FPGA Pong Via VGA

Anand Jasti
College of Engineering
California State University, Long Beach
Long Beach, United States of America
Anand.Jasti@student.csulb.edu

Abhishek Jasti
College of Engineering
California State University, Long Beach
Long Beach, United States of America
Abhishek.Jasti@student.csulb.edu

***Abstract*— The purpose of this project is to implement an FPGA pong game that utilizes the Digilent Nexys A7-100T's on-board VGA port to display the contents of the game on a monitor, as well as use the right two switches to control the movement of player one's paddle and the left two switches to control the movement of player two's paddle, and the seven-segment display to keep track of each player's score. The player's scored are displayed respectively to their seven-segment display, the player on the right side of the screen has their score displayed on the right side of the seven-segment display and the player on the left side of the screen has their score displayed on the left side of the seven-segment display. The ball is always initialized to the middle of the display screen and moves in a random direction. The ball resets to the center when a player scores. The paddles are initialized in the center left and center right of the screen respectively for player one and player two. Each player has two switches to control their paddle's direction, one switch controls the direction up and the other switch controls the direction down. The game should run indefinitely until one of the players reaches three points.**

Keywords---Seven Segment Display, VGA, Switches

I. INTRODUCTION

This project of FPGA Pong game via VGA combines modules from all areas of learning thus far. Pong game via VGA combines both sequential and combinational circuits into one design. The pong game is displayed at 800x600 resolution at 75 Hz. This display is done by using a vertical and horizontal sync generator and a basic clock divider to fit the VGA standard. The Pong game outputs the score to the seven-segment array module which multiplexes the individual displays to show multiple digits in decimal form. The game is controlled by the two most right and the two most left switches with a debounce function to prevent unintentional misinputs.

II. BACKGROUND & PRELIMINARIES

A. Background

In the process of creating this project, our intention was to demonstrate the use of the VGA port on the Nexys A7-100T board and the FPGA as a display adapter/graphics processor to play a simple game of pong, and additionally keep score on the onboard seven-segment display

B. Preliminary

Our research concluded that our Pong game was very similar to the one-player pong game [1] in the textbook. Instead of using the on-board buttons we used the onboard switches to control the player's paddle for our pong game, and instead of one-player, we made it a two-player pong game. We are also using the two seven-segment displays on our Nexys A7-100T board to display the player's score.

Another research that helped implement our project was a sample project report [2] given to us by the professor Amin Rezaei. FPGA Snake Via VGA has very similar implementations to our FPGA Pong Via VGA. The use of VGA in this case however has much better resolution. Instead of using the 640x480 as done is FPGA Snake Via VGA, we realized that our Nexys A7-100T board can implement a much better resolution. Our game displays at 800x600 resolution. The use of our seven-segment display also has a very similar implementation to FPGA Snake Via VGA.

C. GAME RULES

Pong is a two-dimensional sports game that simulates table tennis. The player controls an in-game paddle by moving it vertically across the left or right side of the screen. They can compete against another player controlling a second paddle on the opposing side. Players use the paddles to hit a ball back and forth. The goal is for each player to reach three points before the opponent; points are earned when one fails to return the ball to the other.

III. IMPEMENTATION

A. CLOCK DIVIDER

This clock divider module takes in two inputs (clk, and div_value) and outputs one output(div_clock). At the input clk(100MHz) the module increments a counter until reaching the div_value and toggles the output. The clock divider module is being used in two different modules. It is being used by the VGA Synchronization Generator to generate a 50 MHz pixel clock as per the requirements specified by IBM Supported resolution timing charts [4], for the target resolution: 800x600 pixels resolution with 75 Hz vertical refresh rate and a 50 MHz pixel clock frequency. The clock divider module is also being used in the Seven-Segment Display module to generate a 10 KHz clock which allows us to properly display the digits using a multiplexer. Figure 1 shows the simulation of the clock divider module generating a 50 MHz clock used for the VGA diaplay from a 100 MHz clock. Figure 2 shows the simulation of the clock divider module generating a 10 KHz clock used for the seven-segment display from a 100 MHz clock.

