COMPSYS 305 Mini-Project Final Report

Francis Cho
Department of Electrical, Computer and Software Engineering
The University of Auckland
Auckland, New Zealand
fcho544@aucklanduni.ac.nz

Gayeon Kim
Department of Electrical, Computer and Software Engineering
The University of Auckland
Auckland, New Zealand
gkim902@aucklanduni.ac.nz

This report covers Group 20's design and development processes involved in creating a Flappy Bird clone in VHDL for a COMPSYS 305 mini-project. The program was uploaded to a DE0 FPGA board that acted as a game console which could be connected to a VGA monitor and PS/2 mouse to display and play the game. Due to time constraints and unforeseen circumstances, the full game was not able to be implemented as intended. However, intended implementations such as state control and life counters are discussed in detail, as well as other potential improvements. (*Abstract***)**

I. INTRODUCTION

The game we attempted to implement is a 'Flappy Bird' game clone that is programmed using VHDL and compiled into a DE0 FPGA board.

The game mechanics involve the player controlling a bird/ball object that is controlled with the PS/2 mouse. Random obstacles will be generated, and the player must not let the bird/ball object touch these generated obstacles or else a life point will be lost. If three life points are lost, the bird/ball object dies, and the game ends.

The game will feature two game modes: (1) Training mode; (2) Single-player GAME mode. Training mode allows the player to practise with either infinite life points or easier obstacle generation (to be decided). The single-player GAME mode is the normal game mode with the standard three life points.

The game must be displayed on a VGA display with a resolution of 640 x 480 pixels. The screen must be kept in motion from right to left at a constant speed (this may change depending on the difficulty level).

II. COMPONENTS USED

A. About PS/2 Mouse

The PS/2 mouse has bidirectional synchronous serial protocol, and the interface consists of two bidirectional signals which are clock and data. Data is transmitted one byte at a time. Each byte is sent in a sequence consisting of 11-12 bits. Data is shifted out by the host at clock falling edge and is ready by the mouse at clock rising edge.

The communication protocol of the mouse is host to device communication for sending the commands and device to host communication for sending the data [4].

B. About DE0 Board

DE0 board has many features that allow the user to implement a wide range of design circuits. As the feature of general user interface, the board has green colour LEDs (Active high), 4 seven- segment displays (Active low) and 16*2 LCD interface. The lock input of the board is a 50MHz oscillator, so it had to be halved for our project. For VGA output, it uses a 4-bit resistor-network DAC with 15-pin high density D-sub connector. It supports up to 1280*1024 at 60Hz refresh rate [1].

C. About VGA

The images on the VGA screen are displayed by turning the pixels on and off. The common VGA display standard is 25MHz and we used this frequency for our project as well. The pixel rate is 640*480 resolution, and each pixel takes 40ns at 25MHz pixel rate

VGA video standard contains 5 active signals: Horizontal and vertical synchronization signals, three analog signals for red, green, and blue. By changing the analog voltage levels of the RGB signals, different colours can be produced and depending on the number of bits supported by the development board, different amounts of colours can be represented. [3]

In order to generate the VGA signal at 25MHz, the clock signal provided by DE0 (50MHz) needs to be halved, this was achieved by utilizing the 'altpll' mega function in Quartus.

III. SYSTEM DESIGN MODELLING

The two following figures show our intended implementation versus our actual implementation. Figure 1 shows our actual implementation which has the absence of a proper state control implementation and life counter function. Figure 2 shows our intended implementation with a fully integrated state control FSM and life counter component. Figure 3 shows our intended state control FSM.

