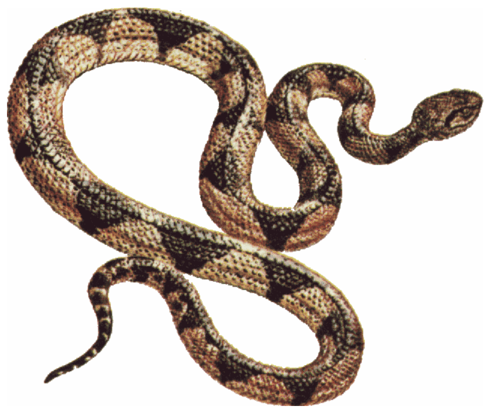
SLITHER



Written By: Aaron Kammula and Jackie Ke

Course: ECE241

Due Date: Monday December 5th, 2016.

Part 1: Introduction

As stated in lecture, the final project for Digital Systems (ECE241) was a comprehensive assessment of our knowledge and skill in designing and implementing hardware with Verilog. Therefore, to show what we have learned from the start of the semester, we have decided to set our focus on the following goals of this project:

- Create a popular game out of hardware that many people can relate to, but with various additional features such as a booster and a random dot generator
- Build on the fundamentals of hardware design by adding at least one of the following: RAM, VGA Controllers, and implementing an interactive mouse pad.
- Incorporate and implement as much as we can out of hardware design from previous labs, lectures and online resources.
- Implement as much knowledge on game design learned from other courses or past experiences as possible.

For this project to initiate any further, an abundant amount of motivation was needed to ensure that this project was on the right track. Therefore, to overcome this burden, we have decided to build up upon a famous game that many craved to play or think about when they had the chance to do so; whether it be during break, lunch, school or sleep time. Additionally, another source of motivation arose from the idea of how we could frame hardware in an unusual manner. In other words, this project could've potentially been a way to move away from the usual method of controlling hardware (I.E: Keys, LEDR's and Switches). Therefore, what motivated us was the fact that we could take a very popular game, and control it with hardware in a way that's simple and interactive.

Part 2: The Design

2.1: Top Level Block Diagram

Description of the blocks

PS2 Mouse Interface

Used to get data from mouse, and receive a signal when data is available (mouse has moved).

Initialize Dots

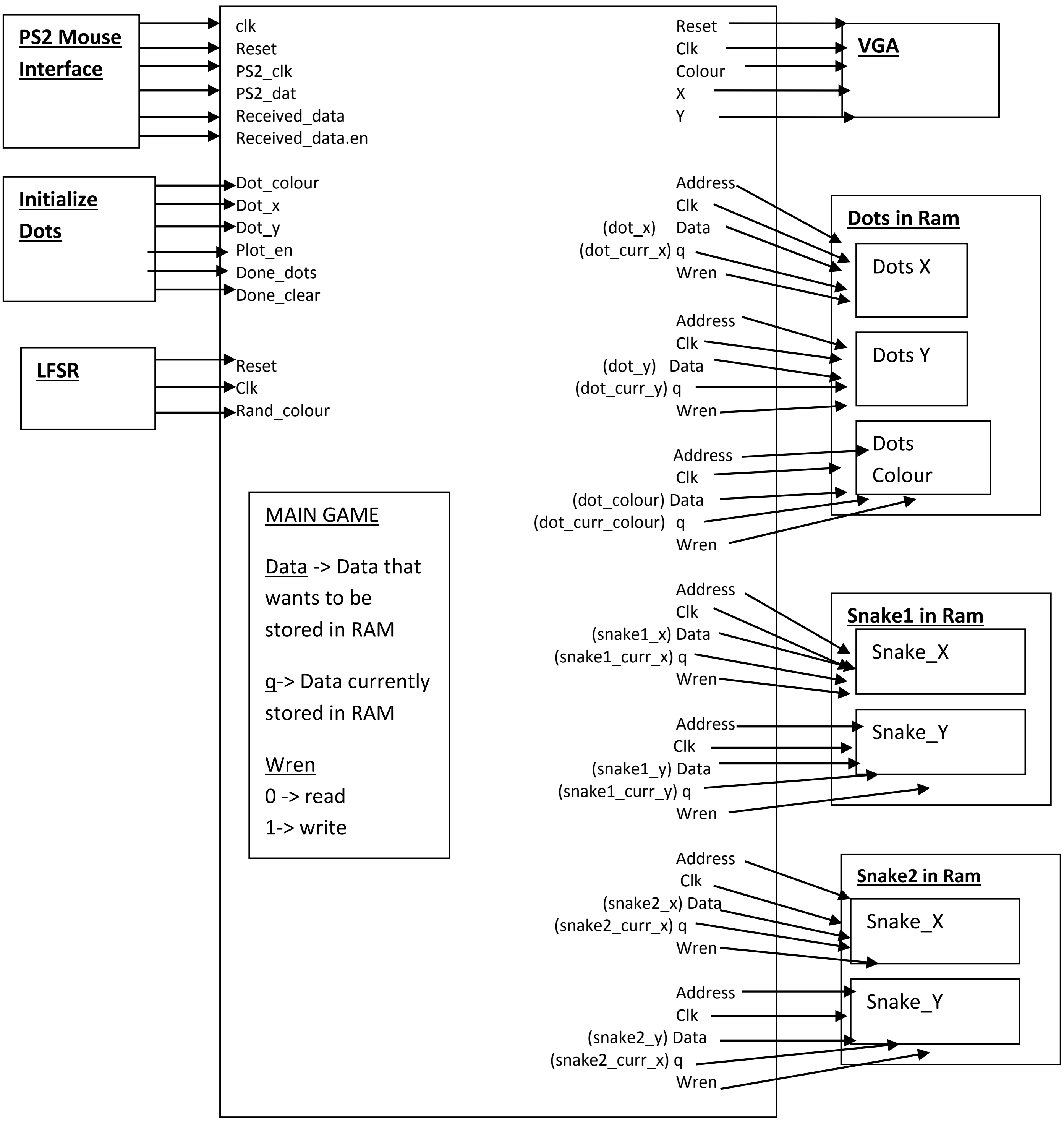
First, the screen is cleared with a black background, then random x,y and colour coordinates are created. A signal is sent when the screen is completely cleared, and another signal is sent after the 100 dots has been received.

LFSR

This is used to generate random colours, x and y coordinates for the dots. When using the boost feature of this game, the LFSR produces random colours when moving the snake around.

MAIN GAME

This module cycles through each and every state, determines when to draw or erase, read or write to RAM, and to update the position for snakes and mouse cursor.


PS2 Mouse Interface
Initialize Dots
LFSR
clk
Reset
PS2_clk
PS2_dat
Received_data
Received_data.en
Dot_colour
Dot_x
Dot_y
Plot_en
Done_dots
Done_clear
Reset
Clk
Rand_colour
MAIN GAME
Data -> Data that wants to be stored in RAM
q-> Data currently stored in RAM
Wren
0 -> read
1-> write
Reset
Clk
Colour
X
Y
VGA
Dots in Ram
Dots X
Dots Y
Dots Colour
Address
Clk
(dot_x) Data
(dot_curr_x) q
Wren
Address
Clk
(dot_y) Data
(dot_curr_y) q
Wren
Address
Clk
(dot_colour) Data
(dot_curr_colour) q
Wren
Snake1 in Ram
Snake_X
Snake_Y
Address
Clk
(snake1_x) Data
(snake1_curr_x) q
Wren
Address
Clk
(snake1_y) Data
(snake1_curr_y) q
Wren
Snake2 in Ram
Snake_X
Snake_Y
Address
Clk
(snake2_x) Data
(snake2_curr_x) q
Wren
Address
Clk
(snake2_y) Data
(snake2_curr_x) q
Wren

2.2: “Main Game” Block Schematics

Description of 2.2 Parts

Reset
When KEY[0] is pressed, all data and counters get initialized.

Initialize Dots
The screen is first cleared. The dots with random x, y and colour, are prepared with the use of an LFSR and stored in RAM. A signal is also sent when screen is cleared and when 100 dots are reached.

Store Update
This part stores the initialized length and position of snakes into RAM.

Plot Dots
For this part of the diagram, the dots are first read from RAM, then displayed on to the screen with the use of the VGA module. This part of the diagram will end once the counter hits 100.

Plot Snakes
Similar to the Plot Dots part of the diagram, data of the snake will be read from Ram, then displayed on the screen with the use of a VGA module.

Plot Mouse
The mouse first gets disabled if new data has been received, and the previous data of the mouse gets erased. Afterwards, the coordinates of the mouse gets updated, the cursor gets redrawn with respect to the mouse, and the mouse gets re-enabled.

Wait Time
This wait time allows users to see everything on the screen. If the mouse data has been received meanwhile, it updates the mouse. Boost effect of the snakes also happen here by setting the wait time counter smal.

Change Snake
After this state, all the tasks will be done to the other snake.

Erase Snake from Screen
This state is similar to Plot Snake, except the output color to VGA is always black.

Update Head of Snake
This changes the coordinates of the head with input directions.

Check Collision with Dots
To check for collisions with dots, the coordinates of the dots are first read from RAM, then the coordinates of the snake head and the dots are compared. This will keep going until all of the dots are compared with each snake.

Increase Length of Snake
If a collision between a dot and the head of a snake is observed, then the size counter of the snake gets positively altered.

Check Collision with Snake

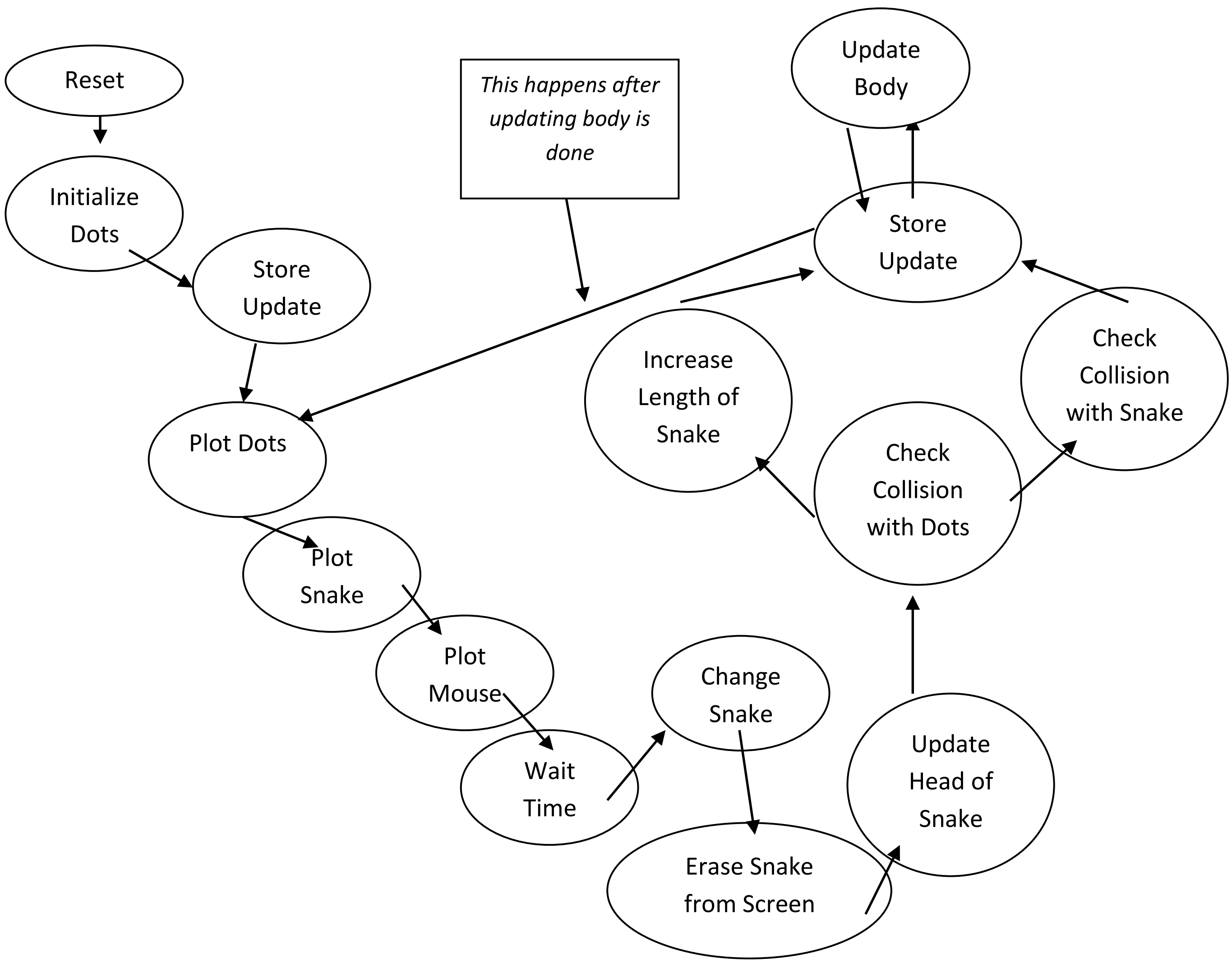
To check for collisions, the coordinates of the other snake are compared with the head of the current snake. A counter is used to traverse through each pixel of the other snake. If any coordinates between the two snakes are equal, then a collision is present, and game over screen appears.

Store Update

If a collision between a dot or a snake has happened, then the information stored within RAM gets changed either for the dot or the snake. If two snakes collide, then the entire memory for the screen gets replaced with a big, white screen. However, if a snake collides with a dot, then the memory gets changed in a way that makes the snake store more pixels, while changing the colour of the dot to a black colour.

Update Body

This part of the diagram is responsible for updating the body of the snake. Their coordinates basically equal to the coordinates of the previous unit. Afterwards, the diagram goes back to plotting dots.



Part 3: The Successes

Overall, with the exception of a fully functional collision module, we have succeeded in reaching all the milestones presented to the TA. As a result, we have come up with a game with the following capabilities: the ability to generate random dots, control multiple snakes with one interface, detect when a snake should grow or fail to move with the use of a collision detector, and control the snakes with a mouse.

However, behind each success, failures are always prevalent within the time span of this entire project. The major problems found within this project arose from the following: Improper resets, and collisions.

To begin, when creating the random dot generator with the use of an LFSR, the bits to produce the random x, y and colour coordinates were not shifting unless if the user pressed KEY[0], which was intended to reset the game in this case. As a result, when the game is started, an additional press of KEY[0] was required to get the game working. A possible reason for a problem like this could arise from one of the conditional statements of the FSM.

Additionally, the collision module would work on the first press of the reset button. However, after the white screen (which is also the “Game Over” screen) appeared after collision, the collisions between on snake and the other didn't seem to work afterwards. A possible reason for a problem like this would arise from the fact that the same method of determining collisions between snakes and dots, and snakes and snakes, was used. In other words, this could be a possible problem because by using the same modular method for both a static and dynamic form of objects, there could be some complications with how the memory was stored for the snakes as the static dots seemed to collide as planned with the snake`s head. Therefore, another plan for detecting the snake`s position on the screen would probably be needed. Another possible reason for this problem would arise from the fact that when the user pressed ``Reset``, the process of storing information in memory is interrupted, so there is some data stored or cleared at times when it is not necessary. In brief, there could’ve possibly been problems with how each FSM interacted with each other after each collision, or how memory was stored and manipulated throughout the entire program.

Part 4: Future Improvements

If we had the chance to start this project from scratch, we would make separate modules for the snakes and the collisions, and then put them into various .v files. That way, the project would be a lot more organized, and the failures found within this entire project would’ve been much easier to debug. Additionally, instead of focusing to just get the project done, each partner should add more parts that they could definitely get done from the start and finish that off as soon as possible. That way, the project will have more features that will help it stand out from other projects. Moreover, another change that we would make is to use as less counters as possible, and try to connect as many states together as possible to maximize the efficiency of the code in attempts to prepare us for finishing future programming projects as quickly and as efficient as possible. Lastly, another change that would`ve been implemented into this project would be to spend more time on how the extensions would be connected. However, if there is no success coming along, then, after three days, the focus would be more on the functionality of the game. For example, instead of spending more than a week on connecting the mouse into the game, the main focus would've been shifted on getting the game to work perfectly with switches or any DE1-SOC connection. That way, even if the mouse extensions weren't ready, the game would still work perfectly fine. Furthermore, by getting extensions or additional connections right before the due date, there was a higher probability of getting nothing or a product that did not work at all. By making the above mistake, the game was harder to display as a multiplayer game. Therefore, less fun would come out of the game. In brief, whenever frustration is faced, moving on and building up on another part of the project is a much better strategy than moving head forward and forgetting all of the other components that need to be built upon for the entire project.

APPENDIX

Part 1: Main Game Code (ram and VGA files are not included)
Part 2: The Plot Dots Module
Part 3: The PS2/Y Mouse Module

Main Game code

```
module FinalProject    (
        CLOCK_50, KEY, LEDR, SW,
        PS2_CLK, PS2_DAT, //PS2_CLK2, PS2_DAT2,

        // The ports below are for the VGA output.  Do not change.
        VGA_CLK,                        //    VGA Clock
        VGA_HS,                         //    VGA H_SYNC
        VGA_VS,                         //    VGA V_SYNC
        VGA_BLANK_N,                    //    VGA BLANK
        VGA_SYNC_N,                     //    VGA SYNC
        VGA_R,                          //    VGA Red[9:0]
        VGA_G,                          //    VGA Green[9:0]
        VGA_B                           //    VGA Blue[9:0]
    );

    input    CLOCK_50;
    input [3:0] KEY;

    input [2:0] SW;

    // Bidirectionals
    inout              PS2_CLK;
    inout              PS2_DAT;
//    inout              PS2_CLK2;
//    inout              PS2_DAT2;
//

    output [9:0] LEDR;

    // Do not change the following outputs
    output           VGA_CLK;                 //    VGA Clock
    output           VGA_HS;                  //    VGA H_SYNC
    output           VGA_VS;                  //    VGA V_SYNC
    output           VGA_BLANK_N;             //    VGA BLANK
    output           VGA_SYNC_N;              //    VGA SYNC
    output    [9:0]    VGA_R;                 //    VGA Red[9:0]
    output    [9:0]    VGA_G;                 //    VGA Green[9:0]
    output    [9:0]    VGA_B;                 //    VGA Blue[9:0]


    /* ***************************************************** */

    wire resetn;
    assign resetn = KEY[0];

    wire clk;
    assign clk = CLOCK_50;

    wire boost;
```

```
assign boost = ~KEY[3];

wire boost2;
assign boost2 = ~KEY[1];

//wire [2:0] direction;
//assign direction = SW[2:0];
reg [7:0] plot_x;
reg [6:0] plot_y;



/* ***************************************************** */
//random colour generator
wire [2:0] rand_colour;

LFSR3 color1 (clk, rand_colour, resetn, 3'b100);


/* ***************************************************** */

reg [7:0] plot_VGA_x;
reg [6:0] plot_VGA_Y;
reg [2:0] plot_VGA_colour;
reg plot_VGA_en;

vga_adapter VGA(
	.resetn(KEY[0]),
	.clock(clk),
	.colour(plot_VGA_colour),
	.x(plot_VGA_x),
	.y(plot_VGA_Y),
	.plot(plot_VGA_en),

	.VGA_R(VGA_R),
	.VGA_G(VGA_G),
	.VGA_B(VGA_B),
	.VGA_HS(VGA_HS),
	.VGA_VS(VGA_VS),
	.VGA_BLANK(VGA_BLANK_N),
	.VGA_SYNC(VGA_SYNC_N),
	.VGA_CLK(VGA_CLK));
   defparam VGA.RESOLUTION = "160x120";
   defparam VGA.MONOCHROME = "FALSE";
   defparam VGA.BITS_PER_COLOUR_CHANNEL = 1;
   defparam VGA.BACKGROUND_IMAGE = "black.mif";


/* ***************************************************** */

wire [7:0] dot_x;
```

```
wire [6:0] dot_y;
wire [2:0] dot_colour;
wire dot_plot_en, dots_done, doneclear;
assign LEDR[0] = dots_done;
assign LEDR[1] = plot_VGA_en;

plotDots    plotDot_i (
        .clk(clk),
        .resetn(resetn),
        .plot_colour(dot_colour),
        .plot_x(dot_x),
        .plot_y(dot_y),
        .plot_en(dot_plot_en),
        .done_ploting_init_dots(dots_done),
        .doneclear(doneclear)
    );



/* ***************************************************** */
//MOUSE1
wire        [7:0]    ps2_key_data1;
wire                 ps2_key_pressed1;
reg [1:0] counter1;

reg [7:0] mouse1_x;
reg [6:0] mouse1_y;
reg [7:0] l3, l2, l1;
reg get_enable1, reset_counter1;

PS2_Controller PS2_1 (
    // Inputs
    .CLOCK_50               (clk),
    .reset              (~KEY[0]),

    // Bidirectionals
    .PS2_CLK           (PS2_CLK),
    .PS2_DAT           (PS2_DAT),

    // Outputs
    .received_data        (ps2_key_data1),
    .received_data_en    (ps2_key_pressed1)
);


always @(posedge CLOCK_50)
begin
    if (resetn == 1'b0) begin
        l3 <= 8'h00;
        counter1 <= 2'b00;
        //get_enable1 <= 1'b1;
```

```
        end

        else begin

            //if (ps2_key_pressed1 && get_enable1) begin
            if (ps2_key_pressed1 && get_enable1 && (counter1 != 2'b11) ) begin
                l3 <= ps2_key_data1;
                l2 <= l3;
                l1 <= l2;
                /*
                if (counter1 == 2'b10) begin
                    //get_enable1 <= 1'b0;
                    counter1 <= 2'b00;
                end
                else counter1 <= counter1 + 1'b1;
                */
                counter1 <= counter1 + 1'b1;
            end

            if (reset_counter1) counter1 <= 2'b00;
            //else get_enable1 <= 1'b1;
        end
    end

    /* ***************************************************** */
    //MOUSE 2
//
//   wire        [7:0]    ps2_key_data2;
//   wire                 ps2_key_pressed2;
//   reg [1:0] counter2;
//
//   reg [7:0] mouse2_x;
//   reg [6:0] mouse2_y;
//   reg [7:0] u3, u2, u1;
//   reg get_enable2, reset_counter2;
//
//   PS2_Controller PS2_1 (
//       // Inputs
//       .CLOCK_50              (clk),
//       .reset              (~KEY[0]),
//
//       // Bidirectionals
//       .PS2_CLK           (PS2_CLK2),
//       .PS2_DAT           (PS2_DAT2),
//
//       // Outputs
//       .received_data       (ps2_key_data2),
//       .received_data_en    (ps2_key_pressed2)
//   );
//
//
```

```
//    always @(posedge CLOCK_50)
//    begin
//        if (resetn == 1'b0) begin
//            u3 <= 8'h00;
//            counter2 <= 2'b00;
//            //get_enable1 <= 1'b1;
//        end
//
//        else begin
//
//            //if (ps2_key_pressed1 && get_enable1) begin
//            if (ps2_key_pressed2 && get_enable2 && (counter2 != 2'b11) ) begin
//                u3 <= ps2_key_data2;
//                u2 <= u3;
//                u1 <= u2;
//                /*
//                if (counter1 == 2'b10) begin
//                    //get_enable1 <= 1'b0;
//                    counter1 <= 2'b00;
//                end
//                else counter1 <= counter1 + 1'b1;
//                */
//                counter2 <= counter2 + 1'b1;
//            end
//
//            if (reset_counter2) counter2 <= 2'b00;
//            //else get_enable1 <= 1'b1;
//        end
//    end
//



    /* ****************************************************** */


    reg dots_to_mem;
    wire [7:0] dot_cur_x;
    wire [7:0] dot_cur_y;
    wire [7:0] dot_cur_colour;
    reg [6:0] counterDot;
    reg [2:0] dot_out_colour;

    reg data_wait_counter1;

    dotsram dotx(
            .address(counterDot),
            .clock(clk),
            .data(dot_x),
            .wren(dots_to_mem),
```

```
		.q(dot_cur_x)
		);

	dotsram doty(
		.address(counterDot),
		.clock(clk),
		.data({1'b0,dot_y}),
		.wren(dots_to_mem),
		.q(dot_cur_y)
		);

	dotsram dotcolour(
		.address(counterDot),
		.clock(clk),
		.data({5'b00000,dot_out_colour}),
		.wren(dots_to_mem),
		.q(dot_cur_colour)
		);


	/* ***************************************************** */
	reg snake_counter;
	reg [3:0] speed_difference;

	reg [3:0] frame_counter; //counts each frame
	reg [19:0] second_counter; //counts 1/60 of a second

	reg [1:0] data_wait_counter2;


	//SNAKE 1
	wire [7:0]  snake1_cur_x,  snake1_cur_y;
	//wire s_draw, s_erase, s_update_head, s_increase_length, s_update_body, write_to_mem,
s_store;
	reg [8:0] snake1_memory_counter1, snake1_total_data;
	reg snake1_to_mem;
	reg snake1_done;

	reg [3:0] speed;

	reg [7:0] snake1_out_x, snake1_out_y;
	reg [7:0] snake1_prev_x, snake1_prev_y;

	reg [2:0] last_direc;
	reg [2:0] direction;
	reg [2:0] snake1_colour;

	ram512x8 getx1(
		.address(snake1_memory_counter1),
		.clock(clk),
		.data(snake1_out_x),
```

```
	.wren(snake1_to_mem),
	.q(snake1_cur_x)
	);

ram512x8 gety1(
	.address(snake1_memory_counter1),
	.clock(clk),
	.data(snake1_out_y),
	.wren(snake1_to_mem),
	.q(snake1_cur_y)
	);

/* ***************************************************** */
//SNAKE 2
wire [7:0]  snake2_cur_x,  snake2_cur_y;
reg [8:0] snake2_memory_counter2, snake2_total_data;
reg snake2_to_mem;
reg snake2_done;

reg [3:0] speed2;

reg [7:0] snake2_out_x, snake2_out_y;
reg [7:0] snake2_prev_x, snake2_prev_y;

reg [2:0] last_direc2;
wire [2:0] direction2;
assign direction2 = SW[2:0];
reg [2:0] snake2_colour;

ram512x8 getx2(
	.address(snake2_memory_counter2),
	.clock(clk),
	.data(snake2_out_x),
	.wren(snake2_to_mem),
	.q(snake2_cur_x)
	);

ram512x8 gety2(
	.address(snake2_memory_counter2),
	.clock(clk),
	.data(snake2_out_y),
	.wren(snake2_to_mem),
	.q(snake2_cur_y)
	);


/* ***************************************************** */
//states
reg [7:0] state;
localparam	PLOT_DOTS = 8'd16,
		INIT_DOTS = 8'd17,
```

```
DOT_DATA_wait = 8'd18,
DOTS_SCREEN_WAIT = 8'd19,
WAIT = 8'd20,

draw = 8'd0,
d_wait = 8'd1,
dprint = 8'd14,
draw_reset_counter = 8'd11,//resetting the memory counter1 and determines the speed of the snake

enable_mouse = 8'd25,
erase_mouse = 8'd26,
update_mouse = 8'd27,
draw_mouse = 8'd28,
disable_mouse =8'd29,

enable_mouse2 = 8'd30,
erase_mouse2 = 8'd31,
update_mouse2 = 8'd37,
draw_mouse2 = 8'd33,
disable_mouse2 =8'd34,

wait_time = 8'd2,
change_snake = 8'd35,

erase = 8'd3,
e_wait = 8'd4,
eprint = 8'd15,
erase_reset_counter = 8'd12,

get_direction = 8'd5,
update_head = 8'd6,

food_data_wait = 8'd21,
check_food = 8'd22,
next_food = 8'd23,
increase_length = 8'd7,

othersnake_data_wait =8'd36,
check_snake = 8'd37,
next_snake_coord = 8'd38,

update_wait = 8'd13,
update_body = 8'd8,

store_wait = 8'd24,
store_update = 8'd9,
u_wait = 8'd10,
game_over = 8'd39;//max
```

```
/* ***************************************************** */
reg temp;

assign LEDR[9] = temp;

/* ***************************************************** */

always@(posedge clk)
begin

if (!resetn) begin //Case reset
   state <= INIT_DOTS;
   plot_VGA_x <= 0;
   plot_VGA_Y <= 0;
   plot_VGA_colour <= 0;
   plot_VGA_en <= 0;

   dots_to_mem <= 1'b0;
   counterDot <= 7'b1111111; //make it 0 the first time
   snake_counter <= 1'b0;

   data_wait_counter1 <= 1'b1;
   data_wait_counter2 <= 2'b10;

   snake1_to_mem <= 1'b0;
   snake1_memory_counter1 <= 9'b000000000;
   snake1_out_x <= 8'b00000010;
   snake1_out_y <= 8'b00000010;
   snake1_total_data <= 9'b000000001;
   snake1_colour <= 3'b011;
   last_direc <= 3'b000;
   direction <= 3'b000;
   snake1_done <= 1'b0;

   snake1_to_mem <= 1'b1;
   snake2_memory_counter2 <= 9'b000000000;
   snake2_out_x <= 8'b01111111;
   snake2_out_y <= 8'b01111100;
   snake2_total_data <= 9'b000000001;
   snake2_colour <= 3'b011;
   last_direc2 <= 3'b111;
   //direction2 <= 3'b000;
   snake2_done <= 1'b0;

   mouse1_x <= 8'b01010000;
   mouse1_y <= 7'b0111100;
   get_enable1 <= 1'b0;
   reset_counter1 <= 1'b0;

   speed_difference <= 4'b0000;
```

```
    plot_x <= 8'b00000000;
    plot_y <=0;

    temp <= 1'b0;
end


else begin  //Moving around states

    case(state)

        INIT_DOTS:
        begin
            if (dots_done) begin
                state <= store_update;
                dots_to_mem <= 1'b0;
                counterDot <= 0;
                plot_VGA_en <= 1'b0;
            end

            else
            begin
                if (doneclear) begin
                    dots_to_mem <= 1'b1;
                    counterDot <= counterDot + 1'b1;
                end

                state <= INIT_DOTS;
                plot_VGA_x <= dot_x;
                plot_VGA_Y <= dot_y;
                plot_VGA_colour <= dot_colour;
                dot_out_colour <= dot_colour;
                plot_VGA_en <= dot_plot_en;
            end
        end



        /* ****************************************************** */
        //draw dots

        //wait for data from memory
        DOT_DATA_wait:
        begin
            temp <= 1'b0;

            if (data_wait_counter1 == 1'b0) begin
                state <= PLOT_DOTS;
                data_wait_counter1 <= 1'b1;
            end
```

```
    else begin
        plot_VGA_en <= 1'b0;
        state <= DOT_DATA_wait;
        data_wait_counter1 <= data_wait_counter1 -1'b1;
    end
end

//send to VGA
PLOT_DOTS:
begin
    plot_VGA_x <= dot_cur_x;
    plot_VGA_Y <= dot_cur_y[6:0];
    plot_VGA_colour <= dot_cur_colour[2:0];
    plot_VGA_en <= 1'b1;
    state <= DOTS_SCREEN_WAIT;
end

//VGA has received data, plot to screen
DOTS_SCREEN_WAIT:
begin
    plot_VGA_en <= 1'b0;

    if(counterDot==7'b1111111) begin
        state <= d_wait;
        counterDot <= 0;
    end

    else begin
        state <= DOT_DATA_wait;
        counterDot <= counterDot + 1'b1;
    end

end

/* ***************************************************** */
//snake 1 draw

d_wait:
begin
    if (data_wait_counter1 == 1'b0) begin
        state <= draw;
        data_wait_counter1 <= 1'b1;
    end

    else begin
        plot_VGA_en <= 1'b0;
        state <= d_wait;
        data_wait_counter1 <= data_wait_counter1 -1'b1;
    end
end
```

```
draw:
begin
   if (snake_counter == 1'b0) begin
      plot_VGA_x <= snake1_cur_x;
      plot_VGA_Y <= snake1_cur_y[6:0];
      plot_VGA_colour <= boost? rand_colour : snake1_colour;
      if (boost) snake1_colour <= (rand_colour == 3'b000) ? 3'b011 :rand_colour;
   end

   else begin
      plot_VGA_x <= snake2_cur_x;
      plot_VGA_Y <= snake2_cur_y[6:0];
      plot_VGA_colour <= boost2? rand_colour : snake2_colour;
      if (boost2) snake2_colour <= (rand_colour == 3'b000) ? 3'b101 :rand_colour;
   end

   state <= dprint;

   plot_VGA_en <= 1'b1;

end

dprint:
begin
   plot_VGA_en <= 1'b0;

   if (snake_counter == 1'b0) begin
      if( snake1_memory_counter1 == (snake1_total_data-1'b1) ) begin
         state <= draw_reset_counter;
         snake1_memory_counter1 <= 0;
      end

      else begin
         state <= d_wait;
         snake1_memory_counter1 <= snake1_memory_counter1 + 1'b1;
      end
   end

   else begin
      if( snake2_memory_counter2 == (snake2_total_data-1'b1) ) begin
         state <= draw_reset_counter;
         snake2_memory_counter2 <= 0;
      end

      else begin
         state <= d_wait;
         snake2_memory_counter2 <= snake2_memory_counter2 + 1'b1;
      end
   end
```

```
end

draw_reset_counter:
begin
   plot_VGA_en <= 1'b0;

   if (snake_counter == 1'b0) begin
      snake1_memory_counter1 <= 9'b000000000;
      speed <= (boost) ? 4'b0001 : 4'b1111;
   end

   else begin
      snake2_memory_counter2 <= 9'b000000000;
      speed2 <= (boost2) ? 4'b0001 : 4'b1111;
   end

   state <= enable_mouse;

   second_counter <= 20'b11001011011100110101; //50 million divide by 60 == 833333
   frame_counter <= 4'b0000;
end

/* ****************************************************** */
//Mouse 1

enable_mouse: begin
   //if ( (counter1 == 2'b11) && ps2_key_pressed1) begin
   if (counter1 == 2'b11) begin
      state <= erase_mouse;
      get_enable1 <= 1'b0;
      reset_counter1 <= 1'b1;
   end

   else begin
      state <= wait_time;
      get_enable1 <= 1'b1;
   end
end

erase_mouse: begin
   state <= update_mouse;

   reset_counter1 <= 1'b0;
   plot_VGA_x <= mouse1_x;
   plot_VGA_Y <= mouse1_y;
   plot_VGA_colour <= 3'b000;
   plot_VGA_en <= 1'b1;
end

update_mouse: begin
```

```
            state <= draw_mouse;

            if (l2 == 8'b0) mouse1_x <= mouse1_x;
            else begin
                if (l1[4]) mouse1_x <= mouse1_x - 1'b1;// - (~l2 ) - 1'b1;
                else mouse1_x <= mouse1_x + 1'b1;//l2;
            end

            if (l3 == 8'b0) mouse1_y <= mouse1_y;
            else begin
                if (!l1[5]) mouse1_y <= mouse1_y - 1'b1;//(~l3[6:0]) - 1'b1;
                else mouse1_y <= mouse1_y + 1'b1;// l3[6:0];
            end

            plot_VGA_en <= 1'b0;
        end

        draw_mouse: begin
            state <= disable_mouse;
            plot_VGA_x <= mouse1_x;
            plot_VGA_Y <= mouse1_y;
            plot_VGA_colour <= 3'b011;
            plot_VGA_en <= 1'b1;
        end

        disable_mouse: begin
            state <= wait_time;
            plot_VGA_en <= 1'b0;

            get_enable1 <= 1'b1;
        end

        /* ***************************************************** */
        //Mouse 2
//
//          enable_mouse2: begin
//              //if ( (counter1 == 2'b11) && ps2_key_pressed1) begin
//              if (counter2 == 2'b11) begin
//                  state <= erase_mouse2;
//                  get_enable2 <= 1'b0;
//                  reset_counter2 <= 1'b1;
//              end
//
//              else begin
//                  state <= wait_time;
//                  get_enable2 <= 1'b1;
//              end
//          end
//
//          erase_mouse2: begin
//              state <= update_mouse2;
```

```
//
//          reset_counter2 <= 1'b0;
//          plot_VGA_x <= mouse2_x;
//          plot_VGA_Y <= mouse2_y;
//          plot_VGA_colour <= 3'b000;
//          plot_VGA_en <= 1'b1;
//       end
//
//       update_mouse2: begin
//          state <= draw_mouse2;
//
//          if (u2 == 8'b0) mouse2_x <= mouse2_x;
//          else begin
//             if (u1[4]) mouse2_x <= mouse2_x - 1'b1;//- (~l2 ) - 1'b1;
//             else mouse2_x <= mouse2_x + 1'b1; //l2;
//          end
//
//          if (u3 == 8'b0) mouse2_y <= mouse2_y;
//          else begin
//             if (!l2[5]) mouse2_y <= mouse2_y - 1'b1;//(~l3[6:0]) - 1'b1;
//             else mouse2_y <= mouse2_y +  1'b1; //l3[6:0];
//          end
//
//          plot_VGA_en <= 1'b0;
//       end
//
//       draw_mouse2: begin
//          state <= disable_mouse2;
//          plot_VGA_x <= mouse2_x;
//          plot_VGA_Y <= mouse2_y;
//          plot_VGA_colour <= 3'b011;
//          plot_VGA_en <= 1'b1;
//       end
//
//       disable_mouse2: begin
//          state <= wait_time;
//          plot_VGA_en <= 1'b0;
//
//          get_enable2 <= 1'b1;
//       end
//

      /* ***************************************************** */



      wait_time:
      begin
         if (frame_counter == speed || frame_counter == speed2) begin
            state <=  change_snake;
            frame_counter <= 4'b0000;
```

```
            second_counter <= 20'b00001011011100110101;
         end

         else begin
            if ( second_counter == 20'b01100101101110011010) //- 2'b11
*(frame_counter/2'b11) == 1'b0)
               state <= (counter1 == 2'b11)? enable_mouse: wait_time; //mouse counter data
            else state <= wait_time;

            //1/60 of a second reached
            if (second_counter == {20{1'b0}}) begin
               frame_counter <= frame_counter + 1'b1;
               second_counter <= 20'b00001011011100110101; //changed
b11001011011100110101
            end

            else begin
               second_counter <= second_counter - 1'b1;//{ {19{1'b0}}, 1'b1};
            end

         end

      end


      //boost_check:
//          if ( ((speed2 != 4'b1111) ^ (speed != 4'b1111)) && (speed_difference != 4'b1111) )
begin
//             speed_difference <= speed_difference + 1'b1;
//          end
//
//
//          else begin
//             speed_difference <= 4'b0000;
//             snake_counter <= snake_counter + 1'b1;
//          end
//          snake_counter <= snake_counter + 1'b1;

      change_snake: begin
         if (snake_counter == 1'b0) begin
            if (snake1_done) begin
               state <= DOT_DATA_wait;
               snake_counter <= 1'b1;
               snake1_done <= 1'b0;
            end

            else begin
               state <= e_wait;
               snake_counter<= 1'b0;
            end
         end
```

```
    else begin
        if (snake2_done) begin
            state <= DOT_DATA_wait;
            snake_counter <= 1'b0;
            snake2_done <= 1'b0;
        end

        else begin
            state <= e_wait;
            snake_counter<= 1'b1;
        end
    end

end


/* ****************************************************** */
//snake erase

e_wait:
begin
    if (data_wait_counter1 == 1'b0) begin
        state <=  erase;
        data_wait_counter1 <= 1'b1;
    end

    else begin
        plot_VGA_en <= 1'b0;
        state <=  e_wait;
        data_wait_counter1 <= data_wait_counter1 -1'b1;
    end

end

erase:
begin
    if (snake_counter == 1'b0) begin
        plot_VGA_x <= snake1_cur_x;
        plot_VGA_Y <= snake1_cur_y[6:0];
        plot_VGA_colour <= 3'b000;
    end

    else begin
        plot_VGA_x <= snake2_cur_x;
        plot_VGA_Y <= snake2_cur_y[6:0];
        plot_VGA_colour <= 3'b000;
    end

    state <= eprint;
```

```
    plot_VGA_en <= 1'b1;
end


eprint:
begin
    plot_VGA_en <= 1'b0;

    if (snake_counter == 1'b0) begin
        if( snake1_memory_counter1 == (snake1_total_data-1'b1) ) begin
            state <= erase_reset_counter;
            snake1_memory_counter1 <= 0;
        end

        else begin
            state <= e_wait;
            snake1_memory_counter1 <= snake1_memory_counter1 + 1'b1;
        end
    end

    else begin
        if( snake2_memory_counter2 == (snake2_total_data-1'b1) ) begin
            state <= erase_reset_counter;
            snake2_memory_counter2 <= 0;
        end

        else begin
            state <= e_wait;
            snake2_memory_counter2 <= snake2_memory_counter2 + 1'b1;
        end
    end
end

erase_reset_counter:
begin
    if (data_wait_counter2 == 2'b00) begin
        state <= get_direction;
        data_wait_counter2 <= 2'b10;
    end

    else begin
        state <= erase_reset_counter;
        plot_VGA_en <= 1'b0;
        data_wait_counter2 <= data_wait_counter2 - 1'b1;

        if (snake_counter == 1'b0) snake1_memory_counter1 <= 9'b000000000;
        else snake2_memory_counter2 <= 9'b000000000;
    end
end

/* ****************************************************** */
```

```
//snake head update

get_direction:
begin
   state <= update_head;

   if ( (mouse1_y == snake1_cur_y) && (mouse1_x > snake1_cur_x) ) direction <=
3'b000;
   if ( (mouse1_y == snake1_cur_y) && (mouse1_x < snake1_cur_x) ) direction <=
3'b001;
   if ( (mouse1_y < snake1_cur_y) && (mouse1_x == snake1_cur_x) ) direction <=
3'b010;
   if ( (mouse1_y > snake1_cur_y) && (mouse1_x == snake1_cur_x) ) direction <=
3'b011;
   if ( (mouse1_y > snake1_cur_y) && (mouse1_x < snake1_cur_x) ) direction <=
3'b100;
   if ( (mouse1_y < snake1_cur_y) && (mouse1_x > snake1_cur_x) ) direction <=
3'b101;
   if ( (mouse1_y > snake1_cur_y) && (mouse1_x > snake1_cur_x) ) direction <=
3'b110;
   if ( (mouse1_y < snake1_cur_y) && (mouse1_x < snake1_cur_x) ) direction <=
3'b111;

end


update_head:
begin
   state <= food_data_wait;
   //food? increase_length :store_update;

   if (snake_counter == 1'b0) begin

      snake1_prev_x <= snake1_cur_x;
      snake1_prev_y <= snake1_cur_y;

      //direction is the opposite as the last_direction
      //we will ignore the current direction b/c snake cant go backward
      if ( (direction[1] == last_direc[1]) && (direction[2] == last_direc[2]) ) begin

         case (last_direc)
            3'b000: begin  //right
               snake1_out_x <= snake1_cur_x + 1'b1;
               snake1_out_y <= snake1_cur_y;
            end
            3'b001: begin  //left
               snake1_out_x <= snake1_cur_x - 1'b1;
               snake1_out_y <= snake1_cur_y;
            end
            3'b010: begin //up
               snake1_out_y <= snake1_cur_y - 1'b1;
```

```
            snake1_out_x <= snake1_cur_x;
        end
        3'b011: begin //down
            snake1_out_y <= snake1_cur_y + 1'b1;
            snake1_out_x <= snake1_cur_x;
        end

        3'b100: begin
            snake1_out_x <= snake1_cur_x - 1'b1;
            snake1_out_y <= snake1_cur_y + 1'b1;
        end

        3'b101: begin
            snake1_out_x <= snake1_cur_x + 1'b1;
            snake1_out_y <= snake1_cur_y - 1'b1;
        end
        3'b110: begin
            snake1_out_y <= snake1_cur_y + 1'b1;
            snake1_out_x <= snake1_cur_x + 1'b1;
        end
        3'b111: begin
            snake1_out_y <= snake1_cur_y - 1'b1;
            snake1_out_x <= snake1_cur_x - 1'b1;
        end

        default: begin
            snake1_out_x <= snake1_cur_x + 1'b1;
            snake1_out_y <= snake1_cur_y;
        end
    endcase

    //last_direction stays the same
    last_direc <= last_direc;
end

//next direction is valid
else begin
    case (direction)
        3'b000: begin  //right
            snake1_out_x <= snake1_cur_x + 1'b1;
            snake1_out_y <= snake1_cur_y;
        end
        3'b001: begin  //left
            snake1_out_x <= snake1_cur_x - 1'b1;
            snake1_out_y <= snake1_cur_y;
        end
        3'b010: begin //up
            snake1_out_y <= snake1_cur_y - 1'b1;
            snake1_out_x <= snake1_cur_x;
        end
        3'b011: begin //down
```

```
            snake1_out_y <= snake1_cur_y + 1'b1;
            snake1_out_x <= snake1_cur_x;
         end

         3'b100: begin
            snake1_out_x <= snake1_cur_x - 1'b1;
            snake1_out_y <= snake1_cur_y + 1'b1;
         end

         3'b101: begin
            snake1_out_x <= snake1_cur_x + 1'b1;
            snake1_out_y <= snake1_cur_y - 1'b1;
         end
         3'b110: begin
            snake1_out_y <= snake1_cur_y + 1'b1;
            snake1_out_x <= snake1_cur_x + 1'b1;
         end
         3'b111: begin
            snake1_out_y <= snake1_cur_y - 1'b1;
            snake1_out_x <= snake1_cur_x - 1'b1;
         end

         default: begin
            snake1_out_x <= snake1_cur_x + 1'b1;
            snake1_out_y <= snake1_cur_y;
         end
      endcase

      last_direc <= direction;
   end
end


else begin
   snake2_prev_x <= snake2_cur_x;
   snake2_prev_y <= snake2_cur_y;

   //direction is the opposite as the last_direction
   //we will ignore the current direction b/c snake cant go backward
   if ( (direction2[1] == last_direc2[1]) && (direction2[2] == last_direc2[2]) ) begin

      case (last_direc2)
         3'b000: begin  //right
            snake2_out_x <= snake2_cur_x + 1'b1;
            snake2_out_y <= snake2_cur_y;
         end
         3'b001: begin  //left
            snake2_out_x <= snake2_cur_x - 1'b1;
            snake2_out_y <= snake2_cur_y;
         end
         3'b010: begin //up
```

```
            snake2_out_y <= snake2_cur_y - 1'b1;
            snake2_out_x <= snake2_cur_x;
        end
        3'b011: begin //down
            snake2_out_y <= snake2_cur_y + 1'b1;
            snake2_out_x <= snake2_cur_x;
        end

        3'b100: begin
            snake2_out_x <= snake2_cur_x - 1'b1;
            snake2_out_y <= snake2_cur_y + 1'b1;
        end

        3'b101: begin
            snake2_out_x <= snake2_cur_x + 1'b1;
            snake2_out_y <= snake2_cur_y - 1'b1;
        end
        3'b110: begin
            snake2_out_y <= snake2_cur_y + 1'b1;
            snake2_out_x <= snake2_cur_x + 1'b1;
        end
        3'b111: begin
            snake2_out_y <= snake2_cur_y - 1'b1;
            snake2_out_x <= snake2_cur_x - 1'b1;
        end

        default: begin
            snake2_out_x <= snake2_cur_x + 1'b1;
            snake2_out_y <= snake2_cur_y;
        end
    endcase

    //last_direction stays the same
    last_direc2 <= last_direc2;
end

//next direction is valid
else begin
    case (direction2)
        3'b000: begin  //right
            snake2_out_x <= snake2_cur_x + 1'b1;
            snake2_out_y <= snake2_cur_y;
        end
        3'b001: begin  //left
            snake2_out_x <= snake2_cur_x - 1'b1;
            snake2_out_y <= snake2_cur_y;
        end
        3'b010: begin //up
            snake2_out_y <= snake2_cur_y - 1'b1;
            snake2_out_x <= snake2_cur_x;
        end
```

```
            3'b011: begin //down
               snake2_out_y <= snake2_cur_y + 1'b1;
               snake2_out_x <= snake2_cur_x;
            end

            3'b100: begin
               snake2_out_x <= snake2_cur_x - 1'b1;
               snake2_out_y <= snake2_cur_y + 1'b1;
            end

            3'b101: begin
               snake2_out_x <= snake2_cur_x + 1'b1;
               snake2_out_y <= snake2_cur_y - 1'b1;
            end
            3'b110: begin
               snake2_out_y <= snake2_cur_y + 1'b1;
               snake2_out_x <= snake2_cur_x + 1'b1;
            end
            3'b111: begin
               snake2_out_y <= snake2_cur_y - 1'b1;
               snake2_out_x <= snake2_cur_x - 1'b1;
            end

            default: begin
               snake2_out_x <= snake2_cur_x + 1'b1;
               snake2_out_y <= snake2_cur_y;
            end
         endcase

         last_direc2 <= direction2;
      end

   end

end

/* ****************************************************** */
//CHECK FOOD
food_data_wait:
begin
   if (data_wait_counter1 == 1'b0) begin
      state <= check_food;
      data_wait_counter1 <= 1'b1;
   end

   else begin
      state <= food_data_wait;
      data_wait_counter1 <= data_wait_counter1 -1'b1;
   end
end
```

```
check_food:
begin
   if (snake_counter == 1'b0) begin
      if ( (snake1_out_x == dot_cur_x) &&    (snake1_out_y == dot_cur_y) &&
(dot_cur_colour[2:0] != 3'b000) ) begin
         state <= increase_length;
         dot_out_colour <= 3'b000;
         dots_to_mem <= 1'b1;
      end

      else begin
         state <= next_food;
      end
   end

   else begin
      if ( (snake2_out_x == dot_cur_x) &&    (snake2_out_y == dot_cur_y) &&
(dot_cur_colour[2:0] != 3'b000) ) begin
         state <= increase_length;
         dot_out_colour <= 3'b000;
         dots_to_mem <= 1'b1;
      end

      else begin
         state <= next_food;
      end
   end
end


next_food:
begin
   if(counterDot==7'b1111111) begin
      //change here to check other collisions
      state <= othersnake_data_wait;
      counterDot <= 0;
   end

   else begin
      state <= food_data_wait;
      counterDot <= counterDot + 1'b1;
   end
end

/* ****************************************************** */
//check snake
othersnake_data_wait:
begin
   if (data_wait_counter1 == 1'b0) begin
```

```
        state <= check_snake;
        data_wait_counter1 <= 1'b1;
    end

    else begin
        state <= othersnake_data_wait;
        data_wait_counter1 <= data_wait_counter1 -1'b1;
    end
end


check_snake:
begin
    if (snake_counter == 1'b0) begin
        if ( (snake1_out_x == snake2_cur_x) &&    (snake1_out_y == snake2_cur_y) )
            state <= game_over;

        else
            state <= next_snake_coord;
    end

    else begin
        if ( (snake2_out_x == snake1_cur_x) &&    (snake2_out_y == snake1_cur_y) )
            state <= game_over;

        else begin
            state <= next_snake_coord;
        end
    end
end


next_snake_coord:
begin
//flip cuz check other snake
    if (snake_counter == 1'b1) begin
        if( snake1_memory_counter1 == (snake1_total_data-1'b1) ) begin
            state <= store_update;
            snake1_memory_counter1 <= 0;
        end

        else begin
            state <= othersnake_data_wait;
            snake1_memory_counter1 <= snake1_memory_counter1 + 1'b1;
        end
    end

    else begin
        if( snake2_memory_counter2 == (snake2_total_data-1'b1) ) begin
            state <= store_update;
```

```
            snake2_memory_counter2 <= 0;
        end

        else begin
            state <= othersnake_data_wait;
            snake2_memory_counter2 <= snake2_memory_counter2 + 1'b1;
        end
    end
end




/* ****************************************************** */


increase_length:
begin
    state <= store_update;

    dots_to_mem <= 1'b0;
    counterDot <= 0;

    if (snake_counter == 1'b0) snake1_total_data <= snake1_total_data + 1'b1;
    else snake2_total_data <= snake2_total_data + 1'b1;
end


/* ****************************************************** */
//snake body update

update_wait:
begin
    if (data_wait_counter1 == 1'b0) begin
        state <= update_body;
        data_wait_counter1 <= 1'b1;
    end

    else begin
        state <= update_wait;
        data_wait_counter1 <= data_wait_counter1 - 1'b1;
    end

end

update_body: begin
    state <= store_update;

    if (snake_counter == 1'b0) begin
        snake1_prev_x <= snake1_cur_x;
        snake1_prev_y <= snake1_cur_y;
```

```
        snake1_out_x <= snake1_prev_x; //store the next unit's x,y in out_x, out_y
        snake1_out_y <= snake1_prev_y; //so we can assign them next_state
    end

    else begin
        snake2_prev_x <= snake2_cur_x;
        snake2_prev_y <= snake2_cur_y;
        snake2_out_x <= snake2_prev_x;
        snake2_out_y <= snake2_prev_y;
    end
end


/* ***************************************************** */
//store update

store_update: begin
    state <= store_wait;

    if (snake_counter == 1'b0) begin
        snake1_to_mem <= 1'b1;
        snake1_out_x <= snake1_out_x;
        snake1_out_y <= snake1_out_y;
    end

    else begin
        snake2_to_mem <= 1'b1;
        snake2_out_x <= snake2_out_x;
        snake2_out_y <= snake2_out_y;
    end
end

store_wait: begin
    if (snake_counter == 1'b0) begin
        if (snake1_memory_counter1 == (snake1_total_data-1'b1)) begin
            state <= u_wait;
        end

        else begin
            state <= update_wait;
            snake1_memory_counter1 <= snake1_memory_counter1 + 1'b1;
        end

        snake1_to_mem <= 1'b0;
    end

    else begin
        if (snake2_memory_counter2 == (snake2_total_data-1'b1)) begin
            state <= u_wait;
        end
```

```
        else begin
            state <= update_wait;
            snake2_memory_counter2 <= snake2_memory_counter2 + 1'b1;
        end

        snake2_to_mem <= 1'b0;
    end
end

u_wait: begin
    if (data_wait_counter2 == 2'b00) begin
        state <= WAIT;
        data_wait_counter2 <= 2'b10;
    end

    else begin
        state <= u_wait;
        data_wait_counter2 <= data_wait_counter2 - 1'b1;

        if (snake_counter == 1'b0) snake1_memory_counter1 <= 0;
        else snake2_memory_counter2 <= 0;
    end
end

/* ***************************************************** */


WAIT:
begin
    state <= DOT_DATA_wait;
    temp <= 1'b1;

    if (snake_counter == 1'b0) snake1_done <= 1'b1;
    else snake2_done <= 1'b1;

end


game_over:
begin
    if (plot_x == 8'b11111111 && plot_y == 7'b1111111) begin
        state <= game_over;
        plot_VGA_en <= 1'b0;
    end

    else begin
        if (plot_x == 8'b11111111) begin
            plot_y <= plot_y + 1'b1;
            plot_x <= 8'b0;
        end
        else begin
```

```
                plot_x <= plot_x + 1'b1;
            end

            state <= game_over;

            plot_VGA_x <= plot_x;
            plot_VGA_Y <= plot_y;
            plot_VGA_colour <= 3'b111;
            plot_VGA_en <= 1'b1;
        end

    end


    default: state <= INIT_DOTS;
    endcase
end


end


endmodule
```

