ECE 385

Fall 2023

Final Project

Pac Man

Rhea Tiwari & Iona Lee-Parnell
ECE 4072 Friday 13:30
Srijan Chakraborty

Introduction

For the final project, we decided to create a version of Pacman. For our game, we decided that we would have a maze, a pacman that would be controlled with the keyboard, four ghosts that would move by themselves, food dots for pacman to collect, and power-ups if we had time. We also wanted pacman to have three lives and for the game to be over either after pacman loses all his lives or he collects all the food. We decided that we would make the majority of this game through System Verilog (hardware). This way we could build off of Lab 6 because in that lab we created a ball and had boundaries, which would be helpful in our game.

Module Descriptions

Module: hex.sv
Inputs: clk, reset, [3:0] in[4]
Outputs: [3:0] hex_grid, [7:0] hex_seg
Description: The Hex Driver was similar to the code we used before with the information about how to represent the leds on the FPGA as hex numbers.
Purpose: To output a desired output onto the LED displays.

Module: VGA_controller.sv
Inputs: pixel_clk, reset
Outputs: hs, vs, active_nblank, sync, drawX, drawY
Description: This module is used to keep track of each pixel of a standard 640x480 standard screen. It is made up of two counters. A horizontal counter and a vertical counter. These keep track of the position of the pixel that is currently being drawn to. If the horizontal counter has reached its maximum, then it will reset and the vertical counter will be incremented by one.
Purpose: To keep track of each screen pixel being displayed.

Module: mb_usb_hdmi_top
Inputs: Clk, reset_rtl_0, gpio_usb_int_tri_i, usb_spi_miso, uart_rtl_0_rxd
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, hdmi_tmds_data_n, hdmi_tmds_data_p, hex_segA, hex_gridA, hex_segB, hex_gridB
Description: This is where the signals being input and output of each module are connected.
Purpose: To act as the connection between the FPGA and the SystemVerilog code.

Module: ball.sv
Inputs: Reset, frame_clk, [7:0] keycode, state1, state2, state3, pac_reset, vga_clk, stop, [3:0] blue
Outputs: [9:0] BallX, [9:0] BallY, [9:0] BallS1
Description: This contains the code for pacman to be able to deal with constraints from the maze

Purpose: The purpose is for the ball (pacman) to be able to move throughout the maze without getting stuck in the borders of the maze

Module: Color_Mapper.sv
Inputs: [9:0] BallX, BallY, DrawX, DrawY, Ball_size, ghostX, ghostY, ghostXi, ghostYi, ghostXp, ghostYp, ghostXc, ghostYc; Clk, pac_life, state1, state2, state3, stop, wire test, vga_clk, blank, font, food_on, [3:0] mazer, mazeg, mazeb
Outputs: [3:0] Red, Green, Blue; ball_on, ghost_on, ghosti_on, ghostp_on, ghostc_on
Description: Create and stores coordinates for the ghosts and pacman and uses them keeps track of when the game restarts and when it ends
Purpose: To show all the ghosts and pacman and to be able to keep track of the number of lives left for pacman/when the game is over

Module: ghost.sv
Inputs: Reset, frame_clk, Clk, vga_clk, [7:0] keycode, scatter, chase, start, scatteri, chasei, starti, pac_reset, stop, [9:0] pacX, pacY
Outputs: [9:0] ghostX, ghostY, ghostS, ghostXi, ghostYi, ghostSi
Description: To make the red ghost and code it to be able to move based on an AI algorithm we created while dealing with maze constraints
Purpose: The purpose is for the red ghost (Blinky) to be able to move around the maze and how to deal with boundaries and corners

Module: ghost_i.sv
Inputs: Reset, frame_clk, Clk, vga_clk, [7:0] keycode, scatter, chase, start, movei, pac_reset, stop, [9:0] pacX, pacY
Outputs: [9:0] ghostX, ghostY, ghostS
Description: To make the green ghost and code it to be able to move based on an AI algorithm we created while dealing with maze constraints
Purpose: The purpose is for the green ghost (Inky) to be able to move around the maze and how to deal with boundaries and corners

Module: ghost_pinky.sv
Inputs: Reset, frame_clk, Clk, vga_clk, [7:0] keycode, scatter, chase, start, movep, pac_reset, stop, [9:0] pacX, pacY
Outputs: [9:0] ghostX, ghostY, ghostS
Description: To make the brown ghost and code it to be able to move based on an AI algorithm we created while dealing with maze constraints
Purpose: The purpose is for the brown ghost (Pinky) to be able to move around the maze and how to deal with boundaries and corners

Module: ghost_c.sv
Inputs: Reset, frame_clk, Clk, vga_clk, [7:0] keycode, scatter, chase, start, movec, pac_reset, stop, [9:0] pacX, pacY
Outputs: [9:0] ghostX, ghostY, ghostS
Description: To make the orange ghost and code it to be able to move based on an AI algorithm we created while dealing with maze constraints
Purpose: The purpose is for the orange ghost (Clyde) to be able to move around the maze and how to deal with boundaries and corners

Module: pac_maze_example.sv
Inputs: vga_clk, blank, [9:0] DrawX, DrawY
Outputs: [3:0] red, green, blue
Description: Contains code to take a sprite and stretch it across the screen based on the dimensions we give it
Purpose: To create the actual maze that pacman and the ghosts will travel through

Module: pac_maze_palette.sv
Inputs: [0:0] index
Outputs: [3:0] red, green, blue
Description: Contains code that will assign a color to the maze, which is blue in our case
Purpose: To make the maze look blue like it is in the actual arcade game

Module: controller.sv
Inputs: Clk, Reset, state1, state2, state3, pac_reset, [7:0] keycode
Outputs: scatter, chase, start, [9:0] counter
Description: This module includes the algorithm for the red ghost to follow. It starts moving as soon as a key is pressed and then alternates between scattering to the top right corner of the move for a few seconds, and then chasing after pacman (using pacman's coordinates and finding the fastest way to get to it) for a few seconds
Purpose: The purpose is for the red ghost (Blinky) to actually be able to move around the maze, alternating between chasing pacman and scattering to one corner of the maze.

Module: controller_i.sv
Inputs: Clk, Reset, state1, state2, state3, pac_reset, [7:0] keycode
Outputs: scatteri, chasei, starti, movei
Description: This module includes the algorithm for the green ghost to follow. It starts moving a few seconds after Blinky and then alternates between scattering to the bottom right corner of the move for a few seconds, and then chasing after pacman (using pacman's coordinates and finding the fastest way to get to it) for a few seconds

Purpose: The purpose is for the green ghost (Inky) to actually be able to move around the maze, alternating between chasing pacman and scattering to one corner of the maze.

Module: controller_p.sv
Inputs: Clk, Reset, state1, state2, state3, pac_reset, [7:0] keycode
Outputs: scatterp, chasep, startp, movep
Description: This module includes the algorithm for the brown ghost to follow. It starts moving a few seconds after Inky and then alternates between scattering to the top left corner of the move for a few seconds, and then chasing after pacman (using pacman's coordinates and finding the fastest way to get to it) for a few seconds
Purpose: The purpose is for the brown ghost (Pinky) to actually be able to move around the maze, alternating between chasing pacman and scattering to one corner of the maze.

Module: controller_c.sv
Inputs: Clk, Reset, state1, state2, state3, pac_reset, [7:0] keycode
Outputs: scatterc, chasec, startc, movec
Description: This module includes the algorithm for the orange ghost to follow. It starts moving a few seconds after Pinky and then alternates between scattering to the bottom left corner of the move for a few seconds, and then chasing after pacman (using pacman's coordinates and finding the fastest way to get to it) for a few seconds
Purpose: The purpose is for the orange ghost (Clyde) to actually be able to move around the maze, alternating between chasing pacman and scattering to one corner of the maze.

Module: controller_pac.sv
Inputs: Clk, Reset, ball_on, ghost_on, ghosti_on, ghostp_on, ghostc_on, [7:0] keycode, [9:0] pacX, pacY, ghostX, ghostY, ghostXi, ghostYi, ghostXp, ghostYp, ghostXc, ghostYc, drawX, drawY
Outputs: pac_life, state1, state2, state3, pac_reset, stop
Description: This module includes the code for how to count down through pacman's lives and how to stop the game once it is over, either because pacman used all his lives or because he collected all the food
Purpose: The purpose is for the game to actually be able to end after a certain point

Module: food_dots.sv
Inputs: reset, Clk, food_on, [9:0] DrawX, drawY, pacX, pacY
Outputs: [9:0] counter, [3:0] one_dig, two_dig
Description: We created each food dot by creating a circle and placing it on a coordinate within the maze. Then we created flags to see if pacman's coordinates and the food's coordinates match, and if they do, make the food disappear and add a point to a counter (for the score).

Purpose: Pacman collects food throughout the game and once he gets all the food, he wins. Each food dot is worth a point, so they also help with getting a higher score

Module: text.sv
Inputs: clk, stop, Reset, [3:0] dig0, dig1; [9:0] x, y
Outputs: [3:0] text_on, reg [11:0] text_rgb, wire text
Description: We used the counter from the food dot module to update the score and print it to the screen. We also use the pacman controller module to determine if we lost or won and which text to print to screen once the game is over.
Purpose: The purpose is for the score to show up and update, as well as the text to indicate if we won or lost to show up.

Module: food_rom.sv
Inputs: clk, wire [10:0] addr, reg [7:0] data
Outputs: scatteri, chasei, starti, movei
Description: This code includes the code for the text that needs to be displayed and how it is stored in memory.
Purpose: The purpose is to help the texts from the text module to be able to display properly on the screen.

Block Design Components

AXI Uart: The AXI Uart connects to the bus of the microcontroller and provides the controller interface for asynchronous serial data transfer. The baud rate of the UART must be the same as the baud rate of the port connection in vitis to ensure that data is being received at the same rate that data is being sent. The AXI uart allows printf statements to be used for debugging purposes.

Microblaze: The microblaze processor is configured as a microcontroller for this lab. By adding memory and additional peripherals, a useful design can be created.

Clocking wizard: The clocking wizard is used to configure a clock circuit to the specifications of the user. The design used in this lab requires two clocks. A 25 MHz clock and a 125 MHz clock. These are required by the SPI peripheral which has a frequency ratio of 4.

Microblaze debug module: The MDM core used in this design enables JTAG-based debugging of a Microblaze processor.

Microblaze local memory: The local memory is used for connection Microblaze instruction and data ports to peripherals on BRAM.

AXI Interconnect: The AXI Interconnect is used to join AXI memory mapped master devices to memory mapped slave devices. It is only intended for memory mapped transfers.

Processor system reset: The processor system reset it used to allow users to adjust the design to better suit their desired application.

Microblaze concat: Concat is used to combine bus signals with different widths. The output is a bus that combines the input signals together. Concat is used in this design to concatenate the individual interrupts into a singular signal that is sent to the interrupt controller.

AXI Quad SPI: The quad Serial Peripheral Interface is used to connect the AXI bus to the SPI slave device. In this lab the SPI peripheral is used to communicate with the MAX3421E. SPI consists of 4 signals; MOSI, MISO, CS and CLK. MOSI is a data signal that transmits data from the Microblaze to the MAX3421E. MISO is a signal which transmits data from the MAX3421E to the Microblaze.

AXI Interrupt Controller: The interrupt controller combines multiple interrupt signals from peripheral devices to a single interrupt output to the system processor.

AXI GPIO: This is the input and output interface. In this lab they are used to control the keycode operations and the reset button.

AXI Timer: The AXI timer is a 32-bit timer. The timer is used in this design as the USB has many timeouts that need precise timekeeping. The timer therefore allows the Microblaze to keep track of when a sufficient amount of time for a task has elapsed.

Description of C code Functions

MAXreg_wr

This function is used to write to the MAX3421E. Its arguments are a register and data. These are used to write to a register and provide it with the required data. Firstly the MAX3421E must be selected using the SetSlaveSelect function. The XSPI_transfer function is used to transmit the data according to the protocol structure.

MAXbytes_wr

This function is used to write multiple bits of data to the MAX3421E. Like the previous function, this function first tells the MAX3421E which register it will write to and then transmits the data using the XSPI_transfer function. The MAX3421E is able to expect multiple data bytes as we select the slave for the entire trnasmitting interval, which means the Max3421E knows that until

it is deselected it must anticipate more data from a write command. This function returns a pointer to a memory address that we have written to last. This makes it easier to call the function multiple times.

MAXreg_rd

This function is used to read from the MAX3421E. The function writes to the register it wants to read from. And then reads back from that register in the same XSPI_tranfer. This is because the communication between the Microblaze and the MAX3421E is fully duplex. When we return the value from the readbuffer which we have created as an array to read the data from the MAX3421E into, we must return the second byte from the readbuffer. This is because the first byte will return a memory address that we do not need. The information we require is instead in the second byte.

MAXbytes_rd

This function is used to read multiple bytes from the MAX3421E. The function writes to the register it wants to read from. The XSPI_Transfer function is placed in a for loop and the data at the second bit of the readbuffer array is then placed in the data array. Then the data array is added to the number of bytes given as an argument. This value is returned.

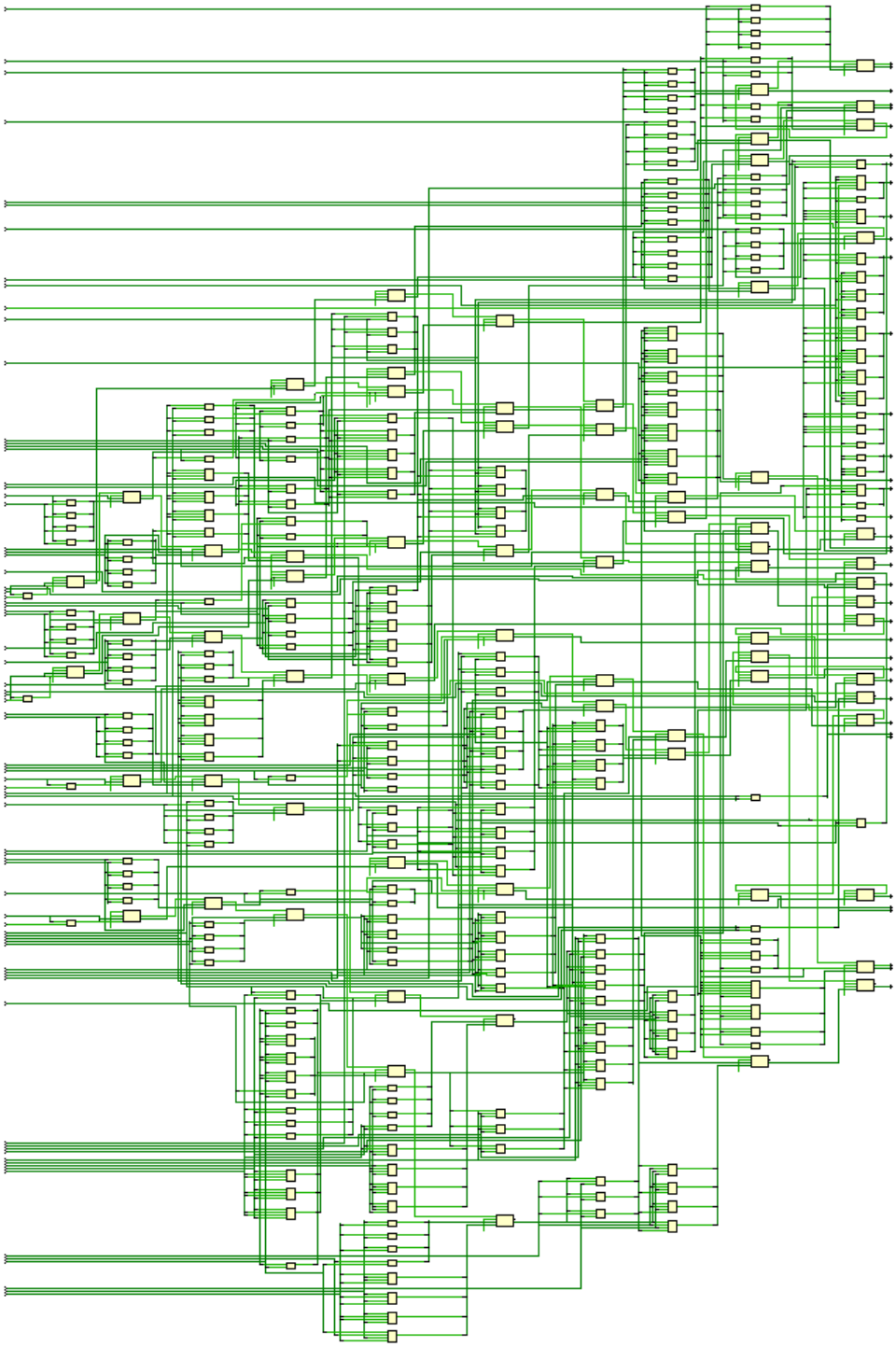

Figure 1: RTL Block Diagram of Pac Man Game

Design Resources and Statistics Table

LUT	23207
DSP	95
Memory (BRAM)	65
Flip-Flop	3080
Latches	0
Frequency	103.3 MHz
Static Power	0.079 W
Dynamic Power	0.837 W
Total Power	0.916 W

Table 1: Design Statistics Table for week 1

The Statistics table shows that a lot of the FPGA's resources were used in the development of this game. This is because the game was developed in hardware instead of software.

Post Lab Questions

Hardware vs Software development of the Game

Our version of Pac Man was developed mainly in System Verilog. The only C code used in our programme is to receive the keyboard controls and pass them into the top level of the project in Vivado. Our game builds on the logic we designed for lab 6 of this course. This meant that we had working keyboard controls which would be used to move pacman around the map. The difficulty with developing our game solely in hardware was that it used up a lot more resources than had we chosen to use software. This led to issues towards the end of the game development. As we were running out of resources on our FPGA, it meant we were not able to implement all 141 food dots that we had originally mapped out and coded for our game. The logic we used proved too costly and in order to add other features such as giving Pac Man multiple lives, we had to reduce the number of food dots in the game. The use of hardware over software also meant that our code was not as sleek as it could have been had we used C.

Pac Man Ghost AI

Similar to the original Pac Man game, each of the four ghosts have two states; scatter and chase. When the ghosts are in the scatter state, they will move towards one of the four corners of the map. Once a ghost is in the chase state, it will calculate the distance between itself and pac man, then determine if one pixel left, one pixel right, one pixel up or one pixel down will give it the closest distance to pac man. The order of the distances is then arranged so the ghost knows which distance is shortest up to which distance is longest. It then determines whether it can move in the direction of the shortest path. If there is a boundary in that direction one pixel ahead, the ghost will then look at the second shortest distance to see if there is a boundary in that direction. This continues until the ghost can find a direction to move towards. Once the ghost reaches the next pixel it will re-calculate each distance with respect to Pacman's new location.

Each ghost follows the same logic for chase. However, each ghost has its own finite state machine which is attached to a counter. This means that there is always a ghost in the chase state. The ghosts are set to leave their base with a staggered exit. The red ghost will leave as soon as the first keycode for pacman is pressed. This also begins a counter so the other three ghosts know when to leave.

If one of the ghosts is in the same location as pacman this sets off a flag which causes pacman to lose a life, while all the ghosts are reset to their original starting location.

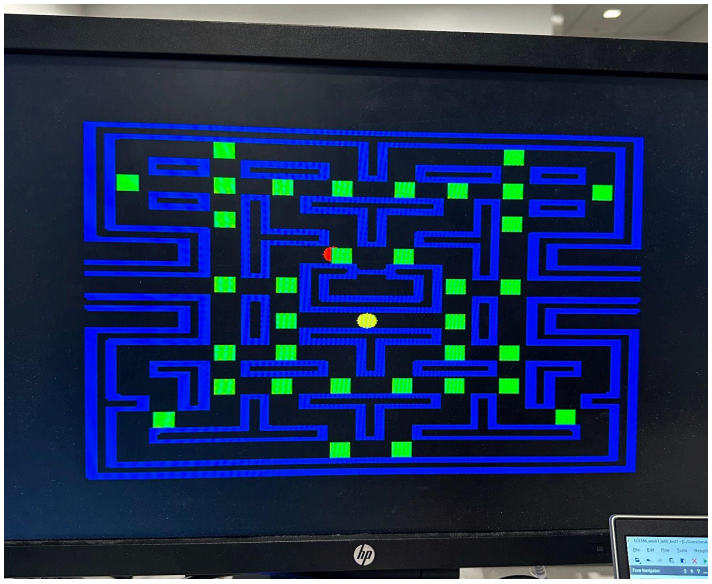

Figure 2: Tiles Being Added to the Maze

Pac Man

Pacman is controlled by the user using the "AWSD" keys of a keyboard. Pacman can only move up, left, right or down. Pacman is not able to cross a boundary. Boundary conditions are checked by using the ROM that contains the coordinates of each pixel that makes up the map. It pacman's top, right, left or bottom boundary overlaps with the boundary of the map, pacman will simply jump back and stop moving.

Pacman has a state machine which gives it three lives. If Pacman collides with a ghost and a flag is set, one of Pacman's lives is lost and the game resets with each character back in their original starting position. If Pacman loses three lives, the game ends with a "GAME OVER" appearing on the screen.
If Pacman moves into the boundary tunnel, it is able to pass through to the other side of the screen. This is done by giving Pacman a new set coordinate once it goes past a certain point of the X-axis.

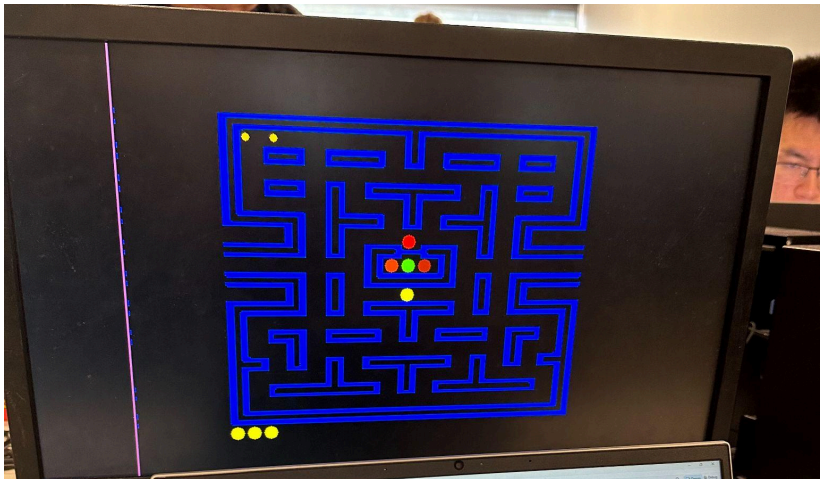

Figure 3:Pacman with lives and Ghosts

Food

If Pacman's coordinates overlap with the coordinates of the food, this will set off a flag to turn off that food dot so that it no longer appears on the screen. If Pacman goes over the coordinate and the flag has not yet been raised, this will increment a counter which is used to show the score. If all the food dots have been collected, Pacman has won the game and "YOU WIN" is projected onto the screen.
To accurately project the score count of the screen, two variables, digit one and digit two, were created. Each was four bits and started their count at 0. Digit one was fed the value of the counter. If the counter reached 10, it was reset and the value of digit two was increased by one.

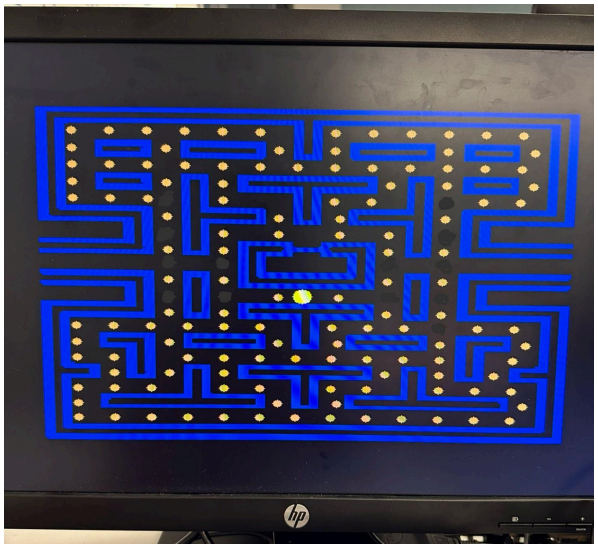

Figure 4: Food Dots Added to Maze

Discuss functionality of your design. If parts of your design did not work, discuss what could be done to fix it?

We implemented a functional Pacman game. Pacman is able to eat the food dots, which increase the player's score. The ghosts chase after Pacman and if a collision occurs, Pacman loses a life. If all of Pacman's lives are lost the game ends with the characters disappearing from the screen. If Pacman consumes all of the food dots, he wins and the game ends.

We had hopes to fully fill the maze with the food dots, however as we began to run out of resources on our fpga, we realised this was not feasible and had to reduce the number of food dots on the screen. This meant the game was fully functional, but lacking some aesthetics.

How could you further develop the game

To further develop the game, We could attempt to add more food dots, but instead of placing them inside always comb blocks and drawing each one out, we could use an array to mark the location and then a for loop that would draw out each dot. This would aim to use less resources and would mean we would be able to add more food dots.

We also could add sprites to animate pacman and give the ghosts their characteristic shape. We chose not to add sprites to our game as we began to run out of time and prioritised functionality of the game over its overall look. Had we added sprites to our game, it would have required us to change the boundary conditions of both pacman and the ghosts as well as collision detection.

Another way we could build on our current design would be to add power-ups. The way we intended to do this was to add some food dots and change their colour. If Pacman had been on the coordinate of the dot it would have moved each of the ghosts to a frightened state where it would not be able to kill pacman. This could be achieved by adding another state to each of the ghosts' controllers and no matter what state it is in, give it a condition to show that if Pacman is on the power up coordinate, then move to the frightened state.

Pitfalls in the Design

Throughout our development of the final project, we came across a few drawbacks. Originally we had hard-coded the maze design for pacman. However, once we did this, we were unable to get Pacman to interact with the map, as it kept moving across the boundaries. In the end we decided to use block memory. This allowed us to be able to access the memory address of each dot in the maze, which meant we knew when Pacman or a ghost was going to overlap with the maze.

Another setback was with the development of the ghost AI. We began by designing the scatter and chase movement of one ghost. When it entered the chase state, it would get stuck in a wall and not move again until the scatter state. This meant we had to implement a look ahead for each of the ghost's directions. This enabled the ghost to look ahead and check if there was going to be a boundary a pixel ahead and not just at its current location.

Conclusion

We successfully designed a version of Pacman, developing our code only in hardware. We drew on our past experience of System Verilog and the labs leading up to the final project to implement our game. While there were a few drawbacks in our design process along the way, in the end we have a fully functioning Pacman game.

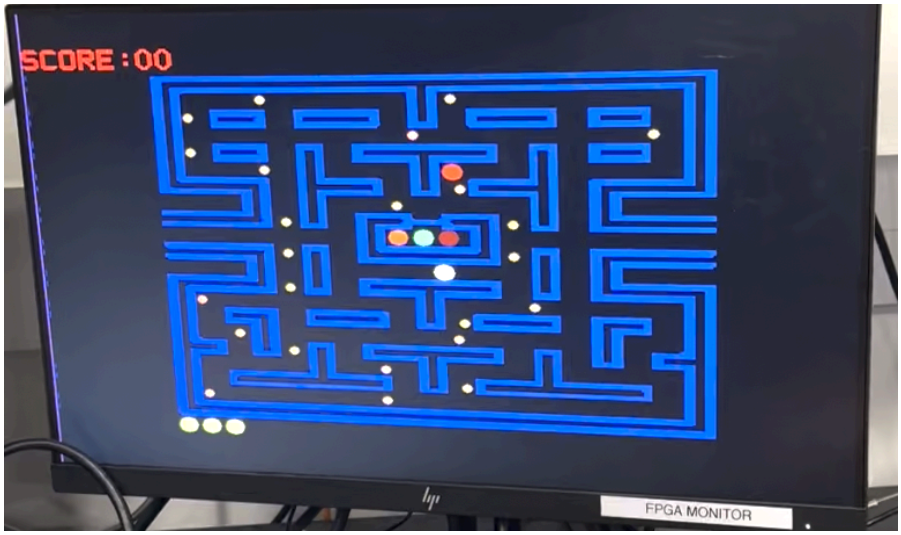

Figure 5: Final Version of Pacman Game