ES3B2 Digital Systems Design: Assignment Report

INTRODUCTION

This report will discuss the design of an interactive game in Vivado Design Suite (developed by Xilinx) using Nexys4DDR FPGA development board. The project replicates the classic video arcade game: *Frogger*. It will analyse: how the module *VGA_OUT* controls the display, the different modules used to create the game logic: *DRAWCON*, *FROGMOVE*, *SEGINTERFACE*, *SEVENSEG* and *GAMETOP*; and it will conclude with a reflective section.

BACKGROUND DISCUSSION ON DISPLAYS: VGA

Our project's VGA signalling protocol (Appendix 1) is based on [1]. In order to declare the output of the screen we addressed each pixel individually. This is achieved by means of their coordinates in the form of two counters: one recording the horizontal distance (11-bit registers: *hcount* and *reg_x*) and another one for the vertical position (10-bit registers: *vcount* and *reg_y*).

In traditional CRT displays, the pixels' output was determined by a cathode ray being deflected from the top-left corner of the screen to the right. When it reached the end of the top line, it would then continue from the left-most position of the second line, and so on. However, moving from the end of a line to the beginning of the following line required extra time. Hence, extra cycles with no pixel output were added to give the beam time to complete the movement (as shown in Figure1 below) [1]. Our project uses this approach mainly for its didactical purpose. The coordinates of each pixel are determined by means of two '*always blocks*' [2]: one with extra no-output coordinates, and a second one to address exclusively the 'active video' region.

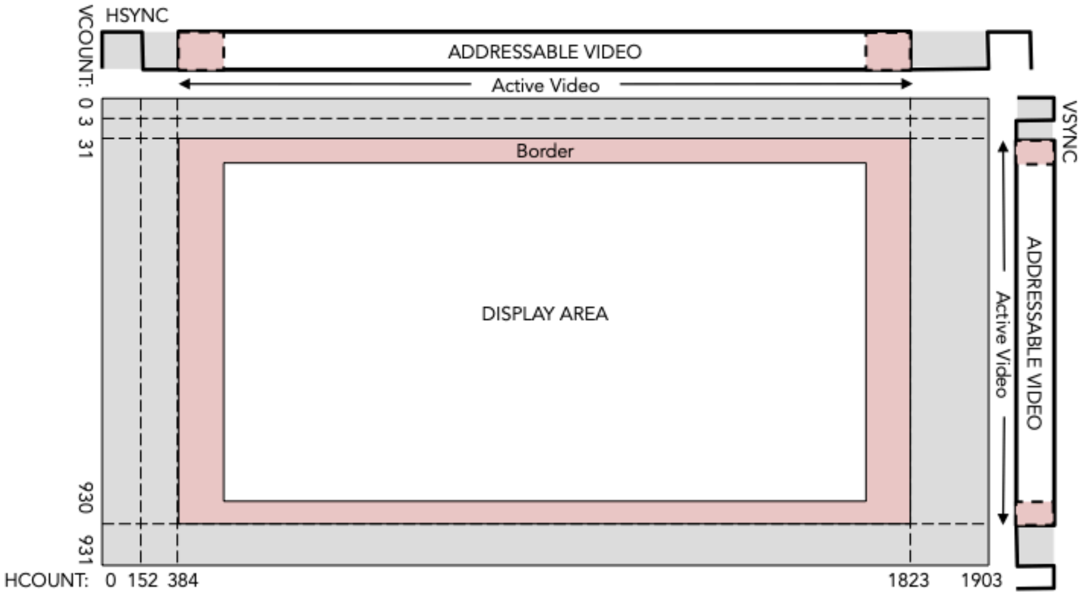

Figure 1: Addressable Pixels Schematic

Both 'always blocks' use an internal clock signal of the FPGA (referred as 'clk'); it has a frequency of 106MHz, ensuring that all pixels are addressed fast enough to draw the screen. The first block creates the entire "video" region. On each clock cycle the horizontal coordinates (*hcount*) will increase by a factor of one until it reaches the horizontal maximum value. At this point the vertical coordinate (*vcount*) will increase by a factor of one, changing the pixel row and restarting the horizontal counter. The second 'always block' is based on the previous counters *hcount* and *vcount*. It creates two new coordinates *reg_x* and *reg_y* for the "addressable video". In our case, this creates a 1439 by 899 pixel screen [1]. These variables indicate the pixel being drawn; thus, they are assigned to the outputs: *curr_x* and *curr_y*, to be used in other modules.

The pixel output uses the RGB colour model, where the colour is the result of the combination of three additive primary colours: red, green and blue, through the declaration of an intensity-value for each [2]. Hence, there are three output values: *pix_r*, *pix*_g and *pix_b*, which will each receive the value of an input variable from our *Drawcon* Module: *draw_r*, *draw_g* and *draw_b*, when they are within the display area and a value of 0 when they are not, thanks to three 'assign' statements [1].

It is important to note that we use two output variables: *hsync* and *vsync*, to generate the correct control pulses to refresh the screen [3]. Given the specific resolution of the screen, *hsync* will be assigned a value of 1 when *hcount* is between 0 and 151 (inclusive) and a value of zero otherwise; similarly, *vsync* will only be one when *vcount* is between 0 and 2 (as shown in Figure1) [1].

DESIGN DESCRIPTION

The game design mimics the classic arcade game: 'Frogger'. It consists on a moving block: the frog, who has to cross a road avoiding the cars and cross a river by jumping on top of moving wooden logs to avoid falling on the water to reach a finish line. The player has a total of 5 opportunities to achieve this goal, which are displayed on the FPGA.

The game graphic design is achieved in the *Drawcon* module. This module takes position data elaborated in *VGA_OUT* and in the *Frogmove* module and sends the pixel output information to the *VGA_OUT* module to build the screen. At the same time, the *Frogmove* module sends the score signals to the top module: *Gametop*, to ensure the correct screen and 'lives score' are displayed. The overall project design is summed in Figure2 below.

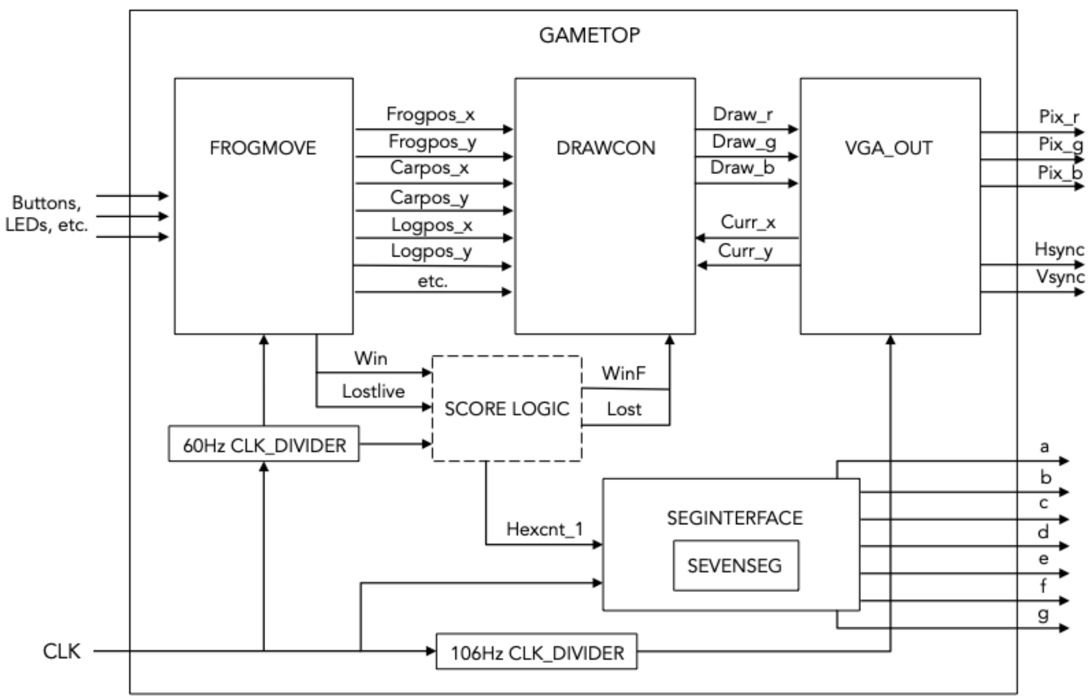

Figure 2: Overall Project Design

A) 'Drawcon' Module (Appendix 2)

As specified in the *VGA_OUT* module, our project uses an area of 1439 pixels width and 899 pixels depth. The graphics can be divided into two sections: the background (which conforms all the static colour blocks) and the moving blocks (which includes the frog, the cars and the logs).

The module starts by specifying the background colour with an 'always block'. Following the RGB colour model, there are three internal background variables: *bg_r*, *bg_g* and *bg_b*. The 'always block's' logic leaves a margin on all borders with a width of 10 pixels, where the player will not be able to enter. This specifies the game 'play' region as the area within the margins. It then continues by dividing the remaining of the display area into sections for the different obstacles of the game (as shown in Figure 3):

Figure 3: Background Display

1. The first section addressed is the river, which consists of a blue horizontal rectangle that fills the whole width of the 'play' region from 100 to 400 pixels depth.
2. Secondly, the road is created. It consists on a grey rectangle comprising the width of the screen from 500 to 800 pixels depth with two dashed white lines imitating road lines, one from 595 to 605 and the other from

695 to 705. Each line is made of individual white rectangles of 10 pixels depth and 48 pixels width repeating at regular intervals of 48 pixels.

3. Thirdly, it draws the finish line area within a depth of 25 to 85 pixels setting a change in the colour pattern at 55.
4. Finally, everything else is given a green colour to recreate grass.

