

Design Procedure

The purpose of this lab is to implement what we have learned through this class by creating a fun/cool project that utilizes the DE1-SoC board and peripherals! We chose to create a classic Pacman game.



We are developing a classic Pacman game using SystemVerilog, a hardware description language, to model the electronic systems on an FPGA (Field-Programmable Gate Array). The game will be displayed on a VGA (Video Graphics Array) screen, and the Pacman character will be controlled using the N8 input controller.

Core Features

- Maze Display: The game will feature a maze layout, similar to the classic Pacman games. This maze will be displayed on the VGA screen.
- Pacman Movement: The Pacman character will be controllable in four directions (up, down, left, right) using the N8 controller.
- Ghosts: There will be a ghost in the maze that moves chasing the Pacman. If Pacman collides with any of the ghosts, the game will end.
- Food: Small food pellets will be placed throughout the maze. Pacman must eat all the food to win the game.

Maze

In creating the maze for the Pacman game, we utilize a given VGA Driver. This driver provides x- and y-coordinates of the current pixel, where the origin is the upper-left corner. The user, in turn, provides the driver with a color to draw at that pixel, which must be valid within two clock cycles of the coordinates changing. Colors are expressed as three 8-bit unsigned integers, each specifying the light intensity in its respective color channel: red, green, or blue.

Since the VGA Driver employs a "scanning" method, rather than the user specifying the color (black or white) of an arbitrary pixel, the driver cycles through one pixel at a time based on the outputs x and y. The user then supplies the RGB color for that pixel. For our maze, we need to provide the specific color, which is blue, to the given coordinates. Thus, as the VGA Driver scans through each pixel, we will supply the blue color for the pixels that form the maze walls.

First, we are using a screen size of 96x72 pixels. The Pacman character will be 3x3 pixels. We scale Pacman to be 3x3 instead of 1x1 because we need to create food items that are smaller than Pacman. Therefore, Pacman cannot be the smallest element. The food items will be 1x1 pixel, positioned in the middle of Pacman's 3x3 grid.

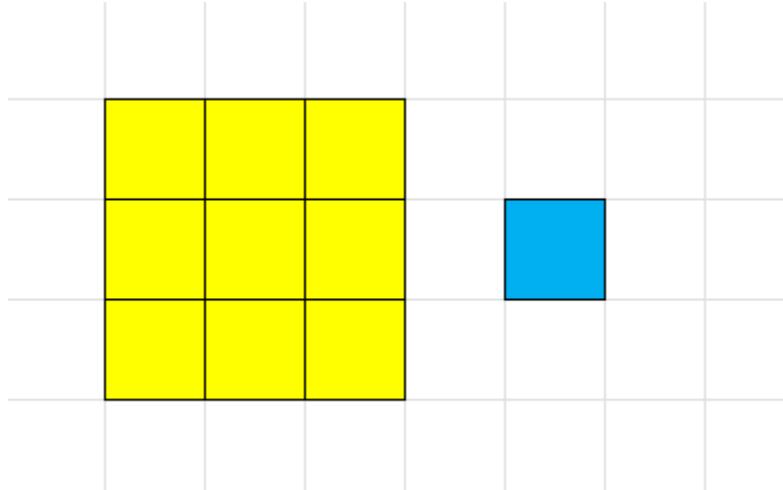


Figure 1. The pixels size for the Pacman and the food

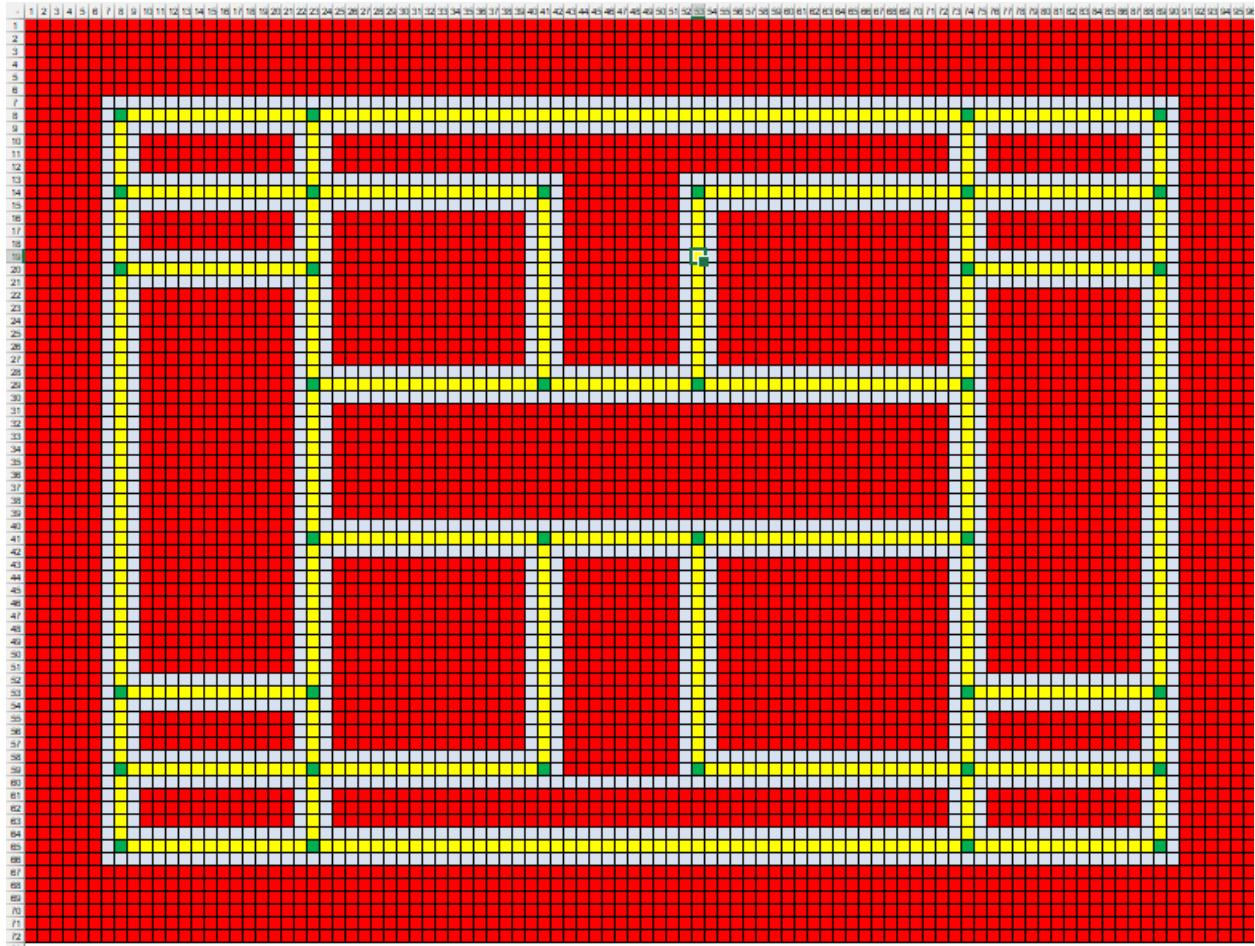


Figure 2. The coordinates for the pacman games.

In the given screenshots, the colors represent various elements of the Pacman game as follows:

- Red represents the maze walls.
- Yellow represents the path movement of Pacman.
- Blue represents the edge tile because Pacman is 3x3 pixels in size.
- Green represents intersections or edge cases.

By using these color codes, we can clearly differentiate between the maze walls, Pacman's path, edge tiles, and intersections. This color scheme helps in visually organizing the game's layout and ensuring that each element is easily identifiable.

```

1 module maze (x, y, r, g, b);
2   input logic [9:0] x;
3   input logic [8:0] y;
4   output logic [7:0] r, g, b;
5
6   always_comb begin
7     // Default color
8     r = 0;
9     g = 0;
10    b = 0;
11
12    if(y == 0 || y == 1 || y == 2 || y == 3 || y == 4 || y == 5 ||
13      y == 66 || y == 67 || y == 68 || y == 69 || y == 70 || y == 71) begin
14      r <= 0;
15      g <= 0;
16      b <= 255;
17    end
18
19    else if(x == 0 || x == 1 || x == 2 || x == 3 || x == 4 || x == 5 ||
20      x == 90 || x == 91 || x == 92 || x == 93 || x == 94 || x == 95) begin
21      r <= 0;
22      g <= 0;
23      b <= 255;
24    end
25
26    // 1
27    else if (y >= 9 && y <= 11 && x >= 9 && x <= 20 || y >= 9 && y <= 11 && x >= 24 && x <= 71 ||
28      y >= 90 && y <= 11 && x >= 75 && x <= 86) begin
29      r <= 0;
30      g <= 0;
31      b <= 255;
32    end
33
34    else if (y >= 60 && y <= 62 && x >= 9 && x <= 20 || y >= 60 && y <= 62 && x >= 24 && x <= 71 ||
35      y >= 60 && y <= 62 && x >= 75 && x <= 86) begin
36      r <= 0;
37      g <= 0;
38      b <= 255;
39    end
40
41    // 2
42    else if (y >= 12 && y <= 14 && x >= 12 && x <= 20) begin
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127 endmodule

```

Figure 3. Systemverilog code of maze

Following the figure 2, we can easily hard-code which x and y coordinates that are going to be colored blue to build the maze.

Pacman

For the pacman, we have 5 main modules that are utilized.

The first module is `pacman_cont`. This module determines the state movement of Pacman, providing the necessary signals to the datapath.

Next is `pacman_datapath`, which controls the data according to the state determined by `pacman_cont`. This module manages the actual movement of Pacman within the game.

The `pacman_display` module is responsible for displaying Pacman at a scale of 3x3 pixels instead of 1x1, ensuring Pacman is visually larger and distinct on the screen.

The `user_direction_hold` module keeps Pacman moving in the specified direction without the need for the user to repeatedly input the direction. This allows for smoother and more continuous movement based on the user's initial input.

Finally, the `pacman` module is the top-level module that integrates all the Pacman-related modules, coordinating their functions to create the complete game.

