Lab 4 Report

Introduction

We designed a game called “Piggy’s Poisonous Farts”, implemented on the Nexys 3 Spartan 6 FPGA board and displayed on the monitor via a VGA cable. The goal is to move Piggy (pink square) with up/down/left/right buttons so that he touches and consumes a randomly placed snack (blue square) each round while never revisiting the orange trail of poisonous farts that he leaves behind him. Every round, Piggy has 15 seconds to eat the snack. When he eats a snack, a new round with a new snack begins, Piggy grows larger, and his trail gets longer. Every 5 rounds, a vegetable (green square) appears in addition to the snack, and eating it shrinks Piggy’s size by a fixed amount. If the vegetable is not eaten before the snack, it disappears. If the timer runs out or if Piggy touches his poison trail, the game ends, and a blue screen with a white frowny face is displayed after a few seconds. During the game, 8 LEDs display the score (number of snacks eaten), and the seven-segment display shows the 15-second timer. When the game ends, the seven-segment display flashes the score. The game can be reset at any time by flipping the reset switch and pressing start (middle button) to begin the game.

Design Description

Our game is implemented with 10 modules. The top level module (top, **Figure 1**) connects 9 inner modules together (clock_divider, counter, ss_display, Debouncer, snack_locator, vegetable_locator, pig_locator, trail_locator, vga640x480). The top module receives input from the FPGA’s 100 MHz clk, rst switch, and up/down/left/right/start buttons. It outputs the following: {[7:0] seg, [3:0] an} for the seven-segment display, {[7:0] led} for the 8 LEDs, and {hsync, vsync, [2:0] red, [2:0] green, [1:0] blue} for the VGA display.

```
3  module top(rst, clk, start, btnU, btnR, btnD, btnL, seg, an, led, hsync, vsync, red, green, blue);
4      `include "constants.v"
5      input rst, start, clk, btnU, btnR, btnD, btnL;
6      output [7:0] seg;
7      output [3:0] an;
8      output [7:0] led;
9      output wire hsync;      //horizontal sync out
10     output wire vsync;      //vertical sync out
11     output wire [2:0] red;  //red vga output
12     output wire [2:0] green; //green vga output
13     output wire [1:0] blue; //blue vga output
```

Figure 1: top module interface: input and output ports

The clock_divider uses the FPGA’s 100 MHz clk to produce clocks of various frequencies that are used by the counter, ss_display, and vga640x480 modules. The counter uses the 1 Hz clk outputted by clock_divider to count down from 15 seconds every round, and if the timer hits 0, it outputs a game over signal to the top module. ss_display receives various inputs (e.g. clk_fast and clk_blink from clock_divider, timer values from counter, and score from pig_locator) and outputs [7:0] ‘seg’ and [3:0] ‘an’ to either display the timer on the seven-segment display during the game or to flash the score after the game ends. Debouncer filters extra noise from the up/down/left/right buttons (intentionally excluding the start button) so that button presses are only counted once. Refer to our Lab 3 report for specific details on clock_divider, counter, ss_display, and Debouncer. We now focus on explaining modules that are new to this lab.

First, snack_locator (**Figure 2**) receives inputs from the top module (e.g. start, new_round) and outputs random snack coordinates each round to pig_locator and vga640x480. Two counters, countX and countY, are initialized to minX and minY (the upper left coordinates of the screen)

when the reset switch is flipped (**Figure 3**, lines 18-21). The counters increment by 1 every clock cycle and wrap around upon reaching the end of the screen (lines 40-47). When the player presses the start button or if a new round begins, the snack's position is set equal to counterX and counterY (lines 36-37). This produces a random location because it takes 2 steps to reset our game: flipping the reset switch and pressing the start button. The variable difference in time between these 2 steps will allow the counters to reach an unpredictable value. New rounds obviously occur at unpredictable times, too. In addition, Piggy's current position is checked to ensure that if the snack is about to spawn on top of him, it is instead placed nearby (lines 23-35). The vegetable_locator's interface is identical to the snack_locator, and its implementation is nearly identical, except that vegetable_locator initializes its counters to different values than snack_locator so that the vegetable does not spawn in the exact same location as the snack.

```
4  module snack_locator(clk, rst, new_round, pigX, pigY, pig_growth, posX, posY, start, posX_end, posY_end);
5      input start, clk, rst, new_round;
6      input wire [10:0] pigX, pigY, pig_growth;
7      output reg [10:0] posX, posY;
8      output wire [10:0] posX_end, posY_end;
```

