

Project: Snake Game via VGA

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

Progress Report

Sections:

- 1. Introduction and Game specifics**
- 2. Overall Game Flowchart**
- 3. FSM design**
- 4. Game Modules**
- 5. Game Top Diagram**
- 6. Vivado implementation**
- 7. Contributions**

Introduction and Game specifics:

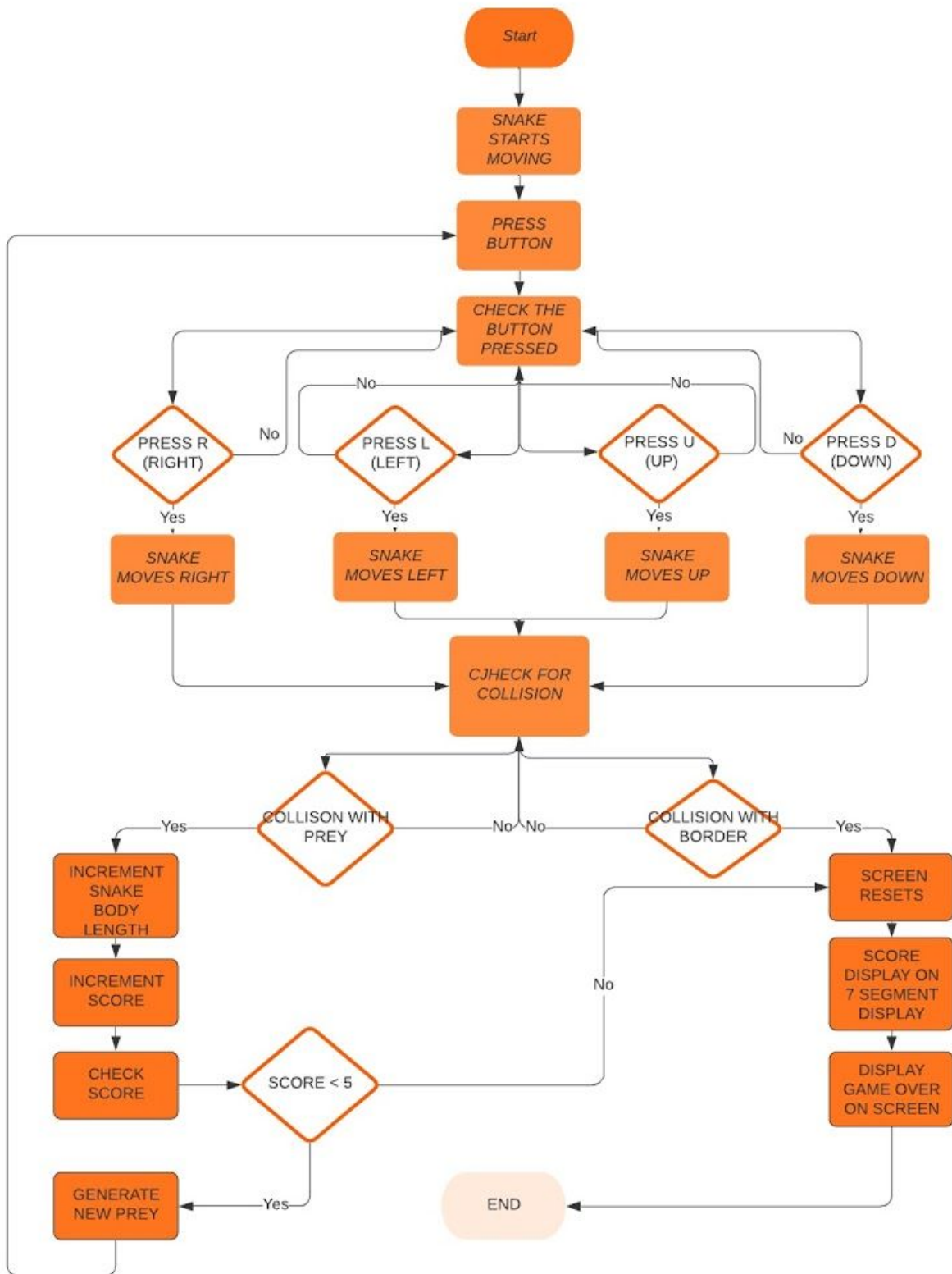
For our final project deliverable for the course Digital Logic Design, we created the classic single- player “Snake” game based on existing snake game architectures using basys 3 fpga board implementation on Xilinx Vivado. The player controls a snake head, presenting a snake of length one unit, on a 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_prej 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

DLD PROJECT REPORT : GROUP X

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

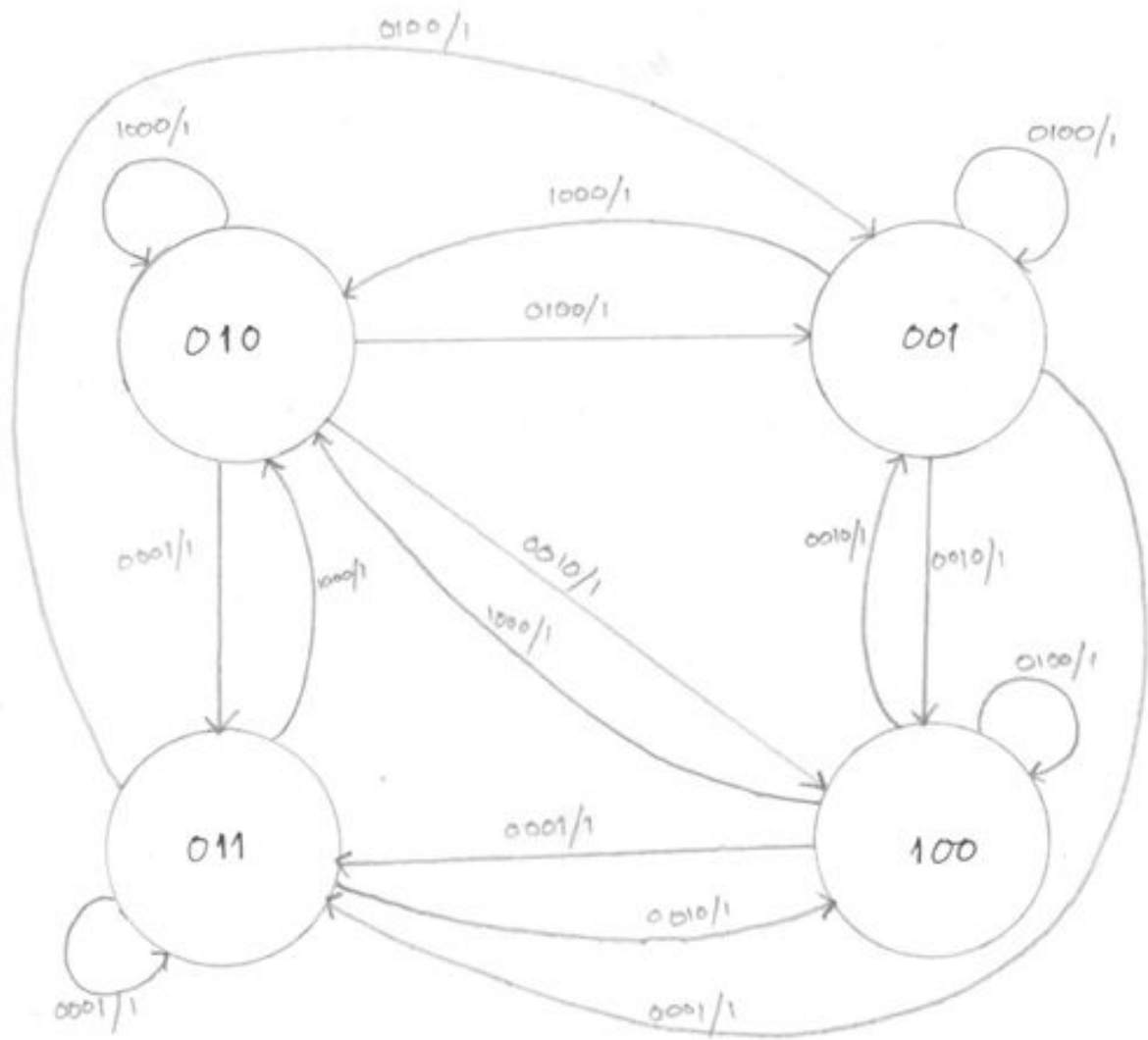
d=100

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

DLD PROJECT REPORT : GROUP X

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



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.

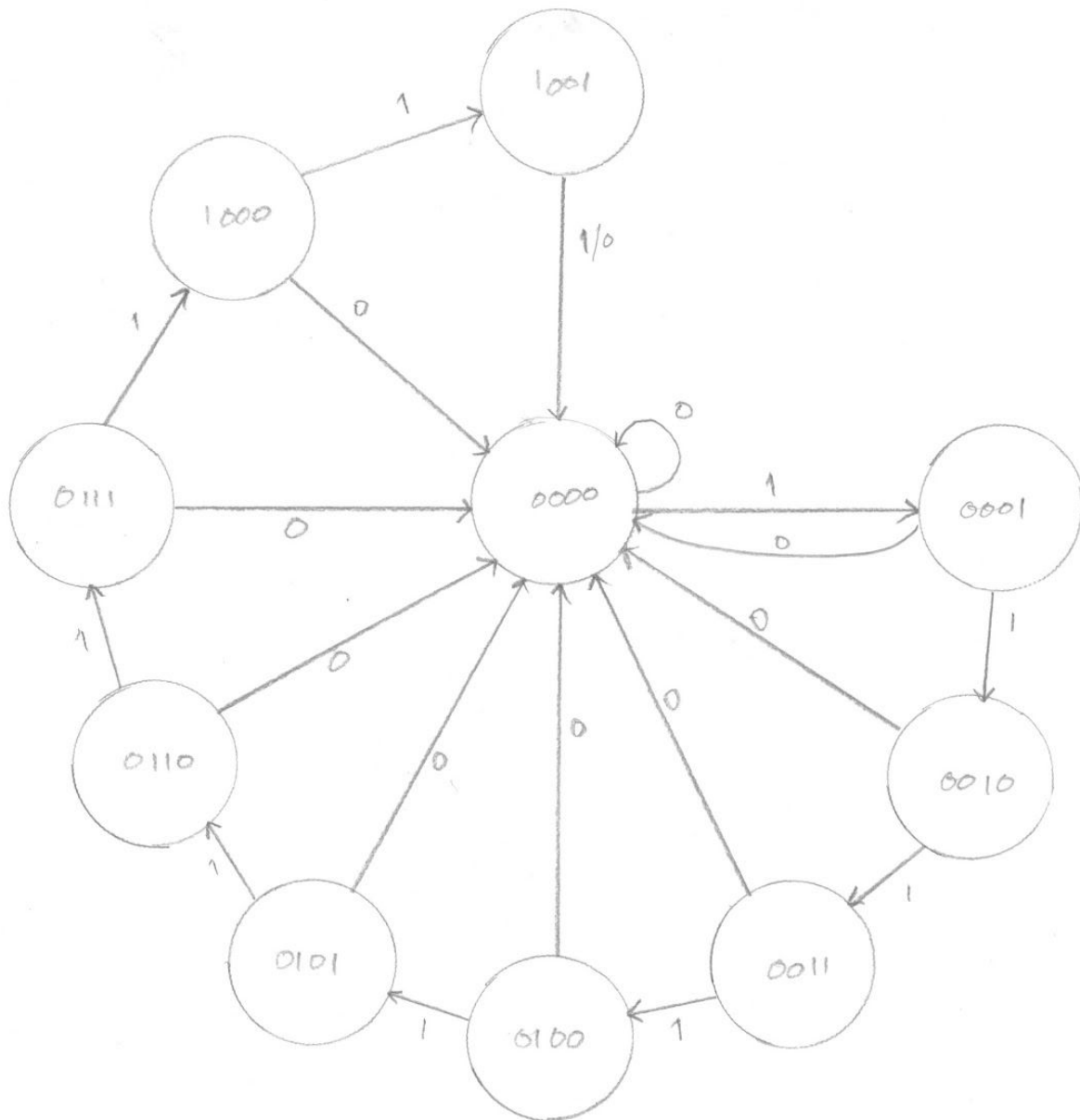
The FSM is a Mealy machine as its output depends on the current state as well as the current input, i.e next state.

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

<u>Present State</u>	<u>Next State</u>	
<u>(score)</u>	<u>Snake</u> <u>1</u>	<u>Border</u> <u>0</u>
0000	0001	0000
0001	0010	0000
0010	0011	0000
0011	0100	0000
0100	0101	0000
0101	0110	0000

DLD PROJECT REPORT : GROUP X

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



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.

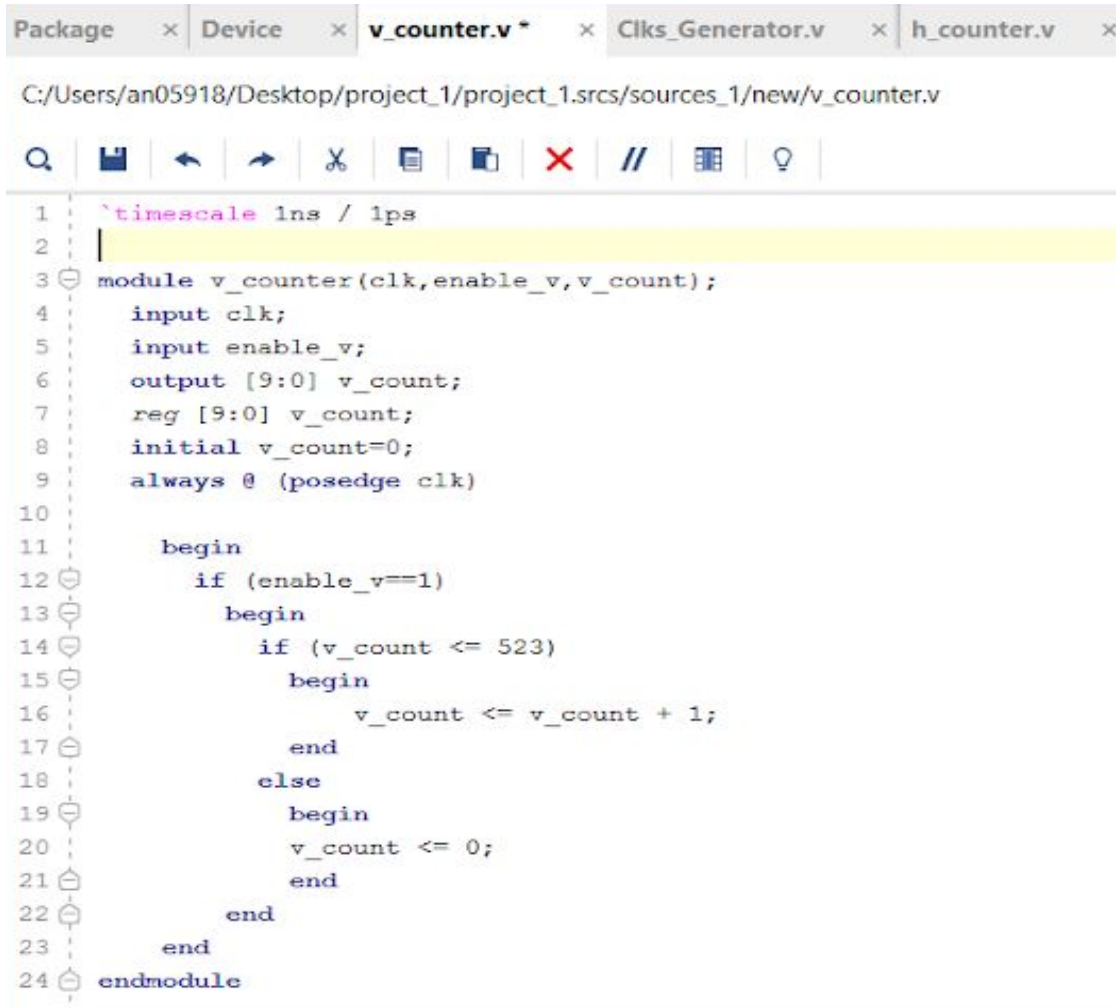
To implement all above described Fsms in their entirety we designed the top module which calls all modules specifically designed for each Fsm.

