Project: Snake Game via VGA

Members : Lama Imam, Ayeza Nasir, Hamna Jamil, Ali Raza

Progress Report

Sections:

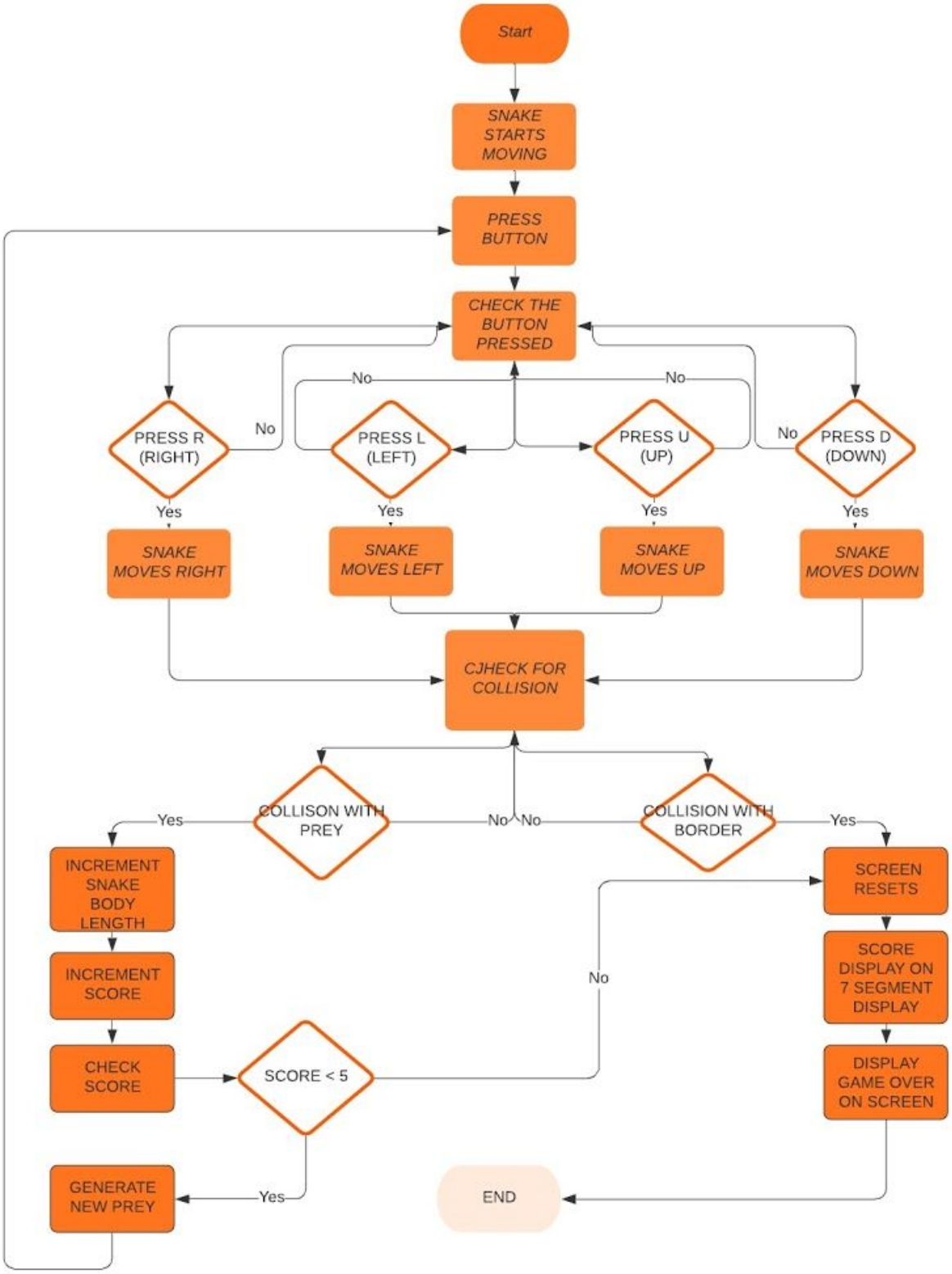
Introduction and Game specifics:

For our final project deliverable for the course Digital Logic Design, we created the classic single- player “Snake” game based on existing snake game architectures using basys 3 fpga board implementation on Xilinx Vivado. The player controls a snake head, presenting a snake of length one unit, on a bordered plane using the up, down, right and left push buttons of the FGPA board and has to pass through a dot, representing prey. Each time it passes through a dot (or eats the prey), another unit will be added to the length and the player will get one point. The player loses when the snake runs into the screen border. After the player loses or his score reaches nine the screen resets indicating game is over and the player’s final score is displayed on the 7 segment display.

Our game will be displayed on Vga monitor and our input peripheral are external push buttons connected to the push buttons of the fpga board.

Here is a screenshot of our game display blue border and green 1 unit snake head shown:

Overall Game flowchart:



FSM design:

FSM 1:

This Fsm is a Moore Machine since output depends on current state only.

As the snake eats its prey, it generates another prey to be eaten.

The modules that execute this FSM are snake_body, random_prey and Prey modules.

Reset screen=00000

1st prey generated=00001

1st prey eaten=00010

2nd prey generated=00011

2nd prey eaten=00100

3rd prey generated=00101

3rd prey eaten=00110

4th prey generated=00111

4th prey eaten=01000

5th prey generated=01001

5th prey eaten=01010

6th prey generated=01011

6th prey eaten=01100

7th prey generated=01101

7th prey eaten=01110

8th prey generated=01111

8th prey eaten=10000

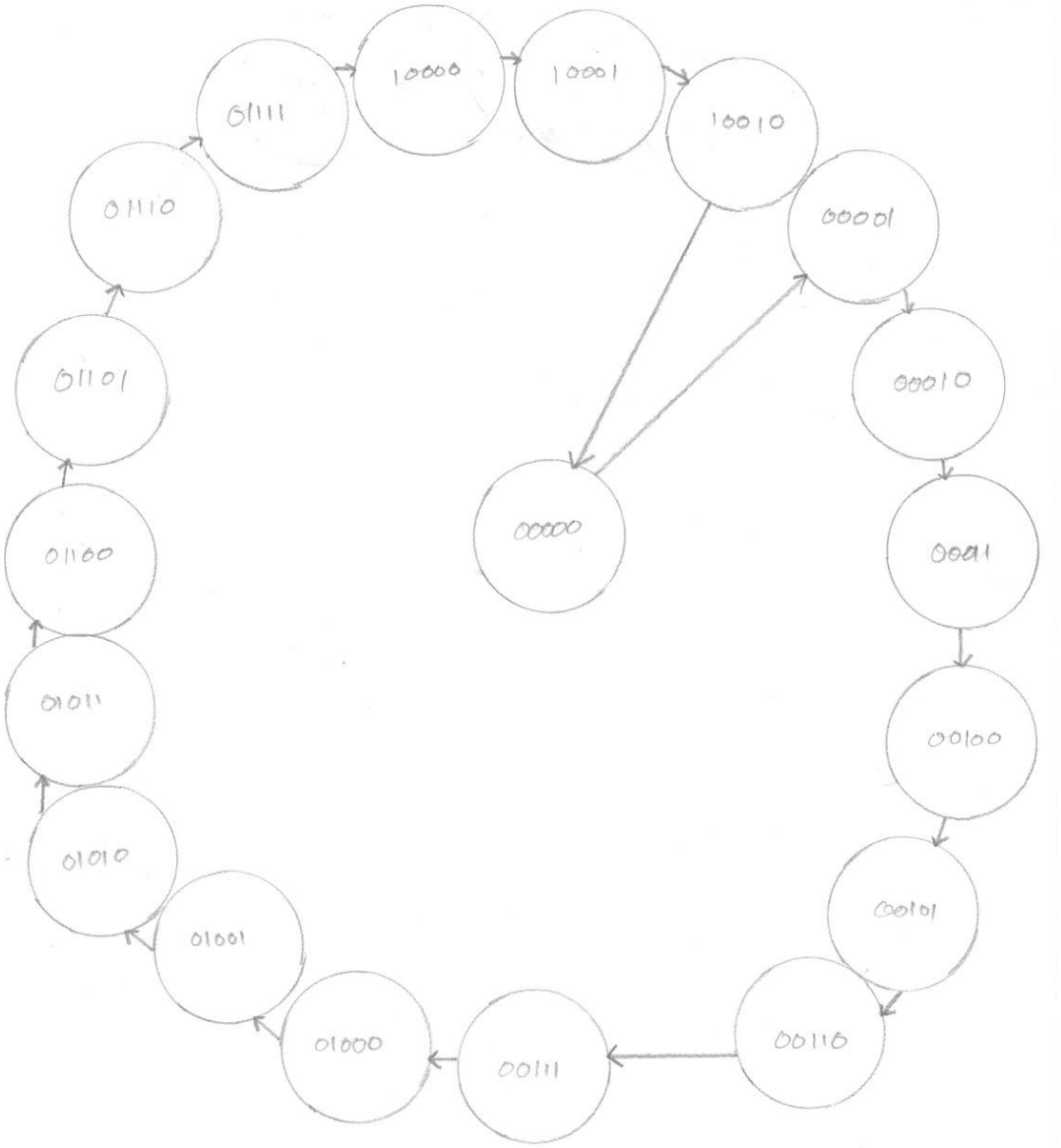
9th prey generated=10001

9th prey eaten=10010

Table 1:

Present State	Next state
00000	00001
00001	00010
00010	00011
00011	00100
00100	00101
00101	00110
00110	00111
00111	01000
01000	01001
01001	01010
01010	01011
01011	01100

01100	01101
01101	01110
01110	01111
01111	10000
10000	10001
10001	10010
10010	00000


10000
10001
10010
00001
00010
00011
00100
00101
00110
00111
01000
01001
01010
01011
01100
01101
01110
01111
00000

FSM 2:

4 states: Movement in left / right / up / down direction in default speed. Changes according to the button input signal .

The FSM is a Mealy machine since output depends on the current state and the current button input.

The modules executing this fsm are the controller module and snake_body module.

l=001

r=010

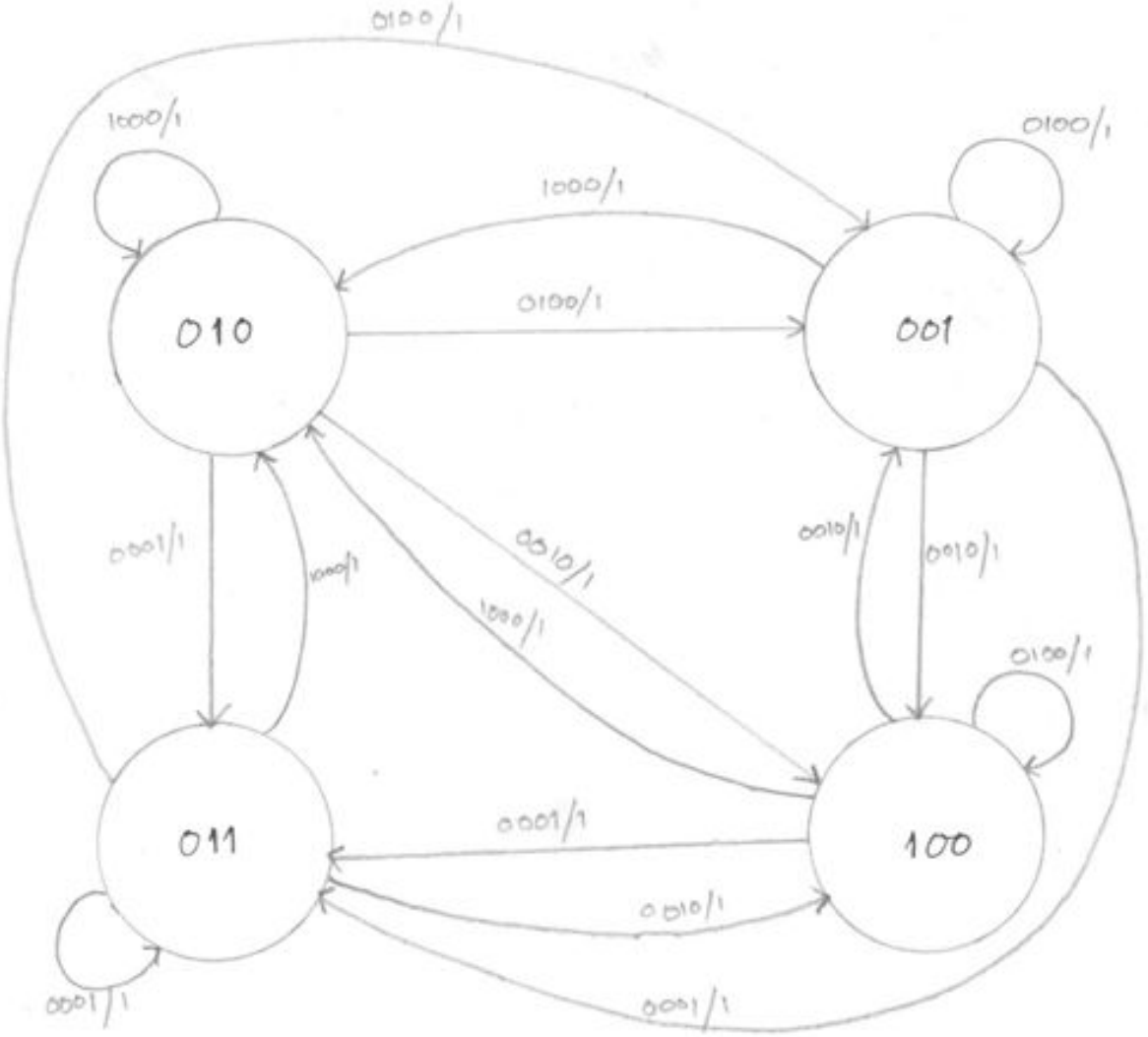
u=011

d=100

Table 2:

Present State	**Next State**				**Button Input**
	r	**l**	**u**	**d**	
010	1	0	0	0	010
010	0	1	0	0	001
010	0	0	1	0	011
010	0	0	0	1	100
001	1	0	0	0	010

001	0	1	0	0	001
001	0	0	1	0	011
001	0	0	0	1	100
011	1	0	0	0	010
011	0	1	0	0	001
011	0	0	1	0	011
011	0	0	0	1	100
100	1	0	0	0	010
100	0	1	0	0	001
100	0	0	1	0	011
100	0	0	0	1	100


0100/1
1000/1
0100/1
1000/1
010
0100/1
001
0001/1
1000/1
0010/1
1000/1
0010/1
0010/1
0100/1
011
0001/1
100
0010/1
0001/1
0001/1

FSM 3:

For good collision: When the snake head collides with a prey (randomly generated), score will be incremented until it reaches a maximum score of 9. After that, screen resets.

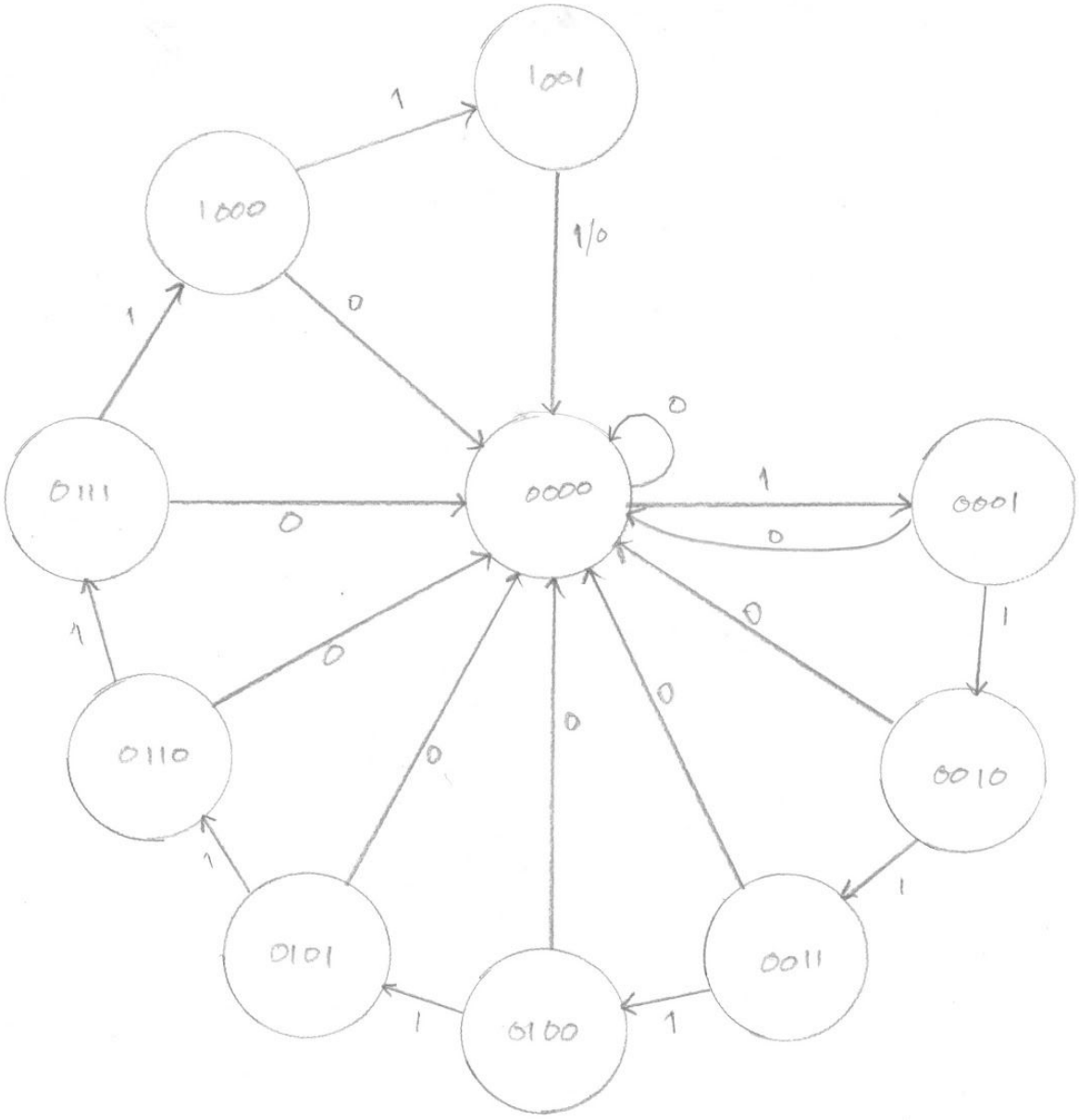
For bad collision: When the snake head collides with the border, the game will be over and screen resets.

