

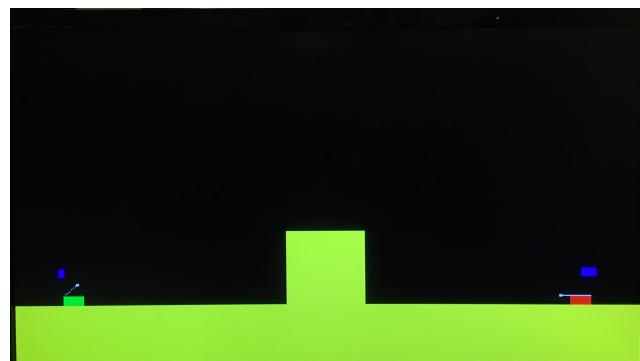


ELEX 7660: Digital System Design

Design Project

VGA Based Arcade Game (ZART)

Terry Calderbank
Kolyn Gray



12 April, 2018

Contents

1	Introduction	2
1.1	Motivation	2
2	Project Overview	2
3	Display Hardware	4
4	VGA Software	5
4.1	Signal Timing	5
4.2	Drawing with VGA	7
4.3	Example Graphics	7
5	Suggestions for Future Work	8
5.1	Randomly Generated Wind	8
5.2	Moving Tanks	9
5.3	Destroyable Terrain	9
5.4	Higher Resolution Colour	9
6	Program Listings	10
6.1	zart.sv	10
6.2	Game_Control.sv	12
6.3	BarrelControl.sv	14
6.4	BulletMove.sv	16
6.5	sqrt.sv	19
6.6	colseq.sv	20
6.7	kpdecode.sv	21
6.8	vga.sv	24
6.9	hvsync_generator.sv	27

1 Introduction

The goal of this project was to implement our new-found knowledge of finite state machine design in order to make a simple arcade game displayed over the VGA protocol. In the project we recreated an old and incredibly obscure arcade game called ZART (Zee-Artillery). As an MS-DOS based game for Windows and PalmOS, ZART was considered retro when it was released by a small shareware company called PLBM games in 1999[1]. While extremely dated looking and soon to be faced with all kinds of compatibility issues with the new versions of windows, ZART was unique in that it only required each player to use a single keyboard key to play. The addictive game play, based around iteratively honing in an artillery round on an opponent's tank, also featured collapsible sand terrain as well as randomly generated wind effects.

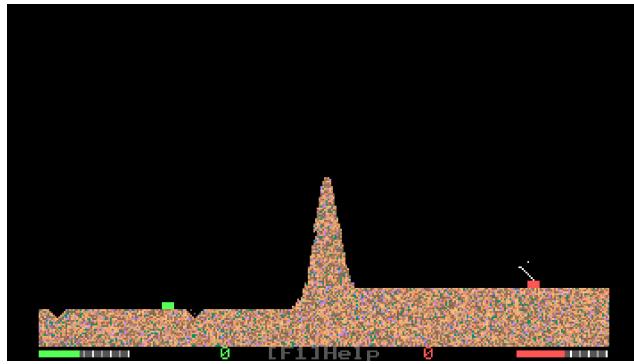


Figure 1: Gameplay of Zart by PLBM Games (1999)

1.1 Motivation

We chose this project because it allowed us to apply everything we have learned about state machines in a complex, but not overwhelming manner. We also feel that implementing somewhat realistic two-dimensional kinematic projectile motion in System Verilog was an interesting challenge. This project required us to learn a lot about VGA display signals as well as the additional hardware required to display additional colour ranges. While VGA is becoming obsolete, we felt that it was a good stepping stone towards understanding more advanced protocols like HDMI and DVI.

2 Project Overview

As seen in Figure 2, the arcade game system is made up of various blocks that all come together to form a system. The system inputs are the push buttons used by the player, and the output is 5 signals going out to the VGA monitor.

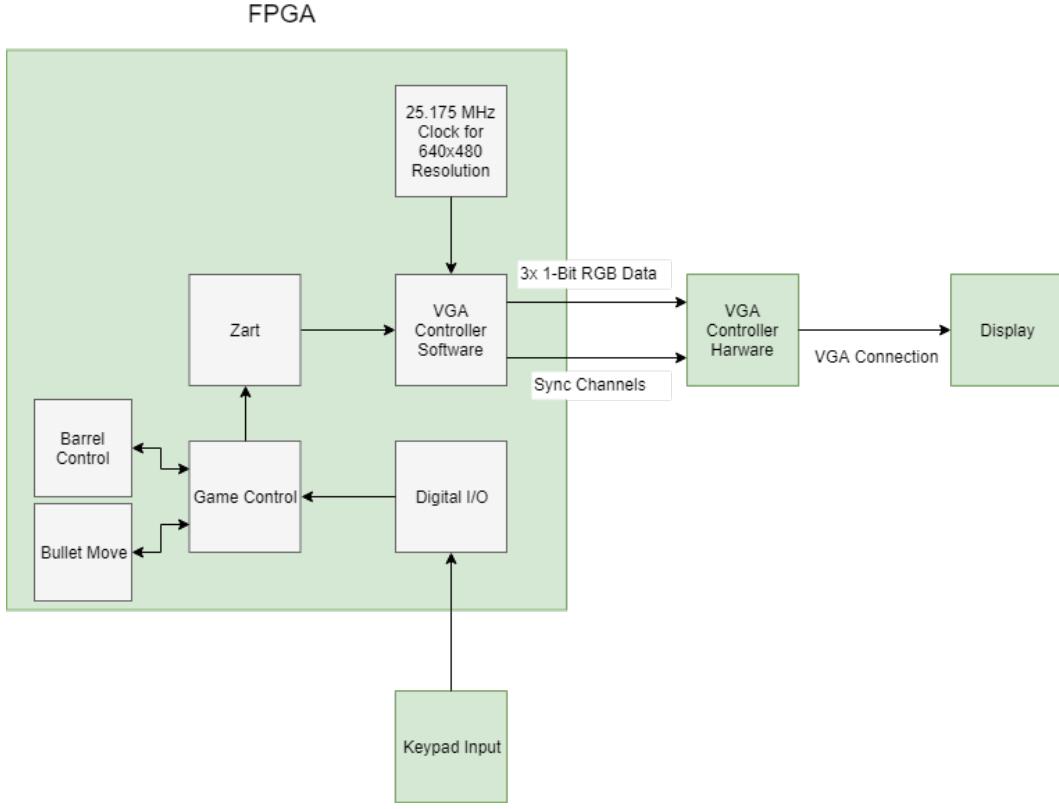


Figure 2: Project block diagram

The game used push buttons to change the state of each tank. One set was to move the player to the next stage of shooting and the other was to select the parameter. The flow of these operations is depicted in Figure 3.

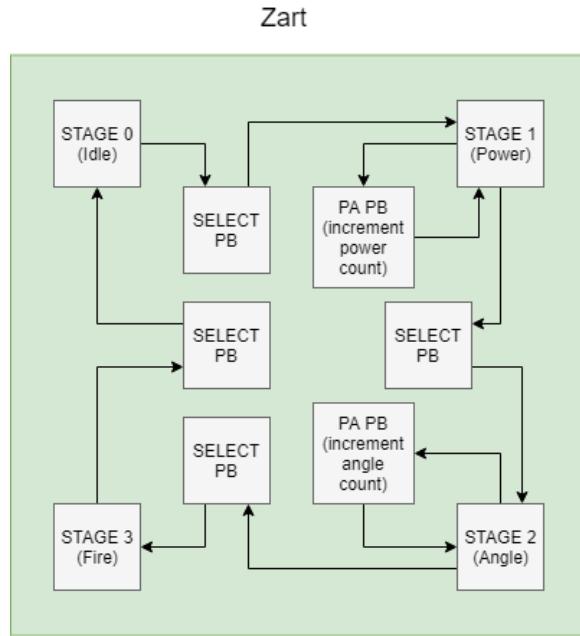


Figure 3: Zart Game State Diagram

We felt that the game would look more refined in the bullets moved pixel by pixel and not jump across the screen. So we made two counters counting from a 2kHz clock, each having max values that changed depending on how fast the bullet was supposed to move. The Y direction counter would update the max count when the ball moved. It would update to ever increasing values to simulate the bullet slowing down in the y direction while rising and speed up in the y direction as the bullet fell. Theoretically for the bullet to completely stop, the counter would need to count to infinity. To get around this we did trials to see at what counter values the ball appeared to have adequately stopped. The code would test for the Y counter max to exceed this value and then the direction would change. This process is visualized in Figure 4.