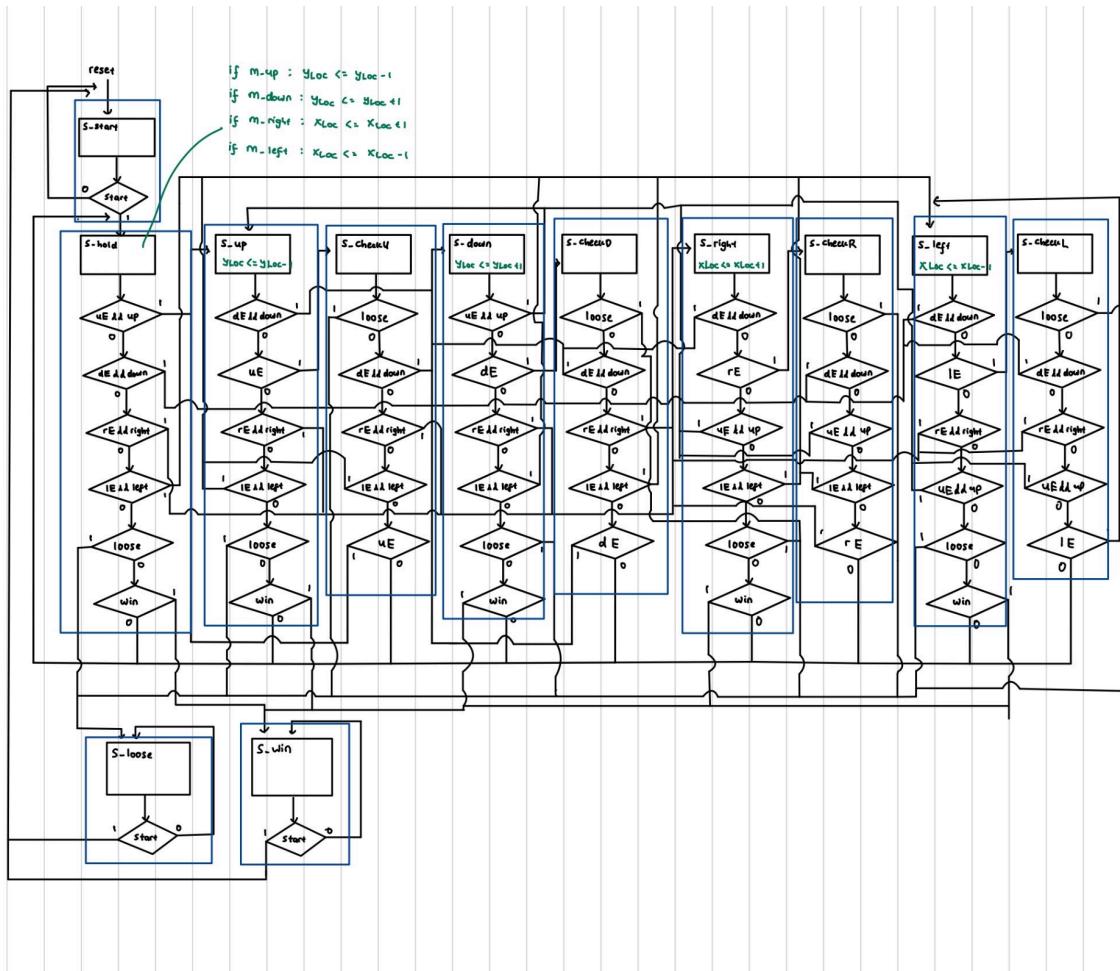


Figure 4. ASMD for Pacman

```

Text Editor - C:/Users/USER/Downloads/lab6_pacmanV1/lab6 pacman - Copy/DE1_SoC - DE1_SoC - [pacMan_cont.v]
File Edit View Project Processing Tool Window Help
Search altera.com

1 // Pacman control module to control the state of the pacman which include
2 // s_start, s_hold, s_right, s_checkU, s_down, s_checkD,
3 // s_right, s_checkR, s_left, s_checkL, s_win, s_loose
4 module pacMan_ctrl(
5   input logic clk,
6   output logic reset,
7   output logic win, loose,
8   output logic dE, rE, IE, //maze enable signals
9   output logic up, down, left, right, start, //controller signals
10  output logic m_up, m_down, m_right, m_left, e_start, m_hold); // output moves
11
12
13 //state variables
14 enum{ s_start, s_hold,
15       s_up, s_down, s_right, s_left,
16       s_checkU, s_checkD, s_checkR, s_checkL,
17       s_win, s_loose} ps, ns;
18
19
20 // Controller logic w/synchronous reset
21 always_ff @(posedge clk)
22   if (reset)
23     ps <= s_start;
24   else
25     ps <= ns;
26
27
28 // Next state logic constant movement
29 always_comb
30 begin
31   case(ps)
32     case(ps)
33       s_start: ns = start ? s_hold : s_start;
34
35       s_up:   if (dE && down) ns = s_down;
36       else if (rE) ns = s_checkU;
37       else if (rE && right) ns = s_right;
38       else if (IE && left) ns = s_left;
39       else if (loose) ns = s_loose;
40       else if (win) ns = s_win;
41       else ns = s_hold;
42
43       s_checkU:if (loose) ns = s_loose;
44
45   endcase
46 end

```

```

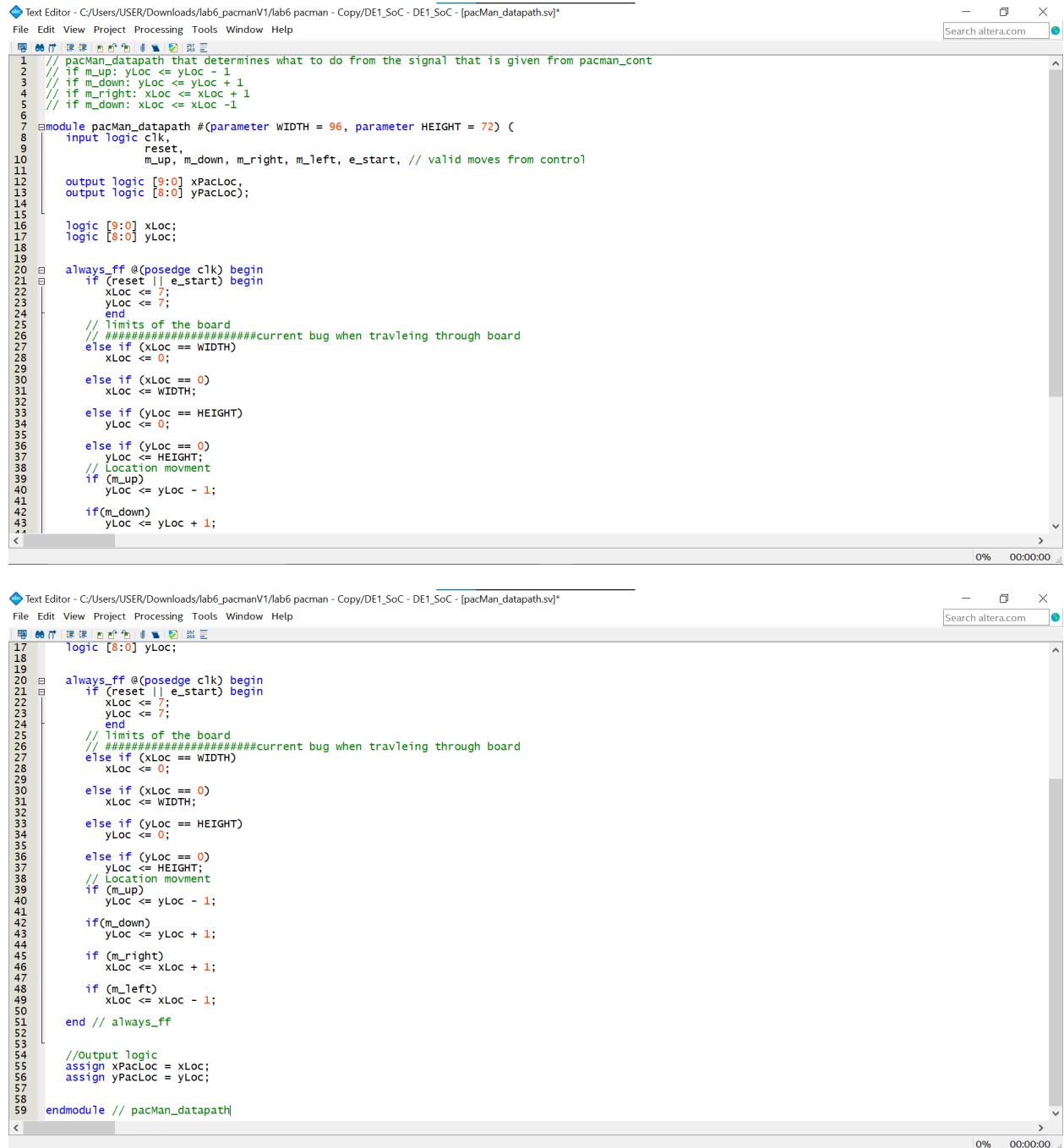
Text Editor - C:/Users/USER/Downloads/lab6_pacmanV1/lab6 pacman - Copy/DE1_SoC - DE1_SoC - [pacMan_cont.sv]*

File Edit View Project Processing Tools Window Help
Search altera.com

29 // Next state logic constant movement
30 always_comb
31 begin
32     case(ps)
33         s_start: ns = start ? s_hold : s_start;
34
35         s_up:    if (de && down) ns = s_down;
36         else if (ue) ns = s_checkU;
37         else if (re && right) ns = s_right;
38         else if (!e && left) ns = s_left;
39         else if (loose) ns = s_loose;
40         else if (win) ns = s_win;
41         else ns = s_hold;
42
43         s_checkU:if (!loose) ns = s_loose;
44         else if (de && down) ns = s_down;
45         else if (re && right) ns = s_right;
46         else if (!e && left) ns = s_left;
47         else if (ue) ns = s_up;
48         else ns = s_hold;
49
50         s_down:   if (ue && up) ns = s_up;
51         else if (de && ns == s_checkD);
52         else if (re && right) ns = s_right;
53         else if (!e && left) ns = s_left;
54         else if (loose) ns = s_loose;
55         else if (win) ns = s_win;
56         else ns = s_hold;
57
58         s_checkD:if (!loose) ns = s_loose;
59         else if (ue && up) ns = s_up;
60         else if (re && right) ns = s_right;
61         else if (!e && left) ns = s_left;
62         else if (de) ns = s_down;
63         else ns = s_hold;
64
65         s_right:  if (!e && left) ns = s_left;
66         else if (re) ns = s_checkR;
67         else if (ue && up) ns = s_up;
68         else if (de && down) ns = s_down;
69         else if (loose) ns = s_loose;
70         else if (win) ns = s_win;
71         else ns = s_hold;
72
73
74         s_left:   if (re && right) ns = s_right;
75         else if (!e) ns = s_checkL;
76         else if (ue && up) ns = s_up;
77         else if (de && down) ns = s_down;
78         else if (loose) ns = s_loose;
79         else if (win) ns = s_win;
80         else ns = s_hold;
81
82         s_checkL:if (loose) ns = s_loose;
83         else if (re && right) ns = s_right;
84         else if (ue && up) ns = s_up;
85         else if (de && down) ns = s_down;
86         else if (!e) ns = s_left;
87         else ns = s_hold;
88
89         s_hold:   if (ue && up) ns = s_up;
90         else if (de && down) ns = s_down;
91         else if (re && right) ns = s_right;
92         else if (!e && left) ns = s_left;
93         else if (loose) ns = s_loose;
94         else if (win) ns = s_win;
95         else ns = s_hold;
96
97         s_loose: ns = start ? s_start : s_loose;
98         s_win:   ns = start ? s_start : s_win;
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122 endmodule
123

