FPGA Implementation of a 2D Strategy Game

Abdullah Emir Göğüsdere,Yahya Alp Akçay, İsmethan Öncel

Department of Electrical and Electronics Engineering

Middle East Technical University

Ankara,Turkey

E-mail: abdullah.gogusdere@metu.edu.tr , akcay.alp@metu.edu.tr ,oncel.ismethan@metu.edu.tr

***Abstract*—We propose a system that implements a 2D strategy game using FPGA board. The system is proposed to take three inputs and shows the output on VGA screen.**

Keywords—FPGA, VGA

I. INTRODUCTION

In that project, we designed a system that displays a 2D strategy game. In that system, we designed a screen that is 640x480 pixels and consists of 300 tiles with each tile being 32x32 bit. The system takes 3 inputs from the FPGA board, which are logic 1, logic 0, and activity button. For that system, we have several circuits in the background. For the logic of the game, we first designed a game manager circuit. Game manager includes all the logical properties of the game such as rules, player orders, and win and lose situations. Furthermore, to convert these logical operations to pixels, we designed a circuit which is a Pixel Drawer circuit. Pixel drawer circuit uses Read Only Memory and Random Access Memory. The first one is used for creating all the tiles, whereas the latter one is used for manipulating the game board pixels in accordance with the game manager circuit, and in addition to the game manager circuit, inputs coming from the FPGA board affect the Random Access Memory. With these memories, we draw pixels in the Pixel Drawer circuit. The next action is screening these properties on the VGA screen. For that operation, we used the 3-bit output of the Pixel Drawer circuit. With the benefit of that 3-bit, we created a multiplexer circuit to choose RGB colors. Moreover, to synchronize our circuit with the VGA screen we benefited from a circuit which is named VGA synchronizer.

In that project, Game Manager part is created by Yahya Alp Akçay and İsmethan Öncel whereas, VGA part is created by Abdullah Emir Göğüsdere.

II. GAME MANAGER CIRCUIT

In our game, we need to change the specific tiles in accordance with the inputs coming from the FPGA board. We need to change the tiles directly according to FPGA inputs such as placing the triangles and the circles on the specific tiles. We also need to change some of the tiles indirectly according to FPGA inputs. For example, total moves win recent positions should be kept in the memory and need to be read from the memory. According to the win or draw situation we are required to write an expression on the board. Before writing that expression, we need to check win, and draw possibilities. For that, we need to check every win position situation.

Positioning the figures on the screen requires coordinate information of the corresponding tiles. These position vectors are outputs of our game manager. The part of the game board, in which we inserted our figures consists of 100 tiles. We designed a logical system that starts from the upper leftmost tile and goes toward the end of the same row, increasing the column number. The logic is similar to the VGA screen which also completes the row first and starts the following row after. Our system takes 8-bit binary input from the FPGA board. The least significant 4-bit of that input represents the row of the position. Whereas the most significant 4-bit represents the column number of the position. However, since with 4-bit a number larger than our row or column can be written, we need to check the validity of the input. If it is not valid, the system should not respond to that input.

In the case of valid input, the system should place the position on the board. With choosing suitable row and column. Moreover, after the placement, there can be a win situation, so the system is required to check that. There are several win situations, the first situation is that the four triangles can be aligned in 45 degrees. In that case, after the placement of the last triangle, we should check its right and left sides. In that case, there are also several scenarios such as the last triangle can be at the rightmost, second from rightmost, leftmost, and second from leftmost. We need to check every situation. They can also be aligned at 90 degrees on the board, and we also check that situation in every manner same as a 45-degree situation. Moreover, there are other win conditions such as 135 degrees aligned, and 180 degrees aligned. For those cases, we need to check every possibility such as done at 45 degrees. All the mentioned win situations are also applied to the circles.

Other than triangles and circles we also need to add squares to our system. The condition for the squares is that, at each 6th turn of the figures we need to replace the first figure which is a triangle or square with the red square. For that logic, we also need to keep the first and seventh positions of the circles and triangles separately.

In addition to the win situation mentioned above, there can be a draw situation. We can check this situation at the

25th turn. If there are not four triangles or circles aligned at 45, 90,135, or 180 degrees, this situation is a draw.

III. Pixel Drawer Circuit

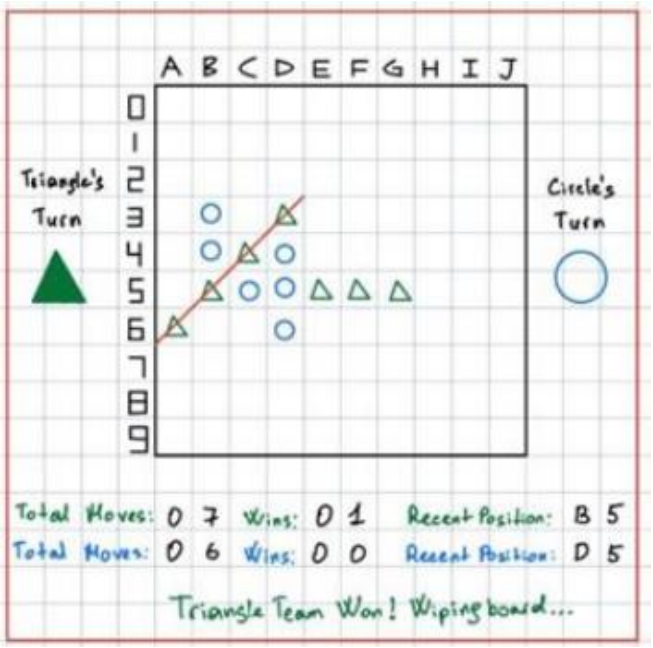

Figure 1: Shows example scenario for the game.

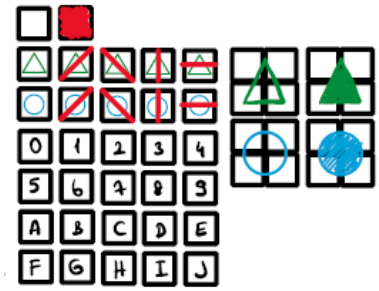

Figure 2: Shows figures included in BROM.

Pixel Drawer circuit uses block read-only memory (BROM) and blocks random access memory (BRAM) to draw tiles. These two components include the necessary information to draw pixels. The first memory that we will take into consideration is BROM. BROM is used for storing permanent data. BROM uses pre-programmed data. Moreover, BROM is read-only memory, it cannot be changed. These qualifications of the BROM are beneficial for our circuit. Since we need to use several figures on our VGA board without changing them. Examples of these figures are circles, triangles, numbers, and letters on the coordinate axis. Every element in the BROM consists of 32 bits. With the benefit of Pixel Drawer, we reached the required elements by addressing BROM.

Secondly, we will take into consideration BRAM. That is a type of memory that is embedded in the FPGA board. With the benefit of BRAM, we manipulate the pixels. In that situation, we benefit from constant data in the BROM. We address figures in the BROM with the benefit of BRAM and obtain the required figure. We need to place triangles or circles according to input coming from the FPGA board. These elements have their addresses in the BROM. Furthermore, in a win situation, we need to replace these elements with their half-crossed corresponding. For aligned 4 triangles in 45, 90, 135, or 180 degrees, we need to address lined triangles and change these triangles with the corresponding tiles. For the circle-win situation same procedure applies. In the 6th turn, we need to replace the first triangle and circle with the square element. The address for that square is also included in the BROM. All the elements included in BROM are given in Figure 2. Elements shown on the right in Figure 2 show the turn of the game. In other words, if triangles turn then the filled triangle and non-filled circle will be shown on the screen, whereas in circles turn filled circle and non-filled triangle will be shown on the screen.

On the VGA board, there will be two types of tiles. The first type is constant tile, which does not change. As shown in Figure 1, the numbers shown on the column and the letters in the row, Triangle's Turn, Circle's Turn, Total Moves, Wins, and Recent Position expressions do not change. Even the win situation occurs, and the board is wiped. Because these are not related to inputs coming from the FPGA board. Thus, these tiles will be shown in all conditions. The second type of tile is dynamic tile, which changes according to inputs coming from the FPGA board. These tiles not only change direction according to inputs but also change indirectly. Directly changing tiles are inserting positions of the triangles and squares. That is a 10x10 tile part, which totally includes 100 tiles. According to input taken by FPGA, triangles are positioned in the corresponding position in these 100 tiles. For that operation, BRAM uses BROM to change empty tile to tile including Triangle. Moreover, in the following turns the same process is repeated. For the circle's turn same procedure also holds. Indirect changes are out of 100 tiles. Indirect changes meaning, the changes occurred because of a set of inputs. For example, changes in the corresponding number to wins and recent position elements are changed after a set of inputs is applied. All of the changes are controlled by BRAM in the relation to Game Manager circuit.

IV. VGA Synchronizer and Multiplexer

A.VGA Synchonizer

VGA synchronizer is a circuit that syncs the timing between "pixel drawer" and monitor. It is composed of five

processes, two input and six output ports.

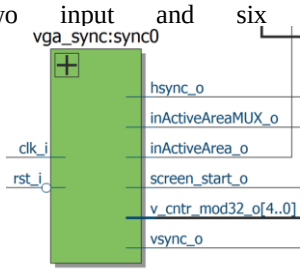

Figure 3: Shows VGA synchronizer circuit.

Ports:

Inputs:

• clk: It is a clock signal that is required to have a particular rate according to the industry VGA standard timing. For this project, it must be approximately 25MHz for a 60Hz refresh rate.

• rst: It is a reset signal to make the circuit go to its initial state.

Outputs:

• hsync: It is a horizontal timing signal that sends logic "1" while the scanline part is in a visible area, front porch, back porch, sends logic "0" while scanline part is in sync pulse. The time intervals for horizontal scanline parts are given in Table 1.

Table 1: Shows horizontal timing.

Horizontal timing (line)

Polarity of horizontal sync pulse is negative.

Scanline part	Pixels	Time [µs]
Visible area	640	25.422045680238
Front porch	16	0.63555114200596
Sync pulse	96	3.8133068520357
Back porch	48	1.9066534260179
Whole line	800	31.777557100298

• vsync: It is a vertical timing signal that sends logic "1" while the scanline part is in a visible area, front porch, back porch, sends logic "0" while scanline part is in sync pulse. The time intervals for vertical scanline parts are given in Table 2.

Table 2: Shows vertical timing.

Vertical timing (frame)

Polarity of vertical sync pulse is negative.

Frame part	Lines	Time [ms]
Visible area	480	15.253227408143
Front porch	10	0.31777557100298
Sync pulse	2	0.063555114200596
Back porch	33	1.0486593843098
Whole frame	525	16.683217477656

• inActiveAreaMUX: It is for signaling the vga_rgb_mux circuit when the visible area (640x480) is sent to the monitor.
• inActiveArea: It is for signaling the pixel_drawer circuit one clock cycle before when the visible area is sent to the monitor.
• screen_start: It is for signaling the pixel_drawer circuit one clock cycle before when the new frame is going to start.
• v_cntr_mod32: It is for signaling the pixel_drawer circuit which row is now sent to the monitor.

Processes:
1. Horizontal and Vertical Line Counting:
This process is to track the current pixel coordinates and is used while sending the required sync signal to the monitor.

2. Creating Horizontal and Vertical Sync Signals:
This process is to inform the monitor about the timing of the pixel sending signal. "Hsync" and "Vsync" signals are sent according to the industry standard timing.

3. Creating and sending the "inActiveArea" Signal:
This process is to apprise the "pixel drawer" of the start of the visible area by sending the signal named "inActiveArea".

B. Multiplexer

RGB Multiplexer is a circuit that converts the 3-bit color representation of the project to the 8-bit RGB data.

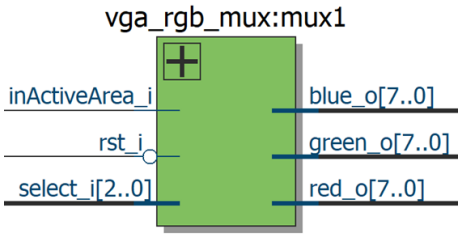

Figure 4: Shows multiplexer circuit.

This circuit is not sequential but combinational since the conversion does not require a significant amount of time.

The conversion behavior of this multiplexer can be explained simply through the "case" part of the source code. Here is the "case" part of the code.

```
// BLACK
'b000    :   begin
                red_o   <= 'h00;
                green_o <= 'h00;
                blue_o  <= 'h00;
                end
// WHITE
'b111    :   begin
                red_o   <= 'hFF;
                green_o <= 'hFF;
                blue_o  <= 'hFF;
                end
// RED
'b100    :   begin
                red_o   <= 'hFF;
                green_o <= 'h00;
                blue_o  <= 'h00;
                end
// GREEN
'b010    :   begin
                red_o   <= 'h00;
                green_o <= 'hFF;
                blue_o  <= 'h00;
                end
// BLUE
```

```
'b001    :    begin
              red_o   <= 'h00;
              green_o <= 'h00;
              blue_o  <= 'hFF;
              end
```

When the VGA synchronizer signals the multiplexer that the scanline part is now the visible area, the mux converts the color data coming from the pixel drawer to its 8-bit RGB data through the port named “select”. When the scanline part is not the visible area, the mux sends black color to the monitor, that is, it sends 8’b00000000” through each red, green, and blue port.

V. Conclusion

In that paper, we mentioned about 2D strategy game using an FPGA board. We had several qualifications and circuits in that system. Such as, we first created a game manager circuit to control the game logic. We had many possibilities for positioning figures and win-or-draw situations. We have taken into consideration all these possibilities. On the other hand, we benefited from BROM and BRAM for drawing pixels. BRAM is controlled by a game manager circuit. Whereas data in BROM is taken from BROM with the benefit of BRAM. Furthermore, for the VGA multiplexer circuit, we benefited from BRAM and BROM. Another circuit is a multiplexer, which we used to draw colors on the board. In that project many circuits are used, we take 3 inputs which are logic 0, logic 1, and activity buttons, according to these inputs the circuit makes required manipulations on the VGA board.

Acknowledgment *(Heading 5)*

References

[1] “VGA Microcontroller projects”. TinyVGA.com. VGA Signal Timing (tinyvga.com)