CompSys 305 Final Report

Eric Wong and Simon Dawidowski
University of Auckland, ewon466@aucklandun.ac.nz, sbud159@aucklanduni.ac.nz

Abstract - **The project was to create a game similar to Flappy Bird on a DE0 board. The goal of the game is to acquire points by utilizing pickups and avoiding the oncoming obstacles while the game progressively speeds up. Features such as pickups and a linear-feedback shift register (LFSR) were added to increase randomness to ensure that each playthrough is unique and challenging. A high-level finite state machine is utilized in order to differentiate between the different states and what the controls do in each state. Although we are happy with the end result, if given more time, we would add more pickups and a way to store high scores.**

Index Terms - DE0 board, game, VGA-ready, VHDL

Introduction

For this project, the task was to create a game similar to *Flappy Bird[1]*. In this game, the player controls a bird which is constantly falling by jumping to move up slightly. The player is to navigate between pipes, while not colliding with the pipes or ground. Jumping in the game costs energy in which the player has a set amount at the start of the game. If the player runs out of energy, they will not be able to jump. Pickups are available throughout the game which give the player energy. The goal of the game is to last for as long as possible while managing energy. The players score increases every time a pipe is passed. The game ends when the player collides with the floor or pipe and the final score is calculated.

Game Strategy

I. Rules and Features

There are many features added which differ from the original *Flappy Bird* that this game was based on. Some of these features include:

- **Training mode**: Designed to make it easy to learn the controls, items, and mechanics. In the training mode the level will not speed up as your score increases, it will remain at the slowest speed.
- **Energy**: When the player makes the bird jump, energy is consumed. The player starts with a set amount of energy and can gain more energy from pickups. If the player runs out of energy, the bird will not jump
- **Pickups**: Placed randomly throughout the game, they give the player more energy when the bird collides with them.
- **Increasing difficulty**: While running the game in normal mode, the screen scrolling speed will increase a
- score increases

II. Man-Machine interfaces

The Player interacts with the game though the different interfaces. These interfaces allow the player to control what happens in the game and provide the player with visual feedback of what they just did. These interfaces include:

- **Mouse**: The main player's main controller. Both the left and right mouse buttons are used to select options while in any menu state. While in the game, the left mouse button is used by the player to make the bird jump while the right mouse button pauses the game.
- **DIP switch**: When on, sets the game to tutorial mode. When off, sets the game to normal mode
- **Push button**: Resets the high score
- **7-seg display**: Displays the player's highest score since the device has been powered on
- **VGA display**: Updates every clock cycle to display changes in the game

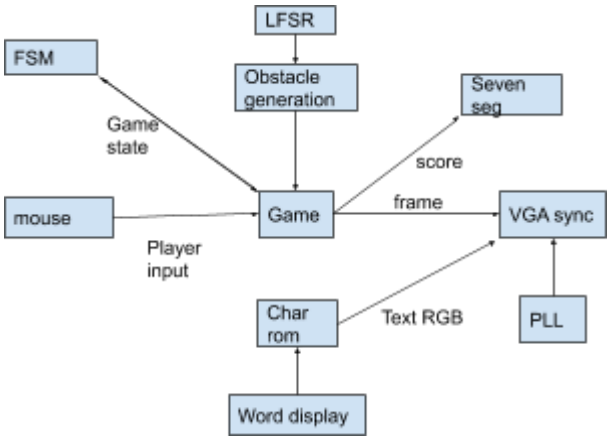
III. How to Play

The player controls the bird's jump with the left mouse button. The main goal is to get as many points as possible before dying. To achieve this, the player must have energy in order to jump through the gaps of the pipes. The player must decide whether to risk going for the pickup, in which they might malposition themselves and not be able to go

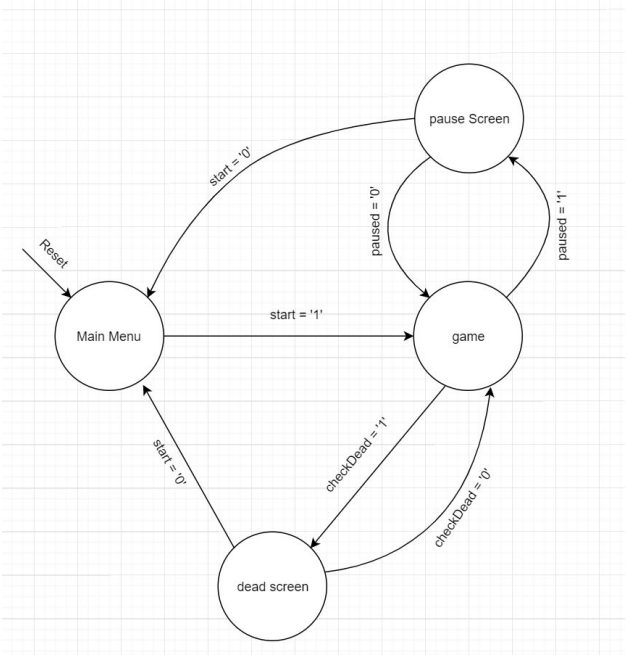
though the next gap, or miss the pickup and risk running out of energy. This choice becomes much harder as the game speeds up and the player may not have time to think about it. When the game ends, the player is shown their score for that run and can click the left mouse button to restart. There are four different speeds that the game runs at, the first change in speed is after the player reaches a score of 6, the next on is at a score of 20 and the last speed change is at a score of 30.

Implementation

I. High level block diagram



Mouse[2]: Takes user input from the mouse and converts it into signals that the device understands.
LFSR: Generates a random number which is used to select the next obstacle to place in the scene
Obstacle generation: Takes the random number and creates pipes and energy power ups. There are only two pipes created, when they get to the left side of the screen they are reset to the right side and a new gap position is generated. There is always one energy power up on screen, it either gets to the end and resets or the player hits it and resets.
Char ROM[2]: Stores pixel information of all characters. Only accessed by 'char disp'
Word display: Depending on the state, this block places the characters in the right position. Outputs to the VGA sync, if the current pixel is part of a char.
Game: Computes the location of where to render objects each frame and the logic of interactions between them.
7 seg display: Receives the score from the current playthrough, decodes it, and displays it on the 7 segment display
VGA sync[2]: Aligns and assigns the correct pixels and colours to the monitor for each frame
PLL: Divides the 50MHz clock signal into the required 25.2MHz required by the monitor



II. FSM description

The FSM has four states; 00,01,10,11. These represent the menu, game, dead, and paused state. This implementation of the FSM is used to track what the game behaviour should be. It selects what text overlay appears on the display, and when to allow the player to control the bird. The FSM is also responsible for setting the reset signal. For the code to function properly it needs to be informed on which state it is in.

Results

I. Resources consumption

The program uses about 17% of the total logic elements, majority is combinational logic. Only about 2% is sequential. The collision and reset code does the most computation, this is where most of the LEs are used. Given more time the code could be optimised. The program uses less than 1% of the total memory bits, this is where the character data for rendering ascii characters is stored. A total of 48 pins were used to connect the various I/O sources used in the project.

Flow Status	Successful - Thu May 30 00:14:59 2019
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	miniproject
Top-level Entity Name	miniproject
Family	Cyclone III
Device	EP3C16F484C6
Timing Models	Final
Total logic elements	2,561 / 15,408 (17 %)
Total combinational functions	2,512 / 15,408 (16 %)
Dedicated logic registers	377 / 15,408 (2 %)
Total registers	377
Total pins	48 / 347 (14 %)
Total virtual pins	0
Total memory bits	4,096 / 516,096 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 112 (0 %)
Total PLLs	1 / 4 (25 %)

II. Time analysis

The maximum operating frequency is 25.2MHz, this is due to the vga standard. We could increase this by running the other components of a 50MHz DEO clock, however is this unnecessary as 25.2Mhz is more than enough for this application.

Conclusion

The project assigned was to recreate a *Flappy Bird* type in VHDL. The game is displayed on a 640 by 480 resolution VGA monitor. The game has two modes, one to allow practice, and the other which progressively speeds up. The game was designed to be repeatable and so randomly generates pickups and pipe spawn locations so that each run is different.

If given extra time, there are many features that could have been added. One of the features that would have been added are more pickups. Currently, there is only one type of pickup in the game which increases energy on picking it up. Different type of pickups such as ones that decrease energy on pick up would be added in order to make the game more challenging. This would make the game more difficult because it means that the player must pay attention to what pickup they are picking up rather than blindly going for every one. Another thing that could have been implemented is the use of images. Instead of hardcoding each pixel to a certain color, the system would render the images instead. This will allow for more complex and interesting models compared to the simple drawn shapes currently. This was originally planned however the lack of knowledge meant that implementation would have taken too long.

References

[1] https://flappybird.io/

[2] Files provided by the course i.e. vga sync, mouse

[3] Course material i.e lecture slides

[4]Muhammad Nadeem

[5] Morteza Biglari-Abhari

Appendix