

Pac-Man

Rated E for Everyone

Andrew Carle, Peter Isho, Zachary Platte, Rohit Timmagi

Electrical and Computer Engineering Department

School of Engineering and Computer Science

Oakland University, Rochester, MI

amcarle@oakland.edu, pdisho@oakland.edu, zdplatte@oakland.edu, rtimmagi@oakland.edu

Abstract—The purpose of this project is to reconstruct a game similar to Pac-Man using the VGA port on the Nexys-4 Board. Throughout this project, the methodology behind implementing the VGA display has been immensely emphasized, in doing this it is possible to see how placing objects on the display and interacting with these objects is done. In a future project, it is recommended to attempt to find a way to implement an easier way to handle the collision with the Pac-bits to be able to add more to the game efficiently.

I. INTRODUCTION

Pac-Man was released in May 1980 and since that time it has been ported to many different types of systems. This project will run a two player Pac-Man game on an FPGA using a keyboard, FPGA buttons, and a VGA display. The essential aspects of the project are broken down into three parts which include receiving an input signal from the keyboard and buttons, utilizing that signal to move Pac-Man, and then outputting the corresponding pixels to a VGA display. Of course there are many more parts to this design, but these three seem to be the most essential for now. The working game will consist of one level with a win or lose screen, and two players. The input from the keyboard will move player one and the buttons will move player two. All of this is constantly being shown on the VGA display.

II. METHODOLOGY

A. Movement Control

Since the game allows for two players each player is moved with a different selection of buttons. Player two is moved using four of the five buttons of the FPGA whereas player one is controlled using inputs from a keyboard. Each of the player's characters is moved with a formula that allows for a movement of 15 pixels per second while the corresponding direction input is held down. There are interactions within the game that allow a player to speed up or cause a ghost to remain still for a short period of time. Each of these are handled within a process triggered by the collision between Pac-Man and the corresponding Pac-bit.

Using a keyboard with the provided USB port on the FPGA isn't that difficult. Within the FPGA there is a system that changes that USB signal into a PS/2 signal which is much easier to handle. The only things to deal with are the

PS/2 clock and PS/2 data, where both are explained in detail on the datasheet. The data that is coming into the FPGA will consist of a start bit, eight data bits, a parity bit, and a stop bit. Since each key has unique data bits it's easy to handle them in the system.

The data coming from the keyboard will be what moves Pac-Man. If the up-arrow key is clicked and the move is allowed, Pac-Man will be moved up by a set number of pixels. The top-level Data Path diagram from movements can be found in Figure 1. Formulas 1-4 are the VHDL code used to rotate the entire picture of Pac-Man right, down, up, and left respectively so the opening is always facing the direction he is moving. These formulas are essentially *rotating* the normal image which is facing right. Normal access of the index of any 16x16 array is found by using the first equation. Rotating the image so that it is essentially facing down is the second equation. The third and fourth equations relate to facing up and left, respectively.

$$(16 * (Vc - y) + (Hc - x)) \quad [1]$$

$$(16 * (Hc - x) + (Vc - y)) \quad [2]$$

$$(16 * (Hc - x) + (15 - (Vc - y))) \quad [3]$$

$$(16 * (Vc - y) + (15 - (Hc - x))) \quad [4]$$

These equations are necessary for accessing each objects memory, without them there is no guarantee that the image will be displayed correctly. This is because the memory is sequential, and without accessing addresses, will output whatever data is next. This would be fine if the image was displayed, and then the rest of the VGA display was displayed, however, this isn't the case. The VGA outputs pixels' row by row, so the memory needs to be accessed at different times. This was a big issue in the beginning of the project, so instead of accessing memory, a yellow block was a substitute for an image. It wasn't until the last few days that the images were implemented in the game.

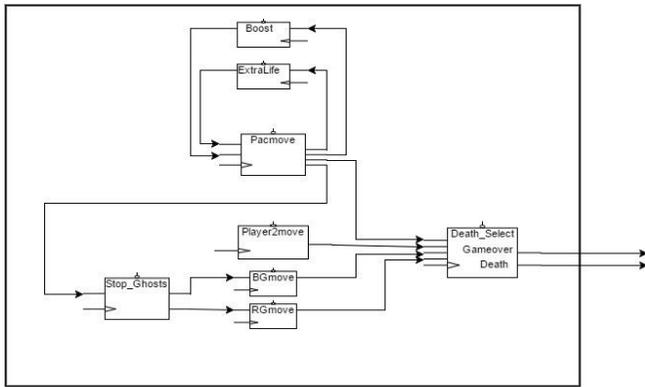


Figure 1. Controllable Movement Data Path

Each of the ghost's movements are controlled using FSM's with the diagrams being found in Figures 4 and 5.

B. Display Control

The game is displayed using the block `vga_ctrl_12b` in memory mode. The top level can be found in Figure 3. With this implementation, the walls, Pac-bits, original ghost locations, and the beginning placement for Pac-Man are placed using specific pixel placements. Whenever a certain pixel location matches a condition that was stated, the necessary data will be placed. This allows for an easier interaction between Pac-Man and the objects, whether they are walls or Pac-bits. Pac-Man, along with all the other non-wall objects were implemented using the conversion file in MATLAB to convert the pictures to text. The text is then read into the VHDL code to bring the original characters into the game. Pac-Man during movement is constantly rendering in the new location it moves to, this is the reason that the formulas executed in the prior section are put in place. The ghost displays are each done with an identical process to Pac-Man.

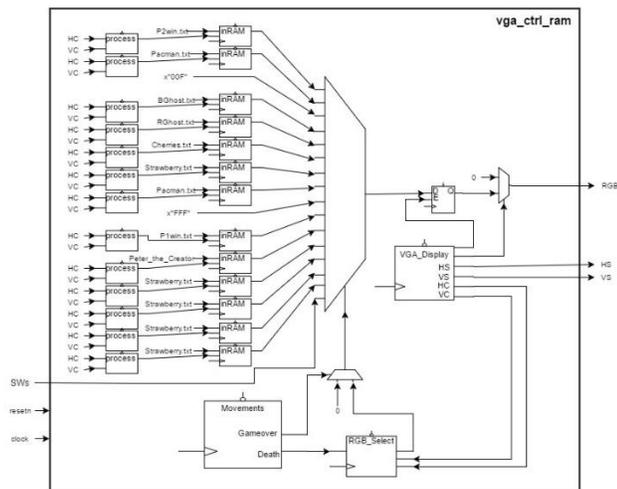


Figure 2. VGA Control Data Path

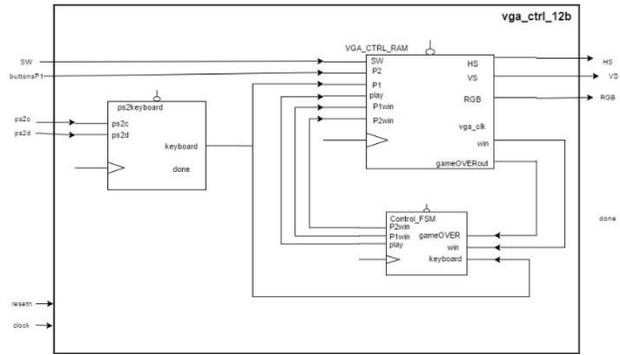


Figure 3. VGA Control Top-level Data Path

C. Collision Handling

Placing the pixels through memory allows for a collision between Pac-Man's and the Pac-bits, walls, and ghosts based on the location of Pac-Man in comparison with each of the objects. Every object starts at a default position, for the non-moving objects this position is crucial in terms of the interactions between said object and a movable object. Each of the Pac-bits has a signal that controls whether the bit remains on the map or not, by default this signal is set to '0' and when the location of Pac-Man collides with the location of the Pac-bit the signal is set to '1' thus removing it from the map. Once the object is removed its location changes color to that of the background. All the Pac-bits are handled in this manner, with the only difference between a Pac-bit that grants the user a "power-up" and a regular one being the location on the map.

The power-ups use a method that is utilized throughout the game. As soon as a collision occurs between Pac-Man and one of the power-ups, a signal gets a '1'. In the case of the speed boost, this signal will activate another signal which will go back into the component that controls Pac-Man's movement. It will allow Pac-Man to move faster for a short amount of time, around 5 seconds. This method of sending a signal back into the component is used when Pac-Man collides with a ghost. That collision will send a reset signal to all movement components which resets them to their default position.

Since the signal that controls the Pac-bits being on the map is Pac-Man's location each time a ghost collides with a Pac-bit it is ignored and the ghost continues moving. The collision that occurs with Pac-Man and a ghost causes Pac-Man's location to be set back to starting position while removing one of the lives from the upper righthand corner of the screen. Arguably the most important collision is with Pac-Man and the edges of the map, without this Pac-Man would be able to "leave" the screen causing issues with the game. To implement this the pixels along the edges of the screen are treated as walls so that Pac-Man and the ghosts cannot move through them.

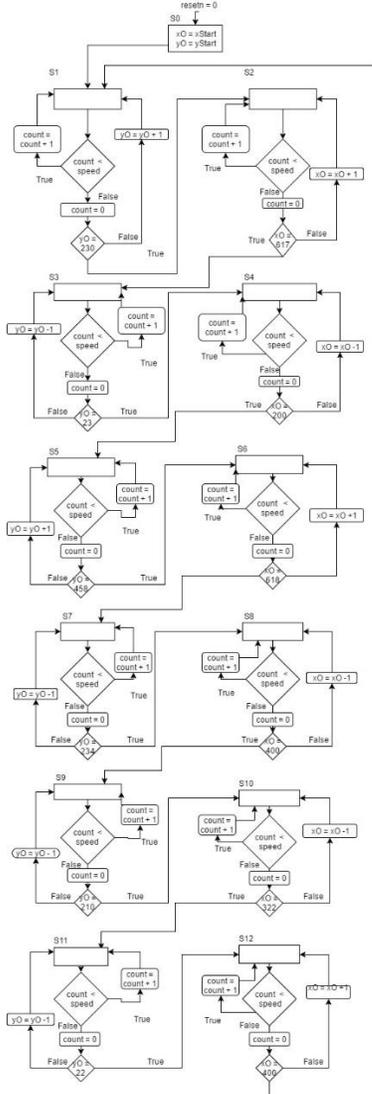


Figure 4. Ghost 1 Movement

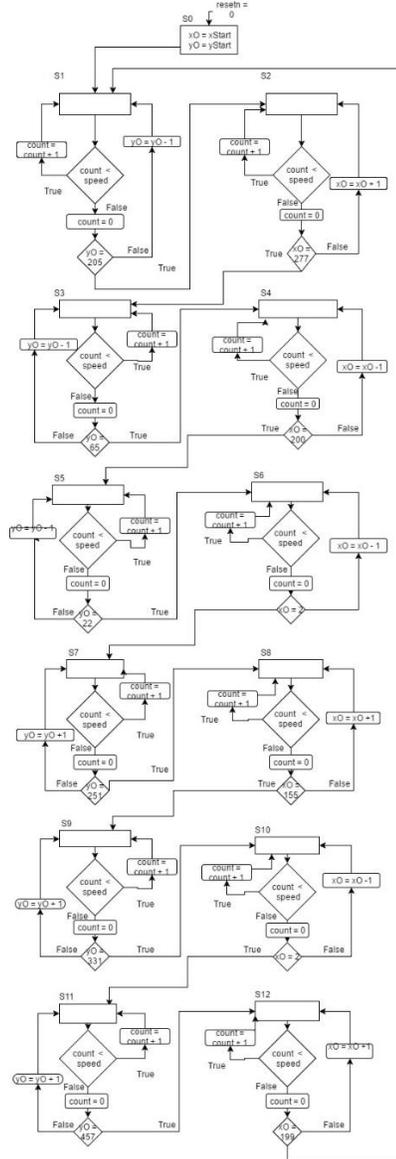


Figure 5. Ghost 2 Movement

III. Technical Approach

The hardware is the Nexys-4 Artix-7 FPGA board and the software is Xilinx Vivado. There are two ports that are utilized: the USB input port and the VGA output port. USB input is for the keyboard and the VGA output is for the display. In addition to the USB input the buttons on the Nexys-4 Artix-7 board are used to move the second player to make the game more challenging since player two can work with the ghosts. In order to properly communicate with each of these ports [1] is careful studied to understand the interactions they have with the board.

The USB to PS/2 component is tested by simply verifying that the LEDs on the Nexys-4 board represented the corresponding hexadecimal number. This hexadecimal number is then mapped to the directional controls in the

game. The VGA component was tested using a .txt file and outputting its signal to a computer monitor. MATLAB is used to convert the images to text files that can be read into the VHDL code, then they are placed within the game. This is done using the file given in class with some changes made to it to fit the size of the images being imported to the game.

The reset button is used for resetting the entire game. The internal clock of the FPGA will be used for all instructions done in the design of this game.

IV. RESULTS

The results of testing the USB input signal were successful. The two-bit hexadecimal number represented by LEDs on the Nexys-4 DDR board correctly corresponded to each key from the keyboard and buttons. The signals of the arrow keys were unknown before this test, but now are recorded so that the arrows may be used to control Pac-Man movements.

The VGA tests were also successful, but with some issues. The VGA component was capable of outputting the correct shape and color of the .txt file on the computer monitor. After adjusting the equation for how and where the .txt file was to be placed on the monitor, the image became distorted with rows, or stripes, of the color. Thoroughly testing this further led to the images appearing in the appropriate manner, in addition to being able to move without the images becoming distorted.

The overall result of this game is the user being able to control both Pac-Man and the player-controlled ghost via the keyboard controls and button controls respectively. How Pac-Man moves depends on the key or button the user pushes. As Pac-Man moves he eats up the Pac-bits. The game ends when either Pac-Man loses all his lives or if he collects all the objects. Depending on which player wins, a corresponding win screen is displayed.

Originally, hitting the space bar after a win screen is shown was supposed to be implemented. The issue with that was unknown, and there wasn't enough time to debug it. Rolling back to the original code allows the user to hit reset whenever a game is complete.

CONCLUSIONS

The state machines used to control the ghosts' movements seem slightly complicated, however when analyzed closely they are extremely similar which made for an easier implementation. Things we learned would be how to code the FPGA to be able to control Pac-Man's movement for both player one and player two via the keyboard and buttons and displaying the game onto a monitor via VGA. We also better improved our skills on how to implement state machines to deal with various types of movement within an environment such as video games. The complexity of the project wasn't extreme, but once a solution was found for a problem it required a lot of time to complete. There was a lot of little details that needed to be kept track of, otherwise many things could go wrong. It was a fun project to work on.

References

- [1] Diligent, Nexys4 DDR FPGA Board Reference Manual, Rev. C, pp. 12 & 14-17, September 2014.
- [2] Llamocca, D. (2017). VHDL Coding for FPGAs. Retrieved April 22, 2017, from <http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html>