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

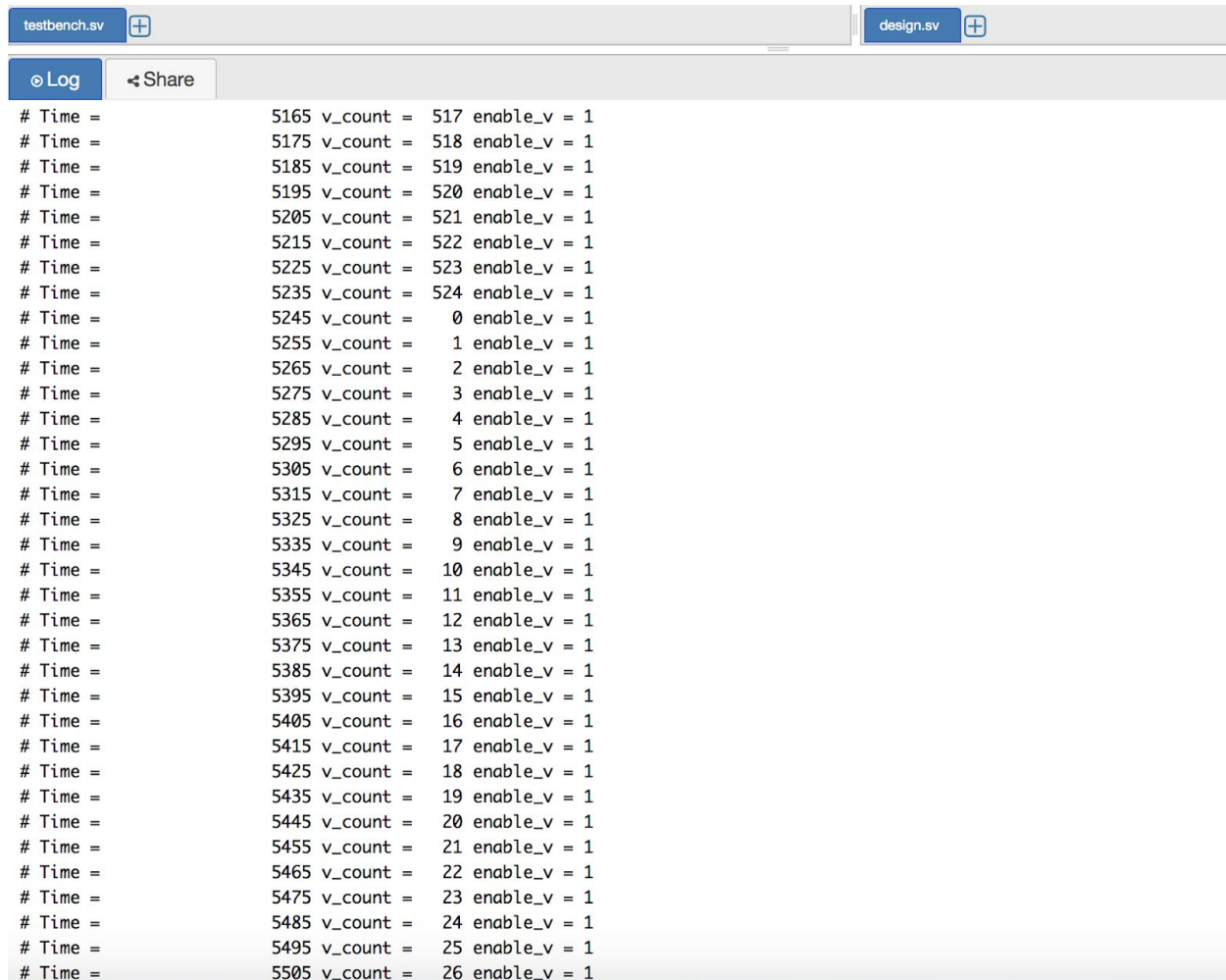
The image shows a screenshot of a Verilog code editor. At the top, there are tabs for 'Package', 'Device', 'v_counter.v *', 'Clks_Generator.v', and 'h_counter.v'. Below the tabs, the file path 'C:/Users/an05918/Desktop/project_1/project_1.srscs/sources_1/new/v_counter.v' is displayed. A toolbar with various icons (search, save, undo, redo, etc.) is visible. The main area contains the Verilog code for the 'v_counter' module. The code starts with a timescale of 1ns / 1ps, followed by the module declaration 'module v_counter(clk,enable_v,v_count);'. It then lists inputs 'clk' and 'enable_v', and an output 'v_count' of type [9:0]. A register 'v_count' of type [9:0] is declared, initialized to 0. An 'always' block is triggered by the positive edge of 'clk'. Inside this block, there is a 'begin' statement. Within the 'begin' block, there is an 'if' statement 'if (enable_v==1)'. Inside this 'if' block, there is another 'begin' statement. Inside this second 'begin' block, there is an 'if' statement 'if (v_count <= 523)'. Inside this third 'if' block, there is a 'begin' statement followed by 'v_count <= v_count + 1;'. This is followed by an 'end' statement for the third 'if' block. Then, there is an 'else' block containing a 'begin' statement followed by 'v_count <= 0;', and an 'end' statement for the 'else' block. This is followed by an 'end' statement for the second 'begin' block. Finally, there is an 'end' statement for the 'always' block, and an 'endmodule' statement at the bottom. Line numbers 1 through 24 are visible on the left side of the code editor.

```
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:



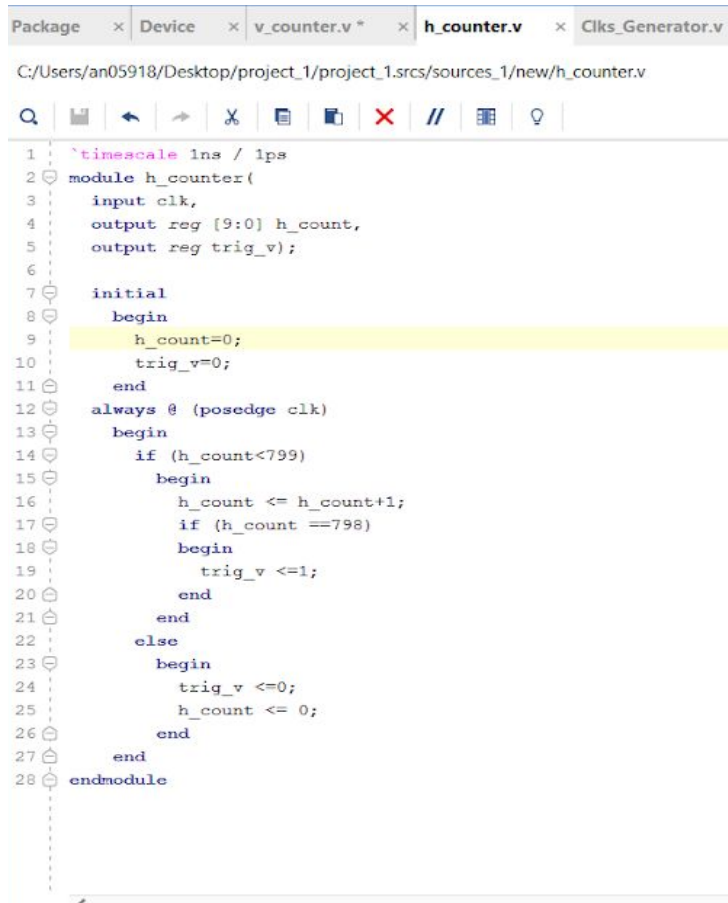
The screenshot shows the output log window of the Eda Playground. At the top, there are tabs for 'testbench.sv' and 'design.sv'. Below the tabs, there are buttons for 'Log' and 'Share'. The log window displays a series of test results, each consisting of a time value, a variable name, and its value. The results are as follows:

Time	v_count	enable_v
5165	165	1
5175	175	1
5185	185	1
5195	195	1
5205	205	1
5215	215	1
5225	225	1
5235	235	1
5245	245	0
5255	255	1
5265	265	2
5275	275	3
5285	285	4
5295	295	5
5305	305	6
5315	315	7
5325	325	8
5335	335	9
5345	345	10
5355	355	11
5365	365	12
5375	375	13
5385	385	14
5395	395	15
5405	405	16
5415	415	17
5425	425	18
5435	435	19
5445	445	20
5455	455	21
5465	465	22
5475	475	23
5485	485	24
5495	495	25
5505	505	26

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 :

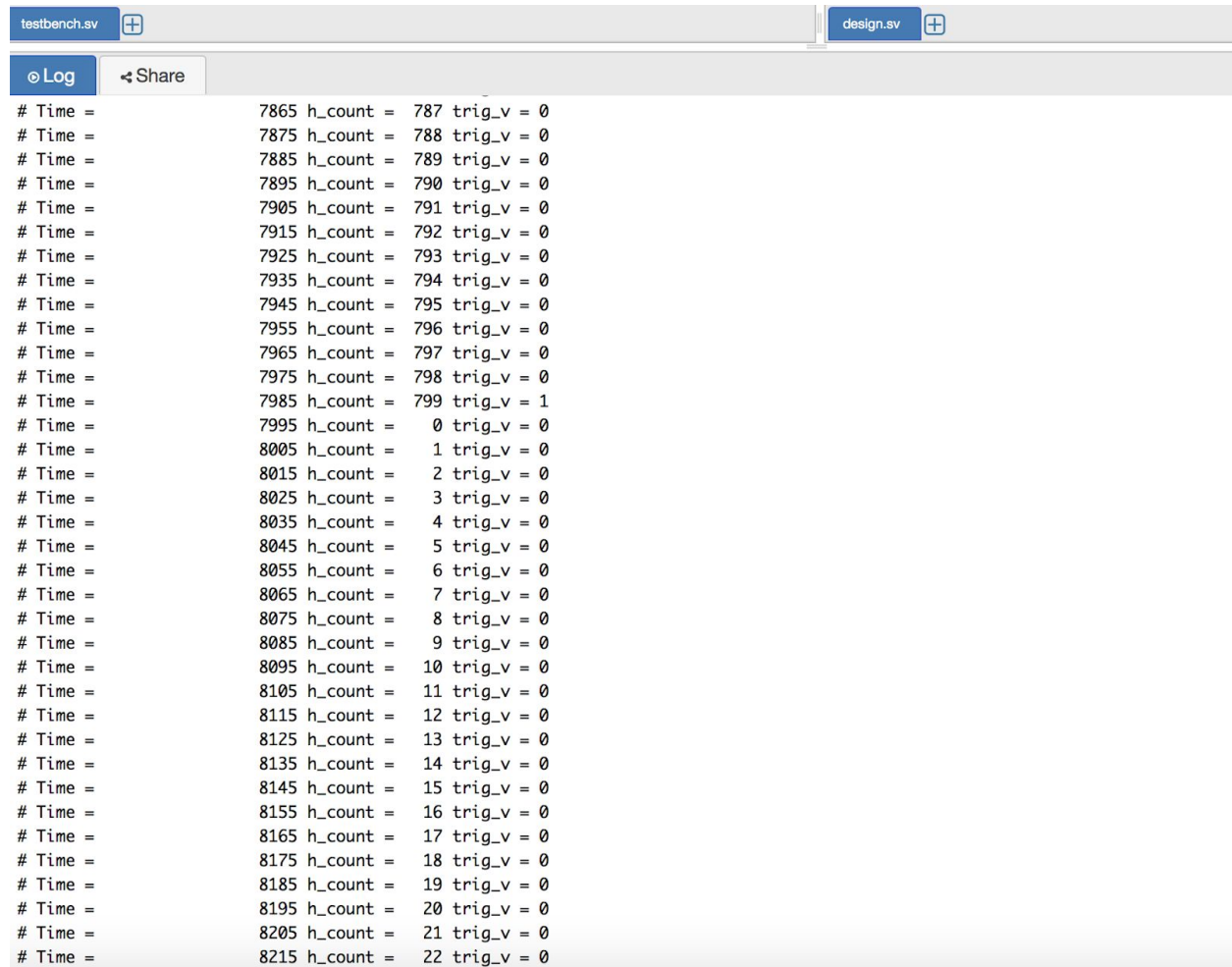
The image is a screenshot of a Verilog code editor. At the top, there are tabs for 'Package', 'Device', 'v_counter.v *', 'h_counter.v', and 'Clks_Generator.v'. Below the tabs, the file path is shown: 'C:/Users/an05918/Desktop/project_1/project_1.srcs/sources_1/new/h_counter.v'. The code is as follows:

```
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 ==799)
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:



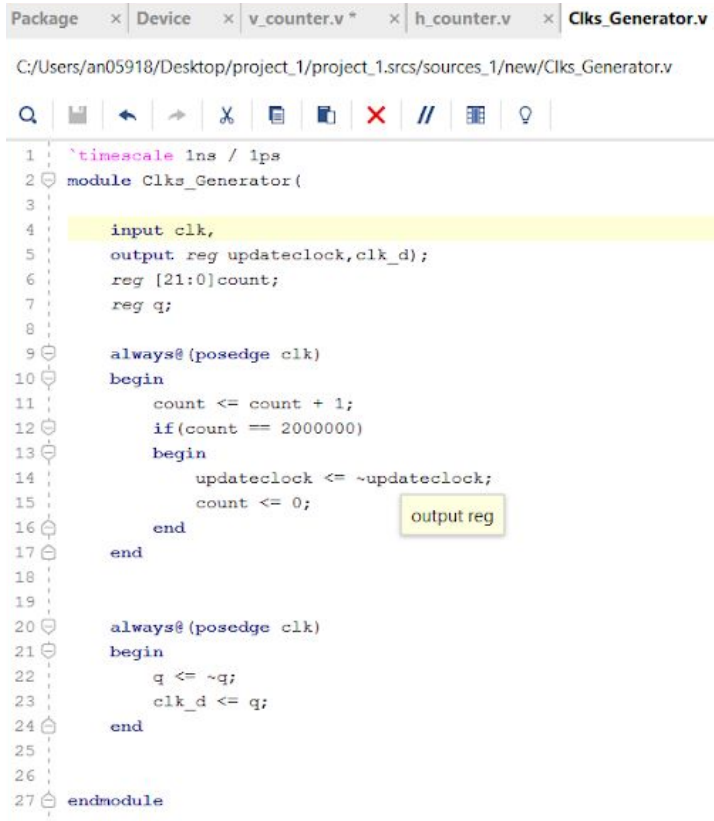
The screenshot shows the output log window of the Eda Playground. At the top, there are tabs for 'testbench.sv' and 'design.sv'. Below the tabs, there are buttons for 'Log' and 'Share'. The log window displays a series of testbench results, each line showing the current time, the value of 'h_count', and the value of 'trig_v'. The results show a sequence of values for 'h_count' from 786 to 821, and 'trig_v' is 0 for most values, but 1 for 'h_count' = 799. The log window is scrollable, and the current view shows the last 22 results.

```
# 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:



The screenshot shows a Verilog code editor with the following content:

```
Package x Device x v_counter.v * x h_counter.v x Clks_Generator.v
C:/Users/an05918/Desktop/project_1/project_1.srcs/sources_1/new/Clks_Generator.v

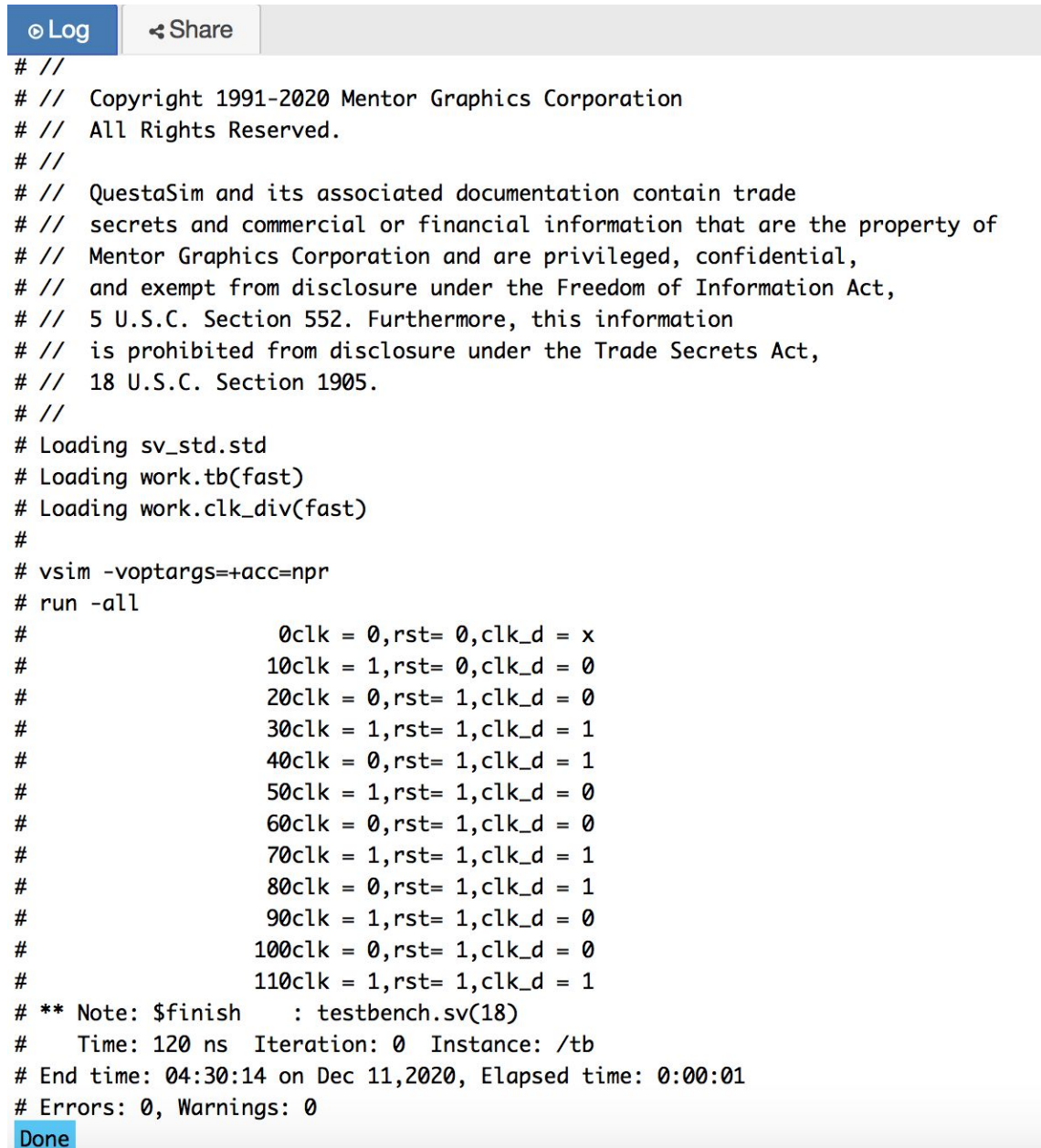
1 `timescale 1ns / 1ps
2 module Clks_Generator(
3
4     input clk,
5     output reg updateclock, clk_d);
6     reg [21:0] count;
7     reg q;
8
9     always@(posedge clk)
10    begin
11        count <= count + 1;
12        if(count == 2000000)
13        begin
14            updateclock <= ~updateclock;
15            count <= 0;
16        end
17    end
18
19
20    always@(posedge clk)
21    begin
22        q <= ~q;
23        clk_d <= q;
24    end
25
26 endmodule
```

Annotations in the image include a yellow highlight on line 4, a yellow box around the 'output reg' text on line 5, and a yellow box around the 'end' keyword on line 16.

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:

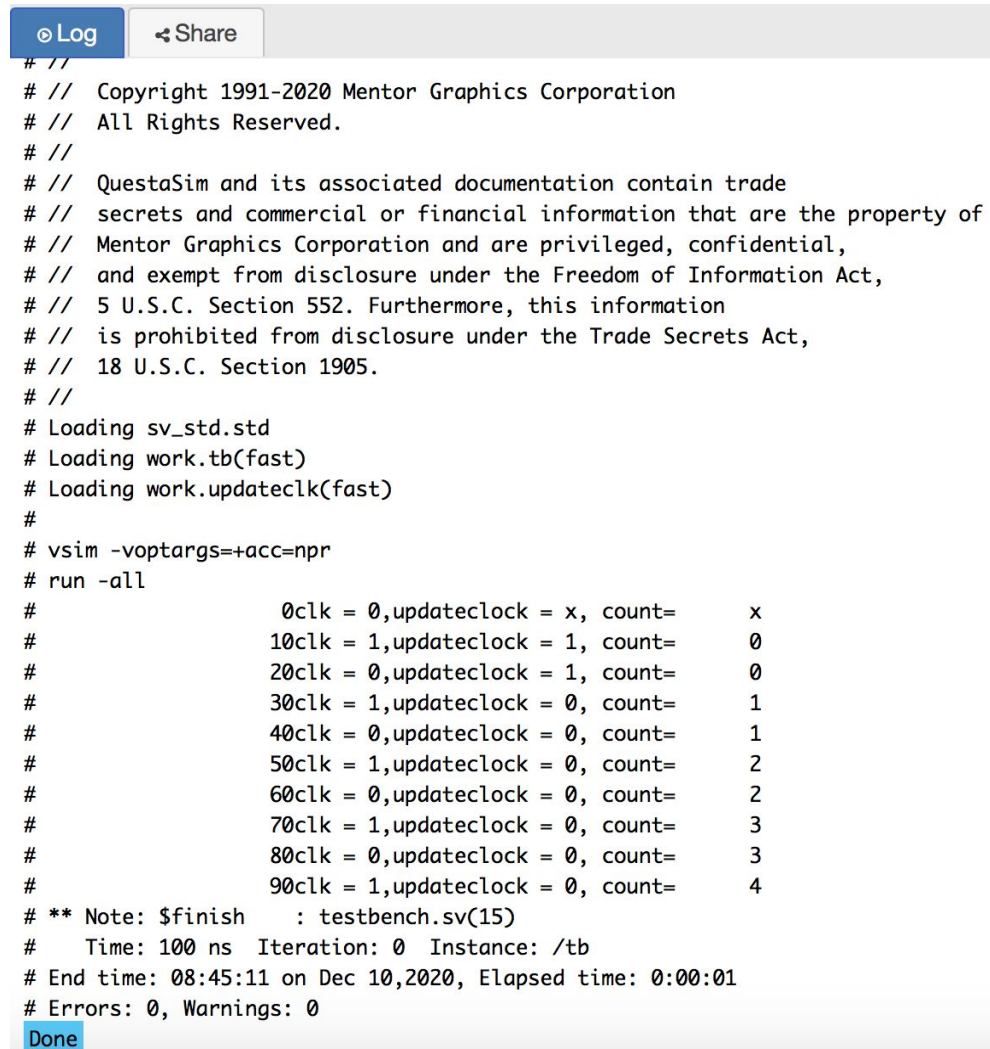
The image shows a screenshot of the Eda Playground web interface. At the top, there are two buttons: 'Log' (active) and 'Share'. Below the buttons, the log window displays the following text:

```
# //  
# // 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:



The screenshot shows the Eda Playground interface with a log window. At the top, there are buttons for 'Log' and 'Share'. The log window displays the following text:

```
# //  
# // 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
```

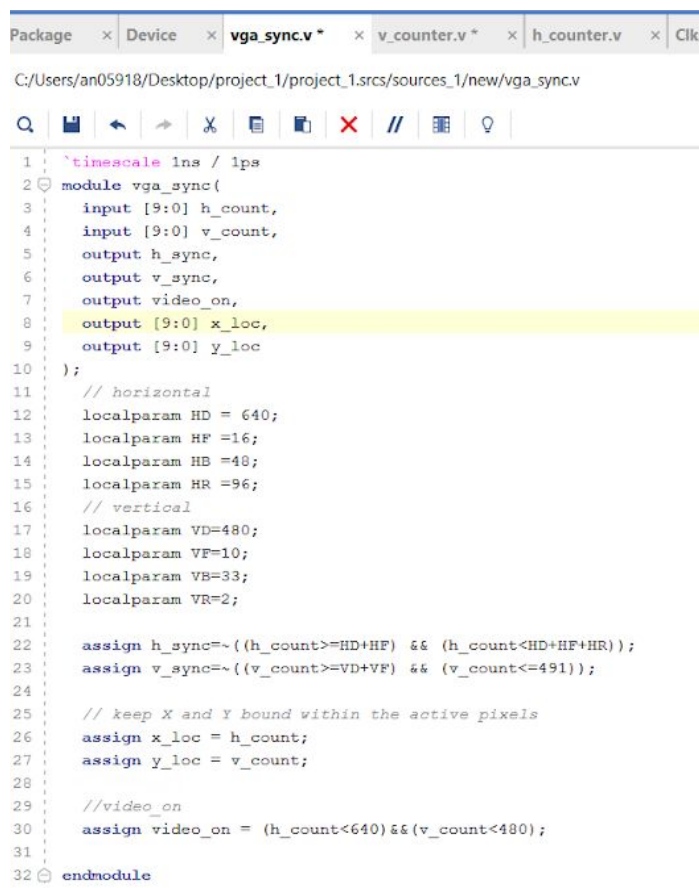
Module name : vga_sync

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:


```

Package x Device x vga_sync.v * x v_counter.v * x h_counter.v x Clk
C:/Users/an05918/Desktop/project_1/project_1.srscs/sources_1/new/vga_sync.v

1 `timescale 1ns / 1ps
2 module vga_sync(
3     input [9:0] h_count,
4     input [9:0] v_count,
5     output h_sync,
6     output v_sync,
7     output video_on,
8     output [9:0] x_loc,
9     output [9:0] y_loc
10 );
11 // horizontal
12 localparam HD = 640;
13 localparam HF = 16;
14 localparam HB = 48;
15 localparam HR = 96;
16 // vertical
17 localparam VD=480;
18 localparam VF=10;
19 localparam VB=33;
20 localparam VR=2;
21
22 assign h_sync=~((h_count>=HD+HF) && (h_count<HD+HF+HR));
23 assign v_sync=~((v_count>=VD+VF) && (v_count<=VD+VF+VR));
24
25 // keep X and Y bound within the active pixels
26 assign x_loc = h_count;
27 assign y_loc = v_count;
28
29 //video_on
30 assign video_on = (h_count<640)&&(v_count<480);
31
32 endmodule

```

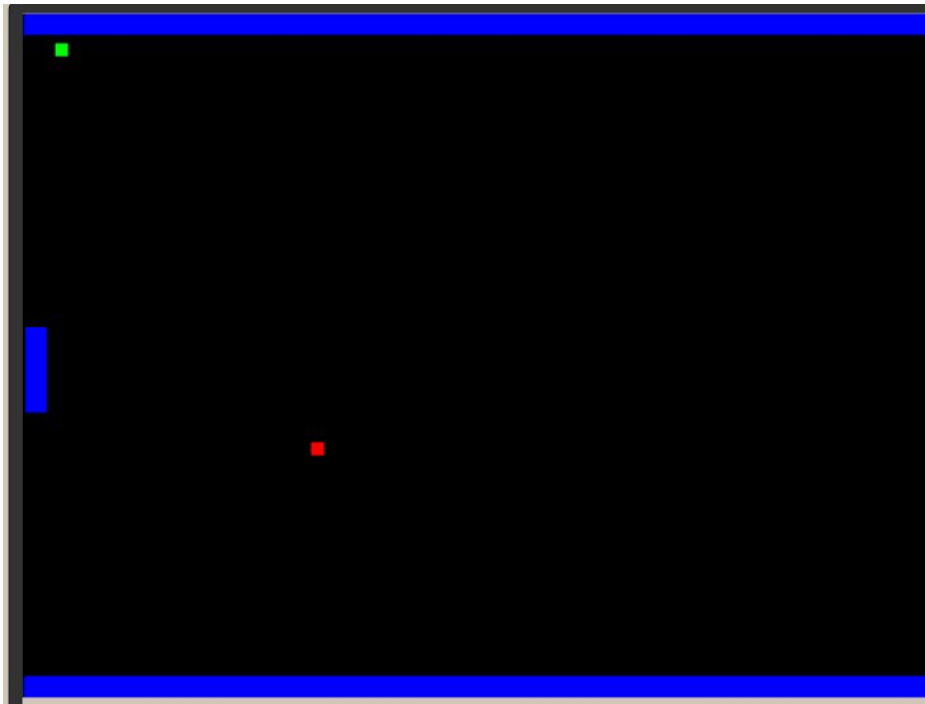

Link to eda playground module design and testbench for vga_sync:

<https://edaplayground.com/x/ZqAW>

Snippet :

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

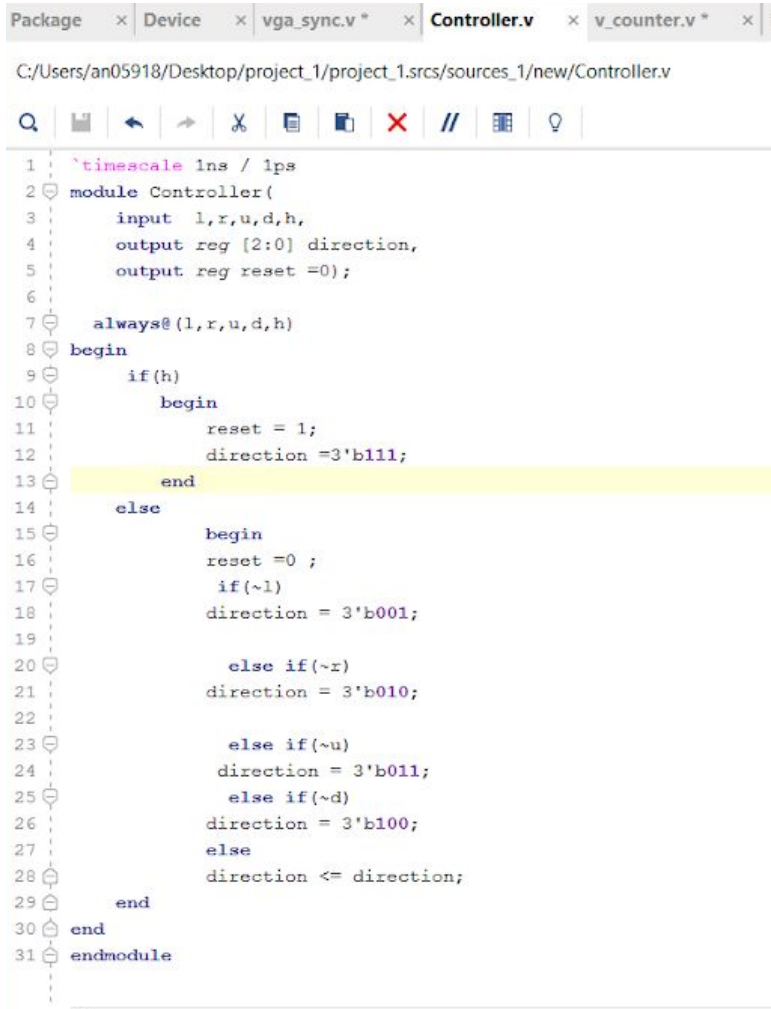
Red block signifying the prey.



Module name: Controller

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

Snippet:



The screenshot shows a Verilog code editor with the following content:

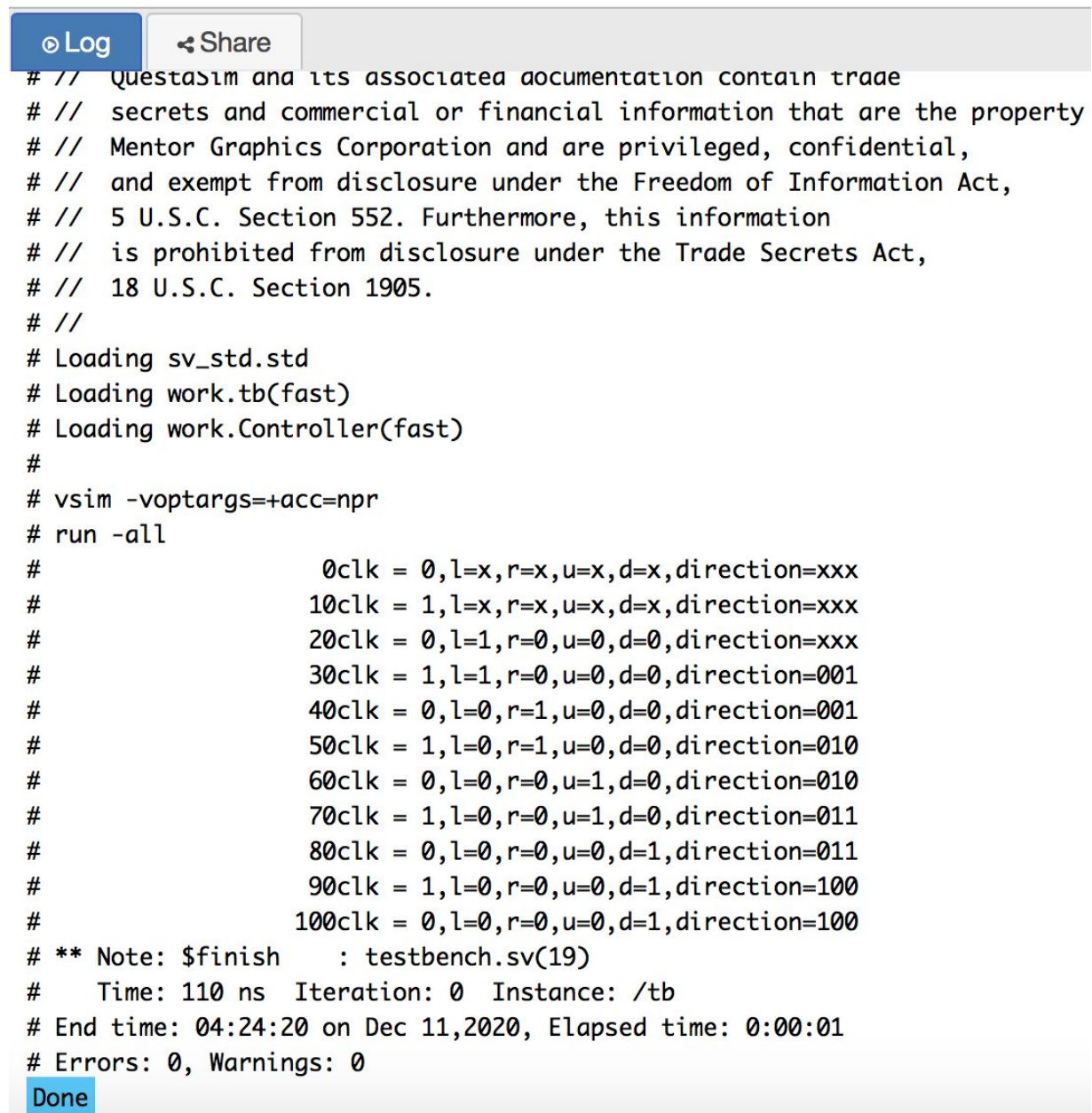
```
Package x Device x vga_sync.v * x Controller.v x v_counter.v * x t
C:/Users/an05918/Desktop/project_1/project_1.srscs/sources_1/new/Controller.v

1 `timescale 1ns / 1ps
2 module Controller(
3     input  l,r,u,d,h,
4     output reg [2:0] direction,
5     output reg reset =0);
6
7     always@(l,r,u,d,h)
8     begin
9         if(h)
10            begin
11                reset = 1;
12                direction =3'b111;
13            end
14        else
15            begin
16                reset =0 ;
17                if(~l)
18                    direction = 3'b001;
19
20                else if(~r)
21                    direction = 3'b010;
22
23                else if(~u)
24                    direction = 3'b011;
25                else if(~d)
26                    direction = 3'b100;
27                else
28                    direction <= direction;
29            end
30    end
31 endmodule
```

Link to eda playground module design and testbench for Controller:

<https://edaplayground.com/x/a5d5>

Snippet of output log window to verify functionality:



The screenshot shows the output log window of the Eda Playground. At the top, there are two buttons: "Log" (active) and "Share". The log content is as follows:

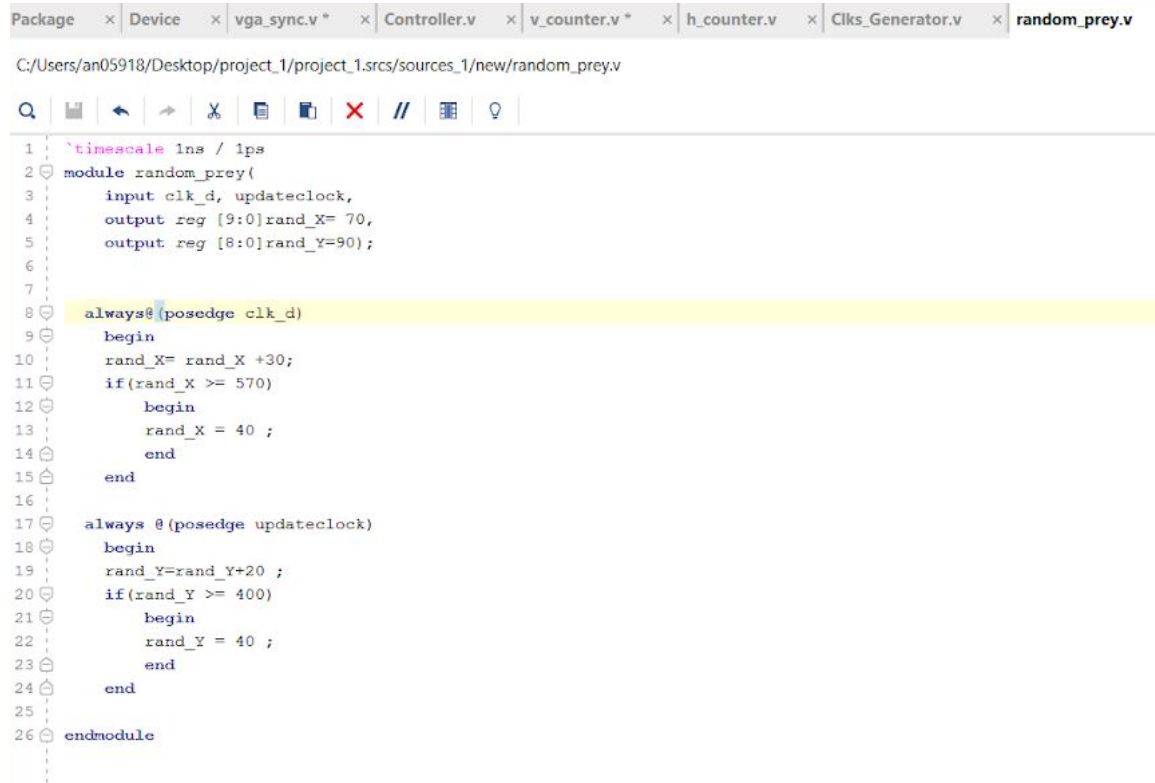
```
# // Questasim and its associated documentation contain trade
# // secrets and commercial or financial information that are the property
# // Mentor Graphics Corporation and are privileged, confidential,
# // and exempt from disclosure under the Freedom of Information Act,
# // 5 U.S.C. Section 552. 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
```

At the bottom left of the log window, there is a blue button labeled "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 :