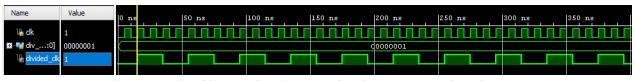

Figure 1: Simulation of 50 MHz clock

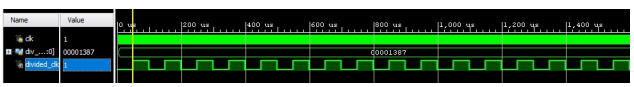

Figure 2: Simulation of 10 KHz clock

B. VGA SYNCHRONIZATION GENERATOR

This module is responsible for generating vertical and horizontal synchronizing signals, which are needed to use the VGA port on the Nexys A7-100T properly. We closely followed a YouTube video uploaded by Simply Embedded named *How to Create VGA Controller in Verilog on FPGA? | Xilinx FPGA Programming Tutorials [3],* and the example given by Pong Chu in Chapter 13 of his book *FPGA Prototyping by Verilog Examples: Xilinx Spartan-3 Edition [1],* to create this module. In the video Simply Embedded is implementing a VGA controller for a 640x408 display monitor at 60 Hz, we made modifications in our implementation by creating a VGA controller for an 800x600 display monitor at 75 Hz. We also implemented the pixel_x and pixel_y from the example given by Pong Chu. The system clock frequency of the Nexys A7-100T is 100 MHz, so we implemented the clock divider mentioned before to create a VGA clock at 50 MHz for the pixel clock. We chose to implement a VGA controller for 800x600 @75 Hz which is one of the supported resolutions for TFT/LCD display units put forth by IBM [4], because the pixel clock needed for this resolution was achievable.

Timing name	800 x 600 @ 60 Hz		800 x 600 @ 75 Hz	
Horizontal frequency and polarity	37.879 kHz	Positive	46.875 kHz	Positive
Vertical frequency and polarity	60.317 Hz	Positive	75 Hz	Positive
Pixel clock	40 MHz		49.5 MHz	
Scan type	Noninterlaced		Noninterlaced	
Horizontal				
Period	26.400 µs	1056 pixels	21.333 µs	1056 pixels
Display	20.000 µs	800 pixels	16.162 µs	800 pixels
Blanking	6.400 µs	256 pixels	5.172 µs	256 pixels
Sync	3.200 µs	128 pixels	1.616 µs	80 pixels
Back porch	2.200 µs	88 pixels	3.232 µs	160 pixels
Front porch	1.000 µs	40 pixels	0.323 µs	16 pixels
Vertical				
Total	16.579 ms	628 lines	13.333 ms	625 lines
Display	15.840 ms	600 lines	12.800 ms	600 lines
Blanking	0.739 ms	28 lines	0.533 ms	25 lines
Sync	0.106 ms	4 lines	0.064 ms	3 lines
Back porch	0.607 ms	23 lines	0.448 ms	21 lines
Front porch	0.026 ms	1 line	0.021 ms	1 line

Figure 3: IBM Supported Resolution Timing Chart for 800 x 600 @60 Hz and @75Hz

Since we chose to implement a clock divider, we can only divide the 100 MHz clock by integers. If we chose to implement the clock divider a different way, we would have been able to divide the 100 MHz clock by non-integers. Since we can only divide the clock by integers and we wanted to maximize the resolution, we looked at the Supported resolution timing charts provided by IBM [4] and chose the closest resolution that met our unique situation. The Timing chart shows that the pixel clock needed 49.5 MHz for 800x600 @75Hz resolution. As mentioned before this pixel clock frequency is not achievable by our clock divider module, the closest achievable frequency by our clock divider was 50 MHz, so we decided to test our VGA controller to see if it would work at a slightly different frequency. We knew that it would work because there was only a .5 MHz difference between what our clock divider produced and what was recommended, and indeed just as we thought this worked providing us with a picture on our monitor.

The Nexys A7 series of FPGAs are only capable of outputting 4-bits each on the Red, Green, and Blue channels. Allowing us to only use 12-bit color on our FPGAs as seen in Figure 4 [7]. Digilent uses voltage dividers in parallel to generate 4-bit color instead of a DAC, which explains the presence of the resistors on each line from the 3 RGB pins.

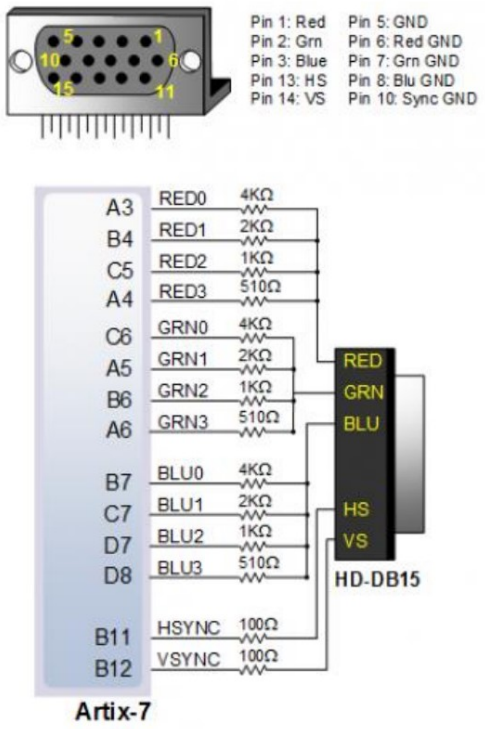

Figure 4: Nexys A7 VGA Interface [7]

C. PONG GRAPHICS

This module takes in the input from the players and outputs an RGB signal. Pong graphics module utilizes the x and y coordinates of the current pixel being drawn by the VGA Generator and determines the RGB value. The boundaries are drawn in fixed locations however the ball and the left and right paddles may vary depending on the switch input. The ball has its location placed initially at the center of the screen, the left and right paddles have their location placed initially at the center left and center right respectively as shown in Figure 5. For this module we took a lot of inspiration from the textbook [1]. In the textbook the implementation of this module includes bitmapping, but we decided not to implement bitmapping to simplify the complexity of this module.

Figure 5: Visual Scenario for pong game

Figure 6: Pong ball and player paddle in motion

D. SEVEN-SEGMENT DISPLAY

This module allows us to display each players score to the 7-segment display on the board. Player one's score is placed in the right most digit of the right 7-segment display and Player two's score is placed in the right most digit of the left 7-segment display. The display is driven by 100KHz clock, this clock frequency is achieved by using the clock divider module. This allows us to properly allow the digits to multiplex and iterate through the individual digits of the input number right to left. Only one digit of the displayed number is illuminated at a time, however the rate at which the module switches between the digits of the input number is fast enough that the human eye perceives it as a static image. The module outputs an 8-bit cathode signal which is tied to the individual segments of the digit and the 8-bit anode signal which is tied to the individual digit itself. Both the anodes and cathodes are active when "low."

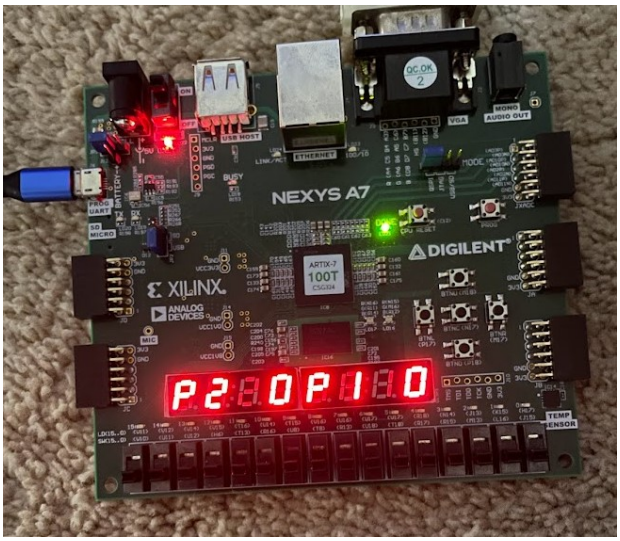

Figure 7: Picture of the seven-segment display

E. RANDOM NUMBER GENERATOR

For this module, we took inspiration from the Worcester Polytechnical Institute's ECE 4514 Digital Design II Lecture 6: A Random Number Generator to make a Linear Feedback Shift Register (LFSR) generate pseudorandom numbers [6]. We used one five-bit LFSR module to provide random direction for the ball. The five-bit Linear Feedback Shift Register (LFSR) provides a pseudo-random number between 0 and 1. If the random number is 1 the ball goes toward right player, if the random number is 0 the ball goes toward the left player. A simulation of the pseudo-random output is visible on Figure 8 below with a repeat period of 15 clock cycles.

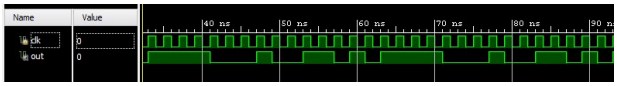

Figure 8: Simulation of Random Number Generator module

F. DEBOUNCING

This module is used for user inputs like the left/right player up and down switch, and the reset switch. This module takes the input from mechanical switches and cleans out the input. When a switch is flipped up/down on FPGA there are many unexpected up and down bounces in the switch signal as shown in Figure 9. This module will clean out the glitches on the input wire. To clean up the glitches from the mechanical switches we implemented a 5ms delay so that when the input switch becomes stable, we assign the input to the output. This module is very important because without this module we would see unexpected behavior while playing the pong game.

Figure 9: Simulation of debounce module

G. TOP MODULE

Top module is used to instantiate all other modules and provide a basis for the game. This includes the setting, updating, and resetting of the VGA output, seven-segment display, and the RGB value out of the pong graphics module which is synchronized between the pixel clock and system clock before it is outputted to the display. Check Figure 10 for the hierarchical structure of the pong game in Verilog.

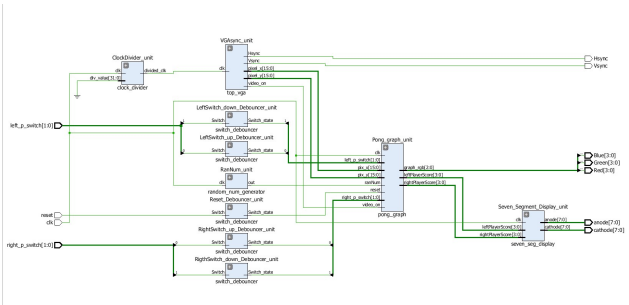

Figure 10: Design Diagram

IV. EVALUTION

Several iterations were required to achieve the current functionality of the game. We did almost everything we set out to do in this project. The only thing we didn't implement in this project was the incrementing of the score

for the left and the right player, which led to us not implementing the end of the game. We spent hours on this one issue of incrementing the score of the players, and the best thing we could come up with is a buggy seven segment display and a working pong game without any score. Proper timing between modules and the graphics being displayed was essential, if we didn't have proper timing between modules would lead to a variety of errors or even preventing the monitor from detecting the board. Initially, the VGA synchronizations was tested and verified within the Vivado simulator, the RGB values were set to the white color while testing the VGA synchronization module. Figures 11, 12, 13, 14, and 15 shows the simulations we used to verify the VGA synchronization module.

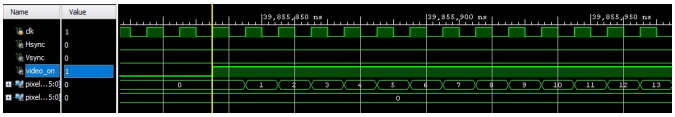

Figure 11: Simulation of VGA Generator – Vertical Sync Front Porch

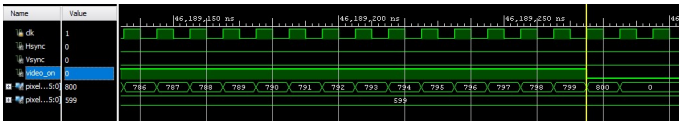

Figure 12: Simulation of VGA Generator – Vertical Sync Back Porch

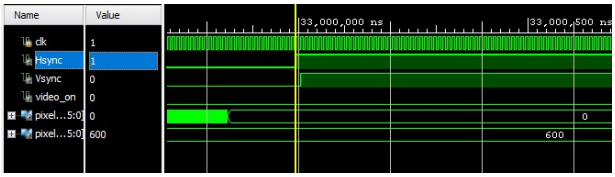

Figure 13: Simulation of VGA Generator – Horizontal Sync Front Porch

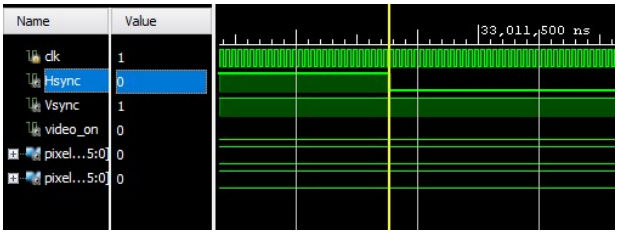

Figure 14: Simulation of VGA Generator – Horizontal Sync Back Porch

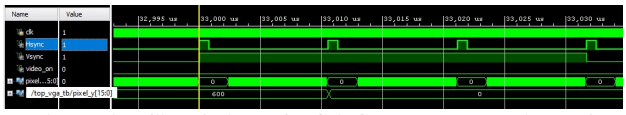

Figure 15: Simulation of VGA Generator – Horizontal Sync Finish

The complexity for pong graphics module quickly ramped up as more characteristic features were implemented. The left and right player score would not properly increment which would display unexpected values on the seven-segment display. This problem was never solved because of the time constraint of the project. The most difficult component by far would be implementing the end of the game, this would produces errors like timing loops, so we decided to not implement the end of the game. The current implementations are satisfactory, despite the difficulties encountered during development. Figures 16, 17 shows pictures of the running board and the setup we used to emulate our pong game.

Figure 16: Picture of the running board

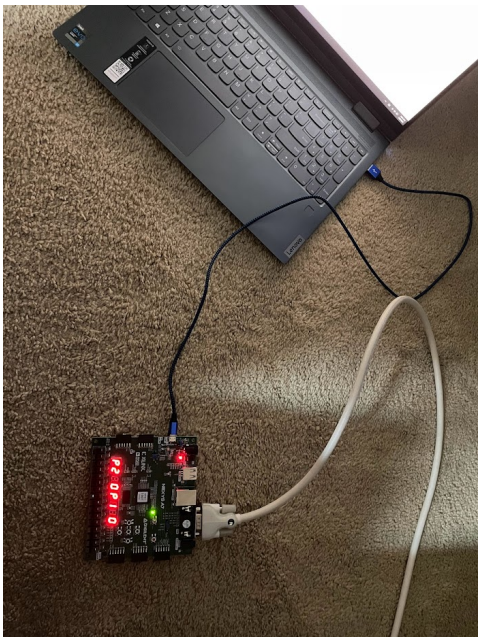

Figure 17: Picture of our setup

V. Conclusion

The complexity of implementing all of the desired features for the pong game was severely underestimated. Many iterations and several hours were required to achieve the current level of functionality of the design. Research into the various concepts of the pong game, and referencing other pong game implemented by other people made it possible to achieve the current level of functionality. Cooperating and relying on each other to implement the various modules required for the pong game required thought and with the time-induced anxiety of the project's deadline approaching was very hard , but we managed to implement a somewhat working pong game.

REFERENCES

[1] P. P. Chu, "VGA Controller I: Graphic" in FPGA Prototyping by Verilog Examples Xilinx Spartan-3 Version. Hoboken, NJ: Wiley, 2008, ch. 13, pp. 309-340.

[2] Mariam Bekhit, Pi Oliver, Justin Salazar, "FPGA Snake Via VGA" California State University Long Beach, Fall 2021, CECS 361.

[3] How to Create VGA Controller in Verilog on FPGA? | Xilinx FPGA Programming Tutorials. (2018, November 29). YouTube. https://www.youtube.com/watch?v=4enWoVHCykI

[4] *Supported resolution timing charts*. (n.d.). © Copyright IBM Corporation 2015. Retrieved March 23, 2022, from https://www.ibm.com/docs/en/power8?topic=display-supported-resolution-timing-charts

[5] J. P. Nicole, Pong Game, fpga4fun [Online]. Available: https://www.fpga4fun.com/PongGame.html. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[6] P. Schaumont, ECE 4514 Digital Design II Lecture 6: A Random Number Generator in Verilog, Worcester Polytechnical Institute [Online]. Available: http://rdsl.csit-sun.pub.ro/docs/PROIECTARE%20cu%20FPGA%20CURS/lecture6[1].pdf

[7] Digilent, Nexys A7 Reference Manual, Digilent [Online]. Available: https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual