CENG 3430 PROJECT FINAL REPORT
FLAPPY BIRD

CENG 3430 Rapid Prototyping of Digital Systems

09/05/2021

—

Name: Steve Dexter B. Tamang

SID: 1155124868

1. Introduction

1.1 Project Overview

Flappy bird was a successful mobile game launched in 2013 and generated 50 million downloads by early January 2014 (Dogtiev, 2020). It is a 2D game in which the player controls the bird, which constantly moves to the right. The player should navigate the bird through the pipes. The pipes are of random heights but with equally sized gaps. The bird will ascend only if the player touches the screen. Otherwise, it will fall to the ground. Each time the bird passes through the pipe, one point will be rewarded to the player. If the bird hits a pipe or falls to the ground, the game ends.

In this project, I will build a VHDL version of this game using ZedBoard, PmodSSD, and a VGA display, and the knowledge learned from this course.

Moreover, I would like to game to another level by adding background colors and cloud images, similar to the original game. I believe that adding those things shall make the game look more pleasing to the eye.

2. Game User Interface

2.1 Gameplay

Below is the prototype design of the game user interface; it consists of a bird, some clouds, pipes, and the ground.

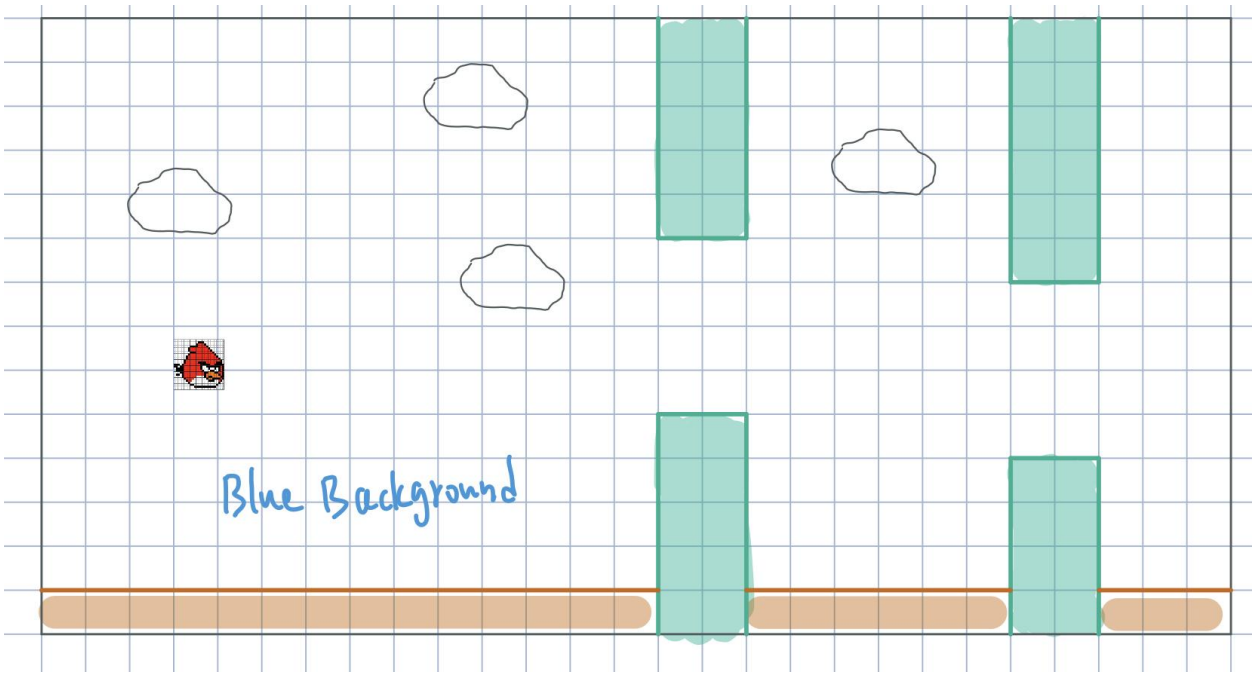


Figure 2.1.1 Prototype for the game UI, drawn on an iPad

2.2 Drawing the Bird

The bird is one of the most essential features of the game. To make the game more interesting and fun, I have decided to use a character in the Angry Birds Franchise, called “Red.” The bird is represented using a 23 x 25 matrix for gameplay. Moreover, I wrote a Python Program to scale the 23 x 25 matrix to a 143 x 149 matrix for displaying the bird on the splash screen.

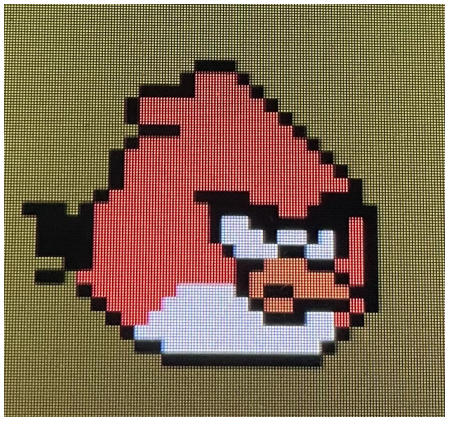


Figure 2.2.1: 143 x 149 bird image

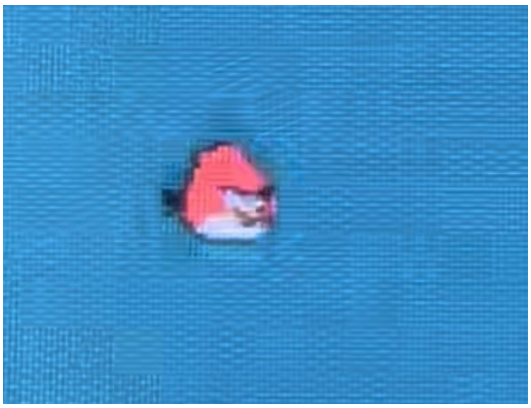


Figure 2.2.2: 23 x 25 bird image for gameplay

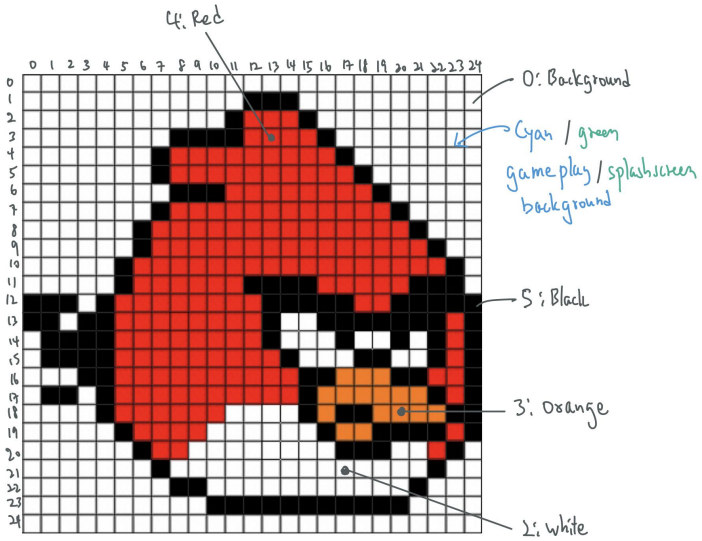


Figure 2.2.3 Prototype of the bird

2.3 Drawing the Pipes

The pipes are also one of the essential components of the game. The idea is to create rectangular-shaped pipes with gaps of random sizes. The pipes are dyed in a light green color, similar to the original game.

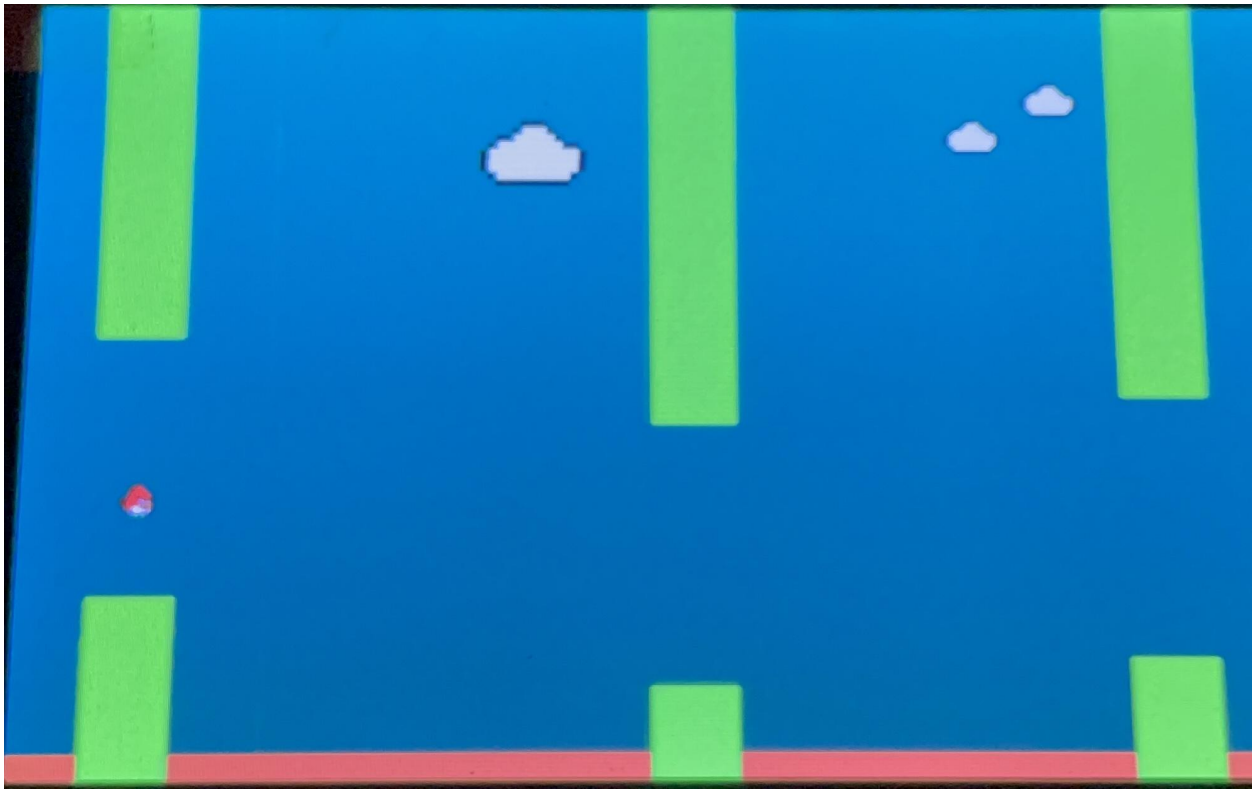


Figure 2.3: The randomly sized pipes

2.4 Drawing the Letters

In the splash screen (initial state) and the game over screen, there are words displayed on the LCD display. The purpose of this is to let the user know how to navigate around the game. This can enhance user-friendliness.

2.4.1 Splash Screen

The Splash Screen consists of the title of the game, instructions on how to start playing the game, together with an image of a bird and some clouds to make the splash screen look nice. The text “FLAPPY BIRD” is represented as a 59 x 593 matrix consisting of values 1 and 0, where the letters are drawn, and 0 are the empty spaces. Similarly, the text “PLAY: BTNR” is represented as a 29 x 269 matrix consisting of values 1 and 0.

Figure 2.4.1: Splash Screen informing the user to press BTNR to play the game

2.4.2 Game Over Screen

The Game Over Screen (end screen) shows the word “Game Over” and an instruction “Play Again: BTND,” informing the player to press BTND to try playing the game again. The text “GAME OVER” is represented as a 39 x 323 matrix that consists of values 1 and 0, where 1 is the letters drawn and 0 is the empty spaces. Similarly, the text “PLAY AGAIN: BTND” is represented as a 19 x 287 matrix consisting of values 1 and 0.

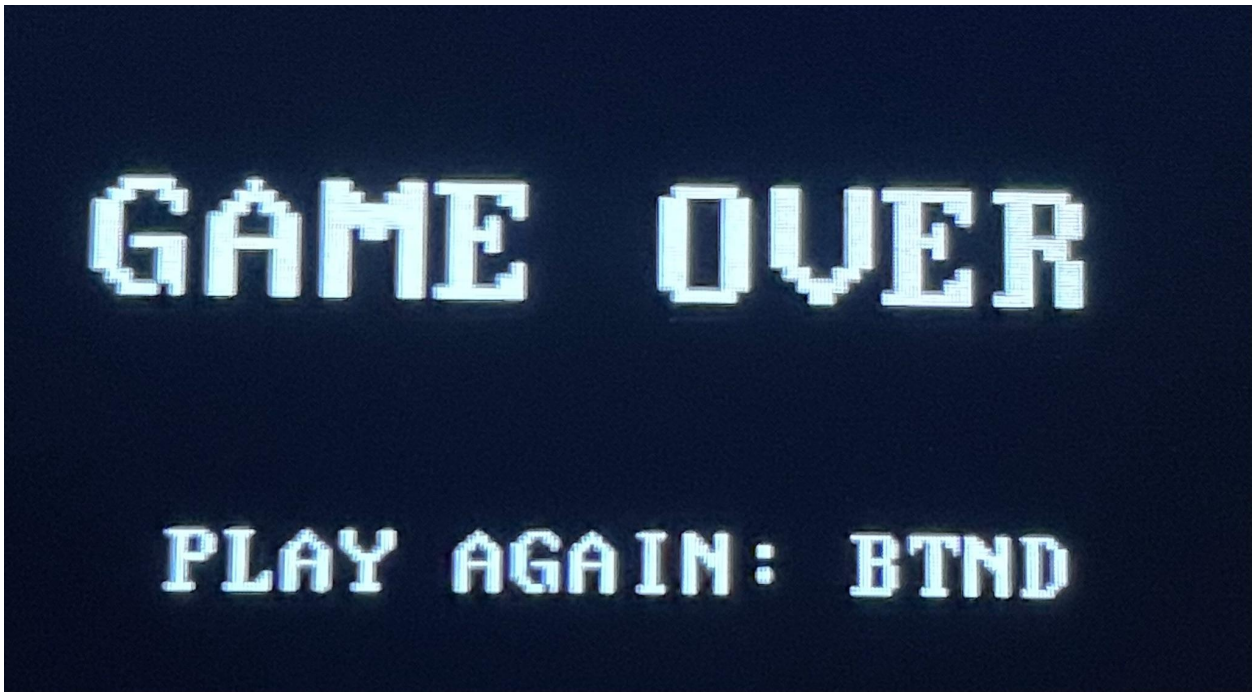


Figure 2.4.2 Game Over Screen informing the player to press BTND to play again

2.5 Drawing the Clouds

The clouds make this game look more like the original flappy bird. This game has clouds of three different sizes, small, medium, and large. The small size is represented as a 26 x 38 matrix, the medium size is represented as a 35 x 51 matrix, and the large size is represented as a 53 x 77 matrix. All the clouds are initially drawn as a 9 x 13 matrix (see Figure 2.5.3) and then scaled to make the three different sizes of clouds.

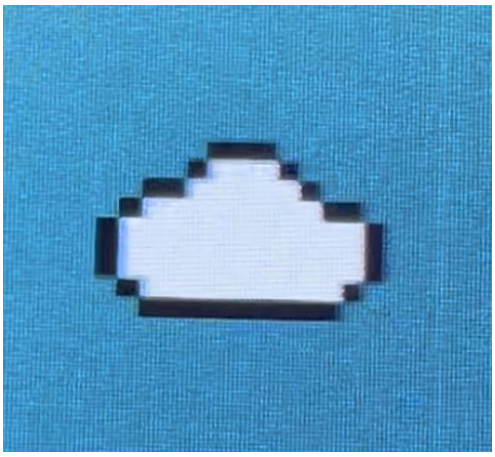


Figure 2.5.1: The Large-sized cloud

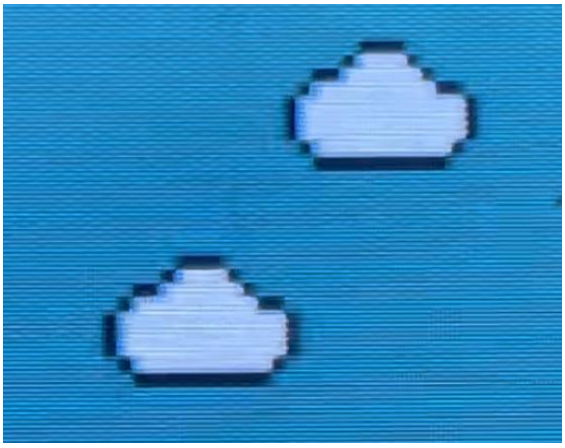


Figure 2.5.2: The 2 medium-sized clouds

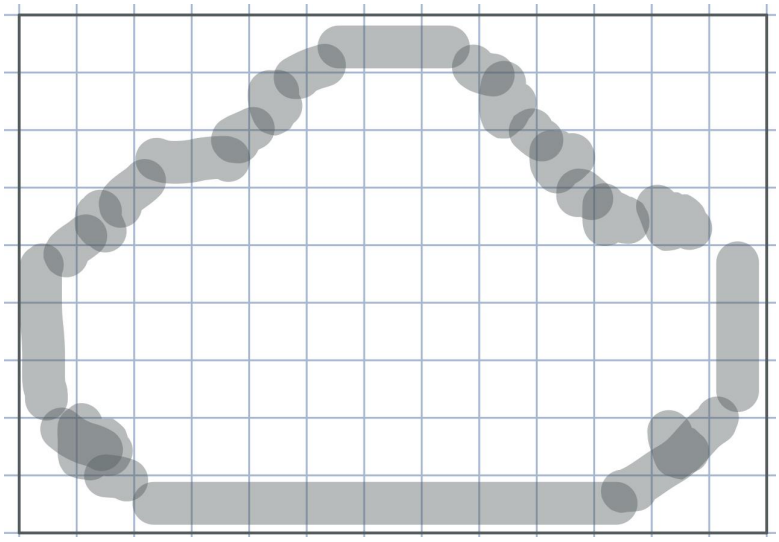
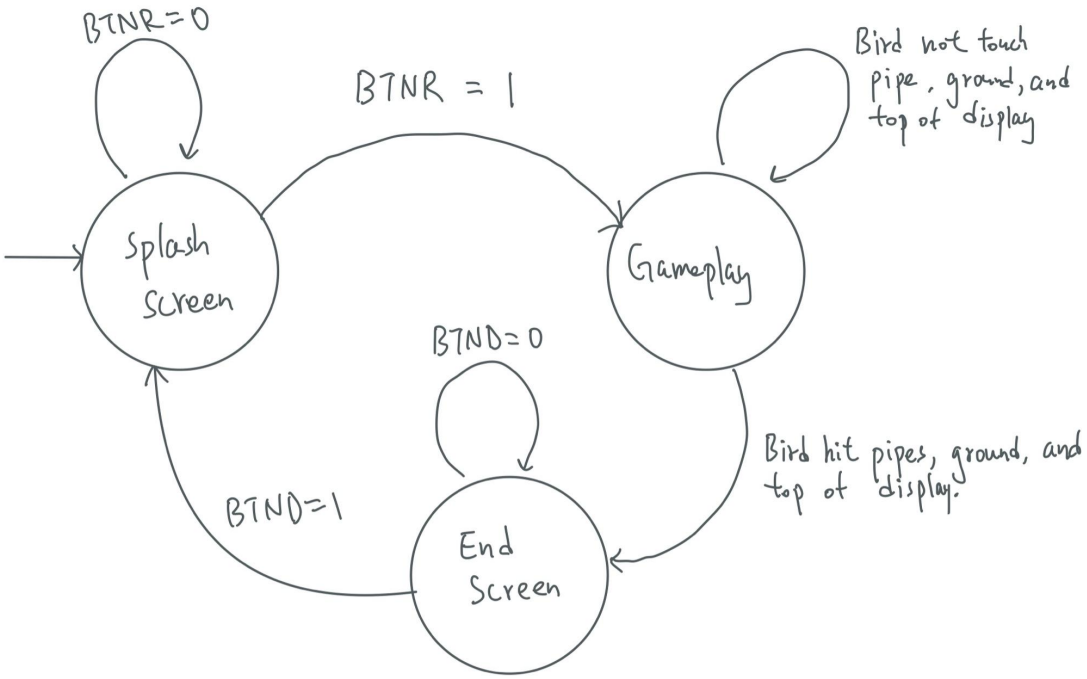


Figure 2.5.3: The 9 x 13 cloud matrix prototype

3. System Design

3.1 Overall Architecture

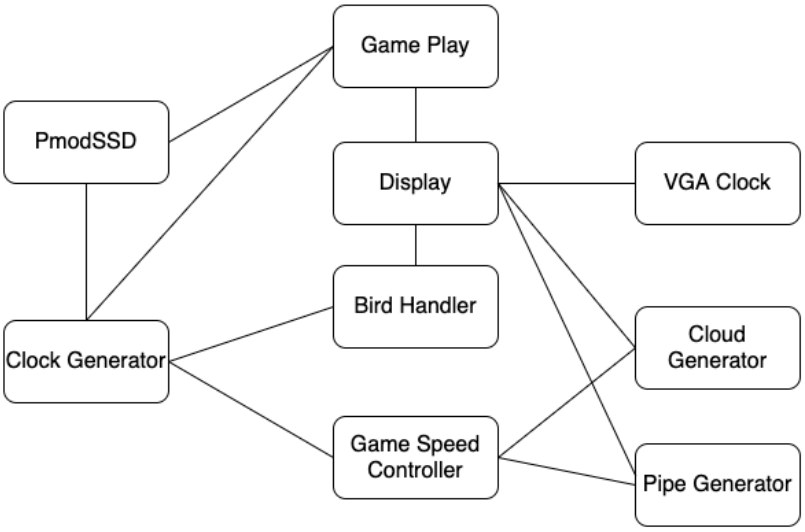
The system has three main states, namely, Splash screen, Gameplay, and End screen. Below is a simplified Finite State Machine of the Game.



- Splash Screen: Initial State, it shows the name of the game, “Flappy Bird,” instruction, “Play: BTNR,” some clouds, and the bird for decoration.
- Gameplay: In this state, the user is playing the game. The clouds of different sizes and pipes move left at a certain speed, depending on the score. If the score is quite high, the pipes and clouds move faster. Also, the bird will move up when the user presses the BTNC button and down when nothing is pressed.
- End Screen: When the bird hits the pipe, ground, and the top of the display, the game is over; the end screen will be shown together with the text “Game Over” and the instruction of “Play Again: BTND.”

3.2 Module Description

This project is built using VHDL; thus, all the modules are executed concurrently.



In this project, some modules use others modules to function. The functions and interactions between modules are described below.

- Clock Generator: It generates four different clocks with different frequencies, including a 100Hz, 150Hz, 200Hz, 250Hz clock. These clocks are used by other modules such as Game Speed Controller and Bird Handler
- .Game Speed Generator: It controls the speed of the game. It is used for generating pipes and clouds. Depending on the current score during the gameplay, the Game Speed Controller will select a faster clock so that the game becomes more difficult.
- Display: This module controls the output of the LCD Display, depending on the position of the pipes, bird, clouds, text. Moreover, the Display uses a VGA Clock Module to function.
- Game Play: This module allows the transition from one state to another depending on the user inputs and game outcome (win/lose).

- Bird Handler: This module controls the position of the bird.
- Cloud Handler: This module controls the position of the clouds.
- Pipe Generator: This module controls the position of the pipe, generates new pipes, and determines if the bird passes through the pipes.
- PmodSSD: This module controls the output of the PmodSSD; the user's score determines the output value.

3.3 Input and Output

This game uses three buttons, BTNC, BTNR, and BTND as inputs, Pmod SSD, and LCD display as outputs.

3.3.1 Buttons and Control

- BTNC: When this button is pressed, the bird will fly upward
- BTND: When this button is pressed, the game state will transition from end_screen to start_screen
- BTNR: When this button is pressed, the game will begin

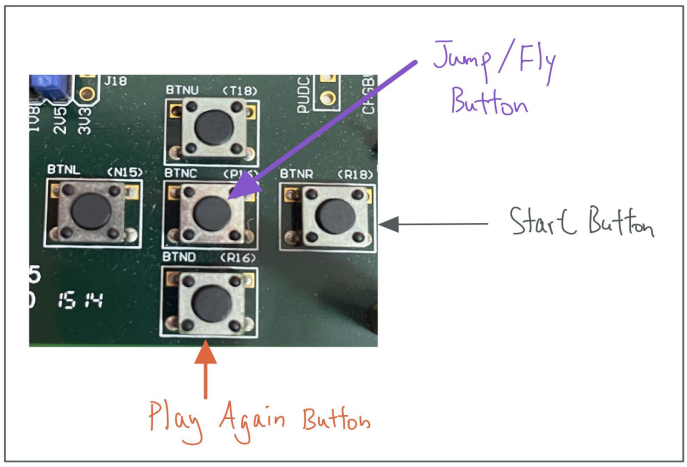


Figure 3.3.1: Button descriptions

3.3.2 LCD Display

The elements that will be displayed on the LCD display are the pipes, clouds, birds, and some texts or words.

3.3.2.1 The Bird Image

A 2D array is used to represent the bird's image. The values in the array are the colors of the bird. The bird has five different color values 0, 2, 3, 4, and 5, representing the colors cyan, white, orange, red, and black.

