Sparsh Gauba, Max Lutton

204600605, 604493477

**CS 152A Lab 4: Creative Project**

**Introduction:**

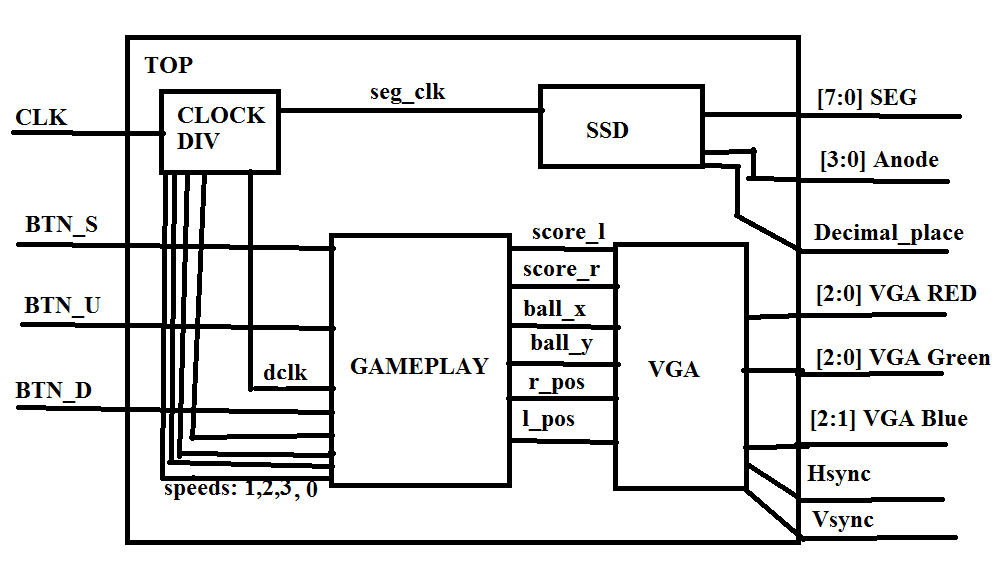
For the capstone project of this course, we were given full creative authority to create whatever we wanted, leveraging all of the knowledge we’ve accumulated throughout the quarter with the FPGA board. We chose to recreate the classic arcade game, Pong. In addition to buttons and creating our own multi-module hierarchical design, we also incorporated VGA into this project, which is something we had not used before.

Our version of Pong is similar to the original. The player competes in a game of virtual ping pong against the computer, controlled with our own AI algorithm. The player and the computer compete by deflecting a ball back and forth across the screen using their paddles. A competitor scores a point when the ball gets behind the other’s paddle, as the paddles can move only in the vertical axis. The score is displayed at the top of the screen, and the winner is decided once either the human or the machine scores 11 points. The game has 3 levels of difficulty, with the AI’s speed increasing at each level. The player’s paddle is green, and the computer’s is red.

The game uses 3 buttons on the Nexys3 FPGA board: up, down, and right. The top and bottom buttons control the motion of the user’s paddle and the right button is used to reset the game. When the top button is pressed, the user’s paddle moves up, unless if it reaches the top limit of the screen. The bottom button does the same, except in the down direction. Resetting the game puts the scores back to 0 and everything back to their original positions. We also display “PONG” on the LED seven segment display of the FPGA board.

We also make use of the VGA port on the FPGA board, as well as a monitor. VGA (Video Graphics Array) is one of the most widely used video formats. It allows us to convert our digital signals into colored pixels on the monitor. The colors are 8-bits, so there are 256 different pixel color choices. We chose to keep the visuals of our game simple and sleek, so we only used a few of the many color possibilities. We used 640x480 resolution, with a refresh rate of 25MHz for each pixel. This allowed us to display seamless motion with a high-quality image.

Figure 4.1 shows the flow of our modules for this project. We have 5 modules, including: pong\_top, pong\_segdisplay, pong\_vga, pong\_gameplay, and clock\_div. The top module inputs the 100MHz clock from the FPGA, as well as the responses from the buttons, and it outputs the final signals to the VGA. The clock\_div module converts this 100MHz clock into the 25MHz clock for the pixels, 381.47Hz clock for the seven segment display, and 3 slower clocks for the artificial intelligence, which we will discuss in depth later in this report. The VGA module is responsible for scanning through the pixels for the screen, and correctly determining the RGB arrays for each position. This uses information from the gameplay module, which manages not only the score of the game, but also the positions of the moving paddles and ball. The seg\_display module simply presents PONG on the seven segment display.



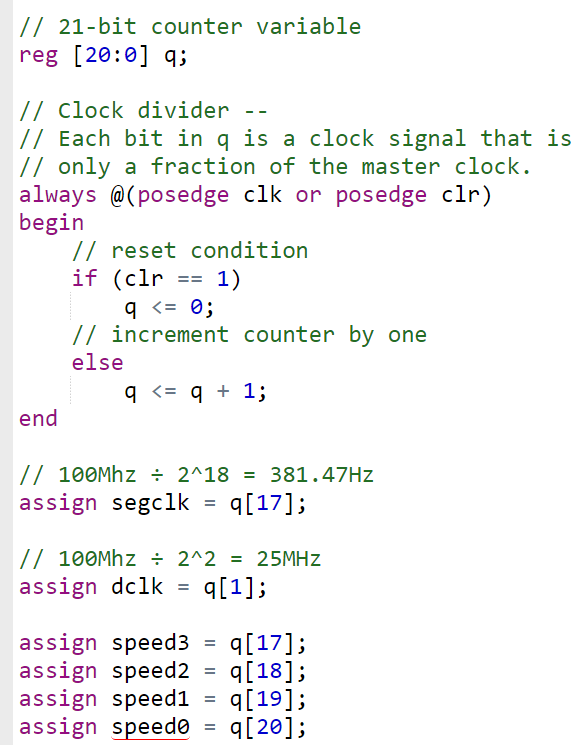
***Figure 4.1: Overall flow of design, showing I/O of each module***

**Design Description:**

Our top level module reads input from the buttons, along with the 100MHz clock signal from the FPGA board. It outputs the signals necessary for the seven segment display (SEG, Anode, and Decimal\_place) and the signals required for the VGA (VGA\_RED, VGA\_Green, VGA\_Blue, Hsync, and Vsync). There are four submodules used to divide the clock and configure the signals required for the seven segment display and VGA. These are described in detail below.

**Clock Div:**

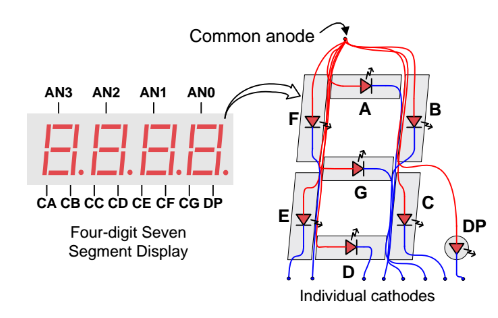
This module divides the 100MHz clock signal from the FPGA into 5 other signals, which are then used for various purposes. A 25MHz clock (dclk) is used by Gameplay (which simply passes this to VGA) for drawing the pixels. Seg\_clk is a 381.47Hz clock used by the seven segment display as its refresh rate. This clock is further divided by factors of 2, 4, and 8 for clock signals speed1, speed2, and speed3. These are used by Gameplay to determine the speed of the AI player. The method for our clock division is displayed in Figure 4.2.



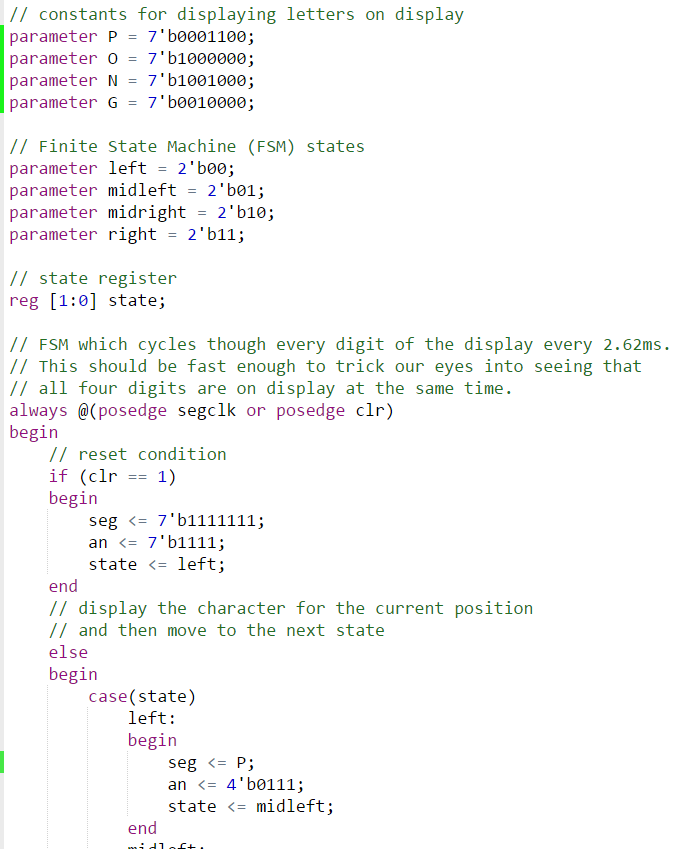
***Figure 4.2: Code snippet from clock\_div., showing how the segclk and speeds are calculated. The dclk is produced in a similar fashion.***

**Seven Segment Display:**

This module is responsible for displaying the name of our game (PONG) in the seven segment display. It only takes in the seg\_clk from the clock divider module, and outputs three signals: seg (for cathodes), anode, and decimal\_place. These values are used by the FPGA to determine which cathodes to illuminate on the display. It maps the individual anodes (each of the 4 anodes represents one character) into the correct representation for the seven segment display by assigning 0 to cathodes that should be on and 1 to cathodes that should be off, with same the order of bits as in Figure 4.3. For instance, in order to display a P, the digit would be assigned the value: 7'b1001000, with A being the most significant bit, G being the second to least most significant bit. We set DP = 1 because we don’t want the decimal places to be illuminated. How this was largely accomplished is seen in Figure 4.4. Using the seg\_clk, we chose when to illuminate each character (since we can only illuminate one at a time), and displayed the appropriate value.



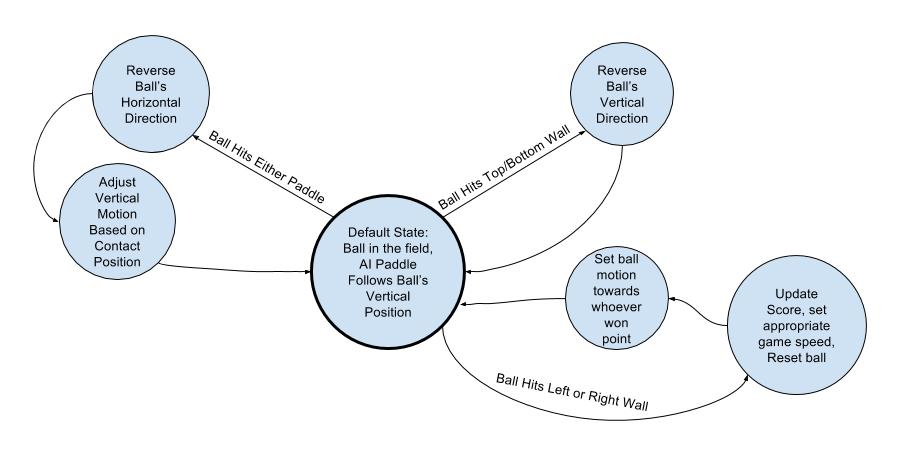
***Figure 4.3: Mapping of anodes and cathodes in seven segment display.***

******

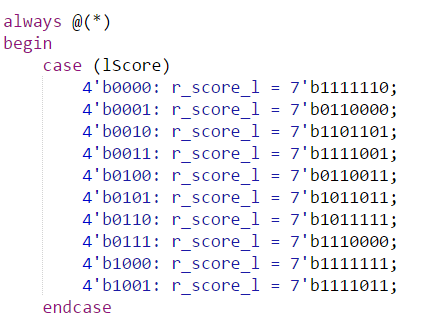
***Figure 4.4: Code snippet from segdisplay, adopted from NERP Demo. This shows how the cathode values are assigned, and one of the anodes are selected.***

**Gameplay:**

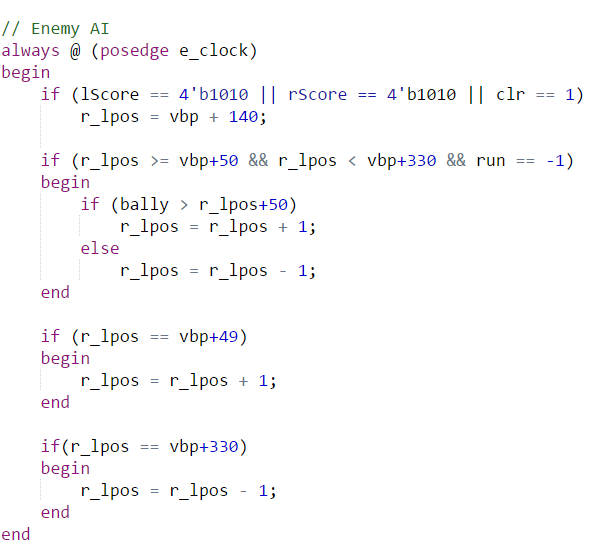
This module was responsible for a large portion of the project. It defined the positions of the moving parts of the game, including the user-controlled paddle and the AI paddle, the ball, and the score. A finite state diagram of the general gameplay of the game is displayed in Figure 4.5.The pixel locations of the various items were outputted to the VGA module, where the appropriate RGB values were assigned. The score was implemented with two counters, each incrementing when its player scored. We used a design similar to that of the seven segment display for the score (as seen in Figure 4.6), where virtual segments were turned on to display the shape desired. The paddle position for the user was controlled by input from the up push button and the down push button. The enemy paddle positioned was determined via an algorithm. The algorithm checked the current ball vertical position, and started moving the paddle in the direction towards that vertical position. It would update its target vertical position very clock cycle, with the clock cycle determined by the difficulty level selected. This is seen in Figure 4.7. We control the ball’s trajectory and position largely by maintaining its rise and run. Every clock cycle, we update the ball’s location according to its rise and run values. When the ball makes contact with another surface, such as a paddle or a wall, we change the rise and run to accurately represent deflections. The ball maintains a constant velocity throughout the game, so deflections only change the ball’s direction. When the ball bounces off of a paddle, its run is flipped and its new rise is determined entirely by the location on the paddle at which it made contact. This is shown in Figure 4.8. A player’s score is incremented when the ball gets beyond the other player’s paddle. The ball is placed back at its original position, with its rise and run reset. This is shown in Figure 4.9.

****

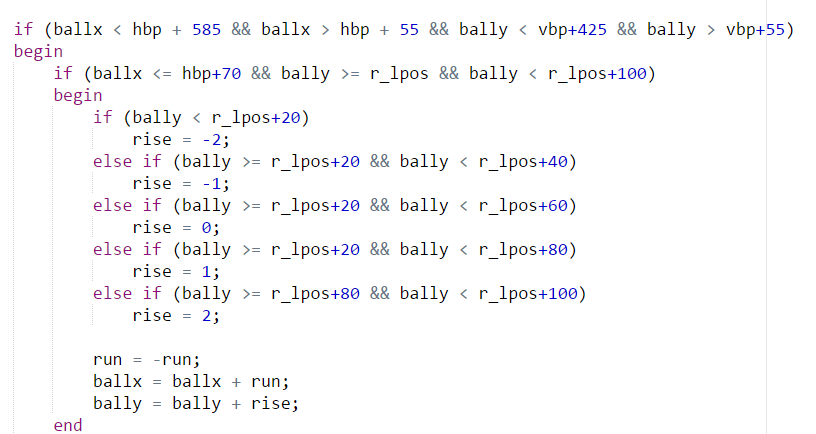
***Figure 4.5: Finite State Diagram of the general functions of the gameplay.v***



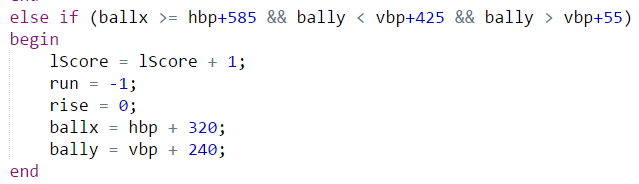
***Figure 4.6: Code snippet from gameplay.v. This converts the player score into a seven segment display-like representation, to be draw by the VGA.***

**

***Figure 4.7: Code snippet from gameplay.v. This controls the position of the enemy paddle.***

******

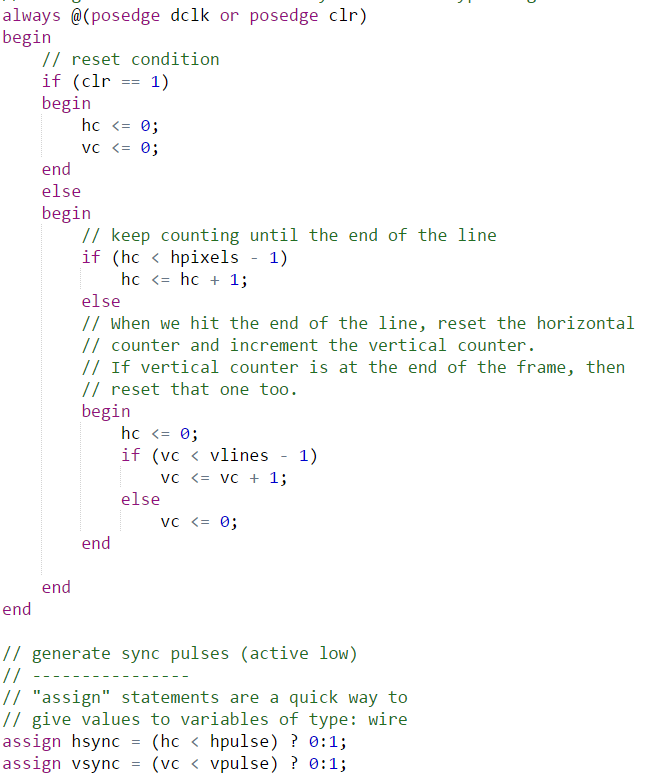
***Figure 4.8: Code snippet from gameplay.v. This shows how we adjusted the ball position and its rise and run upon making contact with various portions of the right paddle.***

******

***Figure 4.9: Code snippet from gameplay.v. This shows how the left player’s score is incremented when the ball reaches a point beyond the paddle.***

**VGA:**

To create a VGA display, we need 5 values: RGB Red, RGB Green, RGB Blue, Vsync, and Hsync. Red and Green are both 3-digit values, Blue is a two-digit value (since our eyes aren’t able to see as many hues of blue), and together these are used to build the 8-bit color vector. This decides what color the pixel we are currently drawing will be. Vsync tells us which row of pixels we are currently working on drawing, and hsync tells us which column of pixels we are drawing. To draw an image, we essentially loop through all the pixels from the top left of the screen down to the bottom right, adjusting each pixel’s RGB values accordingly. The colors themselves are actually chosen in the module above. This module focuses on scanning looping the pixel numbers, setting the current pixel and hsync and vsync values accordingly. This method is displayed in Figure 4.10. We based our methods here largely off of the NERP sample project that was provided in the lab specification.



***Figure 4.10: Code snippet from VGA.v. This assigns the hsync and vsync values.***

**Simulation Documentation:**

Since the game largely revolved around placing the game objects in the right places on the display, it was difficult to test the code without running the code on the FPGA in order to see the issues involved with the code. Due to this, much of the testing and debugging of the gameplay and VGA code was done through trial-and-error of adjusting the placement of various shapes on the screen. This, however, meant that it was often somewhat difficult to accurately pinpoint the origin of many of the issues that had risen during development, which manifested themselves in graphical glitches or other types of oddities that were solved only by closely examining the code and setting the gameplay in its different states in order to observe the glitch.

Even though the code related to graphics was partly adopted from the NERP Demo program (source at the end), it still had to be adjusted to work with our FPGA board, which meant adjusting the clocks and then connecting the right output pins to the appropriate hardware pins on the FPGA board.

Since the game has many conditions and states that are only in effect in rare circumstances, much of the final testing involved playing the game multiple times and forcing the game to execute corner cases that would otherwise be overlooked. Through multiple tests and small corrections, we were able to get rid of any fringe graphical and game logic issues.

**Conclusion:**

**Summary of Design:**

In this lab, we went through the complete project lifecycle--starting with an idea, creating a project proposal, designing, testing, and then presenting a finished project. Our goal was to implement a fully-functioning version of Pong on the Nexys3 Spartan-6 FPGA board, and we were successful. We are able to reliably display dynamic, moving images on a monitor using the VGA. And we are able to control the movement of the images not only with hardcoded algorithms, but also by reading input from a human player through the push buttons.

Our design is comprised of 5 separate but interoperating modules: the top module, clock div, seven segment display, gameplay, and VGA. The top level module gathers input from the 3 push buttons we are using to control the display on the monitor and from the FPGA’s clock. It also instantiates and provides the input for the clock and seven segment display modules, while outputting the final signals required for both the VGA display and the seven segment display.

The clock div module simply divides the 100MHz clock into 5 other signals: 25MHz, 381Hz, 191Hz, 95Hz, 48Hz, and 24Hz. The 25MHz signal is used to draw pixels on the screen by the VGA module. The 191, 95, 48, and 24Hz clocks are used to control the AI paddle’s algorithms at the different difficulty levels. The 381Hz clock is used to refresh the seven segment display at a rate that does not show any flickering. The seven segment display module takes in its clock as input, and by turning on specific cathode and anodes for appropriate sections of the clock period. This enables it to consistently display the name of our game, Pong. The gameplay module was responsible for controlling the various components that you see on the screen. It provided the pixel locations for the moving ball and paddles, as well as the score. It also enabled the functionality for multiple game levels by allowing us to select the speed of the AI player. We adjust the trajectory of the ball by simply adjusting its rise and run when the ball comes into contact with another surface, such as a wall or paddle. The score is displayed in a fashion like a virtual version of the seven segment display--turning on specific segments to get a desired integer shape. The AI-controlled paddle is controlled by, at varying intervals (depending on difficulty level), checking the location of the ball and moving towards the ball’s vertical position. The more rapidly it checks the ball’s location, the more accurate it will be, so it will be harder to get the ball past it. The VGA module then takes the positions of the paddles and ball and the score, and outputs the correct shapes and colors. It does this by continuously looping through the full range of pixels and assigning the appropriate RGB values for the specified Hsync and Vsync values. These are then from the FPGA through its VGA port to a monitor, where everything is displayed.

**Difficulties Encountered:**

The most difficult part of this project was definitely using the VGA. It took us a good amount of time to figure out how to display static images with the VGA, even with the sample VGA And, once we had that figured out, we still had to learn how to control the movement of the various components of our design. Time became a real constraint, especially in the last days of development. On the Tuesday before it was due, we had to wait over 2 hours in the lab to get a work station.

**Work Breakdown and Suggestion:**

Completing this lab was a team effort. Max focused on the general design of the project and then the basics of VGA, before also fixing up the seven segment display. Sparsh focused on the gameplay of the project and figuring out how to make everything move appropriately, while also implementing the AI and button controls.

Overall, we thought that this was a successful lab. The time provided probably would have been sufficient if we had worked more frantically at the beginning of the process. In the end, however, it would have been nice if there were a few more FPGA’s available for when the lab was insanely crowded the day before our project was due.

**Sources**

VGA Display Code: <https://pumpingstationone.org/2013/04/nerp-fpgaok/>