305 Final Report

Produced by Luke Ryan, Darren Ho,
Thomas Yeh
ECSE
University of Auckland
NZ, Auckland
Emails: lrya138@aucklanduni.ac.nz,
dho524@aucklanduni.ac.nz,
cyeh243@aucklanduni.ac.nz

***Abstract*—The project is about creating a version of the mobile game *Flappy Bird*. Using VHDL code to program the Altera Cyclone V FPGA of the DE0-CV Board. It will be a similar game that adds new features like gifts, changing difficulties/modes, and obstacles. We used a finite state machine to handle the current game state. A linear feedback shift register (LFSR) to generate pseudo-random numbers to randomize the obstacles, and gifts. The display will be outputted by the VGA. The player's inputs will be read from both the PS/2 mouse and switches/push buttons of the DE0-CV board. The final product mostly meets how we initially envisioned it. It runs well, and provides a fun experience, which all games should. Trial and error on gameplay, resources, and performance, was a good learning curve that helped us to find the best solutions. In the future if given more time we would want to better optimize the game and add more features, like new gifts.**

***Keywords*—VGA, Flappy Bird, VHDL, FSM, DE0-CV Board, bird/ball**

I. Introduction (*Flappy Bird*)

We're tasked to create a game that emulates the features of the popular mobile game Flappy Bird. The game consists of a bird/ball (yellow block) and pipes, it is up to the player as the bird to jump up and down through the pipes, until the player eventually dies by hitting the ground/sky or a pipe (obstacles). The goal of each playthrough is to reach a higher score each time. In this version of the game, we have created we have added onto the features that already exist. The game will now have a training and normal mode, where the player can practice as much as they need in training mode, until they feel confident enough to begin scoring in normal mode. For normal mode we have added stages of difficulty that each difficulty will trigger based on how long the player survives. The player will now have a health bar that will deplete a single life after each collision with an obstacle. Dying once the health bar is fully depleted. There will be gifts to aid the player as they progress, such as heart (red block) gifts that will replenish some of the player's health bar.

II. Design Specifications

The game will be implemented by programming a DE0-CV board with our designed VHDL code. The player will have control over the upward and downward motion of the bird via a PS/2 mouse. Also, the player can interact with the game by the DIP switches and push buttons.

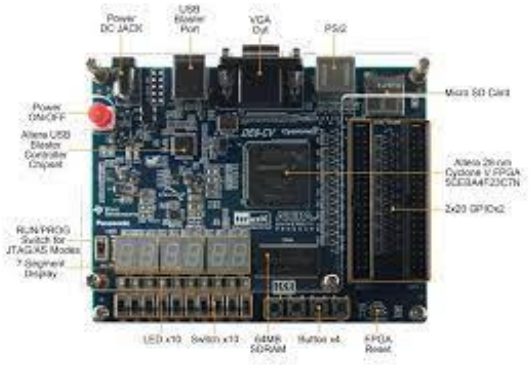

Fig.1. DE0 –CV FPGA board [2]

The game will be displayed on a Video Graphics Array (VGA) screen at a 25MHZ pixel rate with 640 x 480 resolution. The screen will produce 60 frames per second, avoiding flickering.

III. Game Strategy

A. How To Play

The goal of the player is to pass through as many pipes, and to survive as long as possible. Throughout the playthrough, randomly generated gifts will appear to aid the player. A 'heart' will replenish the health of the player's health bar, depending on whether the player is down some health. The player will finally die once the health bar has been fully depleted, which is depleted by either colliding with the ground/sky or the pipes coming towards the player. Once a playthrough is completed the score will be judged by the time indicated on the seven-segment and VGA display for how long the player survived.

B. New Features

With this version of the game, we built upon the original Flappy Bird game, adding many new features. Following of which were:

- Training Mode:
    - This mode was created so that the player could practice the game's functions in an easier setting. The player will have a larger health bar and the game will only be set on the easiest difficulty for the whole playthrough. There will be no score for this mode.

- Normal Mode:

- This mode is the ordinary way to play the game. The difficulty of this mode will increase every 30 seconds, until a maximum of 4 minutes, where then the max difficulty will be reached. The final score will be determined on the time survived.

- Heart
  - This is one of the gifts of the game that will be randomly generated at different locations. It will replenish the health bar loss of the player by one health slot.

- Time Score
  - Differing from the original game where score was only on passed pipes, now it will be based on time survived.

C. *Implementation*

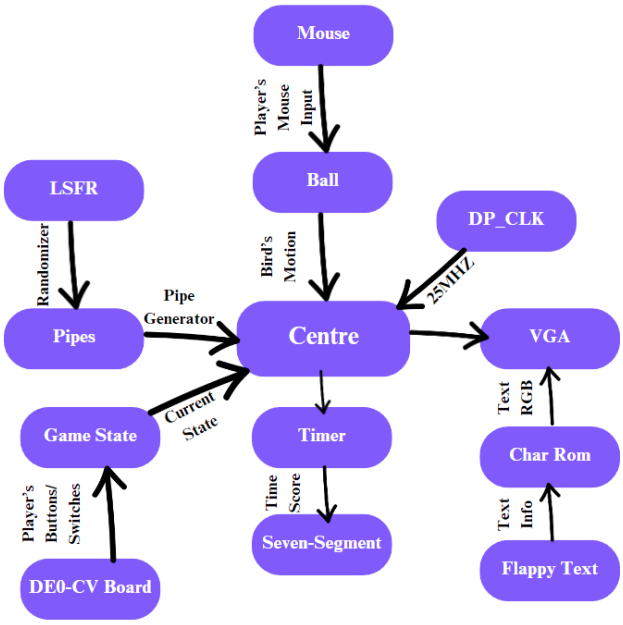

Fig. 2. High Level Block Diagram

Centre: This block indicates where common logic is used across multiple blocks. Such as:

- 25MHZ clock signal.
- Ball's x and y position.
- Pipe's x and y position.
- RGB colours.

LSFR: The Fibonacci Linear Feedback Shift Register (LSFR) is used to generate a random 6-bit binary value to be provided for pipes. This works via a shift register whose input is the linear function of the current state and feedback points (taps). This occurs in a loop that will have a length of 63 iterations. The 'Nextstate' will be calculated by the shifted 'Currstate' and appended feedback bit value. The feedback value is calculated on the 5 LSB of 'Currstate.' This is to ensure that every new clock cycle, 'Nextstate' will register a new 6-bit binary value to the output.

Pipes: Uses generated random 6-bit binary value to select the gap of each pipe. Pipes move from left to right of the screen. Once they get to the left side they are reset back to the right side. Here, detection conditionals will determine based on the ball's y position if the player loses health if they collide with an obstacle (sky/ground or pipe) or gain health if it's the 'heart.'

Mouse: The mouse is the primary interface through which the player interacts with the game. Data is transmitted at one byte at a time, within a sequence of 11-12 bits (1 start bit, 8 data bits, 1 parity bit, 1 stop bit, 1 acknowledge bit). It has three different bus stages:

- Idle – This is when both the Data and Clock lines are kept high, allowing the mouse to begin transmitting data.
- Inhibit – Pulling the Clock line low stops communication
- Host Request-to-Send – A signal indicating to start clock pulses to begin sending data.

Once the mouse runs a self-test and passes it, we can begin receiving inputs from the player based on either left or right clicking the PS/2 mouse.

Ball: Receives commands from either the left or right click of the mouse to change the bird's motion upwards. The bird will show a downward motion (gravity) if there is no command. If the '***pipes'*** block determines a collision, the output will be received here to adjust accordingly.

Timer: A 25MHZ timer that is used to count how long the player survives and show this on the Seven-Segment display.

Seven-Segment: Displays timer value received onto the DE0-CV board' seven-segments.

DP_CLK: Provides the clock signal of 25MHZ, for the VGA sync, and all other components.

Flappy Text: Displays relevant text and objects in desired positions based on current game state.

Char Rom: Used to store and access characters and objects received by 'Flappy Text' to be displayed to VGA.

VGA: Displays the game on a monitor, by redrawing the entire screen 60 times per second (60HZ). Each frame comprises rows (480) and columns (640) of pixels, which are calculated based on the RGB values received at specific positions. These positions are ordered using the pixel rows and columns of the background or objects or text that is synchronised with the vertical and horizontal edges just outside the boundaries of the VGA screen.

Game State: Refer to Section 'IV. Finite State Machine (FSM).'

DE0-CV Board: Receives inputs from push buttons and switches to interact with the game's current state.

IV. Finite State Machine (FSM)

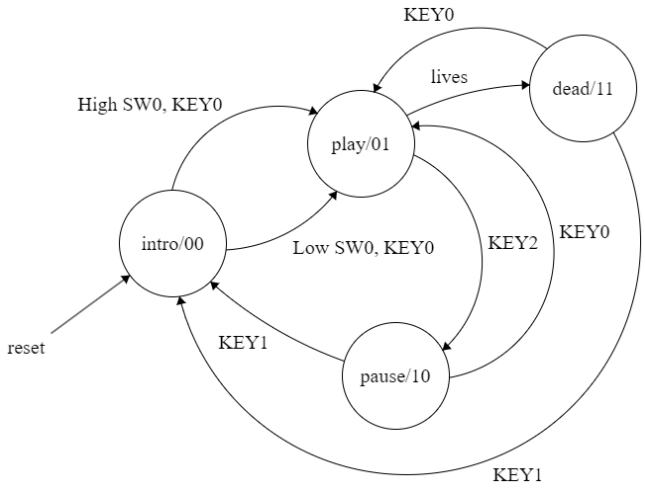

Fig. 3. High Level State Diagram

We have implemented a Moore state machine to handle the game state of where the player will be currently. These are the following four states with their inputs and behaviors:

1. game_state = “00”:

    1.1. The game will be in ‘intro’ state, where the player will be displayed with a home screen. From there, the player can select from two game modes (training mode, normal mode). How the player selects these modes is based on the DIP switch, SW0, and the KEY0 push button. If SW0 is high and KEY0 is pressed, the player will play in training mode. Vice versa, the player will play in normal mode when SW0 is low and KEY0 is pressed. This will then move the player to ‘play’ state.

Fig. 4. Start Screen

2. game_state = “01”:

    2.1. The game will be in ‘play’ state, after the mode is selected in ‘intro’ state. This will be either of the two selected game modes (training/normal mode), where the player using the PS/2 mouse can begin playing the game. The player can pause the game whenever they want. Once the player collides enough with the obstacles to deplete the health bar fully, the player will move into ‘dead’ state.

Fig. 5. Play Screen

3. game_state = “10”:

    3.1. The game will be in ‘pause’ state, when the KEY2 button is pressed, this can only occur in ‘play’ state. While in this state the player and pipes remain where they are. From here the player can either simply continue playing by pushing KEY0 or return to the home screen by pushing KEY1.

Fig. 6. Pause Screen

4. game_state = “11”:

    4.1. The game will be in 'dead’ state after the player’s health bar has been dropped to zero in the ‘play’ state. From here, the player can rerun the game in the same mode by pushing KEY0 going to the ‘play’ state again. Or the player can push KEY1 to transition to ‘intro’ state to select a different mode.

Fig. 7. Dead screen

V. Resources and Performance

Flow Status	Successful - Tue May 28 16:15:24 2024
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	D_H
Top-level Entity Name	main_dd
Family	Cyclone V
Device	5CEBA4F23C7
Timing Models	Final
Logic utilization (in ALMs)	843 / 18,480 (5 %)
Total registers	443
Total pins	53 / 224 (24 %)
Total virtual pins	0
Total block memory bits	4,096 / 3,153,920 (< 1 %)
Total DSP Blocks	0 / 66 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	1 / 4 (25 %)
Total DLLs	0 / 4 (0 %)

Fig. 8. Resource Utilisation

For our design, the game used 443 registers to synthesise the VHDL code. We used 24% of the total pins (53 pins) to connect our I/O from our many components. Our game in the end used less than 1% of the total block memory bits (using 4,096 bits) to store relevant data. We only used 1 phase-locked loop (PLL) out of the total 4. This resource usage of our game was effective. However, something that we wanted to further improve on in the future was to focus on improving our code optimisation so that it could be more synthesisable. Lowering the number of used registers.

	Fmax	Restricted Fmax	Clock Name
1	62.74 MHz	62.74 MHz	VGA_SYNC:inst33|pixel_column[0]
2	69.52 MHz	69.52 MHz	inst|dp_clk_inst|altera_pll_i|general[0].gpll~PLL_OUTPUT_COUNTER|divclk
3	72.98 MHz	72.98 MHz	VGA_SYNC:inst33|pixel_row[4]
4	96.56 MHz	96.56 MHz	VGA_SYNC:inst33|vert_sync_out
5	220.75 MHz	220.75 MHz	timer:inst2|ClockDivider:dummy|vClockOut
6	233.59 MHz	233.59 MHz	MOUSE:inst1|MOUSE_CLK_FILTER

Fig. 9. Timer Analyzer

The maximum frequency of the game given by Quartus' time analyzer tool was 69.52MHZ from the PLL. This frequency was sufficient for the required performance of the game. However, we noticed that if we wanted to implement sprites in the future it is in our best interest to increase the maximum frequency. So, the responsiveness and rendering of the game would not be hindered.

VI. Conclusion

We were tasked with creating a Flappy Bird and designing it with new features on top of its base functionality, while optimizing as best we could. We followed our initial interpretation of the project but found during implementation that changes were required as we better developed our understanding of the project. At the end of this process, with our better understanding of the project in the future we could greatly improve the game.

A. *Decisions/Trade-Offs*

1. Concerning our 'pause' state, we planned to use the KEY2 push button to transition between the player playing and the game being paused. This created a large issue, as the button was the same to move between the 'pause' and 'play' state. We found this to be undesirable, because originally when the clock signal captured KEY2 being pressed, it was impossible to determine which state of the two you would end up in. As even though a press of the button may seem quick to humans, the clock signal is much faster. To fix this problem we changed the button to pause (KEY2) the game and resume it (KEY0).
2. Initially, we planned to implement an invincibility power-up. However, we felt that it gave little advantage to the player, so we decided to remove it.
3. For our planning we wanted to use both a timer and pipes passed counter to determine the player's score. During implementation, we found that using a combination of a timed score and the number of pipes passed score conflicted with each other. So, instead we decided to remove the pipes passed counter.

B. *Future Improvements*

In the future the kind of improvements we would add would be:

- ***Sprites***: This would add an aesthetic look to the game making it look more pleasing. We could switch out the ball with a proper bird design and give the pipes more of a 3D look. This would also allow us to design the background, to give more life to the game.
    - ***Coins***: If the ***sprites*** improvement was successful, we could add a coin gift that could be randomly left throughout the game and be used to purchase different designs for the bird from the home screen.
- Multiplayer Scoring: To help friends play the game in a competitive nature, in the future we could add a recorded leaderboard that shows the top three highest scores with a name associated with them.
- ***More Gifts/Power-Ups***: In the future we could add more gifts/power-ups that can aid the player. This could be a 'slow time' power-up that slows down the pipe's speed for a duration of time, giving the player the ability to readjust.
- ***Music/Sound Effects***: In the future adding an auditory side to the game, such as music or sound effects would better enhance players immersing within the game.
- ***Other Obstacles***: Adding more different kinds of obstacles could both be used for difficulty or variation. These kinds of obstacles could be an opposing bird that flies against the player at a set y position, and if the player hits the bird, they lose some health.
- ***Difficulty Control***: We could add a custom difficulty setting, where the player themselves can decide what difficult they prefer to play at.

ACKNOWLEDGEMENTS

Thank you to A/Prof. Morteza Biglari-Abhari, Dr. Maryam Hemmati, and the teaching assistants for teaching and providing support throughout the course.

REFERENCES

[1] COMPSYS 305 course slides produced by A/Prof. Morteza Biglari-Abhari and Dr. Maryam Hemmati.

[2] DE0-CV Board User Manual author Terasic Inc.