The FSM is a Mealy machine as its output depends on the current state as well as the current input, i.e next state.

To implement this Fsm we designed the snake_body and collision modules.

Present State	**Next State**	
(score)	**Snake** **1**	**Border** **0**
0000	0001	0000
0001	0010	0000
0010	0011	0000
0011	0100	0000
0100	0101	0000
0101	0110	0000

0110	0111	0000
0111	1000	0000
1000	1001	0000
1001	0000	0000


1001
1000
0111
0000
0001
0110
0010
0101
0011
0100
1/0
1
0

Explanation:

A prey(apple) will be randomly generated by default on screen and the snake will be in movement with its default set speed from its chosen coordinates (position initially in the play area).

Once the snake eats the apple as in (collision of snake head with randomly generated point) occurs the snake will grow by a unit, and another apple will be generated .

This will be repeated 9 times till the snake reaches a length of 9 units and the player wins the game.

If before this the snake hits the border the game will be over.

This will be a total of 5 other states 3 of which will repeat the procedure carried out in state 2 , and 1 state we enter if the snake head collides with the border which results in game over. Otherwise we enter a state where score is displayed and the game is won.

To implement all above described Fsms in their entirety we designed the top module which calls all modules specifically designed for each Fsm.

Game Modules:

Module name: v_counter

Description: Takes 1 bit clock signal and enable_v which is the output trig_v from h-count and outputs v_vount.At every positive edge of the clock.

Snippet

Package | Device | v_counter.v * | Clks_Generator.v | h_counter.v

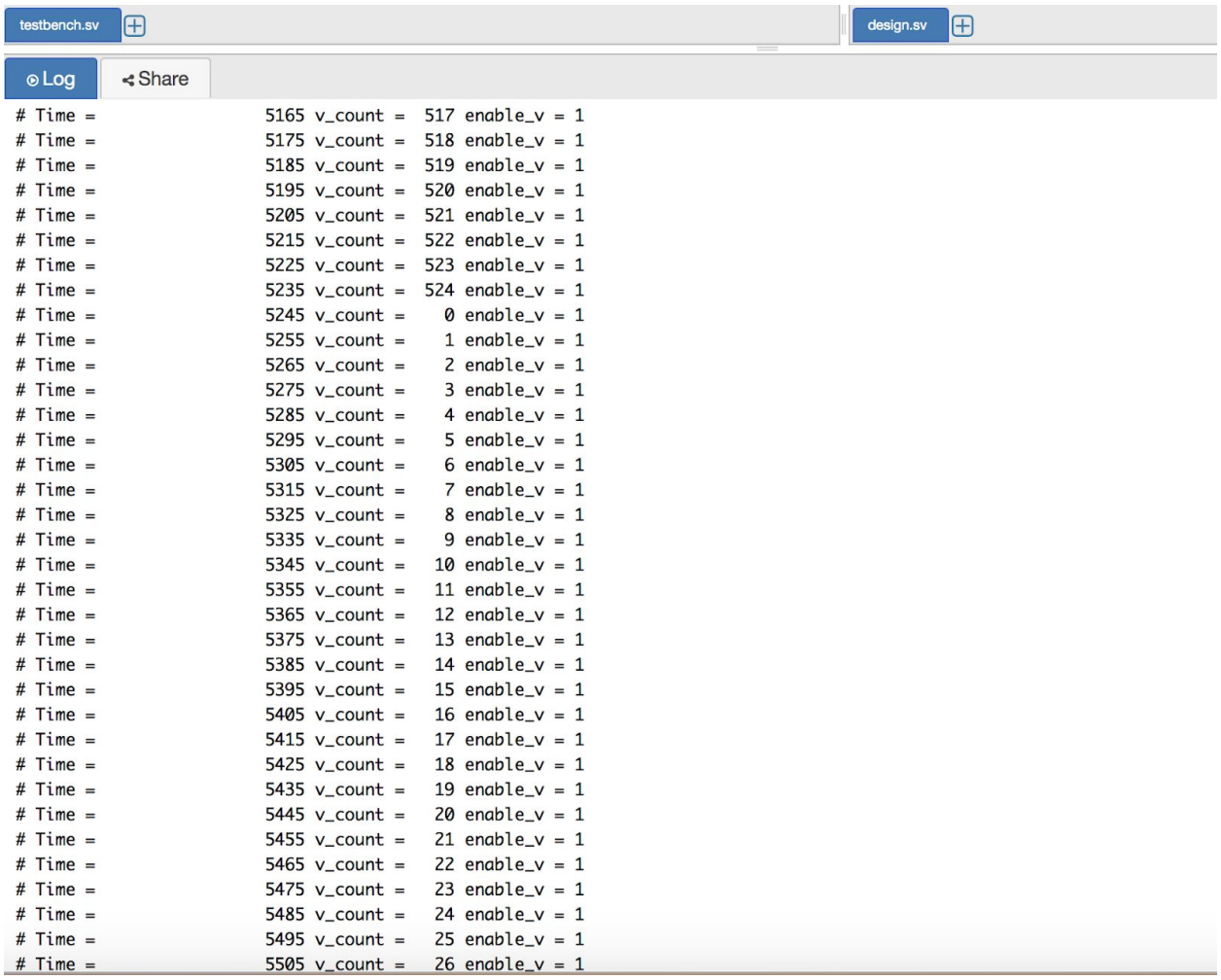
C:/Users/an05918/Desktop/project_1/project_1.srcs/sources_1/new/v_counter.v

```
1  `timescale 1ns / 1ps
2
3  module v_counter(clk,enable_v,v_count);
4    input clk;
5    input enable_v;
6    output [9:0] v_count;
7    reg [9:0] v_count;
8    initial v_count=0;
9    always @ (posedge clk)
10
11     begin
12       if (enable_v==1)
13         begin
14           if (v_count <= 523)
15             begin
16                 v_count <= v_count + 1;
17             end
18           else
19             begin
20             v_count <= 0;
21             end
22         end
23     end
24 endmodule
```

Link to eda playground module design and testbench:

https://edaplayground.com/x/DY46

Snippet of output log window that verifies functionality:



Module name : h_counter

Description : Takes 1 bit clk input (25mhz) and outputs a 10 bit h_count signal and 1 bit trig_v signal. At every positive edge of the clock**.**

Snippet :

