CS M152A - Lab 4

Jair Hinojosa, Kyle Romero
904771873, 204747283
Section 5
TA: Fan Hin Hung
Date Performed: 3/16

1.0 - Introduction

For our final lab, we chose to design and implement a version of the classic arcade game Stacker. The player is able to control the game using buttons on the FPGA board while the VGA protocol is used to display game graphics on a computer monitor.

1.1 - Background

Stacker is a popular arcade game that tasks the player with creating a stack of blocks. The game board consists of a fifteen by seven grid of squares. On each level a certain number of blocks on the highest row reached by the player moves back and forth horizontally. The player makes progress in the game by successively stacking these moving rows on top of each other. The player wins by creating a tower that reaches the top of the game board.

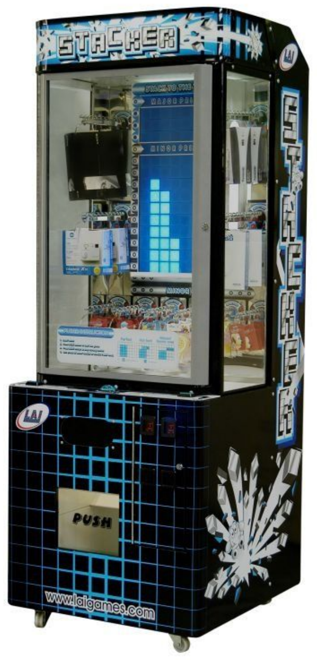


Figure 1.1.1 - A typical arcade implementation of a Stacker game.
Source: http://keywordsuggest.org/gallery/480582.html

VGA Protocol

The Nexys 3 FPGA Board uses an 8-bit color and two sync signals (horizontal and vertical) to generate a display on an VGA compatible monitor. The sync signals are used to indicate when the current pixel being drawn should move to the next line or back to the top of the screen. By manipulating the sync signals according to certain parameters, we are able to display a certain color on the screen that corresponds to the current pixel being drawn.

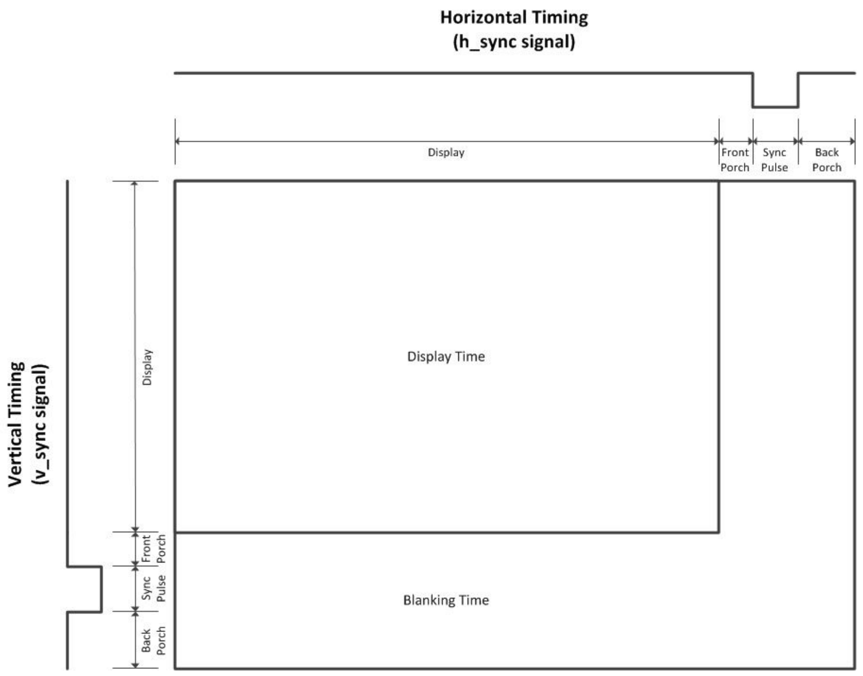


Figure 1.1.2 - A general outline of the key timing signals that must be considered when using a VGA display. Source: https://ericeastwood.com/blog/8/vga-simulator-getting-started

1.2 - Design Requirements

We decided to implement a version of stacker that is very similar to that pictured above. We established the following game rules for our player. The player is able to stop the motion of a moving row by pressing a button. If there are any blocks that overlap with the previous row, then the player continues onto the next level with the number of blocks that overlapped. If all blocks are misaligned, then the player loses. As the player's current level increases, the blocks get faster and the maximum number of blocks in the moving row decreases.

As per the final project specification, we established milestones that are significant on our path to the full implementation described above. These will be used as a rubric for the grading of this project.

Title	Percentage	Features
Basic Display	15%	The game begins and displays a static image of the game board on the monitor
Advanced Display	15%	The game begins and displays a moving image of the first row on the game board
Simple Implementation	40%	Moving blocks are displayed on the board and the player is able to move to the next level using the select button
Reset Implementation	10%	The player is able to reset the game to its initial state by pressing the reset button

Lose Implementation	5%	The player is able to lose the game by pressing the select button when all blocks are misaligned from the previous row
Speed Scaling	5%	The speed of the moving blocks increases as the player gets higher on the board
Win Implementation	5%	The player is able to win by building a tower that reaches the top of the game board
Block Scaling	5%	The number of blocks is limited to a maximum value at each level

Figure 1.1.2 - The development milestones that make up the rubric for this project.

2.0 - Design Documentation

Our digital system consists of a top-level module that combines debouncers for input, a VGA driver, and a game logic controller into the arcade game Stacker. Our implementation uses input from the FPGA board and uses the VGA protocol to write graphics to a monitor. We incorporated into our design a high-level game controller module that controls each lower module and their interactions.

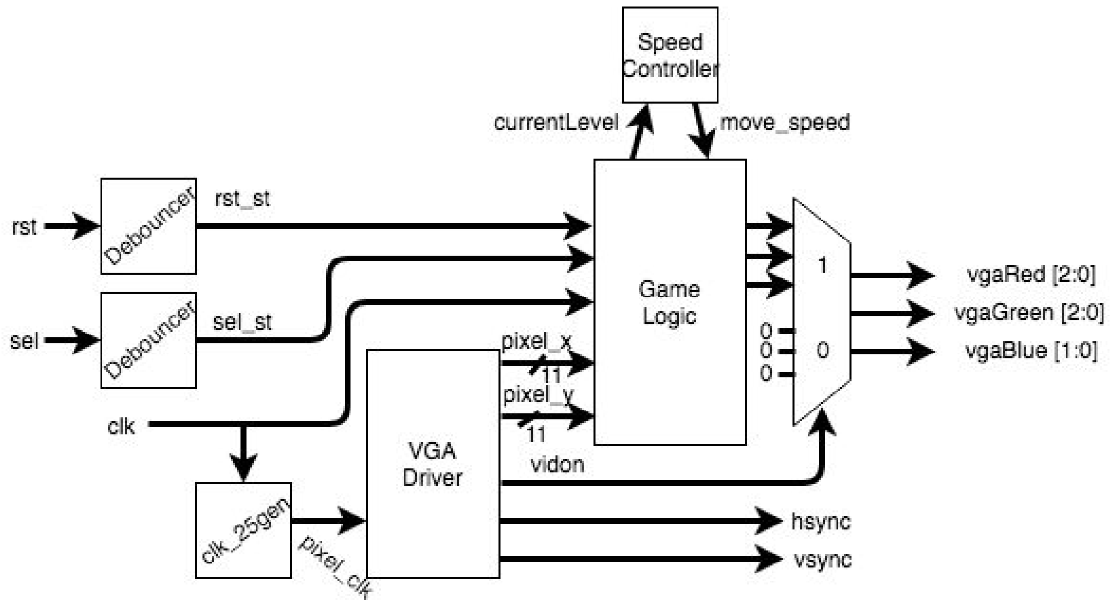


Figure 2.0.1 - A top-level block diagram that outlines the interactions between various components of our system.

The speed controller and clk_25gen module are not outlined below since they are both minor modules. The speed controller takes in the player's current level and outputs the movement speed clock and allows us to difficulty scaling as the player increases their level. The clk_25gen module generates 25 MHz clock that dictates the pixel clock used by the VGA driver.

2.1 - User Interface

We designed our user interface for this project by taking inspiration from the stopwatch lab. The rightmost button is as a reset and forces our game to its initial state. The middle button is used to select the position of moving blocks and, hopefully, advance to the next level.

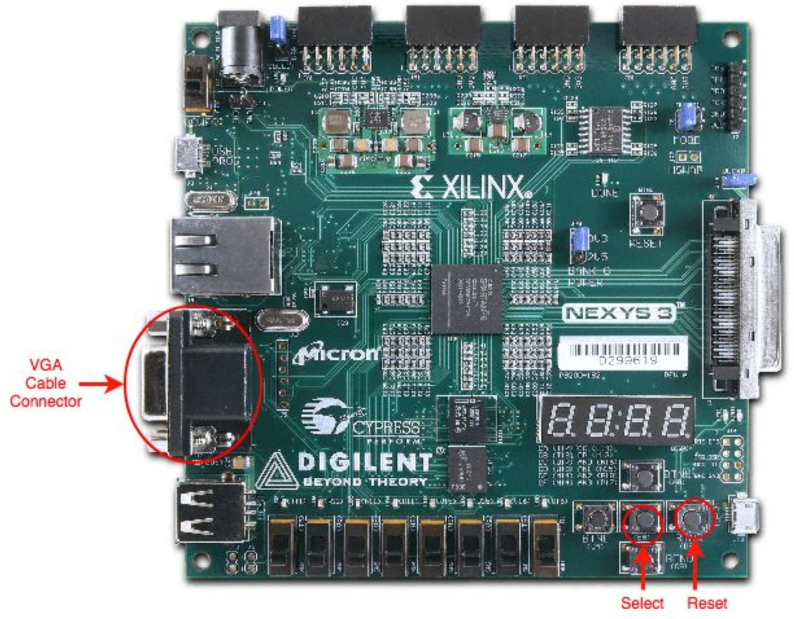


Figure 2.1.1 - The user interface specification of our digital system.

2.2 - Debouncer

Our debouncer implementation is an improved version of the module used in our stopwatch lab. By fine-tuning the timing parameter (CLK_MAX) of our debouncer, we were able to achieve reliable debouncing performance during high speed gameplay.

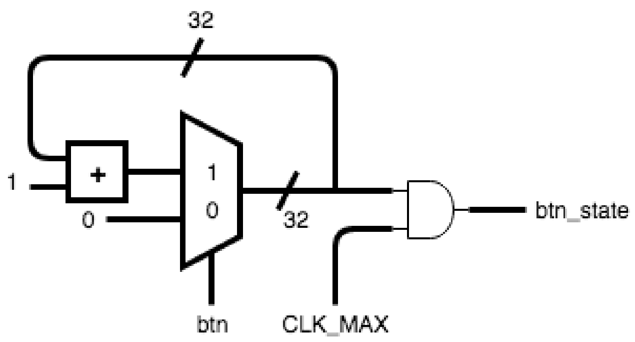


Figure 2.2.1 - A block diagram translation of the Verilog code that implements our debouncer.

2.3 - VGA Driver

Our VGA interface is derived from a combination of the VGA demo provided for our use and a Verilog project that implements a simple maze game using VGA that we found online[1]. The module uses various parameters to generate the signals that correspond to a 640x480 display resolution. Specifically, the horizontal sync is pulsed when the line needs to wrap around to a new line and the vertical sync is pulsed when the current pixel needs to wrap back up to the top of the screen. Furthermore, we output the coordinates of the current pixel being drawn, which is used by the game logic module to determine what color to draw to each pixel.

2.4 - Game Logic

Our logic controller uses the debounced player controls to drive the logic of our game. This is the main module that drives our game and handles: block movement, player interaction, reset functionality, win functionality, lose functionality, difficulty scaling, and maximum block scaling.

Blocks are moved in increments of their blocks size from left to right. If a block is detected to be close to one edge of the screen, a flag is set which indicates that the blocks should start moving in the other direction. If there are multiple blocks, the engine stores the location of each block and ensures that the left or rightmost block is used to reverse to check if the current block group should move in the other direction.

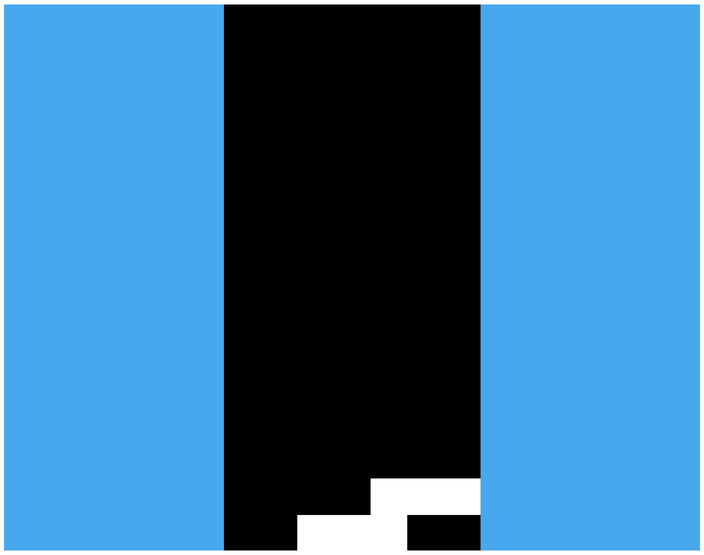


Figure 2.4.1 - Our game board is split into a 7x15 grid that dictates where blocks are able to move and where blocks are spawned on each level. This is a rough sketch (not actual gameplay footage) of a player on level two with the three blocks currently moving.

Player input drives the state machine of our game logic. The reset button is used to force the game to its initial state and the select button is used to stop the motion of a block group. If this group is above previously placed blocks, then the player moves on the the next level, otherwise the player loses. If the player's current level is greater than or equal to sixteen then the player wins.

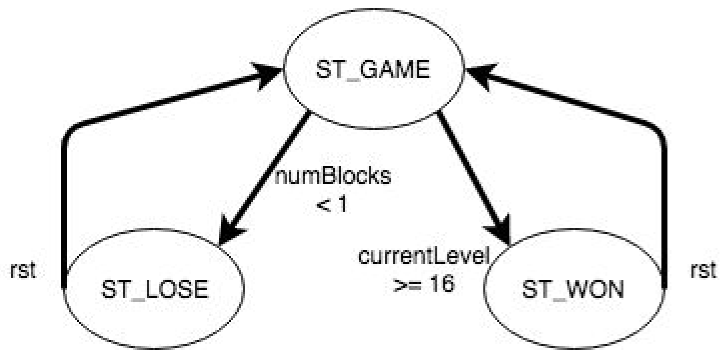


Figure 2.4.2 - An FSM diagram which outlines the states of our game logic module.

3.0 - Simulation Documentation

In order to simulate the design of our digital system, we wrote a simple testbench that could be used to run some simple smoke tests on our program. After correctly simulating each of the features below, we validated their functionality further by synthesizing our system on the FPGA board.

Test Name	Figure	What are we looking for?
Image Test	3.0.2	Ensure image generation signals are changing
Simple Functionality	3.0.3	Ensure player advancement to next level
Reset Functionality	3.0.4	Ensure reset forces game to its initial state
Win Functionality	3.0.5	Ensure the player can win and game state is updated
Speed Scaling	3.0.6	Ensure the movement clock threshold decreases

Figure 3.0.1 - The methods and features that we tested on our game. Note that some features are difficult to test during simulation due to the visual nature of a video game.

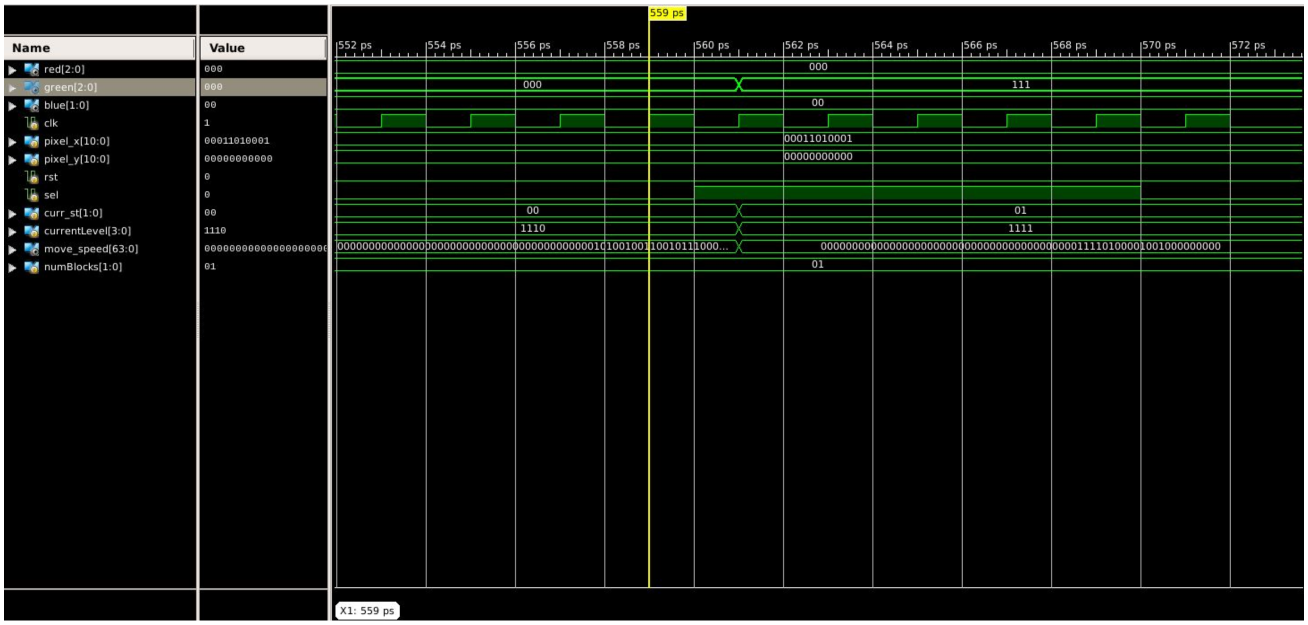


Figure 3.0.2 - In this waveform, one can see the changing signals being output on our VGA pins that correspond to signals for different colors. Note **green** changes to all 111s, which represents the player winning and a green screen being displayed.

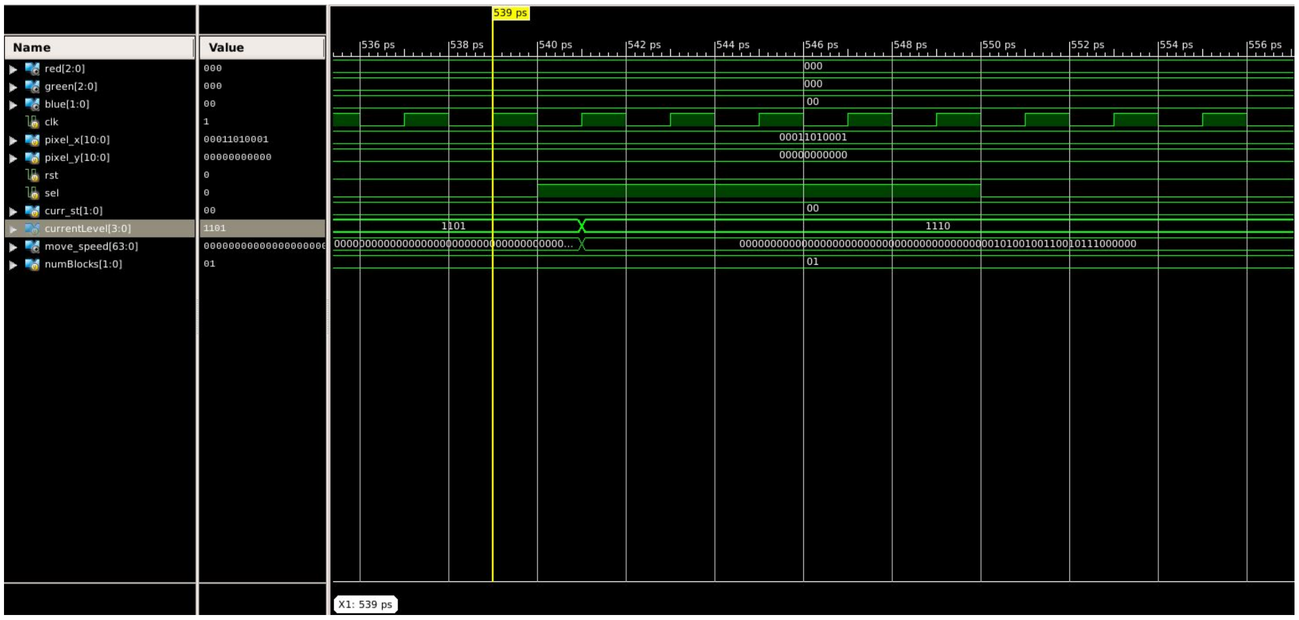


Figure 3.0.3 - In this waveform, one can see the current level changing once the player presses the select button (the **sel** signal is set high). This represents a player successfully progressing to the next level.

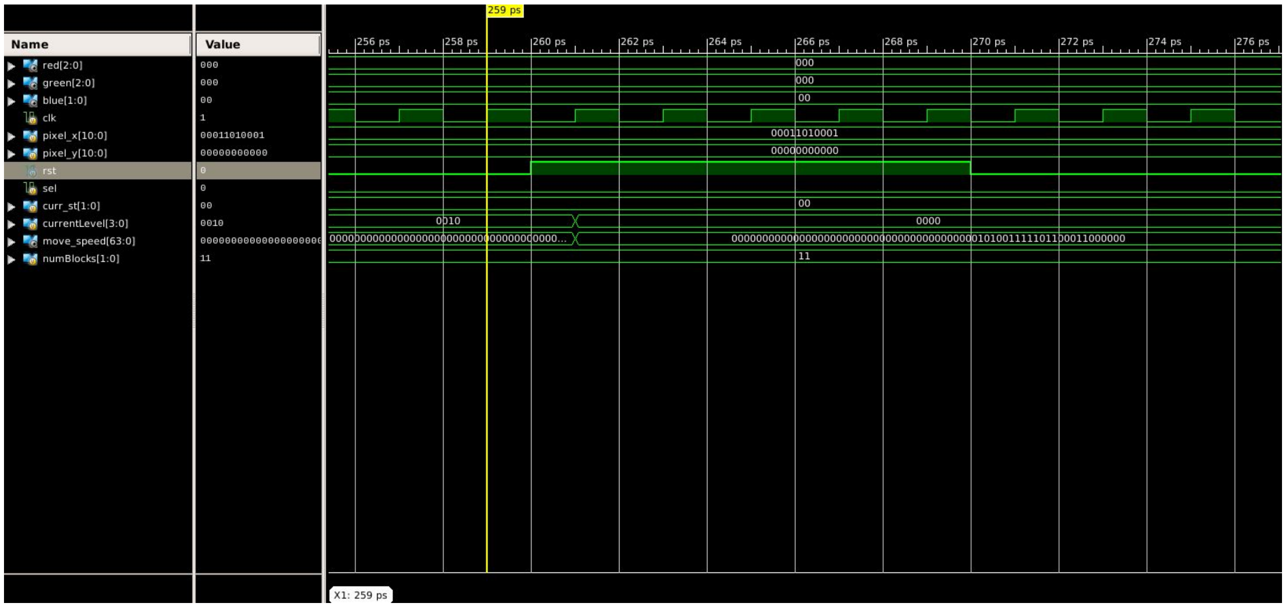


Figure 3.0.4 - In this waveform, one can see the game being forced to its initial state (for example, **currentLevel** is set to 0) when the player presses the reset button (the **rst** signal is set high).

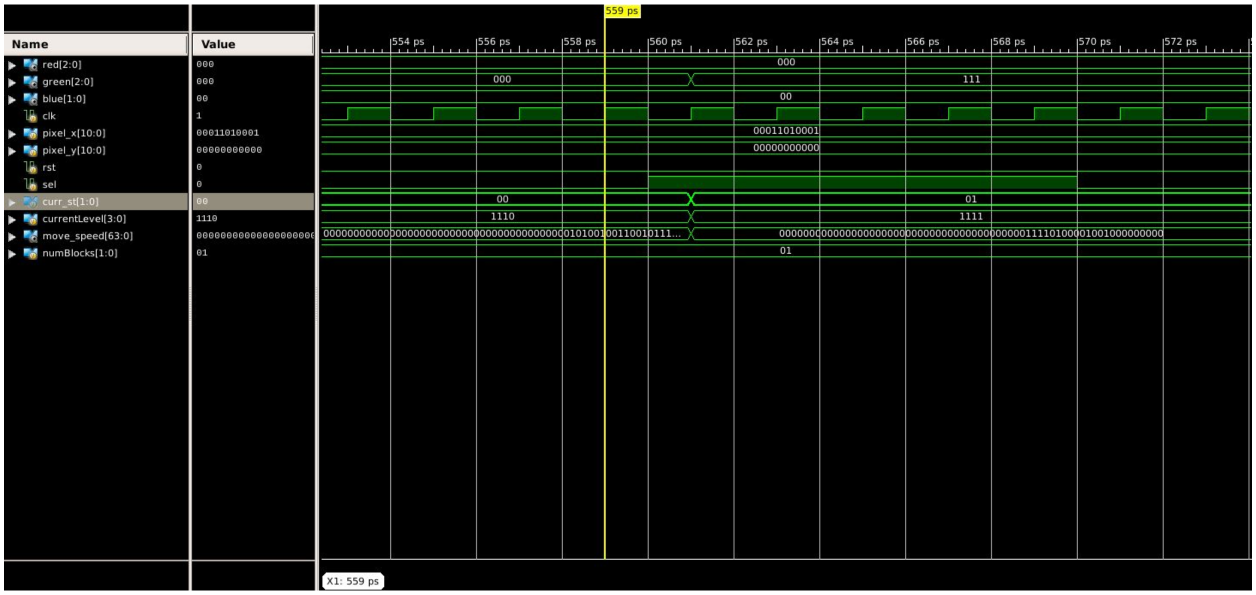
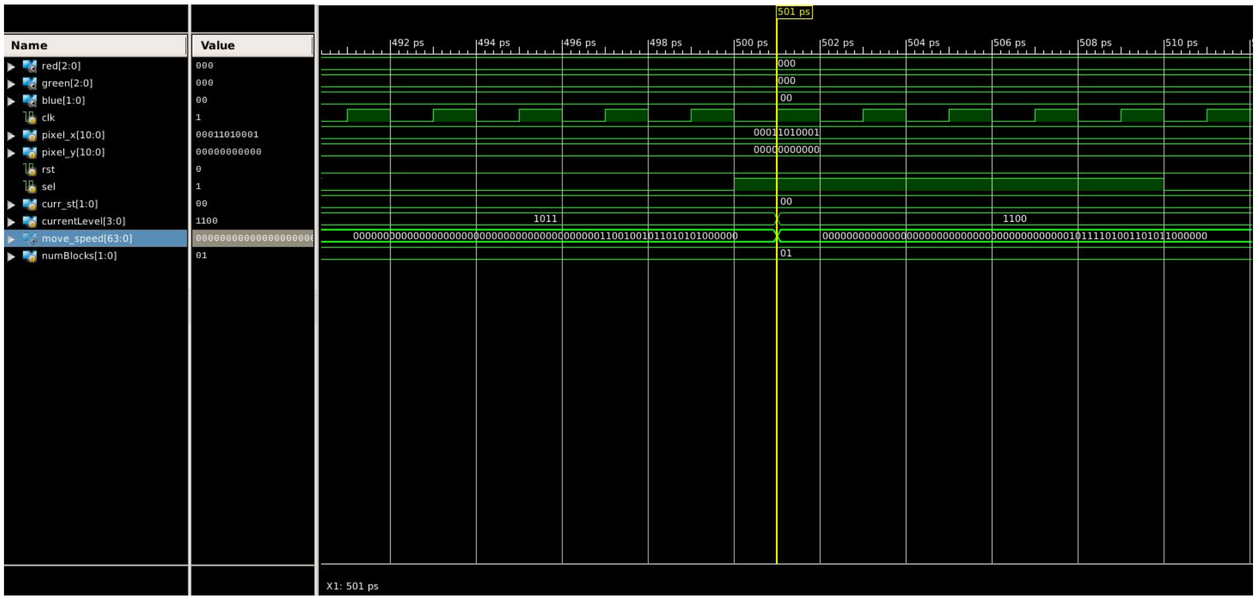


Figure 3.0.5 - In this waveform, one can see the game state being updated once the player aligns a tower of blocks. This forces the all display bits to be green to represent a win.



Name	Value
red[2:0]	000
green[2:0]	000
blue[1:0]	00
clk	1
pixel_x[10:0]	00011010001
pixel_y[10:0]	00000000000
rst	0
sel	1
curr_st[1:0]	00
currentLevel[3:0]	1100
move_speed[63:0]	0000000000000000000
numBlocks[1:0]	01

Figure 3.0.6 - In this waveform, one can see the threshold for the movement clock decreasing (**move_speed**) as progress is being made by the player (**currentLevel** is increasing).

4.0 - Conclusion

During this project, we implemented the arcade game Stacker. A player can control our game by pressing buttons on an FPGA board and the game's graphics are output to a monitor via the VGA protocol. Our design combines debouncers for input, a VGA driver, and a game logic controller into a working game. Furthermore, we wrote a simple testbench that tested the basic functionality of our game.

We encountered several difficulties throughout the design and implementation of our project. Getting the VGA protocol to work properly, getting our button to debounce correctly, and testing the implementation of our game on the FPGA board were all challenges that proved significant during our development process. After much trial and error, we were able to get our VGA output to work by altering our planned screen resolution and utilizing the auto-adjust feature on our monitor to use the proper settings for our board. Proper button debouncing was accomplished by fiddling with timing requirements to detect a valid button press. Due to the fast pace of our game, it often became difficult to discern whether we had bugs in our game's logic. For example, at one point we thought that we were losing at incorrect times. By introducing a temporary "debug mode" to pause the game when we lost, we were able to confirm that no such bug existed and that our game functioned correctly.

If we had time to implement additional features for our project, we would suggest creating a scoring system for the game and adding instructions to the game. For example, by adding a the controls to the side of the display. In regards to the overall lab experience, we recommend purchasing newer computers for the lab to increase the performance of Xilinx. In this course, a plurality of our time was spent waiting for our Verilog to synthesize on the FPGA board. By decreasing the amount of time students spend waiting, you could improve the overall efficiency of this course.

A Note About Unfinished Parts

Our project did not have any unfinished parts with respect to our project proposal. The additional features we would consider adding, if we had more time, are listed above.

Works Cited

[1] FPGA VGA Protocol Implementation
https://github.com/ThePedestrian/FPGA-Simple-Maze-Game-Using-VGA-Output/blob/master/SRC/FPGA-Verilog-Code/vga_800x600.v