```

Figure 5. Systemverilog for pacman_cont



```

1 // pacMan_datapath that determines what to do from the signal that is given from pacman_cont
2 // if m_up: yLoc <= yLoc - 1
3 // if m_down: yLoc <= yLoc + 1
4 // if m_right: xLoc <= xLoc + 1
5 // if m_left: xLoc <= xLoc -1
6
7 module pacMan_datapath #(parameter WIDTH = 96, parameter HEIGHT = 72) (
8   input logic clk,
9   input logic reset,
10  input logic m_up, m_down, m_right, m_left, e_start, // valid moves from control
11
12  output logic [9:0] xPacLoc,
13  output logic [8:0] yPacLoc);
14
15
16  logic [9:0] xLoc;
17  logic [8:0] yLoc;
18
19
20  always_ff @(posedge clk) begin
21    if (reset || e_start) begin
22      xLoc <= 7;
23      yLoc <= 7;
24    end
25    // limits of the board
26    // #####current bug when travleing through board
27    else if (xLoc == WIDTH)
28      xLoc <= 0;
29
30    else if (xLoc == 0)
31      xLoc <= WIDTH;
32
33    else if (yLoc == HEIGHT)
34      yLoc <= 0;
35
36    else if (yLoc == 0)
37      yLoc <= HEIGHT;
38    // Location movement
39    if (m_up)
40      yLoc <= yLoc - 1;
41
42    if(m_down)
43      yLoc <= yLoc + 1;
44
45    if (m_right)
46      xLoc <= xLoc + 1;
47
48    if (m_left)
49      xLoc <= xLoc - 1;
50
51  end // always_ff
52
53
54  //Output logic
55  assign xPacLoc = xLoc;
56  assign yPacLoc = yLoc;
57
58
59 endmodule // pacMan_datapath

```

Figure 6. Systemverilog for pacman_datapath

The screenshot shows a text editor window with the title "Text Editor - C:/Users/USER/Downloads/lab6_pacmanV1/lab6 pacman - Copy/DE1_SoC - DE1_SoC - [pacMan_display.sv]*". The menu bar includes File, Edit, View, Project, Processing, Tools, Window, and Help. A search bar at the top right says "Search altera.com". The code in the editor is:

```
1 // This module is to display the pacman with scale 3x3 pixels.
2 module pacMan_display (
3   input logic clk,
4   input logic reset,
5   input logic m_up, m_down, m_right, m_left, e_start, m_hold,
6   input logic [9:0] x, xPacLoc,
7   input logic [8:0] y, yPacLoc,
8   output logic [7:0]r, g, b);
9
10  always_comb
11    begin
12      if (x == xPacLoc + 1 && y >= yPacLoc - 1 || x == xPacLoc && y >= yPacLoc + 1
13      || x == xPacLoc - 1 && y >= yPacLoc - 1 || x == xPacLoc + 1 && y <= yPacLoc + 1)
14        r = 255;
15        g = 255;
16        b = 255;
17      end
18      else begin
19        r = 0;
20        g = 0;
21        b = 0;
22      end
23    end
24
25
26
27 endmodule // pacMan_display
```

Figure 7. Systemverilog for pacman_display

The screenshot shows a text editor window with the title "Text Editor - C:/Users/USER/Downloads/lab6_pacmanV1/lab6 pacman - Copy/DE1_SoC - DE1_SoC - [SystemVerilog1.sv]*". The menu bar includes File, Edit, View, Project, Processing, Tools, Window, and Help. A search bar at the top right says "Search altera.com". The code in the editor is:

```
1 // holds user input until next user input
2 // also decides start direction
3 module user_direction_hold(
4   input logic clk, reset,
5   input logic upIn, downIn, leftIn, rightIn, startIn,
6   output logic upOut, downOut, leftOut, rightOut);
7
8
9   logic up, down, left, right;
10
11  always_ff @(posedge clk) begin
12    if (reset) begin
13      up <= 0;
14      down <= 0;
15      left <= 0;
16      right <= 0;
17    end
18    else if (startIn) begin
19      up <= 0;
20      down <= 0;
21      left <= 0;
22      right <= 0;
23    end
24    else if (upIn) begin
25      up <= 1;
26      down <= 0;
27      left <= 0;
28      right <= 0;
29    end
30    else if (downIn) begin
31      up <= 0;
32      down <= 1;
33      left <= 0;
34      right <= 0;
35    end
36    else if (leftIn) begin
37      up <= 0;
38      down <= 0;
39      left <= 1;
40      right <= 0;
41    end
42    else if (rightIn) begin
43      up <= 0;
44      down <= 0;
```

Figure 8. Systemverilog for user_direction_hold

```

1 // Top Level module that connects the pacman_cont, pacman_datapath, user_direction_hold, and pacman_display
2 module pacMan #(parameter WIDTH = 96, parameter HEIGHT = 72) (
3   input logic      clk,
4           reset,
5           win, loose,
6           ue, de, rE, lE, //maze enable signals
7           up, down, left, right, start, // controller signals
8           input logic [9:0] x, // from display driver
9           input logic [8:0] y,
10
11  output logic [9:0] xPacLoc,
12  output logic [8:0] yPacLoc,
13  output logic [7:0] r, g, b;
14
15 //internal wires
16 logic m_up, m_down, m_right, m_left, e_start, m_hold;
17
18 pacMan_cont c_unit (//inputs
19   .clk,
20   .reset,
21   .win, loose,
22   .ue, .de, .rE, .lE, //maze enable signals
23   .up, .down, .left, .right, .start, // controller signals
24   //outputs
25   .m_up, .m_down, .m_right, .m_left, .e_start, .m_hold); // output moves
26
27 pacMan_datapath#(.WIDTH(96), .HEIGHT(72)) d_unit (
28   .clk,
29   .reset,
30   .m_up, .m_down, .m_right, .m_left, .e_start, // valid moves from control
31   //outputs
32   .xPacLoc,
33   .yPacLoc);
34
35 pacMan_display pDis ( //inputs
36   .clk,
37   .reset,
38   .m_up, .m_down, .m_right, .m_left, .e_start, .m_hold,
39   .x, .xPacLoc,
40   .y, .yPacLoc,
41   //outputs
42   .r, .g, .b);
43
44 endmodule

```

Figure 9. Systemverilog for pacman

In summary, Pacman will move according to the user's input, which can be one of four directions: left, up, right, or down. The user does not need to repeatedly input commands to keep Pacman moving. Once a direction is specified, Pacman will continue moving in that direction until it encounters a wall.

Here's how the movement system works in detail:

- Direction Input: The user inputs a direction (left, up, right, or down) using the control interface.
- Continuous Movement: The `user_direction_hold` module ensures that Pacman keeps moving in the specified direction continuously. This eliminates the need for the user to repeatedly input the same command to keep Pacman moving.
- State Management: The `pacman_cont` module manages Pacman's state, determining the direction of movement based on the user's input and the current state of the game.
- Datapath Control: The `pacman_datapath` module controls Pacman's movement on the screen, updating Pacman's position according to the current state and direction of movement.
- Collision Detection: Pacman will continue to move in the specified direction until it encounters a wall. The game checks for collisions with the maze walls and stops Pacman's movement when a wall is hit.
- Visual Representation: The `pacman_display` module renders Pacman on the screen, scaled to 3x3 pixels for better visibility. This module ensures that Pacman's movements are accurately displayed according to the direction and position provided by the datapath.
- Top-Level Integration: The `pacman` module integrates all these functionalities, ensuring that the user's input is processed, Pacman's state is managed, movements are controlled, and the display is updated in a cohesive manner.

This system provides a seamless and intuitive control experience, allowing the user to simply specify the direction and watch Pacman move continuously until an obstacle is encountered.

Maze logic

After, we have implemented the pacman, now we want to create the maze logic because we don't want the pacman to be able to go through the maze. The pacman can only go in the path which is shown in the yellow block in figure 2. To implement this, we hard coded the logic according to the position of the pacman.

```

1 // This module takes the xLoc and yLoc and determine which direction the object can go.
2 // This module will prevent the object to go through the maze.
3 // By default all the direction is set to 0.
4 module maze_logic(
5   input logic [9:0] xLoc,
6   input logic [8:0] yLoc,
7   output logic ue, de, re, te); //enable signals
8
9   always_comb begin
10
11     ue = 0;
12     de = 0;
13     re = 0;
14     te = 0;
15
16     // Horizontal top
17     if (yLoc == 7 && xLoc >= 8 && xLoc <= 21 || yLoc == 7 && xLoc >= 23 && xLoc <= 27 || yLoc == 7 && xLoc >= 74 && xLoc <= 87) begin
18       re <= 1;
19       te <= 1;
20     end
21
22     // Horizontal bottom
23     else if (yLoc == 64 && xLoc >= 8 && xLoc <= 21 || yLoc == 64 && xLoc >= 23 && xLoc <= 27 || yLoc == 64 && xLoc >= 74 && xLoc <= 87) begin
24       re <= 1;
25       te <= 1;
26     end
27
28     // Vertical left
29     else if (xLoc == 7 && yLoc >= 8 && yLoc <= 12 || xLoc == 7 && yLoc >= 14 && yLoc <= 18 || xLoc == 7 && yLoc >= 20 && yLoc <= 51 || xLoc == 7 && yLoc >= 53 && yLoc <= 57 || xLoc == 7 && yLoc >= 59 && yLoc <= 63) begin
30       de <= 1;
31       ue <= 1;
32     end
33
34     // Vertical right
35     else if (xLoc == 88 && yLoc >= 8 && yLoc <= 12 || xLoc == 88 && yLoc >= 14 && yLoc <= 18 || xLoc == 88 && yLoc >= 20 && yLoc <= 51 || xLoc == 88 && yLoc >= 53 && yLoc <= 57 || xLoc == 88 && yLoc >= 59 && yLoc <= 63) begin
36       de <= 1;
37       ue <= 1;
38     end
39
40     // Vertical first column (22) and (73)
41     else if (xLoc == 22 && yLoc >= 8 && yLoc <= 12 || xLoc == 22 && yLoc >= 14 && yLoc <= 18 || xLoc == 22 && yLoc >= 20 && yLoc <= 51 || xLoc == 22 && yLoc >= 53 && yLoc <= 57 || xLoc == 22 && yLoc >= 59 && yLoc <= 63) begin
42       de <= 1;
43       ue <= 1;
44     end
45
46     // Vertical first column (22) and (73)
47     else if (xLoc == 73 && yLoc >= 8 && yLoc <= 12 || xLoc == 73 && yLoc >= 14 && yLoc <= 18 || xLoc == 73 && yLoc >= 20 && yLoc <= 51 || xLoc == 73 && yLoc >= 53 && yLoc <= 57 || xLoc == 73 && yLoc >= 59 && yLoc <= 63) begin
48       de <= 1;
49       ue <= 1;
50     end
51
52     // Vertical first column (24) and (52)
53     else if (xLoc == 24 && yLoc >= 8 && yLoc <= 12 || xLoc == 24 && yLoc >= 14 && yLoc <= 18 || xLoc == 24 && yLoc >= 20 && yLoc <= 51 || xLoc == 24 && yLoc >= 53 && yLoc <= 57 || xLoc == 24 && yLoc >= 59 && yLoc <= 63) begin
54       de <= 1;
55       ue <= 1;
56     end
57
58     // Vertical second column (41) and (52)
59     else if (xLoc == 41 && yLoc >= 8 && yLoc <= 12 || xLoc == 41 && yLoc >= 14 && yLoc <= 18 || xLoc == 41 && yLoc >= 20 && yLoc <= 51 || xLoc == 41 && yLoc >= 53 && yLoc <= 57 || xLoc == 41 && yLoc >= 59 && yLoc <= 63) begin
60       de <= 1;
61       ue <= 1;
62     end
63
64
65     // Horizontal first row (13) and (59)
66     else if (yLoc == 13 && xLoc >= 8 && xLoc <= 21 || yLoc == 13 && xLoc >= 23 && xLoc <= 39 || yLoc == 13 && xLoc >= 55 && xLoc <= 72 || yLoc == 13 && xLoc >= 74 && xLoc <= 87) begin
67       re <= 1;
68       te <= 1;
69     end
70
71
72
73
74
75
76
77
78
79
80
81

