CS223 Project Report

Melike KARA
222303094
Section 6

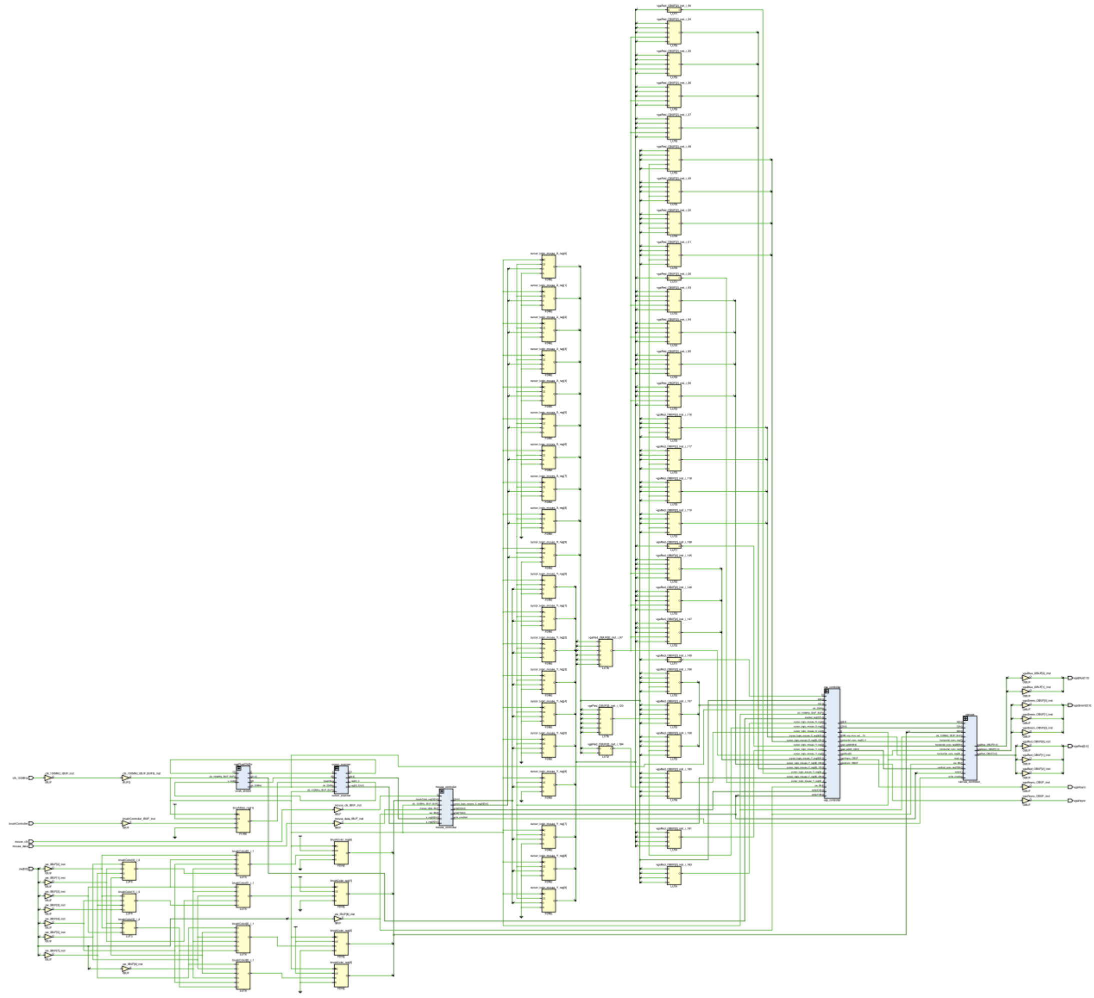
My CS 223 term project involves creating a VGA controller and drawing canvas, with the goal of using the Basys 3 board to display and interact with graphics with different colors(8-bit RGB) controlled by switches on a monitor. Below is a summary of the stages and tasks of my project:

1.VGA Controller

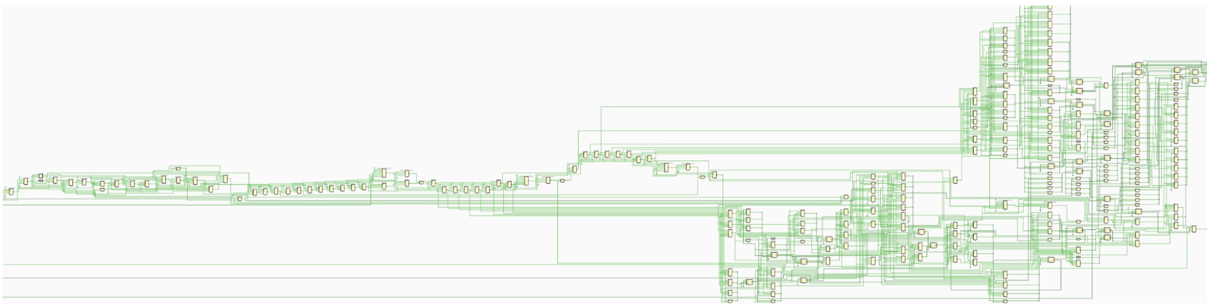
a. **Spesifications**
   Resolution: 640 × 480 pixels
b. **Timing and Synchronization:**
   - Pixel Clock: 25 MHz
   - HSYNC and VSYNC:
     - HSYNC (Horizontal Sync) is active for 640 pixel clock cycles (one row).
     - VSYNC (Vertical Sync) is active for 480 rows (one frame).
     - Both sync signals have front porch, sync pulse, and back porch timing for both horizontal and vertical sync.
c. **Horizontal Timing:**
   - **Visible Area**: 640 pixels
   - **Front Porch**: 16 pixels (time: 0.6355 µs)
   - **Sync Pulse**: 96 pixels (time: 3.8133 µs)
   - **Back Porch**: 48 pixels (time: 1.9067 µs)
   - **Whole Line**: 800 pixels (time: 31.7776 µs)
d. **Vertical Timing:**
   - **Visible Area**: 480 lines (time: 15.2532 µs)
   - **Front Porch**: 10 lines (time: 0.3178 µs)
   - **Sync Pulse**: 2 lines (time: 0.0636 µs)
   - **Back Porch**: 33 lines (time: 1.0487 µs)
   - **Whole Frame**: 525 lines (time: 16.6832 µ)