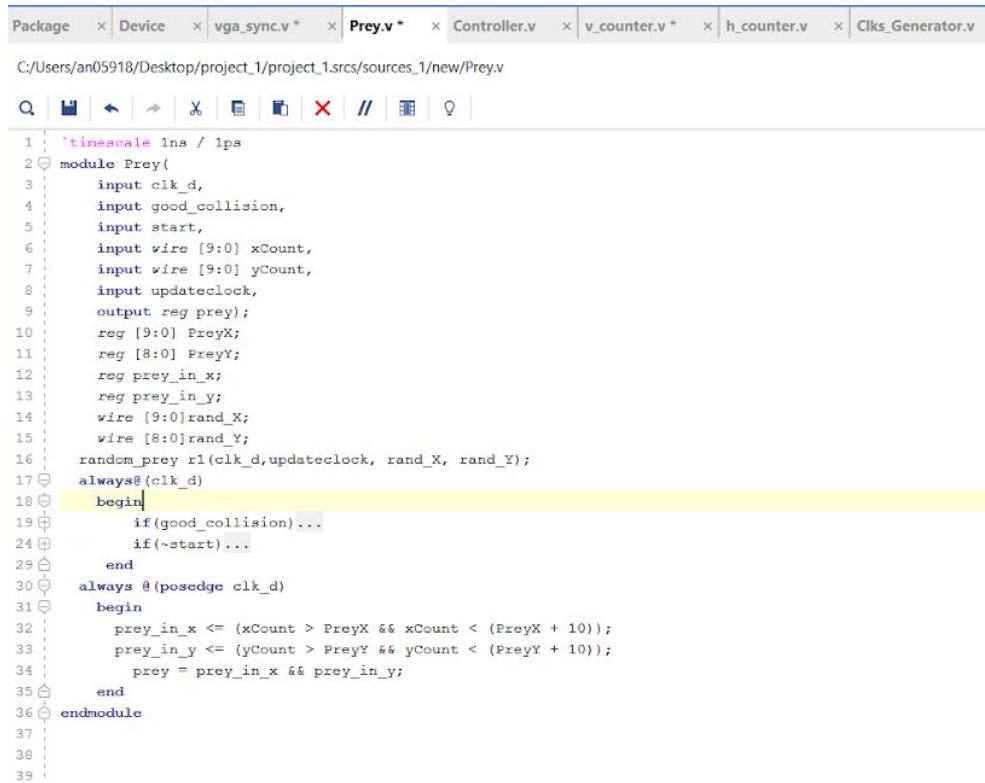
The screenshot shows a Verilog code editor with multiple tabs open: Package, Device, vga_sync.v *, Controller.v, v_counter.v *, h_counter.v, Clks_Generator.v, and random_prey.v. The active tab is random_prey.v, showing the following code:

```
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_pre module and collaborates the collision and regeneration of random prey on the screen .

Snippet:



The screenshot shows a Verilog code editor with the following tabs: Package, Device, vga_sync.v, Prey.v (active), Controller.v, v_counter.v, h_counter.v, and Clks_Generator.v. The file path is C:/Users/an05918/Desktop/project_1/project_1.srscs/sources_1/new/Prey.v. The code defines a module Prey with inputs clk_d, good_collision, start, xCount, yCount, and updateclock, and an output reg prey. It includes registers for PreyX, PreyY, prey_in_x, and prey_in_y, and wires for rand_X and rand_Y. The module instantiates a random_pre module and contains logic for collision detection and prey regeneration.

```

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_pre 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

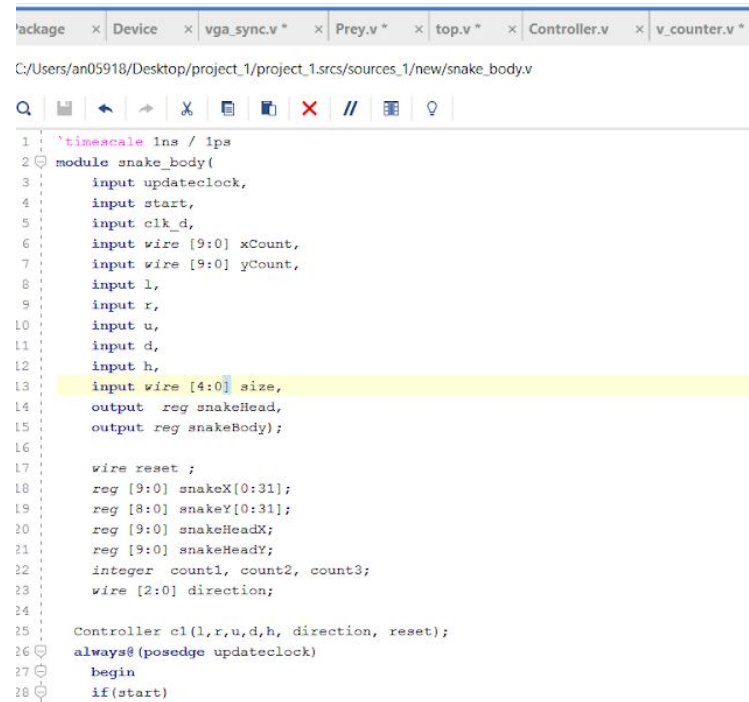
```

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

The image is a screenshot of a Verilog code editor. At the top, there is a tab bar with several open files: 'ackage', 'Device', 'vga_sync.v *', 'Prey.v *', 'top.v *', 'Controller.v', and 'v_counter.v *'. Below the tab bar, the file path 'C:/Users/an05918/Desktop/project_1/project_1.srcs/sources_1/new/snake_body.v' is displayed. The main area shows the Verilog code for the 'snake_body' module. The code includes a timescale, module definition, input/output declarations, register and wire declarations, and an instantiation of the 'Controller' module. The line 'input wire [4:0] size,' is highlighted in yellow. Line numbers 1 through 28 are visible on the left side of the code.

```
1 `timescale 1ns / 1ps
2 module snake_body(
3     input updateclock,
4     input start,
5     input clk_d,
6     input wire [9:0] xCount,
7     input wire [9:0] yCount,
8     input l,
9     input r,
10    input u,
11    input d,
12    input h,
13    input wire [4:0] size,
14    output reg snakeHead,
15    output reg snakeBody);
16
17    wire reset ;
18    reg [9:0] snakeX[0:31];
19    reg [8:0] snakeY[0:31];
20    reg [9:0] snakeHeadX;
21    reg [9:0] snakeHeadY;
22    integer count1, count2, count3;
23    wire [2:0] direction;
24
25    Controller c1(l,r,u,d,h, direction, reset);
26    always@(posedge updateclock)
27    begin
28        if(start)
```

DLD PROJECT REPORT : GROUP X

```
29 begin
30     if(direction != 3'b111)begin
31         for(count1 = 31; count1 > 0; count1 = count1 - 1)
32             begin
33                 if(count1 <= size - 1)
34                     begin
35                         snakeX[count1] = snakeX[count1 - 1];
36                         snakeY[count1] = snakeY[count1 - 1];
37                     end
38                 end
39             end
40         case(direction)
41             3'b001: snakeY[0] <= (snakeY[0] - 10);
42             3'b010: snakeX[0] <= (snakeX[0] - 10);
43             3'b011: snakeY[0] <= (snakeY[0] + 10);
44             3'b100: snakeX[0] <= (snakeX[0] + 10);
45             3'b111:begin snakeX[0] <= snakeX[0];
46                     snakeY[0] <= snakeY[0]; end
47             endcase
48         end
49     else if(~start)
50         begin
51             for(count3 = 1; count3 < 32; count3 = count3+1)
52                 begin
53                     snakeX[count3] = 700;
54                     snakeY[count3] = 500;
55                 end
56             snakeX[0] = 300;
57             snakeY[0] = 300;
58         end
59     end
60

61 always@(posedge clk_d)
62     begin
63         snakeBody = 0 ;
64
65         for(count2 = 1; count2 < size; count2 = count2 + 1)
66             begin
67
68                 if(snakeBody ==0 )
69                     snakeBody = ((xCount > snakeX[count2] && xCount < snakeX[count2]+10) && (yCount > snakeY[count2] && yCount < snakeY[count2]+10));
70
71             end
72         end
73     end
74
75
76
77
78 always@(posedge clk_d)
79     begin
80         snakeHead = (xCount > snakeX[0] && xCount < (snakeX[0]+10)) && (yCount > snakeY[0] && yCount < (snakeY[0]+10));
81     end
82
83
84 endmodule
```

Module name: collision

Description: There are 2 types of cases when collision occurs , as in good collision where the snakehead collides with the prey resulting in incrementing the snake body length. On the other hand if it is a bad collision , 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:

```

1  `timescale 1ns / 1ps
2  module collision(
3      input border,
4      input clk_d,
5      input updateclock,
6      input start,
7      input wire [9:0] xCount,
8      input wire [9:0] yCount,
9      input l,
10     input r,
11     input u,
12     input d,
13     input h,
14     output wire snakeBody,
15     output wire snakeHead,
16     output reg GameOver,
17     output wire prey,
18     output reg[6:0] seg1,
19     output reg[6:0] seg2);
20
21
22     reg good_collision, bad_collision;
23     reg azab = 1;
24     reg [4:0] size =1;
25     snake_body s1(updateclock,start,clk_d,xCount,yCount,l,r,u,d,h,size,snakeHead,snakeBody);
26     Prey p1(clk_d,good_collision,start,xCount,yCount,updateclock,prey);
27
28     reg lethal, nonLethal ;
29     always @(posedge clk_d)
30         lethal = (border|| snakeBody)&& snakeHead ;
31     always @(posedge clk_d)
32         nonLethal = prey && snakeHead && azab;

```

DLD PROJECT REPORT : GROUP X

```
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:

```
Package | Device | vga_sync.v* | Prey.v* | top.v* | Controller.v | v_counter.v* | h_counter.v | Clks_Generator.v | random_pre.v | testbench_vga.v | snake_bc | ? |
C:/Users/an05918/Desktop/project_1/project_1srcs/sources_1/new/top.v

1 | `timescale 1ns / 1ps
2 | module top(
3 |     input clk,
4 |     input r,
5 |     input t,
6 |     input u,
7 |     input d,
8 |     input h,
9 |     input start,
10 |     output reg[3:0]red, green, blue,
11 |     output haync,vaync,clk_d, blank_n,
12 |     output wire [6:0]seg1,
13 |     output wire [6:0]seg2);
14 |
15 |     wire [9:0] c_h;
16 |     wire [9:0] c_v;
17 |     wire p,q,r,s;
18 |     wire R;
19 |     wire G;
20 |     wire B;
21 |     wire snakeHead,snakeBody;
22 |     wire GameOver;
23 |     reg border;
24 |     wire play;
25 |     wire [9:0] a,b;
26 |     assign p<= clk_d;
27 |     Clks_Generator CLKs(.clk(clk),.updateClock(a),.clk_d(p));
28 |     h_counter c3(.clk(p),.h_count(c_h),.trig_v(q));
29 |     v_counter c4(.clk(p),.enable_v(q),.v_count(c_v));
30 |     vga_sync c5(.h_count(c_h),.v_count(c_v),.h_sync(haync),.v_sync(vaync),.video_on(s),.x_loc(a),.y_loc(b));
31 |
32 |
33 |
34 |     collision col(.snakeBody(snakeBody),.snakeHead(snakeHead),.border(border),.GameOver(GameOver),.clk_d(p),.updateClock(a),.start(start),.xCount(c_h),.yCount(c_v),.l(1),.r);
35 |
36 |     always @(posedge p)
37 |     begin
38 |         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)))
39 |     end
40 |     assign R = (r && (~ prey || GameOver));
41 |     assign G = (r && ((snakeBody || snakeHead || border) && ~GameOver));
42 |     assign B = (r && ~GameOver);
43 |     always@(posedge clk_d)
44 |     begin
45 |         red = {4[R]};
46 |         green = {4[G]};
47 |         blue = {4[B]};
48 |     end
49 | endmodule
```