```

Figure 10. Systemverilog for maze_logic

By implementing this module, Pacman is prevented from moving through the maze walls because any direction leading to a wall will be set to 0.

Food

Next, using the same approach we used to build the maze, we will create the food. The food will be placed every three pixels starting from the edge, colored yellow, and sized 1x1. We also need

to implement additional logic so that when Pacman eats the food (when the food location coincides with Pacman's location), it will disappear. We'll use a memory array to track the food state, where 1 indicates food is present and 0 indicates it has been eaten. We assign 1 to every coordinate where we want to place food. When Pacman collides with these coordinates, we assign 0. Finally, we assign the RGB color based on the flag: if it's 1, we assign yellow; otherwise, black. This way, the food will be displayed and will disappear once Pacman eats it.

```
Text Editor - C:/Users/USER/Downloads/lab6_pacmanV1/lab6 pacman - Copy/DE1_SoC - DE1_SoC - [food.sv]*
File Edit View Project Processing Tools Window Help
Search altera.com

1 // This modules display in the path every 3 pixels (yellow colored)
2 // once the food is eaten, it deleted the Food
3 module food (x, y, xPacLoc, yPacLoc, r, g, b, clk, gamewin);
4
5     input logic clk;
6     input logic [9:0] x, xPacLoc;
7     input logic [8:0] y, yPacLoc;
8     output logic [7:0] r, g, b;
9     output logic gamewin;
10    logic [7:0] food_count;
11
12    // Memory array to track food state (1 if food is present, 0 if food is eaten)
13    logic food_state[96][72]; // Assuming VGA resolution, adjust size as needed
14    // Initialize the food state array
15    initial begin
16        food_count = 0;
17        for (int i = 0; i < 96; i++) begin
18            for (int j = 0; j < 72; j++) begin
19                food_state[i][j] = 0;
20            end
21        end
22
23        // Horizontal
24        for (int i = 7; i <= 89; i = i + 3) begin
25            food_state[i][7] = 1;
26            food_count = food_count + 1;
27        end
28
29        for (int i = 7; i <= 89; i = i + 3) begin
30            food_state[i][64] = 1;
31            food_count = food_count + 1;
32        end
33
34        for (int i = 7; i <= 40; i = i + 3) begin
35            food_state[i][13] = 1;
36            food_count = food_count + 1;
37        end
38
39        for (int i = 52; i <= 88; i = i + 3) begin
40            food_state[i][13] = 1;
41            food_count = food_count + 1;
42        end
43
44        // ... (remaining code for food placement)


```

Ln 11 Col 1 SystemVerilog HDL File 0% 00:00:00

```
Text Editor - C:/Users/USER/Downloads/lab6_pacmanV1/lab6 pacman - Copy/DE1_SoC - DE1_SoC - [food.sv]*
File Edit View Project Processing Tools Window Help
Search altera.com

119    for (int j = 40; j <= 58; j = j + 3) begin
120        food_state[52][j] = 1;
121        food_count = food_count + 1;
122    end
123
124
125    end
126
127    // Update food state on Pacman collision
128    always_ff @(posedge clk) begin
129        if (food_state[xPacLoc][yPacLoc]) begin
130            food_state[xPacLoc][yPacLoc] <= 0; // Mark food as eaten
131            food_count = food_count - 1;
132        end
133    end
134
135    // Set pixel color based on food state
136    always_comb begin
137        // Default color (black)
138        r = 0;
139        g = 0;
140        b = 0;
141
142        // Check food state for the current pixel
143        if (food_state[x][y]) begin
144            // Food color (yellow)
145            r = 255;
146            g = 255;
147            b = 0;
148        end
149    end
150
151    // Set gamewin signal based on food count
152    always_comb begin
153        if (food_count == 0) begin
154            gamewin = 1;
155        end else begin
156            gamewin = 0;
157        end
158    end
159
160
161 endmodule


```

0% 00:00:00

Figure 11. Systemverilog for food

Ghost

Now, for ghosts, the implementation is similar to the pacman. However, instead of having the user input to control the movement, we created the ghost logic module to determine the ghost movement because we want the ghost to chase the pacman.

```

1 // ghost control module to control the state of the ghost which include (similar to pacman)
2 // s_start, s_hold, s_right, s_left, s_checkU, s_down, s_checkL,
3 // s_right, s_checkR, s_left, s_checkL, s_win, s_loose
4 module ghost_cont(
5   input logic clk,
6   input logic reset,
7   input logic win, loose,
8   input logic dE, rE, lE, //maze enable signals
9   input logic up, down, left, right, start, //controller signals
10  output logic m_up, m_down, m_right, m_left, e_start, m_hold); // output moves
11
12
13 //state variables
14 enum{ s_start, s_hold,
15       s_up, s_down, s_right, s_left,
16       s_checkU, s_checkL, s_checkR, s_checkL,
17       s_win, s_loose} ps, ns;
18
19
20 // Controller logic w/synchronous reset
21 always_ff @(posedge clk)
22   if (reset)
23     ps <= s_start;
24   else
25     ps <= ns;
26
27
28 // Next state Logic constant movement
29 always_comb
30 begin
31   case(ps)
32     s_start: ns = start ? s_hold : s_start;
33
34     s_up:   if (dE && down) ns = s_down;
35     else if (uE) ns = s_checkU;
36     else if (rE && right) ns = s_right;
37     else if (lE && left) ns = s_left;
38     else if (loose) ns = s_loose;
39     else if (win) ns = s_win;
40     else ns = s_hold;
41
42     s_checkU:if (loose) ns = s_loose;
43     else if (win) ns = s_win;
44
45     s_checkL:if (loose) ns = s_loose;
46     else if (rE && right) ns = s_right;
47     else if (uE && up) ns = s_up;
48     else if (dE && down) ns = s_down;
49     else if (loose) ns = s_loose;
50     else if (win) ns = s_win;
51     else ns = s_hold;
52
53     s_right: if (rE && right) ns = s_right;
54     else if (lE) ns = s_checkL;
55     else if (uE && up) ns = s_up;
56     else if (dE && down) ns = s_down;
57     else if (loose) ns = s_loose;
58     else if (win) ns = s_win;
59     else ns = s_hold;
60
61     s_left:  if (rE && right) ns = s_right;
62     else if (lE) ns = s_checkL;
63     else if (uE && up) ns = s_up;
64     else if (dE && down) ns = s_down;
65     else if (lE) ns = s_left;
66     else ns = s_hold;
67
68     s_loose: ns = start ? s_start : s_loose;
69
70     s_win:  ns = start ? s_start : s_win;
71
72   endcase
73 end
74
75 //output logic
76 assign m_up = (ps == s_up);
77 assign m_down = (ps == s_down);
78 assign m_right = (ps == s_right);
79 assign m_left = (ps == s_left);
80 assign e_start = (ps == s_start);
81 assign m_hold = (ps == s_hold);
82
83
84 endmodule

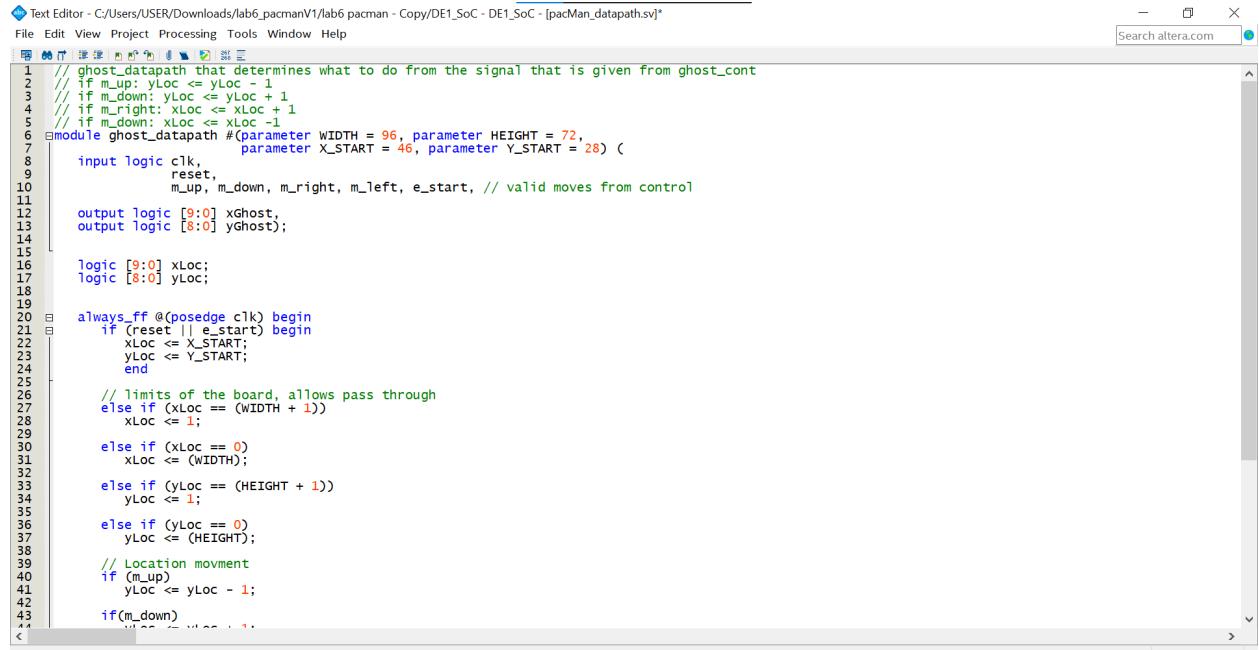
```

```

80   s_left:  if (rE && right) ns = s_right;
81   else if (lE) ns = s_checkL;
82   else if (uE && up) ns = s_up;
83   else if (dE && down) ns = s_down;
84   else if (loose) ns = s_loose;
85   else if (win) ns = s_win;
86   else ns = s_hold;
87
88   s_checkL:if (loose) ns = s_loose;
89   else if (rE && right) ns = s_right;
90   else if (uE && up) ns = s_up;
91   else if (dE && down) ns = s_down;
92   else if (lE) ns = s_left;
93   else ns = s_hold;
94
95   s_hold:  if (uE && up) ns = s_up;
96   else if (dE && down) ns = s_down;
97   else if (rE && right) ns = s_right;
98   else if (lE && left) ns = s_left;
99   else if (loose) ns = s_loose;
100  else if (win) ns = s_win;
101  else ns = s_hold;
102
103
104  s_loose: ns = start ? s_start : s_loose;
105
106  s_win:  ns = start ? s_start : s_win;
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122 endmodule

