Kerem Emre
Bayrak
22303713
Section: 2

The objective of this project was to use the Basys 3 FPGA board to create a VGA driver system that could draw visuals on a monitor in different features. System Verilog was used to implement the design, which was broken down into multiple phases. A VGA controller, two clock dividers (25MHz and 100MHz), debouncing logic for buttons, memory modules for pixel data storage, and interfaces for human interaction using buttons and a PS/2 mouse were the design's main elements. The implementation included the VGA timing specifications, which call for a resolution of 640x480 pixels at a refresh rate of 60 Hz as specified in the requirements.

The VGA controller module, which controls the horizontal and vertical synchronization signals (hsync and vsync) required to drive a VGA display, was the first implementation module. By synchronizing the timing of drawing each row and each frame, the synchronization signals enabled the display to render each frame accurately. By keeping track of counters for the horizontal and vertical positions (pixel_x and pixel_y), the VGA controller produces these signals. The signals are toggled when the counters reach predetermined levels that correspond to the end of the visible area, front porch, sync pulse, and back porch. This timing ensures that the display adheres to the standard 640x480 resolution, with each frame consisting of 525 lines (480 visible lines plus 45 lines for front and back porches and the sync pulse) and each line consisting of 800 clock cycles (640 visible pixels plus 160 clock cycles for horizontal blanking intervals).

The pixel_clock_25MHz module created a 25 MHz pixel clock to accommodate the high-speed operation needed for VGA. This module uses a counter to reduce the system clock, which is normally 100 MHz, to 25 MHz. Since every pixel on the screen must be updated at this frequency in order to reach the intended frame rate of 60 Hz, the 25 MHz clock is used for powering the VGA display. A 100 Hz clock divider was also used to control slower processes like updating cursor positions and debouncing button inputs. To do this, the clock_divider_100Hz module counts clock cycles and toggles the output clock at the right times.

With the debounced buttons only valid button presses were recorded by implementing the debouncing logic, which was implemented in the debounce module and filtered out mechanical noise from the buttons. This managed the mouse movement and scrolling features, as multiple unintentional triggers could cause the unintended outputs in the frame. The VGA controller, scrolling logic, and debouncing modules were all combined into the top-level module vga_stage1_top. By modifying the scroll_x and scroll_y values in response to button inputs, the scrolling feature was accomplished. To give the impression that the image was moving across the screen, these offsets were applied to the pixel coordinates.

In drawing_canvas_top module, the canvas had a white background at first, and the user could use a brush of any size, from a single pixel to a 3x3 block. Pixel data was saved in the bram_dual_port block RAM (BRAM) module, which was used to record the pre synthesized data. The BRAM's dual-port design made it possible to read and write data simultaneously, which made it possible to update the display effectively as the user kept drawing.

For the 3x3 brush size mode for the drawing canvas, desired output couldn't be got. Inaccurate address calculations or logical mistakes in creating the 3x3 block of pixels could be the cause of the brush size problem, especially when addressing edge cases close to canvas borders. A PS/2 mouse interface in the drawing_canvas_top2 module is tried to be implemented. PS/2 mouse's signals were decoded by the ps2_mouse_controller module, which then extracted the movement of the cursor (delta_x, delta_y) and detected left-click events. Compared to button-based movement, this is easier for the user to operate the pointer. The drawing canvas's utility is enhanced by the mouse interface, which was focusing producing drawings simpler. Instead of using the center button (btnC) as in the earlier stages, the left-click button was employed as the trigger for drawing on the canvas.

The testbench waveform (figure 1.) demonstrates the problems with the ps2_mouse_controller module, which reveal that even when Mouse_Clk and Mouse_Data change, the delta_x output stays at 00000000. This implies that the mouse data may not be being properly captured or processed by the module. Possible reasons can be a malfunctioning state machine that doesn't detect when a byte is finished, wrong shift register implementation for collecting data bits, or inaccurate Mouse_Clk.

