Flappy Bird in VHDL

List of Authors (Alex Rice, Brikena Dulaj, Peyton Schmid, Roman Hryntsiv)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: alexrice@oakland.edu, brikenadulaj@oakland.edu, peytonschmid@oakland.edu, rhryntsiv@oakland.edu

Abstract— **Flappy Bird is a mobile game developed by Vietnamese video game artist and programmer Dong Nguyen under his game development company, Gears. This project is a recreation of the game using a Nexys A7-50T FPGA board programmed using VHDL code, a VGA display, a 7-segment display, and a PS-2 keyboard.**

I. Introduction

Playing video games brings children and adults alike great joy. A game can serve as a place to escape from the struggles of everyday life, or in the case of mobile games, a fun way to pass the time while waiting in line at somewhere like the grocery store. The value a video game can bring to life served as the motivation for making this project.

Flappy Bird is a simple, yet highly addictive mobile game that was released in 2013. The objective of the game is to gain as many points as possible. Due to its simplicity and the competition that it created, the game took the world by storm. In the game, the user taps the screen to lift Flappy the bird as it travels through sets of pipes above and below it. Flappy is continuously falling, as the user input mimics the flapping of a bird. If the screen is not tapped, the Flappy sprite plummets, which in turn causes the game to end. With each pipe that the user is able to pass, a point is gained towards their score. The game continues until the user makes an error and Flappy collides with one of the pipes. At this point, the game ends and the user can start over again to try to beat their high score.

The goal of this project was to recreate Flappy Bird using VHDL code and the NEXYS A7-50T FPGA board. The project required using knowledge gained throughout the course of the semester such as FSM design, embedded counters, and how to interface with various external I/O peripherals. To implement the screen control of the original game, a PS-2 keyboard was used. To handle scorekeeping, an on-board 7-segment display was used. A VGA display was used to project the game. Extensive research needed to be done on VGA protocol, as this topic was not covered in class. .

II. Methodology

A. Top File

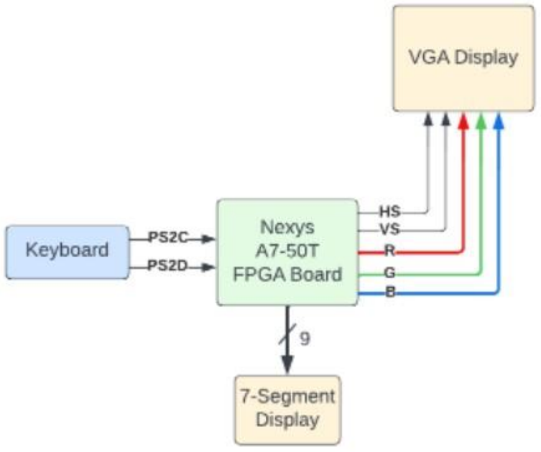

Figure 1. System Block Diagram

To facilitate cleaner design and simplify debugging, the project focused on interconnecting existing program blocks. In a top file, all of the VHDL program blocks were interconnected and port-mapped. The top file includes all system signals, including the PS-2 Keyboard, the 7-Segment display, and the VGA display block. These blocks are then connected accordingly. **Figure 1** depicts an overview of this file. It also includes a finite-state machine (FSM) that is responsible for actually controlling the flow of the game as the user plays. A FSM defines “states” of a program that are entered and exited based on set signal values. This further simplifies program development by defining transitions and signal values based on other signal values. Complicated sequential processes are often optimized by using an FSM to dictate program execution.

The FSM moves throughout the different states of the game and reacts accordingly to signal values generated by the system or received from external input. The FSM places the screen images for the pipes and the bird onto the VGA display and controls their movement during the game. It is also responsible for recognizing collisions between the bird character and the pipe obstacles. The FSM then either increases the score and continues the game, or ends the game upon collision. It also coordinates the timing of the game by only switching between states on the rising edge of the input clock. **Figure 2** below depicts this FSM.

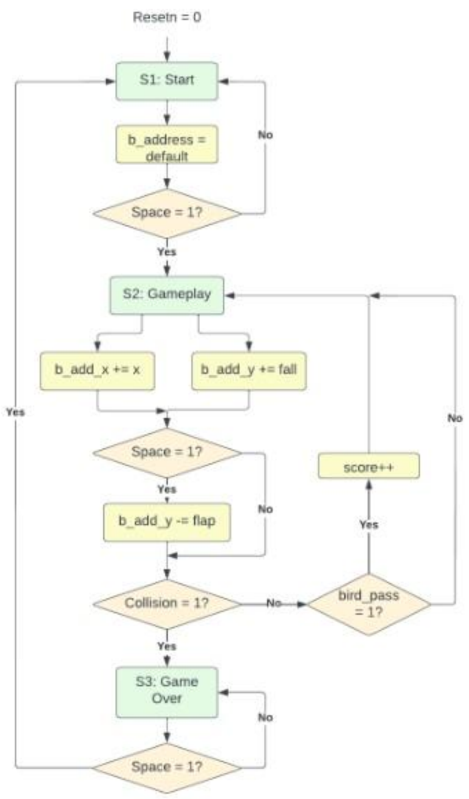

Figure 2. Main FSM

The game begins by placing the bird and obstacles on the screen in the "default" starting position. This is the first state of the game, where the program is idle until the user presses the spacebar to enter the game. Once the spacebar is pressed, the game begins. This gameplay state is the second state of the game, and includes a bulk of the program.

To understand the gameplay state, the design of the game must be understood. The original game moved randomly generated pipes on the right side of the screen, and moved them linearly towards the left side. The bird only moved vertically to avoid these pipes. In this implementation, the pipes remain stationary, while the bird travels across the screen.

During the gameplay state, the bird's address is automatically incremented in both the x and y directions by **25** and **50 pixels** every clock cycle, respectively. If the user presses the spacebar, the y direction is decremented by **50 pixels**, making the bird go up instead of down. If a collision occurs, meaning that the bird position overlaps with the pipe obstacle, the game shifts to the "game over" state. If a collision does not occur, the program checks to see if the bird passed an obstacle. If so, the score is incremented. After this, the program returns to the beginning of the gameplay state. In the third "game over state," the program is again idle until the user hits the spacebar to play the game again.

B. VGA

The original plan for the project included using MATLAB to convert an image into a binary text file. The images that were going to be used are shown in **Figure 3** below.

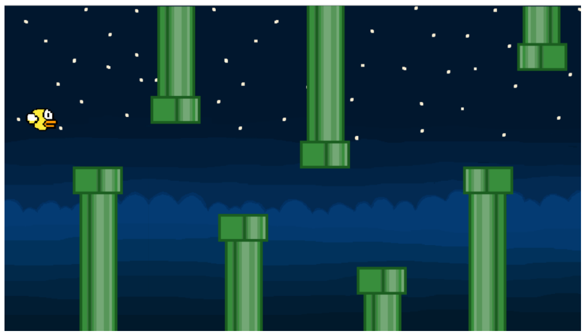

Figure 3. Game Design with RAM Implementation

This file would be stored in a RAM (random access memory) block to implement the VGA images of the pipe obstacles, a background and Flappy. **Figure 4** below displays the block diagram of this setup. However, this was not possible to design adequately in the time allotted (covered in detail in the results section). Instead, a more simple VGA display controller, shown in **Figure 5**, was used to manually create the scenery, obstacles, and sprite in the available 640 by 480 pixel space. Embedded in both the VGA control block is the display block, shown in **Figure 6** below. This block generates the clock tick of the VGA display and cycles through the coordinates of the display.

