Class Project Report

Yoshi's Nightmare

Mohammed Abdrabalamir Alshammasi

06/12/2021

Table of Contents

1. Introduction and Background on Display via VGA

The VGA output consists of a total of 14 output bits, four bits for each red, green, and blue pixel intensity and two control pulse signals for horizontal and vertical synchronization (`hsync & vsync`). There are two periods for VGA signal timing, the Blanking period and the Display Area period. The Blanking period is the period of time between drawing each row of pixels where no pixels are being drawn. The Display Area period is the period of time where we are drawing pixels on the display with red, green, and blue pixel intensities. The pixels are drawn in a raster-scan format, starting from the top left moving right to the top right then going down one row until a full frame is drawn. The horizontal and vertical control signals pulse at regular intervals depending on the resolution to determine the period we are currently at.

The resolution specified for the project is 1280x800. Two counters were built to generate the horizontal and vertical control pulse signals and output colours onto the display. The Horizontal Counter *`hcount`* starts from 0, incrementing every cycle up to 1679 cycles then loops back to 0 repeatedly. The Vertical Counter *`vcount`* starts from 0 counting up to 827 and is only incremented when the horizontal counter has completed a whole cycle. For our resolution `hsync` is active low, hence it is assigned to be low when the `hcount` is in the range 0 to 135 inclusive and high otherwise. On the other hand, `vsync` is not active low, hence it is assigned to be high when `vcount` is in the range 0 to 2 inclusive and low otherwise. These sync signals pulses produce our desired resolution and control the circuitry within the display to function.

Next, to output pixels onto the display we first need to determine whether we are within the display area. To do that, two signals were created *`within_h` and `within_v`* which are set to be high when `hcount` is in the range 336 to 1615 inclusive and `vcount` is in the range 27 to 826 respectively. Two new counters are built, *`curr_x`* and *`curr_y`*, to determine the current pixel coordinates. These counters are only activated if we are within the display area as determined by `within_h` and `within_v`. The counting logic for `curr_x` and `curr_y` is similar to `hcount` and `vcount`, with `curr_x` counting from the range 0 to 1279 inclusive and `curr_y` from the range 0 to 799 inclusive. This fully describes the VGA output module, the generated pixel coordinates counters are finally used in the drawing module `drawcon` to draw objects on the screen.

Every object has a width and a height which can be specified in terms of the number of horizontal and vertical pixels (across an x and y range). The `drawcon` module outputs the red, green, and blue pixel values of the current pixel being drawn according to the pixel coordinates counters. To draw an object, the RGB pixel values of that object are outputted when the pixel coordinate counters are within the specified region of the object. In the case of having overlapping objects (e.g.

distinguish between foreground and background), a priority order is fixed by using a chain of **if-else** statements. Since we can only output the RGB values of one of the objects, the RGB values of the object with the higher priority are outputted.

2. Design Description

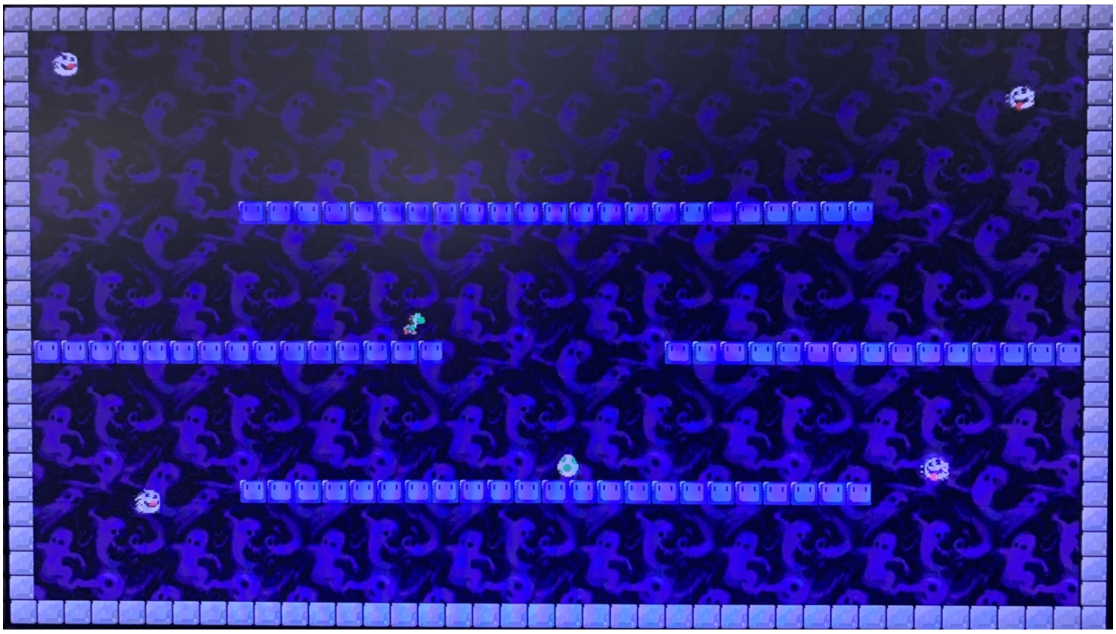

Figure 1: The Game: Yoshi's Nightmare

The game is called Yoshi's Nightmare, it is based on a character named Yoshi from the Nintendo Super Mario series. There are enemy ghosts called Boo's which are the traditional enemy characters in the Super Mario series. These ghosts chase Yoshi around the game map while the player attempts to gather as many Yoshi eggs as possible to gain score points without being caught and terrified to death.

The game design can be broken down into 6 main components: (1) Board I/O (Buttons, Switches, Seven Segment Display, and RGB LED), (2) Movement including Jumping & Gravity effect, (3) Platforms, (4) Enemy Ghosts including Health Points (HP) & Damage, (5) Eggs and Score Points, and lastly (6) Sprites. Below is a list of the game design modules, not including VGA and `drawcon`, with brief descriptions:

- **game_top:** top module where all design modules are interconnected as well as the logic for terminating the game when the character dies.
- **game_logic:** contains the logic for the main character (Yoshi) movement including jumping & gravity, and borders/platforms collision checking.
- **ghosts_logic:** contains the logic for enemy ghosts (Boo's) movement and collision checking with Yoshi to inflict damage.
- **eggs_logic:** contains the logic to randomise the egg spawn location and collision checking with Yoshi to gain score.

All the modules are instantiated in `game_top`: the modules listed above, `vga_out`, `drawcon`, and `multidigit`. The board input/output features such as buttons and LED are connected to the `game_top` input/output ports. Board inputs are wired to the game design modules that use them. An IP block takes the on-board 100MHz clock to generate three different clocks: (1) an 83.46MHz `pixclk` for drawing the screen at the correct pixel rate for our resolution, (2) a 6MHz `posclk` that is further divided using a counter to 60Hz `sixtyhz_clk` for our game design modules, and (3) a 95MHz `displayclk` connected to `multidigit` for the seven segments display. The counter used to divide `posclk` counts up to 50,000 and loops back to 0 repeatedly, flipping the `sixtyhz_clk` signal every 50,000 cycles. The rest of this section describes the circuitry built to produce the logic for each of the main components listed above in detail.

2.1 Board I/O

The design of the game utilizes a number of the board I/O features. Three directional buttons are used for Yoshi's movement. Four switches are used to enable/disable enemy ghosts to adjust the difficulty level the player wants. The four rightmost digits of the seven segment display are used to display the player's score. The four leftmost digits of the seven segment display are used to display the player's health points. Lastly, the RGB LED is used to flash red if a ghost gets close to Yoshi otherwise it flashes blue. The details of how each of these features are utilized are explained in the sections of the game components where they are used.

2.2 Movement, Jumping, and Gravity

All the logic to implement Yoshi's movement and the gravity effect is in the `game_logic` module. The inputs of the module are: the `sixtyhz_clk` for synchronous blocks, three directional buttons left, right, and up for moving left, right, and jumping respectively. The module outputs the x and y coordinates of Yoshi `yoshi_x` and `yoshi_y`.

The left and right movement of Yoshi is simple. When the left button is pressed and the signal goes high `yoshi_x` is decremented by a fixed value, 4 pixels for every clock edge. Similarly for moving right, when the right button is pressed and the signal goes high `yoshi_x` is incremented by a fixed value, 4 pixels for every clock edge.

To implement jumping and gravity effect on Yoshi, four **signed** signals are used: `jmp_velocity`, `gravity`, `negative_limit`, and `pos_y`. The first signal, `pos_y`, is used to tell Verilog to build correct signed circuitry to perform signed arithmetic on the y coordinate of Yoshi with the other signed signal `jmp_velocity`. Although it is a signed signal, it is ensured that this signal does not go below zero. At every clock edge, the value of `pos_y` is casted to unsigned and assigned to the output `yoshi_y`. The second signal, `jmp_velocity`, is the vertical velocity that is applied to the y coordinate to perform the jumping action and gravity effect. The third signal, gravity, is a constant set to one. Finally, `negative_limit` is a negative constant used as a threshold to ensure that `jmp_velocity` does decrease further than its signal width.

Now that we have defined all the required signed signals to be able to perform the necessary arithmetic, jumping and gravity are implemented as follows. At every rising clock edge, `jmp_velocity` is subtracted from `pos_y` and the gravity constant is subtracted from `jmp_velocity`. By default the value of `jmp_velocity` is zero and whenever it reaches the negative threshold `negative_limit` from constantly subtracting the gravity constant, it is reset back to zero. Now since we are subtracting `jmp_velocity` from `pos_y` constantly, **if** statements are used to ensure that `pos_y` does not exceed the floor/platform y coordinate value so that Yoshi does not go through the floor/platform. Figure 2 shows the Verilog code of the described logic for the map ground. Note that the y coordinate increases from top to bottom in our coordinate system so the illusion of jumping is done by subtracting from y.

```verilog
// 768 is ground y coord, 42 is height
else if ((pos_y + 11'd42 - jmp_velocity) >= 11'd768)
begin
    pos_y <= 11'd726; // 726 + 42 = 768, on the ground
    jmp_velocity <= 11'd0;
end
else
    pos_y <= pos_y - jmp_velocity; // apply gravity effect
    jmp_velocity <= jmp_velocity - gravity; // update velocity constantly
```

```verilog
// Limit the velocity negative value, avoids nasty errors
if (jmp_velocity <= negative_limit)
    jmp_velocity <= 11'd0;

// only jump if we are not jumping already
if (up_btn & !jumping)
begin
    jmp_velocity <= 11'd19;
    jumping = 1'b1; // set jumping flag
end
```

Figure 2: Verilog of Jumping Velocity and Gravity updates to Y coordinate.

A jump is triggered when the up button is pressed on the board. This sets the value of the `jmp_velocity` signal to 19. We have the gravity constant equal to one, hence over the next 19 clock cycles the value of `pos_y` is decremented by `jmp_velocity` which has the values 19, 18, 17,…, 2, 1, 0. When `jmp_velocity` reaches 0 this is when Yoshi reaches the peak jump height. Now since `jmp_velocity` is being decremented by the gravity constant, over the next 19 cycles `jmp_velocity` is negative with values -1, -2, …, -18, -19. Hence the subtraction of a negative number turns into a plus sign which creates the illusion of gravity by adding to `pos_y` at the same rate the subtraction was done. If the value of Yoshi's y coordinate is plotted against the described 38 clock cycles when a jump is triggered, we get a parabola shape. Figure 3 shows an example plot of when Yoshi has a y coordinate equal to 750 and a jump is triggered. In our case the parabola curves upwards because y increases from top to bottom hence jumping is done through subtraction.

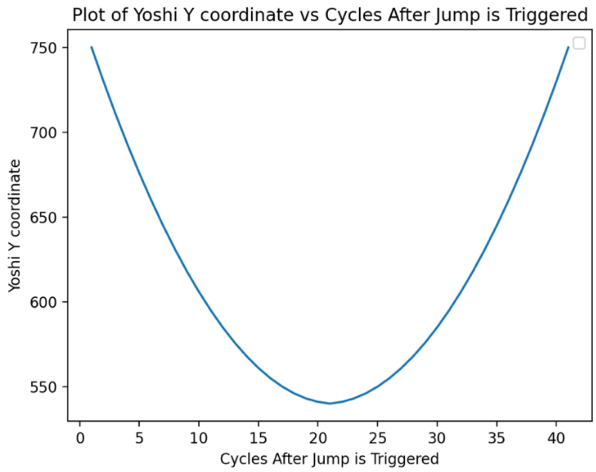

Figure 3: Yoshi Jumping Curve

The final detail is we need to ensure that this effect of jumping cannot be triggered if the character is in the air. This is done by using a 1-bit signal called `jumping` to track if Yoshi is grounded or not. This signal is set to be high when a jump is triggered, otherwise it is set to low when Yoshi's y coordinate is equal to the ground of the map or a platform, setting the `jumping` flag when a jump is triggered can be seen in Fig 2.

