Digital Design I

Pong Game

Mohamed El-Refai 900222334
Hana Shalaby 900223042
Mohamed Azouz 900225230

Introduction

This project implements a classic Pong game using Verilog HDL using FPGA, showcasing real-time performance and modular hardware design. Main features include VGA synchronization for a 640x480 display which can be adjusted as well, paddle and ball control with collision detection, and dynamic text rendering for scores and game messages. Modules such as **vgaSync**, **paddleCtrl**, and **ballCtrl** ensure smooth gameplay, while clock dividers manage timing constraints. The system also includes visual customization options like dark mode, highlighting the potential of FPGA-based systems for interactive gaming applications.

Design Approach

The main module that allows us to display pixels on the screen is called **vgaSync**. It works on the horizontal and vertical sections separately and syncs them to appear to update simultaneously. The module counts the vertical position according to specific parameter values that align with the display on which we tested the game. Analogously for the horizontal pixel position as well. Then it synchronizes the signals. There is an output signal **display_on** that is true if and only if the pixel position satisfies the parameters. The rest of the program uses this signal as an enable.

In order to get the appropriate clock frequency that would work with the display we had, we used a built-in module from the IP-Catalog on Vivado called **Clock Wizard.** With our input clock clk of 100MHz, we generated an output clock frequency of exactly 25.175MHz. In **display.v** we instantiated the module as **clk_wiz0()** that takes as input 100MHz as clk and outputs 25MHz as clk_out.

The ball is controlled by a separate clock divider which is defined in the **display** module and this clock is then passes into **ballCtrl** module which works on the movement and direction control of the ball and that allows us to control the clock of the ball separately allowing us to enable or stop the ball using the **enable** input while keeping the movement of the paddles normal. This allowed us to test all the possibilities of the interactions of the ball with the paddles and screen edges. Moreover, In the module itself it has horizontal **hCountOut** and vertical **vCountOut** positions are controlled using specific counter modules in them. These modules are used to increment or decrement the ball's position based on its current direction (**hDirection and vDirection**) and the direction toggling is triggered by collision signals (**hCol and vCol**) which has it's logic in the display module and that allows the ball to bounce of paddles as well as screen boundaries by flipping the signals everytime there is a needed change in direction. For instance, change the vertical direction when it hits the top of the screen or the bottom. In addition the ballCtrl module is used in the scoring system in the game and that is implemented so that when the ball crosses the left or right screen boundaries, the corresponding player scores are updated (**score1 & score2**). The module also handles reset conditions to initalize or reposition the ball as when reset is triggered the ball is centerd on the screen.

The paddles are controlled by the **clk_wiz** clock divider we used in **display** module, so the module processes user inputs for moving the paddle up or down (**pushup & pushdown**) using debouncing logc to eliminate noise and ensure clean input signals, allowing for a reliable paddle movement. There is also an enable signal which is generated by XORing the debounded pushup and pushdown signals, ensuring the paddle moves only when there is an active user input, preventing unnecessary or unintended motion. Besides that, the paddle positioning is managed by the **paddleCount** submodule which keeps track of the paddle's vertical position within the screen boundaries by incrementing or decrementing based on the input direction (**upwire & downwire**). The paddle then outputs the paddle's current position (**coord)** as an 8 bit value which is used in **display** module for collision detection as well as to determine the paddle's location and interaction with the ball.

For the score text and "GAME OVER" at the end of the game, we have a ROM called **ascii_rom** containing the ASCII codes we need. The display module requests the codes and maps them to the screen according to the specified region we require the text to be in.

The **display** module which we can consider the "top" module the controls everything, integrating VGA synchronization, paddle and ball controls, and score display to create a seamless experience. And it dynamically renders paddles, the ball, and background colors while supporting a dark mode for visual customization. The module includes seven-segment displays for player scores and integrates ASCII ROM-based text rendering for "GAME OVER" messages and scores. Utilizing inputs from **paddleCtrl** and **ballCtrl**, the display module manages collision feedback, game states, and visual updates, ensuring smooth, real-time interaction on a 640x480 VGA screen.

Challenges Faced

The main challenge we faced was displaying the correct score on the screen and fixing glitches regarding the numbers displayed. The issue was related to having multiple areas in the code where we do the reset of the score. Another issue was more hardware related, where we tried to work on a different PC from the one we usually use in the lab, and the clk_wiz module was not configurated correctly. This prevented the ball and paddles from moving. Even after regenerating the output frequency correctly, the other PC was not running the game as expected even though the code was identical. The issue went away when we returned to the initial PC we were working on.

Contributions

Most of the work done on the project is done together in the lab because of the software restriction since it is only available in the lab as well as the FPGA boards. We started off by creating the block diagrams, Mohamed El-Refai worked on the VGA drive module block diagram while Mohamed Azouz and Hana Shalaby worked on the ball logic block diagram. We then started working on the **vgaSync** module to allow us to display just a square on the screen which was required for milestone 1 to display anything meaningful. After that, we started working on the **paddleCtrl** as well as **ballCtrl** to get us something that can be played. However, it was had some bugs in the reset as well as the collisions was not accurate but that was enough for milestone 2. Finally, we started working on the **ballCtrl** to improve it to display the ball as an actual ball not as a square like before in milestone 1. Also, we added scoring system in the **ballCtrl** as mentioned in the design approaches. We also worked on fixing the **paddleCtrl** as it used to go past the edges of the screen which was incorrect. All of that we also had to adjust to these changes in the **display** module which we also added the score display at this point as well as fixed the seven segment display score display.

Conclusion

The implementation of the Pong game on an FPGA using Verilog HDL demonstrates the successful integration of hardware design principles to create a fully functional and interactive gaming system. The modular design of key components, such as **vgaSync**, **paddleCtrl**, and **ballCtrl**, ensures smooth gameplay, real-time responsiveness, and dynamic visual rendering. Despite challenges such as debugging score display glitches and hardware-specific issues, the project highlights the potential of FPGA based systems for real-time applications. This project provided a hands-on opportunity to explore hardware-software co-design, enhance problem-solving skills, and collaborate effectively as a team to deliver an engaging gaming experience.