CS M152A Lab 4

Michael Wu, Haoyu Yun, Jennie Zheng

Introduction and Requirement

We designed and built a parody of the game "Flappy Bird" on the FPGA board. This game consists of a bird moving horizontally on a 2D plane through a series of columns with gaps in them. The bird and the pipes should be displayed on the VGA with full-fledged graphics.

To move, the character can "flap". Flapping will set the character's upward velocity to a set positive upward value. Then over time, the character's velocity will decrease based on a constant downward acceleration that mimics gravity. The bird's position changes according to the bird's velocity.

The goal of the game is to pass through the gap of between each column. After passing through the gap of each column, the game score will increase by one. Furthermore, as the game progresses and the bird passes through more columns, the columns come at the player at a faster and faster rate to gradually increase the game's difficulty.

If the character hits a column or the ground, this will end the game. The player must press reset in order to reset the game.

We decided to score the project using the following grading rubric.

- Hit Detection (15%) - The game should end when the character hits either the ground or a column. The screen should then pause and no more movement should happen. The reset button will be required to start a new game.
- Column Movement (15%) - The columns should move towards the player and there should be infinite columns. A maximum of two columns will be visible at a time.
- Random Column Generation (10%) - The columns should have a random height for the gap that is continuously generated.
- Graphics (10%) - The player character and columns should be displayed with image files instead of basic shapes on the screen.

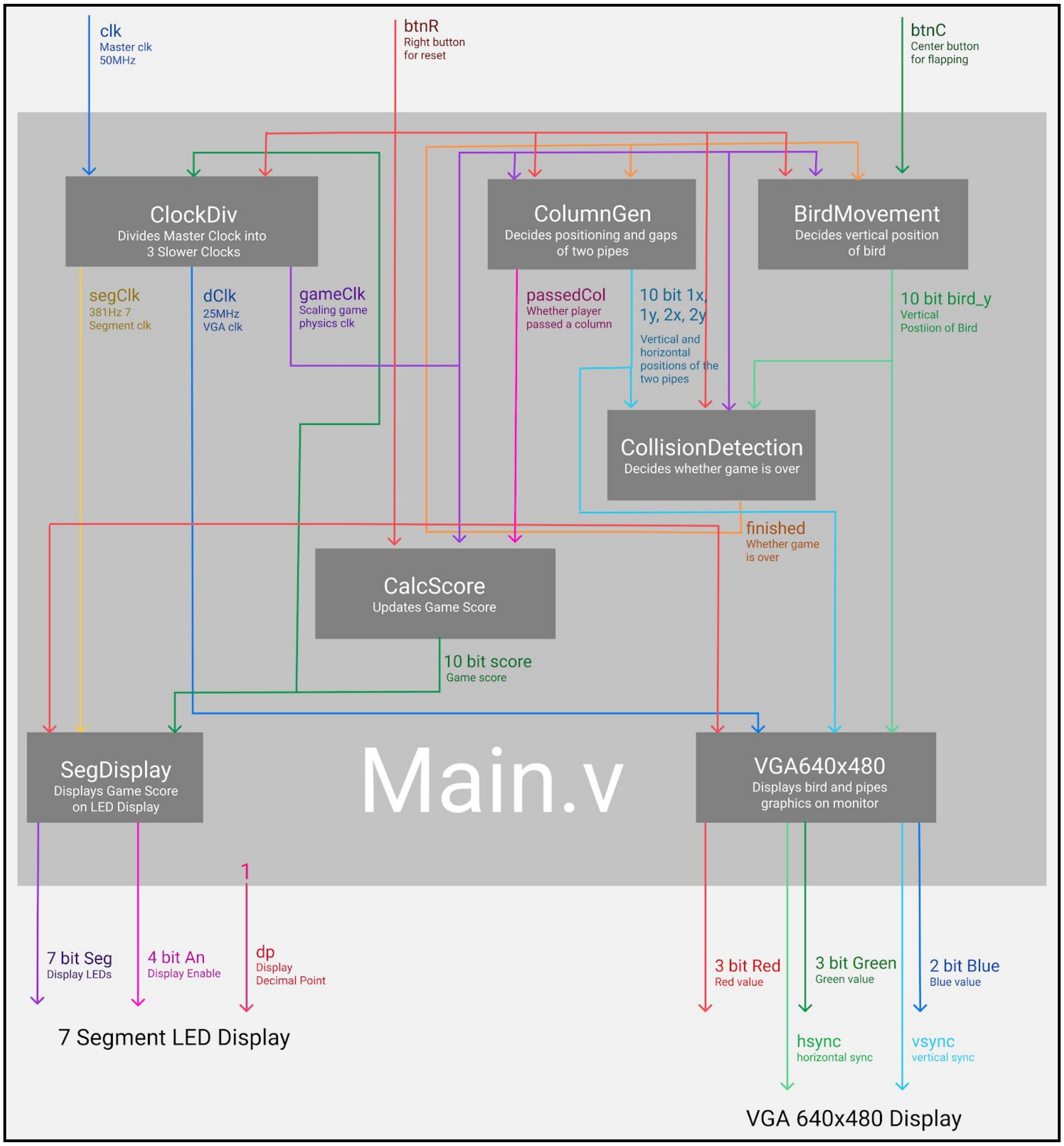
- Reset Button (10%) - When the reset button is pressed the game should reset to its initial state.
- Score display (10%) - The seven segment display should show how many columns the character is able to get through.
- Speed progression (10%) - As the game progresses, the speed of the game should increase in order to increase difficulty.