2.RTL Schematics

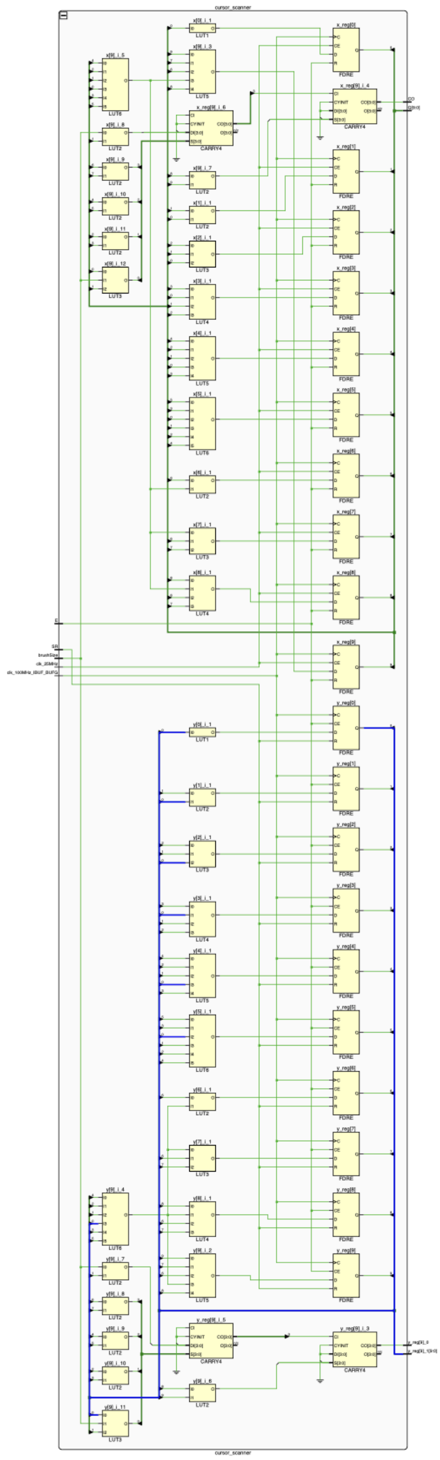
a. project_main



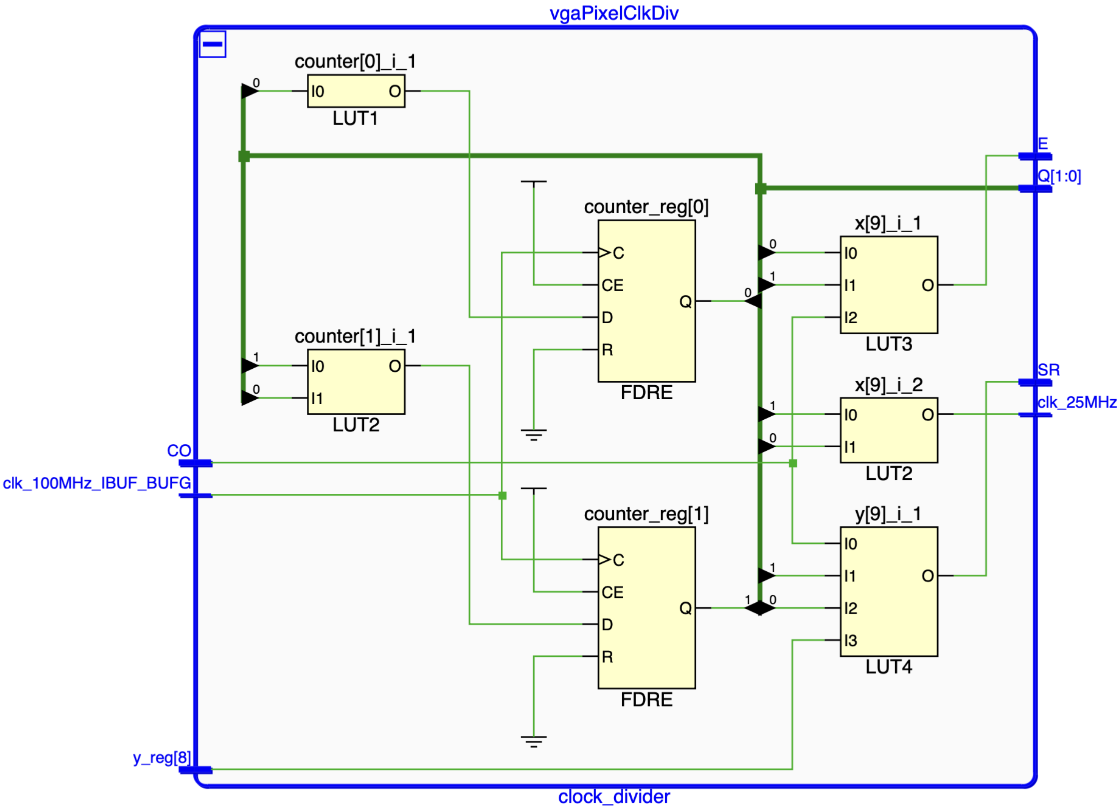
b. mouseController



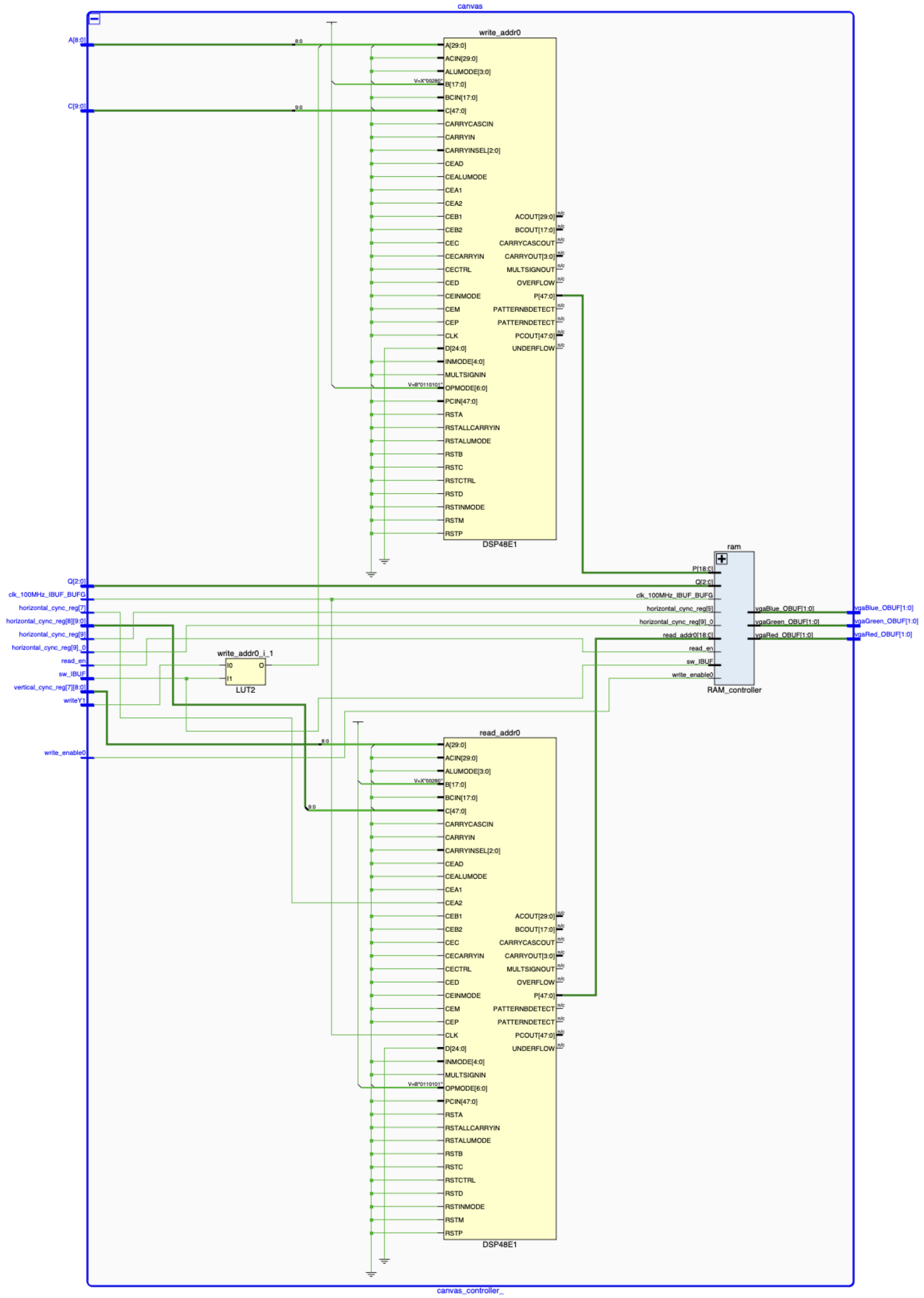
c. cursorScanner



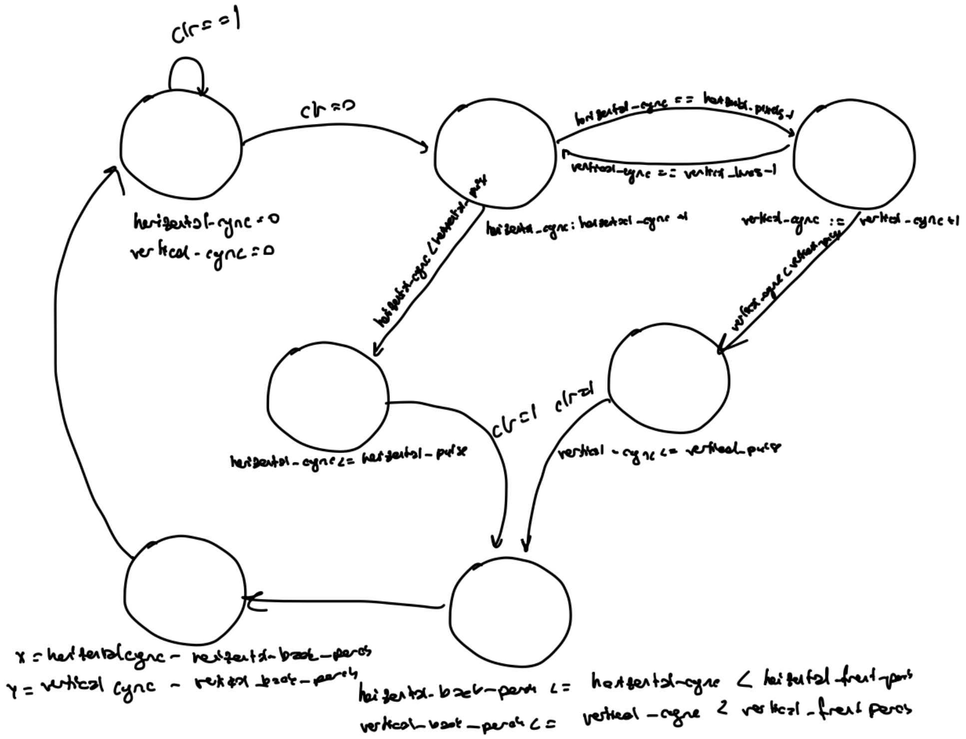
d. clockDivider

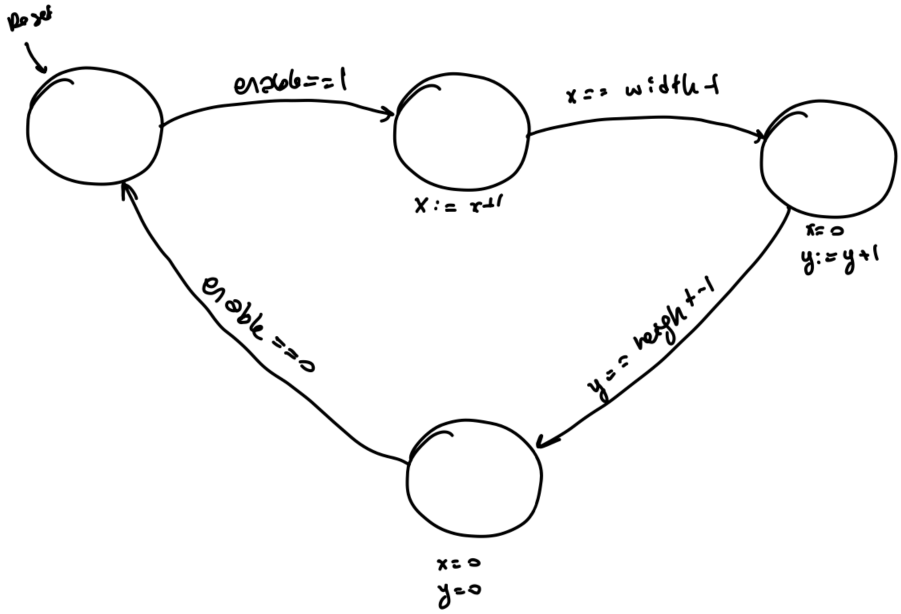


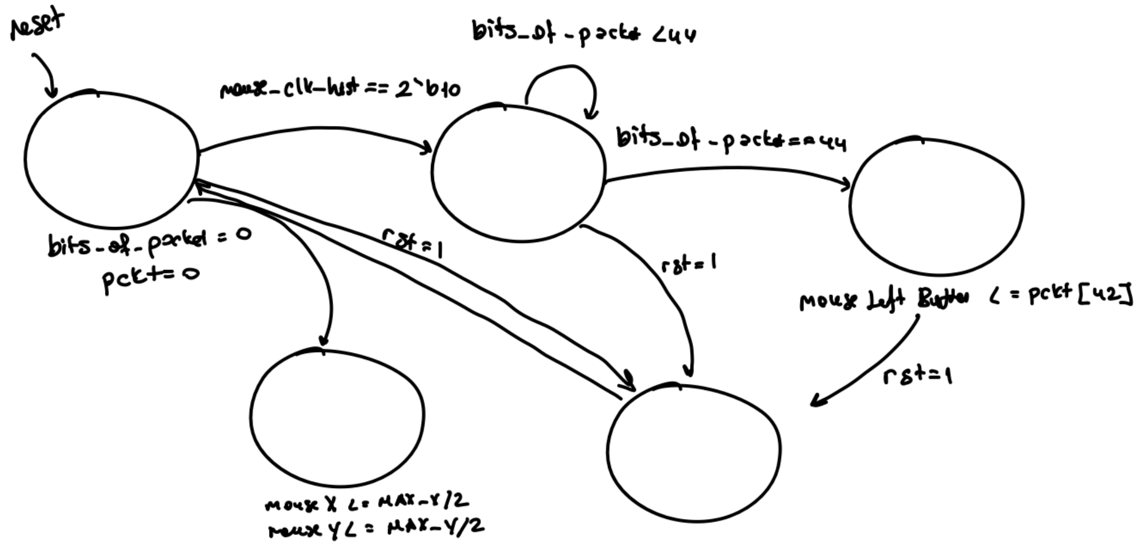
e. canvasController

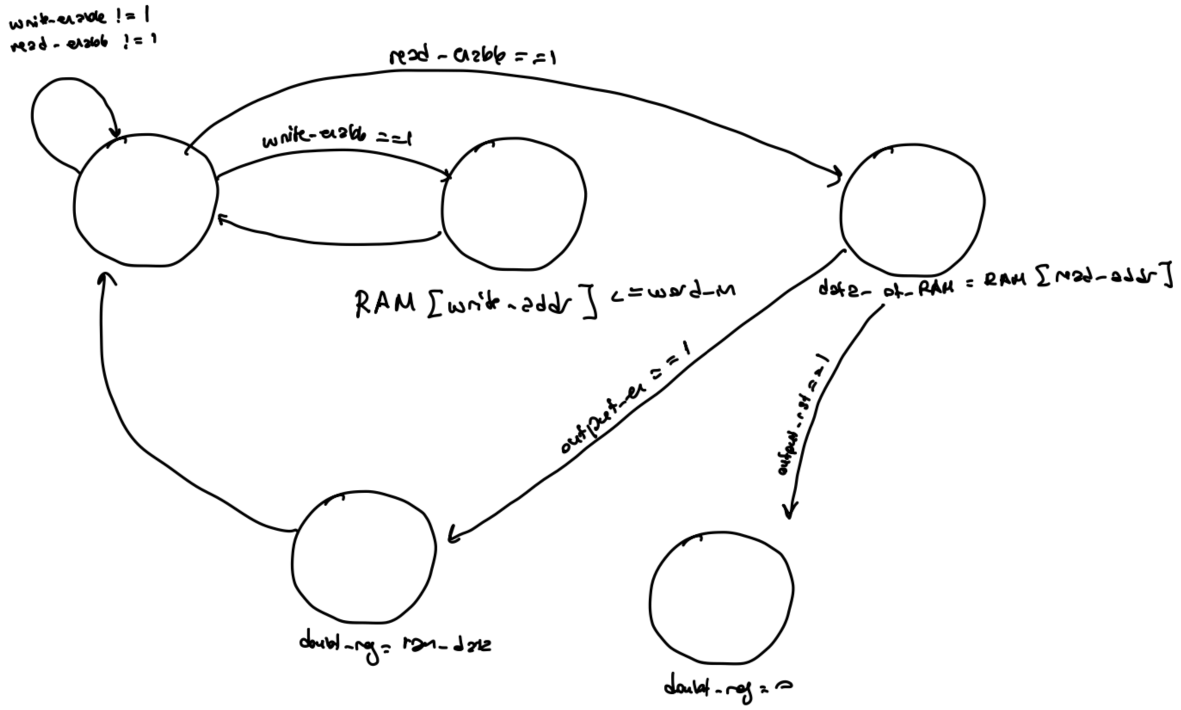


3. State Diagrams

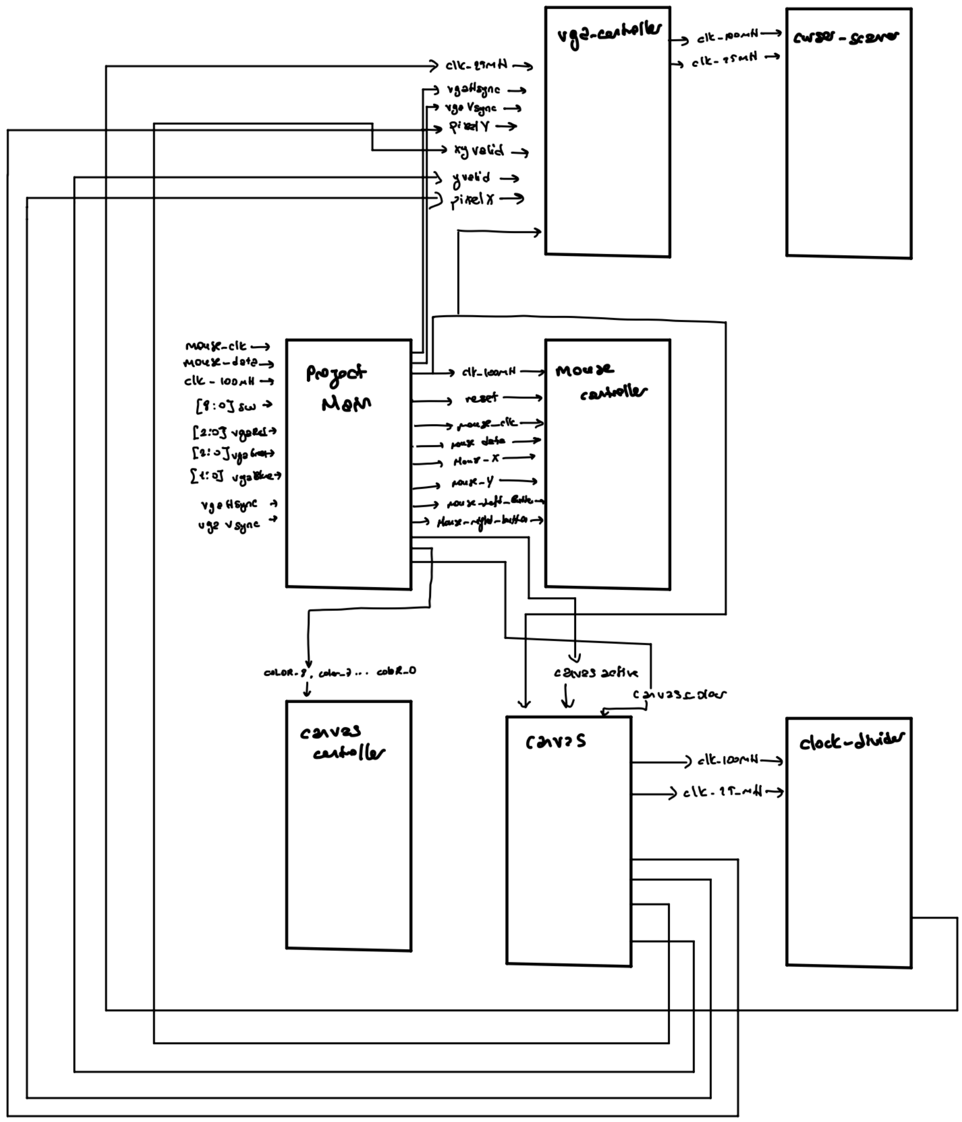





Reset
mouse_clk_hist == 2`b10
