EE314 TERM PROJECT - PINBALL

Burak Topçu -- 2167385
Doğa Tolgay – 2141323
Group number: 109

***Abstract*—**

As a term project for the laboratory of digital electronics and logic design courses, students have to create a game known as pinball. This game is played with 2 flipper that is controlled from out. According to ball movement, some collisions occur on the game and, with respect to these collisions player will win or lose points. At the end of game, gamer reach a point that shows us to success of player. This game will be created by using FPGA's and Verilog coding that manage the logic circuits in FPGA. After the game exists, students have to play this game on a monitor which is connected to the FPGA through VGA. In this project, all of the students learn how VGA operates and how the images on the monitor will appear. In addition to controlling VGA via FPGA, students learn using and manipulating the internal circuits of FPGA with the help of codes written on the Verilog of the Quartus program. As a result, reader will learn some specific how Verilog codes operates and syntax of this language, how to get an image through VGA and how to move the solids which is appeared on the screen by manipulating the codes and assigning the colors of VGA with respect to moving particle.

***Index Terms*— Pinball, VGA, FPGA, Verilog, Quartus.**

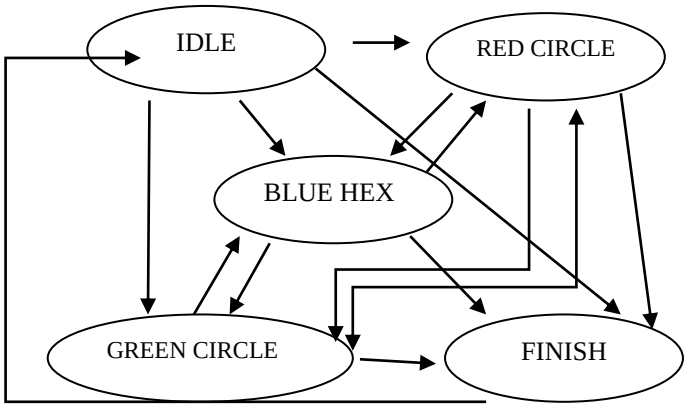
I. INTRODUCTION

In this project we are expected to make a pinball game. Pinball is an arcade game that is played in pinball machine; however we will implement it into a screen and control the flippers via FPGA buttons. On the screen, there will be a plunger, and when the start button is opened the ball will move randomly on the game screen. There are also 4 balls and 2 hexagons. Two of the hexagon will be green and other ones will be red. Also, hexagons will be blue. The player will gain +10 and +20 points from the red circles and blue hexagons respectively by colliding them. However, if the ball collides the green circles, that results in -15 points. With respect to the gained or lost points, scoreboard will follow the results and reveal it on the monitor screen. Also, we have to put a timer counter that will show us timing of game.

In the project, we will have some subunits that are mainly VGA subunit, code that includes solids such as borders, movement codes for ball and flipper, codes that designed for collisions, time and score codes.

In order to control the game from one code, we did not write our code as partially, and collect them into a top module. Instead of that, we wrote our code on a one module that operates all these sub modules.

II. STATE DIAGRAM



Also ball can move from red to red, from blue to blue and from green to green.

III. VGA SUBUNIT

Video graphics array mostly known as VGA for our project concludes 640*480 pixels. That means we have 480 lines, and we have 640 pixels for each line. These lines and pixels are controlled by horizontal and vertical synchronizations. Also the color that result for each pixel will be controlled code. With respect to code, color cables on the VGA transmit the code to the screen (unknown, 2019).

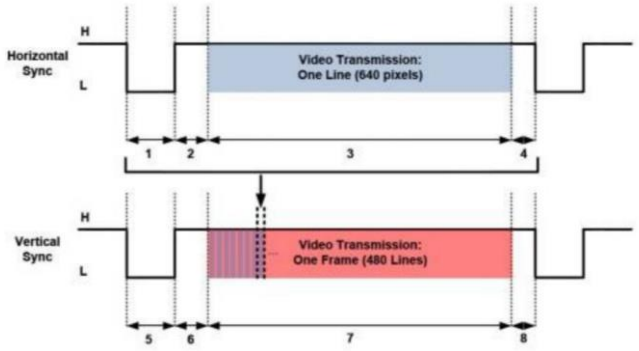


Figure 1: Horizontal and Vertical synchronization (unknown, 2019)

Timeline # on Fig. 1	Name	Duration	Clock Count
1	H. Sync	3.84 µs	96
2	Back Porch (H)	1.92 µs	48
3	Video Signal (One Line)	25.6 µs	640
4	Front Porch (H)	0.64 µs	16
5	V. Sync	0.064 ms	2
6	Back Porch (V)	1.056 ms	33
7	Video Signal (One Frame)	15.36 ms	480
8	Front Porch (V)	0.32 ms	10

Figure 2: Timings of the above Horizontal and Vertical synchronizations (unknown, 2019)

As one can see above synchronization graphs, horizontal part of the transmitted signal will be high for the interval 3. That means, user can get a meaningful signal for between 144-744 clocks. Also, vertical synchronization operate properly for interval 7. But, VGA starts controlling the pixels from top of the left and finish it at the right below point. Thus, the interval identified at 7 that is 35-515 takes more time with respect to horizontal synchronization.

