Pac-Man Game Design

Created by: David Soofian, Dagna Harasim, and Charles Finkel

Introduction

Our project has been designed to take everything we have come to learn so far, from our individual labs and add our own personal flavor to instrument our knowledge of the FPGA, its components, and the VHDL code.

We have designed a video game with no previous base, without integrating a ready-made emulator or using other people's work. We feel that creating the graphics and the game console will be a formidable task for our team and a learning experience for us all.

Our project relies on the integration of VHDL and C. With both languages working, together we will be able to handle inputs from the keyboard, RAM, ROM, and output to the video display.

The Game

The game itself will be a modified version of the original Pac-Man game. The graphics will be similar but the game's objective will be a little different. Instead of having aliens chase after Pac-Man and be a constant obstacle to him, our game is a race for time with a different type of obstacle. Each player will be timed on how long it takes the player to 'eat' every pellet on the screen while guiding the character through the maze we have created. The essence of the game lies within the best timing a player can make around the maze and devour all the pellets. The added obstacles are randomly opening and closing doors, leaving the game to be a more complex mix of luck and ingenuity.

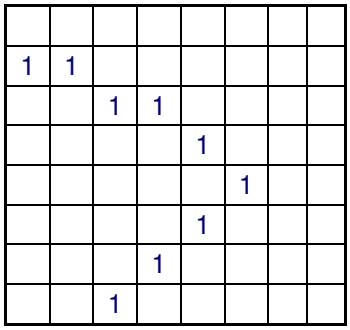
VHDL

In our labs, we have learned to implement a live video display. We were able to store our character sets and graphics on the XSB's RAM. This allowed us to save valuable time in drawing each character for our typewriter one by one by calling pre-existing mapped graphics.

We will generate our graphics in an array representing 8x8 pixels, each coded in hexadecimal. Each 8x8 pixel array on the screen will be represented by two strings of nine hexadecimal values. Each group of nine values will represent a mapping in one color. The first term in the string will represent the value for the color of the following pixel map. We will then proceed to read the second string of values to overlap the same 8x8 pixel space on our screen. The second string will represent the second color, with no pixels overlapping the first layer. This approach allows us to create an 8x8 pixel array with two colors. For convenience, the game engine will read 18 values at a time, since we are allowing for two colors for each 8x8 square.

1	00	C0	30	08	04	08	10	20

2	00	00	C0	F0	F8	F0	E0	C0



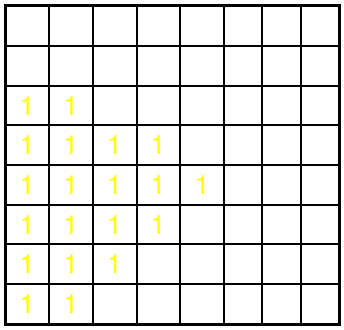


Fig. 1 Example of 1 8x8 pixel block. This is the upper right block of Pac-Man's figure. The outline is in blue color, and the filling is in yellow.

This format should save a considerable amount of space in our ROM. We considered another option of having a color preceding each pixel, which would occupy almost three hundred and eighty times the amount of memory. The version we chose also is neater and contributes to the overall simplicity of the design.

We will be implementing the mechanics and hardware from our lab five assignment, which created a video typewriter using the ROM and a complicated VHDL reading system.

We chose to create Pac-Man as a 16x16 pixel character. Making our character 8x8 pixels would take away from our graphics capabilities, therefore we enlarged Pac-Man by giving him a 2x2 square of 8x8 pixel maps. Keeping in mind that Pac-Man will be facing different directions on each path he takes, graphics must be created for each direction he may take. By splitting Pac-Man up into a 2x2 square, we now have more of a fluid motion in his journey.

Instead of Pac-Man's graphic being moved to the next character space we can now move his character half of its character space at a time, which gives him more of a lifelike movement through our maze. We will leave it to our game engine to construct Pac-Man; with a table of characters, our synchronization should be flawless.

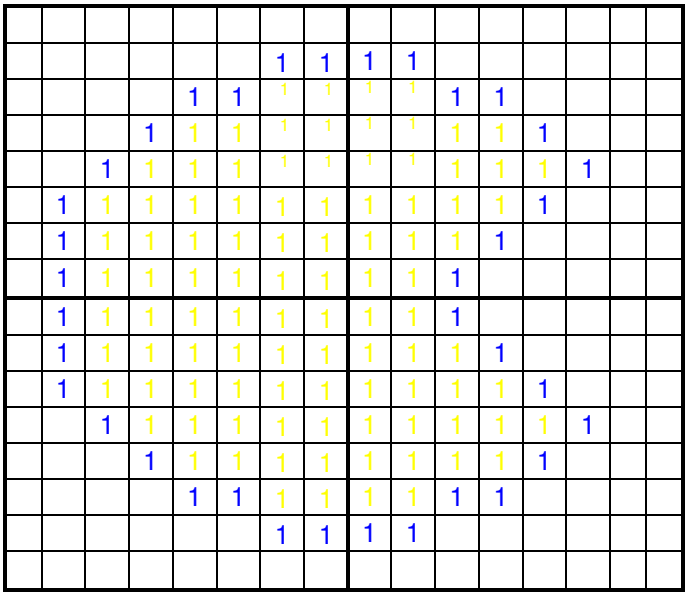


Fig. 2. A sample graphic of Pac-Man.

The actual game "board" will consist of two layers, first with the permanent part of the maze, and second with the pallets and the randomly appearing doors. We have decided to keep the method previously used in the lab to display the puzzle, but with modification. Instead of clearing the screen by setting all of the pixels to black, we will clear the screen by redrawing the maze. The maze, without the pallets, will essentially be the background of the game. This way all the constant components of the maze will stay on the screen, and only the doors, pallets, and Pac-M an will be "redrawn" or erased from the screen as needed. A map of the maze will be read by the game engine, to allow it to "know" where all the permanent obstacles are, and therefore make it easier for us to program Pac-Man's behavior when he encounters a wall.

The second layer with the pellets, which Pac-Man eats for points and the random doors will be laid after the maze. The pellets will be cleared as Pac-Man's character eats them by overwriting their pixel value with Pac-Man's own character values. We will create a random number generator to implement opening and closing doors in specific spots in order to add more difficulty to the maze. As each door is opened or closed, we will update our data to allow the game engine to know where the current walls are.

The Game Engine – C

The game engine will be written all in C. The game will be controlled by a clock, which will in essence be the frame rate of the game. At each clock tick, Pac-Man will move one character.

We have not yet decided on whether we want to include ghosts. If we do, they would probably have a very rudimentary AI scheme since none of us have had a formal education in computer AI.

Input will be taken from the keyboard, and we will allow only four keys to do anything in the game (left, right, up, and down). After each clock tick, the most recently suppressed key will determine the direction Pac-Man will move. For example, if multiple keys were hit within the duration of one clock cycle only the mostly recently pressed key will register. Holding down a key will not be necessary, as Pac-Man will continue to move in the same direction, as he was the previous clock tick if no other key was pressed.

Game Block Diagram