Digital Design 2: Final Project Report

Abhishek Damle
Charles Hall

Table of Contents

System Block Diagrams

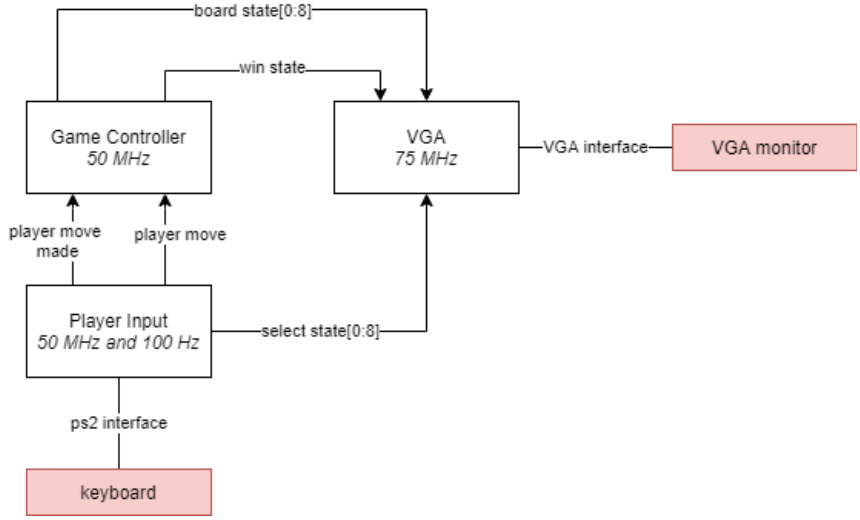

Figure 1: High level system block diagram

Table 1: Description of high level modules signals

Modules	
VGA	Draws the tic tac toe board on the VGA monitor
Game Controller	Contains the logic for the tic tac toe game
Player Input	Moves selected position based on keyboard input and allows player to select a position
Signals	
board state	Stores whether each of the nine places on the tic tac toe board are free, occupied by a circle, or occupied by a cross
win state	Stores whether the game is in progress, has been tied, has been won by the computer, or has been won by the player
select state	Stores which of the nine places has been selected by the player(used for debugging purposes as multiple places can be selected at once)
player move	Stores which of the nine places has been selected by the player
player move made	Pulses for 1 50 MHz clock cycle when the player has selected their position

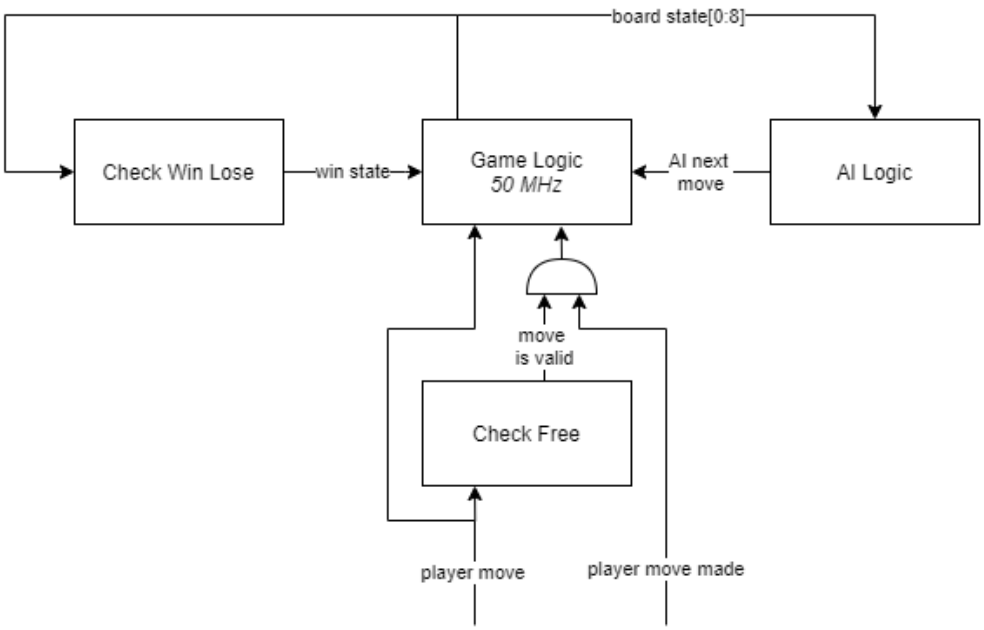

Figure 2: Block diagram of Game Controller module

Table 2: Description of Game Controller submodules and signals

Modules	
Check Win Lose	Checks whether the game is in progress, has been tied, or has been won
Game Logic	FSM that alternates giving turns to the player and AI
AI Logic	Chooses the AI's move
Check Free	Determines whether players move is valid
Signals	
board state	Stores whether each of the nine places on the tic tac toe board are free, occupied by a circle, or occupied by a cross
win state	Stores whether the game is in progress, has been tied, has been won by the computer, or has been won by the player
AI next move	Stores which of the nine places has been selected by AI
player move	Stores which of the nine places has been selected by the player
player move made	Pulses for 1 50 MHz clock cycle when the player has selected their position
Move is valid	Stores whether the player's move is valid

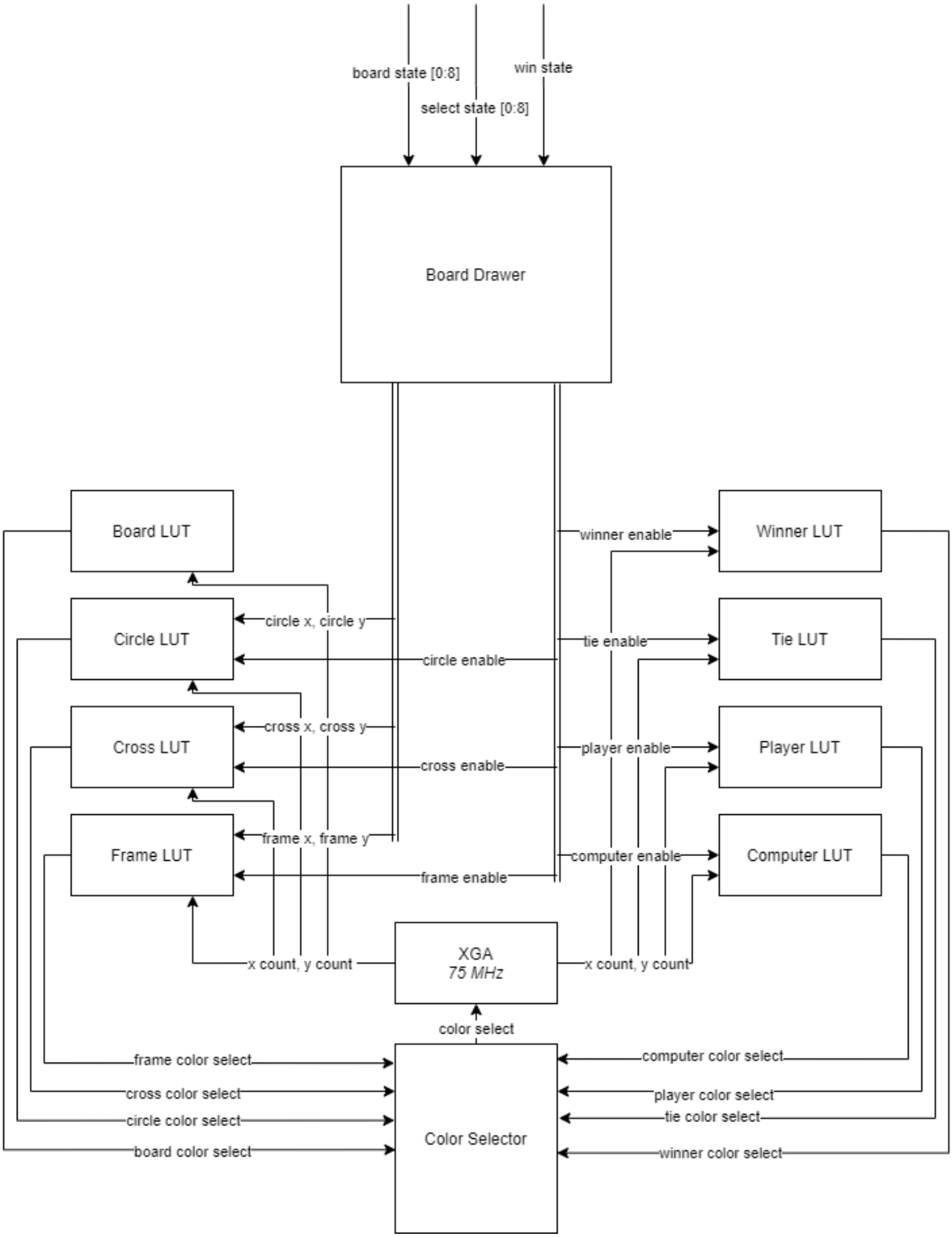

Figure 3: Block diagram of VGA module

Table 3: Description of VGA submodules and signals

Modules	
Board Drawer	Draws game board based on the state of the board, which position is selected, and whether the game has been won or not
Color Selector	Prioritizes outputs of the LUT's
Board LUT	LUT for the board
Circle LUT	LUT for circle sprite
Cross LUT	LUT for the cross sprite
Frame LUT	LUT for the frame sprite which denotes the player's selected position
Winner LUT	LUT for the word "winner"
Tie LUT	LUT for the word "tie"
Player LUT	LUT for the word "player"
Computer LUT	LUT for the word "computer"
Signals	
board state	Stores whether each of the nine places on the tic tac toe board are free, occupied by a circle, or occupied by a cross
win state	Stores whether the game is in progress, has been tied, has been won by the computer, or has been won by the player
select state	Stores which of the nine places has been selected by the player
circle, cross, frame, winner, tie, player, and computer enable	Enables respective LUT
board, circle, cross, frame, winner, tie, player, and computer color select	Output of respective LUT
circle, cross, and frame x circle, cross, and frame y	Stores x and y position of its respective sprite
color select	Stores which color the current pixel should be

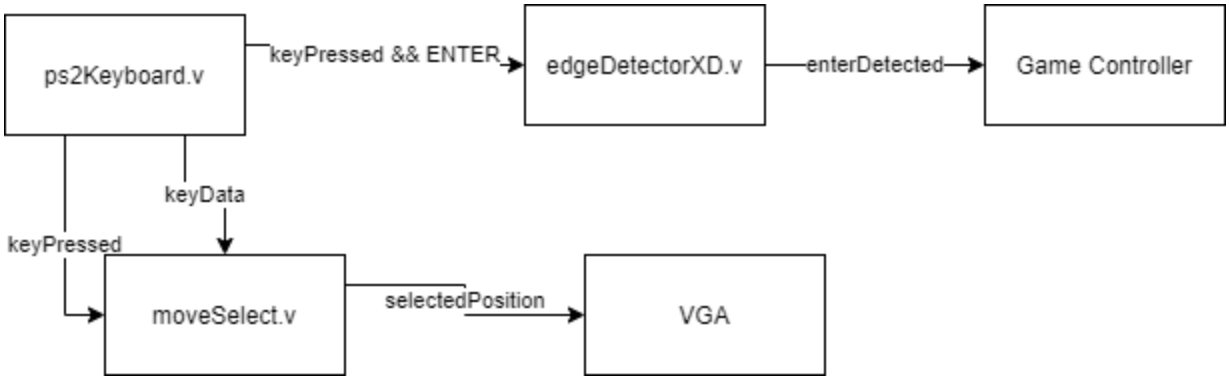

Figure 4: Block diagram of PlayerInput Module

Table 4: Description of PlayerInput Modules

Modules	
ps2Keyboard	Outputs the data read from the keyboard
edgeDetectorXD	Outputs a positive signal when the ENTER key is pressed.
Board LUT	Outputs the current selected position based on the keyData read from the ps2Keyboard module.
Signals	
keyPressed	Stores when the ps2Keyboard module finishes reading in the 8-bit key code from the keyboard
keyData	Stores the 8-bit key code read in from the keyboard.
enterDetected	Stores when the ENTER key is pressed and the positive 50mhz clock edge is detected.
selectedPosition	Stores the player's current selected position

Project File Structure

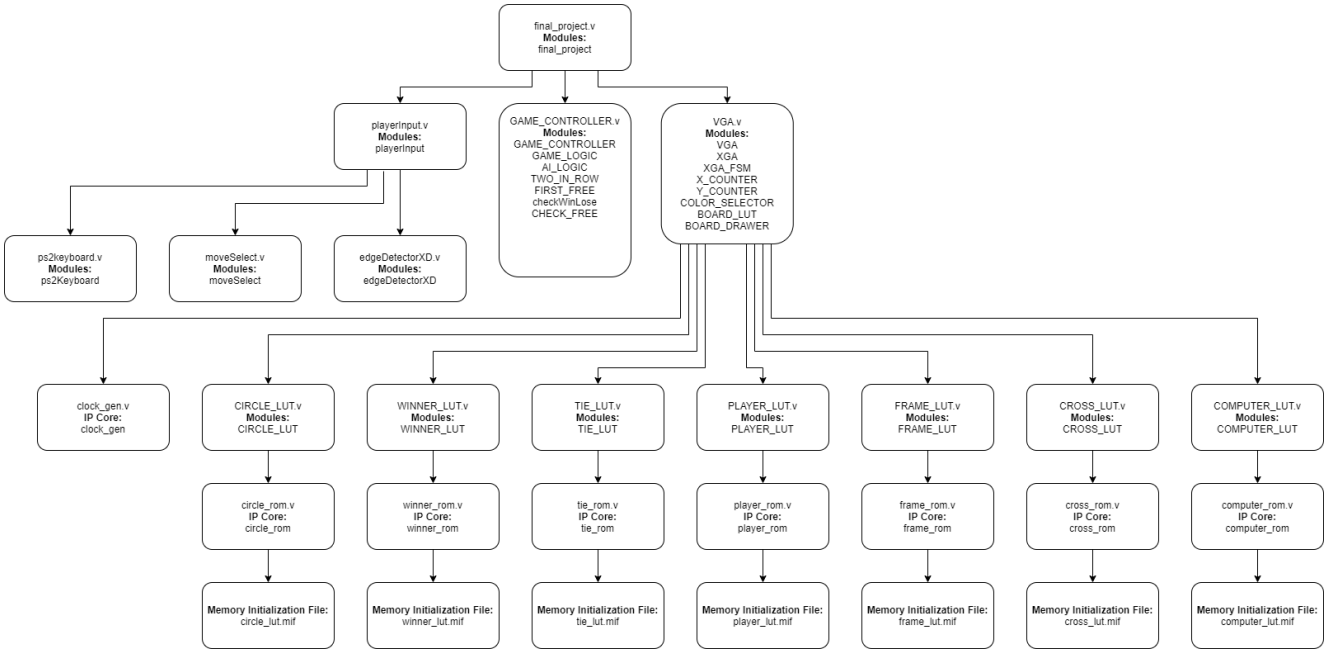

Figure 5: Hierarchy of Files

Technical Discussions

This project was an extension of homework 6 that allows a player to play a tic tac toe game using a keyboard. The fundamental additions to this project were the logic for the tic tac toe game, the module responsible for drawing the board, and the module that reads keyboard inputs through the ps2 protocol. A block diagram of the tic tac toe game logic is shown in figure 2 and descriptions of the modules and signals from figure 2 can be found in table 2. The “Game Logic” module is responsible for alternating between the player and the AI’s turns and takes inputs from purely combinational modules which determine the AI’s move, whether the game has been won, and the player's valid move. The module that decides the AI’s next move is the most complex amongst these modules and to simulate an opponent making “intelligent plays”, the following criteria, in order of importance, are used to determine the AI’s next move.

1. Does position result in 3 in a row for the AI?
2. Does position block 3 in a row for the Player?
3. Is the position the first free position?

The game board was drawn through the “VGA” module which is an adaptation of the top level module from homework 6. The block diagram and descriptions for the sub modules and signals in the “VGA” module can be found in figure 3 and table 3 respectively. The core of the “VGA”

module is the “XGA” module which simply sets the current pixel’s color to an inputted color value and outputs the current pixel’s x and y position. Like the ball from homework 6, the circle, frame, cross, winner, tie, player, and computer sprites determine the color of the current pixel based on their relative x and y position to the current pixel’s x and y position as well as the contents of their look up tables which are implemented as ROM’s. The sprites x, y positions, and whether they are enabled are determined by the “Board Drawer” module based on the board state, select state, and win state. The outputs of all of the sprites are fed into the Color Selector module which prioritizes them and selects the color of the current pixel.

The last fundamental addition was the “playerInput” module. When a key is pressed on the keyboard, it will send an 11-bit data frame to the FPGA. The frame consists of a zero start bit, 8-data bits corresponding to the key pressed, a parity bit, and a stop bit. Once the start bit is detected, the ps2Keyboard module will move into a DATA state, where it reads in 8 bits of data that correspond to the specific key pressed on the keyboard.

Mixed Clock Domain Design

We had to implement an edge detector due to the differences between the main 50 MHz clock and the 100 Hz clock being read in from the ps2 keyboard. The edgeDetectorXD module ensures that the keyPressed signal is set for one clock cycle on the 50 MHz clock instead of the slower 100 hz clock that is read in from the keyboard. Through pulsing the keyPressedsignal for 1 50 MHz clock cycle and holding the hex code for the key entered in a register, the “ps2Keyboard” module is able reliably transmit the key pressed to the rest of the “playerInput” module which is clocked at 50 MHz.
We didn’t have to worry about the mixed clock between the 50 mhz clock and the 75mhz VGA clock. This is because the game state is stored and changed in one module, and the VGA module is only concerned about reading in the updated data from that module and the game is constant for long periods of time. .

Sprite Generation

Since the ROM depth for the pictorial sprites is 65536 while the depth for the text sprites is 6144, an automated method of setting the ROM contents needed to be developed. In order to generate the ROM contents, a sprite was first drawn on a 256 x 256 or 256 x 26 pixel canvas. This sprites were then converted to bmp images and a program called “lcd-image-convertor” was used to get the color values for each of the pixels in the sprites. These color values were then formatted by the “lut_gen.cpp” program into the ”.mif” format which is used to program ROM. This process allowed for rapidly converting sprites in the .bmp image format to ROM contents and greatly reduced the amount of time and effort taken to program the ROM.

Memory Usage Optimization

A naive approach would have been to have instances for the look up tables for the circle, cross, and frame sprites for each of the nine positions which get enabled and disabled based on the state of the game. In order to optimize the memory usage, we employed a method that allowed us to only use only one instance of the look up table for the circle, cross, and frame sprites. In our method, we split the 768x768 pixel, 3x3 game board into 9 individual 256x256 pixel blocks. We then check to see which of the 9 blocks the current pixel position of the “XGA” module lies in. Based on this, we set the x and y positions of the sprites at one of the correct 9 positions.

Figure 6 shows an example game board with circle images. Utilizing our method, these images can be shown using a single sprite and look up table and table 5 shows the x and y positions of the single circle sprite for each current x and y pixel value.

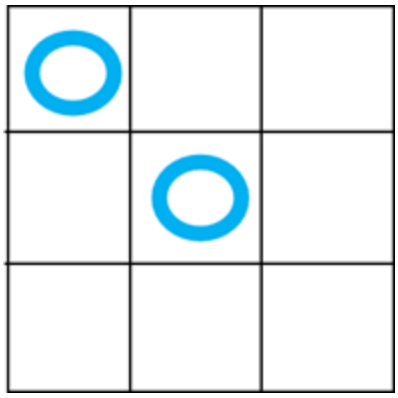

Figure 6: Example game board

Table 5: X and Y positions given to a single sprite to display image from figure 6

		Pixel’s x position		
		0-255	256-511	512-767
Pixel’s y position	0-255	0,0	disabled	disabled
	256-511	disabled	256,256	disabled
	512-767	disabled	disabled	disabled

User Manual

When you start the game, the top left square on the tic-tac-toe board should be highlighted. You can move the yellow square using the **A (Left), S (Down), W (Up), D(Right)** keys on the keyboard. When you want to confirm your position selection, press the **ENTER** key. A blue ‘O’ should appear in the selected position. Then, the computer should select a position, which will

contain a red ‘X’. You can select another position as long as the position isn’t already occupied by an ‘O’ or an ‘X’. If you manage to get 3 ‘O’s in a row, you win the game. However, if the computer gets 3 ‘X’s in a row, you lose the game. If all of the positions of the board are filled, and neither the player nor the computer has 3-in-a-row, then the game result is a tie. Whether you win, lose, or tie, there will be an indicator that appears on the right side of the screen. To reset the game, just press the **KEY0** on the FPGA board.