ENGR 378 Digital Systems Design

Exp. 6: Pong

Submitted by:

Björn Franzén
Kristian Lauszus

Nov. 6, 2014

Demonstration……………………..…..____/5

Report: Presentation …………...____/2

Problem analysis…….…………….____/2

Design work……………………….____/5

Results………………………..………..____/5

Conclusion/Discussion……………..…….____/1

Total…………………………..…….____/ 20

Problem Analysis

In this lab we needed to implement a simple two-player pong game and show it on a computer monitor via VGA. We modularized our design as described in the pre-lab report.

Hardware Design

To modularize our design we broke it down into three different modules, one sub-module and one top module. We had the top module use the positions of the paddles and the ball, output from the respective module, to output to the screen. This was done by setting three outputs, to determine the colour, when the pixel was at the proper position. This was handled by a module provided to us, seen in Figure 1. The modules “paddle”, Figure 4, “Ball”, Figure 5, controls the respective function and outputs the wanted position. “GameScore”, Figure 3, keeps tracks of the current score by watching two inputs were either is set high by the ball module if the ball hits the respective corner. It also contains a sub-module for outputting the score to the segment displays.

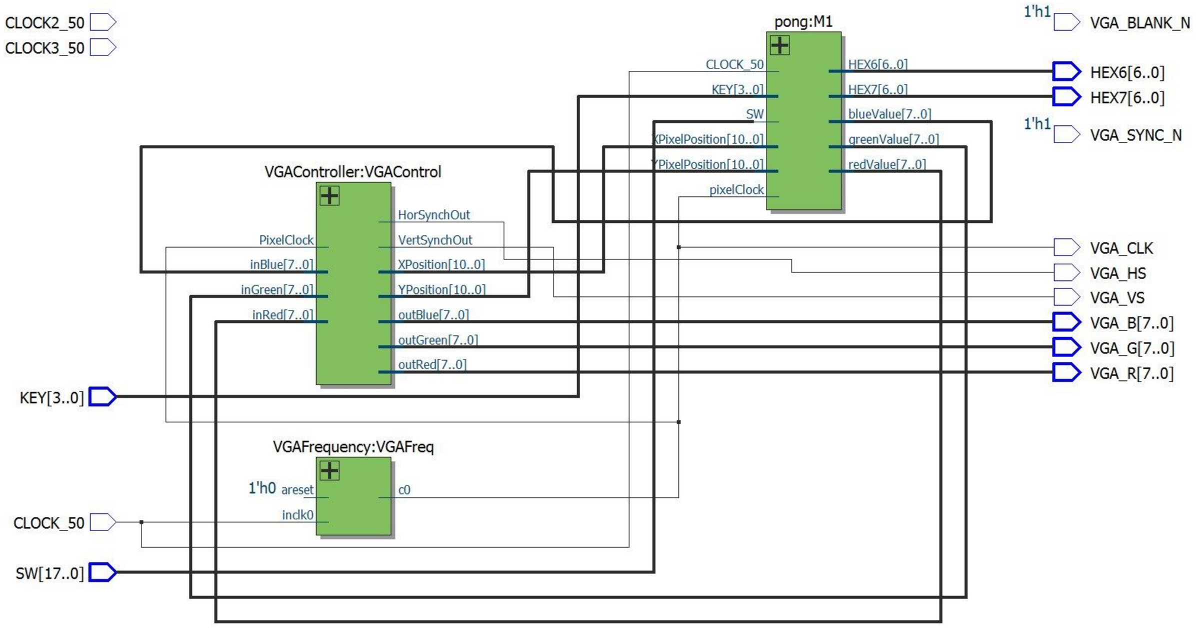


Figure 1 – Hardware synthesis of the top module provided to us.

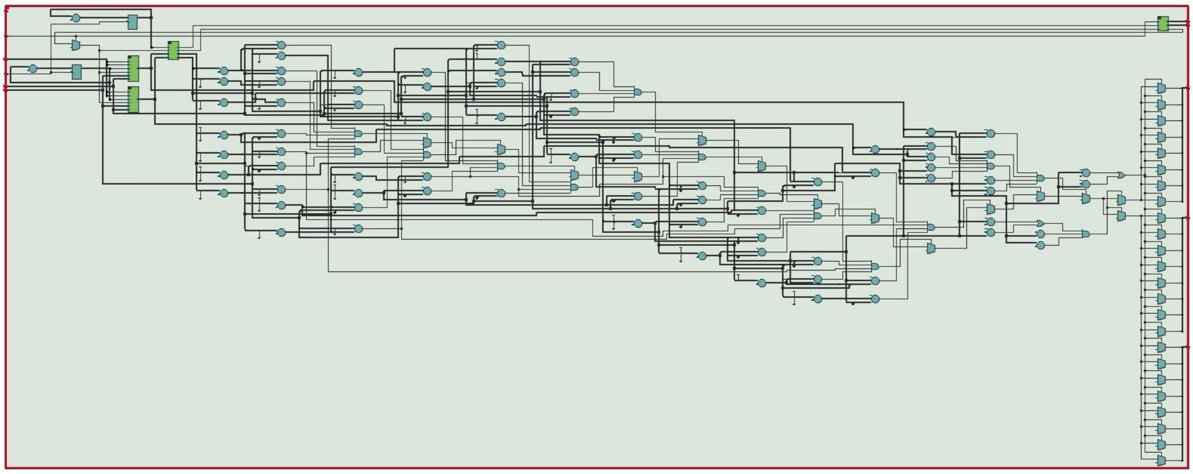


Figure 2 - Hardware synthesis of our top module.

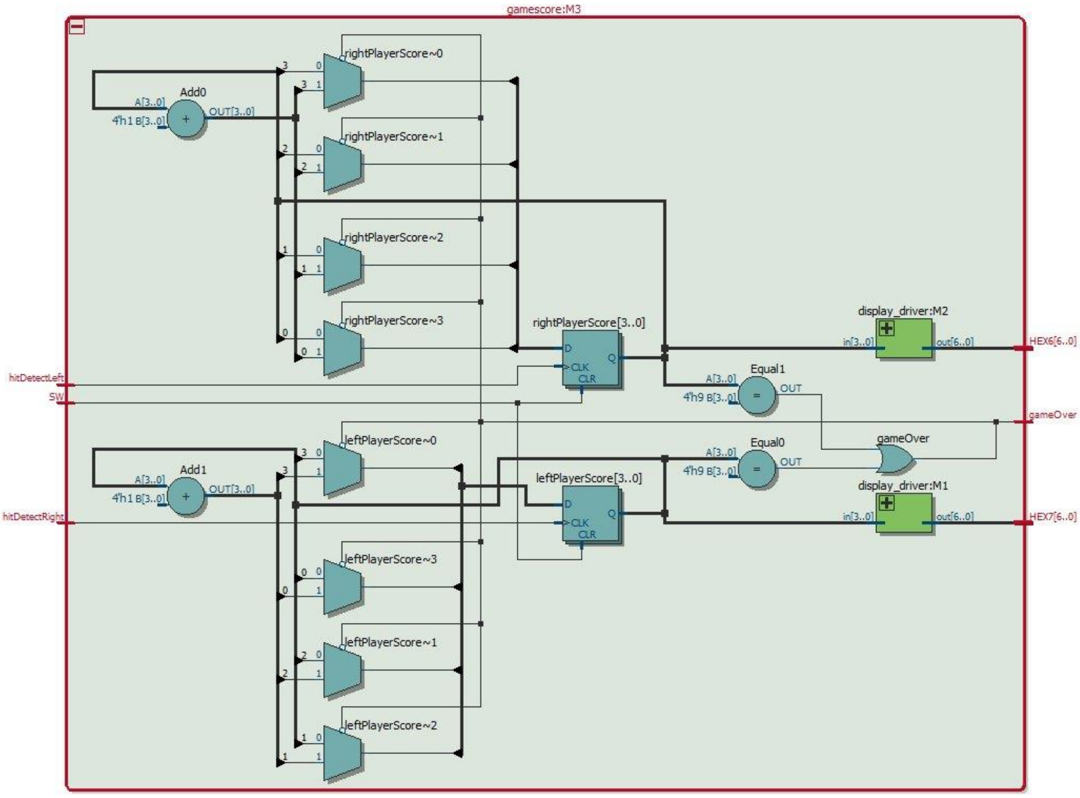


Figure 3 – Hardware synthesis of the module for keeping track and outputting game scores.

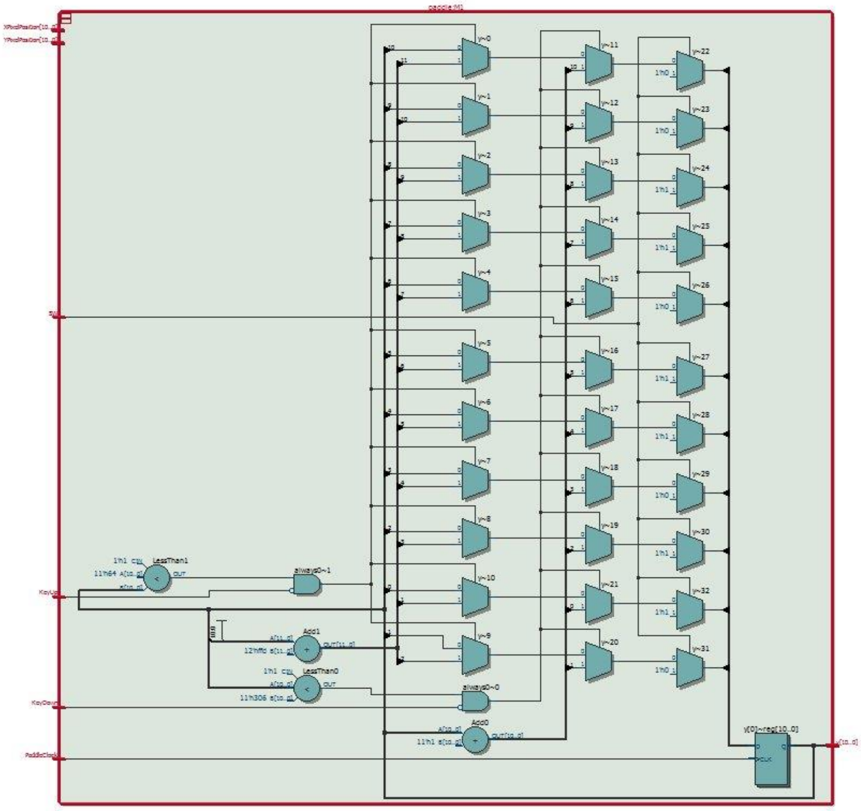


Figure 4 - Hardware synthesis of the module for movement of the paddles.

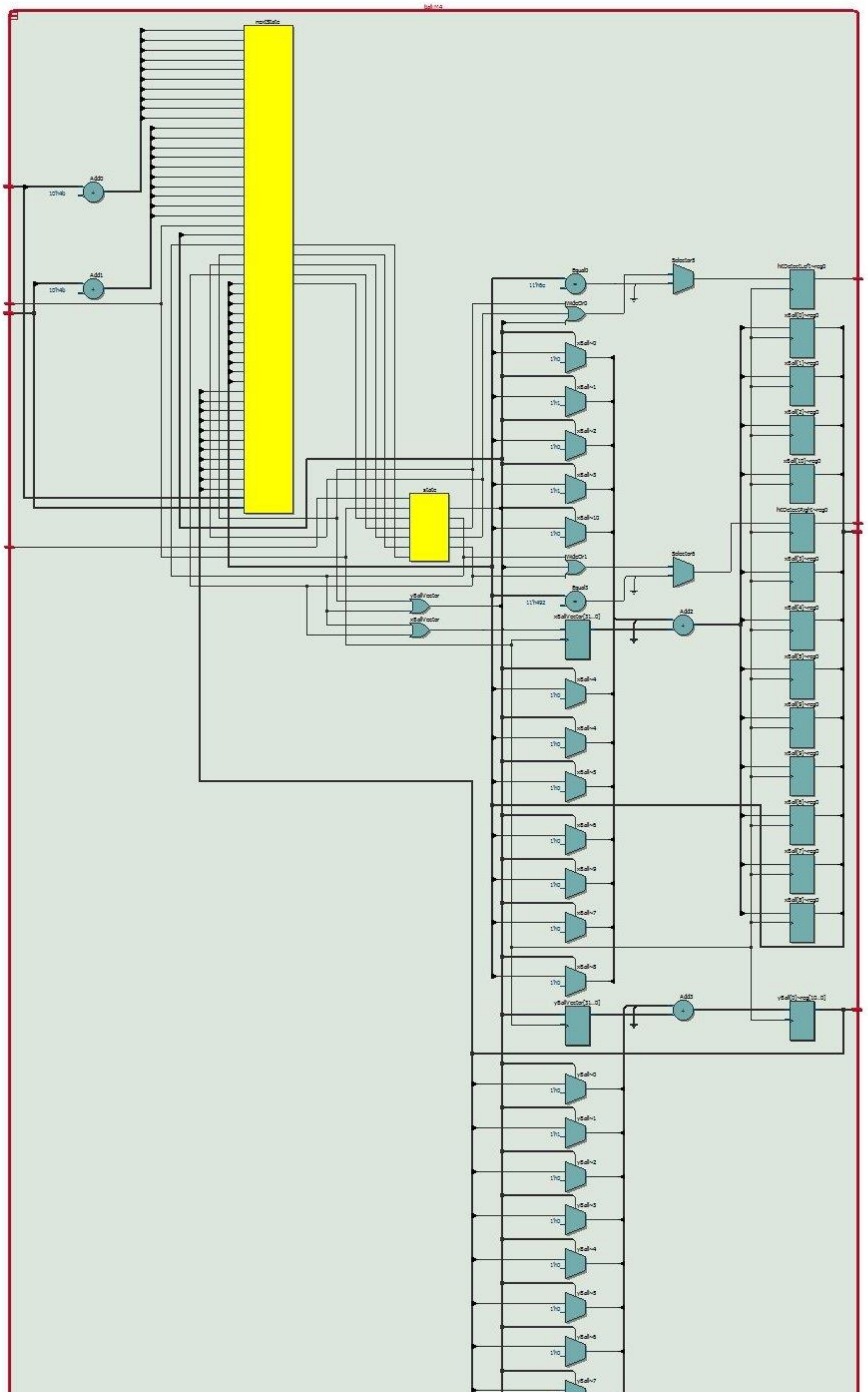


Figure 5 - Hardware synthesis of the module for ball movement.

Verilog Modelling

As described we first created a top-module called pong and created an instance of this in the VGAInterface module.

The pong module then had two instances of the paddle module, where one of them had the two buttons to the left as inputs and then other one had the buttons at the right as inputs. The output from these two modules where then saved in the two wires yPL (y-axis paddle left) and yPR (y-axis paddle right) which indicated the y-coordinate of the left and right paddle respectively.

The next module was the gamescore module, which was responsible for incrementing two counters every time one of the users scored i.e. the ball hit one of the sides. These counter values were then outputted on the 7-segment displays using a display drier we wrote in an earlier lab. The gamescore module also had an output signal which got asserted once of the players reached a score of nine.

The final sub-module was the ball module. This is simply a FSM according to the diagram in the lab assignment. The x- and y-coordinates where output from this module together with two signals indicating if the ball hits one of the sides. This is then used as the input to the gamescore module.

Finally the information from all the modules was then used to set the RGB-values to the monitor. First it checks if it should draw the sidebars, then the top and bottom bar. After that it draws the two paddles and after that it draws a round ball. This is all drawn on a black background.

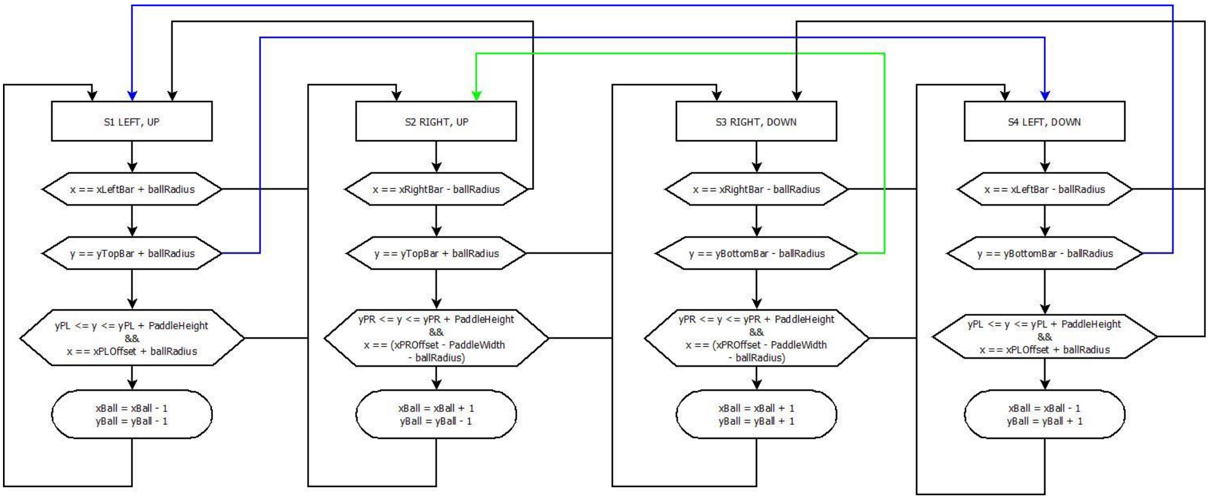
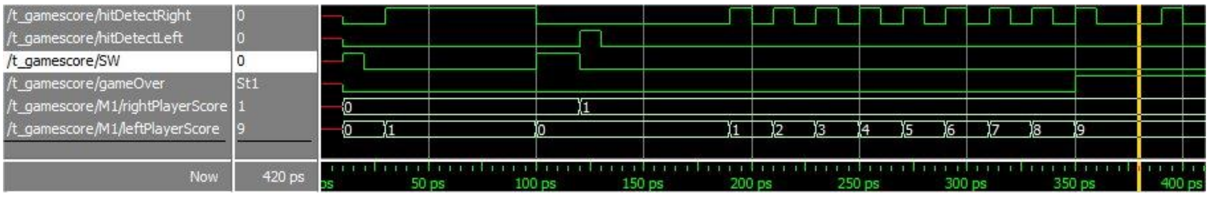

S1 LEFT, UP
x == xLeftBar + ballRadius
y == yTopBar + ballRadius
yPL <= y <= yPL + PaddleHeight && x == xPLOffset + ballRadius
xBall = xBall - 1
yBall = yBall - 1
S2 RIGHT, UP
x == xRightBar - ballRadius
y == yTopBar + ballRadius
yPR <= y <= yPR + PaddleHeight && x == (xPROffset - PaddleWidth - ballRadius)
xBall = xBall + 1
yBall = yBall - 1
S3 RIGHT, DOWN
x == xRightBar - ballRadius
y == yBottomBar - ballRadius
yPR <= y <= yPR + PaddleHeight && x == (xPROffset - PaddleWidth - ballRadius)
xBall = xBall + 1
yBall = yBall + 1
S4 LEFT, DOWN
x == xLeftBar - ballRadius
y == yBottomBar - ballRadius
yPL <= y <= yPL + PaddleHeight && x == xPLOffset + ballRadius
xBall = xBall - 1
yBall = yBall + 1

Figure 6 – ASM chart for the Ball module.

Results/Verification

Since the project in nature is a lot harder to simulate than previous labs, we only assigned a test bench for the GameScore module, which could be run independent of the others. Instead of creating a test-bench for the other modules we used a trial and error approach were we uploaded our programs and evaluated the function on the screen. This is of course not ideal but for our relatively simplistic program, we couldn't justify writing a test-bench which would require a lot of behavioural modelling to capture the correct behaviour. It would also be very tedious to evaluate the results which again is not appropriate for a design on this level.



```
/t_gamescore/hitDetectRight       0
/t_gamescore/hitDetectLeft        0
/t_gamescore/SW                   0
/t_gamescore/gameOver             St1
/t_gamescore/M1/rightPlayerScore  1
/t_gamescore/M1/leftPlayerScore   9
Now                               420 ps
```

Figure 7 – Simulation of the GameScore module.

Conclusions and Discussion

We had some problems connecting the different modules in the top design because of the complexity of the design. For an example we initially constructed the GameScore module as a state machine but then ran in the synchronizing issues when connecting it to the state machine in ball module. We then wrote a combinational module instead which solved the issue.

Overall it was great to have a more complex design when we could split up the different tasks.

Pre-lab

Below is the diagram for the top-module, paddle module and an overview of the game. We did not make a diagram for the ball module, as this was pretty much done in the lab assignment already. Also the score module was so simple that we did not bother to do this as well.

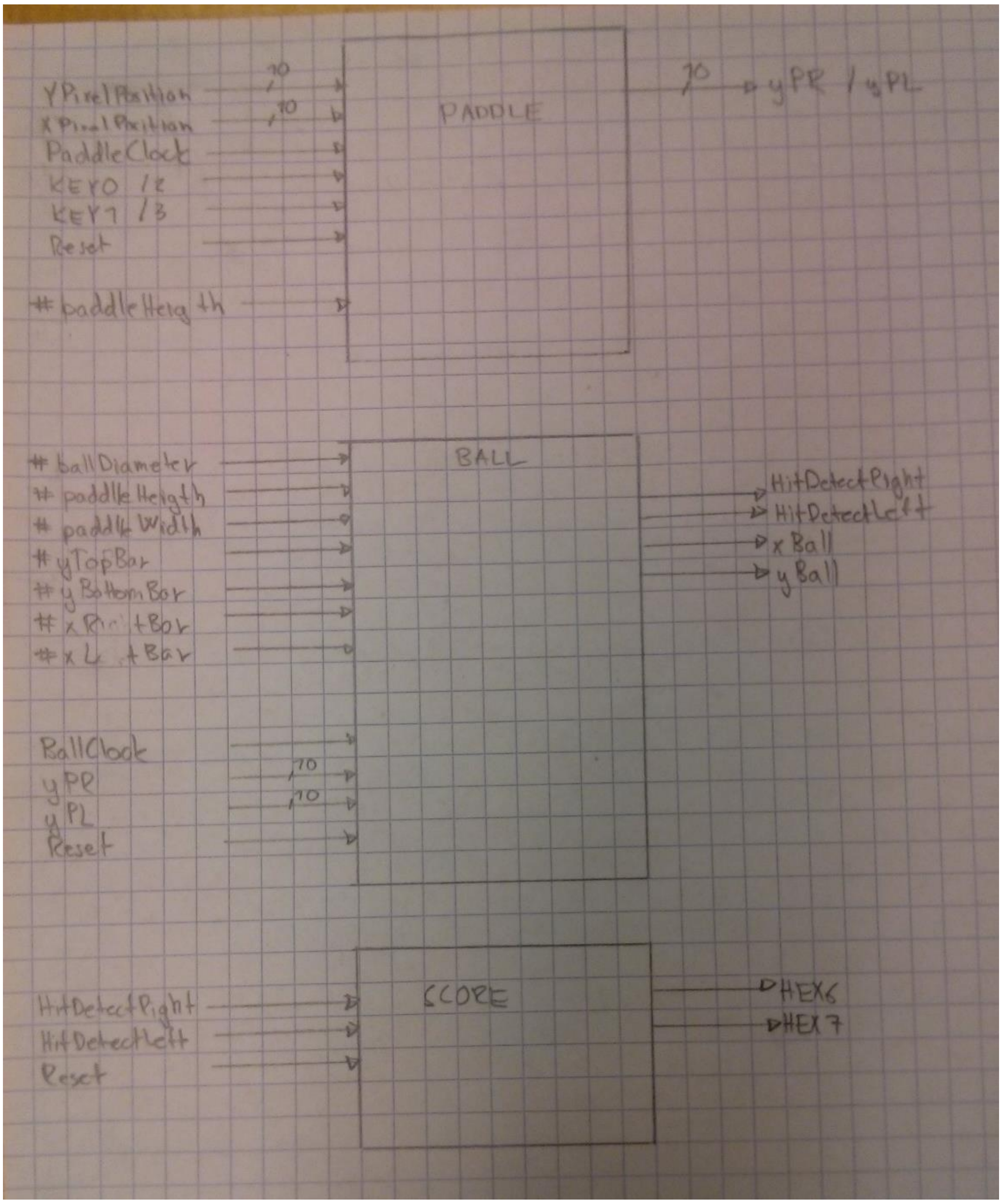


Figure 8 – Top module digram

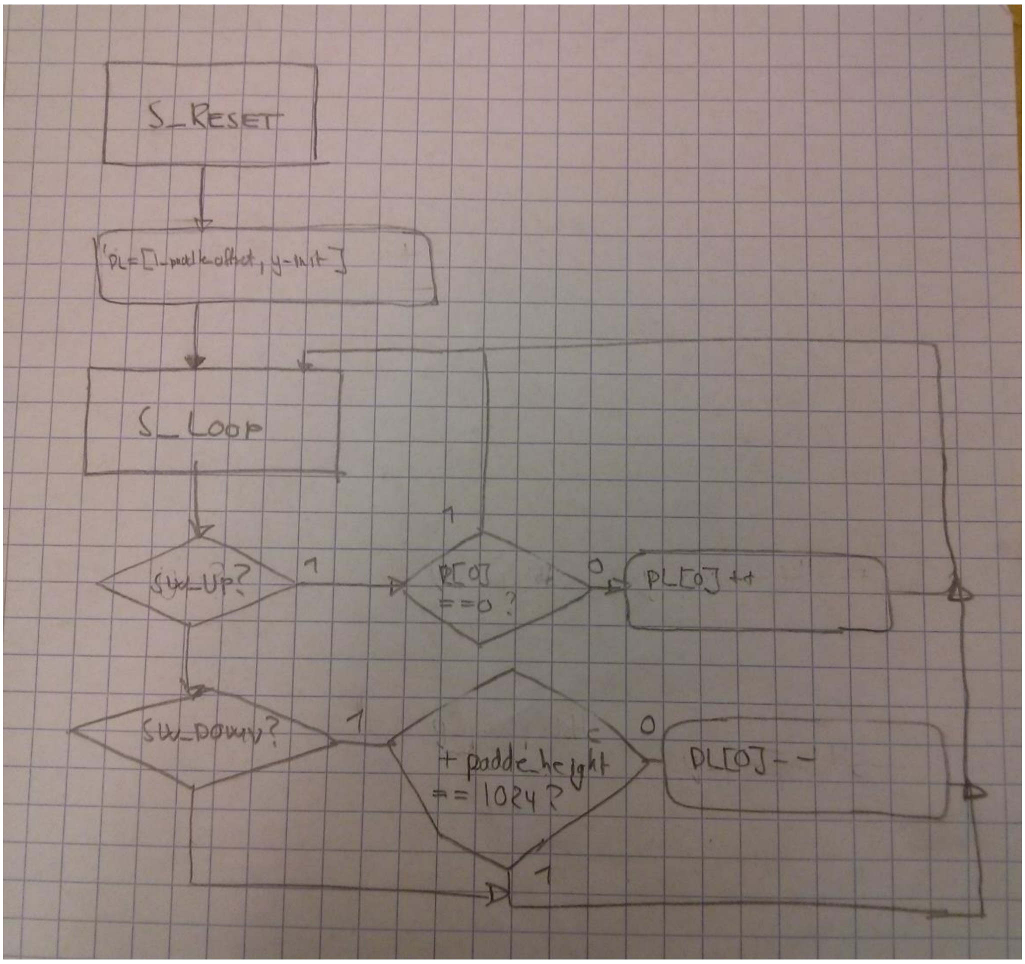


Figure 9 – Paddle module

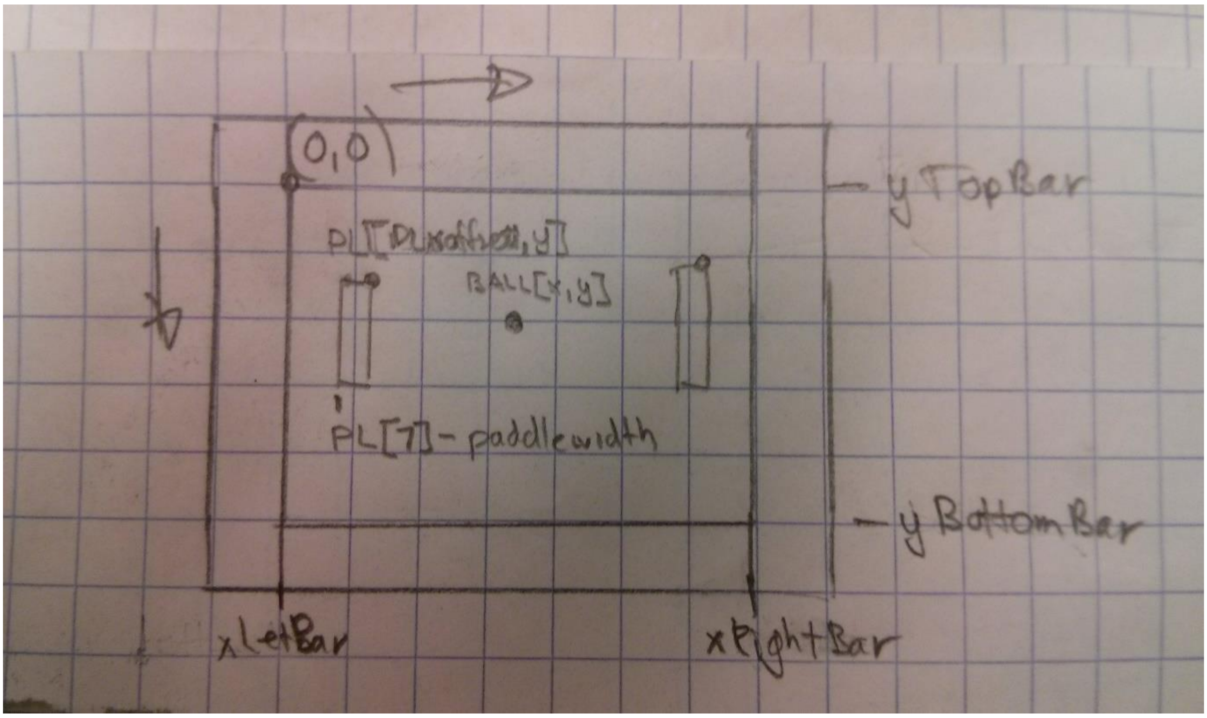


Figure 10 – Game overview

Work Breakdown

We initially broke it down so that Kristian would create the ball movement and Björn the rest. In the end though the work was intertwined to be a combined effort.

HDL Source Code

```
// ENGR 378 (VGA Lab)
// Date: 12/13/11
// Written by: Michael Chan
// San Francisco State University
// Note: This code is used for the VGA lab for ENGR 378 and is written to support
monitor display resolutions of 1280 x 1024 at 60 fps

// (DETAILS OF THE MODULES)
// VGAInterface.v is the top most level module and asserts the red/green/blue
signals to draw to the computer screen
// VGAController.v is a submodule within the top module used to generate the
vertical and horizontal synch signals as well as X and Y pixel positions
// VGAFrequency.v is a submodule within the top module used to generate a 108Mhz
pixel clock frequency from a 50Mhz pixel clock frequency using the PLL

// (USER/CODER Notes)
// Note: User should modify/write code in the VGAInterface.v file and not modify
any code written in VGAController.v or VGAFrequency.v

module VGAInterface(

        //////////// CLOCK //////////
        CLOCK_50,
        CLOCK2_50,
        CLOCK3_50,

        //////////// LED //////////
//      LEDG,
//      LEDR,

        //////////// KEY //////////
        KEY,

        //////////// SW //////////
        SW,

        //////////// SEG7 //////////
//      HEX0,
//      HEX1,
//      HEX2,
//      HEX3,
//      HEX4,
//      HEX5,
        HEX6,
        HEX7,

        //////////// VGA //////////
        VGA_B,
        VGA_BLANK_N,
        VGA_CLK,
        VGA_G,
        VGA_HS,
        VGA_R,
        VGA_SYNC_N,
        VGA_VS
);

//=======================================================
//  PARAMETER declarations
//=======================================================
//=======================================================
//  PORT declarations
//=======================================================

//////////// CLOCK //////////
input                                     CLOCK_50;
input                                     CLOCK2_50;
input                                     CLOCK3_50;

//////////// LED //////////
//output                  [8:0]           LEDG;
//output                 [17:0]           LEDR;

//////////// KEY //////////
input                     [3:0]           KEY;

//////////// SW //////////
input                    [17:0]           SW;

//////////// SEG7 //////////
//output                  [6:0]           HEX0;
//output                  [6:0]           HEX1;
//output                  [6:0]           HEX2;
//output                  [6:0]           HEX3;
//output                  [6:0]           HEX4;
//output                  [6:0]           HEX5;
output                    [6:0]           HEX6;
output                    [6:0]           HEX7;

//////////// VGA //////////
output                    [7:0]           VGA_B;
output                                    VGA_BLANK_N;
output                                    VGA_CLK;
output                    [7:0]           VGA_G;
output                                    VGA_HS;
output                    [7:0]           VGA_R;
output                                    VGA_SYNC_N;
output                                    VGA_VS;

//=======================================================
//  REG/WIRE declarations
//=======================================================
reg      aresetPll = 0; // asynchrous reset for pll
wire     pixelClock;
wire     [10:0] XPixelPosition;
wire     [10:0] YPixelPosition;
wire     [7:0] redValue;
wire     [7:0] greenValue;
wire     [7:0] blueValue;

// slow clock counter variables
reg      [17:0] slowClockCounter = 0;
wire     slowClock;

// variables for the dot
//reg    [10:0] XDotPosition = 500;
//reg    [10:0] YDotPosition = 500;

//=======================================================
//  Structural coding
//=======================================================

// output assignments
assign VGA_BLANK_N = 1'b1;
assign VGA_SYNC_N = 1'b1;
assign VGA_CLK = pixelClock;
```

```
// PLL Module (Phase Locked Loop) used to convert a 50Mhz clock signal to a 108 MHz
clock signal for the pixel clock
VGAFrequency VGAFreq (aresetPll, CLOCK_50, pixelClock);

// VGA Controller Module used to generate the vertial and horizontal synch signals
for the monitor and the X and Y Pixel position of the monitor display
VGAController VGAControl (pixelClock, redValue, greenValue, blueValue, VGA_R,
VGA_G, VGA_B, VGA_VS, VGA_HS, XPixelPosition, YPixelPosition);

// COLOR ASSIGNMENT PROCESS (USER WRITES CODE HERE TO DRAW TO SCREEN)

//eight_colors M1(.LEDR(LEDR[10:0]), .SW1(SW[1]),.XPixelPosition(XPixelPosition),
.YPixelPosition(YPixelPosition), .redValue(redValue), .blueValue(blueValue),
.greenValue(greenValue), .pixelClock(pixelClock), .slowClock(slowClock));
pong M1( .XPixelPosition(XPixelPosition), .YPixelPosition(YPixelPosition),
.redValue(redValue), .blueValue(blueValue), .greenValue(greenValue),
.pixelClock(pixelClock), .CLOCK_50(CLOCK_50), .HEX7(HEX7), .HEX6(HEX6), .KEY(KEY),
.SW(SW[0]));

endmodule
module ball#(
                parameter PaddleHeight,
                                          PaddleWidth,
                                          BallRadius,
                                          yTopBar,
                                          yBottomBar,
                                          xRightBar,
                                          xLeftBar,
                                          xPLOffset,
                                          xPROffset,
                                          xBallInit,
                                          yBallInit
        )(
                output reg [10:0] xBall, yBall,
                output reg hitDetectRight,  hitDetectLeft,
                input BallClock, SW,
                input [10:0] yPL, yPR
        );
        parameter S_RESET        = 0, // Reset position
                                  S_1              = 1, // LEFT_UP
                                  S_2              = 2, // RIGHT_UP
                                  S_3              = 3, // RIGHT_DOWN
                                  S_4              = 4; // LEFT_DOWN
        reg [2:0] state, nextState;
        integer xBallVector, yBallVector;

        always @ (posedge BallClock)
        begin
                if(SW) state <= S_RESET;
                else state <= nextState;
        end

        always @ (posedge BallClock) begin // Will only trigger on state since
yPLReg, yPRReg, xBall, yBall are all synced with the clock. , xBall, yBall, yPLReg,
yPRReg
                hitDetectLeft = 0;
                hitDetectRight = 0;

                /*xBallVector = 0;
                yBallVector = 0;*/

                case(state)
                        S_RESET :
                                                        begin
                                                                xBall =
xBallInit;
```

```
                                                                    yBall =
yBallInit;
                                                                    xBallVector <=
0;
                                                                    yBallVector <=
0;
                                                                    nextState =
S_1;
                                                            end
                          S_1      :                 begin // Left, Up
                                                                    xBallVector <=
-1;
                                                                    yBallVector <=
-1;

                                                                    if(xBall ==
xLeftBar + BallRadius) // Hit left bar
                                                                    begin

        hitDetectLeft = 1;

        nextState = S_2;

                                                                    end
                                                                    else if(yBall
== yTopBar + BallRadius) // Hit top bar

        nextState = S_4;
                                                                    else if(yBall
>= yPL && yBall <= yPL + PaddleHeight && xBall == xPLOffset + BallRadius) // Hit
left paddle

        nextState = S_2;
                                                                    //else
nextState = S_1;
                                                            end
                          S_2      :                 begin // Right, Up
                                                                    xBallVector <=
1;
                                                                    yBallVector <=
-1;
                                                                    if(xBall ==
xRightBar - BallRadius) // Hit right bar
                                                                    begin

        nextState = S_1;

        hitDetectRight = 1;
                                                                    end
                                                                    else if(yBall
== yTopBar + BallRadius) // Hit top bar

        nextState = S_3;
                                                                    else if(yPR <=
yBall && yBall <= yPR + PaddleHeight && xBall == xPROffset - PaddleWidth -
BallRadius) // Hit right paddle

        nextState = S_1;
                                                                    //else
nextState = S_2;
                                                            end
                          S_3      :                 begin // Right, Down
                                                                    xBallVector <=
1;
                                                                    yBallVector <=
1;
                                                                    if(xBall ==
xRightBar - BallRadius) // Hit right bar
                                                                    begin
        nextState = S_4;

        hitDetectRight = 1;
                                                                end
                                                                else if(yBall
== yBottomBar - BallRadius) // Hit bottom bar

        nextState = S_2;
                                                                else if(yPR <=
yBall && yBall <= yPR + PaddleHeight && xBall == xPROffset - PaddleWidth -
BallRadius) // Hit right paddle

        nextState = S_4;
                                                                //else
nextState = S_3;
                                                        end
                        S_4     :               begin   // Left, Down
                                                                xBallVector <=
-1;
                                                                yBallVector <=
1;
                                                                if(xBall ==
xLeftBar + BallRadius) // Hit left bar
                                                                begin

        nextState = S_3;

        hitDetectLeft = 1;
                                                                end
                                                                else if(yBall
== yBottomBar - BallRadius) // Hit bottom bar

        nextState = S_1;
                                                                else if(yPL <=
yBall && yBall <= yPL + PaddleHeight && xBall == xPLOffset + BallRadius) // Hit
left paddle

        nextState = S_3;
                                                                //else
nextState = S_4;
                                                        end
                        default nextState = S_RESET;
                endcase

                xBall = xBall + xBallVector;
                yBall = yBall + yBallVector;
        end
endmodule
module gamescore(
        input hitDetectRight,  hitDetectLeft, SW,
        output gameOver,
        output [6:0] HEX6, HEX7
);
        reg [3:0] rightPlayerScore, leftPlayerScore;

        display_driver M1(.out(HEX7), .in(leftPlayerScore));        // Output score
to segment displays.
        display_driver M2(.out(HEX6), .in(rightPlayerScore));

        assign gameOver = leftPlayerScore == 4'd9 || rightPlayerScore == 4'd9; //
Needed for gameover to update immediately since the always
        //      loop only executes on changes to its event control and there would
be no change if this would be checked directly in S_COUNT.

        always@(posedge hitDetectLeft, posedge SW)
                if(SW) rightPlayerScore = 0;
                else if (!gameOver) rightPlayerScore = rightPlayerScore + 1'd1;
        always@(posedge hitDetectRight, posedge SW)
                if(SW) leftPlayerScore = 0;
                else if(!gameOver) leftPlayerScore = leftPlayerScore + 1'd1;

endmodule
module t_gamescore();
        reg hitDetectRight,  hitDetectLeft, SW;
        wire gameOver;
        wire [6:0] HEX6, HEX7;

        gamescore M1(.hitDetectRight(hitDetectRight),
.hitDetectLeft(hitDetectLeft), .SW(SW), .gameOver(gameOver), .HEX6(HEX6),
.HEX7(HEX7));

        initial begin
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b001;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b000;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b100;

                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b100;
                #50
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b001; // Reset
                #10
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b010;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b000;
                #50
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b100;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b000;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b100;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b000;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b100;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b000;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b100;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b000;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b100;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b000;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b100;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b000;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b100;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b000;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b100;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b000;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b100;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b000;
                #20
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b100;
                #10 {hitDetectRight, hitDetectLeft, SW} = 3'b000;
                #20$stop;
        end

endmodule

module display_driver(
                output reg[6:0] out,
                input [3:0] in
        );

        always @ (in) begin
                case (in)
                        4'b0000 : out = 7'b1000000; // 0
                        4'b0001 : out = 7'b1111001; // 1
                        4'b0010 : out = 7'b0100100; // 2
                        4'b0011 : out = 7'b0110000; // 3
                        4'b0100 : out = 7'b0011001; // 4
                        4'b0101 : out = 7'b0010010; // 5
                        4'b0110 : out = 7'b0000011; // 6
                        4'b0111 : out = 7'b1111000; // 7
                        4'b1000 : out = 7'b0000000; // 8
                        4'b1001 : out = 7'b0011000; // 9

                        4'b1010 : out = 7'b0001000; // A
                        4'b1011 : out = 7'b0000000; // B (8)
                        4'b1100 : out = 7'b1000110; // C
                        4'b1101 : out = 7'b1000000; // D (0)
                        4'b1110 : out = 7'b0000110; // E
                        4'b1111 : out = 7'b0001110; // F

                        default : out = 7'b1111111; // OFF
                endcase
        end
endmodule
module paddle#(
        parameter       PaddleHeight = 200,
                                        yPaddleInit =
600,
                                        yTopBar = 100,
                                        yBottomBar = (1024 - 100)
)
(
        output reg [10:0] y,
        input PaddleClock,
                        SW,
        input [10:0]    XPixelPosition,  // Position on x-axis. Increments on
pixelClock with some deadzone at start and end.
                                                YPixelPosition,  // Position on
y-axis. Increments when XPixelPosition has reached end of screen+deadzone.
        input KeyUp, KeyDown
);

        parameter       S_Reset = 1'b0,
                                        S_Loop = 1'b1;

        always@(posedge PaddleClock) begin
                if(SW == 1)
                begin
                        y = yPaddleInit;
                end
                else if((KeyDown == 0)&&(y <= (yBottomBar - PaddleHeight)))
                        y = y + 1'b1;
                else if((KeyUp == 0)&&(y >= yTopBar))
                        y = y - 1'b1;
        end
endmodule
module pong(
        output reg [7:0] redValue, blueValue, greenValue,
        output [6:0] HEX6, HEX7,
        input pixelClock,       // VGA frequency
                        CLOCK_50,       // 50Mhz
                        SW,
        input [10:0]    XPixelPosition,
                                                YPixelPosition,
        input [3:0] KEY
);
        parameter       PaddleHeight = 11'd150,
                                        PaddleWidth = 11'd10,
                                        BallRadius = 11'd10,
                                        yTopBar = 11'd100,
                                        yBottomBar = (11'd1024 - 11'd100),
                                        xRightBar = (11'd1280-11'd100),
                                        xLeftBar = 11'd100,
                                        xPLOffset = 11'd150,
                                        xPROffset = (11'd1280 - 11'd150 +
PaddleWidth),
                                        yPaddleInit = (11'd1024/11'd2 -
PaddleHeight/11'd2),
                                        xBallInit = 11'd1280/11'd2,
                                        yBallInit = 11'd1025/11'd2;

        wire [10:0] yPL, yPR;
        wire [10:0] yBall, xBall;
        wire gameOver, hitDetectLeft, hitDetectRight, BallClock, PaddleClock,
gameOverOrReset;
        reg [16:0] PaddleCounter;
        reg [16:0] BallCounter;

        // generates a slow clock by selecting the MSB from a large counter
        assign PaddleClock = PaddleCounter[16];
        assign BallClock = BallCounter[16];

        always@ (posedge CLOCK_50)
        begin
                BallCounter <= BallCounter + 1;
                PaddleCounter <= PaddleCounter + 1;
        end

        assign gameOverOrReset = SW == 1 ? 1'b1 : gameOver == 1 ? 1'b1 : 1'b0; //
If gameover then lock paddles and ball movement. This is input to all modules
instead of SW.

        // Controls paddle movement.
        paddle  #(.yBottomBar(yBottomBar), .yTopBar(yTopBar),
.PaddleHeight(PaddleHeight), .yPaddleInit(yPaddleInit))
                                M1(.YPixelPosition(YPixelPosition),
.XPixelPosition(XPixelPosition), .PaddleClock(PaddleClock), .SW(gameOverOrReset),
.y(yPL), .KeyUp(KEY[0]), .KeyDown(KEY[1]));
        paddle  #(.yBottomBar(yBottomBar), .yTopBar(yTopBar),
.PaddleHeight(PaddleHeight), .yPaddleInit(yPaddleInit))
                                M2(.YPixelPosition(YPixelPosition),
.XPixelPosition(XPixelPosition), .PaddleClock(PaddleClock), .SW(gameOverOrReset),
.y(yPR), .KeyUp(KEY[2]), .KeyDown(KEY[3]));

        // Outputs gameOver when a player has a score of 9.
        gamescore M3(.hitDetectLeft(hitDetectLeft),
.hitDetectRight(hitDetectRight), .SW(SW), .gameOver(gameOver), .HEX6(HEX6),
.HEX7(HEX7));

        // Controls ball movement
        ball            #(.PaddleHeight(PaddleHeight), .PaddleWidth(PaddleWidth),
.BallRadius(BallRadius), .yTopBar(yTopBar), .yBottomBar(yBottomBar),
.xRightBar(xRightBar),
                                        .xLeftBar(xLeftBar),
.xPLOffset(xPLOffset), .xPROffset(xPROffset), .xBallInit(xBallInit),
.yBallInit(yBallInit))
                                M4(.xBall(xBall), .yBall(yBall),
.BallClock(BallClock), .SW(gameOverOrReset), .yPL(yPL), .yPR(yPR),
.hitDetectRight(hitDetectRight),  .hitDetectLeft(hitDetectLeft));

        always @(*) begin
                // Outputs to screen.
                if(XPixelPosition <= xLeftBar || XPixelPosition >= xRightBar)
                begin // Side bars.
                        redValue <= 8'b00000000;
                        blueValue <= 8'b00000000;
                        greenValue <= 8'b11111111;
                end
                else if((XPixelPosition > xLeftBar && XPixelPosition < xRightBar)
&& (YPixelPosition <= yTopBar || YPixelPosition >= yBottomBar))
                begin // Top and Bottom Bar
                        redValue <= 8'b11111111;
                        blueValue <= 8'b11111111;
                        greenValue <= 8'b00000000;
                end
                else if((YPixelPosition >= yPL && YPixelPosition <= (yPL +
PaddleHeight)) && (XPixelPosition >= (xPLOffset - PaddleWidth) && (XPixelPosition
<= xPLOffset)))
                begin // Left Paddle
                        redValue <= 8'b11111111;
                        blueValue <= 8'b11111111;
                        greenValue <= 8'b11111111;
                end
                else if((YPixelPosition >= yPR && YPixelPosition <= (yPR +
PaddleHeight)) && (XPixelPosition >= (xPROffset - PaddleWidth) && (XPixelPosition
<= xPROffset)))
                begin // Right Paddle
                        redValue <= 8'b11111111;
                        blueValue <= 8'b11111111;
                        greenValue <= 8'b11111111;
                end

                // Start Ball Plot. All the if statements are there to make the
ball round(ish) shaped.
                else if(((yBall - 11*BallRadius/10) <= YPixelPosition &&
YPixelPosition <= (yBall - BallRadius)) && ((xBall - BallRadius/5) <=
XPixelPosition   && XPixelPosition <= xBall + 1*BallRadius/5))
                begin
                        redValue <= 8'b11111111;
                        blueValue <= 8'b11111111;
                        greenValue <= 8'b11111111;
                end
                else if(((yBall - BallRadius) <= YPixelPosition && YPixelPosition
<= (yBall - 4*BallRadius/5)) && ((xBall - 6*BallRadius/10) <= XPixelPosition   &&
XPixelPosition <= xBall + 6*BallRadius/10))
                begin
                        redValue <= 8'b11111111;
                        blueValue <= 8'b11111111;
                        greenValue <= 8'b11111111;
                end
                else if(((yBall - 4*BallRadius/5) <= YPixelPosition &&
YPixelPosition <= (yBall - 3*BallRadius/5)) && ((xBall - 8*BallRadius/10) <=
XPixelPosition   && XPixelPosition <= xBall + 8*BallRadius/10))
                begin
                        redValue <= 8'b11111111;
                        blueValue <= 8'b11111111;
                        greenValue <= 8'b11111111;
                end
                else if(((yBall - 3*BallRadius/5) <= YPixelPosition &&
YPixelPosition <= (yBall - 2*BallRadius/5)) && ((xBall - 92*BallRadius/100) <=
XPixelPosition   && XPixelPosition <= xBall + 92*BallRadius/100))
                begin
                        redValue <= 8'b11111111;
                        blueValue <= 8'b11111111;
                        greenValue <= 8'b11111111;
                end
                else if(((yBall - 2*BallRadius/5) <= YPixelPosition &&
YPixelPosition <= (yBall - BallRadius/5)) && ((xBall - 98*BallRadius/100) <=
XPixelPosition   && XPixelPosition <= xBall + 98*BallRadius/100))
                begin
                        redValue <= 8'b11111111;
                        blueValue <= 8'b11111111;
                        greenValue <= 8'b11111111;
                end
                else if(((yBall - BallRadius/5) <= YPixelPosition &&
YPixelPosition <= (yBall + BallRadius/5)) && ((xBall - BallRadius) <=
XPixelPosition   && XPixelPosition <= (xBall + BallRadius)))
                begin
                        redValue <= 8'b11111111;
                        blueValue <= 8'b11111111;
                        greenValue <= 8'b11111111;
                end
                else if(((yBall + BallRadius/5) <= YPixelPosition &&
YPixelPosition <= (yBall + 2*BallRadius/5)) && ((xBall - 98*BallRadius/100) <=
XPixelPosition   && XPixelPosition <= xBall + 98*BallRadius/100))
                begin
                        redValue <= 8'b11111111;
                        blueValue <= 8'b11111111;
                        greenValue <= 8'b11111111;
                end
                else if(((yBall + 2*BallRadius/5) <= YPixelPosition &&
YPixelPosition <= (yBall + 3*BallRadius/5)) && ((xBall - 92*BallRadius/100) <=
XPixelPosition   && XPixelPosition <= xBall + 92*BallRadius/100))
                begin
                        redValue <= 8'b11111111;
                        blueValue <= 8'b11111111;
                        greenValue <= 8'b11111111;
                end
                else if(((yBall + 3*BallRadius/5) <= YPixelPosition &&
YPixelPosition <= (yBall + 4*BallRadius/5)) && ((xBall - 8*BallRadius/10) <=
XPixelPosition   && XPixelPosition <= xBall + 8*BallRadius/10))
                begin
                        redValue <= 8'b11111111;
                        blueValue <= 8'b11111111;
                        greenValue <= 8'b11111111;
                end
                else if(((yBall + 4*BallRadius/5) <= YPixelPosition &&
YPixelPosition <= (yBall + BallRadius)) && ((xBall - 6*BallRadius/10) <=
XPixelPosition   && XPixelPosition <= xBall + 6*BallRadius/10))
                begin
                        redValue <= 8'b11111111;
                        blueValue <= 8'b11111111;
                        greenValue <= 8'b11111111;
                end
                else if(((yBall + BallRadius) <= YPixelPosition && YPixelPosition
<= (yBall + 11*BallRadius/10)) && ((xBall - BallRadius/5) <= XPixelPosition   &&
XPixelPosition <= xBall + BallRadius/5))
                begin
                        redValue <= 8'b11111111;
                        blueValue <= 8'b11111111;
                        greenValue <= 8'b11111111;
                end
                // End Ball Plot

                else
                begin
                        redValue <= 8'b00000000;
                        blueValue <= 8'b00000000;
                        greenValue <= 8'b00000000;
                end
        end
endmodule
module eight_colors(
        output reg [7:0] redValue, blueValue, greenValue,
        input pixelClock,                         // 50Mhz.
                        slowClock,                        // pixelClock/2^20 =>
48Hz
        input [10:0]    XPixelPosition,  // Position on x-axis. Increments on
pixelClock with some deadzone at start and end.
                                                   YPixelPosition,  // Position on
y-axis. Increments when XPixelPosition has reached end of screen+deadzone.

        input           SW1,
        input [10:0] LEDR
        );



        always@ (posedge pixelClock) begin
                if(YPixelPosition >= 1024/2)
```

```
                        begin
                                redValue <= 8'b00000000;
                                blueValue <= 8'b00000000;
                                greenValue <= 8'b00000000;
                        end
                else if(XPixelPosition <= 1280/8)
                        begin
                                redValue <= 8'b00000000;
                                blueValue <= 8'b00000000;
                                greenValue <= 8'b00000000;
                        end
                else if(XPixelPosition <= 1280/8*2)
                        begin
                                redValue <= 8'b11111111;
                                blueValue <= 8'b00000000;
                                greenValue <= 8'b00000000;
                        end
                else if(XPixelPosition <= 1280/8*3)
                        begin
                                redValue <= 8'b00000000;
                                blueValue <= 8'b11111111;
                                greenValue <= 8'b00000000;
                        end
                else if(XPixelPosition <= 1280/8*4)
                        begin
                                redValue <= 8'b11111111;
                                blueValue <= 8'b11111111;
                                greenValue <= 8'b00000000;
                        end
                else if(XPixelPosition <= 1280/8*5)
                        begin // Green
                                redValue <= 8'b00000000;
                                blueValue <= 8'b00000000;
                                greenValue <= 8'b11111111;
                        end
                else if(XPixelPosition <= 1280/8*6)
                        begin // Magenta
                                redValue <= 8'b11111111;
                                blueValue <= 8'b00000000;
                                greenValue <= 8'b11111111;
                        end
                else if(XPixelPosition <= 1280/8*7)
                        begin
                                redValue <= 8'b00000000;
                                blueValue <= 8'b11111111;
                                greenValue <= 8'b11111111;
                        end
                else if(XPixelPosition <= 1280/8*8)
                        begin
                                redValue <= 8'b11111111;
                                blueValue <= 8'b11111111;
                                greenValue <= 8'b11111111;
                        end
                else
                        begin
                                redValue <= 8'b00000000;
                                blueValue <= 8'b00000000;
                                greenValue <= 8'b00000000;
                        end
        end
endmodule
```