food_y) and the food is active, the food_eaten signal is asserted. The module also raises the border_collision signal if the head goes out of bounds, as defined by GRID_WIDTH and GRID_HEIGHT.

Inputs and Outputs:

- **Inputs:** Clock (clk), reset (reset), movement direction (direction), current snake length (snake_length), move enable signal (enable_move), food coordinates (food_x, food_y), and a food_active flag.
- **Outputs:** snake_x_flat and snake_y_flat, food_eaten, and border_collision (high if the snake hits the screen boundary).



7. tp_Snake_Movement



This testbench evaluates the behavior of the SnakeMovement module by simulating directional movement, food detection, and snake growth. The test initializes the snake in the center of the grid and repeatedly moves it to the right. At specific positions, food is placed in the snake's path to trigger the food_eaten signal. When food is eaten, the snake_length is incremented to simulate the growth logic handled by external game control. Throughout the simulation, signals such as snake_x_flat, food_eaten, and border_collision are observed to verify correct snake behavior and boundary handling. This test effectively demonstrates the module's ability to update position, grow on food collision, and detect wall collisions.

8. SnakeGameLogic:

The SnakeGameLogic module acts as the main game controller for the Snake game, handling direction input, movement timing, food spawning, growth, and collision detection. It integrates the SnakeMovement module to update the snake's position and maintains internal logic to manage when the snake should grow, when to spawn food, and when the game is over.

Functionality:

The module listens to keyboard scan codes (scancode) to update the direction of movement, ensuring the snake cannot reverse direction directly. A movement pulse (enable_move_pulse) is generated based on a timing counter to control the snake's pace. When a move occurs, the module checks for food collision, border collision, and self-collision. If the snake eats food, it activates a grow flag and disables the food until a new random position is generated. If a collision is detected, the game ends by asserting game_over.

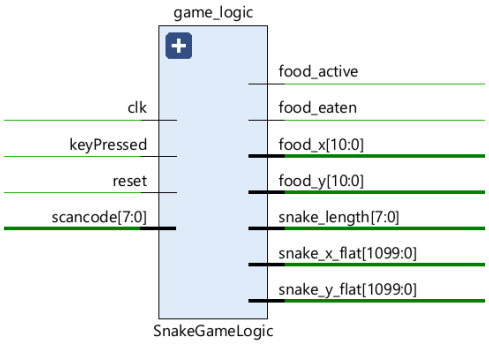
Inputs and Outputs:

- **Inputs:**
  - clk, reset: Clock and synchronous reset

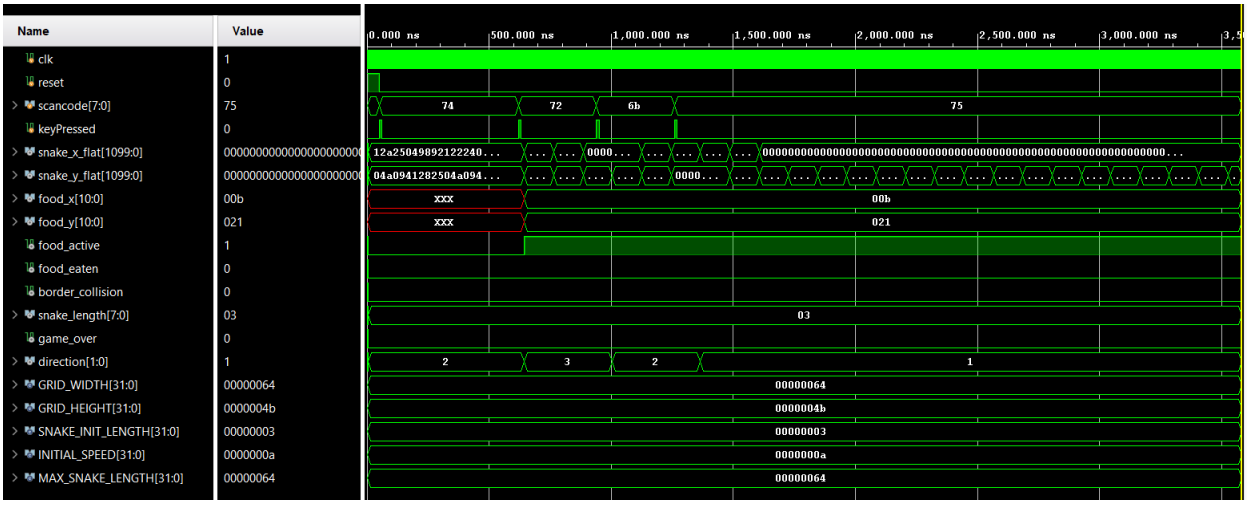
- scancode, keyPressed: Keyboard input signals