```

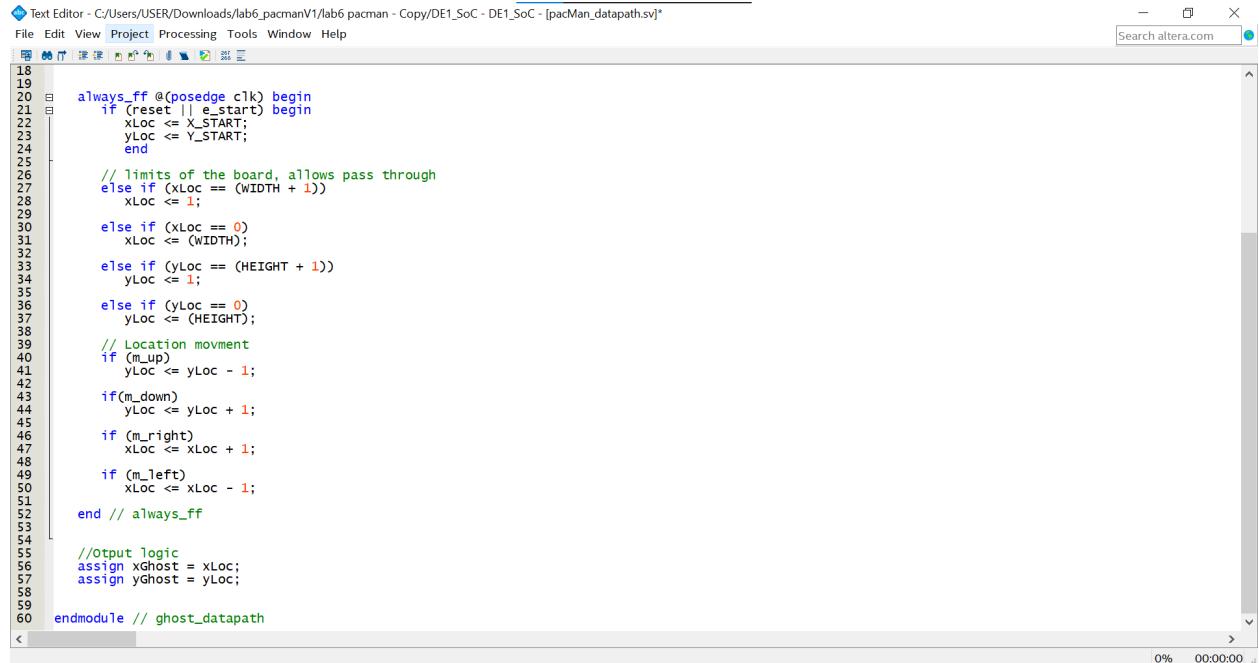
Figure 12. Systemverilog for ghost_cont



```

1 // ghost_datapath that determines what to do from the signal that is given from ghost_ctrl
2 // if m_up: yLoc <= yLoc - 1
3 // if m_down: yLoc <= yLoc + 1
4 // if m_right: xLoc <= xLoc + 1
5 // if m_left: xLoc <= xLoc - 1
6 module ghost_datapath #(parameter WIDTH = 96, parameter HEIGHT = 72,
7                             parameter X_START = 46, parameter Y_START = 28) (
8     input logic clk,
9         reset,
10    m_up, m_down, m_right, m_left, e_start, // valid moves from control
11
12    output logic [9:0] xGhost,
13    output logic [8:0] yGhost;
14
15    logic [9:0] xLoc;
16    logic [8:0] yLoc;
17
18
19    always_ff @(posedge clk) begin
20        if (reset || e_start) begin
21            xLoc <= X_START;
22            yLoc <= Y_START;
23        end
24
25        // limits of the board, allows pass through
26        else if (xLoc == (WIDTH + 1))
27            xLoc <= 1;
28
29        else if (xLoc == 0)
30            xLoc <= (WIDTH);
31
32        else if (yLoc == (HEIGHT + 1))
33            yLoc <= 1;
34
35        else if (yLoc == 0)
36            yLoc <= (HEIGHT);
37
38        // Location movement
39        if (m_up)
40            yLoc <= yLoc - 1;
41
42        if(m_down)
43            yLoc <= yLoc + 1;
44
45        if (m_right)
46            xLoc <= xLoc + 1;
47
48        if (m_left)
49            xLoc <= xLoc - 1;
50
51    end // always_ff
52
53
54    //Output logic
55    assign xghost = xLoc;
56    assign yGhost = yLoc;
57
58
59
60 endmodule // ghost_datapath

```



```

18
19
20    always_ff @(posedge clk) begin
21        if (reset || e_start) begin
22            xLoc <= X_START;
23            yLoc <= Y_START;
24        end
25
26        // limits of the board, allows pass through
27        else if (xLoc == (WIDTH + 1))
28            xLoc <= 1;
29
30        else if (xLoc == 0)
31            xLoc <= (WIDTH);
32
33        else if (yLoc == (HEIGHT + 1))
34            yLoc <= 1;
35
36        else if (yLoc == 0)
37            yLoc <= (HEIGHT);
38
39        // Location movement
40        if (m_up)
41            yLoc <= yLoc - 1;
42
43        if(m_down)
44            yLoc <= yLoc + 1;
45
46        if (m_right)
47            xLoc <= xLoc + 1;
48
49        if (m_left)
50            xLoc <= xLoc - 1;
51
52    end // always_ff
53
54
55    //Output logic
56    assign xghost = xLoc;
57    assign yGhost = yLoc;
58
59
60 endmodule // ghost_datapath

```

Figure 13. Systemverilog for ghost_datapath

The screenshot shows a text editor window with the title "Text Editor - C:/Users/USER/Downloads/lab6_pacmanV1/lab6 pacman - Copy/DE1_SoC - DE1_SoC - [SystemVerilog2.sv]*". The menu bar includes File, Edit, View, Project, Processing, Tools, Window, and Help. A search bar at the top right says "Search altera.com". The code is as follows:

```
1 // This module is to display the ghost with scale 3x3 pixels.
2 module ghost_display #(parameter RED = 255, parameter GRN = 0,parameter BLUE = 0)(
3     input logic clk,
4         reset,
5     m_up, m_down, m_right, m_left, e_start, m_hold,
6     input logic [9:0] x, xGhost,
7     input logic [8:0] y, yGhost,
8     output logic [7:0]r, g, b);
9
10    always_comb
11        begin
12            if (x == xGhost + 1 && y >= yGhost - 1 || x == xGhost && y >= yGhost + 1
13            || x == xGhost - 1 && y >= yGhost - 1 && y <= yGhost + 1)begin
14                r = RED;
15                g = GRN;
16                b = BLUE;
17            end
18            else begin
19                r = 0;
20                g = 0;
21                b = 0;
22            end
23        end
24
25
26
27 endmodule // ghost_display
```

Ln 27 Col 27 SystemVerilog HDL File 0% 00:00:00

Figure 14. Systemverilog for ghost_display

The screenshot shows a text editor window with the title "Text Editor - C:/Users/USER/Downloads/lab6_pacmanV1/lab6 pacman - Copy/DE1_SoC - DE1_SoC - [SystemVerilog1.sv]*". The menu bar includes File, Edit, View, Project, Processing, Tools, Window, and Help. A search bar at the top right says "Search altera.com". The code is as follows:

```
1 // holds user input until next user input
2 // also decides start direction
3 module user_direction_hold
4     input logic clk, reset,
5         upIn, downIn, leftIn, rightIn, startIn,
6     output logic upOut, downOut, leftOut, rightOut;
7
8
9     logic up, down, left, right;
10
11    always_ff @(posedge clk)
12        begin
13            if (reset)
14                up <= 0;
15                down <= 0;
16                left <= 0;
17                right <= 0;
18            else if (startIn)
19                up <= 0;
20                down <= 0;
21                left <= 0;
22                right <= 0;
23            else if (upIn)
24                up <= 1;
25                down <= 0;
26                left <= 0;
27                right <= 0;
28            else if (downIn)
29                up <= 0;
30                down <= 1;
31                left <= 0;
32                right <= 0;
33            else if (leftIn)
34                up <= 0;
35                down <= 0;
36                left <= 1;
37                right <= 0;
38            else if (rightIn)
39                up <= 0;
40                down <= 0;
41                left <= 0;
42                right <= 1;
43        end
44
```

Ln 44 Col 27 SystemVerilog HDL File 0% 00:00:00

Figure 15. Systemverilog for user_direction_hold

```

1 // Top Level module that connects the ghost_cont, ghost_datapath, ghost, and ghost_display
2 module ghost #(parameter WIDTH = 96, parameter HEIGHT = 72,
3   parameter X_START = 46, parameter Y_START = 28,
4   parameter RED = 255, parameter GRN = 0, parameter BLUE = 0) (
5   input logic [1:0] clk,
6   input logic [1:0] reset,
7   input logic [1:0] win, loose,
8   input logic [3:0] ue, de, .E, //maze enable signals
9   input logic [3:0] up, down, left, right, start, // controller signals
10  input logic [8:0] x, // from display driver
11  input logic [8:0] y,
12  input logic [8:0] xPacLoc,
13  input logic [8:0] yPacLoc,
14
15  output logic [9:0] xGhost,
16  output logic [8:0] yGhost,
17  output logic [7:0] r, g, b);
18
19 //internal wires
20 logic m_up, m_down, m_right, m_left, e_start, m_hold,
21   upout, downout, leftout, rightout;
22
23 user_direction_hold g_hold( // holds user input
24   .clk(.clk), .reset(.reset),
25   .upIn(up), .downIn(down), .leftIn(left), .rightIn(right), .startIn(start),
26   .upout, .downout, .leftOut, .rightOut);
27
28 ghost_cont c_unit ( //inputs
29   .clk(.clk),
30   .reset(.reset),
31   .win(.win),
32   .ue(.ue), .de(.E), .E, //maze enable signals
33   .upIn(up), .downIn(down), .leftIn(left), .rightIn(right), .startIn(start),
34   .outputs
35   .m_up, .m_down, .m_right, .m_left, .e_start, .m_hold); // output moves
36
37 ghost_datapath#(WIDTH, HEIGHT) d_unit (
38   .clk(.clk),
39   .reset(.reset),
40   .m_up, .m_down, .m_right, .m_left, .e_start, // valid moves from control
41   .outputs
42   .xGhost,
43   .yGhost);

```

Figure 16. Systemverilog for ghost

```

1 // This modules to determine the movement direction fo the ghost based on the location of the ghost and pacman
2 // If yPacLoc is bigger that means pacman is lower than ghost, thus ghost will try to go down, otherwise go up
3 // If xPacLoc is bigger that means pacman is on the right side, thus ghost will try go'right, otherwise go left
4 module ghostlogic(
5   input logic [10:0] xPacLoc, yPacLoc,
6   input logic [10:0] xGhost, yGhost,
7   input logic glueE, g1E, g1dE, gldE,
8   input clk,
9   output logic gu, gd, gl, gr
10 );
11 logic signed [10:0] dx;
12 logic signed [10:0] dy;
13
14 always_ff @(posedge clk) begin
15   dx = xPacLoc - xGhost;
16   dy = yPacLoc - yGhost;
17   // Default directions to 0
18   gu = 0;
19   gd = 0;
20   gl = 0;
21   gr = 0;
22
23   // Determine the direction based on dx and dy, and the availability of movement
24   if (dx > 0 && g1E) begin
25     gr = 1;
26   end else if (dx < 0 && g1dE) begin
27     gl = 1;
28   end else if (dy > 0 && gldE) begin
29     gd = 1;
30   end else if (dy < 0 && glueE) begin
31     gu = 1;
32   end
33 end
34 endmodule

```

Figure 17. Systemverilog for ghostlogic

The implementation of the ghosts is similar to that of Pacman. However, instead of receiving direction signals from user input, the ghosts use a separate algorithm to determine their movement direction.

The algorithm determines the movement direction of the ghosts based on the relative positions of the ghost and Pacman. The logic is as follows:

Vertical Movement:

- If y_{PacLoc} (Pacman's y-coordinate) is greater than the ghost's y-coordinate, it means Pacman is below the ghost. Therefore, the ghost will try to move down.
- If y_{PacLoc} is less than or equal to the ghost's y-coordinate, Pacman is above the ghost. Thus, the ghost will try to move up.

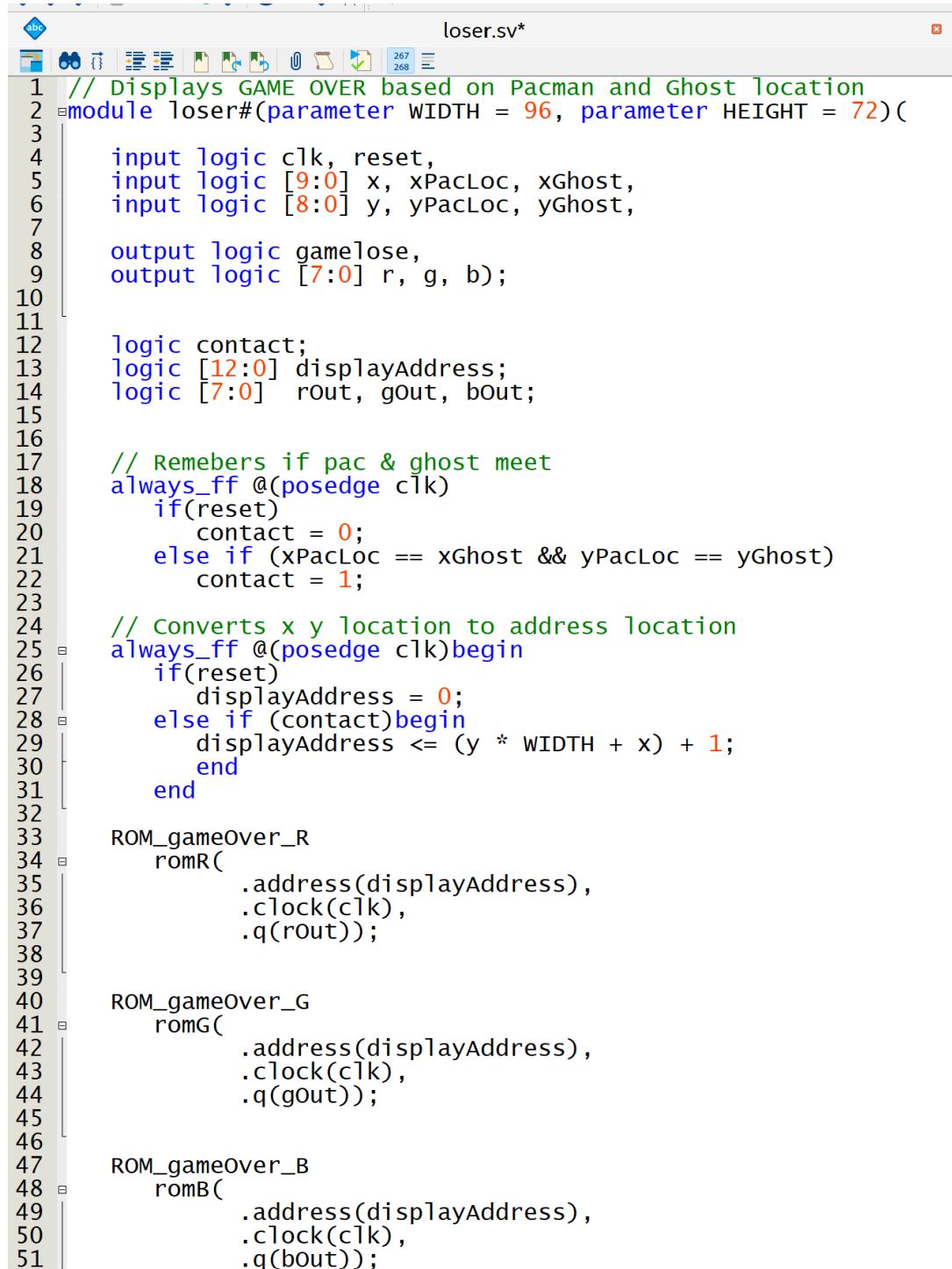
Horizontal Movement:

- If x_{PacLoc} (Pacman's x-coordinate) is greater than the ghost's x-coordinate, it means Pacman is to the right of the ghost. Therefore, the ghost will try to move right.
- If x_{PacLoc} is less than or equal to the ghost's x-coordinate, Pacman is to the left of the ghost. Thus, the ghost will try to move left.

Thus, in this way, the ghost is always trying to chase the pacman until their locations are exactly the same.

Loser

After building the main components we are now ready to make a couple final touches to the game. We want to have significant things happen when the game is won or lost. For example, the whole display screen needs to change and display a message. We know if we have time, giving pacman extra lives would not be hard to implement given that he has his own reset button. Changing the whole display to say something like game over seemed like a necessary goal for a basic implementation. We knew that utilizing a ROM filled with the data would be one of the best ways to make this happen. See figure 18 below for details.



The screenshot shows a text editor window titled "loser.sv*" containing Verilog code. The code defines a module "loser" with parameters WIDTH = 96 and HEIGHT = 72. It takes inputs for clock (clk), reset, Pacman location (x, y), and Ghost location (x, y). It outputs gameover status and RGB colors (r, g, b). The module uses an always_ff block to remember if Pacman and Ghost meet. It also converts the x-y location into a memory address and reads from three ROM blocks (R, G, B) based on the address.

```
1 // Displays GAME OVER based on Pacman and Ghost location
2 module loser#(parameter WIDTH = 96, parameter HEIGHT = 72)(
3     input logic clk, reset,
4     input logic [9:0] x, xPacLoc, xGhost,
5     input logic [8:0] y, yPacLoc, yGhost,
6
7     output logic gamelose,
8     output logic [7:0] r, g, b);
9
10
11    logic contact;
12    logic [12:0] displayAddress;
13    logic [7:0] rout, gout, bout;
14
15
16    // Remebers if pac & ghost meet
17    always_ff @(posedge clk)
18        if(reset)
19            contact = 0;
20        else if (xPacLoc == xGhost && yPacLoc == yGhost)
21            contact = 1;
22
23    // Converts x y location to address location
24    always_ff @(posedge clk)begin
25        if(reset)
26            displayAddress = 0;
27        else if (contact)begin
28            displayAddress <= (y * WIDTH + x) + 1;
29        end
30    end
31
32    ROM_gameOver_R
33    romR(
34        .address(displayAddress),
35        .clock(clk),
36        .q(rout));
37
38
39    ROM_gameOver_G
40    romG(
41        .address(displayAddress),
42        .clock(clk),
43        .q(gout));
44
45
46    ROM_gameOver_B
47    romB(
48        .address(displayAddress),
49        .clock(clk),
50        .q(bout));
```

```

51
52
53
54    // output logic
55    assign gamelose = contact;
56    assign r = rOut * 250;
57    assign g = gOut * 250;
58    assign b = bout;
59
60
61 endmodule // loser

```

Figure 18. Systemverilog for loser

As we can see there are only a couple key points we needed to figure out.

- Trigger loose condition by contact of the ghost and pacman's location
- Convert the x and y coordinates to a ROM address

One lesson learned is that for our implementation we only needed one ROM due to the type of data available to generate for the MIF file. This is why the output values are scaled so great.

More discussion on this in the results section.

Winner

The winner module and situation will use the framework of the loose situation but will be slimmed down due to not needing as many input signals. The win situation only is active when all the food is eaten so this single logic signal will come from the food module.

```

1 // Winning display
2 // Starts reading ROM after winning condition met
3 module winner#(parameter WIDTH = 96, parameter HEIGHT = 72)(
4
5   input logic clk, reset, gamewin,
6   input logic [9:0] x,
7   input logic [8:0] y,
8
9   output logic [7:0] r, g, b);
10
11   logic contact;
12   logic [12:0] displayAddress;
13   logic [7:0] qout;
14
15   // Converts x y location to address location
16   always_ff @(posedge clk)begin
17     if(reset)
18       displayAddress = 0;
19     else if (gamewin)begin
20       displayAddress <= (y * WIDTH + x) + 1;
21     end
22   end
23
24
25   ROM_win
26   romW(
27     .address(displayAddress),
28     .clock(clk),
29     .q(qout));
30
31   // output logic
32   assign gamelose = contact;
33   assign r = qout * 250;
34   assign g = qout * 250;
35   assign b = qout;
36
37
38 endmodule // winner
39

```

Figure 19. Systemverilog for winner

As seen in figure 19 the winner module is just a slimmed down version of the loser module in figure 18. The ROM is at the heart of both the modules. The winner is looking for one signal to take over and celebrate the win.

Top Level module (DE1 - SoC)

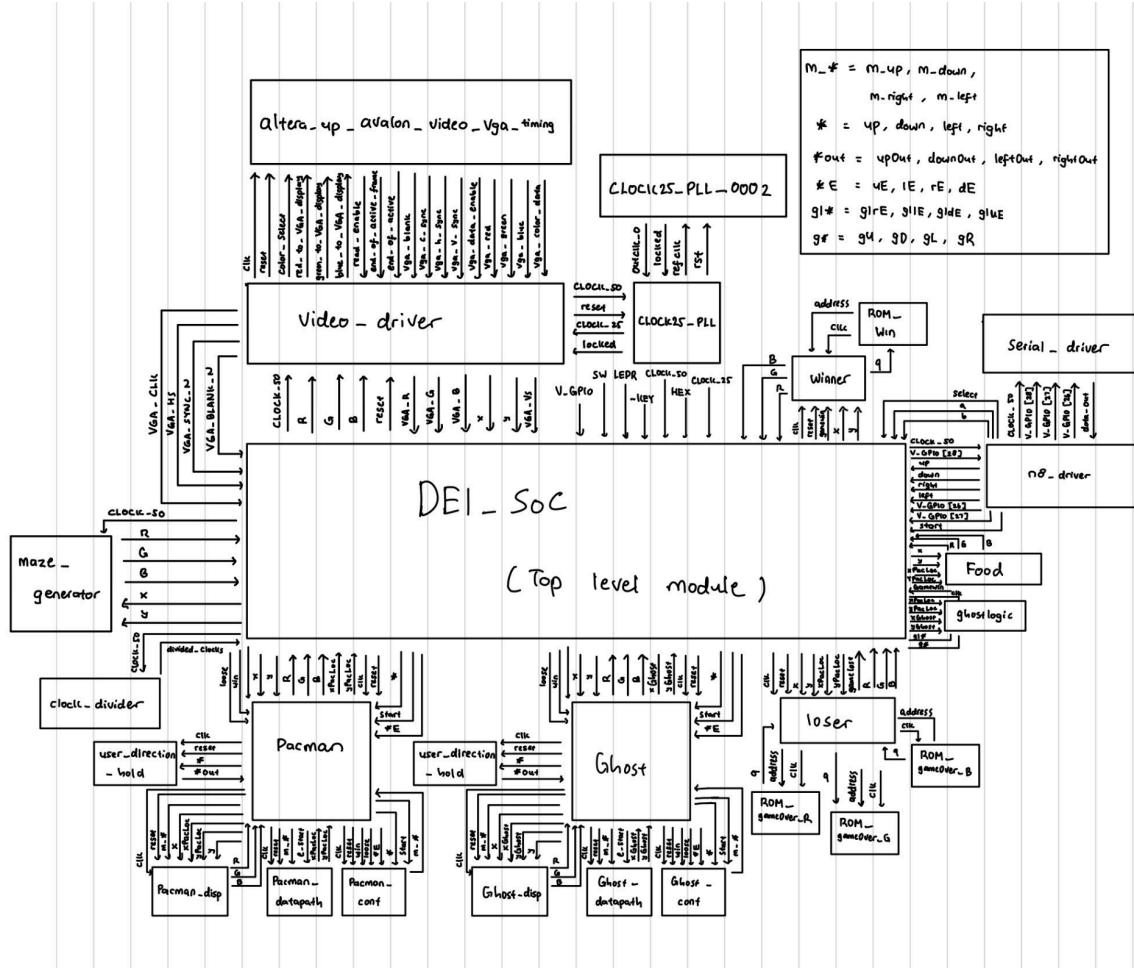


Figure 20. Block diagram of the top level

Once all the necessary modules were created, the next step was to integrate them into a top-level module, which connects the board and all the individual modules to enable the game to be played. Here's a more detailed breakdown of the integration process:

- Connecting the N8 Controller:
 - The N8 controller serves as the primary movement control signal for Pacman.
 - The up signal from the N8 controller makes Pacman move up.
 - The left signal makes Pacman move left.
 - The right signal makes Pacman move right.
 - The down signal makes Pacman move down.
 - Additionally, the start signal, also triggered from the N8 controller, initiates the game.
- Reset Signal:

- The reset signal is controlled by ~KEY0. Pressing ~KEY0 will reset the game, returning Pacman and all game elements to their initial states.
- Integration of Modules: The top-level module integrates the maze, Pacman, ghosts, food, and VGA display modules.
- Maze Module: Defines the layout of the game area, including walls and pathways.
- Pacman Module: Controls Pacman's position, movement, and interactions with other game elements.
- Ghost Modules: Control the movement and behavior of ghosts, including their chasing logic and interaction with Pacman.
- Food Module: Manages the placement and consumption of food items by Pacman.
- VGA Display Module: Handles the graphical output to the display, rendering Pacman, ghosts, food, and the maze.
- Collision Detection and Game Logic:
 - The top-level module coordinates collision detection, ensuring Pacman interacts correctly with walls, food, and ghosts.
 - When Pacman collides with food, the food is consumed, and the score is updated.
 - If Pacman collides with a ghost, the game logic determines whether Pacman loses a life or if special conditions (e.g., power-ups) affect the interaction.
- Gameplay Flow:
 - At the start of the game, the start signal initializes all game elements.
 - Players use the N8 controller to navigate Pacman through the maze, avoiding ghosts and collecting food.
 - The reset signal can be used to restart the game at any time.

By integrating these components into a cohesive top-level module, we create a fully functional Pacman game that can be played on an FPGA with VGA output and N8 controller input. The top-level module ensures seamless interaction between all parts of the game, providing an engaging and responsive gaming experience.

Results

Our Pacman game turned out well with all the main features covered. The movement of Pacman simulated the original game play. This allows the user to push the direction once anytime before the turn and Pacman will make that turn. This all happened correctly within the maze boundaries.

The map is not identical to the original game but still has a similar feel. The food is another key part of the game that we were happy to get working properly. This took more than one try and ended up being the most cleverly written part of the code. Last but not least we also implemented the ghost. The logic for the ghost is a touch different than the original game. Basically our ghost chases Pacman much more from start to finish and never lets up. The framework of the code allows for multiple ghosts and multiple types of ghost logic. This game took lots of testing mostly through labsland but some through the testbench. In this section we will take a look at the game development and testing in the same order as it came together.

Maze

The maze was the beginning of using the new display driver. This set the tone for the rest of the game. One issue that we ran into is scaling. The maze code needed to be rewritten after we realized that life will be easier later on if we had a known set of pixels. We decided on a 96x72 frame to allow for our target size of 3x3 for Pacman. The maze turned out well and started our display logic. See figure 21 below for details

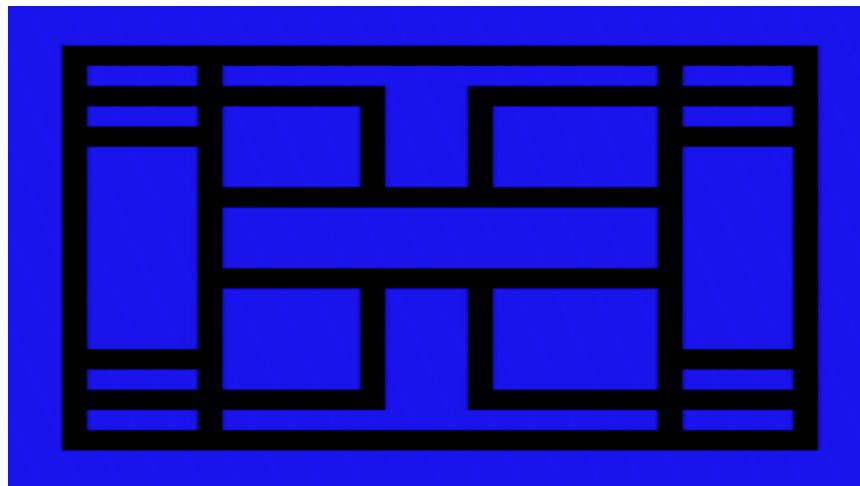


Figure 21. maze

Food

The food is a key part of the game and after understanding the display driver the food was implemented. One problem came about though because the food is not static like the maze it needs to change as Pacman eats it. This meant that the food code also needed to be rewritten. At

this time the focus shifted to getting Pacman moving to see if there are any other issues that we need to keep on radar for combining the modules. Below is the static food implementation that ultimately gets changed but keeps the same look.

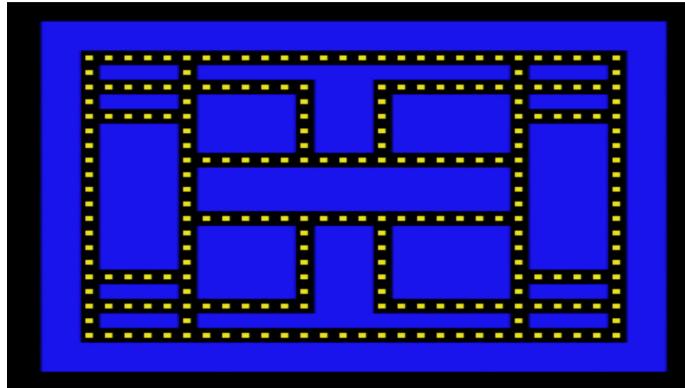


Figure 22. maze and food

Pacman and maze logic

Pacman is obviously the heart of the game so we wanted lots of control and interaction capabilities. This resulted in an AMSD design with lots of signals and outputs. To begin with we made it so that Pacman moves by pushing a button on the directional pad on the N8. This ended up working after a fair amount of struggle. Pacman was able to move anywhere on the screen, including through to the other side. We knew that in the game play of Pacman he needed to move on his own and just turn but we also needed to make a map logic that signaled to Pacman that he can not go through walls. So the map logic was made based on the map. With hindsight we wonder if there was a way to use the map as the map logic. It seems possible but tricky because the map is 3 pixels wide. The maze logic made it easy for Pacman to stay on track and know when there was a corner or turn. The maze logic worked well with the manual diving of Pacman. So then we moved to making Pacman move smoothly through the maze. This proved to be more difficult because of checking the next state. When Pacman was moved manually there was a hold state after each move, this allowed for enough time for the enable signals from the map logic to stop the movement. So when implementing the constant movement Pacman would step one pixel off and then be stuck because he was out of the maze logic. We ended up adding four more states to fix this timing issue. Each direction basically needed a check state which would pass through the movement if available. We needed to keep the hold state as well for when Pacman comes to a

stop after hitting a wall. The figure below shows a transition that is working properly but does not work for hitting walls.