The next stage is to declare the moving objects in the screen. There are a total of thirteen moving blocks: 6 cars, 6 logs and 1 frog (our character). For each object it takes the horizontal and vertical coordinates of the top-right corner of the block as an input value (referred to as: namepos_x and namepos_y). By means of a combinational 'always block', it then determines the width and depth of each object by adding the size values (shown in Table1 below) [4].

Table 1: Moving Blocks' dimensions

OBJECT	Frog	Car	Log (type 1)	Log (type 2)
WIDTH	32	144	144	288
HEIGHT	32	80	80	80

Both car-blocks and log-blocks are set to have a specific uniform colour. On the contrary, the frog-block holds an image (Appendix 3). This is achieved by means of a Block Memory Generator – ROM also known as a "Sprite". This method requires the image to have the same dimensions as the block it will fill (32x32 in the case of the frog-block), and to be a specific format: a COE file. A COE file refers to the Xilinx Coefficient of an image, it stores a set of initialization values for the block specified. The image size can be tailored using several applications (such as Paint) and can be converted into a COE file using a variety of methods (for this project a Matlab file [5] was used (Appendix 4)). Once the COE file is uploaded to a Block Memory Generator, it takes an input address determined by Equation1:

$$\mathrm{addr\_f\_reg} <= \left(\left(\mathrm{draw_y} - \mathrm{frogpos_y}\right) \cdot 6'\mathrm{d}32\right) + \left(\mathrm{draw\_x} - \mathrm{frogpos\_x}\right) \quad (1)$$

And produces a 12-bit output that is then assigned to the 3 colour parameters of the object (digits 1-4 for blue, digits 5-8 for green, and 9-12 for red).

Having declared the pixel output for each element, all objects and the background are combined in an 'always block'. An "object hierarchy" is established to ensure the moving objects appear in the screen. This 'always block' assigns each block's pixel output to 3 internal registers: *draw_r_reg*, *draw_g_reg* and *draw_b_reg*, that will later be assigned to the module output (*draw_r*, *draw_g* and *draw_b*). The logic first addresses the frog-block ensuring it remains traceable, it continues by transmitting the car-blocks and log-blocks to the output variables, and lastly, it sets all other pixels to the corresponding background output, resulting in the blocks moving on top of the background [6].

Finally, another 'always block' controls the screen displayed depending on two signals. These signals are the "Lost" signal and the "Win" signal. The "Lost" signal activates when the player runs out of opportunities (it has used all its lives), setting a red screen to the colour output variables. On the other hand, if it reaches the finish line; that is to say, the player has achieved the purpose of the game; the "Win" signal is triggered displaying a "Win" screen. The win screen uses the Block Memory Generator – ROM (explained above) to display a tailored image (Appendix 3). Nevertheless, while both signals are switched off the game runs normally displaying the different objects.

B) 'Frogmove' Module (Appendix 5)

The module *Drawcon* will build the graphics of the game. However, it needs to receive the position information of every moving block or it will not print them. This information is declared in a separate module: *Frogmove*. The frog-block should be able to move in all directions, interacting with both the static elements: the river and the finish line, and the dynamic elements: the cars and the wooden logs, all while being controlled by the player.

Given the speed of the internal clock, we created and instantiated a new clock divider with a frequency of approximately 60Hz (referred to in the code as *sixclk*) to control the blocks' speed [4]. This will ensure the blocks move at an adequate speed. All blocks require a starting position, for our project they used the coordinates stated in Table2 below:

Table 2: Moving Blocks' Starting Positions

OBJECT	Horizontal Coordinate	Vertical Coordinate
Frog	736	835
Car00	10	506
Car01	638	506
Car10	1286	611
Car11	658	611
Car20	10	711
Car21	638	711
Log00	10	306
Log01	638	306
Log10	1286	206
Log11	658	206
Log20	10	106
Log21	638	106

As it can be seen in table2, the car-blocks and log-blocks are grouped to have two blocks aligned at specific depths, given that their movement will be horizontal, this grants a certain level of difficulty when crossing each line. The movement of each element is then specified by means of an 'always block' using the newly created clock (*sixclk*).

1. FROG-BLOCK MOVEMENT

To allow control over frog-block, the code uses 5 instantiated input variables connected to 5 buttons on the control panel of the FPGA board, the buttons correspond to: Up, Down, Left, Right and Centre. The centre button is the restart button. Hence, the logic first addresses the case when it is pressed. In this scenario it will return the frog-block to its starting position (and deactivate the internal "Win" signal if active). The other four buttons will generate an input signal that will cause the frog-block to move in the direction pressed [4]. Technically, it consists on increasing or decreasing the vertical or horizontal coordinates by a fixed factor as long as it lies within the specific area. This area takes into account the size of the block to avoid covering the border (setting the depth between 15 and 850 and the width from 10 to 1395). The speed was determined to be higher in the vertical than in the horizontal direction to facilitate avoiding obstacles. To be precise, the vertical movement is performed in steps of 10 pixels per clock cycle, whereas the horizontal is done in steps of 4. Yet, the logic for the interactions with all other blocks is determined before the general movement of the frog-block.

Regarding all the cars: If the frog-block and a car-block 'make contact' (occupy consecutive or common pixels) the frog-block returns to its original position and an internal 'Lostlive' signal is activated, which will be output to a different module: "GAMETOP" (the unifying module). In order to determine when both blocks are in contact we use the relative distances of their top-right corner, which can be generalised to the interaction with all cars (as seen in Figure4).

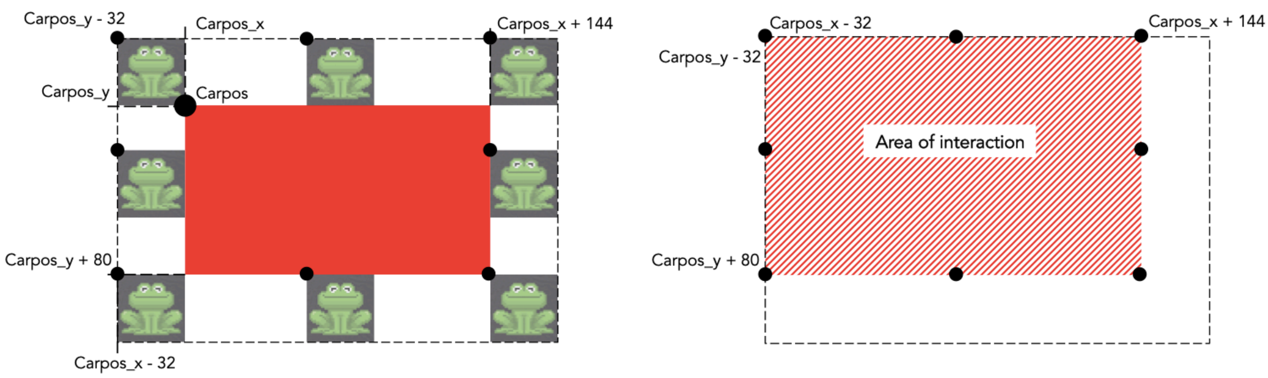

Figure 4: Area of interaction between Frog and Car-block

On the other hand, the frog-block should be able to get on top of the wooden logs and move as if it was on top of them. This can be achieved by setting the frog-block's horizontal coordinate (*blkpos_x_reg* in our project) to increase or decrease by a constant factor equal to the speed of the moving wooden log for as long as it remains on top of it. After that declaration, the frog-block movement logic can be included with certain speed modifications, allowing the player to move normally on top of a moving log-block. Considering that the object will only move horizontally, simply its horizontal speed (Equation2) needs to be adjusted:

$$blkpos\_x\_reg \ = blkpos\_x\_reg + NEW\_SPEED \tag{2}$$

Which can be obtained following equations (3) and (4):

For movement towards the right: $NEW_{SPEED} = LogSpeed_{horizontal} + 4$ (3)

For movement towards the left: $NEW_{SPEED} = LogSpeed_{horizontal} - 4$ (4)

The interplay with the log-blocks is also determined based on both their top-right corners. To interact frog-block will need to lie within the log-blocks' horizontal boundaries, but it will have more flexibility in the vertical direction to avoid falling in the water when transitioning between blocks, following the same process detailed in Figure4.

Next, the river area is addressed. If the log-block interaction conditions are not met and a large proportion of frog-block is found on the river area (anywhere within a depth of 105 and 385 for its top-corner's vertical coordinate), this will equate to falling into the river; thus, returning to the starting position and activating the internal 'Lostlive' signal (to reduce the 'lives' counter). Similarly, if frog-block is found in the 'finish line' area (within a vertical pixel depth between 10 and 80), the "Win" signal will be activated and output to other modules.

As long as none of the above conditions are met: or in other words, while the frog does not interact with any other element, it will move freely around the screen following the operations determined by the player (explained above). Note that for all conditions where the user did not return to its starting position, the internal "Lostlive" register is set to zero to avoid losing lives on loop after losing one.

2. CAR-BLOCKS AND LOG-BLOCKS MOVEMENT

The next section of the module focuses on each individual moving block. In order to hold the horizontal distance between moving blocks constant each aligned couple moves at a specific speed, and moves to the beginning of the path after reaching the screen limit.

In the case of the road, each aligned couple of cars move at a constant speed of 4, -7 (Note the negative sign due to the direction of the middle line being reversed) and 10 pixels per time cycle respectively as it gets closer to the finish line. Following the same scheme, each couple of log-blocks go at 2, -5 (equally, note the reverse direction) and 7 pixels per time cycle, resulting in an increasingly difficult path.

Lastly, the module assigns the values stored in each moving block's coordinates to their respective output variable, as well as the value of the internal "Win" and "Lostlive" registers to send both signals to other modules.