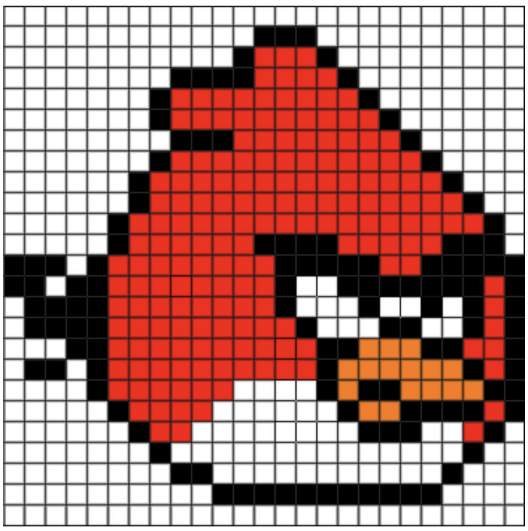


Figure 3.3.2.1: An angry bird Image found on the internet (see reference)

```
type angry2D is array(0 to 23, 0 to 25) of integer range 0 to 9;
signal bird: angry2D;
```

Figure 3.3.2.2.2: The 2D array for the bird (gameplay)

```
type largeAB is array(0 to 143, 0 to 149) of integer;
signal angryBirdLarge: largeAB;
```

Figure 3.3.2.1.3: The 2D array for the bird (splash screen)

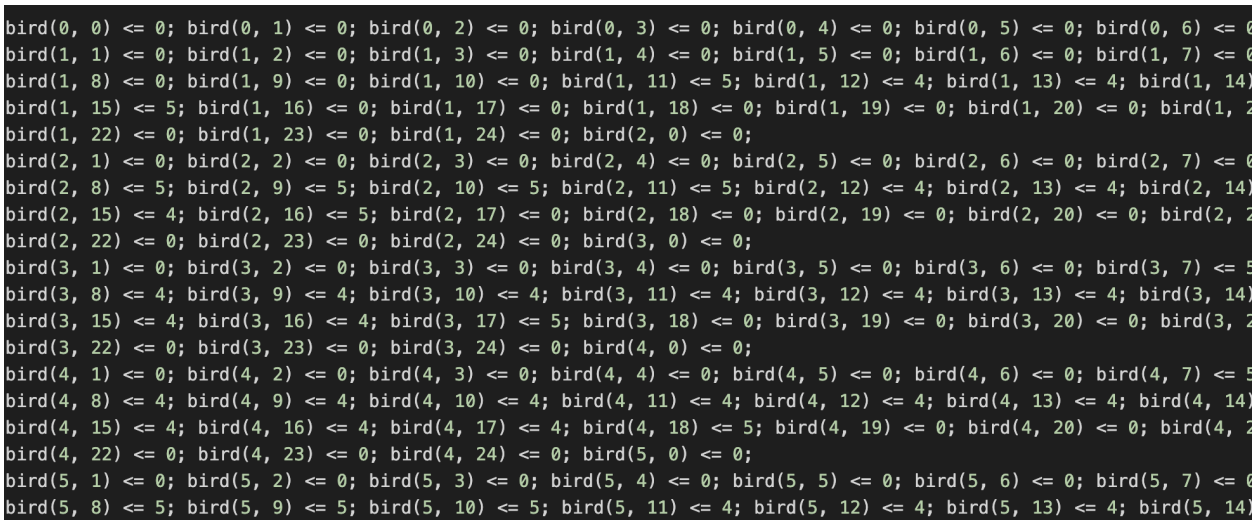


Figure 3.3.2.1.4: The matrix of the bird's image generated by Python Code

The initial position of the bird is defined below:

```
-- Bird area box
signal b_start_x: integer := H_START + 120;
signal b_end_x: integer := H_START + 120 + bird_width;
signal b_start_y: integer := V_START + 285;
signal b_end_y: integer := V_START + 285 + bird_height;
```

Figure 3.3.2.1.5: The initial position of the bird

Next, the implementation of the control of the bird. When the game begins, the bird automatically falls to the ground if the player did not press the BTNC button. If the player presses the BTNC button, the bird will translate in the positive y-direction (fly). Below is the relevant code.

```
--handle the movement of bird
bird_handle: process(clk100Hz) begin
    if(falling_edge(clk100Hz)) then
        if (game_state = game_play) then
            if(BTNC = '1') then
                b_start_y <= b_start_y - 6;
                b_end_y <= b_start_y + bird_height;
                if(b_start_y < V_START) then
                    b_start_y <= V_START + 1;
                    b_end_y <= b_start_y + bird_height;
                end if;
            else
                b_start_y <= b_start_y + 3;
                b_end_y <= b_start_y + bird_height;
                if(b_end_y > V_END) then
                    b_end_y <= V_END;
                    b_start_y <= b_end_y - bird_height;
                end if;
            end if;
        elsif (game_state = end_screen or game_state = splash_screen) then
            b_start_y <= V_START + 285;
            b_end_y <= V_START + 285 + bird_height;
        end if;
    end if;
end process bird_handle;
```

Figure 3.3.2.6: The code for controlling the bird

3.3.2.2 The Cloud Image

A 2D array is used to represent the clouds image. The values in the array are the colors of the cloud. The bird has three different color values, 0, 1, 2, representing the colors white, black, and cyan. In this game, there are three different cloud sizes.

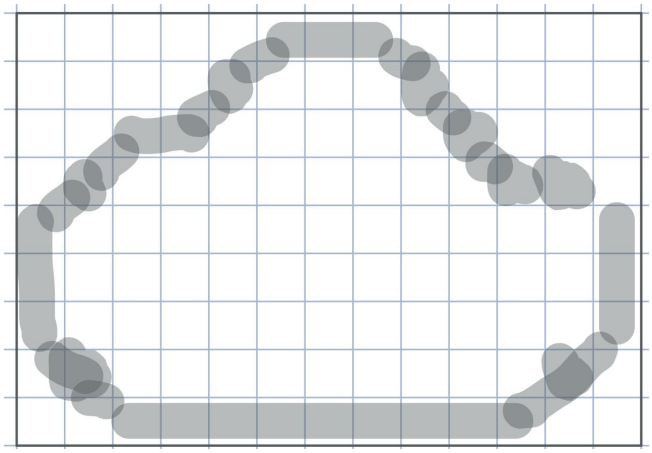


Figure 3.3.2.2.1: prototype design

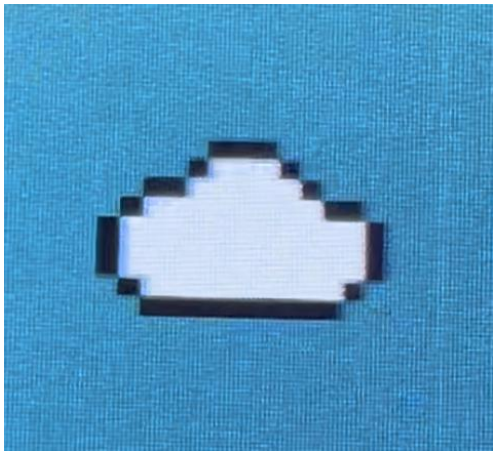


Figure 3.3.2.2.1: cloud on display

```
type cloud_small2D is array(0 to 26, 0 to 38) of integer;
signal cloud_small: cloud_small2D;

type cloud_large2D is array(0 to 35, 0 to 51) of integer;
signal cloud_large: cloud_large2D;

type cloud_xl2D is array(0 to 53, 0 to 77) of integer;
signal cloud_xl: cloud_xl2D;
```

Figure 3.3.2.2.3: The 2D array for the three different cloud sizes

```
cloud_large(0, 0) <= 2; cloud_large(0, 1) <= 2; cloud_large(0, 2) <= 2; cloud_large(0, 3) <= 2;
cloud_large(1, 0) <= 2; cloud_large(1, 1) <= 2; cloud_large(1, 2) <= 2; cloud_large(1, 3) <= 2;
cloud_large(2, 0) <= 2; cloud_large(2, 1) <= 2; cloud_large(2, 2) <= 2; cloud_large(2, 3) <= 2;
cloud_large(3, 0) <= 2; cloud_large(3, 1) <= 2; cloud_large(3, 2) <= 2; cloud_large(3, 3) <= 2;
cloud_large(4, 0) <= 2; cloud_large(4, 1) <= 2; cloud_large(4, 2) <= 2; cloud_large(4, 3) <= 2;
cloud_large(5, 0) <= 2; cloud_large(5, 1) <= 2; cloud_large(5, 2) <= 2; cloud_large(5, 3) <= 2;
cloud_large(6, 0) <= 2; cloud_large(6, 1) <= 2; cloud_large(6, 2) <= 2; cloud_large(6, 3) <= 2;
cloud_large(7, 0) <= 2; cloud_large(7, 1) <= 2; cloud_large(7, 2) <= 2; cloud_large(7, 3) <= 2;
cloud_large(8, 0) <= 2; cloud_large(8, 1) <= 2; cloud_large(8, 2) <= 2; cloud_large(8, 3) <= 2;
cloud_large(9, 0) <= 2; cloud_large(9, 1) <= 2; cloud_large(9, 2) <= 2; cloud_large(9, 3) <= 2;
cloud_large(10, 0) <= 2; cloud_large(10, 1) <= 2; cloud_large(10, 2) <= 2; cloud_large(10, 3) <
```

Figure 3.3.2.2.4: The matrix of the large cloud image generated by Python Code

The initial position of the clouds are defined below:

```
signal cs_start_x: integer := H_START + 100;
signal cs_end_x: integer := H_START + 100 +  cloud_s_width;
signal cs_start_y: integer := V_START + 50;
signal cs_end_y: integer := V_START + 50 + cloud_s_height;

signal cl_start_x: integer := H_START + 400;
signal cl_end_x: integer := H_START + 400 +  cloud_l_width;
signal cl_start_y: integer := V_START + 20;
signal cl_end_y: integer := V_START + 20 + cloud_l_height;

signal cxl_start_x: integer := H_START + 800;
signal cxl_end_x: integer := H_START + 800 +  cloud_xl_width;
signal cxl_start_y: integer := V_START + 80;
signal cxl_end_y: integer := V_START + 80 + cloud_xl_height;
```

Figure 3.3.2.2.4: The initial position of the clouds

Next, the implementation of the movement of the clouds. When the game begins, the clouds will automatically shift left. The shifting speed is determined by the GameSpeedController (discussed in the next section, section 4.5) Below is the relevant code for controlling the clouds.

```
cloud_generator: process(clock_signal) begin
    if(rising_edge(clock_signal)) then
        if(game_state = game_play) then
            -- small cloud
            if(cs_end_x > H_START) then
                cs_start_x <= cs_start_x - 1;
                cs_end_x <= cs_end_x - 1;
            else
                cs_start_x <= H_END + 80;
                cs_end_x <= H_END + 80 + cloud_s_width;
            end if;
            if(cs1_end_x > H_START) then
                cs1_start_x <= cs1_start_x - 1;
                cs1_end_x <= cs1_end_x - 1;
            else
                cs1_start_x <= H_END + 80;
                cs1_end_x <= H_END + 80 + cloud_s_width;
            end if;
```

Figure 3.3.2.2.5: movement of the small cloud (similar implementation for other cloud sizes)

3.3.2.3 The Flappy Bird Word

A 2D array is used to represent the letters in the word “FLAPPY BIRD.” The values in the array are the colors of the word. It has two different color values 0, 1, which represent the colors green and white. The green color is for matching the background color of the Splash Screen. The initialization of the word is defined below.

```
type flappyWord is array (0 to 59, 0 to 593) of integer;
signal flappy_W: flappyWord;
```

Figure 3.3.2.3.1: 2D array for the word

```
flappy_W(0, 0) <= 1; flappy_W(0, 1) <= 1; flappy_W(0, 2) <= 1; flappy_W(0, 3) <= 1; flappy_W(0, 4) <= 1;
6) <= 1; flappy_W(0, 7) <= 1; flappy_W(0, 8) <= 1; flappy_W(0, 9) <= 1; flappy_W(0, 10) <= 1; flappy_W(0
flappy_W(0, 13) <= 1; flappy_W(0, 14) <= 1; flappy_W(0, 15) <= 1; flappy_W(0, 16) <= 1; flappy_W(0, 17)
(0, 19) <= 1; flappy_W(0, 20) <= 1; flappy_W(0, 21) <= 1; flappy_W(0, 22) <= 1; flappy_W(0, 23) <= 1; fl
<= 1; flappy_W(0, 26) <= 1; flappy_W(0, 27) <= 1; flappy_W(0, 28) <= 1; flappy_W(0, 29) <= 1; flappy_W(0
flappy_W(0, 32) <= 1; flappy_W(0, 33) <= 1; flappy_W(0, 34) <= 1; flappy_W(0, 35) <= 1; flappy_W(0, 36)
(0, 38) <= 1; flappy_W(0, 39) <= 1; flappy_W(0, 40) <= 1; flappy_W(0, 41) <= 1; flappy_W(0, 42) <= 0; fl
<= 0; flappy_W(0, 45) <= 0; flappy_W(0, 46) <= 0; flappy_W(0, 47) <= 0; flappy_W(0, 48) <= 0; flappy_W(0
flappy_W(0, 51) <= 0; flappy_W(0, 52) <= 0; flappy_W(0, 53) <= 0; flappy_W(0, 54) <= 1; flappy_W(0, 55)
(0, 57) <= 1; flappy_W(0, 58) <= 1; flappy_W(0, 59) <= 1; flappy_W(0, 60) <= 1; flappy_W(0, 61) <= 1; fl
<= 1; flappy_W(0, 64) <= 1; flappy_W(0, 65) <= 1; flappy_W(0, 66) <= 1; flappy_W(0, 67) <= 1; flappy_W(0
flappy_W(0, 70) <= 1; flappy_W(0, 71) <= 1; flappy_W(0, 72) <= 1; flappy_W(0, 73) <= 1; flappy_W(0, 74)
(0, 76) <= 1; flappy_W(0, 77) <= 1; flappy_W(0, 78) <= 0; flappy_W(0, 79) <= 0; flappy_W(0, 80) <= 0; fl
```

Figure 3.3.2.3.2: The matrix of the word generated by Python Code

The position of the word is set to the center of the screen. As defined below.

```
-- Text Area
signal t_start_x: integer := H_START + 236;
signal t_end_x: integer := H_START + 236 + text_width;
signal t_start_y: integer := V_START + 180;
signal t_end_y: integer := V_START + 180 + text_height;
```

Figure 3.3.2.3.3: The position of the word

The above implementation applies to all words displayed on the LCD display.

3.3.2.4 The Pipes

The pipes are the obstacles for the bird. If the bird hits the pipe, the game is over. Below is the code for initializing the position of the pipes.

```
signal p1xs: integer := H_END - 60 - 341;
signal p1xe: integer := H_END - 341;

signal p2xs: integer := H_END;
signal p2xe: integer := H_END + 60;

signal p3xs: integer := H_END + 341;
signal p3xe: integer := H_END + 60 + 341;

signal p1ys: integer := V_START + 150;
signal p2ys: integer := V_START + 100;
signal p3ys: integer := V_START + 200;
```

Figure 3.3.2.4.1. The initial position of the pipes

Next, the implementation of the movement of the pipes. When the game begins, the pipes will automatically shift left. The speed of the shifting is determined by the GameSpeedController (discussed in the next section, section 4.5) Below is the relevant code for controlling the pipes.

```
pipe_genetator: process(clock_signal) begin
    if(rising_edge(clock_signal)) then
        if (game_state = game_play) then
            -- pipe 1
            if (p1xe > H_START) then
                p1xs <= p1xs - 1;
                p1xe <= p1xe - 1;
```

Figure 3.3.2.4.2: Handling the shifting/movement of pipe 1

3.3.2.5 The Pmod SSD

This component is to display the score to the player. The score will be incremented once the bird passes through a pipe. Since the Pmod SSD can only display two digits, the maximum score is 99. Once the score reaches 99, the game is over, and the score is reset to 0. Below are the relevant codes.

```
if(rising_edge(clk_100Hz)) then
    sel <= not sel;
    if(sel = '0') then
        digit <= value / 10;
    else
        digit <= value mod 10;
```

Figure 3.3.2.5.1: Alternating between the digits

```
-- display the digits in set mode
process(digit) begin
    case digit is
        when 0 => ssd <= "1111110";
        when 1 => ssd <= "0110000";
        when 2 => ssd <= "1101101";
        when 3 => ssd <= "1111001";
        when 4 => ssd <= "0110011";
        when 5 => ssd <= "1011011";
        when 6 => ssd <= "1011111";
        when 7 => ssd <= "1110000";
        when 8 => ssd <= "1111111";
        when 9 => ssd <= "1111011";
        when others => ssd <= "0000000";
    end case;
end process;
```

Figure 3.3.2.5.2: Displaying a digit

4. Implementation

This section will elaborate more on the previous sections and explain how this game is implemented.

4.1 Handling the Pipes

First, I declare the signals to initialize the positions and size of the three pipes.

```
signal sky_x: integer := H_START;
signal sky_y: integer := V_END - 25;

signal p1xs: integer := H_END - 60 - 341;
signal p1xe: integer := H_END - 341;

signal p2xs: integer := H_END;
signal p2xe: integer := H_END + 60;

signal p3xs: integer := H_END + 341;
signal p3xe: integer := H_END + 60 + 341;

signal p1ys: integer := V_START + 150;
signal p2ys: integer := V_START + 100;
signal p3ys: integer := V_START + 200;
signal gap, gap1, gap2, gap3: integer := 200;
```

Figure 4.1.1

The integer signals p1xs, p2xs, p3xs refer to the horizontal starting position of pipe1, pipe2, and pipe3 respectively.

The integer signals p1xe, p2xe, p3xe refer to the horizontal ending position of pipe1, pipe2, and pipe3. The integer signals p1ys, p2ys, p3ys are the starting position of the gaps in between the pipes. Also, the integer signals gap, gap1, gap2, gap3 defines the size of the gap between the pipes.

