Habib University

**EE/CS 172L/130L-T7**

Digital Logic and Design

Fall 2020

# Project Report

Batool Ahmed (ba06180)

Adnan Asif (aa06204)

Haania Siddiqui (hs06188)

Aliza Rafique (ar05986)

## TABLE OF CONTENTS

# Chapter 1

## ABSTRACT

This project aims to implement the ”Snake” game on Basys 3 Artix-7. This is a two player game in which each player has an objective is to eat five food points randomly displayed at the screen without colliding with other objects on the screen. The game will continue till either of the the snake collides with itself or the walls. Aside from FPGA itself, we will be using a monitor connected via VGA port for the display and a keyboard to control the snake.

This report will highlight the progress steps of the project, starting from the explanation of the design of certain feature and its finite state machine design.

Through this project, we were able to learn software based implementations and apply it to hardware. This learning experience, gave us an insight on hardware based projects and their applications in the current era.

# Chapter 2

# TECHNICAL DESCRIPTION

## 2.1 System Overview

Our game will be using a memory segment. It will consider the memory addresses stored in the memory segment to output the new state, i.e. it will compare the previous state of the snake with the new state and will generate output accordingly. The most important functions of our game are the movement of a snake, generation of food, and collision (with itself or food). For the display of the game and its interaction with the user, a monitor is used that is connected via VGA to the FPGA board. For the user control and input, we used a PS/2 keyboard.

### 2.1.1 Components Used

1. Digilent Basys 3 Artix-7 FPGA Board:

   ready-to-use digital circuit development platform based on the latest Artix-7™ Field Programmable Gate Array (FPGA) from Xilinx. With FPGA , many designs can be completed without any additional hardware since it has switches, LEDs and other I/O devices. The designs can be expanded through FPGA I/O pins using cutsom boards and circuits. It can hosts designs from introductory combinational circuits to complex sequential circuits. The Artix-7 FPGA has more capacity and performs better than previous designs. Basys 3 is resourceful in terms of ports and peripherals.

2. Xilinx Vivado

3. VGA Monitor and cable

4. PS/2 Keyboard:

   *PS/2 Keyboard is discussed in detail in input block*

### 2.1.2 Features of the Game

In our snake game, the snakes can move up, down, left, and right and can also move through the walls.

1. Players:

   Our Snake Game comprises of two players which are snakes.

2. Collision

   (a) Food Collision and Increase in score:

   If a snake collides with food, i.e. coordinates of his head become equal to coordinates of the food, then the collision is detected. With this collision, the score increases by one point.

   (b) Collision with another Snake:

   Another collision that might occur is the collision of a snake with the the snake.

3. Food Generation:

   The food is generated at random places one at a time. The snakes have to collect the food that is generated randomly.Only one apple is generated at a time. The snakes have to eat it in order to get a point.As soon as any one of the snake collides and eats apple , another apple is generated at a random place. Whoever collects the five apples first, wins.

4. Score:

   The progress of both the players/snakes is displayed on screen through scores. When a snakes score 5 points, i.e. when it eats five apples, it wins. The score of player 1 is displayed on the top left corner of screen and score of player 2 is displayed on the top right corner of the screen. The colors of snake 1 and snake 2 are cyan and red respectively.

5. Winning: On victory of any of the snake, a winning screen is displayed with the name of respective player.

(a) Score:

If a snake scores five points before the other snake, it wins.

(b) Collision:

If a snake collides with another snake , the latter wins and the former loses.

6. Reset:

Reset is an option that would restart the game. A player can press enter in order to reset the game.

## 2.2 Input Block

After powering on, the Artix 7 FPGA needs to be configured. Files with an extension of .bit and known as bitstreams, stores the configured data. We used the Vivado software to create bitstreams from Verilog source files. For the movement of snake 1 in our game, we are using the arrow keys-up,down, left and right- of PS/2 keyboard to give the input. Data is received from the PS/2 style keyboard in the form of scan codes when a key is pressed. Each key has its scan code. Since we are using four arrow keys -up, down,left and right- hence we used scan codes of the respective keys to make our snake move. Following are the scan codes for given alphabets in hexadecimal:

UP :E075

DOWN :E072

LEFT : E06B

RIGHT : E074

Our snake moves after a key has been released. The keyboard has its clock and sends data on the negative edge of its clock. Keyboard module's output is of 5 bit, which gives the direction for the snake to move.

For the 5 bit output of our keyboard module, we used the following LED ports:

U16

E19

V19

U19

W18

An important point that had been considered was that the snake could not move left if he is moving right and vice versa. Similarly, if a player is moving down, he cannot move up and vice versa.

The USB HID host capability in Basys 3 is provided by Auxiliary Function microcontroller (Microchip PIC24FJ128) which switches to the USB host(application mode) after the FPGA has programmed. The keyboard is attached to a USB connector at J2 and is driven by the firmware in the microcontroller. The ports of Artix 7 that we used to drive the input signals, ”clk” and ”data”, are C17 and B17. Following is the representation of the connections.

For player 2 , the inputs are being given through push buttons of FPGA board. They are momentary switches and at rest gives a low output.They only give a high output when they are pressed.The FPGA board has 5 push buttons but we are using only four for the movement of our second snake. Their configuration is as follows:

For moving up,

BTNU-T18

For moving down,

BTND-U17

For moving left,

BTNL-W19

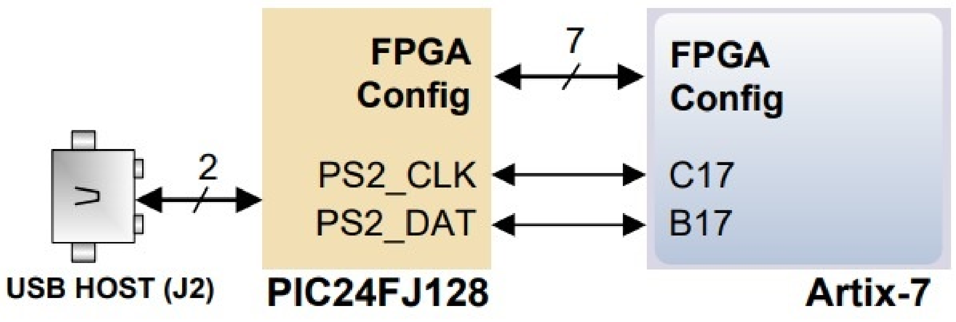


Figure 2.1: BASYS3 PIC24 Connections

For moving right,

BTNU-T17

## 2.3 Output Block

The output is displayed on a VGA screen. The output contains 5 signals: $h_{sync}$, $v_{sync}$, red, green, and blue. When the signal is received from clk, a block converts it to $clk_d$ and passes it to another block containing h$_{counter}$. h$_{counter}$ gets incremented at every positive edge of the clock signal(clk) and counts up to 799, at this value, the trig$_v$ signal is set to 1. This block outputs h$_{count}$ and v$_t rig$. v$_t rig$, along with clk$_d$, is passed to another block containing v$_{counter}$. v$_{counter}$ gets incremented at every positive edge of the clock signal (clk) and counts up to 524, only if the enable$_v$ signal is 1.

The output v$_{count}$ with h$_{count}$ is passed to vga$_{sync}$. This block generates a 5 output signal:h$_s ync$, v$_s ync$, x$_{loc}$, y$_{loc}$, and video$_{on}$. In which three signals: video$_{on}$, x$_{loc}$, and y$_{loc}$ are used for pixel generation in which red, green, and blue signals are generated at the end. They are discussed below:

### 2.3.1 h$_{sync}$ and v$_{sync}$

These signals are connected to the VGA port and control the horizontal and vertical scans of the monitor. A period of h$_{sync}$ signal contains 800 pixels in which 640 pixels are displayed on the screen, 96 pixels in the electron beam region at the left edge of the screen, 16 pixels for the right border, and 48 pixels for the left border. h$_{sync}$ signal is lowest when the h$_{counter}$'s output is between 656 and 751.

A period of the v$_{sync}$ signal is 525 lines in which 480 lines are used for the display, 2 lines for the electron beam region, 10 lines for the bottom border, and 33 lines for the top border. v$_{sync}$ signal is the lowest when the line count is between 490 and 491.

### 2.3.2 x$_{loc}$, y$_{loc}$, and video$_{on}$

The video$_{on}$ signal indicates whether the current targeted pixel is in the displayable region. The x$_{loc}$ and y$_{loc}$ signals specify the location of the current pixel. These signals are used to generate three each 4-bit color signals: red, green, and blue.

## 2.4 User Flow

### 2.4.1 User Diagram

Following is the user diagram for the game:

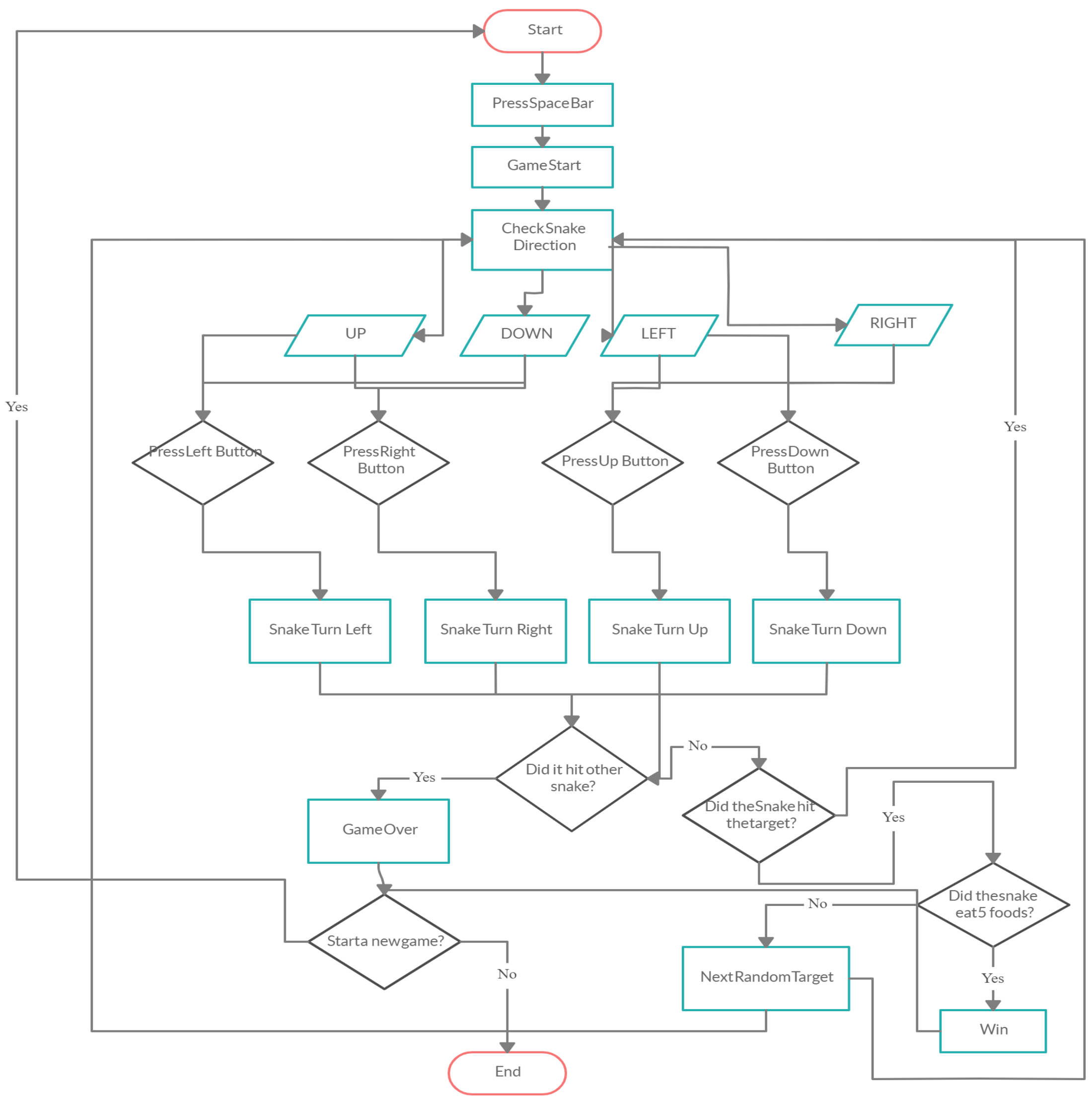


Figure 2.2: User Diagram

### 2.4.2 Explanation

The user diagram indicates the possibilities for a user in this game. Each user has four controls: left, right, up and down. The user can move in either of the direction to reach the food which increases the score of the user. If the user hits another snake then the user loses. The user can win the only if it eats five foods which are randomly generated in the game. After the game is over, the players can start a new game.

# Chapter 3

# IMPLEMENTATION DETAILS AND RESULTS

## 3.1 Movement

### 3.1.1 Finite State Machine

The following are description of snake transition from every current state:

1. INITIAL:

   In the beginning of the game , both the snakes are in initial state. If up key or up push button is pressed , the state of respective snake changes to up. If down key or down push button is pressed , the state of respective snake changes to down.If right key or right push button is pressed , the state of respective snake changes to right. If down key or down push button is pressed , the state of respective snake changes to down.

2. UP:

   From the up state, the snake can transition to LEFT and RIGHT states only. If current state of snake is UP , the next state can not be DOWN i.e. it can not move down if it's facing up.

3. DOWN:

   Again, if the current state of snake is DOWN, the snake can transition to LEFT and RIGHT states only. The next state can not be UP i.e. it can not move up if it's facing downwards

4. LEFT:

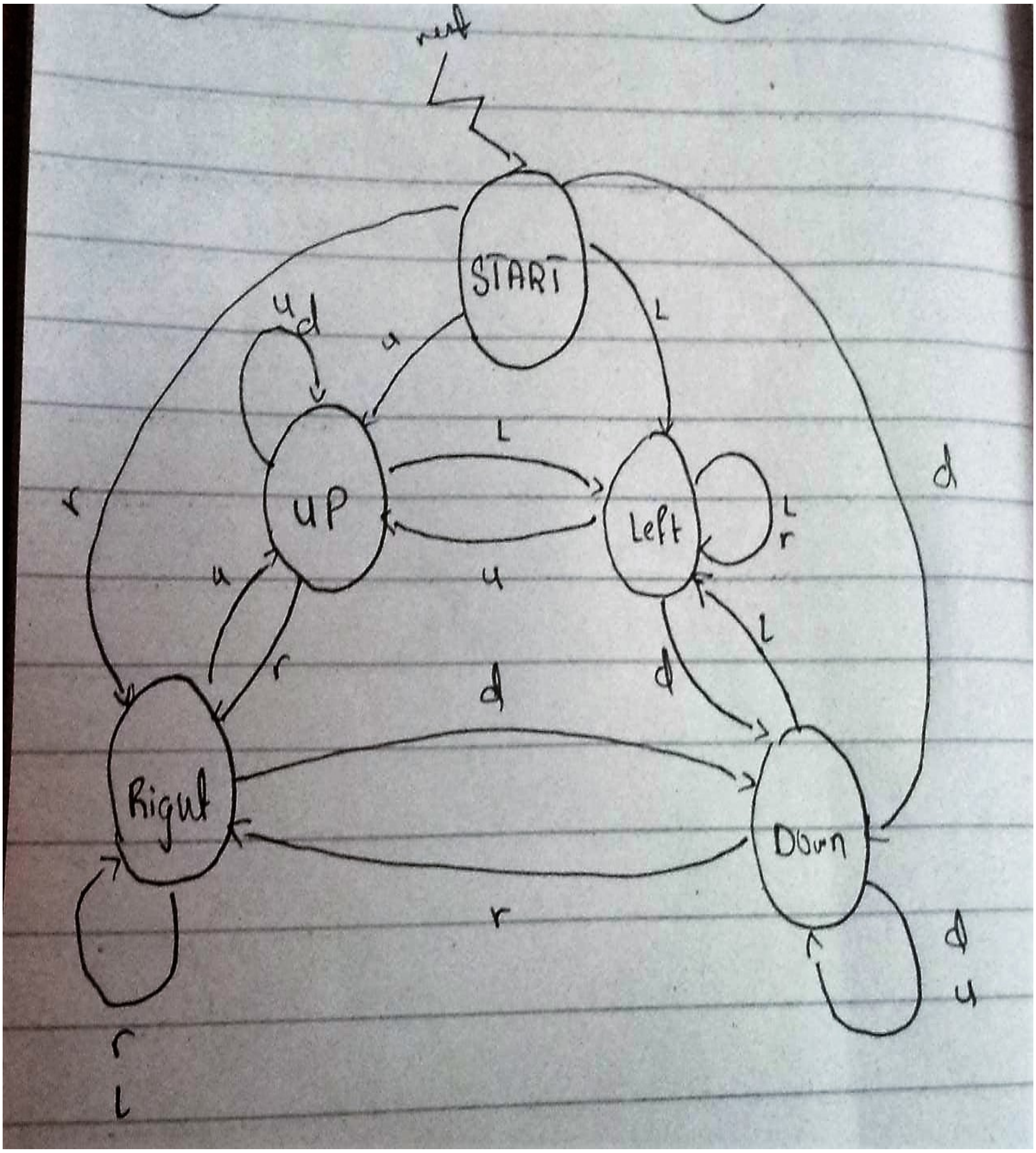
   If the current state is LEFT, the snake can transition to UP and DOWN states only.The next state can not be RIGHT i.e. it can move to right if it's facing left.

5. RIGHT:

   Again, if the current state of snake is RIGHT, the snake can transition to UP and DOWN states only. The next state can not be RIGHT i.e. it can not move to left if it's facing right.

The state of the FSM is directed towards the main module and then according to the current state, the movements of the head of the snake is implemented. Consequentially, wherever the head of the snake moves, the body of the snake follows those coordinates as well. All this is occurring on another clock which is of 25Hz.

## 3.1.2 State Transition Diagram



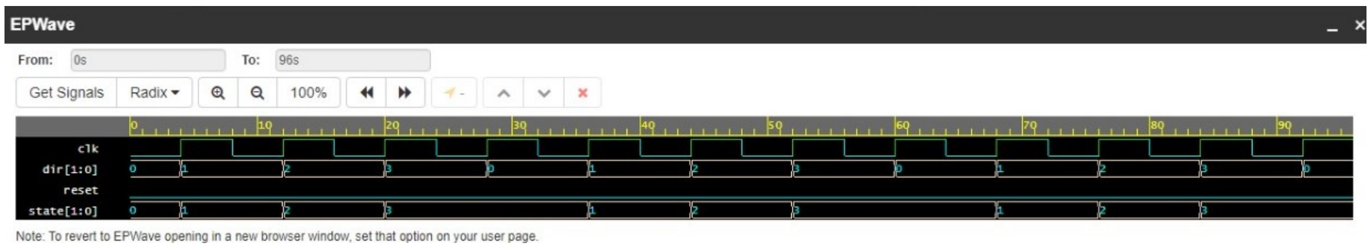
### 3.1.3 Code

```
    module move(clk, dir, state);
  input clk;
  //reset;
  input [4:0] dir;
  output [4:0] state;
  reg [4:0] state;
  reg [4:0] next;
  parameter UP = 5'b00010;
  parameter LEFT = 5'b00100;
  parameter RIGHT = 5'b10000;
  parameter DOWN = 5'b01000;
  parameter START = 5'b0;
  parameter RESET = 5'b11111;
always @(dir)
    begin
      case (state)
        UP:
          begin
            if (dir == 5'b00010)
            next = UP;
            else if (dir == 5'b00100)
              next = LEFT;
            else if (dir == 5'b10000)
              next = RIGHT;
            else if (dir == 5'b01000)
              next = UP; //Since it can't go down when it's up
            else if (dir == 5'b11111)
                next = RESET;
          end
        RIGHT:
          begin
            if (dir == 5’b00010)
            next = UP;
            else if (dir == 5’b00100)
              next = RIGHT; //Since it can’t go left when it’s right
            else if (dir == 5’b10000)
              next = RIGHT;
            else if (dir == 5’b01000)
              next = DOWN;
            else if (dir == 5’b11111)
                next = RESET;
          end
        LEFT:
          begin
            if (dir == 5’b00010)
            next = UP;
            else if (dir == 5’b00100)
              next = LEFT;
            else if (dir == 5’b10000)
              next = LEFT; //Since it can’t go right when it’s left
            else if (dir == 5’b01000)
              next = DOWN;
            else if (dir == 5’b11111)
                next = RESET;
          end
        DOWN:
```

```
begin
  if (dir == 5’b00010)
  next = DOWN; //Since it can’t go right when it’s left
  else if (dir == 5’b00100)
    next = LEFT;
  else if (dir == 5’b10000)
    next = RIGHT;
  else if (dir == 5’b01000)
    next = DOWN;
  else if (dir == 5’b11111)
      next = RESET;
end
START:
begin
  if (dir == 5’b00010)
  next = UP;
  else if (dir == 5’b00100)
    next = LEFT;
  else if (dir == 5’b10000)
    next = RIGHT;
  else if (dir == 5’b01000)
    next = DOWN;
  else if (dir == 5’b11111)
      next = RESET;
end
RESET:
begin
  next=START;
end
```

```
        default:
          next = START;
      endcase
      state = next ;
    end
endmodule
```

### 3.1.4 Timing Diagram



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

Figure 3.1: Timing Diagram for Movement

## 3.2 Collision

### 3.2.1 Collision with Food

*Concept*

To keep a track of the state of the food in the game, three boolean flags/registers are being used. Two of the registers correspond to the two snakes and their values switch to one whenever a snake eats the food. The third register is switched to one whenever the food is being eaten by either of the snake and a new food needs to be randomly generated.

New food generation takes place with respect to the position of both of the snakes in order to generate food at random locations.

*Code*

```
food_collide_S1 = (((headx - foodx < 20) & (headx - foodx > 0)) ||
    ((foodx - headx > 0) & (foodx - headx < 20))) & (((heady - foody < 20)
    & (heady - foody > 0)) || ((foody - heady < 20)) & (foody - heady > 0));

food_collide_S2 = (((head2_x - foodx < 20) & (head2_x - foodx > 0)) ||
((foodx -head2_x > 0) & (foodx - head2_x < 20))) & (((head2_y - foody < 20)
&(head2_y - foody > 0)) || ((foody - head2_y < 20)) & (foody - head2_y > 0));
```

### 3.2.2 Collision with Other Snake

*Concept*

If the head of any snake collides with the body of another snake then the snakes goes to lost state of the game. Consequentially, the other player wins the game. This type of collision is being detected by comparing the $x$ and $y$ axis of the head of a snake with all the coordinates of the body of another snake.

*Code*

```
bCollision_1 =  (((((headx - head2_x1  < 20) & (headx - head2_x1  > 0))
    || ((head2_x1  - headx > 0) & (head2_x1  - headx < 20)))
    & (((heady - head2_y1 < 20) & (heady - head2_y1> 0)) ||
    ((head2_y1- heady < 20)) & (head2_y1 - heady > 0))) ||
    ((((headx - head2_x2  < 20) & (headx - head2_x2  > 0)) ||
    ((head2_x2  - headx > 0) & (head2_x2  - headx < 20))) &
    (((heady - head2_y2 < 20) & (heady - head2_y2 > 0)) ||
    ((head2_y2- heady < 20)) & (head2_y2 - heady > 0))) ||
    ((((headx - head2_x3  < 20) & (headx - head2_x3  > 0)) ||
    ((head2_x3  - headx > 0) & (head2_x3  - headx < 20))) &
    (((heady - head2_y3 < 20) & (heady - head2_y3> 0)) ||
    ((head2_y3- heady < 20)) & (head2_y3 - heady > 0))) ||
    ((((headx - head2_x4  < 20) & (headx - head2_x4  > 0)) ||
    ((head2_x4  - headx > 0) & (head2_x4  - headx < 20))) &
    (((heady - head2_y4 < 20) & (heady - head2_y4> 0)) ||
    ((head2_y4- heady < 20)) & (head2_y4 - heady > 0))));

bCollision_2 = (((((head2_x - headx1 < 20) & (head2_x - headx1 > 0))
    || ((headx1 - head2_x > 0) & (headx1 - head2_x < 20))) &
    (((head2_y - heady1 < 20) & (head2_y - heady1 > 0)) ||
    ((heady1- head2_y < 20)) & (heady1 - head2_y > 0))) ||
    ((((head2_x - headx2 < 20) & (head2_x - headx2 > 0)) ||
    ((headx2 - head2_x > 0) & (headx2 - head2_x < 20))) &
    (((head2_y- heady2 < 20) & (head2_y - heady2 > 0)) ||
    ((heady2- head2_y < 20)) & (heady2 - head2_y > 0))) ||
    ((((head2_x - headx3 < 20) & (head2_x- headx3 > 0)) ||
    ((headx3 - head2_x > 0) & (headx3 - head2_x< 20))) &
    (((head2_y - heady3 < 20) & (head2_y- heady3 > 0)) ||
    ((heady3- head2_y < 20)) & (heady3 - head2_y > 0))) ||
    ((((head2_x - headx4 < 20) & (head2_x- headx4 > 0)) ||
    ((headx4 - head2_x > 0) & (headx4 - head2_x< 20))) &
    (((head2_y - heady4 < 20) & (head2_y- heady4 > 0)) ||
    ((heady4 - head2_y < 20)) & (heady4 - head2_y > 0))));
```

## 3.3 Game FSM

### 3.3.1 Concept

The Game FSM for each snake has the following states: Start, One Point, Two Points, Three Points, Four Points, Game Won, and Game Lost. The game would initially start from the game state and on pressing the reset key at any point in the game would bring it back to the Start state. When snake 1 eats an apple (collides with it), its state would shift to One Point and as it keeps on consuming apples the state keeps traversing until it reaches Four Points. If at Four points it consumes an apple i.e. it reaches the winning condition, its state shifts to Game Won while the other snake's state shifts to Game Lost. If at any point the head of snake 1 collides with the body of snake 2, Snake 1 loses and its state shifts to Game Lost while Snake 2 wins and its state shifts to Game Won. Briefly, the state transitions depend on apple collision, collision with the other snake and reset input. Similar FSM is used for the second snake.

### 3.3.2 Diagram

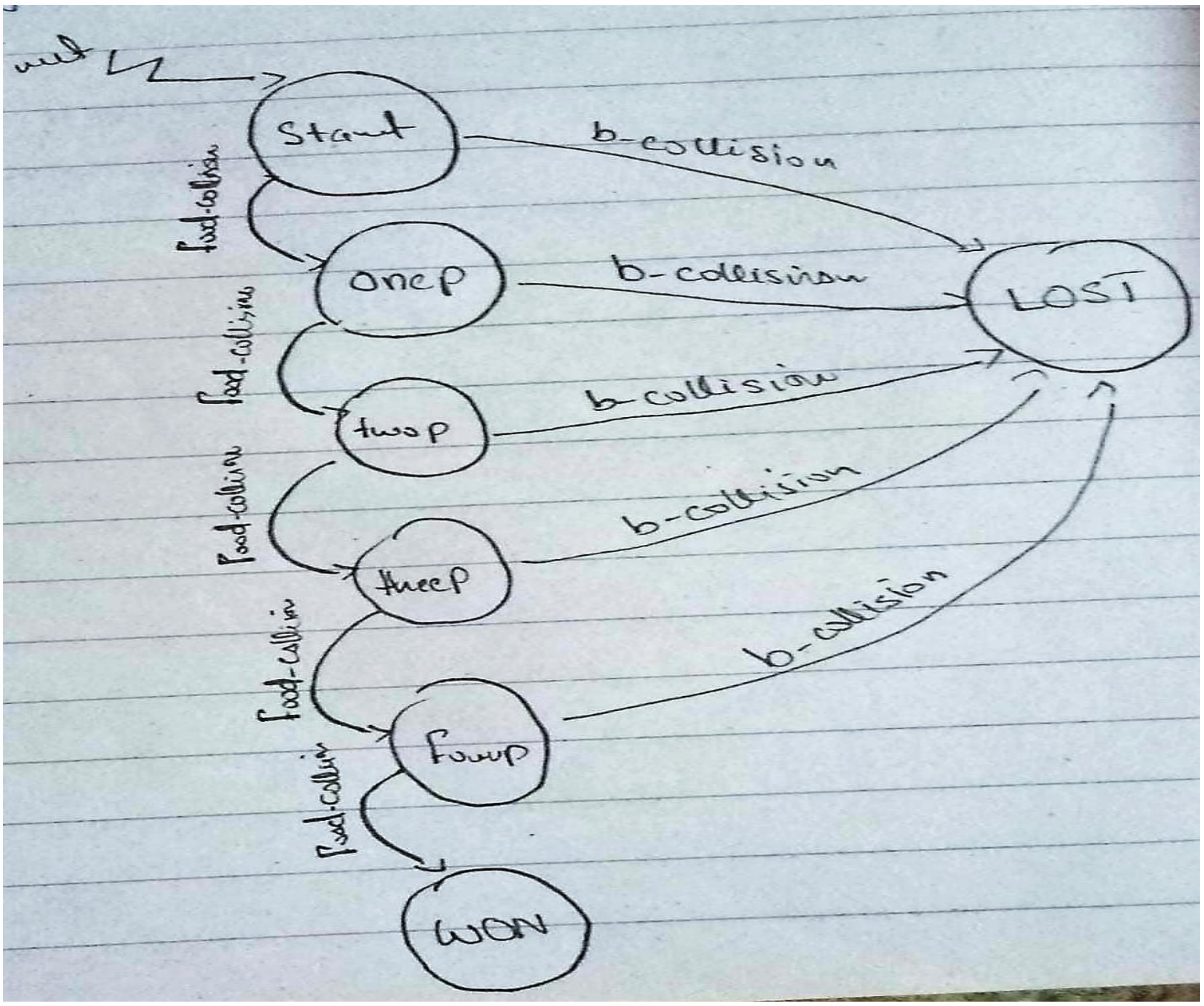


Figure 3.2: Game FSM

### 3.3.3 Code

```
  module Snake_fsm(clk, reset, food_collide_S1 , bCollision_1, gs_S1);
input clk, reset, food_collide_S1 ,bCollision_1;
output [2:0] gs_S1;
reg [2:0] gs_S1;
reg [2:0] ns_S1;
```

```
parameter START = 3’b000;

parameter One_Point = 3’b001;

parameter Two_Points = 3’b010;

parameter Three_Points = 3’b011;

parameter Four_Points = 3’b100;

parameter WON = 3’b101;

parameter LOST = 3’b110;


always @(posedge clk or posedge reset)

begin

    if (reset)

       gs_S1 <= START;


    else gs_S1 <= ns_S1;

 end


always @(posedge food_collide_S1 or  bCollision_1 )

  begin

    case (gs_S1)

  START:

    begin

    if (food_collide_S1)

      ns_S1 = One_Point;

    else if (bCollision_1)

      ns_S1 = LOST;

    else

      ns_S1 = START;
    end
  One_Point:
    begin
    if (food_collide_S1)
      ns_S1 = Two_Points;
    else if (bCollision_1)
      ns_S1 = LOST;
    else
      ns_S1 = One_Point;
    end
  Two_Points:
    begin
    if (food_collide_S1)
      ns_S1 = Three_Points;
    else if (bCollision_1)
      ns_S1 = LOST;
    else
      ns_S1 = Two_Points;
    end
  Three_Points:
    begin
    if (food_collide_S1)
      ns_S1 = Four_Points;
    else if (bCollision_1)
      ns_S1 = LOST;
    else
      ns_S1 = Three_Points;
    end
  Four_Points:
      begin
      if (food_collide_S1)
        ns_S1 = WON;
      else if (bCollision_1)
        ns_S1 = LOST;
      else
        ns_S1 = Four_Points;
      end
    WON:
      begin
        ns_S1 = WON;
      end
    LOST:
      begin
        ns_S1 = LOST;
        end
    default:
      ns_S1 = START;
  endcase
  gs_S1 = ns_S1;


    end

endmodule
```

## 3.4 Block Diagram

### 3.4.1 Explanation

The module receives 3 inputs in total, 2 inputs from the keyboard (keyboard's clock and data) and the FPGA clock. The inputs from the keyboard arereceivedd by the KB$_{input}$ module which handle the data and passes forward the direction according to the key pressed, as previously discussed. This direction is moved forward to a Finite State Machine which handles the movement of the snake, it receives this direction and is synchronized with a divided clock of 25MHz (divided by a clock divider) and moves forward the current state in which the snake is moving. This state is sent to the top module which handles most of the logic and in which resides another FSM to control the general state of the game. Along with the state, the module also receives an hsync and vsync and x and y location to specify the current position of the pixel. It also receives the divided clock and another updated clock divided to 25Hz. All the updates inside the game are synchronized with this clock. Generation of pixel is also handled by this module and it sends forward the values for red, blue, and green along with hsync and vsync to the VGA port.

### 3.4.2 Diagram

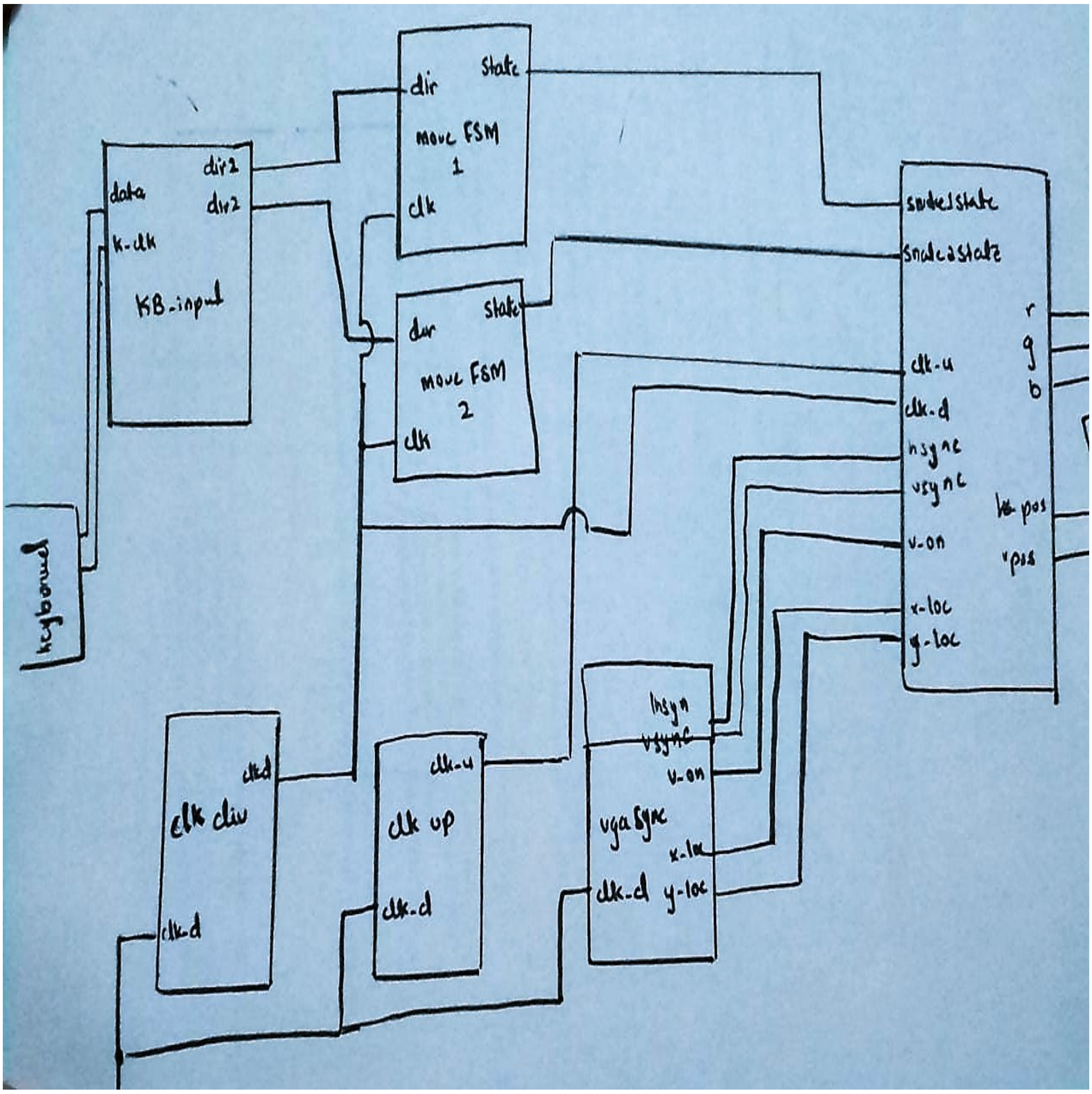


Figure 3.3: Block Diagram

# Chapter 4

# GAME SNIPPETS

## 4.1 Game View

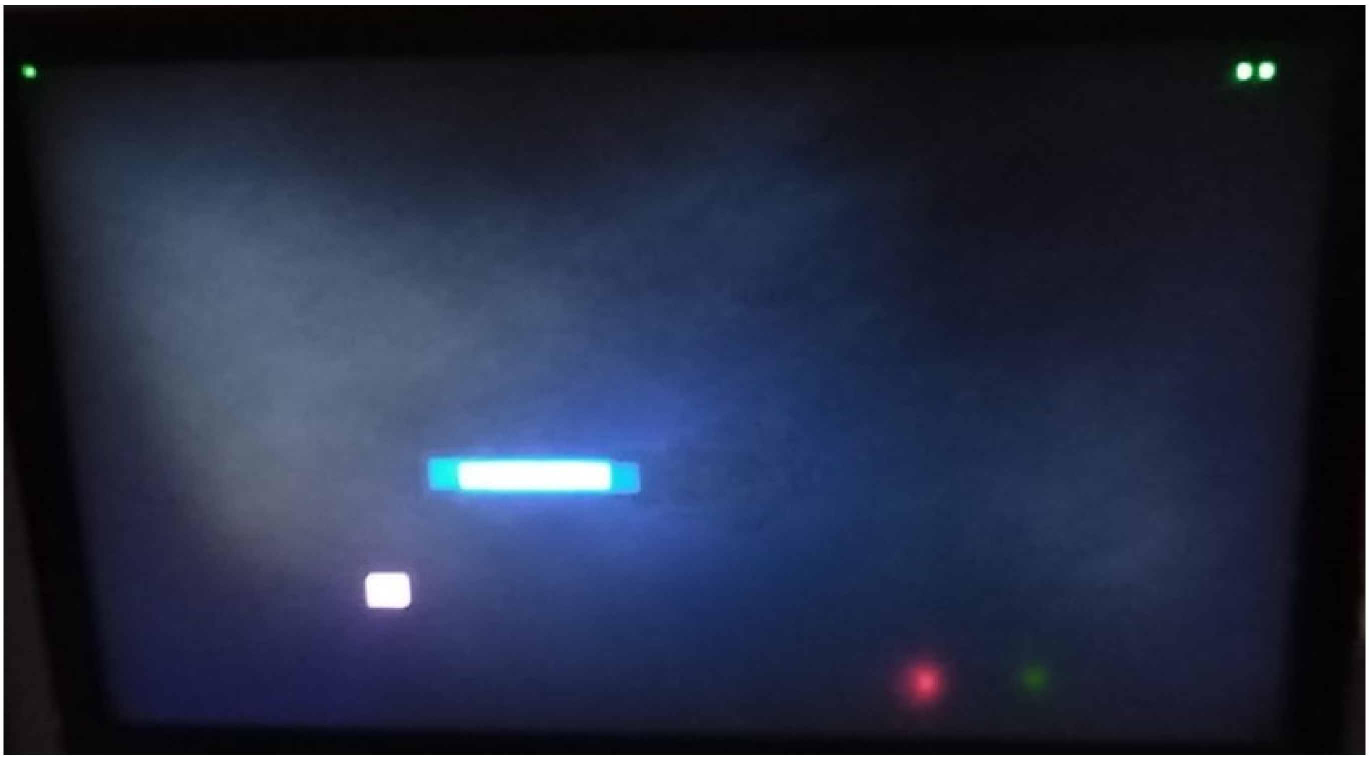


Figure 4.1: Blue Snake on Screen

Figure 4.2: Both Snakes

## 4.2 Score

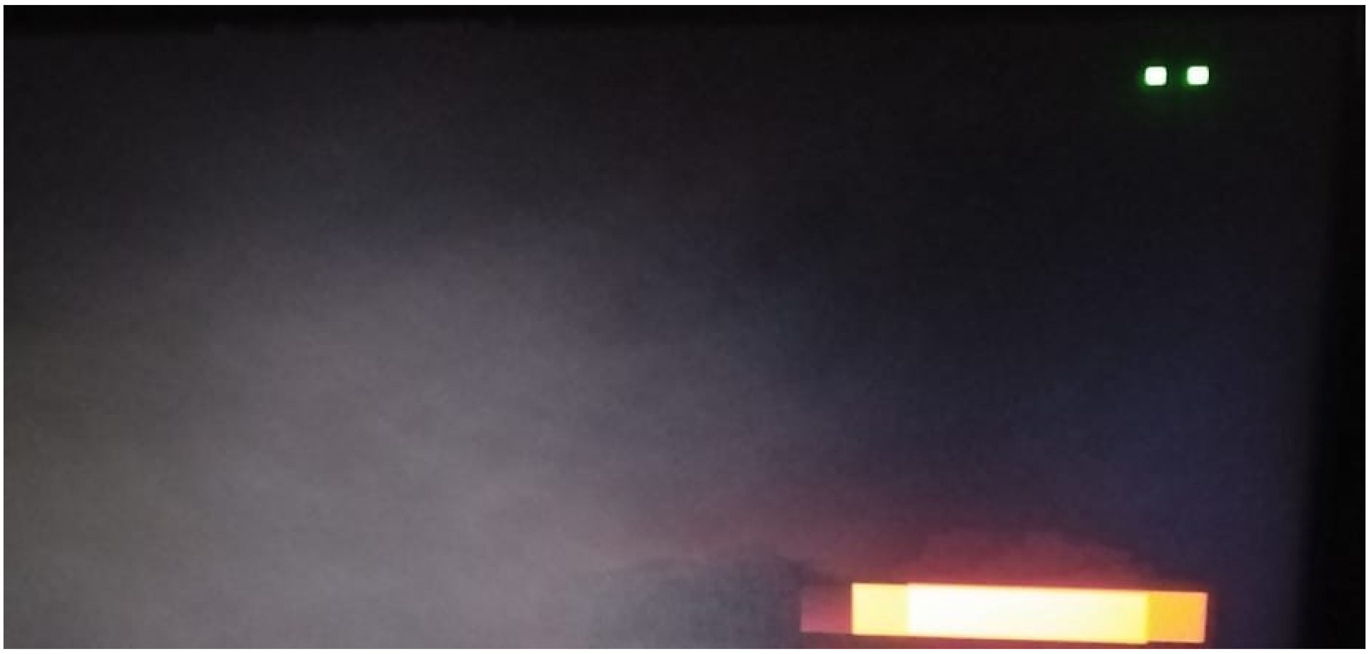


Figure 4.3: Score points

## 4.3 Win screen

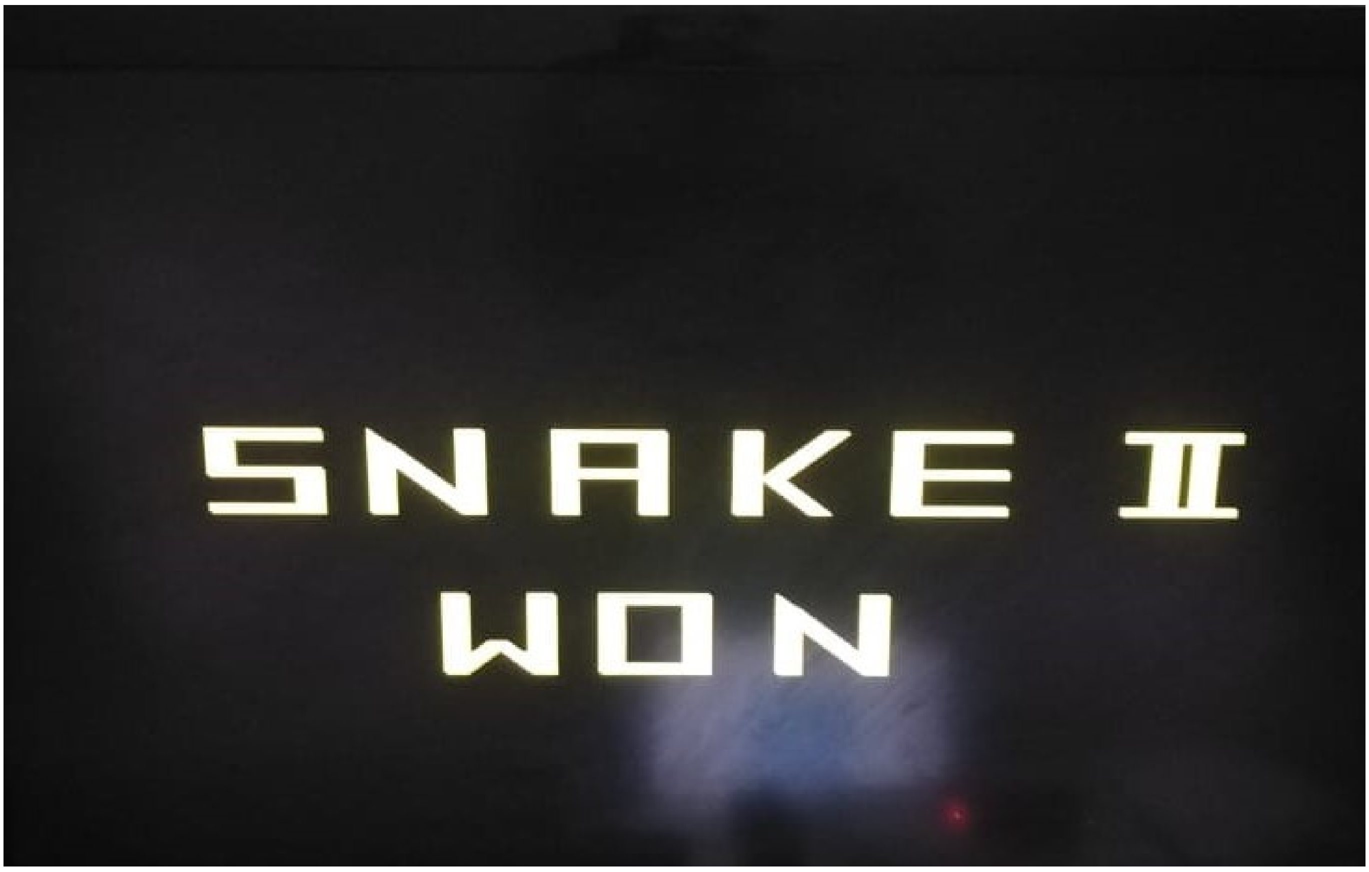


Figure 4.4: Yellow snake winning screen

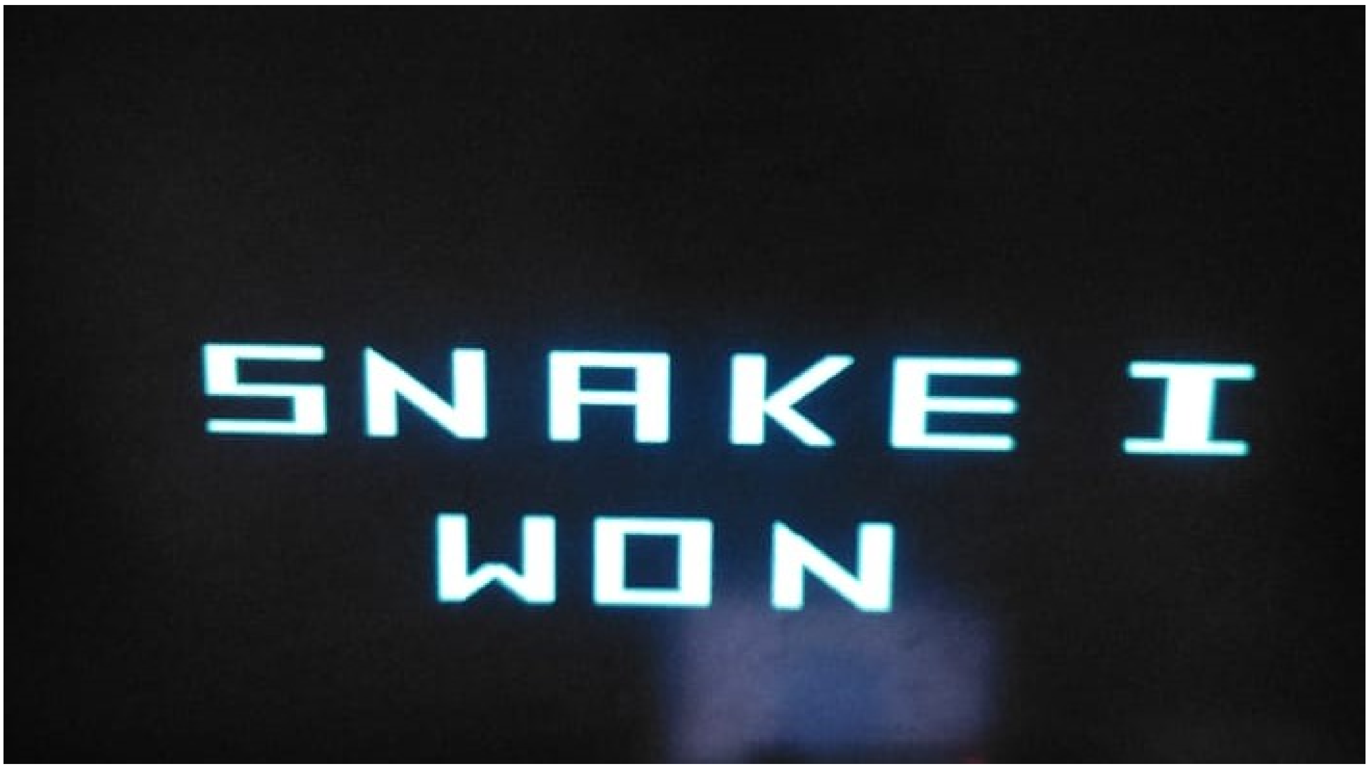


Figure 4.5: Blue snake winning screen