Design Description

We have the following module hierarchy.

constants.v - contains parameter definitions shared between other modules
nexys3.ucf - contains button/key bindings
main.v - controls overall game logic

- clockdiv.v - uses main clock to create significantly slower clocks for game physics and graphics updates
- birdMovement.v - updates bird velocity and bird acceleration over time and in response to button presses
- columnGen.v - generates columns with random gap locations and updates the position of the columns
- collisionDetection.v - checks whether bird collides with a column
- calcScore.v - updates score based on columns passed
- segdisplay.v - updates LED display on Xilinx board to display the player's score
- vga640x480.v - updates the vga to display the bird and the pipes
    - Graphic folder - creates hardcoded pixel representations of the bird and the pipes



Organization of the Modules Within the Game

The game logic all lies within the main module, which calls numerous other modules. Other notable modules are described below.

In particular, the clockdiv module is used to create three secondary clocks - segClk, dClk, and gameClk. dClk has a constant period and is used by Vga640x480 to render the game graphics. segClk also has a constant period and is used by SegDisplay to render the game graphics.

gameClk, on the other hand, has a period which is a function of the game score. The higher the score, the faster the gameClk runs.

The birdMovement module is used to update the position of the bird. At every posedge of the gameClk, the bird moves its position based on its current velocity and updates its velocity based on a) whether or not something pressed the flap button and b) the acceleration of gravity.

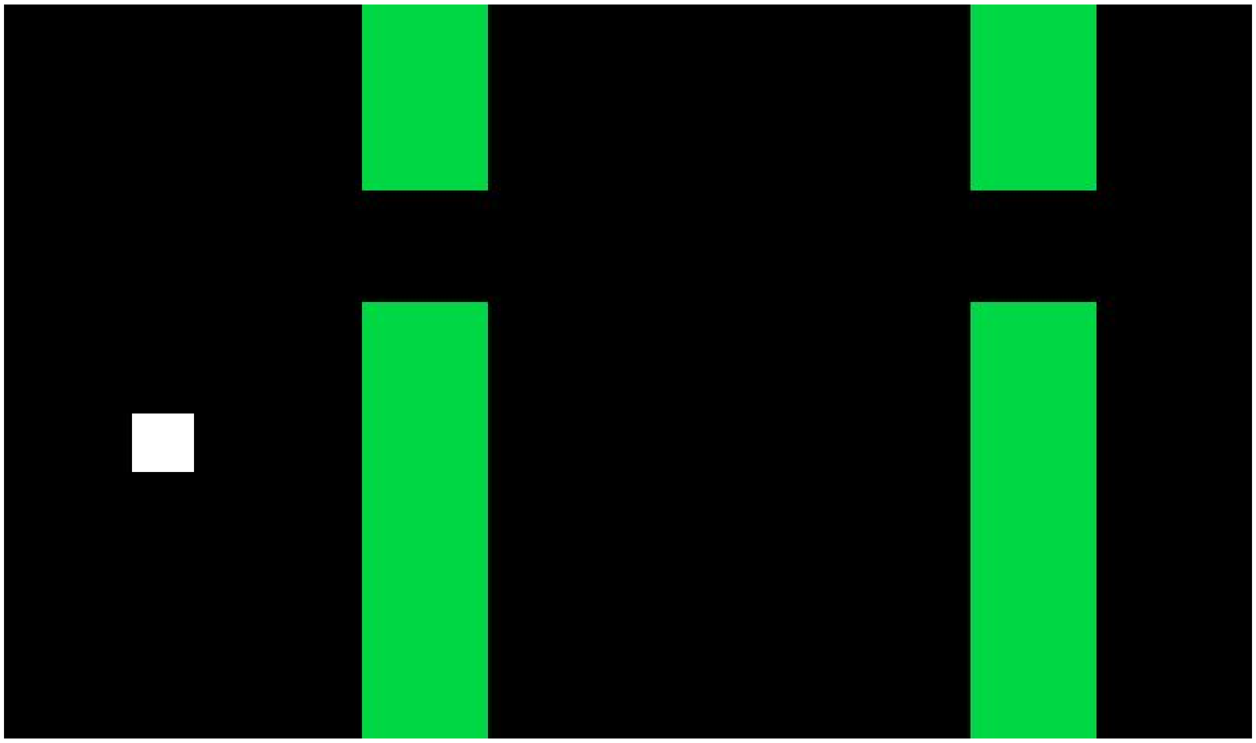
Lastly, the columnGen module is used to update the columns. The game tracks two columns: one at the middle of the screen and another at the right. At every tick of the gameClk, both columns shift a pixel to the left. Whenever a column hits the left wall, its x position is reset to the right side. Furthermore, it’s gap height is changed with a hardware pseudorandom number generator (an LFSR, linear-feedback shift register). We used the pseudorandom number equation q <= {q[14:0], q[7] ^ q[5] ^ q[4] ^ q[3]} in order to create our random gap heights.

