ECE 385

Spring 2024

Final Project

Final Project Report

Eric Vo and Roshni Mathew

Introduction

For our final project, we implemented the classic Pac-Man game. Our goal for the project was to recreate the iconic arcade experience using our FPGA. The game involves graphics rendering, user input handling, and basic game AI. Our design was built upon a combination of lab 6 and 7. We used a microblaze to interface with different peripherals, getting inputs from the keyboard and displaying the game to the monitor. We implemented the majority of our game logic in hardware using SystemVerilog. We used the C code to control the ghost movement and convert our score from hex based to decimal based.

When you press reset (r on the keyboard), our ghosts appear and the game can now begin. Then you can control the pacman using the w,a,s,d keys. The game ends either when the pacman has eaten all of the pellets or when the ghost kills the pacman three times. We also have included the special power up pellets that make the ghosts frightened and allows pacman to eat them.

Description of the system

We broke this project up into six phases. First we started by developing the maze and putting it on the screen. Second, we created the boundary conditions for the pac man so that it couldn't go through walls. Third, we added animations for the pacman and the ghosts. Fourth, we added pellets on the screen and made it so the pacman could eat them and gain points. Fifth, we developed the ghost logic which allowed them to move around the maze and target the pacman as necessary. Finally, we added logic to implement super power pellets, determine when the game was over, and display the score.

Our first phase consisted of maze construction and is what we demoed during our midpoint check in. All of this was done in system verilog. Our game is tile based such that there is a 30 by 30 tile grid where each tile is 16 pixels by 16 pixels. To map our maze to the screen we have a 30 by 30 array where each index holds values up to 8. Each number represents a different maze part. Then we have a mazeRom file which has different maze parts: a top left corner wall, a top right corner wall, a bottom left corner wall, a bottom right corner wall, a vertically straight wall, a horizontally straight wall, and a blank tile. In our color mapper, we calculate the tile we are on using DrawX and DrawY, then check our 30 by 30 array for the part, and then use the mazeRom to draw the corresponding part for that tile. The formula for calculating the tile is similar to lab 7, except instead of 8 by 16 pixels, our tiles are 16 by 16 pixels. We use the maze 3D array for checking the boundary conditions as well.

Our second phase was displaying the pacman correctly to the screen, controlling it with the keyboard, and limiting its movement to the walls. For displaying and controlling the pacman, we used lab 6.2 code and logic. Once the pacman was on the screen, we had to develop additional logic to check for walls. This logic was also tile based and in system verilog. First we limited the pacman movement so that it could only switch tiles once it had reached the center of the tile. This made sure the pacman was always centered correctly. When the pacman reaches

the center of each tile, we check what direction and what tile it's trying to go to next. Then we used the tile x and y values to determine whether there was a wall on that tile using our 30 by 30 maze array. Our maze array was set up such that a 0 represented no wall. If there is a wall on the tile, we then check if the previous movement direction is possible. If it is, we allow the pacman to continue going in its previous direction, otherwise we stop the pacman completely. For example, let's say the pacman was initially moving to the left and then we tried to move upwards. If there is a wall above the pacman but no walls to the left, the pacman will continue moving to the left. If there is a wall both above and to the left of the pacman, we stop the pacman. Then an additional feature is that if the pacman goes offscreen through the passageway on the right and left side, it will teleport to the other side of the screen.

Our third phase was adding animations to the ghosts and pacman. The pacman has three different states that we cycle through as it moves around the maze: closed, half open, and full open. Then it has a dying animation as well which we only cycle through when the ghost kills it. For the ghost, they have two states they cycle through as they move around the maze. Additional animations appear in frightened mode, when the ghost colors alternate between white and blue as the timer nears the end and when it dies. Both the ghosts and pacman also have different states depending on which direction they are moving in. The ghost's eyes and pac man's mouth move in the direction they are going. We used an FSM for both of the sprites. For the pacman FSM, the inputs to it are the direction of movement, whether it's dead or alive, and the current state. For the ghosts, the inputs are the direction of movement, whether it's frightened mode, whether it's dead or alive, and its current state.

Our fourth phase was adding pellets to the screen, which was also controlled by the system verilog side. Initially we wanted to have an algorithm that would go through the maze layout file and put a pellet on each tile with a 0 on it. Then we decided we wanted sections with no pellets on them, so we instead created a separate 30 by 30 array that represented whether there was a pellet or not on a tile. Our color mapper then used the pellet array and colored the center four pixels of each tile if there was a pellet on it. Then to calculate whether a pacman had eaten a pellet, we checked if the pacman was on a tile with a pellet on it, incremented the pellets eaten, and then removed the pellet from the array.

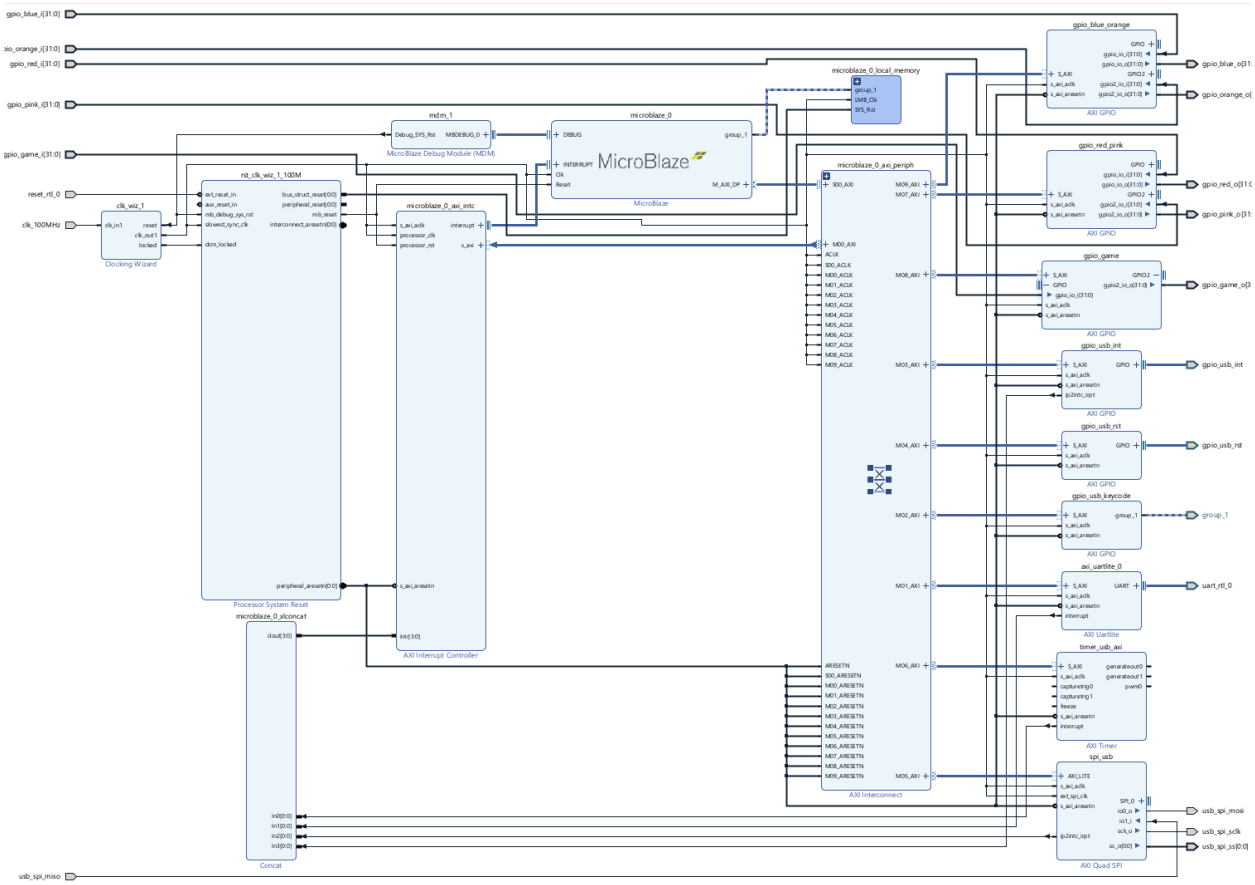
Our fifth phase was implementing ghost logic. This was done on the c side. In order to set this up, we added GPIO blocks to our microblaze. We added a total of three GPIO blocks. One block for sending general information over like the Pacman's tile coordinates. Then one dual port block that sent and received information regarding the red and pink ghosts and another dual port block for the orange and blue ghosts. Each GPIO port was set up to send 14 bits of data and receive 3 bits. Similar to how we controlled the pacman with the keyboard, we set our c code side to act as the keyboard. The microblaze would send the ghost's position, what state it was in, and whether it was on the center of a tile through the UART and then receive the direction of movement the ghost should follow.

On the C side, we have another maze layout denoting the walls within our layout. Then we have two functions: one for general ghost movement that uses the state of the ghost to determine where to go and a target tile function that moves the ghosts towards a specific tile.

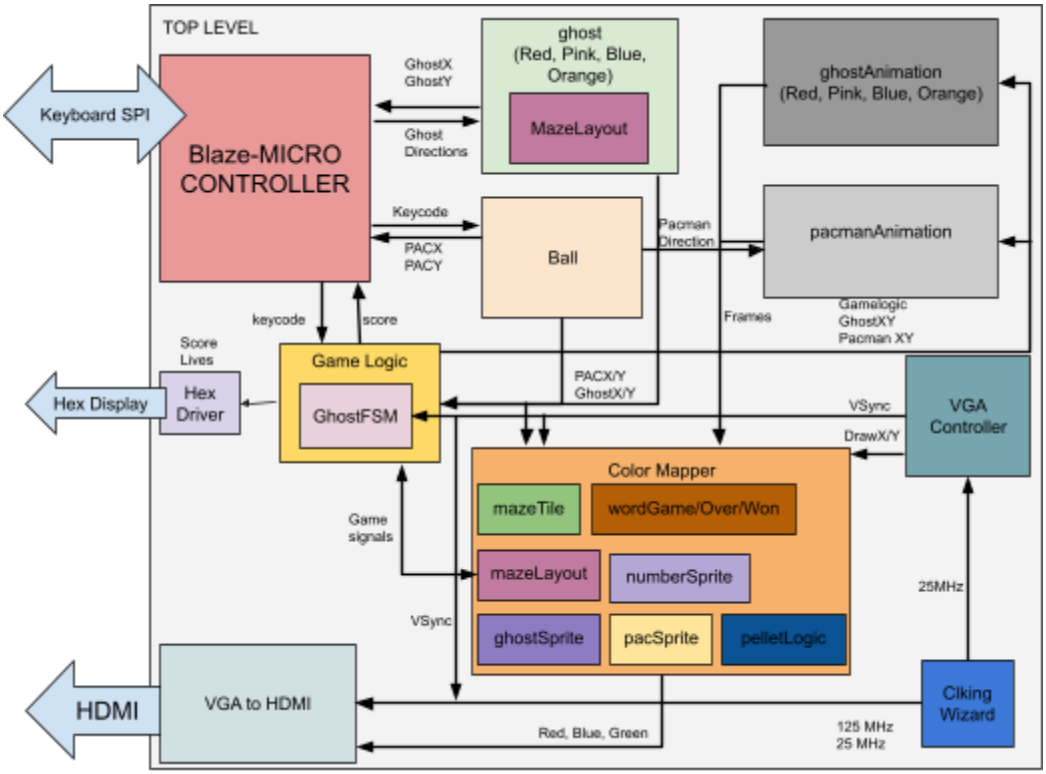
We designed our algorithm to check first whether a ghost was on the center of a tile. If it was in the center, it checked the distance from the target tile to the up, right, left, and down tile. If there was a wall in any of those tiles, it replaced the corresponding distance with a very large number. Then another rule imposed on the ghost is it can't reverse directions so it would also check current direction and change the distance of the opposite tile to a really large number. Then the next tile will be picked based on the minimum distance. The ghost movement function looks at the state the ghost is currently in (chase, scatter, frightened, or dead), and change the target tile accordingly. For chase, all of the ghosts' target tiles were set to the pacman. For scatter, each ghost had a target tile just outside each corner of the maze. For frightened, the target tiles were selected semi-randomly. We had a counter going in the main file and modded it with different numbers to "randomly" pick spots on the maze. Finally when it's dead, the target tile is the ghost house. On the system verilog side, the ghost.sv file would take the direction of movement as an input, double checks to make sure there isn't a wall in the direction the ghost is trying to move, and then update the ghost's location accordingly, similar to how our ball.sv file works.

Our final phase was adding additional features and final touches. We added super power pellets to our pellet array so that when the pacman eats one of them, the ghosts go into frightened mode and the pacman can eat them. Then added logic so that the game ends after the pacman dies three times. We also added end screens for game over when you win and lose. Finally, we changed our score from displaying on the hex drivers in hex to displaying on the screen in decimal form.

Block Diagrams



Block Diagram of the MicroController



Block diagram of Top level

Module Descriptions

Module: Ball.sv
Inputs:

- Reset, frame_clk, ghoseGone
- [7:0] keycode

Outputs:

- [9:0] BallX, BallY
- [2:0] direction, mazePart,
- [4:0] mazeRow, mazeCol,
- [3:0] check

Description: This module used the maze layout and updated our pacman's ball x and y coordinates based on inputs from the keyboard. We used the w,a,s,d keys to make the pacman move up, right, down, or left. Then it also checked for walls before updating the pacman's position to make sure it was limited to the maze. Finally, we added a check to make sure the pacman only changed direction when in the center of the tile. We defined the center of the tile as the 7th row and 7th column. This made sure the pacman was always centered on the tile. All updates to ball x and y were done on the rising edge of a clock.
Purpose: This module controlled the movement of the pacman and limited it so it couldn't move through walls.

Module: Ghost.sv

Inputs:

- Reset, frame_clk
- [2:0] directionInput
- [9:0] ghost_X_Center, ghost_Y_Center

Outputs:

- [9:0] ghostX, ghostY
- onPixel

Description: This module controlled the movement of the ghosts and outputted the ghost's x and y position in pixels. It took inputs from the software side regarding the direction we wanted to move in and then used the maze layout to verify if we could actually move that way. The logic is very similar to the ball.sv file. The ghosts could also only switch directions on the center pixel, and all motion was updated on the rising edge of the clock.
Purpose: This module was responsible for updating ghost's position based on the algorithm and direction instructions from the software side. We had four instantiations of this module: one for each ghost.

Module: gameLogic.sv
Inputs:

- [4:0] pacTileX, pacTileY, redGhostTileX, redGhostTileY, pinkGhostTileX, pinkGhostTileY, blueGhostTileX, blueGhostTileY, orangeGhostTileX, orangeGhostTileY
- [15:0] keycode
- clk, restart, pacAnimationDone, frightenMode
- [8:0] pelletsEaten

Outputs:

- [2:0] redGhostState, blueGhostState, pinkGhostState, orangeGhostState
- reset, pacDead, ghostGone
- [15:0] score
- [8:0] frightenedCounter
- [1:0] lives

Description: This module takes in the x and y coordinates of the ghost and pacman determine which state the ghosts and pacman should be. There is a counter within this module that sends input signals into the 4 ghostFSM instantiations and other signals to determine which state the ghosts should be in. It also uses the coordinates of both ghosts and pacman to find out which ghost kills pacman or is being killed by pacman depending on if the ghost is currently in the frightened state or not which is a signal that gets sent out. This is further explained in the ghostFMS section. This module also takes in the number of pelletsEaten and ghostKills to calculate the total score and it also keeps track of the number of lives left and will decrease each time pac man is killed.
Purpose: This module was responsible for keeping track of the game logic which includes the points, lifes, ghost modes, interaction between ghost and pacman, and game reset and restart.

Module: Color_Mapper.sv
Inputs:

- [9:0] pacmanX, pacmanY, DrawX, DrawY, redGhostX, redGhostY, pinkGhostX, pinkGhostY, blueGhostX, blueGhostY, orangeGhostX, orangeGhostY
- [3:0] redGhostFrame, pinkGhostFrame, blueGhostFrame, orangeGhostFrame
- [4:0] pacmanFrame
- Restart, clk, ghostGone
- [1:0] lives
- [8:0] frightenedCounter
- [19:0] score

Outputs:

- [3:0] Red, Green, Blue
- [9:0] pelletsEaten
- frightenMode

Description: This module takes in the x and y pixels of the ghost and pacman to draw them onto the screen along with the pellets and maze. There is an instantiation of the mazeLayout and mazeTile which the color mapper uses to draw onto the background maze first. Then there is an instantiation of pelletLogic which maps out which tiles have a pellet, super pellet, and which one doesn't along with the signal frigthenMode gets passed out from it. The pacman and ghost are drawn onto the screen using an instantiation of pacmanSprite and ghostSprites which are kept track of using the x and y pixels of the ghosts and pacman. The instantiation of 4 numberSprites, wordGAME, wordOVER, and wordWON are here so it can be draw onto the screen at the appropriate time. NumberSprite is controlled by the score in decimal and the words are controlled with the win or lose condition of the game. The lives are also made up of instantiations of the pacmanSprite and disappear after a life is lost.
Purpose: This module is responsible for outputting the correct color for the specific pixel based on the pacman game for the VGA to HDMI to display the game. This will display all of the ghosts, pacman, pellets, score, lives, and maze.

Module: pelletLogic.sv
Inputs:

- [4:0] row, col, pacTilelX, pacTileY
- Restart, clk

Outputs:

- pelletExists, supperPelletExists, frightenMode
- [9:0] pelletsEaten

Description: This module takes in the row and col from the color mapper so it can output if a pellet/super pellet exists or not. A copy of the maze's pellets are stored in an array and whenever the game is restarted, it copies the original pellets into a usable double array since when pacman eats a pellet, the pellet will disappear. The pacTileX and pacTileY are used to determine which pellets get eaten and the number of pellets eaten will be outputted to help determine the score in gameLogic. When the superPellet is eaten, then the frightenMode output signal goes high to indicate that the pacman can eat the frightened ghosts.
Purpose: This module is responsible for the pellet logic like if a pellet/super pellet exists there or not since it was eaten and displaying it into the color mapper. It also keeps track of the number of pellets that is eaten since it is needed in the score.

Module: animationPacmanFSM.sv
Inputs:

- Clk, reset, ghostGone
- [2:0] directionAsych

Outputs:

- pacAnimationDone
- [4:0] frame

Description: This module is an FSM that chooses a frame for the color mapper to map out. It taks in a clock that is on a counter to slow the 60 hz down to 20 hz to make pacman's animation slower. It starts in the closed state and goes to different states depending on the direction pacman is moving. Also when ghostGone is high which indicates that pacman is dead, the FSM switches to the dying animation and outputs high to pacAnimationDone when the dying animation is done. This is further explained in the Pacman Animation FSM section.
Purpose: This module was responsible for controlling the animation of pacman based on the direction he is facing and if he is dying. This tells the colormapper which frame it should choose from the pacmanSprite.

Module: animationPacmanFSM.sv
Inputs:

- Clk, reset
- [2:0] direction
- [2:0] ghostState

Outputs:

- [3:0] frame

Description: This module is an FSM that chooses a frame for the colormapper to map out. It taks in a clock that is on a counter to slow the 60 hz down to 20 hz to make the ghost's animation slower. The ghost's basic movement is based on the direction it is moving and the states switch between its moving legs. It also uses the ghostState to determine which state it should choose and that outputs the frame should be on. Like when the ghostState is in frighted then the FSM switches to the frightened ghost animation frames and when the ghostState is in dead, then the FSM goes to the eyes animation frame. This is further described in the FSM section.
Purpose: This module was responsible for controlling the animation of the ghosts based on the direction they are facing and the state they are in. This tells the colormapper which frame it should choose from the ghostSprite.

Module: KeycodeDetector.sv
Inputs:

- Clk
- [7:0] keycode

Outputs:

- [7:0] output_keycode

Description: This module takes in the keycode and sets all of the variables to the same keycode expect for keycode 0000. On every clock edge, the variables feed into each other with 0000 being fed into the last variable. So if the keycode is 0000 then the variables will slowly turn back into 0000s but if the keycode is not 0000 then it sets all the variables to the inputted keycode. The first variable is the output of the module and therefore will stay a certain keycode that is not 0000 for 8 clock cycles.
Purpose: This module was responsible for holding the user's input for a little bit logic like a buffer so the user does not have to press the key exact on the clk cycle but has some leeway.

Module: ghostSprite.sv
Inputs:

- [4:0] row, col
- [3:0] frame

Outputs:

- [1:0] color

Description: This module takes in the row, col and frame needed by the color mapper and outputs the pallet color for it. The row and col need to be converted to little endian and the frame selects the unpacked element from the 4d array. The sprite of the ghost is stored as a parameter of [21:0] [21:0] [1:0] ROM [11] which is a 22 by 22 with 4 colors and 11 frames. The color ranges from transparent, ghost's color, white, blue.
Purpose: This module was responsible for storing the ghost's sprite and allowing the colormapper to see which color should be displayed for that specific pixel based on the row, col and frame it is on.

Module: pacman_sprite.sv
Inputs:

- [4:0] row, col
- [4:0] frame

Outputs:

- color

Description: This module takes in the row, col and frame needed by the color mapper and outputs the pallet color for it. The row and col need to be converted to little endian and the frame selects the unpacked element from the 3d array. The sprite of pacman is stored as a parameter of [21:0] [21:0] ROM [12] which is a 22 by 22 with 2 colors and 12 frames. The color ranges from transparent and yellow.
Purpose: This module was responsible for storing the pacman's sprite and allowing the colormapper to see which color should be displayed for that specific pixel based on the row, col and frame it is on.

Module: numberSprite.sv
Inputs:

- [15:0] row
- [7:0] col
- [3:0] number

Outputs:

- color

Description: This module takes in the row, col and number needed by the color mapper and outputs the pallet color for it. The row and col need to be converted to little endian and the number selects the unpacked element from the 3d array. The sprite of number is stored as a parameter of [15:0] [7:0] ROM [10] which is a 16 by 8 with 2 colors and 10 numbers. The color ranges from transparent and white. The numbers 0 - 9 are pre-stored in the parameter.
Purpose: This module was responsible for storing the numbers 0-9 and allowing the colormapper to see which color should be displayed for that specific pixel based on the row, col and number it is on.

Module: numberSprite.sv
Inputs:

- [15:0] row
- [7:0] col
- [3:0] number

Outputs:

- color

Description: This module takes in the row, col and number needed by the color mapper and outputs the pallet color for it. The row and col need to be converted to little endian and the number selects the unpacked element from the 3d array. The sprite of number is stored as a parameter of [15:0] [7:0] ROM [10] which is a 16 by 8 with 2 colors and 10 numbers. The color ranges from transparent and white. The numbers 0 - 9 are pre-stored in the parameter.
Purpose: This module was responsible for storing the numbers 0-9 and allowing the color mapper to see which color should be displayed for that specific pixel based on the row, col and number it is on.

Module: wordGAME.sv
Inputs:

- [9:0] row
- [34:0] col

Outputs:

- color

Description: This module takes in the row and col needed by the color mapper and outputs the pallet color for it. The row and col need to be converted to little endian selects an element from the 2d array. The sprite of the GAME is stored as a parameter of [9:0] [34:0] ROM which is a 10 by 35 with 2 colors that store the word “GAME”. The color ranges from transparent and white.
Purpose: This module was responsible for storing the word “GAME” and allowing the color mapper to see which color should be displayed for that specific pixel based on the row and col it is on.

Module: wordOVER.sv
Inputs:

- [9:0] row
- [34:0] col

Outputs:

- color

Description: This module takes in the row and col needed by the color mapper and outputs the pallet color for it. The row and col need to be converted to little endian selects an element from the 2d array. The sprite of the OVER is stored as a parameter of [9:0] [34:0] ROM which is a 10 by 35 with 2 colors that store the word “OVER”. The color ranges from transparent and white.
Purpose: This module was responsible for storing the word “OVER” and allowing the color mapper to see which color should be displayed for that specific pixel based on the row and col it is on.

Module: wordWON.sv
Inputs:

- [9:0] row
- [23:0] col

Outputs:

- color

Description: This module takes in the row and col needed by the color mapper and outputs the pallet color for it. The row and col need to be converted to little endian selects an element from the 2d array. The sprite of the WON is stored as a parameter of [9:0] [23:0] ROM which is a 10 by 24 with 2 colors that store the word “WON”. The color ranges from transparent and white.
Purpose: This module was responsible for storing the word “WON” and allowing the color mapper to see which color should be displayed for that specific pixel based on the row and col it is on.

Module: mazeTile.sv
Inputs:

- [6:0] addr

Outputs:

- [15:0] data

Description: This module takes in the address that is needed by the color mapper and outputs the row of pixel colors. The output data and address need to be converted to little endian selects an element from the 2d array. The maze tiles is stored as a parameter of [0:127] [15:0] ROM which is a 16 by 16 with 7 tiles and each pixel can have 2 colors. The tiles are blank, top left corner, top right corner, bottom left corner, bottom right corner, horizontal wall and vertical wall. The color ranges from transparent and white.
Purpose: This module was responsible for storing the different types of walls that can be used from the mazeLayout and allows the color mapper to see which color should be displayed for that specific pixel based on the row and col it is on.

Module: mazeLayout
Inputs:

- [4:0] row, col,
- mazenumber

Outputs:

- [2:0] mazepart
- Right, left, up, down

Description: This module is responsible for holding the maze layout. It takes in a tile row and column and then outputs the corresponding maze part for that square. There were 8 different maze parts possible. Right top corner, left top corner, right bottom corner, left bottom corner, blank, vertical edge, horizontal edge, and then no entry squares. We had
Purpose: We hardcoded our maze into a 3d array so we can access whether there were walls and what types of walls were on each tile.

Module: Pacman_game_top
Inputs:

- Clk, reset_rtl_0, usb_spi_miso, uart_rtl_0_rxd,
- [0:0] gpio_usb_int_tri_i

Outputs:

- Gpio_usb_rst_tri_o, usb_spi_mosi, usb_spi_sclk, usb_spi_ss, uart_rtl_0_txd, hdmi_tmds_clk_n, hdmi_tmds_clk_p
- [2:0]hdmi_tmds_data_n, hdmi_tmds_data_p
- [7:0] hex_segA, hex_segB
- [3:0] hex_gridA, hex_gridB

Description: This module had our instantiations for various modules. The main ones we added are the pacman and ghost animations, the logic to control the pacman and ghost movement, and then background game logic. This way our color mapper could take in information regarding ghost and pacman position and ghost and pac man state to draw the sprites correctly on the screen. It also takes in input from the GPIO ports so that we can communicate with the software side, sending over the ghost and pacman position and receiving the ghosts direction of movement. It also sent the score over so it can be converted from hexadecimal to decimal.
Purpose: This module acts as the bridge between FPGA to the physical inputs and outputs of it. The purpose of this module as a whole is to connect the different components of our game together.

Module: VGA controller
Inputs:

- pixel_clk, reset,

Outputs:

- hs, vs, active_nblank, sync
- [9:0] drawX, drawY

Description:

This module uses the counters to keep track of the current pixel position. It also generates sync pulses for hsync and vsync based on whether the pixel is in the horizontal or vertical buffer. For hsync it checks if the horizontal counter (hc) falls within the range of 656 to 752 pixels and for vsync, it checks if the vertical counter (vc) reaches lines 490 or 491. The output active_nblank determines whether the display should be active or in the blanking interval. The drawX and drawY tells us the pixel we are currently on.

Purpose:

This module is responsible for orchestrating all the timing and signals required to drive the VGA display.

Module: HexDriver

Inputs:

- clk, reset,
- [3:0] in[4]

Outputs:

- [7:0] hex_seg,
- [3:0] hex_grid

Description:

This module uses clk to run the counter which is needed to go through the hex grid and hex seg. Within this module, it uses module nibble_to_hex which takes in nibble and outputs hex. It converts the input which is [3:0] into an 8-bit value that can be used for the hex display. It outputs the converted [3:0] into the corresponding hex grid and seg.

Purpose:

This module is used to convert and display the values within the registers A and B to the hex driver for the user to see.

Block Descriptions

Module: MicroBlaze
Description/Purpose: The MicroBlaze is a 32 bit soft processor core that is highly customizable and can be implemented entirely on the FPGA. It can be configured to meet specific performance, area, and feature requirements based on what the used wants and can support various peripherals and interfaces.

Module: Clocking Wizard
Description/Purpose: The Clocking wizard is a tool that generates clocking circuits with desired frequencies. It's main role is ensuring proper timing within the FPGA. It also generates two additional clocks at 25 MHZ and 125 MHZ which are used by the VGA controller and VGA to HDMI converter.

Module: VGA to HDMI
Description/Purpose: The VGA to HDMI IP takes the analog VGA signal as input, processes it, converts it into digital format, and then formats it according to the HDMI specification. This process is further explained below.

Module: Local Memory
Description/Purpose: The Local Memory is an on chip memory that provides fast access and is used to store frequently accessed memory. It is used to store critical data and instructions to minimize access latency.

Module: Debug Module
Description/Purpose: The Debug Module provides capabilities such as breakpoints, watchpoints, and trace features to aid in the debugging process.

Module: Processor System Reset
Description/Purpose: The Processor System Reset is responsible for resetting the processor system, including all processor cores and associated peripherals. It ensures a clean start-up state in response to a reset signal.

Module: AXI Interconnect
Description/Purpose: The AXI Interconnect is the bus architecture used for connecting various IP cores and peripherals in an FPGA design. It is responsible for routing data and control signals between different components.

Module: Concat
Description/Purpose: The Concat component is used for concatenating signals or data streams in an FPGA design. It is often used to combine data from multiple sources.

Module: AXI4 Interrupt Controller

Description/Purpose: The AXI4 Interrupt Controller is a peripheral used for managing interrupt signals in an FPGA design. It handles interrupt requests and helps with efficient handling of asynchronous events.

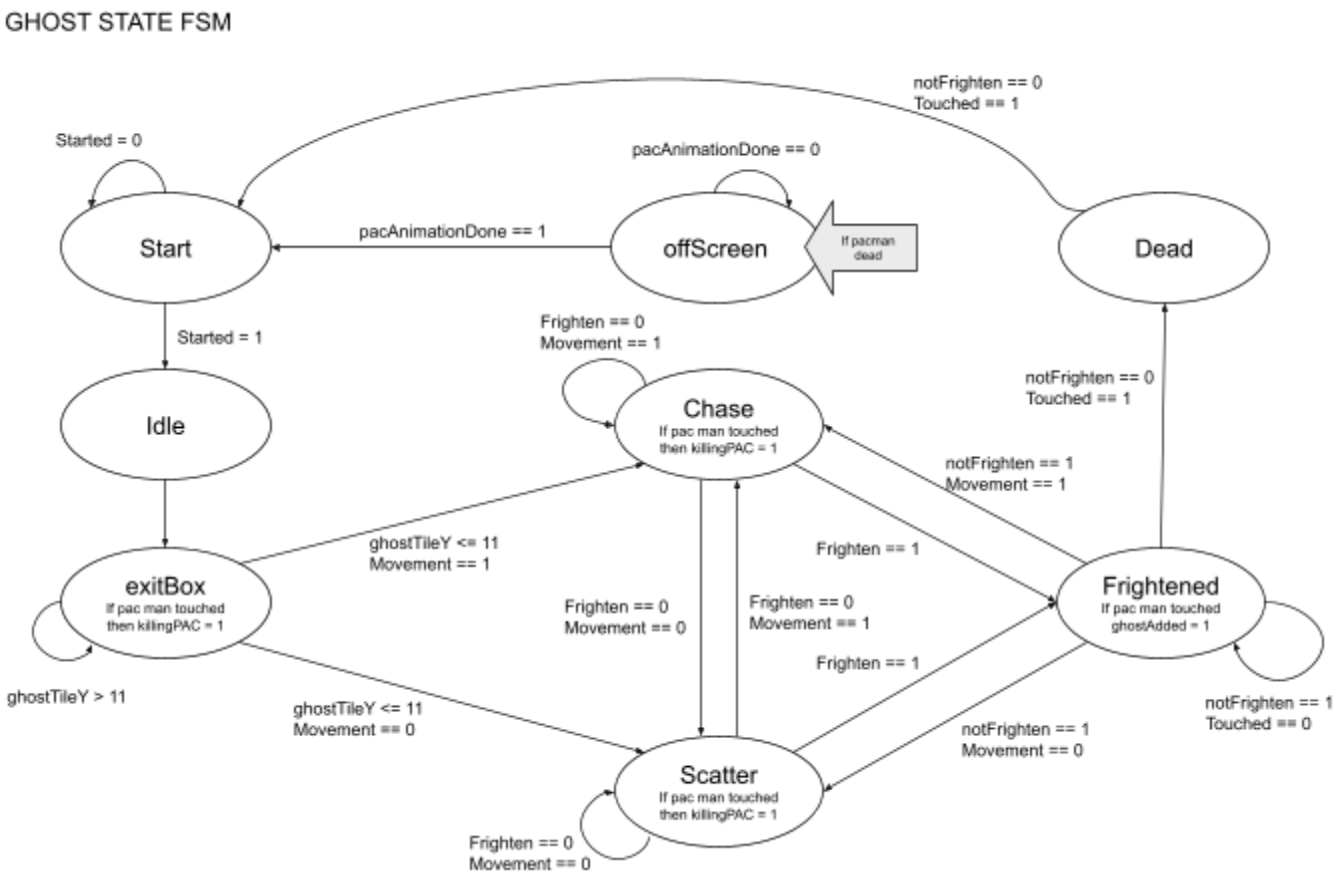
Module: SPI
Description/Purpose: The Quad SPI block is the serial peripheral interface and its a communication protocol for interfacing with external devices. In our lab it interacts with the MAX3421E block and it allows for high speed data transfer. It used four signals: clock, master out slave in, master in slave out, and slave select/chip select.

Module: GPIO
Description/Purpose: The general-purpose input/out module is an IP that provides a simple interface for controlling general-purpose I/O. Its ports include a clock and reset and also directly connect to the AXI interconnect. The GPIO ports can be inputs or outputs depending on the GPIO TRI which allows it to read or write data from the peripheral.

Module: UARTLite
Description/Purpose: The Uartline block is the universal asynchronous receiver/transmitter and is able to serially communicate with external devices or systems. We used it for its debugging capabilities because it is able to send debug messages and/or receive commands.

Module: AXI Timer
Description/Purpose: The AXI Timer provides timing functionalities within a system based on the FPGA clock. It can generate accurate time delays, measure elapsed time, or trigger events at specific intervals.

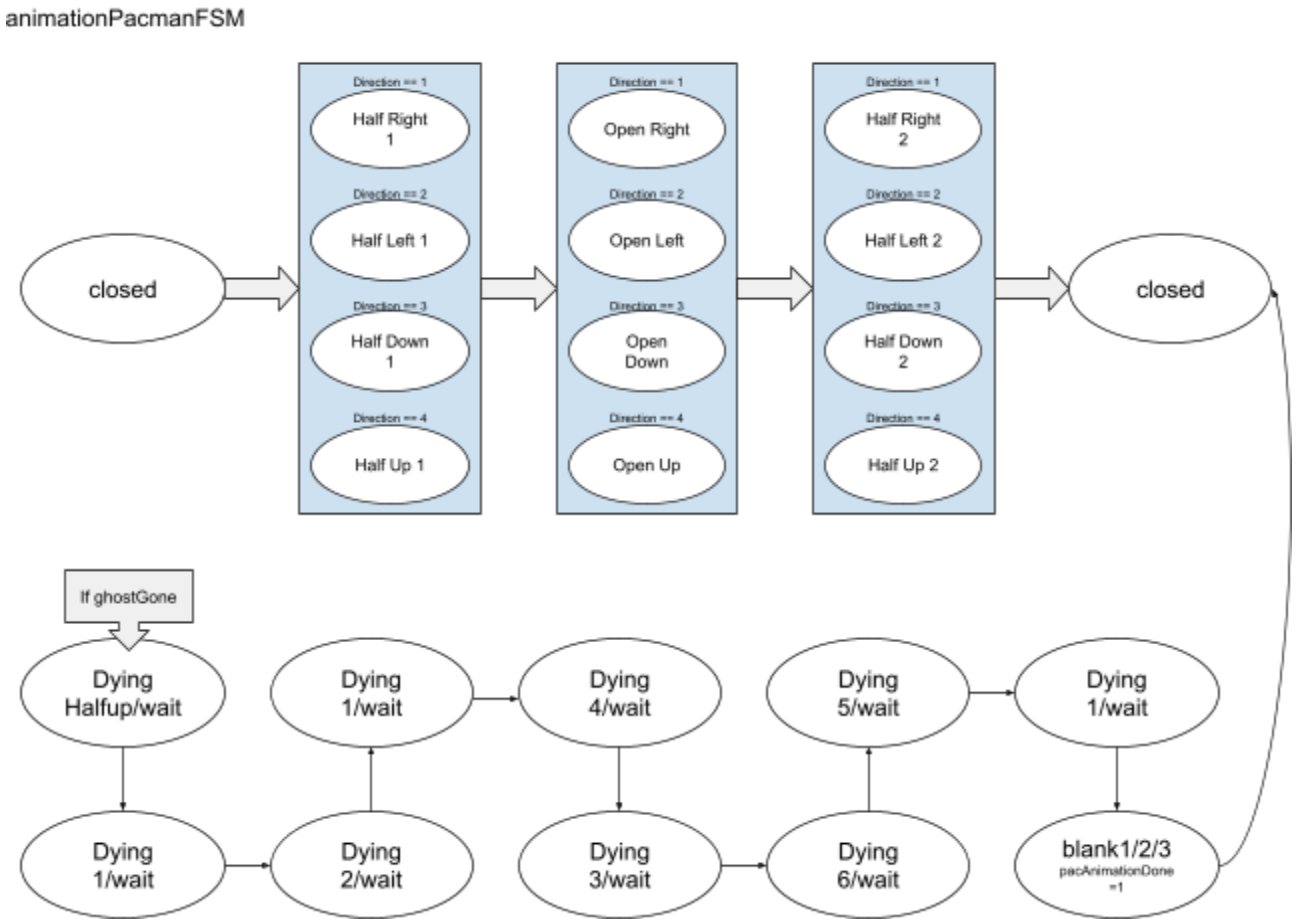
Finite State Machines



Ghost State FSM

This FSM is used to control the state that the ghost is in during the game. The game starts with all the ghosts being in the Start state and once “R” has been pressed “Started” is set to 1 forever. This makes the state go to idle which is when the ghosts are just moving aimlessly around the box and then it moves into exitBox. At this state, the ghost is trying to exit the box and only when it’s ghostTileY is <= 11 which means it has exited the box, may it go to the next state. The next state is determined by the input signal Movement which is from the game logic module. If the movement is == 1 then the ghost must be chasing pacman, else it will be in scatter mode which is just a set of predetermined tiles. During these states, they may become frightened which is controlled by the input signal “Frighten” and will send the states into Frightened state. Otherwise, the state will swap to whichever movement it should be from the input signal “Movement”. During any of the exitBox, Scatter, or Chase states, if pacman touches the ghost, then the output “killingPAC” is set high to indicate that pacman is being killed. Once in the frightened state, the state will stay in the frightened state until the input “NotFrighten” is set high or if pacman is touched. When “NotFrighten” is set high, the state goes back to either scatter or chase set depending on “Movement”. When the ghost is touched while in frighten mode, the ghost dies and therefore the state must go to the Dead state. During the dead state the ghost is just a pair of eyes and will only go back to the start state when it returns to its

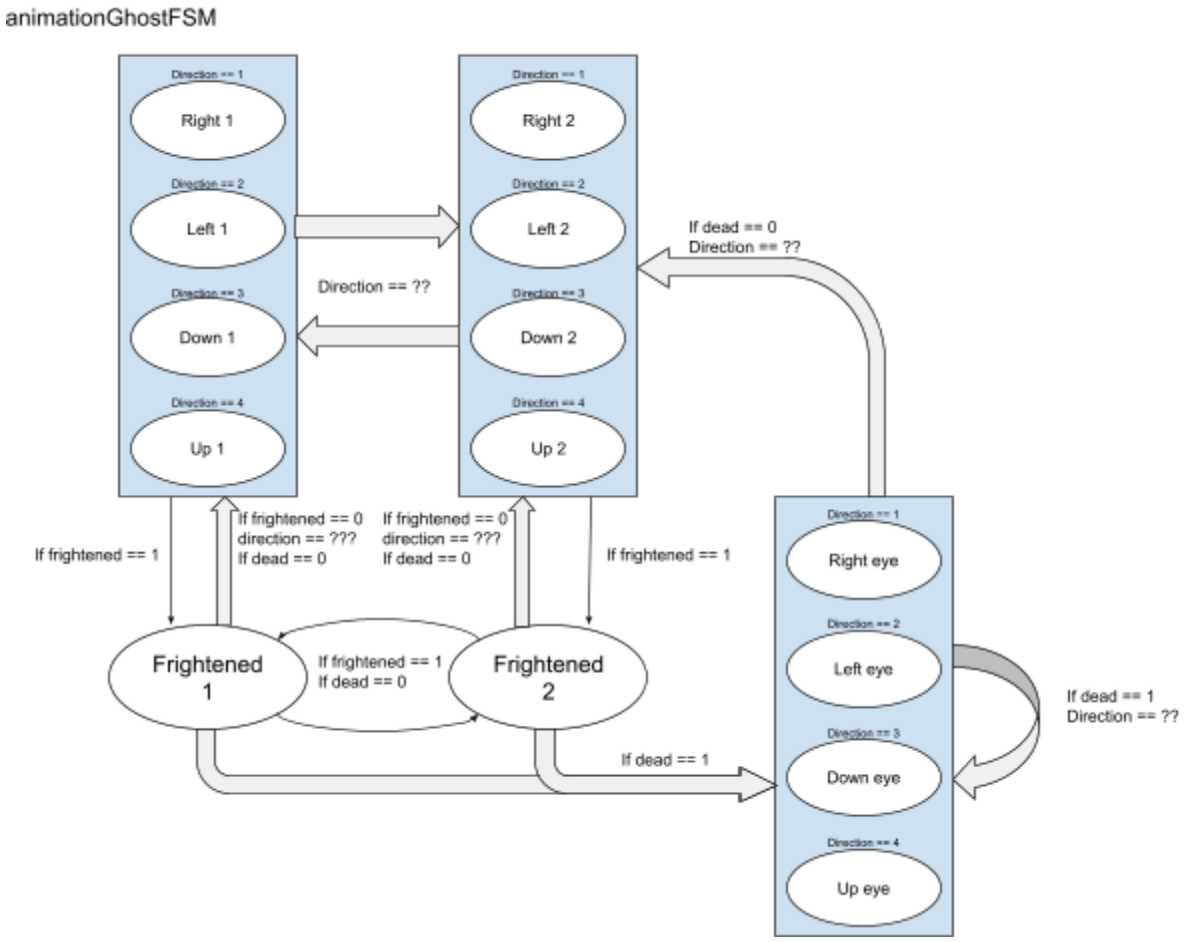
spawn area by tracking the ghostTileX and ghostTileY of the ghost. These states are being passed to other modules like game logic, animation, and color mapper so it can properly do timing, different animations, and coloring. Lastly, the states are also outputted into the microcontroller since the directions of the ghost is controlled differently based on the state of ghost which is done in C.



Pacman's Animation FSM

This FSM is used to control the animation for our pacman, so every 1/20 seconds, it moves to the next state. For each state, they have an output called frame which is used by the color mapper to choose the frame from the sprite sheet. At the very basics, the pacman has 4 repeating states which is closed, half1, open, half2, and then back to closed. However each direction has its own state which meant that for 1 state, there was 4 outputs (1 for every direction). For example, if it is in halfUp1, then there is a if condition that checks the direction then selects OpenRight, OpenLeft, OpenDown, and OpenUp. In the Pacman's Animation FSM drawing, this has been reduced to a larger arrow with the conditions being over the states instead to improve readability. There is another condition which overrides any of the states which is if ghostGone. Since if the ghosts are gone to the offscreen, that means pacman has dead and regardless of the state that pacman is in, it must go to the first dying state which is

DyingHalfUp. Each of the states has a wait state which makes the animation slower. The last states blank1, blank2, and blank3 are used as a buffer so the screen stays black for a bit till the game resets. At the very last state (blank3) the output signal “pacAnimationDone” is set high to indicate to the game logic that the dying animation has finished and the ghosts should reset too.



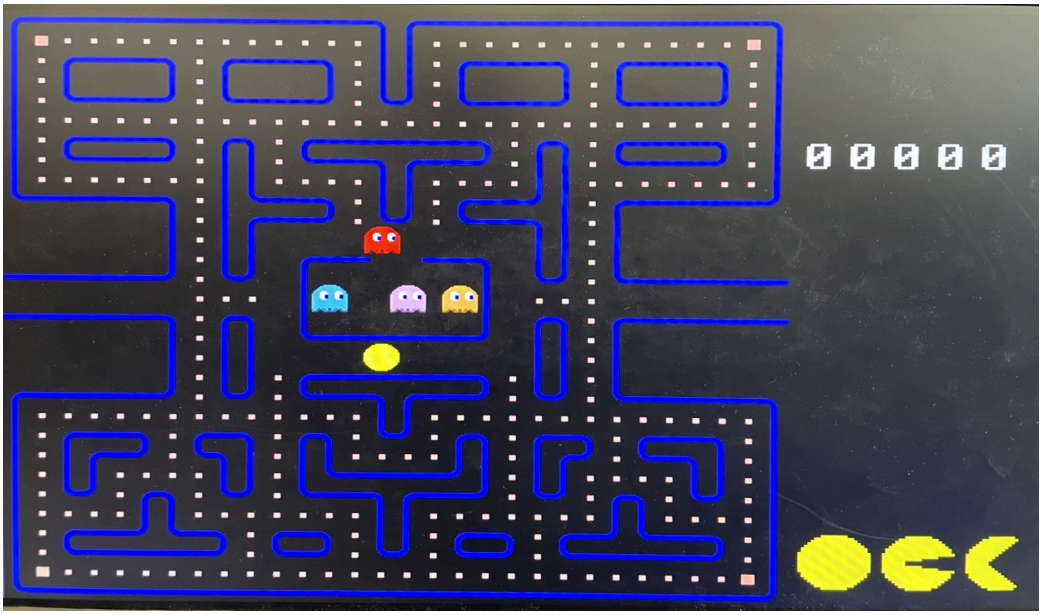
Ghost Animation FSM

This FSM is used to control the animation of the ghost. The ghost has 2 frames per set of animation because its little legs move back and forth and each set of animation needs a direction. The basic movements are called up1, up2, etc which is when the ghost is walking around the map and must switch between the different leg positions. If the ghosts are in frightened mode, then it must switch states to frightened1 and frightened2 if the ghost is currently in basic movement. If the ghosts are no longer frightened then the ghost animation state must go back to basic movement depending on its direction or if the ghost is dead then it must switch to the eyes animation, else it keeps switching between frightened1 and frightened2. Once the ghost is dead, it switches to eyes which also depend on direction and stays in the eyes' animation state until the input signal dead is low which returns the state into the basic movement state depending on the direction.

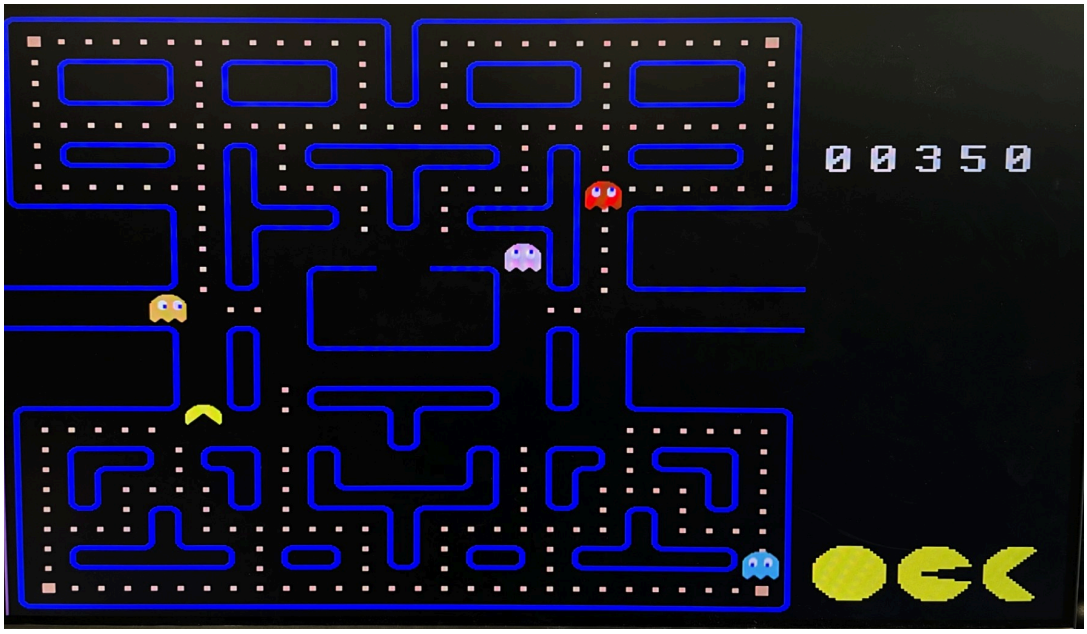
Sprite Sheet

The sprites created for this final project were 22 by 22 and was used for the animation of the ghost and pacman. The ghost has 2 sprites per direction since its legs move for basic movement and frightened mode. Once it dies, they become eyes which have 4 for each direction. Pacman's basic movement is very simple with only 3 sprites per direction which gets recycled since pacman is symmetrical. For example left pacman is just right pacman but reading from the left to the right instead of right to left for the columns. The death animation was created for when pac man touches a ghost and dies.

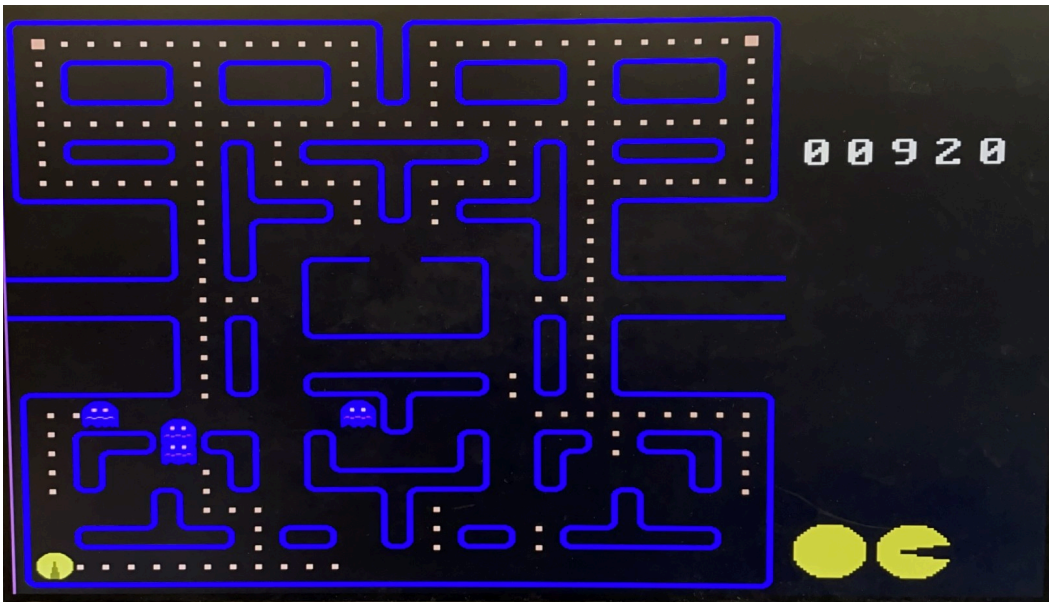
Gameplay



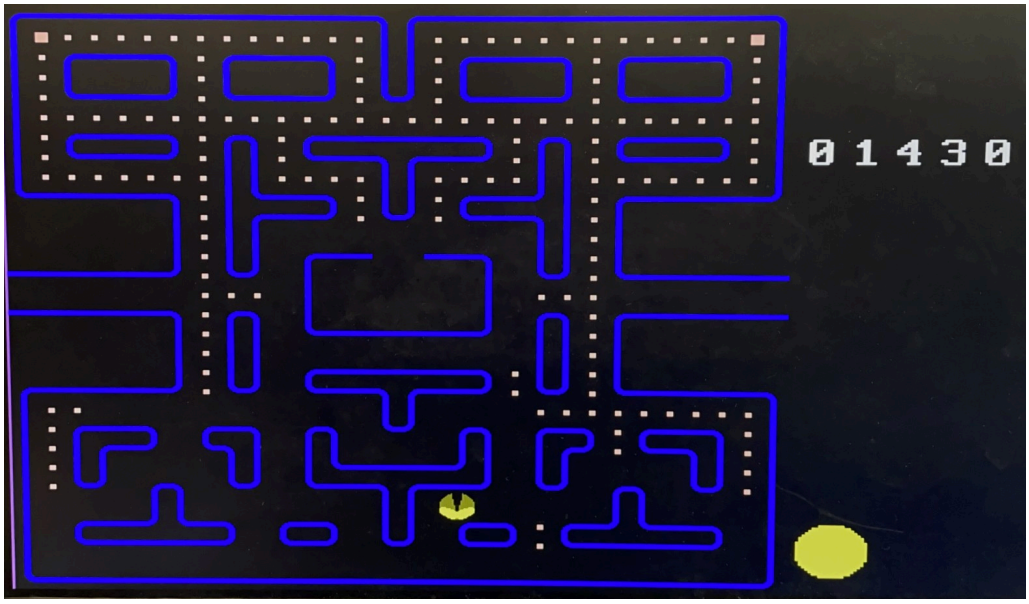
Our initial start screen is blank with just the pacman on it. To initiate gameplay, first press “r” to reset the game, the ghosts appear, and then “w”,”a”,”s”, or “d” to begin moving the pacman. The score and lives are displayed in the right column.



This shows the pacman moving around the screen and the ghosts currently in scatter mode. In this mode, each ghost targets a tile right outside of their respective corners. The red corner is top right, the pink corner is top left, the orange corner is bottom left, and the blue corner is bottom right. In chase mode, the ghosts target the pacman.



The image shows the ghosts in frightened mode, which occurs after a pacman has eaten a super power pellet (located in each corner of the maze). All of the ghosts turn blue and target random tiles on the maze. Now the pacman can eat the ghosts and gain points. The ghosts will flash white when frightened mode is ending.



This shows the pacman dying after a ghost kills it. Right now it is transitioning between animation states. When a pacman dies, all of the ghosts disappear and reset to the ghosts box after its full death. Also notice the number of lives in the bottom right has decreased.

When you lose all of your lives, the maze will disappear and a “game over” sign will appear. We have a separate end screen for when you win.

Design Resources and Statistics

LUT	7114

DSP	3
Memory (BRAM)	8
Flip-Flop	5081
Latches	0
Frequency	114.77 MHz
Static Power	0.076 W
Dynamic Power	0.469 W
Total Power	0.545 W

Conclusion

In conclusion, we both learned a lot from this project. For starters, we better learned how to interface software and C code with the hardware. Initially our plan was to do mostly everything in hardware because we wanted more practice with it and we didn't want to figure out how to set up the C side again. However, once we started implementing ghost logic on the C side, we realized how simple it was and much faster to debug and test.

Looking at features we would have liked to implement with more time. We would have loved to add sound to the game and make it multi level as well. Right now once you finish, you can keep playing to get a higher score but that's only for a while. Adding different mazes and cycling through them would have been cool. We also wanted to try to add different speeds to the pacman and ghosts. Right now they both move at the same speed, but to make it harder, we initially wanted to make the ghost faster but ran out of time before we could have figured that out.