2.3 Platforms & Collison Checking

The first thing needed to create platforms is to define the x and y coordinates ranges for each platform. A platform is essentially just a rectangle with a specified width (x range) and height (y range). Figure 4 below shows a screenshot of the x and y coordinate ranges defined as parameters for all 4 platforms. Platforms 1 and 4 are cantered and have the same x range. Platforms 2 and 3 are defined to be on the left and right respectively, and they have the same y range. The platforms can be seen in Figure 1 above. Now that we have defined the platforms parameters, the logic for collision checking Yoshi with the platforms is explained next.

```
// pl is used as abbreviation for platform
// Platform 1 -> Lower center of the screen
parameter pl1_yrange1 = 10'd609;
// platform 4 -> Higher center of the screen
parameter pl4_yrange1 = 9'd249;
// Shared between platforms 1 and 4
parameter pl1and4_xrange1 = 9'd273;
parameter pl1and4_xrange2 = 10'd1008;

// Platform 2 -> Low left of the screen
parameter pl2_xrange1 = 6'd32;
parameter pl2_xrange2 = 10'd511;
// Platform 3 -> Low right of the screen
parameter pl3_xrange1 = 10'd768;
parameter pl3_xrange2 = 11'd1247;
// Shared between platforms 2 and 3
parameter pl2and3_yrange1 = 9'd429;
```

Figure 4: x Platforms x and y coordinates ranges

Six 1-bit signals in total are defined for all platforms to track if Yoshi is within the x range of a platform and whether Yoshi's y coordinate is above the platform or not. We have six signals only because the x range is shared for platforms 1 and 4 and the y range is shared for platforms 2 and 3 as shown in Fig 4 above. The signals for the x range are set to be high if Yoshi's x coordinate is within the range. The signals for the y are set to be high if Yoshi's y coordinate plus Yoshi's height is greater than or equal to the platform y coordinate. These conditions are checked at every rising edge of the clock in a synchronous always block. Figure 5 shows a screenshot of the Verilog code that implements this.

```
// Platform 1 -> Lower center of the screen collision checking,
// 32 is Yoshi's width, 42 is height
above_pl1_y <= (yoshi_y + 10'd42 <= pl1_yrange1);
// Platform 4 -> Higher center of the screen collision checking
above_pl4_y <= (yoshi_y + 10'd42 <= pl4_yrange1);
// In x range, shared between platforms 1 and 4
in_pl1and4_xrange <= ((pl1and4_xrange1-9'd32) <= yoshi_x) & (yoshi_x <= pl1and4_xrange2);

// Platform 2 -> Low left of the screen collision checking
in_pl2_xrange <= ((pl2_xrange1) <= yoshi_x) & (yoshi_x <= pl2_xrange2);
// Platform 3 -> Low right of the screen collision checking
in_pl3_xrange <= ((pl3_xrange1-9'd32) <= yoshi_x) & (yoshi_x <= pl3_xrange2);
// Above y, shared between platforms 2 and 3
above_pl2and3_y <= (yoshi_y + 10'd42 <= pl2and3_yrange1);
```

Figure 5: Platforms Collision Checking Signals

Now that we have signals that tell us if Yoshi is within the platform horizontal range and above it, all that is left is to apply the collision checking to Yoshi's y coordinate so that he stays on the platform. If both the x range and above y signals for a given platform are high then gravity brings Yoshi down no further than the platform y coordinate and sets Yoshi's y to the platform floor y coordinate. This is done using **if** statements, Figure 6 shows this check for platform 4 as an example. The final detail is we need to ensure a priority order for platforms since Yoshi can be within the range and above more than one platform. This priority order is imposed using a chain of **if-else** statements with the highest priority for the highest platform decreasing as we go lower.

```
if (in_pl1and4_xrange & above_pl4_y) // in x range, above y
begin
    // Make sure Yoshi stays on the platform
    if ((pos_y + 11'd42 - jmp_velocity) >= pl4_yrange1)
    begin
        // platform 4 ground y coordinate
        pos_y <= 11'd207; // 207 + 42 = 249

        // reset velocity to 0 when Yoshi is grounded so
        // that when he walk off the platform the gravity
        // effect starts feels natural
        jmp_velocity <= 11'd0;
    end
    else
        pos_y <= pos_y - jmp_velocity; // apply gravity effect
end
```

Figure 6: Platform 4 Collision Checking Code Example

2.4 Enemy Ghosts including Health Points & Damage

All the logic to implement the ghosts movement is in the `ghosts_logic` module. The inputs of the module are: the `sixtyhz_clk` for synchronous blocks, four switches signals to enable/disable four ghosts, and the character Yoshi x and y coordinates. The module outputs the x and y coordinates of each ghost 1 to 4 `ghost1_x`, `ghost1_y`, …, `ghost4_x`, `ghost4_y`, and the RGB values `led_r`, `led_g`, `led_b` for the

RGB LED. Four 1-bit `got_hit1`, ..., `got_hit4` signals for each ghost are also output. These are used to signal if a ghost has dealt damage to Yoshi.

The logic for all four ghosts movement is exactly the same, the only difference between the ghosts is their movement speed with ghost 1 being the slowest and ghost 4 being the fastest. Therefore, for the sake of brevity, in the rest of this section the implementation details applies to all four ghosts without the need to explicitly mention each one of them individually. In the case of declaring a wire/reg used to check for some event, multiple wires are declared in the code for each ghost.

The ghosts movement is simple, the goal of the ghosts is to catch the character Yoshi hence their movement is dependent on Yoshi's x and y coordinates. For example, when Yoshi's position is to the left of a ghost, the ghost should start moving left towards Yoshi. Similarly for the other directions: up, down, and right the same logic is applied. For the ghost to move left and right, the `ghost_x` is decremented and incremented by a fixed value respectively. For the ghost to move up and down, the `ghost_y` is decremented and incremented by a fixed value respectively. In a synchronous always block, **if** statements are used to compare the `ghost_x` with `yoshi_x` and `ghost_y` with `yoshi_y` to apply the movement towards Yoshi. Figure 7 shows the code that implements this for ghost 1 as an example, the horizontal movement applied to `ghost_x` is on the left and vertical movement applied to `ghost_y` is on the right. When the game is started, the starting position of each ghost is fixed, they start at the corners of the map with ghost 1 at the top left, ghost 2 at the top right, ghost 3 at the bottom left, and ghost 4 at the bottom right. Each ghost only appears if its corresponding switch on the board is enabled.

```
if (ghost1_x < yoshi_x)
begin
    ghost1_x <= ghost1_x + GHOST1_X_SPEED; // move right

    // Distance check for RGB Leds!
    if (yoshi_x - ghost1_x <= CLOSE_DISTANCE)
        ghost1_x_close <= 1'b1;
    else
        ghost1_x_close <= 1'b0;
end
else if (ghost1_x > yoshi_x)
begin
    ghost1_x <= ghost1_x - GHOST1_X_SPEED; // move left

    // Distance check for RGB Leds!
    if (ghost1_x - yoshi_x <= CLOSE_DISTANCE)
        ghost1_x_close <= 1'b1;
    else
        ghost1_x_close <= 1'b0;
end
```

```
if (ghost1_y < yoshi_y)
begin
    ghost1_y <= ghost1_y + GHOST1_Y_SPEED; // move down

    // Distance check for RGB Leds!
    if (yoshi_y - ghost1_y <= CLOSE_DISTANCE)
        ghost1_y_close <= 1'b1;
    else
        ghost1_y_close <= 1'b0;
end
else if (ghost1_y > yoshi_y)
begin
    ghost1_y <= ghost1_y - GHOST1_Y_SPEED; // move up

    // Distance check for RGB Leds!
    if (ghost1_y - yoshi_y <= CLOSE_DISTANCE)
        ghost1_y_close <= 1'b1;
    else
        ghost1_y_close <= 1'b0;
end
```

Figure 7: Ghost 1 Movement Example

The ghosts inflict damage and reduce Yoshi's health points when they reach/touch him. Both Yoshi and the ghosts hitboxes are just rectangles hence the check for inflicting damage is the overlap of two rectangles. A 1-bit `overlap` signal is used to check for overlap using an **assign** statement. An overlap is detected using four simple comparisons of the rectangles corner points which are ANDed together. Figure 8 shows the code that checks for overlap between Yoshi and ghost 1. If the overlap of a ghost goes high, the corresponding got_hit output is set to decrement the health points digits in game_top which are displayed on the four leftmost digits of the seven segment display. If the player's health points reaches zero a game_over signal is set and the display shows a purple colour across the whole frame.

```
// YOSHI_SIZE is the width=32 to be a bit more specific
assign overlap1 = ((yoshi_x <= ghost1_x + GHOSTS_SIZE) &
                   (yoshi_x + YOSHI_SIZE >= ghost1_x) &
                   (yoshi_y <= ghost1_y + GHOSTS_SIZE) &
                   (yoshi_y + YOSHI_SIZE >= ghost1_y)
);
```

Figure 8: Overlap/Collision Check for Yoshi & Ghost 1

The RGB LED lights up red when a ghost gets close to Yoshi otherwise it lights up blue. A 1-bit ghost_close signal is used to track if a ghost is close to Yoshi. Another two 1-bit signals ghost_x_close and ghost_y_close are defined to check if the ghost has gotten close to Yoshi across both x and y coordinates respectively. A ghost is considered to be close to Yoshi if the difference between the ghost_x and yoshi_x, and ghost_y and yoshi_y is less than a specified threshold CLOSE_DISTANCE. If the difference in the x coordinates is below the threshold the ghost_x_close signal is set to be high. Similarly for the y coordinates, if the difference is less than the threshold then the ghost_y_close signal for the ghost is set to be high. These x and y distance difference checks can be seen in Figure 7 above. The AND operator is then used on the ghost_x_close and ghost_y_close signals in the assign statement to ghost_close. Finally, in a combinational always block the outputs led_r, led_g, and led_b are set to red if one of the ghost_close signals (since we have one for each ghost) is high otherwise they are set to blue.

2.5 Eggs and Score Points

All the logic to implement the eggs random spawn location is in the eggs_logic module. The input ports of the module are: the clock for synchronous blocks and the character Yoshi x and y coordinates. The module outputs the x and y coordinates of the egg and the four score digits to be displayed on the seven segments display.

The player gains score points through collecting eggs that spawn randomly around the map on the floor or platforms only. There are 20 possible egg spawn locations in total, they are labelled on figure 9 below. The egg spawn locations x coordinates on the ground and each platform were calculated to be split evenly across the surface. Only 1 egg can exist at a time and its location does not change unless it is collected by the player. The location of the first egg is constant, but the location of subsequent eggs is random and depends on when Yoshi collects the previous egg (more specifically the clock edge).

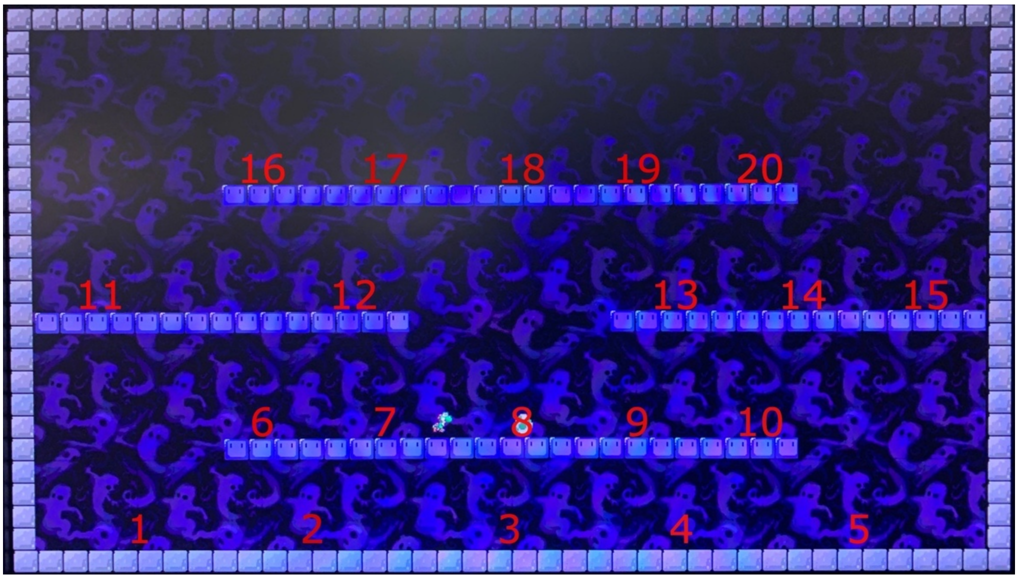

Figure 9: Egg Spawn Locations Labelled in Red

The score for the game is between 0 and 9999, each collected egg gives one score point. An egg is collected when a collision/overlap between Yoshi and the egg occurs, which increments the score digits that are displayed.

The randomness of the egg spawn location relies on the value stored in two registers to determine which spawn location is chosen. A 4-bit register `rand_egg_y`, four bits because we have 4 levels of height (y values) we would like the egg to spawn at: (1) ground, (2) platform 1, (3) platforms 2 and 3, and (4) platform 4. The second value is stored in a 5-bit register `rand_egg_x` for 5 different x coordinates for the egg to spawn at depending on the value of `rand_egg_y`

In order to achieve randomising the egg spawn location our registers should have a single high bit each that corresponds to the active y and x coordinate values to activate a single spawn location at a time. This is an implementation of a serial to parallel shift register to keep shifting the random egg y and x coordinates over time. The idea to implement randomness this way is inspired from shifting the anode activation for the `multidigit` seven segment display. The `rand_egg_y` is initialised at the ground level (equal to 0001). Then using a serial to parallel shift register at each rising edge a '0' bit is fed into the LSB to shift the activated y coordinate. When a cycle is completed from the ground (0001) to platform 4 (1000) a '1' bit is fed into the LSB to start a new cycle. `Rand_egg_x` uses exactly the same procedure, with value 00001 meaning the leftmost x coordinate and 10000 means the rightmost x coordinate.

Next, two registers are defined to store the `next_egg_x` and `next_egg_y` values depending on the values of the `rand_egg_x` and `rand_egg_y` registers. **Case** statements in a combinational always block are used to set the values of `next_egg_x` and `next_egg_y`. Note that the order of the case statements matter inside the always block since the x coordinate values depends on the y coordinate. We have 5 possible x coordinate values for every y level (4 levels, ground and 3 platform levels). Registers `x_option1`, …, `x_option5` are assigned to store these x values temporarily in the case statement of `rand_egg_y`. In the second case statement of `rand_egg_x`, `next_egg_x` is assigned one of these values. Figure 10 shows all case values of `rand_egg_y` and `rand_egg_x` with what they set the `next_egg` and `x_option` registers to. Figure 11 shows the case statements that implement this, only two cases in the `rand_egg_y` case statement is shown for brevity.

rand_egg_y value	**Set next_egg_y to:**
0001 →	Ground y coordinate
0010 →	Platform 1 y coordinate
0100 →	Platforms 2 and 3 y coordinate
1000 →	Platform 4 y coordinate

Set x_option1, ..., x_option5 to:

The x_option 1-5 registers are assigned x coordinate values within the x range of the selected y coordinate surface. For example, platform 1 x range is from 273 to 1008 hence the x coordinate of the egg MUST be within this range to be on the platform. The values for each surface are hard coded.

rand_egg_x value	**Set next_egg_x to:**
00001 →	x_option1
00010 →	x_option2
00100 →	x_option3
01000 →	x_option4
10000 →	x_option5

Figure 10: Assigned values to `next_egg` and `x_option` registers based on `rand_egg`

```
// Case statement to set the y coordinate for
// the next egg position and set values of the
// x_option registers to appropriate values that
// are WITHIN the selected surface x range
case(rand_egg_y)
    4'b0001: begin // ground
                next_egg_y = ground;
                x_option1 = 11'd140;
                x_option2 = 11'd350;
                x_option3 = 11'd560;
                x_option4 = 11'd770;
                x_option5 = 11'd980;
             end
    4'b0010: begin // platform 1
                next_egg_y = pl1_ground;
                x_option1 = 11'd280;
                x_option2 = 11'd450;
                x_option3 = 11'd620;
                x_option4 = 11'd790;
                x_option5 = 11'd960;
             end
```

```
// Case statement to set the x for the next egg
// position. x_option registers values have been
// assigned coordinate values based on rand_egg_y
// in the previous case statement
case(rand_egg_x)
    5'b00001: next_egg_x = x_option1;
    5'b00010: next_egg_x = x_option2;
    5'b00100: next_egg_x = x_option3;
    5'b01000: next_egg_x = x_option4;
    5'b10000: next_egg_x = x_option5;
endcase
```

Figure 11: `rand_egg_x,y` Case Statements to Assign to `next_egg_x,y`

The egg is collected when Yoshi reaches/touches the egg. Similar to ghosts inflicting damage, both the Egg and Yoshi hitboxes are rectangular. Therefore, the same overlap check described in the previous section is used for collecting the eggs, which can be seen in figure 8. Only when the overlap signal goes high, meaning Yoshi has touched the egg, the output `egg_x` and `egg_y` signals are assigned the values stored in `next_egg_x` and `next_egg_y`. It is important to note that the values in `next_egg_x` and `next_egg_y` change on every clock edge since they depend on the values in `rand_egg_x` and `rand_egg_y` which are shifted repeatedly on every clock edge. This constant shifting of the active y and x coordinates creates the desired effect of randomising the egg spawn locations based on the time the current egg is collected.

The final detail is the four digit score points counter which is incremented when Yoshi collides with an egg and collects it. When an egg is collected, the first digit `score_dig1` is always incremented. The second digit `score_dig2` is incremented only when `score_dig1` is 9. The third digit `score_dig3` is incremented only when `score_dig1` and `score_dig2` are 9. The fourth digit `score_dig4` is incremented only when `score_dig1`, `score_dig2`, and `score_dig3` are 9. The first digit loops back to 0 after reaching 9. The second digit loops back to 0 when it reaches 9 and `score_dig1` is 9. The third digit loops back to 0 when it reaches 9 and both `score_dig1` and `score_dig2` are 9. The fourth digit loops back to 0 when it reaches 9 and the first three digits are 9. All these conditions are implemented using **if** statements and only happen when Yoshi collides with an egg.

2.6 Sprites

Once all game components have been implemented, it was finally time to make the game look nice using sprites. All objects in the game use sprites, the border walls, the platforms, the eggs, the character is Yoshi, the enemy ghosts are Boo's, and the background. Eight sprite images are used in total, figure 12 shows all of them and their dimensions. All sprites except Yoshi's are taken from [1], Yoshi's sprites are from [2]. A script from [3] was used to convert the sprites to .coe files which are loaded to initialise memory blocks generated using the Block Memory Generator IP.

Figure 12: Game Sprites

The implementation of sprites requires us to address the memory blocks storing the RGB pixel values properly. The idea of sprites is to fetch these RGB pixel values from memory when we are in the region of the object we are trying to draw instead of just assigning a fixed colour. The general method of addressing the memory blocks correctly is the same for all sprites, with a slight change for the top and bottom wall borders, platforms, and the background because in these cases we are drawing multiple copies of the sprite for a certain horizontal range. Sprites are implemented in the `drawcon` module; all IP memory blocks are instantiated in `drawcon` as well.

Drawing the egg sprites is the simplest case because only 1 sprite is being drawn at a time without connecting it to any extra logic to flip/turn horizontally as done for Yoshi/ghosts. Hence the memory addressing will be explained using the egg sprite as an example. The slight addressing change for drawing multiple copies of the sprite and extra logic connected to flip Yoshi/ghosts horizontally is explained later.

Two offset registers `egg_x_os` and `egg_y_os` are defined to keep track of the x and y coordinates offset within the egg object region we are drawing. The memory address that stores the RGB pixel value for pixels at the `egg_x_os` and `egg_y_os` offsets within the egg drawing region is calculated using the following equation:

$$memory\ address\ at\left(egg_{x_{os}}, egg_{y_{os}}\right) = \ EGG\ WIDTH * egg_{y_{os}} + egg_{x_{os}} \qquad (1)$$

Where EGG WIDTH is the width of the egg sprite image, it is 32 in this case. This equation can be used for any sprite, replacing EGG WIDTH with the image width. This assumes that each address in the memory stores the RGB value of a single pixel hence it depends on the way the RGB values are organised and stored in the memory. In this project all memory blocks store the value of pixel 0,0 at address 0 and as we move in a raster-scan format in the image the address is incremented. Using the egg as an example, pixel 0,1 is stored at address 1, pixel 0,2 is stored at address 2, …, pixel 0,31 is stored at address 31. Moving to the next row in the image, pixel 1,0 is stored at address 32, …, pixel 1, 31 is stored at address 61 and so on. This method of calculating the memory address using offsets within the object allows us to fetch the RGB values and draw the sprite shape for our object. The `egg_x_os` offset tracker is incremented on every clock edge when our `draw_x` and `draw_y` pixel counters are within the egg object region. It is reset to 0 when the egg sprite width is reached. The `egg_y_os` offset tracker is incremented only when the `egg_x_os` tracker has completed a whole row. The logic and implementation of these offset trackers is exactly the same as `hcount, vcount` for the VGA output.

In the case of drawing multiple copies of the sprite across a horizontal range such as when drawing the top and bottom borders, platforms, or background we add a third counter signal for the whole horizontal range. The y offset tracker is now only incremented when this third counter has completed a whole row instead of incrementing when the x offset tracker has completed a row of the sprite.
For example, to draw the top border wall across the whole display width which is 1280 pixels, our third counter counts from 0 to 1279. The y offset tracker for drawing the top border is only incremented when a whole row is completed (1280 cycles). This way we only move to drawing the next row of the sprite when the current sprite

row has been replicated across the whole horizontal range specified. The final detail for this to work correctly is that the horizontal range needs to be divisible by the sprite width. In the case of the top and bottom borders, 1280 is divisible by 32 which gives us 40 replications of the wall sprite. The width of the central platforms 1 and 4 is 736 pixels which is divisible by 32 and gives us 23 platform sprite replications. The width of the left and right platforms 2 & 3 is 480 pixels, dividing by 32 gives us 15 platform sprite replications. The background sprite replication follows exactly the same logic and the implementation is similar. Figure 13 shows screenshot of the code that implements memory addressing for platform 2 sprites.

```
else if (pl2_x & pl2and3_y)
begin
    pl_mem_addr <= PLATFORM_WIDTH * leftpl_y_os + leftpl_x_os;

    if (leftpl_x_os == 5'd31)
        leftpl_x_os <= 6'd0; // reset offset tracker
    else
        leftpl_x_os <= leftpl_x_os + 1'b1; // increment x offset tracker

    // Only increment y when we finished drawing the entire row
    if (pl2_row_count == 11'd479) // we are drawing 15 sprites consecutively, width of pl2 is 480 pixels
    begin
        pl2_row_count = 11'd0;

        if (leftpl_y_os == 5'd31)
            leftpl_y_os <= 6'd0; // reset offset tracker
        else
            leftpl_y_os <= leftpl_y_os + 1'b1;
    end
    else
        pl2_row_count <= pl2_row_count + 1'b1; // increment row counter
end
```

Figure 13: Platform 2 Sprite Addressing, Offset Trackers are Underlined in Red

Ghosts and Yoshi have 2 sprites for facing left and right. The `drawcon` module takes Yoshi's x and y coordinates and all the ghosts x and y coordinates. **If** statements are used to check if Yoshi's x coordinate is to the left or right of a ghost x coordinate similar to the ghosts movement logic explained in section 2.4. Depending on which if statement condition is satisfied, left or right, the ghost left or right sprite is drawn. For Yoshi, the left and right board buttons inputs are used to determine if Yoshi is moving left or right. If the left button is pressed then we draw Yoshi facing left, if the right button is pressed we draw Yoshi facing right. If you press the two buttons simultaneously nothing is drawn. The last bit of extra logic needed to decide which direction sprite to draw when Yoshi is standing still i.e. no buttons are pressed. A 1-bit register `last_dir` is used to remember which button (left or right) was last pressed. This signal is set to low if the left button is pressed and set to high if the right button is pressed. Hence the value stored in this register remembers what was the last direction Yoshi faced so we keep drawing it when he is standing still.

3. Testing Description

In the early stages of the project before the game design was started, two test benches were created. These two testbenches were used to check that the counters and pulses in the `vga_out` module were correctly generated. The waves of the internal signals of the module were added to the simulation window. The simulation was then ran for a limited amount of time, enough clock cycles for the counters to count up to their threshold and loop back to 0. This is how the counters `hcount`, `vcount`, `curr_x`, and `curr_y` were checked to be correct. By ensuring that they loop back to 0 when they reach their thresholds and are incremented according to their specification. `Hcount` was incremented every cycle while `vcount` is only incremented when `hcount` has completed a whole cycle. The `curr` counters were only incremented during the display area and are correctly aligned with the visible pixels. `curr_x` is incremented every cycle while `curr_y` is only incremented when `curr_x` has completed a whole row.

Once the game development has started, all testing was carried out by synthesising the design and running it on the board. The testing of the design was done by visually inspecting the results on the display to see if it matches the expected/desired behaviour. Starting with movement, jumping, and gravity, using the board buttons to interact with the game and make sure that the actions are as desired. Checking the speed of the character moving left and right is not too slow or fast, the jumping duration and height, and the gravity effect is working as intended. Then platforms were added and tested. Testing the collision checking of the character with the platforms by jumping and walking on the entire platform to ensure that there are no false spots and the character does not fall through the platform. Added features like jumping were ensured to be working for the platforms similar to the stage ground.

Next, the ghosts were added to the game and tested. Testing the ghosts movement by moving the character all around the map to ensure that their movement is implemented correctly and no undesired behaviour is observed. Then health points were added to the seven segments display. The health points were tested by colliding the character with the ghosts and checking that the number on the display is decremented correctly. The last game feature added is the eggs and score points. Similar to the health points, eggs were tested by colliding the character with the eggs and checking that the score number on the display is incremented as expected. Finally, sprites were added to the game components one by one. Testing the sprites was done by visually comparing the sprite drawn on the display with the original images to make sure that they match.

4. Reflection

I really enjoyed working on this project, in fact I enjoyed it so much I wished that I did not have other courses. The amount of freedom we had in designing our games and open-endedness of the project allows room for creativity and to demonstrate the skills acquired from the course. I disliked the long waiting times for the design to synthesise and generate a bitstream every time, even when only minimal changes were done. This made testing time consuming. When facing a bug in the implementation, there is no source of debugging information from within the code only warning or errors through the log.

I was only introduced to Verilog in this course and after this project I feel confident in being able to think through problems and build circuitry to solve them. The logic and methods I learnt for implementing some of the game design features do translate closely to software game design.

I feel that the project should be individual since all the project objectives can be achieved individually with the given amount of time. Improvements can be made to the demo marking scheme by including more details for each criteria. For example, the 4 marks for I/O features can be divided to 3 marks for using any three features and the last mark for using a more advanced feature such as accelerometer, audio, etc. The 4 marks for sprites can be detailed to: 1 mark for implementing a sprite, 1 marking for replicating or using multiple sprites, 1 mark for adding logic that interacts with the sprite, and finally 1 mark for animating an object. I feel that these changes to the demo marking criteria could encourage thinking about the design to include more variety.

5. References:

[1] Game idea and Wall, Platform, Ghosts, Eggs, and Background Sprites: https://embeddedthoughts.com/2016/12/09/yoshis-nightmare-fpga-based-video-game/

[2] Yoshi Sprites: https://www.spriters-resource.com/snes/yoshiisland/

[3] Image to COE File Script: https://github.com/Jesse-Millwood/image-2-coe

[4] Sprites In Block ROM: https://www.youtube.com/watch?v=EWQ3s9NxGEI

[5] Jumping and Gravity Explanation: https://medium.com/@brazmogu/physics-for-game-dev-a-platformer-physics-cheatsheet-f34b09064558

[6] Overlap Check: https://stackoverflow.com/questions/19753134/get-the-points-of-intersection-from-2-rectangles

[7] Creating ROM with Vivado: https://web.mit.edu/6.111/volume2/www/f2019/handouts/labs/lab3_19/rom_vivado.html

Appendix A
Verilog Code Files

A.1 Game Top Source File

Source file of the top module where all design modules are interconnected as well as the logic for terminating the game when the character dies.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/27/2021 07:05:44 PM
// Design Name:
// Module Name: game_top
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module game_top(input clk, rst,
                input left_btn, right_btn, up_btn, // character control
                input enemy1, enemy2, enemy3, enemy4, // enemy switches
                output a, b, c, d, e, f, g, // seven segments display
                output [7:0] an, // anode's for multi-digit seven segments
                output [3:0] pix_r, pix_g, pix_b, // pixel RGB output
                output led_b, led_g, led_r, // RGB LED
                output hsync, vsync); // control signals for display


    wire pixclk; // IP block generates an 83.46MHz clock
    wire posclk; // IP block generates a 6MHz clock
    wire displayclk; // IP block generates a 95 MHz clock
    clock_div clock_dividers (.clk_out1(pixclk), .clk_out2(posclk), .clk_out3(displayclk), .clk_in1(clk));

    // We require a 60hz clock for the position logic,using a counter we further divide the 6MHz to 60Hz
    wire [15:0] constant = 16'd49999;
    reg [15:0] clk_count = 16'd0;
    always@(posedge posclk)
    begin
        if (rst)
            clk_count <= 16'd0;
        else if (clk_count == constant)
            clk_count <= 16'd0;
        else
            clk_count <= clk_count + 1'b1;
    end

```

```
    reg sixtyhz_clk;
    always@(posedge posclk)
    begin
        if (rst)
            sixtyhz_clk <= 1'b0;
        else if (clk_count == constant)
            sixtyhz_clk <= ~sixtyhz_clk;
        else
            sixtyhz_clk <= sixtyhz_clk;
    end

    // vga_out module instantiation
    wire [10:0] curr_x;
    wire [9:0] curr_y;
    wire within; // within display area or not
    vga_out vga1 (.clk(pixclk),
                  .rst(rst),
                  .red_v(draw_r),
                  .green_v(draw_g),
                  .blue_v(draw_b),
                  // Outputs
                  .within(within),
                  .pix_r(pix_r),
                  .pix_g(pix_g),
                  .pix_b(pix_b),
                  .hsync(hsync),
                  .vsync(vsync),
                  .curr_x(curr_x),
                  .curr_y(curr_y)
    );

    // game_logic module instantiation
    wire [10:0] yoshi_x;
    wire [9:0] yoshi_y;
    game_logic logic1 (.clk(sixtyhz_clk),
                       .rst(rst),
                       .left_btn(left_btn),
                       .right_btn(right_btn),
                       .up_btn(up_btn),
                       // Outputs
                       .yoshi_x(yoshi_x),
                       .yoshi_y(yoshi_y)
    );

    // ghosts_logic module instantiation
    wire [10:0] ghost1_x, ghost2_x, ghost3_x, ghost4_x;
    wire [9:0] ghost1_y, ghost2_y, ghost3_y, ghost4_y;
    wire got_hit1, got_hit2, got_hit3, got_hit4;
    ghosts_logic logic2 (.clk(sixtyhz_clk),
                         .rst(rst),
                         .ghost1(enemy1),
                         .ghost2(enemy2),
                         .ghost3(enemy3),
                         .ghost4(enemy4),
                         .yoshi_x(yoshi_x),
                         .yoshi_y(yoshi_y),
                         // Outputs
                         // RGB Leds
                         .led_b(led_b),
                         .led_g(led_g),
                         .led_r(led_r),
                         // Inflicting damage signals
                         .got_hit1(got_hit1),
                         .got_hit2(got_hit2),
                         .got_hit3(got_hit3),
                         .got_hit4(got_hit4),
```

```
                                // ghosts positions x,y
                                .ghost1_x(ghost1_x),
                                .ghost2_x(ghost2_x),
                                .ghost3_x(ghost3_x),
                                .ghost4_x(ghost4_x),
                                .ghost1_y(ghost1_y),
                                .ghost2_y(ghost2_y),
                                .ghost3_y(ghost3_y),
                                .ghost4_y(ghost4_y)
        );

        // eggs_logic module instantiation
        wire [10:0] egg_x;
        wire [9:0] egg_y;
        wire [3:0] score_dig1, score_dig2, score_dig3, score_dig4;
        eggs_logic logic3 (.clk(sixtyhz_clk),
                           .rst(rst),
                           .yoshi_x(yoshi_x),
                           .yoshi_y(yoshi_y),
                           // Ouputs
                           .egg_x(egg_x),
                           .egg_y(egg_y),
                           .score_dig1(score_dig1),
                           .score_dig2(score_dig2),
                           .score_dig3(score_dig3),
                           .score_dig4(score_dig4)
        );

        // Drawcon instantiation
        wire [3:0] draw_r, draw_g, draw_b;
        reg game_over = 1'b0;
        drawcon draw1 (.clk(pixclk),
                       .game_over(game_over),
                       .within(within),
                       .left_btn(left_btn),
                       .right_btn(right_btn),
                       // Character related signals
                       .yoshi_x(yoshi_x),
                       .yoshi_y(yoshi_y),
                       // Ghosts related signals
                       // switches
                       .ghost1(enemy1),
                       .ghost2(enemy2),
                       .ghost3(enemy3),
                       .ghost4(enemy4),
                       // x, y coordinates
                       .ghost1_x(ghost1_x),
                       .ghost2_x(ghost2_x),
                       .ghost3_x(ghost3_x),
                       .ghost4_x(ghost4_x),
                       .ghost1_y(ghost1_y),
                       .ghost2_y(ghost2_y),
                       .ghost3_y(ghost3_y),
                       .ghost4_y(ghost4_y),
                       // Eggs related signals
                       .egg_x(egg_x),
                       .egg_y(egg_y),
                       // Current drawing pixel position signals
                       .draw_x(curr_x),
                       .draw_y(curr_y),
                       // Outputs
                       .r(draw_r),
                       .g(draw_g),
                       .b(draw_b)
        );

```

```
// seven segments display module instantiation
wire digits_clk;
multidigit seginterface (.clk(displayclk),
                         .rst(rst),
                         // Score digits
                         .dig0(score_dig1),
                         .dig1(score_dig2),
                         .dig2(score_dig3),
                         .dig3(score_dig4),
                         // Health digits
                         .dig4(hp_dig1),
                         .dig5(hp_dig2),
                         .dig6(hp_dig3),
                         .dig7(hp_dig4),
                         // Outputs
                         .div_clk(digits_clk),
                         .a(a),
                         .b(b),
                         .c(c),
                         .d(d),
                         .e(e),
                         .f(f),
                         .g(g),
                         .an(an)
);

// Our health points digits, we start with 3 health points
// hp_dig2-4 are defined for the sake of completeness, not used
reg [3:0] hp_dig1 = 4'd3;
reg [3:0] hp_dig2 = 4'd0;
reg [3:0] hp_dig3 = 4'd0;
reg [3:0] hp_dig4 = 4'd0;

// Based on the "got_hit" signal from the ghosts_logic module decrement health
// Signal GAME OVER when health is decremented to 0
always @ (posedge digits_clk)
begin
    // Hit signal for each ghost 1-4
    if (got_hit1)
    begin
        if (hp_dig1 == 4'd0)
            game_over <= 1'b1;
        else
            hp_dig1 <= hp_dig1 - 1'b1;
    end

    if (got_hit2)
    begin
        if (hp_dig1 == 4'd0)
            game_over <= 1'b1;
        else
            hp_dig1 <= hp_dig1 - 1'b1;
    end

    if (got_hit3)
    begin
        if (hp_dig1 == 4'd0)
            game_over <= 1'b1;
        else
            hp_dig1 <= hp_dig1 - 1'b1;
    end

    if (got_hit4)
    begin
```

```
                if (hp_dig1 == 4'd0)
                    game_over <= 1'b1;
                else
                    hp_dig1 <= hp_dig1 - 1'b1;
            end
        end
    endmodule
```

A.2 VGA Out Source File

Contains the VGA output signalling protocol where counters are used to generate the correct sync pulses for our display resolution and counters for pixel coordinates.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/27/2021 10:36:43 AM
// Design Name:
// Module Name: vga_out
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module vga_out(input clk, rst,
               input [3:0] red_v, green_v, blue_v,
               output [3:0] pix_r, pix_g, pix_b,
               output reg within,
               output hsync, vsync,
               output reg [10:0] curr_x,
               output reg [9:0] curr_y);

    reg [10:0] hcount;
    reg [9:0] vcount;

    always@(posedge clk)
    begin
        if (rst)
        begin
            hcount <= 11'b0;
            vcount <= 10'b0;
        end
        else
        begin
            if (hcount == 11'd1679)
            begin
                hcount <= 11'b0;

                if (vcount == 10'd827)
                    vcount <= 10'b0;
                else
                    vcount <= vcount + 1'b1;
            end
            else
```

```
                hcount <= hcount + 1'b1;
        end
    end

    // When hcount is between 0 and 135 we assign 0 to hsync, otherwise we assign 1
    assign hsync = (11'd0 <= hcount) & (hcount <= 11'd135) ? 0 : 1;
    assign vsync = (10'd0 <= vcount) & (vcount <= 10'd2) ? 1 : 0;

    // Set the pixel signals to be the input from drawcon when we are within the display area
    wire within_h, within_v;
    assign within_h = (11'd336 <= hcount) & (hcount <= 11'd1615);
    assign within_v = (10'd27 <= vcount) & (vcount <= 10'd826);
    assign pix_r = (within_h & within_v) ? red_v : 4'd0;
    assign pix_g = (within_h & within_v) ? green_v : 4'd0;
    assign pix_b = (within_h & within_v) ? blue_v : 4'd0;

    always@(posedge clk)
    begin
        if (rst)
        begin
            curr_x <= 11'b0;
            curr_y <= 10'b0;
        end
        else
        begin
            if (within_h & within_v)
            begin
                // Signal goes to drawcon to tell if we are within the screen
                // using this makes the blanking period not mess up the offset trackers
                within <= 1'b1;

                if (curr_x == 11'd1279)
                begin
                    curr_x <= 11'b0;

                    if (curr_y == 10'd799)
                        curr_y <= 10'b0;
                    else
                        curr_y <= curr_y + 1'b1;
                end
                else
                    curr_x <= curr_x + 1'b1;
            end
            else
                within <= 1'b0;
        end
   end
      endmodule
```

A.3 Game Logic Source File

This source file contains the logic for the main character (Yoshi) movement including jumping & gravity, and borders/platforms collision checking.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/03/2021 10:42:50 PM
// Design Name:
// Module Name: game_logic
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module game_logic(input clk, rst,
                  input left_btn, right_btn, up_btn,
                  output reg [10:0] yoshi_x,
                  output reg [9:0] yoshi_y);

    // Platforms ranges parameters
    // pl is used as abbreviation for platform
    // Platform 1 -> Lower center of the screen
    parameter pl1_yrange1 = 10'd609;
    // platform 4 -> Higher center of the screen
    parameter pl4_yrange1 = 9'd249;
    // Shared between platforms 1 and 4
    parameter pl1and4_xrange1 = 9'd273;
    parameter pl1and4_xrange2 = 10'd1008;


    // Platform 2 -> Mid left of the screen
    parameter pl2_xrange1 = 6'd32;
    parameter pl2_xrange2 = 10'd511;
    // Platform 3 -> Mid right of the screen
    parameter pl3_xrange1 = 10'd768;
    parameter pl3_xrange2 = 11'd1247;
    // Shared between platforms 2 and 3
    parameter pl2and3_yrange1 = 9'd429;

    // Collision Checking for yoshi with platforms
    reg above_pl1_y = 1'b0; // above platform 1
    reg above_pl4_y = 1'b0; // above platform 4
    reg in_pl1and4_xrange = 1'b0; // shared between platforms 1 and 4
    reg in_pl2_xrange = 1'b0; // within platform 2 x range
```

```
    reg in_pl3_xrange = 1'b0; // within platform 3 x range
    reg above_pl2and3_y = 1'b0; // shared between platforms 2 and 3
    always@(posedge clk)
    begin
        // Platform 1 -> Lower center of the screen collision checking,
        // 32 is Yoshi's width, 42 is height
        above_pl1_y <= (yoshi_y + 10'd42 <= pl1_yrange1);
        // Platform 4 -> Higher center of the screen collision checking
        above_pl4_y <= (yoshi_y + 10'd42 <= pl4_yrange1);
        // In x range, shared between platforms 1 and 4
        in_pl1and4_xrange <= ((pl1and4_xrange1-9'd32) <= yoshi_x) & (yoshi_x <= pl1and4_xrange2);

        // Platform 2 -> Low left of the screen collision checking
        in_pl2_xrange <= ((pl2_xrange1) <= yoshi_x) & (yoshi_x <= pl2_xrange2);
        // Platform 3 -> Low right of the screen collision checking
        in_pl3_xrange <= ((pl3_xrange1-9'd32) <= yoshi_x) & (yoshi_x <= pl3_xrange2);
        // Above y, shared between platforms 2 and 3
        above_pl2and3_y <= (yoshi_y + 10'd42 <= pl2and3_yrange1);
    end


    // Signed jumping velocity, gravity constant, and negative limit
    reg signed [10:0] pos_y; // need a signed pos y signal for calculations
    reg signed [10:0] jmp_velocity = 11'd0;
    reg signed [10:0] negative_limit = -11'd30;
    reg signed [10:0] gravity = 11'd1;
    reg jumping;
    always@(posedge clk)
    begin
        if (rst)
        begin
            yoshi_x <= 11'd640;
            pos_y <= 11'd756;
        end
        else
        begin
            // Move left when the left button is pressed
            if (left_btn)
            begin
                if (yoshi_x >= 11'd36) // Don't hit left border
                    yoshi_x <= yoshi_x - 11'd4;
            end

            // Move right when the right button is pressed
            if (right_btn)
            begin
                if (yoshi_x + 11'd32 <= 11'd1243) // Don't hit the right border
                    yoshi_x <= yoshi_x + 11'd4;
            end

            // If gravity pulls us lower than the floor/platform, set to floor/platform y
            // If the character is on the floor/platform then we are not jumping
            // We set the velocity to 0 while grounded because we want to keep the gravity
```

```
                // effect natural when the character walks off a platform i.e. gravity will start
                // from 0 until the character hits the ground.
                if (in_pl1and4_xrange & above_pl4_y) // in x range, above y
                begin
                    // Make sure Yoshi stays on the platform
                    if ((pos_y + 11'd42 - jmp_velocity) >= pl4_yrange1)
                    begin
                        // platform 4 ground y coordinate
                        pos_y <= 11'd207; // 207 + 42 = 249

