CompSys 305 Mini Project Report

Lexin Lin
Department of EEE
University of Auckland
Auckland, NZ
llin819@aucklandun.ac.nz

Yiqi Zang
Department of EEE
University of Auckland
Auckland, NZ
yzan714@aucklanduni.ac.nz

Katherine Luo
Department of CSE
University of Auckland
Auckland, NZ
yluo985@aucklanduni.ac.nz

***Abstract* - This report details the final design and implementation for COMPSYS 305 Mini project. The purpose of this project was to design a game similar to Flappy Bird that is played using mouse, push buttons and switches on the DE0 board and displayed through a VGA monitor. The target of this game is to reach maximum scores by avoiding touch with the pipes and reaching upcoming gifts between the pipes to gain life points while the game progressively increases its speed. Unique features in our design such as background are changeable between daytime and nighttime using a specific switch on the DE0 board. In order to distinguish different states, a high-level finite state machine is implemented. Although all basic requirements are reached at the end, future potential improvements like increasing the maximum frequency and implementing more levels or gifts are doable if more time is given.**

Keywords - DE0 board, VGA display, VHDL, Finite state machine, Flappy Bird

Introduction

For this project, the purpose was to create a game similar to Flappy Bird. The game console used was the DE0 board. It is displayed on a VGA screen with a resolution of 640*480 pixels. In this game, the player controls a doge using a PS/2 mouse to let it move up and down. Contacts with obstacles like pipes will make the doge lose its life point. Gifts are available in the game which give the player more life points. The player's score is based on the amount of pipes that the doge passes through. The game ends when the number of lives goes down to zero.

Game Strategy

I. Rules and Features

To achieve the highest score in this game, the player needs to use the mouse to tap the screen to ensure the doge survives as long as possible. The score is based on the amount of pipes that the doge passes through. Hence, the strategy of this game is to make sure the doge passes as many pipes as possible.

The followings are the main features in our game:

- **Pipes**: They were placed randomly based on the number of clicks that the player has input. When a collision happens with the doge, the life points will decrease.
- **Gifts:** They were placed randomly between the upper pipe and the lower pipe and allowed the player to increase their life point when colliding with them.
- **Levels of difficulty**: In normal game mode, the speed will get faster after every 10 pipes. The maximum speed is limited when the player passes 40 pipes.
- **Scores:** The player's score is based on the amount of pipes that the doge passes through.
- **Background:** The background is changeable between daytime and nighttime using the sw9 on DE0 board.
- **Lives:** Lives allow the player to continue the game and it is shown on the 7-seg section on DE0 board.

II. Major Machine interfaces

Throughout the game, the player interacts with it using different interfaces . Players can control the game using these interfaces and be provided with feedback of their actions. The followings are the major machine interfaces:

- **Mouse**: Main controller of the game.In the menu state, the player can start the game by pressing the left mouse button. Once the game starts, the left mouse button is used to control the motion of the doge. When the game ends, the left mouse button is used to restart the game while clicking the right mouse button, it will take the player back to the menu section.
- **DIP switch**: Switch zero is used to select the two different modes. When sw0 is on, the mode

selection is training mode, while it is off, the mode changes to normal game mode. Switch 9 is used to select the two different backgrounds. When sw9 is on, the game will show the nighttime background, while it is off, it changes to daytime background.

- **Push button**: Once the game starts, the player can pause the game by pressing push button 1 and continue the game by pressing push button 0.
- **7-seg display**: Displays the player's current life points once the game has started.
- **VGA display**: Updates every clock cycle to display changes in the game.

III. Game modes

The game is required to have two operation modes. The descriptions of the two modes are followed below:

- **Training mode:** The purpose of this mode is to let the player practice as much as possible since it is set as the lowest level. The initial life point is set as 5, once it becomes zero, the game ends.
- **Normal Game mode:** In normal game mode, the player is expected to survive. Although the initial life point is also set as 5, the game will progressively increase its speed. Once the life point becomes zero, the game ends.

IV. How to Play

The player is suggested to start the game under training mode at first as he/she needs to get familiar with the game. The player controls the motion of the doge by clicking the left mouse button. Collision with pipes will reduce the current life points while collision with gifts will help the player gain life points. The initial life points for two modes are both set as 5. During the game, the player is allowed to continue the game when collision happens with pipes as long as his/her life point is more than zero. Once the life point becomes zero, the game ends. However, the player can still retry unlimited times by pressing the left mouse button when the game ends. After switching to normal game mode by switching off switch 9, the game will have three levels of difficulty. The speed will get faster after every 10 pipes. The maximum speed is limited when the player passes 40 pipes.

