EE314

PINBALL PROJECT

Yunus ÇAY, Alperen CEBECİK
Middle East Technical University
Department of Electrical and
Electronics Engineering
Ankara,Turkey
e2261634@metu.edu.tr,
e2166148@metu.edu.tr

***Abstract*—This report contains the overall results of the EE314 Digital Electronics Laboratory project. Project's name is Pinball Game with FPGA-Verilog. Results and content of this project is included in this report.**

Keywords—FPGA, Verilog, VGA, Pinball

I. INTRODUCTION

Today's world is changed very fast and the requirements are replaced with more technological ones. Hence, human needs to also more well-designed equipments to handle issues about their lives. These equipments can be anything; mechanical, electrical stuffs…, but the first comes to mind is computers

The invention of the computers is a significant point for the world and human lives have been affected by this invention from the late 1970's. Of course, when we say computer, we are not just referring to PC. There is a huge variety of computers.

This variety has been occurred due to different needs for human. However, sometimes, we need very specialized machines. For this case, the machine which can be adjusted to provide proper work is needed. This machine is already constructed and its name is FPGA.

FPGA stands for Field Programmable Gate Array. This type of computer is constructed with million and millions of gates or transistors. To specialize the FPGA machine, Verilog language is used. Verilog language is written according to sequential implementation.

As a result, FPGA is a type of computer and any desired logical implementation can be implemented with FPGA and Verilog language. In this way, any need can be handled.

II. COMPONENTS AND CONSTRAINTS

A. *Constraints*

- Plunger: initial movement.
- Flippers with continuous movement
- Specialized targets for score and scattering
- Energy loss of ball
- Collision restrictions

B. *Components*

- FPGA
- VGA cable and Monitor
- Computer and Quartus Software

III. PROJECT PARTS

A. *VGA Synchronization*

In the project, the first thing to be solved is creating proper VGA signal to monitor.

Monitor gives a sight to outside by using pixels which is the smallest increment giving adjusted light. All variations of colors can be obtained with mix of red, green and blue.

The signals we can say that it is continuous have frequency about 60Hz. Therefore, for example, 640*480 pixel monitor pixels should be changed with 60 Hz to enjoy film, which is provided by VGA synchronization.

This synchronization is done with 25MHz clock generated by FPGA.

$$25MHz/(800*524) = 60Hz \qquad (1)$$

800 is the summation of 640 pixel , H.Sync, Back Porch and Front Porch in the Equation 1. 524 is also obtained like that. As we can see in the Equation 1, all pixels are refreshed with frequency 60Hz.

```
always @ (posedge clk_in)       // VGA Sync.
begin
  if(clk_25)
  begin
  if(en_h)
  begin
     if(Counter_X < 799)
     begin
     Counter_X <= Counter_X + 1;
     en_v <=0;
     end else
        begin
           Counter_X <=0;
           en_h <=0;
           en_v <=1;
        end
  end   else
           if(en_v) begin
           if (Counter_Y < 524) begin
              Counter_Y <= Counter_Y +1;
              en_h <=1;
              end else
              begin
                 Counter_Y <= 0;
                 en_v <= 0;
                 en_h <=1;
        end end end
  end
```

Figure 1 Vga Synchronization

```
  end

always @ (posedge clk_in)
    begin
    if(clk_25)
    begin
    if (Counter_X>94)
    vga_h_sync <=1;
      else vga_h_sync <=0;
        if (Counter_Y>1)
          vga_v_sync<= 1;
          else vga_v_sync<=0;

    end end                          //////// End Of the VGA Sync
```

Figure 2 Vga Synchronization

In Figure 1&2,Counter_X is used for horizontal numeration and Counter_Y is used for vertical numerization. Clk_25 is a clock signal having 25MHz frequency.

B. Clock Divider

FPGA have 50 MHz internal clock signal which is used for all clock divisions. There are 3 type clock division and with these division, 3 type clock signal is obtained with respect frequency.

```
 always @ (posedge clk_in)    // Constructing a 25 MHz signal from the 50MHz signal
  begin
     clk_25 <= ~clk_25;
  end

always @(posedge clk_in) begin     // 50 Hz clock for moving ball
  counter_real = counter_real +1;
  if(counter_real == 999999) begin
     clk_real = ~clk_real;
     clk_real_enable=1;
     counter_real = 0;
  end
  else begin
  clk_real_enable=0;
  end
end

always @ (posedge clk_in) begin          // 1 Hz Clock
  counter_second = counter_second +1;
  if(counter_second == 49999999) begin
     clk_second = ~clk_second;
     clk_second_enable = 1;
     counter_second = 0;
  end
  else begin
  clk_second_enable = 0;
  end
end
```

Figure 3 Clk_25,Clk_real and Clk_sec Construction

First one is enabler signal having 25 MHz signal. It is used for VGA synchronization. In the Figure 3, clk_25 is the enabler signal. Using sequential property, enabler signal is created.

Second one is the clock used to provide the ball with movement.

As seen on the Figure 3, this clk_real signal is constructed with counter and 50MHz internal clock signal. Counter duty is divide the internal clock and after a some positive value counter is reseted.

Third clock is used for timer. In other words, this clock signal has 1Hz frequency. This signal is used for the timer part of the project.

In the Figure 3, as we can see the clock is divided to itself to get 1 Hz signal.

C. Timer

Timer is constructed using decimal to digit. After a first counter reaches to 9, it will be reseted and second counter becomes 1. It will continues to 999 point and then whole counters are reseted.

```

always @ (posedge clk_in) begin              //Timer counting up
  if(clk_second_enable) begin
     if(ss != 9)begin
        ss = ss + 1;
     end
     else if (ss==9 && mm != 9) begin
        ss = 0;
        mm = mm + 1;
     end
     else if( ss == 9 && mm == 9 && hh != 9) begin
        ss = 0;
        mm= 0;
        hh = hh +1 ;
     end
end
end
```

Figure 4 Constructing a Timer

.

D. Boundaries and Specialized Shapes

Monitor is divided in three parts and whole visible screen have border frame.

Monitor is divided into three three because timer, score table and play-area is needed. These internal borders are created with simple compare and logical symbols.

```
always @ (posedge clk_in)                ////// Borders
if(clk_25) begin
begin
  display_check <= ((Counter_X>150 && Counter_X<500) || (Counter_Y>45&& Counter_Y<505))?1:0;

  border_h <= ((Counter_Y>40&& Counter_Y<45) || (Counter_Y>505 && Counter_Y<510) ||  ((Counter_X>500 && Counter_X<780)
  &&(Counter_Y>365 && Counter_Y<370)))?1:0;

  border_v <= (Counter_X>145 && Counter_X<150) || (Counter_X>775 && Counter_X<780) ||
  (Counter_X>500 && Counter_X<505)?1:0;

  border_left <= ((Counter_Y == Counter_X + 145 ) && ((Counter_X> 150) && (Counter_X<270)) && ((Counter_Y>295) && (Counter_Y

  border_right <= (((-1)* Counter_Y == Counter_X -795 ) && ((Counter_X> 380) && (Counter_X<500)) && ((Counter_Y>295) && (Cou
end
end
```

Figure 5 Horizontal, Vertical and Inclined Borders

As seen on the Figure 5, border_h and border_y are equal to 1 and when the Counter_X and _Counter_Y are inside determined areas on border_h and border_y. Then, on the part VGA output colours are assigned, borders become white.

Again on the same Figure 5, the inclined boundaries inside play area are constructed with line equaiton. The starting and end points are known, also slope is known. Then, comparison for Counter_X&Y with border start-end coordinates is enough to create boundary.

Specialized shapes are two type circles and hexagon.

Ball = (Counter_X – ball_x)^2 + (Counter_ Y– ball_y)^2

= R^2 (2)

```
always @  (posedge clk_in) // green and red balls
begin
    if (clk_25 && display_check )
      begin

        ball_green1 <= ((Counter_X-ball_g1x)**2 + (Counter_Y-ball_g1y)**2 < 20*20)?1:0;
        ball_green2 <= ((Counter_X-ball_g2x)**2 + (Counter_Y-ball_g2y)**2 < 20*20)?1:0;
        ball_red1 <= ((Counter_X-ball_r1x)**2 + (Counter_Y-ball_r1y)**2 < 20*20)?1:0;
        ball_red2 <= ((Counter_X-ball_r2x)**2 + (Counter_Y-ball_r2y)**2 < 20*20)?1:0;
      end
end
```

Figure 6 Construction of Green and Red balls

Circles are created with standard circle equation as shown in the Equation 2. Mid point of the circle are assigned to ball_x and _ball_y. Then, as in the case for borders, comparison with counter values and range from the mid values colours are determined.

Hexagon shape is determined like circular one, after a mid point is selected, the equation for square is applied. Then, we devolop areas overlapping on each other, but outside of the shape is alike hexagon.

E. Ball Movement and interactions

Ball movement is provided with using mid-point of ball and increments. For ball movement, we increase the direction of ball with increments under effect of the clk_real signal (50Hz). After giving a initial value with a plunger, ball starts to move in random direction. Then, direction of the ball is adjusted with collisions.

```
always @(posedge clk_in) begin     // animated yellow ball
    ball_yellow <= ((Counter_X - yellow_x)**2 + (Counter_Y - yellow_y)**2 < 10*10)?1:0;
    if(clk_real_enable) begin
    yellow_x = yellow_x + yellow_xinc;
    yellow_y = yellow_y + yellow_yinc;
    end
      if (yellow_x > 490 || yellow_x < 160 ) begin // left right bounds
        yellow_xinc = (-1) * yellow_xinc;
      end

      if (yellow_y < 55 || yellow_y >495) begin // top bottom bounds
        yellow_yinc = (-1) * yellow_yinc;
      end

      if ( (yellow_y <  yellow_x + 145) && (yellow_x >150) && (yellow_x <270) && (yellow_y>295) && (yellow_y <415)) begin  // left slope
        swap_holder1 = yellow_xinc;
        yellow_xinc = yellow_yinc;
        yellow_yinc = swap_holder1 * (-1);
      end

      if (( (-1)* yellow_y < yellow_x -795 ) && (yellow_x >380) && (yellow_x <500) && (yellow_y>295) && (yellow_y <415)) begin // right s
        swap_holder2 = yellow_xinc ;
        yellow_xinc = yellow_yinc * (-1);
        yellow_yinc = swap_holder2;
      end

end
```

Figure 7 Animated Ball

Collision detectors are constructed with comparison test. For example magnitude of ball mid point_x – border_x (right border) is less than the radial length of ball, ball mid point_x is reversed and y is kept as fixed value. All collision detectors are created by using same logic.

Another issue about the ball movement is plunger. Plunger provides the ball with random start. It is succeeded with using random function.

F. *Flippers*

There are two flippers in play-area. Flippers are used to force the ball back. Flippers are constructed using the logic: first y the point on the border side is fixed a point and also length of the ball is fixed a positive value. Then, increasing and decreasing the mid point_y of flipper border changes the position of the flipper under the effect of the clock signal having sufficient frequency(50-60Hz)

G. *VGA Outputs and Colours*

In this part, according to the border information, VGA inputs are assigned with a 8-bit color array.

As mentioned above borders are colored with white, 2 balls are with green and other 2 with red, hexagon is with blue and others we have not yet assigned are colored with black.

```
533
534  wire timer_display = s1||s2||s3||s4||s5||s6||s7||m1||m2||m3||m4||m5||m6||m7||h1||h2
535
536     ///// colour outputs
537     always @ (posedge clk_in )
538     begin
539     if (clk_25)
540     begin
541        if (border_h==1||border_v==1 || border_left==1 || border_right ==1)begin
542        VGA_R <=8'b11111111;
543        VGA_G <=8'b11111111;
544        VGA_B <=8'b11111111;
545        end else
546        if (ball_green1==1||ball_green2==1)
547        begin
548        VGA_R <=8'b00000000;
549        VGA_G <=8'b11111111;
550        VGA_B <=8'b00000000;
551        end else
552           if (ball_red1==1||ball_red2==1)
553           begin
554        VGA_R <=8'b11111111;
555        VGA_G <=8'b00000000;
556        VGA_B <=8'b00000000;
557        end else
558        if (ball_yellow ==1) begin
559        VGA_R <=8'b11111111;
560        VGA_G <=8'b11111111;
561        VGA_B <=8'b00000000;
562        end else
```

Figure 8. Outputs

```
562        end else
563        if (timer_display ==1) begin
564        VGA_R <=8'b11111111;
565        VGA_G <=8'b11111111;
566        VGA_B <=8'b11111111;
567        end
568        else begin
569        VGA_R <=8'b00000000;
570        VGA_G <=8'b00000000;
571        VGA_B <=8'b00000000;
572        end
573
574     end
575     end
576
577  endmodule
578
```

Figure 9. Outputs (Continued)

H. *Conclusion*

In this project, we used FPGA to design a simple Pinball game. Using VGA synchronization, we were able to get an image on monitor. We created 25 MHz, 50 Hz and 1 Hz clocks from our FPGA's internal 50 MHz clock. 25 MHz clock was used to synchronize VGA with FPGA so that we can scan our monitor. 50 Hz was used to animate yellow ball, 1 Hz was used in timer as a second indicator. As ball moves in display area, it interacted with area boundaries, flippers and objects such as red, green circles and blue hexagons. With hitting these objects, user scored points or received penalties. In this project; we got an opportunity to create a product with our existing FPGA and Verilog knowledge, also got familiar with visualisation with VGA. It was a rewarding task.

REFERENCES

[1] FPGA VGA Graphics in Verilog Part 1. (2019). Retrieved from https://timetoexplore.net/blog/arty-fpga-vga-verilog-01

[2] Verilog case statement example. (2019). Retrieved from http://referencedesigner.com/tutorials/verilog/verilog_18.php

APPENDIX A:

```
module screen (clk_in, vga_h_sync, vga_v_sync, VGA_R, VGA_G, VGA_B, clk_25);
 input clk_in ;
 output reg vga_h_sync, vga_v_sync ;

 reg [9:0] Counter_X;   // This is needed for horizontal sync.
 reg [9:0] Counter_Y;   // This is needed for vertical sync.
 reg border_h ;            // Boundary for horizontal
 reg border_v ;         // Boundary for vertical
 reg border_left;       // Boundary left
 reg border_right;      //boundary right
 reg display_check;     // control the boundary area
 reg [9:0] yellow_x , yellow_xinc , yellow_y , yellow_yinc ;

 reg ball_green1,ball_green2,ball_red1,ball_red2 , ball_yellow;

 output reg [7:0] VGA_R ;     // Output of the code and input of the VGA red input
 output reg [7:0] VGA_G ;     // Output of the code and input of the VGA blue input
 output reg [7:0] VGA_B ;     // Output of the code and input of the VGA green input
 output reg clk_25;
 reg clk_real;                //  clock for the timer
 reg clk_second;
 reg en_v;                    // enables for VGA sync
 reg en_h;
 reg [30:0] counter_real;          // real clock counter
 reg [30:0] counter_second;           // real clock counter
 reg clk_real_enable , clk_second_enable;
 reg swap_holder1, swap_holder2;                 //placeholder for swapping slopes
 reg [9:0] ball_g1x,ball_g1y,ball_g2x,ball_g2y; // Middle point of the green balls
 reg [9:0] ball_r1x,ball_r1y,ball_r2x,ball_r2y; // Middle point of the red balls

 reg [3:0] hh, mm , ss ; //Timers for hours, minutes, seconds
 reg h1, h2, h3, h4, h5, h6, h7;
 reg m1, m2, m3, m4, m5, m6, m7;
 reg s1, s2, s3, s4, s5, s6, s7;
```

Figure 10 . Module Declaration