Thomas Eckburg, teckburg, and Nicholas Marcopoli, nmarcopo
May 1, 2018
Logic Design Final Project - Group TU21
Video Link: https://youtu.be/-evo9k03yrI

Final Project: Super Breakout: Lazer Edition

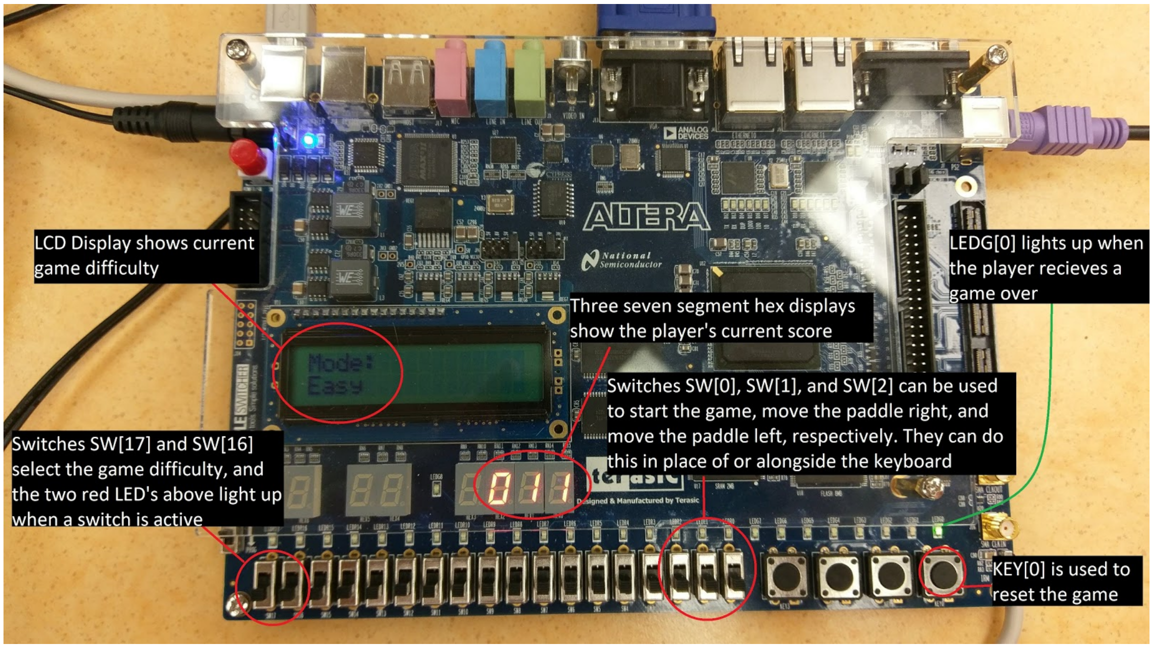
Member Contribution

5 points to each member - Nick and Thomas.

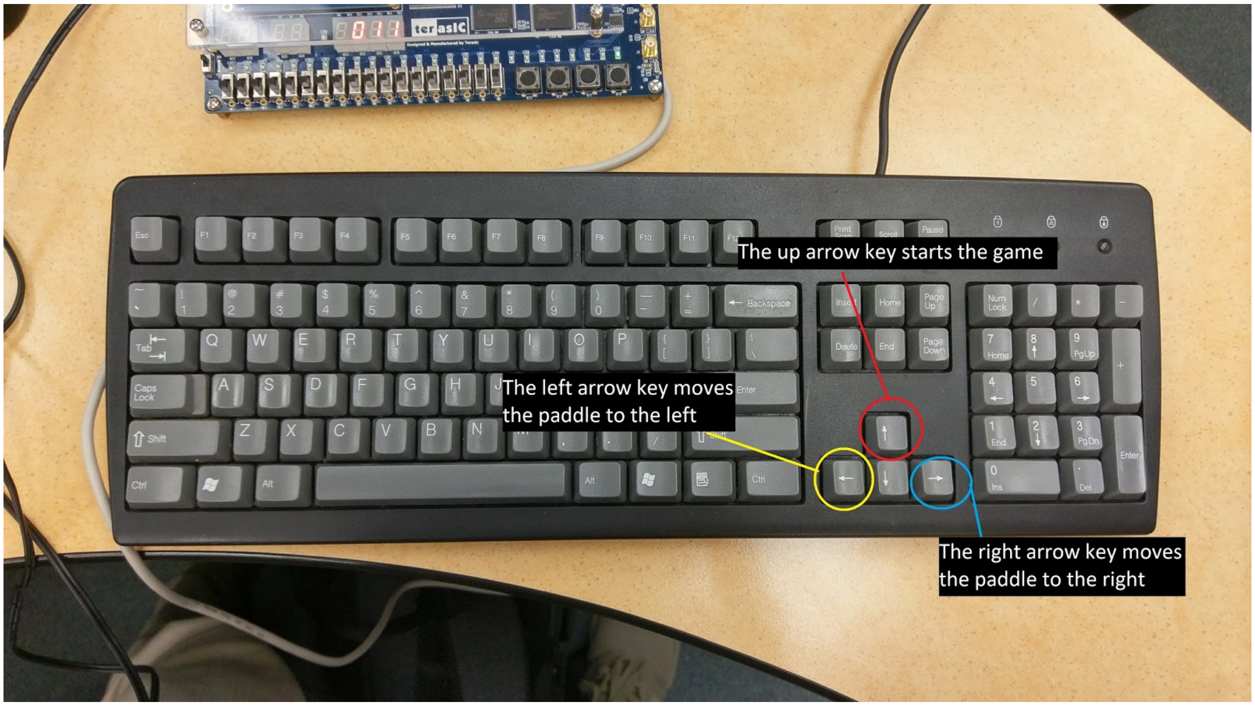
Executive Summary

The Super Breakout: Lazer Edition project uses the following elements of the Altera 115-DE2 board to create an interactive laser game inspired by the popular game "Breakout": LED's, switches, a button, 7-segment displays, the 50 MHz system clock, the LCD display, the VGA display, read-only memory (ROM), a PS/2 keyboard, and audio. The user can use the two leftmost switches (SW17 and SW16) on the board to select a game difficulty shown on the LCD, where the speed of the game increases with higher difficulties. The user can start the game using the "up" arrow key on the keyboard, or SW0 in the absence of the keyboard. The user will then see the laser start at a random position along the top of the screen and move down the screen. He or she must bounce the laser off of the paddle at the bottom of the screen, controlled by either the left and right arrow keys or SW1 and SW2. The laser emits a low tone from speakers plugged into the board when it hits a wall, and a high tone when it hits the blocks near the top of the screen. Whenever the laser cuts through blocks at the top of the screen, the score counter on the 7-segment display increases. The game ends and a green LED is lit when the laser misses the paddle and hits the bottom of the screen. If the user wishes to continue from where he or she left off before they lost the game, they may hit KEY0.

This diagram highlights the features of our project involving the Altera board:

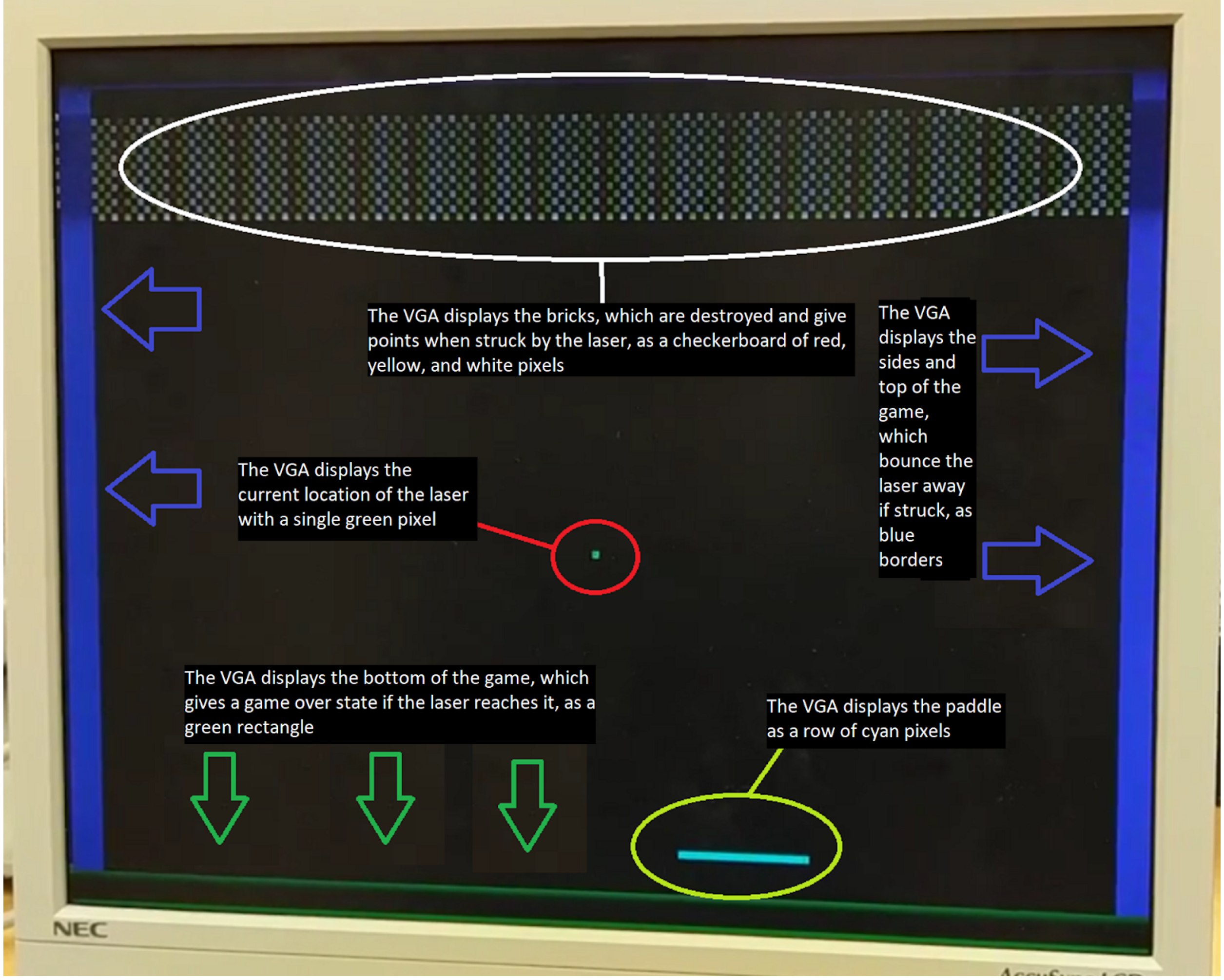

LCD Display shows current game difficulty
Mode:
Easy
LEDG[0] lights up when the player recieves a game over
Three seven segment hex displays show the player's current score
011
Switches SW[0], SW[1], and SW[2] can be used to start the game, move the paddle right, and move the paddle left, respectively. They can do this in place of or alongside the keyboard
Switches SW[17] and SW[16] select the game difficulty, and the two red LED's above light up when a switch is active
KEY[0] is used to reset the game