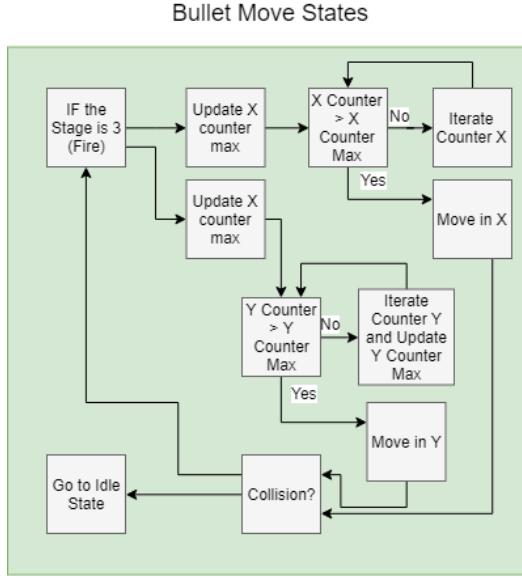


Figure 4: Bullet Moving State Diagram

3 Display Hardware

The hardware required for this project was somewhat minimalist. For basic implementation of 8 colour display, all that was required was a VGA female connector, and $3 \times 270\Omega$ resistors[2]. The purpose of the resistors was to create a voltage divider with the 75Ω monitor termination in order to deliver 0-0.7V from our 3.3V outputs on the DE0 Nano FPGA.

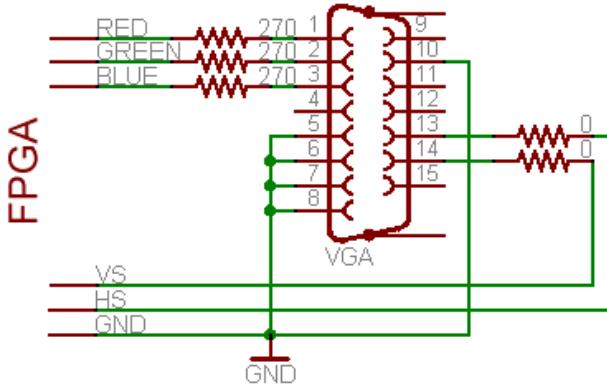


Figure 5: VGA Connector diagram showing minimal connections required

4 VGA Software

We would like to start this section by acknowledging Jean Nicolle's work at www.fpga4fun.com. Without his explanation of the VGA protocol, this project would have been far more difficult.

4.1 Signal Timing

In order to display content onto a VGA monitor proper timing information must be sent over the five signal wires as seen in Figure 7. Choosing this timing depends on your desired display resolution. In an effort to keep things simple we chose a resolution of 640x480. With this information, you can determine the signal information required by looking at a parameter chart like Figure 6.

Format	Pixel Clock (MHz)	Horizontal (in Pixels)				Vertical (in Lines)			
		Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
640x480, 60Hz	25.175	640	16	96	48	480	11	2	31
640x480, 72Hz	31.500	640	24	40	128	480	9	3	28
640x480, 75Hz	31.500	640	16	96	48	480	11	2	32
640x480, 85Hz	36.000	640	32	48	112	480	1	3	25
800x600, 56Hz	38.100	800	32	128	128	600	1	4	14
800x600, 60Hz	40.000	800	40	128	88	600	1	4	23
800x600, 72Hz	50.000	800	56	120	64	600	37	6	23
800x600, 75Hz	49.500	800	16	80	160	600	1	2	21
800x600, 85Hz	56.250	800	32	64	152	600	1	3	27
1024x768, 60Hz	65.000	1024	24	136	160	768	3	6	29
1024x768, 70Hz	75.000	1024	24	136	144	768	3	6	29
1024x768, 75Hz	78.750	1024	16	96	176	768	1	3	28
1024x768, 85Hz	94.500	1024	48	96	208	768	1	3	36

Figure 6: This table lists timing values for several popular resolutions[2].

The selected resolution requires a pixel clock of approximately 25.175 MHz. According to our reference, this frequency does not have to be exact due to the prevalence of 'multisync' monitors that can adjust themselves to frequency differences. The front and back porch values did not have to be set manually as long as we held our sync pulses low long enough.

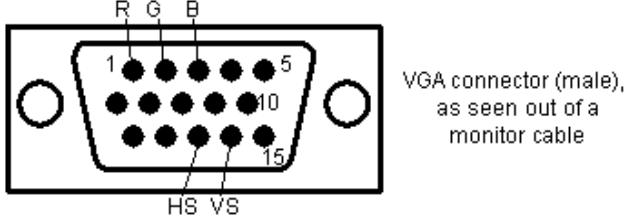


Figure 7: VGA Signal Channels[2]

Once the pixel clock has been selected, two slower pulses can be generated to control drawing on the screen. The pixel clock traces across the screen and when it reaches the end, an active low `h_sync` pulse triggers telling the device to go to the next line. This happens at about 32kHz. Once all the lines have been drawn, an active low `v_sync` pulse triggers telling the monitor to go back to the top and start again. The timing of this pulse is often referred to as refresh rate, and in our case it was 60Hz. See Figure 11.

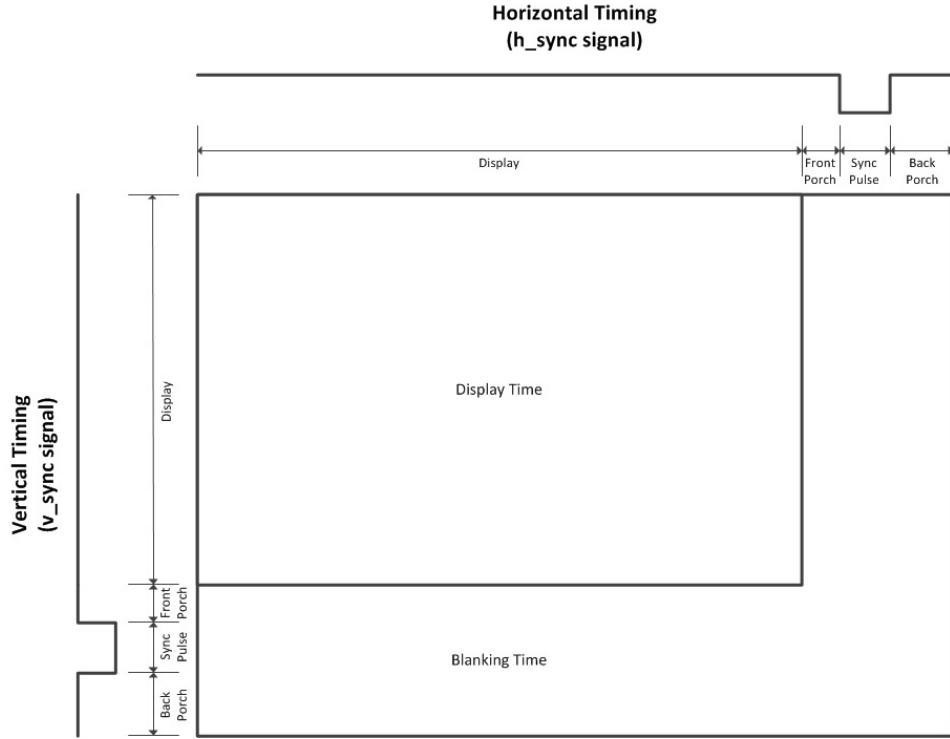


Figure 8: Signal timing for VGA Signal[3]

