PROJECT TOPIC: PING PONG GAME USING VHDL

Group Members:

Onifade Victor Adedamola	EEG/2018/117
Ojo Tomiwa Oyindamola	EEG/2018/056
Zinsu Chibuike Jesutomisin	EEG/2019/092
Ajayi Ogooluwa Emmanuel	EEG/2018/015
Oyeyinka Oyejide	EEG/2019/090

Abstract

This report presents the design and implementation of a Ping Pong game using VHDL (VHSIC Hardware Description Language) and the DE10-Lite FPGA development board. The objective of this project was to create a simple yet entertaining game that showcases the capabilities of FPGA-based hardware design and highlights the versatility of VHDL in digital system development. This report covers the project's design methodology, implementation details, and results.

1. Introduction

1.1 Background

Ping Pong, also known as Pong, is a classic video game that simulates table tennis. It was one of the earliest arcade video games and became immensely popular in the 1970s. The game involves two paddles and a ball, with players controlling the paddles to hit the ball back and forth. The objective is to score points by making the opponent miss the ball.

The Ping Pong Game implemented using VHDL is a digital recreation of the classic table tennis game. This project aims to demonstrate the use of VHDL for creating interactive and engaging games on FPGA platforms. The game involves two players controlling paddles to hit a virtual ball back and forth, simulating the gameplay of ping pong.

1.2 Objectives

The primary objectives of this project are as follows:

- Implement a Ping Pong game using VHDL on the DE10-Lite FPGA board.
- Create a user-friendly interface for player control.
- Display the game graphics on an external monitor via VGA output.
- Implement game logic including ball movement, collision detection, and scoring.
- Demonstrate the capabilities of VHDL for digital system design.

1. Design Methodology

2.1 Hardware Components

The key hardware components used in this project include:

- DE10-Lite FPGA Board: This serves as the platform for implementing the game.
- VGA Monitor: The game graphics are displayed on an external monitor.
- Push Buttons: Used for player input to control the paddles.
- Seven-Segment Displays: Show the score for each player.
- Speaker or Buzzer: Used for sound effects.

2.2 System Architecture:

The system consists of the following major components:

Paddle Controllers: VHDL modules responsible for receiving user input and moving the paddles accordingly.
Ball Movement: Logic to calculate the trajectory and movement of the ball based on its speed and angle.
Collision Detection: Algorithms to detect collisions between the ball, paddles, and game boundaries.
Scoring System: Tracking and updating the players' scores based on successful ball hits.
Display Controller: Module to generate the visual output on the display, representing the game state.

1. **VHDL Modules:**

   A. **Paddle Controller:**
   Input: Player commands (up/down)
   Output: Paddle positions
   Logic: Processes user input and generates control signals to move the paddles.

   A. **Ball Movement:**
   Input: Ball speed, angle
   Output: Ball position
   Logic: Uses basic physics equations to compute ball trajectory and update its position.

   A. **Collision Detection:**
   Input: Ball position, paddle positions, boundaries
   Output: Collision signals
   Logic: Checks for collisions with paddles and boundaries, updates ball direction if collision occurs.

A. **Scoring System:**
Input: Collision signals
Output: Player scores
Logic: Increments the score of the appropriate player when a collision occurs.

A. **Display Controller:**
Input: Ball and paddle positions, player scores
Output: Visual display on screen
Logic: Generates a visual representation of the game state using appropriate graphics.

Interpretation to code.
When interpreting the concepts of the game into code, the project was divided into 5 working parts.
1. **The Clock Divider:** The 480p resolution requires a clock frequency of 25.175MHz to clock its processes. To get this frequency from the board's 50MHz clock, we could either use a PLL to generate a clock of that frequency or write an entity that takes in the board's 50MHz clock and has an architecture that "converts" it to 25MHz. Using the second option was only feasible because the half of the board's clock is almost the same as the required clock frequency. This entity was used a component in the code to generate the clock that controls the pixels, the ball and the paddles.
2. **The Image Generator:** This contains the majority of the code because although the main function of this portion of the code is to generate the images of the paddles and the ball, it also keeps count and monitors the scores. The Paddles' (same sizes) and the ball's sizes are set as generic in the entity for ease of change during generic mapping. Of course the video control signals (namely Horizontal and Vertical Sync Pulses, Visible areas etc) are port inputs into the entity while the color channels are the output. There is also an "enable" button that starts the game.
The working principles of the code is as follows:

First "process" statement.(*controlled by pixel clock , horizontal and vertical active, horizontal and vertical sync pulses. All are standard logics)*
Two counters that monitor the "cursor" both on the vertical and horizontal axes are initalized. They check whether or not the cursor has reach the ended of the either of the axes for which they are assigned to (after checking if the cursor is in an active region. A synchronous reset button is included. The function of this is that, we wish to return the paddles to their starting position when the button is pressed and this, consequently necessitates setting the counters to 0 when it is clicked.

Second "process" statement.*(controlled by paddle clock, reset, direction switch button)*

This portion monitors the paddles movements. It complements the first statement that sets the counters to zero when the reset button is pressed in the sense that, it returns the paddles to their starting position. The direction switch buttons (4 in total, implemented with switches) reverse the motion direction of the paddles. Two buttons are assigned to each paddle and they control the movement across the y-axis. The two buttons can be abstracted to "up" and "down" buttons.

Third "process" statement.

This portion of the code monitors almost everything about the ball and its movement/ motion. There are 6 directions the ball can move in when it collides with either the paddles or the frame of the game. It can move along the x-axis (both positive and negative direction), and can also move diagonally in four directions. These 6 directions were encoded with numbers 0-5.

0 - Movement from top left corner to the bottom right corner (downward diagonal)
1 - Movement from top right corner to the bottom left corner (downward diagonal)
2 - Movement from bottom right corner to the top left corner (upward diagonal)
3 - Movement from bottom left corner to the top right corner (upward diagonal)
4 - Movement along x-axis in the positive direction.
5 - Movement along x-axis in the negative direction.

The code checks for when the ball is either at the beginning (pixel 0,0) or the end (pixel 640,480) of the frame and based on the direction in which the ball is traveling, it either changes the ball direction (in cases where the ball was previously moving towards the frame) or leaves it untouched (in cases where the ball already collided with the frame and has changed direction).

It also checks for collison with the paddles using the same concept. This process is explicitly implemented in the code.

Fourth and fifth "process" statements:

This last portion of the code presents the game as a two-state Moore Finite State Machine (FSM). The state machine tracks the state of the game and monitors the boundary conditions. The score needed to win the game was set as 4. The FSM monitors this and also monitors what counts as a score. It does this by checking the side of the frame the ball passes and increment the score of the user on the opposite side (provided that the score < 4). The first state is the state where the game is idle i.e. either the game has not started or it was just completed. The second state is the state where the game is in play.

The next "process" statements takes in the current state as input and controls whether or not the ball can move.

State 0 => Move <= '0'
State 1 => Move <= '1'
("*Move" is a standard logic signal, it is neither an input nor an output)*

The last "process" statement.

This part of the code is what does probably the most important aspect of the job of this architecture. It assigns value to the color channels. It checks for the position of the paddles and ball and writes white to the locations. The background color was set to black. *(Of course, all displays are controlled by the "display enable")*

3. Score display. This component maps the score to the seven segment display.

4. **Signal generator**: This component checks the cursor to determine if it is in the visible area and generate the "display enable" signal based on the logic AND operation between the Boolean results of checking if the cursor is in the active region both in the horizontal and vertical axes.

5. The VGA control: This is the main entity of the game. Most of its entirety consists of port mapping and generic mapping for all the components described above. There is an inclusion of control of the ball's speed. Four different clock frequencies are set. This is acheived by making use of the clock divider and getting different clock frequencies from the board's 50MHz clock. The different clocks gotten from the port map are now chosen based on the "ball_speed" input.

Simulation and Testing:

Each module should was individually simulated using VHDL simulation tools to ensure proper functionality. Additionally, a full system simulation was carried out to validate the integration of all modules and to test various game scenarios.