**Team members:** Niannian Ji (1002492698), Yizhi Xu (1002082989)
**Station:** 35

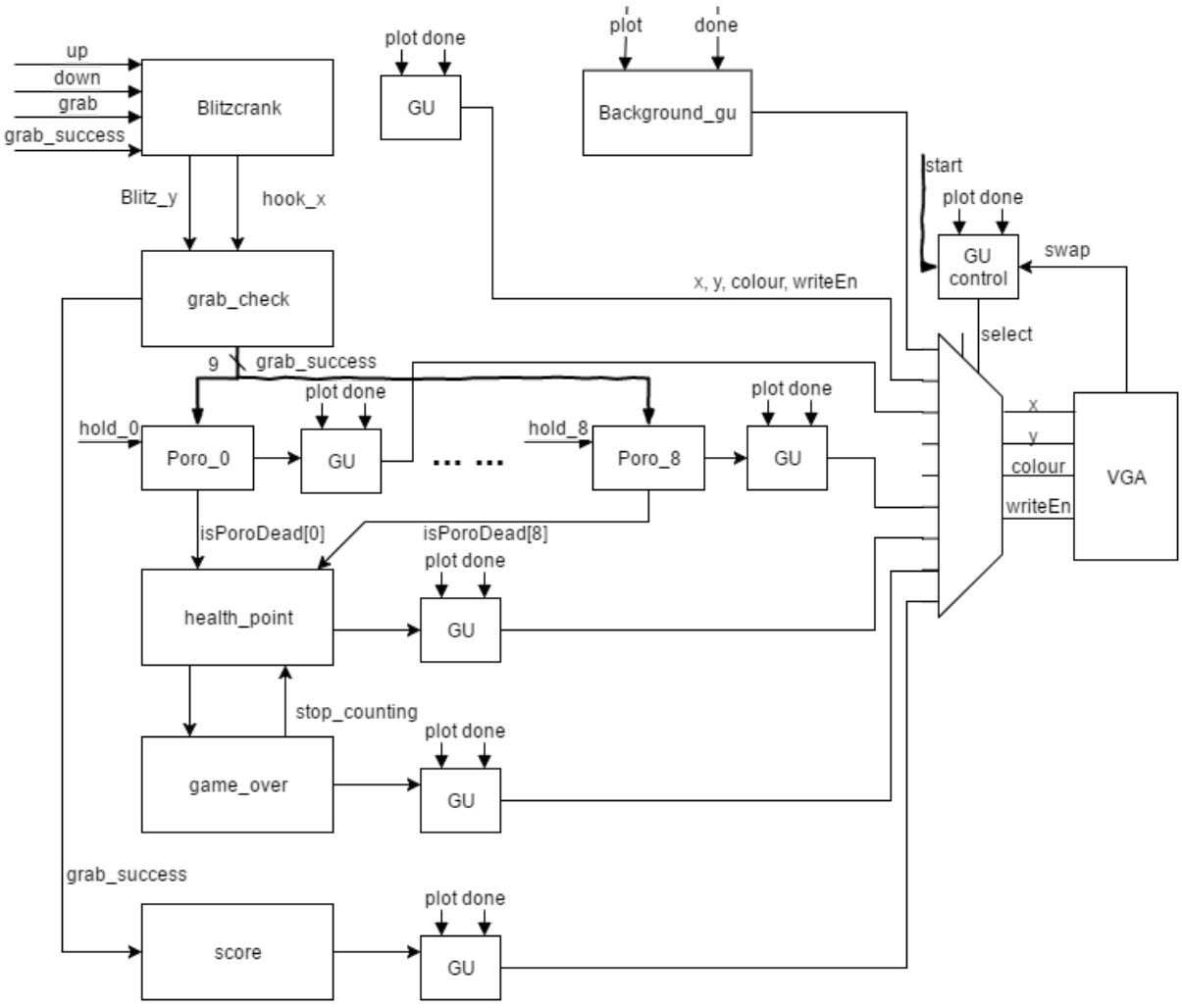
## 1.0 Introduction

This project is a side-scrolling game where the main character -- Blitzcrank is chasing and trying to grab the other character -- poro. We come up with this design idea for several reasons. First, we want to apply what we learned from class to our design project, and game design is a much involved process, thus we can gain a deeper understanding of the course material. Also, we like creating art. Inventing characters, drawing background and creating digital scenes brings this project lots of fun. Moreover, both of our two group members are big fans of the video game --League of Legend. We choose this specific theme so that we actually enjoy and love this project and put all our passion in it.

Our goal is to implement the proposed functionalities of this game, get familiar with the game design process and apply in-class knowledge to hands-on project.

## 2.0 Design Decomposition

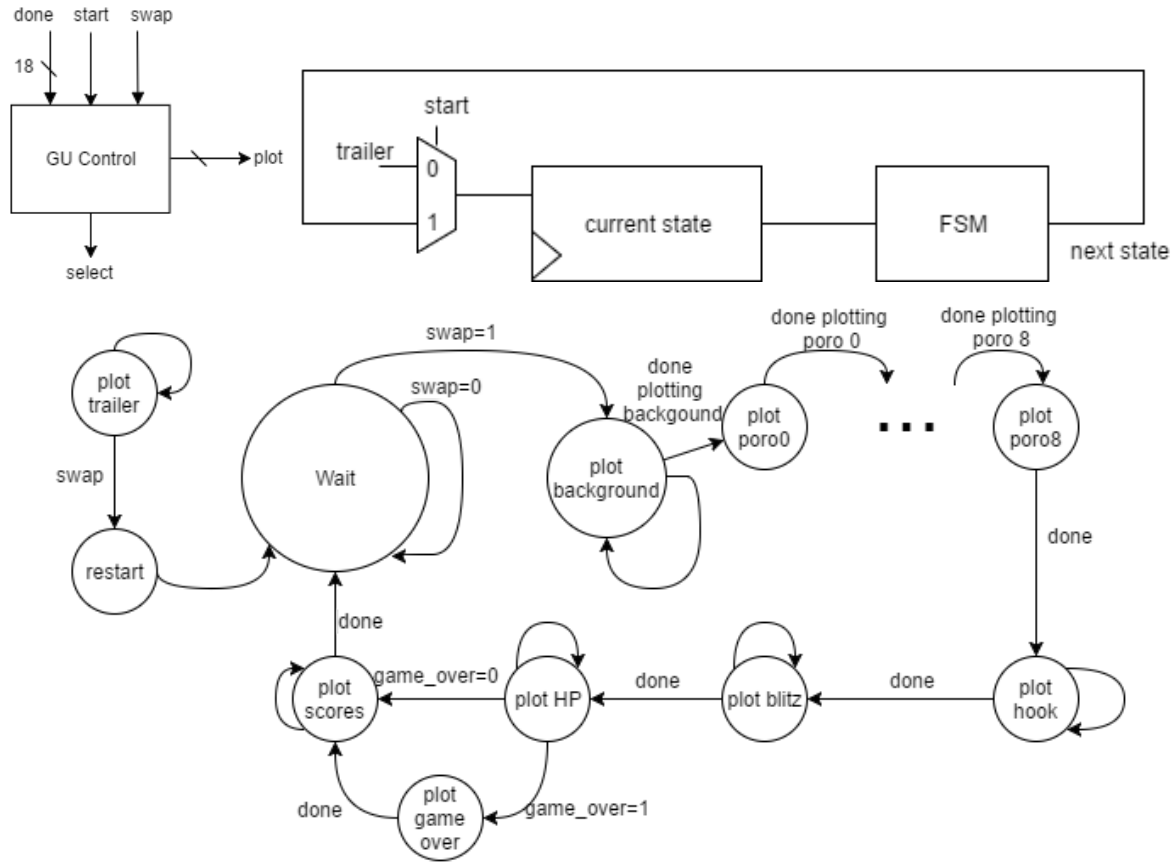
2.1 Controllers:




Graph-2.1: Block Diagram of The Control Unit

Grab_check: detects which poro is grabbed
Poro(0-8): controls poro's position, and check if a poro is dead
HealthPoint: counts from 5 down to 0, the game is over when hp becomes 0
GameOver: shows the game over screen, and disable health point module
Score: counts the score in decimal, can count from 0 to 999

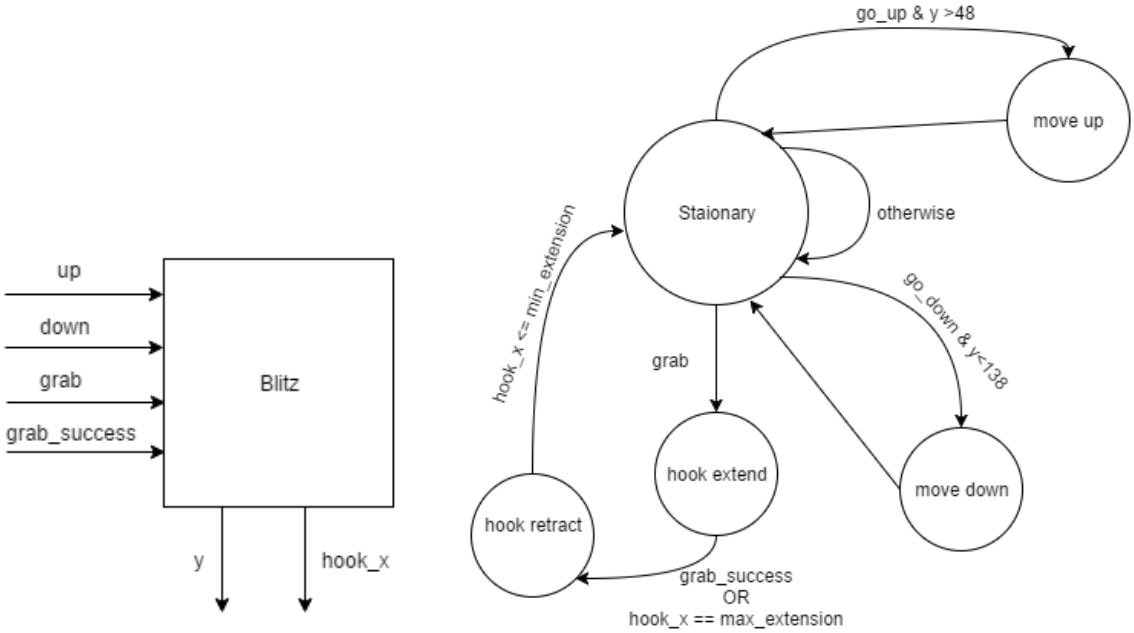
2.2 Graphing Units(GU):




Graph-2.2: Block Diagram and State Diagram of The Graphing Unit

Graphing units are used to draw the characters, scores, game over screen, trailer screen, and health points. They output the coordinate and colour of a picture to VGA. However background GU is slightly different which can draw a picture and scroll it during each frame. When it goes to the plot states, the corresponding "plot" signal becomes 1, and "select" signal is used to select the proper output. "Start" signal is controlled by SW[0], when SW[0] is 0, the trailer is shown, and whenever SW[0] is 1, it goes to state "restart" which will let "resetn" become 0 to reset everything, and the game starts.

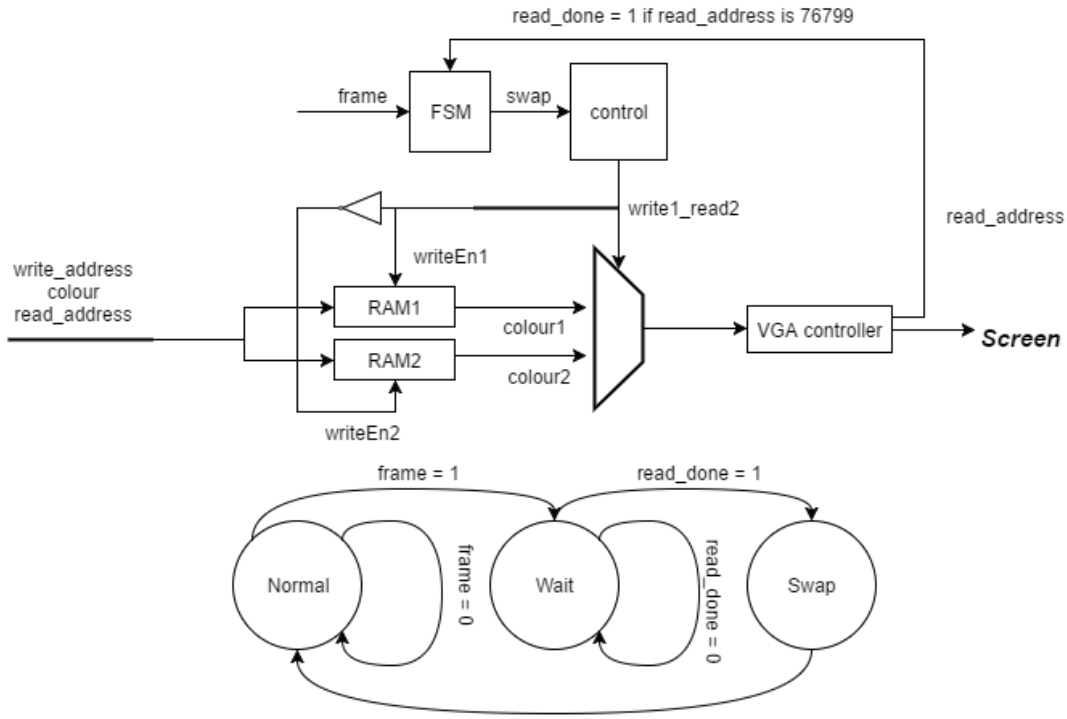
## 2.3 Blitz:




Graph-2.3: Block Diagram and State Diagram of The Blitzcrank Controlling Module

This module controls blitzcrank's movement and grab action, its y is restricted to the range of 48 to 138. The hook will be activated when blitzcrank is in stationary state and it will retract when the hook reaches to maximum range or it successfully grabs a poro.

## 2.4 VGA with double buffers:




Graph-2.4: Block Diagram and State Diagram of The VGA Module

The original version of VGA module has only 1 RAM,. which causes flickering when the game runs at 60FPS. To solve this problem, we added another RAM into this module. SO that the top module can write into one RAM, while the VGA can read the data which are written at last cycle from another RAM.

Swap is 1 when top module is ready to write(frame=1) in data and VGA finishes reading the last memory address.

## 3.0 Report on Success [Appendix A.1]

I. Feature to restart the game and end-of-game check
   - A trailer screen is displayed at the beginning of the game
   - Player is able to start the game at any time in the trailer screen
   - End-of-game screen is displayed when player's health point dropped to zero
   - Player can restart the game at any time

II. Movement of blitzcrank (player controller character), poros (non player controlled character) and scrolling background
   - Player is able to control the up and down movement of blitzcrank
   - Poros keep moving in the horizontal direction until being grabbed or reached left position limit
   - Background keeps moving the the left smoothly

III. Grabbing feature
   - Player is able to extend the hook of Blitzcrank
   - If the hook reached one of the poros in its extension, the hook will retract with the poro together
   - The hook retracts after it reaches maximum extension

IV. Score and health point counting
   - Score counts up whenever the player grabs one poro successfully
   - Health point counts down whenever a poro reaches left position limit without being grabbed, the default health point is set to five
   - End of the game is reached when health point reaches zero
   - Score does not change after end of game is reached

V. Difficulty control
   - Player is able to set the difficulty level of the game by controlling the amount of poros generated on the screen

VI. Flicker-free
   - The VGA output of this project is smooth without any flicker due to the implementation of a double buffer VGA module

## 4.0 Future Improvement

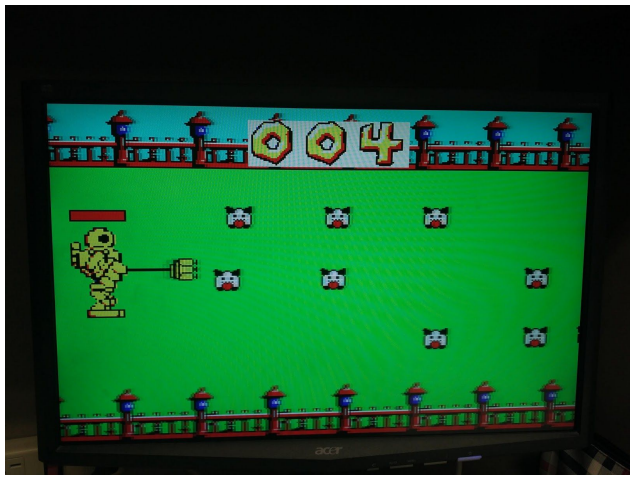
We accomplished most of our goals on this project. And thanks to our TA, we had a clear scope and reasonable milestones through the whole process.

If we were to start all over again, we would like to try more inputs/outputs such as keyboard, audio output etc.

## Appendix

A.1 Visual Presentation of the Project

| | |
|---|---|
| A.1.1 Trailer Screen |  |
| A.1.2 End-of-Game Screen (Health Bar is Empty) |  |
| A.1.3 Movement of Blitzcrank |  |

| | |
|---|---|
| A.1.4 Extension/Retraction of The Hook |  |
| A.1.5 Grabbing (Score Counts Up) |  |

A.2 Schematics

Schematic diagram is attached at the last page. The digital version of this PDF can be zoomed big enough to see the name of wires clearly on the diagram.

A.3 Verilog Code

```
//=====================graph.v (top module)=======================
// top module

module graph
	(
		CLOCK_50,				//	On Board 50 MHz
		KEY,
		SW,
		LEDR,
		HEX0,
		HEX1,
		HEX2,
```

```
                HEX3,
                // Your inputs and outputs here
                // The ports below are for the VGA output.  Do not change.
                VGA_CLK,                                //      VGA Clock
                VGA_HS,                                 //      VGA
H_SYNC
                VGA_VS,                                 //      VGA
V_SYNC
                VGA_BLANK_N,                            //      VGA
BLANK
                VGA_SYNC_N,                             //      VGA
SYNC
                VGA_R,                                  //      VGA Red[9:0]
                VGA_G,                                  //      VGA
Green[9:0]
                VGA_B                                   //      VGA Blue[9:0]
        );
        output [6:0] HEX0, HEX1, HEX2, HEX3;
        input                   CLOCK_50;               //      50 MHz
        input   [3:0]   KEY;
        input   [0:0]   SW;
        output  [9:0]   LEDR;
        // Declare your inputs and outputs here
        // Do not change the following outputs
        output                  VGA_CLK;                //      VGA Clock
        output                  VGA_HS;                 //      VGA
H_SYNC
        output                  VGA_VS;                 //      VGA
V_SYNC
        output                  VGA_BLANK_N;            //      VGA
BLANK
        output                  VGA_SYNC_N;             //      VGA
SYNC
        output  [9:0]   VGA_R;                  //      VGA Red[9:0]
        output  [9:0]   VGA_G;                  //      VGA Green[9:0]
        output  [9:0]   VGA_B;                  //      VGA Blue[9:0]

        wire resetn;
        reg go_up, go_down, grab;
        reg go_up_buf, go_down_buf, grab_buf;
        reg restart;
        assign resetn = KEY[0] && !restart;


        // Create the colour, x, y and writeEn wires that are inputs to the controller.
```

```
	reg [2:0] colour;
	reg [8:0] x;
	reg [7:0] y;


	wire [14:0]	writeEn; //0 -> blitz, 1-9 -> poro, 10 ->BG, ll -> hook, 12 -> arm,
13->trailer, 14->hp
	reg writeEn_vga;

	reg [14:0]	plot; //0 -> blitz, 1-9 -> poro, 10 ->BG, ll -> hook, 12 -> arm, 13->trailer,
14->hp
	wire [14:0]  done; //0 -> blitz, 1-9 -> poro, 10 ->BG, ll -> hook, 12 -> arm, 13->trailer,
14->hp
	wire clk;
	wire frame; // empty for now

	DelayCounter dc(.clock(CLOCK_50), .reset_n(resetn), .maxcount(28'd833334),
.enable(frame)); //60 fps

	assign clk = CLOCK_50;

	always@(posedge clk) begin
		go_up_buf <= ~KEY[3];
		go_down_buf <= ~KEY[2];
		grab_buf <= ~KEY[1];
	end

	always@(posedge clk) begin
		go_up <= go_up_buf;
		go_down <=go_down_buf;
		grab <= grab_buf;
	end

	//========================= control and gu connection
================================
	wire [7:0] blitz_y, blitz_hook_y;
	wire [8:0] blitz_hook_x;
	wire [7:0] plot_y_b, plot_y_bh, plot_y_ba, plot_y_hp; //bh for blitz hook, ba for arm
	wire [8:0] plot_x_b, plot_x_bh, plot_x_ba, plot_x_hp;
	wire [2:0] colour_b, colour_bh, colour_ba, colour_hp;

	wire [8:0] grab_success;
	assign LEDR[8:0] = grab_success;
	assign LEDR[9] = blitz_grab_success;
```

```
	wire blitz_grab_success =
grab_success[0]|grab_success[1]|grab_success[2]|grab_success[3]|grab_success[4]|grab_s
uccess[5]|grab_success[6]|grab_success[7]|grab_success[8];
	// *blitz
	blitz_pos blitz(go_up,go_down,clk,frame,resetn,grab,
blitz_grab_success,blitz_hook_x,blitz_hook_y,blitz_y);
	blitz_gu
b_gu(clk,blitz_y,resetn,plot[0],colour_b,plot_x_b,plot_y_b,writeEn[0],done[0]);
	hp_gu
hpgu1(clk,blitz_y,3'd3,resetn,plot[14],colour_hp,plot_x_hp,plot_y_hp,writeEn[14],done[14]);
	blitz_hook_gu
bh_gu(clk,blitz_hook_x,blitz_hook_y,resetn,plot[11],colour_bh,plot_x_bh,plot_y_bh,writeEn[1
1],done[11]);
	blitz_arm_gu
ba_gu(clk,blitz_hook_x,blitz_hook_y,resetn,plot[12],colour_ba,plot_x_ba,plot_y_ba,writeEn[1
2],done[12]);
	// *poros

	wire [8:0] hold;
	wire [8:0] score_p;

	wire [8:0] x_p0;
	wire [8:0] x_p1;
	wire [8:0] x_p2;
	wire [8:0] x_p3;
	wire [8:0] x_p4;
	wire [8:0] x_p5;
	wire [8:0] x_p6;
	wire [8:0] x_p7;
	wire [8:0] x_p8;

	wire [7:0] y_p0;
	wire [7:0] y_p1;
	wire [7:0] y_p2;
	wire [7:0] y_p3;
	wire [7:0] y_p4;
	wire [7:0] y_p5;
	wire [7:0] y_p6;
	wire [7:0] y_p7;
	wire [7:0] y_p8;

	wire [2:0] colour_p0 ;
	wire [2:0] colour_p1 ;
	wire [2:0] colour_p2 ;
	wire [2:0] colour_p3 ;
	wire [2:0] colour_p4 ;
```

```
wire [2:0] colour_p5 ;
wire [2:0] colour_p6 ;
wire [2:0] colour_p7 ;
wire [2:0] colour_p8 ;

wire [8:0] plot_x_p0;
wire [8:0] plot_x_p1;
wire [8:0] plot_x_p2;
wire [8:0] plot_x_p3;
wire [8:0] plot_x_p4;
wire [8:0] plot_x_p5;
wire [8:0] plot_x_p6;
wire [8:0] plot_x_p7;
wire [8:0] plot_x_p8;

wire [7:0] plot_y_p0;
wire [7:0] plot_y_p1;
wire [7:0] plot_y_p2;
wire [7:0] plot_y_p3;
wire [7:0] plot_y_p4;
wire [7:0] plot_y_p5;
wire [7:0] plot_y_p6;
wire [7:0] plot_y_p7;
wire [7:0] plot_y_p8;


poro_pos p0(clk, frame, resetn, grab_success[0], hold[0], score_p[0], x_p0);
poro_pos p1(clk, frame, resetn, grab_success[1], hold[1], score_p[1], x_p1);
poro_pos p2(clk, frame, resetn, grab_success[2], hold[2], score_p[2], x_p2);
poro_pos p3(clk, frame, resetn, grab_success[3], hold[3], score_p[3], x_p3);
poro_pos p4(clk, frame, resetn, grab_success[4], hold[4], score_p[4], x_p4);
poro_pos p5(clk, frame, resetn, grab_success[5], hold[5], score_p[5], x_p5);
poro_pos p6(clk, frame, resetn, grab_success[6], hold[6], score_p[6], x_p6);
poro_pos p7(clk, frame, resetn, grab_success[7], hold[7], score_p[7], x_p7);
poro_pos p8(clk, frame, resetn, grab_success[8], hold[8], score_p[8], x_p8);

reg [8:0] hold_counter;

assign hold[0] = !(hold_counter >= 9'd0);
assign hold[1] = !(hold_counter >= 9'd63);
assign hold[2] = !(hold_counter >= 9'd127);
assign hold[3] = !(hold_counter >= 9'd191);
assign hold[4] = !(hold_counter >= 9'd255);
assign hold[5] = !(hold_counter >= 9'd319);
assign hold[6] = !(hold_counter >= 9'd383);
assign hold[7] = !(hold_counter >= 9'd447);
```

```
	assign hold[8] = !(hold_counter >= 9'd511);

	//hold counter
	always@ (posedge clk)
	if(frame) begin
		if (!resetn)
			hold_counter <= 9'd0;
		else if (hold_counter < 9'd511)
			hold_counter <= hold_counter + 9'd1;
	end

	// randomize y later temp //23 pixels distance starting 48+22 ending 138+22
	assign y_p0 = 8'd70;
	assign y_p1 = 8'd70;
	assign y_p2 = 8'd70;
	assign y_p3 = 8'd115;
	assign y_p4 = 8'd115;
	assign y_p5 = 8'd115;
	assign y_p6 = 8'd160;
	assign y_p7 = 8'd160;
	assign y_p8 = 8'd160;

	// *connect hook to poros
	grab_check gc0(grab,blitz_hook_x,blitz_hook_y,plot_x_p0,y_p0,grab_success[0]);
	grab_check gc1(grab,blitz_hook_x,blitz_hook_y,plot_x_p1,y_p1,grab_success[1]);
	grab_check gc2(grab,blitz_hook_x,blitz_hook_y,plot_x_p2,y_p2,grab_success[2]);
	grab_check gc3(grab,blitz_hook_x,blitz_hook_y,plot_x_p3,y_p3,grab_success[3]);
	grab_check gc4(grab,blitz_hook_x,blitz_hook_y,plot_x_p4,y_p4,grab_success[4]);
	grab_check gc5(grab,blitz_hook_x,blitz_hook_y,plot_x_p5,y_p5,grab_success[5]);
	grab_check gc6(grab,blitz_hook_x,blitz_hook_y,plot_x_p6,y_p6,grab_success[6]);
	grab_check gc7(grab,blitz_hook_x,blitz_hook_y,plot_x_p7,y_p7,grab_success[7]);
	grab_check gc8(grab,blitz_hook_x,blitz_hook_y,plot_x_p8,y_p8,grab_success[8]);


	// *poro_gu
	poro_gu
p_gu0(clk,x_p0,y_p0,resetn,plot[1],colour_p0,plot_x_p0,plot_y_p0,writeEn[1],done[1]);
	poro_gu
p_gu1(clk,x_p1,y_p1,resetn,plot[2],colour_p1,plot_x_p1,plot_y_p1,writeEn[2],done[2]);
	poro_gu
p_gu2(clk,x_p2,y_p2,resetn,plot[3],colour_p2,plot_x_p2,plot_y_p2,writeEn[3],done[3]);
	poro_gu
p_gu3(clk,x_p3,y_p3,resetn,plot[4],colour_p3,plot_x_p3,plot_y_p3,writeEn[4],done[4]);
	poro_gu
p_gu4(clk,x_p4,y_p4,resetn,plot[5],colour_p4,plot_x_p4,plot_y_p4,writeEn[5],done[5]);
```

```
        poro_gu
p_gu5(clk,x_p5,y_p5,resetn,plot[6],colour_p5,plot_x_p5,plot_y_p5,writeEn[6],done[6]);
        poro_gu
p_gu6(clk,x_p6,y_p6,resetn,plot[7],colour_p6,plot_x_p6,plot_y_p6,writeEn[7],done[7]);
        poro_gu
p_gu7(clk,x_p7,y_p7,resetn,plot[8],colour_p7,plot_x_p7,plot_y_p7,writeEn[8],done[8]);
        poro_gu
p_gu8(clk,x_p8,y_p8,resetn,plot[9],colour_p8,plot_x_p8,plot_y_p8,writeEn[9],done[9]);

        // *background_gu
        wire [8:0] plot_x_bg;
        wire [7:0] plot_y_bg;
        wire [2:0] colour_bg ;
        background_gu
bg_gu(frame,clk,resetn,plot[10],colour_bg,plot_x_bg,plot_y_bg,writeEn[10],done[10]);

        // *trailer_gu
        wire [8:0] plot_x_tl;
        wire [7:0] plot_y_tl;
        wire [2:0] colour_tl ;
        trailer_gu
tl_gu(frame,clk,resetn,plot[13],colour_tl,plot_x_tl,plot_y_tl,writeEn[13],done[13]);

        // *FSM
        reg [4:0] current_state, next_state;
        localparam  S_PLOT_WAIT                 = 5'd0,
                            S_PLOT_BG                   = 5'd1,
           S_PLOT_P0                = 5'd2,
                            S_PLOT_P1                   = 5'd3,
                            S_PLOT_P2                       = 5'd4,
                            S_PLOT_P3                   = 5'd5,
                            S_PLOT_P4                   = 5'd6,
                            S_PLOT_P5                   = 5'd7,
                            S_PLOT_P6                   = 5'd8,
                            S_PLOT_P7                   = 5'd9,
                            S_PLOT_P8                   = 5'd10,
                            S_PLOT_BLITZ                = 5'd11,
                            S_PLOT_BLITZ_HOOK               = 5'd12,
                            S_PLOT_BLITZ_ARM                = 5'd13,
                            S_PLOT_TRAILER                  = 5'd14,
                            S_RESTART                       = 5'd15,
                            S_PLOT_HP                       = 5'd16;

        always@(*)
    begin: state_table
        case (current_state)
```

```
                                S_PLOT_TRAILER: next_state = (swap) ? S_RESTART :
S_PLOT_TRAILER;
                                S_RESTART: next_state = S_PLOT_WAIT;
                                S_PLOT_WAIT: next_state = (swap) ? S_PLOT_BG :
S_PLOT_WAIT;
                                S_PLOT_BG:   next_state = done[10] ? S_PLOT_P0 :
S_PLOT_BG;
                                S_PLOT_P0: next_state = done[1] ? S_PLOT_P1:
S_PLOT_P0;
                                S_PLOT_P1: next_state = done[2] ? S_PLOT_P2:
S_PLOT_P1;
                                S_PLOT_P2: next_state = done[3] ? S_PLOT_P3:
S_PLOT_P2;
                                S_PLOT_P3: next_state = done[4] ? S_PLOT_P4:
S_PLOT_P3;
                                S_PLOT_P4: next_state = done[5] ? S_PLOT_P5:
S_PLOT_P4;
                                S_PLOT_P5: next_state = done[6] ? S_PLOT_P6:
S_PLOT_P5;
                                S_PLOT_P6: next_state = done[7] ? S_PLOT_P7:
S_PLOT_P6;
                                S_PLOT_P7: next_state = done[8] ? S_PLOT_P8:
S_PLOT_P7;
                                S_PLOT_P8: next_state = done[9] ? S_PLOT_BLITZ_ARM:
S_PLOT_P8;
                                S_PLOT_BLITZ_ARM: next_state = done[12] ?
S_PLOT_BLITZ_HOOK: S_PLOT_BLITZ_ARM;
                                S_PLOT_BLITZ_HOOK: next_state = done[11] ?
S_PLOT_BLITZ: S_PLOT_BLITZ_HOOK;
                                S_PLOT_BLITZ: next_state = done[0] ? S_PLOT_HP:
S_PLOT_BLITZ;
                                S_PLOT_HP: next_state = done[14] ? S_PLOT_WAIT:
S_PLOT_HP;
        default:    next_state = S_PLOT_WAIT;
                endcase
    end // state_table

        always@(*)
    begin
        plot = 15'b0;
        restart = 1'b0;
                        case (current_state)
                                S_PLOT_BG:          plot[10]=1'b1;
                                S_PLOT_BLITZ:       plot[0]=1'b1;
                                S_PLOT_P0:          plot[1]=1'b1;
                                S_PLOT_P1:          plot[2]=1'b1;
```

```
                                S_PLOT_P2:          plot[3]=1'b1;
                                S_PLOT_P3:          plot[4]=1'b1;
                                S_PLOT_P4:          plot[5]=1'b1;
                                S_PLOT_P5:          plot[6]=1'b1;
                                S_PLOT_P6:          plot[7]=1'b1;
                                S_PLOT_P7:          plot[8]=1'b1;
                                S_PLOT_P8:          plot[9]=1'b1;
                                S_PLOT_BLITZ_HOOK: plot[11]=1'b1;
                                S_PLOT_BLITZ_ARM:  plot[12]=1'b1;
                                S_PLOT_HP:          plot[14]=1'b1;
                                S_RESTART:          restart = 1'b1;
                                S_PLOT_TRAILER: begin
                                        plot[13]=1'b1;
                                end
                                default: plot = 15'b0;
                endcase
    end

        // current_state registers
    always@(posedge clk)
    begin
                if(SW[0] == 0)
                   current_state <= S_PLOT_TRAILER;
       else
          current_state <= next_state;
    end
        // *mux
        always@(posedge clk)
    begin
                        case (current_state)
                                S_PLOT_BG:          begin x=plot_x_bg; y=plot_y_bg;
colour=colour_bg; writeEn_vga=writeEn[10];          end
                                S_PLOT_BLITZ:       begin x=plot_x_b;  y=plot_y_b;
colour=colour_b;  writeEn_vga=writeEn[0];           end
                                S_PLOT_BLITZ_HOOK:  begin x=plot_x_bh; y=plot_y_bh;
colour=colour_bh; writeEn_vga=writeEn[11];          end
                                S_PLOT_BLITZ_ARM:   begin x=plot_x_ba; y=plot_y_ba;
colour=colour_ba; writeEn_vga=writeEn[12];          end
                                S_PLOT_P0:          begin x=plot_x_p0; y=plot_y_p0;
colour=colour_p0; writeEn_vga=writeEn[1];           end
                                S_PLOT_P1:          begin x=plot_x_p1; y=plot_y_p1;
colour=colour_p1; writeEn_vga=writeEn[2];           end
                                S_PLOT_P2:          begin x=plot_x_p2; y=plot_y_p2;
colour=colour_p2; writeEn_vga=writeEn[3];           end
                                S_PLOT_P3:          begin x=plot_x_p3; y=plot_y_p3;
colour=colour_p3; writeEn_vga=writeEn[4];           end
```

```
                                S_PLOT_P4:          begin x=plot_x_p4; y=plot_y_p4;
colour=colour_p4; writeEn_vga=writeEn[5];          end
                                S_PLOT_P5:          begin x=plot_x_p5; y=plot_y_p5;
colour=colour_p5; writeEn_vga=writeEn[6];          end
                                S_PLOT_P6:          begin x=plot_x_p6; y=plot_y_p6;
colour=colour_p6; writeEn_vga=writeEn[7];          end
                                S_PLOT_P7:          begin x=plot_x_p7; y=plot_y_p7;
colour=colour_p7; writeEn_vga=writeEn[8];          end
                                S_PLOT_P8:          begin x=plot_x_p8; y=plot_y_p8;
colour=colour_p8; writeEn_vga=writeEn[9];          end
                                S_PLOT_HP:          begin x=plot_x_hp; y=plot_y_hp;
colour=colour_hp; writeEn_vga=writeEn[14];         end
                                S_PLOT_TRAILER:     begin x=plot_x_tl; y=plot_y_tl;
colour=colour_tl; writeEn_vga=writeEn[13];         end
                endcase
    end

        reg [7:0] score_counter;

        wire score;
        assign score = |score_p;

        always@(posedge score, negedge resetn)begin
                if(!resetn)begin
                        score_counter <= 8'b0;
                end
                else if(score)
                        score_counter <= score_counter + 1'b1;
        end


        hex_decoder H0(
      .hex_digit(score_counter[3:0]),
      .segments(HEX0)
      );
    hex_decoder H1(
      .hex_digit(score_counter[7:4]),
      .segments(HEX1)
      );
    hex_decoder H2(
      .hex_digit(4'b0),
      .segments(HEX2)
      );
    hex_decoder H3(
      .hex_digit(4'b0),
      .segments(HEX3)
```

```
	);

		wire swap;
		// Create an Instance of a VGA controller - there can be only one!
		// Define the number of colours as well as the initial background
		// image file (.MIF) for the controller.
		vga_adapter_2buffers VGA(
					.frame(frame),
					.swap(swap),
					.resetn(KEY[0]),
					.clock(CLOCK_50),
					.colour(colour),
					.x(x),
					.y(y),
					.plot(writeEn_vga),
					 //Signals for the DAC to drive the monitor.
					.VGA_R(VGA_R),
					.VGA_G(VGA_G),
					.VGA_B(VGA_B),
					.VGA_HS(VGA_HS),
					.VGA_VS(VGA_VS),
					.VGA_BLANK(VGA_BLANK_N),
					.VGA_SYNC(VGA_SYNC_N),
					.VGA_CLK(VGA_CLK));
			defparam VGA.RESOLUTION = "320x240";
			defparam VGA.MONOCHROME = "FALSE";
			defparam VGA.BITS_PER_COLOUR_CHANNEL = 1;
			defparam VGA.BACKGROUND_IMAGE = "background.mif";

		// Put your code here. Your code should produce signals x,y,colour and writeEn
		// for the VGA controller, in addition to any other functionality your design may
require.


endmodule


//====================================Delay
counter=====================================
module DelayCounter(clock, reset_n, maxcount, enable);
		input clock, reset_n;
		input [27:0] maxcount;
		output enable;
		reg [27:0] count;

		assign enable = (count == 0) ? (1'b1) : (1'b0);
```

```
	always @ (posedge clock, negedge reset_n)
	begin
		if (!reset_n)
			count <= 0;
		else if (count == (maxcount-28'b1))
			count <= 0;
		else
			count <= count + 1'b1;
	end

endmodule

module hex_decoder(hex_digit, segments);
	input [3:0] hex_digit;
   output reg [6:0] segments;

   always @(*)
      case (hex_digit)
         4'h0: segments = 7'b100_0000;
         4'h1: segments = 7'b111_1001;
         4'h2: segments = 7'b010_0100;
         4'h3: segments = 7'b011_0000;
         4'h4: segments = 7'b001_1001;
         4'h5: segments = 7'b001_0010;
         4'h6: segments = 7'b000_0010;
         4'h7: segments = 7'b111_1000;
         4'h8: segments = 7'b000_0000;
         4'h9: segments = 7'b001_1000;
         4'hA: segments = 7'b000_1000;
         4'hB: segments = 7'b000_0011;
         4'hC: segments = 7'b100_0110;
         4'hD: segments = 7'b010_0001;
         4'hE: segments = 7'b000_0110;
         4'hF: segments = 7'b000_1110;
         default: segments = 7'h7f;
      endcase
endmodule

//================================blitzcrank.v=========================
module blitz_pos (
	input go_up,
	input go_down,
	input clk,
	input frame,
	input resetn,
```

```
        input grab,
        input grab_success,
        output reg [8:0] blitz_hook_x,
        output reg [7:0] blitz_hook_y,
        output reg [7:0] y
        // output reg writeEn
        );

        localparam S_SATIONARY =3'd0,
                           S_MOVE_UP   =3'd1,
                           S_MOVE_DOWN =3'd2,
                           S_HOOK_EXTENTION  = 3'd3,
                           S_HOOK_RETRACTION = 3'd4;

        localparam MAX_EXTENSION = 7'd94,
                           MIN_EXTENSION = 7'd42,
                           grabV = 9'd2;

        reg [2:0] current_state, next_state;
        reg up_done, down_done;

        // next state logic
        always @(*) begin
                case(current_state)
                        S_SATIONARY: next_state = (go_up&&(y>8'd48)) ? S_MOVE_UP :
((go_down&&(y<8'd138)) ? S_MOVE_DOWN : (grab ? S_HOOK_EXTENTION :
S_SATIONARY));
                        S_MOVE_UP: next_state = up_done ? S_SATIONARY :
S_MOVE_UP;
                        S_MOVE_DOWN : next_state = down_done ? S_SATIONARY :
S_MOVE_DOWN;
                        S_HOOK_EXTENTION : next_state = (grab_success||(blitz_hook_x
>= MAX_EXTENSION)) ? S_HOOK_RETRACTION : S_HOOK_EXTENTION;
                        S_HOOK_RETRACTION : next_state = (blitz_hook_x <=
MIN_EXTENSION) ? S_SATIONARY : S_HOOK_RETRACTION;
                        default: next_state = S_SATIONARY;
                endcase
        end


        //enable signals
        always@ (posedge clk) begin
                if (!resetn) begin
                        y <= 8'd120 - 8'b1;
                        up_done <= 0;
                        down_done <= 0;
```

```
                blitz_hook_x <= 9'd42;
                blitz_hook_y <= 0; //8'd130 - 8'b1;
            end
            else begin
                case(current_state)
                    S_SATIONARY: begin
                        up_done <= 0;
                        down_done <= 0;
                        blitz_hook_x <= 9'd42;
                        blitz_hook_y <= 0;
                    end
                    S_MOVE_UP: if(frame) begin
                        y <= y - 8'd2;
                        up_done <= 1;
                    end
                    S_MOVE_DOWN: if(frame) begin
                        y <= y + 8'd2;
                        down_done <= 1;
                    end
                    S_HOOK_EXTENTION: if(frame) begin
                        blitz_hook_x <= blitz_hook_x + grabV;
                        blitz_hook_y <= y + 8'd22;
                    end
                    S_HOOK_RETRACTION: if(frame) begin
                        blitz_hook_x <= blitz_hook_x - grabV - grabV;
                        blitz_hook_y <= y + 8'd22;
                    end
                endcase

            end

        end

        // current_state registers
    always@(posedge clk)
    begin
        if(!resetn)
            current_state <= S_SATIONARY;
        else
            current_state <= next_state;
    end

endmodule

//================================poro.v============================
module poro_pos (
```

```
        input clk,
        input frame,
        input resetn,
        input grab_success,
        input hold,
        output reg score,
        output reg [8:0] x
        );

        localparam      grabV = 4'd8, //4 pixels per frame
                                poroV = 4'd1; //1 pixels per frame

        reg grab_count;

        reg [3:0] velocity;
        wire respawn;

   always@(posedge clk)
   begin
        if(grab_success)begin
                velocity = grabV;
                grab_count = 1'b1;
        end
        if(frame) begin
     if((!resetn)||(x<9'd43)||hold)begin
        x = 9'd319;
                        grab_count = 1'b0;
                        velocity = poroV;
                        end

                x = x - velocity;
                end
        end


        always@(posedge clk)
        begin
                if(!resetn || (!grab_count))
                score = 1'b0;
                else if(grab_count)
                begin
                        score = 1'b1;
                end
        end

endmodule
```

```
//================================grab_check.v=============================
=
module grab_check(
	input grab,
	input [8:0] blitz_hook_x,
	input [7:0] blitz_hook_y,
	input [8:0] poro_x,
	input [7:0] poro_y,
	output grab_success
	);

	wire x_match, y_match;

	assign y_match = (poro_y >= blitz_hook_y - 9'd14) && (poro_y <= blitz_hook_y +
9'd10);
	assign x_match = poro_x == blitz_hook_x;

	assign grab_success = x_match && y_match;
endmodule

//================================VGA============================
/* VGA Adapter
 * ---------------
 *
 * This is an implementation of a VGA Adapter. The adapter uses VGA mode signalling to
initiate
 * a 640x480 resolution mode on a computer monitor, with a refresh rate of approximately
60Hz.
 * It is designed for easy use in an early digital logic design course to facilitate student
 * projects on the Altera DE2 Educational board.
 *
 * This implementation of the VGA adapter can display images of varying colour depth at a
resolution of
 * 320x240 or 160x120 superpixels. The concept of superpixels is introduced to reduce the
amount of on-chip
 * memory used by the adapter. The following table shows the number of bits of on-chip
memory used by
 * the adapter in various resolutions and colour depths.
 *
 *
-------------------------------------------------------------------------------------------------------------
----
 * Resolution | Mono    | 8 colours | 64 colours | 512 colours | 4096 colours | 32768 colours |
262144 colours | 2097152 colours |
```

```
 *
-----------------------------------------------------------------------------------------------------------------------
 * 160x120    |   19200 |     57600 |     115200 |      172800 |       230400 |        288000 |
345600 |          403200 |
 * 320x240    |   78600 |    230400 | ############### Does not fit
################################################################## |
 *
-----------------------------------------------------------------------------------------------------------------------
 *
 * By default the adapter works at the resolution of 320x240 with 8 colours. To set the
adapter in any of
 * the other modes, the adapter must be instantiated with specific parameters. These
parameters are:
 * - RESOLUTION - a string that should be either "320x240" or "160x120".
 * - MONOCHROME - a string that should be "TRUE" if you only want black and white
colours, and "FALSE"
 *              otherwise.
 * - BITS_PER_COLOUR_CHANNEL  - an integer specifying how many bits are available to
describe each colour
 *                      (R,G,B). A default value of 1 indicates that 1 bit will be used for red
 *                      channel, 1 for green channel and 1 for blue channel. This allows 8 colours
 *                      to be used.
 *
 * In addition to the above parameters, a BACKGROUND_IMAGE parameter can be
specified. The parameter
 * refers to a memory initilization file (MIF) which contains the initial contents of video
memory.
 * By specifying the initial contents of the memory we can force the adapter to initially display
an
 * image of our choice. Please note that the image described by the BACKGROUND_IMAGE
file will only
 * be valid right after your program the DE2 board. If your circuit draws a single pixel on the
screen,
 * the video memory will be altered and screen contents will be changed. In order to restore
the background
 * image your circuti will have to redraw the background image pixel by pixel, or you will have
to
 * reprogram the DE2 board, thus allowing the video memory to be rewritten.
 *
 * To use the module connect the vga_adapter to your circuit. Your circuit should produce a
value for
 * inputs X, Y and plot. When plot is high, at the next positive edge of the input clock the
vga_adapter
```

```
 * will change the contents of the video memory for the pixel at location (X,Y). At the next redraw
 * cycle the VGA controller will update the contants of the screen by reading the video memory and copying
 * it over to the screen. Since the monitor screen has no memory, the VGA controller has to copy the
 * contents of the video memory to the screen once every 60th of a second to keep the image stable. Thus,
 * the video memory should not be used for other purposes as it may interfere with the operation of the
 * VGA Adapter.
 *
 * As a final note, ensure that the following conditions are met when using this module:
 * 1. You are implementing the the VGA Adapter on the Altera DE2 board. Using another board may change
 *    the amount of memory you can use, the clock generation mechanism, as well as pin assignments required
 *    to properly drive the VGA digital-to-analog converter.
 * 2. Outputs VGA_* should exist in your top level design. They should be assigned pin locations on the
 *    Altera DE2 board as specified by the DE2_pin_assignments.csv file.
 * 3. The input clock must have a frequency of 50 MHz with a 50% duty cycle. On the Altera DE2 board
 *    PIN_N2 is the source for the 50MHz clock.
 *
 * During compilation with Quartus II you may receive the following warnings:
 * - Warning: Variable or input pin "clocken1" is defined but never used
 * - Warning: Pin "VGA_SYNC" stuck at VCC
 * - Warning: Found xx output pins without output pin load capacitance assignment
 * These warnings can be ignored. The first warning is generated, because the software generated
 * memory module contains an input called "clocken1" and it does not drive logic. The second warning
 * indicates that the VGA_SYNC signal is always high. This is intentional. The final warning is
 * generated for the purposes of power analysis. It will persist unless the output pins are assigned
 * output capacitance. Leaving the capacitance values at 0 pf did not affect the operation of the module.
 *
 * If you see any other warnings relating to the vga_adapter, be sure to examine them carefully. They may
 * cause your circuit to malfunction.
 *
 * NOTES/REVISIONS:
```

```
 * July 10, 2007 - Modified the original version of the VGA Adapter written by Sam Vafaee in 2006. The module
 *                 now supports 2 different resolutions as well as uses half the memory compared to prior
 *                 implementation. Also, all settings for the module can be specified from the point
 *                 of instantiation, rather than by modifying the source code. (Tomasz S. Czajkowski)
 */

module vga_adapter_2buffers(
                            frame,
                            swap,
                            resetn,
                            clock,
                            colour,
                            x, y, plot,
                            /* Signals for the DAC to drive the monitor. */
                            VGA_R,
                            VGA_G,
                            VGA_B,
                            VGA_HS,
                            VGA_VS,
                            VGA_BLANK,
                            VGA_SYNC,
                            VGA_CLK);

        parameter BITS_PER_COLOUR_CHANNEL = 1;
        /* The number of bits per colour channel used to represent the colour of each pixel. A value
         * of 1 means that Red, Green and Blue colour channels will use 1 bit each to represent the intensity
         * of the respective colour channel. For BITS_PER_COLOUR_CHANNEL=1, the adapter can display 8 colours.
         * In general, the adapter is able to use 2^(3*BITS_PER_COLOUR_CHANNEL ) colours. The number of colours is
         * limited by the screen resolution and the amount of on-chip memory available on the target device.
         */

        parameter MONOCHROME = "FALSE";
        /* Set this parameter to "TRUE" if you only wish to use black and white colours. Doing so will reduce
         * the amount of memory you will use by a factor of 3. */

        parameter RESOLUTION = "320x240";
```

```
        /* Set this parameter to "160x120" or "320x240". It will cause the VGA adapter to
draw each dot on
         * the screen by using a block of 4x4 pixels ("160x120" resolution) or 2x2 pixels
("320x240" resolution).
         * It effectively reduces the screen resolution to an integer fraction of 640x480. It was
necessary
         * to reduce the resolution for the Video Memory to fit within the on-chip memory
limits.
         */

        parameter BACKGROUND_IMAGE = "background.mif";
        /* The initial screen displayed when the circuit is first programmed onto the DE2
board can be
         * defined useing an MIF file. The file contains the initial colour for each pixel on the
screen
         * and is placed in the Video Memory (VideoMemory module) upon programming.
Note that resetting the
         * VGA Adapter will not cause the Video Memory to revert to the specified image. */


        /****************************************************************************/
        /* Declare inputs and outputs.                                              */
        /****************************************************************************/
        output reg swap;
        input resetn;
        input clock;
        input frame;

        /* The colour input can be either 1 bit or 3*BITS_PER_COLOUR_CHANNEL bits
wide, depending on
         * the setting of the MONOCHROME parameter.
         */
        input [((MONOCHROME == "TRUE") ? (0) :
(BITS_PER_COLOUR_CHANNEL*3-1)):0] colour;

        /* Specify the number of bits required to represent an (X,Y) coordinate on the screen
for
         * a given resolution.
         */
        input [((RESOLUTION == "320x240") ? (8) : (7)):0] x;
        input [((RESOLUTION == "320x240") ? (7) : (6)):0] y;

        /* When plot is high then at the next positive edge of the clock the pixel at (x,y) will
change to
         * a new colour, defined by the value of the colour input.
         */
```

```
        input plot;

        /* These outputs drive the VGA display. The VGA_CLK is also used to clock the FSM
responsible for
         * controlling the data transferred to the DAC driving the monitor. */
        output [9:0] VGA_R;
        output [9:0] VGA_G;
        output [9:0] VGA_B;
        output VGA_HS;
        output VGA_VS;
        output VGA_BLANK;
        output VGA_SYNC;
        output VGA_CLK;

        /****************************************************************************/
        /* Declare local signals here.                                              */
        /****************************************************************************/

        wire valid_160x120;
        wire valid_320x240;
        /* Set to 1 if the specified coordinates are in a valid range for a given resolution.*/

        wire writeEn1, writeEn2;
        /* This is a local signal that allows the Video Memory contents to be changed.
         * It depends on the screen resolution, the values of X and Y inputs, as well as
         * the state of the plot signal.
         */

        wire [((MONOCHROME == "TRUE") ? (0) :
(BITS_PER_COLOUR_CHANNEL*3-1)):0] to_ctrl_colour1, to_ctrl_colour2;
        reg  [((MONOCHROME == "TRUE") ? (0) :
(BITS_PER_COLOUR_CHANNEL*3-1)):0] to_ctrl_colour;
        /* Pixel colour read by the VGA controller */

        wire [((RESOLUTION == "320x240") ? (16) : (14)):0] user_to_video_memory_addr;
        /* This bus specifies the address in memory the user must write
         * data to in order for the pixel intended to appear at location (X,Y) to be displayed
         * at the correct location on the screen.
         */

        wire [((RESOLUTION == "320x240") ? (16) : (14)):0]
controller_to_video_memory_addr;
        /* This bus specifies the address in memory the vga controller must read data from
         * in order to determine the colour of a pixel located at coordinate (X,Y) of the screen.
         */
```

```
	wire clock_25;
	/* 25MHz clock generated by dividing the input clock frequency by 2. */

	wire vcc, gnd;

	/*****************************************************************************/
	/* Instances of modules for the VGA adapter.                                 */
	/*****************************************************************************/
	assign vcc = 1'b1;
	assign gnd = 1'b0;

	vga_address_translator user_input_translator(
					.x(x), .y(y),
.mem_address(user_to_video_memory_addr) );
		defparam user_input_translator.RESOLUTION = RESOLUTION;
	/* Convert user coordinates into a memory address. */

	// double buffer
	wire read_done;
	assign read_done = (controller_to_video_memory_addr == 17'd76799);

	reg [4:0] current_state, next_state;
	localparam  S_NORMAL			= 5'd0,
			S_WAIT				= 5'd1,
			S_SWAP				= 5'd2;

	always@(*)
  begin: state_table
	case (current_state)
			S_NORMAL: next_state = frame ? S_WAIT : S_NORMAL;
			S_WAIT:		next_state = read_done ? S_SWAP : S_WAIT;
			S_SWAP:	next_state = S_NORMAL;
	default: next_state = S_NORMAL;
		endcase
  end // state_table

	always@(*)
  begin
	swap = 1'b0;
		case (current_state)
			S_SWAP:		swap=1'b1;
			default: swap = 1'b0;
		endcase
  end

	// current_state registers
```

```
   always@(posedge clock)
   begin
                if(!resetn)
                   current_state <= S_NORMAL;
      else
         current_state <= next_state;
   end

         reg write1_read2;
         always @(posedge clock)begin
         if(swap)begin
         if(!resetn)
                  write1_read2 <= 1'b0;
         else
                  write1_read2 <= !write1_read2;
                  end
         end


         assign valid_160x120 = (({1'b0, x} >= 0) & ({1'b0, x} < 160) & ({1'b0, y} >= 0) & ({1'b0,
y} < 120)) & (RESOLUTION == "160x120");
         assign valid_320x240 = (({1'b0, x} >= 0) & ({1'b0, x} < 320) & ({1'b0, y} >= 0) & ({1'b0,
y} < 240)) & (RESOLUTION == "320x240");
         assign writeEn1 = (plot) & (valid_160x120 | valid_320x240) & (write1_read2);
         assign writeEn2 = (plot) & (valid_160x120 | valid_320x240) & (!write1_read2);
         /* Allow the user to plot a pixel if and only if the (X,Y) coordinates supplied are in a
valid range. */

         /* Create video memory 1. */
         altsyncram      VideoMemory1 (
                                  .wren_a (writeEn1),
                                  .wren_b (gnd),
                                  .clock0 (clock), // write clock
                                  .clock1 (clock_25), // read clock
                                  .clocken0 (vcc), // write enable clock
                                  .clocken1 (vcc), // read enable clock
                                  .address_a (user_to_video_memory_addr),
                                  .address_b (controller_to_video_memory_addr),
                                  .data_a (colour), // data in
                                  .q_b (to_ctrl_colour1) // data out
                                  );
         defparam
                  VideoMemory1.WIDTH_A = ((MONOCHROME == "FALSE") ?
(BITS_PER_COLOUR_CHANNEL*3) : 1),
                  VideoMemory1.WIDTH_B = ((MONOCHROME == "FALSE") ?
(BITS_PER_COLOUR_CHANNEL*3) : 1),
```

```
		VideoMemory1.INTENDED_DEVICE_FAMILY = "Cyclone II",
		VideoMemory1.OPERATION_MODE = "DUAL_PORT",
		VideoMemory1.WIDTHAD_A = ((RESOLUTION == "320x240") ? (17) : (15)),
		VideoMemory1.NUMWORDS_A = ((RESOLUTION == "320x240") ? (76800) :
(19200)),
		VideoMemory1.WIDTHAD_B = ((RESOLUTION == "320x240") ? (17) : (15)),
		VideoMemory1.NUMWORDS_B = ((RESOLUTION == "320x240") ? (76800) :
(19200)),
		VideoMemory1.OUTDATA_REG_B = "CLOCK1",
		VideoMemory1.ADDRESS_REG_B = "CLOCK1",
		VideoMemory1.CLOCK_ENABLE_INPUT_A = "BYPASS",
		VideoMemory1.CLOCK_ENABLE_INPUT_B = "BYPASS",
		VideoMemory1.CLOCK_ENABLE_OUTPUT_B = "BYPASS",
		VideoMemory1.POWER_UP_UNINITIALIZED = "FALSE",
		VideoMemory1.INIT_FILE = BACKGROUND_IMAGE;

	/* Create video memory 2. */
	altsyncram	VideoMemory2 (
				.wren_a (writeEn2),
				.wren_b (gnd),
				.clock0 (clock), // write clock
				.clock1 (clock_25), // read clock
				.clocken0 (vcc), // write enable clock
				.clocken1 (vcc), // read enable clock
				.address_a (user_to_video_memory_addr),
				.address_b (controller_to_video_memory_addr),
				.data_a (colour), // data in
				.q_b (to_ctrl_colour2) // data out
				);
	defparam
		VideoMemory2.WIDTH_A = ((MONOCHROME == "FALSE") ?
(BITS_PER_COLOUR_CHANNEL*3) : 1),
		VideoMemory2.WIDTH_B = ((MONOCHROME == "FALSE") ?
(BITS_PER_COLOUR_CHANNEL*3) : 1),
		VideoMemory2.INTENDED_DEVICE_FAMILY = "Cyclone II",
		VideoMemory2.OPERATION_MODE = "DUAL_PORT",
		VideoMemory2.WIDTHAD_A = ((RESOLUTION == "320x240") ? (17) : (15)),
		VideoMemory2.NUMWORDS_A = ((RESOLUTION == "320x240") ? (76800) :
(19200)),
		VideoMemory2.WIDTHAD_B = ((RESOLUTION == "320x240") ? (17) : (15)),
		VideoMemory2.NUMWORDS_B = ((RESOLUTION == "320x240") ? (76800) :
(19200)),
		VideoMemory2.OUTDATA_REG_B = "CLOCK1",
		VideoMemory2.ADDRESS_REG_B = "CLOCK1",
		VideoMemory2.CLOCK_ENABLE_INPUT_A = "BYPASS",
		VideoMemory2.CLOCK_ENABLE_INPUT_B = "BYPASS",
```

```
		VideoMemory2.CLOCK_ENABLE_OUTPUT_B = "BYPASS",
		VideoMemory2.POWER_UP_UNINITIALIZED = "FALSE",
		VideoMemory2.INIT_FILE = BACKGROUND_IMAGE;

	always @(posedge clock)begin
		if(write1_read2 == 1'b1)begin
			to_ctrl_colour <= to_ctrl_colour2;
		end
		else begin
			to_ctrl_colour <= to_ctrl_colour1;
		end
	end

	vga_pll mypll(clock, clock_25);
	/* This module generates a clock with half the frequency of the input clock.
	 * For the VGA adapter to operate correctly the clock signal 'clock' must be
	 * a 50MHz clock. The derived clock, which will then operate at 25MHz, is
	 * required to set the monitor into the 640x480@60Hz display mode (also known as
	 * the VGA mode).
	 */

	vga_controller controller(
			.vga_clock(clock_25),
			.resetn(resetn),
			.pixel_colour(to_ctrl_colour),
			.memory_address(controller_to_video_memory_addr),
			.VGA_R(VGA_R),
			.VGA_G(VGA_G),
			.VGA_B(VGA_B),
			.VGA_HS(VGA_HS),
			.VGA_VS(VGA_VS),
			.VGA_BLANK(VGA_BLANK),
			.VGA_SYNC(VGA_SYNC),
			.VGA_CLK(VGA_CLK)
		);
		defparam controller.BITS_PER_COLOUR_CHANNEL  =
BITS_PER_COLOUR_CHANNEL ;
		defparam controller.MONOCHROME = MONOCHROME;
		defparam controller.RESOLUTION = RESOLUTION;

endmodule

/* This module implements the VGA controller. It assumes a 25MHz clock is supplied as
input.
 *
 * General approach:
```

```
 * Go through each line of the screen and read the colour each pixel on that line should have
from
 * the Video memory. To do that for each (x,y) pixel on the screen convert (x,y) coordinate to
 * a memory_address at which the pixel colour is stored in Video memory. Once the pixel
colour is
 * read from video memory its brightness is first increased before it is forwarded to the VGA
DAC.
 */
module vga_controller(	vga_clock, resetn, pixel_colour, memory_address,
			VGA_R, VGA_G, VGA_B,
			VGA_HS, VGA_VS, VGA_BLANK,
			VGA_SYNC, VGA_CLK);

	/* Screen resolution and colour depth parameters. */

	parameter BITS_PER_COLOUR_CHANNEL = 1;
	/* The number of bits per colour channel used to represent the colour of each pixel. A
value
	 * of 1 means that Red, Green and Blue colour channels will use 1 bit each to
represent the intensity
	 * of the respective colour channel. For BITS_PER_COLOUR_CHANNEL=1, the
adapter can display 8 colours.
	 * In general, the adapter is able to use 2^(3*BITS_PER_COLOUR_CHANNEL)
colours. The number of colours is
	 * limited by the screen resolution and the amount of on-chip memory available on
the target device.
	 */

	parameter MONOCHROME = "FALSE";
	/* Set this parameter to "TRUE" if you only wish to use black and white colours.
Doing so will reduce
	 * the amount of memory you will use by a factor of 3. */

	parameter RESOLUTION = "320x240";
	/* Set this parameter to "160x120" or "320x240". It will cause the VGA adapter to
draw each dot on
	 * the screen by using a block of 4x4 pixels ("160x120" resolution) or 2x2 pixels
("320x240" resolution).
	 * It effectively reduces the screen resolution to an integer fraction of 640x480. It was
necessary
	 * to reduce the resolution for the Video Memory to fit within the on-chip memory
limits.
	 */

	//--- Timing parameters.
	/* Recall that the VGA specification requires a few more rows and columns are drawn
```

```
	 * when refreshing the screen than are actually present on the screen. This is
necessary to
	 * generate the vertical and the horizontal syncronization signals. If you wish to use a
	 * display mode other than 640x480 you will need to modify the parameters below as
well
	 * as change the frequency of the clock driving the monitor (VGA_CLK).
	 */
	parameter C_VERT_NUM_PIXELS  = 10'd480;
	parameter C_VERT_SYNC_START  = 10'd493;
	parameter C_VERT_SYNC_END    = 10'd494; //(C_VERT_SYNC_START + 2 - 1);
	parameter C_VERT_TOTAL_COUNT = 10'd525;

	parameter C_HORZ_NUM_PIXELS  = 10'd640;
	parameter C_HORZ_SYNC_START  = 10'd659;
	parameter C_HORZ_SYNC_END    = 10'd754; //(C_HORZ_SYNC_START + 96 - 1);
	parameter C_HORZ_TOTAL_COUNT = 10'd800;

	/*****************************************************************************/
	/* Declare inputs and outputs.                                               */
	/*****************************************************************************/

	input vga_clock, resetn;
	input [((MONOCHROME == "TRUE") ? (0) :
(BITS_PER_COLOUR_CHANNEL*3-1)):0] pixel_colour;
	output [((RESOLUTION == "320x240") ? (16) : (14)):0] memory_address;
	output reg [9:0] VGA_R;
	output reg [9:0] VGA_G;
	output reg [9:0] VGA_B;
	output reg VGA_HS;
	output reg VGA_VS;
	output reg VGA_BLANK;
	output VGA_SYNC, VGA_CLK;

	/*****************************************************************************/
	/* Local Signals.                                                            */
	/*****************************************************************************/

	reg VGA_HS1;
	reg VGA_VS1;
	reg VGA_BLANK1;
	reg [9:0] xCounter, yCounter;
	wire xCounter_clear;
	wire yCounter_clear;
	wire vcc;

	reg [((RESOLUTION == "320x240") ? (8) : (7)):0] x;
```

```
reg [((RESOLUTION == "320x240") ? (7) : (6)):0] y;
/* Inputs to the converter. */

/*****************************************************************************/
/* Controller implementation.                                                */
/*****************************************************************************/

assign vcc =1'b1;

/* A counter to scan through a horizontal line. */
always @(posedge vga_clock or negedge resetn)
begin
	if (!resetn)
		xCounter <= 10'd0;
	else if (xCounter_clear)
		xCounter <= 10'd0;
	else
	begin
		xCounter <= xCounter + 1'b1;
	end
end
assign xCounter_clear = (xCounter == (C_HORZ_TOTAL_COUNT-1));

/* A counter to scan vertically, indicating the row currently being drawn. */
always @(posedge vga_clock or negedge resetn)
begin
	if (!resetn)
		yCounter <= 10'd0;
	else if (xCounter_clear && yCounter_clear)
		yCounter <= 10'd0;
	else if (xCounter_clear)		//Increment when x counter resets
		yCounter <= yCounter + 1'b1;
end
assign yCounter_clear = (yCounter == (C_VERT_TOTAL_COUNT-1));

/* Convert the xCounter/yCounter location from screen pixels (640x480) to our
 * local dots (320x240 or 160x120). Here we effectively divide x/y coordinate by 2 or
4,
 * depending on the resolution. */
always @(*)
begin
	if (RESOLUTION == "320x240")
	begin
		x = xCounter[9:1];
		y = yCounter[8:1];
	end
```

```
                else
                begin
                        x = xCounter[9:2];
                        y = yCounter[8:2];
                end
        end

        /* Change the (x,y) coordinate into a memory address. */
        vga_address_translator controller_translator(
                                        .x(x), .y(y), .mem_address(memory_address) );
                defparam controller_translator.RESOLUTION = RESOLUTION;


        /* Generate the vertical and horizontal synchronization pulses. */
        always @(posedge vga_clock)
        begin
                //- Sync Generator (ACTIVE LOW)
                VGA_HS1 <= ~((xCounter >= C_HORZ_SYNC_START) && (xCounter <=
C_HORZ_SYNC_END));
                VGA_VS1 <= ~((yCounter >= C_VERT_SYNC_START) && (yCounter <=
C_VERT_SYNC_END));

                //- Current X and Y is valid pixel range
                VGA_BLANK1 <= ((xCounter < C_HORZ_NUM_PIXELS) && (yCounter <
C_VERT_NUM_PIXELS));

                //- Add 1 cycle delay
                VGA_HS <= VGA_HS1;
                VGA_VS <= VGA_VS1;
                VGA_BLANK <= VGA_BLANK1;
        end

        /* VGA sync should be 1 at all times. */
        assign VGA_SYNC = vcc;

        /* Generate the VGA clock signal. */
        assign VGA_CLK = vga_clock;

        /* Brighten the colour output. */
        // The colour input is first processed to brighten the image a little. Setting the top
        // bits to correspond to the R,G,B colour makes the image a bit dull. To brighten the
image,
        // each bit of the colour is replicated through the 10 DAC colour input bits. For
example,
        // when BITS_PER_COLOUR_CHANNEL is 2 and the red component is set to 2'b10,
then the
```

```
	// VGA_R input to the DAC will be set to 10'b1010101010.

	integer index;
	integer sub_index;

	always @(pixel_colour)
	begin
		VGA_R <= 'b0;
		VGA_G <= 'b0;
		VGA_B <= 'b0;
		if (MONOCHROME == "FALSE")
		begin
			for (index = 10-BITS_PER_COLOUR_CHANNEL; index >= 0; index =
index - BITS_PER_COLOUR_CHANNEL)
			begin
				for (sub_index = BITS_PER_COLOUR_CHANNEL - 1;
sub_index >= 0; sub_index = sub_index - 1)
				begin
					VGA_R[sub_index+index] <= pixel_colour[sub_index +
BITS_PER_COLOUR_CHANNEL*2];
					VGA_G[sub_index+index] <= pixel_colour[sub_index +
BITS_PER_COLOUR_CHANNEL];
					VGA_B[sub_index+index] <= pixel_colour[sub_index];
				end
			end
		end
		else
		begin
			for (index = 0; index < 10; index = index + 1)
			begin
				VGA_R[index] <= pixel_colour[0:0];
				VGA_G[index] <= pixel_colour[0:0];
				VGA_B[index] <= pixel_colour[0:0];
			end
		end
	end

endmodule

// megafunction wizard: %ALTPLL%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altpll

// ============================================================
// File Name: VgaPll.v
```

```
// Megafunction Name(s):
//                          altpll
// ================================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 5.0 Build 168 06/22/2005 SP 1 SJ Full Version
// ************************************************************


//Copyright (C) 1991-2005 Altera Corporation
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, Altera MegaCore Function License
//Agreement, or other applicable license agreement, including,
//without limitation, that your use is for the sole purpose of
//programming logic devices manufactured by Altera and sold by
//Altera or its authorized distributors.  Please refer to the
//applicable agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module vga_pll (
        clock_in,
        clock_out);

        input     clock_in;
        output    clock_out;

        wire [5:0] clock_output_bus;
        wire [1:0] clock_input_bus;
        wire gnd;

        assign gnd = 1'b0;
        assign clock_input_bus = { gnd, clock_in };

        altpll    altpll_component (
                                .inclk (clock_input_bus),
                                .clk (clock_output_bus)
                                );
```

```
	defparam
		altpll_component.operation_mode = "NORMAL",
		altpll_component.intended_device_family = "Cyclone II",
		altpll_component.lpm_type = "altpll",
		altpll_component.pll_type = "FAST",
		/* Specify the input clock to be a 50MHz clock. A 50 MHz clock is present
		 * on PIN_N2 on the DE2 board. We need to specify the input clock frequency
		 * in order to set up the PLL correctly. To do this we must put the input clock
		 * period measured in picoseconds in the inclk0_input_frequency parameter.
		 * 1/(20000 ps) = 0.5 * 10^(5) Hz = 50 * 10^(6) Hz = 50 MHz. */
		altpll_component.inclk0_input_frequency = 20000,
		altpll_component.primary_clock = "INCLK0",
		/* Specify output clock parameters. The output clock should have a
		 * frequency of 25 MHz, with 50% duty cycle. */
		altpll_component.compensate_clock = "CLK0",
		altpll_component.clk0_phase_shift = "0",
		altpll_component.clk0_divide_by = 2,
		altpll_component.clk0_multiply_by = 1,
		altpll_component.clk0_duty_cycle = 50;

	assign clock_out = clock_output_bus[0];

endmodule

// =============================================================
// CNX file retrieval info
// =============================================================
// Retrieval info: PRIVATE: MIRROR_CLK0 STRING "0"
// Retrieval info: PRIVATE: PHASE_SHIFT_UNIT0 STRING "deg"
// Retrieval info: PRIVATE: OUTPUT_FREQ_UNIT0 STRING "MHz"
// Retrieval info: PRIVATE: INCLK1_FREQ_UNIT_COMBO STRING "MHz"
// Retrieval info: PRIVATE: SPREAD_USE STRING "0"
// Retrieval info: PRIVATE: SPREAD_FEATURE_ENABLED STRING "0"
// Retrieval info: PRIVATE: GLOCKED_COUNTER_EDIT_CHANGED STRING "1"
// Retrieval info: PRIVATE: GLOCK_COUNTER_EDIT NUMERIC "1048575"
// Retrieval info: PRIVATE: SRC_SYNCH_COMP_RADIO STRING "0"
// Retrieval info: PRIVATE: DUTY_CYCLE0 STRING "50.00000000"
// Retrieval info: PRIVATE: PHASE_SHIFT0 STRING "0.00000000"
// Retrieval info: PRIVATE: MULT_FACTOR0 NUMERIC "1"
// Retrieval info: PRIVATE: OUTPUT_FREQ_MODE0 STRING "1"
// Retrieval info: PRIVATE: SPREAD_PERCENT STRING "0.500"
// Retrieval info: PRIVATE: LOCKED_OUTPUT_CHECK STRING "0"
// Retrieval info: PRIVATE: PLL_ARESET_CHECK STRING "0"
// Retrieval info: PRIVATE: STICKY_CLK0 STRING "1"
// Retrieval info: PRIVATE: BANDWIDTH STRING "1.000"
// Retrieval info: PRIVATE: BANDWIDTH_USE_CUSTOM STRING "0"
```