Figure 2: snack_loctor interface (identical to vegetable_locator interface): input/output ports

```
17      always @(posedge clk) begin
18          if (rst) begin
19              countX = minX;
20              countY = minY;
21          end
22          else if (start || new_round) begin
23              // If snack is about to spawn on pig, relocate
24              if (countX >= posX - snack_size - pig_growth && countX <= posX + pig_size &&
25                  countY >= posY - snack_size - pig_growth && countY <= posY + pig_size)
26              begin
27                  if (countY - pig_size > minY)
28                      countY = countY - pig_size;  // up
29                  else if (countY + pig_size < maxY - snack_size)
30                      countY = countY + pig_size;  // down
31                  else if (countX - pig_size > minX)
32                      countX = countX - pig_size;  // left
33                  else if (countX + pig_size < maxX - snack_size)
34                      countX = countX + pig_size;  // right
35              end
36              posX = countX;  // set random snack pos
37              posY = countY;
38          end
39          else begin
40              if (countX == maxX - snack_size)
41                  countX = minX;
42              else
43                  countX = countX+1;
44              if (countY == maxY - snack_size)
45                  countY = minY;
46              else
47                  countY = countY+1;
48          end
49      end
```

Figure 3: snack_locator logic: setting a random location for the snack every round

The pig_locator (**Figure 4**) implements most of the game logic, like Piggy's current coordinates, snack/vegetable consumption, growth/shrinkage, and score. It receives various inputs (e.g. up/down/left/right signals from Debouncer, snack position from snack_locator, vegetable position from vegetable_locator) and outputs Piggy's updated position and score to ss_display, trail_locator, and vga640x480. Since this module has very complex code, we will only highlight the most important parts. **Figure 5** shows the game logic for when Piggy moves up. The coordinates (x,y)=(0,0) are the top left corner of the screen, so in lines 146-150 we subtract the move distance from posY while keeping posX constant to move Piggy up in a normal situation (extra_move and extra_move_perp in these lines are explained soon). Lines 120-130 implement snack consumption logic for when Piggy moves up into a snack. First, we check whether Piggy's new coordinates overlap with the snack. If so, a new round begins, the score increments, and Piggy grows. In line 128, we subtract 2 values from posY in addition to the move distance:

'growth' and 'extra_move'. We subtract 'growth' so that Piggy grows upward, in the direction that he is moving (if we did not do this, Piggy would grow downward and automatically overlap himself with the trail that he puts behind him, thus killing himself upon eating a snack). We also subtract 'extra_move', which is nonzero *only* when Piggy's previous direction was left or right, so that Piggy will move up even more when he is turning. Basically, this "extra move when turning" feature prevents misalignment in his trail when he turns. Similarly, line 127 adjusts Piggy's x-coordinate upon turning and eating a snack so that his trail is always centered right behind him. Vegetable consumption logic in lines 131-145 is very similar, but now Piggy shrinks instead of growing when he overlaps himself with a vegetable upon moving up. The code in **Figure 5** is repetitive; we use similar logic for the other 3 directions.

```
3  module pig_locator(clk, game_over, game_state, is_fifth_round, vegetable_consumed, rst, up, down, right, left,
4                     snackX, snackY, snackY_end, snackX_end, vegetableX, vegetableY, vegetableX_end, vegetableY_end,
5                     posX, posY, posX_end, posY_end, pig_growth, score, new_round, ateVegetable);
6
7      `include "constants.v"
8      input clk, game_state, is_fifth_round, rst, up, down, right, left, game_over;
9      input [10:0] snackX, snackX_end, snackY, snackY_end;
10     input [10:0] vegetableX, vegetableX_end, vegetableY, vegetableY_end;
11     output reg signed[10:0] posY, posX, pig_growth;
12     output wire signed[10:0] posY_end, posX_end;
13     output reg [7:0] score;
14     output reg new_round;
15     output reg vegetable_consumed;
```

