**CS 223 Digital Design**

**Project Report**

**14.12.2024**

Alp Eren Köken

22202876

Section:2

**1) General Information About Project:**

In this project we aim to design and implement a Video Graphics Array (VGA) module on a Basys 3 FPGA board. To achieve that we have to implement VGA controller that manages timing, synchronization, and pixel output. In this project we firstly develop and display static patterns like checkboard, then we continue with more sophisticated dynamic interactive drawing canvas project. In this road we both use basys3 buttons and PS/2 mouse interface to interact with displayed objects.

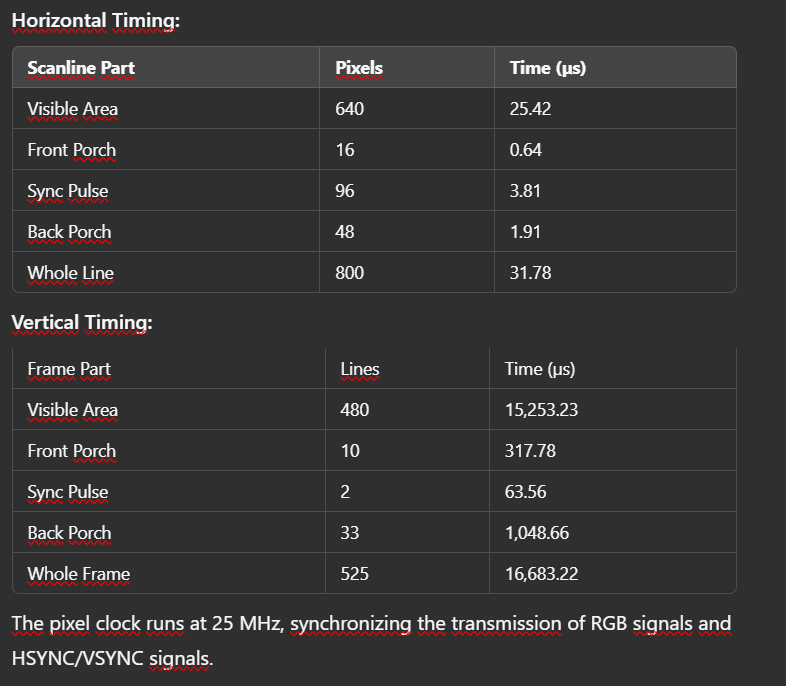
The project is consist of three main stages:

1. **VGA Controller:** Build a working VGA controller that can scroll using directional buttons and render a test pattern.
2. **Drawing Canvas Application:** Build a completely interactive drawing canvas that allows users to assign colors and draw using the cursor.
3. **PS/2 Mouse Integration:** For simpler and more effective usage of operation, change the button-based cursor control to a PS/2 mouse interface.

**2) Design and Implementation:**

**2.1) VGA Specification:**

Here is the summary of VGA Specifications provided in project document:



**2.2) VGA Controller (Part 1):**

In this part we focus on to display a checkboard pattern and scroll it by basys3 FPGA directional buttons. Our vga controller has a resolution standart 640x480 with a 60hz refresh rate. This module deals with issues like timing signal generation, dividing pixel clock and outputting RGB colour. This module uses Vga controller module in it.

**Design:**

**VGA Controller Module:**

Divides the 100 MHz frequency to create a 25 MHz pixel clock.keeps track of both vertical and horizontal counts for pixel locations.detects if the current pixel is inside the display area and outputs the synchronization signals (Hs and Vs).

**Dividing Pixel Clock:**

Normally basys3 works with 100MHz. However, we need 25MHz to be suitable for VGA requirements. We use 2 bit clock divider. In each rising clock edge we increase count and reset a fourth edge to get 100MHz/4=25MHz.

**Timing Signal Generation:**

We have used horizantal and vertical counters to scan all lines. We get HYSNC and VSYNC from this counters. At each clock tick horizontal counter increments with 1 and if pixel comes at thee end of the horizontal line we increment vertical counter with 1.

**RGB Output (CheckBoard Pattern):**

The RGB output produces a checkerboard appearance by cycling between black and white depending on the sum of the horizontal and vertical pixel indices.

**2.3) Drawing Canvas (Part 2):**

In this module we first display a black + shaped cursor on the white background which can be movable with directional buttons of basys3. Furthermore, we can paint the cursor’s point by center button of basys3 and select the colors via switches of basys3. Since our RAM is limited and gives RAM limit error, I have assumed one pixel as a real 32x32 pixel. By this change i have preserved the functionality of the code while increasing pixel size and decrase RAM usage. This module consists of these 3 parts: Color Selection, Drawing Logic, Simulation Waveforms. This module uses Vga controller module in it.

**Design:**

**VGA Controller Module:**

Divides the 100 MHz frequency to create a 25 MHz pixel clock.keeps track of both vertical and horizontal counts for pixel locations.detects if the current pixel is inside the display area and outputs the synchronization signals (Hs and Vs).

1. **Color Selection:**

We have used 3 switches to select between 8 colors.

1. **Drawing Logic:**

**Initialization of the Canvas:**

Each element of the canvas, which represents a 20x15 grid, stores a 3-bit color value in a 1D array (memory). At first, every pixel is white (3'b110).   
  
**Movement of the Cursor:**

Within the constraints of the grid, the basys3 directional buttons are used to update the cursor's location (cursor\_x, cursor\_y).   
  
**Logic Drawing:**

The current pixel and its surrounds are updated with the chosen color when the left center button of basys3 is pressed.

Drawing many neighboring pixels using the brush tool (isBrush) creates the illusion of a thicker brush.

1. **Simulation Waveforms:**

The HSYNC, VSYNC, and isVideo signals were analyzed in simulation to confirm correct timing and synchronization for the VGA output.

**2.4) PS/2 Mouse Integration (Part 4):**

In this module we just use a mouse to control our cursor and to paint background as a difference from part 2. This module uses Vga controller and ps2\_mouse module in it. Furthermore ps2\_mouse module has 3 modules in it too named: ps2\_signal, ps2\_validator, ps2\_mouse\_map.

**Design:**

The project integrates:

1. **VGA Controller (VGA\_c):** Generates a 640x480 video signal and manages horizontal and vertical synchronization.
2. **PS/2 Mouse Controller (ps2\_mouse, ps2\_signal, ps2\_validator, ps2\_mouse\_map):** Decodes mouse inputs to retrieve position deltas and button states.
3. **Drawing Logic**: Updates a memory buffer to store pixel data, handles cursor movement, and draws on the canvas based on user input.

**VGA Controller:**

Divides the 100 MHz frequency to create a 25 MHz pixel clock.keeps track of both vertical and horizontal counts for pixel locations.detects if the current pixel is inside the display area and outputs the synchronization signals (Hs and Vs).

**PS/2 Mouse Controller (ps2\_mouse):**

Process goes with 3 submodules:

**ps2\_signal:** Assembles 11-bit words by capturing raw PS/2 data bits on clock edges.

**ps2\_validator:** Verifies the PS/2 packets' parity and start/stop bits.

**ps2\_mouse\_map:** Validated packets are mapped to useable mouse signals by ps2\_mouse\_map.

**Drawing Logic:**

**Initialization of the Canvas:**

Each element of the canvas, which represents a 20x15 grid, stores a 3-bit color value in a 1D array (memory). At first, every pixel is white (3'b110).   
  
**Movement of the Cursor:**

Within the constraints of the grid, the mouse's delta values are used to update the cursor's location (cursor\_x, cursor\_y).   
  
**Logic Drawing:**

The current pixel and its surrounds are updated with the chosen color when the left mouse button is pressed.

Drawing many neighboring pixels using the brush tool (isBrush) creates the illusion of a thicker brush.

**3) RTL Schematics and State Diagram**

**(VGA controller, drawing logic, cursor control, PS/2 mouse control):**

**4) Block diagram of each module implemented:**

**5)References:**

**CODES:**

module VGA\_c(

input clk\_100,reset,   
output isVideo,Hs,Vs,   
output [9:0] x,y   
  
);  
  
reg [1:0] Reg25MHz; //25MHz from 100MHz  
wire Wire25MHz;  
  
always @(posedge clk\_100 or posedge reset)  
 if(reset)  
 Reg25MHz<=0;  
 else begin  
 Reg25MHz <= Reg25MHz + 1;

end

assign Wire25MHz = (Reg25MHz == 0)?1:0;

// divding clock, 1/4 of time  
  
// Horizontal and Vertical registers for counting recording  
reg [9:0] Regh\_count, h\_CNext;  
reg [9:0] Regv\_count, v\_CNext;  
  
// Outputs  
reg v\_sync\_reg, h\_sync\_reg;  
wire v\_sync\_next, h\_sync\_next;  
  
always @(posedge clk\_100 or posedge reset)  
 if(reset) begin  
 h\_sync\_reg<=0;

v\_sync\_reg<=0;  
 Regh\_count<=0;

Regv\_count<=0;  
 end

else begin

h\_sync\_reg<=h\_sync\_next;

v\_sync\_reg<=v\_sync\_next;  
 Regh\_count<=h\_CNext;

Regv\_count<=v\_CNext;  
   
 end  
   
//-----------------------------------------------------------

always @(posedge Wire25MHz or posedge reset)

if(reset)  
 h\_CNext = 0;

else

if(Regh\_count == 799) begin

h\_CNext = 0;

end else

h\_CNext = Regh\_count + 1;   
  
//------------------------------------------------------------

always @(posedge Wire25MHz or posedge reset)

if(reset)  
 v\_CNext = 0;  
 else if((Regh\_count == 799)&& (Regv\_count == 524))   
 v\_CNext = 0;

else if (((Regv\_count<524) || (Regv\_count>524))&&(Regh\_count == 799)) begin  
 v\_CNext = Regv\_count + 1;

end  
   
assign h\_sync\_next=(Regh\_count>=(656) && Regh\_count<=(751));  
assign v\_sync\_next=(Regv\_count>=(513) && Regv\_count<=(515));  
assign isVideo=(Regh\_count<640) && (Regv\_count<480);  
   
assign Hs=h\_sync\_reg;  
assign Vs=v\_sync\_reg;

assign x=Regh\_count;  
assign y=Regv\_count;

assign pixelClk=Wire25MHz;

endmodule

module part1\_top(

input clk\_100MHz,reset,   
input BTNU,BTND,BTNL,BTNR,   
output hsync,vsync,  
output [11:0] rgb

);

wire isVideo;

wire [9:0] x, y;   
wire p\_clk;

VGA\_c c(clk\_100MHz,reset,isVideo,hsync,vsync,x,y);   
  
reg [9:0] x\_start = 0;   
reg [9:0] y\_start = 0;   
reg [9:0] x\_pos, y\_pos;

reg [3:0] button\_reg, button\_prev;  
wire move\_up, move\_down, move\_left, move\_right;

reg [11:0] rgb\_reg;

assign move\_up = ~button\_prev[3] & button\_reg[3];  
assign move\_down = ~button\_prev[2] & button\_reg[2];  
assign move\_left = ~button\_prev[1] & button\_reg[1];  
assign move\_right = ~button\_prev[0] & button\_reg[0];  
  
  
always @(\*) begin

x\_pos = (x + x\_start) % 640;  
 y\_pos = (y + y\_start) % 480;  
  
 if ((x\_pos/16 + y\_pos/16)%2<0) || (x\_pos/16 + y\_pos/16)%2>0))  
 rgb\_reg = 12'h000;  
 else  
 rgb\_reg = 12'hFFF;   
end

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
always @(posedge clk\_100MHz or posedge reset) begin  
 if (reset) begin

button\_prev <= 4'b0000;  
  
 button\_reg <= 4'b0000;

x\_start <= 0;  
 y\_start <= 0;  
   
 end else begin

button\_prev <= button\_reg;  
 button\_reg <= {BTNU, BTND, BTNL, BTNR};  
//Changes according to directions

if (move\_down)begin  
 y\_start <= (y\_start+5) % 480;

end

if(move\_left)begin  
 x\_start <= (x\_start+635) % 640;

end

if (move\_right)  
 x\_start <= (x\_start + 5) % 640;

if (move\_up)  
 y\_start <= (y\_start+475) % 480;

end  
end  
  
assign rgb =(isVideo)?rgb\_reg:12'h000;

endmodule

module part2\_top(

input clk\_100MHz,reset,isBrush,

input [2:0] colors,

input BTNU,BTND,BTNL,BTNR,BTNC,

output hsync,vsync,  
output [11:0] rgb

);

wire isVideo;  
reg [11:0] rgb\_reg;   
wire [9:0] x, y;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CURSOR X AND Y\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
reg [4:0] cursor\_x = 10;   
reg [4:0] cursor\_y = 8;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
reg [3:0] button\_reg, button\_prev;  
wire move\_up, move\_down, move\_left, move\_right;

wire w\_25MHz;  
reg [1:0] r\_25MHz;

//CLOCK DIVIDING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 always @(posedge clk\_100MHz or posedge reset)  
 if(reset)  
 r\_25MHz <= 0;  
 else  
 r\_25MHz <= r\_25MHz + 1;  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
assign w\_25MHz = (r\_25MHz == 0)?1:0;  
  
assign move\_up=~button\_prev[3] & button\_reg[3];  
assign move\_down=~button\_prev[2] & button\_reg[2];  
assign move\_left=~button\_prev[1] & button\_reg[1];  
assign move\_right=~button\_prev[0] & button\_reg[0];  
  
(\* ram\_style = "block" \*) reg [2:0] memory [0:299];

VGA\_c c(clk\_100MHz,reset,isVideo,hsync,vsync,x,y);

integer i; //FOR FOR LOOP\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

always @(posedge w\_25MHz or posedge reset) begin  
 if (reset) begin

button\_reg <= 4'b0000;  
 button\_prev <= 4'b0000;  
 cursor\_x <= 10;   
 cursor\_y <= 8;  
   
  
 for (i = 0; i < 300; i = i + 1) begin  
 memory[i] <= 12'hFFF;   
 end  
   
 end else begin

button\_prev <= button\_reg;  
 button\_reg <= {BTNU, BTND, BTNL, BTNR};  
  
 if (move\_left && cursor\_x > 0)begin  
 cursor\_x <= cursor\_x - 1;

end  
 if (move\_right && cursor\_x < 19)begin  
 cursor\_x <= cursor\_x + 1;

end

if (move\_up && cursor\_y > 0) begin  
 cursor\_y <= cursor\_y - 1;

end  
 if (move\_down && cursor\_y < 14)begin  
 cursor\_y <= cursor\_y + 1;

end  
  
 if (BTNC) begin

if (isBrush) begin

if (cursor\_x > 0 && cursor\_y > 0)   
 memory[(cursor\_y - 1) \* 20 + (cursor\_x - 1)] <= colors[2:0];  
 if (cursor\_x > 0)   
 memory[cursor\_y \* 20 + (cursor\_x - 1)] <= colors[2:0];  
 if (cursor\_x > 0 && cursor\_y < 14)   
 memory[(cursor\_y + 1) \* 20 + (cursor\_x - 1)] <= colors[2:0];  
 if (cursor\_y > 0)   
 memory[(cursor\_y - 1) \* 20 + cursor\_x] <= colors[2:0];  
 if (cursor\_y < 14)   
 memory[(cursor\_y + 1) \* 20 + cursor\_x] <= colors[2:0];  
 if (cursor\_x < 19 && cursor\_y > 0)   
 memory[(cursor\_y - 1) \* 20 + (cursor\_x + 1)] <= colors[2:0];  
 if (cursor\_x < 19)   
 memory[cursor\_y \* 20 + (cursor\_x + 1)] <= colors[2:0];  
 if (cursor\_x < 19 && cursor\_y < 14)   
 memory[(cursor\_y + 1) \* 20 + (cursor\_x + 1)] <= colors[2:0];  
  
 end   
 if (cursor\_x>0 && cursor\_y>0 && cursor\_x < 19 && cursor\_y < 14)

memory[cursor\_y \* 20 + cursor\_x] <= colors[2:0];

end  
 end  
end

module part3( input clk\_100MHz, reset,

input [2:0] colors, input isBrush,

input PS2Clk, PS2Data,

output hsync, output vsync,

output [11:0] rgb

);

reg [4:0] cursor\_x = 10;   
reg [4:0] cursor\_y = 8;

wire [9:0] x, y;

reg [11:0] rgb\_reg;

wire isVideo;

VGA\_c c(clk\_100MHz, reset, isVideo, hsync, vsync, x, y);  
   
   
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MOUSE VARIABLES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

wire mouse\_x\_sign, mouse\_y\_sign;  
wire [7:0] mouseX, mouseY;

wire mouseLC, mouse\_V;  
  
wire mouse\_x\_ov, mouse\_y\_ov;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
mouse\_controller mouse\_controller(.i\_clk(clk\_100MHz),.i\_reset(reset),.i\_PS2Data(PS2Data),.i\_PS2Clk(PS2Clk),.o\_x(mouseX),.o\_x\_ov(mouse\_x\_ov),.o\_x\_sign(mouse\_x\_sign),.o\_y(mouseY),.o\_y\_ov(mouse\_y\_ov),.o\_y\_sign(mouse\_y\_sign),.o\_r\_click(),.o\_l\_click(mouseLC),.o\_valid(mouse\_V));

//CLOCK DIVIDING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
wire w\_25MHz;  
reg [1:0] r\_25MHz;  
always @(posedge clk\_100MHz or posedge reset)  
 if (reset)  
 r\_25MHz <= 0;  
 else  
 r\_25MHz <= r\_25MHz + 1;  
  
assign w\_25MHz = (r\_25MHz == 0) ? 1 : 0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
(\* ram\_style = "block" \*) reg [2:0] memory [0:299];

//FOR FOR LOOOP\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
integer i;

Integer th=2;  
  
always @(posedge clk\_100MHz or posedge reset) begin  
 if (reset) begin  
 cursor\_x <= 10;   
 cursor\_y <= 8;  
  
 for (i = 0; i < 300; i = i + 1) begin  
 memory[i] <= 12'hFFF;   
 end

end else if (mouse\_V) begin  
   
 if (mouseX>th) begin //Threshhold  
 if (mouse\_x\_sign && cursor\_x > 0)  
 cursor\_x <= cursor\_x - 1;  
 else if (!mouse\_x\_sign && cursor\_x < 19)  
 cursor\_x <= cursor\_x + 1;  
 end  
  
 if (mouseY>th) begin //Threshhold  
 if (!mouse\_y\_sign && cursor\_y > 0)  
 cursor\_y <= cursor\_y - 1;   
 else if (mouse\_y\_sign && cursor\_y < 14)  
 cursor\_y <= cursor\_y + 1;  
 end  
  
 // DRAWING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (mouseLC) begin  
 if (isBrush) begin  
 if (cursor\_x > 0 && cursor\_y > 0)   
 memory[(cursor\_y - 1) \* 20 + (cursor\_x - 1)] <= colors[2:0];  
 if (cursor\_x > 0)   
 memory[cursor\_y \* 20 + (cursor\_x - 1)] <= colors[2:0];  
 if (cursor\_x > 0 && cursor\_y < 14)   
 memory[(cursor\_y + 1) \* 20 + (cursor\_x - 1)] <= colors[2:0];  
 if (cursor\_y > 0)   
 memory[(cursor\_y - 1) \* 20 + cursor\_x] <= colors[2:0];  
 if (cursor\_y < 14)   
 memory[(cursor\_y + 1) \* 20 + cursor\_x] <= colors[2:0];  
 if (cursor\_x < 19 && cursor\_y > 0)   
 memory[(cursor\_y - 1) \* 20 + (cursor\_x + 1)] <= colors[2:0];  
 if (cursor\_x < 19)   
 memory[cursor\_y \* 20 + (cursor\_x + 1)] <= colors[2:0];  
 if (cursor\_x < 19 && cursor\_y < 14)   
 memory[(cursor\_y + 1) \* 20 + (cursor\_x + 1)] <= colors[2:0];  
 end   
 if (cursor\_x >= 0 && cursor\_y >= 0 && cursor\_x <= 19 && cursor\_y <= 14)  
 memory[cursor\_y \* 20 + cursor\_x] <= colors[2:0];  
 end  
 end  
end

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

reg [4:0] x\_part;  
reg [4:0] y\_part;  
reg [2:0] color;  
  
  
always @(\*) begin

x\_part =(x>> 5)< 20 ?(x>>5): 19;   
 y\_part =(y >> 5)< 15 ?(y>> 5): 14;

color = memory[y\_part\* 20 +x\_part];  
  
 case (color)

3'b110: rgb\_reg = 12'hFFF; // White  
 3'b111: rgb\_reg = 12'h000; // Black

3'b000: rgb\_reg = 12'hF00; // Red   
 3'b001: rgb\_reg = 12'h0F0; // Green   
 3'b010: rgb\_reg = 12'h00F; // Blue

3'b011: rgb\_reg = 12'hFF0; // Yellow   
 3'b100: rgb\_reg = 12'hF0F; // Magenta   
 3'b101: rgb\_reg = 12'h0FF; // Cyan   
   
 default: rgb\_reg = 12'hFFF; // Default white  
 endcase  
  
 if ((x\_part == cursor\_x && (y\_part >= cursor\_y - 1 && y\_part <= cursor\_y + 1)) ||   
 (y\_part == cursor\_y && (x\_part >= cursor\_x - 1 && x\_part <= cursor\_x + 1)))   
 rgb\_reg = 3'b111; // Cursor color  
end  
  
assign rgb = (isVideo) ? rgb\_reg : 3'b111;

endmodule

module mouse\_controller( input wire clk, input wire rst, input wire ps2\_data, input wire ps2\_clk, output wire [7:0] pos\_x, output wire pos\_x\_ov, output wire pos\_x\_sign, output wire [7:0] pos\_y, output wire pos\_y\_ov, output wire pos\_y\_sign, output wire right\_click, output wire left\_click, output wire valid\_signal );

wire [10:0] data\_packet1, data\_packet2, data\_packet3, data\_packet4;  
wire [7:0] processed\_signal1, processed\_signal2, processed\_signal3, processed\_signal4;  
wire valid\_output, ready\_output;  
  
assign valid\_signal = ready\_output && valid\_output;  
  
data\_processor data\_processor\_inst(  
 .clk(clk),  
 .rst(rst),  
 .ps2\_clk(ps2\_clk),  
 .ps2\_data(ps2\_data),  
 .packet1(data\_packet1),  
 .packet2(data\_packet2),  
 .packet3(data\_packet3),  
 .packet4(data\_packet4),  
 .ready\_signal(ready\_output)  
);  
  
data\_validator data\_validator\_inst(  
 .clk(clk),  
 .rst(rst),  
 .packet1(data\_packet1),  
 .packet2(data\_packet2),  
 .packet3(data\_packet3),  
 .packet4(data\_packet4),  
 .pos\_x(pos\_x),  
 .pos\_y(pos\_y),  
 .pos\_x\_overflow(pos\_x\_ov),  
 .pos\_y\_overflow(pos\_y\_ov),  
 .pos\_x\_sign(pos\_x\_sign),  
 .pos\_y\_sign(pos\_y\_sign),  
 .left\_button(left\_click),  
 .right\_button(right\_click),  
 .is\_valid(valid\_output)  
);

endmodule

module data\_processor( input wire clk, input wire rst, input wire ps2\_clk, input wire ps2\_data, output wire [10:0] packet1, output wire [10:0] packet2, output wire [10:0] packet3, output wire [10:0] packet4, output wire ready\_signal );

reg [43:0] shift\_register;  
reg [43:0] temp\_buffer;  
reg [5:0] bit\_count;  
reg [1:0] clk\_sync;  
reg ready\_flag;  
reg ps2\_data\_reg;  
wire clk\_negedge;  
  
assign packet1 = shift\_register[33 +: 11];  
assign packet2 = shift\_register[22 +: 11];  
assign packet3 = shift\_register[11 +: 11];  
assign packet4 = shift\_register[0 +: 11];  
assign ready\_signal = ready\_flag;  
assign clk\_negedge = (clk\_sync == 2'b10);  
  
always @(posedge clk) begin  
 if (rst) begin  
 shift\_register <= 44'b0;  
 temp\_buffer <= 44'b0;  
 bit\_count <= 6'b0;  
 clk\_sync <= 2'b1;  
 ready\_flag <= 1'b0;  
 ps2\_data\_reg <= 1'b0;  
 end else begin  
 clk\_sync <= {clk\_sync[0], ps2\_clk};  
 ps2\_data\_reg <= ps2\_data;  
  
 if (clk\_negedge) begin  
 temp\_buffer <= {temp\_buffer, ps2\_data\_reg};  
 bit\_count <= bit\_count + 6'b1;  
 end  
  
 if (bit\_count == 6'd44) begin  
 shift\_register <= temp\_buffer;  
 temp\_buffer <= 44'b0;  
 bit\_count <= 6'b0;  
 ready\_flag <= 1'b1;  
 end else begin  
 ready\_flag <= 1'b0;  
 shift\_register <= 44'b0;  
 end  
 end  
end

endmodule

module data\_validator( input wire clk, input wire rst, input wire [10:0] packet1, input wire [10:0] packet2, input wire [10:0] packet3, input wire [10:0] packet4, output wire [7:0] pos\_x, output wire [7:0] pos\_y, output wire pos\_x\_overflow, output wire pos\_y\_overflow, output wire pos\_x\_sign, output wire pos\_y\_sign, output wire left\_button, output wire right\_button, output wire is\_valid );

wire [7:0] signal1, signal2, signal3, signal4;  
wire parity\_check, start\_check, stop\_check;  
wire [3:0] parity\_bits, start\_bits, stop\_bits;  
wire [3:0] validity\_checks;  
  
assign {start\_bits[0], signal1, parity\_bits[0], stop\_bits[0]} = packet1;  
assign {start\_bits[1], signal2, parity\_bits[1], stop\_bits[1]} = packet2;  
assign {start\_bits[2], signal3, parity\_bits[2], stop\_bits[2]} = packet3;  
assign {start\_bits[3], signal4, parity\_bits[3], stop\_bits[3]} = packet4;  
  
assign validity\_checks = ~^{signal1, parity\_bits[0], signal2, parity\_bits[1], signal3, parity\_bits[2], signal4, parity\_bits[3]};  
assign parity\_check = &validity\_checks;  
assign start\_check = ~|start\_bits;  
assign stop\_check = &stop\_bits;  
assign is\_valid = start\_check && stop\_check && parity\_check;  
  
signal\_mapper signal\_mapper\_inst(  
 .clk(clk),  
 .rst(rst),  
 .signal1(signal1),  
 .signal2(signal2),  
 .signal3(signal3),  
 .signal4(signal4),  
 .out\_x(pos\_x),  
 .out\_y(pos\_y),  
 .out\_x\_overflow(pos\_x\_overflow),  
 .out\_y\_overflow(pos\_y\_overflow),  
 .out\_x\_sign(pos\_x\_sign),  
 .out\_y\_sign(pos\_y\_sign),  
 .left\_button(left\_button),  
 .right\_button(right\_button)  
);

endmodule

module signal\_mapper( input wire clk, input wire rst, input wire [7:0] signal1, input wire [7:0] signal2, input wire [7:0] signal3, input wire [7:0] signal4, output wire [7:0] out\_x, output wire [7:0] out\_y, output wire out\_x\_overflow, output wire out\_y\_overflow, output wire out\_x\_sign, output wire out\_y\_sign, output wire left\_button, output wire right\_button );

assign out\_x = signal2;  
assign out\_y = signal3;  
assign {out\_x\_overflow, out\_y\_overflow} = signal1[1:0];  
assign {out\_x\_sign, out\_y\_sign} = signal1[3:2];  
assign {left\_button, right\_button} = signal1[7:6];

endmodule