Finite-state machine

Our design has four states as shown in figure 1. This FSM is used to keep game behaviour updated. It will be used in the character display component to choose what text needs to be output to the display and will respond when external inputs(switch, button, mouse) or the output signals from other components(dead) are received. The FSM is also responsible for resetting the game when either in the menu state or in the dead state. It will send a signal to reset the game[1].

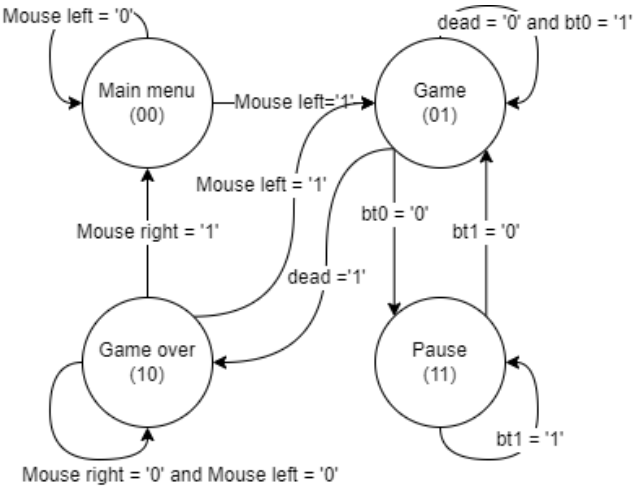

Fig. 1. RAM Summary

Implementation

As below in figure 2 shows our whole system design. Each component will be explained specifically with their purpose and usage later on in this section.

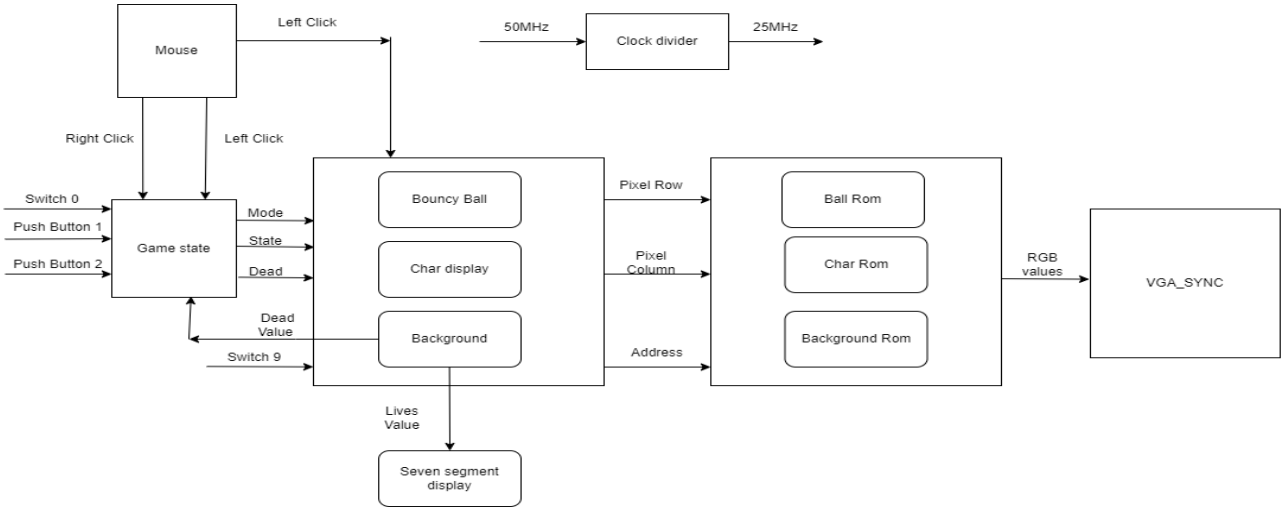

Fig. 2. Block diagram of our system

The clock divider component as shown in figure 3, is used to divide the 50KHz clock of De0 board to 25KHz in order for the VGA signal to be generated,

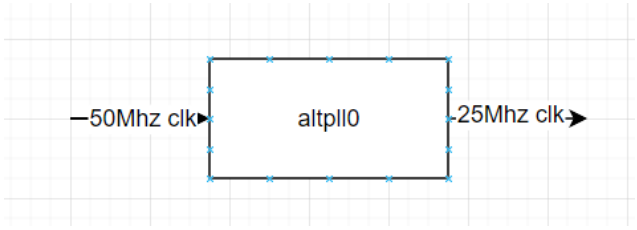

Fig. 3. Clock divider

The VGA_SYNC component as shown in figure 4, is used to display the 12 bits RGB(4 bits each) on a VGA screen every vertical synchronisation. The output sync contains horizontal and vertical sync, in which the vertical sync is used as a sensitive value of process for displaying our game. The output Pixel_pos(row and column) is used for controlling the region where we want to display.

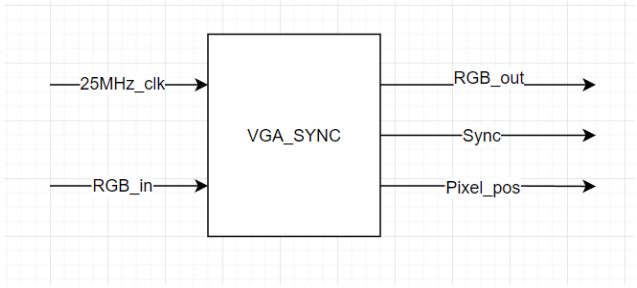

Fig. 4. VGA_SYNC

For char display, instead of defining the bit patterns pixel by pixel, we have used a mif file to store all the bit patterns that will be used at different memory addresses. The char_rom component as shown in figure 5, is used to access to the memory address and read and output the bit patterns being stored correspondingly. Due to the arrangement of addresses and bit patterns in our mif file, we can divide the 8 bit patterns into data_address and font_row and font_column. Data_address stores pattern and font_row and font_column controls the size.

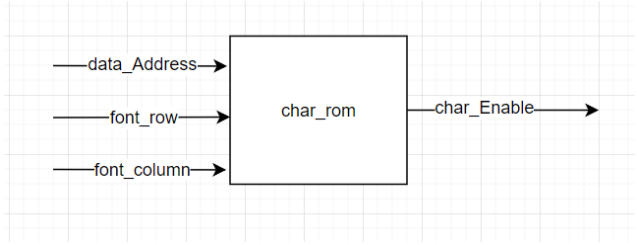

Fig. 5. Char display

With the same method being applied, we have created a background_rom as shown in figure 6. The input SW9 is the switch on the De0 board which is used to change between the daytime and nighttime background we have created. Instead of outputting a single bit, we output a 12 bits RGB value(4 bits each) for each pixel which forms a colourful, vivid background.

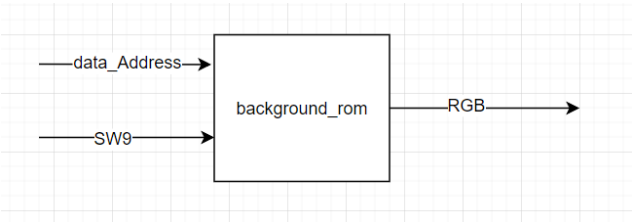

Fig. 6. Background_rom

The Mouse component as shown in figure 7, outputs mouse_Enable = '1' when Pixel_pos is within the range of the mouse cursor icon that we've created. And every vertical sync the mouse cursor will be displayed according to the mouse position. The left and right click are used for game controlling. Depending on different states of the game, they have different functionality.

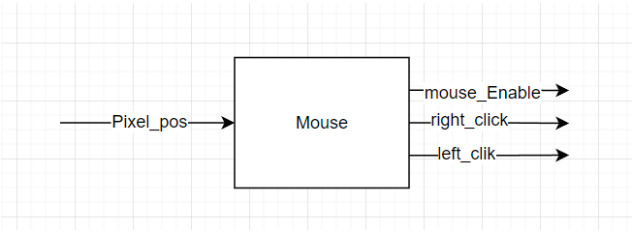

Fig. 7. Mouse

We have created a finite state machine component which is shown as figure 8. The detailed functionality of input and output are shown by figure 1.

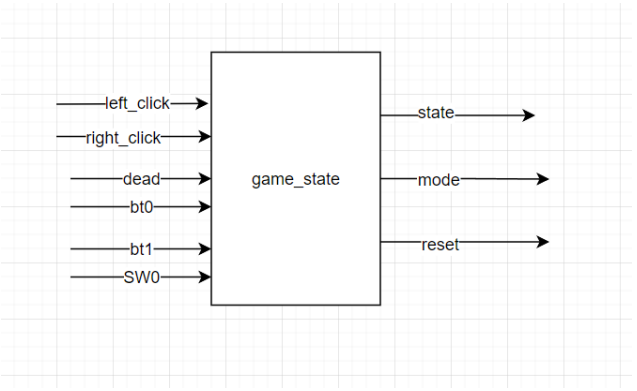

Fig. 8. Finite state machine

The bouncy_ball component is where we implement the function of our doge. With process sensitive to v_sync, if the mouse left clicked, the doge flies up. Otherwise it falls down. If the doge hits the pipe, dead = '1' then use left click and right click to choose whether to restart or back to the menu correspondingly. If the doge passes through a certain number of pipes without dying, the speed_up will input one, which will increase the moving of background and falling speed of the doge. The output click_n signal records how many times the player has left clicked. Depending on the number, the height of the gap of the pipe will change differently. The output ball_pos is the position of the doge to be displayed.

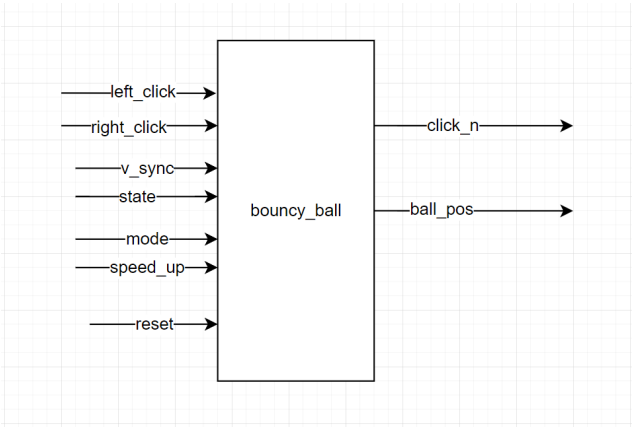

Fig. 9. Bouncy_ball

Finally, all the enable signals which indicate the position to display as mentioned above go into the background component as shown in figure 10. The main functionality of this component is to generate random height of pypes and setting colour for instance at different pixel positions that are being enabled. When the bird passes through a pipe, score_up value will be set to one which tells the char_rom component to increase the address by one. The lives-output is connected to a seven segment display to display remaining life. However, we have decided to use the seven segment display not only displaying the number of lives but also the words 'life' as well. We used a clock generator to count for 2 seconds in real time. And switch the display between 'life' and number of lives.

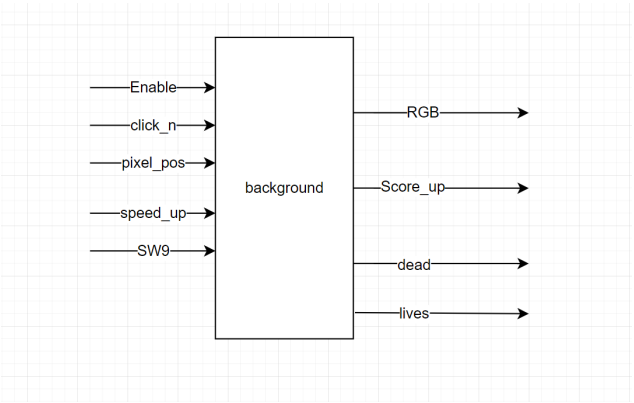

Fig. 10. Background

Design decision/trade-offs

At first, we didn't consider the M9K memory storage. We was trying to store the background with the same size of the monitor, which is 640*480. However, when compiling, the error shows that there is not enough space of memory to store the data. So we have divided the size of the photo by 8 to save some memory. Then we enlarge the background image back to its original size by inputting each memory address to 8 pixel position. In trade off, the quality of the background image is getting worse. And also, we've used arithmetic to achieve the enlargement of the image, which will cause delay and lower out frequency output. For our random height generator of pipe, we have also used arithmetic to achieve, which lowered our frequency.

Results

One ALTPLL used to supply all the components' clocks, because the standard for VGA display is 25 MegaHz pixel rate and have a 680 × 480 resolution.

Figure 11 is the flow summary for this project, which uses 17% of the total logic elements and 53/347 pins.

Flow Summary

Flow Status	Successful - Wed Jun 01 15:50:28 2022
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	VGA
Top-level Entity Name	flappydoge
Family	Cyclone III
Device	EP3C16F484C6
Timing Models	Final
Total logic elements	2,154 / 15,408 (14 %)
Total combinational functions	2,097 / 15,408 (14 %)
Dedicated logic registers	301 / 15,408 (2 %)
Total registers	301
Total pins	53 / 347 (15 %)
Total virtual pins	0
Total memory bits	189,568 / 516,096 (37 %)
Embedded Multiplier 9-bit elements	0 / 112 (0 %)
Total PLLs	1 / 4 (25 %)

Fig. 11. Flow Summary

There are 37% memory bits used, and most of them are used for storing the MIF files of the characters, backgrounds and doge images.

Analysis & Synthesis RAM Summary

	Name	Type	Mode	Port A Depth	Port A Width	Port B Depth	Port B Width	Size	MIF
1	background_...|ALTSYNCRAM	AUTO	ROM	5100	12	--	--	61200	background.mif
2	char_rom:in...|ALTSYNCRAM	AUTO	ROM	512	8	--	--	4096	tcgrom.mif
3	doge_rom:in...|ALTSYNCRAM	AUTO	ROM	256	12	--	--	3072	doge.mif
4	eve_backgro...|ALTSYNCRAM	AUTO	ROM	5100	12	--	--	61200	eve_background.mif
5	title_rom:in...d|ALTSYNCRAM	AUTO	ROM	5000	12	--	--	60000	title.mif

Fig. 12. RAM Summary

Figure 13 showed the maximum frequency for our design is 34.98 Mhz although it meet the design requirement which is for 25 Mhz for all the components but since the vga sync frequency is also 25Mhz, therefore in our design we may not have enough time to transfer data between components which may lead to wrong information or bugs.

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	34.98 MHz	34.98 MHz	VGA_SYNC:inst8|vert_sync_out	
2	82.66 MHz	82.66 MHz	inst2|altpll_component|auto_generated|pll1|clk[0]	
3	231.96 MHz	231.96 MHz	MOUSE:inst4|MOUSE_CLK_FILTER	
4	243.9 MHz	243.9 MHz	background:inst|pipe2_x_pos[0]	
5	243.9 MHz	243.9 MHz	background:inst|pipe3_x_pos[0]	
6	243.9 MHz	243.9 MHz	background:inst|pipe_x_pos[0]	
7	1466.28 MHz	500.0 MHz	clock_generator:inst16|b	limit due to minimum period restriction (tmin)

Fig. 13. Model Fmax Summary

Therefore, in order to deliver a better design, future improvements are needed.

Conclusion and future improvements

In conclusion, the project was to create a game similar to flappy bird code in VHDL. The game is displayed through

a VGA monitor with 640*480 resolution and is controlled by a PS/2 mouse and a DE0 board. There are two modes that are training mode and normal game mode. The game is required to be retryable and randomly generates objects in every run.
For future improvements, there are more features that can be added to the game itself.

For example, unlockable cosmetics (probably different outfits for the doge) granted by getting certain high scores, or just earning a currency as you played. Once the requirement meets, the player can change the cosmetics by the DIP switches(similar to the way to change background).

Increase difficulty by adding energy to the game. When the player clicks, energy is consumed. The game starts with a certain amount of energy(depends on mode) and can gain more energy from gifts or stop clicking for a small period. If the player runs out of energy, the doge will not jump.

The number we used to calculate the height for the pipe is currently use a counter to count the number of mouse been clicked, however in the future we can use a LFSR component to generates a random number, and shift the output to set the height of the pipe with in the range and generates the position of the gift(instead of generate the gift between the gap of the pipe)[2].

And at the current stage we do have a moving cursor icon that will change location with the PS/2 mouse movement. However for future development we can link it to the doge to make it move across x direction.

The most important improvement we need are the codes. Most of the components have more than one process which will create delay for data transfer and especially the max frequency we have is not ideal. Thus, in order to improve the performance of our design. A higher operating frequency is needed and processes within a component could be separated. Also instead of setting up and calculating text coordination by hand, we can create a new component to connect the char rom location and the location we want the text be, and put it to the vga sync component.

Acknowledgment

The completion of this project could not have been possible without the expertise of Dr. Morteza Biglari-Abhari and Dr. Maryam Hemmati, our beloved theory advisers. We would also like to thank Ross Porter, Lucy Wu and David Cole for sitting on our panel and taking the time to help us to improve our design. A debt of gratitude is also owed to Max McDonnell for inspiring us with his flappy bird video game. Last but not least, we would like to thank our teammates and keep up all the excellent work contributed to this project.

References

[1] M. Hemmati, "FSM," iam.auckland.ac.nz. https://canvas.auckland.ac.nz/courses/72184/files/8582352?module_item_id=1480491 (accessed Jun. 01, 2022).

[2] M. Hemmati, "FPGAArchitecture," iam.auckland.ac.nz. https://canvas.auckland.ac.nz/courses/72184/files/8751603?module_item_id=1494622 (accessed Jun. 01, 2022).