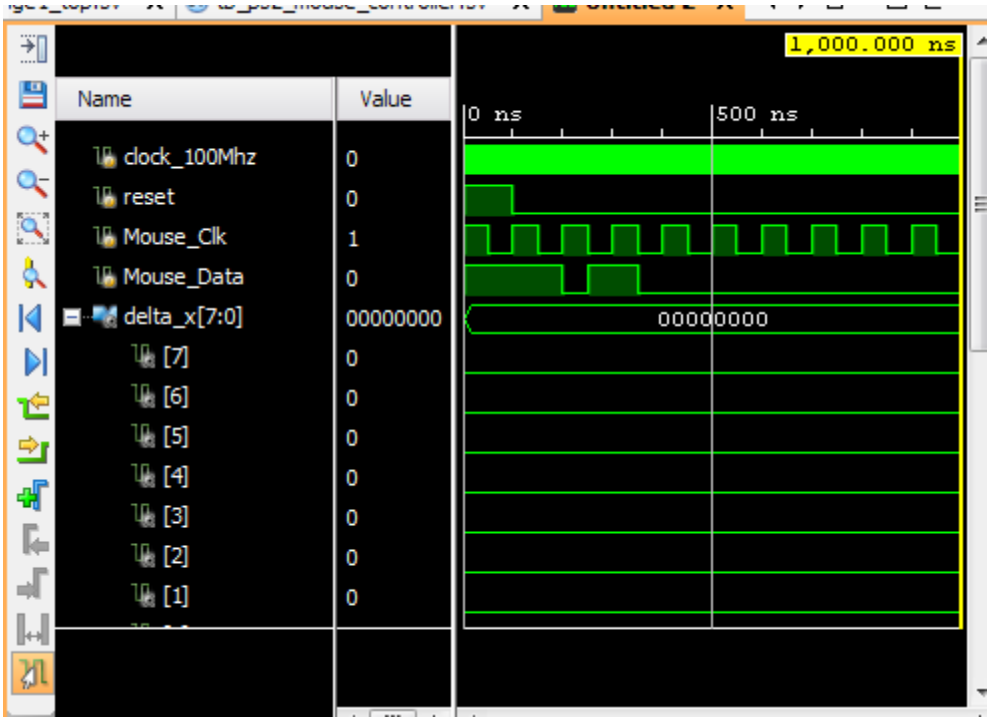


Figure 1. Simulation output for the mouse controller module

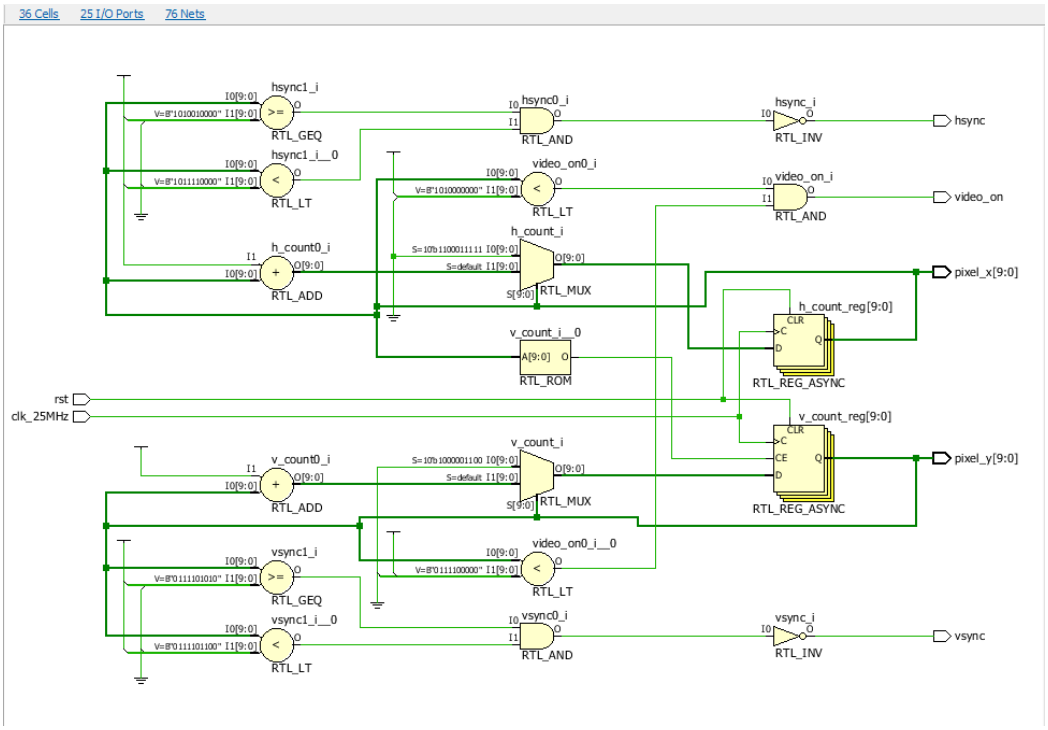


Figure 2. VGA controller RTL Schematics.

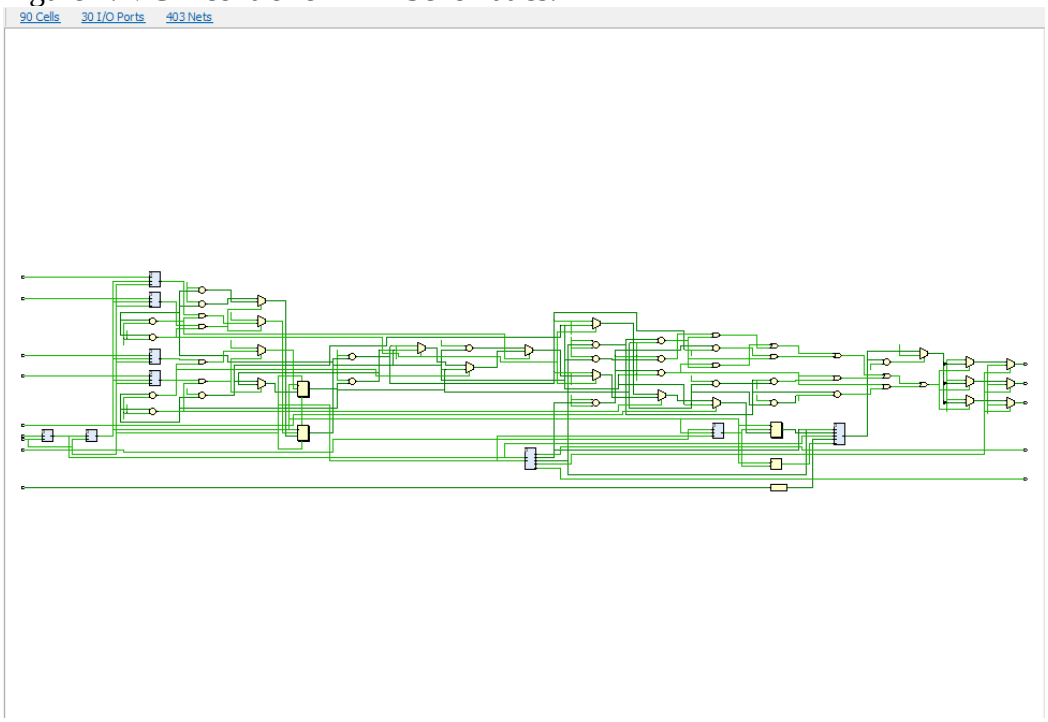


Figure 3. RTL Schematics for drawing logic and cursor control implemented in Drawing canvas top module.

