Compsys 305 Project(Group APA)

*Abstract*

In this report it will cover the block diagram and game design of our project. The goal of our project is to design a game which is based off the game flappy bird using a DE0 board displayed on a VGA board. The design of our game is created in key components such as the pipes which generate the obstacles in the game and the mouse allows the user to control the bird. The user is able to choose from two modes, training mode and game mode. As the player progresses through the game the level will increase, and the bird will move faster. There will be special gifts created which allow the user to pass through objects. The controls are the mouse and the DE0 board switches which allow the user to operate the game. There are 5 main components in the design. These are the mouse, clock divider, game FSM, game, and the VGA sync. The game has a simple design in which will allow the game to use less memory to run.

# Introduction

The goal of this mini project is to design and implement the game Flappy Bird on the DE0 board in VHDL. Users will control and play the game using a PS/2 mouse, DIP switches and push buttons on the DE0 board. The game will be displayed on a VGA board with a resolution of 640 x 480 pixels.The game is a side-scroller where the player controls the vertical movement of a bird to avoid obstacles by flying between the gaps of oncoming rows of pipes. The left click on the mouse raises the bird and the bird falls when there's no user input. The player's score increases whenever a gap between pipes is passed. The aim of the game is to achieve a high score.Our implementation of the game will have two modes: training and regular. In training mode the speed of the oncoming rows of pipes is constant at the lowest speed available in the game. In this mode the bird will have an infinite number of lives. Falling to the bottom of the screen will still prompt the game over screen in which the user can use a push button (PB0) to return to the main menu or restart the run (PB1). The user can pause (prompting a pause screen) and resume the game at any time using DIP switch 0 on the DE0 board.In regular mode the speed of the pipes will increase after ~7.5 seconds, and then increase again after another ~7.5 seconds. The increase in speed increases the difficulty of the game as the player has less time to react to the obstacles. In regular mode, the bird has a finite number of lives which begins at 3 and only increases if the user passes 10 pipes or collects “gifts” which fly across the screen in straight lines. The amount of lives decreases whenever the bird collides with a pipe. When the bird runs out of lives or falls to the bottom of the screen, the game over screen is prompted and the user can either quit to the main menu or restart the run. If the player achieves a score of 100 (passes through 100 pipes), a victory screen will appear. Like the training mode, the user can also pause the game at any time. The gift system that allows the user to gain additional lives is a unique feature to our implementation that wasn’t present in the original game. Additional lives allow the user to reach higher scores and adds another dimension to the gameplay as users won’t only focus on avoiding obstacles but also try to collect the oncoming gifts.

# System Block Diagram

This game is created by key components such as the display, mouse and vga sync. Our entire system hierarchy is shown below in figure 1 and in figure 2.

Our game is comprised of 5 components, the “altpll0” (which is the clock divider from 50MHz to 25 MHz), the “mouse” which controls the mouse, “game\_fsm” which is the FSM for our game, “game” component which contains the logic for bouncy ball, pipes and the gift, and the VGA\_SYNC component which outputs the RGB signal to the monitor. As shown in Fig.2, the game FSM sends the selected mode to the game component, which and sends the RGB signal to the VGA sync depending on the selected mode.

For the game component, we have used structural modelling to create the component. The benefit of this design was that we did not have to have a separate datapath that took in RGB signals from ball, pipes and gift separately and output depending on the state. Instead, we could just have one component in the block diagram called “game” that contained all of these as sub-components. However, the negative of this design was that modifications to one of the interconnected components could affect the other components. Also, making alterations to the design would be difficult without knowing how all the components were connected to each other and what the system hierarchy of the game component looked like.

As shown in the figure 1 the hierarchy model of the game component, from the top level underneath the game component, there are 5 components, and the 4 components menu, gameover, pause\_screen and winner represents the different screens that are shown in the VGA monitor depending on the 3-bit signal from the FSM (eg. the menu screen would be shown if the FSM sends “000”). The bouncy ball component and its sub-components are used to implement the elements of the game (bird/ball, pipes and the gift). The game component determines what RGB signals are sent to the VGA sync component, and we’ve made it so depending on the selected mode from the game FSM. If it is the menu screen or winner screen it will be a green background with white text. If it is the game over screen it will be a red background with white text. If it is the pause screen, it will be a blue background with white text and lastly if it is the training or game mode it will be a blue background with white text, yellow bird, and green pipes.

