Flappy Bird

line 1: Andy Choi
line 2: *Computer Systems Engineering*
line 3: *University of Auckland*
line 4: Auckland, New Zealand
line 5: scho413@aucklanduni.ac.nz

line 1: Charles Lee
line 2: *Computer Systems Engineering*
line 3: *University of Auckland*
line 4: Auckland, New Zealand
line 5: slee705@aucklanduni.ac.nz

line 1: Zion Song
line 2: *Computer Systems Engineering*
line 3: *University of Auckland*
line 4: Auckland, New Zealand
line 5: sson861@aucklanduni.ac.nz

Abstract—In this project, the group made multiple codes to make a game. We connected the code we have built in the project.bdf as this is responsible for the program to compile in the FPGA board and display over the VGA screen.

The code we created in this project includes menu, game over screen, display, game FSM, LFSR, bird_move, bird_mech, mouse and pipe move. The game over and menu codes are responsible for displaying the letters over the screen to show the current status. The display code is responsible for displaying the menu bar, game over screen, and it also sets the colour of the display. The game FSM is used to change the game mode to single or practice mode as well as put the user back to the menu screen and after they die, it goes into the game over screen. LFSR is used to generate the number which will be used for pipes. The code bird_move is responsible for the game mechanic as it contains a collision system, score system and information related to the bird, such as sprite size and lives. The bird mech code displays the current status of the bird during the game. The mouse is used to take input of the mouse click. The code pipe_move uses LFSR to generate the random size of pipes. It moves the pipe from right to left, and it increases the level depending on how well the user plays.

I. Introduction

As part of the COMPSYS 305 curriculum our aim is to build a flappy bird using VHDL which can be played through the use of FPGA board. This base project enables its participants to integrate the basic principles of integrating our understanding of VHDL into FPGA board and displaying over VGA screen.

A. *Mini board bdf*

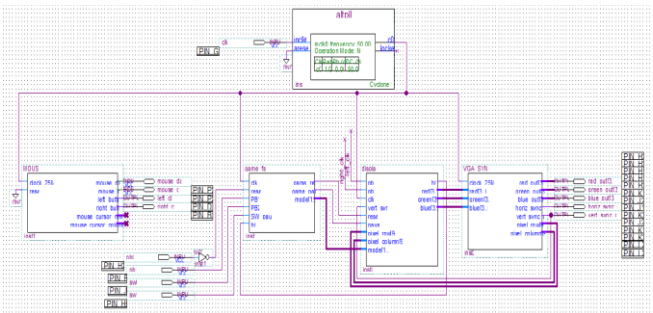

Figure 1

Figure 1, "the MiniProject.bdf" conveys a schematic diagram of the flappy bird project. Instead of connecting every component file, we have decided to use a few selective components which contain multiple components.

We have simplified the overall design to improve the overall design of the circuit. Multiple components are imported into a single component; the display component contains the menu, game over, and flappy_bird component.

B. *Game FSM*

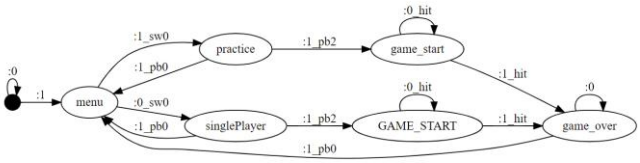

Figure 2

This component is the Finite State Machine (FSM) which decides which window it should be displayed depending on which button has been pressed and/or which switch has been turned on or off and/or if the bird has lost all its lives or hit the ground. At the start, it is in the menu state. When the start button has been pressed, the state will change to either single or practice depending on the switch. If the switch is off, then it will go to single, if it is on, then it will go to practice. When in either state, the component will continuously check for whether the reset button has been pressed or whether the flag, hit, is '1'. When reset is pressed, the state will go back to the previous state. When a hit is '1', then it will move to the next state, game over state. In this state, nothing happens unless reset is pressed. When reset is pressed, the state goes to the menu state to start again.