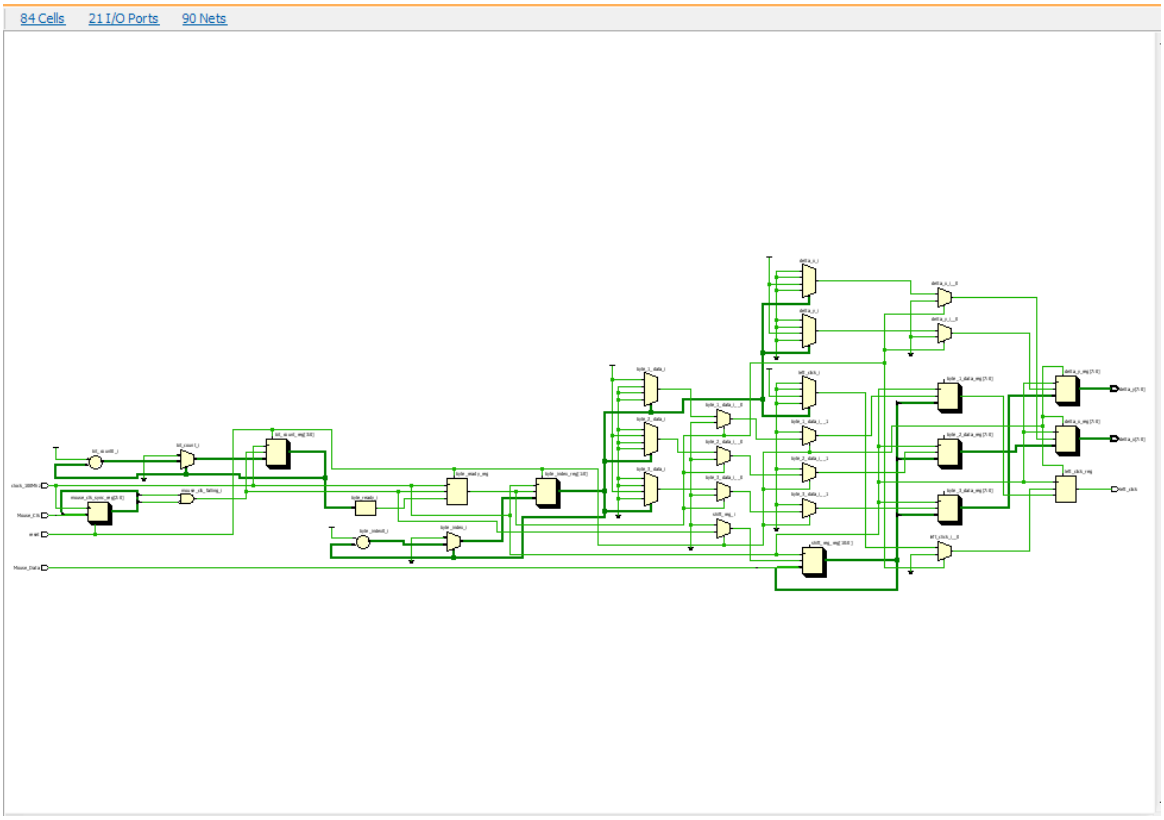
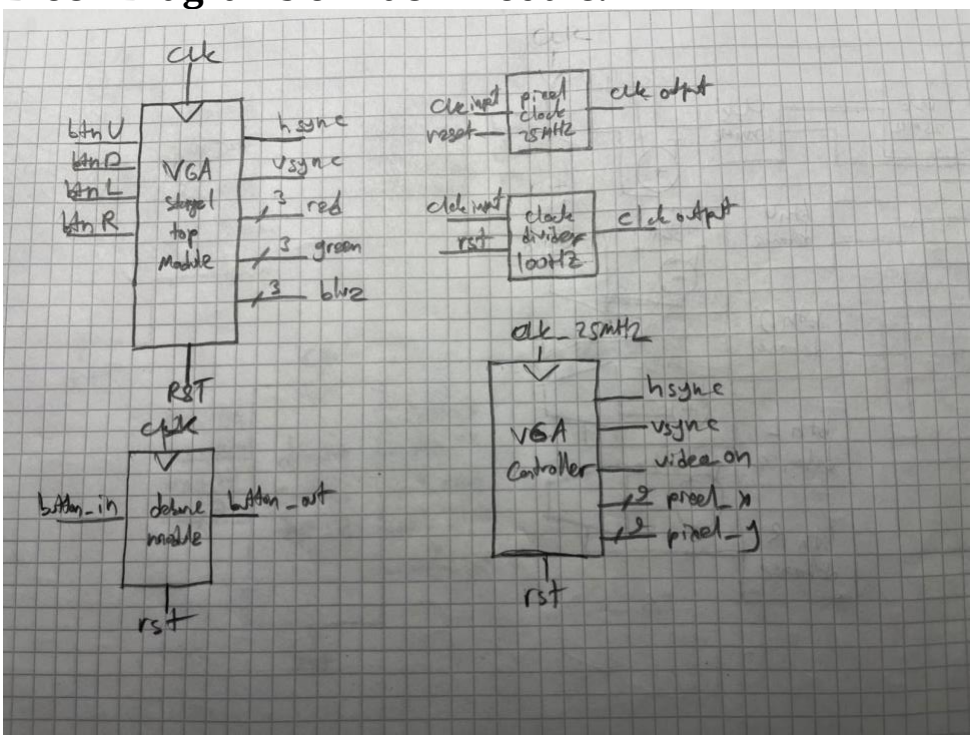
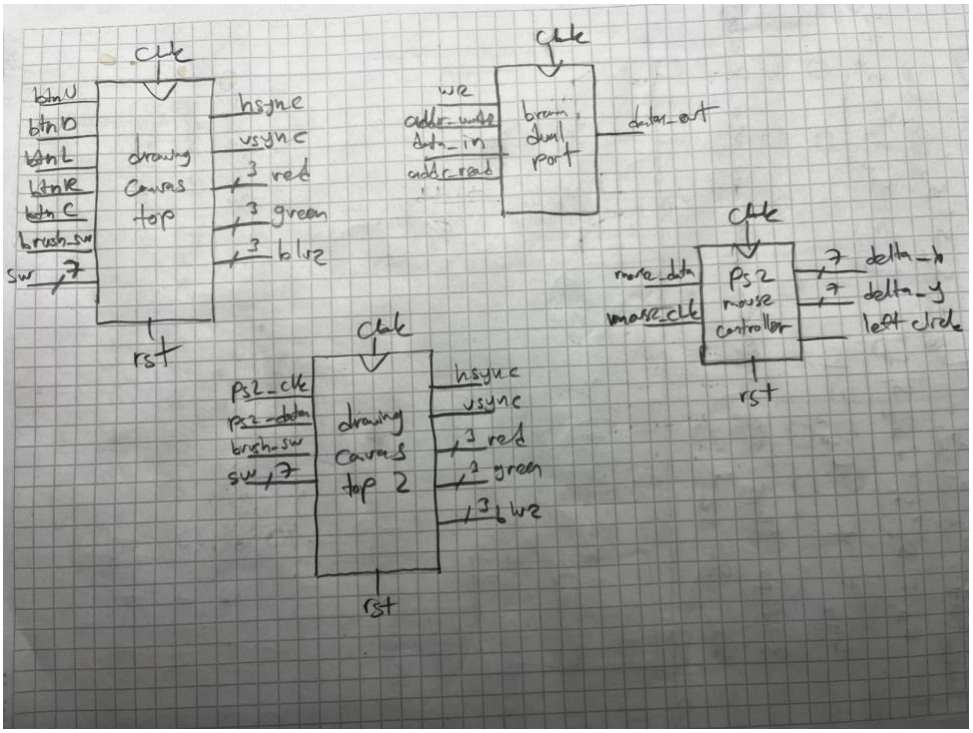