Package | Device | v_counter.v * | h_counter.v | Clks_Generator.v

C:/Users/an05918/Desktop/project_1/project_1.srcs/sources_1/new/h_counter.v

```
1  `timescale 1ns / 1ps
2  module h_counter(
3    input clk,
4    output reg [9:0] h_count,
5    output reg trig_v);
6
7    initial
8      begin
9        h_count=0;
10       trig_v=0;
11     end
12   always @ (posedge clk)
13     begin
14       if (h_count<799)
15         begin
16           h_count <= h_count+1;
17           if (h_count ==798)
18           begin
19             trig_v <=1;
20           end
21         end
22       else
23         begin
24           trig_v <=0;
25           h_count <= 0;
26         end
27     end
28 endmodule
```

Link to eda playground module design and testbench:
https://edaplayground.com/x/6Ww5

Snippet of output log window that verifies functionality:

testbench.sv design.sv

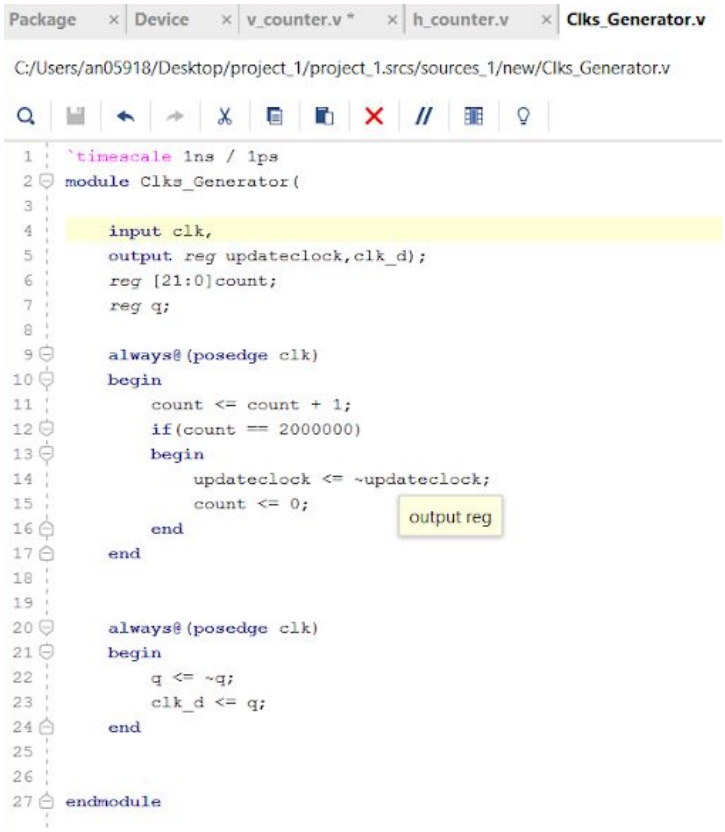
Log Share

```
# Time =              7865 h_count =  787 trig_v = 0
# Time =              7875 h_count =  788 trig_v = 0
# Time =              7885 h_count =  789 trig_v = 0
# Time =              7895 h_count =  790 trig_v = 0
# Time =              7905 h_count =  791 trig_v = 0
# Time =              7915 h_count =  792 trig_v = 0
# Time =              7925 h_count =  793 trig_v = 0
# Time =              7935 h_count =  794 trig_v = 0
# Time =              7945 h_count =  795 trig_v = 0
# Time =              7955 h_count =  796 trig_v = 0
# Time =              7965 h_count =  797 trig_v = 0
# Time =              7975 h_count =  798 trig_v = 0
# Time =              7985 h_count =  799 trig_v = 1
# Time =              7995 h_count =    0 trig_v = 0
# Time =              8005 h_count =    1 trig_v = 0
# Time =              8015 h_count =    2 trig_v = 0
# Time =              8025 h_count =    3 trig_v = 0
# Time =              8035 h_count =    4 trig_v = 0
# Time =              8045 h_count =    5 trig_v = 0
# Time =              8055 h_count =    6 trig_v = 0
# Time =              8065 h_count =    7 trig_v = 0
# Time =              8075 h_count =    8 trig_v = 0
# Time =              8085 h_count =    9 trig_v = 0
# Time =              8095 h_count =   10 trig_v = 0
# Time =              8105 h_count =   11 trig_v = 0
# Time =              8115 h_count =   12 trig_v = 0
# Time =              8125 h_count =   13 trig_v = 0
# Time =              8135 h_count =   14 trig_v = 0
# Time =              8145 h_count =   15 trig_v = 0
# Time =              8155 h_count =   16 trig_v = 0
# Time =              8165 h_count =   17 trig_v = 0
# Time =              8175 h_count =   18 trig_v = 0
# Time =              8185 h_count =   19 trig_v = 0
# Time =              8195 h_count =   20 trig_v = 0
# Time =              8205 h_count =   21 trig_v = 0
# Time =              8215 h_count =   22 trig_v = 0
```

Module name :Clks_Generator

Description: Takes 100 mhz clock as input and outputs two clocks 25 mhz clk_d and 25hz updateclock **.**

Snippet:



Link to eda playground module design and testbench for 25Mhz clock, clk_d:

https://edaplayground.com/x/Hhid

Snippet of output log window to verify functionality:

Log Share

```
# //
# //  Copyright 1991-2020 Mentor Graphics Corporation
# //  All Rights Reserved.
# //
# //  QuestaSim and its associated documentation contain trade
# //  secrets and commercial or financial information that are the property of
# //  Mentor Graphics Corporation and are privileged, confidential,
# //  and exempt from disclosure under the Freedom of Information Act,
# //  5 U.S.C. Section 552. Furthermore, this information
# //  is prohibited from disclosure under the Trade Secrets Act,
# //  18 U.S.C. Section 1905.
# //
# Loading sv_std.std
# Loading work.tb(fast)
# Loading work.clk_div(fast)
#
# vsim -voptargs=+acc=npr
# run -all
#                    0clk = 0,rst= 0,clk_d = x
#                   10clk = 1,rst= 0,clk_d = 0
#                   20clk = 0,rst= 1,clk_d = 0
#                   30clk = 1,rst= 1,clk_d = 1
#                   40clk = 0,rst= 1,clk_d = 1
#                   50clk = 1,rst= 1,clk_d = 0
#                   60clk = 0,rst= 1,clk_d = 0
#                   70clk = 1,rst= 1,clk_d = 1
#                   80clk = 0,rst= 1,clk_d = 1
#                   90clk = 1,rst= 1,clk_d = 0
#                  100clk = 0,rst= 1,clk_d = 0
#                  110clk = 1,rst= 1,clk_d = 1
# ** Note: $finish    : testbench.sv(18)
#    Time: 120 ns  Iteration: 0  Instance: /tb
# End time: 04:30:14 on Dec 11,2020, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
Done
```

Link to eda playground module design and testbench for 25hz clock, updateclock:

https://edaplayground.com/x/Agbc

Snippet of output log window to verify functionality:

Log Share

```
# //
# //  Copyright 1991-2020 Mentor Graphics Corporation
# //  All Rights Reserved.
# //
# //  QuestaSim and its associated documentation contain trade
# //  secrets and commercial or financial information that are the property of
# //  Mentor Graphics Corporation and are privileged, confidential,
# //  and exempt from disclosure under the Freedom of Information Act,
# //  5 U.S.C. Section 552. Furthermore, this information
# //  is prohibited from disclosure under the Trade Secrets Act,
# //  18 U.S.C. Section 1905.
# //
# Loading sv_std.std
# Loading work.tb(fast)
# Loading work.updateclk(fast)
#
# vsim -voptargs=+acc=npr
# run -all
#                    0clk = 0,updateclock = x, count=      x
#                   10clk = 1,updateclock = 1, count=      0
#                   20clk = 0,updateclock = 1, count=      0
#                   30clk = 1,updateclock = 0, count=      1
#                   40clk = 0,updateclock = 0, count=      1
#                   50clk = 1,updateclock = 0, count=      2
#                   60clk = 0,updateclock = 0, count=      2
#                   70clk = 1,updateclock = 0, count=      3
#                   80clk = 0,updateclock = 0, count=      3
#                   90clk = 1,updateclock = 0, count=      4
# ** Note: $finish    : testbench.sv(15)
#    Time: 100 ns  Iteration: 0  Instance: /tb
# End time: 08:45:11 on Dec 10,2020, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
Done
```

Module name : vga_sync

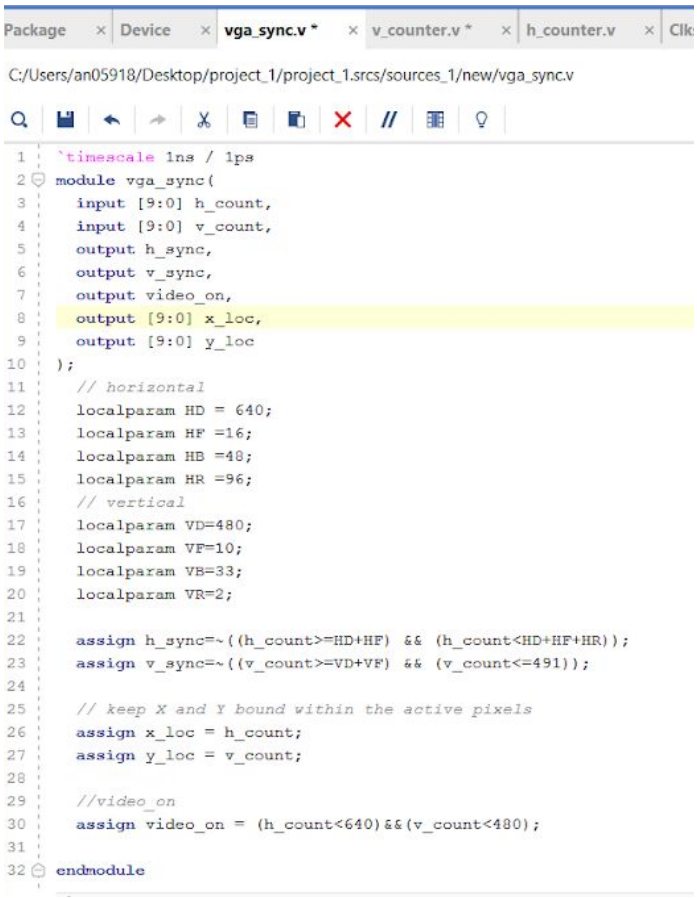
Description : The vga_sync module generates timing and synchronization signals it takes h_count and v_count as input.

2 of its outputs are the h_sync and v_sync signals that are connected to the VGA port to control the horizontal and vertical scans of the monitor in a transverse manner.

Another output is the video_on signal indicating whether the current targeted pixel is in the displayable region. It is asserted only when the h_count is smaller than 640 and v_count is smaller than 480.

The last two outputs are x_loc (the location of pixel x) y_loc (the location of pixel y) signals. The x_loc and y_loc specify the location of the current pixel and can be obtained from h_count and v_count

Snippet:



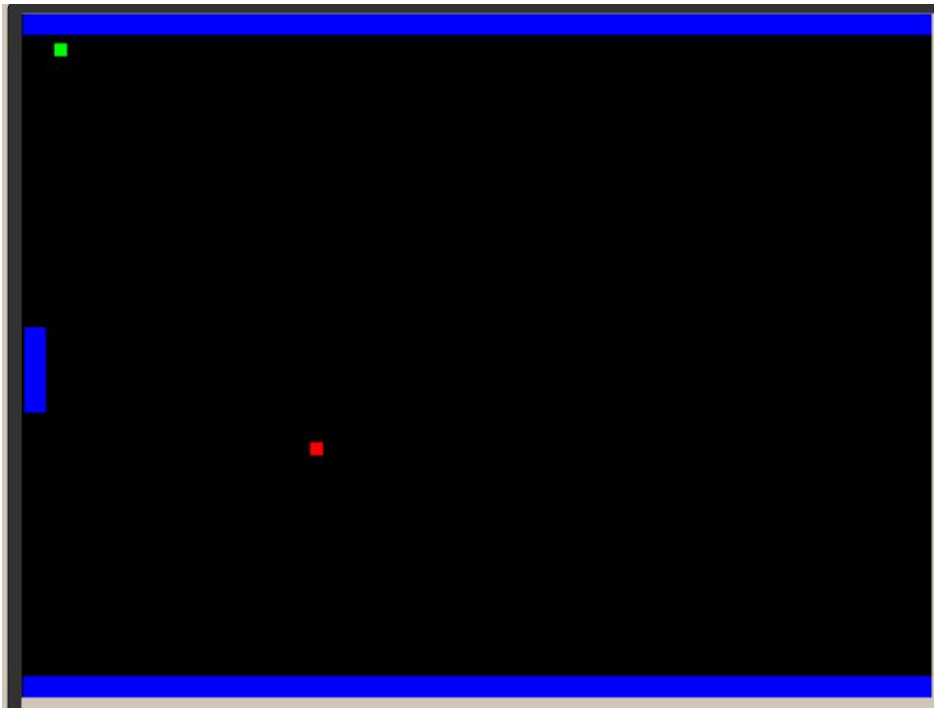
Link to eda playground module design and testbench for vga_sync:

https://edaplayground.com/x/ZqAW

Snippet :

Greenblock signifying snakehead in the initial stage as the game begins.

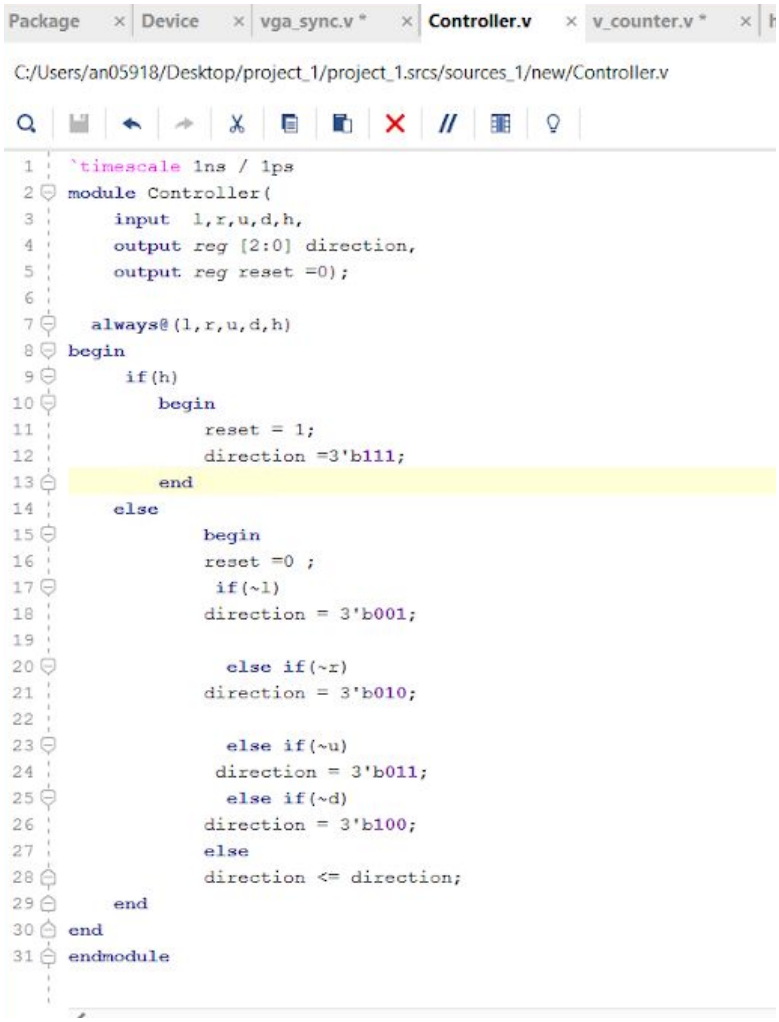
Red block signifying the prey.



Module name: Controller

Description: This module corroborates the movement of the snake in accordance with the assigned key.

Snippet:



Link to eda playground module design and testbench for Controller:

https://edaplayground.com/x/a5d5

Snippet of output log window to verify functionality:

Log Share

```
# //  Questasim and its associated documentation contain trade
# //  secrets and commercial or financial information that are the property
# //  Mentor Graphics Corporation and are privileged, confidential,
# //  and exempt from disclosure under the Freedom of Information Act,
# //  5 U.S.C. Section 552. Furthermore, this information
# //  is prohibited from disclosure under the Trade Secrets Act,
# //  18 U.S.C. Section 1905.
# //
# Loading sv_std.std
# Loading work.tb(fast)
# Loading work.Controller(fast)
#
# vsim -voptargs=+acc=npr
# run -all
#                    0clk = 0,l=x,r=x,u=x,d=x,direction=xxx
#                   10clk = 1,l=x,r=x,u=x,d=x,direction=xxx
#                   20clk = 0,l=1,r=0,u=0,d=0,direction=xxx
#                   30clk = 1,l=1,r=0,u=0,d=0,direction=001
#                   40clk = 0,l=0,r=1,u=0,d=0,direction=001
#                   50clk = 1,l=0,r=1,u=0,d=0,direction=010
#                   60clk = 0,l=0,r=0,u=1,d=0,direction=010
#                   70clk = 1,l=0,r=0,u=1,d=0,direction=011
#                   80clk = 0,l=0,r=0,u=0,d=1,direction=011
#                   90clk = 1,l=0,r=0,u=0,d=1,direction=100
#                  100clk = 0,l=0,r=0,u=0,d=1,direction=100
# ** Note: $finish    : testbench.sv(19)
#    Time: 110 ns  Iteration: 0  Instance: /tb
# End time: 04:24:20 on Dec 11,2020, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
```

Done

Module Name: random_prey

Description: In our game, the snake eats the prey to grow larger. We made a pseudo-random coordinate generating module to generate prey in a random part of the screen when the game starts, and when the snake collides with prey.

Snippet :

Package | Device | vga_sync.v * | Controller.v | v_counter.v * | h_counter.v | Clks_Generator.v | random_prey.v

C:/Users/an05918/Desktop/project_1/project_1.srcs/sources_1/new/random_prey.v

```
1  `timescale 1ns / 1ps
2  module random_prey(
3      input clk_d, updateclock,
4      output reg [9:0]rand_X= 70,
5      output reg [8:0]rand_Y=90);
6
7
8    always@(posedge clk_d)
9      begin
10     rand_X= rand_X +30;
11     if(rand_X >= 570)
12         begin
13         rand_X = 40 ;
14         end
15     end
16
17   always @(posedge updateclock)
18     begin
19     rand_Y=rand_Y+20 ;
20     if(rand_Y >= 400)
21         begin
22         rand_Y = 40 ;
23         end
24     end
25
26 endmodule
```

Module Name : Prey

Description: This module calls random_prey module and collaborates the collision and regeneration of random prey on the screen .

Snippet:

Package | Device | vga_sync.v * | Prey.v * | Controller.v | v_counter.v * | h_counter.v | Clks_Generator.v

C:/Users/an05918/Desktop/project_1/project_1.srcs/sources_1/new/Prey.v

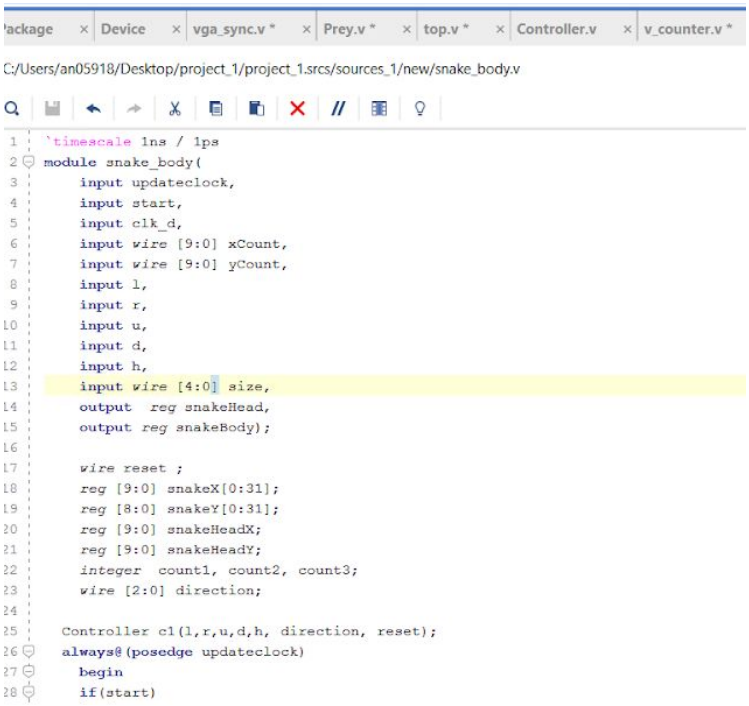
```
1  `timescale 1ns / 1ps
2  module Prey(
3      input clk_d,
4      input good_collision,
5      input start,
6      input wire [9:0] xCount,
7      input wire [9:0] yCount,
8      input updateclock,
9      output reg prey);
10     reg [9:0] PreyX;
11     reg [8:0] PreyY;
12     reg prey_in_x;
13     reg prey_in_y;
14     wire [9:0]rand_X;
15     wire [8:0]rand_Y;
16   random_prey r1(clk_d,updateclock, rand_X, rand_Y);
17   always@(clk_d)
18     begin
19         if(good_collision)...
24         if(~start)...
29      end
30   always @(posedge clk_d)
31     begin
32       prey_in_x <= (xCount > PreyX && xCount < (PreyX + 10));
33       prey_in_y <= (yCount > PreyY && yCount < (PreyY + 10));
34         prey = prey_in_x && prey_in_y;
35     end
36 endmodule
37
38
39
```

Module Name: snake_body

Description: This module increments and manages the length of the snake body as it collides with prey and elongates. It initially begins as a singular unit as in the snake head and grows as it eats.

Controller module has been called in this module.

Snippet



```
    begin
        if(direction != 3'b111)begin
        for(count1 = 31; count1 > 0; count1 = count1 - 1)
            begin
                if(count1 <= size - 1)
                begin
                    snakeX[count1] = snakeX[count1 - 1];
                    snakeY[count1] = snakeY[count1 - 1];
                end
            end
            end
        case(direction)
            3'b001: snakeY[0] <= (snakeY[0] - 10);
            3'b010: snakeX[0] <= (snakeX[0] - 10);
            3'b011: snakeY[0] <= (snakeY[0] + 10);
            3'b100: snakeX[0] <= (snakeX[0] + 10);
            3'b111:begin snakeX[0] <= snakeX[0];
            snakeY[0] <= snakeY[0]; end
            endcase
        end
    else if(~start)
    begin
        for(count3 = 1; count3 < 32; count3 = count3+1)
            begin
            snakeX[count3] = 700;
            snakeY[count3] = 500;
            end
            snakeX[0] = 300;
            snakeY[0] = 300;
    end
    end
```

```

  always@(posedge clk_d)
    begin

        snakeBody =0 ;

        for(count2 = 1; count2 < size; count2 = count2 + 1)
        begin

            if(snakeBody ==0 )
            snakeBody = ((xCount > snakeX[count2] && xCount < snakeX[count2]+10) && (yCount > snakeY[count2] && yCount < snakeY[count2]+10));


        end
    end


  always@(posedge clk_d)
    begin
        snakeHead = (xCount > snakeX[0] && xCount < (snakeX[0]+10)) && (yCount > snakeY[0] && yCount < (snakeY[0]+10));
    end


endmodule
```

Module name: collision

Description: There are 2 types of cases when collision occurs , as in good collison where the snakehead collides with the prey resulting in incrementing the snake body length. On the other hand if it is a bad collison , meaning that the snake head has collided with the border , then the screen is reset and the current score displayed.
This module calls the snakebody and prey module within itself.

Snippet:

```
`timescale 1ns / 1ps
module collision(
    input border,
    input clk_d,
    input  updateclock,
    input start,
    input wire [9:0] xCount,
    input wire [9:0] yCount,
    input l,
    input r,
    input u,
    input d,
    input h,
    output wire snakeBody,
    output wire snakeHead,
    output reg GameOver,
    output wire prey,
    output reg[6:0] seg1,
    output reg[6:0] seg2);


    reg good_collision, bad_collision;
    reg azab = 1;
    reg [4:0] size =1;
  snake_body s1(updateclock,start,clk_d,xCount,yCount,l,r,u,d,h,size,snakeHead,snakeBody);
  Prey p1(clk_d,good_collision,start,xCount,yCount,updateclock,prey);

    reg lethal, nonLethal ;
     always @(posedge clk_d)
      lethal = (border|| snakeBody)&& snakeHead ;
  always @(posedge clk_d)
      nonLethal = prey && snakeHead && azab;
```

```
34
35      wire [4:0] check_size ;
36      assign check_size = (size-1);
37      always @(check_size)
38        begin
39          if(check_size<=9)
40            seg2 <= ~7'b0111111;
41          else if(check_size[4:3] ==2'b01)
42            seg2 <= ~7'b0000110;
43
44          else if(check_size[4:3] ==2'b10)
45            seg2 <= ~7'b1011011;
46
47          else
48            seg2 <= ~7'b1001111;
49
50       case (check_size[3:0] )
51         0 : seg1 <= ~7'b0111111;
52         1 : seg1 <= ~7'b0000110;
53         2 : seg1 <= ~7'b1011011;
54         3 : seg1 <= ~7'b1001111;
55         4 : seg1 <= ~7'b1100110;
56         5 : seg1 <= ~7'b1101101;
57         6 : seg1 <= ~7'b1111101;
58         7 : seg1 <= ~7'b0000111;
59         8 : seg1 <= ~7'b1111111;
60         9 : seg1 <= ~7'b1101111;
61        default : seg1 <= ~7'bX;
62      endcase
63
64      end
```

```
65    always @(posedge clk_d)
66      if(nonLethal) begin
67        good_collision<=1;
68        size = size+1;
69        azab=0 ;
70      end
71      else if(~start)
72        size = 1;
73      else
74        begin
75          good_collision=0; azab =1 ;
76        end
77    always @(posedge clk_d)
78        if(lethal)
79          bad_collision=1;
80        else
81          bad_collision=0;
82    always @(posedge clk_d)
83      if(bad_collision)
84        GameOver<=1;
85      else if(~start)
86        GameOver=0;
87  endmodule
```

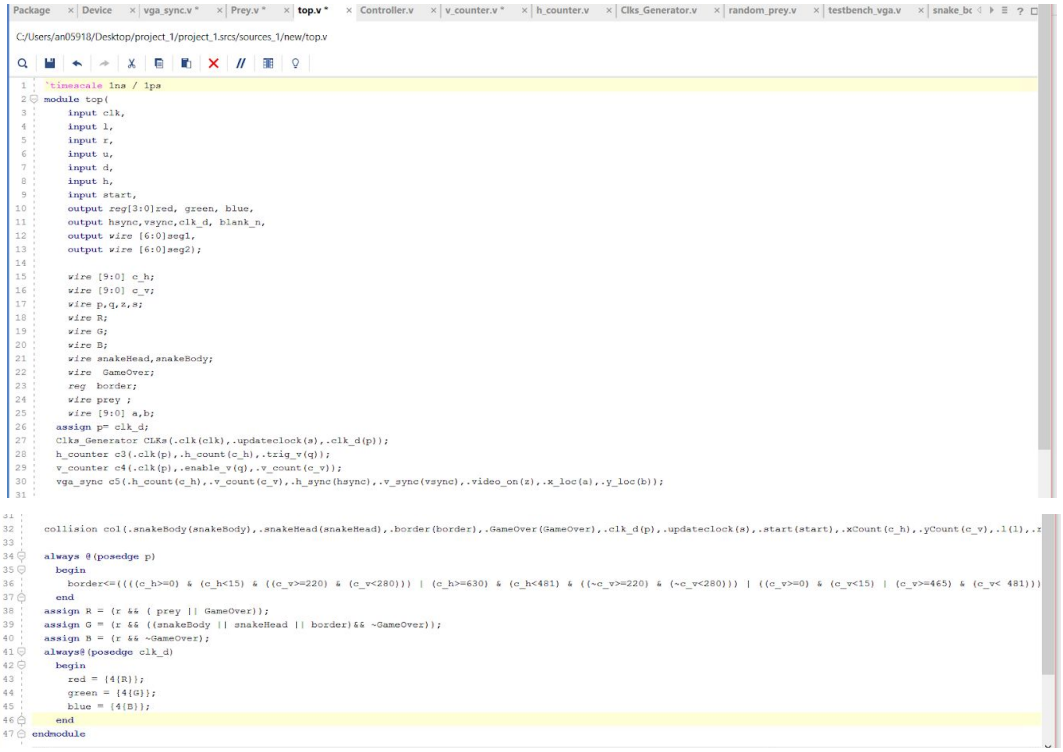
Link to eda playground module design and testbench:

https://edaplayground.com/x/DMeH

Module Name : top

Description: This is the main module within which the h_count , v_count , vga_sync and collisions module has been called. This module is essential to how our game operates and overlooks the collaboration of all the separate modules to ensure the game works according to the flowchart.

Snippet:



```
`timescale 1ns / 1ps
module top(
    input clk,
    input l,
    input r,
    input u,
    input d,
    input h,
    input start,
    output reg[3:0]red, green, blue,
    output hsync,vsync,clk_d, blank_n,
    output wire [6:0]seg1,
    output wire [6:0]seg2);

    wire [9:0] c_h;
    wire [9:0] c_v;
    wire p,q,z,s;
    wire R;
    wire G;
    wire B;
    wire snakeHead,snakeBody;
    wire  GameOver;
    reg  border;
    wire prey ;
    wire [9:0] a,b;
  assign p= clk_d;
  Clks_Generator CLKs(.clk(clk),.updateclock(s),.clk_d(p));
  h_counter c3(.clk(p),.h_count(c_h),.trig_v(q));
  v_counter c4(.clk(p),.enable_v(q),.v_count(c_v));
  vga_sync c5(.h_count(c_h),.v_count(c_v),.h_sync(hsync),.v_sync(vsync),.video_on(z),.x_loc(a),.y_loc(b));


  collision col(.snakeBody(snakeBody),.snakeHead(snakeHead),.border(border),.GameOver(GameOver),.clk_d(p),.updateclock(s),.start(start),.xCount(c_h),.yCount(c_v),.l(l),.r
  always @(posedge p)
    begin
      border<=((((c_h>=0) & (c_h<15) & ((c_v>=220) & (c_v<280))) | (c_h>=630) & (c_h<481) & ((~c_v>=220) & (~c_v<280))) | ((c_v>=0) & (c_v<15) | (c_v>=465) & (c_v< 481)))
    end
  assign R = (r && ( prey || GameOver));
  assign G = (r && ((snakeBody || snakeHead || border)&& ~GameOver));
  assign B = (r && ~GameOver);
  always@(posedge clk_d)
    begin
      red = {4{R}};
      green = {4{G}};
      blue = {4{B}};
    end
endmodule
```

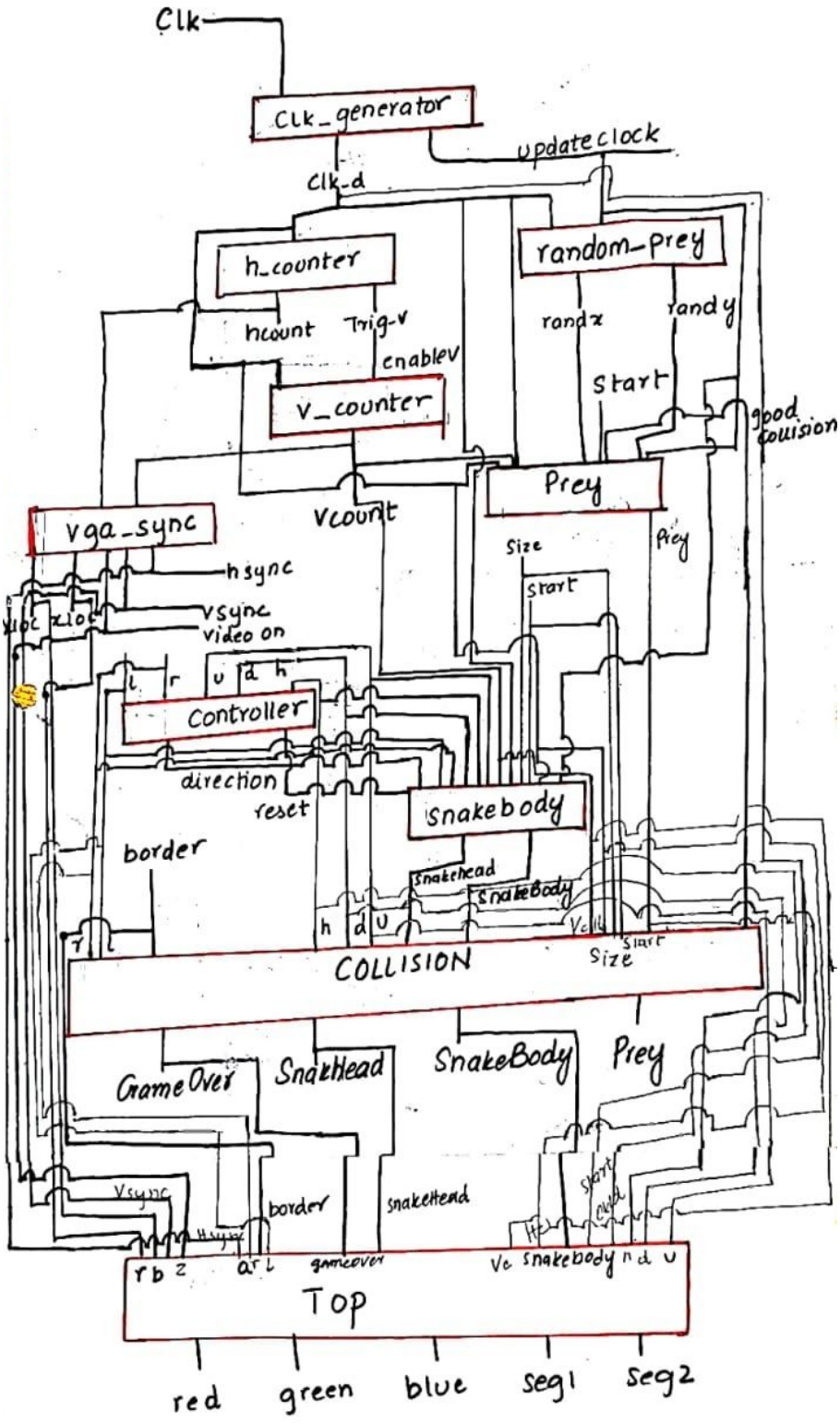
Testbench for Top module:

Package | Device | vga_sync.v * | Prey.v * | top.v * | Controller.v | v_counter.v * | h_counter.v | Cl

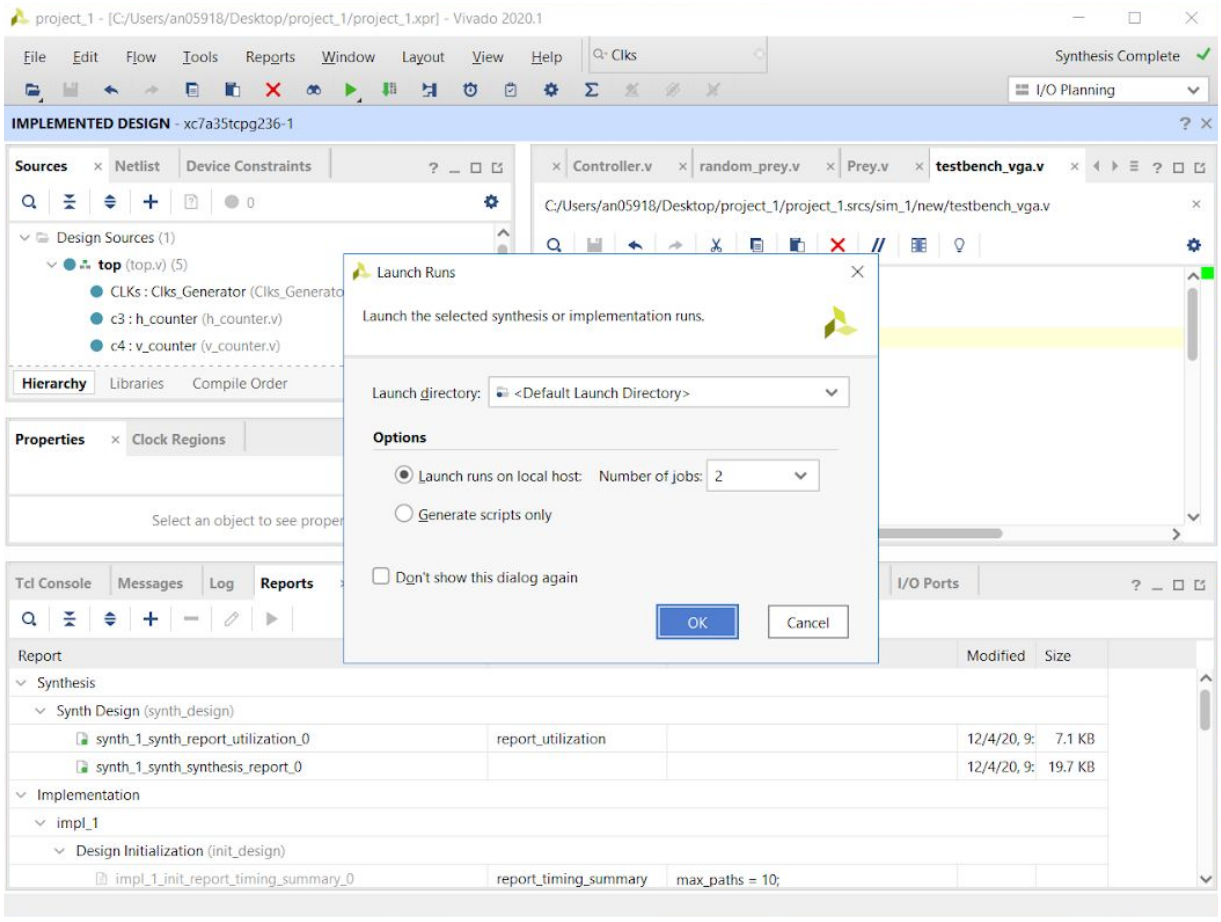
C:/Users/an05918/Desktop/project_1/project_1.srcs/sim_1/new/testbench_vga.v

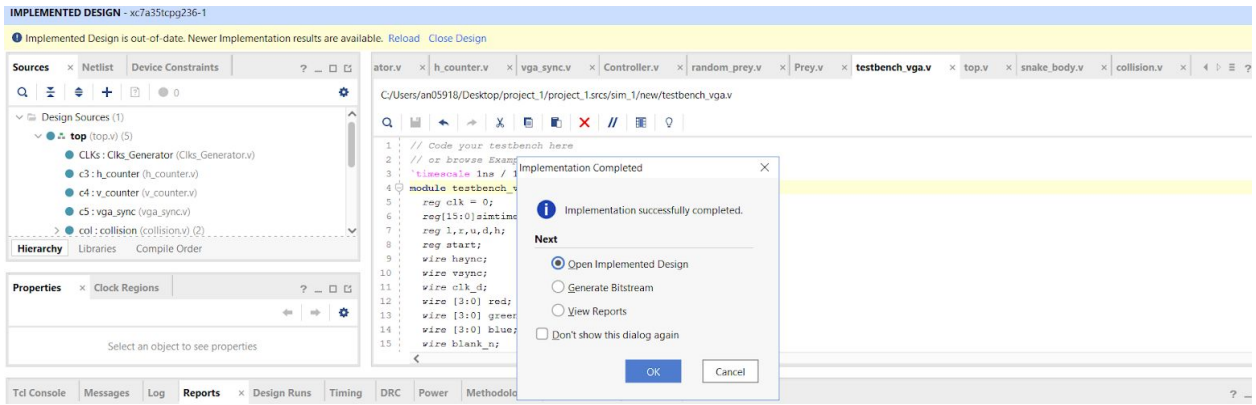
```
1  `timescale 1ns / 1ps
2  module testbench_vga();
3    reg clk = 0;
4    reg[15:0]simtime =16'd5;
5    reg l,r,u,d,h;
6    reg start;
7    wire hsync;
8    wire vsync;
9    wire clk_d;
10   wire [3:0] red;
11   wire [3:0] green;
12   wire [3:0] blue;
13   wire blank_n;
14   wire [6:0] seg1,seg2;
15   integer file_ID;
16
17   top t1(start,clk,clk_d, red, green, blue, hsync, vsync, blank_n,l,r,u,d,h ,seg1,seg2);
18   initial
19     #10 file_ID = $fopen("output_log_file.txt","w");
20   always
21     #5 clk = ~clk;
22   initial begin
23     #50000000 $fclose(file_ID);
24     #10 $finish;
25   end
26   always @(posedge clk_d)
27     $fwrite(file_ID,"%05d ns: %b %b %b %b %b\n", $realtime,hsync,vsync,red,green,blue,l,r,u,d);
28 endmodule
```

Game Top diagram:



Vivado Implementation:





Contributions:

Ayeza Nasir

I designed the overall Game Flow-Chart Design, Prey module, Snake_body module, Collision module and Vivado implementation of above mentioned modules as well as the documentation of above mentioned parts in this report.

Lama Imam

I designed the Vga_sync module that generates horizontal and vertical syncs as well the main Top module implementing our game and tying in all modules together.I also designed the testbench. I did the vivado implementation of modules mentioned before as well as their documentation in this report. I also designed the overall top diagram of our game showcasing all our modules and how they connect.

Hamna Jamil

I designed the random_prey module and did its vivado implementation. I also did the complete design, states definition and state diagram for our 3 game FSMs as well the documentation of above mentioned modules in this report.

Ali Raza

I designed the Clk generator module, H_counter module and V_counter module as well as their vivado implementations. I also made the final version of our Milestone 1 report.