FPGA Implementation of a 2D Strategy Game: Triangles vs. Circles

Ebrar Çakmak
Middle East Technical University
Electrical and Electronics Engineering Department
Ankara, TURKEY
ebrar.cakmak@metu.edu.tr

Yunus Emre Tüysüz
Middle East Technical University
Electrical and Electronics Engineering Department
Ankara, TURKEY
emre.tuysuz@metu.edu.tr

Doruk Yazıcı
Middle East Technical University
Electrical and Electronics Engineering Department
Ankara, TURKEY
doruk.yazici@metu.edu.tr

***Abstract*— This research paper presents the development and implementation of a turn-based 2D strategy game, Triangles vs. Circles. Implementation is done by using the Verilog HDL programming language and the Altera DE1-SoC Board. The implementation allows players to see the game board and their previous moves while they are playing the game with the help of the designed VGA interface.**

Keywords — verilogHDL, game, VGA, states

I. Introduction

In this report, the implementation and development of the Triangles vs. Circles strategy game will be introduced. Moreover, the methodology will be explained, and the results and gameplay will be shown. The game will be displayed by using VGA interface. During the gameplay, inputs will be taken from players which leads to winning or drawing results. The game is coded with VerilogHDL by using Quartus II and displayed with the help of an FPGA 5CSEMF31C6. 3 buttons of FPGA are used to take input. During the game serial input of 8 bit location is taken and by using activity button location is saved, and game is played.

II. Theoretical Background

A. *Gameplay Introduction*

Using a 2D coordinate system, players can place geometric shapes on a 10x10 board. The opposing teams are represented by triangles and circles in green and blue, respectively. When the score is 0-0 triangle starts the game, later a new round starts with the team that won the previous round. The FPGA board is used by players to enter their movements, and the active team can be seen on the gaming screen, since their shape will be fully colored. Forming a row of four identical shapes along an orthogonal or diagonal line will result in a red line drawn over them. Then the board resets, results in victory. Players are only allowed five moves in this game, and every sixth move after that erases the previous move. A draw is declared, the board is reset, and a new game is started if neither team succeeds before it reaches 25% occupancy.

B. *State Diagram*

The game starts with a triangle, after the first move is made by triangle, the state goes to the "Game Ongoing" state. This state consists of valid and invalid inputs. For each input type, the game continues; however, when the invalid move is made, the same player needs to give new input. Game can end under three conditions: draw, circle wins or triangle wins. After that, the game will restart with the last winner. First, a state diagram is drawn to understand the operations and project requirements better. State diagram can be seen in Figure 1.

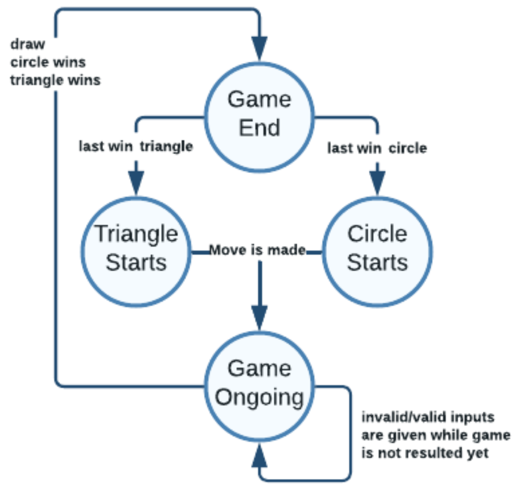
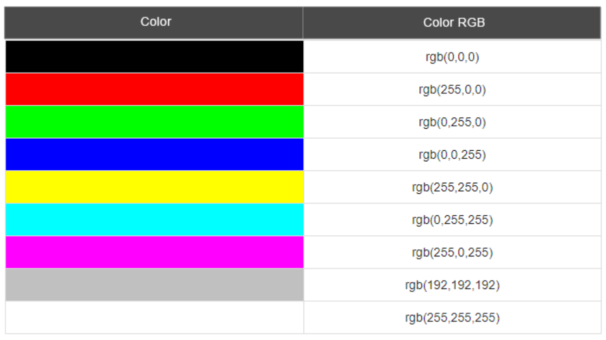


Figure 1 State diagram

III. Overall Design

A. *Display Design*

First, an interface design is required for players to see their movements and the game board itself. For this purpose, a VGA interface is used. Doruk was the one designed the VGA interface. This interface works as a grid of pixels, where each pixel can be set to a desired color by using RGB color codes. RGB color codes can be seen in Figure 2 [1].



Color	Color RGB
⬛	rgb(0,0,0)
🟥	rgb(255,0,0)
🟩	rgb(0,255,0)
🟦	rgb(0,0,255)
🟨	rgb(255,255,0)
🟦	rgb(0,255,255)
🟪	rgb(255,0,255)
⬜	rgb(192,192,192)
⬜	rgb(255,255,255)

Figure 2 RGB color codes [1]

VGA interface consists of 640x480 pixels [2]. Firstly, we made a completely white screen by making each pixel color white. Since VGA code starts counting from Y (vertical), and while Y value stays constant, the X (horizontal) value for the whole row increases. To draw the game board, horizontal and vertical lines are needed. It was easier to draw horizontal lines since during these lines Y is constant, and the X value increases as the natural behavior of the VGA. For vertical lines, Y is changed while X value stays constant [3].

Then, letters and numbers are placed on the board. Ebrar was the one who wrote the numbers and letters. For each letter and number, (x,y) codes are found for pixels and they are colored. Since this process takes too long, some words are shortened for the next parts of the display design. For example, instead of writing “Last (Recent) Position”, “LP” is used to describe it. For “Total Move” and “Total Win”, “TM” and “W” are used respectively. The circle is designed in blue, thus, its TM, W and LP values are also blue. This color is green for triangle.

To give more details on how circle and triangle is drawn:
For circle: The equation of the circle is used: $x^2 + y^2 = r$ where r is the radius of the circle. After deciding on the center, one can decide the size of the circle with r value.
For triangle: To draw the triangle, a range of x values and a height is decided. By changing the height and size of the triangle can be manipulated.

To show the turn, the shape is filled with color, i.e., the triangle (circle) is filled with color green (blue) when it is its turn. In order to fill them with colors, first the shapes are drawn with fully colored, then same shape with white coloring is drawn inside. For example, when it is triangles’ turn, the white triangle inside has zero height. Then players only see solid green colored triangle, etc. The display screen can be seen in Figure 3.

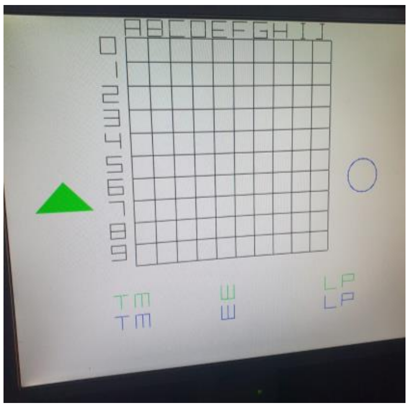


Figure 3 VGA interface (display)

Moreover, a red circle is designed as a warning to show that the given input is invalid. For valid input cases, this red circle does not show up. This red circle can be seen in Figure 4.

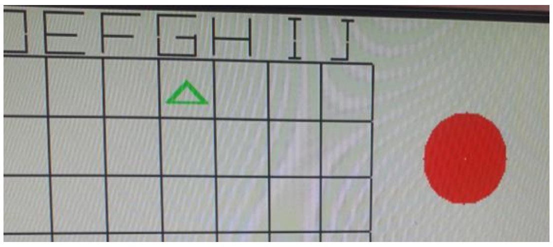


Figure 4 Invalid move warning shape

An important point regarding the display design is the clock division. Since the FPGA board has a 50 MHz clock inside, it needs to be changed to 25 MHz because VGA works with this frequency.

B. Game Algorithm and Mechanism Design

In this part, the game mechanism is designed. For this part Yunus and Doruk worked together to develop the main algorithm. Since the game consists of 25 moves at maximum, 25 turns are defined. Moreover, a 10x10 matrix is defined to save the condition of each square. If a point in array has an input of:

00: Empty, Available
01: Occupied by a Triangle, Unavailable
10: Occupied by a Circle, Unavailable
11: Occupied by a Red Square, Unavailable

The game starts by asking input from the user by using the “1” and “0” keys. Inputs are given as 8-bit array starting from the least significant bit, which is in the form of:

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
$\mathbf{Y_3}$	$\mathbf{Y_2}$	$\mathbf{Y_1}$	$\mathbf{Y_0}$	$\mathbf{X_3}$	$\mathbf{X_2}$	$\mathbf{X_1}$	$\mathbf{X_0}$

Then the player presses the “activity” button. The input is taken, if the input is valid, the movement becomes visible on the screen. If the input is not in the board range or if the chosen slot is already occupied, a red circle appears on the up-right of the screen. This circle indicates the “invalid move” case. As a result, that player needs to enter a new input. When each shape reaches the 6th move, the very first move of them changes to “Red Square” status after they enter their 6th move. These parts are written with if-else and case codes.

When a player wins, a red line is drawn over their winning shapes. To show which side won, the winner shape is filled with the color yellow. An example gameplay screen with the winner circle can be seen in Figure 5.

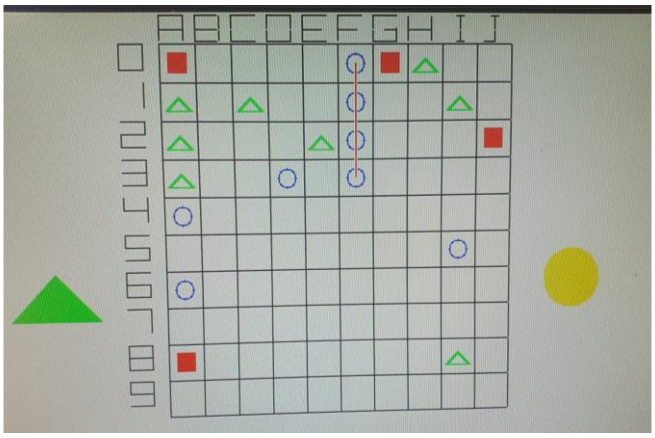


Figure 5 Circle winner example gameplay

When no sides have won after 25% of the gameboard is filled, that is called “draw”. In that case both shapes are light up in yellow. In Figure 6, an example game screen can be seen when the game resulted in a draw.

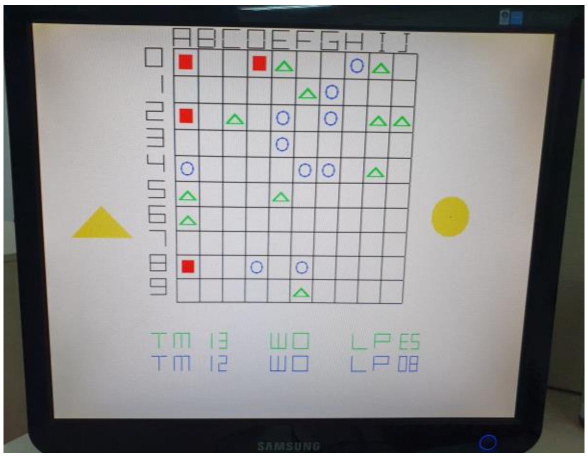


Figure 6 Game screen when the result is draw

Overall game algorithm works like this: In each case total moves and recent position values are added to the memory, then these results are used for calling the numbers and letters. For example, if the last move is A4, then A and 4 are called to be displayed on the screen. The numbers and letters are drawn by using counters and determining the range of x and y values which is explained in detail at the beginning of this section.

There are three main cases in the code: Initial state, triangle starts, and circle starts. After the initial state, the code goes to triangle state if triangle have won previously or goes to circle if circle has won previously. Moreover, location arrays, matrix and counters are reset at this state.

To find the "winner" in this game, the 10x10 matrix is checked in horizontally, vertically, left to right up to down diagonally and lastly, right to left up to down diagonally. For example, if a circle is put on a place the closest 5x5 place is being searched for 10 inputs in the matrix. If this neighbor algorithm manages to find four 10 in the same direction, circle wins.

For each winning case, the start and end values of the winner is recorded. By using these coordinates, a solid red line is drawn connecting these coordinates. Yunus and Ebrar worked on this together. Moreover, if start_x = end_x coordinates that means a horizontal win and draw a horizontal line; if start_y = end_y coordinates that means vertical win and draw a vertical line. And also, when start_x > end_x coordinate it means a diagonal win with right to left top to bottom, draw a diagonal line. Lastly if start_x < end_x coordinates, it is a diagonal win with left to right top to bottom and diagonal line is drawn. By drawing the solid red line, the winning state is concluded, and the board is reset after ten seconds.

IV. FPGA IMPLEMENTATION

For this project to be implemented in real life, Altera DE1-SoC FPGA board is used. All tests are done on FPGA board and no testbench code is written since our group had no problem accessing FPGA during the project.

An 8-bit input is needed from the players. KEY0 is used for the input "0", KEY1 is used for the input "1" [4]. Lastly, KEY3 is used for the "activity" button. When the players write the 8-bit array that is mentioned in the Game Algorithm and Mechanics part, they push the "activity" button. Input is received.

Also, from LEDR0 to LEDR7 is used to show the input. The LSB starts from the LEDR7 then starts shifting as the keys are pressed. At the end MSB of the 8-bit array is seen on LEDR7, LSB is on the LEDR0. When given value is "1" the LED is ON, when "0" the LED is OFF. The input can be observed in FPGA board before the activity button is pressed. An example for input 1001 1001 can be seen in Figure 7.

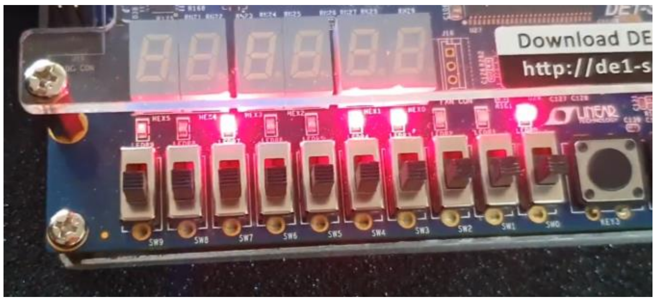


Figure 7 FPGA lights for 1001 1001 code

The input and LED PINs are all defined on Quartus with the addition of VGA PINs [4]. These VGA pins are used to give colors to the screen with the help of VGA cable.

V. CONCLUSION

In conclusion, the design and the implementation of a 2D board game using an FPGA board are demonstrated in this paper. A 10x10 grid is used for the game, and two opposing teams are represented by triangles and circles. To win the game, players must put four identical shapes in rows along orthogonal or diagonal lines. The entire design includes the design and development of a VGA interface for visualizing the game board and player movements, and also the implementation of a game algorithm to manage player inputs, validate moves, keep score, and determine winners. For the project's real-life implementation, the FPGA board was used. The display design, game algorithm and mechanism design, and FPGA implementation were all explained in the paper. The board game was successfully demonstrated in this project.

REFERENCES

[1] Alekhya. (2020, February 25). CSS colors (hex codes, RGB, short hex codes, keywords). FreshersNow Tutorials - Learn Free Courses Online. https://tutorials.freshersnow.com/css/css-colors/

[2] Vafaee, S. (n.d.). *How do VGA monitors work?*. VGA Adapter. https://www.eecg.utoronto.ca/~jayar/ece241_06F/vga/vga-monitors.html#:~:text=The%20VGA%20interface%20works%20serially,Red%2C%20Green%2C%20Blue).

[3] Gianluca Pacchiella. (2018, January 23). *Implementing VGA interface with Verilog*. Gianluca Pacchiella. https://ktln2.org/2018/01/23/implementing-vga-in-verilog/

[4] "De1-SoC user manual", 2014. [Online]. Available: https://courses.cs.washington.edu/courses/cse467/15wi/docs/DE1_SoC_User_Manual.pdf. [Accessed: 3-July -2022].