C) SEVENSEG AND SEGINTERFACE (Appendix 6)

These two modules hold the code of the digits displayed on the FPGA board, which count the "lives/opportunities" remaining.

The module *Sevenseg* sets the logic for the seven-segment display (as shown in Figure5). Essentially, this module establishes which segments need to be switched on to display each number [7]. For instance, (following the model variables from Figure5) in order to represent number 5 in the circuit, segments: a, c, d, f and g should be active. Furthermore, considering that in the module all seven segment variables (a,b,c,d,e,f,g) are assigned to a register called 'intseg', the number 5 would translate to: 1011011 (a=1,b=0,c=1,d=1,e=0,f=1,g=1), as can be seen in the module.

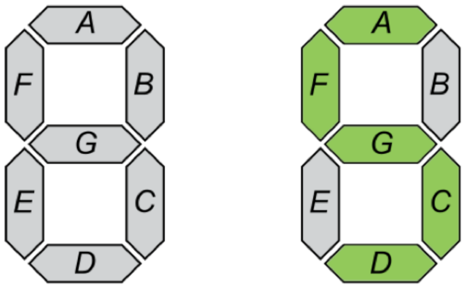

Figure 5: Seven-Segment Circuit Display

The module *Seginterface* takes 8 4-bit inputs to print their values in the 8 seven segment circuits of the FPGA board. This is done with a new internal clock with a frequency of approximately 1Hz and the *Sevenseg* decoder created above, which is instantiated in the module [8].

D) GAMETOP (Appendix 7)

The fact that certain variables are used across different modules shows the modules' interconnection. In *Gametop*, all modules are combined by means of instantiation, creating the bridge that turns the output variables of some modules into the input variables of other modules. In addition, it connects the elements being used from the FPGA board with the modules where they are used by means of the constraints file, such as the movement buttons (Up, Down, Right, Left and Centre). This module also creates the 60Hz clock used to drive the moving block's speed: *sixclk*, which is instantiated in the module: *Frogmove*.

Furthermore, this module also holds the logic controlling the display of both the screen and the FPGA board. This is achieved through an 'always block' based on the clock created above: *sixclk*. This block will use the output signals from module *Frogmove* to control both displays. If the centre button is pressed, the game will restart by setting the screen settings to the normal game (setting the "Win" and "Lost" signal to zero; consequently, causing *Drawcon* to display the game) and returning the lives from the FPGA to 5. When the "Win" signal is switched on, it will transmit the signal to *Drawcon* to display the "win screen". If the "Lostlive" signal is high, indicating the player has fallen on an obstacle, the number displayed in the FPGA will decrease by a factor of one and all the LEDs in the FPGA board will flash to highlight it. If the player had no lives left, the "Lost" signal will be activated, indicating *Drawcon* to display the "Loss screen" (solid red).

TESTING PROCEDURE

The project was tested at several stages: after creating the background, adding a controllable moving block, adding the first moving blocks, setting the interaction with blocks, setting the interaction with areas of the background, introducing Sprites, establishing the conditions for screen changes, and connecting display elements of the FPGA board. Each test ensured the additions generated the expected outcome without modifying the achieved goals, such as having the correct screen display, control over a block's movement, moving objects, correct responses from moving objects and background areas, etc. In order to prevent unspotted errors, this was done by external individuals (friends and colleagues) as well as by the developers.

For the complete duration of the project we ensure the code remained structured and readable by adding annotations and using indentation where useful, facilitating the work of spotting errors. We reused a lot of our own tested code for cases with similar logic (such as the car-block and log-block movement), reducing the probability of introducing new mistakes. However, we still encountered several challenges leading to the development of a "Problem solving" Methodology. Using the 'messages panel' informing of the "Critical Warnings", we located the line of conflict to:

1. Check the variables are correct: wires or registers (with correct bit sizes)
2. Check module variables have been accurately instantiated
3. Run "test-bench" to trace the source of the error, possibly to other variables not working (in that case repeat steps 1 and 2)
4. If possible, try moving the logic of the variable causing the warning to the module where it is used
5. Consider returning to the previous 'error-free' stage and use a different approach to achieve the specification

Our approach was goal-oriented while remaining time-conscious, allowing us to achieve our objectives within the time specified.

REFLECTION SECTION

I would evaluate the task highly positive. The fact that it not only built on the topics covered in lectures and labs, but it evolved from the work completed in the laboratory sessions made it remarkably dynamic. Moreover, the reduce size of the team is also a great element. Despite the groups' differences in ability, it ensures both members get a high exposure to HDL whilst bringing different perspectives to the task. Additionally, the practical work involved combined with the freedom to create the game specification made the project deeply engaging. I particularly enjoyed this aspect as it helped me develop my interest in Hardware programing and fostered my creativity.

I understand how this freedom can generate a wide range of paces across the groups even causing frustration in some cases. Nevertheless, I have experienced that the project remains achievable, proving the current method useful. The element I found more discouraging was the importance of sprites (accounting for 20% of the mark) without giving them enough weight in the lab or offering further resources in Moodle.

Overall, The project consolidated everything learned in lectures as well as encouraged researching extra material, such as the specifics of creating Sprites. Personally, I have acquired a good level of competence in Verilog, further my programing knowledge by understanding how an HDL works, and improved my project management skills by establishing and using work methodologies (such as the "Problem-solving" Methodology described above). On the other hand, I could not look into the introduction of sounds for the game, which could have been very interesting, nor adding all the Sprites planed or refine the code, for instance by setting parameters to manage values more efficiently.

My future recommendations would be to consider the possibility of including more resources concerning the creation and use of Sprites in Moodle and to study the option of increasing the number of laboratory sessions setting a higher volume of work at the beginning of the term. This last suggestion could enable all students to learn about all the characteristics of working with an FPGA board (from using it to control moving blocks to adding a range of sound tracks). For example, it could be achieved by adding more support sessions ensuring the weekly lab requirements are met.

REFERENCES

[1] Warwick School of Engineering (2018). ES3B2 "Digital Systems Design" – Lab Experiment 4 – VGA Output (Lab 4 Instructions (15/11). Available at: https://moodle.warwick.ac.uk/course/view.php?id=25288

[2] Oxford Web Studio. "What is the RGB color system? ". [online] Available at: https://www.oxfordwebstudio.com/en/did-you-know/what-is-the-rgb-color-system [Accessed 3 Jan. 2019].

[3]Sanchez, E. Ecole Polytechnique Fédérale de Lausanne. "A VGA Display Controller" [online] Lslepfl.ch. Available at: http://lslwww.epfl.ch/pages/teaching/cours_lsl/ca_es/VGA.pdf [Accessed 26 Nov. 2018].

[4] Warwick School of Engineering (2018). ES3B2 "Digital Systems Design" – Lab Experiment 5 – Design Project II (Lab 5 Instructions (22/11). Available at: https://moodle.warwick.ac.uk/course/view.php?id=25288

[5] Millwood, J. "Jesse-Millwood/image-2-coe". [online] GitHub. Available at: https://github.com/Jesse-Millwood/image-2-coe [Accessed 30 Nov. 2018].

[6] Warwick School of Engineering (2018). ES3B2 "Digital Systems Design" – Lab Experiment 6 – Design Project III (Lab 6 Instructions (29/11). Available at: https://moodle.warwick.ac.uk/course/view.php?id=25288

[7] Warwick School of Engineering (2018). ES3B2 "Digital Systems Design" – Lab Experiment 1 (Lab 1 Instructions (18/10). Available at: https://moodle.warwick.ac.uk/course/view.php?id=25288

[8] Warwick School of Engineering (2018). ES3B2 "Digital Systems Design" – Lab Experiment 2 (Lab 2 Instructions (01/11). Available at: https://moodle.warwick.ac.uk/course/view.php?id=25288

APPENDIX 1 – VGA_OUT

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 22.11.2018 13:20:23
// 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,
    input [3:0]draw_r,
    input [3:0]draw_g,
    input [3:0]draw_b,
    output [3:0] pix_r,
    output [3:0] pix_g,
    output [3:0] pix_b,
    output hsync,
    output vsync,
    output [10:0] curr_x,
    output [9:0] curr_y
    );

    reg [10:0] hcount=11'd0;
    reg [9:0] vcount=10'd0;
    reg [10:0] reg_x = 11'd0;
    reg [9:0] reg_y = 10'd0;

    always @ (posedge clk)
        begin
            if (hcount==11'd1903) begin
                hcount <= 11'd0;
                if (vcount==10'd931)
                    vcount <= 10'd0;
```

