

COMS4840 Design Document: Pac-Man

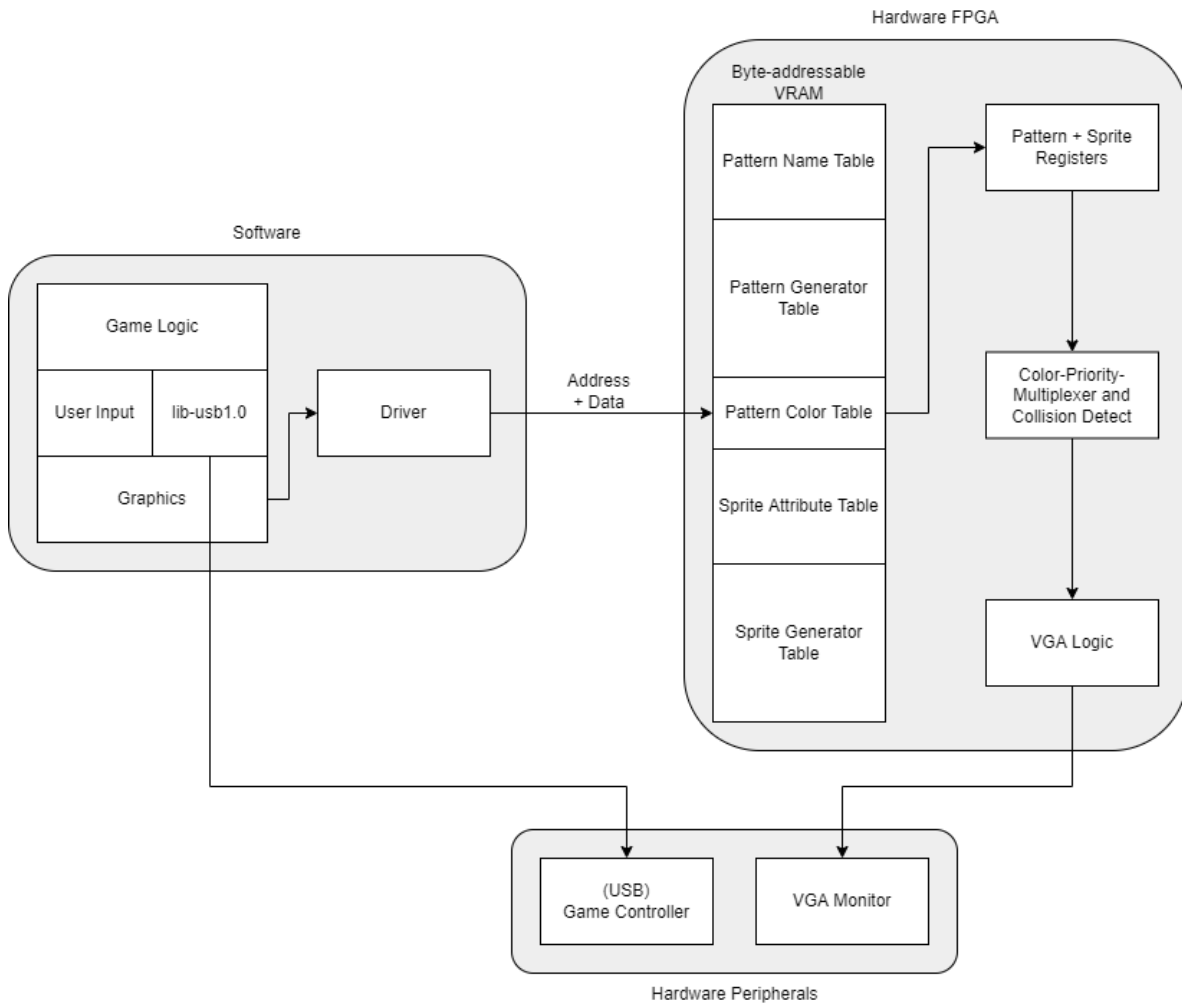
Jerry Lin (ml4686), Leo Qiao (flq2101)

April 1, 2022

1 Introduction

In this project, we will re-create Pac-Man on the FPGA. Pac-Man is a simple and popular game developed in the 1980s where the player controls a character that can move in a maze with the goal of eating all the dots in the maze while avoiding ghosts. On a high level, the FPGA will be responsible for displaying graphics of the game while software will be written to control the FPGA display through a driver. The user will communicate with the software using a game controller through the USB protocol.

2 System Block Diagram



3 Algorithms

Hardware:

The primary algorithm required for our Pac-Man implementation would be the logic to generate the graphics. We plan to use a tile-and-sprite graphics generator, given the regular features of our game graphics. The method in which we will implement tile-and-sprite graphics will be inspired from the graphics architecture of the TMS9918. For the stationary graphics elements such as the maze and the food, we will use tiles to display them in a fixed grid. A pattern name table will hold the memory addresses of the tile to be displayed for each position in the game grid. The pattern generator table will hold the actual pixel for the tiles.

For a given sprite, we will have preloaded the drawings into the FGPA BRAM as with the tiles. This will be represented as a sprite generator table. We will have a sprite attribute table that stores the memory address and location of sprites being displayed. To draw a sprite, the display mechanism operates line-by-line. At each vertical position, if the sprite will be visible, the sprite controller will store the sprite's horizontal position, count down for each horizontal tick and start displaying pixels of sprite once the counter reaches zero. The controller will start the drawing process before the required sprite position (determined by software that will be discussed later) has been reached to take into account memory access and other synchronous logic delays. The controller will also take into account the size of the sprite in order to determine when the drawing is finished. One of many approaches for implementing the controller would be using an FSM to track the status of the sprite drawing.

There will be an implementation of a shared memory bus with arbitration since multiple different sprites will need to be drawn. Additionally, if our memory usage exceeds the maximum on the FPGA, we may need to consider implementing a color lookup table and corresponding logic in order to save memory storing pixel color values.

Software:

The goal of Pac-Man is to eat all the dots located in the maze while avoiding contact with the four ghosts. There will also be four power pellets located within the maze, which if eaten, will grant Pac-Man the ability to eat the ghosts and send them back to the center box. Pac-Man will have a total of 3 lives, and the highest score will be tracked and displayed at the top of the maze. The game will be simplified to only one level.







The behavior for the ghosts would primarily be to chase Pac-Man. We will implement a variation of the AI for the ghosts, where two of the ghosts will be always chasing Pac-Man, while the other two ghosts will alternate between chasing Pac-Man and running away from him. The software will track the position of all the ghosts and Pac-Man using an absolute Cartesian coordinate system. For ghosts to follow Pac-Man, they will calculate the horizontal and vertical distance to Pac-Man and move in the direction closest to the distance vector. For a ghost to flee Pac-Man, they will head in the direction closest to the opposite of the distance vector.

4 Resource Budget

Table 1 shows the on-chip memory usage for the FGPA. The borders for the maze will require 4 bend pieces and two straight pieces, resulting in a total of 6 different images. The dots and power pellets that Pac-Man eats will require 2 images. The four ghosts will require the most amount of memory. For each ghost, we will need 4 models to account for the differences in their eyes depending on the direction they are traveling. Additionally, each ghost will need two different models to simulate movement of their capes. Finally, the blue ghosts and their animations will require a total of 4 images.

The Pac-Man itself will be relatively simple, requiring only two images to model the movement of the mouth. One of the models could be reused to indicate the number of lives left. Displaying the "HIGH SCORE" caption will require 9 images, and so does numbers 0-9 to represent the score. On-chip memory should be the most limiting factor for our design. We do not expect other resources on the FPGA to be strained.

Table 1: Estimated on-chip memory requirements to store bitmaps

Graphics Type	Name	Graphics	Size (bits)	# of Images	Total Size (bits)
Sprite	Ghosts		16 x 16	4 x 4 x 2 + 2 x 2 (36)	345600 221184
Sprite	Pac-Man		16 x 16	2	12288
Tile	Borders		8 x 8	6	9216
Tile	Food		8 x 8	2	3072
Tile	Letters		8 x 8	9	13824
Tile	Numbers		8 x 8	10	15360
Total Memory Requirement (bits)					274944

For the Pattern Name Table, we are on a 640x480 VGA screen with 8x8 tiles, so it will require $(640/8) * (480/8) = 80 * 60 = 4800$ bytes, or equivalently, 38400 bits.

For the Sprite Attribute Table, we will be displaying 5 sprites (1 pac-man and 4 ghosts). Assuming that each entry in the table requires 4 bytes, the entire table would need $5 * 4 = 20$ bytes, or equivalently, 160 bits.

In total, the total amount of VRAM will be roughly $274944 + 38400 + 160 = 313504$ bits. This should easily fit into the 256KB BRAM on the FPGA.

5 HW/SW Interface

The HW/SW interface would resemble a RAM interface. Conceptually, the FPGA component puts the pattern/sprite related data inside the video RAM (VRAM). The exposed interface would be a byte-addressable VRAM, where different addresses have special meanings (e.g. some range of addresses is the pattern generator table, another range of addresses is the sprite attribute table). For our pattern name table and sprite attribute table, we plan to use the same memory layout as the TMS9918.

From the software's perspective, the process of changing anything on the screen (e.g. adding a tile, changing sprite position, etc.) would just be updating the byte stored at certain addresses inside the byte-addressable VRAM. The software driver could hide the intricacies by providing user-space programs with a simple graphics API. To change the position of a certain graphics element from the software programmer's perspective, one could simply pass the address and the x and y coordinates of the sprites. With 4800 tile positions, we will likely need 13 bits or more for the coordinates depending on how smooth we would like movement to be. The exact sizes for address and coordinates will be determined further during implementation.