Figure 4: pig_locator interface: input/output ports

```
112        // Piggy moving logic
113        always @(posedge clk) begin
114            if (rst) begin
115                posY <= startY; posX <= startX; pig_growth <= 0; new_round <= 0; score <= 0; vegetable_consumed <= 0;
116            end
117            else if (!game_over && game_state) begin
118                // Piggy moving up
119                if (up2 && (posY_minus_move >= minY)) begin
120                    // Touching snack
121                    if ((posY_minus_move) > pig_snack_overlapY_lower && (posY_minus_move) < pig_snack_overlapY_upper &&
122                        posX > pig_snack_overlapX_lower && posX < pig_snack_overlapX_upper)
123                    begin
124                        new_round <= 1;
125                        score <= score + 1;
126                        pig_growth <= pig_growth + growth;
127                        posX <= posX - (growth>>1) + extra_move_perp;  // adjust for growth and turns
128                        posY <= posY - growth - move - extra_move;     // grow UP, in direction of Piggy's move
129                        vegetable_consumed <= 0;
130                    end
131                    // Touching vegetable
132                    else if ((posY_minus_move) > pig_veg_overlapY_lower && (posY_minus_move) < pig_veg_overlapY_upper &&
133                        posX > pig_veg_overlapX_lower && posX < pig_veg_overlapX_upper)
134                    begin
135                        if (is_fifth_round && !vegetable_consumed) begin
136                            pig_growth <= pig_growth - shrink;
137                            vegetable_consumed <= 1;
138                            posX <= posX + (shrink >> 1) + extra_move_perp;  // adjust for shrinkage and turns
139                            posY <= posY - move + shrink - extra_move;       // shrink DOWN, opposite of move direction
140                        end
141                        else begin
142                            posY <= posY - move - extra_move;
143                            posX <= posX + extra_move_perp;
144                        end
145                    end
146                    // Touching nothing (Normal case: just moving up)
147                    else begin
148                        posY <= posY - move - extra_move;  // extra_move is nonzero when Piggy is turning
149                        posX <= posX + extra_move_perp;    // extra_move_perp is nonzero when Piggy is turning
150                    end
151                end
```

Figure 5: pig_locator logic: moving up and consuming snack/vegetable

Next, trail_locator (**Figure 6**) receives Piggy's current coordinates (pigX, pigY) from pig_locator and sampled button presses from Debouncer to output poison trail coordinates to the top module and vga640x480. The outputs trailX and trailY are arrays containing each point of the trail in 4-bit increments; the first trail coordinate is contained in the first 4 bits, the second coordinate is in the second 4 bits, and so on. Ordinarily, 10 bits are needed to store the value of each coordinate (since the screen is 640 pixels long). However, in order to save hardware space,

we divided each coordinate value by the trail width (50) so that we can use 4 bits to store values between 0 and 15, and later multiply these values by the trail width to get the original values. In **Figure 7**, registers tempX and tempY track where the next trail point will be located. For example, if Piggy moves to the right, the next trail point will be one move (50 pixels) to the right of the previous trail coordinate, so we increment tempX by one (lines 72-73). The current values of tempX and tempY are stored in the trail arrays (trailX, trailY) when Piggy moves, so Piggy's previous location becomes marked by a trail. The first time Piggy moves, we set the first four bits of our trail arrays to Piggy's predetermined starting position (lines 44-45).

```
4  module trail_locator(clk, rst, game_state, game_over, round, up, down, left, right, pigX, pigY, trailX, trailY, pig_growth);
5      input clk, rst, game_state, game_over, up, down, left, right;
6      input [7:0] round;  // same as score
7      input [9:0] pigX, pigY, pig_growth;
8      output reg [array_size-1:0] trailX, trailY;
```

Figure 6: trail_locator interface: input/output ports

```
42              if (first_move == 0) begin
43                  first_move <= 1;
44                  trailX[ta +: bit_width] <= startX_trail;
45                  trailY[ta +: bit_width] <= startY_trail;
46                  if (up2) begin
47                      tempX <= startX_trail;
48                      tempY <= startY_trail - 1;
49                  end
50                  else if (down2) begin
51                      tempX <= startX_trail;
52                      tempY <= startY_trail + 1;
53                  end
54                  if (right2) begin
55                      tempX <= startX_trail + 1;
56                      tempY <= startY_trail;
57                  end
58                  if (left2) begin
59                      tempX <= startX_trail - 1;
60                      tempY <= startY_trail;
61                  end
62              end
63              else if (up2) begin
64                  tempX <= tempX;
65                  tempY <= tempY - 1;
66              end
67              else if (down2) begin
68                  tempX <= tempX;
69                  tempY <= tempY + 1;
70              end
71              else if (right2) begin
72                  tempX <= tempX + 1;
73                  tempY <= tempY;
74              end
75              else if (left2) begin
76                  tempX <= tempX - 1;
77                  tempY <= tempY;
78              end
```

Figure 7: trail_locator logic: Determining where to place the next point in the trail