```
            else
               vcount <= vcount + 1;
         end else
            hcount <= hcount + 1;
      end

   always @ (posedge clk)
      begin
         if (hcount>=11'd384 && hcount <= 11'd1823 && vcount>= 10'd31 && vcount<= 10'd930)
         begin
            reg_x <= reg_x + 1;
            if (reg_x==11'd1439)
            begin
              reg_x <= 11'd0;
              if (reg_y==10'd899)
                 reg_y <= 10'd0;
              else
                reg_y <= reg_y + 1;
            end
      end
      end

   assign hsync = (hcount>=0 && hcount<=151) ?  1'b0 : 1'b1;
   assign vsync = (vcount>=0 && vcount<=2) ? 1'b1 : 1'b0;

   assign pix_r = (hcount>=384 && hcount <=1823 && vcount>= 31 && vcount<=930) ? draw_r : 4'd0;
   assign pix_g = (hcount>=384 && hcount <=1823 && vcount>= 31 && vcount<=930) ? draw_g : 4'd0;
   assign pix_b = (hcount>=384 && hcount <=1823 && vcount>= 31 && vcount<=930) ? draw_b : 4'd0;

   assign curr_y = reg_y;
   assign curr_x = reg_x;

endmodule
```

APPENDIX 2 – DRAWCON

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 27.11.2018 15:47:40
// Design Name:
// Module Name: drawcon
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
```

```
//
//////////////////////////////////////////////////////////////////////////////////


module drawcon (
    input gen_clk,
    input [10:0]frogpos_x,
    input [9:0]frogpos_y,
    input [10:0] carpos_x,
    input [9:0] carpos_y,
    input [10:0] carpos01_x,
    input [9:0] carpos01_y,
    input [10:0] carpos1_x,
    input [9:0] carpos1_y,
    input [10:0] carpos11_x,
    input [9:0] carpos11_y,
    input [10:0] carpos2_x,
    input [9:0] carpos2_y,
    input [10:0] carpos21_x,
    input [9:0] carpos21_y,
    input [10:0] logpos_x,
    input [9:0] logpos_y,
    input [10:0] logpos01_x,
    input [9:0] logpos01_y,
    input [10:0] logpos1_x,
    input [9:0] logpos1_y,
    input [10:0] logpos11_x,
    input [9:0] logpos11_y,
    input [10:0] logpos2_x,
    input [9:0] logpos2_y,
    input [10:0] logpos21_x,
    input [9:0] logpos21_y,
    input [10:0]draw_x,
    input [9:0]draw_y,
    output [3:0]draw_r,
    output [3:0]draw_g,
    output [3:0]draw_b,
    input winf,
    input lost,
    output [9:0]addr_f,
    input [11:0]dout_f,
    output [18:0]addr_w,
    input [11:0]dout_w
    );

    blk_mem_gen_0 your_instance_name (
      .clka(gen_clk),    // input wire clka
      .ena(1),      // input wire ena
      //.wea(wea),      // input wire [0 : 0] wea
      .addra(addr_f),  // input wire [9 : 0] addra
      //.dina(dina),    // input wire [11 : 0] dina
      .douta(dout_f)  // output wire [11 : 0] douta
    );

    reg [9:0] addr_f_reg;
```

```
blk_mem_gen_1 winscreen (
  .clka(gen_clk),    // input wire clka
  .ena(1),      // input wire ena
  //.wea(wea),      // input wire [0 : 0] wea
  .addra(addr_w),  // input wire [18 : 0] addra
  //.dina(dina),    // input wire [11 : 0] dina
  .douta(dout_w)  // output wire [11 : 0] douta
);

reg [18:0] addr_w_reg;

reg [3:0] draw_r_win;
reg [3:0] draw_g_win;
reg [3:0] draw_b_win;

reg [3:0] draw_r_lost;
reg [3:0] draw_g_lost;
reg [3:0] draw_b_lost;

reg [3:0]bg_r = 4'd0;
reg [3:0]bg_g = 4'd0;
reg [3:0]bg_b = 4'd0;

reg [3:0]blk_r = 0;
reg [3:0]blk_g = 0;
reg [3:0]blk_b = 0;

reg [3:0]draw_r_reg = 0;
reg [3:0]draw_g_reg = 0;
reg [3:0]draw_b_reg = 0;

reg [3:0]draw_r_regF;
reg [3:0]draw_g_regF;
reg [3:0]draw_b_regF;

reg [3:0] car_r = 0;
reg [3:0] car_g = 0;
reg [3:0] car_b = 0;
reg [3:0] car01_r = 0;
reg [3:0] car01_g = 0;
reg [3:0] car01_b = 0;
reg [3:0] car1_r = 0;
reg [3:0] car1_g = 0;
reg [3:0] car1_b = 0;
reg [3:0] car11_r = 0;
reg [3:0] car11_g = 0;
reg [3:0] car11_b = 0;
reg [3:0] car2_r = 0;
reg [3:0] car2_g = 0;
reg [3:0] car2_b = 0;
reg [3:0] car21_r = 0;
reg [3:0] car21_g = 0;
reg [3:0] car21_b = 0;

reg [3:0] log_r = 0;
reg [3:0] log_g = 0;
```

```
reg [3:0] log_b = 0;
reg [3:0] log01_r = 0;
reg [3:0] log01_g = 0;
reg [3:0] log01_b = 0;
reg [3:0] log1_r = 0;
reg [3:0] log1_g = 0;
reg [3:0] log1_b = 0;
reg [3:0] log11_r = 0;
reg [3:0] log11_g = 0;
reg [3:0] log11_b = 0;
reg [3:0] log2_r = 0;
reg [3:0] log2_g = 0;
reg [3:0] log2_b = 0;
reg [3:0] log21_r = 0;
reg [3:0] log21_g = 0;
reg [3:0] log21_b = 0;

reg [10:0] frogpos_x_reg;
reg [9:0] frogpos_y_reg;

// Background:

always @ *
if (draw_x >= 10 && draw_x <= 1430 && draw_y >= 10 && draw_y <= 890)
begin
   // River:
   if (draw_y >= 100 && draw_y <= 400)
   begin
      bg_r <= 4'd0;
      bg_g <= 4'd0;
      bg_b <= 4'd9;
   end
   else
   begin
      //road:
      if (draw_y >= 500 && draw_y <=800)
      begin
         // lines:
         if (draw_y >= 595 && draw_y <= 605 | draw_y >= 695 && draw_y <= 705)
         begin
            //line 1 and 2:
            if (draw_x >= 10 && draw_x <= 58 | draw_x >= 106 && draw_x <= 154 |draw_x >= 202 &&
draw_x <= 250 |draw_x >= 298 && draw_x <= 346 |draw_x >= 394 && draw_x <= 442 |draw_x >= 490
&& draw_x <= 538 |draw_x >= 586 && draw_x <= 634 |draw_x >= 682 && draw_x <= 730 |draw_x >=
778 && draw_x <= 826 |draw_x >= 874 && draw_x <= 922 |draw_x >= 970 && draw_x <= 1018 |draw_x
>= 1066 && draw_x <= 1114 |draw_x >= 1162 && draw_x <= 1210 |draw_x >= 1258 && draw_x <=
1306 |draw_x >= 1354 && draw_x <= 1402)
               begin
               bg_r <= 4'd15;
               bg_g <= 4'd15;
               bg_b <= 4'd15;
               end
            //give lines the color of the road when not white:
            else
               begin
               bg_r <= 4'd7;
```

```
                bg_g <= 4'd7;
                bg_b <= 4'd7;
                end
            end
            // color road:
            else
            begin
                bg_r <= 4'd7;
                bg_g <= 4'd7;
                bg_b <= 4'd7;
            end
        end
        // do other stuff:
        else
        begin
            // finish line:
            if (draw_y >= 10 && draw_y <= 100)
            begin
                // first finish line
                if (draw_y >= 25 && draw_y <= 55)
                begin
                    // set finish line 1
                    if (draw_x >= 10 && draw_x <= 58 | draw_x >= 106 && draw_x <= 154 |draw_x >= 202
&& draw_x <= 250 |draw_x >= 298 && draw_x <= 346 |draw_x >= 394 && draw_x <= 442 |draw_x >=
490 && draw_x <= 538 |draw_x >= 586 && draw_x <= 634 |draw_x >= 682 && draw_x <= 730 |draw_x
>= 778 && draw_x <= 826 |draw_x >= 874 && draw_x <= 922 |draw_x >= 970 && draw_x <= 1018
|draw_x >= 1066 && draw_x <= 1114 |draw_x >= 1162 && draw_x <= 1210 |draw_x >= 1258 && draw_x
<= 1306 |draw_x >= 1354 && draw_x <= 1402)
                    begin
                        bg_r <= 4'd15;
                        bg_g <= 4'd15;
                        bg_b <= 4'd15;
                    end
                    else
                    begin
                        bg_r <= 4'd0;
                        bg_g <= 4'd0;
                        bg_b <= 4'd0;
                    end
                end
                // do the 2nd finish line and grass:
                else
                begin
                    // second finish line:
                    if (draw_y >= 55 && draw_y <= 85)
                    begin
                        //set finish line 2:
                        if (draw_x >= 10 && draw_x <= 58 | draw_x >= 106 && draw_x <= 154 |draw_x >= 202
&& draw_x <= 250 |draw_x >= 298 && draw_x <= 346 |draw_x >= 394 && draw_x <= 442 |draw_x >=
490 && draw_x <= 538 |draw_x >= 586 && draw_x <= 634 |draw_x >= 682 && draw_x <= 730 |draw_x
>= 778 && draw_x <= 826 |draw_x >= 874 && draw_x <= 922 |draw_x >= 970 && draw_x <= 1018
|draw_x >= 1066 && draw_x <= 1114 |draw_x >= 1162 && draw_x <= 1210 |draw_x >= 1258 && draw_x
<= 1306 |draw_x >= 1354 && draw_x <= 1402)
                        begin
                            bg_r <= 4'd0;
                            bg_g <= 4'd0;
```

```
                        bg_b <= 4'd0;
                    end
                    else
                    begin
                        bg_r <= 4'd15;
                        bg_g <= 4'd15;
                        bg_b <= 4'd15;
                    end
                end
                //Give grass color to the rest of that area:
                else
                begin
                    bg_r <= 4'd0;
                    bg_g <= 4'd9;
                    bg_b <= 4'd2;
                end
            end
        end
        else
        begin
            bg_r <= 4'd0;
            bg_g <= 4'd9;
            bg_b <= 4'd2;
        end
    end
  end
end
//Border:
else
begin
  bg_r <= 4'd15;
  bg_g <= 4'd15;
  bg_b <= 4'd15;
end

//Frog:

always @ *
if (draw_x >= frogpos_x && draw_x <= frogpos_x + 32 && draw_y >= frogpos_y && draw_y <=
frogpos_y + 32)
begin
  addr_f_reg <= (((draw_y - frogpos_y) * 6'd32) + (draw_x - frogpos_x));
end

//winscreen
always @ *
if (draw_x >= 360 && draw_x <= 1080 && draw_y >= 225 && draw_y <= 675)
begin
  addr_w_reg <= (((draw_y - 360) * 720) + (draw_x - 225));
end
// Car:

always @ *
if (draw_x >= carpos_x && draw_x <= carpos_x + 144 && draw_y >= carpos_y && draw_y <=
carpos_y + 80)
begin
```

```
        car_r <= 4'd15;
        car_g <= 4'd0;
        car_b <= 4'd0;
    end
    else
        begin
        car_r <= 4'd0;
        car_g <= 4'd0;
        car_b <= 4'd0;
        end

 always @ *
    if (draw_x >= carpos01_x && draw_x <= carpos01_x + 144 && draw_y >= carpos01_y && draw_y <=
carpos01_y + 80)
    begin
        car01_r <= 4'd15;
        car01_g <= 4'd0;
        car01_b <= 4'd0;
    end
    else
        begin
        car01_r <= 4'd0;
        car01_g <= 4'd0;
        car01_b <= 4'd0;
        end

    always @ *
    if (draw_x >= carpos1_x && draw_x <= carpos1_x + 144 && draw_y >= carpos1_y && draw_y <=
carpos1_y + 80)
    begin
            car1_r <= 4'd15;
            car1_g <= 4'd0;
            car1_b <= 4'd0;
        end
        else
            begin
            car1_r <= 4'd0;
            car1_g <= 4'd0;
            car1_b <= 4'd0;
            end

    always @ *
    if (draw_x >= carpos11_x && draw_x <= carpos11_x + 144 && draw_y >= carpos11_y && draw_y <=
carpos11_y + 80)
    begin
            car11_r <= 4'd15;
            car11_g <= 4'd0;
            car11_b <= 4'd0;
        end
        else
            begin
            car11_r <= 4'd0;
            car11_g <= 4'd0;
            car11_b <= 4'd0;
            end
```

```
    always @ *
       if (draw_x >= carpos2_x && draw_x <= carpos2_x + 144 && draw_y >= carpos2_y && draw_y <=
carpos2_y + 80)
       begin
           car2_r <= 4'd15;
           car2_g <= 4'd0;
           car2_b <= 4'd0;
       end
       else
           begin
           car2_r <= 4'd0;
           car2_g <= 4'd0;
           car2_b <= 4'd0;
           end

    always @ *
       if (draw_x >= carpos21_x && draw_x <= carpos21_x + 144 && draw_y >= carpos21_y && draw_y
<= carpos21_y + 80)
       begin
           car21_r <= 4'd15;
           car21_g <= 4'd0;
           car21_b <= 4'd0;
       end
       else
           begin
           car21_r <= 4'd0;
           car21_g <= 4'd0;
           car21_b <= 4'd0;
           end

    // Log:

    always @ *
        if (draw_x >= logpos_x && draw_x <= logpos_x + 144 && draw_y >= logpos_y && draw_y <=
logpos_y + 80)
        begin
            log_r <= 4'd14;
            log_g <= 4'd9;
            log_b <= 4'd5;
        end
        else
            begin
            log_r <= 4'd0;
            log_g <= 4'd0;
            log_b <= 4'd0;
            end

     always @ *
        if (draw_x >= logpos01_x && draw_x <= logpos01_x + 144 && draw_y >= logpos01_y && draw_y
<= logpos01_y + 80)
        begin
            log01_r <= 4'd14;
            log01_g <= 4'd9;
            log01_b <= 4'd5;
        end
        else
```

```
            begin
            log01_r <= 4'd0;
            log01_g <= 4'd0;
            log01_b <= 4'd0;
            end

    always @ *
        if (draw_x >= logpos1_x && draw_x <= logpos1_x + 288 && draw_y >= logpos1_y && draw_y <=
logpos1_y + 80)
        begin
            log1_r <= 4'd14;
            log1_g <= 4'd9;
            log1_b <= 4'd5;
        end
        else
            begin
            log1_r <= 4'd0;
            log1_g <= 4'd0;
            log1_b <= 4'd0;
            end

   always @ *
        if (draw_x >= logpos11_x && draw_x <= logpos11_x + 288 && draw_y >= logpos11_y && draw_y
<= logpos11_y + 80)
        begin
            log11_r <= 4'd14;
            log11_g <= 4'd9;
            log11_b <= 4'd5;
        end
        else
            begin
            log11_r <= 4'd0;
            log11_g <= 4'd0;
            log11_b <= 4'd0;
            end

    always @ *
        if (draw_x >= logpos2_x && draw_x <= logpos2_x + 144 && draw_y >= logpos2_y && draw_y <=
logpos2_y + 80)
        begin
            log2_r <= 4'd14;
            log2_g <= 4'd9;
            log2_b <= 4'd5;
        end
        else
            begin
            log2_r <= 4'd0;
            log2_g <= 4'd0;
            log2_b <= 4'd0;
            end

    always @ *
        if (draw_x >= logpos21_x && draw_x <= logpos21_x + 144 && draw_y >= logpos21_y && draw_y
<= logpos21_y + 80)
        begin
            log21_r <= 4'd14;
```

```
      log21_g <= 4'd9;
      log21_b <= 4'd5;
    end
    else
      begin
      log21_r <= 4'd0;
      log21_g <= 4'd0;
      log21_b <= 4'd0;
      end

  // Recognise blocks instead of background:
  always @ *
  if (draw_x >= frogpos_x && draw_x <= frogpos_x + 32 && draw_y >= frogpos_y && draw_y <=
frogpos_y + 32)
    begin
    draw_r_reg <= dout_f[11:8];
    draw_g_reg <= dout_f[7:4];
    draw_b_reg <= dout_f[3:0];
    end
  else
    if (car_r == 15 && car_g == 0 && car_b == 0)
      begin
      draw_r_reg <= car_r;
      draw_g_reg <= car_g;
      draw_b_reg <= car_b;
      end
    else
    if (car01_r == 15 && car01_g == 0 && car01_b == 0)
    begin
      draw_r_reg <= car01_r;
      draw_g_reg <= car01_g;
      draw_b_reg <= car01_b;
    end
    else
      if ( car1_r == 15 && car1_g == 0 && car1_b == 0)
        begin
        draw_r_reg <= car1_r;
        draw_g_reg <= car1_g;
        draw_b_reg <= car1_b;
        end
        else
        if ( car11_r == 15 && car11_g == 0 && car11_b == 0)
          begin
          draw_r_reg <= car11_r;
          draw_g_reg <= car11_g;
          draw_b_reg <= car11_b;
          end
        else
        if ( car2_r == 15 && car2_g == 0 && car2_b == 0)
          begin
          draw_r_reg <= car2_r;
          draw_g_reg <= car2_g;
          draw_b_reg <= car2_b;
          end
        else
        if ( car21_r == 15 && car21_g == 0 && car21_b == 0)
```

```
begin
draw_r_reg <= car21_r;
draw_g_reg <= car21_g;
draw_b_reg <= car21_b;
end

else
   if (log_r == 14 && log_g == 9 && log_b == 5)
      begin
      draw_r_reg <= log_r;
      draw_g_reg <= log_g;
      draw_b_reg <= log_b;
      end
   else
   if (log01_r == 14 && log01_g == 9 && log01_b == 5)
      begin
      draw_r_reg <= log01_r;
      draw_g_reg <= log01_g;
      draw_b_reg <= log01_b;
      end
   else
   if (log1_r == 14 && log1_g == 9 && log1_b == 5)
         begin
         draw_r_reg <= log1_r;
         draw_g_reg <= log1_g;
         draw_b_reg <= log1_b;
         end
     else
     if (log11_r == 14 && log11_g == 9 && log11_b == 5)
           begin
           draw_r_reg <= log11_r;
           draw_g_reg <= log11_g;
           draw_b_reg <= log11_b;
           end
       else
       if (log2_r == 14 && log2_g == 9 && log2_b == 5)
             begin
             draw_r_reg <= log2_r;
             draw_g_reg <= log2_g;
             draw_b_reg <= log2_b;
             end
         else
         if (log21_r == 14 && log21_g == 9 && log21_b == 5)
             begin
             draw_r_reg <= log21_r;
             draw_g_reg <= log21_g;
             draw_b_reg <= log21_b;
             end
         else
         begin
         draw_r_reg <= bg_r;
         draw_g_reg <= bg_g;
         draw_b_reg <= bg_b;
         end

always @ *
```

```
    if (draw_x >= 360 && draw_x <= 1080 && draw_y >= 225 && draw_y <= 675)
    begin
        draw_r_win <= dout_w[11:8];
        draw_g_win <= dout_w[7:4];
        draw_b_win <= dout_w[3:0];
    end
    else
    begin
        draw_r_win <= bg_r;
        draw_g_win <= bg_g;
        draw_b_win <= bg_b;
    end

    assign addr_f = addr_f_reg;

    assign addr_w = addr_w_reg;

    always @ *
    begin
        if (lost == 1)
        begin
        draw_r_regF <= draw_r_lost;
        draw_g_regF <= draw_g_lost;
        draw_b_regF <= draw_b_lost;
        end
        else
        if (winf == 1)
        begin
        draw_r_regF <= draw_r_win;
        draw_g_regF <= draw_g_win;
        draw_b_regF <= draw_b_win;
        end
        else
        begin
        draw_r_regF <= draw_r_reg;
        draw_g_regF <= draw_g_reg;
        draw_b_regF <= draw_b_reg;
        end
    end

    assign draw_r = draw_r_regF;
    assign draw_g = draw_g_regF;
    assign draw_b = draw_b_regF;

endmodule
```

APPENDIX 3 – Images for the Sprites

Used images:

Image 1: Frog-block Image (From Google Images)

Image 2: Win Screen

Planed images for other elements:

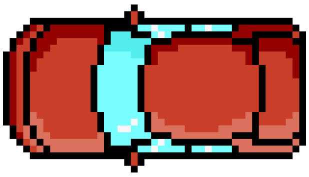

Image 3: Car-Block Image (From Google Images)

Image 4: Lost Game Screen

APPENDIX 4 – COE MATLAB FILE

```
clc; close all;
[fName, pName] = uigetfile('*.*');
I = imresize(imread([pName fName]),[32 32]);
I = (I./17);

fid = fopen('frogie1.coe','w');
fprintf(fid, '%s\n','MEMORY_INITIALIZATION_RADIX=2;');
fprintf(fid, '%s\n','MEMORY_INITIALIZATION_VECTOR=');

for i = 1:size(I,1)
   for j = 1:size(I,2)
      fprintf(fid,'%s',[dec2bin(I(i,j,1),4) dec2bin(I(i,j,2),4) dec2bin(I(i,j,3),4)]);
      if (i == size(I,1) && j == size(I,2))
```

```
            fprintf(fid, '%s', ';');
        else
            fprintf(fid,'%s', ',');
        end
    end
end

fclose(fid);
```

APPENDIX 5 – FROGMOVE

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04.12.2018 17:13:47
// Design Name:
// Module Name: FrogMove
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module FrogMove(
    input centre,
    output [10:0] frogpos_x,
    output [9:0] frogpos_y,
    output [10:0] carpos_x,
    output [9:0] carpos_y,
    output [10:0] carpos01_x,
    output [9:0] carpos01_y,
    output [10:0] carpos1_x,
    output [9:0] carpos1_y,
    output [10:0] carpos11_x,
    output [9:0] carpos11_y,
    output [10:0] carpos2_x,
    output [9:0] carpos2_y,
    output [10:0] carpos21_x,
    output [9:0] carpos21_y,
    output [10:0] logpos_x,
    output [9:0] logpos_y,
    output [10:0] logpos01_x,
    output [9:0] logpos01_y,
    output [10:0] logpos1_x,
```

```
    output [9:0] logpos1_y,
    output [10:0] logpos11_x,
    output [9:0] logpos11_y,
    output [10:0] logpos2_x,
    output [9:0] logpos2_y,
    output [10:0] logpos21_x,
    output [9:0] logpos21_y,
    output win,
    input sixclk,
    input clk,
    input up,
    input down,
    input right,
    input left,
    output [3:0] lostlive
    );

    reg [3:0]lostlive_reg = 0;

    reg win_reg = 0;

    reg [10:0]blkpos_x_reg = 736;
    reg [9:0]blkpos_y_reg = 835;

    reg [10:0] carpos_x_reg = 10;
    reg [9:0] carpos_y_reg = 506;
    reg [10:0] carpos01_x_reg = 638;
    reg [9:0] carpos01_y_reg = 506;
    reg [10:0] carpos1_x_reg = 1286;
    reg [9:0] carpos1_y_reg = 611;
    reg [10:0] carpos11_x_reg = 658;
    reg [9:0] carpos11_y_reg = 611;
    reg [10:0] carpos2_x_reg = 10;
    reg [9:0] carpos2_y_reg = 711;
    reg [10:0] carpos21_x_reg = 638;
    reg [9:0] carpos21_y_reg = 711;

    reg [10:0] logpos_x_reg = 10;
    reg [9:0] logpos_y_reg = 306;
    reg [10:0] logpos01_x_reg = 638;
    reg [9:0] logpos01_y_reg = 306;
    reg [10:0] logpos1_x_reg = 1286;
    reg [9:0] logpos1_y_reg = 206;
    reg [10:0] logpos11_x_reg = 658;
    reg [9:0] logpos11_y_reg = 206;
    reg [10:0] logpos2_x_reg = 10;
    reg [9:0] logpos2_y_reg = 106;
    reg [10:0] logpos21_x_reg = 638;
    reg [9:0] logpos21_y_reg = 106;

always @ (posedge sixclk)
    begin
    if (centre == 1)
    begin
        win_reg <= 0;
        blkpos_x_reg <= 736;
```

```
      blkpos_y_reg <= 835;
   end
   else
   if (blkpos_y_reg >= carpos_y_reg - 32 && blkpos_y_reg <= carpos_y_reg + 80 && blkpos_x_reg >=
carpos_x_reg -32 && blkpos_x_reg <= carpos_x_reg + 144)
   begin
      lostlive_reg <= 1;
      blkpos_x_reg <= 736;
      blkpos_y_reg <= 835;
   end
   else
   if (blkpos_y_reg >= carpos01_y_reg - 32 && blkpos_y_reg <= carpos01_y_reg + 80 && blkpos_x_reg
>= carpos01_x_reg -32 && blkpos_x_reg <= carpos01_x_reg + 144)
   begin
      lostlive_reg <= 1;
      blkpos_x_reg <= 736;
      blkpos_y_reg <= 835;
   end
   else
   if (blkpos_y_reg >= carpos1_y_reg - 32 && blkpos_y_reg <= carpos1_y_reg + 80 && blkpos_x_reg >=
carpos1_x_reg -32 && blkpos_x_reg <= carpos1_x_reg + 144)
   begin
      lostlive_reg <= 1;
      blkpos_x_reg <= 736;
      blkpos_y_reg <= 835;
   end
   else
   if (blkpos_y_reg >= carpos11_y_reg - 32 && blkpos_y_reg <= carpos11_y_reg + 80 && blkpos_x_reg
>= carpos11_x_reg -32 && blkpos_x_reg <= carpos11_x_reg + 144)
   begin
      blkpos_x_reg <= 736;
      blkpos_y_reg <= 835;
      lostlive_reg <= 1;
   end
   else
   if (blkpos_y_reg >= carpos2_y_reg - 32 && blkpos_y_reg <= carpos2_y_reg + 80 && blkpos_x_reg >=
carpos2_x_reg -32 && blkpos_x_reg <= carpos2_x_reg + 144)
   begin
      lostlive_reg <= 1;
      blkpos_x_reg <= 736;
      blkpos_y_reg <= 835;
   end
   else
   if (blkpos_y_reg >= carpos21_y_reg - 32 && blkpos_y_reg <= carpos21_y_reg + 80 && blkpos_x_reg
>= carpos21_x_reg -32 && blkpos_x_reg <= carpos21_x_reg + 144)
   begin
      lostlive_reg <= 1;
      blkpos_x_reg <= 736;
      blkpos_y_reg <= 835;
   end
   else
      begin
      if (blkpos_y_reg >= logpos_y_reg - 12 && blkpos_y_reg <= logpos_y_reg + 80 && blkpos_x_reg >=
logpos_x_reg -2 && blkpos_x_reg <= logpos_x_reg + 112)
         begin
         lostlive_reg <= 0;
```

```
        blkpos_x_reg <= blkpos_x_reg + 2;
        if (up == 1)
          begin
          if (blkpos_y_reg <= 15)
          begin
            blkpos_y_reg <= blkpos_y_reg;
          end
          else
          begin
            blkpos_y_reg <= blkpos_y_reg - 10;
          end
          end
        else
          begin
          if (down == 1)
            begin
            if (blkpos_y_reg >= 850)
              blkpos_y_reg <= blkpos_y_reg;
            else
              blkpos_y_reg <= blkpos_y_reg + 10;
            end
          else
            begin
            if (right == 1)
              begin
              if (blkpos_x_reg >= 1395)
                blkpos_x_reg <= blkpos_x_reg;
              else
                blkpos_x_reg <= blkpos_x_reg + 6;
              end
            else
              begin
              if (left == 1)
                begin
                if (blkpos_x_reg <= 10)
                  blkpos_x_reg <= blkpos_x_reg;
                else
                  blkpos_x_reg <= blkpos_x_reg - 2;
                end
              end
            end
          end
        end
      else
      begin
      if (blkpos_y_reg >= logpos01_y_reg - 12 && blkpos_y_reg <= logpos01_y_reg + 80 &&
blkpos_x_reg >= logpos01_x_reg -2 && blkpos_x_reg <= logpos01_x_reg + 112)
        begin
        lostlive_reg <= 0;
        blkpos_x_reg <= blkpos_x_reg + 2;
        if (up == 1)
          begin
          if (blkpos_y_reg <= 15)
          begin
            blkpos_y_reg <= blkpos_y_reg;
          end
```

```
            else
            begin
               blkpos_y_reg <= blkpos_y_reg - 10;
            end
            end
         else
            begin
            if (down == 1)
               begin
               if (blkpos_y_reg >= 850)
                  blkpos_y_reg <= blkpos_y_reg;
               else
                  blkpos_y_reg <= blkpos_y_reg + 10;
               end
            else
               begin
               if (right == 1)
                  begin
                  if (blkpos_x_reg >= 1395)
                     blkpos_x_reg <= blkpos_x_reg;
                  else
                     blkpos_x_reg <= blkpos_x_reg + 6;
                  end
               else
                  begin
                  if (left == 1)
                     begin
                     if (blkpos_x_reg <= 10)
                        blkpos_x_reg <= blkpos_x_reg;
                     else
                        blkpos_x_reg <= blkpos_x_reg - 2;
                     end
                  end
               end
            end
         end
      else
      begin
      if (blkpos_y_reg >= logpos1_y_reg - 12 && blkpos_y_reg <= logpos1_y_reg + 92 && blkpos_x_reg
>= logpos1_x_reg -2 && blkpos_x_reg <= logpos1_x_reg + 256)
         begin
         lostlive_reg <= 0;
         blkpos_x_reg <= blkpos_x_reg - 5;
         if (up == 1)
            begin
            if (blkpos_y_reg <= 15)
            begin
               blkpos_y_reg <= blkpos_y_reg;
            end
            else
            begin
               blkpos_y_reg <= blkpos_y_reg - 10;
            end
            end
         else
            begin
```

```
            if (down == 1)
              begin
              if (blkpos_y_reg >= 850)
                blkpos_y_reg <= blkpos_y_reg;
              else
                blkpos_y_reg <= blkpos_y_reg + 10;
              end
            else
              begin
              if (right == 1)
                begin
                if (blkpos_x_reg >= 1395)
                  blkpos_x_reg <= blkpos_x_reg;
                else
                  blkpos_x_reg <= blkpos_x_reg - 1;
                end
              else
                begin
                if (left == 1)
                  begin
                  if (blkpos_x_reg <= 10)
                    blkpos_x_reg <= blkpos_x_reg;
                  else
                    blkpos_x_reg <= blkpos_x_reg - 9;
                  end
                end
              end
            end
          end
        else
        begin
        if (blkpos_y_reg >= logpos11_y_reg - 12 && blkpos_y_reg <= logpos11_y_reg + 92 &&
blkpos_x_reg >= logpos11_x_reg -2 && blkpos_x_reg <= logpos11_x_reg + 256)
          begin
          lostlive_reg <= 0;
          blkpos_x_reg <= blkpos_x_reg - 5;
          if (up == 1)
            begin
            if (blkpos_y_reg <= 15)
            begin
              blkpos_y_reg <= blkpos_y_reg;
            end
            else
            begin
              blkpos_y_reg <= blkpos_y_reg - 10;
            end
            end
          else
            begin
            if (down == 1)
              begin
              if (blkpos_y_reg >= 850)
                blkpos_y_reg <= blkpos_y_reg;
              else
                blkpos_y_reg <= blkpos_y_reg + 10;
              end
```

```
          else
            begin
            if (right == 1)
              begin
              if (blkpos_x_reg >= 1395)
                blkpos_x_reg <= blkpos_x_reg;
              else
                blkpos_x_reg <= blkpos_x_reg - 1;
              end
            else
              begin
              if (left == 1)
                begin
                if (blkpos_x_reg <= 10)
                  blkpos_x_reg <= blkpos_x_reg;
                else
                  blkpos_x_reg <= blkpos_x_reg - 9;
                end
              end
            end
          end
        end
      else
      begin
      if (blkpos_y_reg >= logpos2_y_reg - 12 && blkpos_y_reg <= logpos2_y_reg + 92 && blkpos_x_reg
>= logpos2_x_reg -2 && blkpos_x_reg <= logpos2_x_reg + 112)
        begin
        lostlive_reg <= 0;
        blkpos_x_reg <= blkpos_x_reg + 7;
        if (up == 1)
          begin
          if (blkpos_y_reg <= 15)
          begin
            blkpos_y_reg <= blkpos_y_reg;
          end
          else
          begin
            blkpos_y_reg <= blkpos_y_reg - 10;
          end
          end
        else
          begin
          if (down == 1)
            begin
            if (blkpos_y_reg >= 850)
              blkpos_y_reg <= blkpos_y_reg;
            else
              blkpos_y_reg <= blkpos_y_reg + 10;
            end
          else
            begin
            if (right == 1)
              begin
              if (blkpos_x_reg >= 1395)
                blkpos_x_reg <= blkpos_x_reg;
              else
```

```
                blkpos_x_reg <= blkpos_x_reg + 11;
              end
            else
              begin
              if (left == 1)
                begin
                if (blkpos_x_reg <= 10)
                  blkpos_x_reg <= blkpos_x_reg;
                else
                  blkpos_x_reg <= blkpos_x_reg + 3;
                end
              end
            end
          end
        end
      else
      begin
        if (blkpos_y_reg >= logpos21_y_reg - 12 && blkpos_y_reg <= logpos21_y_reg + 92 &&
blkpos_x_reg >= logpos21_x_reg -2 && blkpos_x_reg <= logpos21_x_reg + 112)
          begin
          lostlive_reg <= 0;
          blkpos_x_reg <= blkpos_x_reg + 7;
          if (up == 1)
            begin
            if (blkpos_y_reg <= 15)
            begin
              blkpos_y_reg <= blkpos_y_reg;
            end
            else
            begin
              blkpos_y_reg <= blkpos_y_reg - 10;
            end
            end
          else
            begin
            if (down == 1)
              begin
              if (blkpos_y_reg >= 850)
                blkpos_y_reg <= blkpos_y_reg;
              else
                blkpos_y_reg <= blkpos_y_reg + 10;
              end
            else
              begin
              if (right == 1)
                begin
                if (blkpos_x_reg >= 1395)
                  blkpos_x_reg <= blkpos_x_reg;
                else
                  blkpos_x_reg <= blkpos_x_reg + 11;
                end
              else
                begin
                if (left == 1)
                  begin
                  if (blkpos_x_reg <= 10)
```

```
                    blkpos_x_reg <= blkpos_x_reg;
                  else
                    blkpos_x_reg <= blkpos_x_reg + 3;
                  end
                end
              end
            end
          end
        else
      if(blkpos_y_reg <= 385 && blkpos_y_reg >= 105)
      begin
      blkpos_x_reg <= 736;
      blkpos_y_reg <= 835;
      lostlive_reg <= 1;
      end
      else
      if (blkpos_y_reg <= 80 && blkpos_y_reg >= 10)
        begin
        win_reg <= 1;
        end
      else
      begin
        lostlive_reg <= 0;
        if (up == 1)
          begin
          if (blkpos_y_reg <= 15)
          begin
            blkpos_y_reg <= blkpos_y_reg;
          end
          else
          begin
            blkpos_y_reg <= blkpos_y_reg - 10;
          end
          end
        else
          begin
          if (down == 1)
            begin
            if (blkpos_y_reg >= 850)
              blkpos_y_reg <= blkpos_y_reg;
            else
              blkpos_y_reg <= blkpos_y_reg + 10;
            end
          else
            begin
            if (right == 1)
              begin
              if (blkpos_x_reg >= 1395)
                blkpos_x_reg <= blkpos_x_reg;
              else
                blkpos_x_reg <= blkpos_x_reg + 4;
              end
            else
              begin
              if (left == 1)
                begin
```

```
                    if (blkpos_x_reg <= 10)
                       blkpos_x_reg <= blkpos_x_reg;
                    else
                       blkpos_x_reg <= blkpos_x_reg - 4;
                    end
                 end
              end
           end
        end
      end
      end
      end
      end
      end
     end
  end

  // car mov
  always @ (posedge sixclk)
     if (carpos_x_reg >= 1286)
        begin
        carpos_x_reg <= 10;
        end
     else
        begin
        carpos_x_reg <= carpos_x_reg + 10;
        end

  // carpos0 mov
  always @ (posedge sixclk)
     if (carpos01_x_reg >= 1286)
        begin
        carpos01_x_reg <= 10;
        end
     else
        begin
        carpos01_x_reg <= carpos01_x_reg + 10;
        end


  always @ (posedge sixclk)
    if (carpos1_x_reg <= 10)
       begin
       carpos1_x_reg <= 1286;
       end
    else
       begin
       carpos1_x_reg <= carpos1_x_reg - 7;
       end

  always @ (posedge sixclk)
     if (carpos11_x_reg <= 10)
        begin
        carpos11_x_reg <= 1286;
        end
     else
```

```
        begin
        carpos11_x_reg <= carpos11_x_reg - 7;
        end

always @ (posedge sixclk)
   if (carpos2_x_reg >= 1286)
       begin
       carpos2_x_reg <= 10;
       end
   else
       begin
       carpos2_x_reg <= carpos2_x_reg + 4;
       end

always @ (posedge sixclk)
    if (carpos21_x_reg >= 1286)
        begin
        carpos21_x_reg <= 10;
        end
    else
        begin
        carpos21_x_reg <= carpos21_x_reg + 4;
        end

//log mov
always @ (posedge sixclk)
    if (logpos_x_reg >= 1286)
        begin
        logpos_x_reg <= 10;
        end
    else
        begin
        logpos_x_reg <= logpos_x_reg + 2;
        end

always @ (posedge sixclk)
    if (logpos01_x_reg >= 1286)
        begin
        logpos01_x_reg <= 10;
        end
    else
        begin
        logpos01_x_reg <= logpos01_x_reg + 2;
        end

always @ (posedge sixclk)
    if (logpos1_x_reg <= 10)
        begin
        logpos1_x_reg <= 1286;
        end
    else
        begin
        logpos1_x_reg <= logpos1_x_reg - 5;
        end

 always @ (posedge sixclk)
```

```
    if (logpos11_x_reg <= 10)
        begin
        logpos11_x_reg <= 1286;
        end
    else
        begin
        logpos11_x_reg <= logpos11_x_reg - 5;
        end

 always @ (posedge sixclk)
    if (logpos2_x_reg >= 1286)
        begin
        logpos2_x_reg <= 10;
        end
    else
        begin
        logpos2_x_reg <= logpos2_x_reg + 7;
        end

always @ (posedge sixclk)
    if (logpos21_x_reg >= 1286)
        begin
        logpos21_x_reg <= 10;
        end
    else
        begin
        logpos21_x_reg <= logpos21_x_reg + 7;
        end

    assign frogpos_x = blkpos_x_reg;
    assign frogpos_y = blkpos_y_reg;

    assign carpos_x = carpos_x_reg;
    assign carpos_y = carpos_y_reg;
    assign carpos1_x = carpos1_x_reg;
    assign carpos1_y = carpos1_y_reg;
    assign carpos2_x = carpos2_x_reg;
    assign carpos2_y = carpos2_y_reg;
    assign carpos01_x = carpos01_x_reg;
    assign carpos01_y = carpos01_y_reg;
    assign carpos11_x = carpos11_x_reg;
    assign carpos11_y = carpos11_y_reg;
    assign carpos21_x = carpos21_x_reg;
    assign carpos21_y = carpos21_y_reg;

    assign logpos_x = logpos_x_reg;
    assign logpos_y = logpos_y_reg;
    assign logpos01_x = logpos01_x_reg;
    assign logpos01_y = logpos01_y_reg;
    assign logpos1_x = logpos1_x_reg;
    assign logpos1_y = logpos1_y_reg;
    assign logpos11_x = logpos11_x_reg;
    assign logpos11_y = logpos11_y_reg;
    assign logpos2_x = logpos2_x_reg;
    assign logpos2_y = logpos2_y_reg;
    assign logpos21_x = logpos21_x_reg;
```

```
    assign logpos21_y = logpos21_y_reg;

    assign lostlive = lostlive_reg;

    assign win = win_reg;

endmodule
```

APPENDIX 6 – SEVENSEG & SEGINTERFACE

A) SEVENSEG

```
`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
```

B) SEGINTERFACE

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/31/2016 07:13:38 PM
// Design Name:
// Module Name: seginterface
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module seginterface(
    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 [7:0] an
  );

  wire led_clk;
  reg [3:0] dig_sel;

  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];

  reg [7:0] led_strobe = 8'b11111110;
  always @(posedge led_clk)
    led_strobe <= {led_strobe[6:0],led_strobe[7]};
  assign an = led_strobe;

  reg [2:0] led_index = 3'd0;
  always @(posedge led_clk)
    led_index <= led_index + 1'b1;

  always@*
    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

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

endmodule
```

APPENDIX 7 – GAMETOP

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 27.11.2018 14:14:19
// Design Name:
// Module Name: gametop
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module gametop(
   input clk,
   input centre,
   output [3:0] pix_r,
   output [3:0] pix_g,
   output [3:0] pix_b,
   output hsync,
   output vsync,
   input up,
   input down,
   input right,
   input left,
   output LED0,
   output LED1,
   output LED2,
   output LED3,
   output LED4,
   output LED5,
```

```
output LED6,
output LED7,
output LED8,
output LED9,
output LED10,
output LED11,
output LED12,
output LED13,
output LED14,
output LED15,
output a,
output b,
output c,
output d,
output e,
output f,
output g,
output [7:0] an
);

wire win;
reg winf_reg = 0;
wire lost;
reg lost_reg = 0;

reg [3:0]draw_r_reg;
reg [3:0]draw_g_reg;
reg [3:0]draw_b_reg;

wire [3:0]draw_r;
wire [3:0]draw_g;
wire [3:0]draw_b;

wire gen_clk;
wire [10:0] curr_x;
wire [9:0] curr_y;

reg sixclk;

wire [10:0] frogpos_x;
wire [9:0] frogpos_y;

wire [10:0] carpos_x;
wire [9:0] carpos_y;
wire [10:0] carpos01_x;
wire [9:0] carpos01_y;
wire [10:0] carpos1_x;
wire [9:0] carpos1_y;
wire [10:0] carpos11_x;
wire [9:0] carpos11_y;
wire [10:0] carpos2_x;
wire [9:0] carpos2_y;
wire [10:0] carpos21_x;
wire [9:0] carpos21_y;

wire [10:0] logpos_x;
```

```
wire [9:0] logpos_y;
wire [10:0] logpos01_x;
wire [9:0] logpos01_y;
wire [10:0] logpos1_x;
wire [9:0] logpos1_y;
wire [10:0] logpos11_x;
wire [9:0] logpos11_y;
wire [10:0] logpos2_x;
wire [9:0] logpos2_y;
wire [10:0] logpos21_x;
wire [9:0] logpos21_y;

wire w6;
wire lostlive;
wire winf;

wire return = 0;
reg return_reg = 0;

reg [3:0] hexcnt1 = 4'h5;
reg [3:0] hexcnt2=4'h0;
reg [3:0] hexcnt3=4'h0;
reg [3:0] hexcnt4=4'h0;

reg [3:0] deccnt1= 0;
reg [3:0] deccnt2= 0;
reg [3:0] deccnt3= 0;
reg [3:0] deccnt4= 0;

reg LED_reg = 0;

vga_out VGA (.clk(gen_clk), .pix_r(pix_r), .pix_g(pix_g), .pix_b(pix_b), .hsync(hsync), .vsync(vsync),
.draw_r(draw_r), .draw_g(draw_g), .draw_b(draw_b), .curr_y(curr_y), .curr_x(curr_x));

drawcon DC (.gen_clk(gen_clk), .lost(lost), .winf(winf), .draw_x(curr_x), .draw_y(curr_y),.draw_r(draw_r),
.draw_g(draw_g), .draw_b(draw_b), .frogpos_x(frogpos_x), .frogpos_y(frogpos_y), .carpos_x(carpos_x),
.carpos_y(carpos_y), .carpos01_x(carpos01_x), .carpos01_y(carpos01_y), .carpos1_x(carpos1_x),
.carpos1_y(carpos1_y), .carpos11_x(carpos11_x), .carpos11_y(carpos11_y), .carpos2_x(carpos2_x),
.carpos2_y(carpos2_y), .carpos21_x(carpos21_x), .carpos21_y(carpos21_y), .logpos_x(logpos_x),
.logpos_y(logpos_y), .logpos01_x(logpos01_x), .logpos01_y(logpos01_y), .logpos1_x(logpos1_x),
.logpos1_y(logpos1_y), .logpos11_x(logpos11_x), .logpos11_y(logpos11_y), .logpos2_x(logpos2_x),
.logpos2_y(logpos2_y), .logpos21_x(logpos21_x), .logpos21_y(logpos21_y));

FrogMove FG (.centre(centre), .win(win), .lostlive(lostlive), .clk(w6), .sixclk(sixclk),  .up(up),
.down(down), .right(right), .left(left), .frogpos_x(frogpos_x), .frogpos_y(frogpos_y), .carpos_x(carpos_x),
.carpos_y(carpos_y), .carpos01_x(carpos01_x), .carpos01_y(carpos01_y), .carpos1_x(carpos1_x),
.carpos1_y(carpos1_y), .carpos11_x(carpos11_x), .carpos11_y(carpos11_y), .carpos2_x(carpos2_x),
.carpos2_y(carpos2_y), .carpos21_x(carpos21_x), .carpos21_y(carpos21_y), .logpos_x(logpos_x),
.logpos_y(logpos_y), .logpos01_x(logpos01_x), .logpos01_y(logpos01_y), .logpos1_x(logpos1_x),
.logpos1_y(logpos1_y), .logpos11_x(logpos11_x), .logpos11_y(logpos11_y), .logpos2_x(logpos2_x),
.logpos2_y(logpos2_y), .logpos21_x(logpos21_x), .logpos21_y(logpos21_y));

seginterface SI (.a(a), .b(b), .c(c), .d(d), .e(e), .f(f), .g(g), .an(an), .clk(clk), .rst(rst), .div_clk(w6),
.dig0(hexcnt1), .dig1(hexcnt2), .dig2(hexcnt3), .dig3(hexcnt4),.dig4(deccnt1), .dig5(deccnt2),
.dig6(deccnt3), .dig7(deccnt4));
```

```
reg [20:0]clk_div = 0;

always @ (posedge clk)
begin
clk_div <= clk_div + 1;
if (clk_div == 20'd833333)
begin
   clk_div <= 0;
   sixclk <= !sixclk;
end
end

always @ (posedge sixclk)
begin
if (centre == 1)
begin
   lost_reg <= 0;
   hexcnt1 <= 5;
   return_reg <= 1;
   winf_reg <= 0;
end
else
if (win == 1)
begin
   winf_reg <= 1;
end
else
if (lostlive == 1)
   begin
   LED_reg <= 1;
   winf_reg <= 0;
   if (hexcnt1 <= 0)
      begin
      lost_reg <= 1;
      end
   else
      begin
      hexcnt1 <= hexcnt1 - 1;
      end
   end
else
   begin
   LED_reg <= 0;
   return_reg <= 0;
   winf_reg <= 0;
   end
end

assign return = return_reg;
assign lost = lost_reg;
assign winf = winf_reg;

assign LED0 = LED_reg;
assign LED1 = LED_reg;
assign LED2 = LED_reg;
```

```
    assign LED3 = LED_reg;
    assign LED4 = LED_reg;
    assign LED5 = LED_reg;
    assign LED6 = LED_reg;
    assign LED7 = LED_reg;
    assign LED8 = LED_reg;
    assign LED9 = LED_reg;
    assign LED10 = LED_reg;
    assign LED11 = LED_reg;
    assign LED12 = LED_reg;
    assign LED13 = LED_reg;
    assign LED14 = LED_reg;
    assign LED15 = LED_reg;

    clk_wiz_0 clkwiz
      (
       // Clock out ports
       .clk_out1(gen_clk),     // output clk_out1
      // Clock in ports
       .clk_in1(clk));      // input clk_in1
    // INST_TAG_END ------ End INSTANTIATION Template ---------

endmodule
```