// Retrieval info: PRIVATE: DEVICE_SPEED_GRADE STRING "Any"
// Retrieval info: PRIVATE: SPREAD_FREQ STRING "50.000"
// Retrieval info: PRIVATE: BANDWIDTH_FEATURE_ENABLED STRING "0"
// Retrieval info: PRIVATE: LONG_SCAN_RADIO STRING "1"
// Retrieval info: PRIVATE: PLL_ENHPLL_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: LVDS_MODE_DATA_RATE_DIRTY NUMERIC "0"
// Retrieval info: PRIVATE: USE_CLK0 STRING "1"
// Retrieval info: PRIVATE: INCLK1_FREQ_EDIT_CHANGED STRING "1"
// Retrieval info: PRIVATE: SCAN_FEATURE_ENABLED STRING "0"
// Retrieval info: PRIVATE: ZERO_DELAY_RADIO STRING "0"
// Retrieval info: PRIVATE: PLL_PFDENA_CHECK STRING "0"
// Retrieval info: PRIVATE: CREATE_CLKBAD_CHECK STRING "0"
// Retrieval info: PRIVATE: INCLK1_FREQ_EDIT STRING "50.000"
// Retrieval info: PRIVATE: CUR_DEDICATED_CLK STRING "c0"
// Retrieval info: PRIVATE: PLL_FASTPLL_CHECK NUMERIC "0"
// Retrieval info: PRIVATE: ACTIVECLK_CHECK STRING "0"
// Retrieval info: PRIVATE: BANDWIDTH_FREQ_UNIT STRING "MHz"
// Retrieval info: PRIVATE: INCLK0_FREQ_UNIT_COMBO STRING "MHz"
// Retrieval info: PRIVATE: GLOCKED_MODE_CHECK STRING "0"
// Retrieval info: PRIVATE: NORMAL_MODE_RADIO STRING "1"
// Retrieval info: PRIVATE: CUR_FBIN_CLK STRING "e0"
// Retrieval info: PRIVATE: DIV_FACTOR0 NUMERIC "1"
// Retrieval info: PRIVATE: INCLK1_FREQ_UNIT_CHANGED STRING "1"
// Retrieval info: PRIVATE: HAS_MANUAL_SWITCHOVER STRING "1"
// Retrieval info: PRIVATE: EXT_FEEDBACK_RADIO STRING "0"
// Retrieval info: PRIVATE: PLL_AUTOPLL_CHECK NUMERIC "1"
// Retrieval info: PRIVATE: CLKLOSS_CHECK STRING "0"
// Retrieval info: PRIVATE: BANDWIDTH_USE_AUTO STRING "1"
// Retrieval info: PRIVATE: SHORT_SCAN_RADIO STRING "0"
// Retrieval info: PRIVATE: LVDS_MODE_DATA_RATE STRING "Not Available"
// Retrieval info: PRIVATE: CLKSWITCH_CHECK STRING "1"
// Retrieval info: PRIVATE: SPREAD_FREQ_UNIT STRING "KHz"
// Retrieval info: PRIVATE: PLL_ENA_CHECK STRING "0"
// Retrieval info: PRIVATE: INCLK0_FREQ_EDIT STRING "50.000"
// Retrieval info: PRIVATE: CNX_NO_COMPENSATE_RADIO STRING "0"
// Retrieval info: PRIVATE: INT_FEEDBACK__MODE_RADIO STRING "1"
// Retrieval info: PRIVATE: OUTPUT_FREQ0 STRING "25.000"
// Retrieval info: PRIVATE: PRIMARY_CLK_COMBO STRING "inclk0"
// Retrieval info: PRIVATE: CREATE_INCLK1_CHECK STRING "0"
// Retrieval info: PRIVATE: SACN_INPUTS_CHECK STRING "0"
// Retrieval info: PRIVATE: DEV_FAMILY STRING "Cyclone II"
// Retrieval info: PRIVATE: SWITCHOVER_COUNT_EDIT NUMERIC "1"
// Retrieval info: PRIVATE: SWITCHOVER_FEATURE_ENABLED STRING "1"
// Retrieval info: PRIVATE: BANDWIDTH_PRESET STRING "Low"
// Retrieval info: PRIVATE: GLOCKED_FEATURE_ENABLED STRING "1"
// Retrieval info: PRIVATE: USE_CLKENA0 STRING "0"

```
// Retrieval info: PRIVATE: LVDS_PHASE_SHIFT_UNIT0 STRING "deg"
// Retrieval info: PRIVATE: CLKBAD_SWITCHOVER_CHECK STRING "0"
// Retrieval info: PRIVATE: BANDWIDTH_USE_PRESET STRING "0"
// Retrieval info: PRIVATE: PLL_LVDS_PLL_CHECK NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLK0_DUTY_CYCLE NUMERIC "50"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altpll"
// Retrieval info: CONSTANT: CLK0_MULTIPLY_BY NUMERIC "1"
// Retrieval info: CONSTANT: INCLK0_INPUT_FREQUENCY NUMERIC "20000"
// Retrieval info: CONSTANT: CLK0_DIVIDE_BY NUMERIC "2"
// Retrieval info: CONSTANT: PLL_TYPE STRING "FAST"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "NORMAL"
// Retrieval info: CONSTANT: COMPENSATE_CLOCK STRING "CLK0"
// Retrieval info: CONSTANT: CLK0_PHASE_SHIFT STRING "0"
// Retrieval info: USED_PORT: c0 0 0 0 0 OUTPUT VCC "c0"
// Retrieval info: USED_PORT: @clk 0 0 6 0 OUTPUT VCC "@clk[5..0]"
// Retrieval info: USED_PORT: inclk0 0 0 0 0 INPUT GND "inclk0"
// Retrieval info: USED_PORT: @extclk 0 0 4 0 OUTPUT VCC "@extclk[3..0]"
// Retrieval info: CONNECT: @inclk 0 0 1 0 inclk0 0 0 0 0
// Retrieval info: CONNECT: c0 0 0 0 0 @clk 0 0 1 0
// Retrieval info: CONNECT: @inclk 0 0 1 1 GND 0 0 0 0
// Retrieval info: GEN_FILE: TYPE_NORMAL VgaPll.v TRUE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL VgaPll.inc FALSE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL VgaPll.cmp FALSE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL VgaPll.bsf FALSE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL VgaPll_inst.v FALSE FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL VgaPll_bb.v FALSE FALSE

/* This module converts a user specified coordinates into a memory address.
 * The output of the module depends on the resolution set by the user.
 */
module vga_address_translator(x, y, mem_address);

	parameter RESOLUTION = "320x240";
	/* Set this parameter to "160x120" or "320x240". It will cause the VGA adapter to
draw each dot on
	 * the screen by using a block of 4x4 pixels ("160x120" resolution) or 2x2 pixels
("320x240" resolution).
	 * It effectively reduces the screen resolution to an integer fraction of 640x480. It was
necessary
	 * to reduce the resolution for the Video Memory to fit within the on-chip memory
limits.
	 */

	input [((RESOLUTION == "320x240") ? (8) : (7)):0] x;
```

```
	input [((RESOLUTION == "320x240") ? (7) : (6)):0] y;
	output reg [((RESOLUTION == "320x240") ? (16) : (14)):0] mem_address;

	/* The basic formula is address = y*WIDTH + x;
	 * For 320x240 resolution we can write 320 as (256 + 64). Memory address becomes
	 * (y*256) + (y*64) + x;
	 * This simplifies multiplication a simple shift and add operation.
	 * A leading 0 bit is added to each operand to ensure that they are treated as
unsigned
	 * inputs. By default the use a '+' operator will generate a signed adder.
	 * Similarly, for 160x120 resolution we write 160 as 128+32.
	 */
	wire [16:0] res_320x240 = ({1'b0, y, 8'd0} + {1'b0, y, 6'd0} + {1'b0, x});
	wire [15:0] res_160x120 = ({1'b0, y, 7'd0} + {1'b0, y, 5'd0} + {1'b0, x});

	always @(*)
	begin
		if (RESOLUTION == "320x240")
			mem_address = res_320x240;
		else
			mem_address = res_160x120[14:0];
	end
endmodule

//==============================Graphing
Units============================
//gu for background needs a ram with img imf in datapath

module background_gu
	(
		frame,
		clk,
		resetn,
		plot,

		colour_out,
		x_out,
		y_out,
		writeEn,
		done
	);
	input			frame;
	input			clk;
	input			resetn;
	input			plot;
	// Declare your inputs and outputs here
```

```
        output  [2:0]    colour_out;
        output  [8:0]    x_out;
        output  [7:0]    y_out;
        output                  writeEn;
        output                  done;


        wire [8:0] x_increment;
        wire [7:0] y_increment;

        finite_state_machine f1(
                .plot(plot),
                .clk(clk),
                .resetn(resetn),
                .writeEn(writeEn),
                .x_increment(x_increment),
                .y_increment(y_increment),
                .done(done)
        );

        tl_data_path dp1(
        .frame(frame),
        .resetn(resetn),
        .clk(clk),
        .x_increment(x_increment),
        .y_increment(y_increment),

        .x_out(x_out),
        .y_out(y_out),
        .colour_out(colour_out)
        );

endmodule

module finite_state_machine(
                input plot,
                input clk,
                input resetn,
                output reg writeEn,
                output reg [8:0] x_increment,  //320
                output reg [7:0] y_increment,  //240
                output reg done
        );

        reg [4:0] current_state, next_state;
```

```
        reg [8:0] black_x_counter; // count to 160 pixel
        reg [7:0] black_y_counter; // count to 120 pixel

        localparam              S_RESET                         = 5'd0,
                                        S_BLACK         = 5'd1,
                                        S_BLACK_X       = 5'd2,
                                        S_BLACK_Y       = 5'd3,
                                        S_DONE                          = 5'd4;


        always@(*)
    begin: state_table
        case (current_state)
                                S_RESET: next_state = plot ? S_BLACK : S_RESET;
                                S_BLACK: next_state = (black_y_counter <= 8'd240) ?
S_BLACK_X : S_DONE;
                                S_BLACK_X: next_state = (black_x_counter <= 9'd320) ?
S_BLACK : S_BLACK_Y;
                                S_BLACK_Y: next_state = S_BLACK;
                                S_DONE:         next_state = S_RESET;

        default:     next_state = S_RESET;
                endcase
    end // state_table



        always @(*)
    begin: enable_signals
      // By default make all our signals 0
                writeEn = 1'b0;
                x_increment = 9'b0;
                y_increment = 8'b0;
                done = 1'b0;

      case (current_state)
                        S_BLACK: begin
                                x_increment = black_x_counter;
                                y_increment = black_y_counter;
                        end
                        S_BLACK_X : begin writeEn = 1'b1; end
                        S_DONE : begin done = 1'b1; end

      endcase
    end
```

```
  always @(posedge clk)
  begin
    if(!resetn || (current_state == S_BLACK_Y) || (current_state == S_RESET))
              black_x_counter <= 9'b0;
           if (current_state == S_BLACK_X)
              black_x_counter <= black_x_counter + 9'b1;
  end

  always @(posedge clk)
  begin
    if(!resetn || (current_state == S_RESET))
              black_y_counter <= 8'b0;
           if (current_state == S_BLACK_Y)
              black_y_counter <= black_y_counter + 8'b1;
  end

   // current_state registers
   always@(posedge clk)
   begin: state_FFs
      if(!resetn)
         current_state <= S_RESET;
      else
         current_state <= next_state;
   end // state_FFS

endmodule

module tl_data_path(
         input frame,
         input resetn,
         input clk,

         input [8:0] x_increment, // 320*240
         input [7:0] y_increment, // 320*240

         output reg [8:0] x_out,
         output reg [7:0] y_out,
         output [2:0] colour_out
         );

         reg [8:0] bg_offset;

         always@(posedge clk) begin
                  if(frame)begin
                           if(!resetn|(bg_offset==9'd91))
                                    bg_offset <= 9'd47;
```

```
                else
                        bg_offset <= bg_offset + 9'd2;
            end
        end
        wire [17:0] address;
        assign address = bg_offset + x_increment + y_increment * 10'd640;
        background_image bi1(address,clk,3'b0,1'b0,colour_out);


        always@(posedge clk) begin
            if(!resetn) begin
                x_out <= 9'b0;
                y_out <= 8'b0;
                //colour_out <= 3'b0;
            end
            else begin
                        x_out <= x_increment;
                        y_out <= y_increment;
                //      colour_out <= 3'b000; //set in to blk temp
            end
        end
endmodule

// graphing unit for general first, whenever we are gonna use
// we need to create different module for each controller by adding different RAM inside.

module blitz_gu
        (
            clk,
            y_in,
            resetn,
            plot,

            colour_out,
            x_out,
            y_out,
            writeEn,
            done
        );

        input               clk;
        input   [7:0]   y_in;
        input               resetn;
        input               plot;
        // Declare your inputs and outputs here
        output  [2:0]   colour_out;
```

```
        output  [8:0]    x_out;
        output  [7:0]    y_out;
        output                  writeEn;
        output                  done;


        wire [4:0] x_increment;
        wire [5:0] y_increment; //0-63
        wire ld_xy;

        b_finite_state_machine f1(
                .plot(plot),
                .clk(clk),
                .resetn(resetn),
                .writeEn(writeEn),
                .x_increment(x_increment),
                .y_increment(y_increment),
                .ld_xy(ld_xy),
                .done(done)
        );

        b_data_path dp1(
        .x_in(9'd10),
        .y_in(y_in),
        .resetn(resetn),
        .clk(clk),
        .x_increment(x_increment),
        .y_increment(y_increment),
        .ld_xy(ld_xy),

        .x_out(x_out),
        .y_out(y_out),
        .colour_out(colour_out)
        );

endmodule

module b_finite_state_machine(
                input plot,
                input clk,
                input resetn,
                output reg writeEn,
                output reg [4:0] x_increment,  //0-31
                output reg [5:0] y_increment,  //0-63
                output reg ld_xy,
                output reg done
```

```
        );

        reg [4:0] current_state, next_state;
        reg [10:0] counter;

        localparam  S_LOAD_XY       = 5'd0,
           S_CYCLE_0       = 5'd1,
           S_CYCLE_1       = 5'd2,
                                S_CYCLE_DONE      = 5'd3,
                                size                        = 11'd2047;



        always@(*)
    begin: state_table
        case (current_state)
           S_LOAD_XY: next_state = plot ? S_CYCLE_0 : S_LOAD_XY; // Loop in current
state until value is input
           S_CYCLE_0: next_state = S_CYCLE_1;
                                S_CYCLE_1: next_state = (counter < size) ? S_CYCLE_0 :
S_CYCLE_DONE;
                                S_CYCLE_DONE: next_state = S_LOAD_XY;

        default:     next_state = S_LOAD_XY;
                endcase
    end // state_table



        always @(*)
    begin: enable_signals
      // By default make all our signals 0
                writeEn = 1'b0;
                x_increment = 5'b0;
                y_increment = 6'b0;
                ld_xy = 1'b0;
                done = 1'b0;

       case (current_state)
          S_LOAD_XY: begin
             ld_xy = 1'b1;
             end
          S_CYCLE_0: begin
                                x_increment = counter[4:0];
                                y_increment = counter[10:5];
          end
```

```
                        S_CYCLE_1: begin
                                writeEn = 1'b1;
        end
                        S_CYCLE_DONE: begin
                                done = 1;
                        end

    endcase
  end

  always @(posedge clk)
  begin
                if(!resetn || ((current_state != S_CYCLE_1) && (current_state !=
S_CYCLE_0)))
            counter <= 10'b0;
                if (current_state == S_CYCLE_1)
            counter <= counter + 1'b1;
  end

   // current_state registers
   always@(posedge clk)
   begin: state_FFs
      if(!resetn)
         current_state <= S_LOAD_XY;
      else
         current_state <= next_state;
   end // state_FFS

endmodule

module b_data_path(
        input [8:0] x_in,
        input [7:0] y_in,
        input resetn,
        input clk,

        input [4:0] x_increment, //0-31 depents on size
        input [5:0] y_increment, //0-63
        input ld_xy,

        output reg [8:0] x_out,
        output reg [7:0] y_out,
        output [2:0] colour_out
        );

        reg [8:0] last_x;
```

```
	reg [7:0] last_y;

	wire [10:0] address;
	assign address = {y_increment, x_increment};
	blitz_image bi1(address,clk,3'b0,1'b0,colour_out);

	always@(posedge clk) begin
		if(!resetn) begin
			x_out <= 9'b0;
			y_out <= 8'b0;
			last_x <= 9'b0;
			last_y <= 8'b0;
			//colour_out <= 3'b0;
		end
		else if(ld_xy) begin
			last_x <= x_in;
			last_y <= y_in;
			end
		else begin
				x_out <= (last_x + x_increment);
				y_out <= (last_y + y_increment);
				//colour_out <= ((last_y + y_increment) >= 8'd240 ||(last_x +
x_increment >= 9'd320)) ? 3'b000 : 3'b111;  //set it to white temp
		end
	end
endmodule

module blitz_hook_gu(
		clk,
		x_in,
		y_in,
		resetn,
		plot,

		colour_out,
		x_out,
		y_out,
		writeEn,
		done
	);

	input			clk;
	input	[8:0]	x_in;
	input	[7:0]	y_in;
	input			resetn;
	input			plot;
```

```
	// Declare your inputs and outputs here
	output  [2:0]    colour_out;
	output  [8:0]    x_out;
	output  [7:0]    y_out;
	output                  writeEn;
	output                  done;

	wire [3:0] x_increment; //0-7
	wire [3:0] y_increment; //0-7
	wire ld_xy;

	hook_finite_state_machine f1(
		.plot(plot),
		.clk(clk),
		.y_in(y_in),
		.resetn(resetn),
		.writeEn(writeEn),
		.x_increment(x_increment),
		.y_increment(y_increment),
		.ld_xy(ld_xy),
		.done(done)
	);

	hook_data_path dp1(
	.x_in(x_in),
	.y_in(y_in),
	.resetn(resetn),
	.clk(clk),
	.x_increment(x_increment),
	.y_increment(y_increment),
	.ld_xy(ld_xy),

	.x_out(x_out),
	.y_out(y_out),
	.colour_out(colour_out)
	);

endmodule

module hook_finite_state_machine(
		input plot,
		input clk,
		input resetn,
		input [7:0] y_in,
		output reg writeEn,
		output reg [3:0] x_increment,  //0-7
```

```
            output reg [3:0] y_increment,  //0-7
            output reg ld_xy,
            output reg done
        );

        reg [4:0] current_state, next_state;
        reg [7:0] counter;

        localparam  S_LOAD_XY       = 5'd0,
           S_CYCLE_0       = 5'd1,
           S_CYCLE_1       = 5'd2,
                              S_CYCLE_DONE      = 5'd3,
                              size                        = 8'd255;



        always@(*)
   begin: state_table
        case (current_state)
           S_LOAD_XY: next_state = plot ? ((y_in==0) ? S_CYCLE_DONE : S_CYCLE_0) :
S_LOAD_XY; // Loop in current state until value is input
           S_CYCLE_0: next_state = S_CYCLE_1;
                              S_CYCLE_1: next_state = (counter < size) ? S_CYCLE_0 :
S_CYCLE_DONE;
                              S_CYCLE_DONE: next_state = S_LOAD_XY;

        default:     next_state = S_LOAD_XY;
            endcase
   end // state_table



        always @(*)
   begin: enable_signals
     // By default make all our signals 0
            writeEn = 1'b0;
            x_increment = 4'b0;
            y_increment = 4'b0;
            ld_xy = 1'b0;
            done = 1'b0;

     case (current_state)
        S_LOAD_XY: begin
           ld_xy = 1'b1;
           end
        S_CYCLE_0: begin
```

```
                                x_increment = counter[3:0];
                                y_increment = counter[7:4];
            end
                        S_CYCLE_1: begin
                                writeEn = 1'b1;
            end
                        S_CYCLE_DONE: begin
                                done = 1;
                        end

        endcase
    end

    always @(posedge clk)
    begin
                if(!resetn || ((current_state != S_CYCLE_1) && (current_state !=
S_CYCLE_0)))
            counter <= 6'b0;
                if (current_state == S_CYCLE_1)
            counter <= counter + 1'b1;
    end

     // current_state registers
     always@(posedge clk)
     begin: state_FFs
        if(!resetn)
            current_state <= S_LOAD_XY;
        else
            current_state <= next_state;
     end // state_FFS

endmodule

module hook_data_path(
            input [8:0] x_in,
            input [7:0] y_in,
            input resetn,
            input clk,

            input [3:0] x_increment, //0-7 depents on size
            input [3:0] y_increment, //0-7
            input ld_xy,

            output reg [8:0] x_out,
            output reg [7:0] y_out,
            output [2:0] colour_out
```

```
	);

	reg [8:0] last_x;
	reg [7:0] last_y;

	wire [7:0] address;
	assign address = {y_increment,x_increment};
	hook_image hi1(address,clk,3'b0,1'b0,colour_out);

	always@(posedge clk) begin
		if(!resetn) begin
			x_out <= 9'b0;
			y_out <= 8'b0;
			last_x <= 9'b0;
			last_y <= 8'b0;
			//colour_out <= 3'b0;
		end
		else if(ld_xy) begin
			last_x <= x_in - 9'd8;
			last_y <= y_in;
			end
		else begin
				x_out <= (last_x + x_increment);
				y_out <= (last_y + y_increment);
				//colour_out <= ((last_y + y_increment) >= 8'd240 ||(last_x +
x_increment >= 9'd320)) ? 3'b100 : 3'b100;  //set it to yellow temp
		end
	end
endmodule

module blitz_arm_gu(
		clk,
		x_in,
		y_in,
		resetn,
		plot,

		colour_out,
		x_out,
		y_out,
		writeEn,
		done
	);

	input			clk;
	input	[8:0]	x_in;
```

```
        input   [7:0]   y_in;
        input           resetn;
        input           plot;
        // Declare your inputs and outputs here
        output  [2:0]   colour_out;
        output  [8:0]   x_out;
        output  [7:0]   y_out;
        output          writeEn;
        output          done;

        wire [5:0] x_increment; //0-51
        wire y_increment; //0-1
        wire ld_xy;

        arm_finite_state_machine f1(
                .plot(plot),
                .clk(clk),
                .resetn(resetn),
                .writeEn(writeEn),
                .x_in(x_in),
                .y_in(y_in),
                .x_increment(x_increment),
                .y_increment(y_increment),
                .ld_xy(ld_xy),
                .done(done)
        );

        arm_data_path dp1(
        .x_in(x_in),
        .y_in(y_in),
        .resetn(resetn),
        .clk(clk),
        .x_increment(x_increment),
        .y_increment(y_increment),
        .ld_xy(ld_xy),

        .x_out(x_out),
        .y_out(y_out),
        .colour_out(colour_out)
        );

endmodule

module arm_finite_state_machine(
                input plot,
                input clk,
```

```
            input resetn,
            input [8:0] x_in,
            input [7:0] y_in,
            output reg writeEn,
            output reg [5:0] x_increment,  //0-51
            output reg y_increment,  //0-1
            output reg ld_xy,
            output reg done
        );

        reg [4:0] current_state, next_state;
        reg [5:0] counter_x;
        reg counter_y;

        localparam  S_LOAD_XY       = 5'd0,
           S_CYCLE_0       = 5'd1,
           S_CYCLE_1       = 5'd2,
                           S_CYCLE_DONE      = 5'd3;



        always@(*)
   begin: state_table
        case (current_state)
           S_LOAD_XY: next_state = (plot) ?( (y_in==0) ? S_CYCLE_DONE : S_CYCLE_0) :
S_LOAD_XY; // Loop in current state until value is input
           S_CYCLE_0: next_state = S_CYCLE_1;
                           S_CYCLE_1: next_state = (counter_y&&(counter_x>= x_in -
9'd42)) ? S_CYCLE_DONE : S_CYCLE_0;
                           S_CYCLE_DONE: next_state = S_LOAD_XY;

        default:     next_state = S_LOAD_XY;
            endcase
   end // state_table



        always @(*)
   begin: enable_signals
     // By default make all our signals 0
            writeEn = 1'b0;
            x_increment = 6'b0;
            y_increment = 1'b0;
            ld_xy = 1'b0;
            done = 1'b0;
```

```
        case (current_state)
            S_LOAD_XY: begin
                ld_xy = 1'b1;
                end
            S_CYCLE_0: begin
                                    x_increment = counter_x;
                                    y_increment = counter_y;
            end
                            S_CYCLE_1: begin
                                    writeEn = 1'b1;
            end
                            S_CYCLE_DONE: begin
                                    done = 1;
                            end

        endcase
    end

    always @(posedge clk)
    begin
                    if(!resetn || ((current_state != S_CYCLE_1) && (current_state !=
S_CYCLE_0))) begin
                counter_x <= 6'b0;
                    counter_y <= 1'b0;
                    end
                    if (current_state == S_CYCLE_1)begin
                            if(counter_x >= x_in - 9'd42) begin
                                    counter_x <= 6'd0;
                                    counter_y <= 1'b1;
                                    end
                            else counter_x <= counter_x + 1'b1;
                    end

    end

    // current_state registers
    always@(posedge clk)
    begin: state_FFs
        if(!resetn)
            current_state <= S_LOAD_XY;
        else
            current_state <= next_state;
    end // state_FFS

endmodule
```

```
module arm_data_path(
	input [8:0] x_in,
	input [7:0] y_in,
	input resetn,
	input clk,

	input [5:0] x_increment, //0-51
	input y_increment, //0-1
	input ld_xy,

	output reg [8:0] x_out,
	output reg [7:0] y_out,
	output reg [2:0] colour_out
	);

	reg [8:0] last_x;
	reg [7:0] last_y;

	always@(posedge clk) begin
		if(!resetn) begin
			x_out <= 9'b0;
			y_out <= 8'b0;
			last_x <= 9'b0;
			last_y <= 8'b0;
			colour_out <= 3'b0;
		end
		else if(ld_xy) begin
			last_x <= 9'd42;
			last_y <= y_in + 8'd7;
			end
		else begin
				x_out <= (last_x + x_increment);
				y_out <= (last_y + y_increment);
				colour_out <= ((last_y + y_increment) >= 8'd240 ||(last_x +
x_increment >= 9'd320)) ? 3'b100 : 3'b000;  //set it to yellow temp
		end
	end
endmodule

// graphing unit for general first, whenever we are gonna use
// we need to create different module for each controller by adding different RAM inside.

module poro_gu
	(
		clk,
		x_in,
```

```
        y_in,
        resetn,
        plot,

        colour_out,
        x_out,
        y_out,
        writeEn,
        done
);

input           clk;
input   [8:0]   x_in;
input   [7:0]   y_in;
input           resetn;
input           plot;
// Declare your inputs and outputs here
output  [2:0]   colour_out;
output  [8:0]   x_out;
output  [7:0]   y_out;
output          writeEn;
output          done;

wire [3:0] x_increment; //0-15
wire [3:0] y_increment; //0-15
wire ld_xy;

p_finite_state_machine f1(
        .plot(plot),
        .clk(clk),
        .resetn(resetn),
        .writeEn(writeEn),
        .x_increment(x_increment),
        .y_increment(y_increment),
        .ld_xy(ld_xy),
        .done(done)
);

p_data_path dp1(
.x_in(x_in),
.y_in(y_in),
.resetn(resetn),
.clk(clk),
.x_increment(x_increment),
.y_increment(y_increment),
.ld_xy(ld_xy),
```

```
        .x_out(x_out),
        .y_out(y_out),
        .colour_out(colour_out)
        );

endmodule

module p_finite_state_machine(
                input plot,
                input clk,
                input resetn,
                output reg writeEn,
                output reg [3:0] x_increment,  //0-7
                output reg [3:0] y_increment,  //0-7
                output reg ld_xy,
                output reg done
        );

        reg [4:0] current_state, next_state;
        reg [7:0] counter;

        localparam  S_LOAD_XY       = 5'd0,
           S_CYCLE_0       = 5'd1,
           S_CYCLE_1       = 5'd2,
                                S_CYCLE_DONE      = 5'd3,
                                size                      = 8'd255;



        always@(*)
   begin: state_table
        case (current_state)
           S_LOAD_XY: next_state = plot ? S_CYCLE_0 : S_LOAD_XY; // Loop in current
state until value is input
           S_CYCLE_0: next_state = S_CYCLE_1;
                                S_CYCLE_1: next_state = (counter < size) ? S_CYCLE_0 :
S_CYCLE_DONE;
                                S_CYCLE_DONE: next_state = S_LOAD_XY;

        default:     next_state = S_LOAD_XY;
                endcase
   end // state_table
```

```
        always @(*)
    begin: enable_signals
        // By default make all our signals 0
                writeEn = 1'b0;
                x_increment = 4'b0;
                y_increment = 4'b0;
                ld_xy = 1'b0;
                done = 1'b0;

        case (current_state)
            S_LOAD_XY: begin
                ld_xy = 1'b1;
                end
            S_CYCLE_0: begin
                                x_increment = counter[3:0];
                                y_increment = counter[7:4];
            end
                        S_CYCLE_1: begin
                                writeEn = 1'b1;
            end
                        S_CYCLE_DONE: begin
                                done = 1;
                        end

        endcase
     end

    always @(posedge clk)
    begin
                    if(!resetn || ((current_state != S_CYCLE_1) && (current_state !=
S_CYCLE_0)))
                counter <= 8'b0;
                    if (current_state == S_CYCLE_1)
                counter <= counter + 1'b1;
    end

     // current_state registers
     always@(posedge clk)
     begin: state_FFs
        if(!resetn)
            current_state <= S_LOAD_XY;
        else
            current_state <= next_state;
     end // state_FFS

endmodule
```

```
module p_data_path(
	input [8:0] x_in,
	input [7:0] y_in,
	input resetn,
	input clk,

	input [3:0] x_increment, //0-15 depents on size
	input [3:0] y_increment, //0-15
	input ld_xy,

	output reg [8:0] x_out,
	output reg [7:0] y_out,
	output [2:0] colour_out
	);

	reg [8:0] last_x;
	reg [7:0] last_y;

	wire [7:0] address;
	assign address = {y_increment,x_increment};
	poro_image pi1(address,clk,3'b0,1'b0,colour_out);

	always@(posedge clk) begin
		if(!resetn) begin
			x_out <= 9'b0;
			y_out <= 8'b0;
			last_x <= 9'b0;
			last_y <= 8'b0;
			// colour_out <= 3'b111;
		end
		else if(ld_xy) begin
			last_x <= x_in;
			last_y <= y_in;
			end
		else begin
				x_out <= (last_x + x_increment);
				y_out <= (last_y + y_increment);
				// colour_out <= ((last_y + y_increment) >= 8'd240 ||(last_x +
x_increment >= 9'd320)) ? 3'b100 : realcolour;  //set it to yellow temp
		end
	end
endmodule

//gu for background needs a ram with img imf in datapath
```

```
module trailer_gu
	(
		frame,
		clk,
		resetn,
		plot,

		colour_out,
		x_out,
		y_out,
		writeEn,
		done
	);
	input			frame;
	input			clk;
	input			resetn;
	input			plot;
	// Declare your inputs and outputs here
	output	[2:0]	colour_out;
	output	[8:0]	x_out;
	output	[7:0]	y_out;
	output			writeEn;
	output			done;


	wire [8:0] x_increment;
	wire [7:0] y_increment;

	finite_state_machine f2(
		.plot(plot),
		.clk(clk),
		.resetn(resetn),
		.writeEn(writeEn),
		.x_increment(x_increment),
		.y_increment(y_increment),
		.done(done)
	);

	data_path dp2(
	.frame(frame),
	.resetn(resetn),
	.clk(clk),
	.x_increment(x_increment),
	.y_increment(y_increment),

	.x_out(x_out),
```

```
        .y_out(y_out),
        .colour_out(colour_out)
        );

endmodule

module data_path(
        input frame,
        input resetn,
        input clk,

        input [8:0] x_increment, // 320*240
        input [7:0] y_increment, // 320*240

        output reg [8:0] x_out,
        output reg [7:0] y_out,
        output [2:0] colour_out
        );

        wire [17:0] address;
        assign address = x_increment + y_increment * 10'd320;
        trailer_image ti(address,clk,3'b0,1'b0,colour_out);


        always@(posedge clk) begin
                if(!resetn) begin
                        x_out <= 9'b0;
                        y_out <= 8'b0;
                        //colour_out <= 3'b0;
                end
                else begin
                                x_out <= x_increment;
                                y_out <= y_increment;
                        //      colour_out <= 3'b000; //set in to blk temp
                end
        end
endmodule

// graphing unit for general first, whenever we are gonna use
// we need to create different module for each controller by adding different RAM inside.

module hp_gu
        (
                clk,
                y_in,
      health_point,
```

```
                resetn,
                plot,

                colour_out,
                x_out,
                y_out,
                writeEn,
                done
        );

    input       [2:0]   health_point;
        input                   clk;
        input   [7:0]   y_in;
        input                   resetn;
        input                   plot;
        // Declare your inputs and outputs here
        output  [2:0]   colour_out;
        output  [8:0]   x_out;
        output  [7:0]   y_out;
        output                  writeEn;
        output                  done;


        wire [4:0] x_increment; //0-31
        wire [2:0] y_increment; //0-7
        wire ld_xy;

        hp_finite_state_machine f1(
                .plot(plot),
                .clk(clk),
                .resetn(resetn),
                .writeEn(writeEn),
                .x_increment(x_increment),
                .y_increment(y_increment),
                .ld_xy(ld_xy),
                .done(done)
        );

        hp_data_path dp1(
    .health_point(health_point),
        .x_in(9'd10),
        .y_in(y_in),
        .resetn(resetn),
        .clk(clk),
        .x_increment(x_increment),
        .y_increment(y_increment),
```

```
        .ld_xy(ld_xy),

        .x_out(x_out),
        .y_out(y_out),
        .colour_out(colour_out)
        );

endmodule

module hp_finite_state_machine(
                input plot,
                input clk,
                input resetn,
                output reg writeEn,
                output reg [4:0] x_increment,  //0-31
                output reg [2:0] y_increment,  //0-7
                output reg ld_xy,
                output reg done
        );

        reg [4:0] current_state, next_state;
        reg [7:0] counter;

        localparam  S_LOAD_XY       = 5'd0,
           S_CYCLE_0       = 5'd1,
           S_CYCLE_1       = 5'd2,
                                S_CYCLE_DONE      = 5'd3,
                                size                      = 8'd255;



        always@(*)
   begin: state_table
        case (current_state)
           S_LOAD_XY: next_state = plot ? S_CYCLE_0 : S_LOAD_XY; // Loop in current
state until value is input
           S_CYCLE_0: next_state = S_CYCLE_1;
                                S_CYCLE_1: next_state = (counter < size) ? S_CYCLE_0 :
S_CYCLE_DONE;
                                S_CYCLE_DONE: next_state = S_LOAD_XY;

        default:     next_state = S_LOAD_XY;
                endcase
   end // state_table
```

```
        always @(*)
    begin: enable_signals
        // By default make all our signals 0
                writeEn = 1'b0;
                x_increment = 5'b0;
                y_increment = 3'b0;
                ld_xy = 1'b0;
                done = 1'b0;

        case (current_state)
            S_LOAD_XY: begin
                ld_xy = 1'b1;
                end
            S_CYCLE_0: begin
                                x_increment = counter[4:0];
                                y_increment = counter[7:5];
            end
                        S_CYCLE_1: begin
                                writeEn = 1'b1;
            end
                        S_CYCLE_DONE: begin
                                done = 1;
                        end

        endcase
    end

    always @(posedge clk)
    begin
                if(!resetn || ((current_state != S_CYCLE_1) && (current_state !=
S_CYCLE_0)))
                counter <= 8'b0;
                    if (current_state == S_CYCLE_1)
                counter <= counter + 1'b1;
    end

    // current_state registers
    always@(posedge clk)
    begin: state_FFs
        if(!resetn)
            current_state <= S_LOAD_XY;
        else
            current_state <= next_state;
    end // state_FFS
```

```
endmodule

module hp_data_path(
   input [2:0] health_point,
         input [8:0] x_in,
         input [7:0] y_in,
         input resetn,
         input clk,

         input [4:0] x_increment, //0-31 depents on size
         input [2:0] y_increment, //0-7
         input ld_xy,

         output reg [8:0] x_out,
         output reg [7:0] y_out,
         output [2:0] colour_out
         );

         reg [8:0] last_x;
         reg [7:0] last_y;
         wire [13:0] address;
         assign address = {y_increment, x_increment} + 9'd256 * (health_point);
   hp_image hpi1(address,clk,3'b0,1'b0,colour_out);

         always@(posedge clk) begin
                  if(!resetn) begin
                           x_out <= 9'b0;
                           y_out <= 8'b0;
                           last_x <= 9'b0;
                           last_y <= 8'b0;
                           //colour_out <= 3'b0;
                  end
                  else if(ld_xy) begin
                           last_x <= x_in;
                           last_y <= y_in - 8'd12;
                           end
                  else begin
                                    x_out <= (last_x + x_increment);
                                    y_out <= (last_y + y_increment);
                                    //colour_out <= ((last_y + y_increment) >= 8'd240 ||(last_x +
x_increment >= 9'd320)) ? 3'b000 : 3'b111;  //set it to white temp
                  end
         end
endmodule

//==============================Images============================
```

```
// megafunction wizard: %RAM: 1-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: poro_image.v
// Megafunction Name(s):
//                         altsyncram
//
// Simulation Library Files(s):
//                         altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 16.0.0 Build 211 04/27/2016 SJ Lite Edition
// ************************************************************


//Copyright (C) 1991-2016 Altera Corporation. All rights reserved.
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, the Altera Quartus Prime License Agreement,
//the Altera MegaCore Function License Agreement, or other
//applicable license agreement, including, without limitation,
//that your use is for the sole purpose of programming logic
//devices manufactured by Altera and sold by Altera or its
//authorized distributors.  Please refer to the applicable
//agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module poro_image (
        address,
        clock,
        data,
        wren,
        q);
```

```
	input	[7:0]  address;
	input	  clock;
	input	[2:0]  data;
	input	  wren;
	output  [2:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
	tri1	  clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

	wire [2:0] sub_wire0;
	wire [2:0] q = sub_wire0[2:0];

	altsyncram	altsyncram_component (
				.address_a (address),
				.clock0 (clock),
				.data_a (data),
				.wren_a (wren),
				.q_a (sub_wire0),
				.aclr0 (1'b0),
				.aclr1 (1'b0),
				.address_b (1'b1),
				.addressstall_a (1'b0),
				.addressstall_b (1'b0),
				.byteena_a (1'b1),
				.byteena_b (1'b1),
				.clock1 (1'b1),
				.clocken0 (1'b1),
				.clocken1 (1'b1),
				.clocken2 (1'b1),
				.clocken3 (1'b1),
				.data_b (1'b1),
				.eccstatus (),
				.q_b (),
				.rden_a (1'b1),
				.rden_b (1'b1),
				.wren_b (1'b0));
	defparam
		altsyncram_component.clock_enable_input_a = "BYPASS",
		altsyncram_component.clock_enable_output_a = "BYPASS",
		altsyncram_component.init_file = "../poro16.mif",
		altsyncram_component.intended_device_family = "Cyclone V",
		altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
```

```
		altsyncram_component.lpm_type = "altsyncram",
		altsyncram_component.numwords_a = 256,
		altsyncram_component.operation_mode = "SINGLE_PORT",
		altsyncram_component.outdata_aclr_a = "NONE",
		altsyncram_component.outdata_reg_a = "UNREGISTERED",
		altsyncram_component.power_up_uninitialized = "FALSE",
		altsyncram_component.read_during_write_mode_port_a =
"NEW_DATA_NO_NBE_READ",
		altsyncram_component.widthad_a = 8,
		altsyncram_component.width_a = 3,
		altsyncram_component.width_byteena_a = 1;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrData NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "../poro16.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "256"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegData NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
```

```
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
// Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "8"
// Retrieval info: PRIVATE: WidthData NUMERIC "3"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "../poro16.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "256"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING
"NEW_DATA_NO_NBE_READ"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "8"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "3"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 8 0 INPUT NODEFVAL "address[7..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 3 0 INPUT NODEFVAL "data[2..0]"
// Retrieval info: USED_PORT: q 0 0 3 0 OUTPUT NODEFVAL "q[2..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
// Retrieval info: CONNECT: @address_a 0 0 8 0 address 0 0 8 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 3 0 data 0 0 3 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 3 0 @q_a 0 0 3 0
// Retrieval info: GEN_FILE: TYPE_NORMAL poro_image.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL poro_image.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL poro_image.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL poro_image.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL poro_image_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL poro_image_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf

// megafunction wizard: %RAM: 1-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
```

```
// File Name: background_image.v
// Megafunction Name(s):
//                         altsyncram
//
// Simulation Library Files(s):
//                         altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 16.0.0 Build 211 04/27/2016 SJ Standard Edition
// ************************************************************


//Copyright (C) 1991-2016 Altera Corporation. All rights reserved.
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, the Altera Quartus Prime License Agreement,
//the Altera MegaCore Function License Agreement, or other
//applicable license agreement, including, without limitation,
//that your use is for the sole purpose of programming logic
//devices manufactured by Altera and sold by Altera or its
//authorized distributors.  Please refer to the applicable
//agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module background_image (
        address,
        clock,
        data,
        wren,
        q);

        input    [17:0]  address;
        input     clock;
        input    [2:0]  data;
        input     wren;
        output  [2:0]  q;
`ifndef ALTERA_RESERVED_QIS
```

```
// synopsys translate_off
`endif
	tri1	clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

	wire [2:0] sub_wire0;
	wire [2:0] q = sub_wire0[2:0];

	altsyncram	altsyncram_component (
				.address_a (address),
				.clock0 (clock),
				.data_a (data),
				.wren_a (wren),
				.q_a (sub_wire0),
				.aclr0 (1'b0),
				.aclr1 (1'b0),
				.address_b (1'b1),
				.addressstall_a (1'b0),
				.addressstall_b (1'b0),
				.byteena_a (1'b1),
				.byteena_b (1'b1),
				.clock1 (1'b1),
				.clocken0 (1'b1),
				.clocken1 (1'b1),
				.clocken2 (1'b1),
				.clocken3 (1'b1),
				.data_b (1'b1),
				.eccstatus (),
				.q_b (),
				.rden_a (1'b1),
				.rden_b (1'b1),
				.wren_b (1'b0));
	defparam
		altsyncram_component.clock_enable_input_a = "BYPASS",
		altsyncram_component.clock_enable_output_a = "BYPASS",
		altsyncram_component.init_file = "../background.mif",
		altsyncram_component.intended_device_family = "Cyclone V",
		altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
		altsyncram_component.lpm_type = "altsyncram",
		altsyncram_component.numwords_a = 153600,
		altsyncram_component.operation_mode = "SINGLE_PORT",
		altsyncram_component.outdata_aclr_a = "NONE",
		altsyncram_component.outdata_reg_a = "UNREGISTERED",
		altsyncram_component.power_up_uninitialized = "FALSE",
```

```
		altsyncram_component.read_during_write_mode_port_a =
"NEW_DATA_NO_NBE_READ",
		altsyncram_component.widthad_a = 18,
		altsyncram_component.width_a = 3,
		altsyncram_component.width_byteena_a = 1;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrData NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "../background.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "153600"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegData NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
// Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "18"
// Retrieval info: PRIVATE: WidthData NUMERIC "3"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
```

```
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "../background.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "153600"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING
"NEW_DATA_NO_NBE_READ"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "18"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "3"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 18 0 INPUT NODEFVAL "address[17..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 3 0 INPUT NODEFVAL "data[2..0]"
// Retrieval info: USED_PORT: q 0 0 3 0 OUTPUT NODEFVAL "q[2..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
// Retrieval info: CONNECT: @address_a 0 0 18 0 address 0 0 18 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 3 0 data 0 0 3 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 3 0 @q_a 0 0 3 0
// Retrieval info: GEN_FILE: TYPE_NORMAL background_image.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL background_image.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL background_image.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL background_image.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL background_image_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL background_image_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf

// megafunction wizard: %RAM: 1-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: blitz_image.v
// Megafunction Name(s):
//                         altsyncram
//
// Simulation Library Files(s):
//                         altera_mf
```

```
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 16.0.0 Build 211 04/27/2016 SJ Lite Edition
// ************************************************************


//Copyright (C) 1991-2016 Altera Corporation. All rights reserved.
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, the Altera Quartus Prime License Agreement,
//the Altera MegaCore Function License Agreement, or other
//applicable license agreement, including, without limitation,
//that your use is for the sole purpose of programming logic
//devices manufactured by Altera and sold by Altera or its
//authorized distributors.  Please refer to the applicable
//agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module blitz_image (
	address,
	clock,
	data,
	wren,
	q);

	input	[10:0]  address;
	input	  clock;
	input	[2:0]  data;
	input	  wren;
	output  [2:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
	tri1	  clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif
```

```
	wire [2:0] sub_wire0;
	wire [2:0] q = sub_wire0[2:0];

	altsyncram	altsyncram_component (
				.address_a (address),
				.clock0 (clock),
				.data_a (data),
				.wren_a (wren),
				.q_a (sub_wire0),
				.aclr0 (1'b0),
				.aclr1 (1'b0),
				.address_b (1'b1),
				.addressstall_a (1'b0),
				.addressstall_b (1'b0),
				.byteena_a (1'b1),
				.byteena_b (1'b1),
				.clock1 (1'b1),
				.clocken0 (1'b1),
				.clocken1 (1'b1),
				.clocken2 (1'b1),
				.clocken3 (1'b1),
				.data_b (1'b1),
				.eccstatus (),
				.q_b (),
				.rden_a (1'b1),
				.rden_b (1'b1),
				.wren_b (1'b0));
	defparam
		altsyncram_component.clock_enable_input_a = "BYPASS",
		altsyncram_component.clock_enable_output_a = "BYPASS",
		altsyncram_component.init_file = "../blitz.mif",
		altsyncram_component.intended_device_family = "Cyclone V",
		altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
		altsyncram_component.lpm_type = "altsyncram",
		altsyncram_component.numwords_a = 2048,
		altsyncram_component.operation_mode = "SINGLE_PORT",
		altsyncram_component.outdata_aclr_a = "NONE",
		altsyncram_component.outdata_reg_a = "UNREGISTERED",
		altsyncram_component.power_up_uninitialized = "FALSE",