The num_points register shown in **Figure 8** is used to determine which bits in the trail arrays are replaced by the new tempX and tempY coordinates. It is incremented when Piggy moves so that each trail coordinate will occupy a new set of bits in the array. When num_points is reset to zero, the first trail coordinate stored in the trail array will be overwritten, causing the oldest trail point on the screen to disappear when Piggy moves. As num_points increments through the array, the oldest trail points in the arrays are continuously overwritten. In order for the trail to increase in length each round, num_points is limited to a maximum value of (2*round number), so that num_points will reset less frequently and more trail_points may appear simultaneously as the rounds increase (lines 85-87). The maximum number of trail points is capped at trail_points, a constant (31) based on hardware limitations (lines 88-89), but this is sufficient for our game.

```
80      if (first_move != 0) begin
81          trailX[ta +: bit_width] <= tempX;
82          trailY[ta +: bit_width] <= tempY;
83      end
84
85      if (num_points == 2*round) begin
86          num_points <= 0;
87      end
88      else if (num_points == trail_points-1) begin
89          num_points <= 0;
90      end
91      else begin
92          num_points <= num_points+1;
93      end
```

Figure 8: trail_locator logic: storing trail points in trail arrays, increasing trail length each round

The vga640x480 module (**Figure 9**) receives many inputs from previous modules, such as the 25 MHz clock from clock_divider, Piggy's coordinates from pig_locator, the snack's coordinates from snack_locator, and the vegetable's coordinates from vegetable_locator. It outputs the horizontal/vertical sync and RGB values that drive the real-time state of the game on the monitor. The display has 800 horizontal and 521 vertical pixels per line, but our game screen is only 640 pixels long and 480 pixels tall because the VGA has a brief period of time where no information can be displayed (known as porch intervals). Therefore, to select specific coordinates in our game, we factor in horizontal and vertical back porches. For example, to display an object in the top left corner of our game screen at coordinates (minX, minY), we must use VGA coordinates (hbp+minX, vbp+minY), where hbp and vbp are the number of pixels in the horizontal and vertical back porches respectively. Similarly, there are horizontal and vertical *front* porches that indicate the last coordinates that can be displayed, and all coordinates to be displayed must be less than these. To actually designate a color to each pixel, we have a horizontal counter ('hc') and vertical counter ('vc'). Both 'hc' and 'vc' start at 0, and 'hc' is incremented each clock cycle using the 25 MHz clock from clock_divider. Every time 'hc' hits the end of a row of pixels, 'hc' is reset and 'vc' is incremented by 1. Therefore, we iterate through every pixel row by row, starting from the top and resetting upon hitting the bottom.

```
3   module vga640x480(
4       input wire dclk, input wire clr, input game_state, input game_over, input display_blue, input is_fifth_round,
5       input vegetable_consumed, input wire [10:0] pigX, input wire [10:0] pigY, input wire [10:0] pigX_end, input wire [10:0] pigY_end,
6       input wire [10:0] snackX, input wire [10:0] snackY, input wire [10:0] snackY_end, input wire [10:0] snackX_end,
7       input wire [10:0] vegetableX, input wire [10:0] vegetableY, input wire [10:0] vegetableY_end, input wire [10:0] vegetableX_end,
8       input wire [array_size-1:0] trailX, input wire [array_size-1:0] trailY,
9
10      output wire hsync, output wire vsync, output reg [2:0] red, output reg [2:0] green, output reg [1:0] blue
11  );
12
13  `include "constants.v"
14  parameter hpixels = 800;// horizontal pixels per line
15  parameter vlines = 521; // vertical lines per frame
16  parameter hpulse = 96;  // hsync pulse length
17  parameter vpulse = 2;   // vsync pulse length
18  parameter hbp = 144;    // end of horizontal back porch         (Can't display anything before this point!)
19  parameter hfp = 784;    // beginning of horizontal front porch  (Can't display anything after this point!)
20  parameter vbp = 31;     // end of vertical back porch           (Can't display anything before this point!)
21  parameter vfp = 511;    // beginning of vertical front porch    (Can't display anything after this point!)
22  // active horizontal video is therefore: 784 - 144 = 640
23  // active vertical video is therefore: 511 - 31 = 480
```

Figure 9: vga640x480 module interface: input/output ports and VGA parameters

While iterating through pixels row by row, we determine which game object exists at the coordinates selected by the counters 'hc' and 'vc'. **Figure 10** shows how we set boolean values displayPig, displaySnack, displayVegetable, and displayFart. For example, in line 103 we set displayPig to 1 if 'hc' and 'vc' are currently between Piggy's starting and ending coordinates. A for loop is used to determine if hc and vc are within the bounds of any of the trail coordinates (the top module uses a similar method to determine if Piggy overlaps his trail). These boolean

values are then used in **Figure 11** lines 139-163 to assign the correct values to the RGB registers, which are outputted to the monitor via the VGA cable to display all objects.

```
98   reg [trail_points-1:0] trailCoordinates;
99   integer i;
100
101  always @(dclk) begin
102      // Check: Should we display Piggy at these pixel coordinates? If so, displayPig == 1. Same for snack/vegetable.
103      displayPig <= (hc >= vga_pigX && hc <= vga_pigX_end && vc >= vga_pigY && vc <= vga_pigY_end);
104      displaySnack <= (hc >= vga_snackX && hc <= vga_snackX_end && vc >= vga_snackY && vc <= vga_snackY_end);
105      if (is_fifth_round && !vegetable_consumed)
106          displayVegetable <= (hc >= vga_vegetableX && hc <= vga_vegetableX_end && vc >= vga_vegetableY && vc <= vga_vegetableY_end);
107      else
108          displayVegetable <= 0;
109      // Check all trail coordinates to see if we need to display a trail point at these pixel coordinates.
110      for(i = 0; i < trail_points; i=i+1) begin
111          trailCoordinates[i] <= (trailX[bit_width*i +: bit_width] != 31 && trailY[bit_width*i +: bit_width] != 31 &&
112                      hc >= (hbp + trail_width*trailX[bit_width*i +: bit_width]) &&
113                      hc <= (hbp + trail_width*trailX[bit_width*i +: bit_width] + trail_width) &&
114                      vc >= (vbp + trail_width*trailY[bit_width*i +: bit_width]) &&
115                      vc <= (vbp + trail_width*trailY[bit_width*i +: bit_width] + trail_width));
116      end
117  end
118  assign displayFart = | trailCoordinates;
```

Figure 10: vga640x480 logic: Determining which game object exists at the selected coordinates

Finally, **Figure 12** shows how our top module tracks the game state (e.g. starting a new game, game over mode). In lines 76-82, we set game_state to 1 when the reset switch is flipped and the start button is pressed, thus starting a new game. In line 85, we set game_over to 1 when either the 15-second timer hits 0 or piggy touches the trail, thus ending the game. In line 87, we track every fifth round so that we know when to display the vegetable. All these state wires are fed into the previously described modules so that the game state is synchronized across all modules.

```
134  always @(*) begin
135      // Verify that we're between vertical back and front porches
136      if (vc >= vbp && vc < vfp) begin
137          // Verify that we're between horizontal back and front porches
138          if (hc >= hbp && hc < (hbp+640)) begin
139              if (displayPig) begin  // PINK square for Piggy
140                  red = 3'b111;
141                  green = 3'b000;
142                  blue = 2'b11;
143              end
144              else if (game_state && displaySnack) begin  // BLUE square for Snack
145                  red = 3'b000;
146                  green = 3'b000;
147                  blue = 2'b11;
148              end
149              else if (game_state && displayVegetable) begin  // GREEN square for Vegetable
150                  red = 3'b000;
151                  green = 3'b111;
152                  blue = 2'b00;
153              end
154              else if (game_state && displayFart) begin // ORANGE squares for Trail
155                  red = 3'b111;
156                  green = 3'b100;
157                  blue = 2'b00;
158              end
159              else begin  // WHITE if no game object is at the selected coordinates
160                  red = 3'b111;
161                  green = 3'b111;
162                  blue = 2'b11;
163              end
```

Figure 11: vga640x480 logic: Choosing the right color to display at the selected coordinates

```
76       always @(posedge clk) begin
77           // Start button pressed after reset switch flipped
78           if (NEW_START && !game_state) begin
79               game_state <= 1;  // * Game is now active! *
80               START <= 1;       // Random time difference b/t rst + start --> random 1st round snack location
81           end
82       end
83
84       // Game over when 15-second timer runs out (game_over1) or when Piggy touches trail (game_over2)
85       assign game_over = game_over1 || game_over2;
86       // Vegetable displays every fifth round
87       assign is_fifth_round = score != 0 && ((score % 5) == 0);
```

Figure 12: top module game state logic: Starting/ending the game, tracking every fifth round

Figure 13 is a state diagram of game_state and game_over, described in the previous paragraph. **Figure 14** is a high-level schematic of the entire circuit, including the top module and all 9 inner modules that we have described.

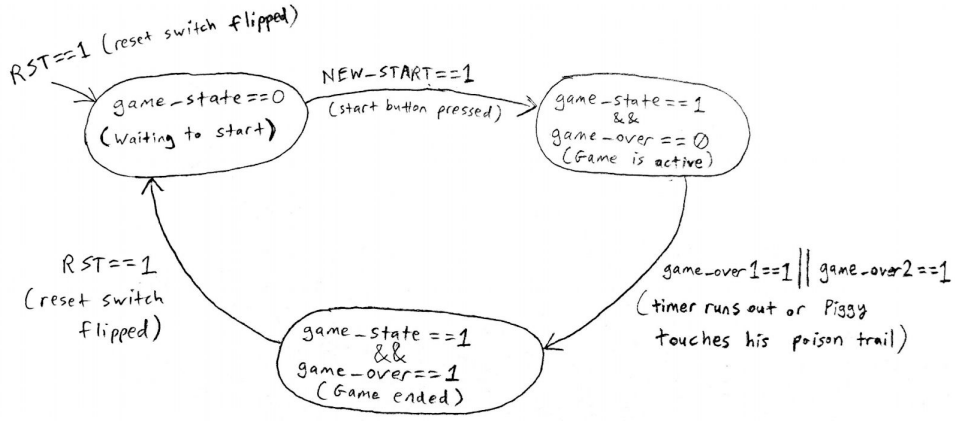

Figure 13: State diagram of game states: game_state and game_over

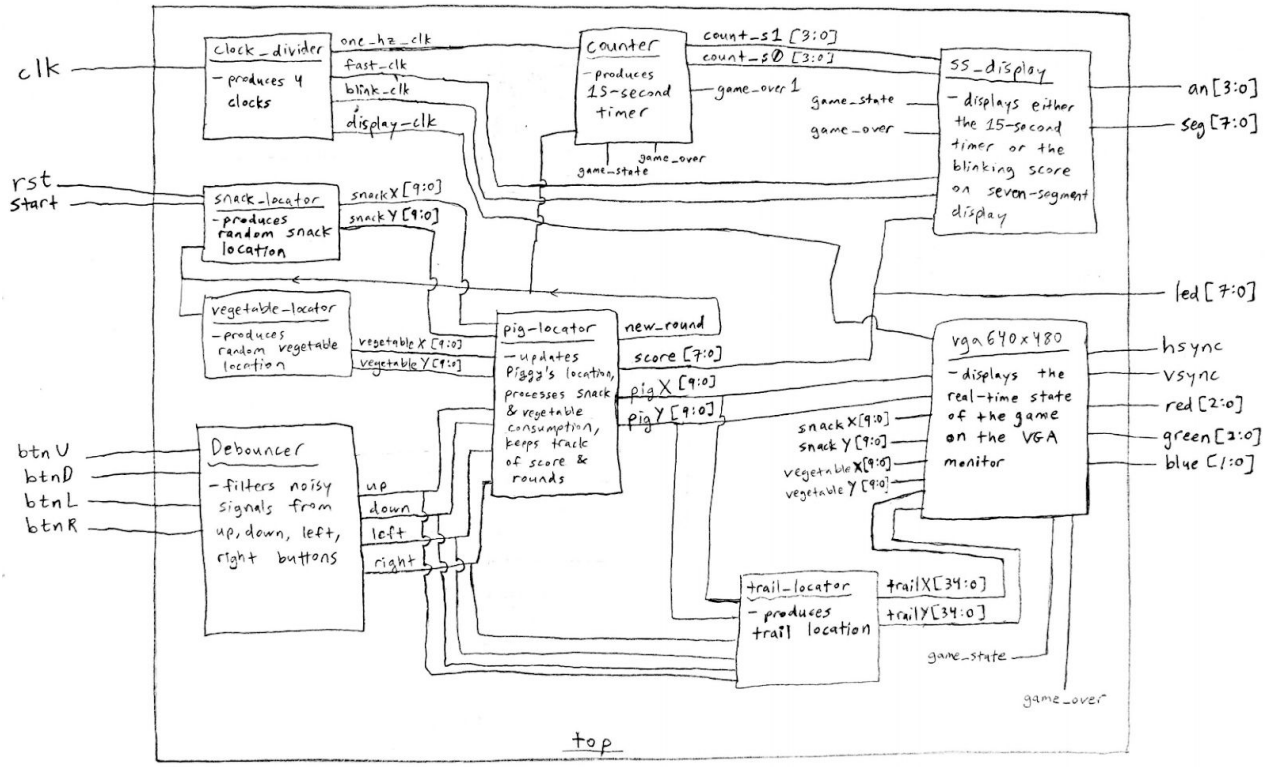

Figure 14: Overall schematic of Piggy's Poisonous Farts: top module + 9 inner modules

Simulation and Documentation

We first tested some modules independently. The clock_divider, counter, Debouncer, and seven-segment display were tested as in Lab 3 (see Lab 3 report). We next tested the trail_locator with the testbench shown in **Figure 15**.

```
27      game_over = 0;
28      round=10;
29      clk = 0;
30      game_state = 0;
31      left = 0;
32      up = 0;
33      right = 0;
34      down = 0;
35      pig_growth = 0;
36      rst = 1;
37
38      #4 rst = 0;
39      #8 game_state = 1;
40      #4 pigX = 300; pigY = 250;
41      #4 down = 1;
42      #2 down = 0;
43      #2 left = 1;
44      #2 left = 0;
45      #2 up = 1;
46      #4 up = 0;
47      #2 right = 1;
48      #4 right = 0;
```

Figure 15: Testbench for trail_locator

The testbench first resets the game and initializes signals to default values. A series of move instructions (down, left, up, up, right, right) are then input to the module. The waveform is shown in **Figure 16**. With each move instruction, the next highest 4 bits of trailX and trailY are updated to contain the coordinate of the next trail point. As displayed, the first move results in the lowest bits of the trail arrays being set to (x,y)==(6,5), which are predetermined starting values corresponding to pixel coordinates (300,250). The first move causes Piggy to move down to (300,300), which correspond to values (6, 6) in the trail arrays since 300/50==6. The next move causes these coordinates to be moved into the trail arrays because the pig's previous location should be marked by a trail. This pattern continues on, with moves to the right causing the next trail coordinate inserted into trailX to be one greater than the previous x-coordinate, and moves to the left to be one less than the previous x-coordinate. Up and down moves behave similarly for the trailY array.

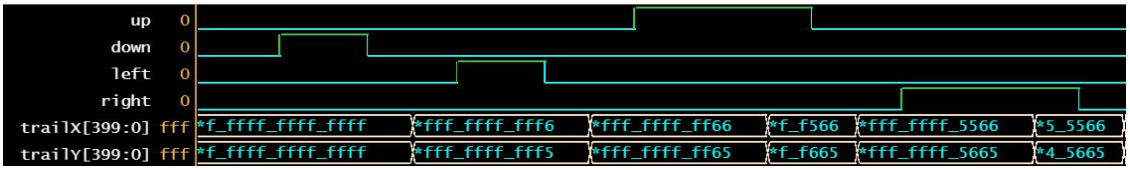

Figure 16: Waveform of trail_locator tests

We then tested the basic movement functionality of the pig_locator with a testbench nearly identical to that shown in **Figure 15** except pigX and pigY are the outputs of the module. The movement pattern input to the module is the same. As shown in **Figure 17**, rst correctly causes pigX to be set to its starting value of 300 (0x12c in hex) and pigY to be set to its starting value of 250 (0xfa in hex). Moving down correctly updates pigY to 300, since the pig moves 50 pixels down the screen. Similarly, each move to the right increases pigX by 50, each move to the left decreases pigX by 50, and each move up decreases pigY by 50. There is a slight delay after the up/down/left/right signals and the new pigX/pigY values so that game logic (e.g. snack/vegetable consumption) can be processed, but this delay is unnoticeable in real time.

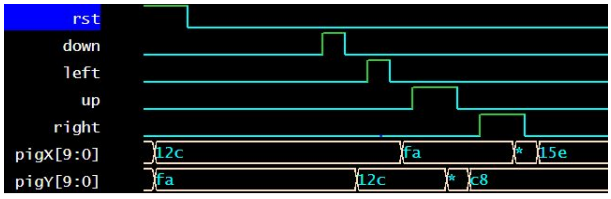

Figure 17: Pig_locator waveform output

Further, we tested that our vga640x480 module outputs correct RGB codes at particular coordinates in the game screen. For one test case, we hard-coded Piggy to be at coordinates (144, 31) in the screen, or (10010000, 11111) in binary. **Figure 18** shows that when VGA counters 'hc' and 'vc' are at these coordinates, displayPig is set to 1 and RGB codes are set to pink. We tested the display of the snack, vegetable, and trail similarly.

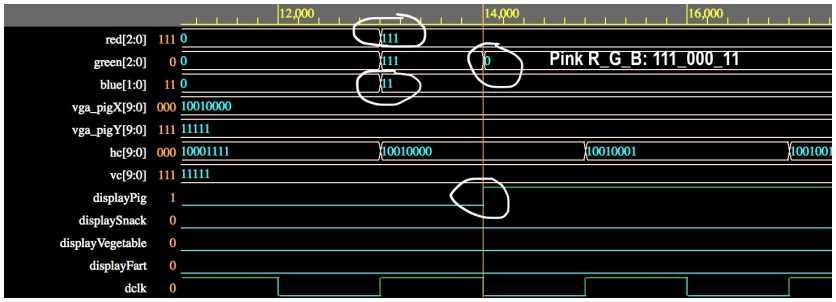

Figure 18: vga640x480 waveform: Pink is displayed for Piggy

We tested the more complicated functionalities of the pig_locator, as well as the vga640x480, snack_locator, and vegetable_locator modules by running the game on the FPGA board and observing the output produced on the screen. **Table 1** shows the tests performed.

Requirement Tested	Test Performed	Result
VGA displays the correct colors and objects on the screen (vga640x480)	Played until 5th round to get all objects on the screen	VGA correctly displays Piggy as pink, snack as blue, trail as orange, and vegetable as green (see **Figure 19**)
2 step restarting process: rst + start (top module)	Flipped reset switch, pressed start button	Game correctly restarted, counter started counting down. Piggy became moveable.
Piggy moves in correct direction (pig_locator and vga640x480)	Turned on VGA display; moved Piggy up, down, left, right	Each directional button correctly moves Piggy in the corresponding direction.
Snacks appear on the screen and are eaten correctly (snack_locator), and LEDs show score	Overlapped the top, left, right, and bottom edges of Piggy with a snack on the screen	Overlapping part of Piggy with a snack causes the snack to reappear in a new location. Also, Piggy's size increases. Score increases, displayed by the LEDs.
Vegetables appear every five rounds and shrink Piggy when eaten (vegetable_locator)	Played until 15th round, overlapped Piggy with vegetable	Vegetable appears in random location every 5 rounds and reduces Piggy's size when eaten
Touching poison trail ends the game (top and	Overlapped Piggy with a poison trail on	The game correctly ended (and displayed correct game over functionality as

trail_locator)	the screen	described below)
The trail gets longer each round until a maximum length (trail_locator)	Played 20 rounds	Observed that the trail got longer each round until staying at a constant length after 15 rounds
Game over functionality (VGA and seven-segment display)	Started game and waited for timer to reach zero	The game ended; a blue screen with a frowny face appeared after a 2 second delay (see **Figure 19**). The seven-segment display blinked the final score.

Table 1: List of test cases

Figure 19: VGA display: Round 5 of active game mode (left) and game over mode (right)

Simulations revealed several bugs that we eventually fixed. For example, the VGA was displaying wrong colors for our objects, which we fixed by bounding the screen by the horizontal and vertical front and back porches. Also, trail_locator initially placed poison on the coordinates that Piggy was about to move to rather than the coordinate he was moving from, which we fixed by using the tempX and tempY registers to keep track of Piggy's movements. Both of these functionalities were explained earlier.

Conclusion

The top module interconnects all 9 inner modules and keeps track of the game state. The snack_locator and vegetable_locator modules place snacks and vegetables at random locations on the screen. The pig_locator keeps track of Piggy's location and increases/decreases Piggy's size when he eats a snack/vegetable. The trail_locator keeps track of the locations of poison on the screen and adjusts the trail length based on the round. The vga640x480 displays all objects on the screen. The clock_divider, Debouncer, counter, and ss_display provide supplementary functionality for the other modules, such as keeping track of a 15-second timer for each round. Initially, we underestimated the difficulty of implementing certain game features. For pig growth, we realized that a variable pig size causes the trail to misalign when turning 90 degrees (even worse when turning and consuming snack/vegetable). Also, we realized that hardware space on the FPGA is very limited, affecting our trail length. After many office hours sessions and hours working from home, we were able to resolve these issues with the logic we described in the explanations for pig_locator and trail_locator ("extra move upon turning" and using 4 bits instead of 10 for each coordinate). We learned to adjust our game logic to overcome obstacles that we did not predict while writing the proposal, and it was a very rewarding experience.