bits_of_packet <44
bits_of_packet==44
bits_of_packet = 0
pckt=0
rst=1
rst=1
Mouse Left Button <= pckt[42]
rst=1
mouse X <= MAX_X/2
mouse Y <= MAX_Y/2


write_enable != 1
read_enable != 1
read_enable ==1
write_enable ==1
RAM [write_addr] <=word_in
data_of_RAM = RAM [read_addr]
output_en == 1
output_rst == 1

4. Block Diagram



5.Detailed Explanations of Modules

a. project_main

i. **Aim:** The aim of this module is to implement an interactive VGA-based drawing application using SystemVerilog. The design enables users to draw on a canvas using mouse or buttons and adjust brush size and color. The project utilizes various modules, such as VGA and mouse controllers, to handle input and display output on a 640x480 VGA screen.

ii. **Code Explanation**: This is the top module of the project. It integrates multiple submodules to create the application. It processes inputs from a mouse/button and switches to allow users to draw on a canvas. The module uses the to decode mouse signals, extracting the X and Y positions, as well as the states of the left and right buttons. These inputs control the drawing logic, which updates the canvas through the canvas_controller_ module. The canvas controller follows pixel color updates based on given input. It manages clearing the canvas when a specific switch is activated. A brush size controller with brushController input, adjusts the brush size with switch, while a set of switches (sw[7:0]) allows users to select colors from a palette from COLOUR_X's.