Figure 23. Pacman debugging waveform

We can see the x location change directions after the right signal. This works unless he hits a wall. Figure below is the corrected version. Here we can see the check states and the hold state when Pacman hits the wall.

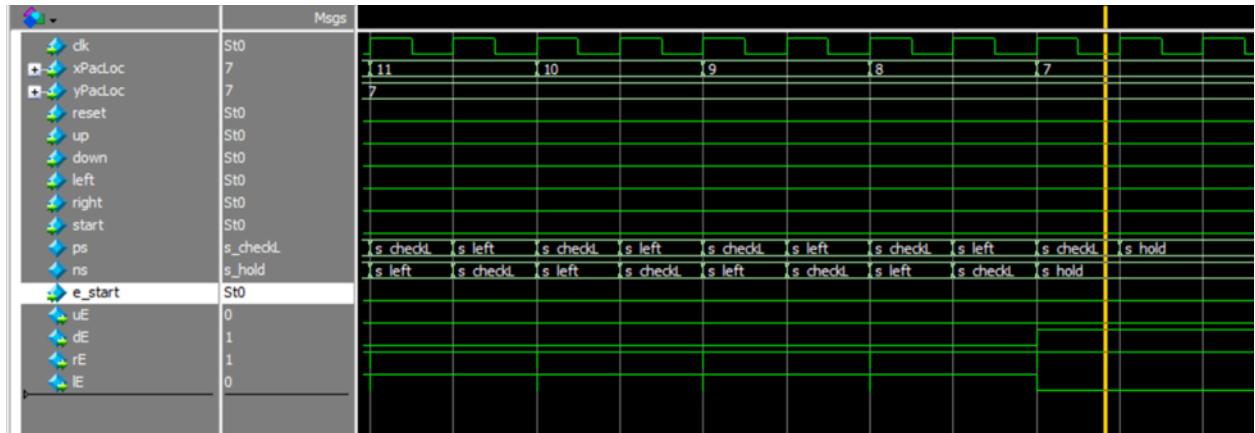


Figure 24. Pacman hitting wall waveform

Next issue for Pacman is remembering the last input signal. This was a relatively easy problem to solve with an additional module that connected to the inputs. It would only allow one signal to be transmitted at a time and then the signal would stay on until the other took over. See figure below for details. The waveform is not the best example but it does show that it is working. Basically the left signal is on and stays on. LeftIn is now the input and we can see it is only on

for a single clock where left is continually on. The reason this is not the best waveform is because the left signal is initially triggered at the start of the game. That's why it's on before leftIn is triggered.

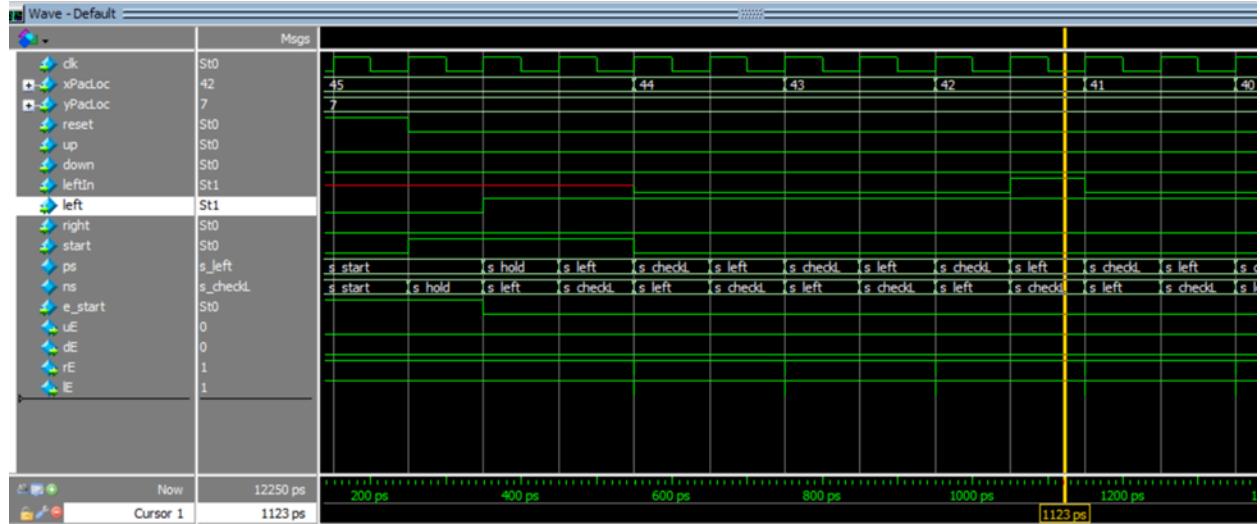


Figure 25. Pacman remember signals waveform

Now that Pacman can move and stop properly, final testing of his movement is done on the VGA display to ensure that nothing was missed. This is where we figured out a proper clock divider speed to be implemented. Pacman is the white cube in test mode. See figure below for details.

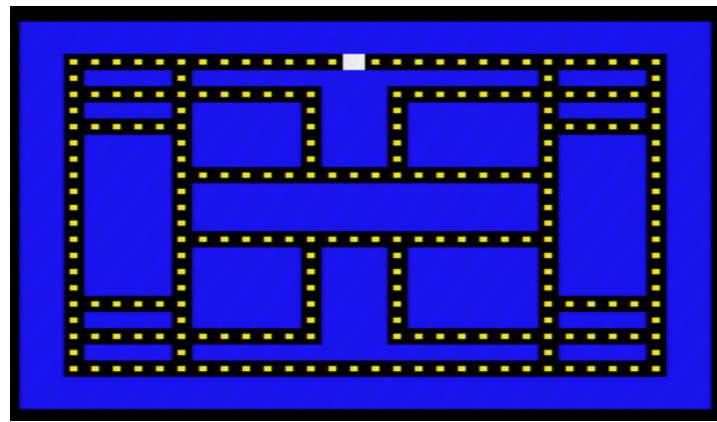


Figure 26. Pacman with movement in maze

Food

The new food module allowed Pacman to finally eat. Each piece of food now had a location in an array memory that would be updated when Pacmans location equaled the food location. This is a lot of locations so we implemented multiple loops to populate the different line locations.

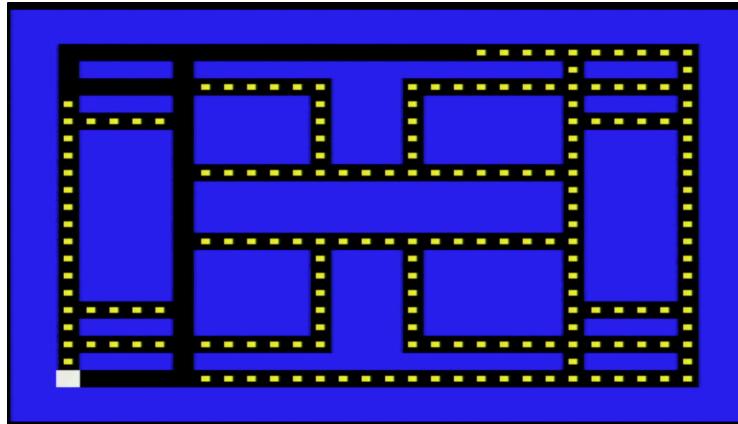


Figure 27. Game play with Pacman eating

Ghost

Pacman would not be fun without the ghosts. For our game we knew if we had Pacman working well that the ghost would be able to use most of the same logic and display modules. This turned out to be true. Only minor name changes were needed. The biggest issue was figuring out what clock should run which component. The one that caused the most trouble was the N8 controller. Even though the ghost does not use the controller, Pacman does and it needs to be at the same clock speed as he is running. We started the ghost the same way where we got it working with our inputs and then built the ghost logic to control after that.

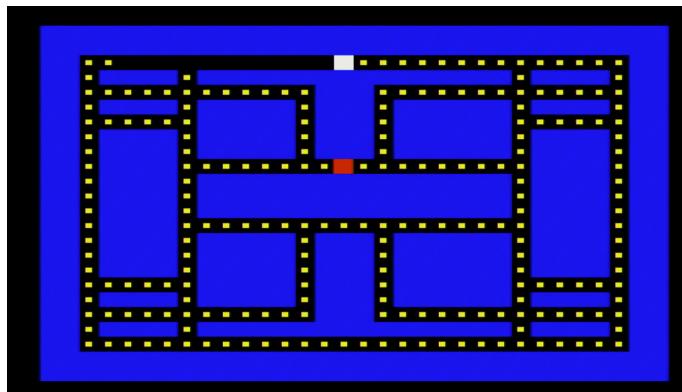


Figure 28. Pacman and ghost

Ghost Logic

The ghost logic determines the feel and stress of the game. Our ghost is able to see Pacmans location at all times. Then based on that location compared to the ghosts location the ghost only has one goal. Which is to be where Pacman is. This makes the game challenging even with only one ghost. If we had more time this is where we can imagine many kinds of ghost logic that could give the game different strategies and such. See figure 29 below for details.

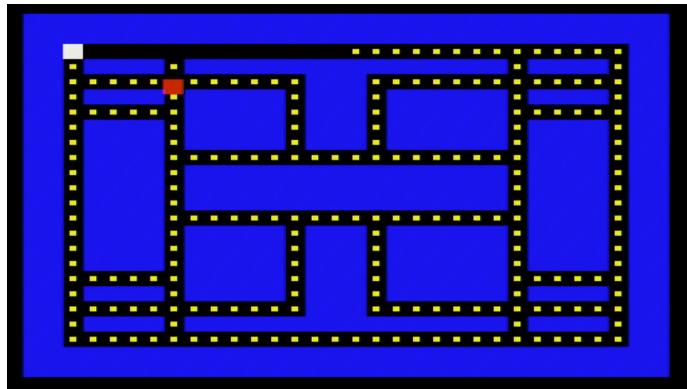


Figure 29. Ghost chasing Pacman

Loose

What happens when the Ghost catches Pacman? See figure 30.



Figure 30. Game Over ROM display

Once the Ghost catches Pacman the game is over and a ROM is then utilized to display this static image. The hardest part of using the ROM is just setting up a memory initialization file. This was possibly more difficult in our situation because of our unusual width and depth. We spent some time trying to figure out how to do an RGB image with multiple colors. This game over image is utilizing three ROMs but they all ended up having the same .mif file for simplicity.

Win

As Pacman was a learning experience for the Ghost the loose was for the win. The win logic comes from a slimmed down version of the loose module which then implements another ROM to display, see the image below in figure 31.



Figure 31. Winning ROM display