Flappy Ghost Report

26th December 2021

Abstract

In this assignment, Group E: NightHawk Star chose Flappy Ghost as the project. This game is played by pressing the button, controlling the ghost up and down to dodge the pillar and keep moving forward.

Contents

1 VGA display principle

For monitor, the three RGB signals are analogue signals whose voltage levels indicate the depth of colour. Using this principle, rich colours could be produced. For example, if R = 3.3V, G = 0V, and B = 0V, the screen will show a very bright red color. If G and B are still 0V, and R is changed to 1.8V, the screen will show a lighter red colour. any combination of R, G, and B, ranging from 0 to 3.3V, can represent a very wide range of colours. The electron beam moves regularly from left to right and from top to bottom on the CRT screen. The electron beam starts from a point at the upper left corner of the screen and scans to the right, forming a horizontal line; after reaching the rightmost end, it returns to the left end of the next horizontal line and repeats the above process. The colours are displayed pixel by pixel, the timing of the pixel changes is so fast that the human eye mistakenly believes that all pixels are displayed together. Therefore, this is used to display the pictures in the game. The pictures are converted to .coe file which can be read by Verilog and it presents the colour of each pixel with Binary, decimal or hexadecimal.

colour	black	blue	red	purple	green	cyan	yellow	white
R	0	0	1	1	0	0	1	1
G	0	0	0	0	1	1	1	1
B	0	1	0	1	0	1	0	1

[2]

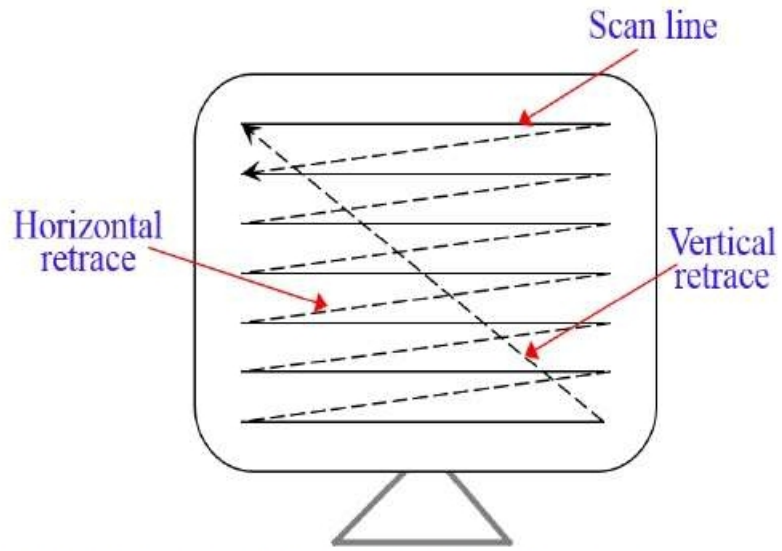

Figure 1: Illustration of raster scanning

[3]

2 Code discription

2.1 Genral

For this pogram, all bird should be ghost. In the beginning, we choose bird as the main character, so is unsuitable for us to change so many parameter's name after. This program contains 10 design sources, game_top.v control all of code and interact with the board. vga_out.v will take control of VGA displayment. It gets input RGB data from game_top, output current x and y coordinate of the screen to other modules, and RGB hsync vsync to VGA output. Each code that will output an image on VGA broad will have a very similar input and output port. They take in the current coordinate from vga_out, output RGB colour and weather is in region. Including bird.v and pillar.v, there x and y is changing, and the code will determine and update display area. Then display.v will accept each displayment moudule's isxxx(isbird, istop, isground...) and 3 bits state (ready, playing, ending) from game_top, determine which RGB data from which file should be used now and pass a number. game_top.v will use a switch case statement to determine and pass RGB data from correct module to vga_out. vga_top also accept signal from fail.v to switch between 3 cases. light.v take the state and switch condition to light up LED. score.v take output score from pillar.v, using 7 segment to display the score. Also, there is ranNum.v under pillar.v that generates random numbers to show the pillar.

2.2 Master control (game_top.v)

It will take the responsibility for changing state, passing correct RGB to vga_out, interacting with board. This game will contain 3 states, player will at the ready state in the biginning, using button to start the game. When the game end (by the signal from fail.v), it switches to the ending state.

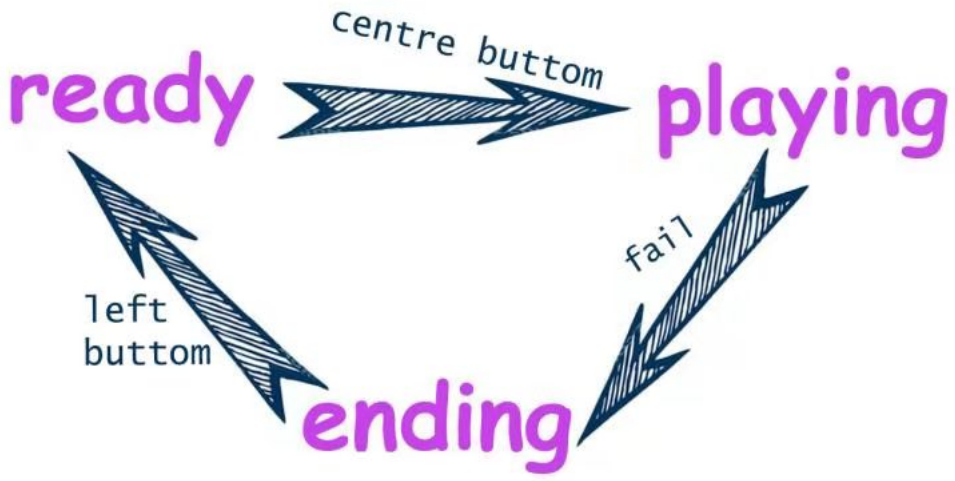

Figure 2: stages changing

The state will affect the ready and ending picture displayed. game_top.v won't judge which image should be displayed here, it will all be done by display.v. It takes a number from display.v, get the mode and switch between each module's output draw_r, draw_g, draw_b. Below is a small part of the codes:

```
always@(posedge pixclk) begin
case(display)
    5'd0 : begin     // default black
        draw_r <= 4'h0;
        draw_g <= 4'h0;
        draw_b <= 4'h0;
    end
    5'd1 : begin     // info bar
        draw_r <= topr;
        draw_g <= topg;
        draw_b <= topb;
    end
    5'd2: begin // left bar
        draw_r <= leftr;
        draw_g <= leftg;
        draw_b <= leftb;
    end
    5'd3: begin // right bar
        draw_r <= rightr;
        draw_g <= rightg;
        draw_b <= rightb;
```

It will generate a slower clock called pixclk use as pixel change. And also frame clock and frame counter, which count for 60 frames per second, are used to provide graph changing.

2.3 Light control (light.v)

It take the input from game_top.v. Including state, and 2 switches. When switch is on, LED above switch is light up to show the switch is on. And for ready two colorful LED will be blue, playing is green, ending is red to show the state of playing to the players.

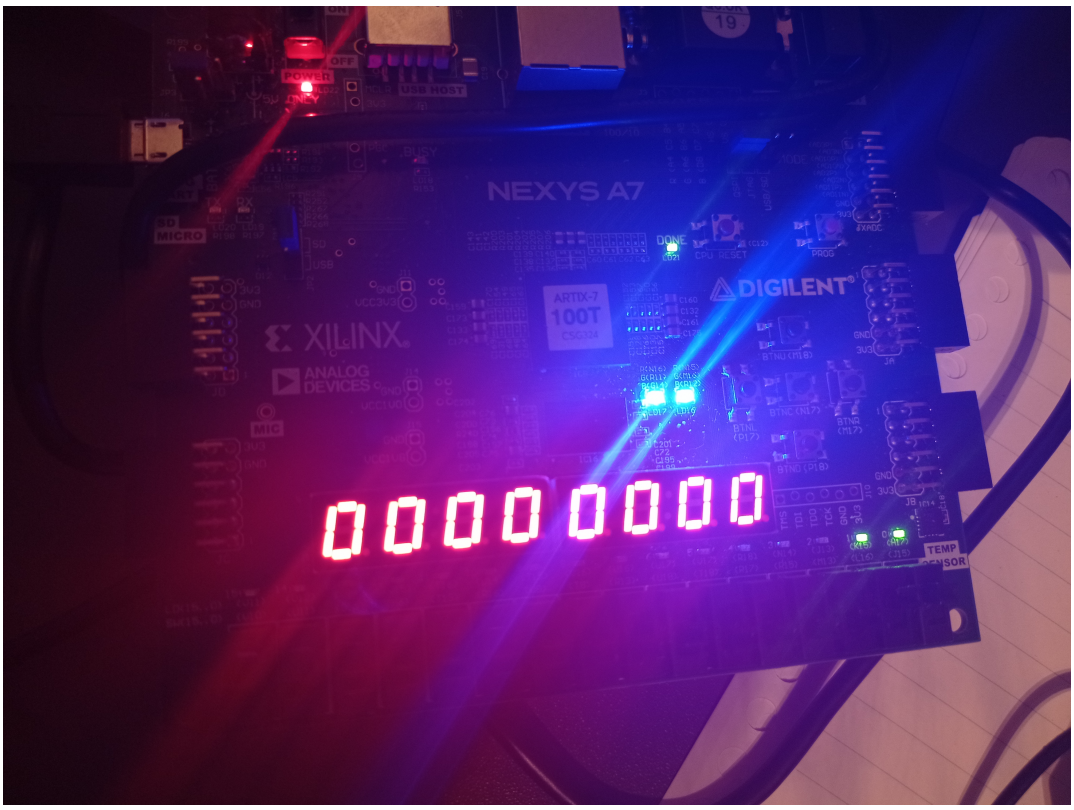

Figure 3: ready

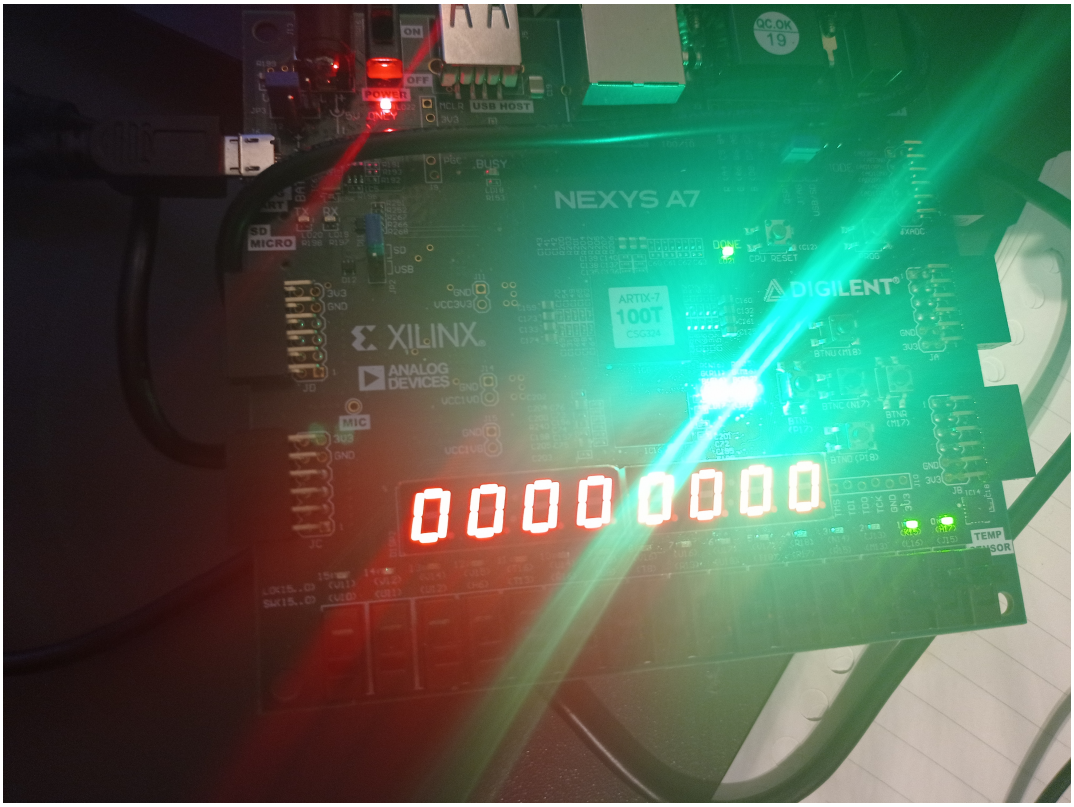

Figure 4: playing

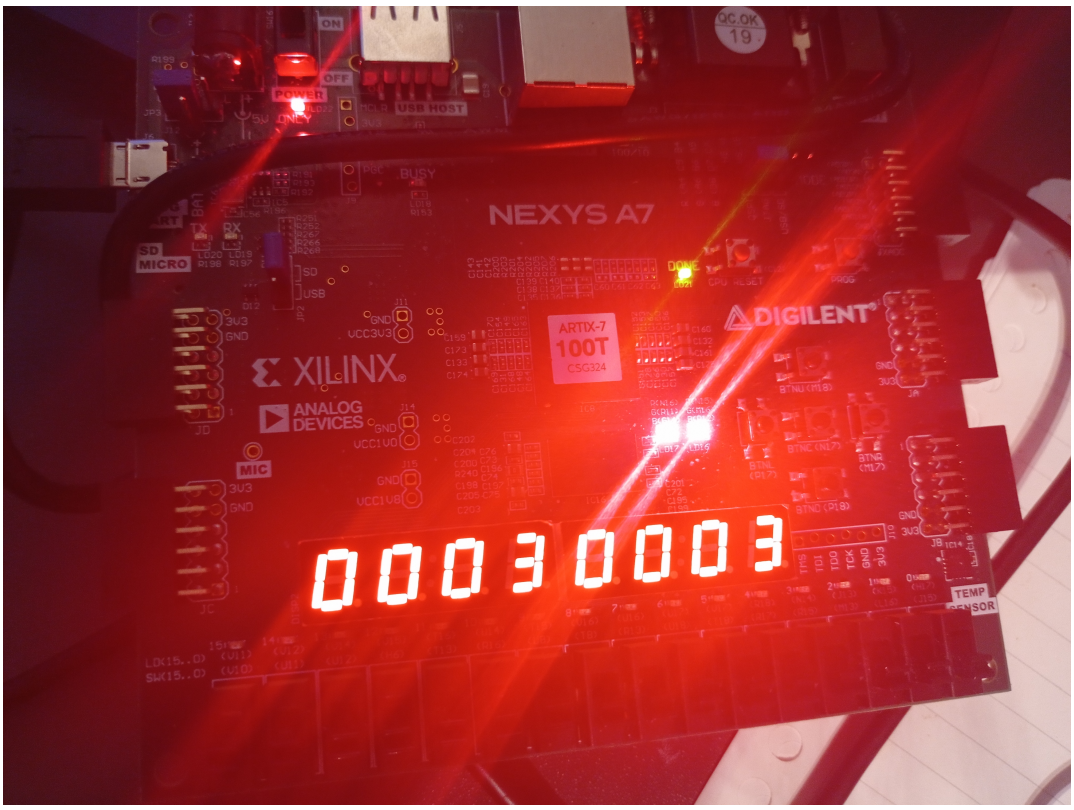

Figure 5: ending

2.4 VGA output (vga_out.v)

This part is the extension of lab 5, the principle of VGA output is described above. It will take RGB input and give output to VGA. It will also calculate current x and y and transfer them to game_top. When the pix is in the display area, current x will increase, else it will be set to 0. Almost the same for y, if it is in the display area and gets line_end signal, y will increase. If out of range, set y into 0.

```
// curr_x
always@(posedge clk) begin
if((hcount >= 11'd384) && (hcount<= 11'd1823)) begin
    curr_x <= curr_x + 1;
end else begin
    curr_x <= 11'd0;
end
end

// curr_y
always@(posedge clk) begin
if((vcount >= 10'd31) && (vcount <= 10'd930)) begin
    if(line_end) curr_y <= curr_y + 11'd1;
    else curr_y <= curr_y;
end
else curr_y <= 11'd0;
end
```

2.5 Stable single original image displayment (bottom.v)

For all of the images, we use python code to convert each pixel's RGB value into string and store in .coe file (see appendix). Firstly, block memory generator IP will be used (in single port ROM state). So it will be a module like implement. The top module should transfer address, that is which line of the data should be read, and get ouptu of data. Now convert this data into RGB array, a image can be shown. The data calculation is relate to the starting point of the picture

that is up left corner. Module will take current x and y as input, if it is in the display region, switch isxxx to 1, calculate and pass picture address to picture IP. The calculation of picture address is (position_y - starpoint_y)* picture width + (position_x - startpoint_x - 1), because of 1 cycle delay. Below is an example:

```
‘timescale 1ns / 1ps
module bottom(
    input clk,
    input [10:0] pos_y,
    input [10:0] pos_x,
    // color of ground
    output reg [3:0] groundr,
    output reg [3:0] groundg,
    output reg [3:0] groundb,
    // if is in ground range
    output reg isground
    );


// image def
    reg [17:0] pic_addr;
    wire [11:0] pic_color;

    always@(posedge clk) begin
        if((pos_y < 900) && (pos_y >= 750) && (350 <= pos_x) &&
           (1090 > pos_x)) begin
            isground <= 1’b1;
            pic_addr <= ((pos_y - 750)*740 + (pos_x - 350 - 1));
            groundr <= pic_color[11:8];
            groundg <= pic_color[7:4];
            groundb <= pic_color[3:0];
        end
        else begin
            groundr <= 4’h7;
            groundg <= 4’hc;
            groundb <= 4’hc;
            isground <= 1’b0;
            pic_addr <= 1;
        end
    end

ground inst_ground (
  .clka(clk),    // input wire clka
  .addra(pic_addr),  // input wire [16 : 0] addra
  .douta(pic_color)  // output wire [11 : 0] douta
);
endmodule
```

2.6 Stable single converted image displayment (top.v leftbar.v rightbar.v)

The image displayment part is as same as 2.5. Because the memory is very limited in the board, we design some images that only contain 2 colours and produce a black and white version of the picture, using oneBitGen.py to store 0 or 1 instead of 12 bits per pixel. And then use an assign statement to give RGB the correct values. This method saves lots of memory, so we can use many images to make our game look nicer. Here is an example:

```
‘timescale 1ns / 1ps

module top(
    input clk,
```

```
    input [5:0] count,
    input [10:0] pos_x,
    input [10:0] pos_y,
    // color of top
    output reg [3:0] topr,
    output reg [3:0] topg,
    output reg [3:0] topb,
    // if is in top range
    output reg istop
    );

    reg [17:0] pic_addr;
    wire [11:0] pic_color;

    // 1 is orange, 0 purple

    always@(posedge clk) begin
        if((pos_x >= 0) && (pos_x < 1440) && (pos_y >= 0) && (
            pos_y < 100)) begin
             istop <= 1'b1;
             pic_addr <= (pos_y * 1440 + pos_x - 1);
             topr <= pic_color[11:8];
             topg <= pic_color[7:4];
             topb <= pic_color[3:0];
        end
        else begin
             topr <= 4'h7;
             topg <= 4'hc;
             topb <= 4'hc;
             istop <= 1'b0;
             pic_addr <= 18'd1;
        end
    end

    assign pic_color = (out)?12'hf73:12'h213;

info img (
  .clka(clk),     // input wire clka
  .addra(pic_addr),  // input wire [17 : 0] addra
  .douta(out)  // output wire [0 : 0] douta
);

endmodule
```

This method help us save lots of memory, so we can display many pictures to make out game look nicer. Below is the ready and ending stage of game:

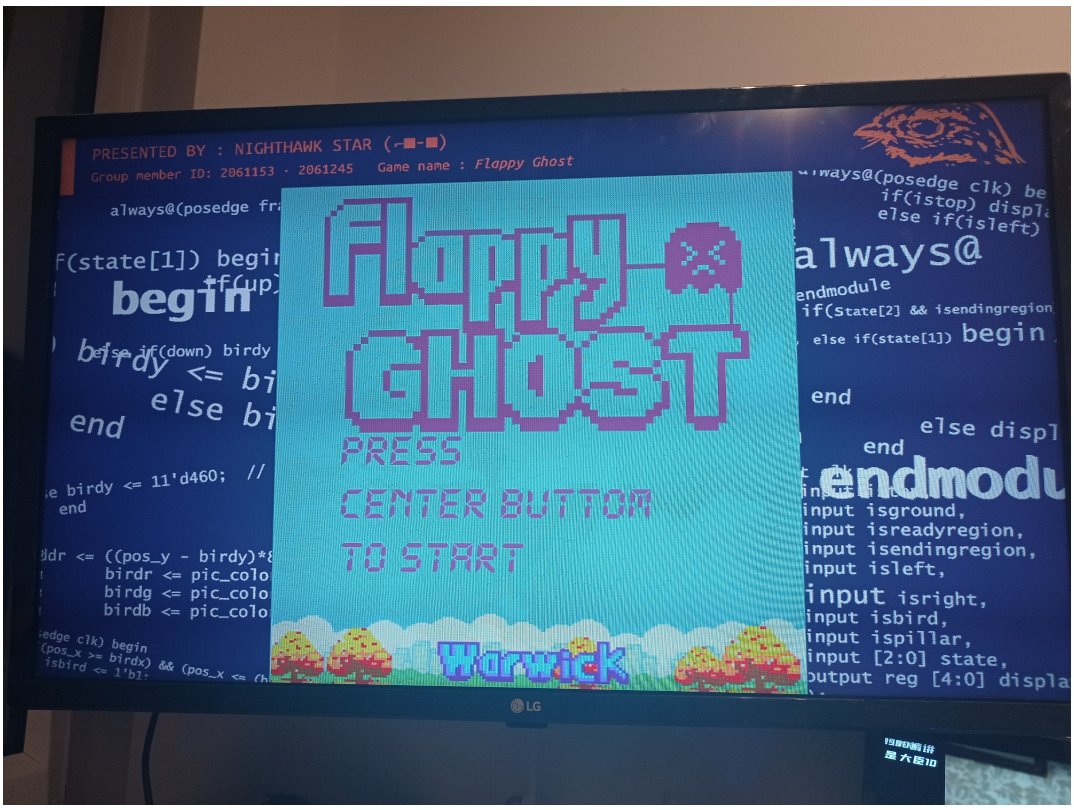

Figure 6: ready stage

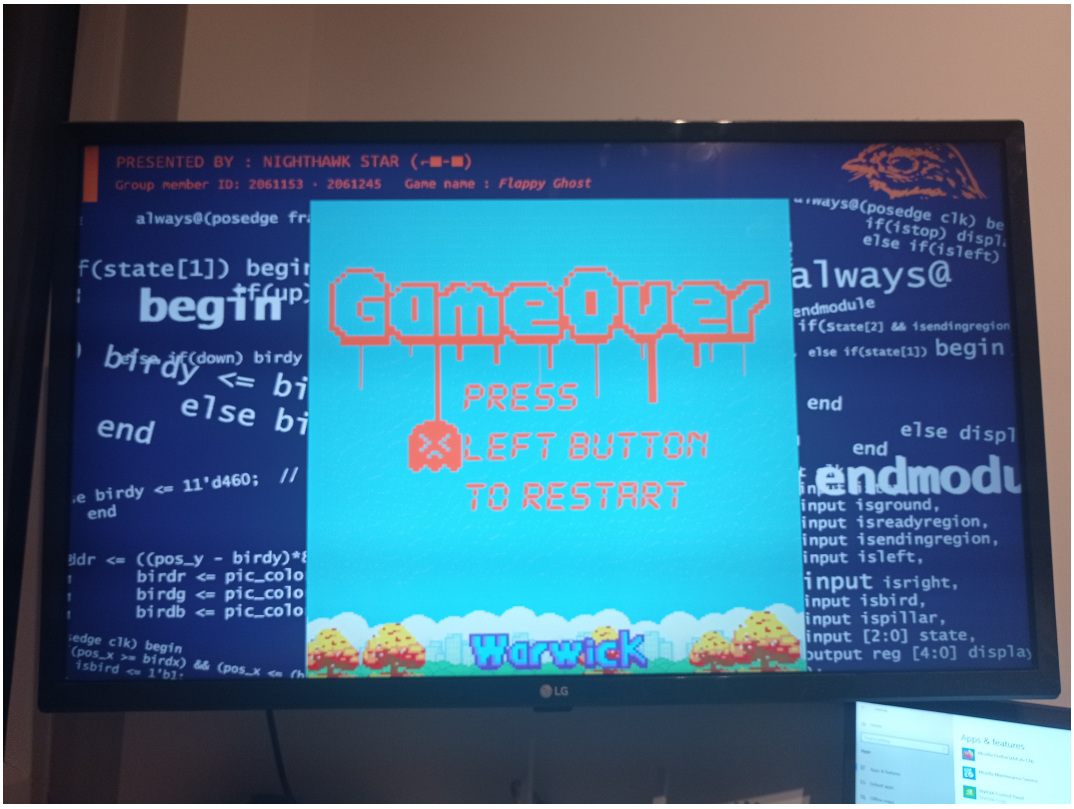

Figure 7: ending stage

2.7 Changing single converted image displayment (ready.v ending.v)

These two image displayment modules extend 2.6, in addition, take frame count as input. So one of the colours (font colour actually) can be changed by time. We use an always statement

to define font colour based on count(framecount) variables:

```
`timescale 1ns / 1ps

module ready(
    input clk,
    input [10:0] pos_x,
    input [10:0] pos_y,
    input [5:0] count,
    output reg [3:0] readyr,
    output reg [3:0] readyg,
    output reg [3:0] readyb,
    output reg isreadyregion
    );

//    for image handling
    reg [18:0] pic_addr;
    wire [11:0] pic_color;
    reg [11:0] font_color;


    always@(posedge clk) begin
        if((pos_y >= 100) && (pos_y < 750) && (350 <= pos_x) &&
           (1090 > pos_x)) begin   // if pix in ready pic region
            isreadyregion <= 1'b1;
            pic_addr <= ((pos_y - 100)*740 + (pos_x - 350 -1));
            readyr <= pic_color[11:8];
            readyg <= pic_color[7:4];
            readyb <= pic_color[3:0];
        end
        else begin  // outside region
            readyr <= 4'h7;
            readyg <= 4'hc;
            readyb <= 4'hc;
            isreadyregion <= 1'b0;
        end
    end

     always@(posedge clk) begin
        if(count <= 30) font_color <= 12'h836;
        else font_color <= 12'hf53;
    end

    assign pic_color = (out)?12'h7cc:font_color;

startPic startpic (
  .clka(clk),    // input wire clka
  .addra(pic_addr),  // input wire [18 : 0] addra
  .douta(out)  // output wire [0 : 0] douta
);

endmodule
```

These two puictures are changing font colour of ready stage.

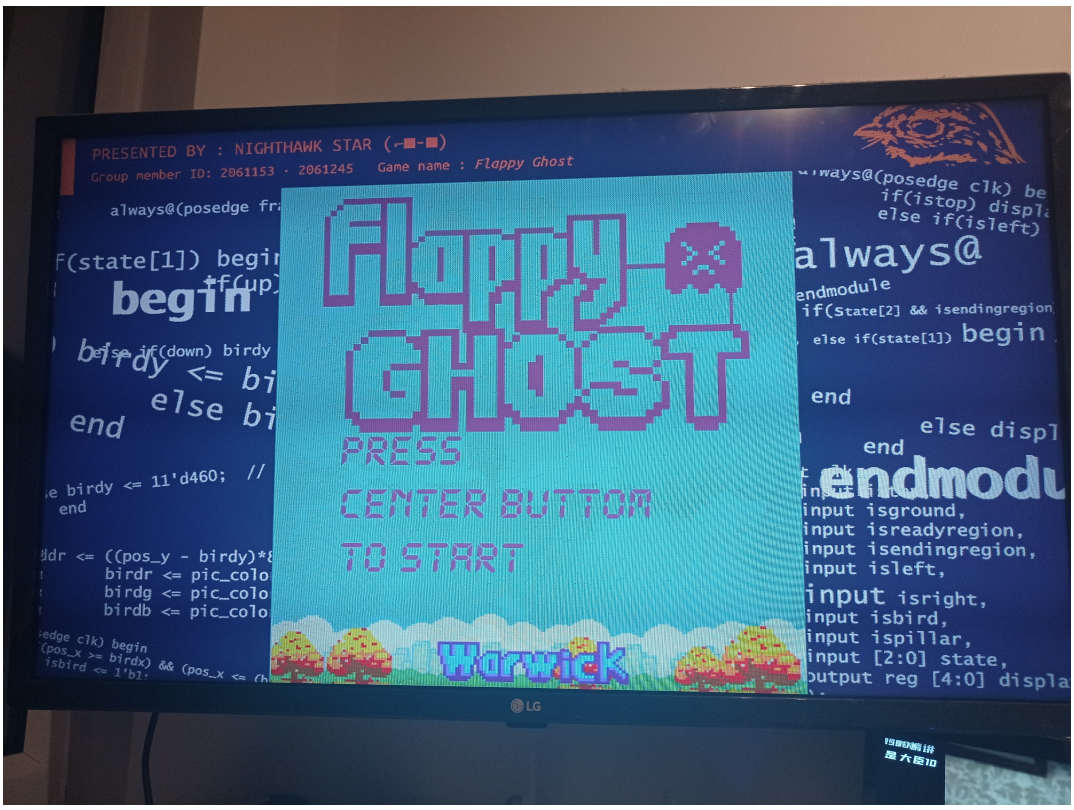

Figure 8: purple font

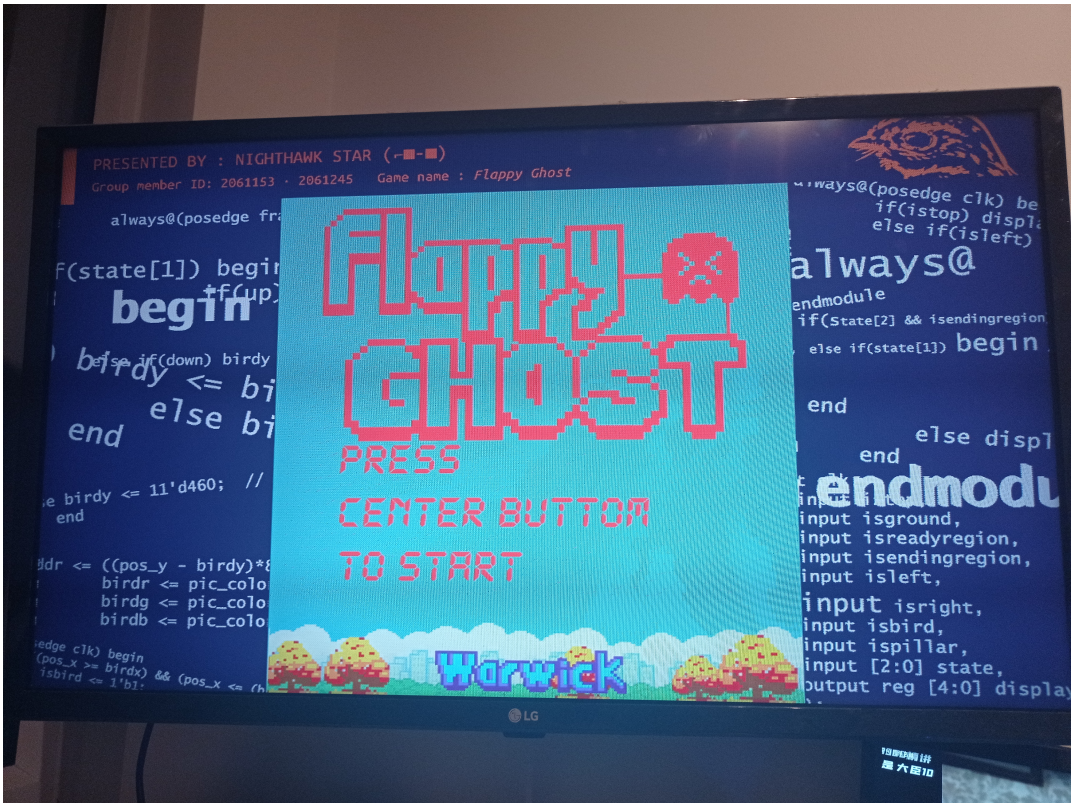

Figure 9: orange font

2.8 Main character control (bird.v)

The ghost is also a picture and implementation is the same as 2.7. We prepare two pictures and will switch between them base on time changing. This module will get the input of two buttons

up and down to control the y position of the ghost, so the display area is varying by time, which will be calculated by codes. It also gets one switch as cheat code, originally ghost will fall if there is no input, but if the cheat switch is on, ghost will stay still. The module will also take state input, if not at playing state the bird will reset to starting position.

```
`timescale 1ns / 1ps
module bird(
    input clk,
    input frameclk,
    input up,
    input down,
    input [2:0] state,
    input [5:0] count,
    input [10:0] pos_x,
    input [10:0] pos_y,
    input sw0,
    output reg isbird,
    output reg [3:0] birdr,
    output reg [3:0] birdg,
    output reg [3:0] birdb,
    output reg [10:0] birdx,
    output reg [10:0] birdy
    );

    // for image handling
    reg [17:0] pic_addr;
    reg [11:0] pic_color;
    wire [11:0] pic1;
    wire [11:0] pic2;

    // init position of bird, left top corner

    initial begin
        birdx = 420;
        birdy = 460;
    end

    always@(posedge frameclk) begin // det bird moving
        if(state[1]) begin  // if is playing
            if(up) birdy <= birdy - 11'd8;
            else if(down) birdy <= birdy + 11'd8;
            else if(sw0) birdy <= birdy;    // cheat code, prevent
                bird falling
            else birdy <= birdy + 11'd3;
        end
        else birdy <= 11'd460;  // reset position
    end

    always@(posedge clk) begin
        if((pos_x >= birdx) && (pos_x <= (birdx+84)) && (pos_y >=
           birdy) && (pos_y <= (birdy+90))) begin
            isbird <= 1'b1;
            pic_addr <= ((pos_y - birdy)*84 + (pos_x - birdx - 1))
               ;
            birdr <= pic_color[11:8];
            birdg <= pic_color[7:4];
            birdb <= pic_color[3:0];
         end
        else begin
            isbird <= 1'b0;
            birdr <= 4'h7;
```

```
            birdg <= 4'hc;
            birdb <= 4'hc;
            pic_addr <= 1;
        end
    end

    always@(posedge clk) begin
    if (count <= 30) pic_color <= pic1;
    else pic_color <= pic2;
    end


ghost1 ghost1 (
  .clka(clk),     // input wire clka
  .addra(pic_addr),  // input wire [12 : 0] addra
  .douta(pic1)  // output wire [11 : 0] douta
);

ghost2 ghost2 (
  .clka(clk),     // input wire clka
  .addra(pic_addr),  // input wire [12 : 0] addra
  .douta(pic2)  // output wire [11 : 0] douta
);

endmodule
```

2.9 Random number generartion (ran.v)

This part of code is a very rough random number generator. Initially, it declares rand arrays for all 1s and prepares another next array with the same size. Then 4 feeds are assigned by xor of two random numbers in rand array. Then concatenate rand and 4 feeds together to generate the next array. When the clock rises, rand will assign by next array. Overall, last 9 bits of rand array will be the random number generated.

```
`timescale 1ns / 1ps

module ran(
    input clk,
    output reg [9:0] out
    );

        reg [20:0] rand;
        initial rand = ~(20'b0);
        reg [20:0] next;
        wire f0, f1, f2, f3;

        assign f0 = rand[20] ^ rand[13];
        assign f1 = rand[2] ^ rand [9];
        assign f2 = rand[0] ^ rand[16];
        assign f3 = rand[5] ^ rand[17];

        always @ (posedge clk)
        begin
                rand <= next;
                out = rand[9:0];
        end

        always@*
        begin
                next = {rand[13:0],f3, f2, rand[4:3], f0, f1};
        end
```

```
endmodule
```

2.10 Pillars generation (pillar.v)

This module will output coordination of two pillars, RGB and is in the region and also score the player get. Two pillars will be kept all the time. Distance between upper and down pillar is 260 pix. So for the detection of isregion, pix y will skip for 260 pix areas left space for ghost passing. Pillars are moving all the time. When pillar passes up left corner of ghost, score + 1, it will move back to another pillar x + 406(406 pix is the distance between two pillars), and set y to a random number that gets from 2.9. The y should be in range of 108 to 632, so we use 108+ran % 524, taking the random of 524.

Figure 10: get acore

```
`timescale 1ns / 1ps

module pillar(
    input sysclk,
    input clk,
    input frameclk,
    input [10:0] pos_x,
    input [10:0] pos_y,
    input [2:0] state,
    input [10:0] birdy,
    output reg ispillar,
    output reg [10:0] p1x,
    output reg [10:0] p1y,
    output reg [10:0] p2x,
    output reg [10:0] p2y,
    output reg [3:0] pillarr,
    output reg [3:0] pillarg,
    output reg [3:0] pillarb,
    output reg [13:0] score
    );
    // totally 2 pillar
    // pillar start with x:1270, y: 400
```

```
// det pillar co
wire [8:0] outy;


always@(posedge frameclk) begin
    if(state[0]) begin   // ready
        p1x <= 11'd1234;
        p1y <= 11'd400;
        p2x <= 11'd1640;
        p2y <= 11'd400;
        score <= 0;
    end
    else if(state[1]) begin // playing
        // moving each pillar
        if(p1x <= 420) begin     // bird pass pillar1
            score <= score + 1;
            p1x <= p2x + 406;
            p1y <= 108 + (outy % 524);
            p2x <= p2x - 5;
            p2y <= p2y;
        end
        else if(p2x <= 420) begin     // bird pass pillar2
            score <= score + 1;
            p2x <= p1x + 406;
            p2y <= 108 + (outy % 524);
            p1x <= p1x - 5;
            p1y <= p1y;
        end
        else begin
            p1x <= p1x - 5;
            p1y <= p1y;
            p2x <= p2x - 5;
            p2y <= p2y;
            score <= score;
        end
    end
    else begin  //end
        score <= score;
        p1x <= 11'd1234;
        p1y <= 11'd400;
        p2x <= 11'd1640;
        p2y <= 11'd400;
    end
end

// output pix info
always@(posedge clk) begin
// pink
    if(((pos_y <= p1y) || (pos_y >= (p1y + 260))) && (pos_x <=
        p1x) && (pos_x >= (p1x - 144))) begin
        pillarr <= 4'hf;
        pillarg <= 4'h0;
        pillarb <= 4'h7;
        ispillar <= 1;
    end
    // brown
    else if(((pos_y <= p2y) || (pos_y >= (p2y + 260))) && (
       pos_x <= p2x) && (pos_x >= (p2x - 144))) begin
        pillarr <= 4'h9;
        pillarg <= 4'h5;
        pillarb <= 4'h0;
```

```
            ispillar <= 1;
        end
        else begin
            pillarr <= 4'h7;
            pillarg <= 4'hc;
            pillarb <= 4'hc;
            ispillar <= 0;
        end
    end

ran ranNum(
    .clk(sysclk),
    .out(outy)
);


endmodule
```

2.11 Failing detection (fail.v)

This module is used to determine whether the game is failed, pass the parameter to game_top to end the game. Firstly it will detect if bird is out of range, if it touches the edge of the region, the game will fail. It also takes coordination of both bird and pillar, and calculating whether these two are overlapped. If the cheat switch is not opened, then the game fails, it can pass isfail to game_top and the state is changed from playing to ending. When the state is not playing, fail.v will automatically reset isfail to 0.

```
wire [8:0] outy;     // get random number output
always@(posedge frameclk) begin
        if(state[0]) begin    // ready
            p1x <= 11'd1234;
            p1y <= 11'd400;
            p2x <= 11'd1640;
            p2y <= 11'd400;
            score <= 0;
        end
        else if(state[1]) begin // playing
            // moving each pillar
            if(p1x <= 420) begin     // bird pass pillar1
                score <= score + 1;
                p1x <= p2x + 406;
                p1y <= 108 + (outy % 524);
                p2x <= p2x - 5;
                p2y <= p2y;
            end
            else if(p2x <= 420) begin     // bird pass pillar2
                score <= score + 1;
                p2x <= p1x + 406;
                p2y <= 108 + (outy % 524);
                p1x <= p1x - 5;
                p1y <= p1y;
            end
            else begin
                p1x <= p1x - 5;
                p1y <= p1y;
                p2x <= p2x - 5;
                p2y <= p2y;
                score <= score;
            end
        end
        else begin  //end
```

```
            score <= score;
            p1x <= 11'd1234;
            p1y <= 11'd400;
            p2x <= 11'd1640;
            p2y <= 11'd400;
        end
    end

    // output pix info
    always@(posedge clk) begin
    // pink
        if(((pos_y <= p1y) || (pos_y >= (p1y + 260))) && (pos_x <=
            p1x) && (pos_x >= (p1x - 144))) begin
            pillarr <= 4'hf;
            pillarg <= 4'h0;
            pillarb <= 4'h7;
            ispillar <= 1;
        end
        // brown
        else if(((pos_y <= p2y) || (pos_y >= (p2y + 260))) && (
            pos_x <= p2x) && (pos_x >= (p2x - 144))) begin
            pillarr <= 4'h9;
            pillarg <= 4'h5;
            pillarb <= 4'h0;
            ispillar <= 1;
        end
        else begin
            pillarr <= 4'h7;
            pillarg <= 4'hc;
            pillarb <= 4'hc;
            ispillar <= 0;
        end
    end

ran ranNum(
    .clk(sysclk),
    .out(outy)
);

endmodule
```

2.12 Score display (score.v)

This module takes input score from pillar, break down to 4 digits and pass it to 7 segment display that done in lab 2.

```
always@* begin
        d1 <= score % 10;
        d2 <= score / 10 % 10;
        d3 <= score / 100 % 10;
        d4 <= score / 1000;
    end
```

These digits are common anode that means only one of them at a time and changing them fast enough so that the persistence of the LEDs makes it seem like they're always on.[1] The score repeat twice on the board, digit 5 to 8 is displayed same digits as 1 to 4. When a digit is light on, pass what the digit should be to SevenSeg.v that light up the correct LED to form the digit. And switch rapidly between all 8 digits. Here is seven-segment display general structure:

Figure 11: Common Anode Circuit Node

[4]

2.13 Image display control (display.v)

This module gets all isregion output from different image display modules and states from game_top module. It will determine which image will be shown now because all of the image modules are continuing output signals. Therefore display.v should tell game_top which of them should be chosen. It uses a number output called mode to let game_top switch between different images.

```
// mode 0 is defalt display black, in case of edge

always@(posedge clk) begin
    if(istop) display <= 5'd1;  // always display
    else if(isleft) display <= 5'd2;    // always has, left
        bar
    else if(isright) display <= 5'd3;   // always has, right
        bar
    else if(!state[1] && isground) display <= 5'd9;   //
        always has, bottom, but at playing is under ghost
    else if(state[0] && isreadyregion) display <= 5'd4;     //
         else if ready state and in ready pic
    else if(state[2] && isendingregion) display <= 5'd5;    //
         else if ready state and in ending pic
    else if(state[1]) begin // when playing
        if(isbird) display <= 5'd8; // bird
        else if(ispillar) display <= 5'd10; // pillar above
            ground
        else if(isground) display <= 5'd6;   // ground should
            be under bird
        else display <= 5'd7;    // playing background
    end
    else display <= 5'd0;   // default black
end
```

3 Testing

We seperate single file to different project and using testbench to test the codes. Here is the example of random number generator 2.9. We make a clock input, flip it in every 1 time delay. Let random number generated output in the terminal:

```
`timescale 1ns / 1ps

module testic(
    );
reg clk;
wire [9:0] out;

initial clk = 0;

always #1 clk = !clk;

ran uut(.clk(clk), .out(out));
initial $monitor("Random number:%d", out);
endmodule
```

And get output at Tcl console like

Random number: 442
Random number: 684
Random number: 822
Random number: 410
Random number: 684
Random number: 790
Random number: 410
Random number: 684

4 Reflection

This project is not perfect, there are something we could not understand. The tops of some columns may be distorted, and we cannot determine whether this is a display problem or something else. Image dispaly will leave black margin on the left hand side. But we do learned a lot from building this project.

4.1 Like

The topic of our project is very creative. As students, just like the little ghosts in the game, rise and fall on the road to knowledge, keep flying forward, never-ending, constantly encountering the pillars and trying to get through them, even if we fail, we will start over and never give up. We chose the ghost of Pac-Man and the background of Flappy Bird as a tribute to the classic game. Through continuous learning and progress, we have been able to make works that are close to them, which is very fulfilling and inspiring.

By adding the clock, it makes the expression of the ghost is constantly switching. This makes the game with better details and better graphics. And the text in the game can also keep changing colour, which also makes the game more colourful.

We not only meet the basic requirements for running the game, but also use seven segment to display the score, and score resets when the ghost hits the pillars. In addition, RGB is also used: it shows different colours when the game is ready, playing and end.

Our code and pictures are all original, we want to create a program that is entirely our own.

Before we started, everyone was assigned a very clear task, so everything was very well organized and efficient during the project.

4.2 Dislike

The memory of the board is very limited, so some of the delicate pictures we created could not be inserted. But we came up with a very good solution: make the picture in black and white, represented by 0 and 1. By reading the values of 0 and 1 and assigning them other colours,

then the colour pictures can be output and the memory is saved at the same time. However, the picture displayed in this way will also be monotonous and simple.

4.3 Improvement

4.3.1 Method of pillars generation

The method of generating pillars can also be improved by making the generation more random. The style of the pillars can also be more complex and detailed, but it cannot be reached because of the lack of memory this time. And we could also try to eliminate the distortion of the pillars

4.3.2 The ghost(movement)

We could try to improve the movement of the ghost, making it be more smooth and less stiff. For example, making it move in a parabolic line, rising or falling along the arc.

4.3.3 Pictures

The pictures of the game could also be improved. We should find a way to create images that take up less space but are more detailed and beautiful(using simple colour or simple lines to draw pictures).

References

[1] Reddit.com. 2021. [online] Available at: <https://www.reddit.com/r/FPGA/comments/qjeez5/nexys_a7100t_7s [Accessed 2 December 2021].

[2] Vafaee, S., 2021. VGA Adapter. [online] Eecg.utoronto.ca. Available at: <https://www.eecg.utoronto.ca/~jayar/ece241_06F/vga/vga-monitors.html> [Accessed 2 December 2021].

[3] Harvey, I., 2007. Illustration of raster scanning. [image] Available at: <https://upload.wikimedia.org/wikipedia/commons/7/72/Raster-scan.svg> [Accessed 2 December 2021].

[4] n.d. Nexys A7 Reference Manual. [image] Available at: <https://digilent.com/reference/_media/reference/programmable-logic/nexys-a7/n4t.png?w=500&tok=d7fb74> [Accessed 2 December 2021].