M152A - Lab 4
Design Lab
The Multiplayer Tank Experience

Markus NOTTI - 904269231
Kyle BAKER - 604273748
Niels PINEDA - 604272353

March 17, 2016

Introduction

In this lab, we implemented a basic multiplayer tank game utilizing the VGA for the main display, buttons and switches on the board for controls, and the seven segment display for additional in game information display.
Our game was played in the following manner: Each of the two players playing the game controls a tank and fires projectiles at his opponent from the opposite side of the map, displayed on the VGA. To fire, the player 1 and player 2 would press the *fire1* and *fire2* buttons respectively. As the players fire the projectiles using the buttons on the FPGA board, the players are also able to adjust the initial velocity and angle of the projectiles using additional buttons. How long the *fire1* and *fire2* buttons are held down by the players determines the projectiles' initial velocity, and the angle can be adjusted by clicking either the *angleUp* button to raise the angle or the *angleDown* button to lower the angle.

If a player's tank is struck by a hostile projectile, the player's tank loses health, which is immediately updated on the 7-segment display. If the projectile makes a direct hit upon the tank, the player's health is reduced by 14HP. If it is only a glancing blow the player's health is reduced by only 7HP. The health display for both of the players' healths was also configured to display the player's own health facing the correct player. In other words, since the players themselves must be on opposite sides of the board to play the game, each player can see his own health, because it is flipped to the perspective of the player who's health it is displaying.
If a tank reaches a health of 0HP, that tank turns red marking the end of the game and victory for the player controlling the other tank. The game can then be reset by flipping the *rst* switch on the board.

Behind the scenes, a clock module, which along with the other modules, will be dis-

cussed in more detail later, created three separate clocks which would drive the program. The three clocks created were 25MHz, 500Hz, and 50Hz. The 25MHz clock was responsible for driving the VGA controller, the 500Hz clock drove the seven segment display, and the 50Hz clock was responsible for updating the position of the tanks and the projectiles on screen.

In order for all of this gameplay to be a reality, we needed to design and implement the following set of features:

Design Description

When designing the lab, we created 8 main modules *clock*, *vga*, *sevenSeg*, *tankL*, *tankR*, *projL*, *projR* and *collisionDetector* all running in the main module *game*. The clock module created 3 slower clocks to regulate other parts of the program, while the rest of the modules, their names relatively self explanatory, contained the program's logic.

Clock Module - *clock*

The clock module took as input a 100MHz clock and outputted three separate clocks to govern the rest of the program: a 25MHz clock, a 500Hz clock, and a 50Hz clock. This was implemented by 3 simple counters. The 25MHz counter would count to 4, the 500Hz clock would count to 200,000, and the 50Hz clock would run to 2,000,000. At the posedge of the 100MHz clock, the counts would increment until they reached their respective maximums, triggering the appropriate output clocks to 1. After reaching this maximum value, the counters would drop back to zero and start their counts over again. This clock module was the beating heart of our software, controlling the rates at which all logic would be calculated, all input would be read and all output would be written.

VGA Module - *vga*

Our VGA module was essentially a controller we designed to handle the VGA functionality for the program. This was probably the most complicated of the lab for us, because it took a lot of trial and error to get it working. Our VGA controller module takes as the 25MHz clock created by the clock module and produces as output 8 wires which make up the color composition of the pixel to be drawn and two wires *Hsync* and *Vsync* to govern the VGA.

In order to implement this controller, two counters, *counterX* and *counterY*, tracked the location of the VGA sweep as it swept across the screen and down the screen. In addition to tracking the location of the VGA sweep on the 640x480 display to figure out when to draw what pixels, the counters were responsible for tracking when exactly it was even appropriate to begin outputting values to the 8 wires governing the pixel color. The tricky part about designing the VGA controller was this specific timing. As illustrated further by the image below, there are very specific periods of time where no color combination must be output, and there are specific times when the *Hsync* and

Vsync must be fired to indicate that a new section of the display is about to be drawn. For every line, there would be a front porch for 8 cycles, a horizontal sync period where Hsync would be fired for 96 cycles, a back porch for 40 cycles, an 8 cycle left border, then a 480 cycle display followed by an 8 cycle right border. For the periods mentioned above, the colors outputted by the vga module to the VGA were all set to 0. *counterX* tracked how many cycles had passed to give the program working knowledge of what period it was in. During the display period is where the appropriate objects were drawn to screen. The Vertical Sync had similar periods with slightly different cycle count. The exact count is displayed below in the image. Note that where the image says 'pixels', we implemented our controller so that each cycle moved onto the next pixel, so in this situation, pixels can be considered equivalent to cycles.

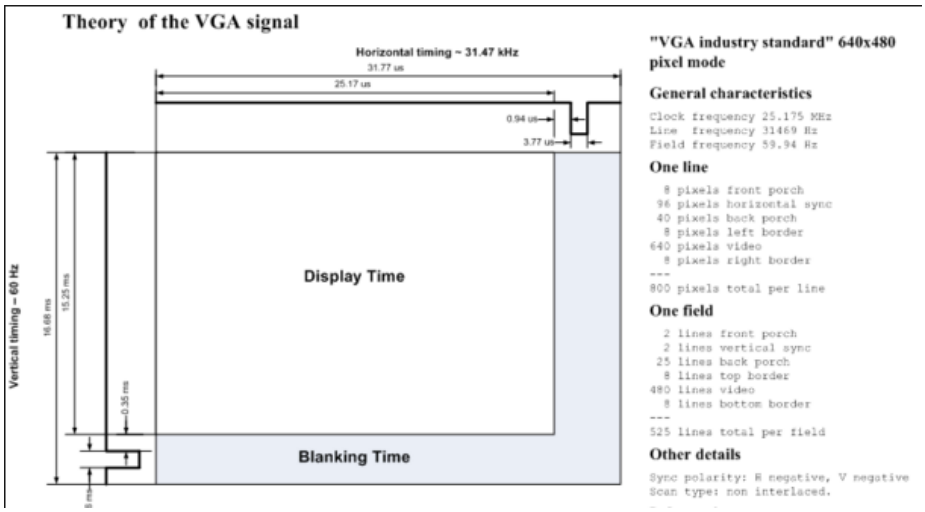

Figure 1: Basic VGA theory upon which we based our VGA controller

Seven Segment Display Module - *sevenSeg*

This module was the controller we designed to handle the seven segment display. It would be passed both of the players' healths and it would display player 1's health facing him, and player 2's health facing opposite, so each of the players can look down at the seven segment display from his respective side of the board and see how much health he has remaining.

The encoding of the tank health into a form readable and displayable by the seven segment display was done using two priority encoders, one for the digits that faced player 1, and one for the digits that faced player 2. Once the digits were all encoded by the priority encoders, the 4 digits were separately displayed at a frequency of 500Hz, governed by a clock that was passed into the module as input after being created by the clock module. By displaying only one digit at a time, and displaying a different digit

500 times a second, it seems to the human eye that all four digits are being displayed simultaneously.

Tank Module - *tankL*, *tankR*

The tank module was designed with the purpose of controlling the location and the movement of both players' tanks on screen. While it didn't make them actively move across the screen, the job of the VGA controller, the tank module looked for movement button presses from the players and updated the tanks' positions accordingly, outputting them to the projectile modules so they knew where to place the projectiles on screen upon initially firing, and to the VGA module so that the VGA knew where to draw them.

Not only did the tank module handle listening for movement buttons *btnL* and *btnR* in order to update the position of the tanks accordingly, it also listened for *angleUp* and *angleDown* buttons to adjust the angles at which the projectiles would be firing from the tanks. Every time either the angle up or angle down buttons would be pressed, the tank module would add or subtract 5 degrees from the angle of the relevant tank. This angle value would then be passed into the projectile module.

It should also be noted that due to some minor issues we had when we created two instances of the tank module, one for player 1 and one for player 2, we decided to do a quick work around and make them two separate modules. Since they are effectively the same, I have considered them the same for the sake of this report.

Projectile Module - *projL*, *projR*

The projectile module contained all of the logic required to simulate the motion of the projectile as if its been fired from the barrel of a cannon. This module takes as input the tank's location and the angle at which that tank is currently set to fire its projectile. The projectile module then takes the angle and uses a priority encoder to retrieve the correct initial x and y velocities at which it should be fired. Then, if the appropriate fired button is pressed, the projectile takes the initial position of the appropriate tank and is assigned the initial velocities found by the priority encoder. Next, the new position of the projectile is determined 50 times a second, and is outputted to vga to be displayed. In order to keep the projectiles flight from being adversely affected by the movement of the tank while the projectile is in flight, updates to the tank's initial position and angle are stopped while the projectile is in flight. Once the projectile hits its target or falls off the screen, the projectile module begins again to listen.

Like the tank module mentioned above, the same problem was occurring for us, so we went ahead and made two separate projectile modules, as it solved our problem. However, in the same way, I have considered them two instances of the same module for the sake of simplicity in explanation.

Collision Detector Module - *collisionDetector*

The collision detector module, as the name suggests, was responsible for monitoring the positions of the tank modules and the projectile modules and listening for any overlap. It was also responsible for relaying that information to the rest of the program. It simply would take in as input all of the positions of the objects mentioned above and check for overlap. If there was overlap, it would simply set a t1_col flag to 1 or a t2_col flag to 1 depending on which tank was hit. The projectile module would pick up this flag and reset, and the seven segment controller module would also pick up this flag and update the health of the tanks on the display accordingly.

Digital Logic Design

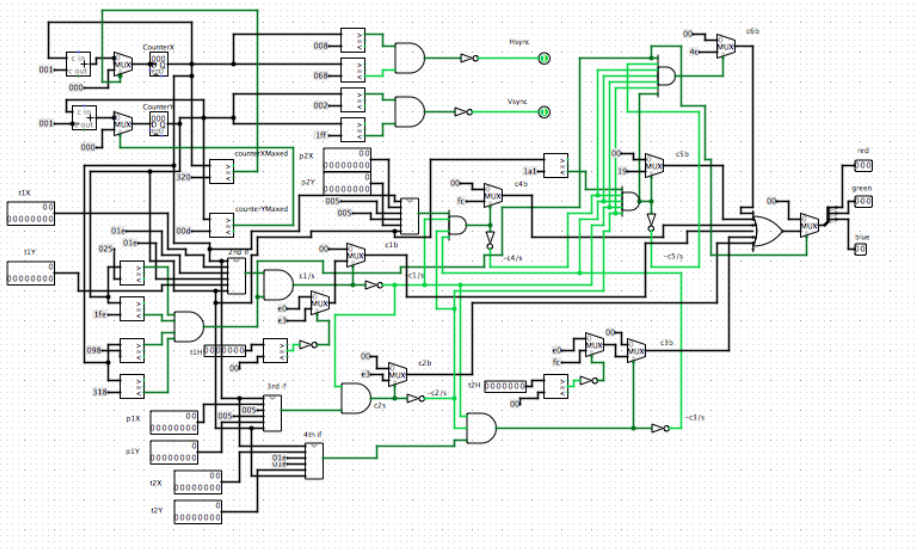

Figure 2: VGA module logic

This vga module takes in the t1X, t1Y, t2X, t2Y, p1X, p1Y, p2X, p2Y, t1H and t2H as input and outputs Hsync, Vsync, red, green and blue. A 25MHz clock is also input and drives the two counter registers that you can see in the top left of the above diagram. For simplicity's sake, the clock is not included in the image above. The above image is the digital design logic for the VGA controller module.

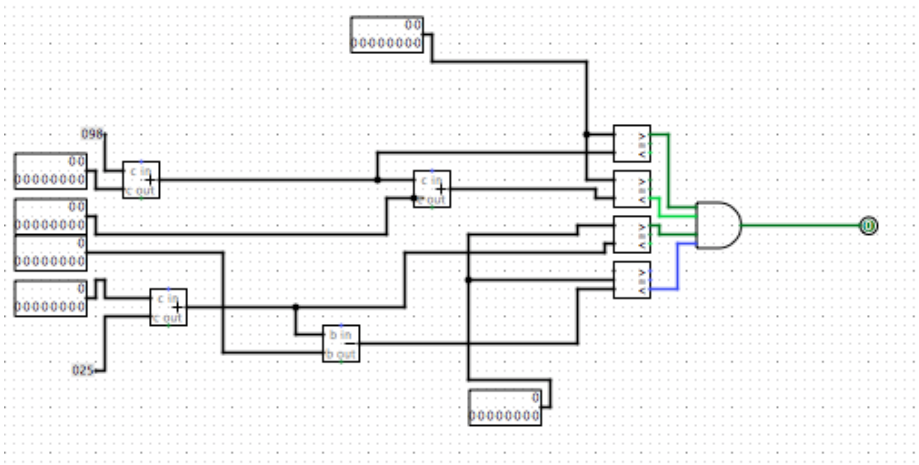

Figure 3: VGA black box 1

The above image is the digital design logic for a black box included in the diagram showing the logic for the VGA controller module. The circuit in the image above corresponds to the long rectangular module in the VGA diagram.

Simulation

We took a relatively dynamic approach to the simulation aspect of our project; whenever we implemented a new module, we tested it immediately as the Game was relatively cumulative. In other words, we had to build up from our simulation and test functionality from there. These efforts included: displaying colors to verify our understanding and implementation of our VGA module, displaying a single tank which passed in it's coordinates to the VGA module, displaying (with the right speed for a correct visual effect) a projectile being shot from the tank, creating two tanks, and finally collision testing.

Initially, our simulation efforts were focused on the VGA module, as having the display was obviously a necessity to testing our overall project. We did so by loading our VGA module onto our FPGA, and then switching the input on our monitor. Originally, we merely attempted to have the screen display a single color (Green) just to verify if our VGA module was working at all, and after many efforts, we were successful. Afterwards, we attempted to display both Green and Blue (Green for the floor/grass and blue for the sky), which again was successful.

The consequent simulation efforts were much more of a challenge, as we were attempting to display a "tank" (a square) at a specific location, with the input coming from a tank module. Because we believed our code to be correct but the tank was not

being displayed, we created a test bench file. This allowed us to create an instantiation and check the coordinates of our first tank via adding them to the waveform and using ”restart” and ”run # ms.” We soon realized that our if statement checking the coordinates to draw a different color for the tank were incorrect, and after fixing and running this on the board once again, our desired outputs were correct.

Afterwards, we attempted to simulate our projectile motion equations. Once again, this was done simply by loading the program onto the FPGA board and continually shooting the projectile until we found it to match up with our expectations. While the displaying of the projectile itself was not the challenge, attempting to find the correct clock speeds to make it look smooth without drag was very difficult. This became exceedingly time consuming as we began using larger integer values which slowed down the synthesizing and generating process a great deal.

With a single tank working correctly, the next objective was to display two tanks, however this proved to be our biggest challenge. For the longest time, we were unable to display two tanks, and so we attempted to simulate via a test bench file. This reaffirmed that the second tank was passing in the correct coordinates at the correct time in our displaying if statement, however the tank would still not be drawn correctly. We finally realized our problem when we changed the second tank to be twice as large; the second tank was simply being drawn over the first tank so it would essentially be invisible. This simulation effort led to our research and finding that there were certain limitations of initializing modules, and thus, we had to make the 2nd tank (as well as its projectile) a separate module altogether.

With two tanks correctly shooting and moving on screen, our final simulation efforts were focused on the collision of the projectiles and tank. The biggest trouble was consistently hitting each other to verify the decrease in health, but we verified that our collision detector was working as hitting one another decreased the number on the FPGA board, which corresponded to the decreasing of health.

The simulation process for this lab was rather repetitive and simple in its process. Once the VGA was working correctly, outside of very specific issues, our simulation came from simply loading the program onto the board and playing the game testing our functionality. The simulation effort allowed us to carefully choose various values to make the game as clean, smooth, and precise as possible.

Conclusion

Compared to our previous labs, this particular project was much more organized. We used separate modules this time, these included one for: power (strength of projectile), the VGA, the seven segment display, and movement. While it wasn't ideal (as it was a bit repetitive), due to initialization of location problems, we had to have two separate, but identical modules for each tank and projectile.

The game itself was relatively simple and straightforward, yet enjoyable. There were two tanks which were capable of moving left and right, and the objective was to shoot one another. This was controlled by buttons to adjust angle and horizontal location, as well as the shoot button adjusting for strength. Depending on how directly you get hit from the projectile, the damage was either 7 or 14 of your total 99 health. Unlike many other tank games, this live multiplayer style was relatively different, and it allowed for a somewhat more novel idea.

The main difficulties we encountered were with the VGA module and projectile motion. Attempting to have output on the monitor via the VGA was so precise, that any small mistakes led to no output at all. The only ”solution to this” was just testing over and over again after trying to fix what small bugs we found. The projectile motion was troublesome at times, especially because of the numbers we had to use. For starters, because we could not use decimals in certain instances, we had to scale everything to a larger number in order emulate decimals. On top of that, because no trigonometric functions exist in Verilog, we had to calculate and hard code various values for the angles.

Overall, this project was absolutely the most enjoyable to work on. It allowed for us to apply our knowledge learned in class to something we were able to exercise our creativity on. Outside of giving more time for the project, the only ”modification” for the lab would have been to allow the students to test the other projects.