This diagram highlights the features of our project involving the keyboard:

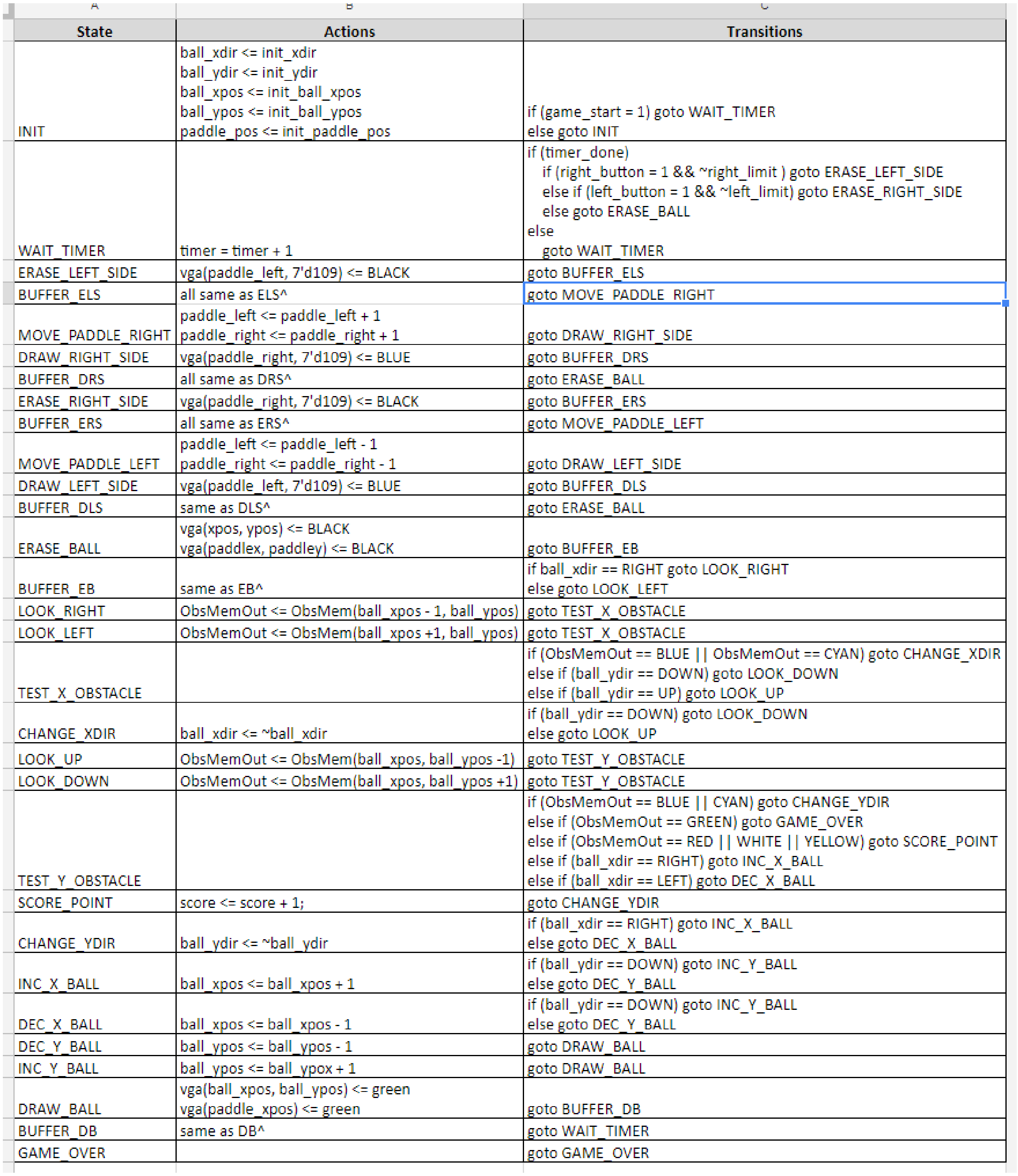


The up arrow key starts the game
The left arrow key moves the paddle to the left
The right arrow key moves the paddle to the right

This diagram highlights the features of our project involving the VGA display:


The VGA displays the bricks, which are destroyed and give points when struck by the laser, as a checkerboard of red, yellow, and white pixels
The VGA displays the sides and top of the game, which bounce the laser away if struck, as blue borders
The VGA displays the current location of the laser with a single green pixel
The VGA displays the bottom of the game, which gives a game over state if the laser reaches it, as a green rectangle
The VGA displays the paddle as a row of cyan pixels
NEC

HLSM Table



State	Actions	Transitions

INIT	ball_xdir <= init_xdir
	ball_ydir <= init_ydir
	ball_xpos <= init_ball_xpos
	ball_ypos <= init_ball_ypos
	paddle_pos <= init_paddle_pos
		if (game_start = 1) goto WAIT_TIMER
		else goto INIT

WAIT_TIMER	timer = timer + 1	if (timer_done)
		if (right_button = 1 && ~right_limit) goto ERASE_LEFT_SIDE
		else if (left_button = 1 && ~left_limit) goto ERASE_RIGHT_SIDE
		else goto ERASE_BALL
		else
		goto WAIT_TIMER

ERASE_LEFT_SIDE	vga(paddle_left, 7'd109) <= BLACK	goto BUFFER_ELS
BUFFER_ELS	all same as ELS^	goto MOVE_PADDLE_RIGHT
MOVE_PADDLE_RIGHT	paddle_left <= paddle_left + 1 paddle_right <= paddle_right + 1	goto DRAW_RIGHT_SIDE
DRAW_RIGHT_SIDE	vga(paddle_right, 7'd109) <= BLUE	goto BUFFER_DRS
BUFFER_DRS	all same as DRS^	goto ERASE_BALL
ERASE_RIGHT_SIDE	vga(paddle_right, 7'd109) <= BLACK	goto BUFFER_ERS
BUFFER_ERS	all same as ERS^	goto MOVE_PADDLE_LEFT
MOVE_PADDLE_LEFT	paddle_left <= paddle_left - 1 paddle_right <= paddle_right - 1	goto DRAW_LEFT_SIDE
DRAW_LEFT_SIDE	vga(paddle_left, 7'd109) <= BLUE	goto BUFFER_DLS
BUFFER_DLS	same as DLS^	goto ERASE_BALL
ERASE_BALL	vga(xpos, ypos) <= BLACK vga(paddlex, paddley) <= BLACK	goto BUFFER_EB

BUFFER_EB	same as EB^	if ball_xdir == RIGHT goto LOOK_RIGHT
		else goto LOOK_LEFT

LOOK_RIGHT	ObsMemOut <= ObsMem(ball_xpos - 1, ball_ypos)	goto TEST_X_OBSTACLE
LOOK_LEFT	ObsMemOut <= ObsMem(ball_xpos +1, ball_ypos)	goto TEST_X_OBSTACLE

TEST_X_OBSTACLE		if (ObsMemOut == BLUE || ObsMemOut == CYAN) goto CHANGE_XDIR
		else if (ball_ydir == DOWN) goto LOOK_DOWN
		else if (ball_ydir == UP) goto LOOK_UP

CHANGE_XDIR	ball_xdir <= ~ball_xdir	if (ball_ydir == DOWN) goto LOOK_DOWN
		else goto LOOK_UP

LOOK_UP	ObsMemOut <= ObsMem(ball_xpos, ball_ypos -1)	goto TEST_Y_OBSTACLE
LOOK_DOWN	ObsMemOut <= ObsMem(ball_xpos, ball_ypos +1)	goto TEST_Y_OBSTACLE

TEST_Y_OBSTACLE		if (ObsMemOut == BLUE || CYAN) goto CHANGE_YDIR
		else if (ObsMemOut == GREEN) goto GAME_OVER
		else if (ObsMemOut == RED || WHITE || YELLOW) goto SCORE_POINT
		else if (ball_xdir == RIGHT) goto INC_X_BALL
		else if (ball_xdir == LEFT) goto DEC_X_BALL

