Final Report for Snake Game

Cristopher Marte (cjm2301), Somya Mehta (sm5169),
Rohan Rabbi (mr4280), Saher Iqbal (si2443)

CSEE4840 Embedded System

Spring 2024

INDEX

I. Project Introduction

This project report details the creation of a grid-based snake game that runs on a VGA Monitor. By using a controller for navigation and a VGA monitor for output, the game is designed to offer an immersive experience, showcasing the fusion of traditional and contemporary gaming technologies.

The game is carefully designed to enable smooth updates of game elements including the snake's position, food items, and score tracking while running at 60 frames per second. The methods used in the game's creation, the technological difficulties faced, and the remedies put in place to attain peak performance and player happiness are all covered in detail in this report.

Game Overview:
The main part of the game is the snake which the user is supposed to navigate throughout the screen. While navigating, as the snake eats a fruit, it grows in length. The score is displayed based on the length of the snake. The game ends if the snake hits its own body or hits the wall.

Game Components:

A. **The Snake Sprite:** The user controls this using a controller.
B. **Wall Sprites:** The boundary of the game.
C. **Apple Sprites:** These are the food items the snake is supposed to eat to grow in length.
D. **Score Display:** This score starts from 0 and increases by one every time the snake consumes an Apple.

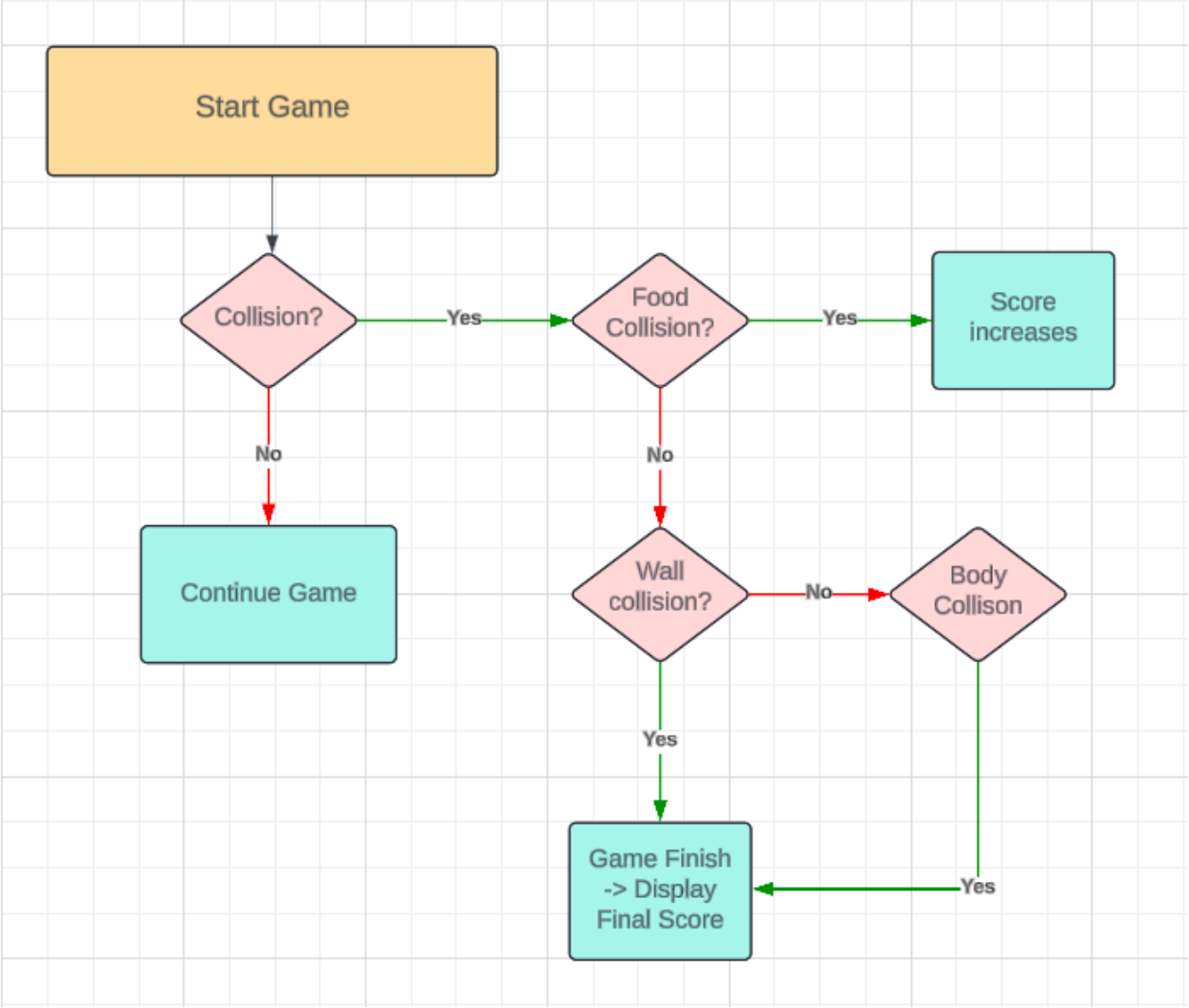

Fig 1. Snake Game Rules and Logic

II. System Overview

The snake game uses both hardware and software components. Hardware part of the project includes the design and implementation of modules for VGA image display, control input via the controller. Specifically, the game logic unit transmits data concerning each object's position and status to the hardware via the device drivers. The hardware peripherals, upon receiving this data, process and decode it to produce the appropriate visual outputs.

The software portion consists of the device drivers and the game logic unit. The game logic manages several key functions, including the display of Apple sprites, Snake sprites, score calculation, food eating, and wall collision detection. The device drivers serve as the bridge connecting the hardware components—such as the controller and VGA monitor—to the logic control unit.

The game logic receives input from the controller given by the user which we use to update the position of the snake. The interaction between these components is structured into a flowchart given below that delineates the overall process, illustrating the continuous data exchange that facilitates the gameplay experience.

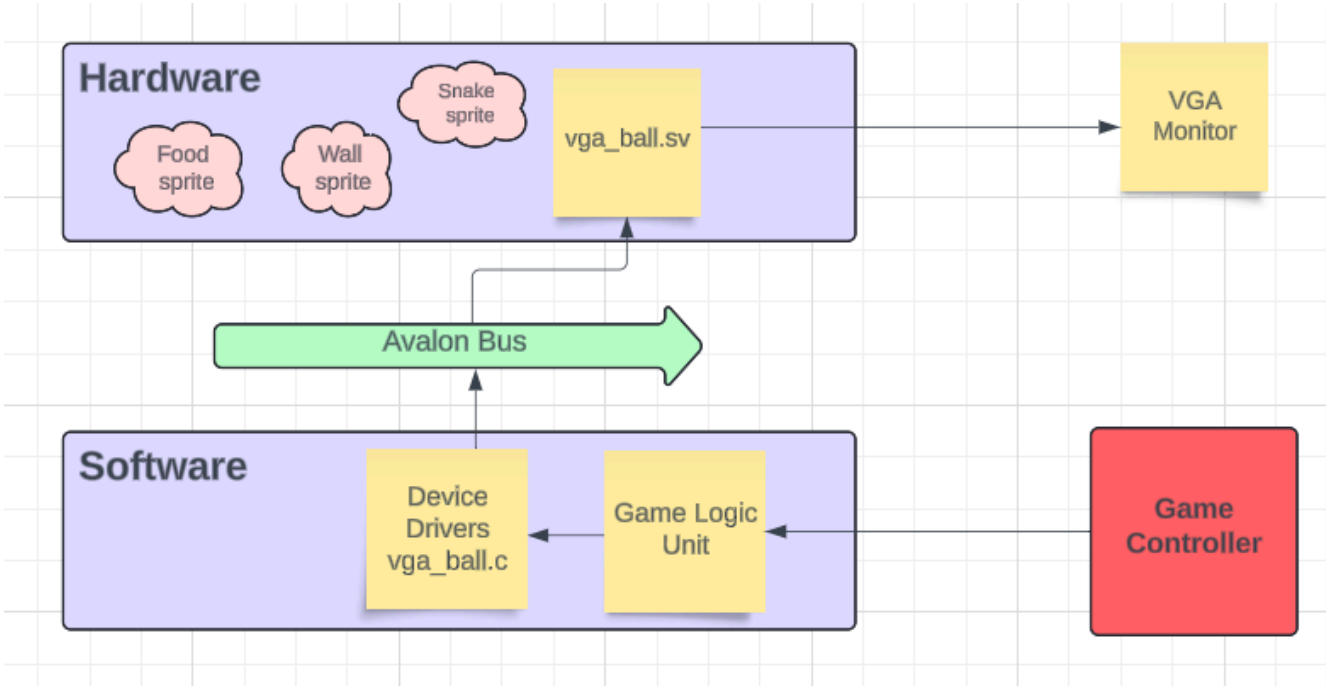

Fig 2. Flowchart for System Overview

The game controller is the input we get from the user playing the game, this input is processed by the game logic unit which is received by the device drivers. The communication between the Software and the Hardware components is done by the Avalon Bus. The Avalon Bus communicates with the System Verilog file and the output is displayed on the VGA Monitor.

The system is primarily divided into several key components:

1. **Input Handling:** The game controller is the input we get from the user playing the game. These inputs dictate the movement of the snake within the game.
2. **Game Logic Unit:** After receiving the user input, the game logic unit processes these inputs to update the game state. This includes moving the snake in the appropriate direction, detecting collisions, and handling the growth of the snake when it eats food.
3. **Avalon Bus:** The communication between the Software and the Hardware components is facilitated by the Avalon Bus. This bus system, provided by Altera, is used for interfacing various components of the system, ensuring smooth data flow between the game logic, device drivers, and other peripherals.
4. **SystemVerilog File:** The vga_ball.sv is the SystemVerilog file that implements the hardware logic. It plays a very important role in displaying the sprites on the VGA monitor and interacting with the software.
5. **Display Output on VGA Monitor:** The VGA signals are received from the FPGA so that the VGA monitor displays the snake movement, the game arena, and the score.

   The controller we are using for the game is a Sony PS5 as shown below:

Fig 3. Sony PS5 Controller

III. Hardware

The hardware portion of our implementation is responsible for displaying the sprites. This is important because as we update the software for the motion of our snake, the sprites should move accordingly to get the perfect visualization and game play for the user.

In our hardware design, display of the sprites is managed through writedata communication using 32-bit data packets. These packets are comprised of four distinct 8-bit packets, effectively encapsulating the necessary data for sprite manipulation in a compact form. To execute this, we have defined a two-dimensional register array reg [7:0] map [39:0][29:0]. This array is structured as follows:

The map array is designed to hold the graphical data for the sprites, with each element of the array corresponding to a specific part of the sprite data. When a sprite needs to be updated or displayed, the corresponding 32-bit data packet is constructed by aggregating four sequential 8-bit values from this array. This method allows for efficient and precise control over the display and positioning of sprites, ensuring that each sprite can be individually addressed and manipulated based on gameplay dynamics.

By organizing the sprite data into this grid of 300 registers, we enable a direct and efficient mapping of sprite information to specific locations on the screen, facilitating rapid updates and high-fidelity visual outputs.

This implementation is as shown below:

```
else if (chipselect && write) begin
  case (address)
    9'h0 : {map[0][0], map[1][0], map[2][0], map[3][0]} <= writedata;
    9'h1 : {map[4][0], map[5][0], map[6][0], map[7][0]} <= writedata;
    9'h2 : {map[8][0], map[9][0], map[10][0], map[11][0]} <= writedata;
    9'h3 : {map[12][0], map[13][0], map[14][0], map[15][0]} <= writedata;
    9'h4 : {map[16][0], map[17][0], map[18][0], map[19][0]} <= writedata;
    9'h5 : {map[20][0], map[21][0], map[22][0], map[23][0]} <= writedata;
    9'h6 : {map[24][0], map[25][0], map[26][0], map[27][0]} <= writedata;
    9'h7 : {map[28][0], map[29][0], map[30][0], map[31][0]} <= writedata;
```

We have divided our VGA Monitor by dividing it into 40x30 tiles such that each tile is 16x16.