		altsyncram_component.read_during_write_mode_port_a =
"NEW_DATA_NO_NBE_READ",
		altsyncram_component.widthad_a = 11,
		altsyncram_component.width_a = 3,
		altsyncram_component.width_byteena_a = 1;
```

```
endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrData NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "../blitz.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "2048"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegData NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
// Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "11"
// Retrieval info: PRIVATE: WidthData NUMERIC "3"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "../blitz.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
```

```
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "2048"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING
"NEW_DATA_NO_NBE_READ"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "11"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "3"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 11 0 INPUT NODEFVAL "address[10..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 3 0 INPUT NODEFVAL "data[2..0]"
// Retrieval info: USED_PORT: q 0 0 3 0 OUTPUT NODEFVAL "q[2..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
// Retrieval info: CONNECT: @address_a 0 0 11 0 address 0 0 11 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 3 0 data 0 0 3 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 3 0 @q_a 0 0 3 0
// Retrieval info: GEN_FILE: TYPE_NORMAL blitz_image.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL blitz_image.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL blitz_image.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL blitz_image.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL blitz_image_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL blitz_image_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf

// megafunction wizard: %RAM: 1-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: hook_image.v
// Megafunction Name(s):
//                         altsyncram
//
// Simulation Library Files(s):
//                         altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 16.0.0 Build 211 04/27/2016 SJ Lite Edition
// ************************************************************
```

```
// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module hook_image (
	address,
	clock,
	data,
	wren,
	q);

	input	[7:0]  address;
	input	  clock;
	input	[2:0]  data;
	input	  wren;
	output  [2:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
	tri1	  clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

	wire [2:0] sub_wire0;
	wire [2:0] q = sub_wire0[2:0];

	altsyncram	altsyncram_component (
				.address_a (address),
```

```
                .clock0 (clock),
                .data_a (data),
                .wren_a (wren),
                .q_a (sub_wire0),
                .aclr0 (1'b0),
                .aclr1 (1'b0),
                .address_b (1'b1),
                .addressstall_a (1'b0),
                .addressstall_b (1'b0),
                .byteena_a (1'b1),
                .byteena_b (1'b1),
                .clock1 (1'b1),
                .clocken0 (1'b1),
                .clocken1 (1'b1),
                .clocken2 (1'b1),
                .clocken3 (1'b1),
                .data_b (1'b1),
                .eccstatus (),
                .q_b (),
                .rden_a (1'b1),
                .rden_b (1'b1),
                .wren_b (1'b0));
        defparam
                altsyncram_component.clock_enable_input_a = "BYPASS",
                altsyncram_component.clock_enable_output_a = "BYPASS",
                altsyncram_component.init_file = "../hook.mif",
                altsyncram_component.intended_device_family = "Cyclone V",
                altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
                altsyncram_component.lpm_type = "altsyncram",
                altsyncram_component.numwords_a = 256,
                altsyncram_component.operation_mode = "SINGLE_PORT",
                altsyncram_component.outdata_aclr_a = "NONE",
                altsyncram_component.outdata_reg_a = "UNREGISTERED",
                altsyncram_component.power_up_uninitialized = "FALSE",
                altsyncram_component.read_during_write_mode_port_a =
"NEW_DATA_NO_NBE_READ",
                altsyncram_component.widthad_a = 8,
                altsyncram_component.width_a = 3,
                altsyncram_component.width_byteena_a = 1;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
```

```
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrData NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "../hook.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "256"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegData NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
// Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "8"
// Retrieval info: PRIVATE: WidthData NUMERIC "3"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "../hook.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "256"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
```

```
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING
"NEW_DATA_NO_NBE_READ"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "8"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "3"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 8 0 INPUT NODEFVAL "address[7..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 3 0 INPUT NODEFVAL "data[2..0]"
// Retrieval info: USED_PORT: q 0 0 3 0 OUTPUT NODEFVAL "q[2..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
// Retrieval info: CONNECT: @address_a 0 0 8 0 address 0 0 8 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 3 0 data 0 0 3 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 3 0 @q_a 0 0 3 0
// Retrieval info: GEN_FILE: TYPE_NORMAL hook_image.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL hook_image.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL hook_image.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL hook_image.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL hook_image_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL hook_image_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf
// megafunction wizard: %RAM: 1-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: hp_image.v
// Megafunction Name(s):
//                         altsyncram
//
// Simulation Library Files(s):
//                         altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 16.0.0 Build 211 04/27/2016 SJ Standard Edition
// ************************************************************


//Copyright (C) 1991-2016 Altera Corporation. All rights reserved.
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
```

```
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, the Altera Quartus Prime License Agreement,
//the Altera MegaCore Function License Agreement, or other
//applicable license agreement, including, without limitation,
//that your use is for the sole purpose of programming logic
//devices manufactured by Altera and sold by Altera or its
//authorized distributors.  Please refer to the applicable
//agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module hp_image (
	address,
	clock,
	data,
	wren,
	q);

	input	[13:0]  address;
	input	  clock;
	input	[2:0]  data;
	input	  wren;
	output  [2:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
	tri1	  clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

	wire [2:0] sub_wire0;
	wire [2:0] q = sub_wire0[2:0];

	altsyncram	altsyncram_component (
				.address_a (address),
				.clock0 (clock),
				.data_a (data),
				.wren_a (wren),
				.q_a (sub_wire0),
				.aclr0 (1'b0),
				.aclr1 (1'b0),
```

```
                .address_b (1'b1),
                .addressstall_a (1'b0),
                .addressstall_b (1'b0),
                .byteena_a (1'b1),
                .byteena_b (1'b1),
                .clock1 (1'b1),
                .clocken0 (1'b1),
                .clocken1 (1'b1),
                .clocken2 (1'b1),
                .clocken3 (1'b1),
                .data_b (1'b1),
                .eccstatus (),
                .q_b (),
                .rden_a (1'b1),
                .rden_b (1'b1),
                .wren_b (1'b0));
        defparam
                altsyncram_component.clock_enable_input_a = "BYPASS",
                altsyncram_component.clock_enable_output_a = "BYPASS",
                altsyncram_component.init_file = "../hp.mif",
                altsyncram_component.intended_device_family = "Cyclone V",
                altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
                altsyncram_component.lpm_type = "altsyncram",
                altsyncram_component.numwords_a = 12288,
                altsyncram_component.operation_mode = "SINGLE_PORT",
                altsyncram_component.outdata_aclr_a = "NONE",
                altsyncram_component.outdata_reg_a = "UNREGISTERED",
                altsyncram_component.power_up_uninitialized = "FALSE",
                altsyncram_component.read_during_write_mode_port_a =
"NEW_DATA_NO_NBE_READ",
                altsyncram_component.widthad_a = 14,
                altsyncram_component.width_a = 3,
                altsyncram_component.width_byteena_a = 1;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrData NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
```

```
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "../hp.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "12288"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegData NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
// Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "14"
// Retrieval info: PRIVATE: WidthData NUMERIC "3"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "../hp.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "12288"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING
"NEW_DATA_NO_NBE_READ"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "14"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "3"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 14 0 INPUT NODEFVAL "address[13..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
```

```
// Retrieval info: USED_PORT: data 0 0 3 0 INPUT NODEFVAL "data[2..0]"
// Retrieval info: USED_PORT: q 0 0 3 0 OUTPUT NODEFVAL "q[2..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
// Retrieval info: CONNECT: @address_a 0 0 14 0 address 0 0 14 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 3 0 data 0 0 3 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 3 0 @q_a 0 0 3 0
// Retrieval info: GEN_FILE: TYPE_NORMAL hp_image.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL hp_image.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL hp_image.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL hp_image.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL hp_image_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL hp_image_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf

// megafunction wizard: %RAM: 1-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: trailer_image.v
// Megafunction Name(s):
//                         altsyncram
//
// Simulation Library Files(s):
//                         altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 16.0.0 Build 211 04/27/2016 SJ Standard Edition
// ************************************************************


//Copyright (C) 1991-2016 Altera Corporation. All rights reserved.
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, the Altera Quartus Prime License Agreement,
//the Altera MegaCore Function License Agreement, or other
//applicable license agreement, including, without limitation,
```

```
//that your use is for the sole purpose of programming logic
//devices manufactured by Altera and sold by Altera or its
//authorized distributors.  Please refer to the applicable
//agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module trailer_image (
	address,
	clock,
	data,
	wren,
	q);

	input	[16:0]  address;
	input	  clock;
	input	[2:0]  data;
	input	  wren;
	output  [2:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
	tri1	  clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

	wire [2:0] sub_wire0;
	wire [2:0] q = sub_wire0[2:0];

	altsyncram	altsyncram_component (
				.address_a (address),
				.clock0 (clock),
				.data_a (data),
				.wren_a (wren),
				.q_a (sub_wire0),
				.aclr0 (1'b0),
				.aclr1 (1'b0),
				.address_b (1'b1),
				.addressstall_a (1'b0),
				.addressstall_b (1'b0),
				.byteena_a (1'b1),
				.byteena_b (1'b1),
				.clock1 (1'b1),
```

```
                                .clocken0 (1'b1),
                                .clocken1 (1'b1),
                                .clocken2 (1'b1),
                                .clocken3 (1'b1),
                                .data_b (1'b1),
                                .eccstatus (),
                                .q_b (),
                                .rden_a (1'b1),
                                .rden_b (1'b1),
                                .wren_b (1'b0));
        defparam
                altsyncram_component.clock_enable_input_a = "BYPASS",
                altsyncram_component.clock_enable_output_a = "BYPASS",
                altsyncram_component.init_file = "../trailer.mif",
                altsyncram_component.intended_device_family = "Cyclone V",
                altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
                altsyncram_component.lpm_type = "altsyncram",
                altsyncram_component.numwords_a = 76800,
                altsyncram_component.operation_mode = "SINGLE_PORT",
                altsyncram_component.outdata_aclr_a = "NONE",
                altsyncram_component.outdata_reg_a = "UNREGISTERED",
                altsyncram_component.power_up_uninitialized = "FALSE",
                altsyncram_component.read_during_write_mode_port_a =
"NEW_DATA_NO_NBE_READ",
                altsyncram_component.widthad_a = 17,
                altsyncram_component.width_a = 3,
                altsyncram_component.width_byteena_a = 1;


endmodule
// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrData NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
```

```
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "../trailer.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "76800"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegData NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
// Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "17"
// Retrieval info: PRIVATE: WidthData NUMERIC "3"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "../trailer.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "76800"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING
"NEW_DATA_NO_NBE_READ"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "17"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "3"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 17 0 INPUT NODEFVAL "address[16..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 3 0 INPUT NODEFVAL "data[2..0]"
// Retrieval info: USED_PORT: q 0 0 3 0 OUTPUT NODEFVAL "q[2..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
// Retrieval info: CONNECT: @address_a 0 0 17 0 address 0 0 17 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 3 0 data 0 0 3 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
```

// Retrieval info: CONNECT: q 0 0 3 0 @q_a 0 0 3 0
// Retrieval info: GEN_FILE: TYPE_NORMAL trailer_image.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL trailer_image.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL trailer_image.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL trailer_image.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL trailer_image_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL trailer_image_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf