Project report on

# Snake and Apple game using Verilog

Submitted by:

Group No. : 1


**Group Members:**

1. Jjateen Gundesha BT22ECI002
2. Ayush Karapagale BT22ECI003
3. Ayush Ambatkar BT22ECI005
4. Nirbhay Raut BT22ECI006


Project report submitted for course

ECL 106: Digital system design with HDL

Submission Date: 03/07/2023


Department of Electronics and Communication Engineering

भारतीय सूचना प्रौद्योगिकी संस्थान, नागपुर

Indian Institute of Information Technology, Nagpur

Table of Contents:

## Chapter 1: Introduction

Our aim for the project was to create a game of snake and apple using Verilog and program a FPGA;

It is a very simple game where we use a virtual snake to collect or eat virtual Apples, which appear on the screen randomly, with the length of snake increasing each time it intakes an apple it becomes more difficult to control and score more points.

It is primarily an endless game but the player is considered lost if the snake accidently collides with either walls or itself.

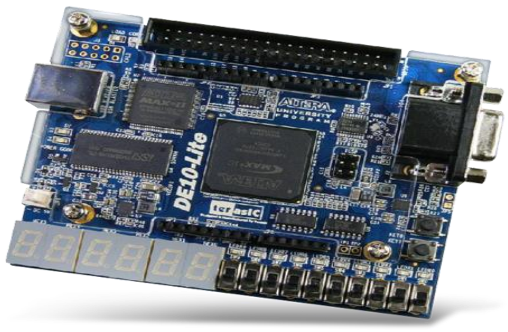
### What is used to execute the project?

- Altera DE10-lite FPGA
- VGA cable and Monitor screen (To display the gameplay)
- Push buttons, resistors and wires (To make an controller)
- USB cable and power cable
- Quartus II Software (To edit and write Verilog code)

### Brief about the components:

- **DE10 lite FPGA**

The Altera DE10-Lite is a development kit designed by Terasic based on the Intel/Altera Cyclone V FPGA. FPGA stands for Field-Programmable Gate Array, which is a programmable logic device that allows users to implement digital circuits



The DE10-Lite board includes different types of memory for storing data. It has 64MB of SDRAM (SDRAM), and 128MB of QSPI Flash memory. These memory options provide flexibility for storing program code and data

The board offers a range of connectivity options, including USB host/device ports, a micro SD card slot and VGA output

The DE10-Lite has a VGA connector for connecting to a monitor or display, allowing you to create graphical user interfaces or display visual information. It also has an audio connector for audio output

- **VGA Monitor**

CRT-based VGA displays use amplitude-modulated moving electron beams to display information on a phosphor-coated screen. LCD displays use an array of switches that can impose a voltage across a small amount of liquid crystal, thereby changing light permittivity through the crystal on a pixel-by- pixel basis. Although the following description is limited to CRT displays, LCD displays have evolved to use the same signal timings as CRT displays. Color CRT displays use three electron beams (one for red, one for blue, and one for green) to energize the phosphor that coats the inner side of the display end of a cathode ray tube.

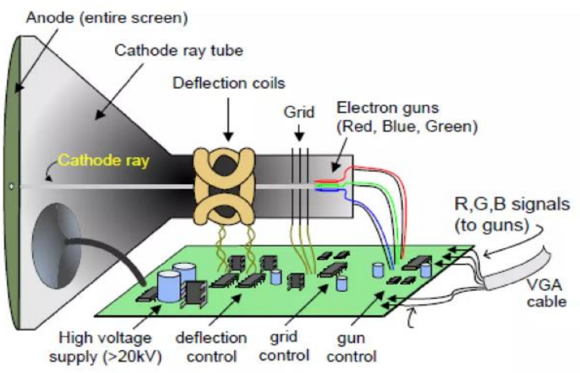



Fig 3: Colour CRT Display

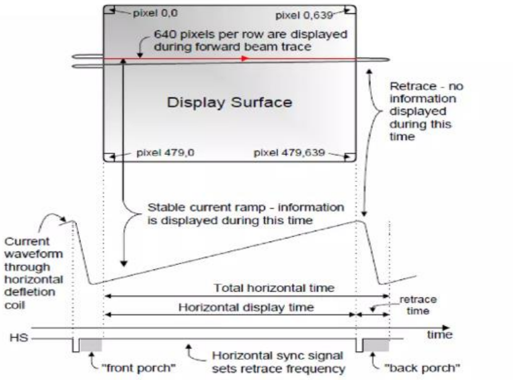
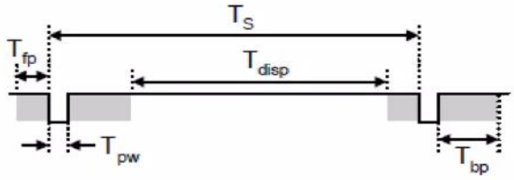



Fig 4: VGA horizontal synchronization

A VGA controller circuit must generate signals and coordinate the delivery of video data based on the pixel clock. The pixel clock defines the time available to display one pixel of information. The VS signal defines the "refresh" frequency of the display, or the frequency at which all information on the display is redrawn.

The minimum refresh frequency is a function of the display's phosphor and electron beam intensity, with practical refresh frequencies falling in the 50Hz to 120Hz range. The number of lines to be displayed at a given refresh frequency defines the horizontal "retrace" frequency.

For a 640-pixel by 480-row display using a 25MHz pixel clock and 60 +/-1Hz refresh, the signal timings can be derived. Timings for sync pulse width and front and back porch intervals (porch intervals are the pre- and post-sync pulse times during which information cannot be displayed) are based on observations taken from actual VGA displays.



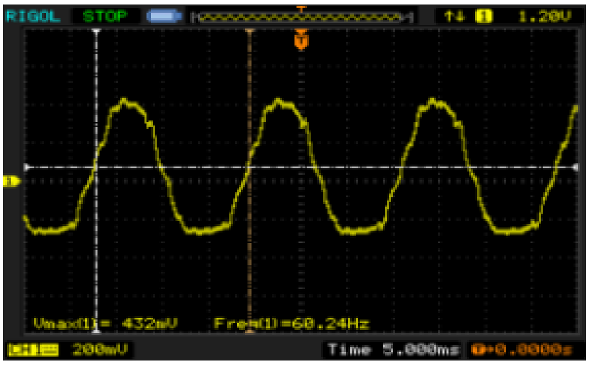

| Symbol | Parameter | Vertical Sync | | | Horiz. Sync | |
|---|---|---|---|---|---|---|
| | | Time | Clocks | Lines | Time | Clks |
| $T_S$ | Sync pulse | 16.7ms | 416,800 | 521 | 32 us | 800 |
| $T_{disp}$ | Display time | 15.36ms | 384,000 | 480 | 25.6 us | 640 |
| $T_{pw}$ | Pulse width | 64 us | 1,600 | 2 | 3.84 us | 96 |
| $T_{fp}$ | Front porch | 320 us | 8,000 | 10 | 640 ns | 16 |
| $T_{bp}$ | Back porch | 928 us | 23,200 | 29 | 1.92 us | 48 |

Figure 5: Signal timings for a 640-pixel by 480 row display using a 25MHz pixel clock and 60Hz vertical refresh.

- **Push Button & Significance of INPUT_PULLUP:**

Consider a circuit, which is a simple normally-open push-button on Breadboard. It has been wired so that one side is tied to GND and the Other side is connected to a Pin We will find that random garbage values appear on serial monitor and fluctuate even when we bring our fingers closer. The pin is "Floating."

This means that any signals in the air, such as from nearby electronics, can cause the pin to "float" to either a HIGH or LOW. For example, this waveform from an oscilloscope shows what the pin is doing when nothing is connected.



With the oscilloscope waveform visible. As a human finger comes close to the pin, the waveform changes. 60Hz noise from the environment is being coupled into the circuit through the finger, which is what causes the "random" behaviour of the input pin. The fix for floating pins is to "pull them up" to a known value when the switch is unpressed. This is done with a Pull-Up resistor.

## Chapter 2: Verilog code

(This is the top level module and contains the fundamental game logic and instantiate the other modules for first time)

```
module snake(start, master_clk, DAC_clk, VGA_R, VGA_G, VGA_B,
VGA_hSync, VGA_vSync, blank_n,x,y,z,w,h ,seg1,seg2);
	input master_clk, x,y,z,w,h;
	output reg [3:0]VGA_R, VGA_G, VGA_B;
	output VGA_hSync, VGA_vSync, DAC_clk, blank_n ;
	wire [9:0] xCount;		wire [9:0] yCount;
	wire displayArea,  VGA_clk;
	wire R, G, B, snakeHead,snakeBody ;
	wire  game_over, apple ;
	reg  border;
	output wire [0:6]seg1, wire [0:6]seg2;
	input start;
	wire update;
```

```
collision col(snakeBody,snakeHead,border,game_over,VGA_clk,update,start,xCount,yCount,x,y,z,w,h,apple,seg1,seg2);

VGA_Controller VGA(VGA_clk, xCount, yCount, displayArea, VGA_hSync, VGA_vSync, blank_n);

	Clks_Generator CLKs(master_clk, update,VGA_clk);

	assign DAC_clk = VGA_clk;

	always @(posedge VGA_clk)

	begin

		border <= (((xCount >= 0) && (xCount < 31) || (xCount >= 610) && (xCount < 641)) || ((yCount >= 0) && (yCount < 31) || (yCount >= 450) && (yCount < 481)));

	end

	assign R = (displayArea && ( apple || game_over));

	assign G = (displayArea && ((snakeBody || snakeHead || border)&& ~game_over));

	assign B = (blank_n && ~game_over) ;

	always@(posedge VGA_clk)

	begin

		VGA_R = {4{R}};

		VGA_G = {4{G}};

		VGA_B = {4{B}};

	end

endmodule
```

(Module "Clks_Generator" reduces the clock frequency signal given by the board 50 MHz to 25MHz which is compatible for data transfer to VGA Monitor, the updated clock signal is now given by VGA_clk)

```
module Clks_Generator(master_clk, update,VGA_clk) ;
        input master_clk;
        output reg update,VGA_clk;
        reg [21:0]count, q ;
        always@(posedge master_clk)
        begin
                count <= count + 1 ;
                if(count == 2020000)
                begin
                        update <= ~update;
                        count <= 0;
                end
        end
        always@(posedge master_clk)
        begin
                q <= ~q;
                VGA_clk <= q;
        end
endmodule
```

(Module "VGA_Controller" generates the necessary signals for VGA display control, including the horizontal and vertical synchronization signals)

```
module VGA_Controller(VGA_clk, xCount, yCount, displayArea, VGA_hSync, VGA_vSync, blank_n);
```

```
input VGA_clk;
output reg [9:0]xCount, yCount;
output reg displayArea;
output VGA_hSync, VGA_vSync, blank_n;
reg p_hSync, p_vSync;
integer porchHF = 650; //start of horizntal front porch
integer syncH = 655;//start of horizontal sync
integer porchHB = 700; //start of horizontal back porch
integer maxH = 800; //total length of line.

integer porchVF = 480; //start of vertical front porch
integer syncV = 490; //start of vertical sync
integer porchVB = 492; //start of vertical back porch
integer maxV = 530; //total rows.

always@(posedge VGA_clk)
begin
	if(xCount === maxH)
		xCount <= 0;
	else
		xCount <= xCount + 1;
end
// 93sync, 46 bp, 640 display, 15 fp
// 2 sync, 33 bp, 480 display, 10 fp
always@(posedge VGA_clk)
begin
	if(xCount === maxH)
	begin
```

```
            if(yCount === maxV)
                yCount <= 0;
            else
            yCount <= yCount + 1;
        end
    end
    always@(posedge VGA_clk)
    begin
        displayArea <= ((xCount < porchHF) && (yCount <
porchVF));
    end
    always@(posedge VGA_clk)
    begin
        p_hSync <= ((xCount >= syncH) && (xCount < porchHB));
        p_vSync <= ((yCount >= syncV) && (yCount < porchVB));
    end
    assign VGA_vSync = ~p_vSync, VGA_hSync = ~p_hSync,
    blank_n = ~displayArea;
endmodule
```

(Module “Collision” defines the conditions of scoring a point (collision between snake and apple) and losing the game (collision between snake and border or body))

```
module
collision(snakeBody,snakeHead,border,game_over,VGA_clk,update,start
,xCount,yCount,x,y,z,w,h,apple,seg1,seg2);
    input border,VGA_clk,update,start,xCount,yCount,x,y,z,w,h;
    output  snakeBody,snakeHead,game_over,apple;
```

```
wire apple ,snakeBody,snakeHead ;
reg game_over, good_collision, bad_collision;
wire [9:0] xCount, [9:0] yCount;
output reg[6:0] seg1;
reg azab = 1;

output reg[6:0] seg2;
 reg [4:0] size =1;
snake_body
snake(update,start,VGA_clk,snakeHead,snakeBody,xCount,yCount,x,y,z,
w,h,size);
Apple
app(VGA_clk,good_collision,apple,start,xCount,yCount,update);
reg lethal, nonLethal ;
always @(posedge VGA_clk) lethal = (border|| snakeBody)&&
snakeHead ;
always @(posedge VGA_clk) nonLethal = apple && snakeHead
&& azab;

wire [4:0] check_size ;
assign check_size = (size-1) ;
always @(check_size)
begin
if(check_size<=9)
        seg2 <= ~7'b0111111;
else if(check_size[4:3] ==2'b01)
        seg2 <= ~7'b0000110;
```

```
else if(check_size[4:3] ==2'b10)
        seg2 <= ~7'b1011011;
else
        seg2 <= ~7'b1001111;

case (check_size[3:0] )
        0 : seg1 <= ~7'b0111111;
        1 : seg1 <= ~7'b0000110;
        2 : seg1 <= ~7'b1011011;
        3 : seg1 <= ~7'b1001111;
       4 : seg1 <= ~7'b1100110;
        5 : seg1 <= ~7'b1101101;
        6 : seg1 <= ~7'b1111101;
        7 : seg1 <= ~7'b0000111;
        8 : seg1 <= ~7'b1111111;
        9 : seg1 <= ~7'b1101111;
        default : seg1 <= ~7'bX;
endcase
end
    always @(posedge VGA_clk)
        if(nonLethal) begin
            good_collision<=1;
            size = size+1;
            azab=0 ;
        end
        else if(~start) size = 1;
```

```
        else    begin
                good_collision=0; azab =1 ;
        end

        always @(posedge VGA_clk)
                if(lethal) bad_collision=1;
                else bad_collision=0;
        always @(posedge VGA_clk) if(bad_collision) game_over<=1;
                else If(~start) game_over=0;

endmodule
```

(module “Apple” takes care of all the properties of the virtual object (apple) used in the game, like generating and managing the position of apple in game with respect to the collision events)

```
module
Apple(VGA_clk,good_collision,apple,start,xCount,yCount,update);
        input VGA_clk , good_collision,start,xCount,yCount,update;
        output reg apple ;
        reg [9:0] appleX;
        reg [8:0] appleY;
        reg apple_inX, apple_inY;
        wire [9:0]rand_X, [9:0]rand_Y;
        wire [9:0] xCount, [9:0] yCount;
random_apple appl(VGA_clk, rand_X, rand_Y,update);
        always@(VGA_clk)
        begin
```

```
		if(good_collision)
		begin
			appleX=rand_X;
			appleY=rand_Y;
		end
		if(~start)
		begin
			appleX=rand_X;
			appleY=rand_Y;
		end
	 end
	always @(posedge VGA_clk)
	begin
		apple_inX <= (xCount > appleX && xCount < (appleX + 10));
		apple_inY <= (yCount > appleY && yCount < (appleY + 10));
		apple = apple_inX && apple_inY;
	end
endmodule
```

(module “random_apple” generates new random positions for our virtual apple in game, taking collisions and borders of the play area into consideration)

```
module random_apple(VGA_clk, rand_X, rand_Y,update);
	input VGA_clk, update ;
	output reg [9:0]rand_X= 70;
```

```
    output reg [8:0]rand_Y=90;
    always@(posedge VGA_clk)
    begin
    rand_X= rand_X +30;
    if(rand_X >= 570)
        begin
        rand_X = 40 ;
        end
    end
    always @(posedge update)
    begin
    rand_Y=rand_Y+20 ;
    if(rand_Y >= 400)
        begin
        rand_Y = 40 ;
        end
    end
endmodule
```

(module “snake_body” defines and manages all the properties of snake’s body and changes it according to the events occurring in the game)

```
module
snake_body(update,start,VGA_clk,snakeHead,snakeBody,xCount,yCount
,x,y,z,w,h,size);
    input update , start,VGA_clk,xCount,yCount,x,y,w,z,h,size;
    wire[4:0] size ;
    wire reset ;
    output  snakeHead,snakeBody;
```

```
reg [9:0] snakeX[0:31];
reg [8:0] snakeY[0:31];
reg [9:0] snakeHeadX;
reg [9:0] snakeHeadY;
integer  count1, count2, count3;
reg snakehead, snakeBody;
wire [9:0] xCount, [9:0] yCount, [2:0] direction;
Controller cont(x,y,z,w,h, direction, reset);
always@(posedge update)
begin
if(start)
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 VGA_clk)
    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 VGA_clk)
    begin
```

```
                snakeHead = (xCount > snakeX[0] && xCount < (snakeX[0]+10)) && (yCount > snakeY[0] && yCount < (snakeY[0]+10));
        end
endmodule
```

(module "Controller" contains the code for the custom gamepad that we created for the game, it is a circuit containing push buttons, resistors and wires mounted on a breadboard)

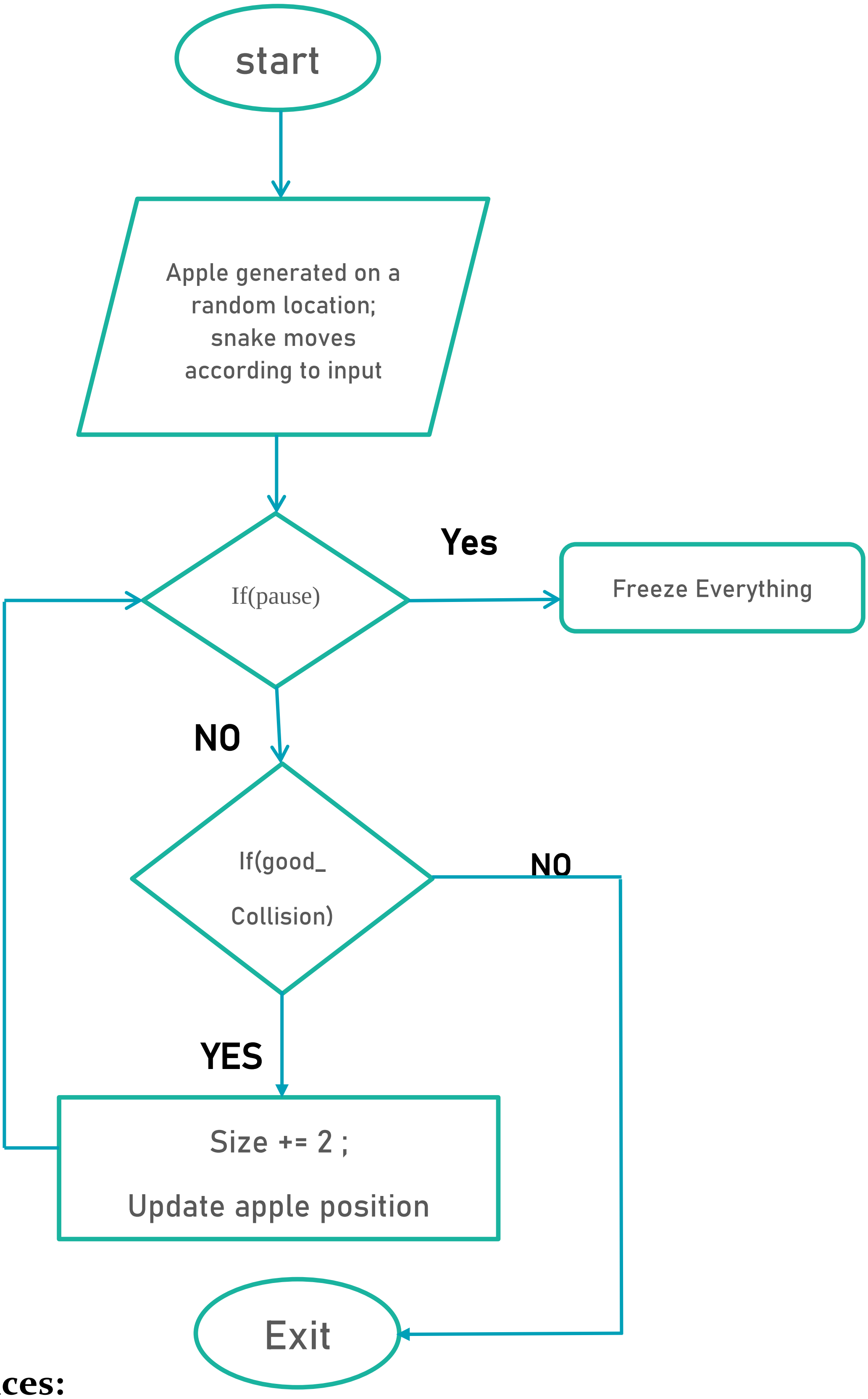
```
module Controller(x,y,z,w,h, direction, reset);
        input   x,y,z,w,h;//  W==RIGHT,X==UP,Y==LEFT,Z==DOWN, H==PAUSE Y
        output reg [2:0] direction;
        output reg reset =0;
        always@(x,y,z,w,h)
begin
         if(h)
                begin
                        reset = 1;
                        direction =3'b111;
                end
        else
                        begin
                        reset =0 ;
                        if(~x)
                                direction = 3'b001;

                        else if(~y)
```

```
                direction = 3'b010;

            else if(~z)
                direction = 3'b011;
            else if(~w)
                direction = 3'b100;
            else
                direction <= direction;
    end
end
endmodule
```

**Flowchart :**




## References:


I. https://www.instructables.com/Snake-on-an-FPGA-Verilog/

II. https://www.slideshare.net/sskrishnajith/snake-game-on-fpga-in-verilog