IV. Software

Software portion of our game plays an important role in moving the snake sprites in the direction as desired by the game player. In order to implement our software logic, we have defined a combine function as shown below:

```
unsigned long int combine(unsigned short int a, unsigned short int b, unsigned short int c, unsigned short int d) {
    unsigned long int x = 0;

    // Combine the values using bitwise OR and bit shifting
    x |= ((unsigned long int)a) << 24;
    x |= ((unsigned long int)b) << 16;
    x |= ((unsigned long int)c) << 8;
    x |= (unsigned long int)d;
    return x;
}
```

This function has been used by us in the code to position the snake sprites as a full snake, we have mapped the value of each snake sprites in terms of numbers such that:

0 → Background
1 → Apple Sprite
2 → Snake Head Looking Up
3 → Snake Head Looking Down
4 → Snake Head Looking Left
5 → Snake Head Looking Right
6 → Body Vertical
7 → Body Horizontal
8 → Snake Body Corner 1
9 → Snake Body Corner 1
10 → Snake Body Corner 1
11 → Snake Body Corner 1
12 → Snake Tail Up
13 → Snake Tail Down

14 → Snake Tail Left
15 → Snake Tail Right

The way we use the combine function is like:

vla.grid.data = combine(0,0,14,5); // Snake head_right and tail_left placed of the first two columns of the corresponding row

or

vla.grid.data = combine(0,14,7,5); // Snake head_right, horizontal snake body and tail_left placed of the first two columns of the corresponding row

After transmitting the 32-bit packet data using iowrite32 from vga_ball.c, we delve into the intricacies of game logic implementation employing the multi-threading technique to execute game logic concurrently. This approach synchronizes with hardware writes while leveraging libusb_interrupt_transfer for capturing controller inputs, facilitating real-time directional changes in the snake's movement based on user commands.

Transitioning to the game logic, while we may not have completed its entirety, we've laid the theoretical groundwork. To effectively monitor the snake's body and its dynamic movements, we've identified the necessity for a deque data structure. Thus, we introduce our customized snake deque, comprising a struct:

```
typedef struct {
  Map arr[MAX_SIZE];
  int front;
  int rear;
} Deque;
```

Within this Deque structure, we employ an array of customized Map structs. These Map structs encapsulate essential variables representing the snake's body positions and types (e.g., body, head, tail). Here's a glimpse:

```
typedef struct {
  unsigned short int x_pos;
  unsigned short int y_pos;
  unsigned short int dir;
  unsigned short int map;
} Map;
```

To facilitate deque manipulation, we've crafted auxiliary functions like insertFront, insertRear, removeRear, removeFront, and getFront, ensuring seamless integration with our game logic.

Continuing our architectural design, we introduce a hashmap structure essential for efficiently updating the snake's position and transmitting it to hardware. The hashmap comprises:

```
typedef struct {
  Key key;
  int value;
} Entry;

typedef struct {
  Entry *entries[HASHMAP_SIZE];
} HashMap;
```

This hashmap structure, complemented by a hash function, serves as a robust mechanism for mapping snake positions, streamlining the process of updating hardware displays post each movement.

As we progress through this report, specifically within hello.c, you'll gain insights into our data transmission strategies and hardware interfacing methodologies,

depicted through nested for loops. While our software remains a work in progress, these meticulously crafted data structures lay a solid foundation for implementing comprehensive game logic seamlessly integrated with hardware interactions.

V. Hardware - Software Interface

The hardware of the DE1-SoC board interacts with the software logic to achieve its functionality. Writing driver or interface code, controlling memory access, and setting up peripherals are all part of this. Following the computation of the game logic, the software modifies the coordinates of the apple, snake, and body length and sends them via a kernel program to the DE1-SoC board's registers. The VGA display is updated by the hardware once it reads the data from the registers.

VI. Conclusion

The following figures show our achieved results while implementing the Snake Game. Goals achieved:

1. Displaying the Apple, Snake and the Wall sprites successfully.
2. Making the vga_ball.sv file successfully.
3. Connecting the controller successfully to move the snake.
4. Interfacing the Hardware-Software correctly.

In progress - Game Design for moving the snake in different directions.

VII. Code Implementation

vga_ball.sv

```
module vga_ball(
   input logic        clk,
          input logic          reset,
                 input logic [31:0]   writedata,
```

```
                input logic             write,
                input                       chipselect,
                input logic [8:0]   address,

                output logic [7:0] VGA_R, VGA_G, VGA_B,
                output logic        VGA_CLK, VGA_HS, VGA_VS,
                                VGA_BLANK_n,
                output logic        VGA_SYNC_n

 );

 logic [10:0]     hcount;
 logic [9:0]     vcount;

 logic [7:0]       background_r, background_g, background_b;

 logic [15:0]     x, y;

//  this is for apple_sprite
 logic [15:0]     apple_sprite_output;
 logic [9:0]      apple_sprite_addr;
 logic [1:0]      apple_sprite_en;

// snake face forwarded to right
 logic [15:0]     snake_head_right_sprite_output;
 logic [9:0]      snake_head_right_sprite_addr;
 logic [1:0]      snake_head_right_sprite_en;
// snake face forwarded to left
 logic [15:0]     snake_head_left_sprite_output;
 logic [9:0]      snake_head_left_sprite_addr;
 logic [1:0]      snake_head_left_sprite_en;
// snake face forwarded to up
 logic [15:0]     snake_head_up_sprite_output;
 logic [9:0]      snake_head_up_sprite_addr;
 logic [1:0]      snake_head_up_sprite_en;
// snake face forwarded to down
 logic [15:0]     snake_head_down_sprite_output;
 logic [9:0]      snake_head_down_sprite_addr;
 logic [1:0]      snake_head_down_sprite_en;

// body bottom left
 logic [15:0]     snake_body_bottomleft_sprite_output;
 logic [9:0]      snake_body_bottomleft_sprite_addr;
 logic [1:0]      snake_body_bottomleft_sprite_en;
```

```
// body bottom right
 logic [15:0]    snake_body_bottomright_sprite_output;
 logic [9:0]     snake_body_bottomright_sprite_addr;
 logic [1:0]     snake_body_bottomright_sprite_en;

// body top left
 logic [15:0]    snake_body_topleft_sprite_output;
 logic [9:0]     snake_body_topleft_sprite_addr;
 logic [1:0]     snake_body_topleft_sprite_en;

// body top right
 logic [15:0]    snake_body_topright_sprite_output;
 logic [9:0]     snake_body_topright_sprite_addr;
 logic [1:0]     snake_body_topright_sprite_en;

// body horizontal
 logic [15:0]    snake_body_horizontal_sprite_output;
 logic [9:0]     snake_body_horizontal_sprite_addr;
 logic [1:0]     snake_body_horizontal_sprite_en;
// body vertical
 logic [15:0]    snake_body_vertical_sprite_output;
 logic [9:0]     snake_body_vertical_sprite_addr;
 logic [1:0]     snake_body_vertical_sprite_en;

// tail up
 logic [15:0]    snake_tail_up_sprite_output;
 logic [9:0]     snake_tail_up_sprite_addr;
 logic [1:0]     snake_tail_up_sprite_en;
// tail down
 logic [15:0]    snake_tail_down_sprite_output;
 logic [9:0]     snake_tail_down_sprite_addr;
 logic [1:0]     snake_tail_down_sprite_en;
// tail left
 logic [15:0]    snake_tail_left_sprite_output;
 logic [9:0]     snake_tail_left_sprite_addr;
 logic [1:0]     snake_tail_left_sprite_en;
// tail right
 logic [15:0]    snake_tail_right_sprite_output;
 logic [9:0]     snake_tail_right_sprite_addr;
 logic [1:0]     snake_tail_right_sprite_en;

 // wall
 logic [15:0]    wall_sprite_output;
```

```
logic [9:0]      wall_sprite_addr;
logic [1:0]      wall_sprite_en;

// zero
logic [15:0]     zero_sprite_output;
logic [9:0]      zero_sprite_addr;
logic [1:0]      zero_sprite_en;

// one
logic [15:0]     one_sprite_output;
logic [9:0]      one_sprite_addr;
logic [1:0]      one_sprite_en;

// two
logic [15:0]     two_sprite_output;
logic [9:0]      two_sprite_addr;
logic [1:0]      two_sprite_en;

// three
logic [15:0]     three_sprite_output;
logic [9:0]      three_sprite_addr;
logic [1:0]      three_sprite_en;

// four
logic [15:0]     four_sprite_output;
logic [9:0]      four_sprite_addr;
logic [1:0]      four_sprite_en;

// five
logic [15:0]     five_sprite_output;
logic [9:0]      five_sprite_addr;
logic [1:0]      five_sprite_en;

// six
logic [15:0]     six_sprite_output;
logic [9:0]      six_sprite_addr;
logic [1:0]      six_sprite_en;

// seven
logic [15:0]     seven_sprite_output;
logic [9:0]      seven_sprite_addr;
logic [1:0]      seven_sprite_en;

// eight
```

```
  logic [15:0]    eight_sprite_output;
  logic [9:0]     eight_sprite_addr;
  logic [1:0]     eight_sprite_en;

  // nine
  logic [15:0]    nine_sprite_output;
  logic [9:0]     nine_sprite_addr;
  logic [1:0]     nine_sprite_en;

  vga_counters counters(.clk50(clk), .*);

 //  apple
 soc_system_apple_sprite apple_sprite(.address(apple_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(apple_sprite_output));
 //  face right
 soc_system_snake_head_right_sprite
snake_head_right_sprite(.address(snake_head_right_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(snake_head_right_sprite_output));
 //  face left
 soc_system_snake_head_left_sprite
snake_head_left_sprite(.address(snake_head_left_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(snake_head_left_sprite_output));
 //  face up
 soc_system_snake_head_up_sprite
snake_head_up_sprite(.address(snake_head_up_sprite_addr), .clk(clk), .clken(1), .reset_req(0),
.readdata(snake_head_up_sprite_output));
 //  face down
 soc_system_snake_head_down_sprite
snake_head_down_sprite(.address(snake_head_down_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(snake_head_down_sprite_output));

 // snake body
 // bottom left
 soc_system_snake_body_bottomleft_sprite
snake_body_bottomleft_sprite(.address(snake_body_bottomleft_sprite_addr), .clk(clk),
.clken(1), .reset_req(0), .readdata(snake_body_bottomleft_sprite_output));
 // bottom right
 soc_system_snake_body_bottomright_sprite
snake_body_bottomright_sprite(.address(snake_body_bottomright_sprite_addr), .clk(clk),
.clken(1), .reset_req(0), .readdata(snake_body_bottomright_sprite_output));
 // top left
 soc_system_snake_body_topleft_sprite
snake_body_topleft_sprite(.address(snake_body_topleft_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(snake_body_topleft_sprite_output));
```

```
  // top right
  soc_system_snake_body_topright_sprite
snake_body_topright_sprite(.address(snake_body_topright_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(snake_body_topright_sprite_output));

  // body horizontal
   soc_system_snake_body_horizontal_sprite
snake_body_horizontal_sprite(.address(snake_body_horizontal_sprite_addr), .clk(clk),
.clken(1), .reset_req(0), .readdata(snake_body_horizontal_sprite_output));
  // body vertical
   soc_system_snake_body_vertical_sprite
snake_body_vertical_sprite(.address(snake_body_vertical_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(snake_body_vertical_sprite_output));
  // tail up
  soc_system_snake_tail_up_sprite snake_tail_up_sprite(.address(snake_tail_up_sprite_addr),
.clk(clk), .clken(1), .reset_req(0), .readdata(snake_tail_up_sprite_output));
  // tail down
  soc_system_snake_tail_down_sprite
snake_tail_down_sprite(.address(snake_tail_down_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(snake_tail_down_sprite_output));
  // tail left
  soc_system_snake_tail_left_sprite snake_tail_left_sprite(.address(snake_tail_left_sprite_addr),
.clk(clk), .clken(1), .reset_req(0), .readdata(snake_tail_left_sprite_output));
  // tail right
  soc_system_snake_tail_right_sprite
snake_tail_right_sprite(.address(snake_tail_right_sprite_addr), .clk(clk), .clken(1), .reset_req(0),
.readdata(snake_tail_right_sprite_output));
  // wall
  soc_system_wall_sprite wall_sprite(.address(wall_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(wall_sprite_output));

  // zero
  soc_system_zero_sprite zero_sprite(.address(zero_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(zero_sprite_output));
  // one
  soc_system_one_sprite one_sprite(.address(one_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(one_sprite_output));
  // two
  soc_system_two_sprite two_sprite(.address(two_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(two_sprite_output));
  // three
  soc_system_three_sprite three_sprite(.address(three_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(three_sprite_output));
  // four
```

```
  soc_system_four_sprite four_sprite(.address(four_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(four_sprite_output));
  // five
  soc_system_five_sprite five_sprite(.address(five_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(five_sprite_output));
  // six
  //soc_system_six_sprite six_sprite(.address(six_sprite_addr), .clk(clk), .clken(1), .reset_req(0),
.readdata(six_sprite_output));
  // seven
  soc_system_seven_sprite seven_sprite(.address(seven_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(seven_sprite_output));
  // eight
  soc_system_eight_sprite eight_sprite(.address(eight_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(eight_sprite_output));
  // nine
  soc_system_nine_sprite nine_sprite(.address(nine_sprite_addr), .clk(clk), .clken(1),
.reset_req(0), .readdata(nine_sprite_output));




  reg [7:0] snake_head_pos_x;
  reg [7:0] snake_head_pos_y;

  reg [7:0] snake_head_up_pos_x;
  reg [7:0] snake_head_up_pos_y;

  reg [7:0] x_pos;
  reg [7:0] y_pos;
  reg [7:0] sprite_type;

  reg [7:0] map [39:0][29:0];
  reg [8:0] x_offset;
  reg [8:0] y_offset;

  always_ff @(posedge clk) begin
    if (reset) begin
      background_r <= 8'h0;
      background_g <= 8'h0;
      background_b <= 8'h0;
      snake_head_pos_x <= 8'b00001010;
      snake_head_pos_y <= 8'b00001010;
      snake_head_up_pos_x <= 8'b1111;
      snake_head_up_pos_y <= 8'b1111;
```

```
  x_offset <= 9'b0;
  y_offset <= 9'b0;
end
else if (chipselect && write) begin
  case (address)
    9'h0 : {map[0][0], map[1][0], map[2][0], map[3][0]} <= writedata;
    9'h1 : {map[4][0], map[5][0], map[6][0], map[7][0]} <= writedata;
    9'h2 : {map[8][0], map[9][0], map[10][0], map[11][0]} <= writedata;
    9'h3 : {map[12][0], map[13][0], map[14][0], map[15][0]} <= writedata;
    9'h4 : {map[16][0], map[17][0], map[18][0], map[19][0]} <= writedata;
    9'h5 : {map[20][0], map[21][0], map[22][0], map[23][0]} <= writedata;
    9'h6 : {map[24][0], map[25][0], map[26][0], map[27][0]} <= writedata;
    9'h7 : {map[28][0], map[29][0], map[30][0], map[31][0]} <= writedata;
    9'h8 : {map[32][0], map[33][0], map[34][0], map[35][0]} <= writedata;
    9'h9 : {map[36][0], map[37][0], map[38][0], map[39][0]} <= writedata;

    9'ha : {map[0][1], map[1][1], map[2][1], map[3][1]} <= writedata;
    9'hb : {map[4][1], map[5][1], map[6][1], map[7][1]} <= writedata;
    9'hc : {map[8][1], map[9][1], map[10][1], map[11][1]} <= writedata;
    9'hd : {map[12][1], map[13][1], map[14][1], map[15][1]} <= writedata;
    9'he : {map[16][1], map[17][1], map[18][1], map[19][1]} <= writedata;
    9'hf : {map[20][1], map[21][1], map[22][1], map[23][1]} <= writedata;
    9'h10 : {map[24][1], map[25][1], map[26][1], map[27][1]} <= writedata;
    9'h11 : {map[28][1], map[29][1], map[30][1], map[31][1]} <= writedata;
    9'h12 : {map[32][1], map[33][1], map[34][1], map[35][1]} <= writedata;
    9'h13 : {map[36][1], map[37][1], map[38][1], map[39][1]} <= writedata;

    9'h14 : {map[0][2], map[1][2], map[2][2], map[3][2]} <= writedata;
    9'h15 : {map[4][2], map[5][2], map[6][2], map[7][2]} <= writedata;
    9'h16 : {map[8][2], map[9][2], map[10][2], map[11][2]} <= writedata;
    9'h17 : {map[12][2], map[13][2], map[14][2], map[15][2]} <= writedata;
    9'h18 : {map[16][2], map[17][2], map[18][2], map[19][2]} <= writedata;
    9'h19 : {map[20][2], map[21][2], map[22][2], map[23][2]} <= writedata;
    9'h1a : {map[24][2], map[25][2], map[26][2], map[27][2]} <= writedata;
    9'h1b : {map[28][2], map[29][2], map[30][2], map[31][2]} <= writedata;
    9'h1c : {map[32][2], map[33][2], map[34][2], map[35][2]} <= writedata;
    9'h1d : {map[36][2], map[37][2], map[38][2], map[39][2]} <= writedata;

    9'h1e : {map[0][3], map[1][3], map[2][3], map[3][3]} <= writedata;
    9'h1f : {map[4][3], map[5][3], map[6][3], map[7][3]} <= writedata;
    9'h20 : {map[8][3], map[9][3], map[10][3], map[11][3]} <= writedata;
    9'h21 : {map[12][3], map[13][3], map[14][3], map[15][3]} <= writedata;
    9'h22 : {map[16][3], map[17][3], map[18][3], map[19][3]} <= writedata;
    9'h23 : {map[20][3], map[21][3], map[22][3], map[23][3]} <= writedata;
```

```
9'h24 : {map[24][3], map[25][3], map[26][3], map[27][3]} <= writedata;
9'h25 : {map[28][3], map[29][3], map[30][3], map[31][3]} <= writedata;
9'h26 : {map[32][3], map[33][3], map[34][3], map[35][3]} <= writedata;
9'h27 : {map[36][3], map[37][3], map[38][3], map[39][3]} <= writedata;

9'h28 : {map[0][4], map[1][4], map[2][4], map[3][4]} <= writedata;
9'h29 : {map[4][4], map[5][4], map[6][4], map[7][4]} <= writedata;
9'h2a : {map[8][4], map[9][4], map[10][4], map[11][4]} <= writedata;
9'h2b : {map[12][4], map[13][4], map[14][4], map[15][4]} <= writedata;
9'h2c : {map[16][4], map[17][4], map[18][4], map[19][4]} <= writedata;
9'h2d : {map[20][4], map[21][4], map[22][4], map[23][4]} <= writedata;
9'h2e : {map[24][4], map[25][4], map[26][4], map[27][4]} <= writedata;
9'h2f : {map[28][4], map[29][4], map[30][4], map[31][4]} <= writedata;
9'h30 : {map[32][4], map[33][4], map[34][4], map[35][4]} <= writedata;
9'h31 : {map[36][4], map[37][4], map[38][4], map[39][4]} <= writedata;

9'h32 : {map[0][5], map[1][5], map[2][5], map[3][5]} <= writedata;
9'h33 : {map[4][5], map[5][5], map[6][5], map[7][5]} <= writedata;
9'h34 : {map[8][5], map[9][5], map[10][5], map[11][5]} <= writedata;
9'h35 : {map[12][5], map[13][5], map[14][5], map[15][5]} <= writedata;
9'h36 : {map[16][5], map[17][5], map[18][5], map[19][5]} <= writedata;
9'h37 : {map[20][5], map[21][5], map[22][5], map[23][5]} <= writedata;
9'h38 : {map[24][5], map[25][5], map[26][5], map[27][5]} <= writedata;
9'h39 : {map[28][5], map[29][5], map[30][5], map[31][5]} <= writedata;
9'h3a : {map[32][5], map[33][5], map[34][5], map[35][5]} <= writedata;
9'h3b : {map[36][5], map[37][5], map[38][5], map[39][5]} <= writedata;

9'h3c : {map[0][6], map[1][6], map[2][6], map[3][6]} <= writedata;
9'h3d : {map[4][6], map[5][6], map[6][6], map[7][6]} <= writedata;
9'h3e : {map[8][6], map[9][6], map[10][6], map[11][6]} <= writedata;
9'h3f : {map[12][6], map[13][6], map[14][6], map[15][6]} <= writedata;
9'h40 : {map[16][6], map[17][6], map[18][6], map[19][6]} <= writedata;
9'h41 : {map[20][6], map[21][6], map[22][6], map[23][6]} <= writedata;
9'h42 : {map[24][6], map[25][6], map[26][6], map[27][6]} <= writedata;
9'h43 : {map[28][6], map[29][6], map[30][6], map[31][6]} <= writedata;
9'h44 : {map[32][6], map[33][6], map[34][6], map[35][6]} <= writedata;
9'h45 : {map[36][6], map[37][6], map[38][6], map[39][6]} <= writedata;


9'h46 : {map[0][7], map[1][7], map[2][7], map[3][7]} <= writedata;
9'h47 : {map[4][7], map[5][7], map[6][7], map[7][7]} <= writedata;
9'h48 : {map[8][7], map[9][7], map[10][7], map[11][7]} <= writedata;
9'h49 : {map[12][7], map[13][7], map[14][7], map[15][7]} <= writedata;
9'h4a : {map[16][7], map[17][7], map[18][7], map[19][7]} <= writedata;
```

```
9'h4b : {map[20][7], map[21][7], map[22][7], map[23][7]} <= writedata;
9'h4c : {map[24][7], map[25][7], map[26][7], map[27][7]} <= writedata;
9'h4d : {map[28][7], map[29][7], map[30][7], map[31][7]} <= writedata;
9'h4e : {map[32][7], map[33][7], map[34][7], map[35][7]} <= writedata;
9'h4f : {map[36][7], map[37][7], map[38][7], map[39][7]} <= writedata;

9'h50 : {map[0][8], map[1][8], map[2][8], map[3][8]} <= writedata;
9'h51 : {map[4][8], map[5][8], map[6][8], map[7][8]} <= writedata;
9'h52 : {map[8][8], map[9][8], map[10][8], map[11][8]} <= writedata;
9'h53 : {map[12][8], map[13][8], map[14][8], map[15][8]} <= writedata;
9'h54 : {map[16][8], map[17][8], map[18][8], map[19][8]} <= writedata;
9'h55 : {map[20][8], map[21][8], map[22][8], map[23][8]} <= writedata;
9'h56 : {map[24][8], map[25][8], map[26][8], map[27][8]} <= writedata;
9'h57 : {map[28][8], map[29][8], map[30][8], map[31][8]} <= writedata;
9'h58 : {map[32][8], map[33][8], map[34][8], map[35][8]} <= writedata;
9'h59 : {map[36][8], map[37][8], map[38][8], map[39][8]} <= writedata;

9'h5a : {map[0][9], map[1][9], map[2][9], map[3][9]} <= writedata;
9'h5b : {map[4][9], map[5][9], map[6][9], map[7][9]} <= writedata;
9'h5c : {map[8][9], map[9][9], map[10][9], map[11][9]} <= writedata;
9'h5d : {map[12][9], map[13][9], map[14][9], map[15][9]} <= writedata;
9'h5e : {map[16][9], map[17][9], map[18][9], map[19][9]} <= writedata;
9'h5f : {map[20][9], map[21][9], map[22][9], map[23][9]} <= writedata;
9'h60 : {map[24][9], map[25][9], map[26][9], map[27][9]} <= writedata;
9'h61 : {map[28][9], map[29][9], map[30][9], map[31][9]} <= writedata;
9'h62 : {map[32][9], map[33][9], map[34][9], map[35][9]} <= writedata;
9'h63 : {map[36][9], map[37][9], map[38][9], map[39][9]} <= writedata;

9'h64 : {map[0][10], map[1][10], map[2][10], map[3][10]} <= writedata;
9'h65 : {map[4][10], map[5][10], map[6][10], map[7][10]} <= writedata;
9'h66 : {map[8][10], map[9][10], map[10][10], map[11][10]} <= writedata;
9'h67 : {map[12][10], map[13][10], map[14][10], map[15][10]} <= writedata;
9'h68 : {map[16][10], map[17][10], map[18][10], map[19][10]} <= writedata;
9'h69 : {map[20][10], map[21][10], map[22][10], map[23][10]} <= writedata;
9'h6a : {map[24][10], map[25][10], map[26][10], map[27][10]} <= writedata;
9'h6b : {map[28][10], map[29][10], map[30][10], map[31][10]} <= writedata;
9'h6c : {map[32][10], map[33][10], map[34][10], map[35][10]} <= writedata;
9'h6d : {map[36][10], map[37][10], map[38][10], map[39][10]} <= writedata;

9'h6e : {map[0][11], map[1][11], map[2][11], map[3][11]} <= writedata;
9'h6f : {map[4][11], map[5][11], map[6][11], map[7][11]} <= writedata;
9'h70 : {map[8][11], map[9][11], map[10][11], map[11][11]} <= writedata;
9'h71 : {map[12][11], map[13][11], map[14][11], map[15][11]} <= writedata;
9'h72 : {map[16][11], map[17][11], map[18][11], map[19][11]} <= writedata;
```

```
9'h73 : {map[20][11], map[21][11], map[22][11], map[23][11]} <= writedata;
9'h74 : {map[24][11], map[25][11], map[26][11], map[27][11]} <= writedata;
9'h75 : {map[28][11], map[29][11], map[30][11], map[31][11]} <= writedata;
9'h76 : {map[32][11], map[33][11], map[34][11], map[35][11]} <= writedata;
9'h77 : {map[36][11], map[37][11], map[38][11], map[39][11]} <= writedata;

9'h78 : {map[0][12], map[1][12], map[2][12], map[3][12]} <= writedata;
9'h79 : {map[4][12], map[5][12], map[6][12], map[7][12]} <= writedata;
9'h7a : {map[8][12], map[9][12], map[10][12], map[11][12]} <= writedata;
9'h7b : {map[12][12], map[13][12], map[14][12], map[15][12]} <= writedata;
9'h7c : {map[16][12], map[17][12], map[18][12], map[19][12]} <= writedata;
9'h7d : {map[20][12], map[21][12], map[22][12], map[23][12]} <= writedata;
9'h7e : {map[24][12], map[25][12], map[26][12], map[27][12]} <= writedata;
9'h7f : {map[28][12], map[29][12], map[30][12], map[31][12]} <= writedata;
9'h80 : {map[32][12], map[33][12], map[34][12], map[35][12]} <= writedata;
9'h81 : {map[36][12], map[37][12], map[38][12], map[39][12]} <= writedata;

9'h82 : {map[0][13], map[1][13], map[2][13], map[3][13]} <= writedata;
9'h83 : {map[4][13], map[5][13], map[6][13], map[7][13]} <= writedata;
9'h84 : {map[8][13], map[9][13], map[10][13], map[11][13]} <= writedata;
9'h85 : {map[12][13], map[13][13], map[14][13], map[15][13]} <= writedata;
9'h86 : {map[16][13], map[17][13], map[18][13], map[19][13]} <= writedata;
9'h87 : {map[20][13], map[21][13], map[22][13], map[23][13]} <= writedata;
9'h88 : {map[24][13], map[25][13], map[26][13], map[27][13]} <= writedata;
9'h89 : {map[28][13], map[29][13], map[30][13], map[31][13]} <= writedata;
9'h8a : {map[32][13], map[33][13], map[34][13], map[35][13]} <= writedata;
9'h8b : {map[36][13], map[37][13], map[38][13], map[39][13]} <= writedata;

9'h8c : {map[0][14], map[1][14], map[2][14], map[3][14]} <= writedata;
9'h8d : {map[4][14], map[5][14], map[6][14], map[7][14]} <= writedata;
9'h8e : {map[8][14], map[9][14], map[10][14], map[11][14]} <= writedata;
9'h8f : {map[12][14], map[13][14], map[14][14], map[15][14]} <= writedata;
9'h90 : {map[16][14], map[17][14], map[18][14], map[19][14]} <= writedata;
9'h91 : {map[20][14], map[21][14], map[22][14], map[23][14]} <= writedata;
9'h92 : {map[24][14], map[25][14], map[26][14], map[27][14]} <= writedata;
9'h93 : {map[28][14], map[29][14], map[30][14], map[31][14]} <= writedata;
9'h94 : {map[32][14], map[33][14], map[34][14], map[35][14]} <= writedata;
9'h95 : {map[36][14], map[37][14], map[38][14], map[39][14]} <= writedata;

9'h96 : {map[0][15], map[1][15], map[2][15], map[3][15]} <= writedata;
9'h97 : {map[4][15], map[5][15], map[6][15], map[7][15]} <= writedata;
9'h98 : {map[8][15], map[9][15], map[10][15], map[11][15]} <= writedata;
9'h99 : {map[12][15], map[13][15], map[14][15], map[15][15]} <= writedata;
9'h9a : {map[16][15], map[17][15], map[18][15], map[19][15]} <= writedata;
```

```
9'h9b : {map[20][15], map[21][15], map[22][15], map[23][15]} <= writedata;
9'h9c : {map[24][15], map[25][15], map[26][15], map[27][15]} <= writedata;
9'h9d : {map[28][15], map[29][15], map[30][15], map[31][15]} <= writedata;
9'h9e : {map[32][15], map[33][15], map[34][15], map[35][15]} <= writedata;
9'h9f : {map[36][15], map[37][15], map[38][15], map[39][15]} <= writedata;

9'ha0 : {map[0][16], map[1][16], map[2][16], map[3][16]} <= writedata;
9'ha1 : {map[4][16], map[5][16], map[6][16], map[7][16]} <= writedata;
9'ha2 : {map[8][16], map[9][16], map[10][16], map[11][16]} <= writedata;
9'ha3 : {map[12][16], map[13][16], map[14][16], map[15][16]} <= writedata;
9'ha4 : {map[16][16], map[17][16], map[18][16], map[19][16]} <= writedata;
9'ha5 : {map[20][16], map[21][16], map[22][16], map[23][16]} <= writedata;
9'ha6 : {map[24][16], map[25][16], map[26][16], map[27][16]} <= writedata;
9'ha7 : {map[28][16], map[29][16], map[30][16], map[31][16]} <= writedata;
9'ha8 : {map[32][16], map[33][16], map[34][16], map[35][16]} <= writedata;
9'ha9 : {map[36][16], map[37][16], map[38][16], map[39][16]} <= writedata;

9'haa : {map[0][17], map[1][17], map[2][17], map[3][17]} <= writedata;
9'hab : {map[4][17], map[5][17], map[6][17], map[7][17]} <= writedata;
9'hac : {map[8][17], map[17][9], map[10][17], map[11][17]} <= writedata;
9'had : {map[12][17], map[13][17], map[14][17], map[15][17]} <= writedata;
9'hae : {map[16][17], map[17][17], map[18][17], map[19][17]} <= writedata;
9'haf : {map[20][17], map[21][17], map[22][17], map[23][17]} <= writedata;
9'hb0 : {map[24][17], map[25][17], map[26][17], map[27][17]} <= writedata;
9'hb1 : {map[28][17], map[29][17], map[30][17], map[31][17]} <= writedata;
9'hb2 : {map[32][17], map[33][17], map[34][17], map[35][17]} <= writedata;
9'hb3 : {map[36][17], map[37][17], map[38][17], map[39][17]} <= writedata;

9'hb4 : {map[0][18], map[1][18], map[2][18], map[3][18]} <= writedata;
9'hb5 : {map[4][18], map[5][18], map[6][18], map[7][18]} <= writedata;
9'hb6 : {map[8][18], map[9][18], map[10][18], map[11][18]} <= writedata;
9'hb7 : {map[12][18], map[13][18], map[14][18], map[15][18]} <= writedata;
9'hb8 : {map[16][18], map[17][18], map[18][18], map[19][18]} <= writedata;
9'hb9 : {map[20][18], map[21][18], map[22][18], map[23][18]} <= writedata;
9'hba : {map[24][18], map[25][18], map[26][18], map[27][18]} <= writedata;
9'hbb : {map[28][18], map[29][18], map[30][18], map[31][18]} <= writedata;
9'hbc : {map[32][18], map[33][18], map[34][18], map[35][18]} <= writedata;
9'hbd : {map[36][18], map[37][18], map[38][18], map[39][18]} <= writedata;

9'hbe : {map[0][19], map[1][19], map[2][19], map[3][19]} <= writedata;
9'hbf : {map[4][19], map[5][19], map[6][19], map[7][19]} <= writedata;
9'he0 : {map[8][19], map[9][19], map[10][19], map[11][19]} <= writedata;
9'hc1 : {map[12][19], map[13][19], map[14][19], map[15][19]} <= writedata;
9'hc2 : {map[16][19], map[17][19], map[18][19], map[19][19]} <= writedata;
```

```
9'hc3 : {map[20][19], map[21][19], map[22][19], map[23][19]} <= writedata;
9'hc4 : {map[24][19], map[25][19], map[26][19], map[27][19]} <= writedata;
9'hc5 : {map[28][19], map[29][19], map[30][19], map[31][19]} <= writedata;
9'hc6 : {map[32][19], map[33][19], map[34][19], map[35][19]} <= writedata;
9'hc7 : {map[36][19], map[37][19], map[38][19], map[39][19]} <= writedata;

9'hc8 : {map[0][20], map[1][20], map[2][20], map[3][20]} <= writedata;
9'hc9 : {map[4][20], map[5][20], map[6][20], map[7][20]} <= writedata;
9'hca : {map[8][20], map[9][20], map[10][20], map[11][20]} <= writedata;
9'hcb : {map[12][20], map[13][20], map[14][20], map[15][20]} <= writedata;
9'hcc : {map[16][20], map[17][20], map[18][20], map[19][20]} <= writedata;
9'hcd : {map[20][20], map[21][20], map[22][20], map[23][20]} <= writedata;
9'hce : {map[24][20], map[25][20], map[26][20], map[27][20]} <= writedata;
9'hcf : {map[28][20], map[29][20], map[30][20], map[31][20]} <= writedata;
9'hd0 : {map[32][20], map[33][20], map[34][20], map[35][20]} <= writedata;
9'hd1 : {map[36][20], map[37][20], map[38][20], map[39][20]} <= writedata;

9'hd2 : {map[0][21], map[1][21], map[2][21], map[3][21]} <= writedata;
9'hd3 : {map[4][21], map[5][21], map[6][21], map[7][21]} <= writedata;
9'hd4 : {map[8][21], map[9][21], map[10][21], map[11][21]} <= writedata;
9'hd5 : {map[12][21], map[13][21], map[14][21], map[15][21]} <= writedata;
9'hd6 : {map[16][21], map[17][21], map[18][21], map[19][21]} <= writedata;
9'hd7 : {map[20][21], map[21][21], map[22][21], map[23][21]} <= writedata;
9'hd8 : {map[24][21], map[25][21], map[26][21], map[27][21]} <= writedata;
9'hd9 : {map[28][21], map[29][21], map[30][21], map[31][21]} <= writedata;
9'hda : {map[32][21], map[33][21], map[34][21], map[35][21]} <= writedata;
9'hdb : {map[36][21], map[37][21], map[38][21], map[39][21]} <= writedata;

9'hdc : {map[0][22], map[1][22], map[2][22], map[3][22]} <= writedata;
9'hdd : {map[4][22], map[5][22], map[6][22], map[7][22]} <= writedata;
9'hde : {map[8][22], map[9][22], map[10][22], map[11][22]} <= writedata;
9'hdf : {map[12][22], map[13][22], map[14][22], map[15][22]} <= writedata;
9'he0 : {map[16][22], map[17][22], map[18][22], map[19][22]} <= writedata;
9'he1 : {map[20][22], map[21][22], map[22][22], map[23][22]} <= writedata;
9'he2 : {map[24][22], map[25][22], map[26][22], map[27][22]} <= writedata;
9'he3 : {map[28][22], map[29][22], map[30][22], map[31][22]} <= writedata;
9'he4 : {map[32][22], map[33][22], map[34][22], map[35][22]} <= writedata;
9'he5 : {map[36][22], map[37][22], map[38][22], map[39][22]} <= writedata;

9'he6 : {map[0][23], map[1][23], map[2][23], map[3][23]} <= writedata;
9'he7 : {map[4][23], map[5][23], map[6][23], map[7][23]} <= writedata;
9'he8 : {map[8][23], map[9][23], map[10][23], map[11][23]} <= writedata;
9'he9 : {map[12][23], map[13][23], map[14][23], map[15][23]} <= writedata;
9'hea : {map[16][23], map[17][23], map[18][23], map[19][23]} <= writedata;
```

```
9'heb : {map[20][23], map[21][23], map[22][23], map[23][23]} <= writedata;
9'hec : {map[24][23], map[25][23], map[26][23], map[27][23]} <= writedata;
9'hed : {map[28][23], map[29][23], map[30][23], map[31][23]} <= writedata;
9'hee : {map[32][23], map[33][23], map[34][23], map[35][23]} <= writedata;
9'hef : {map[36][23], map[37][23], map[38][23], map[39][23]} <= writedata;

9'hf0 : {map[0][24], map[1][24], map[2][24], map[3][24]} <= writedata;
9'hf1 : {map[4][24], map[5][24], map[6][24], map[7][24]} <= writedata;
9'hf2 : {map[8][24], map[9][24], map[10][24], map[11][24]} <= writedata;
9'hf3 : {map[12][24], map[13][24], map[14][24], map[15][24]} <= writedata;
9'hf4 : {map[16][24], map[17][24], map[18][24], map[19][24]} <= writedata;
9'hf5 : {map[20][24], map[21][24], map[22][24], map[23][24]} <= writedata;
9'hf6 : {map[24][24], map[25][24], map[26][24], map[27][24]} <= writedata;
9'hf7 : {map[28][24], map[29][24], map[30][24], map[31][24]} <= writedata;
9'hf8 : {map[32][24], map[33][24], map[34][24], map[35][24]} <= writedata;
9'hf9 : {map[36][24], map[37][24], map[38][24], map[39][24]} <= writedata;

9'hfa : {map[0][25], map[1][25], map[2][25], map[3][25]} <= writedata;
9'hfb : {map[4][25], map[5][25], map[6][25], map[7][25]} <= writedata;
9'hfc : {map[8][25], map[25][9], map[10][25], map[11][25]} <= writedata;
9'hfd : {map[12][25], map[13][25], map[14][25], map[15][25]} <= writedata;
9'hfe : {map[16][25], map[17][25], map[18][25], map[19][25]} <= writedata;
9'hff : {map[20][25], map[21][25], map[22][25], map[23][25]} <= writedata;
9'h100 : {map[24][25], map[25][25], map[26][25], map[27][25]} <= writedata;
9'h101 : {map[28][25], map[29][25], map[30][25], map[31][25]} <= writedata;
9'h102 : {map[32][25], map[33][25], map[34][25], map[35][25]} <= writedata;
9'h103 : {map[36][25], map[37][25], map[38][25], map[39][25]} <= writedata;

9'h104 : {map[0][26], map[1][26], map[2][26], map[3][26]} <= writedata;
9'h105 : {map[4][26], map[5][26], map[6][26], map[7][26]} <= writedata;
9'h106 : {map[8][26], map[9][26], map[10][26], map[11][26]} <= writedata;
9'h107 : {map[12][26], map[13][26], map[14][26], map[15][26]} <= writedata;
9'h108 : {map[16][26], map[17][26], map[18][26], map[19][26]} <= writedata;
9'h109 : {map[20][26], map[21][26], map[22][26], map[23][26]} <= writedata;
9'h10a : {map[24][26], map[25][26], map[26][26], map[27][26]} <= writedata;
9'h10b : {map[28][26], map[29][26], map[30][26], map[31][26]} <= writedata;
9'h10c : {map[32][26], map[33][26], map[34][26], map[35][26]} <= writedata;
9'h10d : {map[36][26], map[37][26], map[38][26], map[39][26]} <= writedata;

9'h10e : {map[0][27], map[1][27], map[2][27], map[3][27]} <= writedata;
9'h10f : {map[4][27], map[5][27], map[6][27], map[7][27]} <= writedata;
9'h110 : {map[8][27], map[9][27], map[10][27], map[11][27]} <= writedata;
9'h111 : {map[12][27], map[13][27], map[14][27], map[15][27]} <= writedata;
9'h112 : {map[16][27], map[17][27], map[18][27], map[19][27]} <= writedata;
```

```
  9'h113 : {map[20][27], map[21][27], map[22][27], map[23][27]} <= writedata;
  9'h114 : {map[24][27], map[25][27], map[26][27], map[27][27]} <= writedata;
  9'h115 : {map[28][27], map[29][27], map[30][27], map[31][27]} <= writedata;
  9'h116 : {map[32][27], map[33][27], map[34][27], map[35][27]} <= writedata;
  9'h117 : {map[36][27], map[37][27], map[38][27], map[39][27]} <= writedata;

  9'h118 : {map[0][28], map[1][28], map[2][28], map[3][28]} <= writedata;
  9'h119 : {map[4][28], map[5][28], map[6][28], map[7][28]} <= writedata;
  9'h11a : {map[8][28], map[9][28], map[10][28], map[11][28]} <= writedata;
  9'h11b : {map[12][28], map[13][28], map[14][28], map[15][28]} <= writedata;
  9'h11c : {map[16][28], map[17][28], map[18][28], map[19][28]} <= writedata;
  9'h11d : {map[20][28], map[21][28], map[22][28], map[23][28]} <= writedata;
  9'h11e : {map[24][28], map[25][28], map[26][28], map[27][28]} <= writedata;
  9'h11f : {map[28][28], map[29][28], map[30][28], map[31][28]} <= writedata;
  9'h120 : {map[32][28], map[33][28], map[34][28], map[35][28]} <= writedata;
  9'h121 : {map[36][28], map[37][28], map[38][28], map[39][28]} <= writedata;

  9'h122 : {map[0][29], map[1][29], map[2][29], map[3][29]} <= writedata;
  9'h123 : {map[4][29], map[5][29], map[6][29], map[7][29]} <= writedata;
  9'h124 : {map[8][29], map[9][29], map[10][29], map[11][29]} <= writedata;
  9'h125 : {map[12][29], map[13][29], map[14][29], map[15][29]} <= writedata;
  9'h126 : {map[16][29], map[17][29], map[18][29], map[19][29]} <= writedata;
  9'h127 : {map[20][29], map[21][29], map[22][29], map[23][29]} <= writedata;
  9'h128 : {map[24][29], map[25][29], map[26][29], map[27][29]} <= writedata;
  9'h129 : {map[28][29], map[29][29], map[30][29], map[31][29]} <= writedata;
  9'h12a : {map[32][29], map[33][29], map[34][29], map[35][29]} <= writedata;
  9'h12b : {map[36][29], map[37][29], map[38][29], map[39][29]} <= writedata;




endcase
/*
if (x_offset == 9'b100111) begin
  x_offset <= 0;
  y_offset <= y_offset + 1;
end else if (y_offset == 29) begin
  x_offset <= 0;
  y_offset <= 0;
end else begin
  x_offset <= x_offset + 4;
end
*/
// map[x_pos][y_pos] <= sprite_type;
```

```
 end
 //map[x_pos][y_pos] <= sprite_type;
end

//logic for generating vga output
reg [7:0] a;
reg [7:0] b;
reg [7:0] c;

reg [7:0] apple_x;
reg [7:0] apple_y;

reg [7:0] head_output1;
reg [7:0] head_output2;
reg [7:0] head_output3;

// ---------------------------------
always_ff @(posedge clk) begin
  if (VGA_BLANK_n) begin
    // Dynamic Sprites
    /*
    if (hcount[10:5] == (d[5:0]-1) && hcount[4:1] >= 4'b1111 && vcount[9:4] == e[5:0]) begin
//coordinates(10,10) 31
      apple_sprite_addr <= hcount[4:1] - 4'b1111 + (vcount[3:0])*16;
      a <= {apple_sprite_output[15:11], 3'b0};
      b <= { apple_sprite_output[10:5], 2'b0};
      c <= {apple_sprite_output[4:0], 3'b0};
    end else if (hcount[10:5] == d[5:0] && hcount[4:1] < 4'b1111 && vcount[9:4] == e[5:0]) begin
      apple_sprite_addr <=  hcount[4:1] + 4'b0001 + (vcount[3:0])*16;
      a <= {apple_sprite_output[15:11], 3'b0};
      b <= { apple_sprite_output[10:5], 2'b0};
      c <= {apple_sprite_output[4:0], 3'b0};
    end
    */
    // apple - 1
    if (map[hcount[10:5]][vcount[9:4]] == 8'b1) begin
      apple_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
      a <= {apple_sprite_output[15:11], 3'b0};
      b <= {apple_sprite_output[10:5], 2'b0};
      c <= {apple_sprite_output[4:0], 3'b0};
    end
    // head up - 2
    else if (map[hcount[10:5]][vcount[9:4]] == 8'b10) begin
      snake_head_up_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
```

```
  a <= {snake_head_up_sprite_output[15:11], 3'b0};
  b <= {snake_head_up_sprite_output[10:5], 2'b0};
  c <= {snake_head_up_sprite_output[4:0], 3'b0};
end
// head down - 3
else if (map[hcount[10:5]][vcount[9:4]] == 8'b11) begin
  snake_head_down_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {snake_head_down_sprite_output[15:11], 3'b0};
  b <= {snake_head_down_sprite_output[10:5], 2'b0};
  c <= {snake_head_down_sprite_output[4:0], 3'b0};
end
// head left - 4
else if (map[hcount[10:5]][vcount[9:4]] == 8'b100) begin
  snake_head_left_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {snake_head_left_sprite_output[15:11], 3'b0};
  b <= {snake_head_left_sprite_output[10:5], 2'b0};
  c <= {snake_head_left_sprite_output[4:0], 3'b0};
end
// head right - 5
else if (map[hcount[10:5]][vcount[9:4]] == 8'b101) begin
  snake_head_right_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {snake_head_right_sprite_output[15:11], 3'b0};
  b <= {snake_head_right_sprite_output[10:5], 2'b0};
  c <= {snake_head_right_sprite_output[4:0], 3'b0};
end
// body vertical - 6
else if (map[hcount[10:5]][vcount[9:4]] == 8'b110) begin
  snake_body_vertical_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {snake_body_vertical_sprite_output[15:11], 3'b0};
  b <= {snake_body_vertical_sprite_output[10:5], 2'b0};
  c <= {snake_body_vertical_sprite_output[4:0], 3'b0};
end
// body horizontal - 7
else if (map[hcount[10:5]][vcount[9:4]] == 8'b111) begin
  snake_body_horizontal_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {snake_body_horizontal_sprite_output[15:11], 3'b0};
  b <= {snake_body_horizontal_sprite_output[10:5], 2'b0};
  c <= {snake_body_horizontal_sprite_output[4:0], 3'b0};
end
// body top left - 8
else if (map[hcount[10:5]][vcount[9:4]] == 8'b1000) begin
  snake_body_topleft_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {snake_body_topleft_sprite_output[15:11], 3'b0};
  b <= {snake_body_topleft_sprite_output[10:5], 2'b0};
```

```
  c <= {snake_body_topleft_sprite_output[4:0], 3'b0};
end
// body bottom left - 9
else if (map[hcount[10:5]][vcount[9:4]] == 8'b1001) begin
  snake_body_bottomleft_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {snake_body_bottomleft_sprite_output[15:11], 3'b0};
  b <= {snake_body_bottomleft_sprite_output[10:5], 2'b0};
  c <= {snake_body_bottomleft_sprite_output[4:0], 3'b0};
end
// body top right - 10
else if (map[hcount[10:5]][vcount[9:4]] == 8'b1010) begin
  snake_body_topright_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {snake_body_topright_sprite_output[15:11], 3'b0};
  b <= {snake_body_topright_sprite_output[10:5], 2'b0};
  c <= {snake_body_topright_sprite_output[4:0], 3'b0};
end
// body bottom right - 11
else if (map[hcount[10:5]][vcount[9:4]] == 8'b1011) begin
  snake_body_bottomright_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {snake_body_bottomright_sprite_output[15:11], 3'b0};
  b <= {snake_body_bottomright_sprite_output[10:5], 2'b0};
  c <= {snake_body_bottomright_sprite_output[4:0], 3'b0};
end
// tail up - 12
else if (map[hcount[10:5]][vcount[9:4]] == 8'b1100) begin
  snake_tail_up_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {snake_tail_up_sprite_output[15:11], 3'b0};
  b <= {snake_tail_up_sprite_output[10:5], 2'b0};
  c <= {snake_tail_up_sprite_output[4:0], 3'b0};
end
// tail down - 13
else if (map[hcount[10:5]][vcount[9:4]] == 8'b1101) begin
  snake_tail_down_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {snake_tail_down_sprite_output[15:11], 3'b0};
  b <= {snake_tail_down_sprite_output[10:5], 2'b0};
  c <= {snake_tail_down_sprite_output[4:0], 3'b0};
end
// tail left - 14
else if (map[hcount[10:5]][vcount[9:4]] == 8'b1110) begin
  snake_tail_left_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {snake_tail_left_sprite_output[15:11], 3'b0};
  b <= {snake_tail_left_sprite_output[10:5], 2'b0};
  c <= {snake_tail_left_sprite_output[4:0], 3'b0};
end
```

```
// tail right - 15
else if (map[hcount[10:5]][vcount[9:4]] == 8'b1111) begin
  snake_tail_right_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {snake_tail_right_sprite_output[15:11], 3'b0};
  b <= {snake_tail_right_sprite_output[10:5], 2'b0};
  c <= {snake_tail_right_sprite_output[4:0], 3'b0};
end

// zero - 16
else if (map[hcount[10:5]][vcount[9:4]] == 8'b10000) begin
  zero_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {zero_sprite_output[15:11], 3'b0};
  b <= {zero_sprite_output[10:5], 2'b0};
  c <= {zero_sprite_output[4:0], 3'b0};
end

// one - 17
else if (map[hcount[10:5]][vcount[9:4]] == 8'b10001) begin
  one_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {one_sprite_output[15:11], 3'b0};
  b <= {one_sprite_output[10:5], 2'b0};
  c <= {one_sprite_output[4:0], 3'b0};
end

// two - 18
else if (map[hcount[10:5]][vcount[9:4]] == 8'b10010) begin
  two_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {two_sprite_output[15:11], 3'b0};
  b <= {two_sprite_output[10:5], 2'b0};
  c <= {two_sprite_output[4:0], 3'b0};
end
// three - 19
else if (map[hcount[10:5]][vcount[9:4]] == 8'b10011) begin
  three_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {three_sprite_output[15:11], 3'b0};
  b <= {three_sprite_output[10:5], 2'b0};
  c <= {three_sprite_output[4:0], 3'b0};
end
//four - 20
else if (map[hcount[10:5]][vcount[9:4]] == 8'b10100) begin
  four_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
  a <= {four_sprite_output[15:11], 3'b0};
  b <= {four_sprite_output[10:5], 2'b0};
  c <= {four_sprite_output[4:0], 3'b0};
```

```
    end
    // five - 21
    else if (map[hcount[10:5]][vcount[9:4]] == 8'b10101) begin
      five_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
      a <= {five_sprite_output[15:11], 3'b0};
      b <= {five_sprite_output[10:5], 2'b0};
      c <= {five_sprite_output[4:0], 3'b0};
    end
    // six - 22

    else if (map[hcount[10:5]][vcount[9:4]] == 8'b10110) begin
      six_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
      a <= {six_sprite_output[15:11], 3'b0};
      b <= {six_sprite_output[10:5], 2'b0};
      c <= {six_sprite_output[4:0], 3'b0};
    end

    // seven - 23
    else if (map[hcount[10:5]][vcount[9:4]] == 8'b10111) begin
      seven_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
      a <= {seven_sprite_output[15:11], 3'b0};
      b <= {seven_sprite_output[10:5], 2'b0};
      c <= {seven_sprite_output[4:0], 3'b0};
    end
    // eight - 24
    else if (map[hcount[10:5]][vcount[9:4]] == 8'b11000) begin
      eight_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
      a <= {eight_sprite_output[15:11], 3'b0};
      b <= {eight_sprite_output[10:5], 2'b0};
      c <= {eight_sprite_output[4:0], 3'b0};
    end
    // nine - 25
    else if (map[hcount[10:5]][vcount[9:4]] == 8'b11001) begin
      nine_sprite_addr <= hcount[4:1] + (vcount[3:0])*16;
      a <= {nine_sprite_output[15:11], 3'b0};
      b <= {nine_sprite_output[10:5], 2'b0};
      c <= {nine_sprite_output[4:0], 3'b0};
    end
    else if (map[hcount[10:5]][vcount[9:4]] == 8'b0 && hcount[10:6] > 8'b1 && hcount[10:6] <
8'b100110 && vcount[9:4] > 8'b011 && vcount[9:4]< 8'b11100) begin
      a <= 8'h0;
      b <= 8'h0;
      c <= 8'h0;
    end
```

```
      // static sprites
      //left wall column
      else if(hcount[10:6] == 5'b00000 && vcount[9:5] > 5'b00001) begin
        wall_sprite_addr <= hcount[5:1] + (vcount[4:0])*32;
        a <= {wall_sprite_output[15:11], 3'b0};
        b <= {wall_sprite_output[10:5], 2'b0};
        c <= {wall_sprite_output[4:0], 3'b0};
      end
      //right
      else if(hcount[10:6] == 5'b10011 && vcount[9:5] > 5'b00001) begin
        wall_sprite_addr <= hcount[5:1] + (vcount[4:0])*32;
        a <= {wall_sprite_output[15:11], 3'b0};
        b <= {wall_sprite_output[10:5], 2'b0};
        c <= {wall_sprite_output[4:0], 3'b0};
      end
      //top
      else if( vcount[9:5] == 5'b00001) begin
        wall_sprite_addr <= hcount[5:1] + (vcount[4:0])*32;
        a <= {wall_sprite_output[15:11], 3'b0};
        b <= {wall_sprite_output[10:5], 2'b0};
        c <= {wall_sprite_output[4:0], 3'b0};
      end
      //bottom
      else if( vcount[9:5] == 5'b01110) begin
        wall_sprite_addr <= hcount[5:1] + (vcount[4:0])*32;
        a <= {wall_sprite_output[15:11], 3'b0};
        b <= {wall_sprite_output[10:5], 2'b0};
        c <= {wall_sprite_output[4:0], 3'b0};
      end
      else begin
        a <= background_r;
        b <= background_g;
        c <= background_b;
      end
    end
  end
// Assign VGA outputs
assign {VGA_R, VGA_G, VGA_B} = {a, b, c};


endmodule
```

```
//-------------------------------------------------------
// I think this is the original template code or lab3 our solution
/*
always_comb begin
  {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0}; // Initialize to black
  if (VGA_BLANK_n) begin
    if ((hcount[10:1]-(x+20))**2 + (vcount-(y+20))**2 <= 20**2) begin
                    {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00}; // Red color for circle
                  end else begin
                    {VGA_R, VGA_G, VGA_B} = {background_r, background_g, background_b};
                  end
          end
end
*/



module vga_counters(
 input logic           clk50, reset,
 output logic [10:0] hcount,  // hcount[10:1] is pixel column
 output logic [9:0]  vcount,  // vcount[9:0] is pixel row
 output logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0            1279       1599 0
 *           _________________           _________
 * ____________|    Video      |_____________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *       _______________________     _____________
 * |____|       VGA_HS          |____|
 */
   // Parameters for hcount
   parameter HACTIVE      = 11'd 1280,
             HFRONT_PORCH = 11'd 32,
             HSYNC        = 11'd 192,
             HBACK_PORCH  = 11'd 96,
             HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                            HBACK_PORCH; // 1600
```

```
// Parameters for vcount
parameter VACTIVE      = 10'd 480,
          VFRONT_PORCH = 10'd 10,
          VSYNC        = 10'd 2,
          VBACK_PORCH  = 10'd 33,
          VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
                         VBACK_PORCH; // 525

logic endOfLine;

always_ff @(posedge clk50 or posedge reset)
  if (reset)         hcount <= 0;
  else if (endOfLine) hcount <= 0;
  else                hcount <= hcount + 11'd 1;

assign endOfLine = hcount == HTOTAL - 1;

logic endOfField;

always_ff @(posedge clk50 or posedge reset)
  if (reset)         vcount <= 0;
  else if (endOfLine)
    if (endOfField)   vcount <= 0;
    else              vcount <= vcount + 10'd 1;

assign endOfField = vcount == VTOTAL - 1;

// Horizontal sync: from 0x520 to 0x5DF (0x57F)
// 101 0010 0000 to 101 1101 1111
assign VGA_HS = !( (hcount[10:8] == 3'b101) &
                   !(hcount[7:5] == 3'b111));
assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

assign VGA_SYNC_n = 1'b0; // For putting sync on the green signal; unused

// Horizontal active: 0 to 1279     Vertical active: 0 to 479
// 101 0000 0000  1280              01 1110 0000  480
// 110 0011 1111  1599              10 0000 1100  524
assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                     !( vcount[9] | (vcount[8:5] == 4'b1111) );

/* VGA_CLK is 25 MHz
 *
 * clk50    __|  |__|  |__|
```

```
   *
   *          _____       __
   * hcount[0]__|     |_____|
   */
  assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge sensitive

endmodule
```

