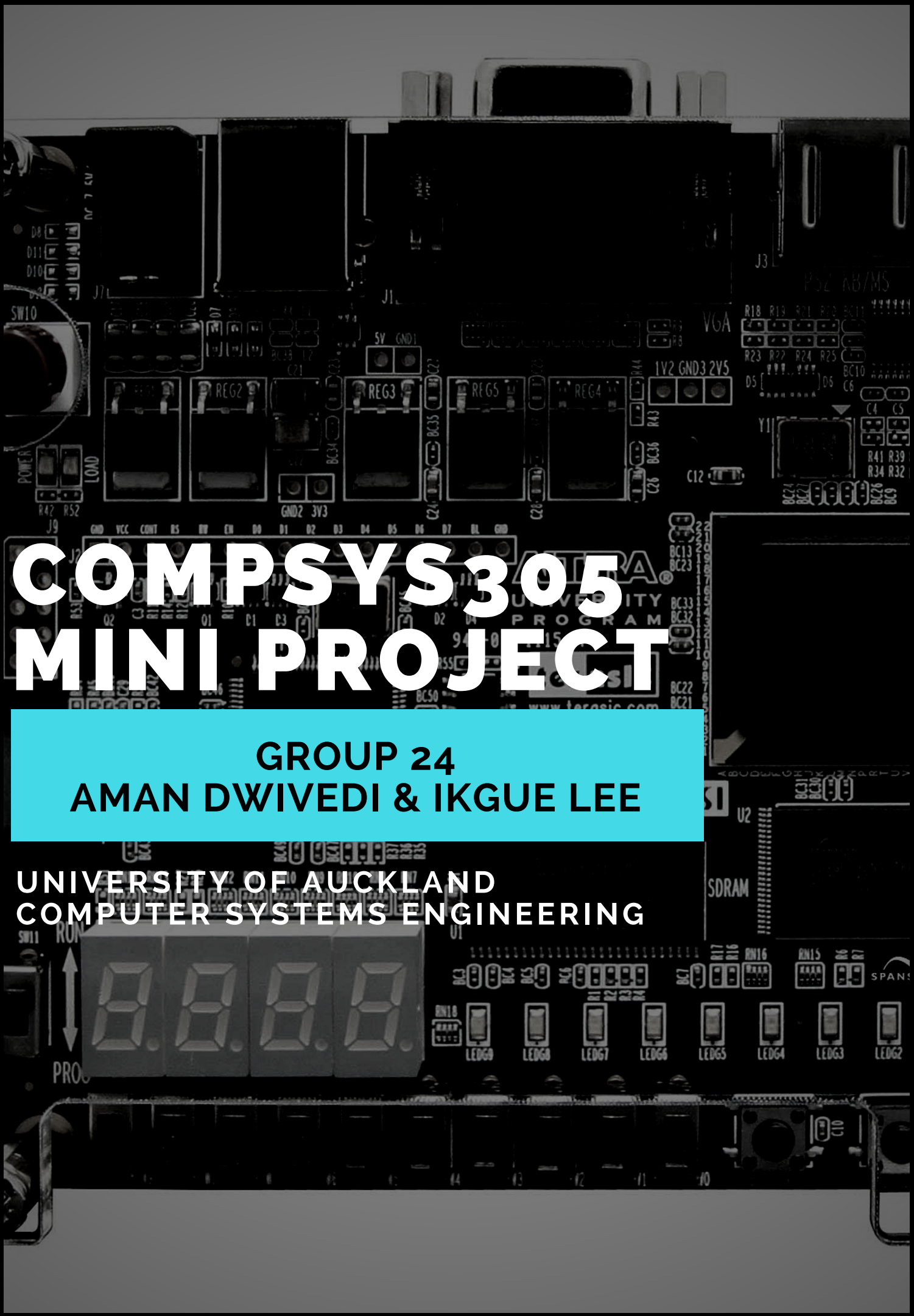


COMPSYS305 MINI PROJECT

GROUP 24
AMAN DWIVEDI & IKGUE LEE

UNIVERSITY OF AUCKLAND
COMPUTER SYSTEMS ENGINEERING

COMPSYS 305 Miniproject Report

Aman Dwivedi and Ikgue Lee
The University of Auckland, adwi888@aucklanduni.ac.nz, ilee471@aucklanduni.ac.nz

***Abstract* - We were assigned to make a game to be played on a VGA screen with a PS/2 mouse using the VHDL programming language on the Altera DE0 board. The game was that of a similar style to the 70s classic game Pong. Minor changes to game features were made to the game to emphasize originality and enjoyment by introducing new elements to intrigue players. We were provided with many helpful resources related to VHDL and code to get us started. This report contains an in-depth analysis of the user interface, game specifications, game strategy and the implementation of the final design with visual representation for the system. Furthermore, it also includes discussion of physical efficiency, the issues faced during development and possible solutions with plans for future improvements as game developers.**

INTRODUCTION

The COMPSYS305 Mini Project was to create a game which integrates some key elements from the 70s classic game Pong onto the Altera DE0 board using the VHDL programming language. The game features balls which spawn at the top of the screen and move down at various angles, bouncing off all four sides. There is also a platform at the bottom of the screen which can be controlled with the PS/2 mouse by the player. It can only move left and right. The aim is to move the platform to catch these balls. When a ball is caught, a point will be rewarded. Another ball will then spawn at a random place along the top of the screen to move down at a random angle. The game offers different levels which introduce increasing difficulty. The score and time left will be displayed on the screen as well as the level number. The total implemented specifications include the score system, timer, the ball physics, the platform mechanics linked to the mouse movement, text display and the main FSM. The final block diagram of the system will be included later to further explain the integrated elements.

GAME STRATEGY

The game introduces two different game modes for the player to choose. The game mode is the main feature - it features different levels in which the difficulties increase depending on the score reached, up to a maximum of 4. There is a time limit to every level - if the score threshold is not met within the time limit, the game will end. The game will also end when the final score objective of 60 is reached.

The practice mode is specifically made for the player to familiarize themselves with the game style. Therefore, it doesn't have a time limit; this would put less pressure on the user to meet the score requirements and give them time to get used to the game mechanics. The player is allowed to play until the score is 20, in which case the game will end and return to the main mode select screen.

The balls spawn along the top of the screen with a fixed vertical coordinate and randomly generated horizontal coordinates. The angle at which the ball moves downwards is also randomly generated as required in the specifications to introduce diversity.

While playing the game, the user can check the current level, acquired score and time left on the top right corner of the screen. As the player progresses through the levels, the level number will be incremented for every stage while the total score is kept consistent and the timer is reset. When the game ends via running out of time or reaching the end, the final score will be shown with instructions to return to the main mode select screen. Restating the obtained score on the end screen motivates the player to keep playing the game to achieve a better result.

Our game proposes points of appeal in both functionality and specifications which make it intriguing for the player. The original Pong game is a two-player game, where each player controls a platform to hit the ball back to the opponent. Because our game is single player, new features were implemented to make the game interesting and add originality. Hence, the levels feature an increase in ball number, speed and a decrease in platform size. To integrate a defined system of "easy to play, hard to master" mechanism, the early levels are very easy to get into while the final level features very fast balls and a very small platform. Because of the high required skill cap, the final objective score is capped at 60.

When the game is first opened, the title "Catch Me If You Can" with the selectable modes are shown. The user can use the switch sw0 to change between normal/practice modes and button pb0 on the Altera DE0 board to start the game. A pair of arrows will indicate which mode is chosen.

During gameplay, the player can use the mouse to move the platform left and right and use button pb1 to pause the game. An asynchronous reset on button pb2 is also available to return to the mode select menu at any time. When the game ends, pb2 can be pressed to return to the main screen.

FINAL BLOCK DIAGRAM

The final design's block diagram, as seen in Figure 1 of the appendices, ended up being slightly different from the planned diagram. The design of the planned diagram did not take into account the linear feedback shift-register (LFSR) and the collision handler. The addition of these components

gave us a total of 11 main components, not including components that were repeated.

The Collision Handler component takes in the position and sizes of the paddle and each ball. These are used to determine whether or not each ball has currently collided with the paddle. If a ball has collided with the paddle, a signal representing the collision for that ball is passed into the Game Logic component.

The Game Logic component handles all logic relating to the game. The component takes in several different inputs which enable it to control when things happen with the game. The component takes in all the enable signals from the fsm to ensure that the rest of the game responds appropriately, for example, when the game is in the pause state, the balls should not display and should not move, or when the game is in the training state, the timer should not tick and the level should not increase. Another important input the component uses is the ball collision status. This allows the Game Logic component to reset the position of a ball based on it's collision status and increase the score on collision. The component also handles incrementing the game level based on score and game mode, changing each ball's vertical velocity based on the game level, changing the paddle size based on the game level and checking if the game is over. The component outputs the current game time, the current score, a divided clock, the enable and reset signals for each ball, the paddle enable signal and the paddle size.

The Control FSM component is a finite-state machine that handles the various states and state transitions of the game. It takes in a clock signal, a signal representing the game over status of the game and the various switches and buttons. The switches and buttons and the game over status signal allow the machine to change between states. Based on the current state, the state machine will output a signal representing that state.

The Ball component handles the position of the ball, the motion of the ball and the drawing of the ball. The ball takes in an initial x position from an LFSR. When the ball is reset, it's new x position will be the initial x position. A few bits are taken from the initial x position in order to determine the horizontal speed and the direction of the ball. The vertical velocity of the ball is passed in as an input which is determined by the Game Logic component. Upon hitting the edges of the screen, the ball will rebound. The ball outputs it's current position to the collision handler and the RGB value of the ball at the current pixel location.

The LFSR component is a Linear Feedback Shift Register which takes in a seed (an initial value) and generates a pseudo-random value consisting of ten bits when the input enable signal is high. This value is then outputted by the LFSR component.

The Game UI component takes in the enable signals provided by the Control FSM component and generates an output UI based on these signals. The text on this UI comes from a character rom, an internal component of the Game UI component. The Game UI component outputs and RGB value representing the RGB of the component at the current pixel location.

The Ball Display component takes in the RGB value of every ball for the current pixel location and outputs a new RGB value based on the inputs.

The Display Logic component takes the RGB values from the Ball Display component, the Paddle component and the Game UI component and outputs an RGB value based on these inputs for the current pixel location to the VGA_SYNC component.

The VGA_SYNC component generates the signals required to display information on the monitor via the VGA interface. The current pixel row and column are passed from the VGA_SYNC component to many other components in order to ensure that different objects display in the correct locations with the correct colours.

The Paddle component takes input from the mouse and decides the position of the paddle based on this input. The component draws the paddle and passes the RGB values of the paddle to the Display Logic component. The ball also outputs its current position to the collision handler.

The Mouse component handles signals being passed in by the physical mouse. The mouse column location is passed onto the paddle.

FINITE-STATE MACHINE

A Mealy FSM was deemed suitable for our design because of the dependence of current inputs. The FSM can be seen in Figure 2 of the appendices. From the initial Menu state when menu_en is 1 with the game title and mode select, there are arrows pointing to Game and Training depending on whether the DIP switch sw0 is on or off. Then, as the buttons are active low, when pb0 is pressed, the machine will move onto the selected state by turning on either game_en or training_en. During either Game or Training, pressing pb1 will make pause_en 1. This moves the game to the Pause state. It can be pressed again to return to the respective previous states depending on prevState. In the Game state, running out of time or reaching a score of 60 will turn on game_over. Similarly, reaching a score of 20 in the Training state will turn on game_over. When game_over is 1, the game will proceed to the Over state where 'THE END' is shown to represent that the game has finished with a prompt to press pb2 to return to the menu. The asynchronous reset button pb2 can be pressed in any state to return to the welcome screen for testing purposes and player convenience.

RESULTS

In terms of efficiency, 1473 logic elements were used in the design. This could have been optimized further by further reducing the number of loops and if statements that were employed and ultimately using less statements which would require logic elements. The flow summary containing the number of logic elements can be seen in Figure 3 of the appendices.

The maximum operating frequency 109.37Mhz - This restriction can be minimized by shortening the critical path

which is from the game_logic node to ball_velocity node. Further information on operating frequencies and nodes can be found in the appendices.

One of the biggest obstacles faced during development was text. Our initial plan to display text made it really difficult to implement different font sizes. Hence, a new solution was formed by specifying particular boundaries for each letter depending on the size needed. Although this required us to restrict the beginning pixel of every letter at a multiple of the font size number, this way resulted in less code and consequently a more efficient design overall.

Another problem was encountered while completing the pause functionality. The FSM would continuously switch between the pause state and game/training state when pb1 was pressed. This caused the VGA screen to flicker between the two screens. We added a debouncer to fix this issue.

Our design contained many signals and variables which were of type integer. There was also a lot of code which were not optimized there was a large number of unnecessary logic elements used. This lead to a lot of unnecessary logic elements used; we spent a lot of time going through the code to try and make it as concise as possible, and we were able to achieve our final result of 1473 logic elements.

CONCLUSION

The brief for this project was to create a game console using an FPGA with a built-in game. The requirements for the game were that two or more balls need to move on a screen and the player needs to catch a ball in order to increase their score. The console needed to operate at a resolution of 640x480 via a VGA interface.

By the end of the project, we had designed a game with four levels of increasing difficulty. As stated by the requirements, the ball speed doubles and the paddle size halves as the level increases. The game screen in both training and normal mode displays relevant information to the user, that is, the current level, their score and the remaining time when in normal mode. The game was designed with the player in mind, attempting to provide a sufficient amount of challenge, in order to keep the player engaged, while still keeping the game beatable.

If given more time, we could further optimize our code to use less logic elements. This would reduce compilation time which ultimately save resources used to implement the system.

We could also potentially add more creative features to the game along with improved graphics and potentially even sound.

Extra features we could add would be the introduction of powerups to increase or decrease the speed of the balls or the size of the paddle or powerups that could temporarily move the paddle in the vertical direction. These could be used to provide more variety to the game which would also give us the opportunity to add more levels to the game.

Graphics can potentially be refined by adding more colour bits. With more colour bits, we could potentially make levels more colourful and differentiate each level from each other.

The addition of sound would allow us to provide the user with information without them having to actually look for it. This could be useful, for example, when only ten seconds remain on the timer. We could play a beeping tone with each timer tick to warn the user that time is running out. We would also be able to add in music.

More variety, more levels, more colour and added sound effects would lead to increased player engagement and player satisfaction which in turn would make the game more enjoyable for the player.

APPENDICES

FIGURE 1
BLOCK DIAGRAM

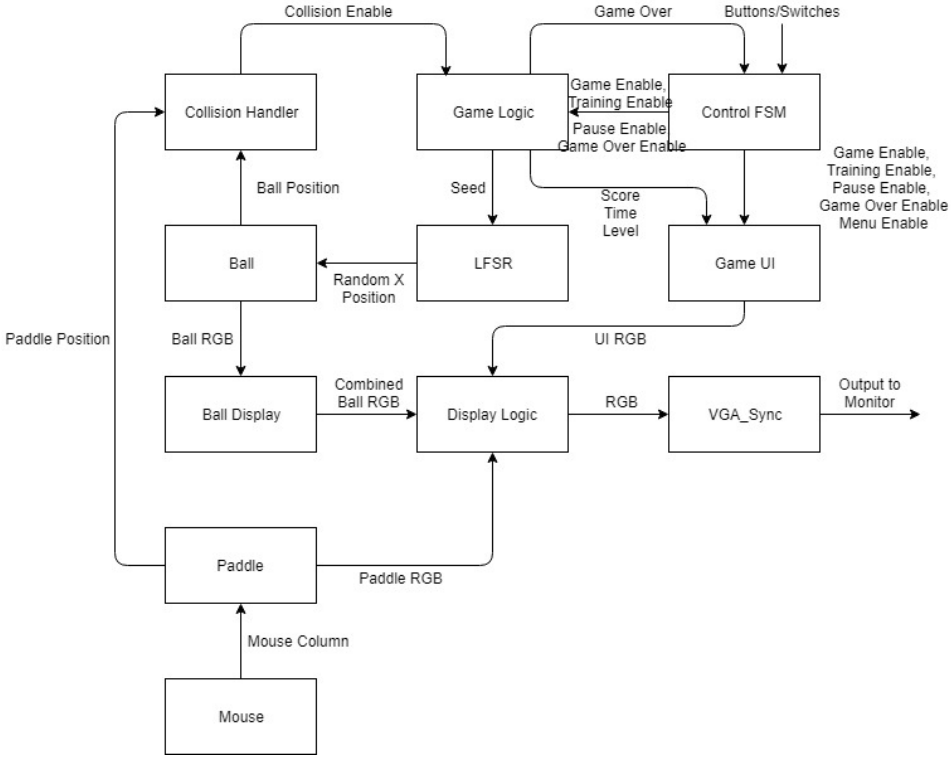


FIGURE 2
FINITE-STATE MACHINE

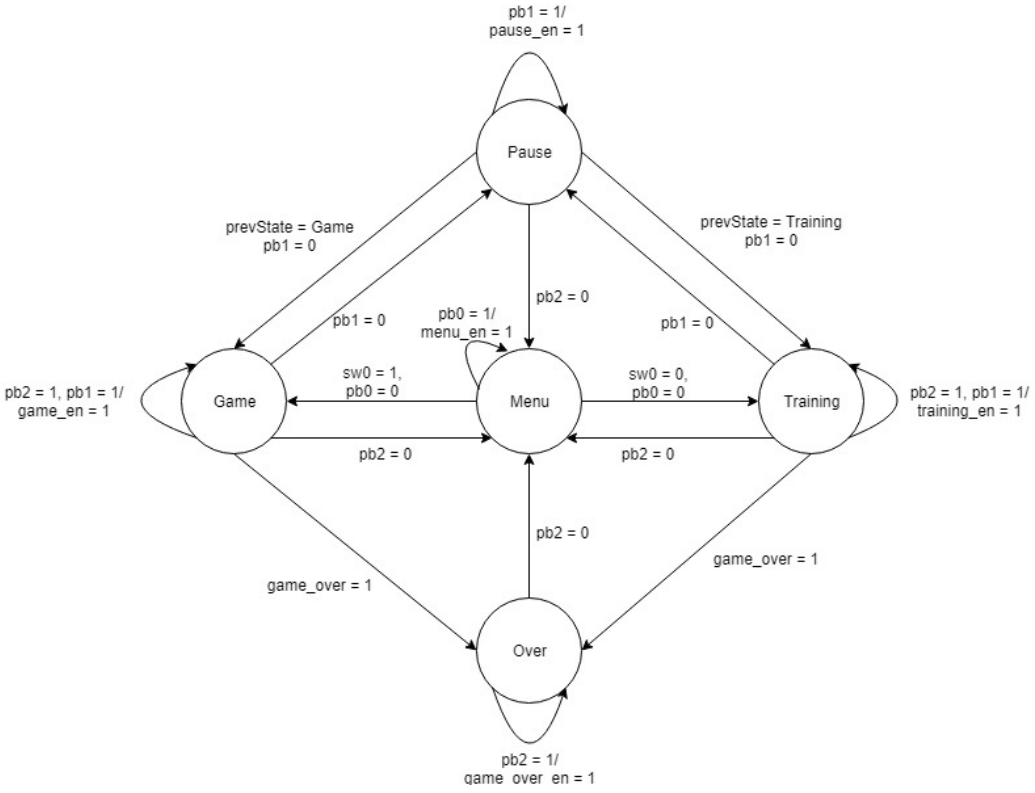
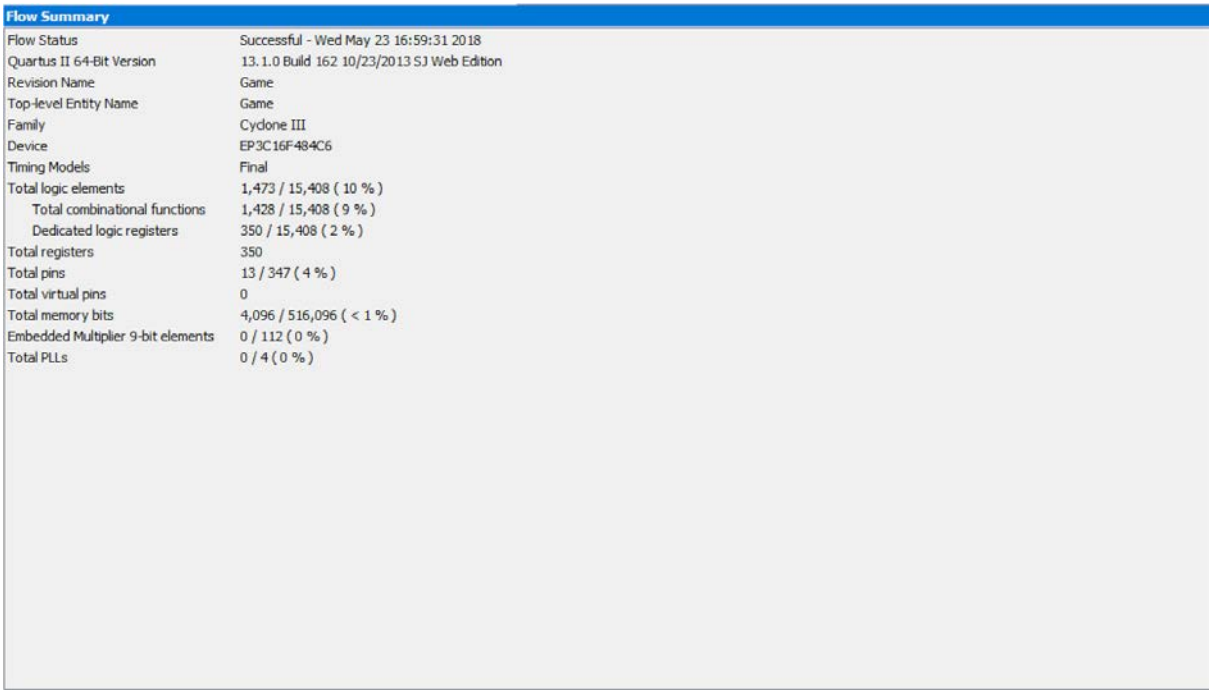


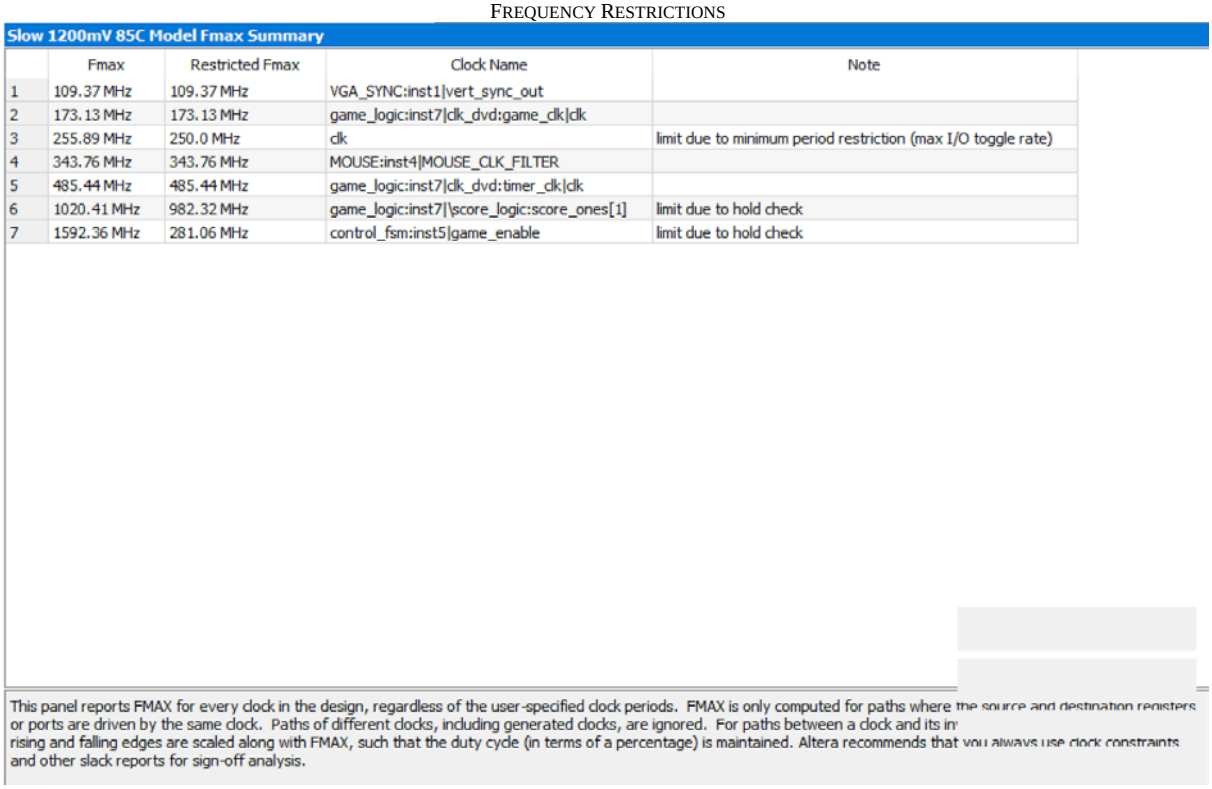
FIGURE 3
FLOW SUMMARY



Flow Summary

Flow Status	Successful - Wed May 23 16:59:31 2018
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Web Edition
Revision Name	Game
Top-level Entity Name	Game
Family	Cyclone III
Device	EP3C16F484C6
Timing Models	Final
Total logic elements	1,473 / 15,408 (10 %)
Total combinational functions	1,428 / 15,408 (9 %)
Dedicated logic registers	350 / 15,408 (2 %)
Total registers	350
Total pins	13 / 347 (4 %)
Total virtual pins	0
Total memory bits	4,096 / 516,096 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 112 (0 %)
Total PLLs	0 / 4 (0 %)

FIGURE 4
FREQUENCY RESTRICTIONS

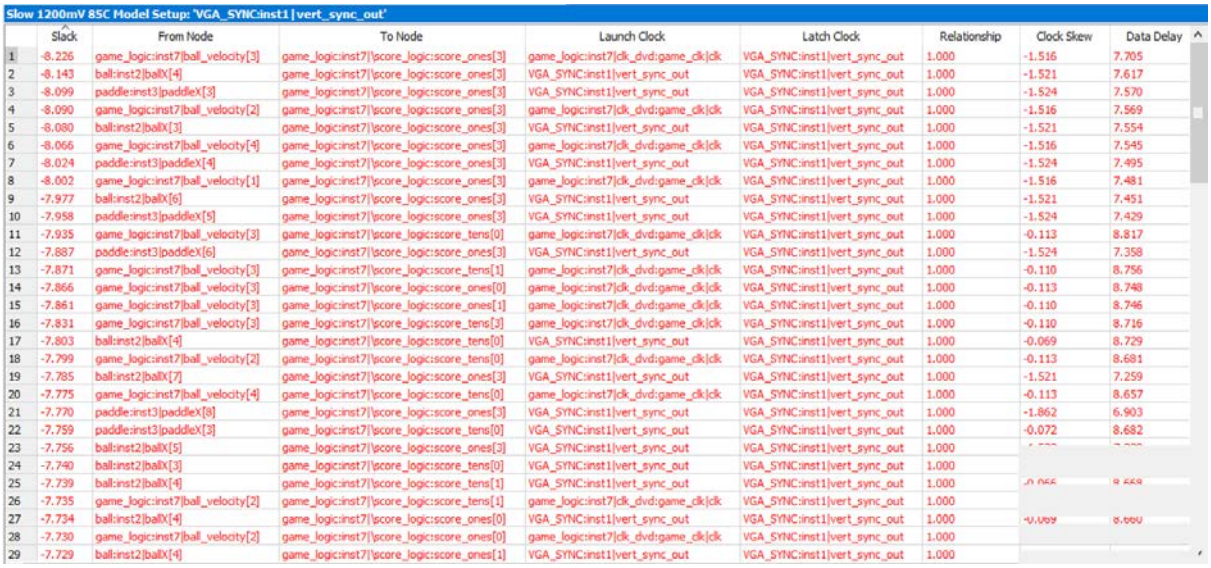


Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	109.37 MHz	109.37 MHz	VGA_SYNC:inst1|vert_sync_out	
2	173.13 MHz	173.13 MHz	game_logic:inst7|clk_dvd:game_clk|clk	
3	255.89 MHz	250.0 MHz	clk	limit due to minimum period restriction (max I/O toggle rate)
4	343.76 MHz	343.76 MHz	MOUSE:inst4|MOUSE_CLK_FILTER	
5	485.44 MHz	485.44 MHz	game_logic:inst7|clk_dvd:timer_clk|clk	
6	1020.41 MHz	982.32 MHz	game_logic:inst7|\score_logic:score_ones[1]	limit due to hold check
7	1592.36 MHz	281.06 MHz	control_fsm:inst5|game_enable	limit due to hold check

This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its in rising and falling edges are scaled along with FMAX, such that the duty cycle (in terms of a percentage) is maintained. Altera recommends that you always use clock constraints and other slack reports for sign-off analysis.

FIGURE 5
CRITICAL PATH



Slow 1200mV 85C Model Setup: 'VGA_SYNC:inst1|vert_sync_out'

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-8.226	game_logic:inst7|ball_velocity[3]	game_logic:inst7|score_logic:score_ones[3]	game_logic:inst7|clk_dvd:game_clk|clk	VGA_SYNC:inst1|vert_sync_out	1.000	-1.516	7.705
2	-8.143	ball:inst2|ballX[4]	game_logic:inst7|score_logic:score_ones[3]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000	-1.521	7.617
3	-8.099	paddle:inst3|paddleX[3]	game_logic:inst7|score_logic:score_ones[3]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000	-1.524	7.570
4	-8.090	game_logic:inst7|ball_velocity[2]	game_logic:inst7|score_logic:score_ones[3]	game_logic:inst7|clk_dvd:game_clk|clk	VGA_SYNC:inst1|vert_sync_out	1.000	-1.516	7.569
5	-8.080	ball:inst2|ballX[3]	game_logic:inst7|score_logic:score_ones[3]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000	-1.521	7.554
6	-8.066	game_logic:inst7|ball_velocity[4]	game_logic:inst7|score_logic:score_ones[3]	game_logic:inst7|clk_dvd:game_clk|clk	VGA_SYNC:inst1|vert_sync_out	1.000	-1.516	7.545
7	-8.024	paddle:inst3|paddleX[4]	game_logic:inst7|score_logic:score_ones[3]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000	-1.524	7.495
8	-8.002	game_logic:inst7|ball_velocity[1]	game_logic:inst7|score_logic:score_ones[3]	game_logic:inst7|clk_dvd:game_clk|clk	VGA_SYNC:inst1|vert_sync_out	1.000	-1.516	7.481
9	-7.977	ball:inst2|ballX[6]	game_logic:inst7|score_logic:score_ones[3]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000	-1.521	7.451
10	-7.958	paddle:inst3|paddleX[5]	game_logic:inst7|score_logic:score_ones[3]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000	-1.524	7.429
11	-7.935	game_logic:inst7|ball_velocity[3]	game_logic:inst7|score_logic:score_tens[0]	game_logic:inst7|clk_dvd:game_clk|clk	VGA_SYNC:inst1|vert_sync_out	1.000	-0.113	8.817
12	-7.887	paddle:inst3|paddleX[6]	game_logic:inst7|score_logic:score_ones[3]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000	-1.524	7.358
13	-7.871	game_logic:inst7|ball_velocity[3]	game_logic:inst7|score_logic:score_tens[1]	game_logic:inst7|clk_dvd:game_clk|clk	VGA_SYNC:inst1|vert_sync_out	1.000	-0.110	8.756
14	-7.866	game_logic:inst7|ball_velocity[3]	game_logic:inst7|score_logic:score_ones[0]	game_logic:inst7|clk_dvd:game_clk|clk	VGA_SYNC:inst1|vert_sync_out	1.000	-0.113	8.748
15	-7.861	game_logic:inst7|ball_velocity[3]	game_logic:inst7|score_logic:score_ones[1]	game_logic:inst7|clk_dvd:game_clk|clk	VGA_SYNC:inst1|vert_sync_out	1.000	-0.110	8.746
16	-7.831	game_logic:inst7|ball_velocity[3]	game_logic:inst7|score_logic:score_tens[3]	game_logic:inst7|clk_dvd:game_clk|clk	VGA_SYNC:inst1|vert_sync_out	1.000	-0.110	8.716
17	-7.803	ball:inst2|ballX[4]	game_logic:inst7|score_logic:score_tens[0]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000	-0.069	8.729
18	-7.799	game_logic:inst7|ball_velocity[2]	game_logic:inst7|score_logic:score_tens[0]	game_logic:inst7|clk_dvd:game_clk|clk	VGA_SYNC:inst1|vert_sync_out	1.000	-0.113	8.681
19	-7.785	ball:inst2|ballX[7]	game_logic:inst7|score_logic:score_ones[3]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000	-1.521	7.259
20	-7.775	game_logic:inst7|ball_velocity[4]	game_logic:inst7|score_logic:score_tens[0]	game_logic:inst7|clk_dvd:game_clk|clk	VGA_SYNC:inst1|vert_sync_out	1.000	-0.113	8.657
21	-7.770	paddle:inst3|paddleX[8]	game_logic:inst7|score_logic:score_ones[3]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000	-1.862	6.903
22	-7.759	paddle:inst3|paddleX[3]	game_logic:inst7|score_logic:score_tens[0]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000	-0.072	8.682
23	-7.756	ball:inst2|ballX[5]	game_logic:inst7|score_logic:score_ones[3]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000		
24	-7.740	ball:inst2|ballX[3]	game_logic:inst7|score_logic:score_tens[0]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000		
25	-7.739	ball:inst2|ballX[4]	game_logic:inst7|score_logic:score_tens[1]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000		
26	-7.735	game_logic:inst7|ball_velocity[2]	game_logic:inst7|score_logic:score_tens[1]	game_logic:inst7|clk_dvd:game_clk|clk	VGA_SYNC:inst1|vert_sync_out	1.000		
27	-7.734	ball:inst2|ballX[4]	game_logic:inst7|score_logic:score_ones[0]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000		
28	-7.730	game_logic:inst7|ball_velocity[2]	game_logic:inst7|score_logic:score_ones[0]	game_logic:inst7|clk_dvd:game_clk|clk	VGA_SYNC:inst1|vert_sync_out	1.000		
29	-7.729	ball:inst2|ballX[4]	game_logic:inst7|score_logic:score_ones[1]	VGA_SYNC:inst1|vert_sync_out	VGA_SYNC:inst1|vert_sync_out	1.000		