MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

EE314
DIGITAL ELECTRONICS LABORATORY
SPRING '24

TERM PROJECT REPORT
FPGA IMPLEMENTATION OF ISOMETRIC SHOOTER GAME

GROUP 45

Emre Kumlu - 2516474
Furkan Akkoyun- 2518504
Şevket Ege Yeşilyurt -2517233
Yusuf Baran - 2574747

I. INTRODUCTION
In this report, we will examine our term project for the Digital Electronics Laboratory course. First, our primary task is creating an isometric shooter game using Verilog HDL on an FPGA platform. Second, we will start with a detailed problem definition, outlining the game's objectives and specifications. Third, we will provide an overview of the VGA interface that we will use for game display. Fourth, we will investigate the solution approach, explain the design and implementation of game components. Fifth, we will also discuss the challenges we encounter during development and give possible solutions to overcome them. An evaluation of the results will happen how the project meets the specified criteria. Finally, we will conclude with a general discussion of the project, its contributions, and potential future developments.
II. PROBLEM DEFINITION
The purpose of this project is the development of the game's logic, visual interface, and the interaction mechanisms via the FPGA hardware. The game is based on classic arcade shooters such as Space Invaders. According to the project specifications, the player directs a spaceship located at the center of the game field. This spaceship can rotate but cannot move and must defend against enemies that appear at the boundaries and move towards the center. The player must strategically rotate the spaceship to aim and fire projectiles, destroying the incoming enemies before they reach and collide with the spaceship, which would end the game. The game field will be displayed using a VGA interface, supporting a resolution of 640 x 480 pixels. The enemies will spawn at predefined angles, move towards the spaceship, and vary in type and health. The player will have two shooting modes to choose from, offering different projectile spreads and damage levels.
III. VIDEO GRAPHICS ARRAY (VGA)
Brief Explanation of VGA Display Standard
VGA is a display protocol standard introduced by IBM. Nowadays, other display standards such as HDMI and DisplayPort are becoming more popular; however, alongside these digital connection standards, VGA, which operates with analog signals, is still in use. The fact that the VGA standard is analog allows it to determine pixel color intensity by continuously varying voltages between 0 and 0.7 volts. This protocol includes parameters such as resolution, signal timing, and color depth. VGA typically operates at a resolution of 640x480 pixels with a refresh rate of 60Hz.
Important definitions for Signal Generation and Basics of VGA
HSYNC: Stands for Horizontal Synchronization, this signal is sent to the monitor to indicate end of the horizontal line of pixels.
VSYNC: Stands for Vertical Synchronization, this signal is sent to the monitor to indicate end of the vertical line of pixels. In other words, it marks the end of a frame, indicating that all HSYNC signals for a frame have been sent.
Blanking intervals: These intervals are periods during which no displayable data is sent to the monitor. They occur after HSYNC and VSYNC signals, allowing time for the electron beam to reset before the start of the next line or frame.
Input Clock: This signal is sent to the VGA driver module (explained further in the rest of this report) and enables pixel processing at 25 MHz.
Reset (Active high): This is an asynchronous reset signal for the VGA driver module.
Color Input (to VGA driver): This signal is sent to the VGA driver module in the format of 8 bits, where the first 3 bits represent the intensity of red, the next 3 bits represent green intensity, and the final 2 bits represent blue intensity for the pixel being processed at that time.
Next X: This signal is output from the VGA driver module and indicates the x-axis coordinate of the pixel to be processed in the next clock cycle.
Next Y: This signal is output from the VGA driver module and indicates the y-axis coordinate of the pixel to be processed in the next clock cycle.
Red, Green, Blue: These signals are sent to the VGA connector's digital-to-analog converter (DAC) and represent the intensities of red, green, and blue for the pixel currently being processed.
SYNC: This signal is sent to the monitor and indicates the start of a new HSYNC or VSYNC.
Clock Input (to VGA connector): This signal is sent to the monitor and ensures that the VGA driver and VGA connector operate synchronously at 25 MHz clock with a refresh rate of 60 Hz. With the signals above, our VGA driver module goes over the pixels on the 640x480 resolution screen at 60Hz refresh rate, HSYNC and VSYNC indicates end of horizontal line and frame respectively to fully control the process, at the same time we assign corresponding color to each pixel with Next X and Next Y. To ensure smooth and accurate display at a 60 Hz refresh rate (at 640x480 resolution), the clock inputs should be approximately 25 MHz.
HSYNC and VSYNC during VGA protocol
Both HSYNC and VSYNC are started as high voltage. For each horizontal line in display, HSYNC remains high for 640 cycles (1 horizontal line), then again high for 16 more cycles (stabilization before ending the current horizontal line which is called front porch), then low for 96 cycles (indicating end of a horizontal line), then high for 48 cycles (preparation for starting to next horizontal line which is called back porch). For each frame in display, VSYNC remains high for 480 horizontal lines (307200 cycles), then again high for 10 lines (stabilization before ending the current frame which is called front porch), then low for 2 lines (indicating end of a frame), then high for 33 lines (preparation for starting to next frame which is called back porch).
Literature Research for VGA
In order to understand VGA protocol and write a well-working module that works as VGA driver, we have read and thoroughly understood the sources below.
Documentation of Manufacturer: DE1-SoC User Manual File is provided to us in ODTUClass, which includes VGA's parameter values and pin assignments in page 34-35.
VGA Driver Page of V. Hunter Adams: This page includes a brief explanation of VGA protocol and a Verilog HDL implementation of VGA driver.
Online Forums: Websites, such as, Stack Overflow and Intel's community forum. These websites include some people's questions about VGA protocol and corresponding answers related to this topic.
8-bit RGB Color Palette from Department of Mathematics, University of Texas at Austin: This pdf file shows corresponding color for each 8'bRRRGGGBB defined color code.

Based on the sources mentioned above, the most critical information we learned are as follows:
- Without the necessary front and back porch timings for HSYNC and VSYNC, unstable images and incorrect color interpretation could occur.
- Despite there being 18432k cycles of data transmission required for a resolution of 640x480 at a refresh rate of 60Hz, we need a 25 MHz clock because HSYNC, VSYNC, and blank signals also need to be accounted for, which totals up to approximately 25k cycles.
- Color depth is encoded in the format 8'bRRRGGGBB, offering an alternative coding method for each color with 255 options in the RGB system.

More findings related to the project explained in **Intersection of the Project and VGA** section below.
First Attempts for a Proper VGA Driver Module
We first tried to create simple VGA signal generator module, we focused on creating correct HSYNC, VSYNC, and SYNC signals. Therefore, we could display pre-determined colors (like 8'b11111111). We set 25Mhz clock and made triggers at necessary instants (after every 640-clock cycle, after every line, after every frame, pulse for porches etc.). However, these attempts resulted in unstable and flickering line segments, and distorted rectangular shapes on the screen. In addition to that, we used the correct 25Mhz clock for generating signals, but there was a synchronization

problem, such as, we wanted to see a disk on the screen, but it was painted as half-disk. We understood that these problems were about timing inaccuracies. We also observed these errors in waveform analysis. To solve these, we adjusted the timing parameters to fine-tune the porch intervals and SYNC pulses by adding extra counters. These attempts partially solved our issues, but the distorted images did not completely disappear.
Then we looked online resources and found the VGA Driver Page of V. Hunter Adams, this page explains the VGA protocol very clearly, and includes a well-working example code for VGA driver module (Adams, n.d). We directly integrated this module into our project.

Intersection of the Project and VGA

To provide a display at 640x480 resolution with a 60Hz refresh rate, similar to the standard VGA protocol, the module we took from VGA Driver Page by V.Hunter Adams was sufficient. This code generates the necessary signals, but to ensure the game is properly shown on the monitor, all pixels on the screen need to be consistently assigned their required colors to show all visual modules (the spaceship, enemies, scoreboard, game over screen, and background image) based on their current positions, shapes, colors, and values. These modules are visible or absent based on specific conditions and have varying colors and shapes depending on current parameters (e.g., shooting mode, enemy shape and health, and the score to be displayed on the scoreboard). We ensure these modules are displayed appropriately at all times according to their current parameters. The backend implementation, and registration of these changes are detailed in the IV. SOLUTION APPROACH section. The programs described in the SOLUTION APPROACH section allow for real-time processing of the registered values. All of the following modules operate with a 25MHz clock and take the processed pixel (next_x and next_y) as input and give the assigned color as output.

```
module yildiz_kumlu
Input: next x, next y
Define inner radius 100, outer radius 140
For pixels within 140 pixels and outside 100 pixels:
    Assign color 8'b11111111
For pixels within 100 pixels:
    Assign color 8'b00100111
    Assign color 8'b00000000 to all other pixels
Repeat across 640x480 resolution
Output: assigned_color
```

Pseudocode 1: background

```
module color hierarchy
Input: next_x, next_y, gameover
Output: assigned color
If gameover == 1:
    If scoreboard_color != 8'b00000000:
        assigned color = scoreboard color
    Else:
        assigned color = game over display color
Else:
    assigned color = game over display color
    If spaceship_body_color != 8'b00000000:
        assigned_color = spaceship_body_color
    Else if scoreboard color != 8'b00000000:
        assigned_color = scoreboard_color
    Else if enemy movement color != 8'b00000000:
        assigned_color = enemy_movement_color
    Else:
        assigned color = yildiz kumlu color
Output: assigned_color
```

Pseudocode 2: color mechanism

For the background image: It is processed in the pseudocode 1 **yildiz_kumlu** module. Two radii are defined: 100 (inner oval radius) and 140 (outer oval radius). Pixels within 140 pixels and outside 100 pixels are assigned the color 8'b11111111. Pixels within 100 pixels are assigned 8'b01000111. Pixels outside both are assigned 8'b00000000. This process is repeated across the entire 640x480 resolution periodically, creating a background pattern of ovals.

For the overall display: All the colors output by these modules need to be systematically assigned. All modules described above assign black (8'b00000000) to pixels that do not require a specific color. Using this, the pseudocode 2 **color_hierarchy** module is created. If a game over occurs and the scoreboard module does not return black, the color from this module is given to the VGA driver module. Else, the color from the game_over_display module is given. If there is no game over, the spaceship_body module's color is used if it does not return black. Else if the scoreboard module does not return black, the scoreboard module's color is used. Else if the enemy_movement module does not return black, the enemy_movement module's color is used. Else, the color from the yildiz_kumlu module is given to the VGA driver module. Thus, the system prioritizes colors. This ensures that all game components are properly displayed on the screen.

IV. SOLUTION APPROACH

This section outlines the proposed solution for developing the game. Firstly, the overall system architecture is illustrated using a block diagram, which highlights the communication and interaction between the submodules such as spaceship control, enemy dynamics, and the VGA interface. Then, each submodule is described in detail, including the design decisions, implementation strategies, and the rationale behind choosing specific methods or algorithms. Diagrams, state diagrams, and pseudocodes are also provided where necessary to illustrate better as shown in figure 1.

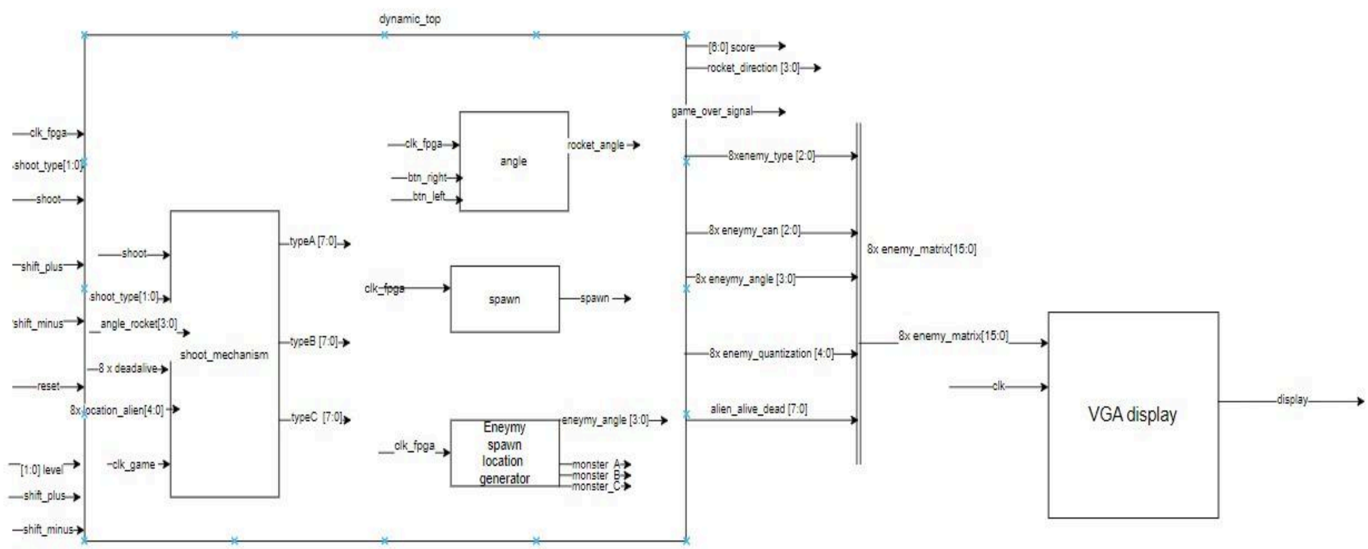

Figure 1: Block Diagram of the Modules

Spaceship Control

We choose circle for the spaceship body, when spaceship rotates thanks to the symmetry of the circle spaceship body remains unchanged. In order to specify the direction of the rocket we added small circle shaped muzzle as shown in figure 2.

Figure 2 Image of the Spaceship

Game screen has been divided 16 angles in order to provide sufficient gaming experience. Spaceship rocket had to rotate between these 16 angles. This angle information modeled with a 4 bit register in "angle" submodule. Rotation of the rocket controlled with buttons of fpga. 2 buttons of fpga had been assigned to rotate clockwise and counter clockwise directions. Algorithm for moving rocket is given below pseudocode 3. Depending the current angle value, display of spaceship and muzzle (on its current position) is done as follows. The spaceship_angle and the first two switches (SW[1:0], used to determine the shooting mode) are processed in the **spaceship_body** module. This module takes spaceship_angle, and SW[1:0] values as input. For the current spaceship_angle value, the position of the small disk representing the spaceship's weapon muzzle is updated within a 7-pixel radius around this position are assigned different colors based on the shooting mode (SW[1:0]): 0,1: 8'b11111111, 2: 8'b0011111, 3: 8'b11110000. Pixels within a 20-pixel radius disk centered at (x-axis=320, y-axis=240) are assigned the color 8'b11100000. All others are assigned 8'b00000000. Below pseudocode 4 is the spaceship_body pseudocode.

```
module angle(clk, btn_right, btn_left, angle);
   input clk;
   input btn_right;
   input btn_left;
   reg angle1;
   reg angle2;
   output angle;
   on negedge btn_right:
      angle1 = angle1 + 1;
   on negedge btn_left:
      angle2 = angle2 + 1;
   always:
      angle = angle1 - angle2;
endmodule
```

Pseudocode 3: angle

```
module spaceship body
Input: spaceship_angle, SW[1:0]
Update muzzle position based on spaceship_angle
For pixels within a 7-pixel radius around muzzle:
   Assign color based on SW[1:0]
   If SW[1:0] == 0 or 1: color = 8'b11111111
   If SW[1:0] == 2: color = 8'b00111100
   If SW[1:0] == 3: color = 8'b11110000
For pixels within a 20-pixel radius disk at center (320, 240):
   Assign color 8'b11110000
   Assign color 8'b00000000 to all other pixels
Output: assigned_color
```

Pseudocode 4: spaceship_body

Shooting Mechanism

The gameplay consists of three distinct shooting type. TypeA damages three health points, and can damage only in alignment of rocket and enemy. TypeB damages 2 health points and can damage in 45 degree view. TypeC damages 1 health points and can damage in 90 degree view. 3 distinct shooting modes provide flexibility for players Player can access different shooting modes by switching SW[0,1]. 00 and 01 refers to typeA, 10 refers to typeB and 11 refers to typeC shooting mode. Also for shooting a button of fpga has used. Player sets desired shooting mode by switches and uses button to shoot enemies. In order to implement this in the shooting mechanism module code decides whether an alive enemy is damaged or not according to the angle between rocket and enemy. To do this we have divided the screen into 16 angle, according to difference between rocket angle and enemy angle shoot mechanism decides if enemy is shooted and by which type. To illustrate, if shooting is typeC, the enemy is damaged 1 health point as long as the angle difference between rocket and the enemy is equal or smaller than 2 (out of 16 angle). As we can see from 1 shoot_mechanism module takes all necessary informations and gives 8 bit 3 registers to model which enemy shooted by which mode of shooting with using the algorithm as follows. (typeA = 'b11111110 means all enemies except enemy 8 have shooted with shooting mode A) Then in the top module necessary health point reducing arrangements are done according to typeA, typeB, and typeC arrays.

A pseudo code which I explained the content in the previous paragraph is provided only for enemy1. In actual code we use following code for 8 different enemies. Pseudocode 5 for shooting is shown below.

```
On the falling edge of shoot:
        If shoot == 0 and shoot_type[1] == 0:
        If angle_rocket == angle_enemy1 and aliveness_of_enemy1 == 1:
        Set typeA[0] to 1
        If shoot == 0 and shoot_type == 2'b10:
        If (angle_rocket == angle_enemy1 or angle_rocket - 1 == angle_enemy1 or
angle_rocket + 1 == angle_enemy1) and aliveness_of_enemy1 == 1:
        Set typeB[0] to 1
        If shoot == 0 and shoot_type == 2'b11:
        If (angle_rocket == angle_enemy1 or angle_rocket - 1 == angle_enemy1 or
angle_rocket + 1 == angle_enemy1 or angle_rocket - 2 == angle_enemy1 or angle_rocket + 2
== angle_enemy1) and aliveness_of_enemy1 == 1:
        Set typeC[0] to 1
```

Pseudocode 5: shooting

Enemy Dynamics

On dynamic_tops module each enemy has an register which represents its different features such as angle, location,aliveness,type and health. Dynamics_top module sends this informations about all 8 different enemies to display unit real time. Main point is all operations are based on changing this registers. Enemies are spawning randomly with a constant frequency. Based on a wide range of player experiences, 0.5 spawn/second frequency and randomly assigned angeles were chosen. In order to generate this functionality, enemy_spawn_signal_generator was used. This module generates 1 pulse signal (1 clock duration) every 2 seconds. Pseudocode zzz is given for signal generator below. Dynamic_top module owns a logic block which uses this pulse signal as following to born one of the enemies which is dead if there is not 8 enemies alive. Thanks to this enemy number never exceeds 8 anyway there is not any extra register that represents enemy 9 so it was already impossible. Also one can observe this logic block also creates enemies immediately if the number of enemy is lower than 2. This process lasts 2 clock cycles and since it uses fpga clock it is actually almost real time. This block looks next dead enemy if previous is alive. Part of the explanation of dynamic top module is given as Pseudocode7 below.

```
module dynamic_top
if (spawn is 1 AND total_alivenumber is not equal to 8)
OR (total_alivenumber is less than 2) then
if alien_1_dead_alive is 0 then
alien_1_dead_alive = 1
alien_1_can = random_type_health alien_1_angle = random_angle
alien_1_type = random_type
alien_1_location=31
```

Pseudocode 6: dynamic_top4

```
module enemy_spawn_signal_generator
   input clk_fpga
   output spawn
   on every positive edge of clk_fpga  if counter equals 100000000 then
        counter = 0,spawn = 1
      else
        spawn = 0  , counter = counter + 1
endmodule
```

Pseudocode 7: signal generator

This logic block used random_type and random_angle signal. Location signal always set to 31 because we divided 26 subparts whole trajectories. All enemies spawn to farthest points of all trajectories. Also spawning healths predetermined since type is setted. For 2 random signals a 16 bits LFSR has been used. 16 bit initially assigned to a random number. This LFSR using 4 bits as taps and gives MS 4 bits as location signal and LS 2 bits as type signal. These taps have been chosen in order to decrease the probability that when we reset the game randomly spawned 2 enemies are not at the same location. So consecutive 2 location signals are different generally. But since there are 3 different types and 2 bit type signal models 4 different codes additional combinational block has been used to assign 2 of the 4 codes to weakest enemy because most probabilistic enemy type should be weakest as desired. (TypeA => 2 health, lateral rectangular / TypeB => 3 health , vertical rectangular / TypeC => 5 health , (+) plus sign). Also instant healt values are visible from enemies colors. (5 => white, 4 => orange, 3 => green, 2 => purple ,1 => red) For further explanation pseudocode 8 is given below for enemy spawn location generation. Also shapes of enemies are visible at results part.

```
module enemy_spawn_location_generator
   input clock
   output output_location
   output monster_A, monster_B, monster_C
  rising_edge (clock)
      internal_reg = (shift internal_reg right by 1 bit) (MSB => XOR of bits 0, 2, 3, and 5 )
   output_location = MSB 4 bits of internal_reg
   if type is 0 or 1 then
      spawn monster A
      do not spawn monster B
      do not spawn monster C
   else if type is 2 then
      do not spawn monster A
      spawn monster B
      do not spawn monster C
   else if type is 3 then
      do not spawn monster A
      do not spawn monster B
      spawn monster C
end module
```

Pseudocode 8: random generator

In order to create 3 different shapes for 3 different types. First one is 20 pixels width and 10 pixels height. Second one is 10 pixels width and 20 pixels height. Last one is the superpose of these two styles (some "+" kind shape).
In order to draw the shapes to monitor, we implement the following mechanism. The type, angle, distance, health, and alive/dead status are processed in the **enemy_movement** module. This module takes all enemy registers as input. For each enemy, if it is alive, the position is calculated based on the angle and distance, and the appropriate color is assigned based on its health. The type information determine the pre-defined shape (geometry), and pixels within this geometry are painted in the assigned color. All other pixels are assigned the color 8'b00000000.

```
If type equals 3'b100: // Shape 1
   If x_position is greater than (next_x - 10) and x_position is less than (next_x + 10) and
     y_position is greater than (next_y - 5) and y_position is less than (next_y + 5):
      Set color to enemy_color.
Else If type equals 3'b010: // Shape 2
   If x_position is greater than (next_x - 5) and x_position is less than (next_x + 5) and
     y_position is greater than (next_y - 10) and y_position is less than (next_y + 10):
      Set color to enemy_color.
Else If type equals 3'b001: // Shape 3
   If (x_position is greater than (next_x - 5) and x_position is less than (next_x + 5) and
      y_position is greater than (next_y - 10) and y_position is less than (next_y + 10)) or
     (x_position is greater than (next_x - 10) and x_position is less than (next_x + 10) and
      y_position is greater than (next_y - 5) and y_position is less than (next_y + 5)):
      Set color to enemy_color.
```

Pseudocode 9: 3 different shape generator for 3 different type

```
module enemy  movement
Input: enemy_registers (type, angle, distance, health, alive/dead)
For each enemy:
  If enemy is alive:
     Calculate position based on angle and distance
     Determine shape based on type
     Assign color based on health
     Paint pixels within pre-defined shape with assigned color
  Else:
     Assign color 8'b00000000 to other pixels
Output: assigned  color
```

Pseudocode 10 : enemy movement

All dynamics of the game based on movement of enemies towards to center. So all enemy locations should change periodically. This frequency should depend on chosen level of game. If player is at hard mode enemies should move faster. One may think that clock divider is useful. Correct but instead clk_divider for this design generated counter_limit signals have used (this is because verilog does not allowed arrangement under two different clock.). For level input 2 switches of fpga used. We have 4 different levels which determine counter limit signal to adjust frequency of movement as following combinational block given in pseudocode 12. As one can observe even at easy mode 2 s is equal to spawning period so only potential error is at the beginning of the game for crossing enemies. On other scenarios enemies already moved to another location before new one spawned . So as explained before for starting of the game and duration there is no probability that 2 enemies are at the same location and angle (checking previous value of random angle and assign it to another location if next one is exactly same).

```
if internal_counter equals counter_limit then
            reset internal_counter
            if alien_location is greater then 5 then
                        alien_location = alien_location -1
else internal_counter increase 1
```

```
level = 00: counter_limit = 25000000; easy 2 s
level = 01: counter_limit = 25000000/2; regular 1 s
level = 10: counter_limit = 25000000/4; hard 0.5 s
level = 11: counter_limit = 25000000/8; extreme 0.25 s
```

Pseudocode 11: enemy moving to center *Pseudocode12: game levels*

Thanks to the code explained in pseudocode 11 above we periodically move enemies towards center of the screen. Counter counts till limit value reached and reset itself simultaneously decreasing location 1. Location module calculate coordinates (x, y) based on a given angle and a location input. It aims at placing objects at specific positions on the screen. It has two input: angle (4 bits) and location_input (5 bits). The angle indicates the direction, with 16 possible values ranging from 0 to 15, while the location_input show distance from center, with 32 possible values between 0 to 31. The outputs are x and y (both 10 bits), which represents calculated coordinates.A register radius is set to 170, representing the maximum distance from the center. The calculations uses approximate sine and cosine values for the angles, scales them by location_input and the radius (170), and then divides by 32 to fit the 5-bit range. The case statement handles each angle value from 0 to 15. For each angle, x and y coordinates are calculated using multipliers for sine and cosine values. For example, at 4'd0 (0 degrees), the coordinates are, x = 320 + (location_input * 0) / 32 and y = 240 - (location_input * 170) / 32. At 4'd1 (22.5 degrees), the coordinates are, x = 320 + (location_input * 65) / 32 and y = 240 - (location_input * 157) / 32. We made similar calculations for other angles.

Player Score and Game Over Conditions

In gameplay, the score increases if the player destroys an enemy. This condition is checked in the dynamic_top module. It checks if an alive enemy died after the shooting, score is incremented by 1.

The score information is processed in the **scoreboard** module. This module takes the score as input. Pre-defined numbers are stored in registers in an 11x10 pixel area (formatted as [10:0] font [0:9] [0:9]). The scoreboard is designed for two digits. The score is divided by 10 to determine the tens digit, and the remainder is the units digit. The scoreboard's center is set at (x-axis=500, y-axis=50). If the processed pixel is within the first half of the scoreboard's width (i.e., the tens digit) and within the determined tens digit, it is assigned the color 8'b00011100. Some process is done for second half of scoreboard's width for units digit. All others are assigned 8'b00000000. An easy explanation for scoreboard display is given as pseudocode 13 below.

```
module scoreboard
Input: score
Calculate tens digit = score // 10
Calculate units_digit = score % 10
Set scoreboard center at (500, 50)
For pixels within the first half of the scoreboard width (tens digit area):
   If pixel within determined tens digit:
      Assign color 8'b00011100
For pixels within the second half of the scoreboard width (units digit area):
   If pixel within determined units digit:
```

Pseudocode 13: scoreboard display

As explained in the previous parts to check the collision between the enemy and the projectile, we have divided the screen into 16 angle; according to the difference between rocket angle and enemy angle, the shoot mechanism decides if the enemy is shooted by which type. To illustrate, if shooting is typeC, the enemy is damaged 1 health point as long as the angle difference between rocket and enemy is equal or smaller than 2(out of 16 angles). More detailed explanation is given in the shoot mechanism subtitle. To calculate the remaining health of the enemies, we used output of the shoot mechanism module, which are typeA, typeB, typeC as input for the health calculation in the dynamic top module. TypeA, typeB, typeC describe which enemies are shooted by which shooting type. At every clock pulse remaining health point is calculated by subtracting the shoot-on-target times potential damage of the shooting type from the previous health point for each enemy. In our gameplay, the alive enemy could have a health point between 1-5; if enemies have other health points (6,7,0), those are accepted as dead and are not displayed on the screen. (worst case 1-2 = -1 =>7) When the spawn signal is 1 one of the dead enemies is born with full health. If the enemy reaches the rocket, the game-over signal becomes 1. We checked that condition by controlling the location of the enemy; for each angle, we divided the movement from the perimeter to the center into 32 locations. After testing the enemy's movement, we realized on the VGA screen the enemy had collided with the rocket at the 5th location. (0th location is the center, 32nd location is the perimeter) Therefore, the game-over signal will be 1 when any alive enemies reach the 5th location. A pseudo code as pseudocode14 and 15 which I explained the content in the previous paragraph, is provided only for enemy1. In actual code, we use the following code for 8 different enemies. For the game over screen, the 1-bit game over information is processed in the **game_over_display** module. This module takes the gameover input. The pixel positions corresponding to the letters "G", "A", "M", "E", " ", "O", "V", "E", "R" are pre-defined. If the gameover information is 1 and the pixel is within these letters, the color is assigned as 8'b00011100; otherwise, it is assigned 8'b00000000.An easy explanation for game over display is given as pseudocode 16 below

```
if alien_1_can is 0 or 6 or 7 then
  set alien_1_dead_alive to 0
```

Pseudocode 14: dead_alive condition

```
if (alien_1_location is 5) and
(alien_1_dead_alive is 1)
set game_over to 1
```

Pseudocode 15: game over condition

```
module game over display
Input: gameover
Pre-define pixel positions for "GAME OVER"
If gameover == 1:
   For pixels within "GAME OVER" letters:
      Assign color 8'b00011100
   Assign color 8'b00000000 to all other pixels
Output: assigned color
```

Pseudocode16: game over display

Coding Approach

The code of the project consists of two main part: Display and game-mechanism. All game mechanism modules (shoot_mechanism, enemy_spawn_signal_generator, rocket_driver, enemy_spawn_location_generator) are integrated into the dynamic_top module and also inside the dynamic_top module same additional parts which do not belong to the separate module(reset, game_over, health reducement, score increment, the enemy movement toward to center, the enemy being born) are added. Display part consists of location, vga_driver, seven_segment_display, scoreboard, color_hiearchy, enemy_matrix, enemy_movement, yildiz_kumlu, game_over_display modules.

Display and game mechanism parts are integrated in the “top” module. We can briefly call dynamic_top as backend and other vga modules as front-end.

V. CHALLENGES

This section addresses the various challenges encountered during the development of the game. Each challenge is discussed in detail, along with the strategies and solutions implemented to overcome them. The spaceship's aiming system presented one of our biggest challenges when writing our

Verilog code. To be more precise, the spacecraft missed targets at positions -1 and -2 while it was at position 0, and it missed targets at positions 1 and 2 when it was at position -1. The if blocks controlling the spaceship's aiming logic were the source of this problem since they failed to consider these edge circumstances. To address this, we added new if blocks to handle these situations, ensuring that the spaceship could precisely target and strike these spots, no matter where it started. This fixed the aiming problem and enhanced gameplay by adding criteria to look for and react to these spots. At the beginning of the project enemy movement has just 8 locations towards the center. After experiencing the gameplay we realize that this leads to non-smooth gameplay. Then we decided to increase the number of location(from perimeter towards center) from 8 to 32 location. We also had a challenge with the game's layout. We encountered an unforeseen problem where the enemies accelerated, and the game exhibited a notable lag when we merged the game over screen into the final code. We tried to solve this, but we decided to completely rewrite some parts of the our main code to fix the issue. At the end, this method corrected the problem and returned the desired enemy speed and game performance. The FPGA sharing issue was another notable challenge that most teams probably faced. When we tried to use the FPGA, it was often delayed and disrupted by our work schedules colliding with another group's. We had more difficulty testing and debugging our code as a result, which made this a major technical problem. In order to address this, we had to collaborate closely with the other group and modify our schedules to provide for equal access to the FPGA, which called for additional organization and flexibility.

VI. RESULTS

This section discusses the outcomes of the project, evaluating how effectively the implementation meets the defined objectives and requirements. It includes an assessment of game functionality, performance metrics, and visual output quality. Additionally, this section provides a comparison of the final product against the initial goals and it highlights the strong and weak parts of the final implementation. Overall the projects meets the almost all the objectives and requirements. We tried animated projectile motion. Our code made a significant progression but it is not completed because of the lack of time and the code did not work .At the project demonstration the assistant realized that even if player kill 2 enemies at the same time scoreboard was increased by 1. After we examined the code we realized our mistake and added new if block. The problem was when the first if block is realized code does not goes into other else if block. We change else if blocks with if blocks therefore problem was solved. Except those 2 mistake overall project works just fine .

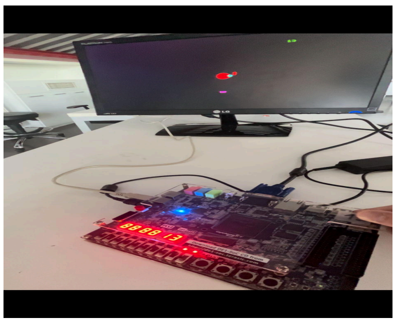

Figure 2: Project Image

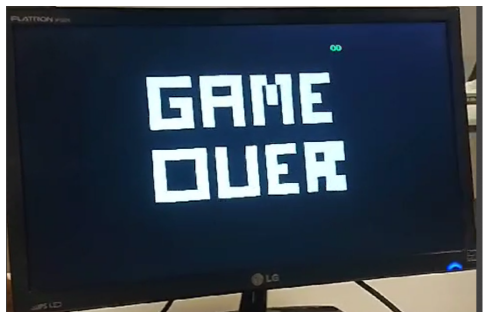

Figure 3: Game Over Screen

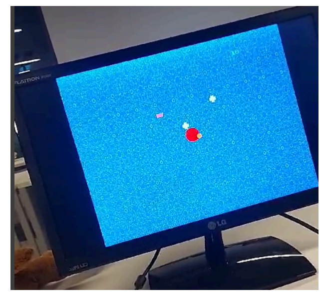

Figure 5: Project Image

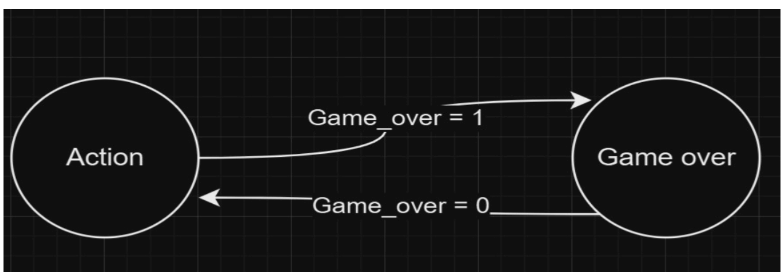

Figure 4: State Diagram

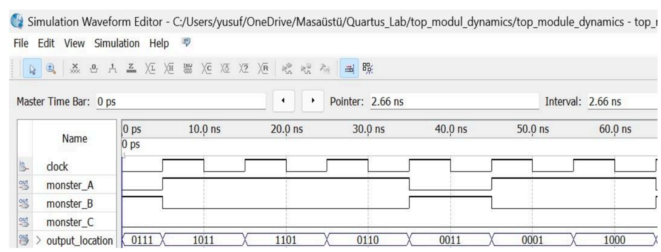

Figure 6: Waveform

Overall we have different enemy types, levels, shooting modes and a rocket. Rocket rotates according to pressed buttons of fpga and enemies are moving towards the center of the screen. Different health levels and shooting modes are visible from figure 4 and 3 .For different shooting modes rocket changes its color. Additionally health values are represented with different enemy colors as explained before. As we can observe from figure 4 both type and location signals randomly generated .Other blocks that have been used for design are too complicated to simulate or obvious. For instance for dynamic top module there are more than 20 output signals. It is very complicated to observe the waveform. Instead direct fpga implementation preferred. Game over condition is another perfectly working mechanism . Dynamics_top module successfully generates game over signal when one of the aliens collides with the spaceship. Vga unit prints this signal like figure 3 . State diagram of this process has shown in figure 4 . At the backend all necessary shooting mechanism and score logic almost function smoothly . When an dead_alive signal is 0 enemy immediately disappears from game field. This change almost flawlessly adds scoreboard a 1 extra point. We also designed a background in order to enhance player experience. We influenced by “nazar boncuğu “ which is one of the key aspects of our culture.This background can be closed. We also used 7 segment display to represent score additionally and used leds to give shooting mode information as feedback to player.

VII. CONCLUSION

In this project, the Video Graphics Array (VGA) is the interface for rendering the game field and various visual elements on the display. We learned about the VGA protocol, signal production, and responsive display through research. First, we worked on creating the fundamental VGA signals (HSYNC, VSYNC, and SYNC) and figuring out when they needed to be generated. By studying documentation, online forums, and example codes, particularly the VGA driver page by V. Hunter Adams, we developed our understanding. This procedure was essential for maintaining display stability and guaranteeing accurate game graphic rendering. The intersection of the project and VGA required integrating various visual modules, such as the spaceship, enemies, scoreboard, game over screen, and background image, into the display system. Using a 25MHz clock, each module evaluated the pixel coordinates to determine the appropriate color assignment according to the current game state. The color_hierarchy module was essential in prioritizing the display of different game elements, ensuring that all components were properly visualized on the screen. The system architecture was described in detail in the section on the solution approach. By carefully designing and carrying out each submodule, we ensured that the game dynamics were accurately represented on the VGA display. This approach enabled us to create a game interface, demonstrating the effectiveness of our VGA driver module and the integration of various game components. Overall, this project taught us the VGA protocol and its application in game rendering. The challenges we encountered and the solutions we developed contributed to a deeper understanding of signal generation, timing synchronization, and visual module integration, which were crucial for any display-based project. In this project, we learned how to use Quartus and Verilog effectively. We made a responsibility list for each group member. This improved teamwork skills. We solved our code problem together. The most educated part of this project was the VGA part. We learned how to visualize our codes on Verilog. This excited us because we could see our code results on the screen simultaneously with the help of FPGA. Debugging became more suitable for us. If we had more time and resources, we would design the movement of the spaceship, allowing it to move in all directions. This would make the game more dynamic and give the impression of flying through space. Additionally, we would implement a smoothly scrolling game environment to enhance the gameplay experience.

VIII. REFERENCES

Adams, V. H. (n.d.). *VGA driver*. V. Hunter Adams. Retrieved from https://vanhunteradams.com/DE1/VGA_Driver/Driver.html#The-VGA-Standard

Department of Mathematics, University of Texas at Austin. (n.d.). *8-bit RGB color palette*.

Various Authors. (n.d.). *DE1-SoC user manual*. ODTUClass. (pp. 34-35).

Various Authors. (n.d.). *Stack Overflow*. Retrieved from https://stackoverflow.com

Various Authors. (n.d.). *Intel's community forum*. Retrieved from https://community.intel.com

IX. APPENDIX / APPENDICES

```
//Module 1
module angle(clk,btn_right,btn_left,angle);
input wire clk;
input wire btn_right;
input wire btn_left;
reg [3:0] angle1 ;
reg [3:0] angle2;
output reg [3:0] angle;
//buton active low dedi kumlu ondan negedge değilse posedge olmalı
always @(negedge btn_right) begin
angle1 <= angle1 + 1;
end
always @(negedge btn_left) begin
angle2 <= angle2 + 1;
end
always @(*) begin
angle<= (angle1-angle2);
end
endmodule

//Module 2
module ClockDivider(
input wire fpga_clock,
input wire [1:0] level,
output reg game_clock
);
reg [25:0] counter = 0;
reg [25:0] DIVISOR;
always @(*) begin
case(level)
2'b00: DIVISOR = 50000000 / 1;
2'b01: DIVISOR = 50000000 / 2;
2'b10: DIVISOR = 50000000 / 4;
2'b11: DIVISOR = 50000000 / 8;
default: DIVISOR=50000000 / 2;
endcase
end
always @(posedge fpga_clock) begin
if (counter < (DIVISOR / 2) - 1) begin
```

```
counter <= counter + 1;
end else begin
counter <= 0;
game_clock <= ~game_clock;
end
end
endmodule

//Module 3
module
color_hierarchy(clk,color_spaceship,color_scoreboard,color_enemy,color_gameover,color_yildiz,color_main,gameover,reset,background_s
witch);
input clk;
input wire background_switch;
input wire [7:0] color_spaceship;
input wire [7:0] color_scoreboard;
input wire [7:0] color_enemy;
input wire [7:0] color_gameover;
input gameover;
input reset;
output reg [7:0] color_main;
input wire [7:0] color_yildiz;
always @(clk) begin
if (reset) begin
color_main = 8'b01000111;
end
else begin
if (gameover) begin //gameover
if (color_scoreboard != 8'b00000000) begin
color_main = color_scoreboard;
end
else begin
color_main = color_gameover;
end
end
else begin
if(color_spaceship != 8'b00000000) begin
color_main = color_spaceship;
end
else if(color_scoreboard != 8'b00000000) begin
color_main = color_scoreboard;
end
else if(color_enemy != 8'b00000000) begin
color_main = color_enemy;
end
else if (color_yildiz != 8'b00000000) begin
if (background_switch == 0) begin
color_main = color_yildiz;
end
else begin
color_main = 8'b00000000;
end
end
else begin
color_main = 8'b00000000;
end
end
end
end
```

```
endmodule

//Module 4
module dynamics_top (
input clock_fpga,
input [1:0] shoot_type,
input shoot,
input shift_plus,
input shift_minus,
input reset,
input [3:0] angle,
input [1:0] level,
output [7:0] alien_alive_dead,
output [4:0] alien_1_quantization,
output [4:0] alien_2_quantization,
output [4:0] alien_3_quantization,
output [4:0] alien_4_quantization,
output [4:0] alien_5_quantization,
output [4:0] alien_6_quantization,
output [4:0] alien_7_quantization,
output [4:0] alien_8_quantization,
output [3:0] alien_1_angle_output,
output [3:0] alien_2_angle_output,
output [3:0] alien_3_angle_output,
output [3:0] alien_4_angle_output,
output [3:0] alien_5_angle_output,
output [3:0] alien_6_angle_output,
output [3:0] alien_7_angle_output,
output [3:0] alien_8_angle_output,
output [2:0] alien_1_type_output,
output [2:0] alien_2_type_output,
output [2:0] alien_3_type_output,
output [2:0] alien_4_type_output,
output [2:0] alien_5_type_output,
output [2:0] alien_6_type_output,
output [2:0] alien_7_type_output,
output [2:0] alien_8_type_output,
//output [3:0] rocket_angle,
output reg game_over,
output reg [6:0] score,
output [2:0] alien_1_can_out,
output [2:0] alien_2_can_out,
output [2:0] alien_3_can_out,
output [2:0] alien_4_can_out,
output [2:0] alien_5_can_out,
output [2:0] alien_6_can_out,
output [2:0] alien_7_can_out,
output [2:0] alien_8_can_out,
output [2:0] sample_total
);
assign sample_total = total_alivenumber;
assign alien_alive_dead =
{alien_1_dead_alive,alien_2_dead_alive,alien_3_dead_alive,alien_4_dead_alive,alien_5_dead_alive,alien_6_dead_alive,alien_7_dead_alive,a
lien_8_dead_alive,};
assign alien_1_quantization = alien_1_location;
assign alien_2_quantization = alien_2_location;
assign alien_3_quantization = alien_3_location;
assign alien_4_quantization = alien_4_location;
assign alien_5_quantization = alien_5_location;
```

```
assign alien_6_quantization = alien_6_location;
assign alien_7_quantization = alien_7_location;
assign alien_8_quantization = alien_8_location;
assign alien_1_angle_output = alien_1_angle ;
assign alien_2_angle_output = alien_2_angle ;
assign alien_3_angle_output = alien_3_angle ;
assign alien_4_angle_output = alien_4_angle ;
assign alien_5_angle_output = alien_5_angle ;
assign alien_6_angle_output = alien_6_angle ;
assign alien_7_angle_output = alien_7_angle ;
assign alien_8_angle_output = alien_8_angle ;
assign alien_1_type_output = alien_1_type;
assign alien_2_type_output = alien_2_type;
assign alien_3_type_output = alien_3_type;
assign alien_4_type_output = alien_4_type;
assign alien_5_type_output = alien_5_type;
assign alien_6_type_output = alien_6_type;
assign alien_7_type_output = alien_7_type;
assign alien_8_type_output = alien_8_type;
assign rocket_angle = rocket_angle;
assign alien_1_can_out = alien_1_can;
assign alien_2_can_out = alien_2_can;
assign alien_3_can_out = alien_3_can;
assign alien_4_can_out = alien_4_can;
assign alien_5_can_out = alien_5_can;
assign alien_6_can_out = alien_6_can;
assign alien_7_can_out = alien_7_can;
assign alien_8_can_out = alien_8_can;
reg [5:0] total_number;
reg [3:0] alien_1_angle;
reg [4:0] alien_1_location;
reg alien_1_dead_alive;
reg [2:0] alien_1_can;
reg [2:0] alien_1_type;
reg [3:0] alien_2_angle;
reg [4:0] alien_2_location;
reg alien_2_dead_alive;
reg [2:0] alien_2_can;
reg [2:0] alien_2_type;
reg [3:0] alien_3_angle;
reg [4:0] alien_3_location;
reg alien_3_dead_alive;
reg [2:0] alien_3_can;
reg [2:0] alien_3_type;
reg [3:0] alien_4_angle;
reg [4:0] alien_4_location;
reg alien_4_dead_alive;
reg [2:0] alien_4_can;
reg [2:0] alien_4_type;
reg [3:0] alien_5_angle;
reg [4:0] alien_5_location;
reg alien_5_dead_alive;
reg [2:0] alien_5_can;
reg [2:0] alien_5_type;
reg [3:0] alien_6_angle;
reg [4:0] alien_6_location;
reg alien_6_dead_alive;
reg [2:0] alien_6_can;
reg [2:0] alien_6_type;
```

```
reg [3:0] alien_7_angle;
reg [4:0] alien_7_location;
reg alien_7_dead_alive;
reg [2:0] alien_7_can;
reg [2:0] alien_7_type;
reg [3:0] alien_8_angle;
reg [4:0] alien_8_location;
reg alien_8_dead_alive;
reg [2:0] alien_8_can;
reg [2:0] alien_8_type;
reg [2:0] shift_reg_previous_1;
reg [2:0] shift_reg_previous_2;
reg [2:0] shift_reg_previous_3;
reg [2:0] shift_reg_previous_4;
reg [2:0] shift_reg_previous_5;
reg [2:0] shift_reg_previous_6;
reg [2:0] shift_reg_previous_7;
reg [2:0] shift_reg_previous_8;
wire spawn_signal;
wire game_clock;
wire [3:0] wire_angle_rocket;
wire [3:0] random_angle;
wire [2:0] monster_type;
wire [7:0] shooted_A;
wire [7:0] shooted_B;
wire [7:0] shooted_C;
reg [31:0] counter_limit;
reg [31:0] internal_counter;
reg [32:0] finito_cokare;
initial begin
finito_cokare = 0;
score = 0;
game_over = 0;
alien_1_dead_alive = 0;
alien_2_dead_alive = 0;
alien_3_dead_alive = 0;
alien_4_dead_alive = 0;
alien_5_dead_alive = 0;
alien_6_dead_alive = 0;
alien_7_dead_alive = 0;
alien_8_dead_alive = 0;
internal_counter = 0;
end
ClockDivider clock_instantiation(.fpga_clock(clock_fpga),.level(level),.game_clock(game_clock));
shoot_mechanism
shoot_mechanism_instantiation(.clk(clock_fpga),.shoot(shoot),.shoot_type(shoot_type),.angle_rocket(angle),.angle_enemy1(alien_1_angle),.
angle_enemy2(alien_2_angle),.angle_enemy3(alien_3_angle),.angle_enemy4(alien_4_angle),.angle_enemy5(alien_5_angle),.angle_enemy6(al
ien_6_angle),.angle_enemy7(alien_7_angle),.angle_enemy8(alien_8_angle),.typeA(shooted_A),.typeB(shooted_B),.typeC(shooted_C),.alivene
ss_of_enemy1(alien_1_dead_alive),.aliveness_of_enemy2(alien_2_dead_alive),.aliveness_of_enemy3(alien_3_dead_alive),.aliveness_of_enem
y4(alien_4_dead_alive),.aliveness_of_enemy5(alien_5_dead_alive),.aliveness_of_enemy6(alien_6_dead_alive),.aliveness_of_enemy7(alien_7_
dead_alive),.aliveness_of_enemy8(alien_8_dead_alive));
enemy_spawn_signal_generator signal_gen_inst(.clk_fpga(clock_fpga),.spawn(spawn));
//rocket_driver
rocket_inst(.reset(reset),.shift_minus(shift_minus),.shift_plus(shift_plus),.clock_fpga(clock_fpga),.rocket_angle(wire_angle_rocket));
enemy_spawn_location_generator
erg_inst(.clock(clock_fpga),.output_location(random_angle),.monster_A(monster_type[2]),.monster_B(monster_type[1]),.monster_C(monst
er_type[0]));
reg [3:0] total_alivenumber;
always@(*)begin
```

```
total_alivenumber = alien_1_dead_alive
+alien_2_dead_alive+alien_3_dead_alive+alien_4_dead_alive+alien_5_dead_alive+alien_6_dead_alive+alien_7_dead_alive
+alien_8_dead_alive;
end
always @(posedge(clock_fpga) ) begin
if (finito_cokare < 100000000) begin
finito_cokare = finito_cokare + 1;
end
else begin
if (reset == 1) begin
score <= 0;
game_over <= 0;
alien_1_dead_alive <= 0;
alien_2_dead_alive <= 0;
alien_3_dead_alive <= 0;
alien_4_dead_alive <= 0;
alien_5_dead_alive <= 0;
alien_6_dead_alive <= 0;
alien_7_dead_alive <= 0;
alien_8_dead_alive <= 0;
internal_counter <= 0;
end
else begin
if (internal_counter == counter_limit) begin
if (alien_1_location>5) begin
alien_1_location = alien_1_location -1;
end
if (alien_2_location>5) begin
alien_2_location = alien_2_location -1;
end
if (alien_3_location>5) begin
alien_3_location = alien_3_location -1;
end
if (alien_4_location>5) begin
alien_4_location = alien_4_location -1;
end
if (alien_5_location>5) begin
alien_5_location = alien_5_location -1;
end
if (alien_6_location>5) begin
alien_6_location = alien_6_location -1;
end
if (alien_7_location>5) begin
alien_7_location = alien_7_location -1;
end
if (alien_8_location>5) begin
alien_8_location = alien_8_location -1;
end
internal_counter <= 0;
end
else begin
internal_counter <= internal_counter + 1;
end
if (((spawn==1) && (total_alivenumber != 6'd8)) || (total_alivenumber < 6'd2)) begin
if (alien_1_dead_alive == 0) begin
case(monster_type)
3'b100:begin
alien_1_dead_alive=1;
alien_1_can=2;
```

```
alien_1_angle=random_angle;
alien_1_type=3'b100;
alien_1_location = 31;
end
3'b010:begin
alien_1_dead_alive=1;
alien_1_can=3;
alien_1_angle=random_angle;
alien_1_type=3'b010;
alien_1_location = 31;
end
3'b001:begin
alien_1_dead_alive=1;
alien_1_can=5;
alien_1_angle=random_angle;
alien_1_type=3'b001;
alien_1_location = 31;
end
endcase
end
else if (alien_2_dead_alive==0) begin
case(monster_type)
3'b100:begin
alien_2_dead_alive=1;
alien_2_can=2;
alien_2_angle=random_angle;
alien_2_type=3'b100;
alien_2_location = 31;
end
3'b010:begin
alien_2_dead_alive=1;
alien_2_can=3;
alien_2_angle=random_angle;
alien_2_type=3'b010;
alien_2_location = 31;
end
3'b001:begin
alien_2_dead_alive=1;
alien_2_can=5;
alien_2_angle=random_angle;
alien_2_type=3'b001;
alien_2_location = 31;
end
endcase
end
else if (alien_3_dead_alive==0) begin
case(monster_type)
3'b100:begin
alien_3_dead_alive=1;
alien_3_can=2;
alien_3_angle=random_angle;
alien_3_type=3'b100;
alien_3_location = 31;
end
3'b010:begin
alien_3_dead_alive=1;
alien_3_can=3;
alien_3_angle=random_angle;
alien_3_type=3'b010;
```

```
alien_3_location = 31;
end
3'b001:begin
alien_3_dead_alive=1;
alien_3_can=5;
alien_3_angle=random_angle;
alien_3_type=3'b001;
alien_3_location = 31;
end
endcase
end
else if (alien_4_dead_alive==0) begin
case(monster_type)
3'b100:begin
alien_4_dead_alive=1;
alien_4_can=2;
alien_4_angle=random_angle;
alien_4_type=3'b100;
alien_4_location = 31;
end
3'b010:begin
alien_4_dead_alive=1;
alien_4_can=3;
alien_4_angle=random_angle;
alien_4_type=3'b010;
alien_4_location = 31;
end
3'b001:begin
alien_4_dead_alive=1;
alien_4_can=5;
alien_4_angle=random_angle;
alien_4_type=3'b001;
alien_4_location = 31;
end
endcase
end
else if (alien_5_dead_alive==0) begin
case(monster_type)
3'b100:begin
alien_5_dead_alive=1;
alien_5_can=2;
alien_5_angle=random_angle;
alien_5_type=3'b100;
alien_5_location = 31;
end
3'b010:begin
alien_5_dead_alive=1;
alien_5_can=3;
alien_5_angle=random_angle;
alien_5_type=3'b010;
alien_5_location = 31;
end
3'b001:begin
alien_5_dead_alive=1;
alien_5_can=5;
alien_5_angle=random_angle;
alien_5_type=3'b001;
alien_5_location = 31;
end
```

```
endcase
end
else if (alien_6_dead_alive==0) begin
case(monster_type)
3'b100:begin
alien_6_dead_alive=1;
alien_6_can=2;
alien_6_angle=random_angle;
alien_6_type=3'b100;
alien_6_location = 31;
end
3'b010:begin
alien_6_dead_alive=1;
alien_6_can=3;
alien_6_angle=random_angle;
alien_6_type=3'b010;
alien_6_location = 31;
end
3'b001:begin
alien_6_dead_alive=1;
alien_6_can=5;
alien_6_angle=random_angle;
alien_6_type=3'b001;
alien_6_location = 31;
end
endcase
end
else if (alien_7_dead_alive==0) begin
case(monster_type)
3'b100:begin
alien_7_dead_alive=1;
alien_7_can=2;
alien_7_angle=random_angle;
alien_7_type=3'b100;
alien_7_location = 31;
end
3'b010:begin
alien_7_dead_alive=1;
alien_7_can=3;
alien_7_angle=random_angle;
alien_7_type=3'b010;
alien_7_location = 31;
end
3'b001:begin
alien_7_dead_alive=1;
alien_7_can=5;
alien_7_angle=random_angle;
alien_7_type=3'b001;
alien_7_location = 31;
end
endcase
end
else if (alien_8_dead_alive==0) begin
case(monster_type)
3'b100:begin
alien_8_dead_alive=1;
alien_8_can=2;
alien_8_angle=random_angle;
alien_8_type=3'b100;
```

```
alien_8_location = 31;
end
3'b010:begin
alien_8_dead_alive=1;
alien_8_can=3;
alien_8_angle=random_angle;
alien_8_type=3'b010;
alien_8_location = 31;
end
3'b001:begin
alien_8_dead_alive=1;
alien_8_can=5;
alien_8_angle=random_angle;
alien_8_type=3'b001;
alien_8_location = 31;
end
endcase
end
end
if ((alien_1_location == 5) && (alien_1_dead_alive == 1)) begin
game_over <= 1;
end
if ((alien_2_location == 5 )&& (alien_2_dead_alive == 1)) begin
game_over <= 1;
end
if ((alien_3_location == 5) && (alien_3_dead_alive == 1)) begin
game_over <= 1;
end
if ((alien_4_location == 5 )&& (alien_4_dead_alive == 1)) begin
game_over <= 1;
end
if ((alien_5_location == 5 )&&(alien_5_dead_alive == 1)) begin
game_over <= 1;
end
if ((alien_6_location == 5) &&(alien_6_dead_alive == 1)) begin
game_over <= 1;
end
if ((alien_7_location == 5 )&&(alien_7_dead_alive == 1)) begin
game_over <= 1;
end
if ((alien_8_location == 5) &&( alien_8_dead_alive == 1)) begin
game_over <= 1;
end
alien_1_can = alien_1_can - 3*shooted_A[0] - 2*shooted_B [0] - shooted_C [0];
alien_2_can = alien_2_can - 3*shooted_A[1] - 2*shooted_B [1] - shooted_C [1];
alien_3_can = alien_3_can - 3*shooted_A[2] - 2*shooted_B [2] - shooted_C [2];
alien_4_can = alien_4_can - 3*shooted_A[3] - 2*shooted_B [3] - shooted_C [3];
alien_5_can = alien_5_can - 3*shooted_A[4] - 2*shooted_B [4] - shooted_C [4];
alien_6_can = alien_6_can - 3*shooted_A[5] - 2*shooted_B [5] - shooted_C [5];
alien_7_can = alien_7_can - 3*shooted_A[6] - 2*shooted_B [6] - shooted_C [6];
alien_8_can = alien_8_can - 3*shooted_A[7] - 2*shooted_B [7] - shooted_C [7];
shift_reg_previous_1 <= alien_1_can;
shift_reg_previous_2 <= alien_2_can;
shift_reg_previous_3 <= alien_3_can;
shift_reg_previous_4 <= alien_4_can;
shift_reg_previous_5 <= alien_5_can;
shift_reg_previous_6 <= alien_6_can;
shift_reg_previous_7 <= alien_7_can;
shift_reg_previous_8 <= alien_8_can;
```

```
if ((shift_reg_previous_1 > 0) && (shift_reg_previous_1 < 6 ) && (alien_1_can == 0 || alien_1_can == 7 || alien_1_can == 6 )) begin
score <= score + 1;
end
if (shift_reg_previous_2 > 0 && shift_reg_previous_2 < 6  && (alien_2_can == 0 || alien_2_can == 7 || alien_2_can == 6 )) begin
score <= score + 1;
end
if (shift_reg_previous_3 > 0 && shift_reg_previous_3 < 6  && (alien_3_can == 0 || alien_3_can == 7 || alien_3_can == 6 )) begin
score<= score + 1;
end
if (shift_reg_previous_4 > 0 && shift_reg_previous_4 < 6  && (alien_4_can == 0 || alien_4_can == 7 || alien_4_can == 6 )) begin
score <= score + 1;
end
if (shift_reg_previous_5 > 0 && shift_reg_previous_5 < 6  && (alien_5_can == 0 || alien_5_can == 7 || alien_5_can == 6 )) begin
score<= score + 1;
end
if (shift_reg_previous_6 > 0 && shift_reg_previous_6 < 6  && (alien_6_can == 0 || alien_6_can == 7 || alien_6_can == 6 )) begin
score <= score + 1;
end
if (shift_reg_previous_7 > 0 && shift_reg_previous_7 < 6  && (alien_7_can == 0 || alien_7_can == 7 || alien_7_can == 6 )) begin
score <= score + 1;
end
if (shift_reg_previous_8 > 0 && shift_reg_previous_8 < 6  && (alien_8_can == 0 || alien_8_can == 7 || alien_8_can == 6 )) begin
score <= score + 1;
end
if ((alien_1_can == 0) || (alien_1_can == 7) || (alien_1_can == 6)) begin
alien_1_dead_alive <= 0;
end
if ((alien_2_can == 0) || (alien_2_can == 7) || (alien_2_can == 6)) begin
alien_2_dead_alive <= 0;
end
if ((alien_3_can == 0) || (alien_3_can == 7) || (alien_3_can == 6)) begin
alien_3_dead_alive<= 0;
end
if ((alien_4_can == 0) || (alien_4_can == 7) || (alien_4_can == 6)) begin
alien_4_dead_alive <= 0;
end
if ((alien_5_can == 0) || (alien_5_can == 7) || (alien_5_can == 6)) begin
alien_5_dead_alive <= 0;
end
if ((alien_6_can == 0) || (alien_6_can == 7) || (alien_6_can == 6)) begin
alien_6_dead_alive<= 0;
end
if ((alien_7_can == 0) || (alien_7_can == 7) || (alien_7_can == 6)) begin
alien_7_dead_alive <= 0;
end
if ((alien_8_can == 0) || (alien_8_can == 7) || (alien_8_can == 6)) begin
alien_8_dead_alive <= 0;
end
end
end
end
end
always@(*) begin
if (reset == 1) begin
case(level)
'b00: counter_limit = 25000000;
'b01: counter_limit = 25000000/2;
'b10: counter_limit = 25000000/4;
'b11: counter_limit = 25000000/8;
endcase
```

```
end
end
endmodule
//Module 5
module enemy_movement(
input clk,
input [15:0] enemy_matrix1,
input [15:0] enemy_matrix2,
input [15:0] enemy_matrix3,
input [15:0] enemy_matrix4,
input [15:0] enemy_matrix5,
input [15:0] enemy_matrix6,
input [15:0] enemy_matrix7,
input [15:0] enemy_matrix8,
input [9:0] next_x,
input [9:0] next_y,
output reg [7:0] color
);
wire [9:0] x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6, x7, y7, x8, y8;
reg [2:0] type;
reg alive_dead;
reg [4:0] location_;
reg [3:0] angle;
reg [2:0] health;
reg [7:0] enemy_color;
location loc1(enemy_matrix1[12:9], enemy_matrix1[8:4],x1, y1);
location loc2(enemy_matrix2[12:9], enemy_matrix2[8:4],x2, y2);
location loc3(enemy_matrix3[12:9], enemy_matrix3[8:4],x3, y3);
location loc4(enemy_matrix4[12:9], enemy_matrix4[8:4],x4, y4);
location loc5(enemy_matrix5[12:9], enemy_matrix5[8:4],x5, y5);
location loc6(enemy_matrix6[12:9], enemy_matrix6[8:4],x6, y6);
location loc7(enemy_matrix7[12:9], enemy_matrix7[8:4],x7, y7);
location loc8(enemy_matrix8[12:9], enemy_matrix8[8:4],x8, y8);
always @(posedge clk) begin
color = 8'b00000000;
type = enemy_matrix1[2:0];
alive_dead = enemy_matrix1[3];
health = enemy_matrix1[15:13];
if (alive_dead) begin
case (health)
5: enemy_color = 8'b11111111;
4: enemy_color = 8'b00000011;
3: enemy_color = 8'b00011100;
2: enemy_color = 8'b11100011;
1: enemy_color = 8'b11100000;
default: enemy_color = 8'b00000000;
endcase
if (type == 3'b100 && (x1 > (next_x - 10)) && (x1 < (next_x + 10)) && (y1 > (next_y - 5)) && (y1 < (next_y + 5))) begin
color = enemy_color;
end else if (type == 3'b010 && (x1 > (next_x - 5)) && (x1 < (next_x + 5)) && (y1 > (next_y - 10)) && (y1 < (next_y + 10))) begin
color = enemy_color;
end else if (type == 3'b001 && (((x1 > (next_x - 5)) && (x1 < (next_x + 5)) && (y1 > (next_y - 10)) && (y1 < (next_y + 10))) || ((x1 > (next_x
- 10)) && (x1 < (next_x + 10)) && (y1 > (next_y - 5)) && (y1 < (next_y + 5))))) begin
color = enemy_color;
end
end
type = enemy_matrix2[2:0];
alive_dead = enemy_matrix2[3];
health = enemy_matrix2[15:13];
```

```
if (alive_dead) begin
case (health)
5: enemy_color = 8'b11111111;
4: enemy_color = 8'b00000011;
3: enemy_color = 8'b00011100;
2: enemy_color = 8'b11100011;
1: enemy_color = 8'b11100000;
default: enemy_color = 8'b00000000;
endcase
if (type == 3'b100 && (x2 > (next_x - 10)) && (x2 < (next_x + 10)) && (y2 > (next_y - 5)) && (y2 < (next_y + 5))) begin
color = enemy_color;
end else if (type == 3'b010 && (x2 > (next_x - 5)) && (x2 < (next_x + 5)) && (y2 > (next_y - 10)) && (y2 < (next_y + 10))) begin
color = enemy_color;
end else if (type == 3'b001 && (((x2 > (next_x - 5)) && (x2 < (next_x + 5)) && (y2 > (next_y - 10)) && (y2 < (next_y + 10))) || ((x2 > (next_x
- 10)) && (x2 < (next_x + 10)) && (y2 > (next_y - 5)) && (y2 < (next_y + 5))))) begin
color = enemy_color;
end
end
type = enemy_matrix3[2:0];
alive_dead = enemy_matrix3[3];
health = enemy_matrix3[15:13];
if (alive_dead) begin
case (health)
5: enemy_color = 8'b11111111;
4: enemy_color = 8'b00000011;
3: enemy_color = 8'b00011100;
2: enemy_color = 8'b11100011;
1: enemy_color = 8'b11100000;
default: enemy_color = 8'b00000000;
endcase
if (type == 3'b100 && (x3 > (next_x - 10)) && (x3 < (next_x + 10)) && (y3 > (next_y - 5)) && (y3 < (next_y + 5))) begin
color = enemy_color;
end else if (type == 3'b010 && (x3 > (next_x - 5)) && (x3 < (next_x + 5)) && (y3 > (next_y - 10)) && (y3 < (next_y + 10))) begin
color = enemy_color;
end else if (type == 3'b001 && (((x3 > (next_x - 5)) && (x3 < (next_x + 5)) && (y3 > (next_y - 10)) && (y3 < (next_y + 10))) || ((x3 > (next_x
- 10)) && (x3 < (next_x + 10)) && (y3 > (next_y - 5)) && (y3 < (next_y + 5))))) begin
color = enemy_color;
end
end
type = enemy_matrix4[2:0];
alive_dead = enemy_matrix4[3];
health = enemy_matrix4[15:13];
if (alive_dead) begin
case (health)
5: enemy_color = 8'b11111111;
4: enemy_color = 8'b00000011;
3: enemy_color = 8'b00011100;
2: enemy_color = 8'b11100011;
1: enemy_color = 8'b11100000;
default: enemy_color = 8'b00000000;
endcase
if (type == 3'b100 && (x4 > (next_x - 10)) && (x4 < (next_x + 10)) && (y4 > (next_y - 5)) && (y4 < (next_y + 5))) begin
color = enemy_color;
end else if (type == 3'b010 && (x4 > (next_x - 5)) && (x4 < (next_x + 5)) && (y4 > (next_y - 10)) && (y4 < (next_y + 10))) begin
color = enemy_color;
end else if (type == 3'b001 && (((x4  > (next_x - 5)) && (x4 < (next_x + 5)) && (y4 > (next_y - 10)) && (y4 < (next_y + 10))) || ((x4 >
(next_x - 10)) && (x4 < (next_x + 10)) && (y4 > (next_y - 5)) && (y4 < (next_y + 5))))) begin
color = enemy_color;
end
```

```
end
type = enemy_matrix5[2:0];
alive_dead = enemy_matrix5[3];
health = enemy_matrix5[15:13];
if (alive_dead) begin
case (health)
5: enemy_color = 8'b11111111;
4: enemy_color = 8'b00000011;
3: enemy_color = 8'b00011100;
2: enemy_color = 8'b11100011;
1: enemy_color = 8'b11100000;
default: enemy_color = 8'b00000000;
endcase
if (type == 3'b100 && (x5 > (next_x - 10)) && (x5 < (next_x + 10)) && (y5 > (next_y - 5)) && (y5 < (next_y + 5))) begin
color = enemy_color;
end else if (type == 3'b010 && (x5 > (next_x - 5)) && (x5 < (next_x + 5)) && (y5 > (next_y - 10)) && (y5 < (next_y + 10))) begin
color = enemy_color;
end else if (type == 3'b001 && (((x5 > (next_x - 5)) && (x5 < (next_x + 5)) && (y5 > (next_y - 10)) && (y5 < (next_y + 10))) || ((x5 > (next_x
- 10)) && (x5 < (next_x + 10)) && (y5 > (next_y - 5)) && (y5 < (next_y + 5))))) begin
color = enemy_color;
end
end
type = enemy_matrix6[2:0];
alive_dead = enemy_matrix6[3];
health = enemy_matrix6[15:13];
if (alive_dead) begin
case (health)
5: enemy_color = 8'b11111111;
4: enemy_color = 8'b00000011;
3: enemy_color = 8'b00011100;
2: enemy_color = 8'b11100011;
1: enemy_color = 8'b11100000;
default: enemy_color = 8'b00000000;
endcase
if (type == 3'b100 && (x6 > (next_x - 10)) && (x6 < (next_x + 10)) && (y6 > (next_y - 5)) && (y6 < (next_y + 5))) begin
color = enemy_color;
end else if (type == 3'b010 && (x6 > (next_x - 5)) && (x6 < (next_x + 5)) && (y6 > (next_y - 10)) && (y6 < (next_y + 10))) begin
color = enemy_color;
end else if (type == 3'b001 && (((x6 > (next_x - 5)) && (x6 < (next_x + 5)) && (y6 > (next_y - 10)) && (y6 < (next_y + 10))) || ((x6 > (next_x
- 10)) && (x6 < (next_x + 10)) && (y6 > (next_y - 5)) && (y6 < (next_y + 5))))) begin
color = enemy_color;
end
end
type = enemy_matrix7[2:0];
alive_dead = enemy_matrix7[3];
health = enemy_matrix7[15:13];
if (alive_dead) begin
case (health)
5: enemy_color = 8'b11111111;
4: enemy_color = 8'b00000011;
3: enemy_color = 8'b00011100;
2: enemy_color = 8'b11100011;
1: enemy_color = 8'b11100000;
default: enemy_color = 8'b00000000;
endcase
if (type == 3'b100 && (x7 > (next_x - 10)) && (x7 < (next_x + 10)) && (y7 > (next_y - 5)) && (y7 < (next_y + 5))) begin
color = enemy_color;
end else if (type == 3'b010 && (x7 > (next_x - 5)) && (x7 < (next_x + 5)) && (y7 > (next_y - 10)) && (y7 < (next_y + 10))) begin
color = enemy_color;
```

```
end else if (type == 3'b001 && (((x7 > (next_x - 5)) && (x7 < (next_x + 5)) && (y7 > (next_y - 10)) && (y7 < (next_y + 10))) || ((x7 > (next_x - 10)) && (x7 < (next_x + 10)) && (y7 > (next_y - 5)) && (y7 < (next_y + 5))))) begin
color = enemy_color;
end
end
type = enemy_matrix8[2:0];
alive_dead = enemy_matrix8[3];
health = enemy_matrix8[15:13];
if (alive_dead) begin
case (health)
5: enemy_color = 8'b11111111;
4: enemy_color = 8'b00000011;
3: enemy_color = 8'b00011100;
2: enemy_color = 8'b11100011;
1: enemy_color = 8'b11100000;
default: enemy_color = 8'b00000000;
endcase
if (type == 3'b100 && (x8 > (next_x - 10)) && (x8 < (next_x + 10)) && (y8 > (next_y - 5)) && (y8 < (next_y + 5))) begin
color = enemy_color;
end else if (type == 3'b010 && (x8 > (next_x - 5)) && (x8 < (next_x + 5)) && (y8 > (next_y - 10)) && (y8 < (next_y + 10))) begin
color = enemy_color;
end else if (type == 3'b001 && (((x8 > (next_x - 5)) && (x8 < (next_x + 5)) && (y8 > (next_y - 10)) && (y8 < (next_y + 10))) || ((x8 > (next_x - 10)) && (x8 < (next_x + 10)) && (y8 > (next_y - 5)) && (y8 < (next_y + 5))))) begin
color = enemy_color;
end
end
end
endmodule

//Module 6
module enemy_spawn_location_generator(
input clock,
output [3:0] output_location,
output reg monster_A,
output reg monster_B,
output reg monster_C
);
reg [15:0] internal_reg = 16'h7891;
reg [1:0] type ;
always@(posedge(clock)) begin
internal_reg <= internal_reg >> 1;
internal_reg[15] <= internal_reg[0] ^ internal_reg[2] ^ internal_reg[3] ^ internal_reg[5];
type <=  internal_reg [1:0];
end
assign output_location = internal_reg [15:12];
always@(*) begin
case (type)
'b00: begin
monster_A <= 1;
monster_B <= 0;
monster_C <= 0;
end
'b01: begin
monster_A <= 1;
monster_B <= 0;
monster_C <= 0;
end
'b10: begin
monster_A <= 0;
```

```
monster_B <= 1;
monster_C <= 0;
end
'b11: begin
monster_A <= 0;
monster_B <= 0;
monster_C <= 1;
end
endcase
end
endmodule

//Module 7
module enemy_spawn_signal_generator (
input clk_fpga,
output reg spawn
);
reg [31:0] counter;
always@(posedge(clk_fpga)) begin
if (counter == 'd100000000) begin
counter <= 'd0;
spawn <= 'd1;
end
else begin
spawn <= 'd0;
counter <= counter + 1;
end
end
endmodule

//Module 8
module game_over_display (
input wire clk,
input wire gameover,
input wire [9:0] next_x,
input wire [9:0] next_y,
output reg [7:0] vga_color
);
always @(posedge clk) begin
if (gameover) begin
if (next_x >= 70 && next_x < 570 && next_y >= 99 && next_y < 400) begin
if (
// G
(next_x >= 120 && next_x < 140 && next_y >= 100 && next_y < 200) ||
(next_x >= 140 && next_x < 200 && next_y >= 100 && next_y < 120) ||
(next_x >= 180 && next_x < 200 && next_y >= 160 && next_y < 200) ||
(next_x >= 160 && next_x < 200 && next_y >= 140 && next_y < 160) ||
(next_x >= 140 && next_x < 200 && next_y >= 180 && next_y < 200) ||
// A
(next_x >= 220 && next_x < 240 && next_y >= 100 && next_y < 200) ||
(next_x >= 240 && next_x < 300 && next_y >= 100 && next_y < 120) ||
(next_x >= 280 && next_x < 300 && next_y >= 100 && next_y < 200) ||
(next_x >= 240 && next_x < 280 && next_y >= 140 && next_y < 160) ||
// M
(next_x >= 320 && next_x < 340 && next_y >= 100 && next_y < 200) ||
(next_x >= 340 && next_x < 350 && next_y >= 100 && next_y < 120) ||
(next_x >= 350 && next_x < 370 && next_y >= 120 && next_y < 160) ||
(next_x >= 370 && next_x < 380 && next_y >= 100 && next_y < 120) ||
(next_x >= 380 && next_x < 400 && next_y >= 100 && next_y < 200) ||
```

```
// E
(next_x >= 420 && next_x < 480 && next_y >= 100 && next_y < 120) ||
(next_x >= 420 && next_x < 440 && next_y >= 100 && next_y < 200) ||
(next_x >= 420 && next_x < 480 && next_y >= 180 && next_y < 200) ||
(next_x >= 420 && next_x < 460 && next_y >= 140 && next_y < 160) ||
// O
(next_x >= 120 && next_x < 140 && next_y >= 240 && next_y < 340) ||
(next_x >= 140 && next_x < 200 && next_y >= 240 && next_y < 260) ||
(next_x >= 200 && next_x < 220 && next_y >= 240 && next_y < 340) ||
(next_x >= 140 && next_x < 200 && next_y >= 320 && next_y < 340) ||
// V
(next_x >= 240 && next_x < 260 && next_y >= 240 && next_y < 340) ||
(next_x >= 300 && next_x < 320 && next_y >= 240 && next_y < 340) ||
(next_x >= 260 && next_x < 300 && next_y >= 320 && next_y < 340) ||
// E
(next_x >= 340 && next_x < 400 && next_y >= 320 && next_y < 340) ||
(next_x >= 340 && next_x < 360 && next_y >= 240 && next_y < 320) ||
(next_x >= 340 && next_x < 400 && next_y >= 240 && next_y < 260) ||
(next_x >= 340 && next_x < 380 && next_y >= 280 && next_y < 300) ||
// R
(next_x >= 420 && next_x < 440 && next_y >= 240 && next_y < 340) ||
(next_x >= 440 && next_x < 500 && next_y >= 240 && next_y < 260) ||
(next_x >= 480 && next_x < 500 && next_y >= 240 && next_y < 280) ||
(next_x >= 460 && next_x < 500 && next_y >= 260 && next_y < 280) ||
(next_x >= 440 && next_x < 480 && next_y >= 280 && next_y < 310) ||
(next_x >= 460 && next_x < 500 && next_y >= 310 && next_y < 340))
begin
vga_color = 8'b11111111; // White
end else begin
vga_color = 8'b00000000; // Black
end
end else begin
vga_color = 8'b00000000; // Black
end
end else begin
vga_color = 8'b00000000; // Black
end
end
endmodule

//Module 9
module location(
input [3:0] angle,
input [4:0] location_input,
output reg [9:0] x,
output reg [9:0] y
);
reg radius = 170;
always @(*) begin
radius = 170;
case (angle)
4'd0: begin
x = 320+((location_input*15'd0)/32);
y = 240-((location_input*15'd170)/32);
end
4'd1: begin
x = 320+((location_input*15'd65)/32);
y = 240-((location_input*15'd157)/32);
end
```

```
4'd2: begin
x = 320+((location_input*15'd120)/32);
y = 240-((location_input*15'd120)/32);
end
4'd3: begin
x = 320+((location_input*15'd157)/32);
y = 240-((location_input*15'd65)/32);
end
4'd4: begin
x = 320+((location_input*15'd170)/32);
y = 240-((location_input*15'd0)/32);
end
4'd5: begin
x = 320+((location_input*15'd157)/32);
y = 240+((location_input*15'd65)/32);
end
4'd6: begin
x = 320+((location_input*15'd120)/32);
y = 240+((location_input*15'd120)/32);
end
4'd7: begin
x = 320+((location_input*15'd65)/32);
y = 240+((location_input*15'd157)/32);
end
4'd8: begin
x = 320+((location_input*15'd0)/32);
y = 240+((location_input*15'd170)/32);
end
4'd9: begin
x = 320-((location_input*15'd65)/32);
y = 240+((location_input*15'd157)/32);
end
4'd10: begin
x = 320-((location_input*15'd120)/32);
y = 240+((location_input*15'd120)/32);
end
4'd11: begin
x = 320-((location_input*15'd157)/32);
y = 240+((location_input*15'd65)/32);
end
4'd12: begin
x = 320-((location_input*15'd170)/32);
y = 240+((location_input*15'd0)/32);
end
4'd13: begin
x = 320-((location_input*15'd157)/32);
y = 240-((location_input*15'd65)/32);
end
4'd14: begin
x = 320-((location_input*15'd120)/32);
y = 240-((location_input*15'd120)/32);
end
4'd15: begin
x = 320-((location_input*15'd65)/32);
y = 240-((location_input*15'd157)/32);
end
endcase
end
endmodule
```

```
//Module 10
module main(fpgaclk,
hsync,
vsync,
red,
green,
blue,
sync,
clk,
blank,
btn_right,
btn_left,
sw_shooting_mode1,
sw_shooting_mode2,
alien1_wire,
alien2_wire,
alien3_wire,
alien4_wire,
alien5_wire,
alien6_wire,
alien7_wire,
alien8_wire,
sample_total,
btn_fire,
level2,
reset_game,
segments1,
segments2,
shooting_led,
background_switch
);
input background_switch;
output [2:0] shooting_led;
output  [6:0] segments1;
output  [6:0] segments2;
input reset_game;
input [1:0] level2;
input btn_fire;
input btn_left;
input btn_right;
input fpgaclk;
input sw_shooting_mode1;
input sw_shooting_mode2;
wire [9:0]  next_x;
wire [9:0]  next_y;
/////////
output  hsync;
output  vsync;
output [7:0] red;
output [7:0] green;
output [7:0] blue;
output sync;
output clk;
output blank;
output [15:0]alien1_wire;
output [15:0]alien2_wire;
output [15:0]alien3_wire;
output [15:0]alien4_wire;
```

```
output [15:0]alien5_wire;
output [15:0]alien6_wire;
output [15:0]alien7_wire;
output [15:0]alien8_wire;
output [2:0] sample_total;
/////////
wire [3:0] current_angle;
wire [7:0] color_spaceship;
wire [7:0] color_yildiz;
wire [7:0] color_scoreboard;
wire [7:0] color_main;
wire [7:0] color_gameover;
wire [7:0] color_enemy;
reg [7:0] score;
//wire [15:0] alien1_wire;
//wire [15:0] alien2_wire;
//wire [15:0] alien3_wire;
//wire [15:0] alien4_wire;
//wire [15:0] alien5_wire;
//wire [15:0] alien6_wire;
//wire [15:0] alien7_wire;
//wire [15:0] alien8_wire;
wire gameover;
//wire [3:0] rocketangle;
wire [6:0] score_number;
reg [3:0] digit1;
reg [3:0] digit2;
////////
reg clk_25mhz;
reg reset;
initial begin
clk_25mhz =0;
reset =0;
score = 8'd12;
end
always @(posedge fpgaclk) begin
clk_25mhz <= !(clk_25mhz);
score <= 8'd12;
digit1 = score_number / 10;
digit2 = score_number % 10;
end
vga_driver u1(clk_25mhz,reset,color_main,next_x,next_y,hsync,vsync,red,green,blue,sync,clk,blank);
spaceship_body
u2(clk_25mhz,next_x,next_y,color_spaceship,current_angle,sw_shooting_mode1,sw_shooting_mode2,shooting_led,reset_game,gameover);
angle u3(clk_25mhz,btn_right,btn_left,current_angle);
//rocket_driver(.reset(1),.shift_minus(btn_left),.shift_plus(btn_right),clock_fpga(fpgaclk));
scoreboard u4(clk_25mhz,next_x,next_y,score_number,color_scoreboard);
color_hierarchy
u5(clk_25mhz,color_spaceship,color_scoreboard,color_enemy,color_gameover,color_yildiz,color_main,gameover,reset_game,background_s
witch);
enemy_movement
u6(clk_25mhz,alien1_wire,alien2_wire,alien3_wire,alien4_wire,alien5_wire,alien6_wire,alien7_wire,alien8_wire,next_x,next_y,color_enemy)
;
dynamics_top
u7(.game_over(gameover),.sample_total(sample_total),.clock_fpga(clk),.shoot_type({sw_shooting_mode1,sw_shooting_mode2}),.shoot(btn_f
ire),.shift_plus(btn_right),.shift_minus(btn_left),.reset(reset_game),.level(level2),.alien_alive_dead({alien1_wire[3],alien2_wire[3],alien3_wir
e[3],alien4_wire[3],alien5_wire[3],alien6_wire[3],alien7_wire[3],alien8_wire[3]}),.alien_1_quantization(alien1_wire[8:4]),.alien_2_quantizati
on(alien2_wire[8:4]),.alien_3_quantization(alien3_wire[8:4]),.alien_4_quantization(alien4_wire[8:4]),.alien_5_quantization(alien5_wire[8:4]
),.alien_6_quantization(alien6_wire[8:4]),.alien_7_quantization(alien7_wire[8:4]),.alien_8_quantization(alien8_wire[8:4]),.alien_1_angle_ou
```

```
tput(alien1_wire[12:9]),.alien_2_angle_output(alien2_wire[12:9]),.alien_3_angle_output(alien3_wire[12:9]),.alien_4_angle_output(alien4_wire[12:9]),.alien_5_angle_output(alien5_wire[12:9]),.alien_6_angle_output(alien6_wire[12:9]),.alien_7_angle_output(alien7_wire[12:9]),.alien_8_angle_output(alien8_wire[12:9]),.alien_1_type_output(alien1_wire[2:0]),.alien_2_type_output(alien2_wire[2:0]),.alien_3_type_output(alien3_wire[2:0]),.alien_4_type_output(alien4_wire[2:0]),.alien_5_type_output(alien5_wire[2:0]),.alien_6_type_output(alien6_wire[2:0]),.alien_7_type_output(alien7_wire[2:0]),.alien_8_type_output(alien8_wire[2:0]),.alien_1_can_out(alien1_wire[15:13]),.alien_2_can_out(alien2_wire[15:13]),.alien_3_can_out(alien3_wire[15:13]),.alien_4_can_out(alien4_wire[15:13]),.alien_5_can_out(alien5_wire[15:13]),.alien_6_can_out(alien6_wire[15:13]),.alien_7_can_out(alien7_wire[15:13]),.alien_8_can_out(alien8_wire[15:13]),.score(score_number),.angle(current_angle)
);
yildiz_kumlu u8(clk_25mhz,next_x,next_y,color_yildiz);
game_over_display u9(clk_25mhz,gameover,next_x,next_y,color_gameover);
seven_segment_display u10(digit1,segments2);
seven_segment_display u11(digit2,segments1);
//   enemy_movement
u12(clk_25mhz,alien1_wire,alien2_wire,alien3_wire,alien4_wire,alien5_wire,alien6_wire,alien7_wire,alien8_wire,next_x,next_y,color_effect)
;
endmodule

//Module 11
module scoreboard (
input clk,
input [9:0] next_x,
input [9:0] next_y,
input [7:0] number,
output reg [7:0] vga_color
);
localparam centerx = 500;
localparam centery = 50;
localparam FONT_WIDTH = 11;
localparam FONT_HEIGHT = 10;
localparam TOTAL_WIDTH = 2 * FONT_WIDTH;
localparam START_X = centerx - (TOTAL_WIDTH) / 2;
localparam START_Y = centery - (FONT_HEIGHT) / 2;
reg [10:0] font [0:9] [0:9];
initial begin
// 0
font[0][0] = 11'b00111111000;
font[0][1] = 11'b01100001100;
font[0][2] = 11'b11000000110;
font[0][3] = 11'b11000000110;
font[0][4] = 11'b11000000110;
font[0][5] = 11'b11000000110;
font[0][6] = 11'b11000000110;
font[0][7] = 11'b11000000110;
font[0][8] = 11'b01100001100;
font[0][9] = 11'b00111111000;
// 1
font[1][0] = 11'b00011100000;
font[1][1] = 11'b00111100000;
font[1][2] = 11'b01111100000;
font[1][3] = 11'b00011100000;
font[1][4] = 11'b00011100000;
font[1][5] = 11'b00011100000;
font[1][6] = 11'b00011100000;
font[1][7] = 11'b00011100000;
font[1][8] = 11'b00011100000;
font[1][9] = 11'b01111111110;
// 2
font[2][0] = 11'b00111111000;
font[2][1] = 11'b01100001100;
```

```
font[2][2] = 11'b11000000110;
font[2][3] = 11'b00000000110;
font[2][4] = 11'b00000001100;
font[2][5] = 11'b00000011000;
font[2][6] = 11'b00000110000;
font[2][7] = 11'b00001100000;
font[2][8] = 11'b00011000000;
font[2][9] = 11'b01111111110;
// 3
font[3][0] = 11'b00111111000;
font[3][1] = 11'b01100001100;
font[3][2] = 11'b11000000110;
font[3][3] = 11'b00000000110;
font[3][4] = 11'b00000111100;
font[3][5] = 11'b00000111100;
font[3][6] = 11'b00000000110;
font[3][7] = 11'b00000000110;
font[3][8] = 11'b01100001100;
font[3][9] = 11'b00111111000;
// 4
font[4][0] = 11'b00000011000;
font[4][1] = 11'b00000111000;
font[4][2] = 11'b00001111000;
font[4][3] = 11'b00011111000;
font[4][4] = 11'b00110111000;
font[4][5] = 11'b01100111000;
font[4][6] = 11'b11111111110;
font[4][7] = 11'b00000111000;
font[4][8] = 11'b00000111000;
font[4][9] = 11'b00000111000;
// 5
font[5][0] = 11'b01111111110;
font[5][1] = 11'b01100000000;
font[5][2] = 11'b01100000000;
font[5][3] = 11'b01111111000;
font[5][4] = 11'b00000001100;
font[5][5] = 11'b00000000110;
font[5][6] = 11'b00000000110;
font[5][7] = 11'b00000000110;
font[5][8] = 11'b01100001100;
font[5][9] = 11'b00111111000;
// 6
font[6][0] = 11'b00111111000;
font[6][1] = 11'b01100001100;
font[6][2] = 11'b11000000110;
font[6][3] = 11'b11000000000;
font[6][4] = 11'b11111111000;
font[6][5] = 11'b11000001100;
font[6][6] = 11'b11000000110;
font[6][7] = 11'b11000000110;
font[6][8] = 11'b01100001100;
font[6][9] = 11'b00111111000;
// 7
font[7][0] = 11'b01111111110;
font[7][1] = 11'b00000001100;
font[7][2] = 11'b00000001100;
font[7][3] = 11'b00000011000;
font[7][4] = 11'b00000011000;
font[7][5] = 11'b00000110000;
```

```
font[7][6] = 11'b00000110000;
font[7][7] = 11'b00001100000;
font[7][8] = 11'b00001100000;
font[7][9] = 11'b00001100000;
// 8
font[8][0] = 11'b00111111000;
font[8][1] = 11'b01100001100;
font[8][2] = 11'b11000000110;
font[8][3] = 11'b11000000110;
font[8][4] = 11'b01100001100;
font[8][5] = 11'b00111111000;
font[8][6] = 11'b01100001100;
font[8][7] = 11'b11000000110;
font[8][8] = 11'b11000000110;
font[8][9] = 11'b01111111100;
// 9
font[9][0] = 11'b00111111000;
font[9][1] = 11'b01100001100;
font[9][2] = 11'b11000000110;
font[9][3] = 11'b11000000110;
font[9][4] = 11'b01111111110;
font[9][5] = 11'b00000000110;
font[9][6] = 11'b00000000110;
font[9][7] = 11'b00000000110;
font[9][8] = 11'b01100001100;
font[9][9] = 11'b00111111000;
end
always @(posedge clk) begin
reg [3:0] digit1;
reg [3:0] digit2;
reg [3:0] digit;
reg [3:0] row;
reg [3:0] col;
digit1 = number / 10;
digit2 = number % 10;
if (next_x >= START_X && next_x < (START_X + TOTAL_WIDTH) &&
next_y >= START_Y && next_y < (START_Y + FONT_HEIGHT)) begin
if (next_x < (START_X + FONT_WIDTH)) begin
digit = digit1;
col = next_x - START_X;
end else begin
digit = digit2;
col = next_x - START_X - FONT_WIDTH;
end
row = next_y - START_Y;
if (font[digit][row][10 - col]) begin
vga_color = 8'b00011100; // beyaz
end else begin
vga_color = 8'b00000000; // siyah
end
end else begin
vga_color = 8'b00000000; // siyah
end
end
endmodule

//Module 12
module seven_segment_display (
input [3:0] digit,
```

```
output reg [6:0] segments
);
always @(*) begin
case (digit)
4'd0: segments <= 7'b1000000;
4'd1: segments <= 7'b1111001;
4'd2: segments <= 7'b0100100;
4'd3: segments <= 7'b0110000;
4'd4: segments <= 7'b0011001;
4'd5: segments <= 7'b0010010;
4'd6: segments <= 7'b0000010;
4'd7: segments <= 7'b1111000;
4'd8: segments <= 7'b0000000;
4'd9: segments <= 7'b0010000;
default: segments <= 7'b1111111;
endcase
end
endmodule

//Module 13
module shoot_mechanism (
input clk,
input shoot,
input [1:0] shoot_type, // 00 doğrusal 3can typea, 01 doğrusal 3can typea, 10 45derece 2can typeb, 11 90derece 1 can typec
input [3:0] angle_rocket,
input [3:0] angle_enemy1, angle_enemy2, angle_enemy3, angle_enemy4, angle_enemy5, angle_enemy6, angle_enemy7, angle_enemy8,
input aliveness_of_enemy1, aliveness_of_enemy2, aliveness_of_enemy3, aliveness_of_enemy4, aliveness_of_enemy5, aliveness_of_enemy6,
aliveness_of_enemy7, aliveness_of_enemy8,
output reg [7:0] typeA, typeB, typeC
);
reg shoot_detected;
initial begin
typeA <= 8'b00000000;
typeB <= 8'b00000000;
typeC <= 8'b00000000;
shoot_detected <= 0;
end
always @(posedge clk) begin
if (shoot == 1) begin
shoot_detected <= 0;
typeA <= 8'b00000000;
typeB <= 8'b00000000;
typeC <= 8'b00000000;
end
else if (shoot == 0 && shoot_detected == 0) begin
shoot_detected <= 1;
if (shoot_type[1] == 0) begin
if (angle_rocket == angle_enemy1 && aliveness_of_enemy1 == 1) typeA[0] <= 1;
if (angle_rocket == angle_enemy2 && aliveness_of_enemy2 == 1) typeA[1] <= 1;
if (angle_rocket == angle_enemy3 && aliveness_of_enemy3 == 1) typeA[2] <= 1;
if (angle_rocket == angle_enemy4 && aliveness_of_enemy4 == 1) typeA[3] <= 1;
if (angle_rocket == angle_enemy5 && aliveness_of_enemy5 == 1) typeA[4] <= 1;
if (angle_rocket == angle_enemy6 && aliveness_of_enemy6 == 1) typeA[5] <= 1;
if (angle_rocket == angle_enemy7 && aliveness_of_enemy7 == 1) typeA[6] <= 1;
if (angle_rocket == angle_enemy8 && aliveness_of_enemy8 == 1) typeA[7] <= 1;
end
if (shoot_type == 2'b10) begin
if ((angle_rocket==4'b1111&&angle_enemy1==4'b0000)&& (aliveness_of_enemy1 == 1)) typeB[0] <= 1;
else if ((angle_rocket==4'b0000&&angle_enemy1==4'b1111)&& (aliveness_of_enemy1 == 1)) typeB[0] <= 1;
```

```
else if ((angle_rocket == angle_enemy1 || angle_rocket - 1 == angle_enemy1 || angle_rocket + 1 == angle_enemy1) && aliveness_of_enemy1
== 1) typeB[0] <= 1;
if ((angle_rocket==4'b1111&&angle_enemy2==4'b0000)&& (aliveness_of_enemy2 == 1) )typeB[1] <= 1;
else if ((angle_rocket==4'b0000&&angle_enemy2==4'b1111)&& (aliveness_of_enemy2 == 1) )typeB[1] <= 1;
else if ((angle_rocket == angle_enemy2 || angle_rocket - 1 == angle_enemy2 || angle_rocket + 1 == angle_enemy2) && aliveness_of_enemy2
== 1) typeB[1] <= 1;
if ((angle_rocket==4'b1111&&angle_enemy3==4'b0000)&& ( aliveness_of_enemy3 == 1)) typeB[2] <= 1;
else if ((angle_rocket==4'b0000&&angle_enemy3==4'b1111)&& ( aliveness_of_enemy3 == 1)) typeB[2] <= 1;
else   if ((angle_rocket == angle_enemy3 || angle_rocket - 1 == angle_enemy3 || angle_rocket + 1 == angle_enemy3) && aliveness_of_enemy3
== 1) typeB[2] <= 1;
if ((angle_rocket==4'b1111&&angle_enemy4==4'b0000) && (aliveness_of_enemy4 == 1)) typeB[3] <= 1;
else if ((angle_rocket==4'b0000&&angle_enemy4==4'b1111) && (aliveness_of_enemy4 == 1)) typeB[3] <= 1;
else if ((angle_rocket == angle_enemy4 || angle_rocket - 1 == angle_enemy4 || angle_rocket + 1 == angle_enemy4) && aliveness_of_enemy4
== 1) typeB[3] <= 1;
if ((angle_rocket==4'b1111&&angle_enemy5==4'b0000)&& (aliveness_of_enemy5 == 1)) typeB[4] <= 1;
else if ((angle_rocket==4'b0000&&angle_enemy5==4'b1111)&& (aliveness_of_enemy5 == 1)) typeB[4] <= 1;
else if ((angle_rocket == angle_enemy5 || angle_rocket - 1 == angle_enemy5 || angle_rocket + 1 == angle_enemy5) && aliveness_of_enemy5
== 1) typeB[4] <= 1;
if ((angle_rocket==4'b1111&&angle_enemy6==4'b0000) && (aliveness_of_enemy6 == 1)) typeB[5] <= 1;
else if ((angle_rocket==4'b0000&&angle_enemy6==4'b1111) && (aliveness_of_enemy6 == 1)) typeB[5] <= 1;
else if ((angle_rocket == angle_enemy6 || angle_rocket - 1 == angle_enemy6 || angle_rocket + 1 == angle_enemy6) && aliveness_of_enemy6
== 1) typeB[5] <= 1;
if ((angle_rocket==4'b1111&&angle_enemy7==4'b0000) && (aliveness_of_enemy7 == 1)) typeB[6] <= 1;
else if ((angle_rocket==4'b0000&&angle_enemy7==4'b1111) && (aliveness_of_enemy7 == 1)) typeB[6] <= 1;
else if ((angle_rocket == angle_enemy7 || angle_rocket - 1 == angle_enemy7 || angle_rocket + 1 == angle_enemy7) && aliveness_of_enemy7
== 1) typeB[6] <= 1;
if ((angle_rocket==4'b1111&&angle_enemy8==4'b0000)&&( aliveness_of_enemy8 == 1) )typeB[7] <= 1;
else if ((angle_rocket==4'b0000&&angle_enemy8==4'b1111)&&( aliveness_of_enemy8 == 1) )typeB[7] <= 1;
else if ((angle_rocket == angle_enemy8 || angle_rocket - 1 == angle_enemy8 || angle_rocket + 1 == angle_enemy8) && aliveness_of_enemy8
== 1) typeB[7] <= 1;
end
if (shoot_type == 2'b11) begin
if ((angle_rocket==4'b1111&&angle_enemy1==4'b0000)&& (aliveness_of_enemy1 == 1) ||
(angle_rocket==4'b1110&&angle_enemy1==4'b0000)&& (aliveness_of_enemy1 == 1) ||
(angle_rocket==4'b1111&&angle_enemy1==4'b0001)&& (aliveness_of_enemy1 == 1)) typeC[0] <= 1;
else if ((angle_rocket==4'b0000 &&angle_enemy1==4'b1111)&& (aliveness_of_enemy1 == 1) ||
(angle_rocket==4'b0001&&angle_enemy1==4'b1111)&& (aliveness_of_enemy1 == 1) || (angle_rocket==4'b0000
&&angle_enemy1==4'b1110)&& (aliveness_of_enemy1 == 1)) typeC[0] <= 1;
else if ((angle_rocket == angle_enemy1 || angle_rocket - 1 == angle_enemy1 || angle_rocket + 1 == angle_enemy1 || angle_rocket - 2 ==
angle_enemy1 || angle_rocket + 2 == angle_enemy1) && aliveness_of_enemy1 == 1) typeC[0] <= 1;
if ((angle_rocket==4'b1111&&angle_enemy2==4'b0000)&& (aliveness_of_enemy2 == 1) ||
(angle_rocket==4'b1110&&angle_enemy2==4'b0000)&& (aliveness_of_enemy2 == 1) ||
(angle_rocket==4'b1111&&angle_enemy2==4'b0001)&& (aliveness_of_enemy2 == 1)) typeC[1] <= 1;
else if ((angle_rocket==4'b0000 &&angle_enemy2==4'b1111)&& (aliveness_of_enemy2 == 1) ||
(angle_rocket==4'b0001&&angle_enemy2==4'b1111)&& (aliveness_of_enemy2 == 1) || (angle_rocket==4'b0000
&&angle_enemy2==4'b1110)&& (aliveness_of_enemy2 == 1)) typeC[1] <= 1;
else if ((angle_rocket == angle_enemy2 || angle_rocket - 1 == angle_enemy2 || angle_rocket + 1 == angle_enemy2 || angle_rocket - 2 ==
angle_enemy2 || angle_rocket + 2 == angle_enemy2) && aliveness_of_enemy2 == 1) typeC[1] <= 1;
if ((angle_rocket==4'b1111&&angle_enemy3==4'b0000)&& (aliveness_of_enemy3 == 1) ||
(angle_rocket==4'b1110&&angle_enemy3==4'b0000)&& (aliveness_of_enemy3 == 1) ||
(angle_rocket==4'b1111&&angle_enemy3==4'b0001)&& (aliveness_of_enemy3 == 1)) typeC[2] <= 1;
else if ((angle_rocket==4'b0000 &&angle_enemy3==4'b1111)&& (aliveness_of_enemy3 == 1) ||
(angle_rocket==4'b0001&&angle_enemy3==4'b1111)&& (aliveness_of_enemy3 == 1) || (angle_rocket==4'b0000
&&angle_enemy3==4'b1110)&& (aliveness_of_enemy3 == 1)) typeC[2] <= 1;
else if ((angle_rocket == angle_enemy3 || angle_rocket - 1 == angle_enemy3 || angle_rocket + 1 == angle_enemy3 || angle_rocket - 2 ==
angle_enemy3 || angle_rocket + 2 == angle_enemy3) && aliveness_of_enemy3 == 1) typeC[2] <= 1;
if ((angle_rocket==4'b1111&&angle_enemy4==4'b0000)&& (aliveness_of_enemy4 == 1) ||
(angle_rocket==4'b1110&&angle_enemy4==4'b0000)&& (aliveness_of_enemy4 == 1) ||
(angle_rocket==4'b1111&&angle_enemy4==4'b0001)&& (aliveness_of_enemy4 == 1)) typeC[3] <= 1;
```

```
else if ((angle_rocket==4'b0000 &&angle_enemy4==4'b1111)&& (aliveness_of_enemy4 == 1) ||
(angle_rocket==4'b0001&&angle_enemy4==4'b1111)&& (aliveness_of_enemy4 == 1) || (angle_rocket==4'b0000
&&angle_enemy4==4'b1110)&& (aliveness_of_enemy4 == 1)) typeC[3] <= 1;
else if ((angle_rocket == angle_enemy4 || angle_rocket - 1 == angle_enemy4 || angle_rocket + 1 == angle_enemy4 || angle_rocket - 2 ==
angle_enemy4 || angle_rocket + 2 == angle_enemy4) && aliveness_of_enemy4 == 1) typeC[3] <= 1;
if ((angle_rocket==4'b1111&&angle_enemy5==4'b0000)&& (aliveness_of_enemy5 == 1) ||
(angle_rocket==4'b1110&&angle_enemy5==4'b0000)&& (aliveness_of_enemy1 == 1) ||
(angle_rocket==4'b1111&&angle_enemy5==4'b0001)&& (aliveness_of_enemy5 == 1)) typeC[4] <= 1;
else if ((angle_rocket==4'b0000 &&angle_enemy5==4'b1111)&& (aliveness_of_enemy5 == 1) ||
(angle_rocket==4'b0001&&angle_enemy5==4'b1111)&& (aliveness_of_enemy5 == 1) || (angle_rocket==4'b0000
&&angle_enemy5==4'b1110)&& (aliveness_of_enemy5 == 1)) typeC[4] <= 1;
else if ((angle_rocket == angle_enemy5 || angle_rocket - 1 == angle_enemy5 || angle_rocket + 1 == angle_enemy5 || angle_rocket - 2 ==
angle_enemy5 || angle_rocket + 2 == angle_enemy5) && aliveness_of_enemy5 == 1) typeC[4] <= 1;
if ((angle_rocket==4'b1111&&angle_enemy6==4'b0000)&& (aliveness_of_enemy6 == 1) ||
(angle_rocket==4'b1110&&angle_enemy6==4'b0000)&& (aliveness_of_enemy6 == 1) ||
(angle_rocket==4'b1111&&angle_enemy6==4'b0001)&& (aliveness_of_enemy6 == 1)) typeC[5] <= 1;
else if ((angle_rocket==4'b0000 &&angle_enemy6==4'b1111)&& (aliveness_of_enemy6 == 1) ||
(angle_rocket==4'b0001&&angle_enemy6==4'b1111)&& (aliveness_of_enemy6 == 1) || (angle_rocket==4'b0000
&&angle_enemy6==4'b1110)&& (aliveness_of_enemy6 == 1)) typeC[5] <= 1;
else if ((angle_rocket == angle_enemy6 || angle_rocket - 1 == angle_enemy6 || angle_rocket + 1 == angle_enemy6 || angle_rocket - 2 ==
angle_enemy6 || angle_rocket + 2 == angle_enemy6) && aliveness_of_enemy6 == 1) typeC[5] <= 1;
if ((angle_rocket==4'b1111&&angle_enemy7==4'b0000)&& (aliveness_of_enemy7 == 1) ||
(angle_rocket==4'b1110&&angle_enemy7==4'b0000)&& (aliveness_of_enemy7 == 1) ||
(angle_rocket==4'b1111&&angle_enemy7==4'b0001)&& (aliveness_of_enemy7 == 1)) typeC[6] <= 1;
else if ((angle_rocket==4'b0000 &&angle_enemy7==4'b1111)&& (aliveness_of_enemy7 == 1) ||
(angle_rocket==4'b0001&&angle_enemy7==4'b1111)&& (aliveness_of_enemy7 == 1) || (angle_rocket==4'b0000
&&angle_enemy7==4'b1110)&& (aliveness_of_enemy7== 1)) typeC[6] <= 1;
else if ((angle_rocket == angle_enemy7 || angle_rocket - 1 == angle_enemy7 || angle_rocket + 1 == angle_enemy7 || angle_rocket - 2 ==
angle_enemy7 || angle_rocket + 2 == angle_enemy7) && aliveness_of_enemy7 == 1) typeC[6] <= 1;
if ((angle_rocket==4'b1111&&angle_enemy8==4'b0000)&& (aliveness_of_enemy8 == 1) ||
(angle_rocket==4'b1110&&angle_enemy8==4'b0000)&& (aliveness_of_enemy8 == 1) ||
(angle_rocket==4'b1111&&angle_enemy8==4'b0001)&& (aliveness_of_enemy8 == 1)) typeC[7] <= 1;
else if ((angle_rocket==4'b0000 &&angle_enemy8==4'b1111)&& (aliveness_of_enemy8 == 1) ||
(angle_rocket==4'b0001&&angle_enemy8==4'b1111)&& (aliveness_of_enemy8 == 1) || (angle_rocket==4'b0000
&&angle_enemy8==4'b1110)&& (aliveness_of_enemy8== 1)) typeC[7] <= 1;
else if ((angle_rocket == angle_enemy8 || angle_rocket - 1 == angle_enemy8 || angle_rocket + 1 == angle_enemy8 || angle_rocket - 2 ==
angle_enemy8 || angle_rocket + 2 == angle_enemy8) && aliveness_of_enemy8 == 1) typeC[7] <= 1;
end
end
else if (shoot == 0 && shoot_detected == 1) begin
typeA <= 8'b00000000;
typeB <= 8'b00000000;
typeC <= 8'b00000000;
end
end
endmodule

//Module 14
module spaceship_body (
input wire clk,
input wire [9:0] x,     // next_x
input wire [9:0] y,     // next_y
output reg [7:0] color,
input wire [3:0] angle,
input wire switch_shooting_mode1,
input wire switch_shooting_mode2,
output reg [2:0] shooting_led,
input reset,
input gameover
```

```
);
// shooting mode
reg [7:0] spaceship_color;
reg shoot_detected;
reg [31:0] led_count;
initial begin
shoot_detected <= 0;
spaceship_color = 8'b11111111; // beyaz;
led_count<=0;
end
localparam CENTER_X = 320;
localparam CENTER_Y = 240;
localparam RADIUS = 20;
localparam RADIUS_SQ = RADIUS * RADIUS;
wire signed [10:0] dx;
wire signed [10:0] dy;
wire [21:0] dist_sq;
assign dx = x - CENTER_X;
assign dy = y - CENTER_Y;
assign dist_sq = dx * dx + dy * dy;
reg [10:0] dot_x;
reg [10:0] dot_y;
localparam RADIUS_dot = 7;
localparam RADIUS_SQ_dot = RADIUS_dot * RADIUS_dot;
wire signed [10:0] d_dotx;
wire signed [10:0] d_doty;
wire [21:0] dist_dot_sq;
always @(*) begin
case(angle)
0: begin dot_x = CENTER_X; dot_y = CENTER_Y - 20; end
1: begin dot_x = CENTER_X + 8; dot_y = CENTER_Y - 18; end
2: begin dot_x = CENTER_X + 14; dot_y = CENTER_Y - 14; end
3: begin dot_x = CENTER_X + 18; dot_y = CENTER_Y - 8; end
4: begin dot_x = CENTER_X + 20; dot_y = CENTER_Y; end
5: begin dot_x = CENTER_X + 18; dot_y = CENTER_Y + 8; end
6: begin dot_x = CENTER_X + 14; dot_y = CENTER_Y + 14; end
7: begin dot_x = CENTER_X + 8; dot_y = CENTER_Y + 18; end
8: begin dot_x = CENTER_X; dot_y = CENTER_Y + 20; end
9: begin dot_x = CENTER_X - 8; dot_y = CENTER_Y + 18; end
10: begin dot_x = CENTER_X - 14; dot_y = CENTER_Y + 14; end
11: begin dot_x = CENTER_X - 18; dot_y = CENTER_Y + 8; end
12: begin dot_x = CENTER_X - 20; dot_y = CENTER_Y; end
13: begin dot_x = CENTER_X - 18; dot_y = CENTER_Y - 8; end
14: begin dot_x = CENTER_X - 14; dot_y = CENTER_Y - 14; end
15: begin dot_x = CENTER_X - 8; dot_y = CENTER_Y - 18; end
default: begin dot_x = CENTER_X; dot_y = CENTER_Y; end
endcase
end
assign d_dotx = x - dot_x;
assign d_doty = y - dot_y;
assign dist_dot_sq = d_dotx * d_dotx + d_doty * d_doty;
always @(posedge clk) begin
led_count = led_count+1;
if(led_count == 25000001) begin
led_count = 0;
end
if (x < 640 && y < 480) begin
if (dist_sq <= RADIUS_SQ) begin
color <= 8'b11100000; //
```

```
end else begin
color <= 8'b00000000; // siyah
end
if (dist_dot_sq <= RADIUS_SQ_dot) begin
color <= spaceship_color; // küre rengi
end
end else begin
color <= 8'b00000000; // siyah
end
if (switch_shooting_mode1 ==0 && switch_shooting_mode2 == 0) begin
spaceship_color <= 8'b11111111; //beyaz
if (led_count == 5 && (reset == 0) && (gameover ==0)) begin
shooting_led[2:0] <= 3'b111;
end
else if(led_count == 12000000) begin
shooting_led[2:0] <= 3'b000;
end
end
else if (switch_shooting_mode1 ==0 && switch_shooting_mode2 == 1) begin
spaceship_color <= 8'b11111111; // beyaz
if (led_count == 5 && (reset == 0) && (gameover ==0)) begin
shooting_led[2:0] <= 3'b111;
end
else if(led_count == 12000000) begin
shooting_led[2:0] <= 3'b000;
end
end
else if (switch_shooting_mode1 ==1 && switch_shooting_mode2 == 1) begin
if (led_count == 5 && (reset == 0) && (gameover ==0)) begin
shooting_led[2:0] <= 3'b001;
end
else if(led_count == 12000000) begin
shooting_led[2:0] <= 3'b000;
end
spaceship_color <= 8'b11110000; // turuncu
end
else if (switch_shooting_mode1 ==1 && switch_shooting_mode2 == 0) begin
spaceship_color <=  8'b0011111;//mavi
if (led_count == 5 && (reset == 0) && (gameover ==0)) begin
shooting_led[2:0] <= 3'b011;
end
else if(led_count == 12000000) begin
shooting_led[2:0] <= 3'b000;
end
end
end
endmodule
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

//Module 15
module term_project(
/////////// CLOCK /////////
input                                   CLOCK2_50,
input                                   CLOCK3_50,
input                                   CLOCK4_50,
input                                   CLOCK_50,
/////////// SEG7 /////////
```

```
output                [6:0]          HEX0,
output                [6:0]          HEX1,
output                [6:0]          HEX2,
output                [6:0]          HEX3,
output                [6:0]          HEX4,
output                [6:0]          HEX5,
/////////// KEY //////////
input                 [3:0]          KEY,
/////////// LED //////////
output                [9:0]          LEDR,
/////////// SW //////////
input                 [9:0]          SW,
/////////// VGA //////////
output                               VGA_BLANK_N,
output                [7:0]          VGA_B,
output                               VGA_CLK,
output                [7:0]          VGA_G,
output                               VGA_HS,
output                [7:0]          VGA_R,
output                               VGA_SYNC_N,
output                               VGA_VS
);
//=======================================================
//  REG/WIRE declarations
//=======================================================
//=======================================================
//  Structural coding
//=======================================================
main(.fpgaclk(CLOCK2_50),.hsync(VGA_HS),
.vsync(VGA_VS),
.red(VGA_R),.green(VGA_G),
.blue(VGA_B),
.sync(VGA_SYNC_N),
.clk(VGA_CLK),
.blank(VGA_BLANK_N),
.btn_right(KEY[0]),
.btn_left(KEY[1]),
.sw_shooting_mode2(SW[0]),
.sw_shooting_mode1(SW[1]),
.btn_fire(KEY[2]),
.level2(SW[3:2]),
.reset_game(SW[9]),
.segments1(HEX0[6:0]),
.segments2(HEX1[6:0]),
.shooting_led(LEDR[2:0]),
.background_switch(SW[8])
);
endmodule

//Module 16
module vga_driver (
input wire clock,     // 25 MHz
input wire reset,     // Active high
input [7:0] color_in, // Pixel color data (RRRGGGBB)
output [9:0] next_x,  // x-coordinate of NEXT pixel that will be drawn
output [9:0] next_y,  // y-coordinate of NEXT pixel that will be drawn
output wire hsync,    // HSYNC (to VGA connector)
output wire vsync,    // VSYNC (to VGA connctor)
output [7:0] red,     // RED (to resistor DAC VGA connector)
```

```
output [7:0] green,   // GREEN (to resistor DAC to VGA connector)
output [7:0] blue,    // BLUE (to resistor DAC to VGA connector)
output sync,          // SYNC to VGA connector
output clk,           // CLK to VGA connector
output blank          // BLANK to VGA connector
);
// Horizontal parameters (measured in clock cycles)
parameter [9:0] H_ACTIVE  =  10'd_639 ;
parameter [9:0] H_FRONT   =  10'd_15 ;
parameter [9:0] H_PULSE   =  10'd_95 ;
parameter [9:0] H_BACK    =  10'd_47 ;
// Vertical parameters (measured in lines)
parameter [9:0] V_ACTIVE   =  10'd_479 ;
parameter [9:0] V_FRONT    =  10'd_9 ;
parameter [9:0] V_PULSE =  10'd_1 ;
parameter [9:0] V_BACK  =  10'd_32 ;
// Parameters for readability
parameter   LOW     = 1'b_0 ;
parameter   HIGH    = 1'b_1 ;
// States (more readable)
parameter   [7:0]    H_ACTIVE_STATE    = 8'd_0 ;
parameter   [7:0]   H_FRONT_STATE     = 8'd_1 ;
parameter   [7:0]   H_PULSE_STATE   = 8'd_2 ;
parameter   [7:0]   H_BACK_STATE     = 8'd_3 ;
parameter   [7:0]    V_ACTIVE_STATE    = 8'd_0 ;
parameter   [7:0]   V_FRONT_STATE    = 8'd_1 ;
parameter   [7:0]   V_PULSE_STATE   = 8'd_2 ;
parameter   [7:0]   V_BACK_STATE     = 8'd_3 ;
// Clocked registers
reg             hysnc_reg ;
reg             vsync_reg ;
reg     [7:0]   red_reg ;
reg     [7:0]   green_reg ;
reg     [7:0]   blue_reg ;
reg             line_done ;
// Control registers
reg     [9:0]   h_counter ;
reg     [9:0]   v_counter ;
reg     [7:0]    h_state ;
reg     [7:0]    v_state ;
// State machine
always@(posedge clock) begin
// At reset . . .
if (reset) begin
// Zero the counters
h_counter   <= 10'd_0 ;
v_counter   <= 10'd_0 ;
// States to ACTIVE
h_state     <= H_ACTIVE_STATE  ;
v_state     <= V_ACTIVE_STATE  ;
// Deassert line done
line_done   <= LOW ;
end
else begin
//////////////////////////////////////////////////////////////////////////
/////////////////////////// HORIZONTAL //////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
if (h_state == H_ACTIVE_STATE) begin
// Iterate horizontal counter, zero at end of ACTIVE mode
```

```
h_counter <= (h_counter==H_ACTIVE)?10'd_0:(h_counter + 10'd_1) ;
// Set hsync
hysnc_reg <= HIGH ;
// Deassert line done
line_done <= LOW ;
// State transition
h_state <= (h_counter == H_ACTIVE)?H_FRONT_STATE:H_ACTIVE_STATE ;
end
if (h_state == H_FRONT_STATE) begin
// Iterate horizontal counter, zero at end of H_FRONT mode
h_counter <= (h_counter==H_FRONT)?10'd_0:(h_counter + 10'd_1) ;
// Set hsync
hysnc_reg <= HIGH ;
// State transition
h_state <= (h_counter == H_FRONT)?H_PULSE_STATE:H_FRONT_STATE ;
end
if (h_state == H_PULSE_STATE) begin
// Iterate horizontal counter, zero at end of H_PULSE mode
h_counter <= (h_counter==H_PULSE)?10'd_0:(h_counter + 10'd_1) ;
// Clear hsync
hysnc_reg <= LOW ;
// State transition
h_state <= (h_counter == H_PULSE)?H_BACK_STATE:H_PULSE_STATE ;
end
if (h_state == H_BACK_STATE) begin
// Iterate horizontal counter, zero at end of H_BACK mode
h_counter <= (h_counter==H_BACK)?10'd_0:(h_counter + 10'd_1) ;
// Set hsync
hysnc_reg <= HIGH ;
// State transition
h_state <= (h_counter == H_BACK)?H_ACTIVE_STATE:H_BACK_STATE ;
// Signal line complete at state transition (offset by 1 for synchronous state transition)
line_done <= (h_counter == (H_BACK-1))?HIGH:LOW ;
end
//////////////////////////////////////////////////////////////////////////
///////////////////////// VERTICAL //////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
if (v_state == V_ACTIVE_STATE) begin
// increment vertical counter at end of line, zero on state transition
v_counter<=(line_done==HIGH)?((v_counter==V_ACTIVE)?10'd_0:(v_counter+10'd_1)):v_counter ;
// set vsync in active mode
vsync_reg <= HIGH ;
// state transition - only on end of lines
v_state<=(line_done==HIGH)?((v_counter==V_ACTIVE)?V_FRONT_STATE:V_ACTIVE_STATE):V_ACTIVE_STATE ;
end
if (v_state == V_FRONT_STATE) begin
// increment vertical counter at end of line, zero on state transition
v_counter<=(line_done==HIGH)?((v_counter==V_FRONT)?10'd_0:(v_counter + 10'd_1)):v_counter ;
// set vsync in front porch
vsync_reg <= HIGH ;
// state transition
v_state<=(line_done==HIGH)?((v_counter==V_FRONT)?V_PULSE_STATE:V_FRONT_STATE):V_FRONT_STATE;
end
if (v_state == V_PULSE_STATE) begin
// increment vertical counter at end of line, zero on state transition
v_counter<=(line_done==HIGH)?((v_counter==V_PULSE)?10'd_0:(v_counter + 10'd_1)):v_counter ;
// clear vsync in pulse
vsync_reg <= LOW ;
// state transition
```

```
v_state<=(line_done==HIGH)?((v_counter==V_PULSE)?V_BACK_STATE:V_PULSE_STATE):V_PULSE_STATE;
end
if (v_state == V_BACK_STATE) begin
// increment vertical counter at end of line, zero on state transition
v_counter<=(line_done==HIGH)?((v_counter==V_BACK)?10'd_0:(v_counter + 10'd_1)):v_counter ;
// set vsync in back porch
vsync_reg <= HIGH ;
// state transition
v_state<=(line_done==HIGH)?((v_counter==V_BACK)?V_ACTIVE_STATE:V_BACK_STATE):V_BACK_STATE ;
end
//////////////////////////////////////////////////////////////////////////
//////////////////////////////// COLOR OUT ///////////////////////////////
//////////////////////////////////////////////////////////////////////////
// Assign colors if in active mode
red_reg<=(h_state==H_ACTIVE_STATE)?((v_state==V_ACTIVE_STATE)?{color_in[7:5],5'd_0}:8'd_0):8'd_0 ;
green_reg<=(h_state==H_ACTIVE_STATE)?((v_state==V_ACTIVE_STATE)?{color_in[4:2],5'd_0}:8'd_0):8'd_0 ;
blue_reg<=(h_state==H_ACTIVE_STATE)?((v_state==V_ACTIVE_STATE)?{color_in[1:0],6'd_0}:8'd_0):8'd_0 ;
end
end
// Assign output values - to VGA connector
assign hsync = hysnc_reg ;
assign vsync = vsync_reg ;
assign red = red_reg ;
assign green = green_reg ;
assign blue = blue_reg ;
assign clk = clock ;
assign sync = 1'b_0 ;
assign blank = hysnc_reg & vsync_reg ;
// The x/y coordinates that should be available on the NEXT cycle
assign next_x = (h_state==H_ACTIVE_STATE)?h_counter:10'd_0 ;
assign next_y = (v_state==V_ACTIVE_STATE)?v_counter:10'd_0 ;
endmodule

//Module 17
module yildiz_kumlu (
input clk,
input [9:0] next_x,
input [9:0] next_y,
output reg [7:0] color
);
localparam [9:0] OVAL_CENTER_X = 320;
localparam [9:0] OVAL_CENTER_Y = 240;
localparam [9:0] OVAL_RADIUS_X = 100;
localparam [9:0] OVAL_RADIUS_Y = 140;
localparam [9:0] BORDER_THICKNESS = 50;//5
reg [19:0] dist_to_oval;
reg [19:0] dist_to_inner_oval;
always @(posedge clk) begin
color <= 8'b11111111;
dist_to_oval <= (((next_x - OVAL_CENTER_X) * (next_x - OVAL_CENTER_X)) * OVAL_RADIUS_Y * OVAL_RADIUS_Y) +
(((next_y - OVAL_CENTER_Y) * (next_y - OVAL_CENTER_Y)) * OVAL_RADIUS_X * OVAL_RADIUS_X);
dist_to_inner_oval <= (((next_x - OVAL_CENTER_X) * (next_x - OVAL_CENTER_X)) * (OVAL_RADIUS_Y - BORDER_THICKNESS)
* (OVAL_RADIUS_Y - BORDER_THICKNESS)) +
(((next_y - OVAL_CENTER_Y) * (next_y - OVAL_CENTER_Y)) * (OVAL_RADIUS_X - BORDER_THICKNESS) * (OVAL_RADIUS_X
- BORDER_THICKNESS));
if (dist_to_oval <= (OVAL_RADIUS_X * OVAL_RADIUS_X) * (OVAL_RADIUS_Y * OVAL_RADIUS_Y)) begin
if (dist_to_inner_oval > ((OVAL_RADIUS_X - BORDER_THICKNESS) * (OVAL_RADIUS_X - BORDER_THICKNESS)) *
((OVAL_RADIUS_Y - BORDER_THICKNESS) * (OVAL_RADIUS_Y - BORDER_THICKNESS))) begin
color <= 8'b01000111;
```

```
end else begin
color <= 8'b01000111; // cream
end
end
end
endmodule
```