In order to reach and use this meaningful portion of the clock we create two different 10-bits registers as CounterX and CounterY. The reason that we chose as 10 bit is that we count for both of the synchronizations over $512=2^9$ means that 9 bit. Since we do not use the above counts for the horizontal 800 and for the vertical 525, we setup an algorithm that will assign 0 above 800 for horizontal and 0 above 525 for vertical synchronization. You can see the written code below for the above situation.

```
8     wire CounterXmaxed = (CounterX == 799); // 16 + 48 + 96 + 640
9     wire CounterYmaxed = (CounterY == 524); // 10 + 2 + 33 + 480
10    always @(posedge clk) begin
11      clk_en <= !clk_en;
12    end
13    always @(posedge clk)
14      begin
15      if (CounterXmaxed)
16        CounterX <= 0;
17      else
18        CounterX <= CounterX + 1;
19    end
20
21    always @(posedge clk)
22    begin
23        if ((CounterY==1)&&(CounterX==799))
24          begin
25             vga_VerticalSynch <= (CounterY < (524))&&( CounterY ==(1));
26             CounterY <= CounterY + 1;
27          end
28        else if (CounterXmaxed)
29          begin
30             if (CounterYmaxed)
31             CounterY <=0;
32             else
33             CounterY <= CounterY + 1;
34        end
35        else
36          begin
37             vga_VerticalSynch <= ((CounterY < (524))&&( CounterY >(1)));
38          end
39          vga_HorizontalSynch <= ((CounterX > (94))&& CounterX < (799));   //
40    end
41    assign vga_h_sync = ~vga_HorizontalSynch; //(CounterX>95) ? 1'b1:1'b0;
42    assign vga_v_sync = ~vga_VerticalSynch;   //(CounterY>1) ? 1'b1:1'b0;
43  endmodule
```

Figure 3: VGA code written on Verilog

Vga_HorizontalSynch and Vga_VerticalSynch are obtained as we expected. Since synchronization inputs of VGA operate in active low mode, we invert the data of Vga_HorizontalSynch and Vga_VerticalSynch, and assign them into vga_h_sync and vga_v_sync. Afterwards, we run the program and observe the white image on the monitor. Below, you can also see the Modelsim results of this VGA code.

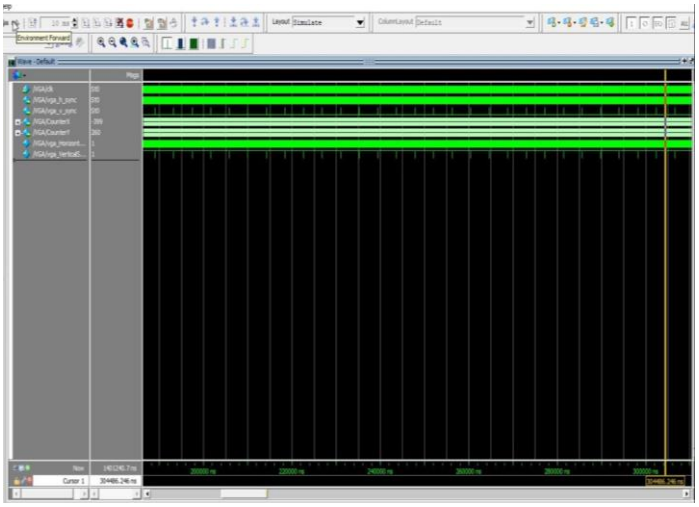


Figure 4: Modelsim simulation to observe vertical synchronization

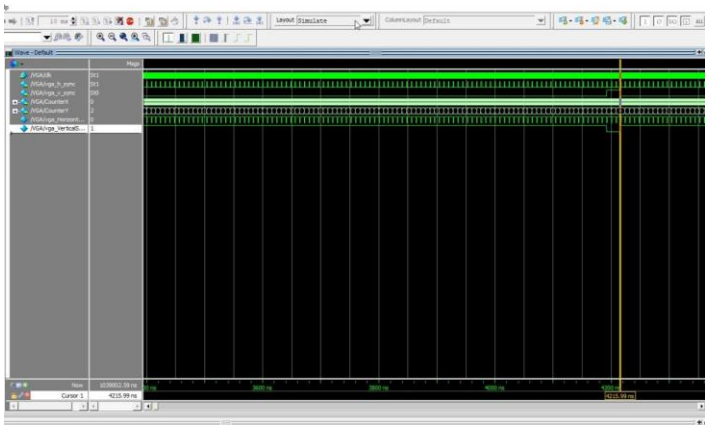


Figure 5:Modelsim simulation to observe horizontal synchronization at CounterX returns to 0, CounterY increase by 1.

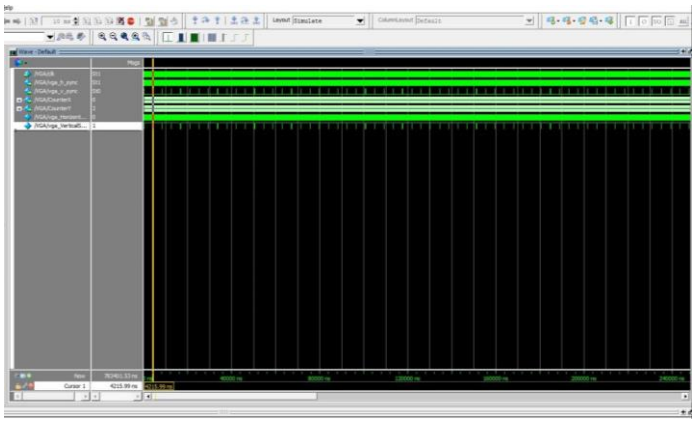


Figure 6: Modelsim simulation to observe vertical synchronization

As a result, after synchronizations of VGA are adjusted, this code run and then we can observe white screen on monitor.

IV. SOLIDS AND MOVEMENTS

A. Solids

On the game screen, there needs to be borders, scoreboard and timing, hexagons and circles.

The algorithm for the circles are obtained with respect to the circle equation that is simply $x^2+y^2=r^2$. For the circles we identified some specific centers and radius for all circles equals to 20 bit.

The algorithm for the hexagons are a bit complicated compared to the circles. Firstly, we write a code in order to get square on the screen. Then upward and downward of the square, we draw four different triangular. Obtaining square is simple since it will be identified with respect to x and y limits. With respect to these limits, we draw 4 different lines to limit the triangles. By choosing the area between lines and squares, triangles are obtained, and afterwards integrated part of triangles and square as shown a hexagon on the monitor.

The algorithm for borders is identified with respect to x and y limits of the screen. In a same manner, scoreboard and timing board are obtained as you can see below. Also the code for above algorithms is shared below.

The game should end after a time therefore a timer is created. This timer is done by using the idea of the seven segment display. Just like seven segment display, we have colored the necessary pixels in order to represent the numbers. By counting upwards we have obtained the playing time of the game.

change, in the contrast when ball reflects from the upper part

screen.v*

```
108
109         //ilk altıgen
110         engel_1<= ((CounterX>200)&&(CounterX<=(200+26))&&((CounterY>100)&&(CounterY<=(100+26))));
111         engel_1_1<= ((CounterX>200)&&(CounterX<=(200+13))&&((CounterY>(87))&&(CounterY<=(100)))&&(CounterY>-CounterX+300));
112         engel_1_2<= (((CounterX<=226)&&(CounterX>(200+13))&&(CounterY>(87))&&(CounterY<=(100))&&(CounterY>CounterX-126)));
113         engel_1_3<= ((CounterX>200)&&(CounterX<=(200+13))&&((CounterY>(126))&&(CounterY<=(100+26+13)))&&(CounterY<CounterX-74));
114         engel_1_4<= ((CounterX<=226)&&(CounterX>(200+13))&&((CounterY>(126))&&(CounterY<=(100+26+13)))&&(CounterY<-CounterX+352));
115
116         // ikinci altıgen
117         engel_2<= ((CounterX-300>200)&&(CounterX-300<=(200+26))&&((CounterY>100)&&(CounterY<=(100+26))));
118         engel_2_1<= ((CounterX-300>200)&&(CounterX-300<=(200+13))&&((CounterY>(87))&&(CounterY<=(100)))&&(CounterY>-CounterX+300+300));
119         engel_2_2<= (((CounterX-300<=226)&&(CounterX-300>(200+13))&&(CounterY>(87))&&(CounterY<=(100))&&(CounterY>CounterX-126-300)));
120         engel_2_3<= ((CounterX-300>200)&&(CounterX-300<=(200+13))&&((CounterY>(126))&&(CounterY<=(100+26+13)))&&(CounterY<CounterX-74-300));
121         engel_2_4<= ((CounterX-300<=226)&&(CounterX-300>(200+13))&&((CounterY>(126))&&(CounterY<=(100+26+13)))&&(CounterY<-CounterX+352+300));
122
123         //Yuvarlaklar
124         engel_3<= ((((CounterX-283)*(CounterX-283))+((CounterY-183)*(CounterY-183)))<=20*20);
125         engel_4<= ((((CounterX-450)*(CounterX-450))+((CounterY-250)*(CounterY-250)))<=20*20);
126         engel_5<= ((((CounterX-550)*(CounterX-550))+((CounterY-328)*(CounterY-328)))<=20*20);
127         engel_6<= ((((CounterX-380)*(CounterX-380))+((CounterY-350)*(CounterY-350)))<=20*20);
128         border <=((CounterX>=143)&&(CounterX<=(16+143))&&(CounterY<=450))
129                 |||((CounterX>(783-16))&&(CounterX<783))|||((CounterY>=35)&&(CounterY<(35+9)));
130         border2 <=(((CounterX>=674-30)&&(CounterX<=670-14))&&((CounterY<=450)&&(CounterY>=35)));
131         // timing border
132         bosluk_1<= ((CounterX>=670&&CounterX<=676)&&(CounterY>=130&&CounterY<=210))||
133                    ((CounterX>=744&&CounterX<=750)&&(CounterY>=130&&CounterY<=210))||
134                    ((CounterX>=670&&CounterX<=750)&&(CounterY>=130&&CounterY<=136))||
135                    ((CounterX>=670&&CounterX<=750)&&(CounterY>=210-6&&CounterY<=210));
136         //scoreboard
137         bosluk_2<= ((CounterX>=670&&CounterX<=676)&&(CounterY>=130+150&&CounterY<=210+150))||
138                    ((CounterX>=744&&CounterX<=750)&&(CounterY>=130+150&&CounterY<=210+150))||
139                    ((CounterX>=670&&CounterX<=750)&&(CounterY>=130+150&&CounterY<=136+150))||
140                    ((CounterX>=670&&CounterX<=750)&&(CounterY>=210-6+150&&CounterY<=210+150));
141         //flippers
142         sol_kol_ust<= ((((CounterY-400)*220)>=((CounterX-(144+16))*(90)))
143                       &&(((CounterY-405)*220)<=((CounterX-(144+16))*(90))))
```