SCORE_POINT	score <= score + 1;	goto CHANGE_YDIR

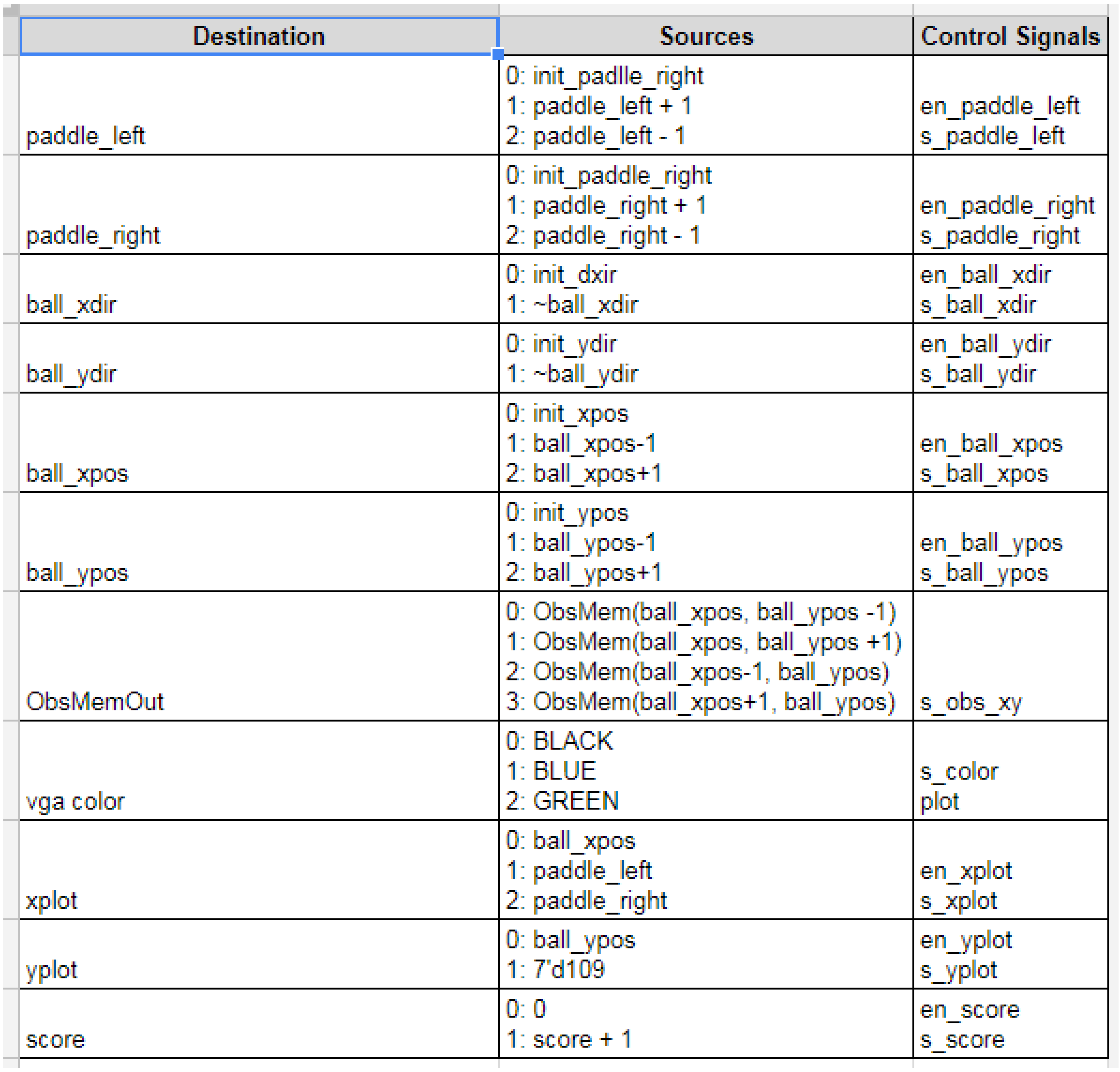
CHANGE_YDIR	ball_ydir <= ~ball_ydir	if (ball_xdir == RIGHT) goto INC_X_BALL
		else goto DEC_X_BALL

INC_X_BALL	ball_xpos <= ball_xpos + 1	if (ball_ydir == DOWN) goto INC_Y_BALL
		else goto DEC_Y_BALL

DEC_X_BALL	ball_xpos <= ball_xpos - 1	if (ball_ydir == DOWN) goto INC_Y_BALL
		else goto DEC_Y_BALL

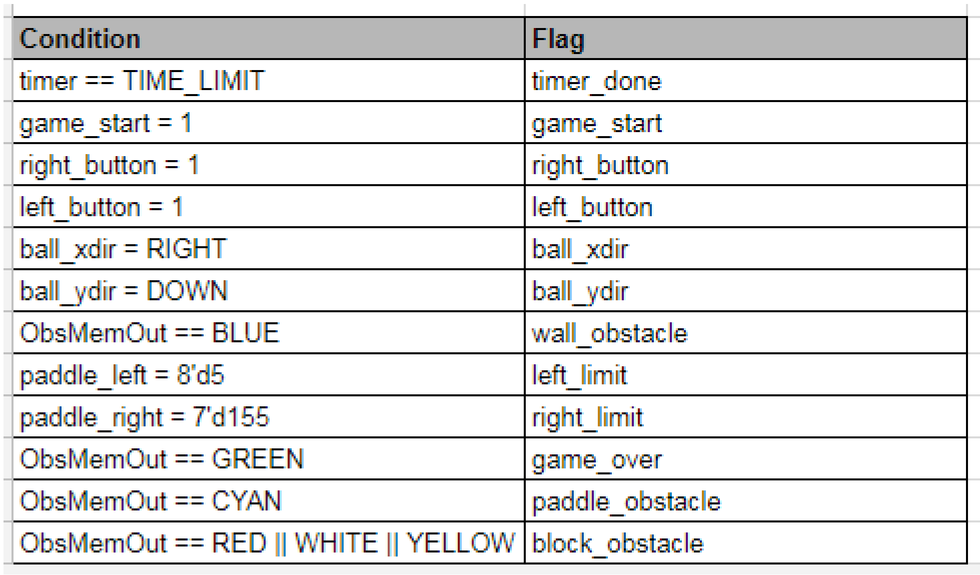
DEC_Y_BALL	ball_ypos <= ball_ypos - 1	goto DRAW_BALL
INC_Y_BALL	ball_ypos <= ball_ypox + 1	goto DRAW_BALL
DRAW_BALL	vga(ball_xpos, ball_ypos) <= green vga(paddle_xpos) <= green	goto BUFFER_DB
BUFFER_DB	same as DB^	goto WAIT_TIMER
GAME_OVER		goto GAME_OVER

Datapath Stages:



Destination	Sources	Control Signals
paddle_left	0: init_padlle_right 1: paddle_left + 1 2: paddle_left - 1	en_paddle_left s_paddle_left
paddle_right	0: init_paddle_right 1: paddle_right + 1 2: paddle_right - 1	en_paddle_right s_paddle_right
ball_xdir	0: init_dxir 1: ~ball_xdir	en_ball_xdir s_ball_xdir
ball_ydir	0: init_ydir 1: ~ball_ydir	en_ball_ydir s_ball_ydir
ball_xpos	0: init_xpos 1: ball_xpos-1 2: ball_xpos+1	en_ball_xpos s_ball_xpos
ball_ypos	0: init_ypos 1: ball_ypos-1 2: ball_ypos+1	en_ball_ypos s_ball_ypos
ObsMemOut	0: ObsMem(ball_xpos, ball_ypos -1) 1: ObsMem(ball_xpos, ball_ypos +1) 2: ObsMem(ball_xpos-1, ball_ypos) 3: ObsMem(ball_xpos+1, ball_ypos)	s_obs_xy
vga color	0: BLACK 1: BLUE 2: GREEN	s_color plot
xplot	0: ball_xpos 1: paddle_left 2: paddle_right	en_xplot s_xplot
yplot	0: ball_ypos 1: 7'd109	en_yplot s_yplot
score	0: 0 1: score + 1	en_score s_score

Signal Definitions:



Condition	Flag
timer == TIME_LIMIT	timer_done
game_start = 1	game_start
right_button = 1	right_button
left_button = 1	left_button
ball_xdir = RIGHT	ball_xdir
ball_ydir = DOWN	ball_ydir
ObsMemOut == BLUE	wall_obstacle
paddle_left = 8'd5	left_limit
paddle_right = 7'd155	right_limit
ObsMemOut == GREEN	game_over
ObsMemOut == CYAN	paddle_obstacle
ObsMemOut == RED || WHITE || YELLOW	block_obstacle

Controller Design:

State	Actions	Transitions

INIT	ball_xdir <= init_xdir
	ball_ydir <= init_ydir
	ball_xpos <= init_ball_xpos
	ball_ypos <= init_ball_ypos
	paddle_left <= init_paddle_left
	paddle_right <= init_paddle_right
	
	s_paddle_left = 0; en_paddle_left = 1;
	s_paddle_right = 0; en_paddle_right = 1;
	s_ball_xdir = 0; en_ball_xdir = 1;
	s_ball_ydir = 0; en_ball_ydir = 1;
	s_ball_xpos = 0; en_ball_xpos = 1;
	s_ball_ypos = 0; en_ball_ypos = 1;
		if (game_start) goto WAIT_TIMER
		else goto INIT

WAIT_TIMER	timer = timer + 1
	
	s_timer = 1; en_timer = 1;
		if (timer_done)
		if (right_button = 1 && ~right_limit) goto ERASE_LEFT_SIDE
		else if (left_button = 1 && ~left_limit) goto ERASE_RIGHT_SIDE
		else goto ERASE_BALL
		else
		goto WAIT_TIMER

ERASE_LEFT_SIDE	vga(paddle_left, 7'd109) <= BLACK
	
	s_plot = 1; en_plot = 1;
	plot = 1; s_color = 0;
		goto BUFFER_ELS

BUFFER_ELS	same as ELS^	goto MOVE_PADDLE_RIGHT

MOVE_PADDLE_RIGHT	paddle_left <= paddle_left + 1
	paddle_right <= paddle_right+1
	
	s_paddle_left = 1; en_paddle_left = 1;
	s_paddle_right = 1; en_paddle_right = 1;
		goto DRAW_RIGHT_SIDE

DRAW_RIGHT_SIDE	vga(paddle_right, 7'd109) <= BLUE plot = 1; s_color = 1;	goto BUFFER_DRS
BUFFER_DRS	all same as DRS^	goto ERASE_BALL

ERASE_RIGHT_SIDE	vga(paddle_right, 7'd109) <= BLACK
	s_xplot = 2; en_xplot = 1;
	s_yplot = 1; en_yplot = 1
	plot = 1; s_color = 0;
		goto BUFFER_ERS

BUFFER_ERS	all same as ERS^	goto MOVE_PADDLE_LEFT

MOVE_PADDLE_LEFT	paddle_left <= paddle_left - 1
	paddle_right <= paddle_right - 1
	s_paddle_left = 2; en_paddle_left = 1;
	s_paddle_right = 2; en_paddle_right = 1;
		goto DRAW_LEFT_SIDE

DRAW_LEFT_SIDE	vga(paddle_left, 7'd109) <= BLUE
	s_plot = 1; en_plot = 1;
	plot = 1; s_color = 1;
		goto BUFFER_DLS

BUFFER_DLS	same as DLS^	goto ERASE_BALL

ERASE_BALL	vga(ball_xpos, ball_ypos) <= BLACK
	s_plot = 0; en_plot = 1;
	plot = 1; s_color = 0;
		goto BUFFER_EB

BUFFER_EB	same as EB^	if ball_xdir == RIGHT goto LOOK_RIGHT
		else goto LOOK_LEFT

LOOK_RIGHT	ObsMemOut <= ObsMem(ball_xpos + 1, ball_ypos) s_obs_xy = RIGHT;	goto TEST_X_OBSTACLE
LOOK_LEFT	ObsMemOut <= ObsMem(ball_xpos - 1, ball_ypos) s_obs_xy = LEFT	goto TEST_X_OBSTACLE
TEST_X_OBSTACLE		if (wall_obstacle || paddle_obstacle) goto CHANGE_XDIR

		else if (ball_ydir) goto LOOK_DOWN
		else goto LOOK_UP

CHANGE_XDIR	ball_xdir <= ~ball_xdir
	s_ball_xdir = 1; en_ball_xdir = 1;
		if (ball_ydir) goto LOOK_DOWN
		else goto LOOK_UP

