**CMPEN 371: Advanced Digital Design**

**Fall 2017**

**VHDL Tic-Tac-Toe**

**Partial Project Report**

**Team 2C**

**Karina Cuadrado, kmc6355**

**Kyle Jonas, krj5137**

***Do not include the attachments sections when you print the hard copy to be delivered at the instructor’s office.***

**ACKNOWLEDGEMENT**

This work is entirely our own except as noted. The approximate contribution of each team member is as follows:

50% Karina Cuadrado \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

50% Kyle Jonas \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Grading Rubric**

|  |  |
| --- | --- |
| **Component** | **Grade** |
| *Proposal* | / 5 |
| *Design Review*  *· Design documents* | / 10 |
| *Progress Demonstration*  *· Demo of Milestones* | / 10 |
| *Partial Project Report*  *· Abstract and Introduction*  *· Functional Specification*  *· Design* | / 15 |
| *Project Presentation*  *· Gives good overview of the project*  *· Gives something interesting to the class*  *· Participation during other presentations* | / 10 |
| *Final Project Demonstration*  *· Significant accomplishments working*  *· Project shows that the team has met objectives of the course* | / 40 |
| *Final Project Report*  *· Completes the partial project report*  *· Verification, Performance, Conclusions* | / 10 |
| *Bonus/Penalty* |  |
| TOTAL | / 100 |

**ABSTRACT**

For the CMPEN 371 final project our team chose to develop a functioning Tic-Tac-Toe game implemented on the Spartan 7 FPGA. The game is played using keyboard control to place red on blue blocks onto the 3x3 grid, which is then displayed to a monitor using VGA.

The project was split into two main components, GameControl and VGAcontrol. GameControl, the Tic-Tac-Toe logic was implemented while VGAcontrol handled drawing game pieces and the 3x3 grid to the monitor.

**INTRODUCTION**

In this project, we managed to implement multiple interfaces such as keyboard (PS/2) and VGA display. We also familiarized ourselves with the designing and verifying of a combinational logic circuit. It provided a challenge to test our VHDL coding and logic abilities to implement such a game. Following up on the basic concepts of a Tic-Tac-Toe game, we provide the users the opportunity to reset the board at any time throughout their game. From the classes where the interfaces PS/2 and VGA were introduced, we developed and implemented our design with modifications. Overall, the purpose of this program is to sharpen our skills in logic circuits and VHDL coding as well as to provide the users an easy and clean platform to play the two-player pencil-and-paper game Tic-Tac-Toe.

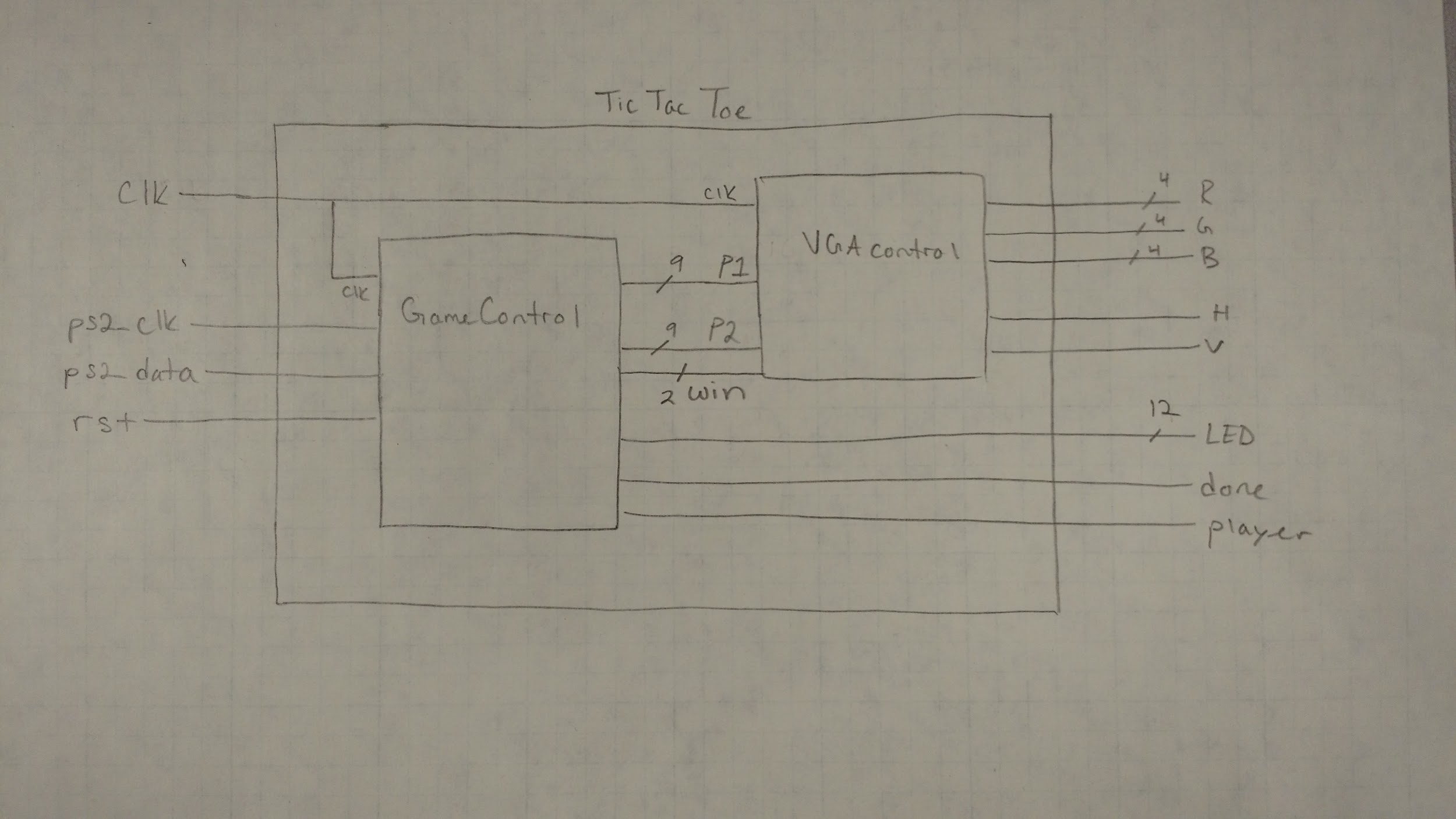
**FUNCTIONAL SPECIFICATION**

This Tic-Tac-Toe program takes in as input the position (numbers 1 through 9 from keyboard assigned from top-left to bottom-right of the gameboard) of where the player wants to place their colored box. The other inputs are reset, mapped to the CPU RESET button in FPGA board, which clears the game board from all drawn boxes, and the system’s clock to be used by our PS/2 and VGA interfaces. This system assumes player one (blue) will place their box first. Once they have pressed the key corresponding to their desired position, an internal signal, which keeps track of which player’s turn it is, will be negated to allow the other player to place their colored box and so on. Internally, the system has assigned 9-bit signal/array for drawing the boxes to every 1-9 key. The way the system determines the color is by the internal signal that tracks the player’s turn as well as the 9-bit signals that determine which player has what colored box in the game board at the moment.. The final outputs of the system are VGA display (R, G, B, H and V signals) where the game board and boxes will be drawn, active-high LEDs 0-8 where the number of the key pressed will be displayed for debugging purposes and visual display, and the player’s turn (LED 15) where ON = player 1 and OFF = player 2.

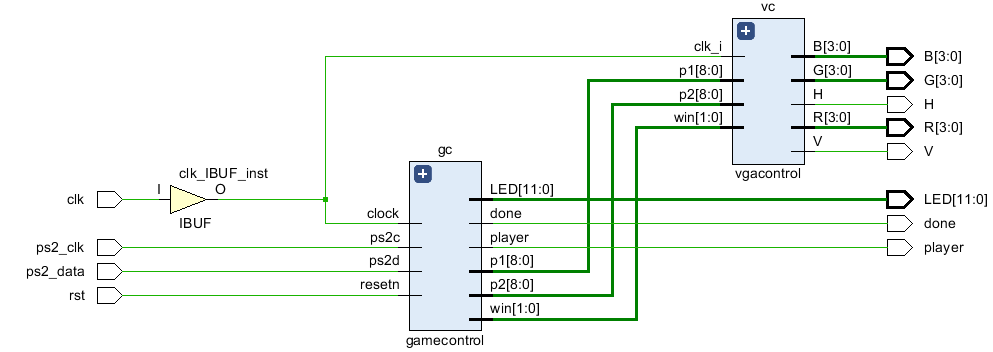
**DESIGN**

The Tic-Tac-Toe game is built off of the main TicTacToe module and implements two primary components, GameControl, and VGAcontrol. GameControl handles taking in keyboard commands, controlling the player's game pieces, and managing the player's turn. VGAcontrol takes in the player game pieces from GameControl and then draws both the gameboard as well as the players blocks onto the monitor.

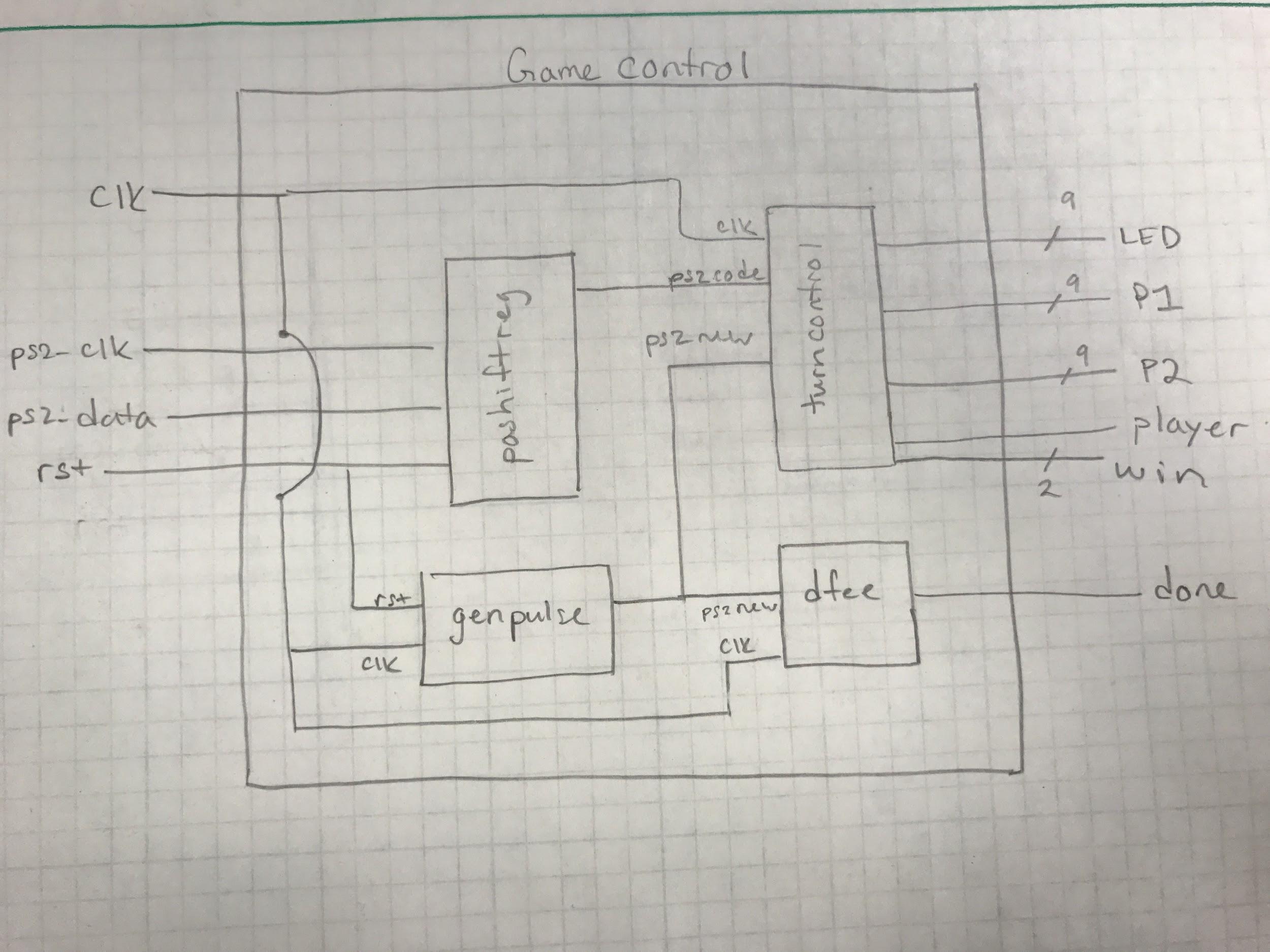
Top Level Block Diagram



Vivado Schematic

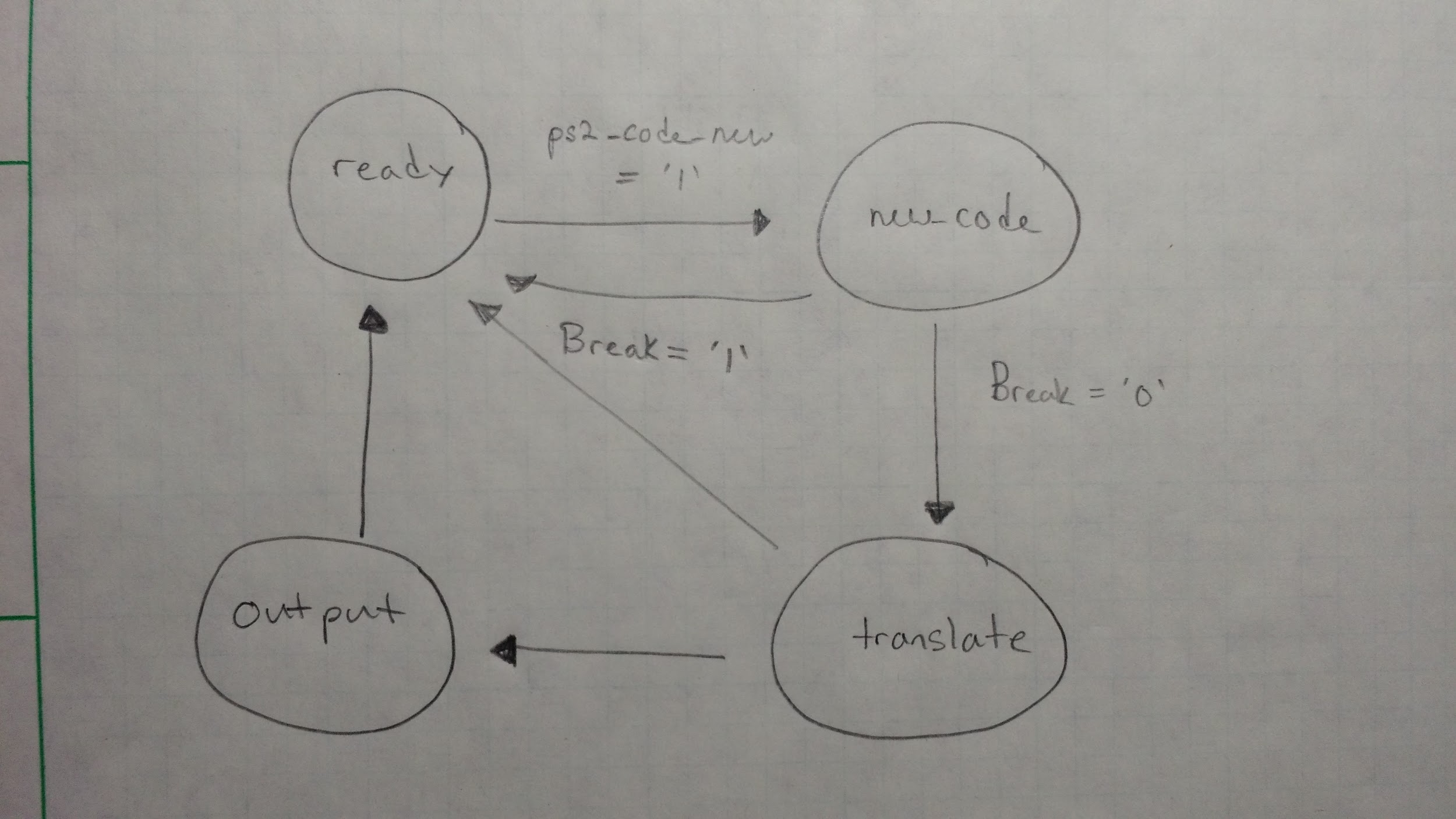


GameControl



The GameControl module is split into receiving keyboard inputs and controlling the player's turn. The shift register and genpulse both handle taking in inputs from the keyboard and passing the ps2\_code and ps2\_new flag. The majority of GameControl focuses on the TurnControl module. TurnControl handles the Tic-Tac-Toe logic by taking the ps2\_code for each player and keeping track of where they placed each game piece on the board using the P1 and P2 signals. Additionally TurnControl manages switching between the player's turn after they select their game piece and displays what cell on the board was pressed using the LEDs. Lastly, dfee uses the ps2\_new flag to to output a done signal to a LED.

TurnControl State Diagram



TurnControl uses a state machine with 4 states: ready, new code, translate, and output.

*Ready*: The ready state waits for a new key press and advances to new code once the new ps2 code flag is received.

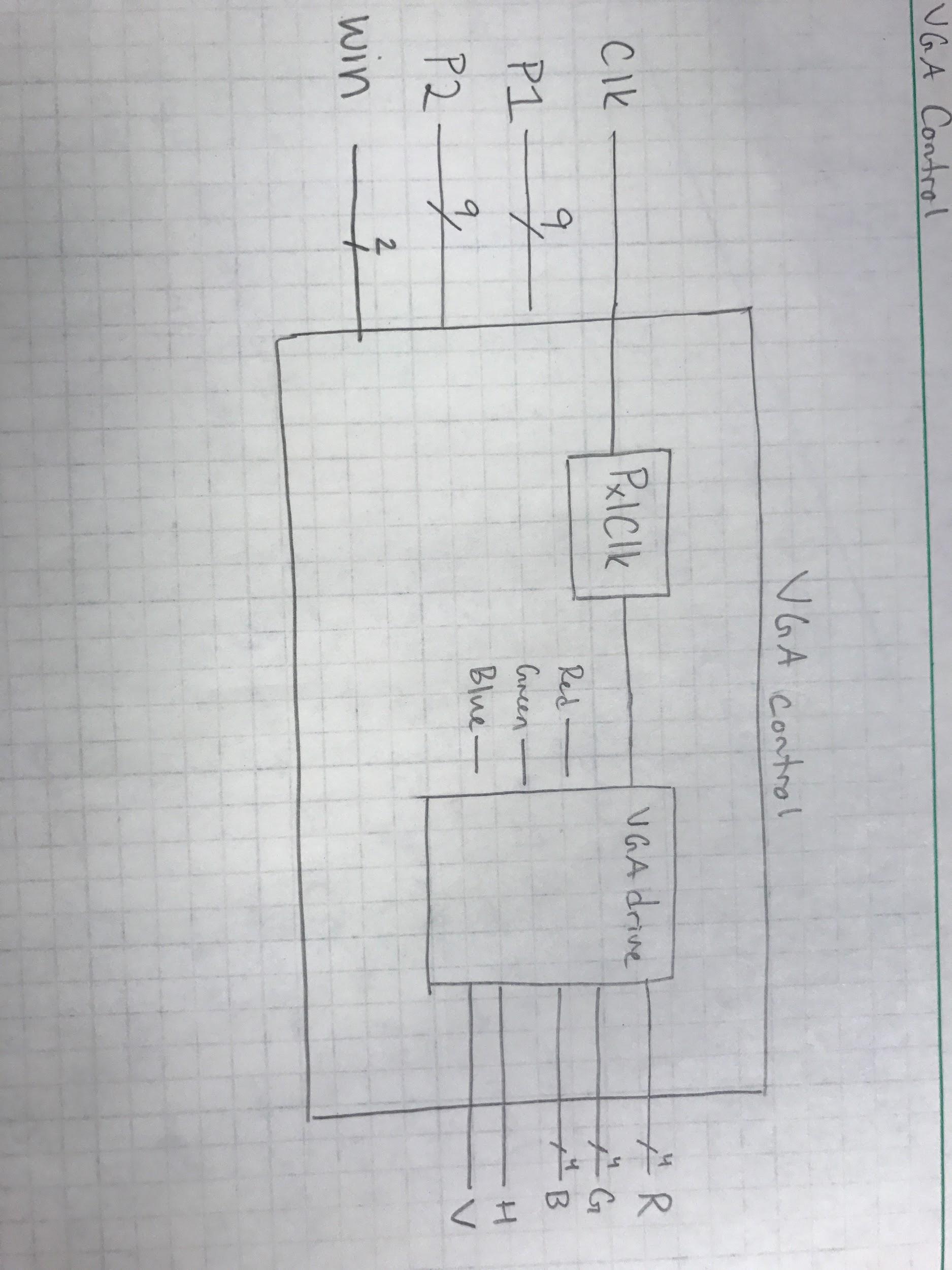
*New Code*: Checks for breaks and continues to translate if the ps2 code is good or moves back to ready of there is a break.

*Translate*: Converts the ps2 code to ASCII and sets the output LED signal. Falls back to ready if there is a break or else advances to output.

*Output*: Checks to see if the gameboard location is open. If available, saves the cell in the player's array and alternates the turn going to the ready state for the new player. If the cell is unavailable, moves back to the ready state waiting for a valid key press.

By saving each player's game piece location into signal P1 or P2, we can then use these signals in the VGAcontrol module to draw them onto the gameboard.

VGAcontrol



VGAcontrol takes in the player 1 and player 2 signals in order to know where to draw each game piece Each bit of P1 and P2 represent a grid on the game board and VGA control checks checks each bit to see if there is a game piece in the cell.VGAcontrol the draws both the game pieces and the game board using the VGAdrive module to send the RGB, Hsync, and Vsync signals to the monitor.

**VERIFICATION**

Developed test cases for every working feature of our design:

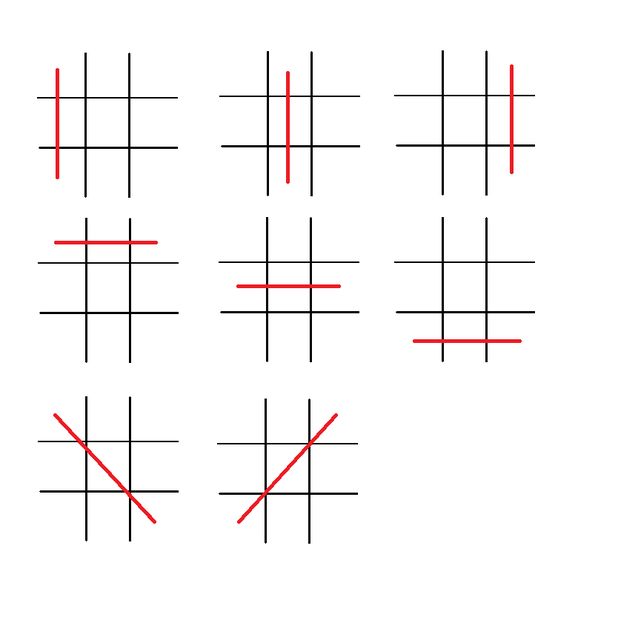
*Inability to overwrite other player’s piece*

Pressed same key in keypad repeatedly for both players’ turns. LED mapped to the player’s turn signal did not flicker, nor did the piece placed on the game board change color. This means that the key was not registered and our test PASSED.

*Inability to press other keys in keyboard (switching player’s turn)*

Pressed random keys in keyboard that are not the required 1-9 keypad keys. Expected the player’s turn to not switch nor any game piece mysteriously appear on game board. Our program successfully PASSED this test.

*Detect when a player has won the game*

Try all eight occasions where a user wins for both players (as displayed below). Grid lines should turn the color of the player who won. Our application successfully passed this test with the exception of the other player “winning” after not resetting the game. This causes the grid lines to turn the color of that player’s instead.

Win conditions for either player on game board

*Erase game pieces when reset button is pressed*

Ensure that the game runs smoothly when reset is not on. When on, the game board must be cleared with the exception of the gridlines. Our program successfully PASSED this test.

*Examine LEDs when pressing a number in keypad*

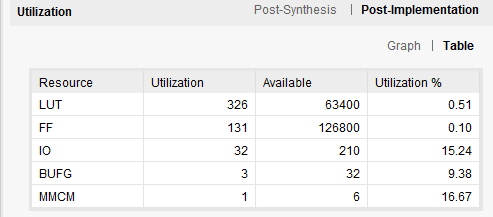
LEDs should display the number assignment once pressed in keypad. This validates the user’s decision. If other key has been pressed, LEDs do not change nor does the player’s turn LED. We tested these 3 different scenarios in this section, and our program PASSED.

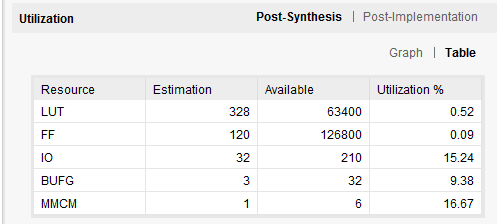
Overall, our program proves itself robust. We prepared backup scenarios for misuse of keyboard or other features. If all fails, the reset feature also provides a safe restoration of the program.

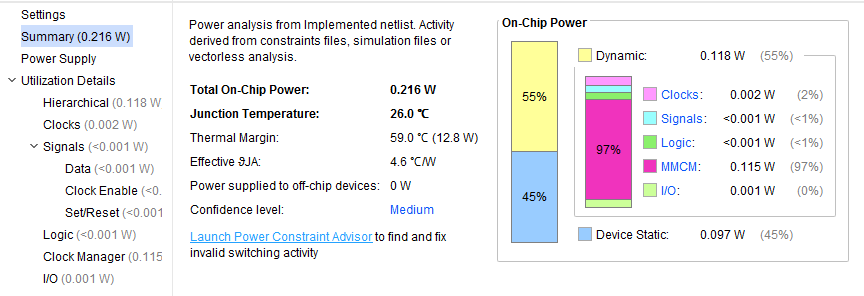
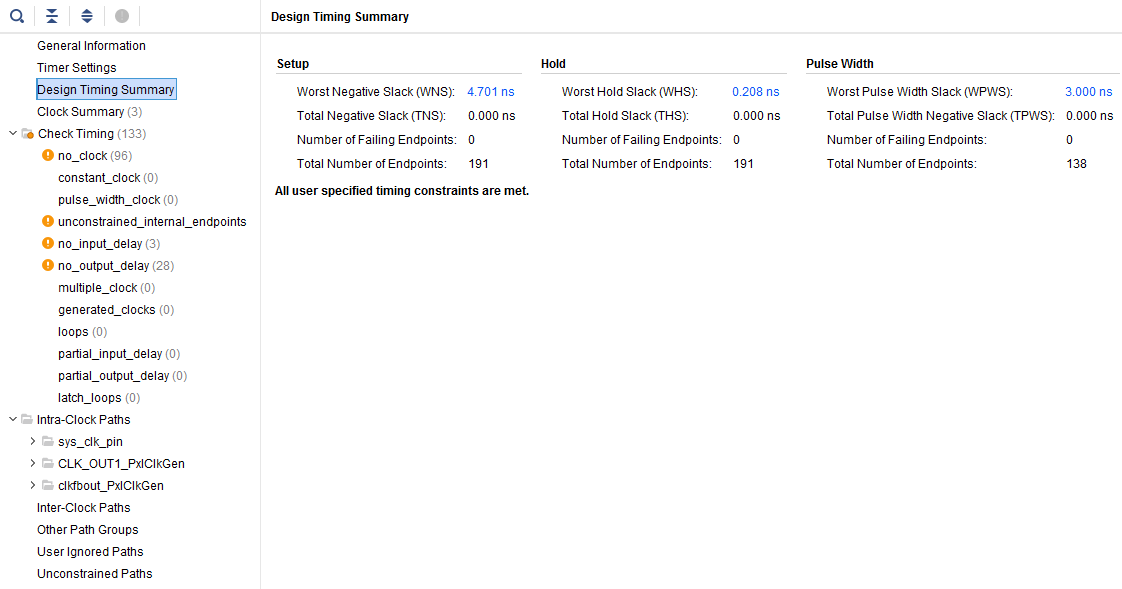
**PERFORMANCE**

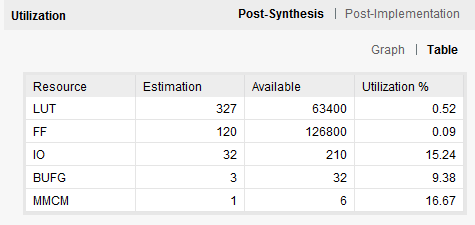
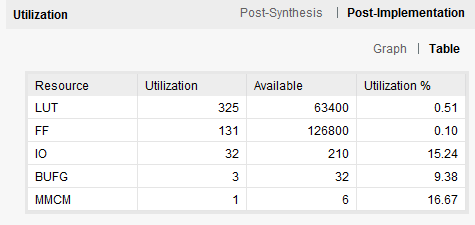
The following snapshots were taken from our synthesis and implementation reports. We are comparing “RuntimeOptimized” with “AreaOptimized - High” just to provide a second perspective. From these reports we concluded that our design has been sufficiently optimized in terms of resources used, area used and power consumed. In terms of delay, however, AreaOptimized synthesis managed to pull 0.2 ns from negative slack and hold slack.

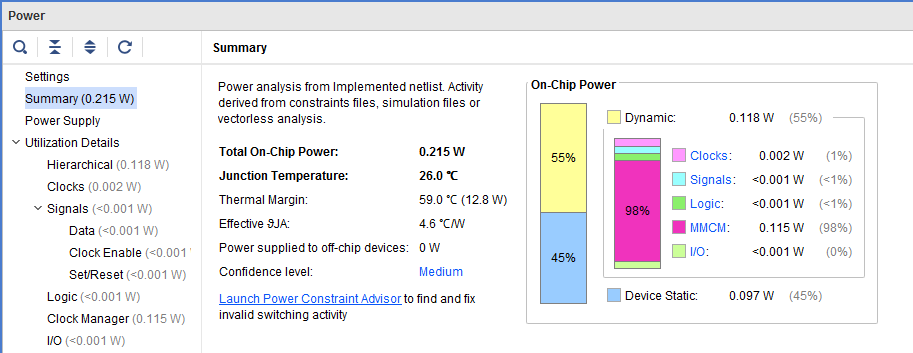
We believe that by tweaking the VGA controller and PS/2 interface given in class, we managed to get rid of a lot of overhead. For example, instead of translating the whole make code we just focused on the 9 keys we cared about from the board. Another thing that could be redacted from our design is the usage of LEDs since it is redundant from the VGA display.

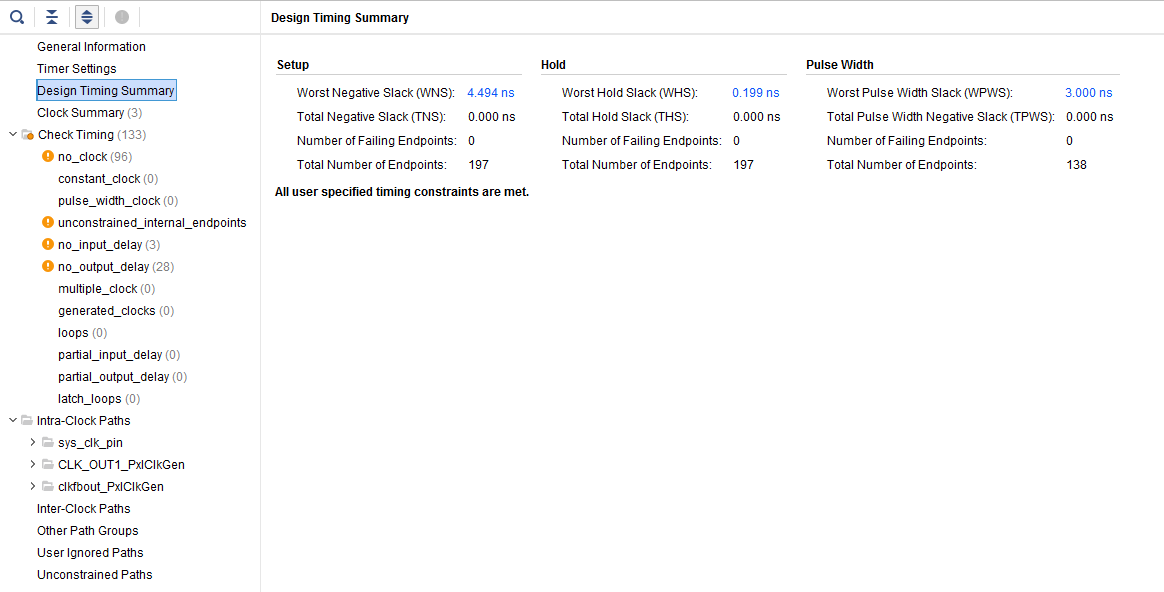
**Runtime Optimized:**





**Area Optimized - High:**





**CONCLUSIONS**

A lot of concepts learned in the lecture were implemented into this two-player Tic-Tac-Toe game. For example, the VGA and PS/2 interfaces introduced in class were major components in our design. These were tweaked to reduce the overhead and meet our functional specifications. Overall, it turned out to be a robust program that can withstand many failure conditions. Possible improvements in the future would be adding an AI feature with a good enough algorithm to beat a human player as well as a score tracker in the 7-segment displays for both players. If developers are bothered by the use of colored blocks, X’s and O’s can be implemented with a ROM array as well. Generally, we are pleased with our project and how well we sharpened our VHDL skills.

**APPENDIX/ATTACHMENTS**

*Top-module:* TicTacToe.vhd

*Internal modules:* GameControl.vhd, VGAControl.vhd, VGADrive.vhd, TurnControl.vhd, pashiftreg.vhd, dfee.vhd, genpulse.vhd and the PxlClk IP.

*Constraints file:* Nexys4DDR\_C.xdc

**MEDIA ATTACHMENTS**

· *Copy of your presentation slides, embedded PowerPoint file.*

<https://docs.google.com/presentation/d/13ehdYYK4q8wTVQuwQ8LOrP5dST_PTjMxneJwNvUJTNc/edit?usp=sharing>

· *List of attached pictures or videos for your project*

Demonstration video

<https://youtu.be/OSVrUFwqetc>

**REFERENCES**

R. Frawley, “FPGA Tic Tac Toe,” *Instructables*, 11-Oct-2017. [Online]. Available: http://www.instructables.com/id/FPGA-Tic-Tac-Toe/. [Accessed: 04-Dec-2017].

P. P. Chu, *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version*, vol. 3. Cleveland, OH: Cleveland State University, 1999.

“Nexys 3 ™ FPGA Board Reference Manual,” *Digilentic*, 11-Apr-2016. [Online]. Available: https://reference.digilentinc.com/\_media/reference/programmable-logic/nexys-3/nexys3\_rm.pdf. [Accessed: 04-Dec-2017].

B. Cuzeau, *Simple PS/2 Interface*, vol. 1.1. ALSE, 2003.