- **Outputs:**
  - snake_x_flat, snake_y_flat: Flattened coordinates of the snake body
  - food_x, food_y, food_active: Current food state and position
  - food_eaten, border_collision: Status signals from SnakeMovement
  - snake_length: Current length of the snake
  - game_over: Indicates if the game has ended
  - direction: Current movement direction

The module ensures smooth game progression by managing movement intervals, directional changes, and game state updates like growth and collisions, forming the heart of the Snake game's logic.



9. tb_SnakeGameLogic:



This testbench verifies the core gameplay logic of the Snake game. It simulates user input through keyboard scancodes to control the snake's direction and monitors

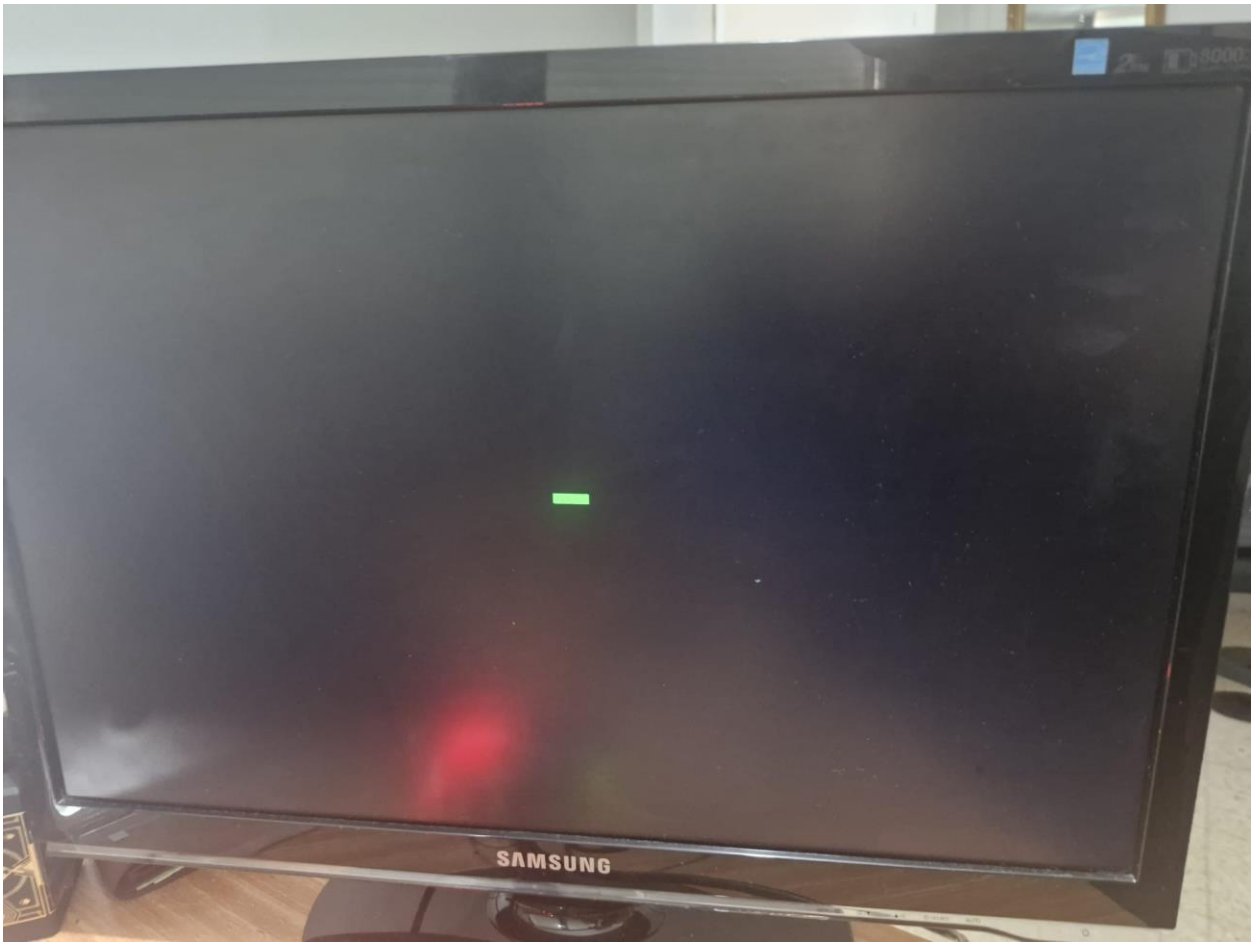
responses such as position updates, food spawning, snake growth, and game-over conditions.
The test begins by applying a reset, then simulates a sequence of directional keypresses (Right, Down, Left, Up) using press_key. The movement timing is accelerated by setting a low INITIAL_SPEED parameter to allow for quick simulation. The testbench also observes the food_eaten and snake_length signals to confirm that the snake grows correctly after eating and checks game_over to verify proper collision detection.
Inputs and Outputs Used:

- Inputs: clk, reset, scancode, keyPressed
- Outputs: snake_x_flat, snake_y_flat, food_x, food_y, food_active, food_eaten, border_collision, snake_length, game_over, direction

The test provides effective functional coverage of input response, directional changes, growth logic, and collision handling, making it suitable for validating the integration of game control and movement modules.

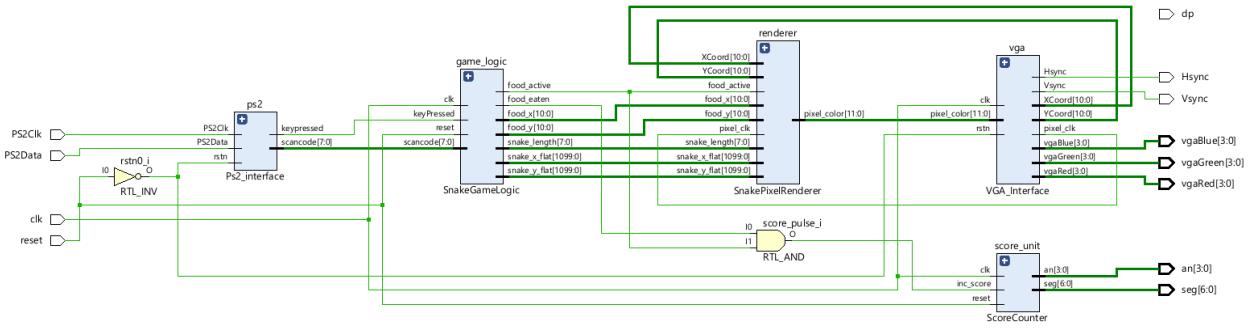
10. Snake_game:

The SnakeGameTop module serves as the top-level wrapper for the Snake game, integrating all subsystems including VGA display, keyboard input, game logic, rendering, and scoring. It takes a 100 MHz clk, active-high reset, and PS/2 keyboard inputs (PS2Clk, PS2Data). Its outputs drive the VGA interface (vgaRed, vgaGreen, vgaBlue, Hsync, Vsync) and a 4-digit 7-segment display (seg, an, dp) to show the score.
The core gameplay logic is handled by the SnakeGameLogic module, which processes direction input via decoded keyboard scancode, manages snake movement, food spawning, colli

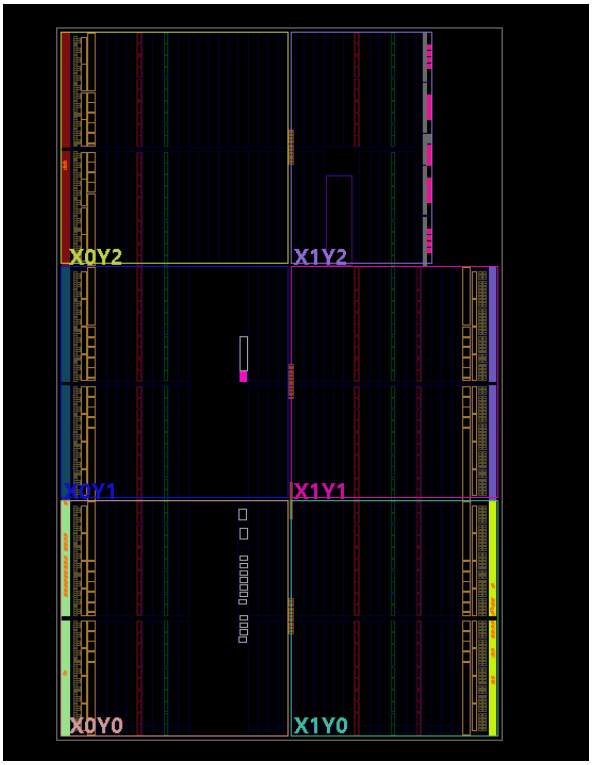


sion detection, and score updates. The snake's pixel positions and the food coordinates are passed to the SnakePixelRenderer module, which determines the correct RGB value for each pixel. The resulting color is forwarded to the VGA_Interface, which generates proper VGA timing and downsampled pixel coordinates for a 100x75 grid. Additionally, whenever food is eaten, a score_pulse is triggered, incrementing the score shown on the 7-segment display via the ScoreCounter module.

11. Elaborated design:



12.Synthesis



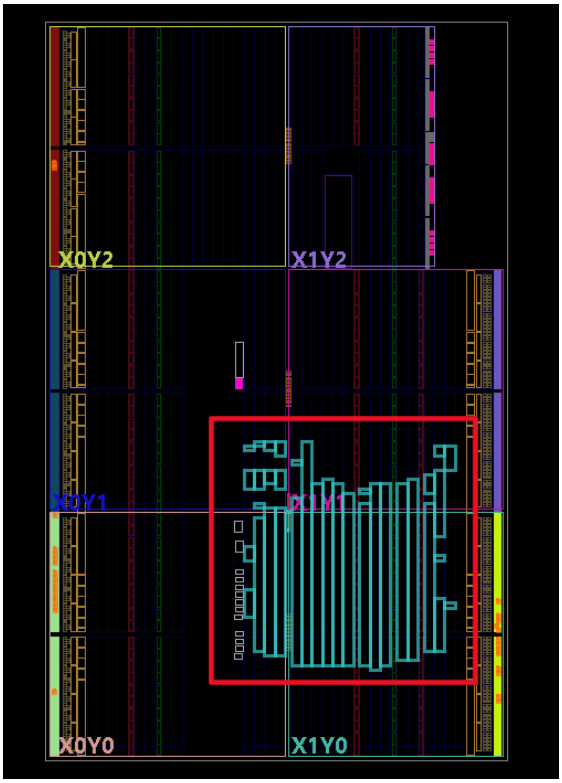
13.Setting up constraints:

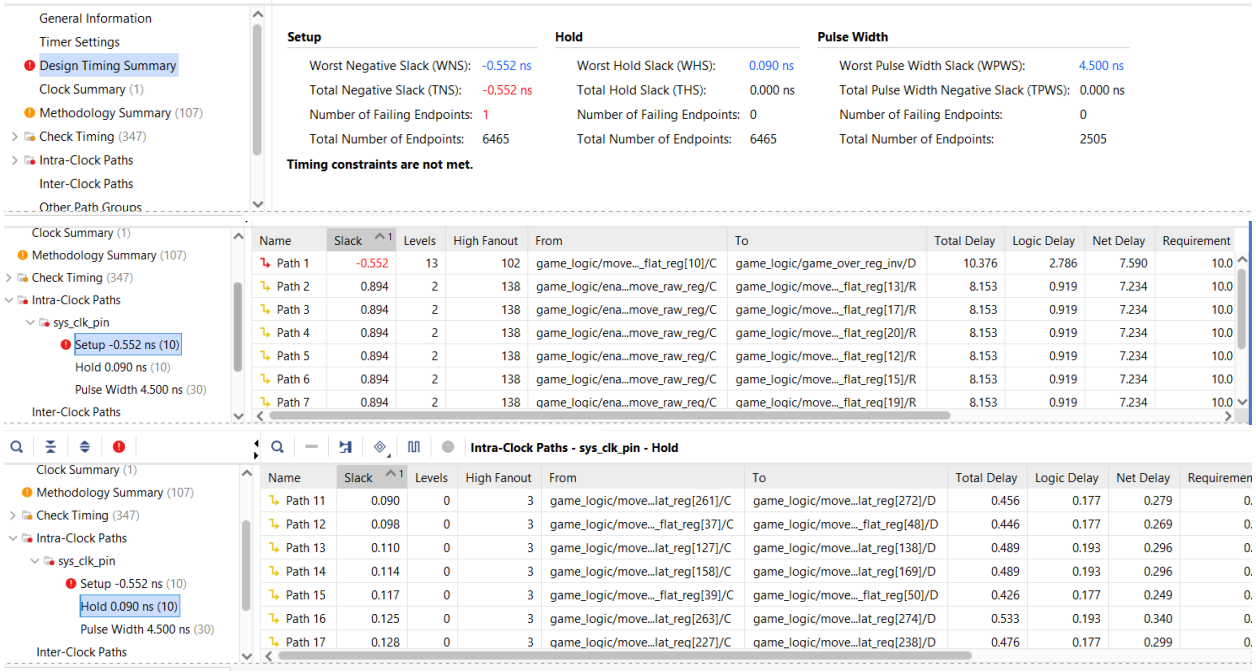
```
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
set_property CFGBVS VCCO [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]


set_property PACKAGE_PIN C17 [get_ports PS2Clk]
set_property IOSTANDARD LVCMOS33 [get_ports PS2Clk]
set_property PULLUP true [get_ports PS2Clk]
set_property PACKAGE_PIN B17 [get_ports PS2Data]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports PS2Data]
set_property PULLUP true [get_ports PS2Data]



set_property PACKAGE_PIN G19 [get_ports {vgaRed[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[0]}]
set_property PACKAGE_PIN H19 [get_ports {vgaRed[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[1]}]
set_property PACKAGE_PIN J19 [get_ports {vgaRed[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[2]}]
set_property PACKAGE_PIN N19 [get_ports {vgaRed[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[3]}]

set_property PACKAGE_PIN N18 [get_ports {vgaBlue[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[0]}]
set_property PACKAGE_PIN L18 [get_ports {vgaBlue[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[1]}]
set_property PACKAGE_PIN K18 [get_ports {vgaBlue[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[2]}]
set_property PACKAGE_PIN J18 [get_ports {vgaBlue[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[3]}]

set_property PACKAGE_PIN J17 [get_ports {vgaGreen[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[0]}]
set_property PACKAGE_PIN H17 [get_ports {vgaGreen[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[1]}]
set_property PACKAGE_PIN G17 [get_ports {vgaGreen[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[2]}]
set_property PACKAGE_PIN D17 [get_ports {vgaGreen[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[3]}]

set_property PACKAGE_PIN P19 [get_ports Hsync]
set_property IOSTANDARD LVCMOS33 [get_ports Hsync]
set_property PACKAGE_PIN R19 [get_ports Vsync]
set_property IOSTANDARD LVCMOS33 [get_ports Vsync]



# 7-segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]

set_property PACKAGE_PIN V7 [get_ports dp]
set_property IOSTANDARD LVCMOS33 [get_ports dp]

set_property PACKAGE_PIN U2 [get_ports {an[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]


#Buttons
set_property PACKAGE_PIN U18 [get_ports reset]
set_property IOSTANDARD LVCMOS33 [get_ports reset]




set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]
```

14.Implementing the design:





15.FPGA programming and debugging: