COMPSYS 305: Intercept Pong

George Xu and Jerry Haoyuan Yang
The University of Auckland, gxu630@aucklanduni.ac.nz, hyan927@aucklanduni.ac.nz

***Abstract* – Intercept Pong, a ball catching game created by Group 22 is a game run off the Altera (Development & Education 0) board, with the video display through Video Graphics Array to a monitor, and a (Personal System/2) mouse to control the in-game platform. The objective of the game is to catch the specified number of balls or more with the platform within the specified time frame. The game progresses to more difficult levels when the objective of the previous level is met. There are two game modes, Training and Game. On training mode, the player can practice the game with no time limit until they have learnt the game. In game mode, the player progresses through the four levels before completing the game. Intercept Pong is easy to learn, and is very simplistic in nature, which will add to the replayability of it. While there is room for improvement, the game is very enjoyable in its current state, and implementations of future improvements will be easy with the modularity of the existing code.**

Index Terms - Altera, DE0 Board, Game, Intercept Pong.

INTRODUCTION

In COMPSYS 305: Digital Systems Design I, our team was tasked with the development of a Pong game to be run off a programmed Altera Development Board outputting to a 640x480 VGA screen, while using a PS/2 mouse as the primary user input. This game involves a ball actively and randomly bouncing off the walls of a screen. The objective of the game is to catch the ball with the mouse controlled moving platform at the bottom of the screen. Once caught, balls respawn at a random location along the top wall and immediately start moving at a random angle. Catching the ball increases the score, and obtaining the set score threshold for a level within a set amount of time will allow the player to advance to the next level. There are 4 levels, with increasing difficulty reflected by changes in variables within the game.

DESIGN SPECIFICATION

The objective of this project is to design a game console with a built-in computer game using only digital logics and digital design. This system will be designed to work on a Terasic DE0 board containing an Altera Cyclone® III 3C16 FPGA device for the logic to be implemented on. This particular FPGA contains 15,408 LEs, so the entire game logic must utilize less than this amount of LEs. The system will require and utilize a PS/2 Mouse to allow the player to send inputs into the system and interact with the game elements. The video output from the board will be through VGA at a resolution of 640x480 (4:3) at 60Hz. The Terasic DE0 board is equipped with 10 dip switches and 3 push buttons, where a number of them will be utilized as an additional method for the player to interact with the system, such as changing game modes, starting or pausing the game. A graphical user interface (GUI) with textual messages will be implemented to advise the player of the controls and available game modes. Time and score values will be displayed on the screen while in game.

GAME STRATEGY

Players can choose from training mode or game mode with the use of a DIP switch, and the game is able to be started with the use of a push button.

The purpose of training mode is to help the player familiarize themselves with the basic game mechanics before entering game mode. Training mode will run until the player reaches the score threshold for the level, with the difference being that there will be no time limit, and the player cannot level up through training mode. We have decided to use the last saved played level on game mode in training mode, so if the player finds a particular level too challenging, they are able to practice on that level without a time limit before re-attempting it in game mode. When the console reboots, the last saved game mode is reset to 1, so initiating training mode after a reboot will load level 1.

For game mode, a constant time constraint across all levels will be introduced, to place pressure on the player, and to make the game more enjoyable, but will not be challenging enough to make the game impossible. The difficulty will increase as the player reaches the later levels, where the ball motion speed, score threshold, time, platform size and platform position boundaries change, so the player will need to demonstrate faster reaction speeds and higher levels of accuracy to pass the levels. The player will need to advance to and complete level 4 to win the game. The PS/2 mouse is very responsive, with the platform being able to move as fast as the player chooses, so the difficulty is purely based off the player's skills rather than the refresh frequency of the computer system.

Upon exiting to the main menu, the game saves the last played level, and continues it when the player attempts to re-enter training or game mode. This will eliminate the frustration of starting from level 1 after losing or returning to main menu.

FINAL DESIGN AND IMPLEMENTATION

The final design we have created includes everything in the design specification, and allows the player to challenge themselves with the increasing difficulty throughout the game. The game contains a graphical user interface in the form of a main menu where the player can choose between the two game modes. There are win, lose and pause screens to indicate to the player of what events have occurred in the game. The platform and ball are colored red and blue respectively to make them distinct while in-game. The text is displayed in a subtle yellow to minimize distraction while in-game.

The high-level components were implemented using a block diagram to maximize understandability while viewed from the top level. The game uses a finite state machine to transition between the different states, such as Main Menu, Game, Game_Win etc. The logic has been split into multiple components with specialized functions, increasing cohesion and modularity. The 7 segment displays have been utilized as an alternate display for mouse position. A linear feedback shift register has been implemented to assist the pong_game component in generating random spawn locations for the ball. Two clock dividers were used, one for the common frequency of 25MHz and the other for generating the 1Hz signal for the in-game timer. Debouncers have been implemented to eliminate the unstable states that occur during button presses.

BLOCK DIAGRAM AND DESCRIPTIONS

The implementation of the game consists of a VHDL program made in Quartus 13.0 using functional blocks. They are connected through wires or buses to form the game. The functionality of the blocks is as of follows:

- **pll_25MHz:** This takes the 50MHz generated by the System Clock and halves it to generate the 25MHz clock required by the vga_sync to generate an image at a refresh rate of 60Hz as required by the monitor.
- **Clock Divider 1Hz:** This takes the 25MHz signal generated by the pll_25MHz and generates a 1Hz signal to be used by the game to increment the timer once every second.
- **LFSR:** This generates a random string of 11 bits to be used by the game to randomly spawn the ball at the top of the screen and begin moving in a random direction.
- **Mouse:** This processes the mouse speed input signals and outputs the position of the mouse fixed to the boundaries of the screen resolution of 640x480.
- **7 Seg Decoder:** This takes 4 bits as input as input and converts this data to the data representing the 7 bits illuminating each of the segments on the display.
- **Text Generator:** This functional block handles the first stage of text generation for the game. It can take numbers in bits and outputs the font address and location data that char_rom uses to output text onto the screen. It also stores hardcoded game text.

- **char_rom:** This functional block handles the second stage of text generation for the game. It takes the font address and location data outputted by Text Generator and outputs signal data to generate visible text onto the screen.

- **Game:** This functional block contains the heart of the game. It uses signals provided by other functional blocks to implement its functionality, as well as taking mouse inputs to control the platform and animating the ball. It outputs video signals to vga_sync.

- **Game Controller:** This controls the main menu of the game. It also controls whether the video output to vga_sync is the main menu, other relates states, or the game. It takes user inputs via the switches and buttons to allow the user to select game mode, and outputs this data to the Game block where the game mode will be run.

- **vga_sync:** This controls horizontal and vertical synchronization signals. The horizontal sync tells the screen when to refresh a row of pixels, while the vertical sync tells it when to display a new frame, i.e. when all 480 rows have been refreshed.

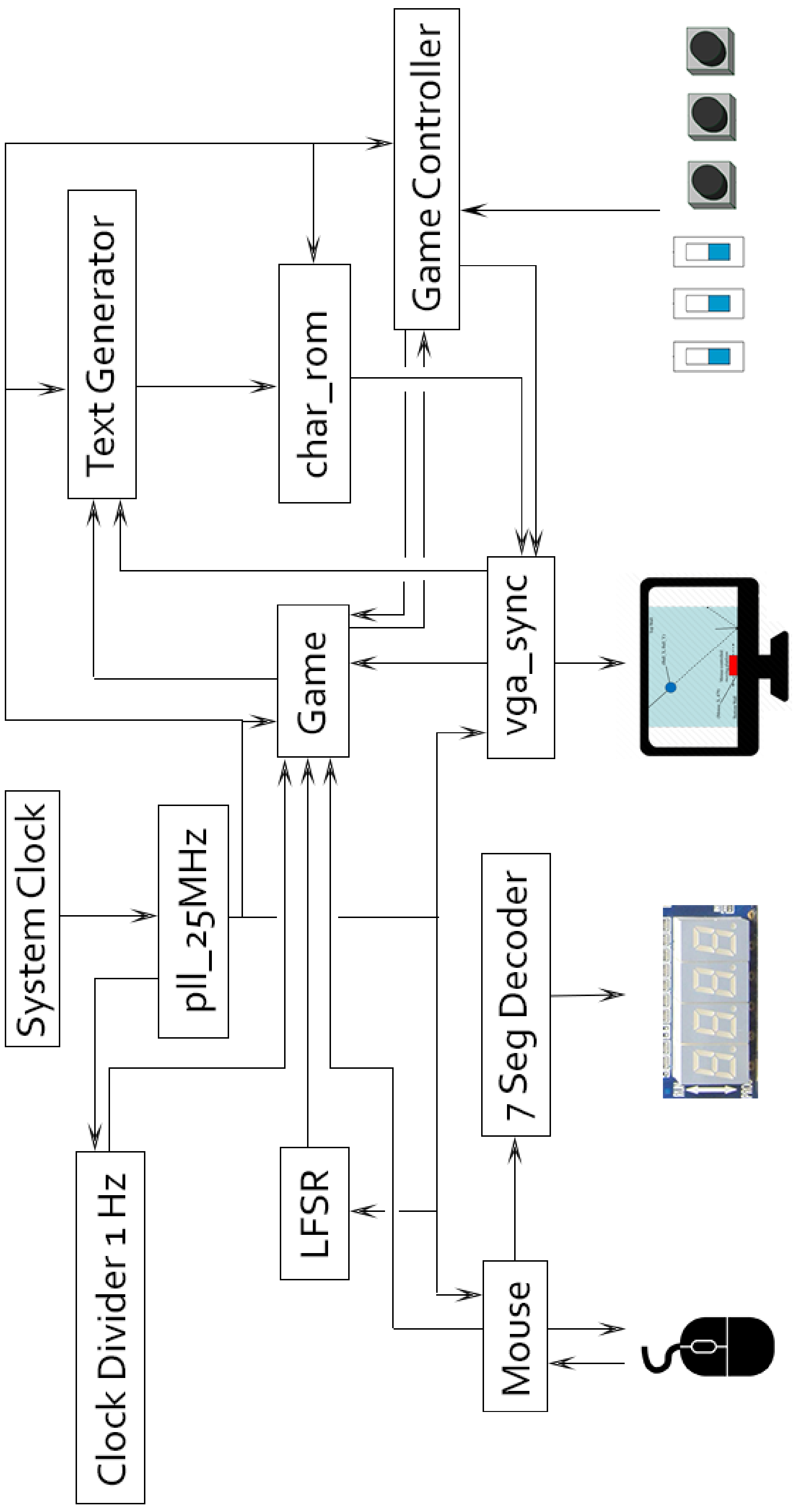

FIGURE 1:
BLOCK DIAGRAM

FINITE STATE MACHINE

In order to control the transition of the different states of Intercept Pong Game, we have created an FSM.

- **Main menu:** The start state of the game
- **Training 1:** Training mode 1 of the Intercept Pong Game
- **Training 2:** Training mode 2 of the Intercept Pong Game
- **Training 3:** Training mode 3 of the Intercept Pong Game
- **Training 4:** Training mode 4 of the Intercept Pong Game
- **Level 1:** Level 1 of the Intercept Pong Game.
- **Level 2:** Level 2 of the Intercept Pong Game.
- **Level 3:** Level 3 of the Intercept Pong Game.
- **Level 4:** Level 4 of the Intercept Pong Game.
- **Pause:** Pause state of the Game when press button_1. This stop all animated objects and reanimates them upon resuming
- **Game win:** The game has been won. It reaches this state when the player has advanced through all the level and passes Level 4.
- **Game loss:** The game is lost by either time running out without satisfying the score threshold or by pressing Button 0.

The FSM begins at the Main menu screen. When button_2 is pressed and DIP switch 0 is switched to 0, the game transits to the training mode where the player will be able to practice the game at the current saved level (Level 1 initially) without a time limit. If DIP switch 0 is 1, the game transits to the current saved level (Level 1 initially) when button_2 is pressed.

Each state will display different visual contents to the player. The static states display button and dip switch functionality information to the player. All levels have their own score requirements. If the player does not pass a level, they will transit to the game over state, where they are able to reattempt the level, otherwise they will transit to the next level. If the player passes level four, they will reach the Game Win state where they will be notified they have won the game. The player may press button_1 and button_2 during the game to pause and resume the game.

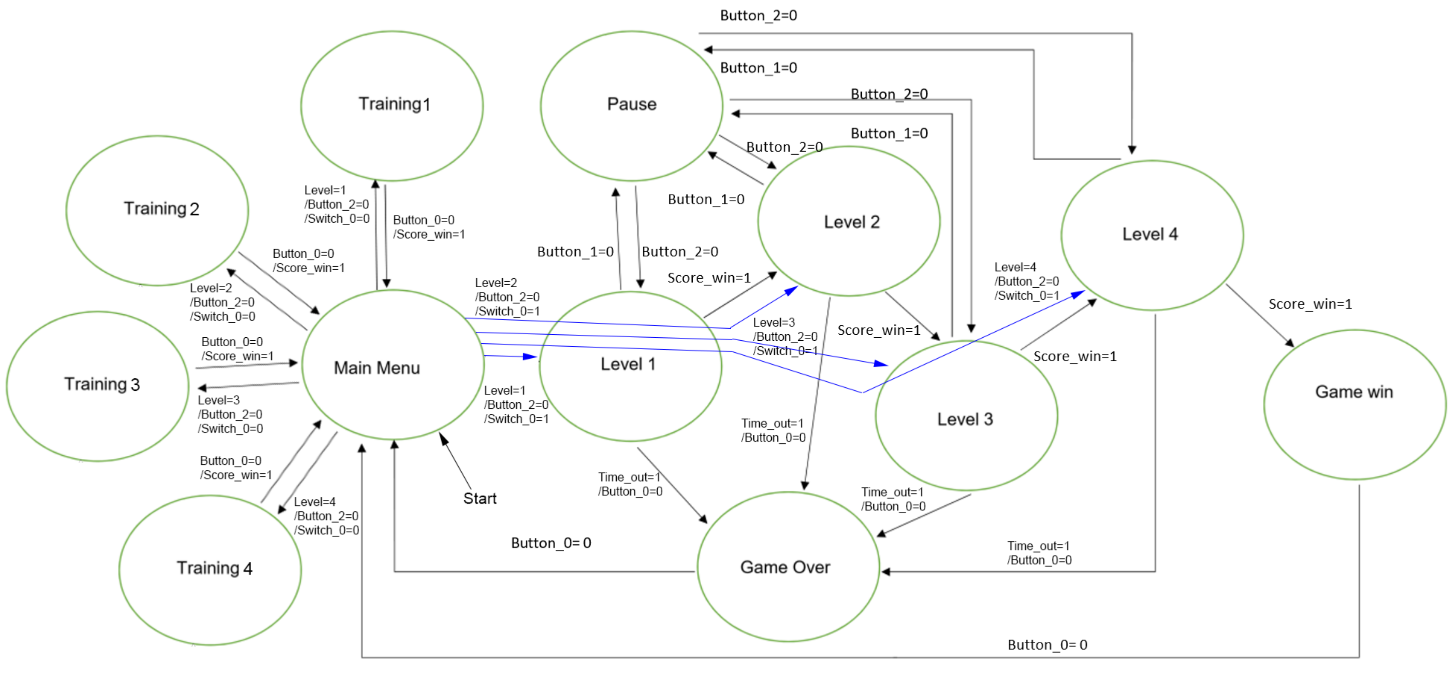

FIGURE 2:
FINITE STATE MACHINE

KEY DESIGN DECISIONS

One of the main design decisions we made for the intercept Pong game was how to increase the difficulty as a function of the level. There were many variables we considered such as time, platform size, ball speed, score threshold, ball size and platform position bounds, though through careful consideration, we chose to go with time, platform size, ball speed and platform position bounds. We wanted to implement as many features as possible, however we recognized that the more features implemented, the more difficult the game would to learn, so we decided against these extra features.

One of the design constraints that we met was the speed of the ball. As the speed was increased, the travel distance would increase during refresh cycles, and the ball would occasionally travel out of the screen. The workaround of this was to decrease the screen boundary, however that would cause the ball to seemingly rebound without touching the wall. Because of this reason, we decided to limit the speed of the ball to the point where it would not exhibit these behaviors.

The difficulty of the game can be increased by increasing the score threshold for each level. The caveat for this is that we were using a random number generator for the spawn angle of the balls, so the maximum possible score for each level, assuming constant ball travel speed, is randomized. This made it in some cases impossible for the player to achieve the score threshold for the level. We have worked around this by decreasing the score threshold to the point where this occurrence is highly unlikely.

We decided to split the finite state machine into two components, one inside the game block and the other in the game controller block. The game block would handle the state switches between each level, and the game controller block would handle the rest of the state switches. Both blocks can communicate with each other to cross-switch between the blocks. This was done so that our design team were able to work in parallel to produce a single FSM in a much shorter period. In addition, it increased modularity and decreased the number of signals and busses between the blocks.

We have introduced smooth transitions between each level, where the positions of the balls do not change during each level up, however the speed of the balls increase. This was contrary to returning the player to a level selection screen, or respawning animated objects to their original position. We have done to make the game more immersive, however this also makes the game more addictive, a positive or negative trade-off depending on the player.

RESOURCE USAGE AND PERFORMANCE

The game runs on an Altera Cyclone® III 3C16 DE0 board with a 50MHz clock frequency, where it is divided in to 25MHz and 1Hz frequencies for our internal counters. We have chosen these frequencies as they are the most commonly used frequencies in our logic blocks. The final design used 3063 logic elements, 20% of the 15408 total logic elements available, as well as 394 dedicated logic registers. 20480 of the total memory bits were used, around 4% of the total available of 516,096. We have used less memory bits than other projects, and that is likely due to the fact that we have not used any bitmaps.

We have utilized all the push buttons and one dip switch to allow the user to interface with the UI. In addition, the current mouse position is displayed on the 7-segment display, where the user can use it to debug an incompatible mouse. We have utilized all the VGA pins available to us to display a contrasting and vibrant image.

POSSIBLE IMPROVEMENTS

While creating this game, there were features we have thought of implementing, but due to time constraints, we were unable to do so. Improvements we pondered on include lives for each level, multiuser challenges, high scores, as well as embossed graphics. We decided against these features as we would rather allocate time for optimization of gameplay mechanics to make the game more enjoyable, and to preserve the simplistic nature of pong.

If we were to make any improvements, we would have implemented the multiuser challenges functionality. This functionality lets the player play at level 4, and records the number of balls caught within the 60 second time frame. The score is then stored and is able to be accessed via a separate state extending from the main menu. This will add competitiveness and increase replayability of the game.

CONCLUSION

Intercept Pong has been a comprehensive learning experience that has consolidated the knowledge we have previously learnt from Computer Engineering, Analogue and Digital Design and Digital Systems Design. It was also our first time creating a game in VHDL. Though simple in nature, we believe the game will have a lot of replayability, due to the difficulty increases with each level, as well as the player being able to improve their skills and challenge themselves to complete levels in the shortest period of time. If we had more time, we would have implemented the multiuser challenges functionality to further improve replayability.