vga_ball.c

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "vga_ball.h"

#include <asm/io.h>
#include <linux/types.h>

#define DRIVER_NAME "vga_ball"

//int a;

/* Device registers */
#define X(x) (x)
// #define Y(x) ((x)+1)
// #define KEY(x) ((x)+2)


struct vga_ball_dev{
        struct resource res; /* Resource: our registers */
        void __iomem *virtbase; /* Where registers can be accessed in memory */
```

```
        //unsigned long int type
        grid grid;
} dev;


//created write coordinate for all the sprites
static void write_coordinate(grid *grid){
    // Write the data to some register using iowrite64
        printk("%d \n",grid->data);
    iowrite32(grid->data, X(dev.virtbase + grid->offset));
        // dev.data = *data;
        dev.grid = *grid;
}


/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */

//this will write backgrounds for apple and snake sprites
static long vga_ball_ioctl(struct file *f, unsigned int cmd, unsigned long int arg)
{
        vga_ball_arg_t vla;

        switch (cmd) {
        case VGA_BALL_WRITE_COORDINATE:
                if (copy_from_user(&vla, (vga_ball_arg_t *) arg,
                                   sizeof(vga_ball_arg_t)))
                        return -EACCES;
                write_coordinate(&vla.grid);
                break;
        default:
                return -EINVAL;
        }

        return 0;
}


/* The operations our device knows how to do */
static const struct file_operations vga_ball_fops = {
        .owner          = THIS_MODULE,
```

```
	.unlocked_ioctl = vga_ball_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice vga_ball_misc_device = {
	.minor		= MISC_DYNAMIC_MINOR,
	.name		= DRIVER_NAME,
	.fops		= &vga_ball_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_ball_probe(struct platform_device *pdev)
{
	vga_ball_color_t beige = { 0xf9, 0xe4, 0xb7 };
	//vga_ball_color_t beige = { 0xff, 0x00, 0x00 };
	int ret;

	/* Register ourselves as a misc device: creates /dev/vga_ball */
	ret = misc_register(&vga_ball_misc_device);

	/* Get the address of our registers from the device tree */
	ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
	if (ret) {
		ret = -ENOENT;
		goto out_deregister;
	}

	/* Make sure we can use these registers */
	if (request_mem_region(dev.res.start, resource_size(&dev.res),
			       DRIVER_NAME) == NULL) {
		ret = -EBUSY;
		goto out_deregister;
	}

	/* Arrange access to our registers */
	dev.virtbase = of_iomap(pdev->dev.of_node, 0);
	if (dev.virtbase == NULL) {
		ret = -ENOMEM;
		goto out_release_mem_region;
	}
```

```
	return 0;

	out_release_mem_region:
	release_mem_region(dev.res.start, resource_size(&dev.res));
	out_deregister:
	misc_deregister(&vga_ball_misc_device);

	return ret;
}

/* Clean-up code: release resources */
static int vga_ball_remove(struct platform_device *pdev)
{
	iounmap(dev.virtbase);
	release_mem_region(dev.res.start, resource_size(&dev.res));
	misc_deregister(&vga_ball_misc_device);
	return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_ball_of_match[] = {
	{ .compatible = "csee4840,vga_ball-1.0" },
	{},
};
MODULE_DEVICE_TABLE(of, vga_ball_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_ball_driver = {
	.driver  = {
		.name  = DRIVER_NAME,
		.owner = THIS_MODULE,
		.of_match_table = of_match_ptr(vga_ball_of_match),
	},
	.remove	= __exit_p(vga_ball_remove),
};

/* Called when the module is loaded: set things up */
static int __init vga_ball_init(void)
{
	pr_info(DRIVER_NAME ": init\n");
	return platform_driver_probe(&vga_ball_driver, vga_ball_probe);
```

```
}

/* Calball when the module is unloaded: release resources */
static void __exit vga_ball_exit(void)
{
        platform_driver_unregister(&vga_ball_driver);
        pr_info(DRIVER_NAME ": exit\n");
}

module_init(vga_ball_init);
module_exit(vga_ball_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA ball driver");
```

hello.c

```
#include <stdio.h>
#include "vga_ball.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
// we added these libraries
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "sony.h"


/*---------------------------------------- Deque ----------------------------------------------*/

#define MAX_SIZE 1200

typedef struct {
   unsigned short int x_pos;
   unsigned short int y_pos;
   unsigned short int dir;
   unsigned short int map;
```

```
    unsigned short int change_x;
    unsigned short int change_y;
} Map;

typedef struct {
    Map arr[MAX_SIZE];
    int front;
    int rear;
} Deque;

void initializeDeque(Deque* dq) {
    dq->front = -1;
    dq->rear = 0;
}

int isFull(const Deque* dq) {
    return (dq->front == 0 && dq->rear == MAX_SIZE - 1) || (dq->front == dq->rear + 1);
}

int isEmpty(const Deque* dq) {
    return dq->front == -1;
}


void insertFront(Deque* dq, Map pos) {
    if (isFull(dq)) {
        printf("Deque is full. Cannot insert.\n");
        return;
    }
    if (dq->front == -1) {
        dq->front = dq->rear = 0;
    } else if (dq->front == 0) {
        dq->front = MAX_SIZE - 1;
    } else {
        dq->front--;
    }
    dq->arr[dq->front] = pos;
}

void insertRear(Deque* dq, Map pos) {
    if (isFull(dq)) {
        printf("Deque is full. Cannot insert.\n");
        return;
```

```
    }
    if (dq->front == -1) {
        dq->front = dq->rear = 0;
    } else if (dq->rear == MAX_SIZE - 1) {
        dq->rear = 0;
    } else {
        dq->rear++;
    }
    dq->arr[dq->rear] = pos;
}

Map getFront(const Deque* dq) {
    Map frontMap;
    if (isEmpty(dq)) {
        printf("Deque is empty. No front element.\n");
        frontMap.x_pos = frontMap.y_pos = frontMap.dir = frontMap.map = 0; // Default values
    } else {
        frontMap = dq->arr[dq->front];
    }
    return frontMap;
}

Map removeFront(Deque* dq) {
    Map removed;
    if (isEmpty(dq)) {
        printf("Deque is empty. Cannot remove.\n");
        removed.x_pos = removed.y_pos = removed.map = 0; // Default values
        return removed;
    }
    removed = dq->arr[dq->front];
    if (dq->front == dq->rear) {
        dq->front = dq->rear = -1;
    } else if (dq->front == MAX_SIZE - 1) {
        dq->front = 0;
    } else {
        dq->front++;
    }
    return removed;
}
Map removeRear(Deque* dq) {
    Map removed;
    if (isEmpty(dq)) {
        printf("Deque is empty. Cannot remove.\n");
        removed.x_pos = removed.y_pos = removed.map = 0; // Default values
```

```
        return removed;
    }
    removed = dq->arr[dq->rear];
    if (dq->front == dq->rear) {
        dq->front = dq->rear = -1;
    } else if (dq->rear == 0) {
        dq->rear = MAX_SIZE - 1;
    } else {
        dq->rear--;
    }
    return removed;
}

/*---------------------------------------- Hash Map ----------------------------------------------*/
#define NUM_ROWS 30
#define NUM_COLS 40
#define HASHMAP_SIZE (NUM_ROWS * NUM_COLS)

// Define a struct for the key (row, column)
typedef struct {
    int row;
    int col;
} Key;

// Define a struct for the hashmap entry
typedef struct {
    Key key;
    int value;
} Entry;

// Define the hashmap structure
typedef struct {
    Entry *entries[HASHMAP_SIZE];
} HashMap;

// Hash function for the key
int hash(Key key) {
    return (key.row * NUM_COLS + key.col) % HASHMAP_SIZE;
}

// Function to initialize the hashmap
HashMap *createHashMap() {
    HashMap *map = (HashMap *)malloc(sizeof(HashMap));
    for (int i = 0; i < HASHMAP_SIZE; i++) {
```

```
        map->entries[i] = NULL;
    }
    return map;
}

// Function to insert a key-value pair into the hashmap
void insert(HashMap *map, Key key, int value) {
    int index = hash(key);
    Entry *entry = (Entry *)malloc(sizeof(Entry));
    entry->key = key;
    entry->value = value;
    map->entries[index] = entry;
}

// Function to retrieve the value associated with a key from the hashmap
int get(HashMap *map, Key key) {
    int index = hash(key);
    if (map->entries[index] != NULL && map->entries[index]->key.row == key.row &&
map->entries[index]->key.col == key.col) {
        return map->entries[index]->value;
    } else {
        return -1; // Key not found
    }
}

// Function to update the value associated with a key in the hashmap
void update(HashMap *map, Key key, int value) {
    int index = hash(key);
    if (map->entries[index] != NULL && map->entries[index]->key.row == key.row &&
map->entries[index]->key.col == key.col) {
        map->entries[index]->value = value;
    }
}

// Function to initialize the hashmap with all values set to 0
void initializeHashMap(HashMap *map) {
    for (int i = 0; i < NUM_ROWS; i++) {
        for (int j = 0; j < NUM_COLS; j++) {
            Key key = {i, j};
            insert(map, key, 0);
        }
    }
}
```

```
/*---------------------------------- Rest of the code ------------------------------------*/

int direction;
int vga_ball_fd;

pthread_t sony_thread;
void *sony_thread_f(void *);

//set the ball position
void set_ball_coordinate(const grid *grid)
{
   vga_ball_arg_t vla;
   vla.grid = *grid;
   if (ioctl(vga_ball_fd, VGA_BALL_WRITE_COORDINATE, &vla)) {
      perror("ioctl(VGA_BALL_WRITE_COORDINATE) failed");
      return;
   }
}


// Define a structure to hold the arguments
struct ThreadArgs {
   // Define the arguments here
   struct libusb_device_handle *sony;
   uint8_t endpoint_address;
   // Add more arguments as needed
};

// Function to be executed in the new thread
void *sony_thread_f(void *args) {
   // Cast the argument pointer to the correct type
   struct ThreadArgs *threadArgs = (struct ThreadArgs *)args;

   // Now you can use the arguments
   struct libusb_device_handle *sony = threadArgs->sony;
   uint8_t endpoint_address = threadArgs->endpoint_address;
   // Use the arguments as needed

   // Don't forget to free the memory allocated for args if necessary
   struct usb_sony_packet packet;
   int transferred;
   for(;;){
      libusb_interrupt_transfer(sony, endpoint_address,
            (unsigned char *) &packet, sizeof(packet),
```

```
            &transferred, 0);

        if (transferred > 0 && packet.keycode[8] != 0x08 ) {
            //printf("%02x \n", packet.keycode[8]);
            int temp_d = packet.keycode[8];
            if (temp_d == 0x02){
                direction = 1;
            } else if (temp_d == 0x06){
                direction = 2;
            } else if (temp_d == 0x00){
                direction = 3;
            } else if (temp_d == 0x04){
                direction = 4;
            }
            //direction_flag = 1;
        } //else direction_flag = 0;
    }

    return NULL;
}

unsigned long int combine(unsigned short int a, unsigned short int b, unsigned short int c,
unsigned short int d) {
    unsigned long int x = 0;

    // Combine the values using bitwise OR and bit shifting
    x |= ((unsigned long int)a) << 24;
    x |= ((unsigned long int)b) << 16;
    x |= ((unsigned long int)c) << 8;
    x |= (unsigned long int)d;
    //printf("%lu\n", x);
    return x;
}

void clear_Display( vga_ball_arg_t vla){
    int offset = 0;
     for(int r = 0; r < 30; r++, offset+=40){
            for(int c = 0; c < 40; c+=4){
                vla.grid.data = combine(0,0,0,0);
                vla.grid.offset = offset+c;
                set_ball_coordinate(&vla.grid);
            }
     }
}
```

```
Key generate_random_coordinates(int min_x, int max_x, int min_y, int max_y) {
    Key coord;
    coord.col = rand() % (max_x - min_x + 1) + min_x;
    coord.row = rand() % (max_y - min_y + 1) + min_y;
    return coord;
}

int main()
{

    struct ThreadArgs args;
    printf("VGA ball Userspace program started\n");
    // opening and connecting to controller
    uint8_t endpoint_address_temp;
    struct libusb_device_handle *sony_temp;
    if ((sony_temp = opensony(&endpoint_address_temp)) == NULL ) {
        fprintf(stderr, "Did not find sony\n");
        exit(1);
    }
    args.sony = sony_temp;
    args.endpoint_address = endpoint_address_temp;

    // Cast the argument pointer to the correct type
    // pthread_create(&sony_thread, NULL, sony_thread_f, NULL);
    pthread_create(&sony_thread, NULL, sony_thread_f, (void *)&args);
    printf("After pthread create\n");

    static const char filename[] = "/dev/vga_ball";
    if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
        fprintf(stderr, "could not open %s\n", filename);
        return -1;
    }