Figure 7: Written codes for solids on Verilog

There exist both penalty and goals in our game. When ball collides with a green circle is results with a -15 points and when ball collides with red circle the player gets +10 points, and when ball collides with a hexagonal shaped obstacle it results with a +20 points. Therefore in order to represent this goals and penalties in our scoreboard we have thought to create three cases. For each case we will determine the places of the obstacles and by conditioning the movement of the ball the number on the scoreboard will be changed. For example suppose that case 1 is the red circle i.e. ball will reflect from the red circle, and then by coloring the necessary bits we will increase the number on the scoreboard.

B. Ball Movements and Collisions

For the game, the only solid that will continuously move is the ball. This ball will move the entire time move after start switch on. In order to move it, the centers of it, are stored in the X_top_merkez and Y_top_merkez. We divide clock 50 MHz clock with by 500k and we get 100Hz signal. There are some directories depends of the movements. The directories of movements are adjusted with respect to either collisions or starting point from plunger. We defined two integers namely X_v and Y_v in order to represent the velocity of the ball. Depending on the velocity of the ball reflects from the objects and borders.

When ball reflects from the borders one can say that, for left and right side of the border X_v is inverted and Y_v does not

of the border X_v does not changes however Y_v is inverted.

When ball reflects from the arms of the flippers the reflection angle has to become 90^{o}. Therefore both X_v and Y_v is converted according to the coming direction.

When ball reflects from the circles it reflects with respect to octagonal approximation which is found in literature research (Cheney, 2019).

The reflection from hexagon is a bit more complicated since the reflection angle will change depending on which side that ball will collide. For the rectangular sides the idea is same with reflection from borders but for triangular sides one has to think angles. Therefore we have said that ball will reflect randomly.

C. Flipper Movement

Flippers are done by generating two line equations and limiting the CounterX values. By limiting the CounterY values between these two lines we have colored the pixels between two lines. One can see code for flippers as a sol_kol_alt and sag_kol_alt in our code [Fig 8].

```
148         sol_kol_alt<= ((((CounterY-466)*220)>=((CounterX-(342))*(90-doga_sol)))
149                       &&(((CounterY-471)*220)<=((CounterX-(342))*(90-doga_sol)))
150                       &&(CounterX>=160+220-60)&&(CounterX<=160+220));
151         sag_kol_alt<= ((((CounterY-466)*220)>=((CounterX-462)*(-90+doga_sag)))
152                       &&(((CounterY-471)*220)<=((CounterX-462)*(-90+doga_sag)))
153                       &&(CounterX<644-220+60)&&(CounterX>=644-220));
154
```

Figure 8: Written codes for solids on Verilog

The flipper's can be move by chancing the slopes of the lines. We have defined a button for each flipper and conditioned that if a button is pushed the slope for right arm increases and the slope for left arm decreases. The flippers should be seen continuous therefore for every 500000 clock cycle (100 Hz) the slope changes in magnitude 1.

V. Conclusion

In this project the pinball game is created by using FPGA, Quartus and Verilog. While doing this project we have make a lot of literature research and learned a lot about VGA such as the missions of internal cables. We have increased our knowledge about Verilog and implemented to a screen using VGA. We make literature research about how does VGA works and able to set it work.

After the VGA works we generated the obstacles and flippers. Since the game is played with respect to movements of the ball, we create a moving blue ball. After that, we make collisions with borders and obstacles. Moreover we have managed to move the flippers, in order to save the ball from falling out of game. Also we will try to create the scoreboard by SSD logic and some algorithm that will count the gained or lost points. Furthermore, we add some time counter that will follow the timing of the game.

To conclusion, we can say that we learned how to write codes on verilog, how to drive VGA module and control their synchronizations, and how to draw various solid materials. In addition, we learned how to move obstacles and how to control their reaction when the ball collides these solid materials or flippers. Moreover we have learned to use Modelsim to examine and observe results of our code with respect to clock cycles.

Thanks to this project, since we have improved our projection about the continuous images and their existence via VGA. In addition to VGA knowledge, also we improved our skills about the coding of the FPGA.

Although our project does not fully satisfy the project requirements we are able to accomplish the large part of the project.

NOTE: You can see our full code and current screen of our project in above Appendix 1 and Appendix 2 respectively.

VI. REFERENCE

Cheney, E. W. (2019). Best Simple Octagonal Distances in Digital Geometry . *Journal of Approximation Theory*, 155-174.

unknown. (2019). *EE314 DIGITAL ELECTRONICS LABORATORY DECLERATİONS.* Retrieved from Metuclass: https://odtuclass.metu.edu.tr/pluginfile.php/273378/mod_resource/content/2/ee314%20project%202018-2019.pdf

unknown. (2019). *VGA Signal 640 x 480 @ 60 Hz Industry standard timing.* Retrieved from TinyVGAprojects: http://tinyvga.com/vga-timing/640x480@60Hz

APPENDIX 1:

```
module screen(
    input clk,
   output vga_h_sync,
   output vga_v_sync,
   output reg inDisplayArea,
   output reg [9:0] CounterX,
    output reg [9:0] CounterY,
    output reg [7:0] Red,
    output reg [7:0] Blue,
    output reg [7:0] Green,
    output reg clk_en,
    output reg [9:0] X_top_merkez,
    output reg [9:0] Y_top_merkez,
    output reg [30:0] counterball,
    output reg [30:0] counterssd,
    //output reg [5:0] X_v,
    //output wire Y_v,
    input startbutton, start_sagkol, start_solkol,
    output reg [5:0] doga_sag,
    output reg [5:0] doga_sol

 );

    //horizontal and vertical synch generator
```

```
    reg vga_HorizontalSynch,
vga_VerticalSynch,border,border2,
        engel_1,engel_1_1,engel_1_2,engel_1_3,engel_1_4,
        engel_2,engel_2_1,engel_2_2,engel_2_3,engel_2_4,
        engel_3,engel_4,engel_5,engel_6,
        sag_kol_alt,ssd11,ssd12,ssd13,ssd14,ssd15,ssd16,ssd17,

sag_kol_ust,ssd21,ssd22,ssd23,ssd24,ssd25,ssd26,ssd27,
        sol_kol_alt,ssd31,ssd32,ssd33,ssd34,ssd35,ssd36,ssd37,
        sol_kol_ust,
        bosluk_1,
        bosluk_2,
        top,
        plunger,
        clock_sayaci

        ;

    integer X_v,Y_v;
   wire CounterXmaxed = (CounterX == 799); // 16 + 48 + 96
+ 640
   wire CounterYmaxed = (CounterY == 524); // 10 + 2 + 33
+ 480


    initial
    begin
     X_top_merkez<=180;
     Y_top_merkez<=242;
     counterball <= 0;
     X_v = 5;
     Y_v = -1;
     doga_sag <= 1;
     doga_sol <=1;
     end

    //Clock divider
    always @(posedge clk) begin
      clk_en <= !clk_en;
    end

    //For horizonatal and vertical synchs, counter incrementer
    always @(posedge clk_en)
        begin
      if (CounterXmaxed)
         CounterX <= 0;
       else
         CounterX <= CounterX + 1;
    end

   always @(posedge clk_en)
   begin
         if ((CounterY==1)&&(CounterX==799))
            begin
               vga_VerticalSynch <= (CounterY < (524))&&(
CounterY ==(1));
               CounterY <= CounterY + 1;
            end
         else if (CounterXmaxed)
            begin
               if (CounterYmaxed)
               CounterY <=0;
               else
               CounterY <= CounterY + 1;

            end
            else
               begin
                  vga_VerticalSynch <= ((CounterY < (524))&&(
CounterY >(1)));
               end
               vga_HorizontalSynch <= ((CounterX > (94))&&
CounterX < (799));   // active for 96 clocks
   end
   always @(posedge clk)
   begin
        counterball <= counterball + 1'b1;
        counterssd <= counterssd + 1'b1;
      inDisplayArea <= (CounterX < (784) && CounterX >
(143) ) && (CounterY < (515)&& CounterY > (34));

        //ilk altıgen
        engel_1<=
((CounterX>200)&&(CounterX<=(200+26))&&((CounterY>
100)&&(CounterY<=(100+26))));
        engel_1_1<=
((CounterX>200)&&(CounterX<=(200+13))&&((CounterY>(
87))&&(CounterY<=(100)))&&(CounterY>-CounterX+300));
        engel_1_2<=
(((CounterX<=226)&&(CounterX>(200+13))&&(CounterY>(
87))&&(CounterY<=(100))&&(CounterY>CounterX-126)));
        engel_1_3<=
((CounterX>200)&&(CounterX<=(200+13))&&((CounterY>(
126))&&(CounterY<=(100+26+13)))&&(CounterY<Counter
X-74));
        engel_1_4<=
((CounterX<=226)&&(CounterX>(200+13))&&((CounterY>(
126))&&(CounterY<=(100+26+13)))&&(CounterY<-
CounterX+352));

       // ikinci altıgen
        engel_2<= ((CounterX-300>200)&&(CounterX-
300<=(200+26))&&((CounterY>100)&&(CounterY<=(100+2
6))));
        engel_2_1<= ((CounterX-300>200)&&(CounterX-
300<=(200+13))&&((CounterY>(87))&&(CounterY<=(100))
)&&(CounterY>-CounterX+300+300));
        engel_2_2<= (((CounterX-300<=226)&&(CounterX-
300>(200+13))&&(CounterY>(87))&&(CounterY<=(100))&
&(CounterY>CounterX-126-300)));
        engel_2_3<= ((CounterX-300>200)&&(CounterX-
300<=(200+13))&&((CounterY>(126))&&(CounterY<=(100
+26+13)))&&(CounterY<CounterX-74-300));
        engel_2_4<= ((CounterX-300<=226)&&(CounterX-
300>(200+13))&&((CounterY>(126))&&(CounterY<=(100+
26+13)))&&(CounterY<-CounterX+352+300));
```

```
        //Yuvarlaklar
         engel_3<= ((((CounterX-283)*(CounterX-
283))+((CounterY-183)*(CounterY-183)))<=20*20);
         engel_4<= ((((CounterX-450)*(CounterX-
450))+((CounterY-250)*(CounterY-250)))<=20*20);
         engel_5<= ((((CounterX-550)*(CounterX-
550))+((CounterY-328)*(CounterY-328)))<=20*20);
         engel_6<= ((((CounterX-380)*(CounterX-
380))+((CounterY-350)*(CounterY-350)))<=20*20);
        border
<=((CounterX>=143)&&(CounterX<=(16+143))&&(Counter
Y<=450))
                ||((CounterX>(783-
16))&&(CounterX<783))||((CounterY>=35)&&(CounterY<(3
5+9)));
         border2 <=(((CounterX>=674-30)&&(CounterX<=670-
14))&&((CounterY<=450)&&(CounterY>=35)));

         bosluk_1<=
((CounterX>=670&&CounterX<=676)&&(CounterY>=130&
&CounterY<=210))||

((CounterX>=744&&CounterX<=750)&&(CounterY>=130&
&CounterY<=210))||

((CounterX>=670&&CounterX<=750)&&(CounterY>=130&
&CounterY<=136))||

((CounterX>=670&&CounterX<=750)&&(CounterY>=210-
6&&CounterY<=210));

         ssd11<=
((CounterX>=721&&CounterX<725)&&(CounterY>=137&&
CounterY<=137+30));
         ssd12<=
((CounterX>=725&&CounterX<=737)&&(CounterY>=137&
&CounterY<=137+13));
         ssd13<=
((CounterX>738&&CounterX<=742)&&(CounterY>=137&&
CounterY<=137+30));
         ssd14<=
((CounterX>=725&&CounterX<=737)&&(CounterY>=208&
&CounterY<=221));
         ssd15<=
((CounterX>=721&&CounterX<725)&&(CounterY>=208&&
CounterY<=208+30));
         ssd16<=
((CounterX>738&&CounterX<=742)&&(CounterY>=208+30
&&CounterY<=221+30));
         ssd17<=
((CounterX>=725&&CounterX<=690)&&(CounterY>=239&
&CounterY<=239+13));

        /* sdd11<=ssd2||ssd3||ssd4||ssd5||ssd7
         sdd12<=
         sdd13<=
         sdd14<=
         sdd15<=
         sdd16<=
         sdd17<=*/

         bosluk_2<=
((CounterX>=670&&CounterX<=676)&&(CounterY>=130+
150&&CounterY<=210+150))||

((CounterX>=744&&CounterX<=750)&&(CounterY>=130+
150&&CounterY<=210+150))||

((CounterX>=670&&CounterX<=750)&&(CounterY>=130+
150&&CounterY<=136+150))||

((CounterX>=670&&CounterX<=750)&&(CounterY>=210-
6+150&&CounterY<=210+150));

         sol_kol_ust<= ((((CounterY-400)*220)>=((CounterX-
(144+16))*(90)))
                  &&(((CounterY-405)*220)<=((CounterX-
(144+16))*(90)))
                  &&(CounterX<=160+220-60));
         sag_kol_ust<= ((((CounterY-400)*220)>=((CounterX-
644)*(-90)))
                  &&(((CounterY-405)*220)<=((CounterX-
644)*(-90)))
                  &&(CounterX<644)&&(CounterX>=644-
220+60));
         sol_kol_alt<= ((((CounterY-466)*220)>=((CounterX-
(342))*(90-doga_sol)))
                  &&(((CounterY-471)*220)<=((CounterX-
(342))*(90-doga_sol)))
                  &&(CounterX>=160+220-
60)&&(CounterX<=160+220));
         sag_kol_alt<= ((((CounterY-466)*220)>=((CounterX-
462)*(-90+doga_sag)))
                  &&(((CounterY-471)*220)<=((CounterX-
462)*(-90+doga_sag)))
                  &&(CounterX<644-
220+60)&&(CounterX>=644-220));



         plunger<=
(((CounterX<=(143+30))&&(CounterX>143+16))&&(Counte
rY>=237-2&&CounterY<247+2));
         top<= ((((CounterX-X_top_merkez)*(CounterX-
X_top_merkez))+((CounterY-Y_top_merkez)*(CounterY-
Y_top_merkez)))<=7*7);

          // tam orta noktada x=402, y=499
           // sol ve sağdaki fark x 60 olsun, y farkı  25
           // sağ alt için başlangıç noktası, x=462, y=474
           // sol alt için başlangıç noktası, x=342, y=474

           // Topun Hareketi Burada
                    if (counterball==500000) begin  //500000
                             counterball <= 0;
                                if (startbutton == 1)
                                   begin
                                        X_top_merkez <=
X_top_merkez + X_v;
```

```
                                    Y_top_merkez <=
Y_top_merkez + Y_v;
                                        if (start_sagkol==0)
                                        doga_sag<=doga_sag +
1'b1;
                                     else if (start_solkol==0)
                                        doga_sol<=doga_sol +
1'b1;
                                end
                               end
        //else if (counterssd==49000000) begin
                //counterssd<=0;
           /*   if (counterball==50000000) begin
                else if(ssd3) //1
                begin
                Green <=8'b11111111;
                Red <=8'b11111111;
                Blue <=8'b00000000;
                end
                else if(ssd2||ssd3||ssd4||ssd5||ssd7) //2
                begin
                Green <=8'b11111111;
                Red <=8'b11111111;
                Blue <=8'b00000000;
                end
                else if(ssd2||ssd3||ssd4||ssd6||ssd7) //3
                begin
                Green <=8'b11111111;
                Red <=8'b11111111;
                Blue <=8'b00000000;
                end
                else if(ssd1||ssd5||ssd7||ssd6) //4
                begin
                Green <=8'b11111111;
                Red <=8'b11111111;
                Blue <=8'b00000000;
                end
                if(ssd2||ssd1||ssd4||ssd6||ssd7) //5
                begin
                Green <=8'b11111111;
                Red <=8'b11111111;
                Blue <=8'b00000000;
                end
                if(ssd1||ssd2||ssd4||ssd5||ssd7||ssd6) //6
                begin
                Green <=8'b11111111;
                Red <=8'b11111111;
                Blue <=8'b00000000;
                end
                if(ssd2&&ssd3&&ssd4&&ssd6) //7
                begin
                Green <=8'b11111111;
                Red <=8'b11111111;
                Blue <=8'b00000000;
                end

   if(ssd1&&ssd2&&ssd3&&ssd4&&ssd5&&ssd6&&ssd7)
//8
                begin
                Green <=8'b11111111;
                Red <=8'b11111111;
                Blue <=8'b00000000;
                end
                if(ssd2ssd3&&ssd4&&ssd6&&ssd1)
                begin
                Green <=8'b11111111;
                Red <=8'b11111111;
                Blue <=8'b00000000;
                end
               end*/
          else  if (inDisplayArea)
              begin
                if (border)
                  begin
                    Green <=8'b11111111;
                    Red <=8'b11111111;
                    Blue <=8'b00000000;

                  end
                  // ilk üçgen
                else if (border2)
                  begin
                    Green <=8'b11111111;
                    Red <=8'b11111111;
                    Blue <=8'b00000000;

                  end

                else if(top) // Topun yeri

                    begin
                            Green <=8'b11111111;
                            Red <=8'b00000000;
                            Blue <=8'b11111111;

   if((startbutton==1)&&(X_top_merkez<=160+8))
                            begin
                            X_v = 5;
                           end
                        else if((X_top_merkez>=644-
8)&&(startbutton==1))
                            begin
                            X_v = -5;
                            end
                        else if
((Y_top_merkez<=52)&&(startbutton==1))
                             begin
                             Y_v = 1;
                            end
                        else if((((X_top_merkez-
283)*(X_top_merkez-283))+((Y_top_merkez-
183)*(Y_top_merkez-183)))<=27*27)
                            begin
                            if(X_top_merkez-
110<=Y_top_merkez)
                            Y_v = -5;
                            else if (X_top_merkez-
110>Y_top_merkez)
                            X_v = -8;
                           end
```

```
                        else if ((((X_top_merkez-
550)*(X_top_merkez-550))+((Y_top_merkez-
328)*(Y_top_merkez-328)))<=27*27)
                            begin
                               Y_v = -4;
                               X_v = -5;
                            end

                        else if ((((X_top_merkez-
380)*(X_top_merkez-380))+((Y_top_merkez-
350)*(Y_top_merkez-350)))<=27*27)
                            begin
                               Y_v <= -6;
                               X_v<= -3;
                            end
                        else if ((((X_top_merkez-
450)*(X_top_merkez-450))+((Y_top_merkez-
250)*(Y_top_merkez-250)))<=27*27)
                            begin
                               Y_v <= -5;
                               X_v <= -5;
                            end

                        /*else if((((Y_top_merkez-
400)*220)>=((X_top_merkez-
(144+16))*(90)))&&(startbutton==1))  // sol engel
                            begin
                               X_v <= 5;
                               Y_v <= -3;
                               end
                        else if((((Y_top_merkez-
400)*220)>=((X_top_merkez-644)*(-
90)))&&(startbutton==1)) // sağ engel
                            begin
                               X_v <= -X_v;
                               Y_v <= -Y_v;
                               end*/
                    end


            else if (engel_1)
                    begin
                        Red <=8'b00000000;
                        Blue<= 8'b11111111;
                        Green<= 8'b00000000;
                    end

             else if(engel_1_1) //üst üçgen
                    begin

                        Red <=8'b00000000;
                        Blue<= 8'b11111111;
                        Green<= 8'b00000000;
                      end

            else if(engel_1_2) //üst üçgen
                   begin

                        Red <=8'b00000000;
                        Blue<= 8'b11111111;
                        Green<= 8'b00000000;
                    end

            else if(engel_1_3) //üst üçgen
                   begin

                        Red <=8'b00000000;
                        Blue<= 8'b11111111;
                        Green<= 8'b00000000;
                    end

            else if(engel_1_4) //üst üçgen
                   begin

                        Red <=8'b00000000;
                        Blue<= 8'b11111111;
                        Green<= 8'b00000000;
                    end

            //ikinci üçgen
            else if (engel_2)
                   begin
                        Red <=8'b00000000;
                        Blue<= 8'b11111111;
                        Green<= 8'b00000000;
                   end

             else if(engel_2_1)
                   begin

                        Red <=8'b00000000;
                        Blue<= 8'b11111111;
                        Green<= 8'b00000000;
                      end

            else if(engel_2_2)
                   begin
                        Red <=8'b00000000;
                        Blue<= 8'b11111111;
                        Green<= 8'b00000000;
                    end

           else if(engel_2_3) //üst üçgen
                   begin
                        Red <=8'b00000000;
                        Blue<= 8'b11111111;
                        Green<= 8'b00000000;
                    end

            else if(engel_2_4)
                   begin
                        Red <=8'b00000000;
                        Blue<= 8'b11111111;
                        Green<= 8'b00000000;
                    end

            else if(engel_3) // daire 1 M=> y=183-x=283
                   begin
                        Red <=8'b00000000;
                        Blue<= 8'b00000000;
```

```
                Green<= 8'b11111111;
            end

        else if(engel_4) // daire 1 M=> y=80-x=310
            begin
                Red <=8'b00000000;
                Blue<= 8'b00000000;
                Green<= 8'b11111111;
            end

        else if(engel_5) // daire 1 M=> y=80-x=310
            begin
                Red <=8'b11111111;
                Blue<= 8'b00000000;
                Green<= 8'b00000000;
            end

        else if(engel_6) // daire 1 M=> y=190-x=450
            begin
                Red <=8'b11111111;
                Blue<= 8'b00000000;
                Green<= 8'b00000000;
            end

        else if(sag_kol_alt) // sağ altta vuracak olan kol

                begin
                    Green <=8'b11111111;
                    Red <=8'b11111111;
                    Blue <=8'b00000000;
                end
        else if(sag_kol_ust) // sağ altta vuracak olan kol

                begin
                    Green <=8'b11111111;
                    Red <=8'b11111111;
                    Blue <=8'b00000000;
                end
        else if(sol_kol_alt) // sol altta vuracak olan kol

                begin
                    Green <=8'b11111111;
                    Red <=8'b11111111;
                    Blue <=8'b00000000;
                end
        else if(sol_kol_ust) // sol altta vuracak olan kol

                begin
                    Green <=8'b11111111;
                    Red <=8'b11111111;
                    Blue <=8'b00000000;
                end
        else if(bosluk_1) // sağ üst, skor tutacağımız
boşluk
                begin
                    Green <=8'b00000000;
                    Red <=8'b00000000;
                    Blue <=8'b11111111;
                end
            else if(bosluk_2) // sağ orta, zaman tutacağımız
boşluk
                begin
                    Green <=8'b00000000;
                    Red <=8'b00000000;
                    Blue <=8'b11111111;
                end
            else if(plunger) // Plunger'ın yeri
                begin
                    Green <=8'b11111111;
                    Red <=8'b00000000;
                    Blue <=8'b11111111;
                end

            //else if(CounterYmaxed)
               //  if(top birşeye çarpyorsa)
                  //else if (counter)
            else
                begin
                   Red<= 8'b00000000;
                   Blue<= 8'b00000000;
                   Green<= 8'b00000000;
                end
            end

        end


   assign vga_h_sync = ~vga_HorizontalSynch;
//(CounterX>95) ? 1'b1:1'b0;
   assign vga_v_sync = ~vga_VerticalSynch;   //(CounterY>1)
? 1'b1:1'b0;


endmodule

            // sag_kol<= ((((CounterY-451)*220)>=((CounterX-
650)*(-60+doga)))&&(((CounterY-455)*220)<=((CounterX-
650)*(-
60+doga)))&&(CounterX<=649)&&(CounterX>=549));
            // sol_kol<= ((((CounterY-451)*220)>=((CounterX-
144)*(60)))&&(((CounterY-455)*220)<=((CounterX-
144)*(60)))&&(CounterX>144+17));


       /*(((CounterY-443)(220))>=((CounterX-624)*(-60))) //
L 1 line
                &&(((CounterY-453)(220))<=((CounterX-
630)(-60))) // L 4 line
                &&((CounterY-463)60)<=((220)(CounterX-
614))  // L 2 line
                &&((CounterY-453)60)>=((220)(CounterX-
630))); // L 3
        //top<=((((CounterX-380)(CounterX-
380))+((CounterY-350)(CounterY-350)))<=14*14);*/
```

APPENDIX 2: