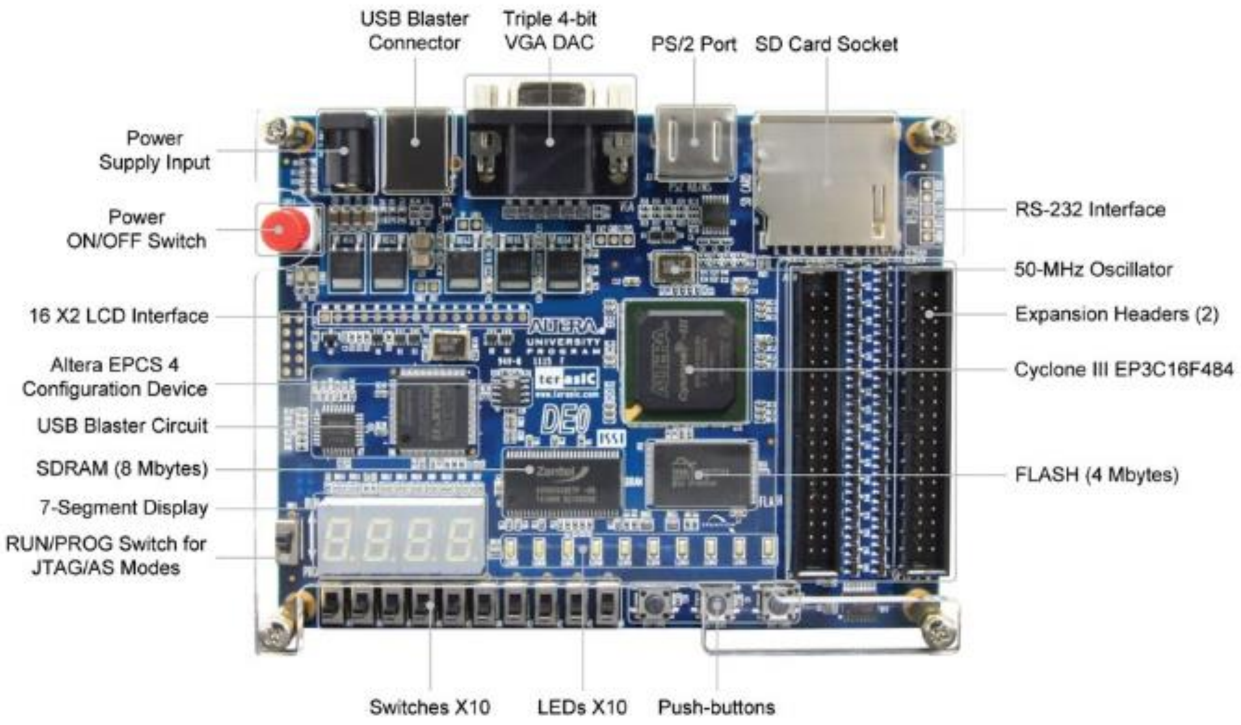
Group 26

Flappy Bird

305 Mini-Project



Sophia Ritchie Nipoon Patel

1.0 Abstract

This report details the Mini Project undertaken by Group 26 as part of the COMPSYS 305 course at The University of Auckland, 2016. A Flappy Bird game was developed by the two members of the group using a programmable DE0 board, mouse and connecting it directly to a monitor using a VGA cable, meaning the entire game uses hardware components. The finished game met all of the brief requirements and included extra features and graphics for player satisfaction and enjoyment.

The results of the final game show that 1955 logic elements were used in the implementation and the minimum operating frequency is 114.53 MHz. The total power dissipation is 76.85mW.

2.0 Introduction

This project required the development of the game Flappy Bird using only hardware components. The game console used was a DE0 board and the user input available was a mouse as well as the switches and buttons on the game console. The game requirements outlined in the project brief were as follows:

- Two playing modes are to be implemented, game mode and training mode
- Game mode will have levels that increment in difficulty
- An interface will be designed to select the mode when the game is loaded
- The user will control a bird that can move up and down with the mouse click
- Pipes will scroll from right to left as obstacles for the bird to maneuver through
- Life points are depleted when collisions occur
- The game ends when the bird drops out of the screen or all life points are depleted

The group contributions consisted of a VHDL file with two main processes. The display process chooses which interface to display, accesses the letters from the supplied char-rom file for the game titles and displays all the objects in the game. The move process handles the physics of the game, including moving all the objects, detecting collisions, and changing the game state.

3.0 Game Strategy

3.1 Rules and features

Our game fulfilled all the minimum specifications, as well as implementing some extra features to maximise user experience. The user interfaces for our game consisted of the mouse left click, switch SW1 and buttons PB1 and PB2 on the DE0 board. These are used for different purposes as explained below.

3.1.1 Game description overview

To play the game, the user will connect the VGA cable to a display, and the mouse into the DE0 board. When the game is loaded, a home screen will show, with the two modes to choose from. Using the dip switch, the user selects the mode and uses the first button across from the dipswitch to begin the game. The user is presented with a frozen screen until they click the left click of the mouse, which starts the pipes moving in a right to left direction across the screen. Each click bounces the user-controlled 'bird' up, otherwise it falls to the bottom of the screen. If the bird goes beyond the bottom of the screen, the game is over. The user has 3 life points to begin with that are displayed at the bottom of the screen. If the bird collides with the pipes, the life points are depleted by one and the bird is moved to the beginning position and frozen until the user clicks the left click of the mouse to start the game moving again. If the bird collides with the pipes 4 times, then the game will be over, unless life points are topped up by a powerup. Powerups randomly appear the user has to collide with them to use them. The score of the game increases the longer the user stays alive. The game can be paused using the second button along from the dipswitch and unpaused by pressing the button again and clicking the left mouse click. Once the game is over, the final score is presented in a game over screen, and the third button along from the dip switch returns the user to the main menu.

3.1.2 Game objects physics

The user controlled bird is acted upon by gravity unless a left mouse click accelerates it upwards. The goal was to create a user intuitive game experience. We found that if the bird accelerated too quickly then the game became very difficult to play, but a constant downwards speed was too easy. The implementation we decided on has the bird moving at a constant speed downwards of 1 pixel per refresh for 5 pixels and then increasing the constant speed by one pixel per refresh. When the left mouse click is pressed the bird moves up 15 pixels per refresh for 3 cycles. It was important not to jump straight up 45 pixels as this would mean that the bird would not collide with power ups and jump straight over them. The bird is fixed at 90 pixels from the left hand side of the screen so the user can look ahead and see the gaps in the pipes and have sufficient time to react and prepare.

In Game Mode, there are levels that get harder as the user progresses. The pipes and power ups move at a constant 1 pixel per refresh from right to left in Training mode and level 1 in Game mode. Each level corresponds to the speed of the pipes moving from right to left e.g level 3 has pipes and powerups moving at 3 pixels per refresh. The fastest speed is 5 pixels per refresh and we found that any faster than that and the game became near impossible to play.

3.1.3 Score and Levels

Levels are determined by the score the player has achieved during the game. Every 20 points that are scored by the user, the level increases and the speed of the pipes and power ups increases by 1 pixel per refresh. Every 20 points scored increases the level of the game, and the highest level is level 5, achieved when the user reaches 80 points. The score of the game is increased by one when the bird passes the rightmost side of a pipe.

3.1.4 Power up types

Our game featured four types of power up outlined below.

- Restore one life point
- Restore all life points
- Slow down game speed to level 1 for a time
- Speed up game speed to level 4 and disable collisions for a time

The timed power ups use the score to determine when to return to normal gameplay. When a power up is consumed, the score 10 score points above the current score is calculated. When the player reaches that score, the power up ceases.

3.1.5 Game Graphics

We implemented engaging background, bird and pipe graphics to enhance user experience. We also used the supplied char_rom and edited the .MIF file in order to use heart shapes for the life points and the letters for all the text. We also wanted to change the display in some way when the bird collided with the pipes, so a flash occurs that inverts the colours for a short time and reinforces to the user that there has been a collision.

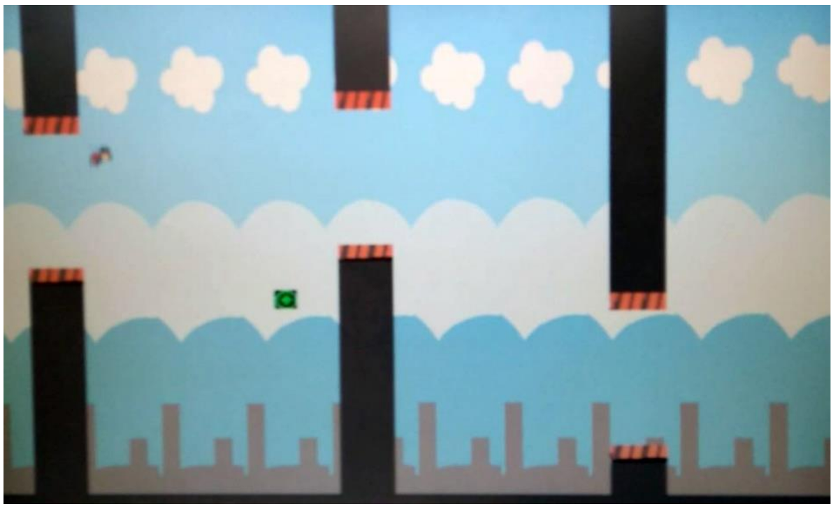


Figure 1 : Preview of our submitted game

4.0 Implementation

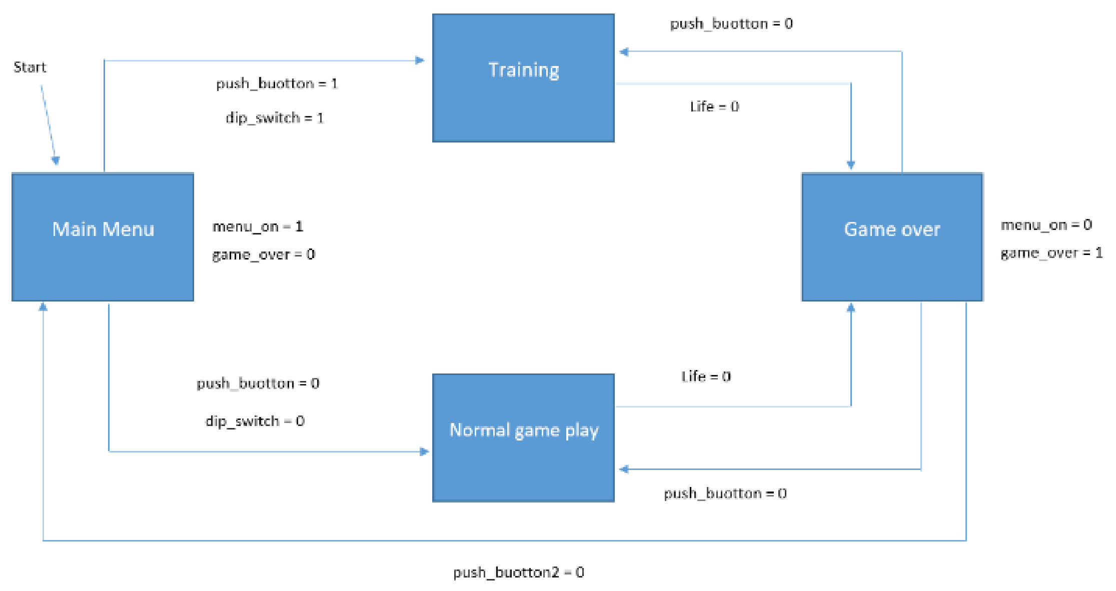


Figure 2 : FSM of our system

This FSM controls which state the game is currently in. The game initially starts up in main menu state. Two buttons (BUTTON1 and BUTTON2) and switch (SW0) of the DE0 board will trigger change in states. Note here the push buttons are active low (0) when pushed. The signals used are menu-on and game_over.