iii. **Code:**

```
module project_main(

    //button inputs

    /*input logic up_btn,  input logic down_btn, input logic
left_btn,input logic right_btn,input logic center_btn,*/



    //if you want to usae buttons instead mouse use this variables



    //mouse inputs

    input logic mouse_clk,input logic mouse_data,

    //Clock

    input logic clk_100MHz, input logic brushController, input logic
[8:0] sw,

    //9-bit vga color

output logic [2:0] vgaRed,output logic [2:0] vgaGreen,utput logic [1:0]
vgaBlue,

//blue is 2 bit

    output logic vgaHsync,output logic vgaVsync

);

    //brush logics

    logic [9:0] brushSize = 1;   logic [3:0] brushColor = 4'b0000;

    //mouse logics
```

```
    logic [9:0] mouseX, mouseY;logic mouseLeftButton;logic
mouseRightButton;ogic [9:0] cursor_logic_mouse_X,
cursor_logic_mouse_Y;

    logic [9:0] addPixelX, addPixelY; logic reset = 0;

    logic clk_25MHz;

    logic xyvalid;logic yvalid;



    logic [9:0] pixelX, pixelY;logic isClearing = sw[8];//it was created
for debugging purposes, it is n



    always_ff @(posedge clk_100MHz) begin

        if (xyvalid) begin

            cursor_logic_mouse_X <= mouseX;

            cursor_logic_mouse_Y <= mouseY;

        end

    end

    //mouse logic

    mouse_controller mouse_controller(

        .clk(clk_100MHz),

        .reset(reset),

        .mouse_clk(mouse_clk),

        .mouse_data(mouse_data),

        .mouseX(mouseX),

        .mouseY(mouseY),

        .mouseLeftButton(mouseLeftButton),

        .mouseRightButton(mouseRightButton)

    );

    //button logic
```

```
    /*buttonController buttonController(

            .clk(clk_100MHz),

            .reset(reset),

            .up_btn(up_btn),

            .down_btn(down_btn),

            .left_btn(left_btn),

            .right_btn(right_btn),

            .center_btn(center_btn),

            .mouseX(mouseX),

            .mouseY(mouseY),

            .pixel_will_painted(mouseLeftButton)

        );*/

    //if you want to control with buttons instead mous use this

    logic canvasActive;

    assign canvasActive = xyvalid;

    logic [7:0] canvasColor;



    // cursor logic

    logic active_pointer1 = cursor_logic_mouse_X <= pixelX && 
pixelX < cursor_logic_mouse_X + 1 &&

                            cursor_logic_mouse_Y <= pixelY && pixelY < 
cursor_logic_mouse_Y + 5; // Downward line

    logic active_pointer2 = cursor_logic_mouse_X <= pixelX && 
pixelX < cursor_logic_mouse_X + 5 &&

                            cursor_logic_mouse_Y <= pixelY && pixelY < 
cursor_logic_mouse_Y + 1;  // Rightward line

    logic active_pointer3 = cursor_logic_mouse_X - 5 <= pixelX && 
pixelX < cursor_logic_mouse_X &&

                            cursor_logic_mouse_Y <= pixelY && pixelY < 
cursor_logic_mouse_Y + 1;  // Leftward line
```

```
    logic active_pointer4 = cursor_logic_mouse_X <= pixelX &&
pixelX < cursor_logic_mouse_X + 1 &&

                         cursor_logic_mouse_Y - 5 <= pixelY && pixelY <
cursor_logic_mouse_Y; // Upward line


    logic [7:0] pointerColor;


    localparam COLOUR_0 = 8'b111_111_11;

    localparam COLOUR_1 = 8'b000_111_00;

    localparam COLOUR_2 = 8'b000_000_11;

    localparam COLOUR_3 = 8'b111_111_00;

    localparam COLOUR_4 = 8'b111_000_11;

    localparam COLOUR_5 = 8'b000_111_11;

    localparam COLOUR_6 = 8'b011_011_10;

    localparam COLOUR_7 = 8'b111_000_00;

    localparam COLOUR_8 = 8'b000_000_00;




    canvas_controller_ #(

        .color_palette_bit(3),

        .color_choices({COLOUR_8, COLOUR_7, COLOUR_6,
COLOUR_5, COLOUR_4, COLOUR_3, COLOUR_2, COLOUR_1,
COLOUR_0}),

        .width(640),

        .height(480)

    )

    canvas(

        .clk(clk_100MHz),

        .xyvalid(canvasActive),
```

```
        .x(pixelX),
        .y(pixelY),
        .writeX(isClearing ? pixelX : mouseX + addPixelX),
        .writeY(isClearing ? pixelY : mouseY + addPixelY),
        .write_enable(isClearing || (mouseLeftButton && (mouseX +
addPixelX < 640) && brushColor != 0)),
        .write_color(isClearing ? 0 : brushColor),
        .red(canvasColor[7:5]),
        .green(canvasColor[4:2]),
        .blue(canvasColor[1:0]));
    //brush size logic
    always_ff @(posedge clk_100MHz) begin
        case (brushController)
            1'b0: brushSize <= 1;
            1'b1: brushSize <= 3;
            default: brushSize <= 1;
        endcase

        case (sw[7:0])
            8'b00000001: brushColor <= 1;
            8'b00000010: brushColor <= 2;
            8'b00000100: brushColor <= 3;
            8'b00001000: brushColor <= 4;
            8'b00010000: brushColor <= 5;
            8'b00100000: brushColor <= 6;
            8'b01000000: brushColor <= 7;
            8'b10000000: brushColor <= 8;
```

```
        default: brushColor <= 0;
      endcase
    end

    assign pointerColor = 8'h000;

    //vga color logic to show the cursor
    assign vgaRed = (active_pointer1 || active_pointer2 || 
active_pointer3 || active_pointer4) ? pointerColor[7:5] : 
canvasColor[7:5];
    assign vgaGreen = (active_pointer1 || active_pointer2 || 
active_pointer3 || active_pointer4) ? pointerColor[4:2] : 
canvasColor[4:2];
    assign vgaBlue = (active_pointer1 || active_pointer2 || 
active_pointer3 || active_pointer4) ? pointerColor[1:0] : 
canvasColor[1:0];

    clock_divider #(.factor_div(4)) vgaPixelClkDiv(
      .clk(clk_100MHz),
      .reset(0),
      .clock_divided(clk_25MHz));

    vga_controller vga_controller(
      .clk(clk_100MHz),
      .clk_25MHz(clk_25MHz),
      .clr(0),
      .horizontal_sync(vgaHsync),
      .vertical_sync(vgaVsync),
      .xyvalid(xyvalid),
```

```
.yvalid(yvalid),
.x(pixelX),
.y(pixelY));

cursor_scanner #(.x_bits(10), .y_bits(10)) cursor_scanner (
.clk(clk_100MHz),
.enable(clk_25MHz),
.x(addPixelX),
.y(addPixelY),
.width(brushSize),
.height(brushSize) );

endmodule
```

b. mouse_controller

i. **Aim**: This module is created for controlling mouse with PS/2 mouse.
ii. **Code Explanation**: Module decodes given input signals to make coordinates. Mouse left button states followed. It moves in predefined 2D space and boundaries are followed too. The right button signal is not utilized in this implementation but is included for standardization purposes.
iii. **Code**:

```
module mouse_controller #(
parameter MAX_X = 639, parameter MAX_Y = 479) (
    input logic mouse_clk,input logic mouse_data,input logic clk, input logic reset,
output logic [9:0] mouseX,output logic [9:0] mouseY,output logic mouseLeftButton,output logic mouseRightButton);
    //right button is not used it is here just for standartise purpose
    `define clamped_plus(coord, minus, mag, max) \
```

```
    minus ? ((coord < mag) ? 0 : (coord - mag)) \
        : ((coord + mag > max) ? max : (coord + mag))

logic [43:0] pckt;
logic [1:0] mouse_clk_hist;
logic [5:0] bits_of_packet;

logic signed [8:0] xDelta, yDelta;
logic [9:0] xDeltaMag, yDeltaMag;
logic xDeltaSign, yDeltaSign;
logic xOverflow, yOverflow;
//These flags indicate if the movement exceeds the range
//that can be represented by the 8-bit signed delta values (xDelta and
yDelta).

//packet assigns
assign xDelta = { pckt[38], pckt[24], pckt[25], pckt[26], pckt[27],
pckt[28], pckt[29], pckt[30], pckt[31] };
assign yDelta = { pckt[37], pckt[13], pckt[14], pckt[15], pckt[16],
pckt[17], pckt[18], pckt[19], pckt[20] };
assign { xOverflow, yOverflow } = pckt[36:35];
//comparison logic
assign xDeltaSign = xDelta < 0;
assign yDeltaSign = yDelta > 0;
assign xDeltaMag = xDelta < 0 ? (-xDelta) : xDelta;
assign yDeltaMag = yDelta < 0 ? (-yDelta) : yDelta;

initial begin
```

```
        mouse_clk_hist <= 2'b11;
    end
    //the logic starts
    always_ff @(posedge clk or posedge reset)
    begin
        if (reset) //if reset is on
        begin

            bits_of_packet <= 0;
            pckt <= 0;
            mouse_clk_hist <= 2'b11;
            mouseX <= MAX_X / 2;
            mouseY <= MAX_Y / 2;
            mouseLeftButton <= 1'b0;
            mouseRightButton <= 1'b0;//unused but is is here to be
standadize
        end
        else
        begin
            mouse_clk_hist <= { mouse_clk_hist[0], mouse_clk };
            if (mouse_clk_hist == 2'b10) begin

                pckt <= { pckt[42:0], mouse_data };
                bits_of_packet <= bits_of_packet + 1;

            end
            else if (bits_of_packet == 44)
```

```
begin

    mouseLeftButton <= pckt[42];

    mouseRightButton <= pckt[41];

    // clamped_plus ensures coordinate updates respect screen
boundaries.

    mouseX <= `clamped_plus(mouseX, xDeltaSign,
xDeltaMag, MAX_X);

    mouseY <= `clamped_plus(mouseY, yDeltaSign,
yDeltaMag, MAX_Y);


    pckt <= 0;

    bits_of_packet <= 0;

begin
end

end

endmodule
```

c. canvas_controller

i. **Aim:** The aim of this module is to manage the drawing and color rendering of 2D canvas in our application.

ii. **Code Explanation**: The module integrates memory with stored color. Data for each pixel on the canvas is generated with RGB signals. It enables interactive updating of canvas with different colors for users.

iii. **Code**:

```
module canvas_controller_#

(                       parameter int color_palette_bit = 2,

                        parameter logic [8*(1 << color_palette_bit)-1:0]
color_choices = -1,

                        parameter int width = 100,

                        parameter int height = 100
```

```
                        ) (

   //input logics input logic clk, input logic xyvalid,input logic [9:0]
x,input logic [9:0] y,input logic [9:0] writeX,input logic [9:0] writeY,
input logic write_enable, //enable

input logic [color_palette_bit-1:0] write_color,

   //output logics

   output logic [2:0] red, output logic [2:0] green,output logic [1:0]
blue

   );


   logic [color_palette_bit-1:0] color_palletes;


   RAM_controller #(

      .width_of_ram(color_palette_bit),

      .depth_of_ram(width * height)

   )

   ram (

      .write_addr(writeX + writeY * width),

      .read_addr(x + y * width),

      .word_in(write_color),

      .clk(clk),

      .write_en(write_enable),

      .read_en(xyvalid),

      .output_rst(0),

      .output_en(1),

      .word_out(color_palletes));

   //assigning the color bits from volor luts respect to xyvalid

   assign red   = xyvalid ? color_choices[8 * color_palletes + 7 -: 3] : 0;
```

```
    assign green = xyvalid ? color_choices[8 * color_palletes + 4 -: 3] :
0;

    assign blue  = xyvalid ? color_choices[8 * color_palletes + 1 -: 2] :
0;


endmodule
```

d. RAM_controller

i. **Aim:** This module is designed to implement efficient memory(RAM). Module is designed with configurable word width and depth.
ii. **Code Explanation**:It is similar to classical BRAM modules. It supports reading and writing operations with separate control signals, making it suitable for use in a variety of digital systems, including our VGA-based application.
iii. **Code**:

```
module RAM_controller#(

parameter int width_of_ram = 1, // Width of each memory word

parameter int depth_of_ram = 10 // Depth of the memory (number of
words)

) (

    input logic clk, input logic write_en,input logic read_en,input logic
output_rst, input logic output_en,

    input logic [clogb2(depth_of_ram-1)-1:0] write_addr,  // Address for
writing data

    input logic [clogb2(depth_of_ram-1)-1:0] read_addr,  // Address for
reading data

    input logic [width_of_ram-1:0] word_in,   // Data to be written to
memory

    output logic [width_of_ram-1:0] word_out  // Data output

    );
```

```
// Memory array to store words

logic [width_of_ram-1:0] memory_array [0:depth_of_ram-1];


// Register to hold data temporarily for output logic

[width_of_ram-1:0] temp_data = {width_of_ram{1'b0}};

logic [width_of_ram-1:0] output_register = {width_of_ram{1'b0}};

// Memory initialization

initial begin

 foreach (memory_array[index]) begin

            memory_array[index] = {width_of_ram{1'b0}};

// Set all memory locations to zero end end

// Write and read operations

always_ff @(posedge clk)

            begin if (write_en)

                        begin

                                    memory_array[write_addr] <= word_in;

                                    // Writing data into memory

                        end

            if (read_en) begin

            temp_data <= memory_array[read_addr];

             // Reading data from memory end end

             // Control logic for output register

                        always_ff @(posedge clk)

                                    begin if (output_rst) begin

                                    output_register <= {width_of_ram{1'b0}};
```

```
// Reset output register
end
else if (output_en) begin
    output_register <= temp_data;
    // Load read data into the output register
end end
    // Assign output register to module's
output
    assign word_out = output_register;
    // Function to calculate address width
function automatic integer
    clogb2;
    input integer depth;
    integer count; begin clogb2 = 0;
    for (count = depth; count > 0; count =
    count >> 1)
    begin clogb2 = clogb2 + 1;
end
end
endfunction
endmodule
```

e. clock_divider

i. **Code**:

```
module clock_divider#(
  parameter factor_div = 1
```

```
) (
    input logic clk,
    input logic reset,
    output logic clock_divided);
     // Counter with the required number of bits to represent factor_div
    logic [$clog2(factor_div)-1:0] counter;

     // Assign divided clock signal
    assign clock_divided = (counter == (factor_div - 1));

    //logic starts
     always_ff @(posedge clk or posedge reset)
     begin
          if (reset) // if the reset 1
          begin
              counter <= 0;
          end else if (counter == (factor_div - 1)) //counter check
          begin
              counter <= 0;
          end else
          begin
              counter <= counter + 1;
          end
       end
   endmodule
```

f. vga_controller

i. **Aim**: The aim of this module is implementing a VGA controller which is the core module for our project.

