University of California, Los Angeles.
Computer Science & Engineering Department.
M152A : Introductory Digital Design Laboratory
Final Project : Pong Game (Using FPGA and VGA display)
Submitted to : Dr. Miodrag Potkonjak &
TA: Brinda Alluri

Submitted by: Fnu Shiva Sandesh, Brandon Tai and Junyoung Kim.
June 08, 2017.

1. *Introduction and requirement*

In this project, we chose the Pong game because Pong is one of the first computer games ever created, and it could be a reasonable project as computer science students. Normally, Pong has two paddles and a ball to let two users play together; however, in this implementation, we will only implement one paddle for the user to play against him/herself. Once the player misses the ball, then he/she loses a life, and everytime the ball bounces off the paddle, then the user gets a point. When user gets five points with the current life, then the number of balls increase to two. We implemented walls on the upper side, right side, and bottom side, and the obstacle at bottom right corner to increase the difficulty of the game.

To implement the Pong game, we were required to use an FPGA board to play the game, a VGA monitor to display, and the Verilog ISE to implement our code. We used the buttons on the FPGA board instead of using joystick because it has the same functionality to play the game. To display the graphics on the VGA monitor, we used the nexys3.ucf and NERP_demo_top.v files provided by Brinda.

The Pong game requires many mathematical calculations such as calculating the degree to bounce the ball when it hits the walls, obstacle, and paddle, the speed of the ball when user acquires more points, and the boundary of the display to properly show the graphics of the game.

2. *Design description* :

Game Tick:

The entire game runs on a tick 2^{21} times slower than the built-in 100 MHz clock. A 21-bit counter is incremented at every clock tick and a game tick occurs every time it overflows. During each game tick, the program detects collisions, updates moving object positions, and updates score.

Collisions:

All collisions occur along axes, so a vertical collision inverts the ball's y-velocity, and a horizontal collision inverts the ball's x-velocity. Collisions with the walls are detected by comparing the ball's position to a screen edge, and using a fixed wall length as an offset. Board collision uses the board's y-position variable in conjunction with the ball's (x,y)-position to detect a collision. Collision with the obstacle was a bit trickier to figure out since it needed to know which side was being collided with. The solution we ended up implementing was to break the obstacle into four one sided walls for collision calculation purposes. So, for collisions, the obstacle is treated as a 4-pixel wide boundary of the obstacle, that way if it collides with the left

"wall" of the obstacle, the program knows to bounce the ball off to the left and so on. This obstacle collision code is shown in Figure 1 below.

```
// Obstacle 1 Left Collision
if (ball_posX >= obs1_posX-8 && ball_posX < (obs1_posX + obs1_edgewidth) && ball_posY >= obs1_posY
&& ball_posY <= (obs1_posY + obs1_height))
begin
        ball_dirX = 0;
end

// Obstacle 1 Right Collision
else if (ball_posX > (obs1_posX + obs1_width - obs1_edgewidth) && ball_posX <= (obs1_posX + obs1_width)
&& ball_posY >= obs1_posY && ball_posY <= (obs1_posY + obs1_height))
begin
        ball_dirX = 1;
end

// Obstacle 1 Top Collision
else if (ball_posX >= obs1_posX && ball_posX <= (obs1_posX + obs1_width) && ball_posY >= obs1_posY-8
&& ball_posY < (obs1_posY + obs1_edgeheight))
begin
        ball_dirY = 0;
end

// Obstacle 1 Bottom Collision
else if (ball_posX >= obs1_posX && ball_posX <= (obs1_posX + obs1_width) && ball_posY > (obs1_posY +
obs1_height - obs1_edgeheight) && ball_posY <= (obs1_posY + obs1_height))
begin
        ball_dirY = 1;
end
```

Figure 1. Obstacle collision code snippet

Pausing:

To implement pausing, we used a debounced button to toggle the game tick counter from incrementing. This essentially halts all game functionality, but we would run into some problems where the game would reset into a paused state and reset the counter to 0. But, since the game tick was programmed to run when the counter overflows to 0, this would effectively cause a game tick to occur every 100 MHz at the built-in clock. So, we implemented a safety net for this case where, it would set the counter to 1 rather than 0 to avoid this from occurring.

Reset:

Since the entire game is run on variables keeping track of all current game data (ball position, ball speed, board position, score, lives, etc.), all we had to do to reset the game was reset all game variables back to their initial states at the push of a debounced reset button.

7-Segment Display:

We use 7-segment displays to display the number of remaining lives (in addition to on-screen graphic) and the current score. To do this, we created a function which takes an integer from 0 to 9 and converts it into the appropriate 8-bit display controller. Then, once we had this function, implementing the display was the trivial task of giving it its own 256 Hz clock and cycling through each digit to update.

VGA Display:

For all the static objects and graphics on the screen such as the “PONG GAME!” text, the walls, and the obstacle, displaying them was as simple as bounding their coordinates on the screen and filling them in with a different color than the background black. For dynamic objects like the board and ball, displaying was similar to the static objects, except that the variables storing their position would change with updates of the game tick. Since all the displaying used the game variables in its calculations, the graphics would update automatically to follow the logic in the background. For all the synchronization between the FPGA board and the VGA display, we used a UCF file.

3. *Simulation /Implementation Documentation*

The simulation of the Pong Game was only possible by programming the FPGA to interact with the VGA display to produce the graphics that were required during the active playing of the game. The simulation required a synchronization between the different components that were being used as the part of the game. The synchronization between the VGA output and the FPGA was most important to produce the correct pixel map for the given state of the game.

The FPGA was used for the real time mathematical calculation for position of the board, ball and obstacle. We used the UCF file to interact with the VGA along with other buttons required for various functionalities of the Pong Game. Configuring the target device resulted in a .bit file which was burnt onto the FPGA for simulating the game. The following figure depicts the position of outside walls (green), obstacle (green), ball (yellow), number of lives left (red hearts) and board (blue) when we first switched to VGA after successfully targeting the FPGA for the game.

Figure 2 : Initial Setup of the game.

However, when we first ran the code and when we tried to change the position of the board it went out of bounds of the VGA display. Therefore, we set the outside walls as parameter to ensure that it does not go out of the bounds. We faced some other issues as well, including one where whenever the ball hit the corner of the board, the score increased by twice rather than just once. The reason for this error was that ball hit two faces of the board. We fix this by adding additional constraints to the motion of the ball. By this we ensure that ball bounced after colliding with one face of the board this ensured that the above mentioned corner case would not happen. In our implementation we added an extra feature that whenever the score in the current life went past 5, the number of balls would increase to two rather than one. The following FPGA figure depicts an instance of total score so far and and number of lives left, and the second picture depicts the graphical representation of the two balls when the score is more than 5 for current life.

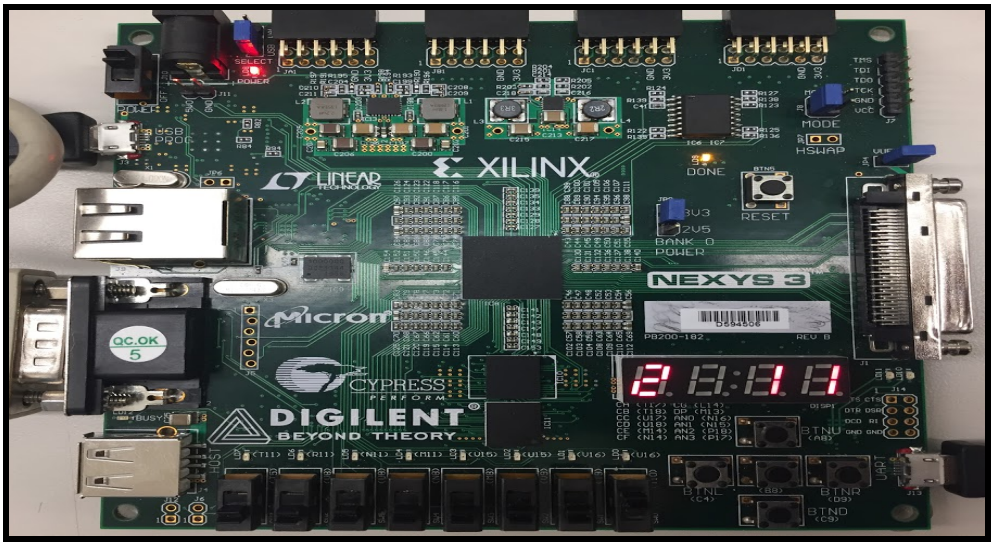

Figure 3 : FPGA showing the number of lives and total score in an iteration of game.

Figure 4: Depicting two balls (purple and yellow) and number of lives left.

While testing game with two balls we faced some problems which revealed slight errors in our logic. One such error was that when the score passed 5 in the current life we did not see the second ball because the first ball was overlapping the second. This occurred because we initialized the speed and position for the second ball to be same as the first ball thus making them completely overlap each other. We fixed this by initializing the second ball to have the opposite y-direction so it would appear to split off from the first ball, and gave it lower values of initial speed so it wouldn't be too difficult. One of the other errors that we noticed was that after the player looses all the lives the ball started shooting on a fixed path directions on its own and thus resulting in changing the score. This due the that fact the clock counter was stuck on zero, causing the game to tick with every passing cycle (See Design Description: Pausing for more information). In order to fix this we set the counter to 1 instead of 0, so when the player lost all their lives, the game would stop and end.

Since we are using buttons instead of a joystick to handle the controls of the game, handling the metastability of the buttons and debouncing them was one of the issues that we focussed on while programming the game. To avoid anomalous behaviour we used a simple counter in the signal input that waits until that button is pressed for long enough before updating that it should change the state of one or all the components in the game.

4. *Conclusion*

This project was an extremely helpful in strengthening our understanding of some the fundamental FPGA principles along with its interaction with VGA displays. We learned how to produce graphics using FPGA. On the surface, the Pong game might appear to be easy; however, there are many constraints that need to be considered to ensure the proper functionality.

The algorithm of designing a functional game was another huge benefit we have obtained from the project. Since the game requires us to implement logic such as collisions, player lives, obstacle, points, balls with varying speeds, and board movement, we had many nested loops and if else statements. We wrote over 800 lines of code to implement all this logic, and it was great practice to strengthen our understanding of programing algorithms.

CSM152A Updated- Lab Proposal, Pong Game

Fnu Shiva Sandesh, Junyoung Kim, and Brandon Tai
May 22, 2017
M152 A - Miodrag Potkonjak
TA: Brinda Alluri

1. Overview

In this project, we will implement the Pong game. Pong was one of the first computer games ever created. The Pong game imitates game features from "Ping Pong" (aka, Table Tennis). The game was first released in 1972 by Atari Corporations, originally developed by Allan Alcorn. Normally, Pong has two paddles and a ball to let two users play together; however, in this implementation, we will only implement one paddle for the user to play against him/herself. Once the player misses the ball, then he/she loses a life, and everytime the ball bounces off the paddle, then the user gets a point. We implemented walls on the upper side, right side, and bottom side, and the paddle moves up and down on the left side.

2. Design description

As we mentioned in the overview section, we will implement three wall on the upper, right and bottom side and a ball to bounce between the walls. When the user achieves a certain score, the speed of ball increases, and we are also thinking of implementing a two player game. However, the primary concern would be to get the one player game working before we move on to the two player game.

The design is divided into three sections, background, graphics during the playing and lastly the score tally. As the name indicates, in the background section we will implement the static graphics and initial conditions for the game. The second part is the dynamic graphic during the game is being played. Lastly, the score tracker will keep an active score tally for the current player.

3. Grading Rubric

1. Background/layout Functionality (20 %) : Graphics and background that will facilitate and help for the clear distinction between the different components of the game, ball, walls, obstacle and score tally. This is achieved through coloring the different components differently for clear indication. We also added additional feature to the game by generating the pixel map for the text "Pong Game" to appear on the screen.

2. Propagation Functionality (15 %) : The ball used in the pong game must stay on its trajectory unless it hits any of the surrounding walls or the obstacle. This will indicate to the user where he/she should move their board/paddle so that they do not to miss the ball and lose a life. This is challenging because it would need a constant refreshing while running the background calculation for its position.

3. Side-Wall and Obstacle Hit Functionality (15 %) : If the ball hits the surrounding walls or the obstacle then its trajectory would be changed. Keeping track of the location of the ball and then updating it based on the position of the incidence on the wall. Player would be able to see the change in trajectory as it hits the wall. The hardest part is to keep the ball from phasing through the obstacle at high speeds.

4. Board Up/ Down button functionality (15 %): We will use buttons on the FPGA to control the motion of the board. This would help the user to position the board to the point where he/she expects the ball would hit. The board can move both up and down.

5. Score Display Functionality (15 %): As the name indicates, the score-tally would display the number of times a player was able to hit the ball. Display would be on the seven-segment display on the FPGA.

6. Reset and Pause Functionality (10 %) : The reset button will allow player to start another round and reset the score. The pause button will allow the player to pause the game in the current state.

7. Number of Lives (5%) : At any time the player can know the number of lives he/she has either by looking at the heart graphic on the VGA display, or the number of lives remaining is also depicted by the first digit on the 7-segment display.

8. Ease of Use (5 %) : Ease of use is an important factor that determines the success of any game. If the user is comfortable playing it, the game is bound to do good. So ease of use is an important consideration for us.