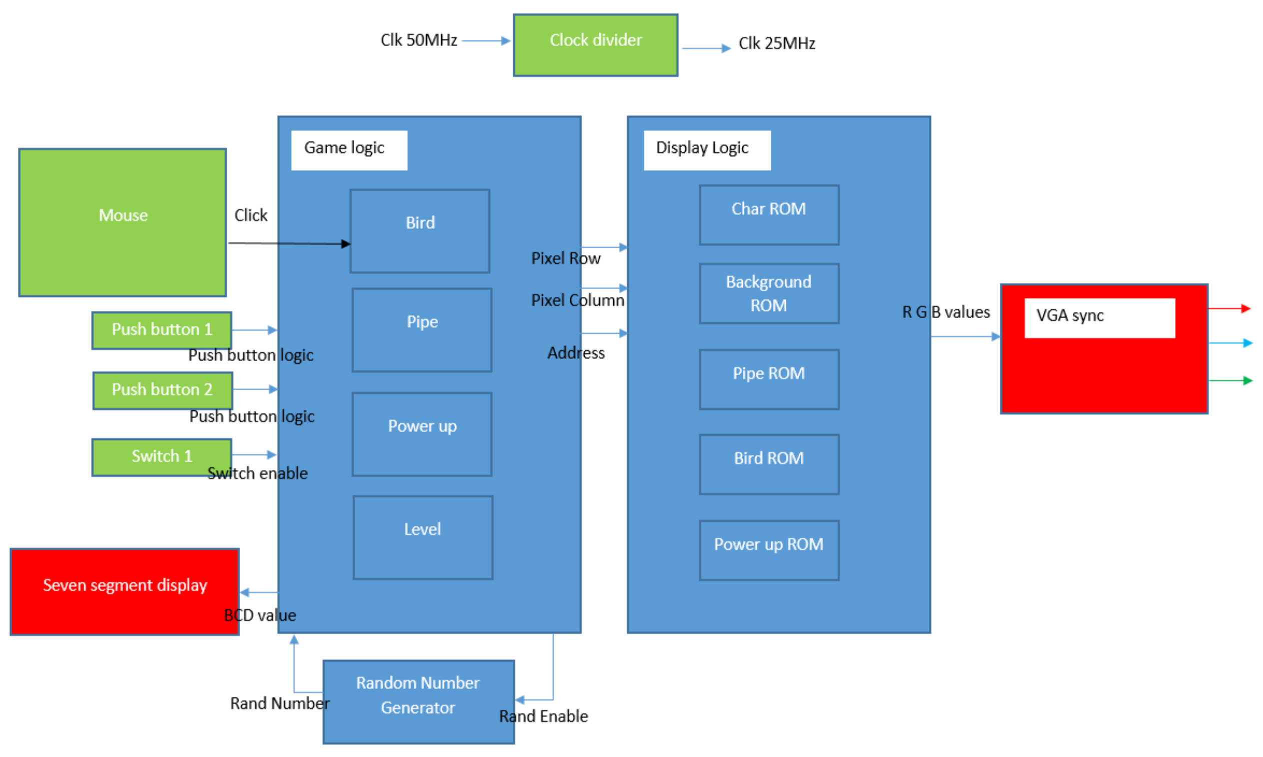


Figure 3 : Block diagram of our system

4.1 Clock Divider Block

Clock divider divides the incoming clock from 50MHz to 25MHz.

4.2 Mouse Block

Controls the position, left and right clicks of the mouse used in the design. The control signals will then be passed onto the game logic block where the mouse events are processed and used to move the bird.

4.3 Random Number Generator Block

This is used to design the randomly generated gaps in the pipes. We used an 8_bit LSFR with three taps to generate random numbers that will come within the screen height range. The random number generator is also used to generate the random power ups which appear on the screen and to randomly determine the type of power up generated.

4.4 Game Logic Block

The game logic block processes the movements of all the components in the system (birds, pipes and power ups). The collision detection is also done in this component alongside the FSM to display welcome screen, pause screen and game over screen as necessary. All this information is sent over to the display logic block as required to draw each pixel on the screen.

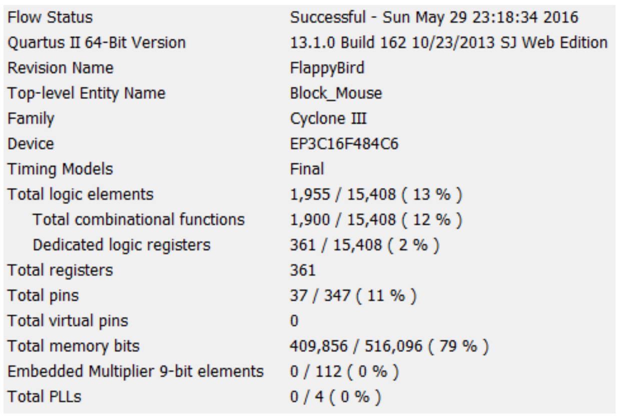
4.5 Display Logic Block

Display Logic Block is responsible for displaying each of the objects on the screen. It will read from the Char ROM, Backgroud ROM, Pipe ROM and Bird ROM and process the data to give each pixel a RGB value. This value is passed on to the VGA Sync Block where it is displayed.

4.6 VGA Sync Block

The value passed on by the Display Logic Block is used to colour in each pixel on the screen.

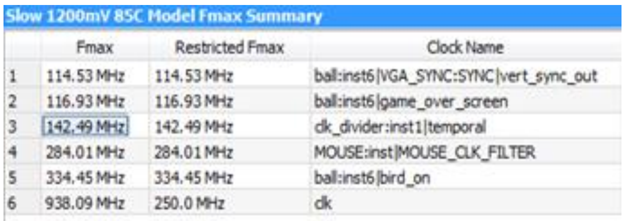
5.0 Result



Flow Status	Successful - Sun May 29 23:18:34 2016
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Web Edition
Revision Name	FlappyBird
Top-level Entity Name	Block_Mouse
Family	Cyclone III
Device	EP3C16F484C6
Timing Models	Final
Total logic elements	1,955 / 15,408 (13 %)
Total combinational functions	1,900 / 15,408 (12 %)
Dedicated logic registers	361 / 15,408 (2 %)
Total registers	361
Total pins	37 / 347 (11 %)
Total virtual pins	0
Total memory bits	409,856 / 516,096 (79 %)
Embedded Multiplier 9-bit elements	0 / 112 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 4 : Figure to show the number of logic elements

Our project used 1955 'logic elements'. This is slightly higher compared to other groups. This is due to our implementation of extra features such as powerups and graphical image background (using the ROM). The background image stored in the ROM using a .MIF also explains why there is a high use of the 'total memory bits' and M9K memory blocks.



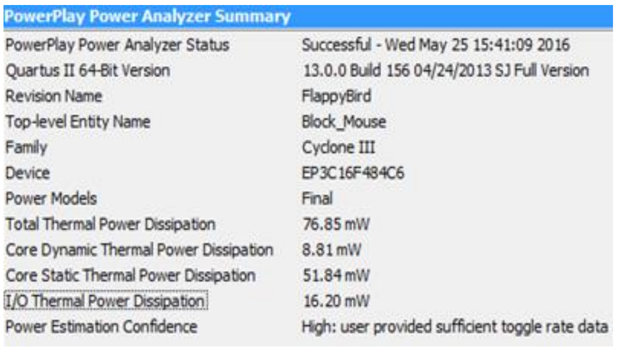
Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name
1	114.53 MHz	114.53 MHz	ball:inst6|VGA_SYNC:SYNC|vert_sync_out
2	116.93 MHz	116.93 MHz	ball:inst6|game_over_screen
3	142.49 MHz	142.49 MHz	clk_divider:inst1|temporal
4	284.01 MHz	284.01 MHz	MOUSE:inst|MOUSE_CLK_FILTER
5	334.45 MHz	334.45 MHz	ball:inst6|bird_on
6	938.09 MHz	250.0 MHz	clk

Figure 5 : Table showing the frequencies of different processes

In the table above the maximum speed of our circuit can be determined by looking at the minimum operating frequency. Vert_sync_out has the minimum operating frequency Fmax (114.53 MHz). This is also our critical path in the circuit. While Fmax is a function of the longest propagation delay between two registers in the circuit, it does not indicate the delays with which output signals appear at the pins of the chip.

The table below (at the bottom of the page) shows the worst case timing path for our slowest path, in our case it is vert_sync_out.



PowerPlay Power Analyzer Summary

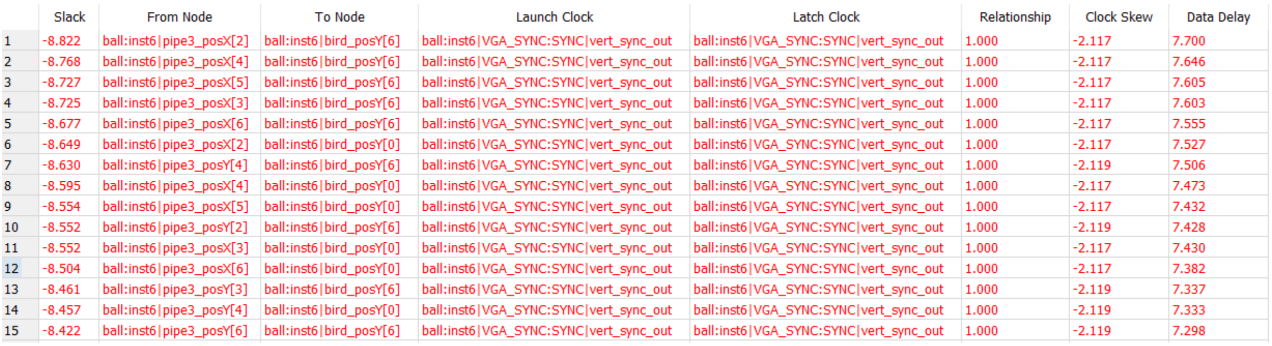
PowerPlay Power Analyzer Status	Successful - Wed May 25 15:41:09 2016
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Full Version
Revision Name	FlappyBird
Top-level Entity Name	Block_Mouse
Family	Cyclone III
Device	EP3C16F484C6
Power Models	Final
Total Thermal Power Dissipation	76.85 mW
Core Dynamic Thermal Power Dissipation	8.81 mW
Core Static Thermal Power Dissipation	51.84 mW
I/O Thermal Power Dissipation	16.20 mW
Power Estimation Confidence	High: user provided sufficient toggle rate data

Figure 7 : Figure to show the total power consumption

From the table of figures above you can see that our total power dissipation is 76.85mW. We are more interested in looking at dynamic power dissipation as it is the power dissipated when active current flows and switching takes place in the transistor. e.g For a signal going from low to high, 8.81mW of power is dissipated.

6.0 Conclusion and Discussion

In conclusion we have done our very best to deliver an enjoyable game for the players and have implemented several features which were beyond the project specifications. If we had time to improve our design, we would focus on making the design more efficient as our design uses almost 2,000 logic elements and over 400,000 memory bits. We would try to reduce that by avoiding latches in our code and removing unused signals. Another area that would add value to the game would be to have even more detailed graphics. In particular, making the bird image animated when flapping upwards.



	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-8.822	ball:inst6|pipe3_posX[2]	ball:inst6|bird_posY[6]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.117	7.700
2	-8.768	ball:inst6|pipe3_posX[4]	ball:inst6|bird_posY[6]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.117	7.646
3	-8.727	ball:inst6|pipe3_posX[5]	ball:inst6|bird_posY[6]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.117	7.605
4	-8.725	ball:inst6|pipe3_posX[3]	ball:inst6|bird_posY[6]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.117	7.603
5	-8.677	ball:inst6|pipe3_posX[6]	ball:inst6|bird_posY[6]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.117	7.555
6	-8.649	ball:inst6|pipe3_posX[2]	ball:inst6|bird_posY[0]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.117	7.527
7	-8.630	ball:inst6|pipe3_posY[4]	ball:inst6|bird_posY[6]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.119	7.506
8	-8.595	ball:inst6|pipe3_posX[4]	ball:inst6|bird_posY[0]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.117	7.473
9	-8.554	ball:inst6|pipe3_posX[5]	ball:inst6|bird_posY[0]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.117	7.432
10	-8.552	ball:inst6|pipe3_posY[2]	ball:inst6|bird_posY[6]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.119	7.428
11	-8.552	ball:inst6|pipe3_posX[3]	ball:inst6|bird_posY[0]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.117	7.430
12	-8.504	ball:inst6|pipe3_posX[6]	ball:inst6|bird_posY[0]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.117	7.382
13	-8.461	ball:inst6|pipe3_posY[3]	ball:inst6|bird_posY[6]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.119	7.337
14	-8.457	ball:inst6|pipe3_posY[4]	ball:inst6|bird_posY[0]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.119	7.333
15	-8.422	ball:inst6|pipe3_posY[6]	ball:inst6|bird_posY[6]	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	ball:inst6|VGA_SYNC:SYNC|vert_sync_out	1.000	-2.119	7.298

Figure 6 : Table to show the delay in the system