ii. **Code Explanation**: VGA controller is used for generating synchronized horizontal and vertical signals necessary for VGA display operation. It provides pixel coordinates (x and y) and validates the screen area to display content, forming the backbone of any VGA-based graphics system.

iii. **Code**:

```
module vga_controller(

    input logic clk, input logic clk_25MHz, input logic clr,

    output logic xyvalid, output logic yvalid, output logic
horizontal_sync,output logic vertical_sync,output logic [9:0] x,output
logic [9:0] y  );

    //parameters written from the project file

    parameter int horizontal_back_porch = 144;

    parameter int horizontal_front_porch = 784;

    parameter int vertical_back_porch = 0;

    parameter int vertical_front_porch = 480;

    parameter int horizantal_pixels = 800;

    parameter int vertical_lines = 480;

    parameter int horizantal_pulse = 96;

    parameter int vertical_pulse = 2;



    logic [9:0] horizontal_cync;

    logic [9:0] vertical_cync;

    //logic starts

   always_ff @(posedge clk or posedge clr) begin

          if (clr) //clr currently is not used it was here for debugging
purposes
```

```
            begin
                horizontal_cync <= 0;
                vertical_cync <= 0;
            end else if (clk_25MHz)
            begin
                if (horizontal_cync < horizantal_pixels - 1) //checking for
boundries
                begin
                    horizontal_cync <= horizontal_cync + 1;
                end else
                begin
                    horizontal_cync <= 0;
                    if (vertical_cync < vertical_lines - 1) //checking for
boundries
                    begin
                        vertical_cync <= vertical_cync + 1;
                    end else
                    begin
                        vertical_cync <= 0;
                    end
                end
            end
        end
    //assignments starts here to proporley assigning the screen
boundirais
    assign horizontal_sync = (horizontal_cync < horizantal_pulse) ? 1'b0
: 1'b1;
```

```
  assign vertical_sync = (vertical_cync < vertical_pulse) ? 1'b0 : 1'b1;

  assign xyvalid = (vertical_back_porch <= vertical_cync &&
vertical_cync < vertical_front_porch && horizontal_back_porch <=
horizontal_cync && horizontal_cync < horizontal_front_porch);

  assign yvalid  = (vertical_cync < vertical_front_porch);

  /*assign x      = horizontal_cync;

  assign y      = vertical_cync;*/

  assign x      = horizontal_cync - horizontal_back_porch;

  assign y      = vertical_cync - vertical_back_porch;


endmodule
```

g. cursor_scanner

i. **Aim**: The aim of this module is to simulate a scanning cursor that traverses the grid which is our screen.

ii. **Code Explanation**:This grid is defined with width and height parameters. This module is typically used to control or interact with graphical displays, grids, or memory areas in a systematic manner.

iii. **Code**:

```
module cursor_scanner #(parameter x_bits = 3, parameter y_bits = 3)

(

  input logic [x_bits+1-1:0] width,input logic  [y_bits+1-1:0] height,
input logic clk, //clock

  input logic enable, //enable for cursor always able

  output logic [x_bits-1:0] x, output logic [y_bits-1:0] y);

  /*  always_ff @(posedge clk)

  begin //it was mistake part, does not work properly

     if (enable) begin

         if (x < width)

         begin

               x <= x + 1;
```

```
        end
        else
        begin
            x <= 0;
            if (y < height)
            begin
                y <= y + 1;
            end else
            begin
                y <= 0;
            end
        end
    end
end*/
//logic starts here
always_ff @(posedge clk)
begin
    if (enable) begin
         if (x < width - 1)
         begin
                x <= x + 1;
        end
        else
        begin
            x <= 0;
            if (y < height - 1)
            begin
```

```
                y <= y + 1;
            end else
            begin
                y <= 0;
            end
        end
    end
end


endmodule
```

h. buttonController

i. **Explanation:** This module for 2nd stage. Instead of controlling the mouse, we can draw with buttons. If proper lines in the project_main are got out of comment line it will properly work

ii. **Code:**

```
module buttonController #(
    parameter int boundiries_X = 639,
    parameter int boundiries_Y = 479
)(
    input  logic clk, input  logic reset,
    //button inputs
    input  logic right_btn,input  logic left_btn, input  logic up_btn,input  logic down_btn,input
logic center_btn,
    //mouse inputs
    output logic [9:0] mouseX,output logic [9:0] mouseY,output logic pixel_will_painted
);


    logic [19:0] counter;
    //slwowing down with counter for movements
```

```
    logic enable;


    initial begin
        mouseX = boundiries_X / 2;
        mouseY = boundiries_Y / 2;
        pixel_will_painted = 0;
        counter = 0;
    end
    //initially cursor will born in the middle of the screen
/*initial begin //for inittializing cursor in corner
        mouseX = boundiries_X;
        mouseY = boundiries_Y;
        pixel_will_painted = 0;
        counter = 0;
    end*/
     //logic beginds
    always_ff @(posedge clk or posedge reset) begin
        if (reset) begin
            counter <= 0;
            enable <= 0;
        end else begin
            counter <= counter + 1;
            if (counter == 0)
            //eneable is triggerde due to counter made over floe
                enable <= 1;
            else //else case anable is 0
                enable <= 0;
```

```
        end
    end


    //cursor movement logic begins here
    always_ff @(posedge clk or posedge reset) begin
    //reset both resets mouuse movement and pixel drawing
        if (reset) begin
            mouseX <= boundiries_X / 2;
            mouseY <= boundiries_Y / 2;
            pixel_will_painted <= 0;
        end else begin
        //movement logic starts
if (enable) begin
    if (right_btn) begin // Logic for right button
        if (mouseX < boundiries_X)
            mouseX <= mouseX + 1;
        else
            mouseX <= boundiries_X;
    end else if (left_btn) begin // Logic for left button
        if (mouseX > 0)
            mouseX <= mouseX - 1;
        else
            mouseX <= 0;
    end else if (up_btn) begin // Logic for up button
        if (mouseY > 0)
            mouseY <= mouseY - 1;
        else
```

```
            mouseY <= 0;
    end else if (down_btn) begin // Logic for down button
        if (mouseY < boundiries_Y)
            mouseY <= mouseY + 1;
        else
            mouseY <= boundiries_Y;
    end
end



            if (center_btn)//center button logic is not bothered from others
                pixel_will_painted <= 1;
            else
                pixel_will_painted <= 0;
        end
    end

endmodule
```