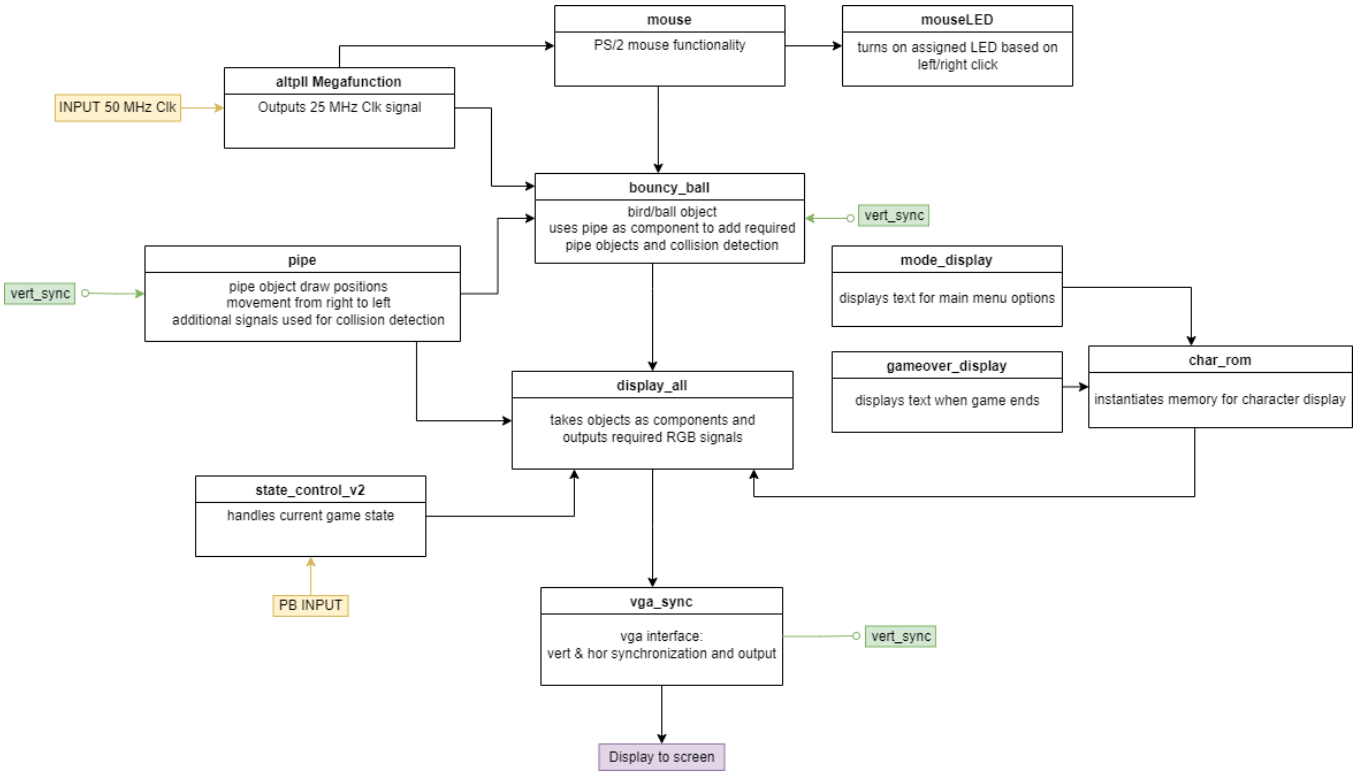

Figure 1

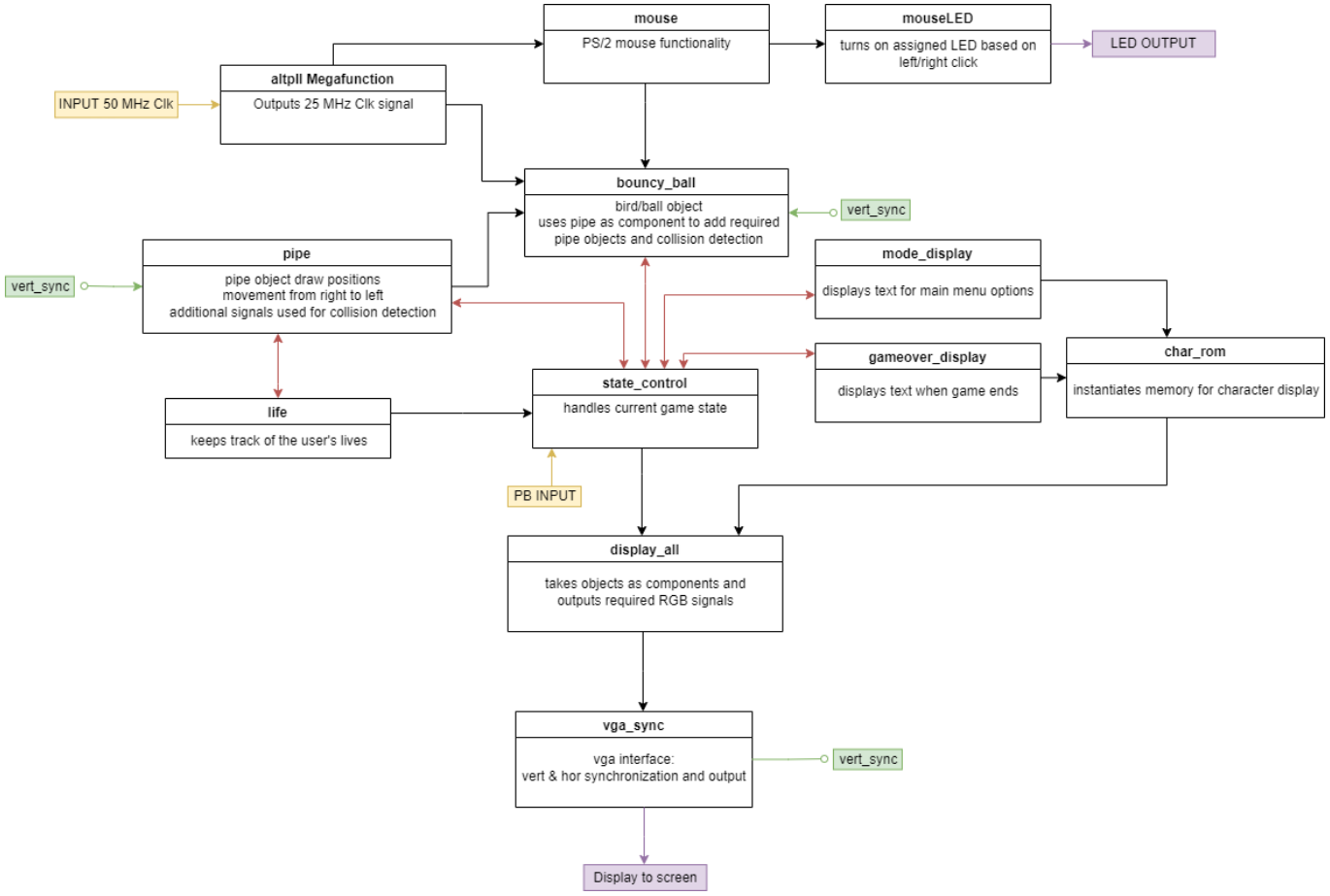

Figure 2

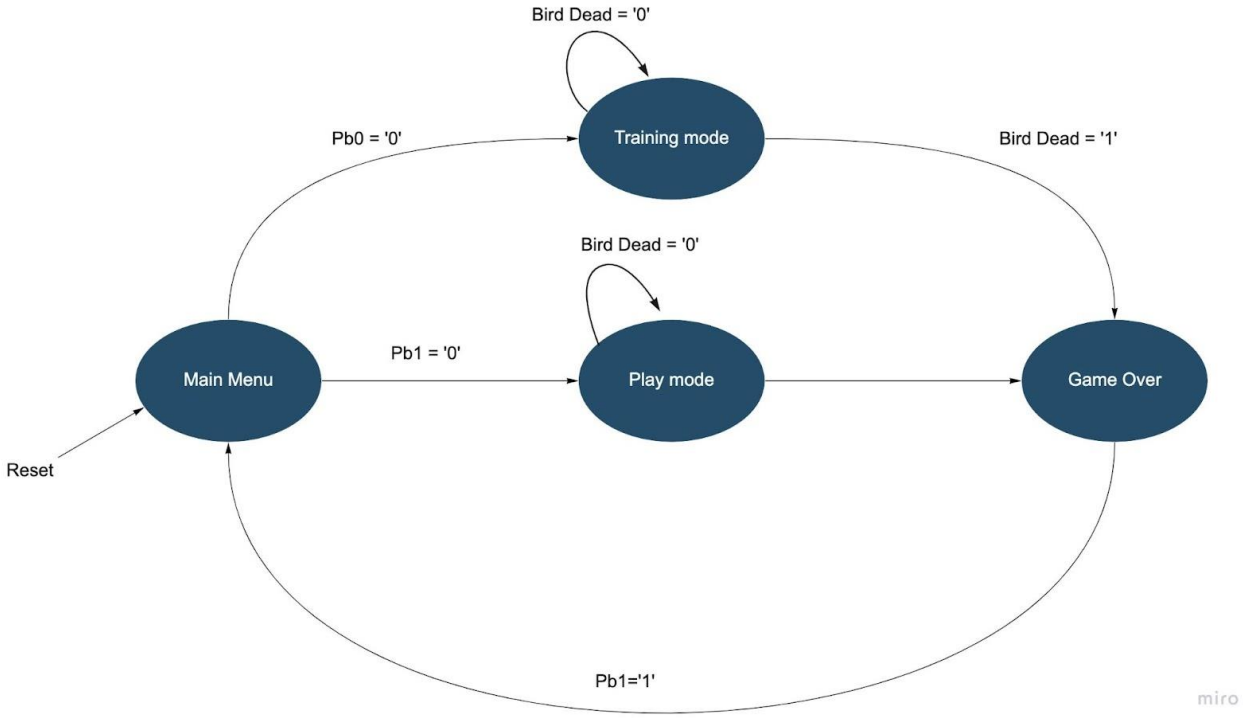

Figure 3

IV. DESIGN IMPLMENTATION

A. *Design Decision and Trade-offs*

For our design, we decided to create separate entities for the bird object, pipes, and text displays. This was to promote maintainability of code and make debugging easier.

To implement collision detection, there had to be some sort of connection between the bird and pipe objects so we used pipes as a component inside the bird object file (bouncy_ball.vhd) and port mapped all the relevant inputs and outputs.

In order to have all the required objects to be displayed, a display function (display_all.vhd) was created which essentially works as a GPU for our game. This function uses all objects as components (bird, pipes, menu text display, game over text display) and with appropriate port mapping and logic implementation the outputs are assigned to the red, green, blue outputs signals which are then connected to our VGA sync file (vga_sync.vhd).

B. *State Control*

Our intended implementation: Our game was to have a default main menu screen which displayed two options: training and play mode. When PB0 (push button 0) is used it was to allow the user to play the training mode and PB1 (push button 1) to play the normal game mode. In both of these modes, when the bird's life reached zero, it was to end the game and display the game over screen. From the game over screen, the user could push PB2 (push button 2) to go back to the main menu screen. This process is shown by Figure 3.

Unfortunately, we were not able to integrate all the components together and implement the state control.

C. *Pipe and bird movement*

The positioning and drawing of the bird and pipe objects are based on the bouncy_ball.vhd example provided to us in the project resources folder [5].

The bouncy_ball function acts as our 'flappy bird'. It takes a left mouse click signal as an input which is used to move the bird upwards (or 'fly') with each mouse button click. By default, the bird is falling downwards so the user needs to continuously click to keep the bird airborne to travel through the pipe gaps. In our design, 3 different pipe objects are drawn.

D. *Collision detection*

In order to detect collision between the bird and pipe objects, we extracted dimension values for the pipes (width, gap, positional values) and ported them into the bouncy_ball function. Logic statements were written and raised a 'collision flag' once a collision was detected. Once a collision was detected, the bird object was respawned at a set position.

E. *Life counter*

We intended to implement a life counter system to decrement from a set value for every collision that was detected. Unfortunately, we were not able to successfully implement this feature.

F. *Text Display*

A group of characters are stored in a memory block in the FPGA. This memory is instantiated in the char_rom.vhd file. The ROM contents are defined by the constant named "ROM" and it will be preloaded into the memory [2]. A mif file is used to initialize the memory and TCGROM.mif contains the pattern of 64 characters which works as the memory initialization file. Each character in a .mif file is described through 8 lines of memory address and is translated to a block of 8*8 pixels [3]. In our projects, we used pixel-row and pixel-column value as the address to the CharRom to define the size of the text.

V. IMPROVEMENTS

Our current version is incomplete and given more time we would have got it fully functional with some various features implemented. These include the following:

A. *Successful State Control System*

The relevant state control was designed and coded but was unable to be integrated into our game.

B. *Random gap placement in pipe*

Our current version only generates a fixed pipe gap value with each pipe object creation. Varying pipe gaps could be implemented using a random number generator.

C. *Level difficulty*

After a fixed amount of time is elapsed, the speed of pipes are increased in order to increase the game's difficulty.

VI. RESOURCE USAGE AND PERFORMANCE

Flow Summary	
Flow Status	Successful - Thu Jun 02 19:28:44 2022
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	FlappyBird_Mini_Project
Top-level Entity Name	FlappyBird_Mini_Project
Family	Cyclone III
Device	EP3C16F484C6
Timing Models	Final
Total logic elements	585 / 15,408 (4 %)
Total combinational functions	549 / 15,408 (4 %)
Dedicated logic registers	160 / 15,408 (1 %)
Total registers	160
Total pins	15 / 347 (4 %)
Total virtual pins	0
Total memory bits	8,192 / 516,096 (2 %)
Embedded Multiplier 9-bit elements	0 / 112 (0 %)
Total PLLs	1 / 4 (25 %)

Figure 4

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	129.13 MHz	129.13 MHz	inst1|altpll_component|auto_generated|pll1|clk[0]	
2	255.3 MHz	255.3 MHz	VGA_SYNC:inst3|vert_sync_out	
3	289.86 MHz	289.86 MHz	MOUSE:inst|MOUSE_CLK_FILTER	

Figure 5

Figure 4 shows the flow summary and provides information about total logic elements and resource usages. Figure 5 shows the Fmax values. Our altpll mega function had an Fmax value of 129.13 Mhz. Our VGA_SYNC component had Fmax value of 255.3 Mhz. And our mouse had the highest Fmax value of 289.86 Mhz.

REFERENCES

[1] Terasic Technologies, "Altera DE0 User Manual", 2009

[2] Lee, S., 2006. "Advanced digital logic design". Toronto (Ont.): Thomson.

[3] Maryam Hemmatai, "Mini project- VGA Interface and LFSR", 2022

[4] Maryam Hemmatai, "Mini project- Mouse Interface", 2022

[5] Maryam Hemmatai, "Mini-Project Resources Folder", 2022