FPGA Pong

For your happy entertainment

Matthew Michalewicz, Steven Stewart
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: mrmichal@oakland.edu, spstewar@oakland.edu

***Abstract*— This project implements the game known as Pong onto an Artix 7 FPGA on the Nexys 4 development board. The video signal is output to a VGA monitor and players interface with the game through on-board push buttons. A game is the best way to develop and exhibit mastery of machinery since it is the highest form of connection between man and machine.**

I. INTRODUCTION

The purpose of this project is for the team members to develop mastery over the FPGA by interfacing with peripheral devices. This was accomplished through a recreation of Pong, a classic video game originally released in 1972. Pong is a simple game, composed of a ball and two paddles, as well as a wall on the top and bottom of the screen for the ball to bounce against. Players lose the game by failing to hit the ball.

In order to implement the game, a VGA interface needed to be implemented as well as control via push buttons onboard the Nexys 4. The VGA controller was covered in class, but dynamic image generation had to be learned. As far as the implications of our project, it helped demonstrate the concepts that were learned and were applied in a unique manner.

II. METHODOLOGY

A. Top Level

The top level design of the project can be found below. Figure 1 shows the block that was in charge of generating and controlling all of the objects on the screen. This was the main area of the project, and is explained further in Section B.. Figure 2 shows the VGA controller that was made available by Professor Llamocca. For this code, the switches were not used as the inputs as they were in the original code. Instead, the output of the first block, RGB, was fed into as an input to the VGA controller. This allowed the colors generated from that block to be effectively displayed to the VGA screen. Also, the 3-bit RGB version of the VGA controller was utilized. Finally, the overall ASM is shown in Figure 3. This was used to handle the different rounds of the game. It is discussed more in Section C.

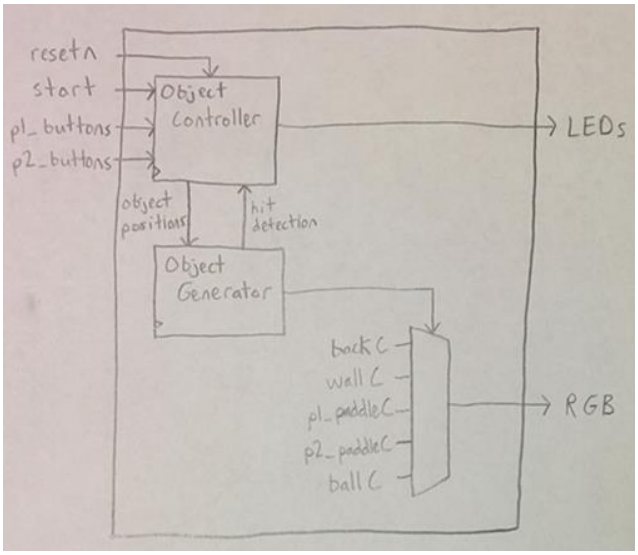

Figure 1: Top Level Module

B. Datapath Circuit

The main part of the project dealt with creating and moving the objects on the VGA screen. The VGA display and VGA controller were taken from the code that was made available by Professor Llamocca. The objects were generated by assigning constant values to the sides of each object. For example, both paddles were given a width of 10 pixels and a height of 120 pixels. Also, the distances from the sides of the screen were initially assigned for each object. Every distance was measured in relation to the top and left sides of each object. If the bottom and right sides of each object were required, the height and width were added, respectively. These distances were chosen in reference to the top and left sides of the screen, since this is the manner in which the VGA display code writes to the screen. Some of these values, like the positions of the walls and the horizontal positions of the paddles, were static and were assigned as constants. The vertical positions of the paddles and both the horizontal and vertical positions of the ball were constantly changing, so these were assigned as signals.

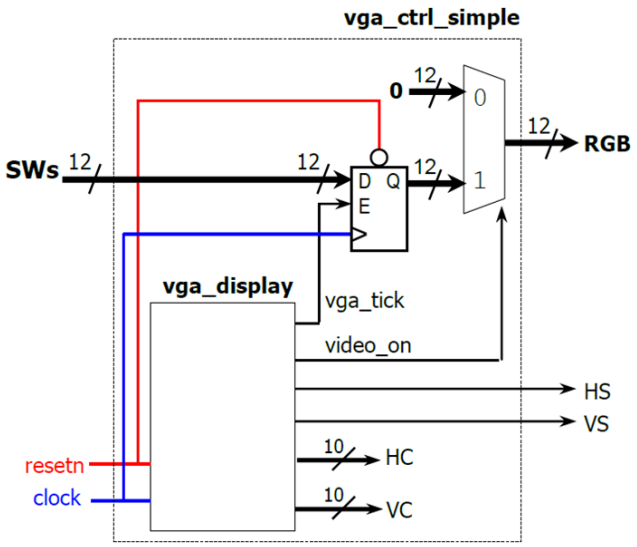

Figure 2: VGA Module

The objects were drawn to the screen by the VGA controller. This was done by having a statement that checked the current position of the object with the vertical and horizontal count of the VGA controller. When the count was in between coordinates that were assigned to an object, an "on" signal went high. There was one of these signals for every object, and they were all concatenated together with the "video_on" signal from the VGA display. This new signal was then used as a select line for a multiplexer. Each object also had its own RGB color code assigned to it, and these codes were the inputs to the multiplexer. Whenever the count reached the area on the screen where an object was supposed to be drawn, the RGB color of the object was drawn on the screen. The purpose of this multiplexer is to toggle through the different objects, depending on what "on" signal is currently high. It toggles fast enough that it looks like every object is displayed on the screen at the same time.

The paddles and the balls needed to be able to move around the screen. To do this, a constant was assigned for each object, and this was denoted as the velocity of the object. This velocity was added to the position of the object so at the next tick, the object would move as many pixels as the velocity was defined as. For example, the paddles had a velocity of 10, so they moved 10 pixels every clock tick. However, since the clock cycle was in the order of nanoseconds, one press of the of the button would almost instantly send the paddle from one end of the screen to the other, which would not be very useful. To account for this, a counter was implemented that created a signal that acted as a 60 Hz clock. Every object update occurred whenever this signal went high. For the paddle example, this allowed the paddle to move 10 pixels 60 times a second at the max. This allowed the paddle to move from one end of the screen to the other in about a second, which was reasonable for the purpose of the game.

The animation of the ball was much more challenging than the paddle, since the ball had to move both vertically and horizontally, and it was not controlled by an input like the paddles were. The vertical and horizontal speeds were equal, so that the ball constantly moved at a 45 degree angle. This occurred as long as the ball was in the bounds of the screen. The ball also needed to have its direction change whenever it came into contact with another object. This hit detection was the most challenging area of the project. To accomplish this, the coordinates of the ball were checked with the coordinates of the other objects through comparison operators. For example, if the distance from the top of the screen to the bottom of the ball was greater than or equal to the distance from the top of the screen to the top of the bottom wall, then the ball's vertical velocity was reversed so it bounced up. The detection with the paddles was more complicated because both horizontal and vertical components needed to be compared. A hit was detected when the coordinates of the top and bottom of the ball were in between the coordinates of the top and bottom of the corresponding paddle, respectively, for the vertical count. The ball also had to have its horizontal coordinates match up with the position of the side of the paddle. This would then reverse the horizontal speed of the ball, sending it in the opposite direction. However, if the ball missed the paddle then the round would end, the winner would be displayed on the LEDs, and a new round would begin once the start button was pressed.

C. Control Circuit

The ASM for the "Object Controller" FSM is as follows:

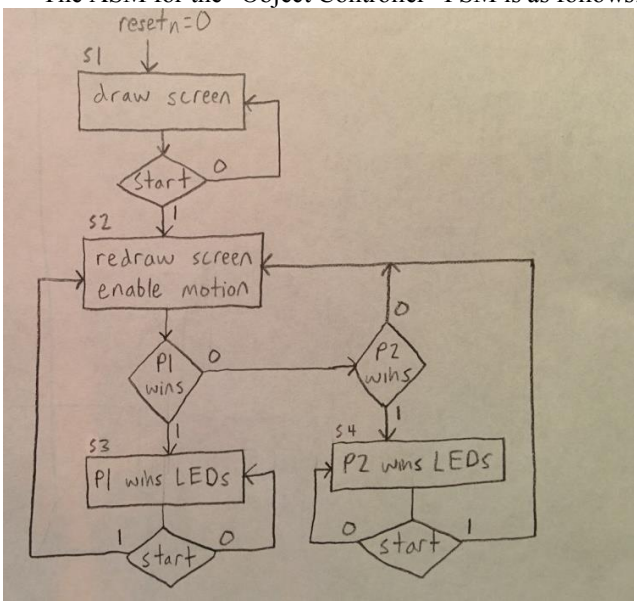

Figure 3: Main finite state machine

State 1: This is the initialization state that is entered upon resetting the circuit. During this state, neither the ball nor the paddles are able to be moved. State 2 is entered when the start button is pushed.

State 2: The game is run within this state. The ball moves and the paddles can be controlled. If no hit is detected, the screen is updated with the new positions of the ball and paddles.

States 3 and 4: These states are entered when either player scores. This illuminates either the leftmost 8 LEDs or the

rightmost 8, depending on which player scored. The game remains in this state until the start button is pressed. This allows the score LEDs to remain illuminated until the players are ready for a new round. Once the start button is pushed, the program returns to State 2 and the game begins again.

The draw screen and enable motion outputs are fed to the "Object Generator" module, which then fed information to the VGA display module. The "Object Generator" sends hit detection signals back to the "Object Controller," allowing for the state to update.

III. Experimental Setup

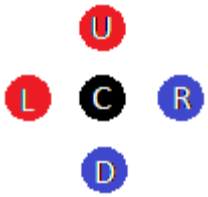

Figure 4: Control Scheme

Player 1 (Red): U to move paddle up, L to move down
Player 2 (Blue): R to move paddle up, D to move down
C to start a match

The project was verified by observing the output to the VGA monitor and testing whether the push buttons acted appropriately. Additionally, the game's response to a collision or miss between the ball and paddle was observed to ensure scoring was accurately tracked. Unfortunately, simulation test benches were not very useful for testing since the game includes both high frequency pixel updates as well as game events which happen every few billion nanoseconds. As a result, simulations were difficult to work with and provided extremely limited usefulness in system validation. Testing the FSM that kept track of score would have required additional input signals for strictly diagnostic purposes.

IV. Results

Overall, the project turned out pretty well. While it is not perfect, playing the game is still satisfying. The paddles respond exactly the way they should whenever the appropriate button is pressed. The ball responds whenever it hits another object by moving in the appropriate direction. It resets to the middle of the screen correctly and will not move unless the start button is pressed. The appropriate LEDs are lit up whenever the ball is scored as well. The reset button accurately resets the screen and the LEDS. A link to a video of the game being played is found below:

https://www.dropbox.com/s/09thoe6zeugumkm/pong_video.mp4?dl=0

There were still a few issues with the performance, though. The hit detection between the ball and the paddle was inaccurate at times. For example, there was a case that occurred at times where the ball would pass right through the top of the paddle. In this case, a collision should have been detected, but the ball passes through and scores, which is frustrating for the user. A link to a video of this occurrence is found below:

https://www.dropbox.com/s/3k5yb1f94keazg7/hit_detection_issue.MOV?dl=0

The only major possible source of issues was brought to attention by Professor Llamocca. Since the objects needed to respond in a manner that was slower than the initial clock speed, a divider was used to slow it down for the object animations. However, the conditions were all set to occur when the clock signal was high, and the event condition was not included. This causes what is known as a latch, and can cause numerous issues in VHDL. It is likely that this had at least some effect on all of the issues that were encountered. For the most part, this error was able to be avoided in the final project to produce a working game.

Conclusions

There are a lot of improvements that could be made to this project. The obvious choice would be to address the issues that still lie in the code, such as the latch issue, and improve the overall reliability of the program. Score keeping for multiple rounds was another goal that was desired, but never reached. If this was achieved, a max score could be set and a winner could be determined, and some sort of victory screen could accompany it to make the experience more satisfying. Sound could also be added for the bounces of the ball to add more immersion to the game. Ultimately the project was a success, despite the issues that were had. Perhaps if the latching issue had been discovered earlier, more features could have been implemented instead of troubleshooting intermittent errors that made no sense. For instance, score keeping functionality was coded, but was never successfully implemented. Regardless, the project was a great learning experience. Even though the final product was not able to utilize all of the functionality that coded, the project ended in a respectable state with the core functionality of Pong.

References

[1] G. O'Brien, "VHDL VGA Pong," http://www.digital-circuitry.com/VHDL_VGA_PONG.htm