# Game FSM

Our overall FSM diagram of our system is shown in figure 3. The initial state that the user starts is the menu state. From here, depending on the input, the next state will either be the regular or training mode. This is determined by either push button 1 or push button 2. From the training or regular state, the next state will either be the pause, win or game over state. The pause state is determined if switch 0 is enabled. The win is enabled if the user reaches a score of 100 and the game over state is determined if the life count reaches 0. The next state for the win and gameover state is the initial menu state. This is enabled when the reset signal is on. For the pause state, the next state will be the regular or training state depending on the previous state when the switch is off.

# Game Design/ Components

## FSM

A key component of our design is the game FSM. The FSM determines the transitions from one state to another.

As shown in figure 4, the FSM takes in 7 input signals, clk, reset, PB1, PB2, SW\_pause, dead and win. Depending on the input, the FSM sends the output signals game\_reset, game\_pause and selected\_mode. The reset, PB1, PB2 and SW\_pause are connected to the buttons/switch on the altera DE0 board, which are PIN\_H2, G3, F1 and J6.For the fsm, there are 6 different states, menu, regular, training, gameover, pause and win. Depending on the input to the fsm, the fsm sends a 3-bit signal out called “selected mode” which is inputted to the “game” component. If the player switches up SW0 (SW\_pause) then the game FSM sends the game\_pause signal, which is used to display the “paused” screen. If the player presses the reset button, the game FSM sends the game\_reset signal, which is used to show the main menu. The outputs of the FSM component that are sent to the game component depending on the state which is shown in figure 5.

## Main Menu

The design for the Main menu screen can be found under the menu.vhd. The main menu is shown below in figure 6.

The main menu has a green background with white text. It displays the Team name with the controls.

## Bouncy Ball

The bouncy ball component encompasses the logic for the “bird” or the character of our game. The implementation is simple, with the ball being an 8x8 pixel cube with a fixed x-position of 250 (only moves up and down). As shown in the figure above the bouncy ball takes in 8 inputs and has 8 outputs. For the user to control the ball, during every vertical sync, if the user is pressing the left mouse button, the ball\_y\_motion would be increased and added onto the y-position of the ball and decreased if the user is not pressing the button to emulate the element of falling due to gravity. To detect the collision with the pipes, we would compare the x,y position of the ball and the x,y position of the pipes, and have a flag called “ball\_collision”. If the position of the ball is within the size/position of the pipes, and if ball\_collision is 0, then we would set ball\_collision to 1. This was to avoid the ball from colliding with the pipes multiple times. The variable life would be reduced by one and the ball\_collision flag would be reset back to 0. To detect collision with the gift, the same logic was used with a different flag called “touch\_ball” and the life would be increased by 1 with a cap of 9 max. If the life is equal to 0, signal “dead” would be set to 1, and would send the user to the gameover screen as shown in the figure below. We have also created a variable called “score” and allowed the user to win the game if the user achieves 99 points. Through a conditional statement, if the score goes to 99, the user would be sent to the winner screen, where they are greeted with a “You Win!” message.

## Pipes

The pipes component encompasses the logic for the pipes or the obstacles for our game. The pipes are 30 pixels thick and with a 140-pixel wide gap between the top pipe and the bottom pipe that the ball can safely pass through without losing its life. Two pipes at a time would be shown on the screen, with pipe 1 starting at x-position of 630 and pipe 2 starting at 960. The pipes then would move at a constant speed to the left at every vertical sync. For the actual game mode, we added 3 levels, with the pipe speed increasing incrementally with every level. The gaps/center of the pipe1 and pipe2 are determined by the LFSR random number generator. When the ball collides with the pipe, the pipe1’s x-position would be reset to 630, and the pipe1’s gap/center would be replaced with pipe2 (the pipe that was supposed to come after pipe1 prior to collision)’s center. A new value for the next pipe2 would be created using the LFSR random number generator. For the levels we would increase the level by every 7.5 seconds. Using the vertical sync, which is incremented by 1 roughly every 0.015 second (699\*524 / 25MHz), we have a variable “count”, that will be counted to 500 before it is reset to 0 and increase the level by 1. Hence, the level would be increased every (0.015 \* 500 = 7.5 sec) 7.5 seconds with the level capping at 3.

## LSFR

The LFSR we used for this project was a galois LFSR, where the XOR gates are placed between the registers 0, 2, 3, and 4 as shown in figure 11. It also starts with a seed value of “10101010”, which is a non-zero value that ensures the output is not stuck at seed value. This generates a “temp” bit, which is then put into the register 0 and each other bits in the other registers are shifted up one position, generating a new random 8-bit number. The 8-bit number is then appended with a 0 to create a 9-bit number (to match the 9-bit y-position value) and we add 90 at the end so it could produce the range values that we want. This output number is then used for generating the random y-position for the gaps between the pipes and the gifts. The reset input signal will reset the seed value back at “10101010”.

## Clock divider

This key component is the clock divider which will take in the input of a 50MHz clock. However, the common VGA display standard is 25 MHz pixel rate. The clock divider will halve the 50MHz clock and output a 25MHz clock. This output will be connected to all the components in the system including the mouse, game, FSM, and the VGA sync components.

## Gifts

The gifts component encompasses the logic for the gift, which gives an extra life if the ball touches it. The gift has a random x and y position, both determined by the LFSR random generator. The x-position of the gift is calculated by: 1000 + random number generated by the LFSR, and is decremented by a constant number every rising edge of vertical sync to make it move to the left. If the speed of the gift is slower/equal to the pipes, we had an issue where the gift appeared to be moving to the right/fixed in place. To combat this issue, we made the gift to move faster (speed = 5) than the pipes, and the pipes at max level of 3 would move at a speed of 3 (incremented by 1 every level). This made the game more challenging, as the user would have to face the pipes getting faster with each level and the gift moving faster than the pipes meant that getting extra lives was difficult. To ensure that the y-position of the gift was different from the gap of the pipes (since if they were the same it would make the game very easy as players would get extra lives as they passes through the gap of the pipes), we added bits 6 to 0 of the random number and would add onto the random number (randnum + randnum(6 downto 0) to vary the gift’s y-position. This made the gifts appear at a different place to the pipe gaps, and the users are faced with a decision to go for an extra life at a risk of colliding with the pipes or to play it safe and not get the gifts.

# Performance

Our team’s implementation of the game is relatively simple as we didn’t use any sprites or detailed graphics for the various visual components such as the bird or pipes (fig 14). Because of the graphical simplicity, our design only uses 1242 logic elements as shown in figure 15. This is one of the design tradeoffs we encountered during development. While implementing sprites would have made the game more appealing to users, we believe that the lower logic elements and more optimized performance makes the gameplay experience more enjoyable for the player. This would allow the game to run at a lower memory cost and would allow the game to run more smoothly.

The maximum operating frequency of our game is 611.25 MHz, restricted to 559.91 MHz in the game fsm component, which is sending a state\_gameover clock, shown in figure 16.

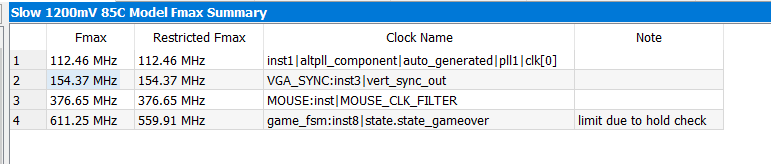


Figure 16: Fmax summary

##### Acknowledgment

Our team would like to acknowledge the lecturers Morteza and Maryam for their lectures on teaching VHDL and FSM. Also, to thank the TA assistance during the lab while implementing VHDL and using the DE0 board.

##### References

1. Projects, E., 2021. Linear-feedback shift register (LFSR) design in vhdl. [online] Engineers Garage. Available at: <https://www.engineersgarage.com/vhdl/feed-back-register-in-vhdl>