Plot dots module (for initializing the dots after reset)

```
// Part 2 skeleton

module plotDots (
    clk,  resetn, plot_colour,
    plot_x,
    plot_y,
    plot_en,
    done_ploting_init_dots,
    doneclear

);



input clk;
input resetn;
output plot_en;
output [7:0] plot_x;
output [6:0] plot_y;
output [2:0] plot_colour;
output done_ploting_init_dots;
output doneclear;
wire [2:0] rand_colour;
```

```
wire [6:0] counterDot;
reg [7:0] counterAsync = 8'b00110110;
wire [7:0] offset_x;
wire [6:0] offset_y;
always @ (posedge clk)//Counter
begin
    //if (!resetn) counterAsync <= 8'b00110110;
    //else
    counterAsync <= counterAsync + 1'b1;
end  // Coutner
LFSR8 lfsrX(.clk(clk), .shift(offset_x), .resetn(resetn), .counterAsync(counterAsync));
LFSR7 lfsrY(clk, offset_y, resetn, counterAsync[6:0]);
LFSR3 lsfrCOLOUR (clk, rand_colour, resetn, counterAsync[2:0]);
//assign LEDR[0] = done_xy_ctrl;
xy_plot_control xy_plot_controller (
 .clk(clk),
 .resetn(resetn),
 //.move_next(KEY[1]),
 //.set_x(SW[7:0]),
 .rand_colour(rand_colour),
 //.set_y(SW[6:0]),
 .offset_x(offset_x),
 .offset_y(offset_y),
 .plot_x(plot_x),
 .plot_y(plot_y),
 .plot_en(plot_en),
 .plot_colour(plot_colour),
 .counterDot(counterDot),
 .done(done_ploting_init_dots),
 .doneclear(doneclear)
);

endmodule




module LFSR8 (clk, shift, resetn, counterAsync);
input clk;
input [7:0]counterAsync;
input resetn;
output reg [7:0] shift = 7'd1;
always@(posedge clk) begin
    if (!resetn) begin
    shift <= counterAsync;
    end

    else begin
     shift <= shift<<1;
     shift[0] <= (shift[1] ^ shift[2]) ^ (shift[3] ^ shift[7]);
```

```
    end

end
endmodule




module LFSR7 (clk,shift, resetn, counterAsync);
input clk;
input [6:0]counterAsync;
input resetn;
output reg [6:0] shift= 6'd1;
always@(posedge clk) begin
 if (!resetn) begin
shift <= counterAsync;
end
else begin
 shift <= shift<<1;
 shift[0] <= (shift[1] ^ shift[2]) ^ (shift[3] ^ shift[6]);
 end
end
endmodule




module LFSR3 (clk,shift, resetn, counterAsync);
input clk;
input [2:0]counterAsync;
input resetn;
output reg [2:0] shift = 2'd1;
always@(posedge clk) begin
if (!resetn) begin
shift <= counterAsync;
end
else begin
 shift <= shift<<1;
 shift[0] <= (shift[1] ^ shift[2]);
 end
end
endmodule




module xy_plot_control (clk, resetn,  rand_colour,  offset_x,  offset_y, plot_x, plot_y, plot_colour,
plot_en, counterDot, done, doneclear);
```

```
input clk, resetn;
//input [7:0] set_x;
//input [6:0] set_y;
input [7:0] offset_x;
input [6:0] offset_y;
input [2:0] rand_colour;
// Output signals to VGA controller
output reg plot_en;
output reg [7:0] plot_x;
output reg [6:0] plot_y;
output reg [2:0] plot_colour;
output reg [6:0] counterDot;
output reg done, doneclear;
// States
reg [3:0] state;
localparam [3:0] INIT = 4'd0;
localparam[3:0] PLOT = 4'd1;
localparam [3:0] DONE = 4'd2;
localparam [3:0] CLEAR_VGA_MEM = 4'd3;
// Internal variables DELETE???
reg [7:0] x;
reg [6:0] y;
reg [2:0] set_colour;
// State machine
always@(posedge clk) begin
 if (!resetn) begin //Case reset
  state <= INIT;
  done <= 0;
 end

 else begin  //Moving around states

  case(state)

   INIT:
    begin
     plot_x <= 0;
     plot_y <= 0;
     plot_en <= 0;
     state <= CLEAR_VGA_MEM;
        counterDot <= 0;
        doneclear <= 1'b0;
//        write_to_mem <= 1'b0;
   end


    CLEAR_VGA_MEM:
```

```
     begin
         if (plot_x == 8'b11111111 && plot_y == 7'b1111111) begin
                state <= PLOT;
                plot_colour <= 0;
                plot_en <= 1;
                doneclear <= 1'b1;
            end else begin
                if (plot_x == 8'b11111111) begin
                    plot_y <= plot_y + 1'b1;
                    plot_x <= 8'b0;
                end
                else begin
                    plot_x <= plot_x + 1'b1;
                end
                state <= CLEAR_VGA_MEM;
                plot_colour <= 0;
                plot_en <= 1;
            end

     end


   PLOT:
    begin
     //if (move_next||counterDot==7'b1111111) begin
       if(counterDot==7'b1111111) begin
      //state <= INIT;  //If counter reaches 100 or w.e, the game has to start. Also, store each dot
in memory.
      plot_en <= 0;
       state <= DONE;
       counterDot <= 0;

     end

       else begin
      plot_x <= offset_x;
      plot_y <= offset_y;
      plot_en <= 1;
       counterDot <= counterDot+1'b1;
      plot_colour <= rand_colour;
      state <= PLOT;

     end
    end


    DONE: begin
      plot_en <= 0;
       done <= 1;
    end
```

```
    default:
     begin
     //state <= INIT;
        state <= INIT;
     plot_x <= 0;
     plot_y <= 0;
     plot_en <= 0;
     end

  endcase
 end
end

endmodule
```

PS2 mouse controller
(obtained from http://www.Computer-Engineering.org
Author: Adam Chapweske)

```
module PS2_Controller #(parameter INITIALIZE_MOUSE = 1) (
    // Inputs
    CLOCK_50,
    reset,

    the_command,
    send_command,

    // Bidirectionals
    PS2_CLK,                    // PS2 Clock
    PS2_DAT,                    // PS2 Data

    // Outputs
    command_was_sent,
    error_communication_timed_out,

    received_data,
    received_data_en            // If 1 - new data has been received
);
```

```
/***************************************************************************
*                        Parameter Declarations                          *
***************************************************************************/


/***************************************************************************
*                        Port Declarations                               *
***************************************************************************/
// Inputs
input           CLOCK_50;
input           reset;

input   [7:0]   the_command;
input           send_command;

// Bidirectionals
inout           PS2_CLK;
inout           PS2_DAT;

// Outputs
output          command_was_sent;
output          error_communication_timed_out;

output  [7:0]   received_data;
output          received_data_en;

wire [7:0] the_command_w;
wire send_command_w, command_was_sent_w, error_communication_timed_out_w;

generate
   if(INITIALIZE_MOUSE) begin
      assign the_command_w = init_done ? the_command : 8'hf4;
      assign send_command_w = init_done ? send_command : (!command_was_sent_w &&
!error_communication_timed_out_w);
      assign command_was_sent = init_done ? command_was_sent_w : 0;
      assign error_communication_timed_out = init_done ? error_communication_timed_out_w :
1;

      reg init_done;

      always @(posedge CLOCK_50)
         if(reset) init_done <= 0;
         else if(command_was_sent_w) init_done <= 1;

   end else begin
      assign the_command_w = the_command;
      assign send_command_w = send_command;
      assign command_was_sent = command_was_sent_w;
      assign error_communication_timed_out = error_communication_timed_out_w;
   end
```

```
endgenerate

/*****************************************************************************
*                   Constant Declarations                    *
*****************************************************************************/
// states
localparam    PS2_STATE_0_IDLE          = 3'h0,
          PS2_STATE_1_DATA_IN          = 3'h1,
          PS2_STATE_2_COMMAND_OUT       = 3'h2,
          PS2_STATE_3_END_TRANSFER    = 3'h3,
          PS2_STATE_4_END_DELAYED       = 3'h4;

/*****************************************************************************
*             Internal wires and registers Declarations              *
*****************************************************************************/
// Internal Wires
wire          ps2_clk_posedge;
wire          ps2_clk_negedge;

wire          start_receiving_data;
wire          wait_for_incoming_data;

// Internal Registers
reg      [7:0]    idle_counter;

reg           ps2_clk_reg;
reg           ps2_data_reg;
reg           last_ps2_clk;

// State Machine Registers
reg      [2:0]    ns_ps2_transceiver;
reg      [2:0]    s_ps2_transceiver;

/*****************************************************************************
*                   Finite State Machine(s)                  *
*****************************************************************************/

always @(posedge CLOCK_50)
begin
   if (reset == 1'b1)
      s_ps2_transceiver <= PS2_STATE_0_IDLE;
   else
      s_ps2_transceiver <= ns_ps2_transceiver;
end

always @(*)
begin
   // Defaults
   ns_ps2_transceiver = PS2_STATE_0_IDLE;
```

```
 case (s_ps2_transceiver)
  PS2_STATE_0_IDLE:
     begin
        if ((idle_counter == 8'hFF) &&
              (send_command == 1'b1))
           ns_ps2_transceiver = PS2_STATE_2_COMMAND_OUT;
        else if ((ps2_data_reg == 1'b0) && (ps2_clk_posedge == 1'b1))
           ns_ps2_transceiver = PS2_STATE_1_DATA_IN;
        else
           ns_ps2_transceiver = PS2_STATE_0_IDLE;
     end
  PS2_STATE_1_DATA_IN:
     begin
        if ((received_data_en == 1'b1)/* && (ps2_clk_posedge == 1'b1)*/)
           ns_ps2_transceiver = PS2_STATE_0_IDLE;
        else
           ns_ps2_transceiver = PS2_STATE_1_DATA_IN;
     end
  PS2_STATE_2_COMMAND_OUT:
     begin
        if ((command_was_sent == 1'b1) ||
           (error_communication_timed_out == 1'b1))
           ns_ps2_transceiver = PS2_STATE_3_END_TRANSFER;
        else
           ns_ps2_transceiver = PS2_STATE_2_COMMAND_OUT;
     end
  PS2_STATE_3_END_TRANSFER:
     begin
        if (send_command == 1'b0)
           ns_ps2_transceiver = PS2_STATE_0_IDLE;
        else if ((ps2_data_reg == 1'b0) && (ps2_clk_posedge == 1'b1))
           ns_ps2_transceiver = PS2_STATE_4_END_DELAYED;
        else
           ns_ps2_transceiver = PS2_STATE_3_END_TRANSFER;
     end
  PS2_STATE_4_END_DELAYED:
     begin
        if (received_data_en == 1'b1)
        begin
           if (send_command == 1'b0)
              ns_ps2_transceiver = PS2_STATE_0_IDLE;
           else
              ns_ps2_transceiver = PS2_STATE_3_END_TRANSFER;
        end
        else
           ns_ps2_transceiver = PS2_STATE_4_END_DELAYED;
     end
  default:
        ns_ps2_transceiver = PS2_STATE_0_IDLE;
  endcase
end
```

```
/*****************************************************************************
 *                             Sequential logic                              *
 *****************************************************************************/

always @(posedge CLOCK_50)
begin
    if (reset == 1'b1)
    begin
        last_ps2_clk    <= 1'b1;
        ps2_clk_reg     <= 1'b1;

        ps2_data_reg    <= 1'b1;
    end
    else
    begin
        last_ps2_clk    <= ps2_clk_reg;
        ps2_clk_reg     <= PS2_CLK;

        ps2_data_reg    <= PS2_DAT;
    end
end

always @(posedge CLOCK_50)
begin
    if (reset == 1'b1)
        idle_counter <= 6'h00;
    else if ((s_ps2_transceiver == PS2_STATE_0_IDLE) &&
            (idle_counter != 8'hFF))
        idle_counter <= idle_counter + 6'h01;
    else if (s_ps2_transceiver != PS2_STATE_0_IDLE)
        idle_counter <= 6'h00;
end

/*****************************************************************************
 *                            Combinational logic                            *
 *****************************************************************************/

assign ps2_clk_posedge =
            ((ps2_clk_reg == 1'b1) && (last_ps2_clk == 1'b0)) ? 1'b1 : 1'b0;
assign ps2_clk_negedge =
            ((ps2_clk_reg == 1'b0) && (last_ps2_clk == 1'b1)) ? 1'b1 : 1'b0;

assign start_receiving_data     = (s_ps2_transceiver == PS2_STATE_1_DATA_IN);
assign wait_for_incoming_data   =
            (s_ps2_transceiver == PS2_STATE_3_END_TRANSFER);

/*****************************************************************************
 *                             Internal Modules                              *
 *****************************************************************************/
```

```
Altera_UP_PS2_Data_In PS2_Data_In (
	// Inputs
	.clk						(CLOCK_50),
	.reset						(reset),

	.wait_for_incoming_data		(wait_for_incoming_data),
	.start_receiving_data		(start_receiving_data),

	.ps2_clk_posedge			(ps2_clk_posedge),
	.ps2_clk_negedge			(ps2_clk_negedge),
	.ps2_data					(ps2_data_reg),

	// Bidirectionals

	// Outputs
	.received_data				(received_data),
	.received_data_en			(received_data_en)
);

Altera_UP_PS2_Command_Out PS2_Command_Out (
	// Inputs
	.clk						(CLOCK_50),
	.reset						(reset),

	.the_command				(the_command_w),
	.send_command				(send_command_w),

	.ps2_clk_posedge			(ps2_clk_posedge),
	.ps2_clk_negedge			(ps2_clk_negedge),

	// Bidirectionals
	.PS2_CLK					(PS2_CLK),
	.PS2_DAT					(PS2_DAT),

	// Outputs
	.command_was_sent			(command_was_sent_w),
	.error_communication_timed_out	(error_communication_timed_out_w)
);

endmodule
```