                        // reset velocity to 0 when Yoshi is grounded so
                        // that when he walk off the platform the gravity
                        // effect starts feels natural
                        jmp_velocity <= 11'd0;
                    end
                    else
                        pos_y <= pos_y - jmp_velocity; // apply gravity effect
                end
                else if (in_pl3_xrange & above_pl2and3_y) // platform 3
                begin
                    if ((pos_y + 11'd42 - jmp_velocity) >= pl2and3_yrange1)
                    begin
                        pos_y <= 11'd387;
                        jmp_velocity <= 11'd0;
                    end
                    else
                        pos_y <= pos_y - jmp_velocity; // apply gravity effect
                end
                else if (in_pl2_xrange & above_pl2and3_y) // platform 2
                begin
                    if ((pos_y + 11'd42 - jmp_velocity) >= pl2and3_yrange1)
                    begin
                        pos_y <= 11'd387;
                        jmp_velocity <= 11'd0;
                    end
                    else
                        pos_y <= pos_y - jmp_velocity; // apply gravity effect
                end
                else if (in_pl1and4_xrange & above_pl1_y) // platform 1
                begin
                    if ((pos_y + 11'd42 - jmp_velocity) >= pl1_yrange1)
                    begin
                        pos_y <= 11'd567;
                        jmp_velocity <= 11'd0;
                    end
                    else
                        pos_y <= pos_y - jmp_velocity; // apply gravity effect
                end
                // 768 is ground y coord, 42 is height
                else if ((pos_y + 11'd42 - jmp_velocity) >= 11'd768)
                begin
                    pos_y <= 11'd726; // 726 + 42 = 768, on the ground
                    jmp_velocity <= 11'd0;
```

```
            end
            else
                pos_y <= pos_y - jmp_velocity; // apply gravity effect
                jmp_velocity <= jmp_velocity - gravity; // update velocity constantly

            // Limit the velocity negative value, avoids nasty errors
            if (jmp_velocity <= negative_limit)
                jmp_velocity <= 11'd0;

            // only jump if we are not jumping already
            if (up_btn & !jumping)
            begin
                jmp_velocity <= 11'd19;
                jumping = 1'b1; // set jumping flag
            end

            if (pos_y - jmp_velocity <= 11'd31) // dont jump through the ceiling
            begin
                pos_y <= 11'd32;
                jmp_velocity <= 11'd0;
            end

            if (pos_y == 11'd726) // on floor
            begin
                jumping <= 1'b0;
            end
            else if (pos_y == 11'd567) // on platform 1
            begin
                jumping <= 1'b0;
            end
            else if (pos_y == 11'd387) // on platform 2 or 3
            begin
                jumping <= 1'b0;
            end
            else if (pos_y == 11'd207) // on platform 4
            begin
                jumping <= 1'b0;
            end

            yoshi_y <= $unsigned(pos_y); // assign unsigned to output signal
        end
    end
endmodule
```

A.4 Ghosts Logic Source File

This source file contains the logic for enemy ghosts (Boo's) movement and collision checking with Yoshi to inflict damage as well as Yoshi & Ghosts distance check for the RGB LED.

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/09/2021 05:35:43 PM
// Design Name:
// Module Name: ghosts_logic
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////


module ghosts_logic(input clk, rst,
                    input ghost1, ghost2, ghost3, ghost4,
                    input [10:0] yoshi_x,
                    input [9:0] yoshi_y,
                    output reg led_b, led_g, led_r,
                    output reg got_hit1, got_hit2, got_hit3, got_hit4,
                    output reg [10:0] ghost1_x, ghost2_x, ghost3_x, ghost4_x,
                    output reg [9:0] ghost1_y, ghost2_y, ghost3_y, ghost4_y);

    parameter YOSHI_SIZE = 6'd32;
    // Ghosts parameters: size, and speed in the x and y directions
    parameter GHOSTS_SIZE = 6'd32;
    // Ghost 1
    parameter GHOST1_X_SPEED = 11'd1;
    parameter GHOST1_Y_SPEED = 10'd1;
    // Ghost 2
    parameter GHOST2_X_SPEED = 11'd2;
    parameter GHOST2_Y_SPEED = 10'd2;
    // Ghost 3
    parameter GHOST3_X_SPEED = 11'd3;
    parameter GHOST3_Y_SPEED = 10'd3;
    // Ghost 4
    parameter GHOST4_X_SPEED = 11'd4;
    parameter GHOST4_Y_SPEED = 10'd4;

    // Checks if there is any overlap between the character and a ghost to apply damage
```

```
    // YOSHI_SIZE is the width=32 to be a bit more specific
    assign overlap1 = ((yoshi_x <= ghost1_x + GHOSTS_SIZE) &
                       (yoshi_x + YOSHI_SIZE >= ghost1_x) &
                       (yoshi_y <= ghost1_y + GHOSTS_SIZE) &
                       (yoshi_y + YOSHI_SIZE >= ghost1_y)
    );

    assign overlap2 = ((yoshi_x <= ghost2_x + GHOSTS_SIZE) &
                       (yoshi_x + YOSHI_SIZE >= ghost2_x) &
                       (yoshi_y <= ghost2_y + GHOSTS_SIZE) &
                       (yoshi_y + YOSHI_SIZE >= ghost2_y)
    );

    assign overlap3 = ((yoshi_x <= ghost3_x + GHOSTS_SIZE) &
                       (yoshi_x + YOSHI_SIZE >= ghost3_x) &
                       (yoshi_y <= ghost3_y + GHOSTS_SIZE) &
                       (yoshi_y + YOSHI_SIZE >= ghost3_y)
    );

    assign overlap4 = ((yoshi_x <= ghost4_x + GHOSTS_SIZE) &
                       (yoshi_x + YOSHI_SIZE >= ghost4_x) &
                       (yoshi_y <= ghost4_y + GHOSTS_SIZE) &
                       (yoshi_y + YOSHI_SIZE >= ghost4_y)
    );


    // Signals used to check if any of the ghosts is close to the character
    // these are used to light up the RGB Leds to red if a ghost is near
    parameter CLOSE_DISTANCE = 8'd150;
    reg ghost1_x_close, ghost1_y_close;
    reg ghost2_x_close, ghost2_y_close;
    reg ghost3_x_close, ghost3_y_close;
    reg ghost4_x_close, ghost4_y_close;
    assign ghost1_close = (ghost1_x_close & ghost1_y_close);
    assign ghost2_close = (ghost2_x_close & ghost2_y_close);
    assign ghost3_close = (ghost3_x_close & ghost3_y_close);
    assign ghost4_close = (ghost4_x_close & ghost4_y_close);

    always @ *
    begin
        if (ghost1_close)
        begin
            led_b = 1'b0;
            led_g = 1'b0;
            led_r = 1'b1;
        end

        if (ghost2_close)
        begin
            led_b = 1'b0;
            led_g = 1'b0;
            led_r = 1'b1;
        end

        if (ghost3_close)
        begin
            led_b = 1'b0;
            led_g = 1'b0;
            led_r = 1'b1;
        end

```

```
            if (ghost4_close)
            begin
                led_b = 1'b0;
                led_g = 1'b0;
                led_r = 1'b1;
            end

            if (!ghost1_close & !ghost2_close & !ghost3_close & !ghost4_close)
            begin
                led_b = 1'b1;
                led_g = 1'b0;
                led_r = 1'b0;
            end
        end

        always @ (posedge clk)
        begin
            if (rst)
            begin
                // Ghost 1
                if (ghost1)
                begin
                    ghost1_x <= 11'd12;
                    ghost1_y <= 10'd12;
                end

                // Ghost 2
                if (ghost2)
                begin
                    ghost2_x <= 11'd1250;
                    ghost2_y <= 10'd12;
                end

                // Ghost 3
                if (ghost3)
                begin
                    ghost3_x <= 11'd12;
                    ghost3_y <= 10'd760;
                end

                // Ghost 4
                if (ghost4)
                begin
                    ghost4_x <= 11'd1250;
                    ghost4_y <= 10'd760;
                end
            end
            else
            begin
                // Ghost 1
                if (ghost1)
                begin
                    if (ghost1_x < yoshi_x)
                    begin
                        ghost1_x <= ghost1_x + GHOST1_X_SPEED; // move right

                        // Distance check for RGB Leds!
                        if (yoshi_x - ghost1_x <= CLOSE_DISTANCE)
                            ghost1_x_close <= 1'b1;
```

```
                    else
                        ghost1_x_close <= 1'b0;
                end
                else if (ghost1_x > yoshi_x)
                begin
                    ghost1_x <= ghost1_x - GHOST1_X_SPEED; // move left

                    // Distance check for RGB Leds!
                    if (ghost1_x - yoshi_x <= CLOSE_DISTANCE)
                        ghost1_x_close <= 1'b1;
                    else
                        ghost1_x_close <= 1'b0;
                end

                if (ghost1_y < yoshi_y)
                begin
                    ghost1_y <= ghost1_y + GHOST1_Y_SPEED; // move down

                    // Distance check for RGB Leds!
                    if (yoshi_y - ghost1_y <= CLOSE_DISTANCE)
                        ghost1_y_close <= 1'b1;
                    else
                        ghost1_y_close <= 1'b0;
                end
                else if (ghost1_y > yoshi_y)
                begin
                    ghost1_y <= ghost1_y - GHOST1_Y_SPEED; // move up

                    // Distance check for RGB Leds!
                    if (ghost1_y - yoshi_y <= CLOSE_DISTANCE)
                        ghost1_y_close <= 1'b1;
                    else
                        ghost1_y_close <= 1'b0;
                end

                // If the ghost touches the character set signal for hit
                if (overlap1)
                    got_hit1 <= 1'b1;
                else
                    got_hit1 <= 1'b0;
            end

            // Ghost 2
            if (ghost2)
            begin
                if (ghost2_x < yoshi_x)
                begin
                    ghost2_x <= ghost2_x + GHOST2_X_SPEED;

                    // Distance check for RGB Leds!
                    if (yoshi_x - ghost2_x <= CLOSE_DISTANCE)
                        ghost2_x_close <= 1'b1;
                    else
                        ghost2_x_close <= 1'b0;
                end
                else if (ghost2_x > yoshi_x)
                begin
                    ghost2_x <= ghost2_x - GHOST2_X_SPEED;
```

```

                            // Distance check for RGB Leds!
                            if (ghost2_x - yoshi_x <= CLOSE_DISTANCE)
                                ghost2_x_close <= 1'b1;
                            else
                                ghost2_x_close <= 1'b0;
                        end

                        if (ghost2_y < yoshi_y)
                        begin
                            ghost2_y <= ghost2_y + GHOST2_Y_SPEED;

                            // Distance check for RGB Leds!
                            if (yoshi_y - ghost2_y <= CLOSE_DISTANCE)
                                ghost2_y_close <= 1'b1;
                            else
                                ghost2_y_close <= 1'b0;
                        end
                        else if (ghost2_y > yoshi_y)
                        begin
                            ghost2_y <= ghost2_y - GHOST2_Y_SPEED;

                            // Distance check for RGB Leds!
                            if (ghost2_y - yoshi_y <= CLOSE_DISTANCE)
                                ghost2_y_close <= 1'b1;
                            else
                                ghost2_y_close <= 1'b0;
                        end

                        // If the ghost touches the character set signal for hit
                        if (overlap2)
                            got_hit2 <= 1'b1;
                        else
                            got_hit2 <= 1'b0;
                    end

                    // Ghost 3
                    if (ghost3)
                    begin
                        if (ghost3_x < yoshi_x)
                        begin
                            ghost3_x <= ghost3_x + GHOST3_X_SPEED;

                            // Distance check for RGB Leds!
                            if (yoshi_x - ghost3_x <= CLOSE_DISTANCE)
                                ghost3_x_close <= 1'b1;
                            else
                                ghost3_x_close <= 1'b0;
                        end
                        else if (ghost3_x > yoshi_x)
                        begin
                            ghost3_x <= ghost3_x - GHOST3_X_SPEED;

                            // Distance check for RGB Leds!
                            if (ghost3_x - yoshi_x <= CLOSE_DISTANCE)
                                ghost3_x_close <= 1'b1;
                            else
                                ghost3_x_close <= 1'b0;
                        end

```

```
                if (ghost3_y < yoshi_y)
                begin
                    ghost3_y <= ghost3_y + GHOST3_Y_SPEED;

                    // Distance check for RGB Leds!
                    if (yoshi_y - ghost3_y <= CLOSE_DISTANCE)
                        ghost3_y_close <= 1'b1;
                    else
                        ghost3_y_close <= 1'b0;
                end
                else if (ghost3_y > yoshi_y)
                begin
                    ghost3_y <= ghost3_y - GHOST3_Y_SPEED;

                    // Distance check for RGB Leds!
                    if (ghost3_y - yoshi_y <= CLOSE_DISTANCE)
                        ghost3_y_close <= 1'b1;
                    else
                        ghost3_y_close <= 1'b0;
                end

                // If the ghost touches the character set signal for hit
                if (overlap3)
                    got_hit3 <= 1'b1;
                else
                    got_hit3 <= 1'b0;
            end

            // Ghost 4
            if (ghost4)
            begin
                if (ghost4_x < yoshi_x)
                begin
                    ghost4_x <= ghost4_x + GHOST4_X_SPEED;

                    // Distance check for RGB Leds!
                    if (yoshi_x - ghost4_x <= CLOSE_DISTANCE)
                        ghost4_x_close <= 1'b1;
                    else
                        ghost4_x_close <= 1'b0;
                end
                else if (ghost4_x > yoshi_x)
                begin
                    ghost4_x <= ghost4_x - GHOST4_X_SPEED;

                    // Distance check for RGB Leds!
                    if (ghost4_x - yoshi_x <= CLOSE_DISTANCE)
                        ghost4_x_close <= 1'b1;
                    else
                        ghost4_x_close <= 1'b0;
                end

                if (ghost4_y < yoshi_y)
                begin
                    ghost4_y <= ghost4_y + GHOST4_Y_SPEED;

                    // Distance check for RGB Leds!
                    if (yoshi_y - ghost4_y <= CLOSE_DISTANCE)
                        ghost4_y_close <= 1'b1;
                    else
```

```
                                ghost4_y_close <= 1'b0;
                        end
                        else if (ghost4_y > yoshi_y)
                        begin
                            ghost4_y <= ghost4_y - GHOST4_Y_SPEED;

                            // Distance check for RGB Leds!
                            if (ghost4_y - yoshi_y <= CLOSE_DISTANCE)
                                ghost4_y_close <= 1'b1;
                            else
                                ghost4_y_close <= 1'b0;
                        end

                        // If the ghost touches the character set signal for
   hit
                        if (overlap4)
                            got_hit4 <= 1'b1;
                        else
                            got_hit4 <= 1'b0;
                    end
                end
            end
      endmodule
```

A.5 Eggs Logic Source File

This source file contains the logic to randomise the egg spawn location and collision checking with Yoshi to gain score.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 11/13/2021 06:06:10 PM
// Design Name: 
// Module Name: eggs_logic
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module eggs_logic(input clk, rst,
                  input [10:0] yoshi_x,
                  input [9:0] yoshi_y,
                  output reg [10:0] egg_x,
                  output reg [9:0] egg_y,
                  output reg [3:0] score_dig1, score_dig2, score_dig3, score_dig4);


    // Serial to parallel shift register to keep shifting the random egg x & y positions
    // over time. These x and y indices are used to select the next egg position which only
    // changes when the player consumes the current egg. Hence randomness is achieved by
    // making the next egg position depend on the time (more specifically the clock edge)
    // that the egg was consumed at.
    reg x_index = 1'b0, y_index = 1'b0;
    reg [4:0] rand_egg_x = 5'b00001; // 5 bits just to increase randomness and not align with y
    reg [3:0] rand_egg_y = 4'b0001; // 4 bits because we have 4 levels of height (y value) floor, pl1, pl2&3, pl4

    // Similar logic to anode activation shifting for sevengsegment display
    always @ *
    begin
        x_index = 1'b0;
        y_index = 1'b0;
        if (rand_egg_x == 5'b10000)
```

```
            x_index = 1'b1;

        if (rand_egg_y == 4'b1000)
            y_index = 1'b1;
    end

    // Shift register, shift to the left
    always @ (posedge clk)
    begin
        rand_egg_x[0] <= x_index;
        rand_egg_x[4:1] <= rand_egg_x[3:0];

        rand_egg_y[0] <= y_index;
        rand_egg_y[3:1] <= rand_egg_y[2:0];
    end


    // Parameters for the ground/platform values for the eggs on each level
    parameter ground = 10'd732;
    parameter pl1_ground = 10'd573;
    parameter pl2and3_ground = 10'd393;
    parameter pl4_ground = 10'd213;

    reg [10:0] next_egg_x = 11'd240;
    reg [9:0] next_egg_y = 10'd732;
    reg [10:0] x_option1, x_option2, x_option3, x_option4, x_option5;
    always @ *
    begin
        // Case statement to set the y coordinate for the next egg position and set values of the
        // x_option registers to appropriate values that are WITHIN the selected surface x range
        case(rand_egg_y)
            4'b0001: begin // ground
                        next_egg_y = ground;
                        x_option1 = 11'd140;
                        x_option2 = 11'd350;
                        x_option3 = 11'd560;
                        x_option4 = 11'd770;
                        x_option5 = 11'd980;
                     end
            4'b0010: begin // platform 1
                        next_egg_y = pl1_ground;
                        x_option1 = 11'd280;
                        x_option2 = 11'd450;
                        x_option3 = 11'd620;
                        x_option4 = 11'd790;
                        x_option5 = 11'd960;
                     end
            4'b0100: begin // platforms 2 and 3
                        next_egg_y = pl2and3_ground;
                        x_option1 = 11'd80;
                        x_option2 = 11'd420;
                        x_option3 = 11'd785;
                        x_option4 = 11'd971;
                        x_option5 = 11'd1157;
                           end
                  4'b1000: begin // platform 4, similar to platform 1
                              next_egg_y = pl4_ground;
                              x_option1 = 11'd280;
                              x_option2 = 11'd450;
```

```
                            x_option3 = 11'd620;
                            x_option4 = 11'd790;
                            x_option5 = 11'd960;
                         end
            endcase

            // Case statement to set the x for the next egg position. x_option registers values have been
            // assigned coordinate values based on rand_egg_y in the previous case statement
            case(rand_egg_x)
                5'b00001: next_egg_x = x_option1;
                5'b00010: next_egg_x = x_option2;
                5'b00100: next_egg_x = x_option3;
                5'b01000: next_egg_x = x_option4;
                5'b10000: next_egg_x = x_option5;
            endcase
        end

        // Overlap check for egg consumption
        parameter CHARACTER_SIZE = 6'd32; // wdith, to be more specific
        parameter EGG_SIZE = 6'd32; // width, to be more specific
        assign overlap = ((yoshi_x <= egg_x + EGG_SIZE) &
                          (yoshi_x + CHARACTER_SIZE >= egg_x) &
                          (yoshi_y <= egg_y + EGG_SIZE) &
                          (yoshi_y + CHARACTER_SIZE >= egg_y)
        );

        // Synchronous block to signal consumption of current egg and fetch next egg position
        // Also increment the score counter
        always @ (posedge clk)
        begin
            if (rst)
            begin
                // Egg position
                egg_x <= 11'd640;
                egg_y <= pl1_ground;

                // Score counter
                score_dig1 <= 4'd0;
                score_dig2 <= 4'd0;
                score_dig3 <= 4'd0;
                score_dig4 <= 4'd0;
            end
            else
            begin
                if (overlap)
                begin
                    // Change egg position when collected
                    egg_x <= next_egg_x;
                    egg_y <= next_egg_y;

                    // Increment score counter, this can possibly be in a separte module
                    score_dig1 <= score_dig1 + 1'b1; // increment by 1

                    // score_dig2 increment if score_dig1 is 9
                    if (score_dig1 == 4'd9)
                        score_dig2 <= score_dig2 + 1'b1;

```

```
                    // score_dig3 increment if both score_dig1, score_dig2 are 9
                    if (score_dig1 == 4'd9 & score_dig2 == 4'd9)
                        score_dig3 <= score_dig3 + 1'b1;

                    // score_dig4 increment if all score_dig1-3 are 9
                    if (score_dig1 == 4'd9 & score_dig2 == 4'd9 & score_dig3 == 4'd9)
                        score_dig4 <= score_dig4 + 1'b1;

                    // Loop back to 0 after reaching 9 for the decimal counters
                    if (score_dig1 == 4'd9 & score_dig2 == 4'd9 & score_dig3 == 4'd9 & score_dig4 == 4'd9)
                        score_dig4 <= 4'b0000;

                    if (score_dig1 == 4'd9 & score_dig2 == 4'd9 & score_dig3 == 4'd9)
                        score_dig3 <= 4'b0000;

                    if (score_dig1 == 4'd9 & score_dig2 == 4'd9)
                        score_dig2 <= 4'b0000;

                    if (score_dig1 == 4'd9)
                        score_dig1 <= 4'b0000;
                end
            end
        end
    endmodule
```

A.6 Drawcon Source File

This source file contains the code to draw all objects of the game, including the sprites memory blocks instantiation and addressing.

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/31/2021 05:10:20 PM
// Design Name:
// Module Name: drawcon
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////////////////////


module drawcon(input clk, game_over, within, left_btn, right_btn,
               input ghost1, ghost2, ghost3, ghost4,
               input [10:0] yoshi_x, draw_x, ghost1_x, ghost2_x, ghost3_x, ghost4_x, egg_x,
               input [9:0] yoshi_y, draw_y, ghost1_y, ghost2_y, ghost3_y, ghost4_y, egg_y,
               output reg [3:0] r, g, b);

    // Border and background drawing logic, wall sprite image size is 32x32
    parameter WALL_WIDTH = 6'd32;
    assign left_border = (11'd0 <= draw_x) & (draw_x <= 11'd31) & (10'd32 <= draw_y) & (draw_y <= 10'd767);
    assign right_border = (draw_x >= 11'd1248) & (draw_x <= 11'd1279) & (10'd32 <= draw_y) & (draw_y <= 10'd767);
    assign top_border = (10'd0 <= draw_y) & (draw_y <= 10'd31);
    assign bottom_border = (draw_y >= 10'd768) & (draw_y <= 10'd799);

    // Need to use different signals for left and right border because they get drawn
    // in parallel technically hence the need to use different signals
    reg [5:0] topAndBottom_x_os = 6'd0, topAndBottom_y_os = 6'd0; // top&bottom border x,y offset trackers
    reg [5:0] left_x_os = 6'd0, left_y_os = 6'd0; // left border x,y offset trackers
    reg [5:0] right_x_os = 6'd0, right_y_os = 6'd0; // right border x,y offset trackers
    reg [10:0] row_count = 11'd0;
    reg [3:0] bg_r, bg_g, bg_b;

    // Wall memory instantiation, stores the wall rgb values
```

```
    reg [9:0] wall_mem_addr;
    wire [11:0] wall_rgb;
    wall_mem wall_sprite (.clka(clk),    // input wire clka
                          .addra(wall_mem_addr),  // input wire [9 : 0] addra
                          .douta(wall_rgb)  // output wire [11 : 0] douta
    );

    always @ (posedge clk)
    begin
        if (within) // must be within display area
        begin
            // All the border conditions are mutually exclusive to not mess up the
            // sprite memory addressing
            if (top_border | bottom_border)
            begin
                wall_mem_addr <= WALL_WIDTH * topAndBottom_y_os + topAndBottom_x_os;

                if (topAndBottom_x_os == 5'd31)
                    topAndBottom_x_os <= 6'd0; // reset offset tracker
                else
                    topAndBottom_x_os <= topAndBottom_x_os + 1'b1; // increment x offset tracker

                // Only increment y when we finished drawing the entire row
                if (row_count == 11'd1279)
                begin
                    row_count = 11'd0;

                    if (topAndBottom_y_os == 5'd31)
                        topAndBottom_y_os <= 6'd0; // reset offset tracker
                    else
                        topAndBottom_y_os <= topAndBottom_y_os + 1'b1;
                end
                else
                    row_count <= row_count + 1'b1; // increment row counter
            end
            else if (left_border)
            begin
                wall_mem_addr <= WALL_WIDTH * left_y_os + left_x_os;

                if (left_x_os == 5'd31)
                begin
                    left_x_os <= 6'd0; // reset x offset tracker

                    // reset y when done drawing a tile
                    if (left_y_os == 5'd31)
                        left_y_os <= 6'd0;
                    else
                        left_y_os <= left_y_os + 1'b1; // inc y
                end
                else
                    left_x_os <= left_x_os + 1'b1;
            end
            else if (right_border)
            begin
                wall_mem_addr <= WALL_WIDTH * right_y_os + right_x_os;

                      if (right_x_os == 5'd31)
```

```
                begin
                    right_x_os <= 6'd0; // reset x offset tracker

                    // reset y when done drawing a tile
                    if (right_y_os == 6'd31)
                        right_y_os <= 6'd0;
                    else
                        right_y_os <= right_y_os + 1'b1; // inc y
                end
                else
                    right_x_os <= right_x_os + 1'b1;
            end
        end
    end


    // background sprite drawing
    // background memory instantiation, stores the bg rgb values
    parameter BACKGROUND_WIDTH = 8'd160; // background sprite image width
    reg [7:0] bg_x_os = 8'd0, bg_y_os = 8'd0; // x,y offset trackers
    reg [10:0] bg_row_count = 11'd0; // horizontal range row counter, upto 1280
    reg [14:0] bg_mem_addr;
    wire [11:0] bg_rgb;
    background_mem bg_sprite (.clka(clk),    // input wire clka
                              .addra(bg_mem_addr),  // input wire [9 : 0] addra
                              .douta(bg_rgb)  // output wire [11 : 0] douta
    );

    always @ (posedge clk)
    begin
        if (within) // must be within display area
        begin
            bg_mem_addr <= BACKGROUND_WIDTH * bg_y_os + bg_x_os;

            if (bg_x_os == 8'd159)
                bg_x_os <= 8'd0; // reset offset tracker
            else
                bg_x_os <= bg_x_os + 1'b1; // increment x offset tracker

            // Only increment y when we finished drawing the entire row
            if (bg_row_count == 11'd1279)
            begin
                bg_row_count = 11'd0;

                if (bg_y_os == 8'd159)
                    bg_y_os <= 8'd0; // reset offset tracker
                else
                    bg_y_os <= bg_y_os + 1'b1;
            end
            else
                bg_row_count <= bg_row_count + 1'b1; // increment row counter
        end
    end

```

```
        always @ *
        begin
            if (top_border | left_border | right_border | bottom_border)
            begin
                // Set the fetched colours from wall memory
                bg_r = wall_rgb[11:8];
                bg_g = wall_rgb[7:4];
                bg_b = wall_rgb[3:0];
            end
            else // background
            begin
                bg_r = bg_rgb[11:8];
                bg_g = bg_rgb[7:4];
                bg_b = bg_rgb[3:0];
            end
        end


        // Character/yoshi drawing
        assign character_x = (yoshi_x <= draw_x) & (draw_x <= yoshi_x + 11'd31); // -1 since sprite indexing starts at 0
        assign character_y = (yoshi_y <= draw_y) & (draw_y <= yoshi_y + 10'd41); // -1 since sprite indexing starts at 0
        parameter YOSHI_WIDTH = 6'd32;
        reg [5:0] yoshi_x_os = 6'd0, yoshi_y_os = 6'd0; // x,y offset trackers

        // Yoshi right sprite memory
        reg [10:0] yoshi_right_mem_addr;
        wire [11:0] yoshi_right_rgb;
        yoshi_right_mem yoshi_right_sprite (.clka(clk),    // input wire clka
                                            .addra(yoshi_right_mem_addr),  // input wire [10 : 0] addra
                                            .douta(yoshi_right_rgb)  // output wire [11 : 0] douta
        );

        // Yoshi left sprite memory
        reg [10:0] yoshi_left_mem_addr;
        wire [11:0] yoshi_left_rgb;
        yoshi_left_mem yoshi_left_sprite (.clka(clk),    // input wire clka
                                          .addra(yoshi_left_mem_addr),  // input wire [10 : 0] addra
                                          .douta(yoshi_left_rgb)  // output wire [11 : 0] douta
        );

        // Signal to remember what direction was faced last to know which sprite to draw
        reg last_dir; // 0 means left, 1 means right

        always @ (posedge clk)
        begin
            // Update the last facing direction tracking signal
            if (left_btn)
                last_dir <= 1'b0;
            else if (right_btn)
                last_dir <= 1'b1;

```

```

            // yoshi reset trackers whenever we start fetching rgb sprite values
            if (draw_x == yoshi_x & draw_y == yoshi_y)
            begin
                yoshi_x_os <= 6'd0;
                yoshi_y_os <= 6'd0;
            end

            // yoshi sprite memory addresing logic
            if (character_x & character_y)
            begin
                if (yoshi_x_os == 5'd31)
                begin
                    yoshi_x_os <= 6'd0; // reset x offset tracker

                    // reset y when done drawing 1 row of sprite
                    if (yoshi_y_os == 6'd41)
                        yoshi_y_os <= 6'd0;
                    else
                        yoshi_y_os <= yoshi_y_os + 1'b1; // inc y
                end
                else
                    yoshi_x_os <= yoshi_x_os + 1'b1;

                if (right_btn) // draw right yoshi
                    yoshi_right_mem_addr <= YOSHI_WIDTH * yoshi_y_os + yoshi_x_os;
                else if (left_btn) // draw left yoshi
                    yoshi_left_mem_addr <= YOSHI_WIDTH * yoshi_y_os + yoshi_x_os;
                else // standing, not moving using the buttons
                begin
                    if (last_dir == 1'b0) // standing facing left side coz last input was left
                        yoshi_left_mem_addr <= YOSHI_WIDTH * yoshi_y_os + yoshi_x_os;
                    else if (last_dir == 1'b1) // standing facing right side coz last input was right
                        yoshi_right_mem_addr <= YOSHI_WIDTH * yoshi_y_os + yoshi_x_os;
                end
            end
        end

        reg [3:0] yoshi_r = 4'd0, yoshi_g = 4'd0, yoshi_b = 4'd0;
        always @ *
        begin
            if (character_x & character_y)
            begin
                if (left_btn)
                begin
                    yoshi_r = yoshi_left_rgb[11:8];
                    yoshi_g = yoshi_left_rgb[7:4];
                    yoshi_b = yoshi_left_rgb[3:0];
                end
                else if (right_btn)
                begin
                    yoshi_r = yoshi_right_rgb[11:8];
                    yoshi_g = yoshi_right_rgb[7:4];
                    yoshi_b = yoshi_right_rgb[3:0];
```

```
            end
            else
            begin
                if (last_dir == 1'b0)
                begin
                    yoshi_r = yoshi_left_rgb[11:8];
                    yoshi_g = yoshi_left_rgb[7:4];
                    yoshi_b = yoshi_left_rgb[3:0];
                end
                else if (last_dir == 1'b1)
                begin
                    yoshi_r = yoshi_right_rgb[11:8];
                    yoshi_g = yoshi_right_rgb[7:4];
                    yoshi_b = yoshi_right_rgb[3:0];
                end
            end
        end
        else
        begin
            yoshi_r = 4'd0;
            yoshi_g = 4'd0;
            yoshi_b = 4'd0;
        end
    end

    // Enemies/Ghosts drawing
    parameter GHOST_SIZE = 5'd31; // -1, for the sake of sprites, indexing starts at 0
    parameter GHOST_WIDTH = 6'd32; // Sprite image width
    // Ghost 1
    assign draw_ghost1_x = (ghost1_x <= draw_x) & (draw_x <= ghost1_x + GHOST_SIZE);
    assign draw_ghost1_y = (ghost1_y <= draw_y) & (draw_y <= ghost1_y + GHOST_SIZE);

    // Ghost 2
    assign draw_ghost2_x = (ghost2_x <= draw_x) & (draw_x <= ghost2_x + GHOST_SIZE);
    assign draw_ghost2_y = (ghost2_y <= draw_y) & (draw_y <= ghost2_y + GHOST_SIZE);

    // Ghost 3
    assign draw_ghost3_x = (ghost3_x <= draw_x) & (draw_x <= ghost3_x + GHOST_SIZE);
    assign draw_ghost3_y = (ghost3_y <= draw_y) & (draw_y <= ghost3_y + GHOST_SIZE);

    // Ghost 4
    assign draw_ghost4_x = (ghost4_x <= draw_x) & (draw_x <= ghost4_x + GHOST_SIZE);
    assign draw_ghost4_y = (ghost4_y <= draw_y) & (draw_y <= ghost4_y + GHOST_SIZE);


    // Ghost left and right sprite image memory instantiation, address, rgb, and trackers
    reg [5:0] ghost1_x_os = 6'd0, ghost1_y_os = 6'd0;
    reg [5:0] ghost2_x_os = 6'd0, ghost2_y_os = 6'd0;
    reg [5:0] ghost3_x_os = 6'd0, ghost3_y_os = 6'd0;
    reg [5:0] ghost4_x_os = 6'd0, ghost4_y_os = 6'd0;
    reg [9:0] ghost_right_mem_addr;
```

```
wire [11:0] ghost_right_rgb;
ghost_right_mem ghost_right_sprite (.clka(clk),    // input wire clka
                                   .addra(ghost_right_mem_addr),  // input wire [9 : 0] addra
                                   .douta(ghost_right_rgb)  // output wire [11 : 0] douta
);

reg [9:0] ghost_left_mem_addr;
wire [11:0] ghost_left_rgb;
ghost_left_mem ghost_left_sprite (.clka(clk),    // input wire clka
                                  .addra(ghost_left_mem_addr),  // input wire [9 : 0] addra
                                  .douta(ghost_left_rgb)  // output wire [11 : 0] douta
);

// Synchronous block to address the ghost memory to fetch the rgb colours
always @ (posedge clk)
begin
    // ghost 1 reset trackers whenever we start fetching rgb sprite values
    if (draw_x == ghost1_x & draw_y == ghost1_y)
    begin
        ghost1_x_os <= 6'd0;
        ghost1_y_os <= 6'd0;
    end

    // ghost 2 reset trackers whenever we start fetching rgb sprite values
    if (draw_x == ghost2_x & draw_y == ghost2_y)
    begin
        ghost2_x_os <= 6'd0;
        ghost2_y_os <= 6'd0;
    end

    // ghost 3 reset trackers whenever we start fetching rgb sprite values
    if (draw_x == ghost3_x & draw_y == ghost3_y)
    begin
        ghost3_x_os <= 6'd0;
        ghost3_y_os <= 6'd0;
    end

    // ghost 4 reset trackers whenever we start fetching rgb sprite values
    if (draw_x == ghost4_x & draw_y == ghost4_y)
    begin
        ghost4_x_os <= 6'd0;
        ghost4_y_os <= 6'd0;
    end

    // Ghost 1 sprite memory addresing logic
    if (ghost1)
    begin
        if (draw_ghost1_x & draw_ghost1_y)
        begin
```

```
                if (ghost1_x_os == 5'd31)
                begin
                    ghost1_x_os <= 6'd0; // reset x offset tracker

                    // reset y when done drawing a tile
                    if (ghost1_y_os == 6'd31)
                        ghost1_y_os <= 6'd0;
                    else
                        ghost1_y_os <= ghost1_y_os + 1'b1; // inc y
                end
                else
                    ghost1_x_os <= ghost1_x_os + 1'b1;

                if (ghost1_x < yoshi_x) // fetch from ghost right mem
                    ghost_right_mem_addr <= GHOST_WIDTH * ghost1_y_os + ghost1_x_os;
                else if (ghost1_x > yoshi_x) // fetch from ghost left mem
                    ghost_left_mem_addr <= GHOST_WIDTH * ghost1_y_os + ghost1_x_os;
            end
        end


        // Ghost 2 sprite memory addressing
        if (ghost2)
        begin
            if (draw_ghost2_x & draw_ghost2_y)
            begin

                if (ghost2_x_os == 5'd31)
                begin
                    ghost2_x_os <= 6'd0; // reset x offset tracker

                    // reset y when done drawing a tile
                    if (ghost2_y_os == 6'd31)
                        ghost2_y_os <= 6'd0;
                    else
                        ghost2_y_os <= ghost2_y_os + 1'b1; // inc y
                end
                else
                    ghost2_x_os <= ghost2_x_os + 1'b1;

                if (ghost2_x < yoshi_x)
                    ghost_right_mem_addr <= GHOST_WIDTH * ghost2_y_os + ghost2_x_os;
                else if (ghost2_x > yoshi_x)
                    ghost_left_mem_addr <= GHOST_WIDTH * ghost2_y_os + ghost2_x_os;
            end
        end

        // Ghost 3 sprite memory addressing
        if (ghost3)
        begin
            if (draw_ghost3_x & draw_ghost3_y)
            begin

                if (ghost3_x_os == 5'd31)
                begin
                    ghost3_x_os <= 6'd0; // reset x offset tracker
```

```

                        // reset y when done drawing a tile
                        if (ghost3_y_os == 6'd31)
                            ghost3_y_os <= 6'd0;
                        else
                            ghost3_y_os <= ghost3_y_os + 1'b1; // inc y
                    end
                    else
                        ghost3_x_os <= ghost3_x_os + 1'b1;

                    if (ghost3_x < yoshi_x)
                        ghost_right_mem_addr <= GHOST_WIDTH * ghost3_y_os + ghost3_x_os;
                    else if (ghost3_x > yoshi_x)
                        ghost_left_mem_addr <= GHOST_WIDTH * ghost3_y_os + ghost3_x_os;
                end
            end

            // Ghost 4 sprite memory addressing
            if (ghost4)
            begin
                if (draw_ghost4_x & draw_ghost4_y)
                begin

                    if (ghost4_x_os == 5'd31)
                    begin
                        ghost4_x_os <= 6'd0; // reset x offset tracker

                        // reset y when done drawing a tile
                        if (ghost4_y_os == 6'd31)
                            ghost4_y_os <= 6'd0;
                        else
                            ghost4_y_os <= ghost4_y_os + 1'b1; // inc y
                    end
                    else
                        ghost4_x_os <= ghost4_x_os + 1'b1;

                    if (ghost4_x < yoshi_x)
                        ghost_right_mem_addr <= GHOST_WIDTH * ghost4_y_os + ghost4_x_os;
                    else if (ghost4_x > yoshi_x)
                        ghost_left_mem_addr <= GHOST_WIDTH * ghost4_y_os + ghost4_x_os;
                end
            end
        end

        reg [3:0] ghost1_r = 4'd0, ghost1_g = 4'd0, ghost1_b = 4'd0; // ghost 1 rgb
        reg [3:0] ghost2_r = 4'd0, ghost2_g = 4'd0, ghost2_b = 4'd0; // ghost 2 rgb
        reg [3:0] ghost3_r = 4'd0, ghost3_g = 4'd0, ghost3_b = 4'd0; // ghost 3 rgb
        reg [3:0] ghost4_r = 4'd0, ghost4_g = 4'd0, ghost4_b = 4'd0; // ghost 4 rgb
        always @ *
        begin
            // If ghost 1 is enabled
            if (ghost1)
            begin
```

```
            if (draw_ghost1_x & draw_ghost1_y)
            begin
                if (ghost1_x < yoshi_x)
                begin
                    ghost1_r = ghost_right_rgb[11:8];
                    ghost1_g = ghost_right_rgb[7:4];
                    ghost1_b = ghost_right_rgb[3:0];
                end
                else if (ghost1_x > yoshi_x)
                begin
                    ghost1_r = ghost_left_rgb[11:8];
                    ghost1_g = ghost_left_rgb[7:4];
                    ghost1_b = ghost_left_rgb[3:0];
                end
            end
            else
            begin
                ghost1_r = 4'd0;
                ghost1_g = 4'd0;
                ghost1_b = 4'd0;
            end
        end
        else
        begin
            ghost1_r = 4'd0;
            ghost1_g = 4'd0;
            ghost1_b = 4'd0;
        end


        // If ghost 2 is enabled
        if (ghost2)
        begin
            if (draw_ghost2_x & draw_ghost2_y)
            begin
                if (ghost2_x < yoshi_x)
                begin
                    ghost2_r = ghost_right_rgb[11:8];
                    ghost2_g = ghost_right_rgb[7:4];
                    ghost2_b = ghost_right_rgb[3:0];
                end
                else if (ghost2_x > yoshi_x)
                begin
                    ghost2_r = ghost_left_rgb[11:8];
                    ghost2_g = ghost_left_rgb[7:4];
                    ghost2_b = ghost_left_rgb[3:0];
                end
            end
            else
            begin
                ghost2_r = 4'd0;
                ghost2_g = 4'd0;
                ghost2_b = 4'd0;
            end
        end
        else
        begin
            ghost2_r = 4'd0;
            ghost2_g = 4'd0;
            ghost2_b = 4'd0;
        end
```

```


            // If ghost 3 is enabled
            if (ghost3)
            begin
                if (draw_ghost3_x & draw_ghost3_y)
                begin
                    if (ghost3_x < yoshi_x)
                    begin
                        ghost3_r = ghost_right_rgb[11:8];
                        ghost3_g = ghost_right_rgb[7:4];
                        ghost3_b = ghost_right_rgb[3:0];
                    end
                    else if (ghost3_x > yoshi_x)
                    begin
                        ghost3_r = ghost_left_rgb[11:8];
                        ghost3_g = ghost_left_rgb[7:4];
                        ghost3_b = ghost_left_rgb[3:0];
                    end
                end
                else
                begin
                    ghost3_r = 4'd0;
                    ghost3_g = 4'd0;
                    ghost3_b = 4'd0;
                end
            end
            else
            begin
                ghost3_r = 4'd0;
                ghost3_g = 4'd0;
                ghost3_b = 4'd0;
            end


            // If ghost 4 is enabled
            if (ghost4)
            begin
                if (draw_ghost4_x & draw_ghost4_y)
                begin
                    if (ghost4_x < yoshi_x)
                    begin
                        ghost4_r = ghost_right_rgb[11:8];
                        ghost4_g = ghost_right_rgb[7:4];
                        ghost4_b = ghost_right_rgb[3:0];
                    end
                    else if (ghost4_x > yoshi_x)
                    begin
                        ghost4_r = ghost_left_rgb[11:8];
                        ghost4_g = ghost_left_rgb[7:4];
                        ghost4_b = ghost_left_rgb[3:0];
                    end
                end
                else
                begin
                    ghost4_r = 4'd0;
                    ghost4_g = 4'd0;
                    ghost4_b = 4'd0;
                end
            end
            else
```

```
            begin
                ghost4_r = 4'd0;
                ghost4_g = 4'd0;
                ghost4_b = 4'd0;
            end
        end // always @ * block end



        // Eggs/Score objects drawing
        parameter EGG_WIDTH = 6'd32;
        parameter EGG_HEIGHT = 6'd36;

        // -1 pixel because we start indexing for sprites from 0
        assign draw_egg_x = (egg_x <= draw_x) & (draw_x <= egg_x + (EGG_WIDTH-1'b1));
        assign draw_egg_y = (egg_y <= draw_y) & (draw_y <= egg_y + (EGG_HEIGHT-1'b1));

        // Sprite image memory instantiation, address, rgb,and tracker. sprite is 32x36
        reg [5:0] egg_x_os = 6'd0, egg_y_os = 6'd0;
        reg [10:0] egg_mem_addr;
        wire [11:0] egg_rgb;
        egg_mem egg_sprite (.clka(clk),    // input wire clka
                            .addra(egg_mem_addr),  // input wire [10 : 0] addra
                            .douta(egg_rgb)  // output wire [11 : 0] douta
        );


        // Synchronous block to address the egg memory to fetch the rgb colours
        always @ (posedge clk)
        begin
            // reset trackers
            if (draw_x == egg_x & draw_y == egg_y)
            begin
                egg_x_os <= 6'd0;
                egg_y_os <= 6'd0;
            end
            if (draw_egg_x & draw_egg_y)
            begin
                egg_mem_addr <= EGG_WIDTH * egg_y_os + egg_x_os;

                if (egg_x_os == 5'd31)
                begin
                    egg_x_os <= 6'd0; // reset x offset tracker

                    // reset y when done drawing a tile
                    if (egg_y_os == 6'd35) // height is 36 pixels
                        egg_y_os <= 6'd0;
                    else
                        egg_y_os <= egg_y_os + 1'b1; // inc y
                end
                else
                    egg_x_os <= egg_x_os + 1'b1;
            end
        end

```

```
    reg [3:0] egg_r = 4'd0, egg_g = 4'd0, egg_b = 4'd0;
    always @ *
    begin
        if (draw_egg_x & draw_egg_y)
        begin
            egg_r = egg_rgb[11:8];
            egg_g = egg_rgb[7:4];
            egg_b = egg_rgb[3:0];
        end
        else
        begin
            egg_r = 4'd0;
            egg_g = 4'd0;
            egg_b = 4'd0;
        end
    end


    // Drawing platforms, using pl as an abbreviation for platform
    // Platform 1 drawing -> lower center of the screen
    parameter pl1_yrange1 = 10'd609;
    parameter pl1_yrange2 = 10'd640;
    assign pl1_y = (pl1_yrange1 <= draw_y) & (draw_y <= pl1_yrange2);
    // Platform 4 drawing -> high center of the screen
    parameter pl4_yrange1 = 9'd249;
    parameter pl4_yrange2 = 9'd280;
    assign pl4_y = (pl4_yrange1 <= draw_y) & (draw_y <= pl4_yrange2);
    // Shared between platforms 1 and 4
    parameter pl1and4_xrange1 = 9'd273;
    parameter pl1and4_xrange2 = 10'd1008;
    assign pl1and4_x = (pl1and4_xrange1 <= draw_x) & (draw_x <= pl1and4_xrange2);

    // Platform 2 drawing -> second lowest left of the screen
    parameter pl2_xrange1 = 6'd32;
    parameter pl2_xrange2 = 10'd511;
    assign pl2_x = (pl2_xrange1 <= draw_x) & (draw_x <= pl2_xrange2);
    // Platform 3 drawing -> second lowest right of the screen
    parameter pl3_xrange1 = 10'd768;
    parameter pl3_xrange2 = 11'd1247;
    assign pl3_x = (pl3_xrange1 <= draw_x) & (draw_x <= pl3_xrange2);
    // Shared between platforms 2 and 3
    parameter pl2and3_yrange1 = 9'd429;
    parameter pl2and3_yrange2 = 9'd460;
    assign pl2and3_y = (pl2and3_yrange1 <= draw_y) & (draw_y <= pl2and3_yrange2);

    // Sprite image for platforms is 32x32
    parameter PLATFORM_WIDTH = 6'd32;
    // Platforms memory instantiation, stores the platform rgb values
    reg [5:0] centerpl_x_os = 6'd0, centerpl_y_os = 6'd0;
    reg [5:0] leftpl_x_os = 6'd0, leftpl_y_os = 6'd0;
    reg [5:0] rightpl_x_os = 6'd0, rightpl_y_os = 6'd0;
    reg [9:0] pl1and4_row_count = 10'd0, pl2_row_count = 10'd0, pl3_row_count = 10'd0;
    reg [9:0] pl_mem_addr;
    wire [11:0] pl_rgb;
    platforms_mem platforms_sprite (.clka(clk),    // input wire clka
                                    .addra(pl_mem_addr),  // input wire [9 : 0] addra
```

```
                                        .douta(pl_rgb)  // output wire [11 : 0] douta
        );

        // Synchronous block to address the platforms memory
        always @ (posedge clk)
        begin
            if ((pl1and4_x & pl1_y) | (pl1and4_x & pl4_y))
            begin
                pl_mem_addr <= PLATFORM_WIDTH * centerpl_y_os + centerpl_x_os;

                if (centerpl_x_os == 5'd31)
                    centerpl_x_os <= 6'd0; // reset offset tracker
                else
                    centerpl_x_os <= centerpl_x_os + 1'b1; // increment x offset tracker

                // Only increment y when we finished drawing the entire row
                if (pl1and4_row_count == 11'd735) // we are drawing 23 sprites consecutively, width of pl is 736 pixels
                begin
                    pl1and4_row_count = 11'd0;

                    if (centerpl_y_os == 5'd31)
                        centerpl_y_os <= 6'd0; // reset offset tracker
                    else
                        centerpl_y_os <= centerpl_y_os + 1'b1;
                end
                else
                    pl1and4_row_count <= pl1and4_row_count + 1'b1; // increment row counter
            end
            else if (pl2_x & pl2and3_y)
            begin
                pl_mem_addr <= PLATFORM_WIDTH * leftpl_y_os + leftpl_x_os;

                if (leftpl_x_os == 5'd31)
                    leftpl_x_os <= 6'd0; // reset offset tracker
                else
                    leftpl_x_os <= leftpl_x_os + 1'b1; // increment x offset tracker

                // Only increment y when we finished drawing the entire row
                if (pl2_row_count == 11'd479) // we are drawing 15 sprites consecutively, width of pl2 is 480 pixels
                begin
                    pl2_row_count = 11'd0;

                    if (leftpl_y_os == 5'd31)
                        leftpl_y_os <= 6'd0; // reset offset tracker
                    else
                        leftpl_y_os <= leftpl_y_os + 1'b1;
                end
                else
                    pl2_row_count <= pl2_row_count + 1'b1; // increment row counter
            end
```

```
            else if (pl3_x & pl2and3_y)
            begin
                pl_mem_addr <= WALL_WIDTH * rightpl_y_os + rightpl_x_os;

                if (rightpl_x_os == 5'd31)
                    rightpl_x_os <= 6'd0; // reset offset tracker
                else
                    rightpl_x_os <= rightpl_x_os + 1'b1; // increment x offset tracker

                // Only increment y when we finished drawing the entire row
                if (pl3_row_count == 11'd479) // we are drawing 15 sprites consecutively, width of pl3 is 480 pixels
                begin
                    pl3_row_count = 11'd0;

                    if (rightpl_y_os == 5'd31)
                        rightpl_y_os <= 6'd0; // reset offset tracker
                    else
                        rightpl_y_os <= rightpl_y_os + 1'b1;
                end
                else
                    pl3_row_count <= pl3_row_count + 1'b1; // increment row counter
            end
        end

        reg [3:0] pl_r = 4'd0, pl_g = 4'd0, pl_b = 4'd0;
        always @ *
        begin
            if ((pl1and4_x & pl1_y) | (pl2_x & pl2and3_y) | (pl3_x & pl2and3_y) | (pl1and4_x & pl4_y)) // all platforms rgb
            begin
                pl_r = pl_rgb[11:8];
                pl_g = pl_rgb[7:4];
                pl_b = pl_rgb[3:0];
            end
            else
            begin
                pl_r = 4'b0;
                pl_g = 4'b0;
                pl_b = 4'b0;
            end
        end

        // Decide between background and foreground colour
        always @ *
        begin
            // Assign colours to draw for the character/block
            // if-else chain asserts priority
            if (game_over)
            begin
                r = 4'b0011;
                g = 4'b0000;
                b = 4'b0110;
            end
            else if ((ghost1_r != 4'd0) & (ghost1_g != 4'd0) & (ghost1_b != 4'd0)) // ghost 1
            begin
                r = ghost1_r;
```

```
                g = ghost1_g;
                b = ghost1_b;
            end
            else if ((ghost2_r != 4'd0) & (ghost2_g != 4'd0) & (ghost2_b != 4'd0)) // ghost 2
            begin
                r = ghost2_r;
                g = ghost2_g;
                b = ghost2_b;
            end
            else if ((ghost3_r != 4'd0) & (ghost3_g != 4'd0) & (ghost3_b != 4'd0)) // ghost 3
            begin
                r = ghost3_r;
                g = ghost3_g;
                b = ghost3_b;
            end
            else if ((ghost4_r != 4'd0) & (ghost4_g != 4'd0) & (ghost4_b != 4'd0)) // ghost 4
            begin
                r = ghost4_r;
                g = ghost4_g;
                b = ghost4_b;
            end
            else if ((yoshi_r != 4'd0) | (yoshi_g != 4'd0) | (yoshi_b != 4'd0)) //  character/block
            begin
                r = yoshi_r;
                g = yoshi_g;
                b = yoshi_b;
            end
            else if ((egg_r != 4'd0) & (egg_g != 4'd0) & (egg_b != 4'd0)) // eggs
            begin
                r = egg_r;
                g = egg_g;
                b = egg_b;
            end
            else if ((pl_r != 4'd0) & (pl_g != 4'd0) & (pl_b != 4'd0)) // platforms
            begin
                r = pl_r;
                g = pl_g;
                b = pl_b;
            end
            else // background
            begin
                r = bg_r;
                g = bg_g;
                b = bg_b;
            end
        end
    endmodule
```

A.7 Multidigit Source File

This source file contains the code for displaying the digits on the seven segment display, this is the code provided to us in exercise 2.

```
module multidigit(
        input clk, rst,
        input [3:0] dig7, dig6, dig5, dig4, dig3, dig2, dig1, dig0,
        output div_clk,
        output a, b, c, d, e, f, g,
        output reg [7:0] an
    );

    wire led_clk;

    reg [3:0] dig_sel;
    reg [2:0] led_index = 3'd0;

    reg [28:0] clk_count = 11'd0;

    always @(posedge clk)
        clk_count <= clk_count + 1'b1;

    assign led_clk = clk_count[16];
    assign div_clk = clk_count[25];

    always@(posedge led_clk)
        if(rst)
            led_index <= 3'd0;
        else
            led_index <= led_index + 1'b1;

    always@*
    begin
        an = 8'b11111111;
        an[led_index] = 1'b0;
        case (led_index)
            3'd0: dig_sel = dig0;
            3'd1: dig_sel = dig1;
            3'd2: dig_sel = dig2;
            3'd3: dig_sel = dig3;
            3'd4: dig_sel = dig4;
            3'd5: dig_sel = dig5;
            3'd6: dig_sel = dig6;
            3'd7: dig_sel = dig7;
        endcase
    end

    sevenseg M1
 (.num(dig_sel), .a(a), .b(b), .c(c), .d(d), .e(e), .f(f), .g(g));

endmodule
```

A.8 Sevenseg Source File

This source file contains the code that implements the seven segments decoder, this is the code provided to us in exercise 2.

```
`timescale 1ns / 1ps

module sevenseg(
    input [3:0] num,
    output a,
    output b,
    output c,
    output d,
    output e,
    output f,
    output g
    );

    reg [6:0] intseg;
    assign {a,b,c,d,e,f,g} = ~intseg;

    always@*
    begin
        case(num)
            4'h0: intseg = 7'b1111110;
            4'h1: intseg = 7'b0110000;
            4'h2: intseg = 7'b1101101;
            4'h3: intseg = 7'b1111001;
            4'h4: intseg = 7'b0110011;
            4'h5: intseg = 7'b1011011;
            4'h6: intseg = 7'b1011111;
            4'h7: intseg = 7'b1110000;
            4'h8: intseg = 7'b1111111;
            4'h9: intseg = 7'b1111011;
            4'ha: intseg = 7'b1110111;
            4'hb: intseg = 7'b0011111;
            4'hc: intseg = 7'b1001110;
            4'hd: intseg = 7'b0111101;
            4'he: intseg = 7'b1001111;
            4'hf: intseg = 7'b1000111;
        endcase
    end


endmodule
```

Appendix B
Bitbucket and Dropbox Links

The project source files can be found on bitbucket on the following link:
https://bitbucket.org/mohshammasi/class-project/src/master/

Over the course of the project I recorded videos of each iteration, sometimes bugs that occurred, these are uploaded and can be found at the following dropbox link:
https://www.dropbox.com/sh/50zko3acd4y3bur/AAAuFaNNmcYOtVoF8aBZHRt4a?dl=0