

**GEBZE TECHNICAL UNIVERSITY**

**ELECTRONIC ENGINEERING DEPARTMENT**

ELEC 457 FPGA BASED SYSTEM DESIGN

**PONG GAME**

**PROJECT REPORT**

*Ali Sacid Karadoğan 161024086*

*Ömer Konan 171024085*

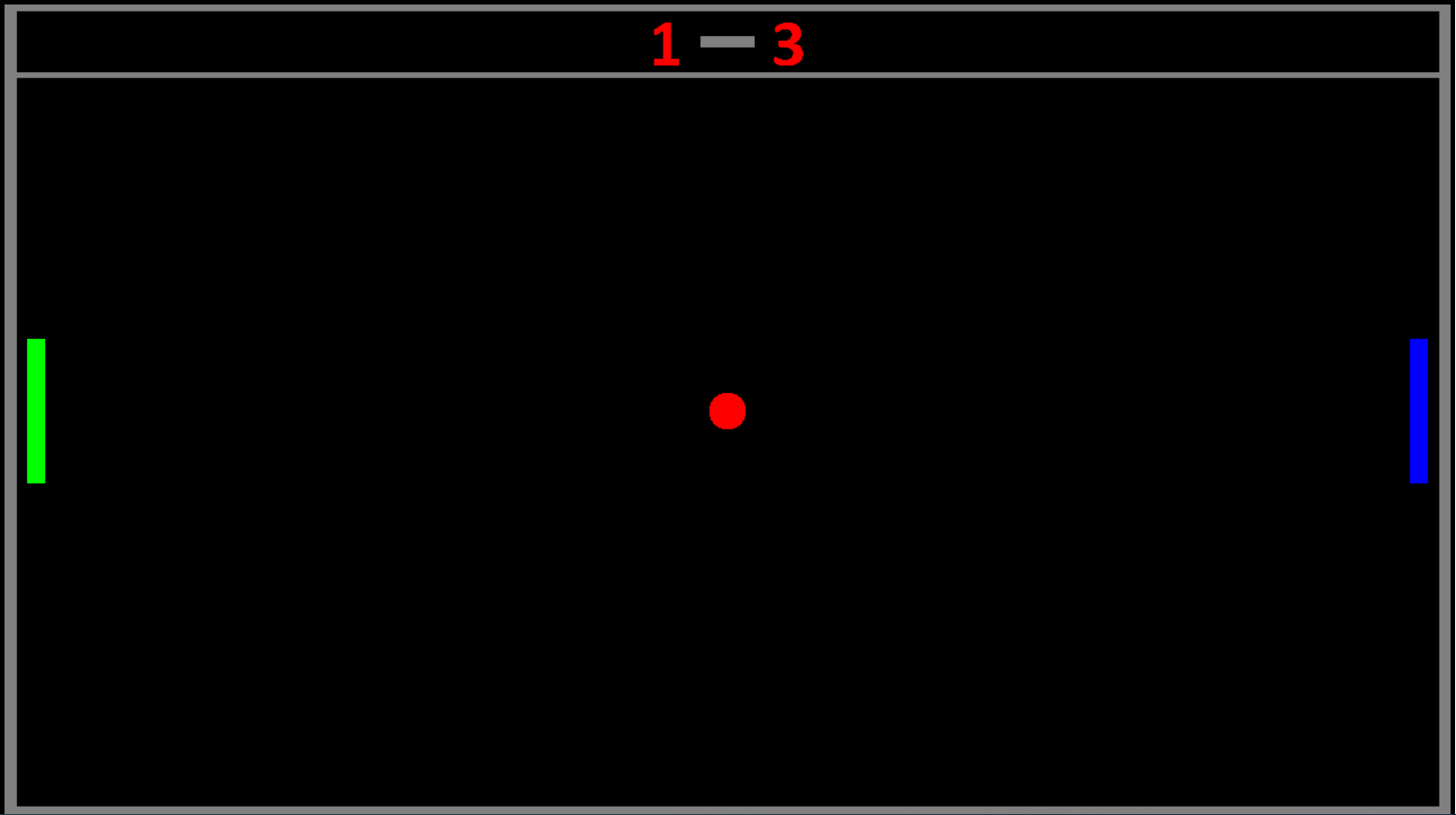
*Yusuf Hamdi Saylam 151024069*

**Introduction**

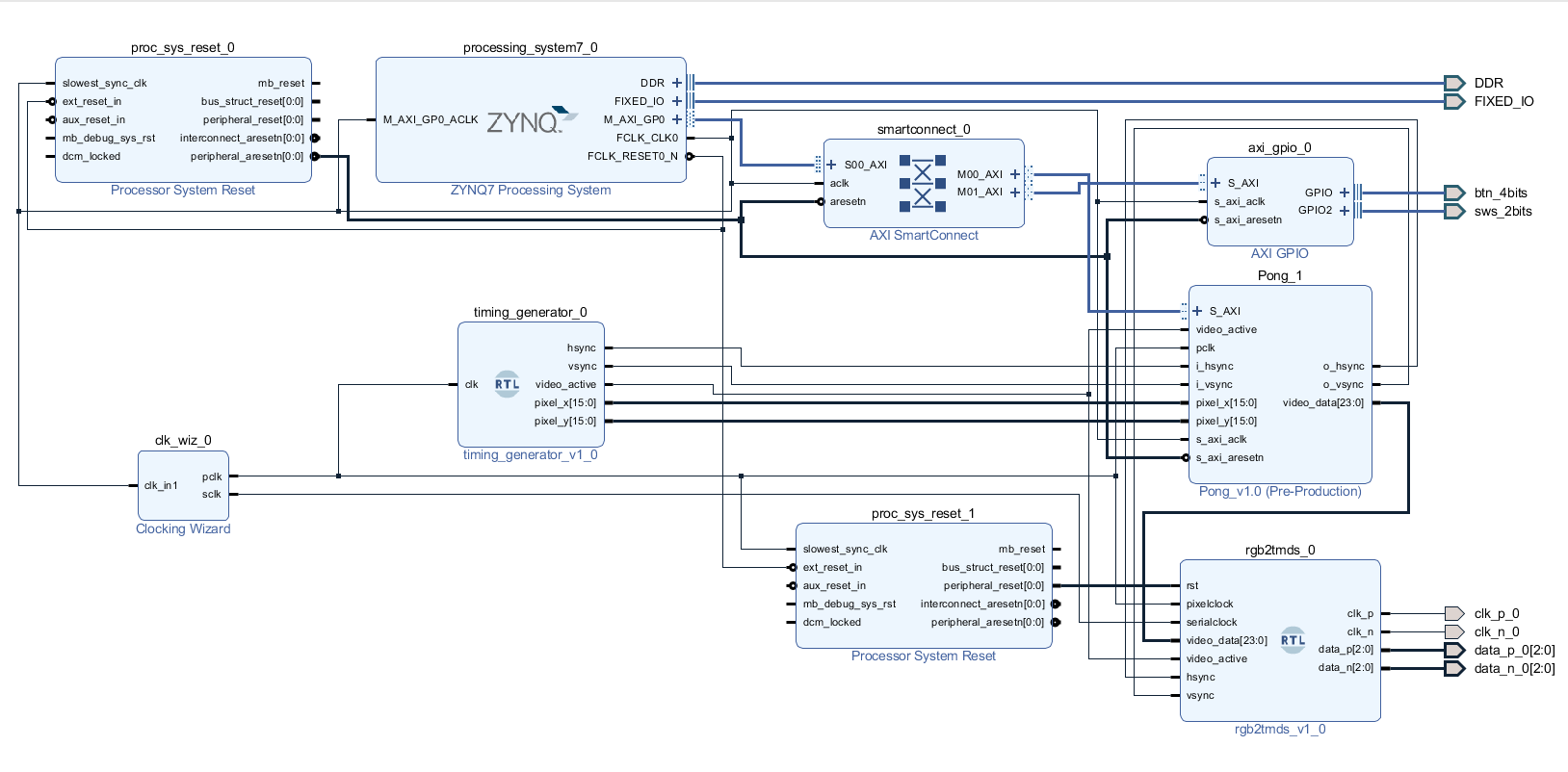
This project contains a remaking of the popular “Pong” game using Zynq-7000 model FPGA.

Pong is a table tennis sports game featuring simple two-dimensional graphics, manufactured by Atari and originally released in 1972.

In this game, there are two players, each user controls the bar located sides of the screen moving up or down to hit the ball back and score. If the ball touches the gray bar where behind its opponent, it is counted as score. Top side includes score board and if one of the players reaches 5 before the other does, the game ends and player X wins.







Block Diagram

The game gets the input signal from GPIO buttons and the C based game code decides what movement will happen to the player bars. The game also gets input signal form switches to choose the game mode. Currently there are 3 different modes in the game. The modes are, Single Player, Multiplayer and Showcase mode.

In Single Player mode;

The player uses the buttons 0 and 1 to move the Player 1 Bar up or down while the other bar controlled by the source code. Well, this is a “hard to play” mode since the source code always aligns the bar within the balls position.

In Multiplayer mode;

The player 1 uses the buttons 0 and 1 to move the Player 1 Bar up or down and the player 2 uses the buttons 2 and 3 to move the player 2 bar up or down. This is the recommended mode to play for the game.

In Showcase mode;

All player bars are controlled by the source code. As it showed in the name of this mode, this mode is for showcase of game.

**Details of Connection between C Source Code and FPGA’s VHDL Side**

C source code generates and sends the coordinate information of the movable objects to the Pong IP. Then, VHDL code obtains the coordinate information from the Pong IP and generates the objects on the screen. All objects are generated within the objectbuffer. Non moving parts put on the screen without any information from the C side except user scores. For user scores, game looks for the values of two variables in the C source code which changes during the game dynamically. Then, prints the score numbers in respect of the values of those variables. If the score is 5, “Player X Wins” appears.

**The Path the Transfer the Data**

player1\_py -> slv\_reg0 -> slv\_reg0\_s2 -> user1\_y -> user1\_y\_i -> user1\_rgb

player2\_py -> slv\_reg1-> slv\_reg1\_s2 -> user2\_y -> user2\_y\_i -> user2\_rgb

ball\_px -> slv\_reg2 -> slv\_reg2\_s2 -> ball\_x -> -----|

|---🡪 ball\_rgb

ball\_py -> slv\_reg3 -> slv\_reg3\_s2 -> ball\_y -> -----|

player1\_score -> slv\_reg4 -> slv\_reg4\_s2 -> score1-> scr1 -> usr1\_scr\_rgb

player2\_score -> slv\_reg5 -> slv\_reg5\_s2 -> score2-> scr2 -> usr2\_scr\_rgb

**Challenges**

**-**Timing: Our first implementation’s timing 19 ns. It causes broken display fault. We solved this problem by pipelining hsync, vsync, video\_active, pixel\_x, pixel\_y in objectbuffer.vhd and slv\_regs in PONG\_AXI.

-Score: While displaying scores, we used same ROM for both player1 and player2. We noticed that if we try assign same ROM both of two players, it assigned for one of them. To solve this challenge, we assigned the score variables in the same process.

-Sleep: There wasn’t any holder for C code being work way faster than we expect. Everything were moving so fast that we couldn’t even be able to see any moving object on the screen. To prevent this we added sleep function call with 2 seconds to source code. This solved the existing problem.

**Workload**

Ali Sacid Karadoğan: C source code, game mechanics, block diagram.

Ömer Konan: Simulator, VHDL(objectbuffer, PONG\_AXI), block diagram.

Yusuf Hamdi Saylam: Creating objectbuffer arrays, block diagram, report.