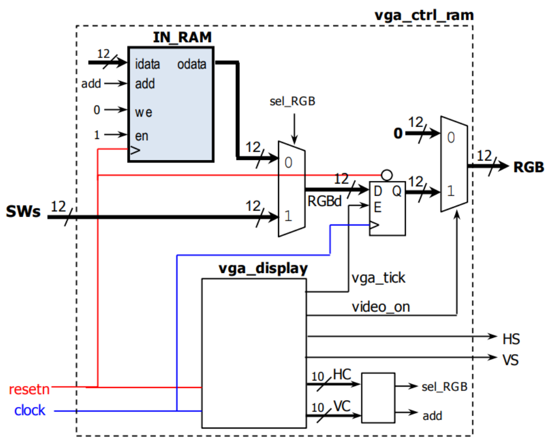

Figure 4. VGA Control Block with RAM

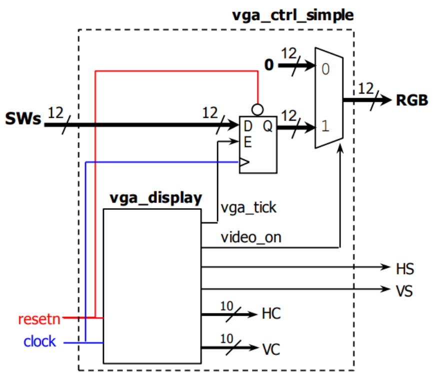

Figure 5. VGA Control Block Without RAM

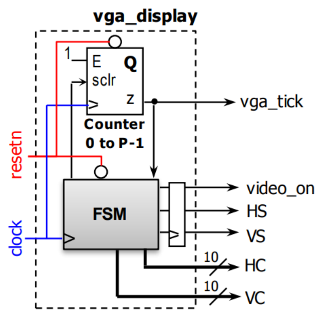

Figure 6. VGA Display Block

To simplify the project during design all images were originally created as rectangles and using only two colors. **Figure 7** below shows the original design of the game without the incorporation of RAM.

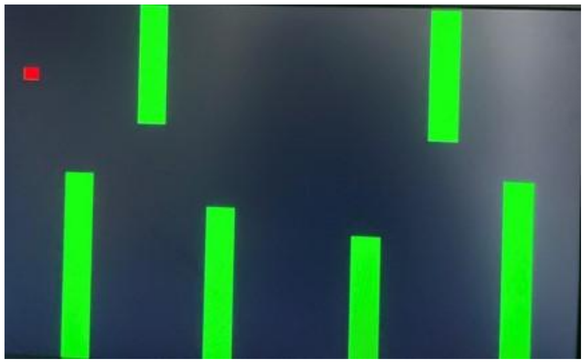

Figure 7. Original Game Design Without RAM

Each pipe was created by defining a space on the screen using the variables HC and VC. These ranges then had RGB outputs set to green instead of the background color. The color of Flappy was defined similarly, except using red instead of green. The main FSM was then used to accomplish the moving of the images and facilitate gameplay. In the initial state, the coordinates are set so they are suspended in the top left corner waiting for the user to start the game. When the user presses space the game begins and moving of the VGA image starts. An embedded counter was used within the main FSM to move between states every second where the coordinates of Flappy move down and to the right with each state change. If the user then presses the up key on the PS-2, Flappy's coordinates are moved up to avoid the pipes on the bottom of the screen.

To track when collisions occur, 'sensitive' areas of the VGA field corresponding to pipe placement were defined. If the position of the bird is ever in the same coordinates as one of the pipes, a collision has occurred, causing the program to enter the "game over" state. In this state, the game ends and the display flashes "GAME OVER" in black to alert the user. Flappy will then be placed back in the start position of the first state until the spacebar on the PS-2 keyboard is pressed. Once this was all created using the simple blocks, a better bird was implemented by using pixel art and more specific coordinates. This updated design is shown in **Figure 8** below. .

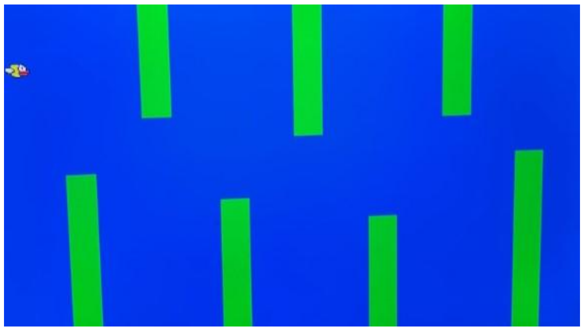

Figure 8. Final Design of Game

C. *PS-2 Keyboard*

To implement the user controls, a PS-2 keyboard was used. PS-2 (Personal System 2) refers to a hardware interface used to connect a keyboard or mouse to a PC. A PS/2 port has a bidirectional, synchronous serial channel. Although this protocol is bidirectional, data transmission favors the device over the host. It transmits data from a 6-pin Mini-DIN port, displayed in **Figure 9** below. The device sends a byte in a serial frame on the data line as the clock toggles for each bit. The host controls the clock transmission, which gives it overall control, although it does not normally perform the majority of data transmission. If the host requires communication to be stopped, it pulls the clock signal low.

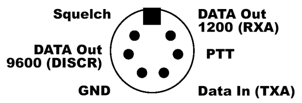

Figure 9. 6-pin Mini-DIN Connection of PS/2 Devices

The system favors device-driven data transmission because it does not need to obtain control of the channel before it can transmit. If the host is to communicate with the device under PS/2 protocol, it must first pull the clock and data signals low. Then, it waits for the device to release control and transmit a clock signal while the host sends a serial frame on the data line. This data is captured on the rising edge of the clock, while data transmission to the host from the device is read on the falling edge of the clock.

The keyboard transmits one byte to represent the press of a key. Exceptions to this one byte in length are the 'SHIFT,' 'CTRL,' and 'PAUSE' keys. Keys that are not extended have a prefix of 'F0,' while extended keys have a prefix of 'E0.' Every key has its own scan code definition after the prefix. **Figures 10** and **11** below summarize these codes with reference to a physical keyboard. This code is sent to the host so that user input can be displayed on the PC display. When the key is repeatedly pressed, the byte is sent to the host every 100 ms until the key is released.

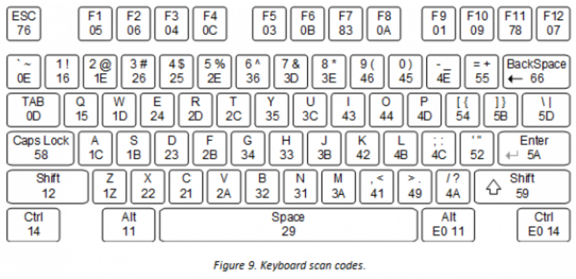

Figure 10. PS/2 Keyboard Scan Codes

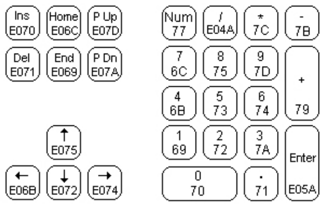

Figure 11. PS/2 Extended Key Codes

To read from the PS/2 keyboard, code was used from the course database. This code gives an output of the keyboard scan code. This was implemented as a block in the game control system. The keyboard scan code was decoded manually within the main program. This PS/2 control block is depicted in **Figure 12** below. From here, this user input was used to control the movement of the character, in addition to starting and stopping the game. Continuous presses of the keyboard were ignored, as the Flappy Bird interface requires a key to be pressed every time that the bird 'flaps.'

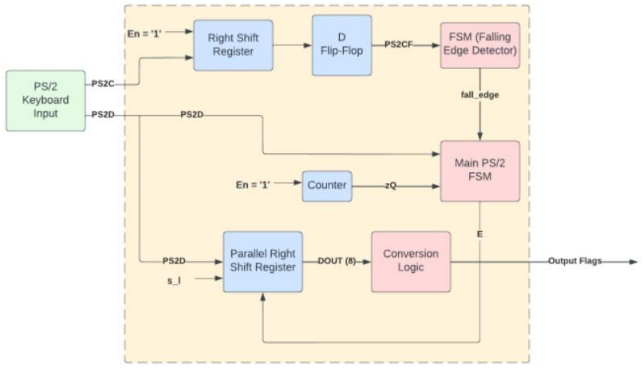

Figure 12. PS/2 Keyboard Block Diagram

In the user interface for the Flappy Bird game, only two keys on the keyboard were used as controls. The spacebar key of the PS-2 is used as the input to start and restart the game, and the press of the up arrow controls the movement of Flappy. The game begins with Flappy stationary in the first state of the FSM and remains there until the 'start' signal is asserted. A flag is set when the spacebar is pressed, which is used within the FSM as the 'start' signal of the game. Once the game begins, the user must continue pressing the up key to keep Flappy in the air. When not pressed, Flappy will continue falling towards the bottom of the screen and run into a pipe or the bottom border.

D. 7-Segment Display

The Game keeps a tally of the users score, or how many pipes they have successfully passed, with the use of one of the on board 7-segment displays of the NEXYS A7-50T FPGA board. A 7-segment display is an electronic display device which is used to display hexadecimal characters. **Figure 13** below shows a seven segment LED display. As indicated by the diode equivalent circuit on the left, each segment is an individual LED. This is a Common Cathode (CC) display meaning all of the cathodes (or negative terminals) of the segment LEDs are connected together.

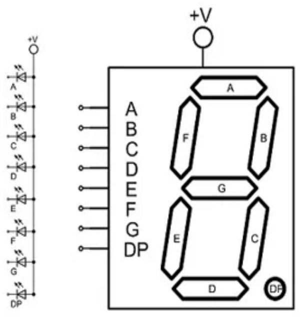

Figure 13. 7-Segment Display

Digital equipment systems use 7-segment Displays for converting digital signals into a form that can be easily displayed and understood by the user. This information is often numerical data in the form of numbers, characters and symbols. Common anode and common cathode seven-segment displays produce the required number by illuminating the individual segments in various combinations. LED based 7-segment displays are easy to use and understand. Because of these factors, it was decided to use 7-segment displays for the project to keep track of the score. Every time the player successfully passes an obstacle, the count on a 7 segment display increases by 1. This continues until the player hits an obstacle.

To implement the 7-segment displays, the control block displayed in **Figure 14** below was included in the project. In this block, a multiplexor selects which display to work with. The value to be shown on the display is sent on through the multiplexor to a hexadecimal to 7-segment decoder. A FSM of a counter from 0 to 7 with enable sends an output to the multiplexor and the display, which enables the correct segment and sends over the corresponding decoded data for that segment.

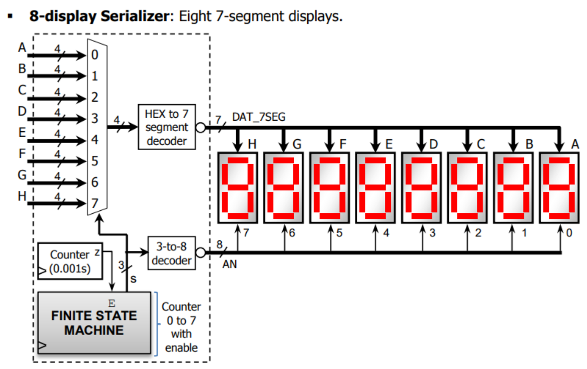

Figure 14. Eight Display 7-Segment Serializer Block

III. Experimental Setup

The heart of the experiment setup is the Nexys A7-50T FPGA Trainer Board. The Nexys board was used to control the external peripherals used in the project (VGA and PS-2), as well as the on board 7-segment display, using code written in VHDL on vivado. To start developing the code for the game the individual pieces of the system were worked on separately to make sure they all worked before attempting to connect them in the top file. The VGA was the first thing handled.

The VGA screen was tested extensively with many different ways of being implemented (covered also in methodology and results). It was attempted to be operated with a block RAM being used to place images on the screen which were converted to the form of txt files. Also a method of creating VGA animations called 'sprites' was attempted. These two approaches were unsuccessful and so a third method using a simple VGA controller to place color in specific horizontal and vertical coordinates was used. Once the simple VGA method allowed images to be created on the screen, the coordinates for all of the pipes and the bird were designed and implemented using trial and error as there was no way to see that the screen image is correct within a test bench simulation.

To implement the keyboard, a VHDL example code from Professor Llamocca's library of VHDL programs was used as a starting point. The code was run through a test bench, shown in **Figure 15** below, to verify that it was functional from a software perspective. Then, a preliminary hardware test was performed by controlling two of the onboard LEDs with space and up arrow, the two keys we intended to use for the game. The test worked, but needed to be adjusted slightly for the purpose of the game by making sure the key was no longer recognized once released.

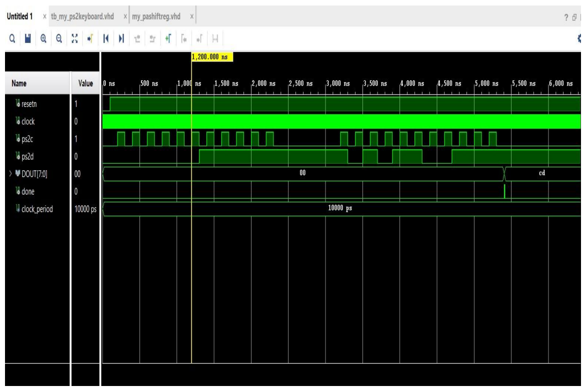

Figure 15. PS/2 Keyboard TestBench

The 7-segment display was handled in almost the exact same fashion as the keyboard. An example code was used and it was tested first in a testbench. After the testbench, a hardware test was performed by incrementing a count on one 7-segment display with the press of a button. The 7-segment and keyboard were both then ready to be added to the topfile.

The final step in creating the game was building out the FSM in the topfile that would control everything happening during play of the game. At this point in the design it was again not really possible to test the game using something purely software based like a timing diagram from a test bench. So to finish a trial and error approach was taken. Small chunks of the FSM were written at a time, for example the movement of the bird with the counter, to verify each section was working and then a new feature was added each time the last was successful. First the falling of the bird was implemented, followed by the flying up with the keyboard up arrow button. Then the realization that a collision had occurred was added and was used to end the game. Finally the 7-segment display was added to increment each time a pipe was passed without collision keeping track of the users score.

IV. Results

Initial testing of the game did not go well. While attempting to use a more complex VGA display controller that included a block RAM many issues occurred. MATLAB was used to convert hand drawn images of Flappy the bird, the pipes and an intricate background into text files which were then to be placed on the VGA from the RAM. However, the block RAM caused synthesizing of the circuit to take a very very long time (near an hour or longer). This became an infeasible way to code as debugging became impossible. Even trying to correct a small error would take hours. When the circuit did finally synthesize and implement, the images were only placed on a miniscule slice of the screen due to the low memory. An attempt was also then made to accomplish the animation using sprites. This too was unsuccessful and no solution could be found for getting the sprite method to work. Because of this, the complex VGA approach was switched to the much more simple one explained in the methodology section above.

Once this switch was made, a more basic but functional version of the game was able to be created. The main portion of the code written was the FSM. Challenges were faced within the FSM with getting the bird falling continuously using an embedded counter while also recognizing the press of the keyboard button to lift the bird. This was solved by increasing the time of the counter slightly so that more time was available to recognize the press of the keyboard. This did lower the difficulty of the game for the user as Flappy now would fall much slower. However, it was deemed a worthy trade off for the time being so as to have a fully functioning program which included all of the intended peripherals.

V. Conclusions

During this project, a successful working version of Flappy bird was recreated. It may be a little less aesthetically pleasing and a lot more simple to play, but the game is functional and could be expanded upon to make it more like the original. If given more time, it could be possible to implement a more functional RAM so that the better looking artwork could be used for the game. More time to debug would also make it possible to speed the game up, and possibly even find a way to make the pipes regenerate so that the game will continuously run till the user makes an error. As it stands, the game is limited and a bit crude, but it's fully functioning with no errors.

VI. References

1. U. Zoltán, "Nexys-A7-50T-OOB" *GitHub*, 2006. [Online]. Available: https://github.com/Digilent/Nexys-A7-50T-OOB/blob/master/src/hdl/Ps2Interface.vhd. [Accessed: 01-Apr-2022].
2. D. Llamocca, "VHDL Coding for FPGAs," *Reconfigurable Computing Research Laboratory*.

[Online]. Available: http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html. [Accessed: 1-Apr-2022].

3. S. K, “VGA Display Controller,” *Digilent Reference*. [Online]. Available: https://digilent.com/reference/learn/programmable-logic/tutorials/vga-display-congroller/start. [Accessed: 01-Apr-2022].
4. A. Brown, “Nexys A7 Reference Manual,” *Digilent Reference*. [Online]. Available: https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual. [Accessed: 1-Apr-2022].