Lab Code [0 points]
Filename: Ball.sv

```
module Ball
    (input logic clock, reset, update,
    input logic [8:0] left_paddle_top, right_paddle_top,
    input logic [8:0] row,
    input logic [9:0] col,
    output logic left_scored, right_scored,
    output logic display);

     logic x_dir;
     logic [1:0] y_dir;
     logic [9:0] left, top;

     always_ff@(posedge clock) begin
        if (reset) begin
            left <= 320 - 2;
            top <= 240 - 2;
            if (left_scored)
                x_dir <= 0;
            else if (right_scored)
                x_dir <= 1;
            y_dir <= 2;
            right_scored <= 0;
            left_scored <= 0;
        end
        else if (update) begin
            // hits left paddle
            if (x_dir &
                top >= left_paddle_top & top < left_paddle_top + 48 &
                left >= 60 & left < 64) begin
                y_dir <= (top >= left_paddle_top + 16) +
                        (top >= left_paddle_top + 32);
                x_dir <= 0;
            end

            // hits right paddle
            else if (~x_dir &
                top >= right_paddle_top & top < right_paddle_top + 48 &
                left + 4 >= 577 & left + 4 < 581) begin
                y_dir <= (top >= right_paddle_top + 16) +
                        (top >= right_paddle_top + 32);
                x_dir <= 1;
            end

            // hits top
            else if (y_dir == 0 & top <= 0)
                y_dir <= 2;

            // hits bottom
            else if (y_dir == 2 & top + 4 > 480)
                y_dir <= 0;

            // right score
            else if (left <= 4) begin
                right_scored <= 1;
            end

            // left score
            else if (left > 632) begin
                left_scored <= 1;
            end

            // normal
            else begin
                if (x_dir)
                    left <= left - 2;
                else
                    left <= left + 2;

                if (y_dir == 2)
                    top <= top + 1;
                else if (y_dir == 0)
                    top <= top - 1;
            end

        end

    end

    assign display = (
        col >= left &
        col < left + 4 &
        row >= top &
        row < top + 4
    );

endmodule: Ball
```

Lab Code [0 points]
Filename: ChipInterface.sv

```
`default_nettype none

module RangeCheck
    #(parameter WIDTH = 8)
    (input logic [WIDTH-1:0] val,high,low,
    output logic is_between);

    assign is_between = (val<=high) && (val>=low);

endmodule : RangeCheck


module RangeCheck_test;
    logic [7:0] val,high,low;
    logic is_between;
    RangeCheck #(8) dut (.*);
    initial begin
        $monitor("val:%b high:%b low:%b is_between:%b",
                val, high, low, is_between);
        val = 8'd10;
        high = 8'd20;
        low = 8'd5;
        #10
        high = 8'd7;
        #10
        high = 8'd20;
        low = 8'd15;
        #10
        $finish;
    end
endmodule: RangeCheck_test



module OffsetCheck
    #(parameter WIDTH = 8)
    (input logic [WIDTH-1:0] val,delta,low,
    output logic is_between);

    assign is_between = (val<=(low+delta)) && (val>=low);

endmodule: OffsetCheck


module OffsetCheck_test;
    logic [7:0] val, delta, low;
    logic is_between;
    OffsetCheck #(8) dut (.*);
    initial begin
        $monitor("val:%b delta:%b low:%b is_between:%b",
                val, delta, low, is_between);
        val = 8'd5;
        low = 8'd1;
        delta = 8'd9;
        #10
        delta = 8'd2;
        #10
        low = 8'd5;
        #10
        low = 8'd6;
        #10
        $finish;
    end
endmodule: OffsetCheck_test


module vga
    (input logic CLOCK_50, reset,
    output logic HS, VS, blank,
    output logic [8:0] row,
    output logic [9:0] col);

    logic blank_en; //enable/disable the clock
    logic [15:0] clock_count; //how many clocks

    logic pulse_width;
    // logic back_porch;
    logic display;
    // logic front_porch;

    logic colcountclear_n;
    logic cc_en;

    logic [19:0] line_counter;
    logic line_counter_reset_n;

    assign blank = ~(display & blank_en);
    assign HS = ~pulse_width;

    OffsetCheck #(16) ocpw (clock_count,191,0, pulse_width);

    // OffsetCheck #(16) ocbp (clock_count, 96, 192, back_porch);

    OffsetCheck #(16) ocd (clock_count, 1279, 288, display);

    // OffsetCheck #(16) ocfp (clock_count, 32, 1568, front_porch);

    //if clock passes one timeline, we reset the column count
    RangeCheck #(16) colrange (clock_count, 1598, 0, colcountclear_n);

    //incrementing clock
    Counter #(16) c ( , 1, ~colcountclear_n | reset, 0,
                    CLOCK_50, 1'b1, clock_count);

    //checking col range and incrementing col
    assign cc_en = display && clock_count[0];
    Counter #(16) col_count ( , cc_en, ~colcountclear_n | reset,
                                0, CLOCK_50, 1'b1, col);

    Counter #(16) row_count (, ~colcountclear_n & blank_en,
                ~line_counter_reset_n | reset, 0, CLOCK_50, 1, row);

    OffsetCheck #(20) VS_check (line_counter, 900000, 3200, VS);

     OffsetCheck #(20) clock_check (line_counter, 817399, 49600, blank_en);

    // define line counter
    RangeCheck #(20) line_counter_check (line_counter,
                    833598, 0, line_counter_reset_n);
    Counter #(20) (, 1, ~line_counter_reset_n | reset, 0,
                    CLOCK_50, 1, line_counter);

endmodule : vga

/*
module vga_test();

    logic clock, reset, HS, VS, blank;
    logic [8:0] row;
    logic [9:0] col;
    logic [31:0] i;

    vga v (clock, reset, HS, VS, blank, row, col);

    initial begin
        // $monitor("%d, clock: %d, row: %d, col: %d, blank: %d, reset: %d",
            $time, clock, row, col, blank, reset);
        clock = 1;
        #1800000
        reset = 1;
        #5
        reset = 0;
        $finish;
    end
    initial begin
        reset = 1;
        @(posedge clock);
        @(posedge clock);
        reset = 0;
        @(posedge clock);
        @(posedge clock);
        @(posedge clock);
        @(posedge clock);
        @(posedge clock);
        @(posedge clock);
        @(posedge clock);
        @(posedge clock);
        @(posedge clock);
        @(posedge clock);
    end
    always
        #1 clock = ~clock;

endmodule: vga_test;
*/

module ChipInterface
    (input logic CLOCK_50,
    input logic [3:0] KEY,
    input logic [17:0] SW,
    output logic [6:0] HEX0, HEX1, HEX2, HEX3,
                       HEX4, HEX5, HEX6, HEX7,
    output logic [7:0] VGA_R, VGA_G, VGA_B,
    output logic VGA_BLANK_N, VGA_CLK, VGA_SYNC_N,
    output logic VGA_VS, VGA_HS);

    logic [8:0] row;
    logic [9:0] col;

    logic serve, reset, update, left_scored, right_scored,
        ball_display,
        left_paddle_display, right_paddle_display,
        left_score_display, right_score_display,
        left_scored_display, right_scored_display,
        left_win, right_win;

    logic [9:0] left_paddle_top, right_paddle_top, reset_counter;
    logic [9:0] left_score_counter, right_score_counter;

    always_ff@(posedge CLOCK_50) begin

        update <= (col == 639) & (row == 479) & (~update);

        if (~KEY[0]) begin
            reset <= 1;
            serve <= 1;
            reset_counter <= 0;
        end

        else if (reset_counter < 8) begin
            reset <= 1;
            serve <= 1;
            reset_counter <= reset_counter + 1;
            left_score_counter <= 20;
            right_score_counter <= 20;
        end

        else if (left_score_counter < 40 | left_win) begin
            if (update)
                left_score_counter <= left_score_counter + 1;
            left_scored_display <= ~left_score_counter[3];
        end

        else if (right_score_counter < 40 | right_win) begin
            if (update)
                right_score_counter <= right_score_counter + 1;
            right_scored_display <= ~right_score_counter[3];
        end

        else begin
            reset <= 0;
            left_scored_display <= 0;
            right_scored_display <= 0;

            if (left_scored) begin
                serve <= 1;
                left_score_counter <= 0;
            end

            else if (right_scored) begin
                serve <= 1;
                right_score_counter <= 0;
            end

            else if (~KEY[3]) begin
                serve <= 0;
            end
        end
    end

    Ball b(
        .clock(CLOCK_50),
        .reset(serve | reset),
        .update(update),
        .left_paddle_top(left_paddle_top),
        .right_paddle_top(right_paddle_top),
        .row(row),
        .col(col),
        .left_scored(left_scored),
        .right_scored(right_scored),
        .display(ball_display)
     );

    Paddle left_paddle(
        .clock(CLOCK_50),
        .reset(reset),
        .input_up(SW[17]),
        .input_down(SW[16]),
        .left(10'd60),
        .update(update),
        .row(row),
        .col(col),
        .top(left_paddle_top),
        .display(left_paddle_display)
        );

    Paddle right_paddle(
        .clock(CLOCK_50),
        .reset(reset),
        .input_up(SW[1]),
        .input_down(SW[0]),
        .left(10'd577),
        .update(update),
        .row(row),
        .col(col),
        .top(right_paddle_top),
        .display(right_paddle_display)
    );

   Score left_score(
        .clock(CLOCK_50),
        .reset(reset),
        .scored(left_scored),
        .row(row),
        .col(col),
        .top(16),
        .left(280),
          .win(left_win),
        .display(left_score_display),
        .segs(HEX6)
    );

     Score right_score(
        .clock(CLOCK_50),
        .reset(reset),
        .scored(right_scored),
        .row(row),
        .col(col),
        .top(16),
        .left(328),
          .win(right_win),
        .display(right_score_display),
        .segs(HEX4)
    );

     Color color(
        .left(left_paddle_display | left_score_display |
                        left_scored_display),
        .right(right_paddle_display | right_score_display |
                        right_scored_display),
        .ball(ball_display),
        .R(VGA_R),
        .G(VGA_G),
        .B(VGA_B)
     );

    logic blank;
    assign VGA_BLANK_N = ~blank;
    vga dut(CLOCK_50, 0, VGA_HS, VGA_VS, blank, row, col);

    assign HEX7 = 7'b1111111;
     assign HEX5 = 7'b1111111;
     assign HEX3 = 7'b1111111;
     assign HEX2 = 7'b1111111;
     assign HEX1 = 7'b1111111;
     assign HEX0 = 7'b1111111;
    assign VGA_SYNC_N = 1;
    assign VGA_CLK = ~CLOCK_50;
endmodule: ChipInterface
```

Lab Code [0 points]
Filename: Color.sv

```
`default_nettype none

module Color(
    input  logic left, right, ball,
    output logic [7:0] R, G, B);

    always_comb begin

        if (left) begin
            R = 255;
            G = 255;
            B = 0;
        end

        else if (right) begin
            R = 0;
            G = 255;
            B = 255;
        end

        else if (ball) begin
            R = 255;
            G = 255;
            B = 255;
        end

        else begin
            R = 0;
            G = 0;
            B = 0;
        end

    end

endmodule: Color
```

Lab Code [0 points]
Filename: Paddle.sv

```
`default_nettype none

module Paddle(
    input  logic clock, reset,
    input  logic input_up, input_down,
    input  logic [9:0] left,
    input  logic update,
    input  logic [9:0] row, col,
    output logic [9:0] top,
    output logic display);

    logic up, down;

    always_ff@(posedge clock) begin
        up <= input_up;
        down <= input_down;

        if (reset)
            top <= 240 - 24;
        else if (update & up & ~down & (top < (480 - 48)))
            top <= top + 4;
        else if (update & down & ~up & (top >= 4))
            top <= top - 4;
    end

    assign display = (
        col >= left &
        col < left + 4 &
        row >= top &
        row < top + 48
    );

endmodule: Paddle
```

Lab Code [0 points]
Filename: Score.sv

```
`default_nettype none

module Score(
    input  logic clock, reset,
    input  logic scored,
    input  logic [9:0] row, col, top, left,
    output logic [3:0] score,
    output logic win, display,
    output logic [6:0] segs);

    logic [4:0] scoreX2;

    assign score = scoreX2[4:1];

    always_ff@(posedge clock) begin
        if (reset) begin
            win <= 0;
            scoreX2 <= 0;
        end
        else if (score == 9)
            win <= 1;
        else if (scored)
            scoreX2 <= scoreX2 + 1;
    end

    /*
     ___s0___
    |        |
    |s5      |s1
    |        |
     ___s6___
    |        |
    |s4      |s2
    |        |
     ___s3___


    */

    logic [9:0] rx, ry;
    logic s0, s1, s2, s3, s4, s5, s6, s7;
    always_comb begin
        rx = col - left;
        ry = row - top;


        s0 = rx >=  0 & rx < 32 & ry >=  0 & ry <  4;
        s1 = rx >= 28 & rx < 32 & ry >=  0 & ry < 24;
        s2 = rx >= 28 & rx < 32 & ry >= 24 & ry < 48;
        s3 = rx >=  0 & rx < 32 & ry >= 44 & ry < 48;
        s4 = rx >=  0 & rx <  4 & ry >= 24 & ry < 48;
        s5 = rx >=  0 & rx <  4 & ry >=  0 & ry < 24;
        s6 = rx >=  0 & rx < 32 & ry >= 22 & ry < 26;

        if (score == 0) begin
            display = s0 | s1 | s2 | s3 | s4 | s5;
            segs = {1'b1, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0}; end
        else if (score == 1) begin
            display = s1 | s2;
            segs = {1'b1, 1'b1, 1'b1, 1'b1, 1'b0, 1'b0, 1'b1}; end
        else if (score == 2) begin
            display = s0 | s1 | s3 | s4 | s6;
            segs = {1'b0, 1'b1, 1'b0, 1'b0, 1'b1, 1'b0, 1'b0}; end
        else if (score == 3) begin
            display = s0 | s1 | s2 | s3 | s6;
            segs = {1'b0, 1'b1, 1'b1, 1'b0, 1'b0, 1'b0, 1'b0}; end
        else if (score == 4) begin
            display = s1 | s2 | s5 | s6;
            segs = {1'b0, 1'b0, 1'b1, 1'b1, 1'b0, 1'b0, 1'b1}; end
        else if (score == 5) begin
```

```
            display = s0 | s2 | s3 | s5 | s6;
            segs = {1'b0, 1'b0, 1'b1, 1'b0, 1'b0, 1'b1, 1'b0}; end
        else if (score == 6) begin
            display = s0 | s2 | s3 | s4 | s5 | s6;
            segs = {1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b1, 1'b0}; end
        else if (score == 7) begin
            display = s0 | s1 | s2;
            segs = {1'b1, 1'b1, 1'b1, 1'b1, 1'b0, 1'b0, 1'b0}; end
        else if (score == 8) begin
            display = s0 | s1 | s2 | s3 | s4 | s5 | s6;
            segs = {1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0}; end
        else if (score == 9) begin
            display = s0 | s1 | s2 | s3 | s5 | s6;
            segs = {1'b0, 1'b0, 1'b1, 1'b0, 1'b0, 1'b0, 1'b0}; end
        else begin
            display = 0;
            segs = {1'b1, 1'b1, 1'b1, 1'b1, 1'b1, 1'b1, 1'b1}; end
    end

endmodule: Score
```

Lab Code [0 points]
Filename: library.sv

```
`default_nettype none

module MagComp
  #(parameter   WIDTH = 8)
  (output logic             AltB, AeqB, AgtB,
   input  logic [WIDTH-1:0] A, B);

  assign AeqB = (A == B);
  assign AltB = (A <  B);
  assign AgtB = (A >  B);

endmodule: MagComp

// module MagComp_test;

//   logic AltB, AeqB, AgtB;
//   logic [1:0] A, B;
//   logic [3:0] vector;

//   assign {A, B} = vector;

//   MagComp #(2) dut(.*);

//   initial begin
//     $monitor("A:%b B:%b ->> AltB(%b) AeqB(%b) AgtB(%b)",
       //A, B, AltB, AeqB, AgtB);
//     for (vector = 4'b0; vector != 4'b1111; vector++)
//       #1;
//     #1
//     $finish;
//   end
// endmodule : MagComp_test

module Adder
  #(parameter WIDTH=8)
  (input  logic [WIDTH-1:0] A, B,
   input  logic             Cin,
   output logic [WIDTH-1:0] S,
   output logic             Cout);

   assign {Cout, S} = A + B + Cin;

endmodule : Adder

// module Adder_test;

//   logic [3:0] A, B;
//   logic       Cin;
//   logic [3:0] S;
//   logic       Cout;

//   logic [8:0] vector;
//   assign {Cin, A, B} = vector;

//   Adder #(4) dut(.*);

//   initial begin
//     $monitor("Cin:%b A:%b B:%b ->> Cout:%b S:%b", Cin, A, B, Cout, S);
//     for (vector = 9'b0; vector != 9'b1_1111_1111; vector++)
//       #1;
//     #1;
//     $finish;
//   end

// endmodule : Adder_test

module Multiplexer
  #(parameter WIDTH=8)
  (input  logic [WIDTH-1:0]         I,
   input  logic [$clog2(WIDTH)-1:0] S,
   output logic                     Y);

   assign Y = I[S];

endmodule : Multiplexer

// module Multiplexer_test;

//   logic [7:0] I;
//   logic [2:0] S;
//   logic       Y;

//   Multiplexer dut(.*);

//   initial begin
//     $monitor("I(%b), Sel(%b) --> Y(%b)", I, S, Y);
//     I = 8'b1011_0011;
//     for (S=3'b000; S != 3'b111; S++)
//       #1;
//     #1;
//     $finish;
//   end

// endmodule : Multiplexer_test

module Mux2to1
  #(parameter WIDTH = 8)
  (input  logic [WIDTH-1:0] I0, I1,
   input  logic             S,
   output logic [WIDTH-1:0] Y);

  assign Y = (S) ? I1 : I0;

endmodule : Mux2to1

// module Mux2to1_test;

//   logic [1:0] I0, I1;
//   logic       S;
//   logic [1:0] Y;

//   logic [4:0] vector;
//   assign {S, I1, I0} = vector;

//   Mux2to1 #(2) dut(.*);

//   initial begin
//     $monitor("Sel(%b) I1(%h) I0(%h) -> Y(%h)", S, I1, I0, Y);
//     for(vector = 5'b0; vector != 5'b11111; vector++)
//       #1;
//     #1;
//     $finish;
//   end

// endmodule : Mux2to1_test

module Decoder
  #(parameter WIDTH=8)
  (input  logic [$clog2(WIDTH)-1:0] I,
   input  logic                     en,
   output logic [WIDTH-1:0]         D);

  always_comb begin
    D = 0;
    if (en)
      D = 1'b1 << I;
  end

endmodule : Decoder

// module Decoder_test;

//   logic [2:0] I;
//   logic       en;
//   logic [7:0] D;

//   logic [3:0] vector;
//   assign {en, I} = vector;

//   Decoder #(8) dut(.*);

//   initial begin
//     $monitor("I(%b) en(%b) -> D(%b)", I, en, D);
//     for(vector = 4'd0; vector != 4'b1111; vector++)
//       #1;
//     #1;
//     $finish;
//   end

// endmodule : Decoder_test

module Register
  #(parameter WIDTH=8)
  (input  logic [WIDTH-1:0] D,
   input  logic             en, clear, clock,
   output logic [WIDTH-1:0] Q);

  always_ff @(posedge clock)
    if (en)
      Q <= D;
    else if (clear)
      Q <= 0;

endmodule : Register

// module Register_test;

//   logic [7:0] D;
//   logic       en, clear, clock;
//   logic [7:0] Q;

//   Register dut(.*);

//   initial begin
//     clock = 0;
//     forever #5 clock = ~clock;
//   end

//   initial begin
//     $monitor("D(%b) clear(%b) en(%b) -> Q(%b)", D, clear, en, Q);
//     D <= 8'b0111_0001; clear <= 0; en <= 1;
//     #7;
//     D <= 8'b1000_1110; en <= 0;
//     #20;
//     clear <= 1;
//     #10;
//     $finish;
//   end

// endmodule : Register_test

module Counter
#(parameter WIDTH=8)
(input logic [WIDTH-1:0] D,
input logic en, clear, load, clock, up,
output logic [WIDTH-1:0] Q);

always_ff @(posedge clock)
if(clear)
Q <= 0;
else if(load)
Q <= D;
```

```
else if(en&up)
Q <= Q+1;
else if(en&~up)
Q <= Q-1;

endmodule: Counter

module Counter_test;
logic [7:0] D;
logic en, clear, load, clock, up;
logic [7:0] Q;
Counter dut(.*);
initial begin
clock = 0;
forever #5 clock = ~clock;
end

initial begin
$monitor("D(%b) clear(%b) en(%b) load(%b) up(%b) -> Q(%b)",
  D, clear, en, load, up, Q);

D <= 8'd10;
clear <= 1;
load <= 0;
en <= 1;
up <= 1;
@(posedge clock);
@(posedge clock);
clear <= 0;
@(posedge clock);
@(posedge clock);
@(posedge clock);
@(posedge clock);
@(posedge clock);
@(posedge clock);
@(posedge clock);
@(posedge clock);

#10
$finish;
end
endmodule: Counter_test

module ShiftRegister
#(parameter WIDTH = 8)
(input logic [WIDTH-1:0] D,
input logic en, left, load, clock,
output logic [WIDTH-1:0] Q);

always_ff @(posedge clock)
if(load)
Q <= D;
else if(en && left)
Q <= {Q[WIDTH-2:0], 1'b0};
else if(en && ~left)
Q <= {1'b0, Q[WIDTH-1:1]};

endmodule: ShiftRegister
/*
module ShiftRegister_test;
logic [7:0] D;
logic en, left, load, clock;
logic [7:0] Q;
ShiftRegister dut(.*);

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

initial begin
```

```
$monitor("D(%b) en(%b) load(%b) left(%b) -> Q(%b)", D, en, load, left, Q);
D <= 8'b1010_1010;
en <= 1;
load <= 1;
left <= 1;
@(posedge clock);
@(posedge clock);
load <= 0;
@(posedge clock);
left <= 0;
@(posedge clock);
en <= 0;
@(posedge clock);
@(posedge clock);
#10
$finish;
end
endmodule: ShiftRegister_test
*/
module BarrelShiftRegister
#(parameter WIDTH = 8)
(input logic [WIDTH-1:0] D,
input logic load, en, clock,
input logic [1:0] by,
output logic [WIDTH-1:0] Q);

always_ff @(posedge clock)
if(load)
Q <= D;
else if(en)
Q <= Q << by;
endmodule: BarrelShiftRegister
/*
module BarrelShiftRegister_test;
logic [7:0] D;
logic load, en, clock;
logic [1:0] by;
logic [7:0] Q;
BarrelShiftRegister dut(.*);

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

initial begin
$monitor("D(%b) en(%b) load(%b) by(%b) -> Q(%b)", D, en, load, by, Q);
D <= 8'b1111_1111;
en <= 0;
load <= 0;
by <= 2'd3;
@(posedge clock);
en <= 1;
@(posedge clock);
by <= 2'd1;
@(posedge clock);
by <= 2'd0;
@(posedge clock);
load <= 1;
@(posedge clock);
@(posedge clock);
#10
$finish;
end
endmodule: BarrelShiftRegister_test
*/
module Memory
#(parameter DW=16,
W=256,
AW=$clog2(W))
```

```
(input logic re,we,clock,
input logic [AW-1:0] Addr,
inout wire [DW-1:0] Data);

logic [DW-1:0] M[W];
logic [DW-1:0] out;

assign Data = (re) ? out:'bz;

always_ff @(posedge clock)
if(we) M[Addr] <= Data;

always_comb
out = M[Addr];

endmodule: Memory
/*
module Memory_test;
logic re, we, clock, tri_en;
logic [7:0] Addr;
tri [3:0] Data;
logic [3:0] driveData;

Memory #(.AW(8), .DW(4)) dut(.*);

assign Data = (tri_en) ? driveData : 4'bz;

initial begin
clock = 0;
forever #5 clock = ~clock;
end
initial begin
for (Addr = 8'd0; Addr < 8'd4; Addr += 1) begin
      re <= 1'b0;
      we <= 1'b1;
      driveData <= Addr[3:0];
      tri_en = 1'b1;
      @(posedge clock);
    end
#5 $finish;
end
endmodule: Memory_test
*/
```