Testbench for Top module:

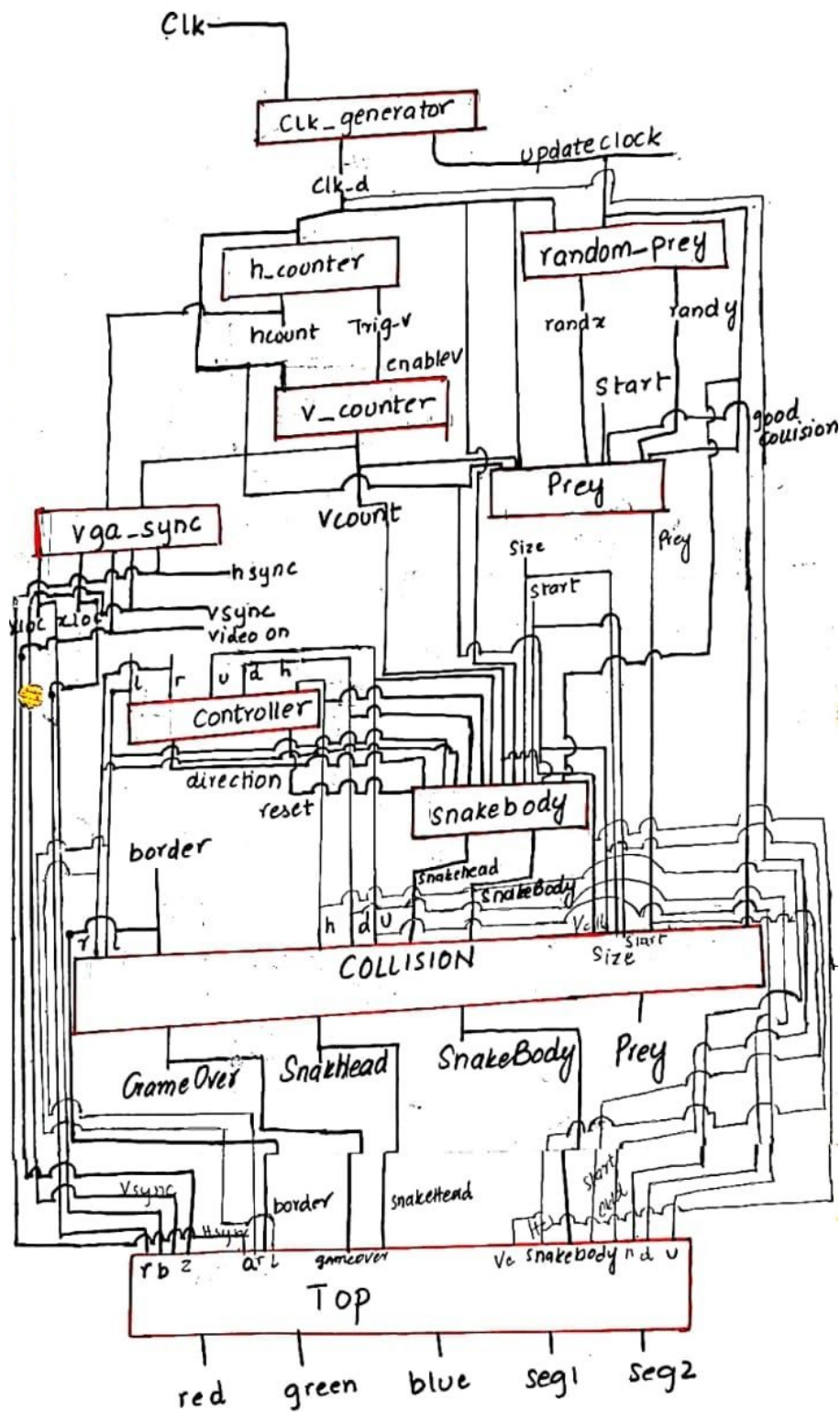
The screenshot shows a Verilog testbench file named `testbench_vga.v` in a simulation environment. The top of the window displays a list of open files: `Package`, `Device`, `vga_sync.v`, `Prey.v`, `top.v`, `Controller.v`, `v_counter.v`, `h_counter.v`, and `Cl`. The file path is `C:/Users/an05918/Desktop/project_1/project_1.srscs/sim_1/new/testbench_vga.v`. The code is as follows:

```

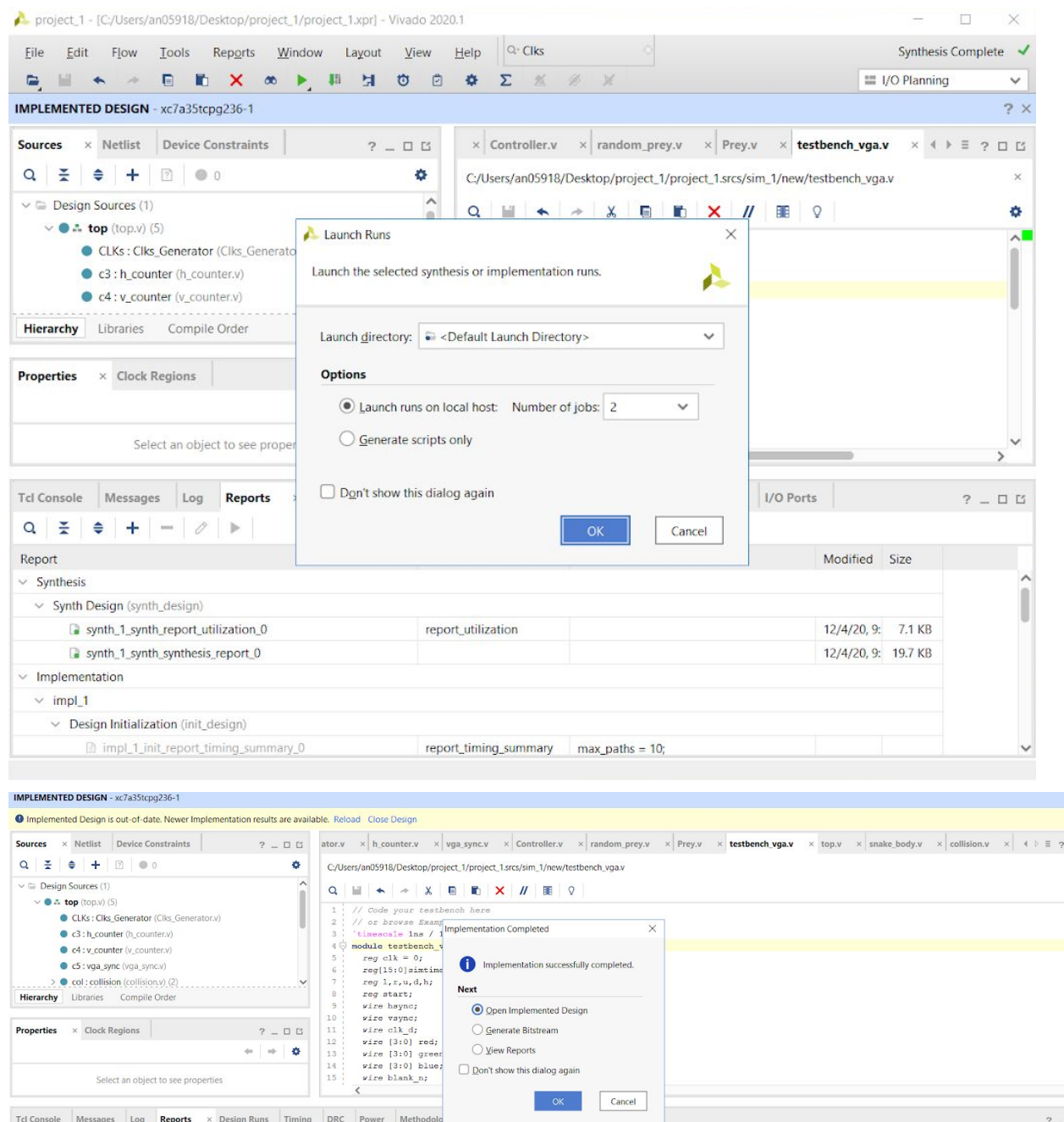
1  `timescale 1ns / 1ps
2  module testbench_vga();
3      reg clk = 0;
4      reg[15:0] simtime = 16'd5;
5      reg l,r,u,d,h;
6      reg start;
7      wire hsync;
8      wire vsync;
9      wire clk_d;
10     wire [3:0] red;
11     wire [3:0] green;
12     wire [3:0] blue;
13     wire blank_n;
14     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         #500000000 $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_preymodule 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.