C. *Menu Screen*

When the user installs the code into the FPGA board, it displays the main menu. The Menu file displays team name and game options which briefly describes what each button will do. These include different game modes which users can select by changing the switch 0, pausing the game by turning on switch 1, resetting to the menu screen by pushing the pb 0 button and playing the game by pressing the pb2 button. We imported the char_rom component, and for libraries, we imported NUMERIC_STD, MATH_REAL, LOGIC_SIGNED, LOGIC_ARITH, LOGIC_1164. The use of char_rom enabled us to set a range for characters to show

over the VGA screen. Then, we set the position of each character. The address of each letter is in decimal format, and we called the address to display on the screen. The trade of our design for the menu screen is that setting the wrong range and position results in the screen showing broken characters.

D. Game Over

When the game ends, it displays “game over” on the screen. The concept behind the code is similar to the menu screen as we imported the same libraries and char_rom as the menu file. First, we set up the range of where it can display the characters, and it will show “GAME OVER” and “SCORE-”. We get the score from the score output of bird_move which is then carried over to the output of flappy_bird.

E. Display

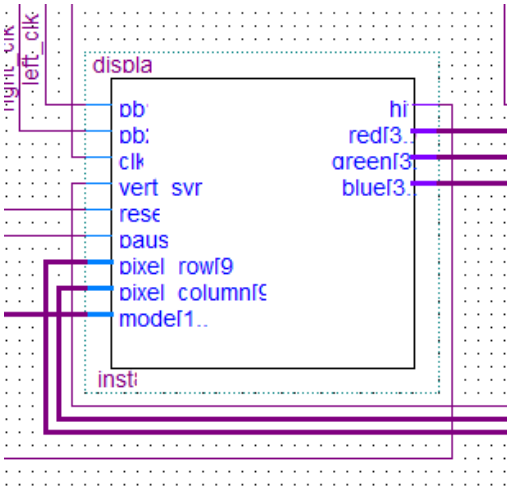

Figure 3

This display component is responsible for displaying the image over the VGA screen as it selects the colour of the objects to be displayed on the screen. It then gives an output of Red, Green, and Blue that is carried to vga_sync so that it is displayed on the screen.

Depending on which mode the game is in, the output of Red, Green, and Blue are different because the address used for each of the colours are different thus giving a different display.

When the game starts, the colour of the pipes is set as green, the colour of the bird is red and then uses blue as a background colour. The colours are set using the output Red, Green, and Blue.

It is also responsible for displaying the game over and the menu screen. The background colour of the game over screen is red, and the text is white, whereas the background of the menu screen is blue and the text is still white.

*F. **LFSR***

We have used the Galois LFSR (linear feedback shift register) over Fibonacci LFSR as Fibonacci LFSR causes latency as the XOR gates are placed on the feedback path. In contrast, Galois LFSR has no latency as the XOR gates are placed between the registers.

Galois LFSR will generate a random number to assign a random length to the pipe. We initialised the size of the vector as seven and a value of random as “01010101”. All components are synchronous with the clock. The initial value of the rand value changes to “10101010” when it resets, as this randomises the pattern of the number generated. The shift register will examine XOR between arrays 6, 4, 3, 2 and 0. The result of this will be put into the least significant bit of the rand value.

When the group was designing an LFSR, instead of making multiple registers and connecting them to create a single module, we decided to make one component that contains the same functionality. This method improved the overall performance of the code as it probably uses fewer logic elements.

*G. **bird_move***

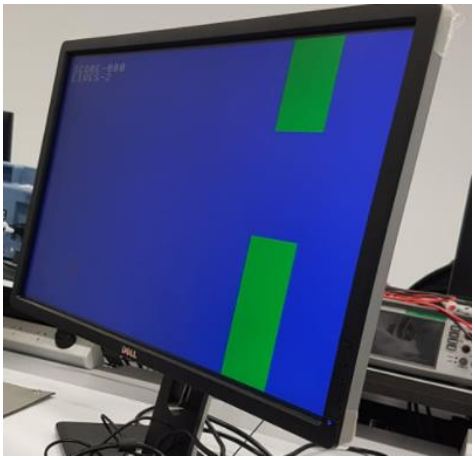

Figure 4

The component bird_move is responsible for containing the bird mechanic and the collision system of the game. When the game starts, the bird’s starts at 100 on the y-axis, and we initialise the bird to have 3 lives. The game physics we applied to our game is simple as a bird moves downward constantly when a mouse is not pressed, and every time the player left clicks a mouse, it will jump upwards. We set the gravity as 3, and it will move upward by 5. Although the bird moves upwards and downwards, they cannot move on a horizontal axis. When a bird jumps, it constantly jumps upwards when the player presses on the mouse. As a result, we installed a flag that prevents a bird from constantly jumping while the mouse is pressed. Thus, the bird only moves upward once when the mouse is clicked.

The life mechanic of the game applied to a bird is that it loses 1 life every time it hits a pipe. So, when a bird loses 3 lives, the game will end. However, if it hits the ground, the game ends instantaneously. The score mechanic of the game is also in this component, and it is made so that when the bird goes through the pipes, the score is increased by 1. These are possible because in the input for the component, there are the positions of the pipes which we can then use to see if the bird has hit the pipes or gone through the pipes.

The bird size is set as 16 pixels, and we initialised the bird as character O as a sprite because we could not import the

mif file for the image we have created for the pipes and the bird. However, there was also an issue with our sprite as the VGA screen does not display clear O, but instead, O is displayed across the whole column and it displays the section where the bird position is at. This is because our sprite is trying to show all across the column instead of in a fixed state.

For the bonus feature, the group did not have enough time to implement the health package system. As a result, we decided to give a bird extra life every time it passes 25 pipes. The maximum number of lives it can get is up to 3. This means even if the bird goes past 25 pipes, if the bird has 3 lives, the number of lives does not go up.

The maximum number of possible points you can get is 999, and if a player gets over the maximum points, it will reset back to 0. The point is divided into 3 separate 6 bits arrays; hundredth, tenth and ones. It is in decimal format which is then changed to text in char_rom, and each time it hits "111001", it will update the following array and reset the current one.

For further improvement, we would also like to implement sprite for a bird and import images for pipe and background to improve the overall quality of the game.

*H. **pipe move***

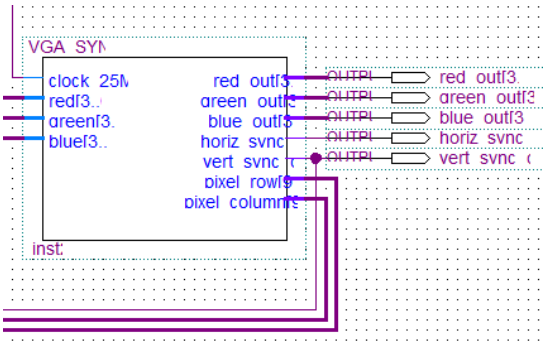

Figure 5

This component generates the position and the movement of the pipe. The top pipe length is determined by the randomised length we get by using the number generated by the LFSR. Then we add to the bottom of the top pipe a fixed length which we have decided to be 75. This gap is the section the bird goes through. We have decided it should be 75 because any more than this would make it too easy and any less would make it too hard. After the gap, it would fill up the rest with a pipe which is the bottom pipe.

Initially, the speed of the pipes is set to 2 by setting the pipe_x_motion signal to 2. The pipes will move from right to left, and this is done by subtracting pipe_x_motion from the pipe_x_position. The same logic is applied to both modes: practice and single-player mode.

In practice mode, there is no levelling system; hence the players can freely practice and expand their skills to understand our game mechanic for this game. In single game mode, we have a levelling system to make it challenging for the players to test their limits. We have set the level to increase when a player passes every ten pipes with a maximum level of 10. However, if the player hits a pipe, the pipe count reset back to 0 and start counting again. So, for example, if the player dies at the 9th pipe, the next level will begin at the 19th pipe. To accomplish this, we have made a variable called level_score, and every time a player hits a pipe, reset is called to update the value for level_score by adding ten extra points. This indicates that the level_score value is updated, and an updated value is the point where the game level increases. When the level is increased, the speed of the pipe movement will also increase by one. This incrementation is done by adding one to the pipe_x_motion. Also, when the player dies, the level resets to 1. This is to reduce the difficulty at higher levels. The special feature is not affected by the player's death. This means even if the player dies at 9th pipe, the next extra life is at 25th pipe, 50th pipe and so on.

*I. **Flappy bird mech***

When the game is played, the player needs to see how many points they have accumulated and how many lives they have. The component Flappy bird mech is used to display the score and the total amount of life a bird has during the game. It is also used to bring in the components to create and move the pipes and the birds. To achieve this, we have imported other components such as char_rom to display the text, pipe_move to create the pipe and move the pipe across from right to left, and bird_move to create and move the bird down and up when the left click is pressed as well as get the score and the number of lives left.

The lives and score values are collected from the bird_move component and it is printed using the char_rom.

However, the texts are hidden when the pipes intersect with the text positions. This is because the pipes have higher priority. In the future, we would like to solve this problem. To solve this problem, we would need to get the position values of the texts and check if the pipes overlap those positions. If the pipes intersect text positions, we need to set the pipe so that it does not display pipe at those positions.

*J. **mouse***

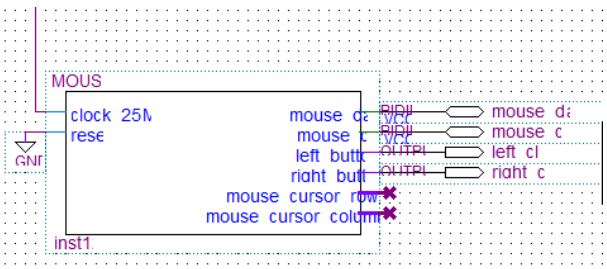

Figure 6

This component gets the input of the PS/2 mouse and gives an output of the position of the mouse and whether the mouse has been clicked or not for both left and right. We can check if this code is working by looking at the 7

segments led display and see if the full stop in the led display is on. This full stop will turn off with the mouse click.

*K. **Resource usage and performance***

Flow Summary	
Flow Status	Successful - Sun May 30 01:10:53 2021
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	MiniProject
Top-level Entity Name	MiniProject
Family	Cyclone III
Device	EP3C16F484C6
Timing Models	Final
Total logic elements	1,090 / 15,408 (7 %)
Total combinational functions	1,059 / 15,408 (7 %)
Dedicated logic registers	265 / 15,408 (2 %)
Total registers	265
Total pins	23 / 347 (7 %)
Total virtual pins	0
Total memory bits	45,056 / 516,096 (9 %)
Embedded Multiplier 9-bit elements	0 / 112 (0 %)
Total PLLs	1 / 4 (25 %)

Figure 7

As shown in figure 7, we have used 1090 logic elements out of possible 15,408 logic elements which is 7% of the total. This means that the code will run fast and consume less power. Also, for the memory bits, we have used 9% of our memory. Our code does not take up much memory and hence when we run the game, it does not lag or shows no underperformance.

Slow 1200mV 85C Model Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	94.78 MHz	94.78 MHz	inst|altpll_component|auto_generated|pll1|clk[0]	
2	183.96 MHz	183.96 MHz	VGA_SYNC:inst2|vert_sync_out	
3	401.28 MHz	401.28 MHz	MOUSE:inst12|MOUSE_CLK_FILTER	

Figure 8

We have used the clock frequency of 50MHz because the DE0 board includes the 50MHz clock. As shown in the figure 8, the maximum operating frequency of our implementation is 94.78MHz. This gives the maximum frequency of VGA of 183.96MHz and the maximum frequency of mouse of 401.28MHz.

Our flappy bird game runs smoothly without any lag and the system is not taking up much of the memory, hence our code does not need improvement at this stage.

ACKNOWLEDGMENT *(Heading 5)*

We give special thanks to TA's and supervisors Maryam and Morteza for spending their time to help students to complete the code by answering questions.