CS M152A Lab 4
Awesome Pong

HUAN-SONY NGUYEN

Ryan Trihernawan (904063131)
Walter Qian (204301927)

Introduction

For our Lab 4 we decided to create a 2-player Pong game. Our Pong game uses the VGA display. Pong is a 2-D game that features two paddles on opposing sides and a ball. Each player controls an in-game paddle that they can move vertically. The players will use the buttons on the FPGA board to control the in-game paddles. A point is scored when the ball is hit passed the opposing player's paddle. The score is displayed at the top center of the screen and the goal of the game is to score as many points as possible. The center button on the FPGA board can be used to reset the game as well. The switches on the board represent the level. Increasing the level increases the speed of the ball. Finally we added some obstacles around the center of the stage that reflect the ball to create more of a challenge.

FPGA Board	Effect
Up Button	Player 1 moves up.
Left Button	Player 1 moves down.
Right Button	Player 2 moves up.
Down Button	Player 2 moves down.
Center Button	Resets the score and positions of ball and paddle.
Switches	Changes the speed of the ball. Leftmost switch is the fastest speed.

Table 1: Controls of our Pong game

Design Description

The overall circuit is composed of four modules and takes fourteen inputs. The fourteen inputs include the five buttons, the eight switches and one clock. The first module is the ball clock module and it is very similar to the clock module in lab 3. The ball clock module takes the clock, the reset button and the switches as input. It then outputs another clock depending on the current switch activated. The second module is the 50Mhz clock module. It takes the master clock as an input and outputs the 50Mhz clock, which is fed to the VGA controller, our third module. The VGA controller is responsible for managing hsync and vsync, a pair of signals to reset our current position horizontally and vertically. Our horizontal counter is based on the 50Mhz clock and the vertical counter counts once for each full horizontal line. This module also outputs pixel_x and pixel_y, which is the location of the pixel being updated. It has one more output, vidon, which simply confirms that the game is being played and the display should be updated. Our fourth and final module is the game logic module. It takes two clocks, the master clock and the ball clock, the five buttons, the eight switches, and the pixel_x and pixel_y as inputs. Its first purpose is the move the ball forward based on its current angle. It is also responsible for detecting collisions between the players, obstacles and the ball. It determines the new angle of the ball if a collision occurs. It has three outputs for the red and green pixels, and two outputs for the blue pixels for the VGA display. Below is a detailed schematic of the circuit. Each module will be expanded on below.