Figure 4. PS/ 2 mouse controller RTL schematics.

Block Diagrams of Each Module:





Appendix:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer:
//
// Create Date: 12/12/2024 10:57:10 PM
// Design Name:
// Module Name: vga_stage1_top
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////


module vga_stage1_top (
    input logic clk,
    input logic rst,
    input logic btnU,
    input logic btnD,
    input logic btnL,
    input logic btnR,
    output logic hsync,
    output logic vsync,
    output logic [3:0] red,
    output logic [3:0] green,
    output logic [3:0] blue
);

    logic clk_25MHz;
    logic clk_100Hz;
    logic video_on;
    logic [9:0] pixel_x;
    logic [9:0] pixel_y;

    logic [9:0] scroll_x = 0;
    logic [9:0] scroll_y = 0;

    logic btnU_debounced;
    logic btnD_debounced;
    logic btnL_debounced;
    logic btnR_debounced;

    pixel_clock_25MHz clk_div_inst (
        .clk_input(clk),
```

```
    .rst(rst),
    .clk_output(clk_25MHz)
);

clock_divider_100Hz clk_div_100Hz_inst (
    .clk_input(clk_25MHz),
    .rst(rst),
    .clk_output(clk_100Hz)
);

debounce btnU_debounce_inst (
    .clk(clk_100Hz),
    .rst(rst),
    .button_in(btnU),
    .button_out(btnU_debounced)
);

debounce btnD_debounce_inst (
    .clk(clk_100Hz),
    .rst(rst),
    .button_in(btnD),
    .button_out(btnD_debounced)
);

debounce btnL_debounce_inst (
    .clk(clk_100Hz),
    .rst(rst),
    .button_in(btnL),
    .button_out(btnL_debounced)
);

debounce btnR_debounce_inst (
    .clk(clk_100Hz),
    .rst(rst),
    .button_in(btnR),
    .button_out(btnR_debounced)
);

vga_controller vga_inst (
    .clk_25MHz(clk_25MHz),
    .rst(rst),
    .hsync(hsync),
    .vsync(vsync),
    .video_on(video_on),
    .pixel_x(pixel_x),
    .pixel_y(pixel_y)
);

always_ff @(posedge clk_100Hz or posedge rst) begin
    if (rst) begin
        scroll_x <= 0;
        scroll_y <= 0;
    end else begin
        if (btnU_debounced) scroll_y <= scroll_y + 1;
        if (btnD_debounced) scroll_y <= scroll_y - 1;
        if (btnL_debounced) scroll_x <= scroll_x + 1;
        if (btnR_debounced) scroll_x <= scroll_x - 1;
```

```
        end
    end

    logic checkerboard;
    always_comb begin
        checkerboard = ((pixel_x + scroll_x) >> 5) ^ ((pixel_y + scroll_y) >>
5);
    end

    always_comb begin
        if (video_on) begin
            if (checkerboard) begin
                red   = 4'hF;
                green = 4'hF;
                blue  = 4'hF;
            end else begin
                red   = 4'h0;
                green = 4'h0;
                blue  = 4'h0;
            end
        end else begin
            red   = 4'h0;
            green = 4'h0;
            blue  = 4'h0;
        end
    end

endmodule


`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer:
//
// Create Date: 12/12/2024 09:28:19 PM
// Design Name:
// Module Name: pixel_clock_25MHz
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////


module pixel_clock_25MHz (
    input logic clk_input,
    input logic rst,
```

```
    output logic clk_output
);

    logic [1:0] counter = 2'b00;

    always_ff @(posedge clk_input or posedge rst) begin
        if (rst) begin
            counter <= 2'b00;
            clk_output <= 1'b0;
        end else begin
            counter <= counter + 1;
            if (counter == 2'b01) begin
                clk_output <= ~clk_output;
                counter <= 2'b00;
            end
        end
    end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer:
//
// Create Date: 12/12/2024 11:09:46 PM
// Design Name:
// Module Name: clock_divider_100Hz
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////


module clock_divider_100Hz (
    input logic clk_input,
    input logic rst,
    output logic clk_output
);

    logic [19:0] counter = 0;

    always_ff @(posedge clk_input or posedge rst) begin
        if (rst) begin
            counter <= 0;
            clk_output <= 0;
        end else begin
```

```
            if (counter == 499999) begin
                clk_output <= ~clk_output;
                counter <= 0;
            end else begin
                counter <= counter + 1;
            end
        end
    end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer:
//
// Create Date: 12/12/2024 11:08:59 PM
// Design Name:
// Module Name: debounce
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////


module debounce (
    input logic clk,
    input logic rst,
    input logic button_in,
    output logic button_out
);

    logic [2:0] shift_reg = 3'b000;

    always_ff @(posedge clk or posedge rst) begin
        if (rst) begin
            shift_reg <= 3'b000;
        end else begin
            shift_reg <= {shift_reg[1:0], button_in};
        end
    end

    assign button_out = &shift_reg;

endmodule

`timescale 1ns / 1ps
```

```
//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer:
//
// Create Date: 12/12/2024 09:27:15 PM
// Design Name:
// Module Name: vga_controller
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////


module vga_controller (
    input logic clk_25MHz,
    input logic rst,
    output logic hsync,
    output logic vsync,
    output logic video_on,
    output logic [9:0] pixel_x,
    output logic [9:0] pixel_y
);


    localparam H_VISIBLE_AREA = 640;
    localparam H_FRONT_PORCH  = 16;
    localparam H_SYNC_PULSE   = 96;
    localparam H_BACK_PORCH   = 48;
    localparam H_TOTAL        = 800;


    localparam V_VISIBLE_AREA = 480;
    localparam V_FRONT_PORCH  = 10;
    localparam V_SYNC_PULSE   = 2;
    localparam V_BACK_PORCH   = 33;
    localparam V_TOTAL        = 525;

    logic [9:0] h_count = 0;
    logic [9:0] v_count = 0;

    always_ff @(posedge clk_25MHz or posedge rst) begin
        if (rst) begin
            h_count <= 0;
            v_count <= 0;
        end else begin
            if (h_count == H_TOTAL - 1) begin
                h_count <= 0;
```

```
                if (v_count == V_TOTAL - 1) begin
                    v_count <= 0;
                end else begin
                    v_count <= v_count + 1;
                end
            end else begin
                h_count <= h_count + 1;
            end
        end
    end

    assign hsync = ~(h_count >= H_VISIBLE_AREA + H_FRONT_PORCH &&
                     h_count < H_VISIBLE_AREA + H_FRONT_PORCH +
H_SYNC_PULSE);

    assign vsync = ~(v_count >= V_VISIBLE_AREA + V_FRONT_PORCH &&
                     v_count < V_VISIBLE_AREA + V_FRONT_PORCH +
V_SYNC_PULSE);

    assign video_on = (h_count < H_VISIBLE_AREA) && (v_count <
V_VISIBLE_AREA);

    assign pixel_x = h_count;
    assign pixel_y = v_count;

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer:
//
// Create Date: 12/13/2024 08:55:59 PM
// Design Name:
// Module Name: drawing_canvas_top
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////


module drawing_canvas_top (
    input logic clk,
    input logic rst,
    input logic btnU,
    input logic btnD,
    input logic btnL,
```

```
    input logic btnR,
    input logic btnC,
    input logic brush_sw,
    input logic [7:0] sw,
    output logic hsync,
    output logic vsync,
    output logic [3:0] red,
    output logic [3:0] green,
    output logic [3:0] blue
);

    logic clk_25MHz;
    logic clk_100Hz;
    logic video_on;
    logic [9:0] pixel_x;
    logic [9:0] pixel_y;

    logic [6:0] cursor_x = 40;
    logic [5:0] cursor_y = 30;

    logic [12:0] write_address;
    logic [12:0] read_address;
    logic [11:0] write_data;
    logic [11:0] read_data;
    logic write_enable;

    logic [3:0] brush_state;

    logic btnU_debounced, btnD_debounced, btnL_debounced, btnR_debounced,
btnC_debounced;

    pixel_clock_25MHz clk_div_inst (
        .clk_input(clk),
        .rst(rst),
        .clk_output(clk_25MHz)
    );

    clock_divider_100Hz cursor_clk_div (
        .clk_input(clk_25MHz),
        .rst(rst),
        .clk_output(clk_100Hz)
    );

    vga_controller vga_inst (
        .clk_25MHz(clk_25MHz),
        .rst(rst),
        .hsync(hsync),
        .vsync(vsync),
        .video_on(video_on),
        .pixel_x(pixel_x),
        .pixel_y(pixel_y)
    );

    debounce debounceU (.clk(clk_100Hz), .rst(rst), .button_in(btnU),
.button_out(btnU_debounced));
    debounce debounceD (.clk(clk_100Hz), .rst(rst), .button_in(btnD),
.button_out(btnD_debounced));
```

