EE314 Spring 2023 Project Work Final Report – Group 29

Ahmet Akman 2442366 – Yusuf Toprak Yıldıran 2444149 – Murathan Kutaniş 2516482

I. Introduction

This project report presents a tic-tac-toe game's successful design and implementation using an FPGA development board. The objective was to create a game with triangle and circle symbols on a 10x10 grid, where players aim to achieve four consecutive symbols for a win. The FPGA development board's buttons served as inputs, while the output was displayed on a VGA port. The DE-1 SoC was the FPGA development board used to create the game. The report is structured into four main sections, starting with a discussion on the general design philosophy that focuses on modularity, efficiency, and resource utilization. The subsequent sections provide detailed descriptions of the input sequence, game logic, and VGA interface. The input sequence section explains the button-based input system and debouncing techniques for accurate user inputs. The game logic section elaborates on the algorithms and state machines used to manage the game's rules, win conditions, and symbol placement. Lastly, the VGA interface section highlights the synchronization and display generation processes, ensuring a seamless visual representation of the game on a VGA monitor. This project report serves as a valuable resource for FPGA-based game development, showcasing the successful integration of hardware and software components to create an engaging and interactive tic-tac-toe game. It also discusses design decisions, technical challenges, and potential areas for future enhancements.

II. General Structure and Design Philisophy

The outline of this project uses a modular design approach consisting of three main modules and a top-level module that connects them. The first module is responsible for handling the serial input, allowing players to interact with the game using logic 0, logic 1, and activity buttons. This module takes coordinates of the desired movement from players and sends it to the game logic module for necessary calculations. The serial input module acts as an interface between the players and the game. The latter module, the game logic module, includes the state machine that manages the state game flow. It manages the current state of the game according to the given coordinates of the shapes, keeps track of the positions of the triangles and circles on the game board, players' movement counts, and recent movements, and checks the validity of moves made by the players if an invalid movement is logged, the player is asked to play again. It also detects winning and final conditions, win counts, and deleted coordinates and then determines the outcomes and following state. This module communicates with the serial input module to receive the moves of players and updates the game state. The Vga module logs the triangles and circles onto

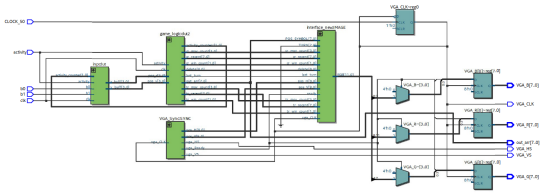

Fig. 1: Module structure from Quartus II.

the screen according to the given outputs of the game logic module. Different states of the game logic module give different output variables, which will be used in the VGA module later. This module also prints some texts declaring winning or draws statements on the screen. The top-level module connects these three modules by making the correct connections between the outputs of the serial input module and the inputs of the game logic module, and outputs of the game logic module, and the inputs of the VGA module as well as making sure of providing all the inputs and outputs properly. Figure 1 represents the basic overview of the module structure of our project.

III. Input Sequence

The Input logic function is a crucial component in the process of transferring a player's input to the game logic in the correct format. This process is essential to ensure that the player's input is accurately reflected in the game, providing an optimal playing experience. The input is received by pressing a button on the FPGA board, which is then indicated by the resulting lights on the board. The

lights help the player see their input, making it easier to confirm that their input has been registered correctly.

To ensure that the input is processed effectively, it is converted to an 8-bit format, with the first 4 bits representing the x-axis and the last 4 bits representing the y-axis. This format is essential to ensure that the game logic can accurately interpret the player's input. To convert the serial inputs to parallel outputs, a shift register is utilized, as demonstrated in Figure 1b. This method is effective in ensuring that the input is correctly processed and interpreted by the game logic.

It is also important to note that if a player inputs more than 8 bits, the extra bits will not be considered. For instance, if a player enters 8-bit input and then enters an additional input, the rightmost bit (1) will be ignored. This feature is designed to prevent any errors that may occur as a result of extra inputs, ensuring that the gameplay remains fair and accurate at all times. After

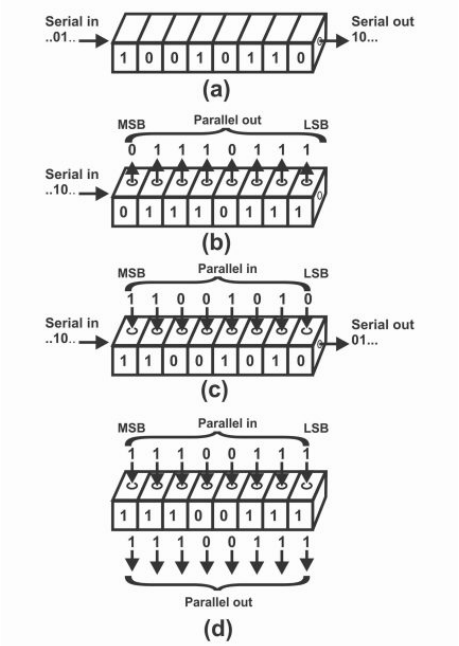

Fig 7.8 Shift register characteristics. (a)Serial in-serial out.(b)Serial in-parallel out. (c)Parallel in-serial out.(d)Parallel in-parallel out.

Fig. 2: Basic scheme for shift register.

the input sequence is completed, the activation button is pressed and the output is send to the game logic part.

IV. Game Logic

This module waits for some input from the serial input module until pressed activity button is. A matrix called mat_val is used for tracking filled places on board by only triangle, only circle, and triangle or circle shapes. Therefore, it is a ten-by-ten matrix where each index has 3 bits of depth where. The first is a triangle, the second is a circle, and the last bit is used for filled indexes. If we look closer into the states that form the module, we can see that the first state is idle, which makes the matrix zero in order to be able to start the game again after the final state. Afterward, the idle state updates the state by giving the order of play to the winner since the $last_turn$ output variable is always updated according to any case of win or turn change. The next state can be $triangle_turn$ or $circle_turn$ according to who the winner is; in these states, the validity of the player's input is checked, and if it is invalid, the player is asked to give another input. Invalid inputs may be the binary coordinates larger than the decimal number 9 and already filled coordinates. Then, to be able to easily check if there is any winning score in the next state, the player's input is kept in the same matrix in another bit where the matrix is formed by only that player's movements. In the case of valid input, 7-bit out_arr where the first 4 bits are the row number and the last four ones are the column numbers of the given coordinate is returned as output as well as the movement count and recent position of that player to the VGA module. At the end of each one of these states, the state is updated to win_our. In this state, it is checked if there are any four sequential shapes that are the same or not by scanning rows and columns sequentially, and if there is one, then the corresponding player's win count is incremented. As understood above, our code was scanning the board only horizontally and vertically, not diagonally, because of our lack of time of us. Lastly, the state is updated to a final state where $movement_count$'s of each team are checked, and if any of the players made the 6th or 12th movement, then corresponding recent coordinates will be deleted. To do this, these coordinates are returned to the VGA module for them to be deleted. After some controls, winning and warning texts are popped up on the screen via the inputs given to the VGA module, and the next state is updated as idle and goes on like this. State diagram used here can be seen Figure 3

V. VGA Interface

In this section, we will explain how we used the Video Graphics Array (VGA) protocol in our FPGA-based game project. The VGA protocol is a well-established standard for video display, known for its distinctive signal timing and synchronization mechanisms. These mechanisms, such as horizontal and vertical sync pulses, play a crucial role in defining the image's structure and position on the screen. By leveraging these principles, we were able to create an engaging and responsive visual interface for our game. This section will explore the details of the VGA protocol and demonstrate how

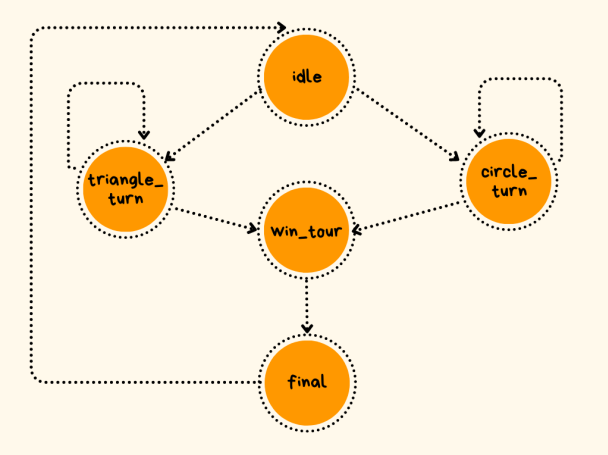

Fig. 3: State diagram.

we creatively adapted it to meet our project's needs, resulting in a seamless gaming experience.

A. VGA Timing and Synchronization

The VGA (Video Graphics Array) screen with a resolution of 640x480 pixels operates based on a specific timing and synchronization mechanism. This mechanism is crucial for the correct display of images on the screen.

The VGA screen operates at a refresh rate of 60 Hz, which means the entire screen is redrawn 60 times per second. The pixel clock for this resolution is approximately 25.175 MHz. This clock rate determines the time taken to display one pixel.

The timing mechanism involves several stages:

- Horizontal Sync: Each line of pixels (640 in this case) is drawn from left to right. After each line, there's a brief period called the Horizontal Blank during which no pixels are drawn. This period allows the electron beam in CRT monitors to move back to the start of the next line. The Horizontal Sync pulse, which is a part of this blanking period, signals this return.
- Vertical Sync: After all lines (480 in this case) are drawn top to bottom, there's a longer pause called the Vertical Blank. This allows the electron beam to move from the bottom right back to the top left of the screen. The Vertical Sync pulse signals this return.

The timing for a 640x480 VGA screen is being utilized as follows:

- Horizontal: 31.46875 kHz (period of 31.777 microseconds)
  - Visible area: 25.422 microseconds (640 pixels)
  - Horizontal sync pulse: 3.813 microseconds (96 pixels)
  - Front porch: 0.636 microseconds (16 pixels)
  - Back porch: 1.907 microseconds (48 pixels)
- Vertical: 59.94 Hz (period of 16.683 milliseconds)
  - Visible area: 15.253 milliseconds (480 lines)
  - Vertical sync pulse: 0.064 milliseconds (2 lines)
  - Front porch: 0.318 milliseconds (10 lines)
  - Back porch: 1.048 milliseconds (33 lines)

These timings ensure that each pixel is displayed at the correct location and time, providing a stable and flicker-free image.

B. Our Approach on Utilization

As shown in Figure 4, the development board we have been working on can define 8 bits per channel. That is, we can actually use the whole 24 bits of color depth to make our interface more exquisite. However, here, the

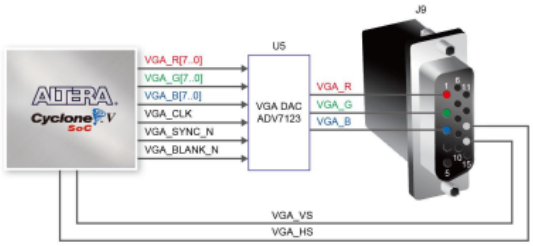

Fig. 4: VGA port basic diagram.

challenge lies in the hardware. First, it is not that feasible to hold 640x480x3 data in the memory. Secondly, it is also not necessary for us to use all 8 bits of each channel to have a good-looking interface. We have followed the steps given as follows.

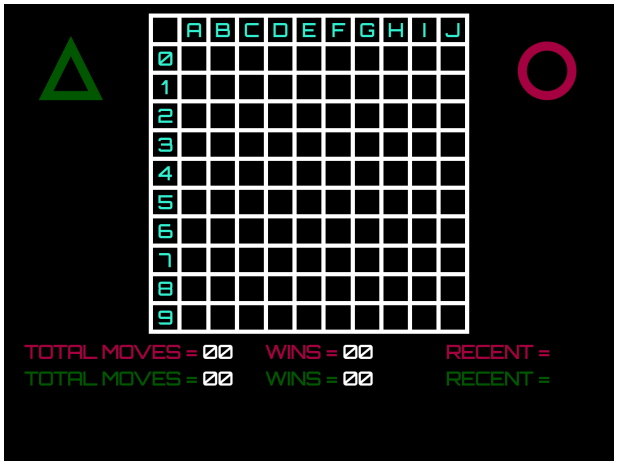

Fig. 5: Base interface design.

- We created the base interface images at draw.io and used basic colors with a distinctive font (Orbitron).
- We create the symbols and numbers messages at draw.io.
- We rescale the base image to 320x240x3 in Matlab and reduce the color depth to 4 bits per channel, totaling 12 bits. The image can be seen in the Figure 5
- We use the base image specs to determine the local positions and the pixel-wise scales of the respective symbols, numbers, letters, and messages.
- Lastly, we write those reduced images to a hexadecimal coded .txt file.
- We read those hexadecimal values at FPGA.
- We first view the base interface in the main loop. **Importantly** we publish each pixel twice to have a 640x480 image on the screen.
- According to the game dynamics, we manipulate the base interface.

As a result, we obtain a clear, fancy, responsive interface on the screen, as can be seen from Figure 6. That is to say, we have used 320x240 base information and 4 bits per channel to effectively deal with memory management and compilation time reduction.

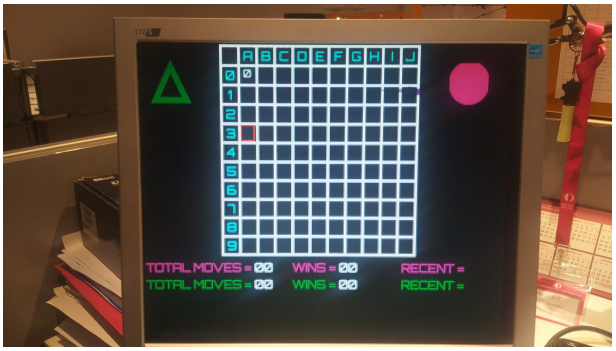

Fig. 6: Interface in action.

VI. Conclusion

In this detailed report, we delve into the creation and implementation of a tic-tac-toe game using FPGA-based hardware as the final project for EE314 digital electronics design laboratory course. Our main focus was on the input sequence, game logic, and VGA interface aspects of the game. We wanted to ensure that the player inputs were handled efficiently, so we incorporated a shift register in the input sequence module to capture and process the serial input seamlessly. The game logic module was responsible for managing the gameplay mechanics and player interaction, while the VGA interface module utilized the VGA protocol to create a visually appealing display on the screen.

One of our goals was to optimize color depth and memory management to ensure that the interface was clear and responsive. We also wanted to reduce compilation time to streamline the game's development process. Through rigorous testing and optimization, we were able to achieve our objectives, resulting in an engaging and well-executed FPGA-based game.

Our project was successful in incorporating digital electronics concepts and practical game design principles, which allowed us to produce a game that was both engaging and educational. We believe that our findings will be useful to anyone interested in developing FPGA-based games or learning more about digital electronics design. Lastly, it can be said that our relatively ineffective performance on the project demonstration is a proof-of-concept for the difference between the responsibility perception of group members.

References.

- DE1-SoC user manual. http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/de1-soc_user_manual.pdf
- Online resource on FPGA interfacing approach. https://www.instructables.com/Image-from-FPGA-to-VGA/