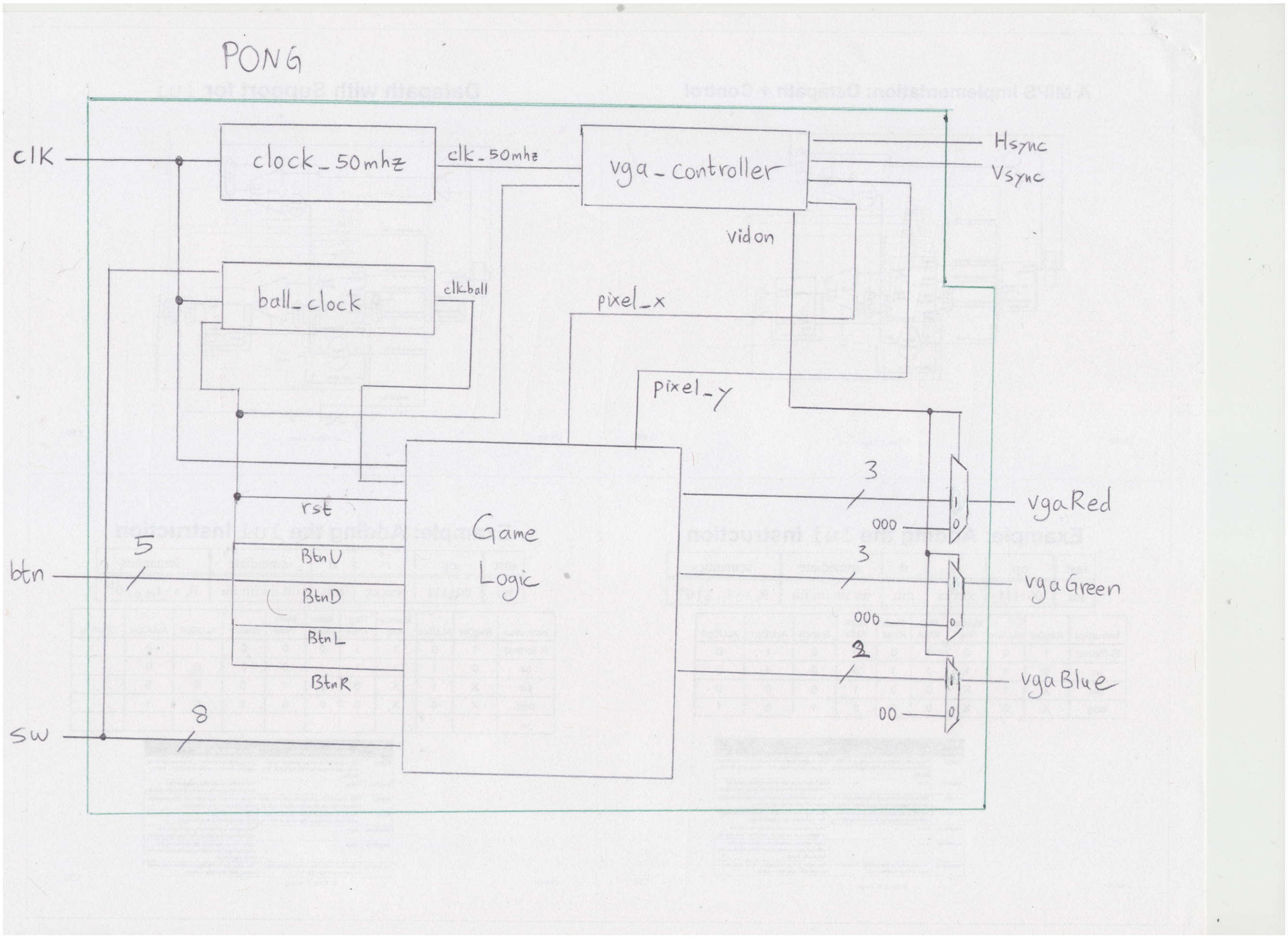

Image 1: Circuit schematic

The first module, ball clock, outputs a clock for the current speed of the ball. In order to get this clock it takes in the master 100Mhz clock along with the eight switches as an input. The module creates eight clocks with varying speeds, using the same technique used in Lab 3. We once again used ModNCounter, which took our 100Mhz master clock and divided it by a divisor to create a slower clock. In this project our divisors ranged from 50,000 to 1,000,000. This creates clocks that range in speed from 50hz to 1,000Hz. Each clock corresponds to a switch, with the fastest state being the leftmost switch being activated and the slowest state corresponding to no switches activated. If multiple switches are activated we simply take the leftmost (fastest) one.

Our second module is the 50Mhz clock module. It is the simplest module in our circuit. It takes the 100Mhz master clock and converts it to a 50Mhz clock by registering every other tick. The output, a 50Mhz clock, is passed onto our VGA controller.

The VGA controller is our third module and takes the 50Mhz clock along with a reset button as input. To display VGA on a screen an electron beam moves across the screen horizontally and vertically. At every location the Red, Green and Blue values are specified and the entire screen is displayed using this technique. The goal of the VGA controller is to control the horizontal sync and vertical sync, which are used to reset the beam horizontally and vertically. If the beam was not reset it would get stuck on the bottom right corner of the screen. To manage the hsync and vsync we need a pair of counters to count the number of ticks. The horizontal counter is in sync with our 50Mhz clock and the vertical counter is incremented for every full horizontal line. We use an enabler, activated every full horizontal line, to help us, as the vertical counter can only be changed when the enabler is set to one. We also have a buffer from when the hsync or vsync is activated to when we can start painting pixels on the screen again. The buffer is called porch time and we have front porch time for before the timing pulse and back porch time for after the pulse. This time allows the beam to travel the length of the screen and get back in position. Every time our horizontal or vertical counter hits the maximum we call hsync and vsync and reset the counters. If both counters are within the porch time range so they are not traveling, then we report back the pixel position of the electron beam. Below is a diagram displaying the uses of the porch times. When the pulse is synced there is a delay before pixels are displayed again.

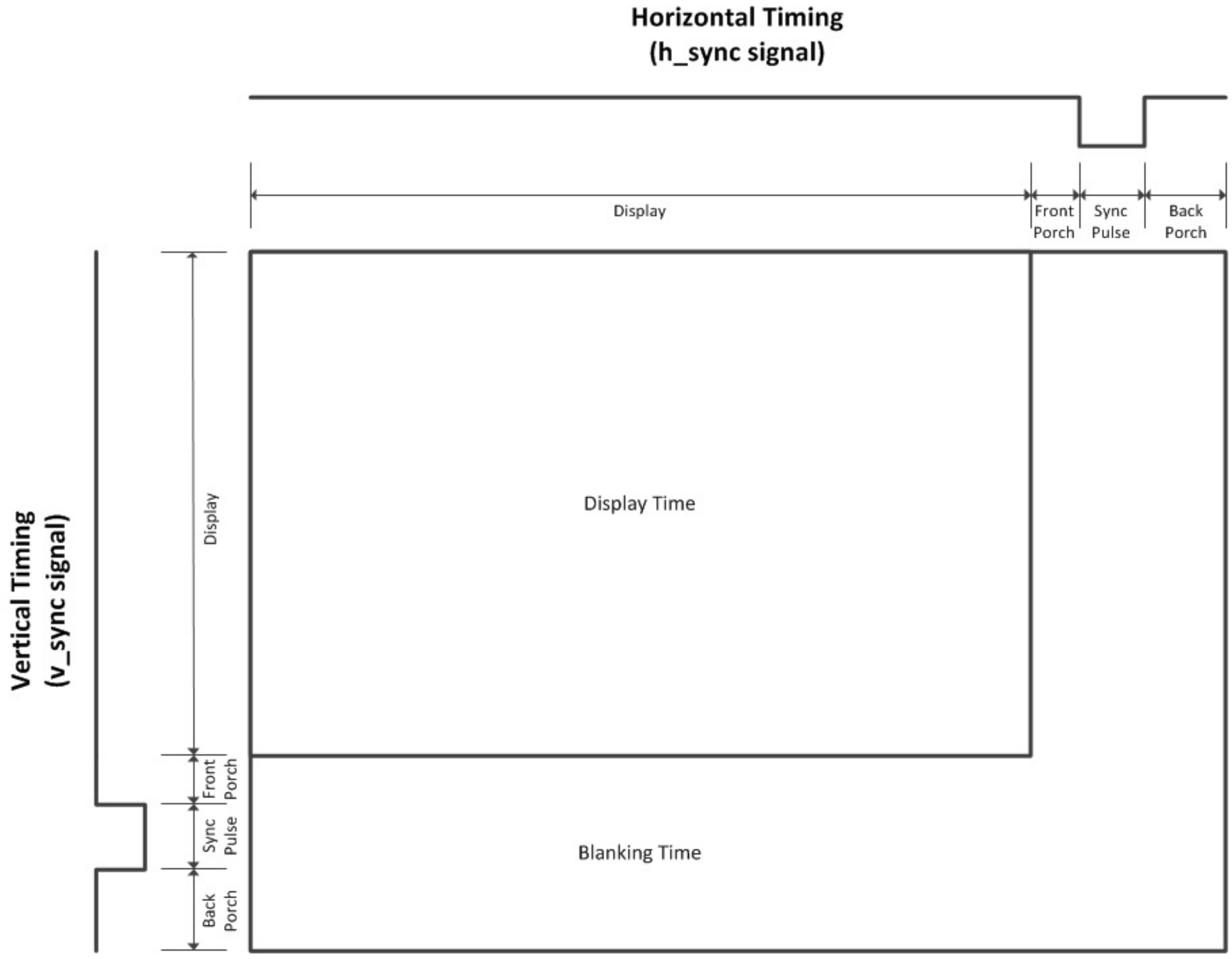

Image 2: VGA signal timing diagram.

The final module is the game logic module and its goal is to update the Red, Green and Blue values of the VGA based on the inputs. It takes all five buttons, eight switches, the x and y positions of the current pixel and two clocks as inputs. First the module checks if the game is reset or if a player has been scored on and if so reassigns the positions of the paddles and ball accordingly. Then it checks for movement of the two paddles from the button inputs. It then updates the position of the ball based on the ball clock. After the positions of the ball and paddles are updated, it checks the current pixel_x and pixel_y input from the VGA controller. If the current pixel location happens to be a paddle or ball the

Red, Green and Blue values are displayed in accordance. It also checks for detections at this point between the ball and any paddles or obstacles or walls. Our logic for bouncing off walls was to simply reflect the ball using the angle of incidence. For the paddle and obstacles the logic was a bit more complicated. When the ball collides with a paddle or obstacle its initial direction is completely disregarded. Its new angle depends solely on how far from the center of the paddle it is located. If it hits the center it is sent directly straight and if it hits on the edge it is sent off an extreme angle (45 degrees). We have 3 potential angles upwards and downwards and they are in increments of 15 degrees. So the ball can fly 0, 15, 30, or 45 degrees in either direction depending on where on the paddle the ball lands. The master clock is used for detecting collisions so it is faster than the ball clock and thus, no glitches occur. If the pixel is in the correct location, the VGA display is properly updated. We use an 8-bit array to simulate the combined RGB values.

Simulation Documentation

We tested the game extensively. Most of our debugging was done by playing the game and finding errors. However we did some special testing to ensure all possible scenarios were accounted for. The first aspect we tested was ball speed from the ball_clock module. We did this by testing all the switches to see if the speeds worked. Multiple switches were tested as well to ensure the fastest speed was enabled. Below is the table of selected tests.

Switches Activated	Expected	Result	Correct?
None	550Hz	50Hz	Yes
1	63Hz	63Hz	Yes
7	1000Hz	1000Hz	Yes
3,5	250Hz	250Hz	Yes
1,2,3,4,5,6	500Hz	500Hz	Yes

Table 2: Ball Clock module tests

Next we decided to test the button inputs. All five buttons were tested at different points to see if they worked as intended. Below is the table. We also confirmed the paddles did not move past the walls.

Button	Expected	Result	Correct?
Up	Player 1 moves up	Player 1 moves up	Yes
Left	Player 1 moves down	Player 1 moves down	Yes
Right	Player 2 moves up	Player 2 moves up	Yes
Down	Player 2 moves down	Player 2 moves down	Yes
Center	Reset	Reset	Yes

Table 3: Button effect tests.

Next we tested the angle of reflection of the ball on the paddle and obstacles. The overall height of the paddle is 61 pixels. The center pixel is a perfect horizontal angle. From then every ten pixels is 15 degrees angle. Below is a table showing our tests.

Location on Paddle	Expected	Result	Correct?
55 Pixels	45 Degrees upward	45 Degrees upward	Yes
45 Pixels	30 Degrees upward	30 Degrees upward	Yes
35 Pixels	15 Degrees upward	15 Degrees upward	Yes
31 Pixels	0 Degrees	0 Degrees	Yes
25 Pixels	15 Degrees downward	15 Degrees downward	Yes
15 Pixels	30 Degrees downward	30 Degrees downward	Yes
5 Pixels	45 Degrees downward	45 Degrees downward	Yes

Table 4: Ball collision on Paddle and obstacles

For testing the angle of the ball and the scoring we simply played the game and saw the VGA display update itself. To our knowledge everything worked as intended.

Conclusion

This final lab let us explore the possible applications of the FPGA and other modules such as ROM, HID, and VGA. We decided to use the VGA to create a Pong game, which seemed challenging due to our inexperience with VGA and designing a fully functional 2-D game. The biggest challenge was displaying the pixels in a VGA monitor. We read a myriad of online tutorials on VGA to understand the inner workings of a VGA and essential parameters to set up a VGA including back porch, front porch, horizontal frequency, sync pulse length, and many others. Implementing the VGA controller module took the most time because we had to tinker with different values for the aforementioned parameters to finally get the FPGA to successfully display some pixels on the VGA monitor. The other major hurdle was implementing a smooth movement of the paddle pixels when a user presses the FPGA buttons. Initially we implemented the button using a clock divider similar to an implementation in lab 1. However, we soon realized that the paddle pixels were not moving smoothly; instead they looked like they were blinking as they moved vertically. After testing different frequencies of the clock dividers, we experimented with using a counter instead of a clock divider. The difference lies in how long the output is asserted to 1. A clock divider asserts its output to 1 for several ticks depending on the frequency, however a counter always asserts its output to 1 for only one tick when it resets it count from its maximum value to 0. This implementation allows for a smooth movement of the paddle pixels even when a user presses and holds the buttons for a few seconds or longer because the counter acts as a debouncer that only recognizes the button signals when it resets. Lab 4 really pushed us to the limit in terms of our understanding of the FPGA and modules on the Nexys3 board as well as our persistence in successfully implementing a fully functional game.