ECE 211 Pong Project
Technical Report

Nakul Talwar, Thomas Clagett

December 21, 2015

Abstract

The completed FPGA design of the Pong game is presented, including a high-level description of the design, technical details of internal operation, and information on system validation. This document demonstrates that the project was successfully completed, that the final design meets all requirements of the specification, and that the design is now ready for commercialization.

1 Introduction

Everyone likes to play video games and Pong is definitely a classic. This Pong game is a modern re-creation of the original game designed to work on any VGA monitor. This FPGA based design overcomes those limitations and has the following objectives:

1. Provide a simple playable version of Pong for FPGA board owners.
2. Allow the game to be played on any display that supports VGA.
3. Present a visually appealing design to players who just want to play a basic Pong game.
4. Include technical documentation, allowing advanced users to extend the game to change designs or add functionality.

This technical report describes the completed FPGA design of the Pong game project in detail. Section 2 presents the theory of operation of the game, including a high-level description of the system, a top-level diagram showing the organization of modules that compose the system, and individual descriptions of the modules. Section 3 details how the project test plan was carried out, demonstrating that the final design meets all of the requirements of the project. Supporting material, such as the full test plans and code listings may be found in the Appendix.

2 Theory of Operation

This section details the theory and technical design of the FPGA Pong game project.

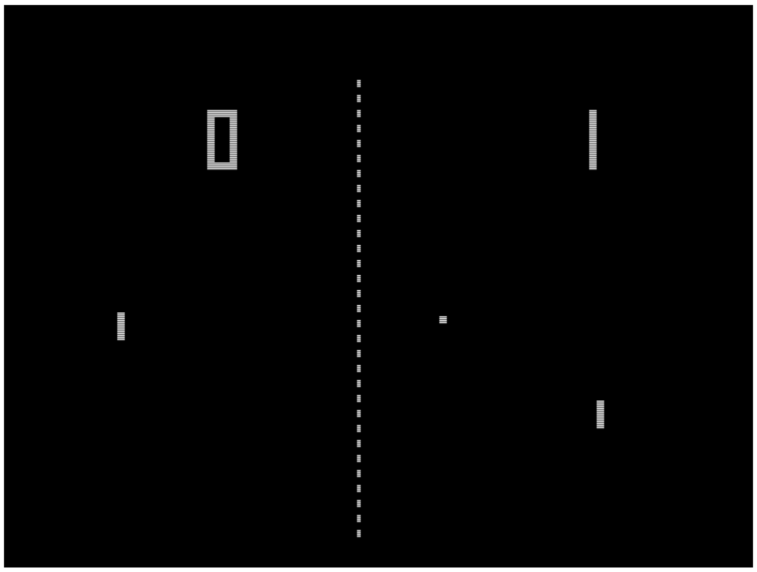

Figure 1: Desired operation of the Pong game.

2.1 High-Level Description

Figure 1 depicts the desired operation of the Pong game. The system is based on VGA output on a screen display and seven segment display for the score. The outputs are connected to a state machine that controls when objects on the screen move. The system varies its output based on different inputs. The operation responds based on user inputs as described below:

- Reset - This button unconditionally resets the game back to initial state.
- Left Up - This button activates upward movement for the left paddle. When active, the left side paddle, as shown in Figure 1, will move upwards.
- Left Down - This button activates downward movement for the left paddle. When active, the left side paddle, as shown in Figure 1, will move downwards.
- Right Up - This button activates upward movement for the right paddle. When active, the right side paddle, as shown in Figure 1, will move upwards.
- Right Down - This button activates downward movement for the right paddle. When active, the right side paddle, as shown in Figure 1, will move downwards.
- Serve - This button can be used to perform a serve. At the end of a round or beginning/end of the game, this button can be used to serve.

Internal details of the system in Figure 1 are described in Section 2.2.

2.2 Low-Level Description

This section describes the individual modules that make up the system shown in Figure 1. The top-level module and submodules were designed in SystemVerilog using the Xilinx Vivado design suite.

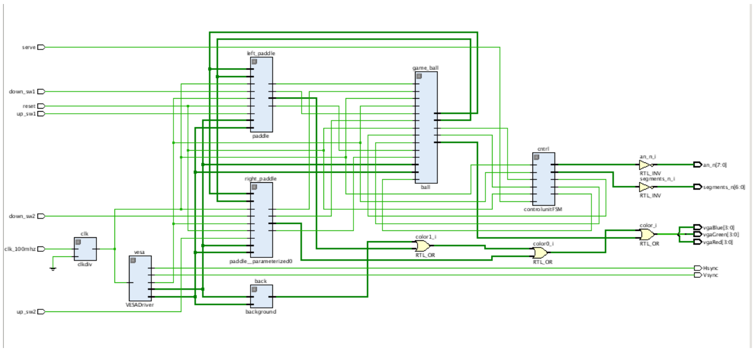

Figure 2: Top level module for the project.

2.2.1 Top-level Module

Figure 2 depicts the top-level module for the Pong game system. The external inputs to the design consist of the following:

clk_100mhz A 100 MHz clock signal provided by the FPGA board.

serve A pushbutton input that is used to serve.

reset A pushbutton input that is used to reset the game.

up_sw1 A switch used to control upward movement for the left paddle.

down_sw1 A switch used to control downward movement for the left paddle.

up_sw2 A switch used to control upward movement for the right paddle.

down_sw2 A switch used to control downward movement for the right paddle.

The design also provides the following outputs

an_n Anode control output for the seven segment display.

segments_n Segment control output for the seven segment display.

vgaBlue Blue color output for the VGA display.

vgaGreen Green color output for the VGA display.

vgaRed Red color output for the VGA display.

Hsync Horizontal sync output for the VGA display.

Vsync Vertical sync output for the VGA display.

The heart of the design is the control unit FSM which changes the VGA output every clock cycle based on the user inputs. The clock used to drive the FSM is set at 50 MHz. This clock is provided by the clock divider module that takes the 100 MHz clock as input and divides it by a factor of $N = 2$ to provide the 50 MHz clock signal to the FSM. For more information on the clock divider module clkdiven, please see the Lab 7 description on the ECE211 Moodle Website.

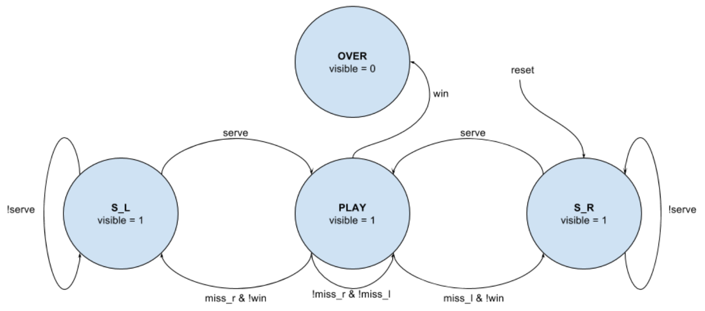

Figure 3: State transition diagram of the FSM.

2.2.2 Control Unit FSM

The Control Unit FSM module is used to control the state of the Pong game based on the previous state and state of user inputs. Figure 3 depicts a state-transition diagram for the T-Bird Turn Signal FSM.

The FSM operates as follows. The operation begins at an initial state S_R where the right paddle serves first. After serving, the state transitions to PLAY and does not change while the ball remains in play. If the right player misses, the state changes to S_L where the left player serves. If the left player misses, the state changes to S_R where the right player serves. These states change in such a manner until one of the players reaches a score of 11 after which the state transitions to OVER. This is the game over state and in order to restart the game, the reset input must be asserted at which point the game restarts from the S_R state again.

The code for this module can be found in Appendix B for convenience.

2.2.3 Paddle

The Paddle module is used to control the visible state and movement of the paddles based on user inputs.

The Paddle operates as follows. If no user inputs to the paddle are asserted the paddle position does not change. If either the up or down inputs to the paddle are asserted the paddle will move in the respective direction by 2 pixels for every clock cycle that the input is asserted. If both up and down are asserted at the same time the paddle will not move. The object itself is created using a movable rectangle generator.

The code for this module can be found in Appendix C for convenience.

2.2.4 Ball

The Ball module is used to control the visible state and movement of the ball based on collisions and position on the screen.

The Ball operates as follows. If the ball is not in contact with any wall or paddle it continues to move in the same direction. If the ball comes in contact with either wall it reverses its vertical velocity. If the ball ball comes in contact with either paddle it reverses its x velocity. In addition, if it hits the top third of a paddle its vertical velocity is decremented; if it hits the bottom third of a paddle its vertical velocity is incremented; if it hits the middle third of a paddle its vertical velocity does not change. The object itself is created using a movable rectangle generator.

The code for this module can be found in Appendix D for convenience.

2.2.5 Background

The Background module is used to display the top and bottom walls on the screen, as well as the dividing line in the middle of the screen.

The module prevents the ball and paddles from travelling past the walls on each side. The dividing line is created by only displaying alternating parts of a rectangle. The objects themselves are created using a stationary rectangle generator.

The code for this module can be found in Appendix E for convenience.

3 System Validation

The test plan from the FPGA design of the Pong game project is provided in Appendix A for convenience. Proper operation of the Pong game was validated by following this test plan as detailed in the table below.

Line	Action	Result	Pass/ Fail	Initial
1	Connected FPGA, powered on, turned all switches off.	Board powered up correctly.	PASS	nt
2	Downloaded bitstream file.	Programming successful.	PASS	nt
3	Checked display.	Game at initial state with ball in the center, both scores at zero.	PASS	nt
4	Asserted up input for left paddle for a few seconds.	Watched left paddle move up.	PASS	nt
5	Asserted down input for left paddle for a few seconds.	Watched left paddle move down.	PASS	nt
6	Asserted both up and down inputs for left paddle for a few seconds.	Watched left paddle not move.	PASS	nt
7	Asserted up input for right paddle for a few seconds.	Watched right paddle move up.	PASS	nt
8	Asserted down input for right paddle for a few seconds.	Watched right paddle move down.	PASS	nt

9	Asserted both up and down inputs for right paddle for a few seconds.	Watched right paddle not move.	PASS	nt
10	Assert the serve pushbutton.	Watched ball move towards the left.	PASS	nt
11	Allowed the game to play for several seconds.	Watched ball bounce off both walls and both paddles atleast once.	PASS	nt
12	Allow right player to miss the ball.	Watched the left score update by one.	PASS	nt
13	Assert the serve pushbutton.	Watched ball move towards the right.	PASS	nt
14	Allow left player to miss the ball.	Watched the right score update by one.	PASS	nt
15	Assert the serve pushbutton.	Watched ball move towards the left.	PASS	nt
16	Allow the game to play until one of the players reaches a score of 11.	The ball should not be visible on screen.	PASS	nt
17	Assert the serve pushbutton.	Nothing happens.	PASS	nt
18	Assert the reset pushbutton.	Game at initial state with ball in the center, both scores at zero.	PASS	nt

A Test Plan

This appendix contains a test plan for the FPGA Pong game project. The test plan was created to test each of the required specifications of the project.

1. Connect a NEXYS-4 FPGA board to the lab computer using a provided USB cable. Ensure the FPGA board is powered on and all the siwtches are off. Verify that the LEDs on the board light up properly, showing that the board is ready to be programmed.
2. Use Vivado to download the provided relevant bitstream file to the FPGA board. Check the Vivado status to ensure that programming was successful.
3. Verify the initial state by making sure the ball is in the center and both scores are at zero.
4. Assert the up input for the left paddle and see that the paddle moves up.
5. Assert the down input for the left paddle and see that the paddle moves down.
6. Assert both the up and down inputs for the left paddle and see that the paddle does not move.
7. Assert the up input for the right paddle and see that the paddle moves up.
8. Assert the down input for the right paddle and see that the paddle moves down.
9. Assert both the up and down inputs for the right paddle and see that the paddle does not move.
10. Assert the serve pushbutton and see the ball serve towards the left side.
11. Allow the game to play and see that the ball bounces of both walls and paddles.

12. Allow the right player to miss the ball and see that the left score updates.
13. Assert the serve pushbutton and see the ball serve towards the right side.
14. Allow the left player to miss the ball and see that the right score updates.
15. Assert the serve pushbutton and see the ball serve towards the left side.
16. Allow the game to play until one of the scores reaches 11 and see that the ball is no longer visible.
17. Assert the serve pushbutton and see that nothing happens.
18. Assert the reset pushbutton and see that the game is in the initial state by making sure the ball is in the center and both scores are at zero.

B Control Unit FSM Code

```
module controlunitFSM(
    input logic serve,
    input logic reset,
    input logic clk,
    output logic visible,
    output logic srv_l,
    output logic srv_r,
    input logic miss_l,
    input logic miss_r,
    input logic frame,
    output logic [7:0] an,
    output logic [6:0] segments
    );

    typedef enum logic [1:0] {S_L, S_R, PLAY, OVER} state_type;
       state_type state, next;

       logic update_left;
       logic update_right;
       logic win;

    sevenseg_control control(.win (win), .clk(clk), .an(an), .segments(segments), .reset(reset

       always_ff @(posedge clk)
       if(reset) begin state <= S_R; end
       else if(frame) state <= next;

       always_comb
       begin
       srv_l = 1'b0;
       srv_r = 1'b0;
```

```
visible = 1'b0;
update_left = 1'b0;
update_right = 1'b0;
next = S_R;
case(state)
 PLAY:

     begin
         visible = 1'b1;
         if(miss_r)
         begin
         update_left = 1'b1;
         if(win)
         next = OVER;
         else
         next = S_L;
         end
         else if(miss_l)
         begin
         update_right = 1'b1;
         if(win)
         next = OVER;
         else
         next = S_R;
         end
         else
         next = PLAY;
     end
 S_R:
     begin
         visible = 1'b1;
         if(serve)
         begin
         srv_r = 1'b1;
         next = PLAY;
         end
         else
         next = S_R;
     end
 S_L:
         begin
             visible = 1'b1;
             if(serve)
             begin
             srv_l = 1'b1;
             next = PLAY;
             end
             else
```

```
                    next = S_L;
                end
          OVER:
                begin
                    visible = 1'b0;
                    next = OVER;
                end
           endcase
         end
endmodule
```

C Paddle Code

```
module paddle(
    input logic reset,
    input logic up_I,
    input logic down_I,
    input logic [9:0] vga_x,
    input logic [9:0] vga_y,
    output logic [11:0] rgb,
    input logic frame,
    input logic clk,
    output logic tophit,
    output logic midhit,
    output logic bothit,
    input logic [9:0] ball_x,
    input logic [9:0] ball_y
    );

    parameter XLOC = 0;
    parameter COLOR = 12'hfff;
    parameter OFFSET = 0;

    mvrectgen #(.OFFSET(OFFSET), .XLOC(XLOC), .YLOC(), .WIDTH(), .HEIGHT(), .COLOR(COLOR), .UP

endmodule

module mvrectgen(
    input logic clk,
    input logic reset,
    input logic en,
    input logic [9:0] x,
    input logic [9:0] y,
    input logic up,
    input logic down,
    output logic [11:0] rgb,
    input logic [9:0] ball_x,
 input logic [9:0] ball_y,
 output logic tophit,
 output logic midhit,
 output logic bothit
 );

parameter WIDTH = 10; // dimension
parameter HEIGHT = 75;
parameter OFFSET = 0;
parameter XLOC = 0; // location on reset
parameter YLOC = (600-HEIGHT)/2;
parameter COLOR = 12'hfff; // output color
parameter UP_LIMIT = 0;
parameter DOWN_LIMIT = 600;

logic [9:0] Y_LOC_new = YLOC;

    always_ff @(posedge clk)
    begin
    if (reset) begin Y_LOC_new <= YLOC; end
    else if(en && up && ~down)
     begin
         if(Y_LOC_new <= UP_LIMIT)
             Y_LOC_new <= Y_LOC_new;
         else
             Y_LOC_new <= Y_LOC_new - 2;
     end
    else if(en && down && ~up)
      begin
          if(Y_LOC_new >= (DOWN_LIMIT-HEIGHT))
              Y_LOC_new <= Y_LOC_new;
          else
              Y_LOC_new <= Y_LOC_new + 2;
      end
     end

     always_comb
     begin

         tophit <= 1'b0;
         midhit <= 1'b0;
         bothit <= 1'b0;

             if(((ball_x+OFFSET) >= XLOC) && ((ball_x+OFFSET) <= XLOC + WIDTH))
             begin

                 if(ball_y >= (Y_LOC_new) && ball_y < (Y_LOC_new + (HEIGHT/3)))
                     tophit <= 1'b1;
```

```
                    else if(ball_y >= (Y_LOC_new + (HEIGHT/3)) && ball_y < (Y_LOC_new + (2*HEI
                        midhit <= 1'b1;
                    else if(ball_y >= (Y_LOC_new + (2*HEIGHT/3)) && ball_y < (Y_LOC_new + (HEI
                        bothit <= 1'b1;

                end


        end


      always_comb
      begin
      if(x >= (XLOC) && x < (XLOC + WIDTH) && y >= (Y_LOC_new) && y < (Y_LOC_new + HEIGHT))
         rgb = COLOR;
      else
         rgb = 12'h000;
      end


endmodule
```

D Ball Code

```
module ball(
    input logic reset,
    input logic tophit_l,
    input logic midhit_l,
    input logic bothit_l,
    input logic tophit_r,
    input logic midhit_r,
    input logic bothit_r,
    input logic clk,
    input logic frame,
    input logic [9:0] vga_x,
    input logic [9:0] vga_y,
    output logic [11:0] rgb,
    input logic visible,
    input logic srv_l,
    input logic srv_r,
    output logic miss_l,
    output logic miss_r,
    output logic [9:0] ball_x,
    output logic [9:0] ball_y
    );

   parameter WIDTH = 10; // dimension
parameter HEIGHT = 10;
parameter XLOC = 400; // location on reset
parameter YLOC = 300;
parameter COLOR = 12'hff0; // output color
parameter UP_LIMIT = 0;
parameter DOWN_LIMIT = 600;
parameter LEFT_LIMIT = 0;
parameter RIGHT_LIMIT = 800;

logic [9:0] Y_LOC_new = YLOC;
logic [9:0] X_LOC_new = XLOC;
logic [9:0] X_vel = 0;
logic [9:0] Y_vel = 0;
logic [2:0] speed_counter = 1;
logic y_direction_counter = 0;

always_ff @(posedge clk)
begin
if(frame)
begin
y_direction_counter <= y_direction_counter + 1;
if(speed_counter == 5)
speed_counter <= 1;
else
speed_counter <= speed_counter + 1;
end
end


 always_ff @(posedge clk)
 begin

     if (reset)
         begin
         Y_LOC_new <= YLOC;
         X_LOC_new <= XLOC;
         X_vel <= 0;
         Y_vel <= 0;
         miss_l <= 0;
         miss_r <= 0;
         end

     else if(frame)
         begin
         miss_l <= 0;
         miss_r <= 0;

         if(srv_l)
```

```
    begin
    X_vel <= -2;
    if(y_direction_counter == 1)
        Y_vel <= speed_counter;
    else
        Y_vel <= -speed_counter;

    X_LOC_new <= X_LOC_new + X_vel;
    Y_LOC_new <= Y_LOC_new + Y_vel;
    end

else if(srv_r)
    begin
    X_vel <= 2;
    if(y_direction_counter == 1)
        Y_vel <= speed_counter;
    else
        Y_vel <= -speed_counter;

    X_LOC_new <= X_LOC_new + X_vel;
    Y_LOC_new <= Y_LOC_new + Y_vel;
    end

else
    begin

    if(X_LOC_new <= (LEFT_LIMIT+30))
        begin
        miss_l <= 1'b1;
        Y_LOC_new <= YLOC;
        X_LOC_new <= XLOC;
        X_vel <= 0;
        Y_vel <= 0;
        end

    else if(X_LOC_new >= (RIGHT_LIMIT-30))
        begin
        miss_r <= 1'b1;
        Y_LOC_new <= YLOC;
        X_LOC_new <= XLOC;
        X_vel <= 0;
        Y_vel <= 0;
        end

    else if(Y_LOC_new <= UP_LIMIT || Y_LOC_new >= (DOWN_LIMIT-HEIGHT))
        begin
        Y_vel <= -Y_vel;
        X_LOC_new <= X_LOC_new + X_vel;
```

```
                Y_LOC_new <= Y_LOC_new - Y_vel;
                end

            else if(tophit_l || tophit_r)
                begin
                X_vel <= -X_vel;
                X_LOC_new <= X_LOC_new - (X_vel*2);
                if(Y_vel>1)
                    Y_vel <= Y_vel-1;
                else if(Y_vel<1)
                    Y_vel <= Y_vel+1;
                end

            else if(midhit_l || midhit_r)
                begin
                X_vel <= -X_vel;
                X_LOC_new <= X_LOC_new - (X_vel*2);
                end

            else if(bothit_l || bothit_r)
                begin
                X_vel <= -X_vel;
                X_LOC_new <= X_LOC_new - (X_vel*2);
                if(Y_vel>0 && Y_vel<5)
                    Y_vel <= Y_vel+1;
                else if(Y_vel<0 && Y_vel>-5)
                    Y_vel <= Y_vel-1;
                end

            else
                begin
                X_LOC_new <= X_LOC_new + X_vel;
                Y_LOC_new <= Y_LOC_new + Y_vel;
                end
            end
        end
 end


always_comb
begin
ball_x = X_LOC_new;
ball_y = Y_LOC_new;
if(vga_x >= (X_LOC_new) && vga_x < (X_LOC_new + WIDTH) && vga_y >= (Y_LOC_new) && vga_y <
   rgb = COLOR;
else
   rgb = 12'h000;
```

```
    end

endmodule
```

E Background Code

```
module background(
    input logic [9:0] vga_x,
    input logic [9:0] vga_y,
    output logic [11:0] rgb
    );

    logic [11:0] rgb1;
    logic [11:0] rgb2;
    logic [11:0] rgb3;
    logic [11:0] rgb3_n;
    logic [11:0] alt_bit;
    assign alt_bit = {12{vga_y[4]}};

    rectgen #(.XLOC(0), .YLOC(128), .WIDTH(799), .HEIGHT(8), .COLOR(12'hfff)) top_wall(.x(vga_
    rectgen #(.XLOC(0), .YLOC(512), .WIDTH(799), .HEIGHT(8), .COLOR(12'hfff)) bottom_wall(.x(v
    rectgen #(.XLOC(396), .YLOC(136), .WIDTH(8), .HEIGHT(376), .COLOR(12'hfff)) net(.x(vga_x),


    assign rgb3_n = rgb3 & alt_bit;
    assign rgb = rgb1 | rgb2 | rgb3_n;


endmodule

module rectgen(
    input logic [9:0] x,
    input logic [9:0] y,
    output logic [11:0] rgb
    );

     // Upper-left-hand corner
   parameter XLOC = 100;
   parameter YLOC = 100;
   // Dimensions
   parameter WIDTH = 100;
   parameter HEIGHT = 100;
   // RGB
   parameter COLOR = 12'hf00;

   always_comb
     if(x >= XLOC && x < (XLOC + WIDTH) && y >= YLOC && y < (YLOC + HEIGHT))
```

```
            rgb = COLOR;
        else
            rgb = 12'h000;


endmodule
```