Moreover, I have defined an integer array of 100 “random” values of gap positions. The purpose of this is to make the pipes generated in the game appear to be different or random.

```
type INT_ARRAY is array (integer range <>) of integer;
signal pipe_ran: INT_ARRAY(0 to 99);
signal pipe_count: integer := 0;
```

Figure 4.1.2

To generate the 100 “random” values of the gap position, I wrote a Python Program to generate the code below (Figure 4.1.3); the values assigned are the randomly generated values by the Python Program.

```
pipe_ran(0) <= 136; pipe_ran(1) <= 291; pipe_ran(2) <= 281; pipe_ran(3) <= 342; pipe_ran(4) <= 100; pipe_ran(5) <= 269; pipe_r
pipe_ran(8) <= 127; pipe_ran(9) <= 330; pipe_ran(10) <= 111; pipe_ran(11) <= 385; pipe_ran(12) <= 175; pipe_ran(13) <= 341; pi
pipe_ran(15) <= 309; pipe_ran(16) <= 198; pipe_ran(17) <= 269; pipe_ran(18) <= 236; pipe_ran(19) <= 168; pipe_ran(20) <= 292;
pipe_ran(22) <= 110; pipe_ran(23) <= 218; pipe_ran(24) <= 264; pipe_ran(25) <= 175; pipe_ran(26) <= 230; pipe_ran(27) <= 316;
pipe_ran(29) <= 176; pipe_ran(30) <= 165; pipe_ran(31) <= 122; pipe_ran(32) <= 312; pipe_ran(33) <= 107; pipe_ran(34) <= 147;
pipe_ran(36) <= 114; pipe_ran(37) <= 229; pipe_ran(38) <= 132; pipe_ran(39) <= 386; pipe_ran(40) <= 277; pipe_ran(41) <= 328;
pipe_ran(43) <= 297; pipe_ran(44) <= 289; pipe_ran(45) <= 123; pipe_ran(46) <= 127; pipe_ran(47) <= 177; pipe_ran(48) <= 252;
pipe_ran(50) <= 317; pipe_ran(51) <= 205; pipe_ran(52) <= 161; pipe_ran(53) <= 389; pipe_ran(54) <= 136; pipe_ran(55) <= 122;
pipe_ran(57) <= 105; pipe_ran(58) <= 378; pipe_ran(59) <= 203; pipe_ran(60) <= 237; pipe_ran(61) <= 312; pipe_ran(62) <= 323;
pipe_ran(64) <= 236; pipe_ran(65) <= 219; pipe_ran(66) <= 243; pipe_ran(67) <= 220; pipe_ran(68) <= 171; pipe_ran(69) <= 384;
pipe_ran(71) <= 389; pipe_ran(72) <= 359; pipe_ran(73) <= 244; pipe_ran(74) <= 145; pipe_ran(75) <= 191; pipe_ran(76) <= 114;
pipe_ran(78) <= 315; pipe_ran(79) <= 290; pipe_ran(80) <= 112; pipe_ran(81) <= 162; pipe_ran(82) <= 203; pipe_ran(83) <= 149;
pipe_ran(85) <= 111; pipe_ran(86) <= 277; pipe_ran(87) <= 388; pipe_ran(88) <= 214; pipe_ran(89) <= 264; pipe_ran(90) <= 234;
pipe_ran(92) <= 192; pipe_ran(93) <= 194; pipe_ran(94) <= 217; pipe_ran(95) <= 137; pipe_ran(96) <= 114; pipe_ran(97) <= 391;
pipe_ran(99) <= 320;
```

Figure 4.1.3

Here is the Python code for generating the above VHDL code:

```
import random
f = open("pipe.txt", "a")

write = ""

for i in range(100):
    write += ("pipe_ran(" + str(i) + ") <= " + str(random.randint(100, 400)) + "; ")
    if(i % 7 == 0 and i > 6):
        write += "\n"

f.write(write)
f.close()
```

Figure 4.1.4

Next, the implementation of the pipe movement.

The pipe will be displayed and moved only in the game_play state. The pipes will shift left if the pipe's position is greater than H_START. Once the pipe's position is less than or equal to the H_START, the position of the pipes will be placed to the right side of the display, and the gap position of the pipes will also be updated by accessing the value stored in the array of pipe_ran (see Figure 4.1.2 & 4.1.3). Moreover, when the bird successfully passes through the pipe, the score will be incremented by one. Below is the relevant code for pipe 1.

```
pipe_genetator: process(clock_signal) begin
    if(rising_edge(clock_signal)) then
        if (game_state = game_play) then
            -- pipe 1
            if (p1xe > H_START) then
                p1xs <= p1xs - 1;
                p1xe <= p1xe - 1;
            else
                p1xs <= H_END;
                p1xe <= H_END + 60;
                p1ys <= pipe_ran(pipe_count);
                value <= value + 1;
                gap1 <= gap;
                pipe_count <= pipe_count + 1;
            end if;
```

Figure 4.1.5

The code for pipe 2 and 3 are similar to the code in Figure 4.1.5; the only differences are in the signal names.

4.2 Handling the Bird

The process named bird_handler is used to control the movement of the bird. The bird will be displayed and moved only in the game_play state. The bird moves up when the user presses the BTNC button and moves down towards the ground when the user does not press the BTNC button.

When the state is not at game_play state, the bird's position will have resorted to its original position. Below is the relevant code for handling the control of the bird

```
--handle the movement of bird
bird_handle: process(clk100Hz) begin
    if(falling_edge(clk100Hz)) then
        if (game_state = game_play) then
            if(BTNC = '1') then
                b_start_y <= b_start_y - 6;
                b_end_y <= b_start_y + bird_height;
                if(b_start_y < V_START) then
                    b_start_y <= V_START + 1;
                    b_end_y <= b_start_y + bird_height;
                end if;
            else
                b_start_y <= b_start_y + 3;
                b_end_y <= b_start_y + bird_height;
                if(b_end_y > V_END) then
                    b_end_y <= V_END;
                    b_start_y <= b_end_y - bird_height;
                end if;
            end if;
        elsif (game_state = end_screen or game_state = splash_screen) then
            b_start_y <= V_START + 285;
            b_end_y <= V_START + 285 + bird_height;
        end if;
    end if;
end process bird_handle;
```

Figure 4.2.1

4.3 Creating Characters and Drawings

In this project, characters were displayed on the LCD screen. I use Python to help ease the process of drawing the characters and strings to the LCD display.

First, I draw the characters using a 10 x 9 matrix. Although it is small, the goal is to draw a simple character first and then scale it up. For example, a letter 'F ':

```
f1 = [[1],[1],[1],[1],[1],[1],[1],[0],[0]]     ## 2 *******
f2 = [[0],[1],[1],[0],[0],[1],[1],[0],[0]]     ## 3  **  **
f3 = [[0],[1],[1],[0],[0],[0],[1],[0],[0]]     ## 4  **   *
f4 = [[0],[1],[1],[0],[1],[0],[0],[0],[0]]     ## 5  ** *
f5 = [[0],[1],[1],[1],[1],[0],[0],[0],[0]]     ## 6  ****
f6 = [[0],[1],[1],[0],[1],[0],[0],[0],[0]]     ## 7  ** *
f7 = [[0],[1],[1],[0],[0],[0],[0],[0],[0]]     ## 8  **
f8 = [[0],[1],[1],[0],[0],[0],[0],[0],[0]]     ## 9  **
f9 = [[0],[1],[1],[0],[0],[0],[0],[0],[0]]     ## a  **
f10 = [[1],[1],[1],[1],[0],[0],[0],[0],[0]]    ## b ****
```

Figure 4.3.1

In figure 4.3.1, I use 1 to represent the character and 0 for the spaces around the character. Then I use the python code below (Figure 4.3.2) that I wrote to scale up the characters or strings.

```python
def CharScale(array, height, width, scale):
    write = ""
    temp_w = ""
    arr = []
    for i in range(height):
        write += '\n'
        a = []
        for j in range(width):
            temp = array[i][j] * scale

            for k in temp:
                write += str(k) + ' '
                temp_w += str(k) + ' '
                a.append(k)

        for s in range(scale):
            write += '\n'
            write += temp_w
            arr.append(a)
        temp_w = ""
    return arr
```

Figure 4.3.2

After scaling up the character 'F,' it is then converted into VHDL code like so:

```
F(0, 0) <= 1; F(0, 1) <= 1; F(0, 2) <= 1; F(0, 3) <= 1; F(0, 4) <= 1; F(0, 5
F(1, 0) <= 1; F(1, 1) <= 1; F(1, 2) <= 1; F(1, 3) <= 1; F(1, 4) <= 1; F(1, 5
F(2, 0) <= 0; F(2, 1) <= 0; F(2, 2) <= 1; F(2, 3) <= 1; F(2, 4) <= 1; F(2, 5
F(3, 0) <= 0; F(3, 1) <= 0; F(3, 2) <= 1; F(3, 3) <= 1; F(3, 4) <= 1; F(3, 5
F(4, 0) <= 0; F(4, 1) <= 0; F(4, 2) <= 1; F(4, 3) <= 1; F(4, 4) <= 1; F(4, 5
F(5, 0) <= 0; F(5, 1) <= 0; F(5, 2) <= 1; F(5, 3) <= 1; F(5, 4) <= 1; F(5, 5
F(6, 0) <= 0; F(6, 1) <= 0; F(6, 2) <= 1; F(6, 3) <= 1; F(6, 4) <= 1; F(6, 5
F(7, 0) <= 0; F(7, 1) <= 0; F(7, 2) <= 1; F(7, 3) <= 1; F(7, 4) <= 1; F(7, 5
F(8, 0) <= 0; F(8, 1) <= 0; F(8, 2) <= 1; F(8, 3) <= 1; F(8, 4) <= 1; F(8, 5
F(9, 0) <= 0; F(9, 1) <= 0; F(9, 2) <= 1; F(9, 3) <= 1; F(9, 4) <= 1; F(9, 5
F(10, 0) <= 0; F(10, 1) <= 0; F(10, 2) <= 1; F(10, 3) <= 1; F(10, 4) <= 1; F
F(11, 0) <= 0; F(11, 1) <= 0; F(11, 2) <= 1; F(11, 3) <= 1; F(11, 4) <= 1; F
```

Figure 4.3.3

The exact process goes for drawing all kinds of letters, birds, and clouds into the LCD display.

4.4 Displaying Score

To display the score on the Pmod SSD, the sel pin will be used for selecting the right of the left seven-segment display. If the sel pin is low, it means the left seven-segment display is selected. Otherwise, the right one is selected. The sel pin alternates between high and low using a 100Hz clock, making the alternation indistinguishable to the naked eye. Below is the relevant code (Figure 4.4.1).

```
Display_Digits: process(clk_100Hz) begin
    if(rising_edge(clk_100Hz)) then
       sel <= not sel;
       if(sel = '0') then
           digit <= value / 10;
        else
           digit <= value mod 10;
        end if;
    end if;
end process Display_Digits;
```

Figure 4.4.1

In figure 4.4.1, with sel is low (‘0’), the left digit is displayed, and when sel is high(“1”), then the right digit is displayed.

```
process(digit) begin
    case digit is
        when 0 => ssd <= "1111110";
        when 1 => ssd <= "0110000";
        when 2 => ssd <= "1101101";
        when 3 => ssd <= "1111001";
        when 4 => ssd <= "0110011";
        when 5 => ssd <= "1011011";
        when 6 => ssd <= "1011111";
        when 7 => ssd <= "1110000";
        when 8 => ssd <= "1111111";
        when 9 => ssd <= "1111011";
        when others => ssd <= "0000000";
    end case;
end process;
```

Figure 4.4.2

Also, the player's score should be added when the bird passes through the pipe. Figure 4.4.3 slow shows the relevant code.

```
if (game_state = game_play) then
    -- pipe 1
    if (p1xe > H_START) then
        p1xs <= p1xs - 1;
        p1xe <= p1xe - 1;
    else
        p1xs <= H_END;
        p1xe <= H_END + 60;
        p1ys <= pipe_ran(pipe_count);
        value <= value + 1;
        gap1 <= gap;
        pipe_count <= pipe_count + 1;
    end if;
```

Figure 4.4.3

Moreover, the code in Figure 4.4.1 and 4.4.2 is designed so that when the signal value is 10, it will show ten and not the hexadecimal of 10 (which is A).

4.5 Increasing Difficulty

To increase the difficulty, the game's speed will gradually increase depending on the players' current score. The process called GameSpeedController will determine the speed of the game. The higher the score, the faster the speed of the game.

Below is the relevant code.

```
GameSpeedController: process(clock_signal, value) begin
        if(value < 30) then
            clock_signal <= clk100Hz;
        elsif(value >= 30 and value <= 50) then
            clock_signal <= clk150Hz;
        elsif(value > 50 and value <= 70) then
            clock_signal <= clk200Hz;
        else
            clock_signal <= clk250Hz;
        end if;
end process;
```

Figure 4.5.1

4.6 Game Over

In this game, the player loses when the bird hits the ground, pipe, and the top of the screen. To determine if the bird hits the ground or the top of the screen, simply check its vertical position. To determine if the bird hits the pipe, simply check if the bird's position overlaps the pipes. If the bird hits the ground, pipe, and on top of the screen, the state of the game will transition from game_play to end_screen, where the word "Game Over" will be displayed to the user. The relevant code can be found in Figure 4.6.1.

```
when game_play =>
    if ((b_end_x > p1xs and b_start_x < p1xe) and (b_start_y < p1ys or b_end_y > p1ys + gap)) or
       ((b_end_x > p2xs and b_start_x < p2xe) and (b_start_y < p2ys or b_end_y > p2ys + gap)) or
       ((b_end_x > p3xs and b_start_x < p3xe) and (b_start_y < p3ys or b_end_y > p3ys + gap)) or
       ((b_end_y > V_END and vcount <= ground_y) or (b_start_y < V_START)) then
       game_state <= end_screen;
    end if;
```

Figure 4.6.1

After the game is over, the position of the pipes, birds, and clouds is restored to its initial value. The score on the Pmod SSD will remain so that the user can see their scores. When the user presses the BTND button, the game state will transition from end_screen to splash_screen, where the user can start a new game.

```
elsif (game_state = end_screen or game_state = splash_screen) then
    p1xs <= H_END - 60 - 341;
    p1xe <= H_END - 341;
    p2xs <= H_END;
    p2xe <= H_END + 60;
    p3xs <= H_END + 341;
    p3xe <= H_END + 60 + 341;

    p1ys <= V_START + 150;
    p2ys <= V_START + 200;
    p3ys <= V_START + 100;
```

Figure 4.6.2: reset the pipe positions

4.7 LCD Display

In this project, the process to drive the VGA display has been implemented on a module called “vga_controller” on a separate VHDL file. This module aims to drive the red, blue, and green signals to display the desired colors inside the 1024 x 600 addressable video area.

To display the pipes, only the green RGB signal is needed when vcount is not between p1ys and (p1ys + gap size), and hcount is between p1xs and p1xe. The same goes for pipes 2 and 3. Below is the relevant code (Figure 4.7.1)

```
if((hcount >= p1xs and hcount <= p1xe) and (vcount >= p1ys + gap or vcount <= p1ys)) then
    red   <= "0000";
    green <= "1111";
    blue  <= "0000";
elsif((hcount >= p2xs and hcount <= p2xe) and (vcount >= p2ys+gap or vcount <= p2ys)) then
    red   <= "1111";
    green <= "0000";
    blue  <= "0000";
elsif((hcount >= p3xs and hcount <= p3xe) and (vcount >= p3ys+gap or vcount <= p3ys)) then
    red   <= "0000";
    green <= "0000";
    blue  <= "1111";
```

Figure 4.7.1: Code for displaying pipe 1,2 and 3

To display the Bird, two integer variables are defined, namely dy, dx. Then dy, dx is assigned as the difference between the current hcount and vcount values, respectively.

```
display_proc: process(hcount, vcount)
variable dx, dy, tx, ty, tx2, ty2, tx3,ty3, tx4, ty4, csx, csy, clx, cly: integer;
variable cs1x, cs1y, cl1x, cl1y, cxlx, cxly, abx, aby: integer;
begin
    dx := (hcount - b_start_x);
    dy := (vcount - b_start_y);
```

Figure 4.7.2: Code for the position of the bird

After defining the variables, the value of the bird's image can be accessed using the 23 x 25 2D array representation of the bird's image. The array representation has five values, 0, 2, 3, 4, 5. The values of 0, 2, 3, 4, and 5 represent the colors cyan, white, orange, red, and black. For example, If bird(dy, dx) = 5, black color is produced on this pixel. If bird(dy, dx) = 3, then orange color is produced. Below is the relevant code.

```
if(bird(dy, dx) = 5) then -- black
    red <= "0000"; green <= "0000"; blue <= "0000";
elsif(bird(dy, dx) = 4) then -- red
    red <= "1111"; green <= "0000"; blue <= "0000";
elsif(bird(dy,dx) = 3) then -- orange
    red <= "1101"; green <= "0110"; blue <= "0000";
elsif(bird(dy,dx) = 2) then -- white
    red <= "1111"; green <= "1111"; blue <= "1111";
else
    red <= "0000"; green <= "0101"; blue <= "0100";
end if;
```

Figure 4.7.3

The code defining the 2D array which represents the bird is:

```
type angry2D is array(0 to 23, 0 to 25) of integer range 0 to 9;
signal bird: angry2D;
```

```
bird(0, 1) <= 0; bird(0, 2) <= 0; bird(0, 3) <= 0; bird(
bird(1, 2) <= 0; bird(1, 3) <= 0; bird(1, 4) <= 0; bird(
bird(1, 9) <= 0; bird(1, 10) <= 0; bird(1, 11) <= 5; bir
; bird(1, 16) <= 0; bird(1, 17) <= 0; bird(1, 18) <= 0; b
; bird(1, 23) <= 0; bird(1, 24) <= 0; bird(2, 0) <= 0;
bird(2, 2) <= 0; bird(2, 3) <= 0; bird(2, 4) <= 0; bird(
bird(2, 9) <= 5; bird(2, 10) <= 5; bird(2, 11) <= 5; bir
; bird(2, 16) <= 5; bird(2, 17) <= 0; bird(2, 18) <= 0; b
```

Figures 4.7.4 and 4.7.5: defining the bird's 2D array and assigning values (partial)

To display the two small, two medium, and one large-sized cloud, a total of 5 pairs of integer variables are needed.

After defining the variables, the value of the clouds image can be accessed using the 26 x 38, 35 x 51, and 53 x 77 2D for the array representation of small, medium, and large clouds, respectively.

The array representation has three values, 0, 1, 2. The values of 0, 1, 2 represent the colors white, black, and cyan.

For example, If cloud_small(csy, csx) = 1, black color should be produced on this pixel. Below is the relevant code.

```
if(cloud_small(csy, csx) = 1) then
    red <= "0000"; blue <= "0000"; green <= "0000";
elsif(cloud_small(csy, csx) = 0) then
    red <= "1111"; blue <= "1111"; green <= "1111";
else
    red <= "0000"; green <= "0101"; blue <= "0100";
end if;
```

Figure 4.7.6

The code defining the 2D arrays of the clouds is:

```
package cloudpkg is
    type cloud_small2D is array(0 to 26, 0 to 38) of integer;
    type cloud_large2D is array(0 to 35, 0 to 51) of integer;
    type cloud_xl2D is array(0 to 53, 0 to 77) of integer;
end package;
```

Figure 4.7.7: The array representations of 3 cloud sizes

```
cloud_small(0, 0) <= 2; cloud_small(0, 1) <= 2; cloud_small(0, 2)
cloud_small(1, 0) <= 2; cloud_small(1, 1) <= 2; cloud_small(1, 2)
cloud_small(2, 0) <= 2; cloud_small(2, 1) <= 2; cloud_small(2, 2)
cloud_small(3, 0) <= 2; cloud_small(3, 1) <= 2; cloud_small(3, 2)
cloud_small(4, 0) <= 2; cloud_small(4, 1) <= 2; cloud_small(4, 2)
cloud_small(5, 0) <= 2; cloud_small(5, 1) <= 2; cloud_small(5, 2)
cloud_small(6, 0) <= 2; cloud_small(6, 1) <= 2; cloud_small(6, 2)
cloud_small(7, 0) <= 2; cloud_small(7, 1) <= 2; cloud_small(7, 2)
cloud_small(8, 0) <= 2; cloud_small(8, 1) <= 2; cloud_small(8, 2)
cloud_small(9, 0) <= 2; cloud_small(9, 1) <= 2; cloud_small(9, 2)
```

Figures 4.7.8: assigning representation values to the small cloud image (partial)

To display the words, take the word “Flappy Bird” as an example. First, defined the integer variable, ty, tx.

After defining the variables, the value of “Flappy Bird” can be accessed using the 59 x 593 2D array representation. The array representation has two values, 0 and 1. The values of 1 and 0 represent the colors white, and green respectively.

For example, If flappy(ty, tx) = 1, white color should be produced on this pixel. Below is the relevant code. The relevant codes are shown below:

```
if (flappy_W(ty, tx) = 1) then
    red   <= "1111"; green <= "1111"; blue  <= "1111";
else
    red <= "0111"; green <= "0110"; blue <= "0000";
end if;
```

Figure 4.7.9

```
type flappyWord is array (0 to 59, 0 to 593) of integer;
signal flappy_W: flappyWord;
```

Figure 4.7.10: Array representation for the word “Flappy Bird”

```
flappy_W(0, 0) <= 1; flappy_W(0, 1) <= 1; flappy_W(0, 2
6) <= 1; flappy_W(0, 7) <= 1; flappy_W(0, 8) <= 1; flap
flappy_W(0, 13) <= 1; flappy_W(0, 14) <= 1; flappy_W(0,
(0, 19) <= 1; flappy_W(0, 20) <= 1; flappy_W(0, 21) <=
<= 1; flappy_W(0, 26) <= 1; flappy_W(0, 27) <= 1; flapp
flappy_W(0, 32) <= 1; flappy_W(0, 33) <= 1; flappy_W(0,
(0, 38) <= 1; flappy_W(0, 39) <= 1; flappy_W(0, 40) <=
<= 0; flappy_W(0, 45) <= 0; flappy_W(0, 46) <= 0; flapp
```

Figure 4.7.11: Assigning representation values to the word “Flappy Bird” (partial)

After implementing the above codes, the result is:

Figure 4.7.12: “Flappy Bird” word on the display

5. Results and Discussions

5.1 Achievements

In the project, I have implemented a 2D game that resembles the original game. For example, the splash screen, bird movements, pipe, and ground collision are from the original game. Also, I made some modifications, such as using the “Angry Bird” image instead of the flappy bird image for this game. Moreover, I have built some Python helper functions to draw and scale up the characters or images, making life easy to draw things into the LCD display.

5.2 Limitations

5.2.1 Pmod SSD Score Display

In this project, since only a 2-digit Pmod SSD is provided to display the player’s score, the maximum score is limited to 99. Once the player scores more than 100 points, the player will not see their actual score.

One solution that came to my mind is once the score is 100, the LED on the ZedBoard will light up, indicating the user has reached 100 points. Then, reset the Pmod SSD. For example, suppose that we set up the LED as std_logic_vector(7 downto 0) for LD7 to LD0 when the user scores 220, then the LED will be “00000010”, and Pmod SSD will display the number 20. Essentially, the LEDs show the hundreds digit.

Another solution is to display the score on the screen; although I have displayed words and characters on the LCD display, displaying the scores on the screen requires more time and effort.

5.2.2 VGA Display

In this project, only a VGA Display is provided. The limitations lie in the number of colors I can create since the 4-bits of Red, green, and blue signals are not enough to create a more accurate color.

5.3 Changes

To tackle the limitations above, the game ends once the score reaches 99. Also, making the game much more difficult once the player reaches a certain score. For example, increasing the speed of the game.

5.4 Further Improvements and Possibilities

5.4.1 User Interface

Instead of having only one solid color, I think that the pipes can have stripes of alternating colors. Like the way I draw the birds, clouds, and words on the LCD Display, it is possible to use a 2D array to draw a better-looking pipe if I have extra time.

5.4.2 Record High Scores

It would be great to record the players' high scores if permanent storage is available.

5.4.3 Sound Effects

It would be great to have sound effects during gameplay, for example, when the bird passes through the pipes, or when the bird hits the pipe or ground, etc....

6. Division of Labour and Development Timeline

Since this is a one-person project, division of labor is not applicable.

6.1 Initial Stage

- Read and try to understand the report sent by the teaching staff.
- Draw the letters to display some text into the LCD display.
- Write Python code to scale the drawings and convert them to VHDL code.

6.2 Implementation Stage

- Write VHDL codes
- Combine the drawings into the VHDL project
- Add background colors and some images to the splash screen
- Show “Game Over” once the game is over
- Add instructions to help users navigate around the game.

6.3 Final Stage

- Testing the game. For example, a test for losing the game.
- Black Box Testing, check if the game is implemented according to specification.
- Start drafting the final report.

7. References

Dogtiev, A. (2020, June 23). Flappy bird revenue – how much did flappy bird make? Retrieved March 19, 2021, from https://www.businessofapps.com/data/flappy-bird-revenue/

Flappy bird wallpapers - wallpaper cave. (n.d.). Retrieved March 19, 2021, from https://wallpapercave.com/flappy-bird-wallpapers

Li, M., & Lam, H. (n.d.). Final Report CENG 3430 The Flappy Bird (Rep.). Hong Kong.

YANG, M. (2021, March). Lecture 05: Use of Clock Sources and Peripheral Modules on ZedBoard. Retrieved April, 2021, from http://www.cse.cuhk.edu.hk/~mcyang/ceng3430/2021S/Lec05%20Use%20of%20Clock%20Sources%20and%20Peripheral%20Modules%20on%20ZedBoard.pdf

YANG, M. (2021, March). Lecture 06: Driving VGA Display with ZedBoard. Retrieved April, 2021, from http://www.cse.cuhk.edu.hk/~mcyang/ceng3430/2021S/Lec06%20Driving%20VGA%20Display%20with%20ZedBoard.pdf

Yatesy, K. (n.d.). Angry bird red. Retrieved May 09, 2021, from http://www.minecraftpixelarttemplates.com/2012/08/angry-bird-red.html