LOOK_UP	ObsMemOut <= ObsMem(ball_xpos, ball_ypos -1) s_obs_xy = UP;	goto TEST_Y_OBSTACLE
LOOK_DOWN	ObsMemOut <= ObsMem(ball_xpos, ball_ypos +1) s_obs_xy = DOWN;	goto TEST_Y_OBSTACLE

TEST_Y_OBSTACLE		if (wall_obstacle || paddle_obstacle) goto CHANGE_YDIR
		else if (block_obstacle) goto SCORE_POINT
		else if (game_over) goto GAME_OVER
		else if (ball_xdir) goto INC_X_BALL
		else goto DEC_X_BALL

SCORE_POINT	score <= score + 1;
	s_score = 1; en_score = 1;
		if (ball_xdir) goto INC_X_BALL
		else goto DEC_X_BALL

CHANGE_YDIR	ball_ydir <= ~ball_ydir
	s_ball_ydir = 1; en_ball_ydir = 1;
		if (ball_xdir) goto INC_X_BALL
		else goto DEC_X_BALL

INC_X_BALL	ball_xpos <= ball_xpos + 1
	s_ball_xpos = 2; en_ball_xpos = 1;
		if (ball_ydir) goto INC_Y_BALL
		else goto DEC_Y_BALL

DEC_X_BALL	ball_xpos <= ball_xpos - 1
	s_ball_xpos = 1; en_ball_xpos =1;
		if (ball_ydir) goto INC_Y_BALL
		else goto DEC_Y_BALL

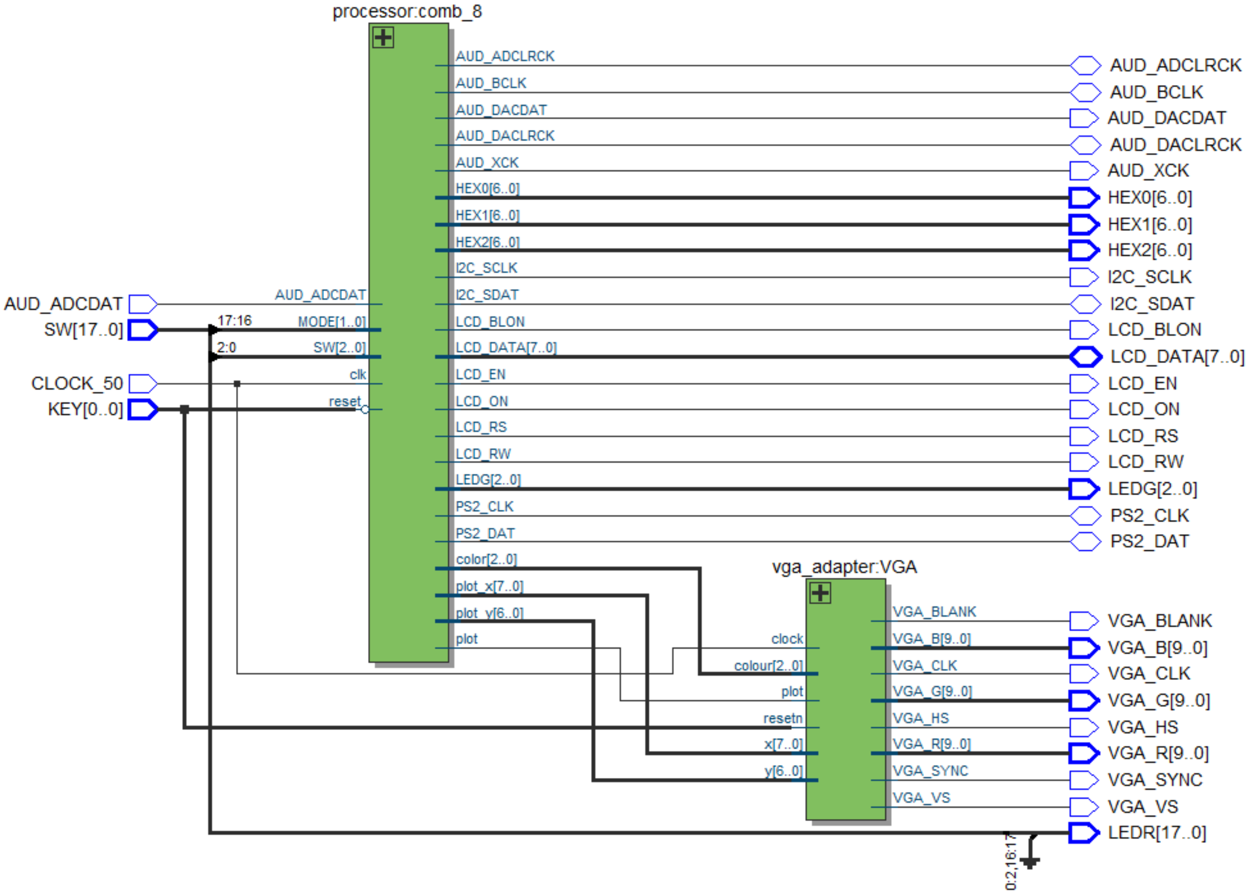
DEC_Y_BALL	ball_ypos <= ball_ypos - 1 s_ball_ypos = 1; en_ball_ypos = 1;	goto DRAW_PADDLE
INC_Y_BALL	ball_ypos <= ball_ypos + 1 s_ball_ypos = 2; en_ball_ypos = 1;	goto DRAW_PADDLE

DRAW_BALL	vga(ball_xpos, ball_ypos) <= GREEN
	s_plot = 0; en_plot = 1;
	s_color = 2; plot = 1;
		goto WAIT_TIMER

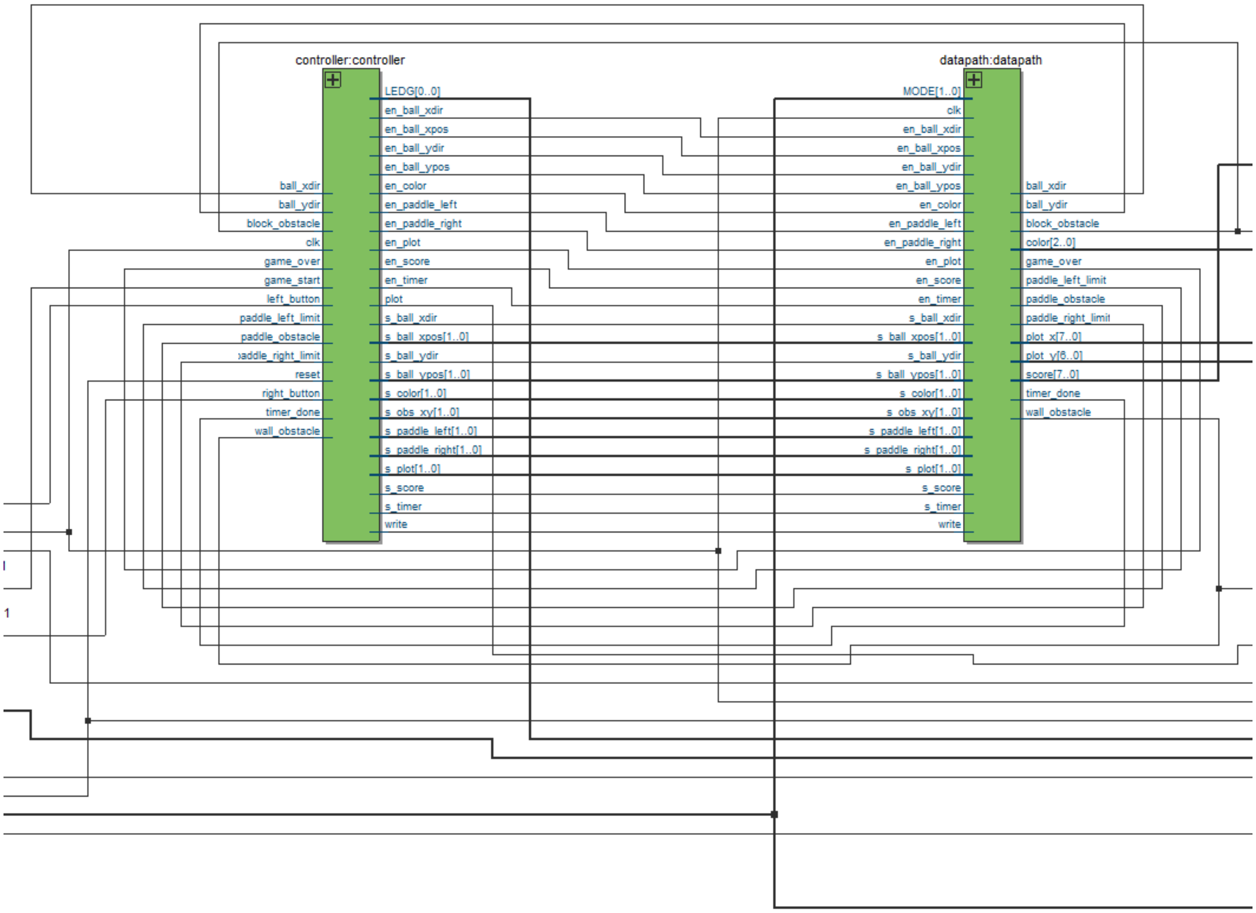
BUFFER_DB	same as DB^	goto WAIT_TIMER
GAME_OVER		goto GAME_OVER

<u>RTL Netlist Views:</u>

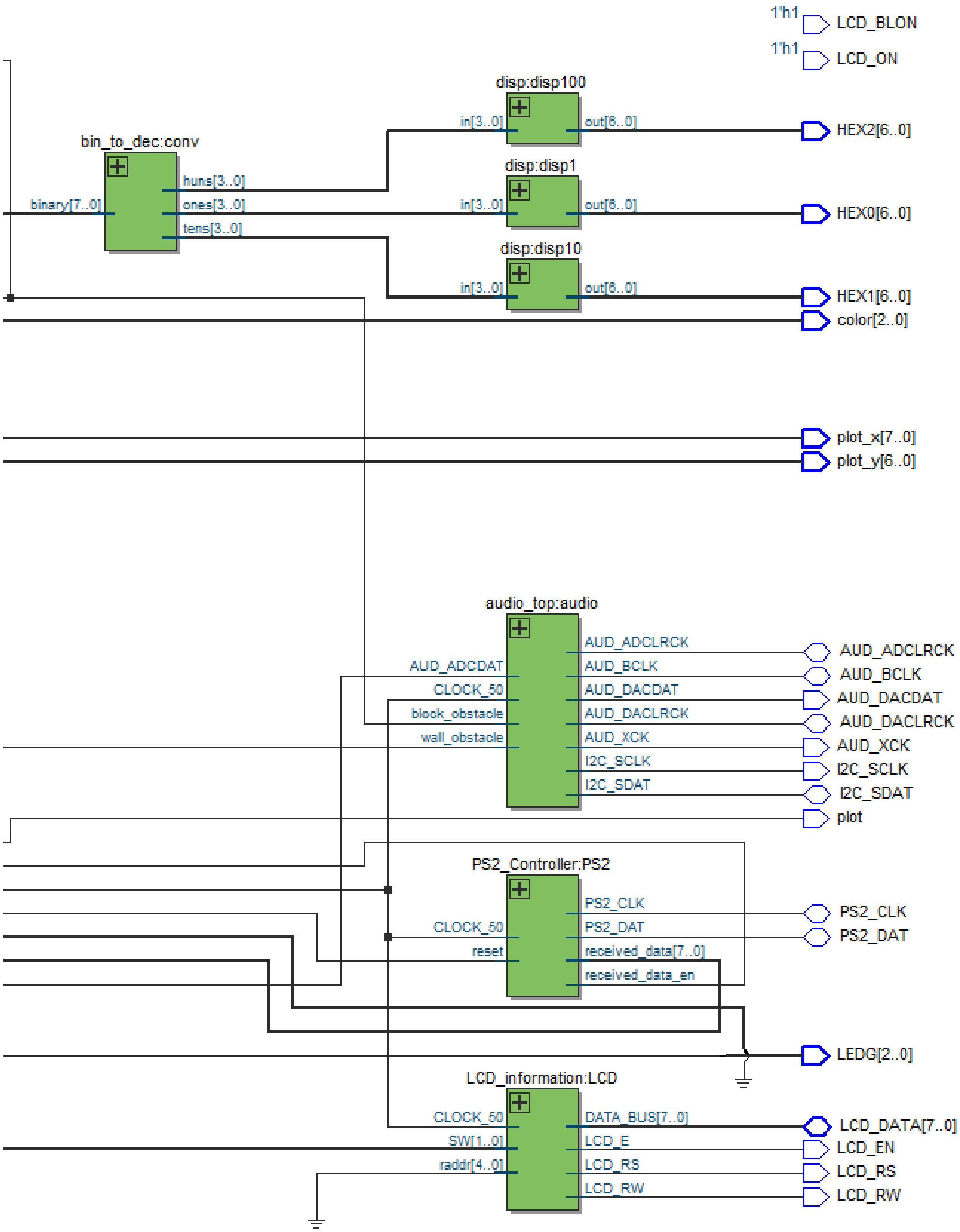
breakout_de2:



processor (just datapath and controller):



processor (custom modules):



Verilog Model

breakout_de2.v

```
module breakout_DE2 (
        input                         CLOCK_50,                            //      50 MHz
        input    [0:0]  KEY,
        input    [17:0] SW,

        output                VGA_CLK,                             //      VGA Clock
        output                VGA_HS,                              //      VGA H_SYNC
        output                VGA_VS,                              //      VGA V_SYNC
        output                VGA_BLANK,                           //      VGA BLANK
        output                VGA_SYNC,                            //      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]
        output [17:0]  LEDR,

        output [2:0]   LEDG,

        // Audio IO
        input       AUD_ADCDAT,
        inout       AUD_BCLK,
        inout       AUD_ADCLRCK,
        inout       AUD_DACLRCK,
        inout       I2C_SDAT,
        output      AUD_XCK,
        output      AUD_DACDAT,
        output      I2C_SCLK,

        // Keyboard stuff
        inout                 PS2_CLK,

        inout                 PS2_DAT,


        // Score to 7 segment hex display
        output [6:0] HEX0,
        output [6:0] HEX1,
        output [6:0] HEX2,

        //LCD Module 16X2
        output                        LCD_ON,        // LCD Power ON/OFF
        output                        LCD_BLON,      // LCD Back Light ON/OFF
        output                        LCD_RW,        // LCD Read/Write Select, 0 = Write, 1 = Read
        output                        LCD_EN,        // LCD Enable
        output                        LCD_RS,        // LCD Command/Data Select, 0 = Command, 1 =
Data
        inout [7:0]           LCD_DATA               // LCD Data bus 8 bits

        );
        assign LEDR[2:0] = SW[2:0];
        assign LEDR[17:16] = SW[17:16];

        wire [2:0]     color;
        wire [7:0]     plot_x;
        wire [6:0]     plot_y;
        wire                  plot;

        processor (
               .clk           (CLOCK_50),
               .reset  (~KEY[0]),
               .SW            (SW[2:0]),
               .MODE          (SW[17:16]),
               //.game_start  (SW[0]),
               //.right_button        (SW[1]),
               //.left_button (SW[2]),

               .plot_x        (plot_x),
               .plot_y        (plot_y),
               .color  (color),
               .plot          (plot),
               .LEDG          (LEDG[2:0]),


               // Audio
               .AUD_ADCDAT            (AUD_ADCDAT),
               .AUD_BCLK              (AUD_BCLK),
               .AUD_ADCLRCK   (AUD_ADCLRCK),
               .AUD_DACLRCK   (AUD_DACLRCK),
               .I2C_SDAT              (I2C_SDAT),
               .AUD_XCK                       (AUD_XCK),
               .AUD_DACDAT            (AUD_DACDAT),
               .I2C_SCLK              (I2C_SCLK),

               // Keyboard
               .PS2_CLK                       (PS2_CLK),
               .PS2_DAT                       (PS2_DAT),

               // score to seven segment hex display
               .HEX0                          (HEX0),
               .HEX1                          (HEX1),
               .HEX2                          (HEX2),

               // LCD Display
               .LCD_ON                (LCD_ON),
               .LCD_BLON              (LCD_BLON),
               .LCD_RW                (LCD_RW),
               .LCD_EN                (LCD_EN),
               .LCD_RS                (LCD_RS),
               .LCD_DATA              (LCD_DATA)

        );

        vga_adapter VGA(
               .resetn(KEY[0]),
               .clock(CLOCK_50),
               .colour(color),
               .x(plot_x),
               .y(plot_y),
               .plot(plot),
               /* 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),
               .VGA_SYNC(VGA_SYNC),
               .VGA_CLK(VGA_CLK)
        );
        defparam VGA.RESOLUTION = "160x120";
        defparam VGA.MONOCHROME = "FALSE";
        defparam VGA.BITS_PER_COLOUR_CHANNEL = 1;
        defparam VGA.BACKGROUND_IMAGE = "breakout_background.mif";


        // score stuff with RAM

endmodule
```

processor.v

```
module processor (
   input             clk,
   input             reset,
        input          [2:0]          SW,
        input          [1:0]          MODE,
        /*input                               game_start,
        input                                 right_button,
        input                                 left_button,
        */
   output      [7:0]          plot_x,
   output      [6:0]          plot_y,
   output      [2:0]          color,
   output            plot,
        output  [2:0]         LEDG,


        // Audio IO
        input      AUD_ADCDAT,
        inout      AUD_BCLK,
        inout      AUD_ADCLRCK,
        inout      AUD_DACLRCK,
        inout      I2C_SDAT,
        output     AUD_XCK,
        output     AUD_DACDAT,
        output     I2C_SCLK,

        // Keyboard IO
        inout                 PS2_CLK,
        inout                 PS2_DAT,

        // Score 7 segment hex display
        output [6:0] HEX0,
        output [6:0] HEX1,
        output [6:0] HEX2,

        //LCD Module 16X2
        output                       LCD_ON,        // LCD Power ON/OFF
        output                       LCD_BLON,      // LCD Back Light ON/OFF
        output                       LCD_RW,        // LCD Read/Write Select, 0 = Write, 1 = Read
        output                       LCD_EN,        // LCD Enable
        output                       LCD_RS,        // LCD Command/Data Select, 0 = Command, 1 =
Data
        inout [7:0]           LCD_DATA              // LCD Data bus 8 bits

   );



        wire                  en_paddle_left;
        wire [1:0]     s_paddle_left;
        wire                  en_paddle_right;
        wire [1:0]     s_paddle_right;
        wire                  en_plot;
        wire [1:0]     s_plot;

        wire                  write;

   wire        en_ball_xpos;
   wire [1:0]  s_ball_xpos;
   wire        en_ball_ypos;
   wire [1:0]  s_ball_ypos;
   wire        en_ball_xdir;
   wire        s_ball_xdir;
   wire        en_ball_ydir;
   wire        s_ball_ydir;
   wire        en_timer;
   wire        s_timer;
        wire                   en_color;
   wire [1:0]  s_color;
   wire [1:0]  s_obs_xy;
   wire        ball_xdir;
   wire        ball_ydir;
   wire        timer_done;
   wire        wall_obstacle;
        wire                   paddle_obstacle;
        wire                   block_obstacle;

        // game over flag
        wire                   game_over;

   controller controller (
      .clk               (clk        ),
      .reset             (reset      ),
               .game_start              (game_start  ),
               .right_button    (right_button),
               .left_button     (left_button),


               .en_paddle_left        (en_paddle_left),
               .s_paddle_left         (s_paddle_left),
               .en_paddle_right       (en_paddle_right),
               .s_paddle_right        (s_paddle_right),
               .en_plot                              (en_plot),
               .s_plot                       (s_plot),

      .en_ball_xpos    (en_ball_xpos    ),
      .s_ball_xpos     (s_ball_xpos     ),
      .en_ball_ypos    (en_ball_ypos    ),
      .s_ball_ypos     (s_ball_ypos     ),
      .en_ball_xdir    (en_ball_xdir    ),
      .s_ball_xdir     (s_ball_xdir     ),
      .en_ball_ydir    (en_ball_ydir    ),
      .s_ball_ydir     (s_ball_ydir     ),
      .en_timer   (en_timer   ),
      .s_timer    (s_timer    ),
               .en_score      (en_score),
               .s_score              (s_score),

               .en_color      (en_color      ),
      .s_color    (s_color    ),
      .s_obs_xy   (s_obs_xy   ),
      .plot       (plot       ),
               .write         (write),

               .LEDG                  (LEDG[0]),


      .ball_xdir       (ball_xdir       ),
      .ball_ydir       (ball_ydir       ),
      .timer_done        (timer_done ),
      .wall_obstacle   (wall_obstacle   ),
               .paddle_obstacle (paddle_obstacle),
               .block_obstacle  (block_obstacle),
               //game over flag
               .game_over               (game_over),

               .paddle_left_limit     (paddle_left_limit),
               .paddle_right_limit    (paddle_right_limit)
   );

   datapath datapath (
      .clk                    (clk        ),
               .write                        (write),
               // new
               .en_paddle_left        (en_paddle_left),
               .s_paddle_left         (s_paddle_left),
               .en_paddle_right       (en_paddle_right),
               .s_paddle_right        (s_paddle_right),
               .en_plot                              (en_plot),
               .s_plot                       (s_plot),

      .en_ball_xpos    (en_ball_xpos    ),
      .s_ball_xpos     (s_ball_xpos     ),
      .en_ball_ypos    (en_ball_ypos    ),
      .s_ball_ypos     (s_ball_ypos     ),
      .en_ball_xdir    (en_ball_xdir    ),
      .s_ball_xdir     (s_ball_xdir     ),
      .en_ball_ydir    (en_ball_ydir    ),
      .s_ball_ydir     (s_ball_ydir     ),
      .en_timer               (en_timer   ),
      .s_timer                (s_timer    ),
               // score
               .en_score                     (en_score),
               .s_score                              (s_score),

               .en_color                     (en_color),
      .s_color                (s_color    ),
      .s_obs_xy               (s_obs_xy   ),
      .ball_xdir       (ball_xdir       ),
      .ball_ydir        (ball_ydir       ),
      .timer_done             (timer_done ),
      .wall_obstacle          (wall_obstacle   ),
               .paddle_obstacle       (paddle_obstacle),
               .block_obstacle        (block_obstacle),
               // game over flag
               .game_over                    (game_over),

               .paddle_left_limit     (paddle_left_limit),
               .paddle_right_limit    (paddle_right_limit),
      .plot_x          (plot_x       ),
      .plot_y                 (plot_y      ),
      .color                  (color      ),
               .score                        (score),
               .MODE                                 (MODE)
   );

        audio_top audio(
               .AUD_ADCDAT            (AUD_ADCDAT),
               .AUD_BCLK              (AUD_BCLK),
               .AUD_ADCLRCK   (AUD_ADCLRCK),
               .AUD_DACLRCK   (AUD_DACLRCK),
               .I2C_SDAT              (I2C_SDAT),
               .AUD_XCK                      (AUD_XCK),
               .AUD_DACDAT            (AUD_DACDAT),
               .I2C_SCLK              (I2C_SCLK),

               .wall_obstacle (wall_obstacle),
               .block_obstacle (block_obstacle),
               .CLOCK_50              (clk)
        );

        // Keyboard stuff
        wire    [7:0] ps2_key_data;
        wire                  ps2_key_en;
        wire                  keycode_ready;
        wire    [7:0] keycode;
        wire                  ext;
   wire                   make;

   // button wires
   wire                   game_start;
   wire                   left_button;
   wire                   right_button;

   assign game_start = ((keycode == 8'h75 && make) || SW[0]);
   assign left_button = ((keycode == 8'h6b && make) || SW[2]);
   assign right_button = ((keycode == 8'h74 && make) || SW[1]);
   assign LEDG[1] = make;

   PS2_Controller PS2(
           .CLOCK_50                     (clk),
           .reset                        (reset),
           .PS2_CLK                             (PS2_CLK),
           .PS2_DAT                             (PS2_DAT),
           .received_data        (ps2_key_data),
           .received_data_en     (ps2_key_en)
   );

   keycode_recognizer key(
           .clk                          (clk),
           .reset_n                      (~reset),
           .ps2_key_en           (ps2_key_en),
           .ps2_key_data  (ps2_key_data),
           .keycode                      (keycode),
           .ext                          (ext),
           .make                         (make),
           .keycode_ready (keycode_ready)
   );



   // score for 7 segment hex display
   wire [7:0]     score;

   wire [3:0]conv_to_disp100;
   wire [3:0]conv_to_disp10;
   wire [3:0]conv_to_disp1;

   bin_to_dec conv (

           .binary        (score),

           .huns                 (conv_to_disp100),

           .tens                 (conv_to_disp10),

           .ones                 (conv_to_disp1)
   );

   //display modules
   disp disp100(
           .in                           (conv_to_disp100),
           //check to see if this is actually the 100's place on the board
           .out                          (HEX2)
   );

   disp disp10(
           .in            (conv_to_disp10),
           .out           (HEX1)
   );

   disp disp1(
           .in            (conv_to_disp1),
                .out            (HEX0)
        );


        // LCD Modules
        assign LCD_BLON = 1'b1;
        assign LCD_ON   = 1'b1;
        LCD_information LCD(
                .CLOCK_50               (clk),

                .LCD_RS                 (LCD_RS),
                .LCD_E                  (LCD_EN),
                .LCD_RW                 (LCD_RW),
                .DATA_BUS               (LCD_DATA),

                .SW                             (MODE),
                .raddr                  (raddr)
        );


endmodule
```

datapath.v

```
module datapath (
        input                                    clk,
        input                                    write,
        // new
        input                                    en_paddle_left,
        input          [1:0]          s_paddle_left,
        input                                    en_paddle_right,
        input          [1:0]          s_paddle_right,
        input                                    en_plot,
        input          [1:0]          s_plot,

   input         en_ball_xpos,
   input    [1:0]    s_ball_xpos,
   input         en_ball_ypos,
   input    [1:0]    s_ball_ypos,
   input         en_ball_xdir,
   input         s_ball_xdir,
   input         en_ball_ydir,
   input         s_ball_ydir,
   input         en_timer,
   input         s_timer,
        //score stuff
        input                                    en_score,
        input                                    s_score,


        // new
        input                                    en_color,
   input    [1:0]    s_color,
   input    [1:0]    s_obs_xy,
   output reg        ball_xdir,
   output reg        ball_ydir,
   output            timer_done,
   output            wall_obstacle,
        output                        paddle_obstacle,
        output                        block_obstacle,

        //game over flag
        output                        game_over,

        output                        paddle_left_limit,
        output                        paddle_right_limit,
        output reg [7:0]      plot_x,
        output reg [6:0]      plot_y,
        output reg [2:0]      color,

        output reg [7:0]  score,

        input     [1:0] MODE
   );

/***************************************************************************
 *                            Parameter Declarations                       *
 ***************************************************************************/
   parameter UP               = 2'd0;
   parameter DOWN             = 2'd1;
   parameter LEFT             = 2'd2;
   parameter RIGHT            = 2'd3;

/***************************************************************************
 *                 Internal Wire and Register Declarations                 *
 ***************************************************************************/
   reg   [25:0]   timer;
```

```
       reg     [7:0]          paddle_left;
       reg     [7:0]          paddle_right;
       reg     [7:0]          ball_xpos;
       reg     [6:0]          ball_ypos;

       wire    [7:0]          xobs;       // x-coordinate to obstacle memory
       wire    [6:0]          yobs;       // y-coordinate to obstacle memory
   wire  [2:0]    dout_obs;   // output of obstacle memory

       reg     [8:0]          initial_xball;


       reg     [7:0]          score_slower;

       reg     [25:0]  TIMER_LIMIT;//         = 26'd1_000_000; // CHANGE BACK TO 26'd1_000_000


/**************************************************************************
 *                            Sequential Logic                            *
 **************************************************************************/

       // Mode of game
       always @(posedge clk)
               case (MODE)
                      0:      TIMER_LIMIT = 26'd1_500_000;
                      1:      TIMER_LIMIT = 26'd1_000_000;
                      2:      TIMER_LIMIT = 26'd750_000;
                      3:      TIMER_LIMIT    = 26'd250_000;
               endcase

       // randomization of ball starting position
       always @(posedge clk)
               if (initial_xball < 8'd65)
                              initial_xball <= initial_xball + 1;
               else
                              initial_xball <= 8'd15;


               //initial_xball <= initial_xball * 2;


       // score stuff
       always @(posedge clk)
               if      (en_score) begin
                      if (s_score) begin
                              score_slower <= score_slower + 1;
                                     if (score_slower > 5) begin
                                             score <= score + 1;
                                             score_slower <= 0;
                                     end
                      end
               else
                      score <= 8'd1;
               end

       // new
       always @(posedge clk)
               if (en_paddle_left)
                      case (s_paddle_left)
                              0: paddle_left <= 8'd71;      //8'd115;
                              1: paddle_left <= paddle_left - 1;
                              2: paddle_left <= paddle_left + 1;
                              default :      paddle_left <= 0;
                      endcase
```

```
     always @(posedge clk)
            if (en_paddle_right)
                   case (s_paddle_right)
                          0: paddle_right <= 8'd89;//   8'd133;
                          1: paddle_right <= paddle_right - 1;
                          2: paddle_right <= paddle_right + 1;
                          default :      paddle_right <= 0;
                   endcase

     always @(posedge clk)
            if      (en_plot)
                   case    (s_plot)
                          0: begin
                                 plot_x <= ball_xpos;
                                 plot_y <= ball_ypos;
                                 end
                          1: begin
                                 plot_x <= paddle_left;
                                 plot_y <= 7'd109;
                                 end
                          2: begin
                                 plot_x <= paddle_right;
                                 plot_y <= 7'd109;
                                 end
                          default :      begin
                                                       plot_x  <= 0;
                                                       plot_y  <= 0;
                                                       end
                   endcase



     always @(posedge clk)
   if (en_ball_xdir)
      if (s_ball_xdir)
         ball_xdir <= ~ball_xdir;
      else
         ball_xdir <= 1;

always @(posedge clk)
   if (en_ball_ydir)
      if (s_ball_ydir)
         ball_ydir <= ~ball_ydir;
      else
         ball_ydir <= 1;

always @(posedge clk)
   if (en_ball_xpos)
      case (s_ball_xpos)
                          0: ball_xpos <= initial_xball * 2;
      //   0: ball_xpos <= 8'd80;
         1: ball_xpos <= ball_xpos - 1;
         2: ball_xpos <= ball_xpos + 1;
         default: ball_xpos <= 0;
      endcase

always @(posedge clk)
   if (en_ball_ypos)
      case (s_ball_ypos)
         0: ball_ypos <= 7'd30;
         1: ball_ypos <= ball_ypos - 1;
         2: ball_ypos <= ball_ypos + 1;
         default: ball_ypos <= 0;
      endcase

always @(posedge clk)
```

```
      if (en_timer)
         if (s_timer)
            timer <= timer + 1;
         else
            timer <= 0;


       always @(posedge clk)
               if (en_color)
                      case (s_color)
                              0: color <= 3'b000;
                              1: color <= 3'b010;
                              2: color <= 3'b001;
                              3: color <= 3'b011;
                      endcase


/***************************************************************************
 *                            Combinational Logic                          *
 ***************************************************************************/

       // obstacle memory coordinate addresses
   assign xobs =
      s_obs_xy == 0 ? ball_xpos :
      s_obs_xy == 1 ? ball_xpos :
      s_obs_xy == 2 ? ball_xpos - 1 :
      ball_xpos + 1;

   assign yobs =
      s_obs_xy == 0 ? ball_ypos - 1 :
      s_obs_xy == 1 ? ball_ypos + 1 :
      ball_ypos;

   /* pixel color to VGA adapter
   assign color      =
               s_color == 0 ? 3'b000 :
               s_color == 1 ? 3'b010 :
               s_color == 2 ? 3'b001 :
               3'b110;
   */
   // flags
   assign timer_done = (timer > TIMER_LIMIT);

   assign wall_obstacle   = (dout_obs == 3'b001 /*|| dout_obs == 3'b010*/);

       assign paddle_obstacle = (dout_obs == 3'b011);

       assign block_obstacle  = (dout_obs == 3'b111 || dout_obs == 3'b110 || dout_obs == 3'b100);

       // game over flag
       assign game_over                = (dout_obs == 3'b010);

       // Outer walls are four pixels thick
       assign paddle_left_limit      = (paddle_left == 8'd5);

       assign paddle_right_limit     = (paddle_right == 8'd154);


 /**************************************************************************
 *                               Internal Modules                          *
 ***************************************************************************/
   image_ram obstacle_mem (
               .clk                                  (clk),
               .x_read                        (xobs),
               .y_read                        (yobs),
               .color_out                     (dout_obs),
               .x_write                             (plot_x),
               .y_write                             (plot_y),
               .color_in                    (color),
               .wren                                (write)
       );
       defparam obstacle_mem.BACKGROUND_IMAGE = "breakout_background_shift.mif";

endmodule
```

controller.v

```
module controller (
        input                           clk,
        input                           reset,
        input                                       game_start,
        input                                       right_button,
        // new
        input                                       left_button,

        output reg                           en_paddle_left,
        output reg       [1:0] s_paddle_left,
        output reg                           en_paddle_right,
        output reg       [1:0] s_paddle_right,
        output reg                           en_plot,
        output reg       [1:0] s_plot,


   output reg          en_ball_xpos,
   output reg    [1:0]  s_ball_xpos,
   output reg          en_ball_ypos,
   output reg    [1:0]  s_ball_ypos,
   output reg          en_ball_xdir,
   output reg          s_ball_xdir,
   output reg          en_ball_ydir,
   output reg          s_ball_ydir,
   output reg          en_timer,
   output reg          s_timer,

        // score stuff
        output reg                           en_score,
        output reg                           s_score,


        output reg                           en_color,
   output reg    [1:0]  s_color,
   output reg    [1:0]  s_obs_xy,
   output reg          plot,
        output reg                           write,

        output reg       [0:0] LEDG,

   input                ball_xdir,
   input                ball_ydir,
   input                timer_done,
   input                wall_obstacle,
        input                                       paddle_obstacle,
        input                                       block_obstacle,

        // game over flag
        input                                       game_over,

        input                                       paddle_left_limit,
        input                                       paddle_right_limit
        );

   parameter UP               = 2'd0;
   parameter DOWN             = 2'd1;
   parameter LEFT             = 2'd2;
   parameter RIGHT            = 2'd3;

   parameter INIT                            = 5'd0;
   parameter WAIT_TIMER                      = 5'd1;
        parameter ERASE_LEFT_SIDE                   = 5'd2;
        // buffer state
        parameter BUFFER_ELS                               = 5'd3;
```

```
    parameter MOVE_PADDLE_RIGHT           = 5'd4;
    parameter DRAW_RIGHT_SIDE                    = 5'd5;
    // buffer
    parameter BUFFER_DRS                                 = 5'd6;

    parameter ERASE_RIGHT_SIDE                   = 5'd7;
    // buffer
    parameter BUFFER_ERS                                 = 5'd8;
    parameter MOVE_PADDLE_LEFT                   = 5'd9;
    parameter DRAW_LEFT_SIDE                     = 5'd10;
    // buffer
    parameter BUFFER_DLS                                 = 5'd11;
    // end of new states
parameter ERASE_BALL                      = 5'd12;
    // buffer
    parameter BUFFER_EB                                  = 5'd13;
parameter LOOK_LEFT                       = 5'd14;
parameter LOOK_RIGHT                      = 5'd15;
parameter TEST_X_OBSTACLE                 = 5'd16;
parameter CHANGE_BALL_XDIR       = 5'd17;
parameter LOOK_UP                         = 5'd18;
parameter LOOK_DOWN                       = 5'd19;
parameter TEST_Y_OBSTACLE                 = 5'd20;
    // score stuff
    parameter SCORE_POINT                        = 5'd21;

parameter CHANGE_BALL_YDIR       = 5'd22;
parameter DECREMENT_XPOS                  = 5'd23;
parameter INCREMENT_XPOS                  = 5'd24;
parameter DECREMENT_YPOS                  = 5'd25;
parameter INCREMENT_YPOS                  = 5'd26;
parameter DRAW_BALL              = 5'd27;
    // buffer
    parameter BUFFER_DB                                  = 5'd28;

    // game over state
    parameter GAME_OVER                                  = 5'd29;

reg [4:0] state, next_state;

always @(posedge clk)
   if (reset)
      state <= INIT;
   else
      state <= next_state;

always @(*) begin
            // new
            en_paddle_left        = 0;
            s_paddle_left                 = 0;
            en_paddle_right               = 0;
            s_paddle_right        = 0;
            en_plot                              = 0;
            s_plot                               = 0;

   en_ball_xpos                   = 0;
   s_ball_xpos                    = 0;
   en_ball_ypos                   = 0;
   s_ball_ypos                    = 0;
   en_ball_xdir                   = 0;
   s_ball_xdir                    = 0;
   en_ball_ydir                   = 0;
   s_ball_ydir                    = 0;
   en_timer                               = 0;
   s_timer                                = 0;
   s_color                                = 0;
```

```
      s_obs_xy                                = 0;
      plot                                    = 0;
               write                                 = 0;
               LEDG[0]                               = 0;

               // score
               en_score                                      = 0;
               s_score                               = 0;

      next_state = INIT;
      case (state)
         INIT              : begin
                       // new
                              s_paddle_left  = 0;  en_paddle_left   = 1;
                              s_paddle_right = 0;    en_paddle_right       = 1;
                              s_score                = 0;    en_score                = 1;
            s_ball_xdir = 0;  en_ball_xdir = 1;
            s_ball_ydir = 0;  en_ball_ydir = 1;
            s_ball_xpos = 0;  en_ball_xpos = 1;
            s_ball_ypos = 0;  en_ball_ypos = 1;
            s_timer = 0; en_timer = 1;

                              if      (game_start)
                              next_state = WAIT_TIMER;
                              else
                              next_state = INIT;
         end
         WAIT_TIMER        : begin
            //s_color = 1; en_color = 1;
            //plot = 1;       write = 1;
            s_timer = 1; en_timer = 1;
            if (timer_done)
                                      if (right_button)             begin

                                              if (paddle_right_limit)
                                                     next_state = ERASE_BALL;
                                              else
                                                     next_state = ERASE_LEFT_SIDE;
                                              end
                                      else if (left_button)  begin

                                              if (paddle_left_limit)
                                                     next_state = ERASE_BALL;
                                              else
                                                     next_state = ERASE_RIGHT_SIDE;
                                              end
                                      else                 begin                 next_state =
ERASE_BALL; end
            else                              next_state = WAIT_TIMER;
         end

                       ERASE_LEFT_SIDE       : begin
                              s_timer = 0; en_timer = 1;

                              // new
                              s_plot = 1;    en_plot = 1;
                              plot = 0;      write = 1;
                              s_color = 0; en_color = 1;// color

                              next_state = BUFFER_ELS;
                       end

                       BUFFER_ELS                    : begin
                              plot = 1; write = 1;
                              s_plot = 1;    en_plot = 1;
                              s_color = 0; en_color = 1;
```

```
        next_state = MOVE_PADDLE_RIGHT;

end


MOVE_PADDLE_RIGHT      : begin



        // new
        s_paddle_left = 2; en_paddle_left = 1;
        s_paddle_right = 2;   en_paddle_right = 1;

        next_state = DRAW_RIGHT_SIDE;

end

DRAW_RIGHT_SIDE        : begin
        // new
        plot = 0;      write = 1;
        s_color = 3; en_color = 1;// color
        s_plot = 2;    en_plot = 1;

        next_state = BUFFER_DRS;
end

BUFFER_DRS                    : begin
        plot = 1; write = 1;
        s_plot = 2;    en_plot = 1;
        s_color = 3; en_color = 1;
        next_state = ERASE_BALL;
end

ERASE_RIGHT_SIDE       : begin
        s_timer = 0; en_timer = 1;

        // new
        s_plot = 2;    en_plot = 1;
        plot = 0;      write = 1;
        s_color = 0; en_color = 1;// color

        next_state = BUFFER_ERS;
end

BUFFER_ERS                    : begin
        plot = 1; write = 1;
        s_plot = 2;    en_plot = 1;
        next_state = MOVE_PADDLE_LEFT;
        s_color = 0; en_color = 1;
end

MOVE_PADDLE_LEFT       : begin

        // new
        s_paddle_left = 1; en_paddle_left = 1;
        s_paddle_right = 1;   en_paddle_right = 1;

        next_state = DRAW_LEFT_SIDE;

end

DRAW_LEFT_SIDE : begin
        // new
        plot = 0;      write = 1;
        s_color = 3; en_color = 1;// color
        s_plot = 1;    en_plot =1;
```

```
                     next_state = BUFFER_DLS;
              end

              BUFFER_DLS            : begin
                     plot = 1; write = 1;
                     s_plot = 1;    en_plot =1;
                     s_color = 3; en_color = 1;
                     next_state = ERASE_BALL;
              end


ERASE_BALL             : begin
   plot = 0;  write = 1;
                     s_color = 0;   en_color = 1;
   s_timer = 0; en_timer = 1;
                     // new
                     s_plot = 0;    en_plot = 1;

                     next_state = BUFFER_EB;

end

              BUFFER_EB                            : begin
                     plot = 1; write = 1;

                     if (ball_xdir)         next_state = LOOK_RIGHT;
   else                next_state = LOOK_LEFT;
              end

LOOK_LEFT         : begin
   s_obs_xy = LEFT;
   next_state = TEST_X_OBSTACLE;
end
LOOK_RIGHT        : begin
   s_obs_xy = RIGHT;
   next_state = TEST_X_OBSTACLE;
end
TEST_X_OBSTACLE   : begin
   if (wall_obstacle || paddle_obstacle)     next_state = CHANGE_BALL_XDIR;
                     else if (block_obstacle)              next_state = SCORE_POINT;
   else if (ball_ydir)    next_state = LOOK_DOWN;
   else              next_state = LOOK_UP;
end
CHANGE_BALL_XDIR       : begin
   s_ball_xdir = 1; en_ball_xdir = 1;
   if (ball_ydir)         next_state = LOOK_DOWN;
   else              next_state = LOOK_UP;
end
LOOK_UP           : begin
   s_obs_xy = UP;
   next_state = TEST_Y_OBSTACLE;
end
LOOK_DOWN         : begin
   s_obs_xy = DOWN;
   next_state = TEST_Y_OBSTACLE;
end
TEST_Y_OBSTACLE   : begin
   if (wall_obstacle || paddle_obstacle)     next_state = CHANGE_BALL_YDIR;
                     //else if (block_obstacle)            next_state = SCORE_POINT;

                     // game over transition!!
                     else if (game_over)      next_state = GAME_OVER;

   else if (ball_xdir)    next_state = INCREMENT_XPOS;
   else              next_state = DECREMENT_XPOS;
```

```
         end

                      SCORE_POINT                   : begin
                                     s_score = 1;   en_score = 1;

                                     if (ball_xdir)    next_state = INCREMENT_XPOS;
                                     else                                 next_state = 
DECREMENT_XPOS;
                      end


         CHANGE_BALL_YDIR       : begin
            s_ball_ydir = 1; en_ball_ydir = 1;
            if (ball_xdir)         next_state = INCREMENT_XPOS;
            else              next_state = DECREMENT_XPOS;
         end
         DECREMENT_XPOS    : begin
            s_ball_xpos = 1; en_ball_xpos = 1;
            if (ball_ydir)         next_state = INCREMENT_YPOS;
            else              next_state = DECREMENT_YPOS;
         end
         INCREMENT_XPOS    : begin
            s_ball_xpos = 2; en_ball_xpos = 1;
            if (ball_ydir)         next_state = INCREMENT_YPOS;
            else              next_state = DECREMENT_YPOS;
         end
         DECREMENT_YPOS    : begin
            s_ball_ypos = 1; en_ball_ypos = 1;
            next_state = DRAW_BALL;
         end
         INCREMENT_YPOS    : begin
            s_ball_ypos = 2; en_ball_ypos = 1;
            next_state = DRAW_BALL;
         end
         DRAW_BALL              : begin
            s_color = 1; en_color = 1;
                              plot = 0;      write = 1;
                              s_plot = 0;    en_plot = 1;

            next_state = BUFFER_DB;
         end


                      BUFFER_DB                            : begin
                              plot = 1; write = 1;

                              next_state = WAIT_TIMER;

                      end

                      GAME_OVER                            : begin



                              LEDG[0] = 1;
                              next_state = GAME_OVER;
                      end

         default           :;
      endcase
   end


endmodule
```

bin_to_dec.v

```
module bin_to_dec (
        input [7:0] binary,
        output reg [3:0] huns,
        output reg [3:0] tens,
        output reg [3:0] ones
        );

        // Concept of decimal to BCD converter taken from:
        // http://www.eng.utah.edu/~nmcdonal/Tutorials/BCDTutorial/BCDConversion.html

        integer count;

        always @(binary)
                begin
                        // zero out each binary representation of the decimals
                        huns = 4'd0;
                        tens = 4'd0;
                        ones = 4'd0;

                for ( count = 7; count >=0; count = count-1)
                        begin   // first always check to see if three needs to be added to any of
the columns
                                if (huns >= 5)
                                        huns = huns + 3;
                                if (tens >= 5)
                                        tens = tens + 3;
                                if (ones >= 5)
                                        ones = ones + 3;

                                // now shift everything to the left
                                huns = huns << 1;
                                huns[0] = tens[3];
                                tens = tens << 1;
                                tens[0] = ones[3];
                                ones = ones << 1;
                                ones[0] = binary[count];
                        end
                end
endmodule
```

audio_top.v

```
module audio_top(
        // Audio Items
        input       AUD_ADCDAT,
        inout       AUD_BCLK,
        inout       AUD_ADCLRCK,
        inout       AUD_DACLRCK,
        inout       I2C_SDAT,
        output      AUD_XCK,
        output      AUD_DACDAT,
        output      I2C_SCLK,
        // End of Audio Items

        input wall_obstacle,
        input block_obstacle,
        input CLOCK_50
        //input reset_beep
        );

        /******* Audio Items *******/
        wire                            audio_out_allowed;
   wire     [31:0]   osc_out;


        // timer stuff
        reg [31:0] beepCount = 0;
        reg              beep_obs     = 0;
        always@ (posedge(CLOCK_50)) begin

                if(wall_obstacle || block_obstacle) begin
                        beepCount <= 0;
                        beep_obs <= 1;
                end
                else begin
                        beepCount <= beepCount + 1;
                end

                if(beepCount > 10000000) begin
                        beep_obs <= 0;
                        beepCount <= 0;
                end

        end


        /******* Audio Module Initialization *******/
        square_wave_osc osc (
      .CLOCK_50                                      (CLOCK_50),
      .reset                                            (~beep_obs),
                .wall_obstacle                               (wall_obstacle),
                .block_obstacle                              (block_obstacle),
      .out                       (osc_out)
   );

        Audio_Controller Audio_Controller (
      // Inputs
      .CLOCK_50                                      (CLOCK_50),
      .reset                                            (~beep_obs),
      .left_channel_audio_out        (osc_out),
      .right_channel_audio_out(osc_out),
      .write_audio_out                  (audio_out_allowed),
      .AUD_ADCDAT                                       (AUD_ADCDAT),
      // Bidirectionals
      .AUD_BCLK                                 (AUD_BCLK),
      .AUD_ADCLRCK                              (AUD_ADCLRCK),
      .AUD_DACLRCK                              (AUD_DACLRCK),
      // Outputs
      .audio_out_allowed                     (audio_out_allowed),
      .AUD_XCK                                     (AUD_XCK),
      .AUD_DACDAT                                       (AUD_DACDAT)
   );

   avconf avc (
      .I2C_SCLK                              (I2C_SCLK),
      .I2C_SDAT                              (I2C_SDAT),
      .CLOCK_50                              (CLOCK_50),
      .reset                                         (~beep_obs)
   );
endmodule
```

LCD_message.v

```
module LCD_message (
        input                  [1:0] SW,
        input                       [4:0] raddr,
        output reg      [7:0]   dout
        );

   always @(raddr, SW)
               case (SW[1:0])

                       0: begin
                              case(raddr)
                                      0: dout = "M";
                                      1: dout = "o";
                                      2: dout = "d";
                                      3: dout = "e";
                                      4: dout = ":";
                                      16: dout = "E";
                                      17: dout = "a";
                                      18: dout = "s";
                                      19: dout = "y";
                                      default: dout = " ";
                              endcase
                       end

                       1: begin
                              case(raddr)
                                      0: dout = "M";
                                      1: dout = "o";
                                      2: dout = "d";
                                      3: dout = "e";
                                      4: dout = ":";
                                      16: dout = "M";
                                      17: dout = "e";
                                      18: dout = "d";
                                      19: dout = "i";
                                      20: dout = "u";
                                      21: dout = "m";
                                      default: dout = " ";
                              endcase
                       end

                       2: begin
                              case(raddr)
                                      0: dout = "M";
                                      1: dout = "o";
                                      2: dout = "d";
                                      3: dout = "e";
                                      4: dout = ":";
                                      16: dout = "H";
                                      17: dout = "a";
                                      18: dout = "r";
                                      19: dout = "d";
                                      default: dout = " ";
                              endcase
                       end

                       default: begin
                              case(raddr)
                                      0: dout = "M";
                                      1: dout = "o";
                                      2: dout = "d";
                                      3: dout = "e";
                                      4: dout = ":";
                                      16: dout = "E";
```

```
                    17: dout = "X";
                    18: dout = "T";
                    19: dout = "R";
                    20: dout = "E";
                    21: dout = "M";
                    22: dout = "E";
                    default: dout = " ";
                endcase
            end
        endcase

endmodule
```

LCD_information.v

```
module LCD_information(
        // LCD Display Inputs and outputs
        // host side
        input                 CLOCK_50,
        // LCD side
        output                LCD_RS,
        output                LCD_E,
        output                LCD_RW,
        inout   [7:0] DATA_BUS,

        // LCD Message inputs and outputs
        input             [1:0] SW,
        input                 [4:0] raddr
        );

        wire DLY_RST;
        wire [4:0] disp_addr;
        wire [7:0] disp_data;

        Reset_Delay r0 (
      .iCLK            (CLOCK_50),
      .oRESET  (DLY_RST)
        );

        LCD_message lm(
                .SW            (SW),
                .raddr  (disp_addr),
                .dout          (disp_data)
        );

        LCD_Display u1 (
                // Host Side
                .iCLK_50MHZ    (CLOCK_50),
                .iRST_N        (DLY_RST),
                .oMSG_INDEX    (disp_addr),
                .iMSG_ASCII    (disp_data),
                // LCD Side
                .DATA_BUS      (DATA_BUS),
                .LCD_RW        (LCD_RW),
                .LCD_E         (LCD_E),
                .LCD_RS        (LCD_RS)
        );

endmodule
```