4.2 Drawing with VGA

Drawing to VGA was achieved using 3 colour channels. This gave us 3 bits to work with, resulting in 8 available colours. Rather than using up the resources required to implement a 640x480 3 colour-channel array, we opted to use conditional statements in combination with the X and Y counters to draw objects to each of the 3 channels. In this way, an object could be stored as a number range rather than a lengthy array of values. For example, to draw a simple red rectangle, the RGB output code would look something like the following:

```
R = (CounterX >= leftbound)&&(CounterX <= rightbound)&&(CounterY >= topbound)&&(CounterY <= bottombound);
```

```
G = 0;
```

```
B = 0;
```

Using the same idea an equation for a line could be used as a test statement for drawing lines. The only problem with this method would be lines with changing slopes. An example of defining conditions for changing slopes can be seen in the program listing in section 6.8.

4.3 Example Graphics

The following pictures are of various in game graphics generated using conditional statements. See the vga.sv program listing for the code required to draw a BLOCK, POINT, LINEPOS, and LINENEG.

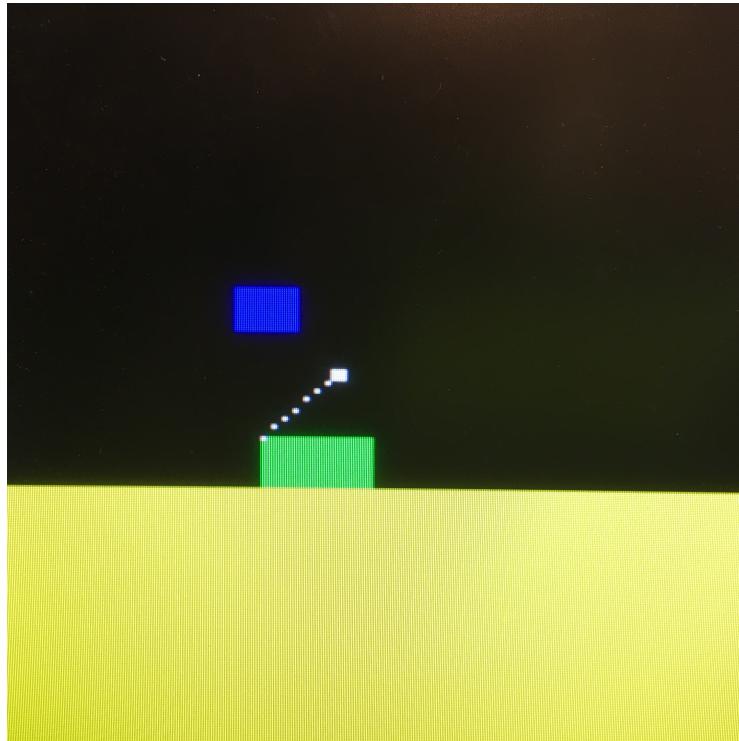


Figure 9: Tank Image Depicting using LINEPOS, and BLOCK close up

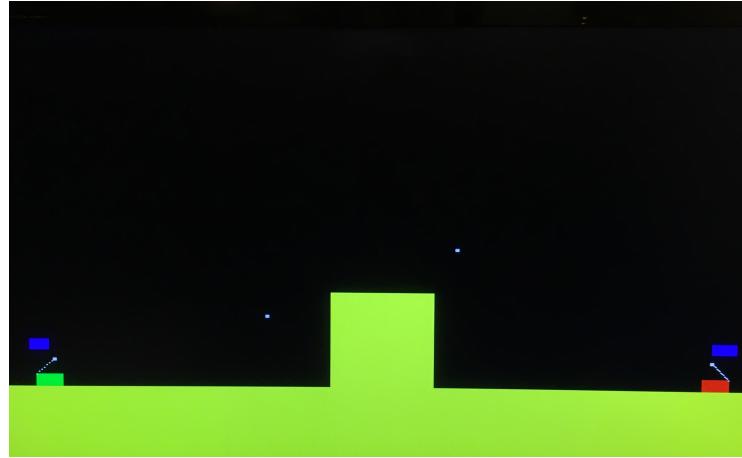


Figure 10: Image Depicting using LINEPOS, LINENEG, POINT, and BLOCK

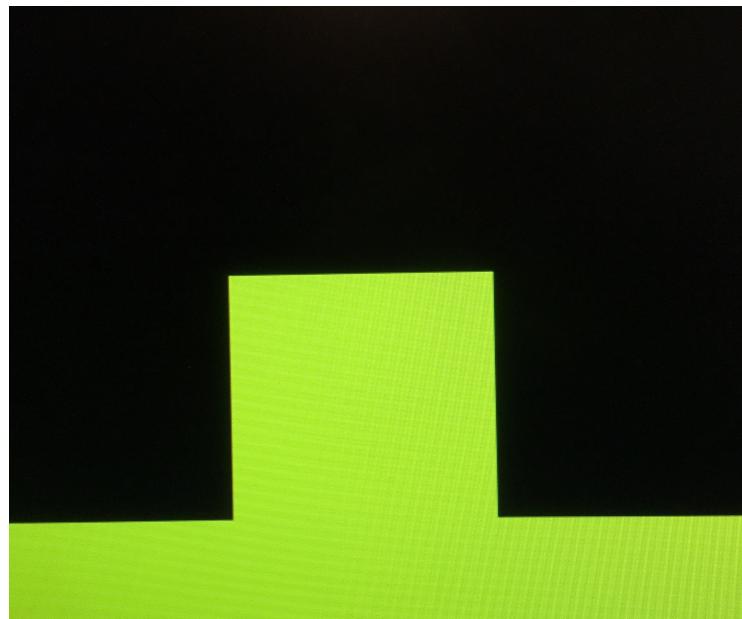


Figure 11: Image for the hill Depicting the usage of BLOCK

5 Suggestions for Future Work

Some additional items we would have liked to implement given more time were the following:

5.1 Randomly Generated Wind

This could be implemented by creating a variable that gets randomized in magnitude and left or right direction at the beginning of the round. The variable would then get added or subtracted from a player's bullet speed during the firing state. This would manifest itself as a constant accelerating force that would influence the trajectories of the projectiles.

5.2 Moving Tanks

This could easily be accomplished by adding more player state variables to include an overall tank X position. The position could be changed using additional buttons. This would increase competitiveness of gameplay.

5.3 Destroyable Terrain

This would involve some work implementing the FPGA's larger memory to store each and every pixel of the terrain in a large array. If an intersection between a bullet and the terrain was detected, those specific pixels could be removed from the array.

5.4 Higher Resolution Colour

The initial planning of this project involved using a 10 bit video DAC to implement higher resolution colour. Due to timing constraints, this was not achieved. An easier way of increasing colour resolution in the future would be to use a resistor network to create a 3 bit DAC. This would be easily done by adding an extra two outputs per colour channel and then summing the 3 together using a buffer. More information can be found online by searching 3 or 4 bit resistor DAC.

References

- [1] DOS Games Archive, “Zart - zee artillery game,” accessed 2018-04-12. [Online]. Available: https://archive.org/details/ZARTZeeArtilleryGame_1020
- [2] J. P. Nicolle, “Pong Game,” accessed 2018-04-12. [Online]. Available: <http://www.fpga4fun.com/PongGame.html>
- [3] S. Larson, “VGA Controller (VHDL),” accessed 2018-04-12. [Online]. Available: <https://eewiki.net/pages/viewpage.action?pageId=15925278>

6 Program Listings

6.1 zart.sv

```
1 module zart(input logic clk_50MHz,
2               output logic clk,
3               (* altera_attribute = "-name=WEAK_PULL_UP_RESISTOR=ON" *)
4               input logic [3:0] kpr,
5               output logic [3:0] kpc,
6               output logic [7:0] LED,
7               output logic vga_h_sync, vga_v_sync, vga_R, vga_G, vga_B
8 );
9
10 logic clk_4Hz;
11 logic kphit; // flag for keypad hit
12 logic [15:0] num; //keypad register
13
14 // shifted values ( (0,0) top left )
15 logic [11:0] pos_x_i_p1;
16 logic [11:0] pos_y_i_p1;
17 logic [11:0] pos_x_i_p2;
18 logic [11:0] pos_y_i_p2;
19 logic [11:0] pos_x_p1;
20 logic [11:0] pos_y_p1;
21 logic [11:0] pos_x_p2;
22 logic [11:0] pos_y_p2;
23
24 // unshifted values ( (0,0) bottom left)
25 logic [11:0] pos_y_i_p1_tmp;
26 logic [11:0] pos_y_i_p2_tmp;
27 logic [11:0] pos_y_p1_tmp;
28 logic [11:0] pos_y_p2_tmp;
29
30 logic [11:0] shift = 480; // the shift value
31
32 // shift the values by the shift value
33 always_comb begin
34     pos_y_i_p1 = shift - pos_y_i_p1_tmp;
35     pos_y_i_p2 = shift - pos_y_i_p2_tmp;
36     pos_y_p1 = shift - pos_y_p1_tmp;
37     pos_y_p2 = shift - pos_y_p2_tmp;
38 end
39
40
41
42 clockgen2 clockgen(clk_50MHz, clk_25MHz, clk_4kHz);
43 clockdiv clockdiv_0(clk_4kHz,clk_4Hz);
44 colseq colseq_0(kpr, kpc, clk_4kHz);
45 kpdecode kpdecode_0(kpr, kpc, kphit, num);
46
47 Game_Control GAME_0( num[3],
48                     num[13],
49                     num[0],
50                     num[6],
51                     pos_x_i_p1,
52                     pos_y_i_p1_tmp,
53                     pos_x_i_p2,
54                     pos_y_i_p2_tmp,
55                     pos_x_p1,
56                     pos_y_p1_tmp,
57                     pos_x_p2,
58                     pos_y_p2_tmp);
59
60 vga vga( pos_x_p1,
61           pos_y_p1,
62           pos_x_p2,
63           pos_y_p2,
64           pos_x_i_p1,
65           pos_y_i_p1,
66           pos_x_i_p2,
67           pos_y_i_p2,
68           clk_50MHz,
69           vga_h_sync,
70           vga_v_sync,
71           vga_R,
72           vga_G,
73           vga_B);
74
75 // turn on leds on fpga when a key on the keypad is hit.
```

```
76     assign LED = { num[13] ,
77                     num[6] ,
78                     num[3] ,
79                     num[0] ,
80                     4'b0000} ;
81
82     assign clk = clk_4Hz ;
83
endmodule
```

6.2 Game_Control.sv

```

1 module Game_Control( input logic pb_select_p1, // player 1 stage select button
2                     input logic pb_select_p2, // player 2 stage select button
3                     input logic pb_pa_p1, // player 1 power or angle select button
4                     input logic pb_pa_p2, // player 2 power or angle select button
5                     output logic [11:0] pos_x_i_p1, // player 1 barrel x postion
6                     output logic [11:0] pos_y_i_p1, // player 1 barrel y postion
7                     output logic [11:0] pos_x_i_p2, // player 2 barrel x postion
8                     output logic [11:0] pos_y_i_p2, // player 2 barrel y postion
9                     output logic [11:0] pos_x_p1, // player 1 ball x postion
10                    output logic [11:0] pos_y_p1, // player 1 ball y postion
11                    output logic [11:0] pos_x_p2, // player 2 ball x postion
12                    output logic [11:0] pos_y_p2); // player 2 ball y postion
13
14 // temporary values for all the modules
15 logic [3:0] stage_p1 = 0;
16 logic [3:0] stage_p2 = 0;
17 logic [3:0] stage_p1_n = 0;
18 logic [3:0] stage_p2_n = 0;
19
20 logic [11:0] angle_x_p1;
21 logic [11:0] angle_y_p1;
22 logic [11:0] angle_x_p2;
23 logic [11:0] angle_y_p2;
24
25 logic [11:0] angle_div;
26 logic [11:0] angle_div2;
27
28 logic [3:0] power_p1;
29 logic [3:0] power_p2;
30
31 // flags to detect any out of bounds balls
32 logic collision_p1;
33 logic collision_p2;
34
35 // flags to detect wins or self deaths
36 logic [1:0] winner_p1;
37 logic [1:0] winner_p2;
38
39 // barrel and bullet control modules for player 1
40 // 1 = because this tank shoots right
41 // (50,93) = position of the tank for player 1
42 BarrelControl #(1,50,93) BC_p1( pb_pa_p1,
43                         pb_select_p1,
44                         stage_p1,
45                         pos_x_i_p1,
46                         pos_y_i_p1,
47                         angle_x_p1,
48                         angle_y_p1,
49                         angle_div,
50                         power_p1);
51 // 1 = becuase the bullets move to the right
52 BulletMove #( 1 ) BM_p1( clk_2kHz ,
53                         power_p1,
54                         angle_x_p1,
55                         angle_y_p1,
56                         angle_div ,
57                         stage_p1 ,
58                         pos_x_i_p1 ,
59                         pos_y_i_p1 ,
60                         pos_x_p1 ,
61                         pos_y_p1 ,
62                         collision_p1 ,
63                         winner_p1 );
64
65 // barrel and bullet control modules for player 2
66 // -1 = because this tank shoots left
67 // (589,93) = position of the tank for player 2
68 BarrelControl #(-1,589,93) BC_p2 ( pb_pa_p2 ,
69                         pb_select_p2 ,
70                         stage_p2 ,
71                         pos_x_i_p2 ,
72                         pos_y_i_p2 ,
73                         angle_x_p2 ,
74                         angle_y_p2 ,
75                         angle_div2 ,
76                         power_p2 );
77 // 0 = becuase the bullets move to the left
78 BulletMove #( 0 ) BM_p2( clk_2kHz ,

```

```

79         power_p2 ,
80         angle_x_p2 ,
81         angle_y_p2 ,
82         angle_div2 ,
83         stage_p2 ,
84         pos_x_i_p2 ,
85         pos_y_i_p2 ,
86         pos_x_p2 ,
87         pos_y_p2 ,
88         collision_p2 ,
89         winner_p2);
90
91 // changes the stage of player 1
92 // the stage can change when the player push the pb_select or
93 // the bullet goes out of bounds or
94 // the player hits themselves or their enemy
95 always @(pb_select_p1 or collision_p1 or winner_p1 or winner_p2) begin
96     if(collision_p1 == 1) begin
97         stage_p1_n = 0;
98     end else begin
99         if(winner_p1 == 1 || winner_p1 == 2 || winner_p2 == 1 || winner_p2 == 2) begin
100             stage_p1_n = 0;
101         end else begin
102             if(pb_select_p1 == 1) begin
103                 if(stage_p1 >= 4) begin
104                     stage_p1_n = 0;
105                 end else begin
106                     stage_p1_n = stage_p1 + 1;
107                 end
108             end
109         end
110     end
111 end
112
113 // changes the stage of player 2
114 // the stage can change when the player push the pb_select or
115 // the bullet goes out of bounds or
116 // the player hits themselves or their enemy
117 always @(pb_select_p2 or collision_p2 or winner_p2 or winner_p2) begin
118     if(collision_p2 == 1) begin
119         stage_p2_n = 0;
120     end else begin
121         if(winner_p1 == 1 || winner_p1 == 2 || winner_p2 == 1 || winner_p2 == 2) begin
122             stage_p2_n = 0;
123         end else begin
124             if(pb_select_p2 == 1) begin
125                 if(stage_p2 >= 4) begin
126                     stage_p2_n = 0;
127                 end else begin
128                     stage_p2_n = stage_p2 + 1;
129                 end
130             end
131         end
132     end
133 end
134
135 always_comb begin
136     stage_p1 = stage_p1_n;
137     stage_p2 = stage_p2_n;
138 end
139
140 endmodule

```

6.3 BarrelControl.sv

```

1 module BarrelControl #(DIR=1,x_i=240,y_i=320) // dir = 1 for right, dir = 0 for left
2                                         // x_i = the x position of the tank
3                                         // y_i = the y position of the tank
4     (input logic pb_pa, // push button to choose the next power or angle
5      input logic pb_select, // psuh button to lock in power or angle or fire
6      input logic [3:0] stage, // current stage: idle, power select,
7                                         // angle select, fire
8      output logic [11:0] pos_x_i, // initial x position of the ball,
9                                         // and the barrel end
10     output logic [11:0] pos_y_i, // initial y position of the ball,
11                                         // and the barrel end
12     output logic [11:0] angle_x, // scaled initial angle for the
13                                         // ball in the x dir
14     output logic [11:0] angle_y, // scaled initial angle for the
15                                         // ball in the y dir
16     output logic [11:0] angle_div, // scaling factor for the angles
17     output logic [3:0] power); // selected power, 1 - 10
18
19 logic [7:0] barrel_length = 20; // barrel length in pixels
20 logic [3:0] power_n = 0; // counter for the pb_pa push button to select power
21 logic [11:0] angle_n = 0; // counter for the pb_pa push button to select angle
22
23 initial begin
24     angle_x = 1000*DIR;
25     angle_y = 0;
26     power = 0;
27     angle_div = 1000;
28 end
29
30 // gives the calculates the position of the barrel, which is also
31 // the initial position of the ball
32 always_comb begin
33     if(DIR == 1 || DIR== -1) begin
34         pos_x_i = x_i + (barrel_length*angle_x)/angle_div;
35         pos_y_i = y_i + (barrel_length*angle_y)/angle_div;
36     end else begin
37         pos_x_i = 640/2;
38         pos_y_i = 480/2;
39     end
40 end
41
42 // selects the angle or power
43 // if the stage is 1 (power select) then the counter is outputed as the power
44 // if the stage is 2 (angle select) then a scaled angle for each direction is given
45 always @ (posedge pb_select) begin
46     unique case (stage)
47         1: power <= power_n;
48         2: begin
49             unique case (angle_n)
50                 1: begin //0 deg
51                     angle_x <= 1000*DIR;
52                     angle_y <= 0;
53                 end
54                 2: begin // 9 deg
55                     angle_x <= 988*DIR;
56                     angle_y <= 156;
57                 end
58                 3: begin // 18 deg
59                     angle_x <= 951*DIR;
60                     angle_y <= 309;
61                 end
62                 4: begin // 27 deg
63                     angle_x <= 891*DIR;
64                     angle_y <= 454;
65                 end
66                 5: begin // 36 deg
67                     angle_x <= 809*DIR;
68                     angle_y <= 588;
69                 end
70                 6: begin // 45 deg
71                     angle_x <= 707*DIR;
72                     angle_y <= 707;
73                 end
74                 7: begin // 54 deg
75                     angle_x <= 588*DIR;
76                     angle_y <= 809;
77                 end
78                 8: begin // 63 deg

```

```

79          angle_x <= 454*DIR;
80          angle_y <= 891;
81      end
82  9: begin // 72 deg
83      angle_x <= 309*DIR;
84      angle_y <= 951;
85  end
86 10: begin // 81 deg
87      angle_x <= 156*DIR;
88      angle_y <= 988;
89  end
90  default: begin
91      angle_x <= 1000*DIR;
92      angle_y <= 0;
93  end
94  endcase
95 end
96 default: begin
97     angle_x <= 1000*DIR;
98     angle_y <= 0;
99     power <= 0;
100 end
101 endcase
102 end
103
104 // depending on the stage pushing the pb_pa push button
105 // either increases the power or angle counter
106 always @ (posedge pb_pa) begin
107     unique case (stage)
108         1: begin
109             if (power_n > 10) begin
110                 power_n <= 10;
111             end else begin
112                 power_n <= power_n + 1;
113             end
114         end
115         2: begin
116             if (angle_n > 10) begin
117                 angle_n <= 10;
118             end else begin
119                 angle_n <= angle_n + 1;
120             end
121         end
122         default: begin
123             power_n <= 0;
124             angle_n <= 0;
125         end
126     endcase
127
128 end
129
130 endmodule

```

6.4 BulletMove.sv

```

1 module BulletMove #( DIR=1 ) // 1 = ball traveling to the right, 0 = left
2     (input logic clk_2kHz,
3      input logic [3:0] power, // scaled 1 - 10 of how fast the ball travels
4      input logic [15:0] angle_x, // the balls scaled initial
5          // velocity in the x dir
6      input logic [15:0] angle_y, // the balls scaled initial
7          // velocity in the y dir
8      input logic [15:0] angle_div, // velocity scaling factor
9      input logic [3:0] stage, // current stage of this player
10     input logic [11:0] pos_x_i, // initial position of the ball in x dir
11     input logic [11:0] pos_y_i, // initial position of the ball in y dir
12     output logic [11:0] pos_x, // current position of the ball in x dir
13     output logic [11:0] pos_y, // current position of the ball in y dir
14     output logic collision, // flag for a collision
15     output logic [1:0] winner); // flag for win or lose
16
17 logic [19:0] CMf = 10; // the square of counter max for the bullet speed
18 logic [19:0] ans = 0; // answer to a sqrt operation for the counter max
19
20 assign CMf = 4000000 / (4000000/BPPS_PERIOD_y_i/BPPS_PERIOD_y_i - 2*40*(pos_y-pos_y_i));
21
22 sqrt sqrt_0(CMf, ans);
23
24 int BPPS_PERIOD_x = 6; // counter max for the x direction
25 int BPPS_PERIOD_y = 6; // counter max for the y direction
26 int BPPS_PERIOD_y_i = 6; // initial counter max for the y direction
27 logic [15:0] BPPS_count_x = 1; // counter to count up to the time to move in x
28 logic [15:0] BPPS_count_y = 1; // counter to count up to the time to move in y
29 logic [15:0] BPPS_count_x_next = 1; // temp for BPPS_count_x
30 logic [15:0] BPPS_count_y_next = 1; // temp for BPPS_count_y
31
32 logic B_move_x = 0; // a flag to move the ball in the x direction
33 logic B_move_y = 0; // a flag to move the ball in the y direction
34
35 logic y_dir = 1; // changes to -1 when the ball begins to fall
36
37 // this is always block is to detect when the ball has moved out of bounds or
38 // hit a player
39 always @(pos_y or pos_x) begin
40     if(stage == 3) begin // stage 3 is the fired ball phase
41         if(pos_y > 479 || pos_y < 80) begin
42             collision <= 1;
43         end else begin
44             if(pos_x > 639 || pos_x < 1) begin
45                 collision <= 1;
46             end else begin
47                 if(pos_x > 280 && pos_x < 360 && pos_y < 180) begin
48                     collision <= 1;
49                 end else begin
50                     if( DIR ) begin
51                         if(pos_x > 50 && pos_x < 70 && pos_y > 80 && pos_y < 93) begin
52                             winner <= 2;
53                         end else begin
54                             if(pos_x > 569 && pos_x < 589 && pos_y > 80 && pos_y < 93)
55                             begin
56                                 winner <= 1;
57                             end else begin
58                                 collision <= 0;
59                                 winner <= 0;
60                             end
61                         end
62                     end else begin
63                         if(pos_x > 50 && pos_x < 70 && pos_y > 80 && pos_y < 93) begin
64                             winner <= 1;
65                         end else begin
66                             if(pos_x > 569 && pos_x < 589 && pos_y > 80 && pos_y < 93)
67                             begin
68                                 winner <= 2;
69                             end else begin
70                                 collision <= 0;
71                                 winner <= 0;
72                             end
73                         end
74                     end
75                 end
76             end
77         end
78     end
79 end
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
623
624
625
625
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
148
```

```

79         collision <= 0;
80         winner <= 0;
81     end
82 end
83
84
85 // signals a move then the direction counters achieve the right count
86 always @(*) begin
87     if(BPPS_count_x >= BPPS_PERIOD_x) begin
88         B_move_x = ~B_move_x;
89         BPPS_count_x_next = 1;
90     end else begin
91         BPPS_count_x_next = BPPS_count_y + 1;
92     end
93
94     if(BPPS_count_y >= BPPS_PERIOD_y) begin
95         B_move_y = ~B_move_y;
96         BPPS_count_y_next = 1;
97     end else begin
98         BPPS_count_y_next = BPPS_count_y + 1;
99     end
100 end
101
102 // calculates the period of the counters depending on the
103 // angles and power of each direction
104 always @(posedge clk_2kHz) begin
105     if(stage == 2) begin
106         BPPS_PERIOD_x <= ((4*2000*10*angle_div)/(905*angle_x*power));
107         BPPS_PERIOD_y_i <= ((4*2000*10*angle_div)/(905*angle_y*power));
108         BPPS_count_x <= 1;
109         BPPS_count_y <= 1;
110     end else begin
111         if(stage == 3) begin
112             BPPS_count_x <= BPPS_count_x_next;
113             BPPS_count_y <= BPPS_count_y_next;
114             if(ans > 150) begin
115                 BPPS_PERIOD_y <= 150;
116                 y_dir = 0;
117             end else begin
118                 BPPS_PERIOD_y <= ans;
119                 y_dir = y_dir;
120             end
121         end else begin
122             y_dir = 1;
123             BPPS_count_x <= 1;
124             BPPS_count_y <= 1;
125         end
126     end
127 end
128
129
130 // moves the ball in the x dir when the counter finishes
131 // if the player push pbselect the ball will move back to the tank barrel
132 // DIR:
133 // 1 = moves bullet right
134 // 0 = moves bullet left
135 always @(posedge B_move_x) begin
136
137     if(stage == 3) begin
138         if(DIR) begin
139             pos_x <= pos_x + 1;
140         end else begin
141             pos_x <= pos_x - 1;
142         end
143     end else begin
144         pos_x <= pos_x_i;
145     end
146 end
147
148 // moves the ball in the y dir when the counter finishes
149 // if the player push pbselect the ball will move back to the tank barrel
150 always @(posedge B_move_y) begin
151
152     if(stage == 3) begin
153         if(y_dir) begin
154             pos_y <= pos_y + 1;
155         end else begin
156             pos_y <= pos_y - 1;
157         end
158     end else begin

```

```
159         pos_y <= pos_y_i;  
160     end  
161  
162 endmodule  
163
```

6.5 sqrt.sv

```
1 module sqrt(input logic [19:0] num, output logic [19:0] ans);
2
3     logic [19:0] min = 0;
4     logic [19:0] max = 0;
5     logic [19:0] mid = 0;
6     logic [19:0] guess = 0;
7     logic [19:0] count = 0;
8
9 // uses a successive approximation to find the sqrt or a number
10 always @(*) begin
11     min = 0;
12     max = num;
13     count = 0;
14
15     while((max - min) > 1 && count < 50) begin
16         mid = (max + min) / 2;
17         guess = (mid*mid);
18         if(guess <= num) begin
19             min = mid;
20         end
21         if(guess >= num) begin
22             max = mid;
23         end
24         count = count + 1;
25     end
26
27     ans = min;
28 end
29
30 endmodule
```

6.6 colseq.sv

```
1 // colseq.sv - ELEX 7660
2 // Kolyn.Gray 2018-01-22
3
4 module colseq (input logic [3:0] kpr, output logic [3:0] kpc, input logic clk);
5
6     // if no rows are low cycle through making the columns low
7     always_ff@(posedge clk) begin
8         if(kpr == 4'b1111)
9             begin
10                 unique case (kpc)
11                     4'b0111: kpc = 4'b1011;
12                     4'b1011: kpc = 4'b1101;
13                     4'b1101: kpc = 4'b1110;
14                     4'b1110: kpc = 4'b0111;
15                     default: kpc = 4'b1011;
16             endcase
17         end
18     end
19
20 endmodule
```

6.7 kpdecode.sv

```
1 // kpdecode.sv - ELEX 7660
2 // Kolyn.Gray 2018-01-22
3
4 module kpdecode ( input logic [3:0] kpr,
5                   input logic [3:0] kpc,
6                   output logic kphit,
7                   output logic [15:0] num);
8
9 // allows for 4 buttons to be pressed in the same column at once
10 // the idea being depending on the rows that are low
11 // and using the current columns that are low you can determine what buttons are pressed
12 always_comb begin
13   if(kpr != 4'b1111 && kpc != 4'b1111)
14     begin
15       kphit <= 1;
16       unique case (kpr)
17         4'b0000:
18
19           unique case (kpc)
20             4'b0111: num = 16'b0010_0000_0100_1001;
21             4'b1011: num = 16'b0100_0000_1001_0010;
22             4'b1101: num = 16'b1000_0001_0010_0100;
23             4'b1110: num = 16'b0001_1110_0000_0000;
24             default: num = 16'hx;
25           endcase
26         4'b0001:
27           unique case (kpc)
28             4'b0111: num = 16'b0000_0000_0100_1001;
29             4'b1011: num = 16'b0000_0000_1001_0010;
30             4'b1101: num = 16'b0000_0001_0010_0100;
31             4'b1110: num = 16'b0000_1110_0000_0000;
32             default: num = 16'hx;
33           endcase
34         4'b0010:
35           unique case (kpc)
36             4'b0111: num = 16'b0010_0000_0000_1001;
37             4'b1011: num = 16'b0100_0000_0001_0010;
38             4'b1101: num = 16'b1000_0000_0010_0100;
39             4'b1110: num = 16'b0001_0110_0000_0000;
40             default: num = 16'hx;
41           endcase
42         4'b0011:
43           unique case (kpc)
44             4'b0111: num = 16'b0000_0000_0000_1001;
45             4'b1011: num = 16'b0000_0000_0001_0010;
46             4'b1101: num = 16'b0000_0000_0010_0100;
47             4'b1110: num = 16'b0000_0110_0000_0000;
48             default: num = 16'hx;
49           endcase
50         4'b0100:
51           unique case (kpc)
52             4'b0111: num = 16'b0010_0000_0100_0001;
53             4'b1011: num = 16'b0100_0000_1000_0010;
54             4'b1101: num = 16'b1000_0001_0000_0100;
55             4'b1110: num = 16'b0001_1010_0000_0000;
56             default: num = 16'hx;
57           endcase
58         4'b0101:
59           unique case (kpc)
60             4'b0000: num = 16'b0000_1011_1100_0111;
61             4'b0001: num = 16'b0000_0001_1100_0111;
62             4'b0010: num = 16'b0000_1010_1100_0011;
63             4'b0011: num = 16'b0000_0000_1100_0011;
64             4'b0100: num = 16'b0000_1011_0100_0101;
65             4'b0101: num = 16'b0000_0001_0100_0101;
66             4'b0110: num = 16'b0000_1010_0100_0001;
67             4'b0111: num = 16'b0000_0000_0100_0001;
68             4'b1000: num = 16'b0000_1011_1000_0110;
69             4'b1001: num = 16'b0000_0001_1000_0110;
70             4'b1010: num = 16'b0000_1010_1000_0010;
71             4'b1011: num = 16'b0000_0000_1000_0010;
72             4'b1100: num = 16'b0000_1011_0000_0100;
73             4'b1101: num = 16'b0000_0001_0000_0100;
74             4'b1110: num = 16'b0000_1010_0000_0000;
75             4'b1111: num = 16'b0000_0000_0000_0000;
76             default: num = 16'hx;
77           endcase
78         4'b0110:
```

```

79         unique case (kpc)
80             4'b0111: num = 16'b0010_0000_0000_0001;
81             4'b1011: num = 16'b1000_0000_0000_0010;
82             4'b1101: num = 16'b1000_0000_0000_0100;
83             4'b1110: num = 16'b0001_0010_0000_0000;
84             default: num = 16'hx;
85         endcase
86     4'b0111:
87         unique case (kpc)
88             4'b0000: num = 16'b0000_0010_0000_0111;
89             4'b0001: num = 16'b0000_0000_0000_0111;
90             4'b0010: num = 16'b0000_0010_0000_0011;
91             4'b0011: num = 16'b0000_0000_0000_0011;
92             4'b0100: num = 16'b0000_0010_0000_0101;
93             4'b0101: num = 16'b0000_0000_0000_0101;
94             4'b0110: num = 16'b0000_0010_0000_0001;
95             4'b0111: num = 16'b0000_0000_0000_0001;
96             4'b1000: num = 16'b0000_0010_0000_0110;
97             4'b1001: num = 16'b0000_0000_0000_0110;
98             4'b1010: num = 16'b0000_0010_0000_0010;
99             4'b1011: num = 16'b0000_0000_0000_0010;
100            4'b1100: num = 16'b0000_0010_0000_0100;
101            4'b1101: num = 16'b0000_0000_0000_0100;
102            4'b1110: num = 16'b0000_0010_0000_0000;
103            4'b1111: num = 16'b0000_0000_0000_0000;
104            default: num = 16'hx;
105        endcase
106    4'b1000:
107        unique case (kpc)
108            4'b0111: num = 16'b0010_0000_0100_1000;
109            4'b1011: num = 16'b1000_0000_1001_0000;
110            4'b1101: num = 16'b1000_0001_0010_0000;
111            4'b1110: num = 16'b0001_1100_0000_0000;
112            default: num = 16'hx;
113        endcase
114    4'b1001:
115        unique case (kpc)
116            4'b0111: num = 16'b0000_0000_0100_1000;
117            4'b1011: num = 16'b0000_0000_1001_0000;
118            4'b1101: num = 16'b0000_0001_0010_0000;
119            4'b1110: num = 16'b0000_1100_0000_0000;
120            default: num = 16'hx;
121        endcase
122    4'b1010:
123        unique case (kpc)
124            4'b0111: num = 16'b0010_0000_0000_1000;
125            4'b1011: num = 16'b0100_0000_0001_0000;
126            4'b1101: num = 16'b1000_0000_0010_0000;
127            4'b1110: num = 16'b0001_0100_0000_0000;
128            default: num = 16'hx;
129        endcase
130    4'b1011:
131        unique case (kpc)
132            4'b0111: num = 16'b0000_0000_0000_1000;
133            4'b1011: num = 16'b0000_0000_0001_0000;
134            4'b1101: num = 16'b0000_0000_0010_0000;
135            4'b1110: num = 16'b0000_0100_0000_0000;
136            default: num = 16'hx;
137        endcase
138    4'b1010:
139        unique case (kpc)
140            4'b0111: num = 16'b0010_0000_0100_0000;
141            4'b1011: num = 16'b0100_0000_1000_0000;
142            4'b1101: num = 16'b1000_0001_0000_0000;
143            4'b1110: num = 16'b0001_1000_0000_0000;
144            default: num = 16'hx;
145        endcase
146    4'b1101:
147        unique case (kpc)
148            4'b0000: num = 16'b0000_1001_1100_0000;
149            4'b0001: num = 16'b0000_0001_1100_0000;
150            4'b0010: num = 16'b0000_1000_1100_0000;
151            4'b0011: num = 16'b0000_0000_1100_0000;
152            4'b0100: num = 16'b0000_1001_0100_0000;
153            4'b0101: num = 16'b0000_0001_0100_0000;
154            4'b0110: num = 16'b0000_1000_0100_0000;
155            4'b0111: num = 16'b0000_0000_0100_0000;
156            4'b1000: num = 16'b0000_1001_1000_0000;
157            4'b1001: num = 16'b0000_0001_1000_0000;
158            4'b1010: num = 16'b0000_1000_1000_0000;

```

```

159      4'b1011: num = 16'b0000_0000_1000_0000;
160      4'b1100: num = 16'b0000_1001_0000_0000;
161      4'b1101: num = 16'b0000_0001_0000_0000;
162      4'b1110: num = 16'b0000_1000_0000_0000;
163      4'b1111: num = 16'b0000_0000_0000_0000;
164      default: num = 16'hx;
165      endcase
166  4'b1110:
167      unique case (kpc)
168          4'b0111: num = 16'b0010_0000_0000_0000;
169          4'b1011: num = 16'b0100_0000_0000_0000;
170          4'b1101: num = 16'b1000_0000_0000_0000;
171          4'b1110: num = 16'b0001_0000_0000_0000;
172          default: num = 16'hx;
173      endcase
174  4'b1111: num = 16'b0000_0000_0000_0000;
175      default: num = 16'hx;
176  endcase
177 end
178 else
179 begin
180     kphit <= 0;
181     num = 16'hx;
182 end
183 end
184
185 endmodule

```

6.8 vga.sv

```

1 //bitwise operation to define a rectangle during the sweep
2 `define BLOCK(a) (((CounterX >= `a`.l)&&(CounterX <= `a`.r))\
3     &&((CounterY >= `a`.t)&&(CounterY <= `a`.b)))
4
5 //bitwise operation to draw a rectangular point with a "radius" of 2*thickness
6 `define POINT(x,y,thickness) ((CounterX >= x - thickness)&&(CounterX <= x + thickness))\
7     &&((CounterY >= y - thickness)&&(CounterY <= y + thickness))
8
9 //line drawing algorithm for positive slopes
10 `define LINEPOS(line,thickness) (\`CounterY ==(`line`.y2 + (`line`.my*(CounterX -`line`.x2))/`line`.mx))\
11     && ((CounterX >= `line`.x1) && (CounterX <= `line`.x2))\
12     && (CounterY >= `line`.y2) && (CounterY <= `line`.y1))
13
14 //line drawing algorithm for negative slopes
15 `define LINENEG(line,thickness) (\`CounterY ==(`line`.y1 + (`line`.my*(CounterX -`line`.x1))/`line`.mx))\
16     && ((CounterX >= `line`.x1) && (CounterX <= `line`.x2))\
17     && (CounterY >= `line`.y1) && (CounterY <= `line`.y2))
18
19 module vga(bulletx,bullety,barrel1x,barrel1y,barrel2x,barrel2y,
20             clk, vga_h_sync, vga_v_sync, vga_R, vga_G, vga_B);
21
22 //Position variables
23 input logic[9:0] bulletx;
24 input logic[8:0] bullety;
25 input logic[9:0] barrel1x;
26 input logic[8:0] barrel1y;
27 input logic[9:0] barrel2x;
28 input logic[8:0] barrel2y;
29 input logic clk;
30 output vga_h_sync, vga_v_sync, vga_R, vga_G, vga_B;
31
32 //modified sync generator from fpga4fun.com pong code
33 hvsync_generator syncgen(.CLOCK_50(clk),
34                         .vga_h_sync(vga_h_sync),
35                         .vga_v_sync(vga_v_sync),
36                         .inDisplayArea(inDisplayArea),
37                         .CounterX(CounterX),
38                         .CounterY(CounterY));
39
40
41 ///////////////////////
42 logic inDisplayArea; //allows writing to only 640x480 area as seen below
43 logic [9:0] CounterX; //x write position
44 logic [8:0] CounterY; //y write position
45
46 //rectangles go left right top bottom
47 typedef struct {
48     logic[9:0] l;
49     logic[9:0] r;
50     logic[8:0] t;
51     logic[8:0] b;
52     logic[2:0] ob_col; //3 bit RGB
53 } rect;
54
55
56 //lines work better if pt1 is to the left of pt2
57 typedef struct {
58     logic[9:0] x1;
59     logic[9:0] x2;
60     logic[8:0] y1;
61     logic[8:0] y2;
62     logic[9:0] mx; //dx
63     logic[8:0] my; //dy
64     logic[2:0] line_col; //3 bit RGB
65 } line_seg;
66
67
68 //3 channel output variables
69 logic R, G, B;
70
71 //MAIN TERRAIN
72 rect land = '{0,639,400,479,3'b110};
73 logic land_draw;
74
75 //Middle obstacle
76 rect barrier = '{280,360,300,400,3'b110};
77 logic barrier_draw;
78

```

```

79
80 //TANKS
81 rect tank1 = '{50,70,387,400,3'b010};
82 logic tank1_draw;
83
84 rect tank2 = '{569,589,387,400,3'b100};
85 logic tank2_draw;
86
87 //BARREL
88 line_seg barrel = '{50,70,387,375,0,0,3'b111};
89 logic barrel_draw;
90
91 line_seg barrel2 = '{569,589,375,387,0,0,3'b111};
92 logic barrel2_draw;
93
94 //barrel position assignments
95 assign barrel.y2 = barrel1y;
96 assign barrel.x2 = barrel1x;
97
98 assign barrel2.y1 = barrel2y;
99 assign barrel2.x1 = barrel2x;
100
101 //bullet on off drawing logic
102 logic bullet = 0;
103 logic bullet2 = 0;
104
105 //calculate line slopes, check if divide by zero
106 always_comb
107 begin
108   if(barrel.x2-barrel.x1)
109     barrel.mx = barrel.x2-barrel.x1;
110   barrel.my = barrel.y2-barrel.y1;
111
112   if(barrel2.x2-barrel2.x1)
113     barrel2.mx = barrel2.x2-barrel2.x1;
114   barrel2.my = barrel2.y2-barrel2.y1;
115 end
116
117 //draw it all
118 always_comb
119 begin
120
121   //conditions under which each object is drawn using defines above
122   land_draw = 'BLOCK(land);
123   barrel_draw = 'POINT(barrel.x1,barrel.y1,0) ||
124           'POINT(barrel.x2,barrel.y2,1) ||
125           'LINEPOS(barrel,1);
126
127   barrel2_draw = 'POINT(barrel2.x2,barrel2.y2,0) ||
128           'POINT(barrel2.x1,barrel2.y1,1) ||
129           'LINENEG(barrel2,1);
130
131   tank1_draw = 'BLOCK(tank1);
132   tank2_draw = 'BLOCK(tank2);
133   barrier_draw = 'BLOCK(barrier);
134   bullet = 'POINT(p1bulletx,p2bullety,3);
135   bullet2 = 'POINT(p2bulletx,p2bullety,3);
136
137
138   //Output channels before checking for display area
139   R = bullet || (land_draw & land.ob_col[2]) || (barrier_draw & barrier.ob_col[2]) ||
140       (barrel2_draw & barrel2.line_col[2]) ||(barrel_draw & barrel.line_col[2]) ||
141       (tank1_draw & tank1.ob_col[2]) || (tank2_draw & tank2.ob_col[2]);
142
143   G = bullet || (land_draw & land.ob_col[1]) || (barrier_draw & barrier.ob_col[1]) ||
144       (barrel2_draw & barrel2.line_col[2]) ||(barrel_draw & barrel.line_col[1]) ||
145       (tank1_draw & tank1.ob_col[1]) || (tank2_draw & tank2.ob_col[1]);
146
147   B = bullet || (land_draw & land.ob_col[0]) || (barrier_draw & barrier.ob_col[0]) ||
148       (barrel2_draw & barrel2.line_col[2]) ||(barrel_draw & barrel.line_col[0]) ||
149       (tank1_draw & tank1.ob_col[0]) || (tank2_draw & tank2.ob_col[0]);
150
151 end
152
153 //final output variables
154 logic vga_R, vga_G, vga_B;
155
156 always @(posedge clk)
157 begin
158   vga_R <= R & inDisplayArea;

```

```
159     vga_G <= G & inDisplayArea;  
160     vga_B <= B & inDisplayArea;  
161  
162  
163  
164 endmodule
```

6.9 hvsync_generator.sv

```
1 module hvsync_generator(CLOCK_50 ,
2                         vga_h_sync ,
3                         vga_v_sync ,
4                         inDisplayArea ,
5                         CounterX ,
6                         CounterY);
7
8   input CLOCK_50;
9   output vga_h_sync , vga_v_sync ;
10  output inDisplayArea ;
11  output [9:0] CounterX ;
12  output [8:0] CounterY ;
13
14 //Counter ranges sized according to screen size
15 ///////////////////////////////
16 logic [9:0] CounterX ;
17 logic [8:0] CounterY ;
18 logic CounterXmaxed ;
19
20 logic clk ;
21 logic clk2 ;
22 //25.175 MHz clock source
23 clock25 clockgen(CLOCK_50 , clk );
24
25 //counter x maxed flag variable
26 always_comb
27 begin
28   CounterXmaxed = (CounterX==813) ;
29 end
30
31
32 always @ (posedge clk)
33 if(CounterXmaxed)
34   CounterX <= 0 ;//reset counter at max
35 else
36   CounterX <= CounterX + 1 ;//otherwise increment counter
37
38 always @ (posedge clk) begin
39   //This variable is allowed to overflow instead of resetting
40   if(CounterXmaxed) CounterY <= CounterY + 1 ;
41 end
42
43
44 logic vga_HS , vga_VS ;
45 always @ (posedge clk)
46 begin
47 // change this value to move the display horizontally
48   vga_HS <= (CounterX[9:4]==47) ;
49   // change this value to move the display vertically
50   vga_VS <= (CounterY==500) ;
51 end
52
53 logic inDisplayArea ;
54 always @ (posedge clk)
55 if(inDisplayArea==0)
56   inDisplayArea <= (CounterXmaxed) && (CounterY<480) ;
57 else
58   inDisplayArea <= !(CounterX==639) ;
59
60 assign vga_h_sync = ~vga_HS ;
61 assign vga_v_sync = ~vga_VS ;
62
63 endmodule
```