Simulation Documentation

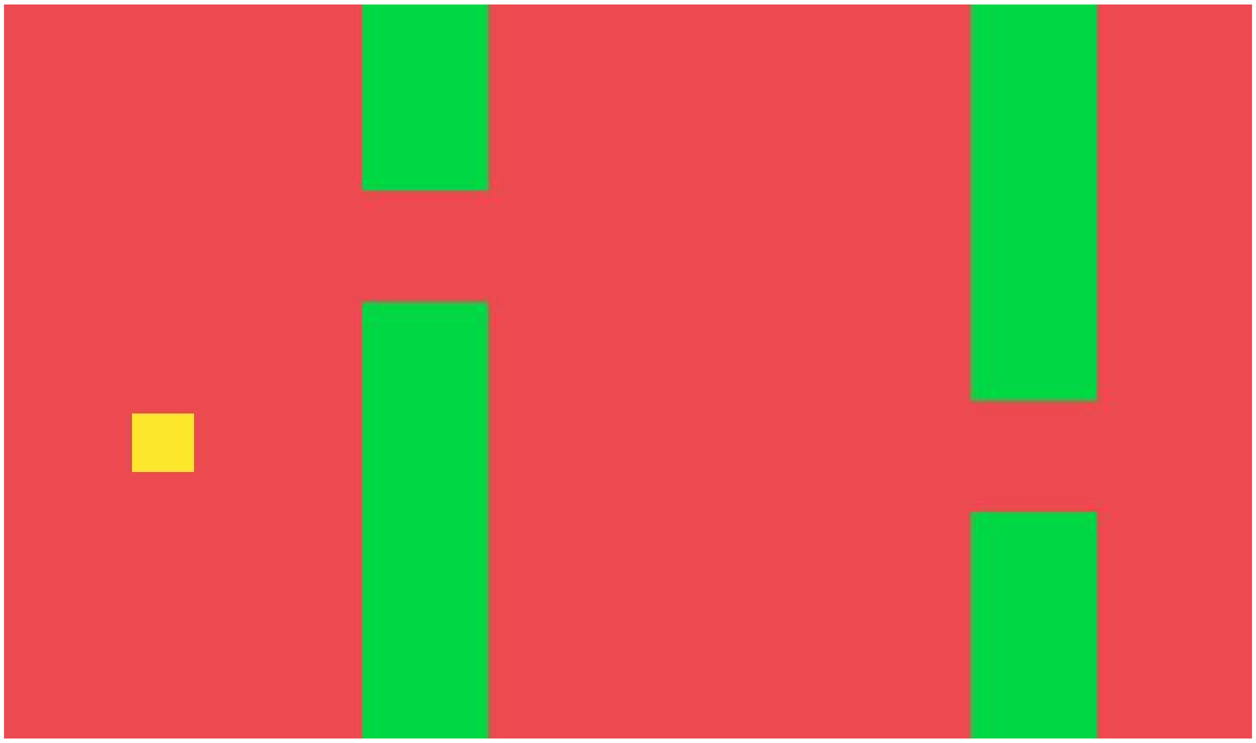
This was a major project where we built many modules independently and subsequently tested them by adding modules one after another to our core program.

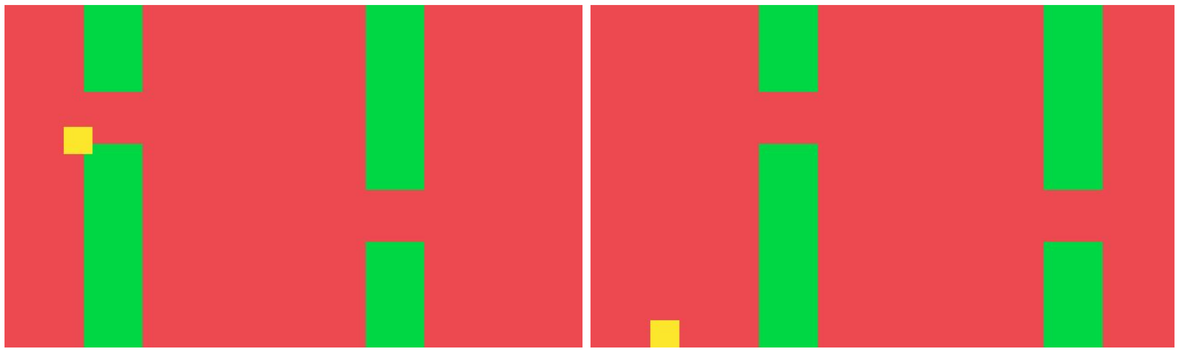
We started out by figuring out how to use the Vga640x480. We cloned the nerf repository on git and modified that project’s clock period, key bindings, and project type until the nerf code worked on our lab computer.

Next, we edited the nerf code’s Vga640x480 to change the demo’s colorful columns into a small white square representing a bird and two green columns representing two pipes.

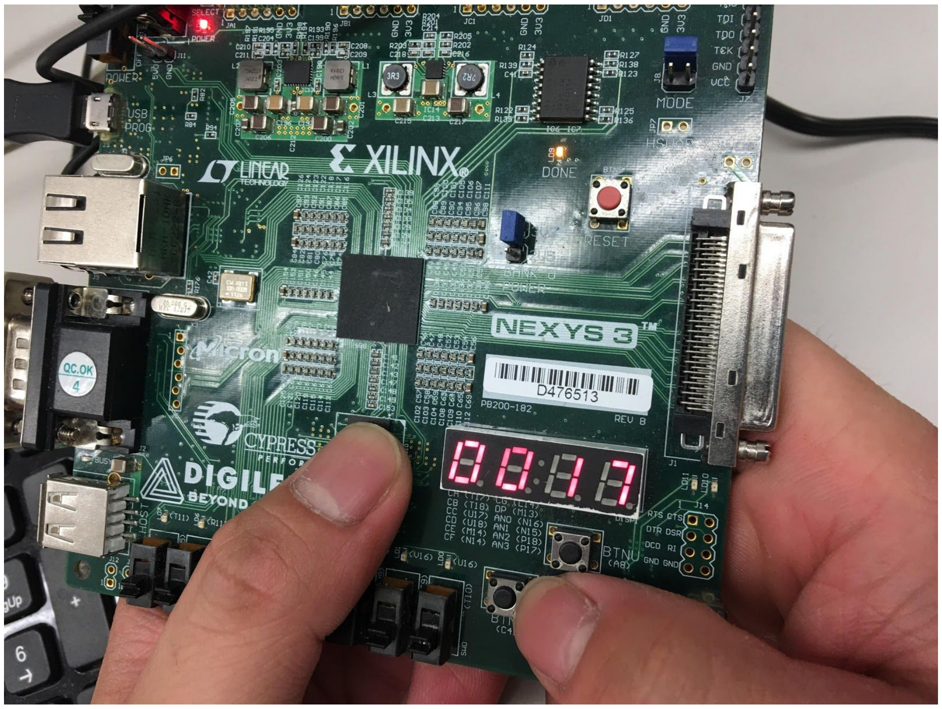

Testing the Vga640x480 with a simplified bird (white square) and two pipes (blue columns)

After this, the rest was a matter of adding modules one by one and then testing the module on our core program. We added the clockDiv, birdMovement, columnGen, and collision Detection modules, in that order, verifying that one module worked fully before moving on to add the next module.

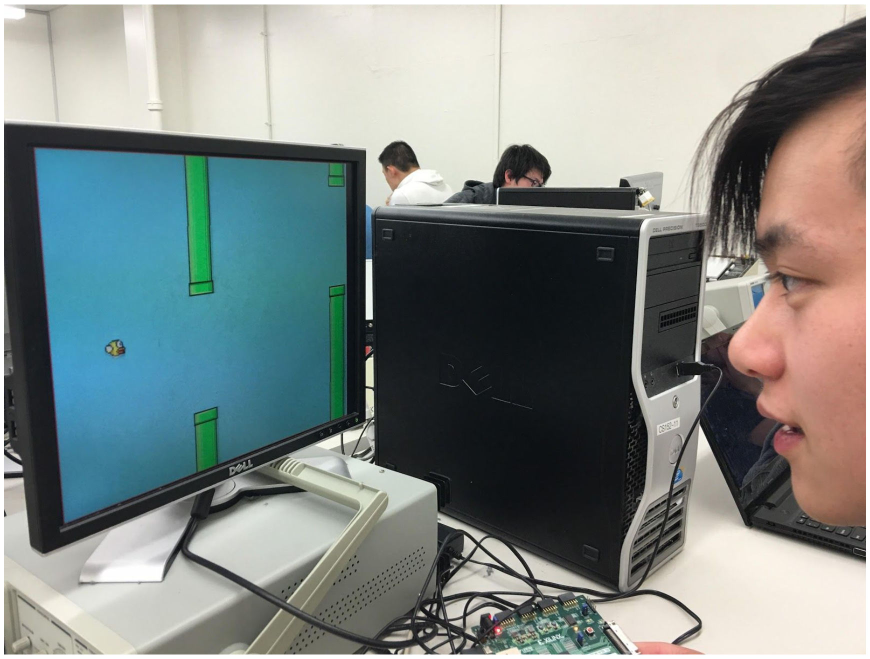

One highlight in our development process was when we managed to get the pipes to be generated with varying heights using a pseudorandom number generator


A second success was when we managed to get the game to come to a full stop when the bird collided into a pipe or hit the ground

Later on, we included the calcScore module to increment the score every time the bird successfully passed a column. Furthermore, we added the segDisplay module to display the score in decimal notation on the board.


Xilinx board displaying button and score during gameplay

Lastly, we improved the graphics in the flappy bird game. We changed the background to a more natural color, sky blue. Furthermore, we obtained a pixelated image of a bird as well as a pipe. Then, we ran the text representation of the images through a Python script in order to generate Verilog code for graphics depiction.


Graphics with pixelated bird and two pipes during gameplay.

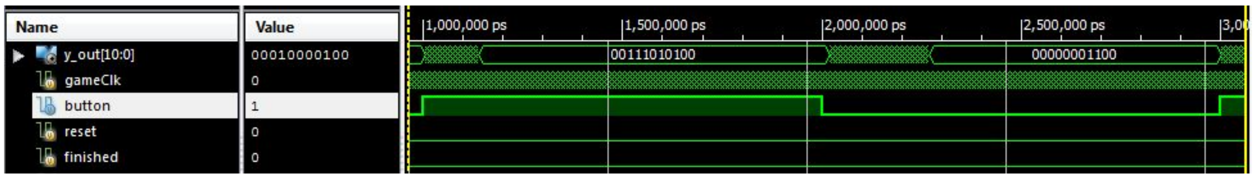
We encountered many bugs throughout our project and employed the debugging techniques learned from previous labs to address them.

In earlier stages of the project, we utilized a testbench to check values related to our core functionality, such as game physics and randomized column generation. We used the Waveforms to verify how the velocity and position of the bird changed. We also used Waveforms to test and achieve random number generation.

In later stages, we integrated multiple modules together, and checked this through actually synthesizing our program and running it on our board. We were able to see how the game performed visually, from the flying path of the bird, to evenly-spaced generation of the columns with randomized gap heights, to collision detection between the bird and column walls.

These observations advised us of places of necessary improvement and also introduced new problems. We further modified our game code to reduce bird velocity to make the bird less jumpy, debugged inconsistent column and gap generation, and added graphics.

Many bugs were caused by incorrect array sizes or initialization our wires and registers. Many variables were accidentally initialized as one-bit values instead of values of nine or ten bits values. We also had to add an extra bit to the wires tracking bird coordinates to account for signedness.



The testing of the gameClk and birdMovement after running into multiple clock issues

Implementing the gameClk, we ran into multiple challenges. Early in development, we used a clock divider to keep a constant-frequency gameClk. We used Waveforms to test the gameClk and verify that bird movement changed based on gameClk ticks. Later, to increase the difficulty of the game over time, we modified our program to gradually lower the number of master clock ticks between when the gameClk is high. This change significantly increased game speed over time, so that none of the play testers could surpass a final score of 13. We used Waveforms to debug gameClk and visually verified that gameClk, combined with the other game components, behaved correctly.

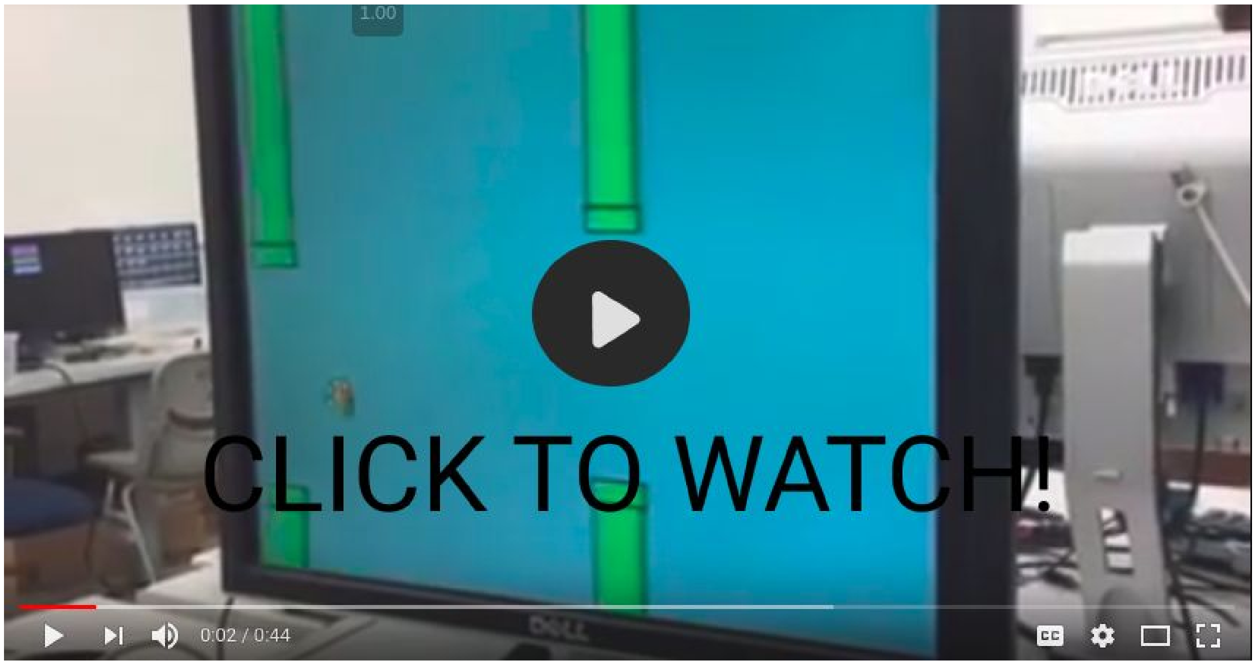
Conclusion

In conclusion, we designed and built a variant of “Flappy Bird” on the FPGA board. We generated columns with randomized gap heights, using a button to maximize bird velocity to “flap” the bird.

This project brought together the skills and techniques we acquired from previous projects - how to write and debug Verilog code and how to design modules to incorporate into a core project. We were able to build a functional project and demonstrate what we learned in the lab course.

Whereas previous projects exclusively used the Xilinx board as well as the IDE Suite software, we were pleased to utilize the computer screen, connected via VGA, for this final project.

Check out our video demonstration below! (Apologies for the poor video quality)


1.00
CLICK TO WATCH!
0:02 / 0:44