    // unsigned short int a = 0;
    // unsigned short int b = 0;
    // unsigned short int c = 0;
    // unsigned short int d = 0;

    vga_ball_arg_t vla;

    Deque snake;
    initializeDeque(&snake);
```

```
HashMap *screen_map = createHashMap();
// Initialize the hashmap with all values set to 0
initializeHashMap(screen_map);

// Map: {x_pos, y_pos, direction, spriteType}
// direction:
//      1->right
//      2->left
//      3->up
//      4->down

//basic snake
//right snake head
Map initial_snake = {4, 4, 1, 5};
insertFront(&snake, initial_snake);
// horizontal snake body directed right
Map initial_snake1 = {3, 4, 1, 7};
insertRear(&snake, initial_snake1);
//right snake tail
Map initial_snake2 = {2, 4, 1, 14};
insertRear(&snake, initial_snake2);

Deque change_point;
initializeDeque(&change_point);
Map initial_change_point = {2, 4, 1, 1};
insertRear(&change_point, initial_change_point);

Key apple_coordinate = generate_random_coordinates(3, 37, 2, 27);
update(screen_map, apple_coordinate , 1);

int offset;
while(1){
    usleep(100000);
    while(1){
        //sleep(1);
        Map temp;
        Map temp_head_up;
        Map temp_h_body;
        Map temp_tail_left;
        Map temp_h_body_cp; //idk if this is supposed to be Map
        Map temp_head_down;
        Map temp_turn_1;
        Map temp_head_left;
```

```
        switch (getFront(&snake).map){
            case 2:
                temp_head_up = removeFront(&snake);
                if (direction == 3 || direction == 4){
                    temp_head_up.y_pos -= 1;
                }
                else if (direction == 1){
                    temp_head_up.x_pos += 1;
                    temp_head_up.dir = direction;
                    temp_head_up.map = 5;
                    Map temp_cp = {temp_head_up.x_pos, temp_head_up.y_pos,
temp_head_up.dir, 0};
                    insertRear(&change_point, temp_cp);
                } else if (direction == 2){
                    temp_head_up.x_pos -= 1;
                    temp_head_up.dir = direction;
                    temp_head_up.map = 4;

                    Map temp_cp = {temp_head_up.x_pos, temp_head_up.y_pos,
temp_head_up.dir, 0};
                    insertRear(&change_point, temp_cp);
                }
                if(apple_coordinate.row == temp_head_up.y_pos && apple_coordinate.col ==
temp_head_up.x_pos){
                    apple_coordinate = generate_random_coordinates(3, 37, 2, 27);
                    update(screen_map, apple_coordinate , 1);
                }
                Key coords_head_up = {temp_head_up.x_pos, temp_head_up.y_pos};
                update(screen_map, coords_head_up, temp_head_up.map);
                insertRear(&snake, temp_head_up);
                break;
            case 3:
                temp_head_down = removeFront(&snake);
                if (direction == 3 || direction == 4){
                    temp_head_down.y_pos += 1;
                }
                 else if (direction == 1){
                    temp_head_down.x_pos += 1;
                    temp_head_down.dir = direction;
                    temp_head_down.map = 5;
                    Map temp_cp = {temp_head_down.x_pos, temp_head_down.y_pos,
temp_head_down.dir, 0};
                    insertRear(&change_point, temp_cp);
                } else if (direction == 2){
```

```
                temp_head_down.x_pos -= 1;
                temp_head_down.dir = direction;
                temp_head_down.map = 4;
                Map temp_cp = {temp_head_down.x_pos, temp_head_down.y_pos,
temp_head_down.dir, 0};
                insertRear(&change_point, temp_cp);
            }
            if(apple_coordinate.row == temp_head_down.y_pos && apple_coordinate.col ==
temp_head_down.x_pos){
                apple_coordinate = generate_random_coordinates(3, 37, 2, 27);
                update(screen_map, apple_coordinate , 1);
            }
            Key coords_head_down = {temp_head_down.x_pos, temp_head_down.y_pos};
            update(screen_map, coords_head_down, temp_head_down.map);
            insertRear(&snake, temp_head_down);
            break;

        case 4:
            temp_head_left = removeFront(&snake);
            if (direction == 2){
                temp_head_left.x_pos -= 1;
            }
             else if (direction == 3){
                temp_head_left.y_pos -= 1;
                temp_head_left.dir = direction;
                temp_head_left.map = 2;
                Map temp_cp = {temp_head_left.x_pos, temp_head_left.y_pos,
temp_head_left.dir, 0};
                insertRear(&change_point, temp_cp);
            } else if (direction == 4){
                temp_head_left.y_pos += 1;
                temp_head_left.dir = direction;
                temp_head_left.map = 3;
                Map temp_cp = {temp_head_left.x_pos, temp_head_left.y_pos,
temp_head_left.dir, 0};
                insertRear(&change_point, temp_cp);
            }
            if(apple_coordinate.row == temp_head_left.y_pos && apple_coordinate.col ==
temp_head_up.x_pos){
                apple_coordinate = generate_random_coordinates(3, 37, 2, 27);
                update(screen_map, apple_coordinate , 1);
            }
            Key coords_head_left = {temp_head_left.x_pos, temp_head_left.y_pos};
            update(screen_map, coords_head_left, temp_head_left.map);
```

```
            insertRear(&snake, temp_head_left);
            break;
        case 5:
            temp = removeFront(&snake);
            if (direction == 1){ //right or left
                temp.x_pos += 1;

            } else if (direction == 3){ // up
                Key temp_c = {temp.x_pos, temp.y_pos};
                update(screen_map, temp_c, 8);
                temp.y_pos -= 1;
                temp.dir = direction;
                temp.map = 2;
                //Map new_coords = {temp.x_pos, temp.y_pos, direction, 0};
                //insertRear(&chan ge_point, new_coords);
            }  else if (direction == 4){ // down
                temp.y_pos += 1;
                temp.dir = direction;
                temp.map = 3;
            }
            if(apple_coordinate.row == temp.y_pos && apple_coordinate.col == temp.x_pos){
                apple_coordinate = generate_random_coordinates(3, 37, 2, 27);
                update(screen_map, apple_coordinate , 1);
            }
            Key coords = {temp.x_pos, temp.y_pos};
            // printf("map val for head: %d\n", temp.map);
            // printf("coords val for head: %d, %d\n", coords.col, coords.row);
            update(screen_map, coords, temp.map);
            // printf("mapping for head: %d\n", get(screen_map, coords));
            insertRear(&snake, temp);

            break;
        case 7:
            temp_h_body = removeFront(&snake);
            temp_h_body_cp = getFront(&change_point);
            if (temp_h_body_cp.x_pos == temp_h_body.x_pos && temp_h_body_cp.y_pos ==
temp_h_body.y_pos){
                printf("change_point\n");
                if(temp_h_body_cp.dir == 3 && temp_h_body.dir == 1){
                    Key new_coords = {temp_h_body.x_pos, temp_h_body.y_pos};
                    update(screen_map, new_coords, 11);
                    break;
                }
            }
```

```
    else if (temp_h_body.dir == direction && direction == 1){
        temp_h_body.x_pos += 1;
    } else if (temp_h_body.dir == direction && direction == 2){
        temp_h_body.x_pos -= 1;
    }
    Key coords_h = {temp_h_body.x_pos, temp_h_body.y_pos};
    //printf("map val for b: %d\n", temp_h_body.map);
    //printf("coords val for b: %d, %d\n", coords_h.col, coords_h.row);
    update(screen_map, coords_h, temp_h_body.map);
    //printf("mapping for body: %d\n", get(screen_map, coords_h));
    insertRear(&snake, temp_h_body);
    break;

case 8:
    temp_turn_1 = getFront(&snake);
    Key temp_h = {temp_turn_1.x_pos - 1, temp_turn_1.y_pos};
    int temp_val = get(screen_map, temp_h);
    if (temp_val == 7){
        Key temp_c = {temp_turn_1.x_pos, temp_turn_1.y_pos-1};
        update(screen_map, temp_c, 6);
    }


    break;




case 14:
    temp_tail_left = removeFront(&snake);
    Key temp_c = {temp_tail_left.x_pos, temp_tail_left.y_pos};
    update(screen_map, temp_c, 0);
    if (direction == 1){
        temp_tail_left.x_pos += 1;
    }
    // else if (direction == 3){
    //     temp_tail_left.y_pos -= 1;
    //     temp_tail_left.dir = 3;
    //     temp_tail_left.map = 13;
    // }

    Key coords_t_l = {temp_tail_left.x_pos, temp_tail_left.y_pos};
    //printf("map val for tail: %d\n", temp_tail_left.map);
```

```
            //printf("coords val for tail: %d, %d\n", coords_t_l.col, coords_t_l.row);
            update(screen_map, coords_t_l, temp_tail_left.map);
            //printf("mapping for tail: %d\n", get(screen_map, coords_t_l));
            insertRear(&snake, temp_tail_left);
            goto writeScreen;
            break;

        }
    }

    writeScreen:
    //printf("line 405\n");
    offset = 0;
    //writing the whole screen
    for(int r = 0; r < 30; r++, offset+=40){
        for(int c = 0; c < 40; c+=4){
            //get(map, (Key){0, 0})
            unsigned short int sprite1 = get(screen_map, (Key){c, r});
            unsigned short int sprite2 = get(screen_map, (Key){c+1, r});
            unsigned short int sprite3 = get(screen_map, (Key){c+2, r});
            unsigned short int sprite4 = get(screen_map, (Key){c+3, r});
            // if(sprite1 > 0 ){
            //     printf("sprite1 %d\n", sprite1);
            // }
            // if(sprite2 > 0 ){
            //     printf("sprite2 %d\n", sprite2);
            // }
            // if(sprite3 > 0 ){
            //     printf("sprite3 %d\n", sprite3);
            // }
            // if(sprite4 > 0 ){
            //     printf("sprite4 %d\n", sprite4);
            // }
            vla.grid.data = combine(sprite1,sprite2,sprite3,sprite4);
            vla.grid.offset = offset + c;
            set_ball_coordinate(&vla.grid);
        }

    }
}


// clear_Display(vla); //clear the display independently rather than depending on a for loop
//initalize snake body and apple
```

```
 /*
   for(int r = 0; r < 30; r++, offset+=40){
      for(int c = 0; c < 40; c+=4){
         if(r == 15 && c == 12){
            vla.grid.data = combine(0,14,7,5); // Snake head_right and tail_left placed of the first
two columns of the corresponding row
            vla.grid.offset = offset+c;
            set_ball_coordinate(&vla.grid);
         } else if (r == 15 && c == 16){
            vla.grid.data = combine(0,0,0,1); // Snake head_right and tail_left placed of the first
two columns of the corresponding row
            vla.grid.offset = offset+c;
            set_ball_coordinate(&vla.grid);
         }
         else {
            vla.grid.data = combine(0,0,0,0); // Snake head_right and tail_left placed of the first
two columns of the corresponding row
            vla.grid.offset = offset+c;
            set_ball_coordinate(&vla.grid);
         }
      }
   }*/

   return 0;

}
```

vga_ball.h

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>

// int offset;
// this will set the coordinates for any sprites
typedef struct {
  unsigned long int data;
  unsigned short int offset;
} grid;
```

```
typedef struct {
        unsigned char red, green, blue;
  //vga_ball_coordinate coordinate;
} vga_ball_color_t;

typedef struct {
  // unsigned long int data
  grid grid;
  // unsigned short int offset;
} vga_ball_arg_t;


#define VGA_BALL_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_BALL_WRITE_COORDINATE _IOW(VGA_BALL_MAGIC, 1, vga_ball_arg_t *)
// #define VGA_HEAD_UP_WRITE_COORDINATE _IOW(VGA_BALL_MAGIC, 2,
vga_ball_arg_t *)
// #define VGA_FRUIT_WRITE_COORDINATE _IOW(VGA_BALL_MAGIC, 3, vga_ball_arg_t *)


#endif
```