```
    debounce debounceL (.clk(clk_100Hz), .rst(rst), .button_in(btnL),
.button_out(btnL_debounced));
    debounce debounceR (.clk(clk_100Hz), .rst(rst), .button_in(btnR),
.button_out(btnR_debounced));
    debounce debounceC (.clk(clk_100Hz), .rst(rst), .button_in(btnC),
.button_out(btnC_debounced));

    bram_dual_port #(
        .DATA_WIDTH(12),
        .ADDR_WIDTH(13)
    ) bram_inst (
        .clk(clk_25MHz),
        .we(write_enable),
        .addr_write(write_address),
        .data_in(write_data),
        .addr_read(read_address),
        .data_out(read_data)
    );

    always_ff @(posedge clk_100Hz or posedge rst) begin
        if (rst) begin
            cursor_x <= 40;
            cursor_y <= 30;
        end else begin
            if (btnU_debounced && cursor_y > 0) cursor_y <= cursor_y - 1;
            if (btnD_debounced && cursor_y < 59) cursor_y <= cursor_y + 1;
            if (btnL_debounced && cursor_x > 0) cursor_x <= cursor_x - 1;
            if (btnR_debounced && cursor_x < 79) cursor_x <= cursor_x + 1;
        end
    end

    always_comb begin
        case (sw)
            8'b0000_0001: write_data = 12'hF00; // Red
            8'b0000_0010: write_data = 12'h0F0; // Green
            8'b0000_0100: write_data = 12'h00F; // Blue
            8'b0000_1000: write_data = 12'hFF0; // Yellow
            8'b0001_0000: write_data = 12'h0FF; // Cyan
            8'b0010_0000: write_data = 12'hF0F; // Magenta
            8'b0100_0000: write_data = 12'hF80; // Orange
            8'b1000_0000: write_data = 12'hFFF; // White
            default:      write_data = 12'hFFF; // Default
        endcase
    end

always_ff @(posedge clk_100Hz) begin
    write_enable <= 1'b0;

    if (btnC_debounced) begin
        write_address <= {cursor_y, cursor_x};
        write_enable <= 1'b1;

        if (brush_sw) begin
            if (cursor_y > 0) begin
                if (cursor_x > 0) begin
                    write_address <= {cursor_y - 1, cursor_x - 1};
                    write_enable <= 1'b1;
```

```
                end
                write_address <= {cursor_y - 1, cursor_x};
                write_enable <= 1'b1;
                if (cursor_x < 79) begin
                    write_address <= {cursor_y - 1, cursor_x + 1};
                    write_enable <= 1'b1;
                end
            end

            if (cursor_x > 0) begin
                write_address <= {cursor_y, cursor_x - 1};
                write_enable <= 1'b1;
            end
            if (cursor_x < 79) begin
                write_address <= {cursor_y, cursor_x + 1};
                write_enable <= 1'b1;
            end

            if (cursor_y < 59) begin
                if (cursor_x > 0) begin
                    write_address <= {cursor_y + 1, cursor_x - 1};
                    write_enable <= 1'b1;
                end
                write_address <= {cursor_y + 1, cursor_x};
                write_enable <= 1'b1;
                if (cursor_x < 79) begin
                    write_address <= {cursor_y + 1, cursor_x + 1};
                    write_enable <= 1'b1;
                end
            end
        end
    end
end

    assign read_address = {pixel_y[9:3], pixel_x[9:3]};

    always_comb begin
        if (video_on) begin
            {red, green, blue} = (read_data != 12'h000) ? read_data :
12'hFFF;

            if ((pixel_x >> 3) == cursor_x && (pixel_y >> 3) == cursor_y ||
                (pixel_x >> 3) == cursor_x - 1 && (pixel_y >> 3) == cursor_y
||
                (pixel_x >> 3) == cursor_x + 1 && (pixel_y >> 3) == cursor_y
||
                (pixel_x >> 3) == cursor_x && (pixel_y >> 3) == cursor_y - 1
||
                (pixel_x >> 3) == cursor_x && (pixel_y >> 3) == cursor_y + 1)
begin
                {red, green, blue} = 12'h000;
            end
        end else begin
            {red, green, blue} = 12'h000;
        end
    end
```

```
endmodule


`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer:
//
// Create Date: 12/14/2024 09:58:34 PM
// Design Name:
// Module Name: bram_dual_port #
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////


module bram_dual_port #(
    parameter DATA_WIDTH = 12,
    parameter ADDR_WIDTH = 13
) (
    input logic clk,
    input logic we,
    input logic [ADDR_WIDTH-1:0] addr_write,
    input logic [DATA_WIDTH-1:0] data_in,
    input logic [ADDR_WIDTH-1:0] addr_read,
    output logic [DATA_WIDTH-1:0] data_out
);


    (* ram_style = "block" *) logic [DATA_WIDTH-1:0] memory [0:(1 <<
ADDR_WIDTH) - 1];

    always_ff @(posedge clk) begin
        if (we) begin
            memory[addr_write] <= data_in;
        end
    end

    always_ff @(posedge clk) begin
        data_out <= memory[addr_read];
    end

endmodule

`timescale 1ns / 1ps
```

```
//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer:
//
// Create Date: 12/15/2024 01:23:33 AM
// Design Name:
// Module Name: drawing_canvas_top2
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////

module drawing_canvas_top2 (
    input logic clk,
    input logic rst,
    input logic ps2_clk,
    input logic ps2_data,
    input logic brush_sw,
    input logic [7:0] sw,
    output logic hsync,
    output logic vsync,
    output logic [3:0] red,
    output logic [3:0] green,
    output logic [3:0] blue
);


    logic clk_25MHz;
    logic clk_100Hz;
    logic video_on;
    logic [9:0] pixel_x;
    logic [9:0] pixel_y;


    logic [6:0] cursor_x = 40;
    logic [5:0] cursor_y = 30;

    logic signed [7:0] delta_x;
    logic signed [7:0] delta_y;
    logic left_click;

    logic [12:0] write_address;
    logic [12:0] read_address;
    logic [11:0] write_data;
    logic [11:0] read_data;
    logic write_enable;
```

```
    pixel_clock_25MHz clk_div_inst (
        .clk_input(clk),
        .rst(rst),
        .clk_output(clk_25MHz)
    );

    clock_divider_100Hz cursor_clk_div (
        .clk_input(clk_25MHz),
        .rst(rst),
        .clk_output(clk_100Hz)
    );

    vga_controller vga_inst (
        .clk_25MHz(clk_25MHz),
        .rst(rst),
        .hsync(hsync),
        .vsync(vsync),
        .video_on(video_on),
        .pixel_x(pixel_x),
        .pixel_y(pixel_y)
    );

    ps2_mouse_controller mouse_ctrl_inst (
        .clock_100Mhz(clk),
        .reset(rst),
        .Mouse_Data(ps2_data),
        .Mouse_Clk(ps2_clk),
        .delta_x(delta_x),
        .delta_y(delta_y),
        .left_click(left_click)
    );

    bram_dual_port #(
        .DATA_WIDTH(12),
        .ADDR_WIDTH(13)
    ) bram_inst (
        .clk(clk_25MHz),
        .we(write_enable),
        .addr_write(write_address),
        .data_in(write_data),
        .addr_read(read_address),
        .data_out(read_data)
    );

    always_ff @(posedge clk_100Hz or posedge rst) begin
        if (rst) begin
            cursor_x <= 40;
            cursor_y <= 30;
        end else begin
            cursor_x <= (cursor_x + delta_x < 0)   ? 0   :
                        (cursor_x + delta_x > 79)  ? 79  : cursor_x +
delta_x;
            cursor_y <= (cursor_y - delta_y < 0)   ? 0   :
                        (cursor_y - delta_y > 59)  ? 59  : cursor_y -
delta_y;
        end
    end
```

```
    always_comb begin
        case (sw)
            8'b0000_0001: write_data = 12'hF00; // Red
            8'b0000_0010: write_data = 12'h0F0; // Green
            8'b0000_0100: write_data = 12'h00F; // Blue
            8'b0000_1000: write_data = 12'hFF0; // Yellow
            8'b0001_0000: write_data = 12'h0FF; // Cyan
            8'b0010_0000: write_data = 12'hF0F; // Magenta
            8'b0100_0000: write_data = 12'hFFF; // White
            8'b1000_0000: write_data = 12'hF80; // Orange
            default:      write_data = 12'hFFF; // Default to White
        endcase
    end

    always_ff @(posedge clk_100Hz) begin
        write_enable <= 1'b0;

        if (left_click) begin
            write_address <= {cursor_y, cursor_x};
            write_enable <= 1'b1;
        end
    end

    assign read_address = {pixel_y[9:3], pixel_x[9:3]};

    always_comb begin
        if (video_on) begin
            {red, green, blue} = (read_data != 12'h000) ? read_data :
12'hFFF;

            if ((pixel_x >> 3) == cursor_x && (pixel_y >> 3) == cursor_y)
begin
                {red, green, blue} = 12'h000;
            end
        end else begin
            {red, green, blue} = 12'h000;
        end
    end

endmodule


`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer:
//
// Create Date: 12/15/2024 01:22:58 AM
// Design Name:
// Module Name: ps2_mouse_driver
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
```

```
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////


module ps2_mouse_controller(
    input wire         clock_100Mhz,
    input wire         reset,
    input wire         Mouse_Data,
    input wire         Mouse_Clk,
    output reg signed [7:0] delta_x,
    output reg signed [7:0] delta_y,
    output reg         left_click
);

    reg [10:0] shift_reg;
    reg [5:0]  bit_count;
    reg        byte_ready;
    reg [1:0]  bytes_received;
    reg [7:0]  byte_1_data, byte_2_data, byte_3_data;

    always @(negedge Mouse_Clk or posedge reset) begin
        if (reset) begin
            bit_count <= 0;
        end else begin
            shift_reg[bit_count] <= Mouse_Data;
            bit_count <= bit_count + 1;

            if (bit_count == 10) begin
                bit_count <= 0;
                byte_ready <= 1;
            end
        end
    end

    always @(posedge clock_100Mhz or posedge reset) begin
        if (reset) begin
            bytes_received <= 0;
            delta_x <= 0;
            delta_y <= 0;
            left_click <= 0;
        end else if (byte_ready) begin
            byte_ready <= 0;
            case (bytes_received)
                2'd0: byte_1_data <= shift_reg[8:1];
                2'd1: byte_2_data <= shift_reg[8:1];
                2'd2: begin
                    byte_3_data <= shift_reg[8:1];
                    left_click <= byte_1_data[0];
                    delta_x <= byte_2_data;
                    delta_y <= byte_3_data;
                end
```

```
            endcase
            bytes_received <= bytes_received + 1;
            if (bytes_received == 2'd2) bytes_received <= 0;
        end
    end

endmodule

set_property IOSTANDARD LVCMOS33 [get_ports {blue[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {blue[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {blue[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {blue[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {green[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {green[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {green[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {green[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {red[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {red[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {red[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {red[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports hsync]
set_property IOSTANDARD LVCMOS33 [get_ports rst]
set_property IOSTANDARD LVCMOS33 [get_ports vsync]
set_property PACKAGE_PIN P19 [get_ports hsync]
set_property PACKAGE_PIN R19 [get_ports vsync]
set_property PACKAGE_PIN G19 [get_ports {red[0]}]
set_property PACKAGE_PIN H19 [get_ports {red[1]}]
set_property PACKAGE_PIN J19 [get_ports {red[2]}]
set_property PACKAGE_PIN N19 [get_ports {red[3]}]
set_property PACKAGE_PIN N18 [get_ports {blue[0]}]
set_property PACKAGE_PIN L18 [get_ports {blue[1]}]
set_property PACKAGE_PIN K18 [get_ports {blue[2]}]
set_property PACKAGE_PIN J18 [get_ports {blue[3]}]
set_property PACKAGE_PIN J17 [get_ports {green[0]}]
set_property PACKAGE_PIN H17 [get_ports {green[1]}]
set_property PACKAGE_PIN G17 [get_ports {green[2]}]
set_property PACKAGE_PIN D17 [get_ports {green[3]}]




set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN W5 [get_ports clk]


set_property IOSTANDARD LVCMOS33 [get_ports brush_sw]
set_property PACKAGE_PIN R2 [get_ports brush_sw]


set_property PACKAGE_PIN T1 [get_ports rst]


set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
set_property PACKAGE_PIN W13 [get_ports {sw[7]}]



set_property IOSTANDARD LVCMOS33 [get_ports btnC]
set_property IOSTANDARD LVCMOS33 [get_ports btnD]
set_property IOSTANDARD LVCMOS33 [get_ports btnL]
set_property IOSTANDARD LVCMOS33 [get_ports btnR]
set_property IOSTANDARD LVCMOS33 [get_ports btnU]
set_property PACKAGE_PIN T18 [get_ports btnU]
set_property PACKAGE_PIN T17 [get_ports btnR]
set_property PACKAGE_PIN W19 [get_ports btnL]
set_property PACKAGE_PIN U17 [get_ports btnD]
set_property PACKAGE_PIN U18 [get_ports btnC]

set_property IOSTANDARD LVCMOS33 [get_ports ps2_clk]
set_property IOSTANDARD LVCMOS33 [get_ports ps2_data]
set_property PACKAGE_PIN C17 [get_ports ps2_clk]
set_property PACKAGE_PIN B17 [get_ports ps2_data]

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer:
//
// Create Date: 12/15/2024 10:59:41 PM
// Design Name:
// Module Name: tb_ps2_mouse_controller
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////


`timescale 1ns / 1ps
```

```
module tb_ps2_mouse_controller;

    reg         clock_100Mhz;
    reg         reset;
    reg         Mouse_Clk;
    reg         Mouse_Data;
    wire signed [7:0] delta_x;
    wire signed [7:0] delta_y;
    wire        left_click;

    ps2_mouse_controller uut (
        .clock_100Mhz(clock_100Mhz),
        .reset(reset),
        .Mouse_Clk(Mouse_Clk),
        .Mouse_Data(Mouse_Data),
        .delta_x(delta_x),
        .delta_y(delta_y),
        .left_click(left_click)
    );

    initial begin
        clock_100Mhz = 0;
        forever #5 clock_100Mhz = ~clock_100Mhz;
    end

    initial begin
        Mouse_Clk = 1;
        forever #50 Mouse_Clk = ~Mouse_Clk;
    end


    initial begin
        reset = 1;
        #100 reset = 0;
    end

    task send_byte(input [7:0] data);
        integer i;
        for (i = 0; i < 8; i = i + 1) begin
            Mouse_Data = data[i];
            @(negedge Mouse_Clk);
        end
    endtask

    task send_ps2_packet(input [7:0] status, input [7:0] x_move, input [7:0]
y_move);
        Mouse_Data = 0;
        @(negedge Mouse_Clk);

        send_byte(status);
        Mouse_Data = 1;
        @(negedge Mouse_Clk);

        Mouse_Data = 0;
        @(negedge Mouse_Clk);

        send_byte(x_move);
```

```
        Mouse_Data = 1;
        @(negedge Mouse_Clk);

        Mouse_Data = 0;
        @(negedge Mouse_Clk);

        send_byte(y_move);
        Mouse_Data = 1;
        @(negedge Mouse_Clk);
    endtask

    initial begin
        Mouse_Data = 1;

        #200;

        send_ps2_packet(8'b0000_0001, 8'd10, 8'd5);
        #100000;

        send_ps2_packet(8'b0000_0000, 8'hEC, 8'hF1);
        #100000;

        $stop;
    end

    initial begin
        $monitor("Time: %0t | delta_x: %0d | delta_y: %0d | left_click: %b",
$time, delta_x, delta_y, left_click);
    end

endmodule
```