Car Racing Game

-- Final Report

CSEE 4840 Embedded System Design

Jing Shi (js4559), Mingxin Huo (mh3452)

Yifan Li (yl3250), Siwei Su (ss4483)

Jing Shi	Sprite Display Modules and Driver
Mingxin Huo	Audio Modules and Driver
Yifan Li	Software for game logic
Siwei Su	Software for game logic

Contents

1 Overview

For this Car Racing Game, we designed and accomplished a video game imitating the existing game showed as Figure 1 with a projective view. In this video game, the player control one of the cars to compete with other opponent computer players. The major elements of this game consists of accelerate, break, score, speed, rank and bump. The player uses keyboard to control the car to turn left or right, to hit a break or accelerate while obtain over 5 golden coins. The theme of our game is to compete with the other opponents that are controlled by computer in a racing tournament, the player's goal is to get to the destination as soon as possible while trying to avoid bumping to other cars or road object, which slows down your speed, within a certain time limitation. The difficulty level of this game is determined by the time limitation, golden coins probability, which refers to the chance you can accelerate and get higher score.

Figure 1 is a screenshot of our Car Racing Game when it begins. The top three columns are rank, timer and score respectively. The bottom right demonstrates the current speed.

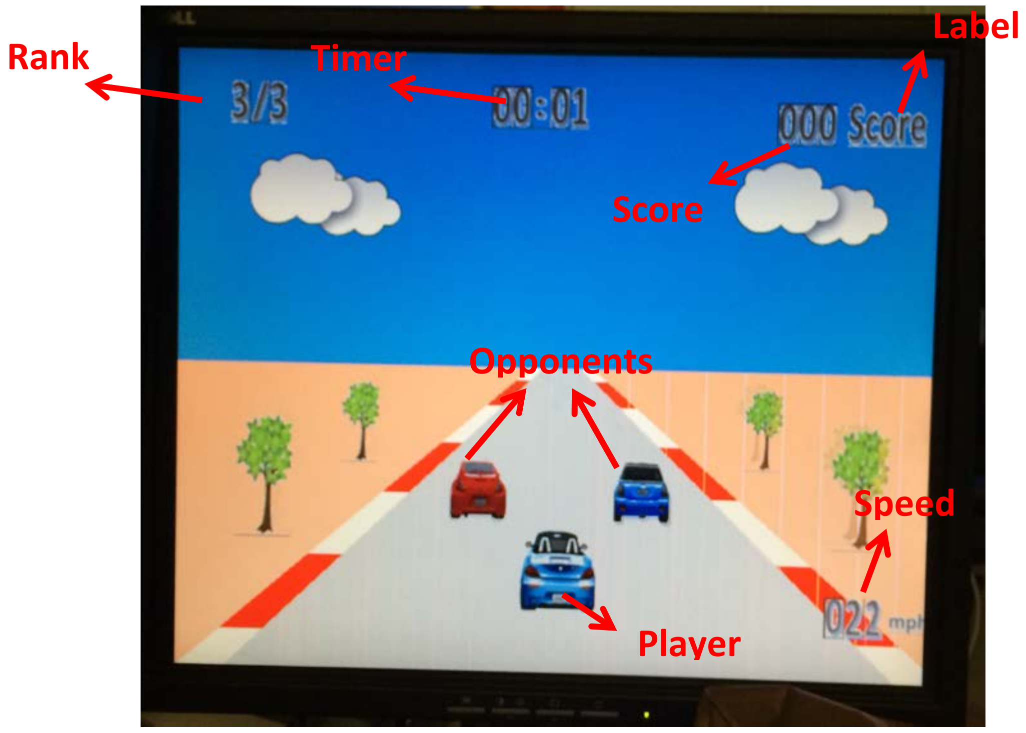


Figure 1 The Car Racing Game

2 Overall View of Design

This part shows the overall design of our software and hardware. This game consists of three major modules. First part is the Game Logic Generator which calculate the logic of this game, such as to detect bumps to obstacle, speed control based on keyboard input, opponents control and road generation, and this module is based on software. The second part is the Screen Rendering Module, we adopt the Sprite Graphics technique to decompose the display screen into 7 layers, which will be explained in derails in later section. The last part is the Audio Module which generate the proper sound under the control of game logic. The overall design of the game is demonstrated in figure 2. The following content of this paper explain each module in details.

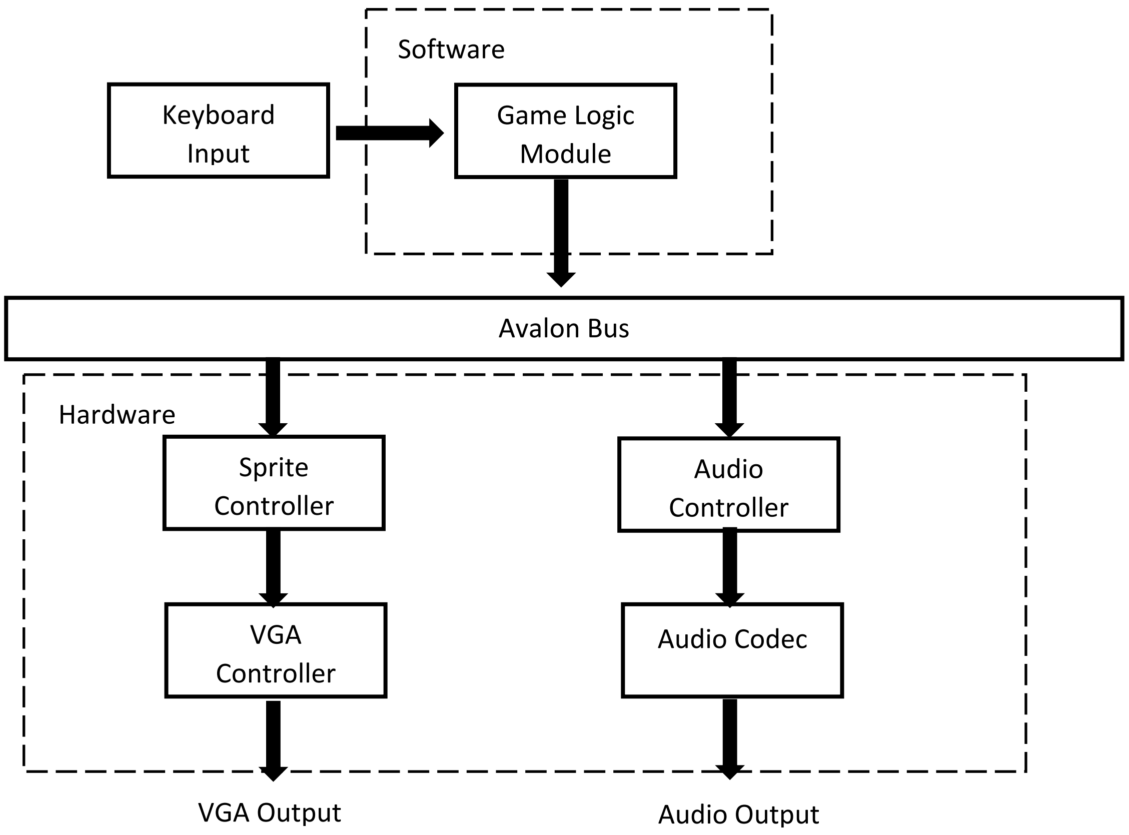


Figure 2 The overall design of Car Racing Game including Software and Hardware.

3 Display Modules

3.1 Sprite

The core of our video game display technology is Sprite, which utilizes small size of different elements to produce repeatedly pattern of pictures, minimizing the requirement for large screen pixels RGB information. The Sprite use multiple layers to overlap the pictures and generate a movement effect by control the movement of small elements in different layers, by this way obtain a video game display function.

The following Figure 3 shows the layers we use in our Car Racing Game, and the rank of layers, the layer whose priority is higher will be displayed top over those with lower priority.

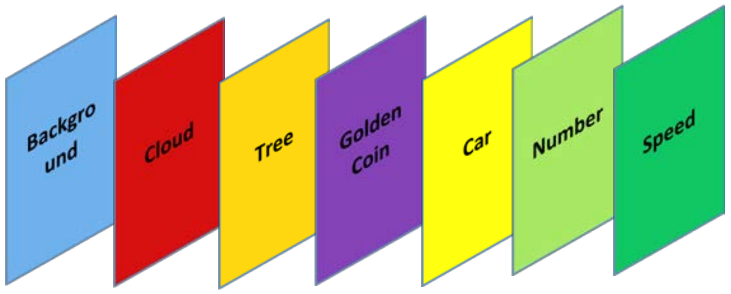
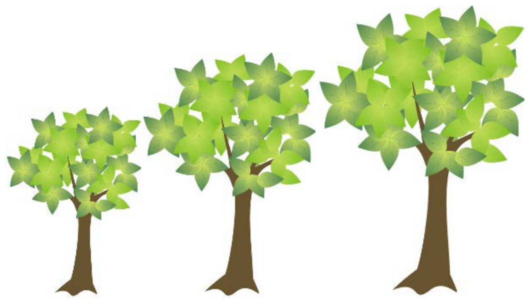


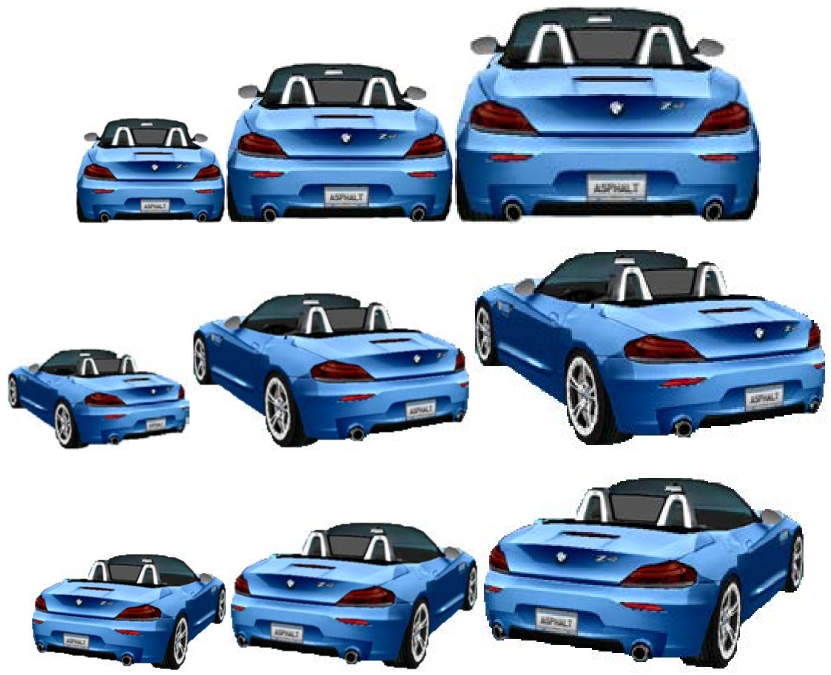
Figure 3 Sprite Merging (the priority ranks from low to high from left to right)

Each layer stands for one or several Sprite picture cluster, Figure 4 demonstrates several Sprite Element that we use.

Three sizes of Cloud (32*32, 48*48, 64*64)



Three sizes of Tree (16*32, 24*48, 32*64)



Three sizes of Car in different directions (32*32, 48*48, 64*64)

Figure 4 Car, Tree and Cloud layer Sprite Elements.

According the Figure 2, the Sprite Control Module's job is to generate the 640*480 pixels RGB value for VGA Controller based on the control signal provided by the Game Logic Module. The Game Logic Module provides the X and Y coordinates for different objects, like trees, clouds, cars, scores, speed, which is also called Sprites. These Sprites are like label pictures store in ROM, and can be pasted to the screen according the X and Y coordinates. We list Sprites we will use in our games.

Sprite	Amount	Pixel Size	Total ROM (Bytes)
Cloud	3	32*32, 48*48, 64*64	22272
Tree	3	16*32, 24*48, 32*64	11136
Car	27	32*32*9, 48*48*9, 64*64*9	200448
Golden Coin	2	64*64, 32*32	15360
Number	10	32*32*10	30720
Label	4	128*64 ("Score") 64*32 ("Speed") 128*32 ("Start") 128*32 ("Finish")	55296
Flag	1	64*64	12288
Total	50	115840	347520

The estimated total ROM size = 348 KB = 0.35 MB , each Pixel is defined by 8-bit R + 8-bit G + 8-bit B.

3.2 Sprite Controller

The diagram for the Sprite Controller Module is demonstrated in Figure 5.

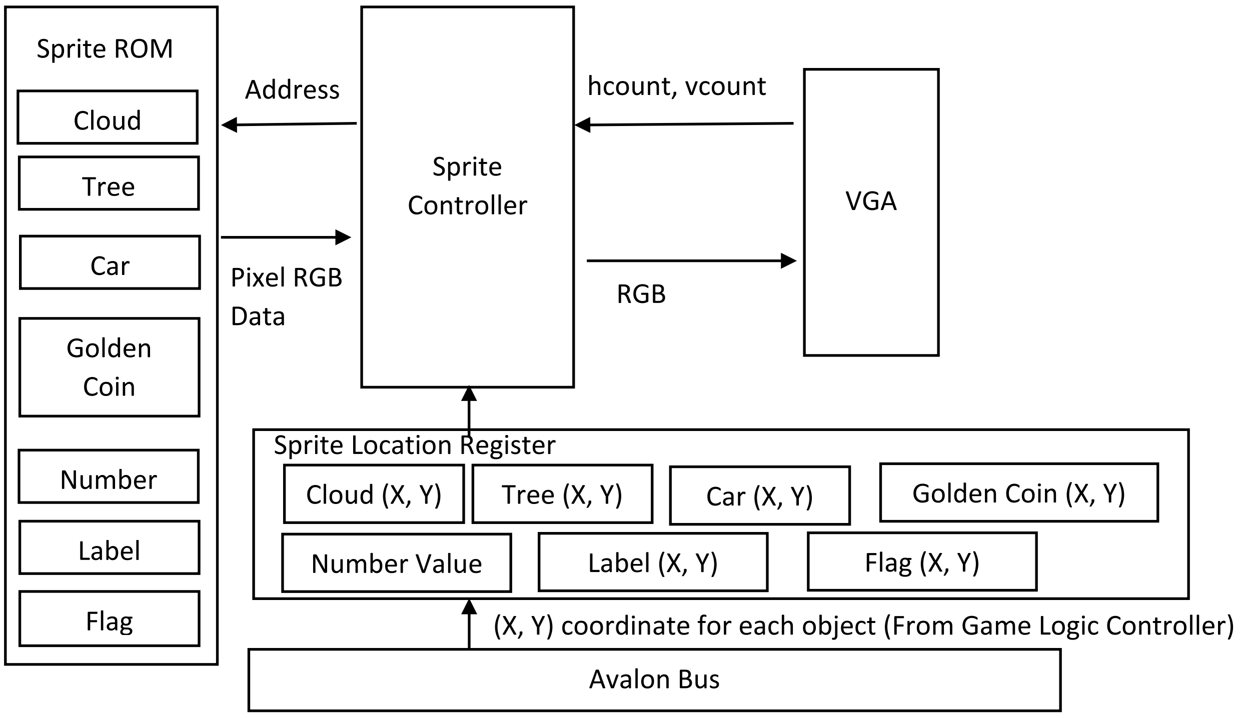


Figure 5 The Sprite Controller Module

The VGA module is the same as Lab 3. The Sprite Controller Module merges the different Sprites to generate the final RGB information for displaying on the screen.

On the Linux side, the software generates the coordinate (X, Y) of different Sprites and passes coordinates to the Avalon Bus through the Kernel Driver. Then these data accompanied with address information in update the register file related to the specific Spite, the Sprite Controller fetches the RGB information from specific Sprite picture ROM, which finally merges all the pictures according to the priority and send to VGA module. The picture is finally showed on the screen. To make the Sprite move, the software just update the (X, Y) coordinate register file continuously.

3.3 Background Generator

The lowest layer is the background picture, which cover the whole 640*480 screen, which is so large if we use ROM to store all the pixels RGB information. Therefore, in our design, the background, which contains road, land, sky and border line, is generated by hardware module using algorithm. That is different from other Sprites whose RGB information is fetched from ROM. We design a special module for Background Generation. The background is showed in Figure 6.

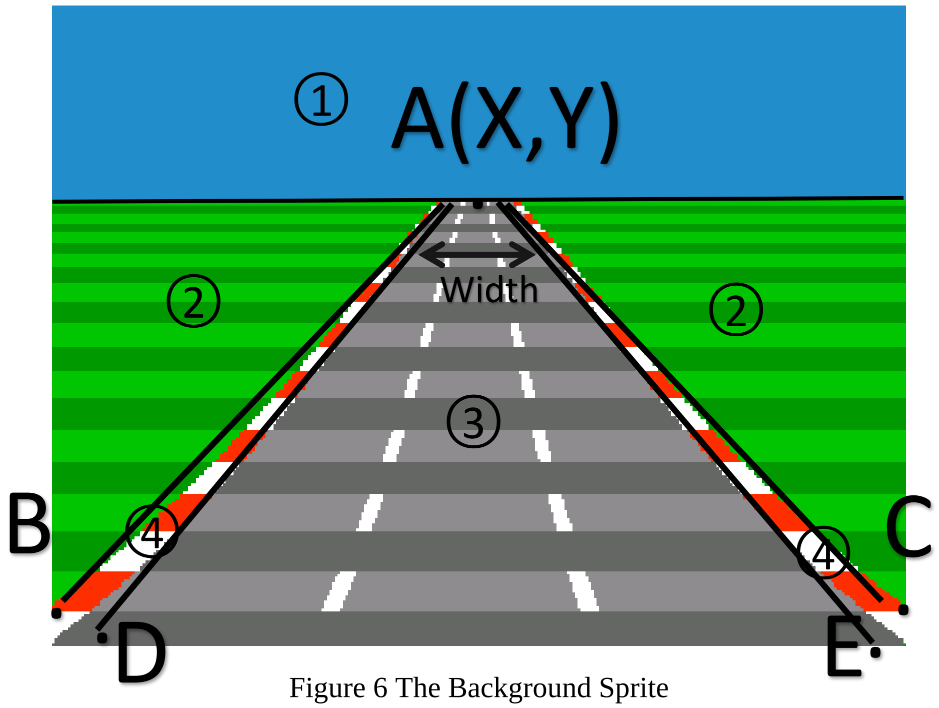


Figure 6 The Background Sprite

The rendering algorithm of this background is based on the coordinate of point A, B, C, D, E and F as well as the Width. Using the geometrical knowledge, we can simply divide the screen into 4 regions, which is demonstrates in Figure 6. Every pixel above Y horizon is set to Blue because it is Sky Region. All pixels within Triangle ABC are set to Gray, because they belong to the Road Region. The rest pixels are set to Green, because they belong to the Grass Region.

3.4 Challenge and Solution

I think the most difficult part in Sprite Module that I have concurred is how to debug in hardware, especially debug the SystemVerilog code. Because Hardware Description Language is totally different from other language that I have learned like C or Java, and the Quartus doesn't have anything like Debug Step by Step, this confuses me a lot at the beginning of programing for this project, I seems lost every time my code doesn't run as I expected, and I have no idea what is going wrong. Then I learn from other experienced classmates and searching in the Internet to find some hardware debug methodologies, I learned that the debug in hardware is not like in the C or Java, it's mainly about the timing analysis, you should use the test bench code to generate the exact test signal to test how your designed module will react to your test signals, and using the waveform function to have a virtual picture for debug. This method help me a lot when I designed the Sprite Module, I use the test bench to debug every small modules and see how these modules response to my clock, check every clock if the output signals is right, it's very tedious but it's also powerful to find what is going wrong in your module. After I have learned this method, it accelerates my progress dramatically.

4 Audio Modules

4.1 Audio Files Preparation

In the audio design, one background music, and two sound effects namely acceleration and crashing are included in this project. All the three audio files are configured with sample rate of 44100 Hz and 16 bits quantization to ensure the sound quality. In order to store the sound files in the on-chip ROM, we have to prepare MIF files with sizes small enough so that the audio files do not consume too much memory of the on-chip ROM.

The first step is find appropriate sound files. The acceleration and crashing sound effects were downloaded in .wav format online. But the original files cannot be implemented directly in this project due to the fact that the files' sizes are much larger than expected. So we modified the .wav format files using MATLAB. We plotted the sample data of the two original files, and picked representative pieces from the plots. Next, we reassembled the selected pieces to create new sample data sets to represent acceleration and crashing sound effects. Then we tested the new sound effects by simulating the looping playback environment in MATLAB, and sounded the files. Finally, we chose the version that performed best in this quality and data size tradeoff. In the next step, we converted the modified sample data into MIF using MATLAB. The codes for generating "accelerate.mif" and "crash.mif" files are in the MATLAB Code section of this report.

The background music was found in the files of the Columbia University in the City of New York Spring 2014 CSEE4840 Embedded System Design Project "NUNY: Ninja University in the City of New York" contributed by Kshitij Bhardwaj, Van Bui, Vinti Vinti, and Kuangya Zhai. The "city.mif" file was proved to be in good quality and suitable size for this project, so we quoted this file and renamed it "background.mif" in this project.

Finally, the well prepared background music, acceleration, and crashing sound effects files were loaded to the on-chip ROM, taking 65536 words (word=16bits) memory in total.

4.2 Audio Controller

The Sockit board in the lab provides the Analog Devices SSM2603 Audio Codec (Encoder/Decoder). This chip supports microphone-in, line-in, and line-out ports, with a sample rate adjustable from 8kHz to 96kHz. The SSM2603 is controlled via a serial I^2C bus interface, which is connected to pins on the CycloneV SoC FPGA. The schematic diagram of the audio circuitry is shown in the figure below.

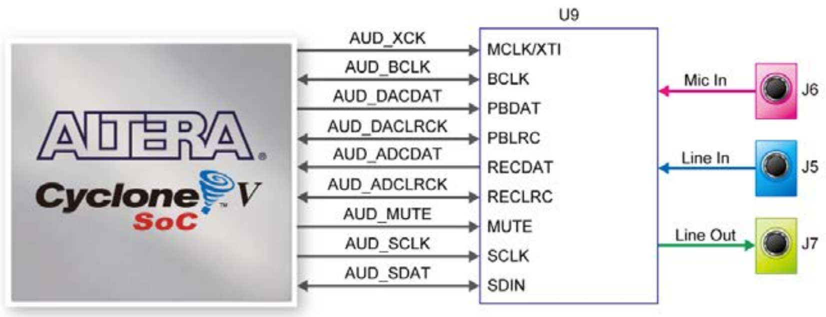


Figure 7 Audio Circuitry in Sockit Board

In this project, the Audio Controller is mainly consisted of four components, namely Audio ROM blocks, Audio Effect block, Audio Codec Interface, and Audio Codec Configuration Interface. The figure below is the function block diagram of the Audio Controller.

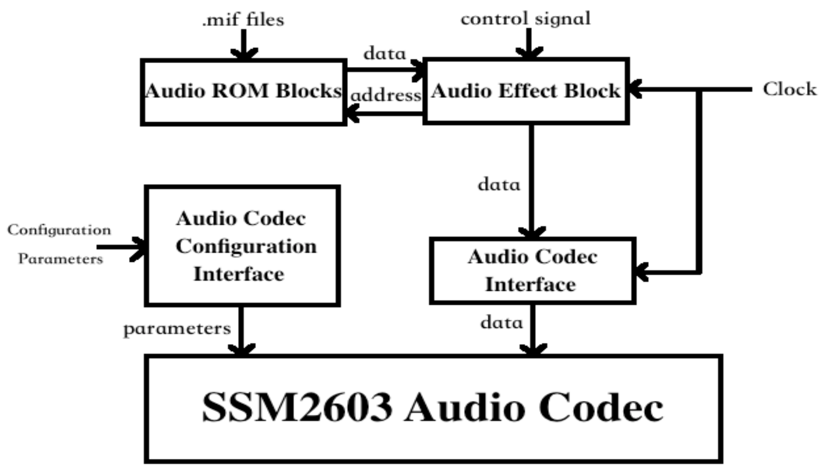


Figure 8 Audio Controller Function Block Diagram

1. Audio ROM blocks. The audio data are well prepared in advance to take as little on-chip ROM space as possible, while preserving the sound quality. The acceleration sound

(“accelerate.mif”), crashing sound (“crash.mif”), and background music (“background.mif”) files are used to create ROM audio data blocks using MegaWizard Plug-in Manager. The background music takes 32768 words (word=16bits) memory, while each of the acceleration and crashing sound effects takes 16384 words (word=16bits) memory. In total, 65536 16-bit ROM space are taken for the audio data.

2. Audio Effect block. This block operates at 50 MHz clock rate and receives the control signals passed from the software. When a valid control signal is passed to this block, the corresponding audio data will be fetched from the Audio Data blocks, and then passed to the SSM2603 Audio Codec through Audio Codec Interface.
3. Audio Codec Interface. This interface operates at 50 MHz clock rate. It sends the audio data to the SSM2603 Audio Codec using shift registers.
4. Audio Codec Configuration Interface. This interface is used to configure the SSM2603 Audio Codec parameters, such as 44100Hz sampling rate and so on, using the I^2C protocol. The two wires in the I^2C protocol are labeled SDAT and SCLK for Serial Data and Serial Clock respectively. In I^2C, data is sent a bit at a time over the SDAT wire, with the separation between bits determined by clock cycles on the SCLK wire. In this project, the FPGA is the master and the audio codec in slave mode.

When the game is on, the software pass control signal “0” to the Audio Controller, and the background music will be played. When acceleration happens in the game, control signal “1” will be passed to the Audio Controller, and the acceleration sound effect will take place repeatedly until the acceleration period ends. When there is a crash on either the boundaries or the other competitors, control signal “2” will be passed to the Audio Controller, and the crashing sound effect will take place. Each sound effect is played individually in this project, and we do not support concurrent sounding in this game.

4.3 Problems Encountered and Solutions

The most severe problem we encountered during the audio design was to find proper audio files that can fit into the on-chip ROM, while leaving enough storage for the rest of the project. Initially, to play high quality audios, we loaded large audio files that took over 30% of the on-chip ROM, such design is too costly to be implemented. Then we decided to search for small audio files that are between one to two seconds long as alternatives. However, the files were still larger than expected. Finally, we decided to customize the audio files using the method described in the “Audio Files Preparation” section of this report. And it turned out that we were able to load the properly modified audio files into the on-chip ROM and play the sound effects that meet the design requirements.

The second problem we encountered during the audio design was converting the .wav files into .mif files correctly. Initially, we found MIF file samples online, and used MATLAB to convert .wav format files into the MIF. However, when we loaded the files on chip and sound it, there were nothing but noises. After several failed testings resulted from using other audio files, we decided to reverse engineer the “city.mif” file from the Columbia University in the City of New York Spring 2014 CSEE4840 Embedded System Design Project “NUNY: Ninja University in the City of New York" contributed by Kshitij Bhardwaj, Van Bui, Vinti Vinti, and Kuangya Zhai. Then we realized that the issue was that MIF files require unsigned integer data type instead of signed integer. So we corrected the error, and finally solved this problem.

4.4 Future Improvements

Background music and sound effects can be played concurrently. A circular queue can be designed to handle the background music only. And a mixer can be designed to combine the sound effects in circular queue and FIFO, and then send the mixed data to SSM2603 Audio Codec in order to play the background music and the sound effects at the same time.

5 Game Logic Module (Software)

5.1 Game Play Flowchart

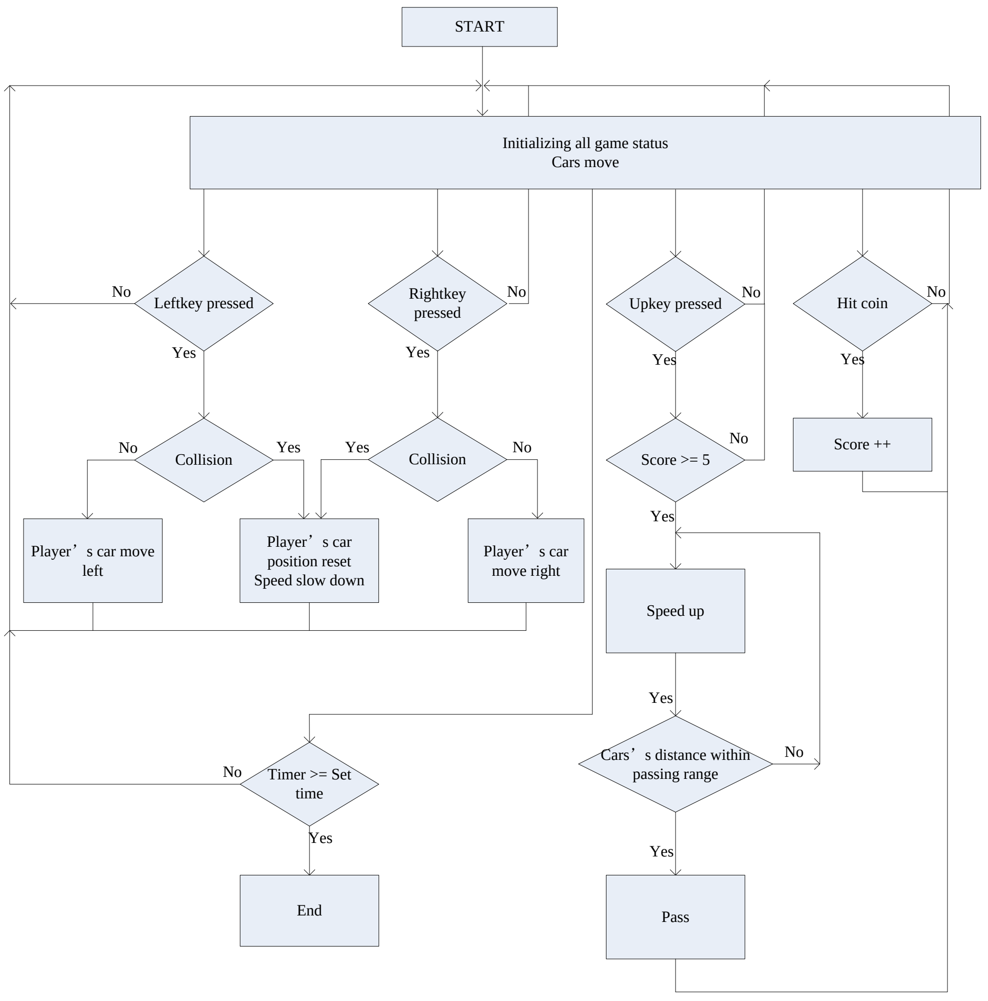


Figure 9 Software Flowchart

When the game starts, a traffic light will change from red to green signaling the start of the game and all cars will start going forward. The other two opponent cars are set to have higher speeds than the player's car and one of them is the fastest. The detection of the USB keyboard is in a separate thread and the system will be detecting if a key is pressed all the time. When the left key is pressed, the player's car will be moving towards left. When the right key is pressed, the player's car will be moving towards right. If the player's car collides with the road or other cars, its position will be reset to the middle of the road and its speed will be slowed down. If the player's car hits a coin, the score will be added up. When the score is larger than or equal to 5 and the up key is pressed, the player's car will speed up. If the distance between the player's car and the opponent cars is within the passing range and the player's car is still speeding up, the player's car will be passing the opponent cars. If the player's car is not speeding up and the distance is within passing range, the player's car will be passed by the opponent cars.

5.2 Hardware – Software Interface

Each sprite stored in the ROM will have a center position with X and Y coordinates. The function write_register is used to communicate between hardware and software which takes an index of a sprite and its center location x and y as inputs. What we need to do in the software is to use write_register to call sprites and put it to the positions we want.

5.3 Game Play Algorithm

- Trees, clouds and road moving

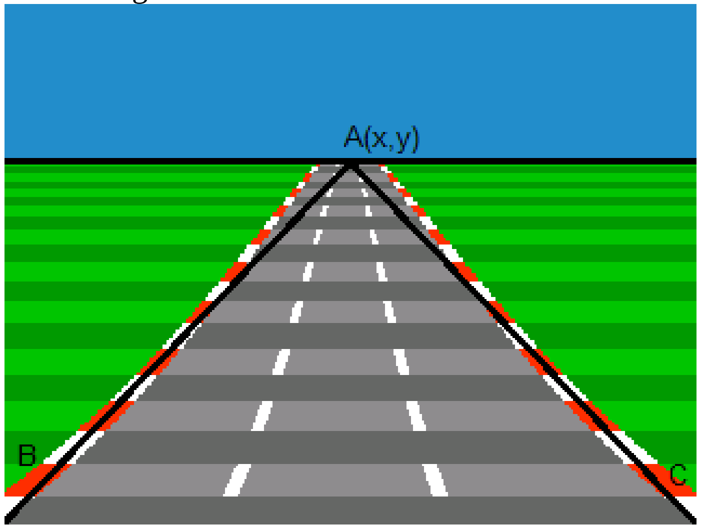


Figure 10 Coordinates of three important points

In our game environment logic, the location and movement of environmental elements are all have a relationship with the coordinates of central point 'A' which is given by the hardware. By identifying vector AB and AC, we could easily put trees and clouds in right location. The unit vector of AB and AC are as follows:

$$ab(x,y) = \left(\frac{A(x)-B(x)}{\sqrt{(A(x)-B(x))^2+(A(y)-B(y))^2}}, \frac{A(y)-B(y)}{\sqrt{(A(x)-B(x))^2+(A(y)-B(y))^2}}\right)$$

$ac(x,y) = (\frac{A(x)-C(x)}{\sqrt{(A(x)-C(x))^2+(A(y)-C(y))^2}}, \frac{A(y)-C(y)}{\sqrt{(A(x)-C(x))^2+(A(y)-C(y))^2}})$ Then by multiplying a proper constant, the increment of trees movement could be calculated dynamically during each loop in codes.

$$Tree\_left(x) = Tree\_left(x) - const \times ab(x)$$
$$Tree\_left(y) = Tree\_left(y) + const \times ab(y)$$
$$Tree\_right(x) = Tree\_rignt(x) + const \times ac(x)$$
$$Tree\_right(y) = Tree\_right(y) + const \times ac(y)$$

When encountered curve roads, the position and movement of trees could also be calculated dynamically according to the movement of central point 'A'.

- Collision with road, coins and cars

In our game, when player's car hit the road margin or other cars, it will be set back to initial position and stay twinkling in place for a while as penalty. The judgment for collision is by comparing the distance of the two central points' coordinates.

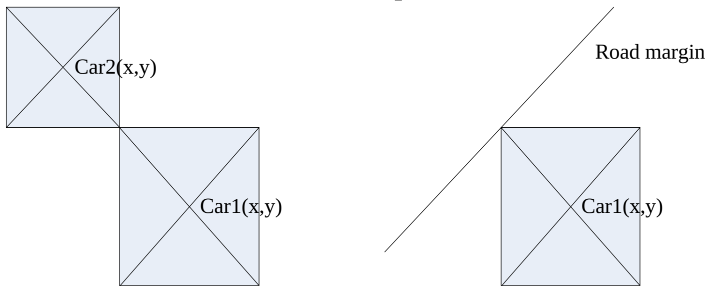


Figure 11 Distance judgment

As for coins, when cars hit the coins, it will get a reward for speedup if the accumulated hit number of coins reaches five.

- Surpass

To be able to know when the player's car is passing the opponent cars or when it is being passed by the opponent cars, we keep track of how far each car has travelled by three variables "car1_dist", "car2_dist" and "car3_dist". These three variables keep increasing based on their different speeds – "car1_speed", "car2_speed" and "car3_speed". In this way we are able to know two cars' distance by subtracting two distance variables. And we set a passing range so that when two cars' distance is within the range, there is a passing event. The player's car can only be passing the opponent cars during speed up and their distance is within the passing range, where its speed is faster than the opponent cars' speeds. And the opponent cars can pass the player's car if the player's car is not speeding up and their distance is within the passing range. During the process of speeding up, all the cars' speeds are set so that the player's car's speed is the fastest. If a collision happens, the player's car's speed will be set to zero during the process of reset. The rank will be displayed based on their distances travelled.

- Numbers displayed (score, time, rank, speed)

We wrote three functions – score_display, time_display and speed_display – to display corresponding numbers on the screen. The score_display function takes the variable "car_score" as input and output the correct score on the screen. If the number is less than 10, it will be put directly on the units' digit. If the number is larger than and equal to 10 and smaller than 100, it will be divided by 10. The quotient will be put on the tens' digit and the remainder will be put on the units' digit. If the number is larger than and equal to 100 and smaller than 1000, it will first be divided by 100 and the quotient is put on the hundreds' digit. Then the quotient is divided by 10 and its quotient is put on the tens' digit and its remainder is put on the units' digit. The other functions use similar algorithm to display numbers.

- Timer
  We used a variable "gametimer" to keep track of the time elapsed. We increase "gametimer" by an increment of 1 with frame rate and then divide it by the number of how many frames per second to get the correct time.

5.4 Future Work Suggestions

1. The game modes could be set in different difficulty levels for player to choose.

2. We may add joystick control in the game to make our game fancier.

3. The curve road part in codes could be added in a new thread in order to make the curve appears more natural and smoothly.

4. We may possibly add more cars.

5.The score can be replaced by a nitrogen bar and the coins can be replaced by nitrogen cans.

6. We may possibly add drift effect for player's car.

7. The game should be able to pause.

5.5 Problems Encountered and Solutions:

1. At the beginning when discussed the game logic, we thought the movement of environmental elements was difficult to control. Because when curve road appears, the trees must move in different directions dynamically with the road direction. Then we came up with a solution which is to set two vectors that could be controlled by only one point's coordinates. This method simplified our game logic and appears to be really helpful and elegant.

2. We had a hard time trying to get the keyboard to work. We kept getting the same error from the system of not being able to find the libusb file. We found out that in our lab3 file, which was the file we were using, the libusb directory was actually missing. Then we copied the directory from lab2 file to lab3. In this way we were able to get rid of the previous error but we got a new error. Then we found out that the error occurred because lab2 and lab3 used different versions.

After we move everything from lab3 to lab2, the problem got fixed and we were finally able to use the keyboard.

3. We were not able to make the keyboard function work in the main function. We found out that using keyboard functions in the main function would always interfere with our main game logic. Thus we used a thread and put the whole keyboard function in the thread. And we opened the thread at the beginning of our main function and end the thread at the end of our main function.

4. We were not able to make the car move continuously by holding the key. In the libusb_interrupt_transfer function, the last variable was set to 0. We tried to increase this number and we found out that we were able to detect a continuous hold of a key. This variable was called timeout (in milliseconds) that is the time the function should wait before giving up due to no response received. You put 0 for an unlimited timeout.

5. We were not able to define an accurate passing range. We were using three variables named "car1_dist", car2_dist" and car3_dist" to keep track of how far each car has travelled and the distances between each other. However, these variables were increasing very fast because they were increasing along with the frame rate. The distances between them were increasing very fast as well. Thus it was very hard to define a range where a passing event should happen. We then divide each distance variable with an integer and were able to make them increase slowly and get a reasonable passing range.

6 Conclusion

From this project, we improved our skills in programming both hardware language and software language.

In the Sprite Display Module design, we have gone through programing, testing, modifying and assembling. We adopt Top-Down Design Methodology, divide the big project into several small task and modules, design general port interface. By this way, we improve our progress and make our project easy to locate where the problems are because we use small modules to limit the bug region. We believe the most valuable lesson we have learned in this project is how to debug in hardware and how to locate the bug. The bugs and problems we encountered through this project help us to enhance our knowledge of hardware design.

During the audio design in this project, we have learnt that the physical on-chip ROM limit should always be kept in mind during hardware design, and all the data designed to be loaded on board should be modified to the correct format beforehand. When preparing the proper audio file, rather than repeatedly loading different testing audio files into the system and compiling the whole program every time we change the audio files, we can simulate the looping playback environment using MATLAB, and then load the qualified audio file version on board for further testing. The simulation strategy is much more efficient. Before integrating the audio module into the whole game logic program, we should test the audio module by sending the expected signals to examine whether the audio system can output correctly. Following such procedure, we can save a lot of debugging, because we have divided the whole design and testing procedure into multiple subsections which make it easier and faster to locate the error locations.

We used the software to implement the game logic and hardware to display graphics and audio sounds and communicated each other with software-hardware interface. During this procedure, we've encountered a lot of problems in implementing the game logic and keyboard control but we were finally able to solve all of them though we still have a lot to improve.

7 Major Codes

7.1 Hardware Part

7.1.1 ROM MIF File Converter (MATLAB)

```
clear all; close all;
row = 32;
colum = 32;
A = imread('light_green2.png');
figure(1)
imshow(A);
B = imresize(A, [row, colum]);
B1 = B(:,:,1);
B2 = B(:,:,2);
B3 = B(:,:,3);
figure(2)
imshow(B);


C = zeros(1,row*colum);

for i = 1:row
    for j = 1:colum
       C(colum*(i-1)+j) = bitshift(int32(B1(i,j)),16) +
bitshift(int32(B2(i,j)),8) + int32(B3(i,j));
%         C(64*(i-1)+j) = B1(i,j)*2^(16) + B2(i,j)*2^(8) + B3(i,j);
    end
end


depth =row*colum; %???????

widths = 24;%?????24?


fidc = fopen('light_green.mif','wt');

fprintf(fidc, 'WIDTH = %d;\n',widths);

fprintf(fidc , 'DEPTH = %d;\n',depth);

fprintf(fidc, 'ADDRESS_RADIX = UNS;\n');

fprintf(fidc,'DATA_RADIX = UNS;\n');
fprintf(fidc,'CONTENT BEGIN\n');
for x = 1 : depth
    fprintf(fidc,'%d:%d;\n',x-1,C(x));
end
    fprintf(fidc, 'END;');
fclose(fidc);
```

7.1.2 VGA_Sprite.sv

```
module VGA_BALL(input logic       clk,
            input logic     reset,
            input logic [7:0]  writedata,
            input logic     write,
            input                 chipselect,
            input logic [7:0]  address,


                    inout   AUD_ADCLRCK, AUD_DACLRCK,AUD_BCLK, AUD_I2C_SDAT,
                    input   AUD_ADCDAT,
                    output  AUD_DACDAT, AUD_XCK, AUD_I2C_SCLK, AUD_MUTE,


            output logic [7:0] VGA_R, VGA_G, VGA_B,
            output logic            VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
            output logic            VGA_SYNC_n);

      parameter HEIGHT      = 10'd 480,
     WIDTH = 11'd 640;
```

```
	logic [7:0] VGA_R_TEMP, VGA_G_TEMP, VGA_B_TEMP;
	logic [15:0] centerX, centerY, road_width;
	logic [9:0] posX;
	logic [8:0] posY;
	logic [15:0] hc, vc;
//	logic [31:0] is_road1, is_road2, is_road3, is_road4;
	integer is_road1, is_road2, is_road3, is_road4, is_road5, is_road6, is_road7, is_road8;
	logic color_flag, road_flag, is_red;
//	logic [7:0] road_section;

// Audio Control
	 logic [3:0]		KEY;
	 logic [3:0]		SW;
	 logic [3:0]		LED;

// Car1_Ahead parameter
	logic [15:0] car1_ahead_x, car1_ahead_y;
	logic [7:0] car1_ahead_r, car1_ahead_g, car1_ahead_b;
	logic car1_ahead_valid;
// Car1_Left parameter
	logic [15:0] car1_left_x, car1_left_y;
	logic [7:0] car1_left_r, car1_left_g, car1_left_b;
	logic car1_left_valid;
// Car1_Right parameter
	logic [15:0] car1_right_x, car1_right_y;
	logic [7:0] car1_right_r, car1_right_g, car1_right_b;
	logic car1_right_valid;

// Car1_Medium_Ahead parameter
```

```
            logic [15:0] car1_medium_ahead_x, car1_medium_ahead_y;
            logic [7:0] car1_medium_ahead_r, car1_medium_ahead_g, car1_medium_ahead_b;
            logic car1_medium_ahead_valid;
// Car1_Medium_Left parameter
            logic [15:0] car1_medium_left_x, car1_medium_left_y;
            logic [7:0] car1_medium_left_r, car1_medium_left_g, car1_medium_left_b;
            logic car1_medium_left_valid;
// Car1__Medium_Right parameter
            logic [15:0] car1_medium_right_x, car1_medium_right_y;
            logic [7:0] car1_medium_right_r, car1_medium_right_g, car1_medium_right_b;
            logic car1_medium_right_valid;

// Car1_Small_Ahead parameter
            logic [15:0] car1_small_ahead_x, car1_small_ahead_y;
            logic [7:0] car1_small_ahead_r, car1_small_ahead_g, car1_small_ahead_b;
            logic car1_small_ahead_valid;
// Car1_Small_Left parameter
            logic [15:0] car1_small_left_x, car1_small_left_y;
            logic [7:0] car1_small_left_r, car1_small_left_g, car1_small_left_b;
            logic car1_small_left_valid;
// Car1_Small_Right parameter
            logic [15:0] car1_small_right_x, car1_small_right_y;
            logic [7:0] car1_small_right_r, car1_small_right_g, car1_small_right_b;
            logic car1_small_right_valid;

// Car2_Ahead parameter
            logic [15:0] car2_ahead_x, car2_ahead_y;
            logic [7:0] car2_ahead_r, car2_ahead_g, car2_ahead_b;
            logic car2_ahead_valid;
```

```
// Car2_Left parameter
        logic [15:0] car2_left_x, car2_left_y;
        logic [7:0] car2_left_r, car2_left_g, car2_left_b;
        logic car2_left_valid;
// Car2_Right parameter
        logic [15:0] car2_right_x, car2_right_y;
        logic [7:0] car2_right_r, car2_right_g, car2_right_b;
        logic car2_right_valid;

// Car2_Medium_Ahead parameter
        logic [15:0] car2_medium_ahead_x, car2_medium_ahead_y;
        logic [7:0] car2_medium_ahead_r, car2_medium_ahead_g, car2_medium_ahead_b;
        logic car2_medium_ahead_valid;
// Car2_Medium_Left parameter
        logic [15:0] car2_medium_left_x, car2_medium_left_y;
        logic [7:0] car2_medium_left_r, car2_medium_left_g, car2_medium_left_b;
        logic car2_medium_left_valid;
// Car2__Medium_Right parameter
        logic [15:0] car2_medium_right_x, car2_medium_right_y;
        logic [7:0] car2_medium_right_r, car2_medium_right_g, car2_medium_right_b;
        logic car2_medium_right_valid;

// Car2_Small_Ahead parameter
        logic [15:0] car2_small_ahead_x, car2_small_ahead_y;
        logic [7:0] car2_small_ahead_r, car2_small_ahead_g, car2_small_ahead_b;
        logic car2_small_ahead_valid;
// Car2_Small_Left parameter
        logic [15:0] car2_small_left_x, car2_small_left_y;
        logic [7:0] car2_small_left_r, car2_small_left_g, car2_small_left_b;
        logic car2_small_left_valid;
// Car2_Small_Right parameter
        logic [15:0] car2_small_right_x, car2_small_right_y;
        logic [7:0] car2_small_right_r, car2_small_right_g, car2_small_right_b;
        logic car2_small_right_valid;


// Car3_Ahead parameter
        logic [15:0] car3_ahead_x, car3_ahead_y;
        logic [7:0] car3_ahead_r, car3_ahead_g, car3_ahead_b;
        logic car3_ahead_valid;
// Car3_Left parameter
        logic [15:0] car3_left_x, car3_left_y;
        logic [7:0] car3_left_r, car3_left_g, car3_left_b;
        logic car3_left_valid;
// Car3_Right parameter
        logic [15:0] car3_right_x, car3_right_y;
        logic [7:0] car3_right_r, car3_right_g, car3_right_b;
        logic car3_right_valid;


// Car3_Medium_Ahead parameter
        logic [15:0] car3_medium_ahead_x, car3_medium_ahead_y;
        logic [7:0] car3_medium_ahead_r, car3_medium_ahead_g, car3_medium_ahead_b;
        logic car3_medium_ahead_valid;
// Car3_Medium_Left parameter
        logic [15:0] car3_medium_left_x, car3_medium_left_y;
        logic [7:0] car3_medium_left_r, car3_medium_left_g, car3_medium_left_b;
        logic car3_medium_left_valid;
// Car3__Medium_Right parameter
        logic [15:0] car3_medium_right_x, car3_medium_right_y;
        logic [7:0] car3_medium_right_r, car3_medium_right_g, car3_medium_right_b;
        logic car3_medium_right_valid;

// Car3_Small_Ahead parameter
        logic [15:0] car3_small_ahead_x, car3_small_ahead_y;
        logic [7:0] car3_small_ahead_r, car3_small_ahead_g, car3_small_ahead_b;
        logic car3_small_ahead_valid;
// Car3_Small_Left parameter
        logic [15:0] car3_small_left_x, car3_small_left_y;
        logic [7:0] car3_small_left_r, car3_small_left_g, car3_small_left_b;
        logic car3_small_left_valid;
// Car3_Small_Right parameter
        logic [15:0] car3_small_right_x, car3_small_right_y;
        logic [7:0] car3_small_right_r, car3_small_right_g, car3_small_right_b;
        logic car3_small_right_valid;

// Tree1_Big parameter
        logic [15:0] tree1_x, tree1_y, tree4_x, tree4_y;
        logic [7:0] tree1_r, tree1_g, tree1_b;
        logic tree1_valid;
// Tree_Medium parameter
        logic [15:0] tree2_x, tree2_y, tree5_x, tree5_y;
        logic [7:0] tree2_r, tree2_g, tree2_b;
        logic tree2_valid;
// Tree_Small parameter
        logic [15:0] tree3_x, tree3_y, tree6_x, tree6_y;
        logic [7:0] tree3_r, tree3_g, tree3_b;
        logic tree3_valid;
```

```
// Cloud_Big parameter
            logic [15:0] cloud1_x, cloud1_y, cloud3_x, cloud3_y;
            logic [7:0] cloud1_r, cloud1_g, cloud1_b;
            logic cloud1_valid;
// Cloud_Small parameter
            logic [15:0] cloud2_x, cloud2_y, cloud4_x, cloud4_y;
            logic [7:0] cloud2_r, cloud2_g, cloud2_b;
            logic cloud2_valid;

//Light parameter
            logic [15:0] light_x, light_y;
            logic [7:0] light_r, light_g, light_b;
            logic [1:0] light_signal;
            logic light_valid;
//Numbers parameters
            logic [3:0] pos, pos_total;
            logic [3:0] min_ten, min_uni, sec_ten, sec_uni;
            logic [3:0] speed_hund, speed_ten, speed_uni;
            logic [3:0] score_hund, score_ten, score_uni;
            logic [7:0] numbers_r, numbers_g, numbers_b;
            logic numbers_valid;

//Finish Flag parameter
            logic [15:0] flag_x, flag_y;
            logic [7:0]  flag_r, flag_g, flag_b;
            logic flag_valid;

//Coin  big parameter
            logic [15:0] coin1_x, coin1_y;
```

```
	logic [7:0]		coin1_r, coin1_g, coin1_b;
	logic coin1_valid;

//Coin small parameter
	logic [15:0] coin2_x, coin2_y;
	logic [7:0]		coin2_r, coin2_g, coin2_b;
	logic coin2_valid;

//Score label parameter
	logic [15:0] score_label_x, score_label_y;
	logic [7:0] score_label_r, score_label_g, score_label_b;
	logic score_label_valid;

//mph label parameter
	logic [15:0] mph_label_x, mph_label_y;
	logic [7:0] mph_label_r, mph_label_g, mph_label_b;
	logic mph_label_valid;

//left_arrow_small parameter
	logic [15:0] left_arrow_small_x, left_arrow_small_y;
	logic [7:0] left_arrow_small_r, left_arrow_small_g, left_arrow_small_b;
	logic left_arrow_small_valid;

//left_arrow_big parameter
	logic [15:0] left_arrow_big_x, left_arrow_big_y;
	logic [7:0] left_arrow_big_r, left_arrow_big_g, left_arrow_big_b;
	logic left_arrow_big_valid;

//right_arrow_small parameter
```

```
            logic [15:0] right_arrow_small_x, right_arrow_small_y;

            logic [7:0] right_arrow_small_r, right_arrow_small_g, right_arrow_small_b;

            logic right_arrow_small_valid;


//right_arrow_big parameter

            logic [15:0] right_arrow_big_x, right_arrow_big_y;

            logic [7:0] right_arrow_big_r, right_arrow_big_g, right_arrow_big_b;

            logic right_arrow_big_valid;



  VGA_BALL_Emulator ball_emulator(.clk50(clk), .*);


// audio module

            Audio_Top audio_top(.OSC_50_B8A(clk), .*);

//          Car1_Sprite car1_sprite(.*);

            Car1_Ahead_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car1_ahead_x), .car1_y(car1_ahead_y), .car
1_r(car1_ahead_r), .car1_g(car1_ahead_g), .car1_b(car1_ahead_b), .car1_valid(car1_ahead_valid));

            Car1_Left_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car1_left_x), .car1_y(car1_left_y), .car1_r(car1
_left_r), .car1_g(car1_left_g), .car1_b(car1_left_b), .car1_valid(car1_left_valid));

            Car1_Right_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car1_right_x), .car1_y(car1_right_y), .car1_r(
car1_right_r), .car1_g(car1_right_g), .car1_b(car1_right_b), .car1_valid(car1_right_valid));


            Car2_Ahead_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car2_ahead_x), .car1_y(car2_ahead_y), .car
1_r(car2_ahead_r), .car1_g(car2_ahead_g), .car1_b(car2_ahead_b), .car1_valid(car2_ahead_valid));

            Car2_Left_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car2_left_x), .car1_y(car2_left_y), .car1_r(car2
_left_r), .car1_g(car2_left_g), .car1_b(car2_left_b), .car1_valid(car2_left_valid));

            Car2_Right_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car2_right_x), .car1_y(car2_right_y), .car1_r(
car2_right_r), .car1_g(car2_right_g), .car1_b(car2_right_b), .car1_valid(car2_right_valid));


            Car3_Ahead_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car3_ahead_x), .car1_y(car3_ahead_y), .car
1_r(car3_ahead_r), .car1_g(car3_ahead_g), .car1_b(car3_ahead_b), .car1_valid(car3_ahead_valid));
```

```
        Car3_Left_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car3_left_x), .car1_y(car3_left_y), .car1_r(car3_left_r), .car1_g(car3_left_g), .car1_b(car3_left_b), .car1_valid(car3_left_valid));

        Car3_Right_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car3_right_x), .car1_y(car3_right_y), .car1_r(car3_right_r), .car1_g(car3_right_g), .car1_b(car3_right_b), .car1_valid(car3_right_valid));

//////////////////////////////////////////////

        Car1_Medium_Ahead_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car1_medium_ahead_x), .car1_y(car1_medium_ahead_y), .car1_r(car1_medium_ahead_r), .car1_g(car1_medium_ahead_g), .car1_b(car1_medium_ahead_b), .car1_valid(car1_medium_ahead_valid));

        Car1_Medium_Left_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car1_medium_left_x), .car1_y(car1_medium_left_y), .car1_r(car1_medium_left_r), .car1_g(car1_medium_left_g), .car1_b(car1_medium_left_b), .car1_valid(car1_medium_left_valid));

        Car1_Medium_Right_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car1_medium_right_x), .car1_y(car1_medium_right_y), .car1_r(car1_medium_right_r), .car1_g(car1_medium_right_g), .car1_b(car1_medium_right_b), .car1_valid(car1_medium_right_valid));


        Car2_Medium_Ahead_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car2_medium_ahead_x), .car1_y(car2_medium_ahead_y), .car1_r(car2_medium_ahead_r), .car1_g(car2_medium_ahead_g), .car1_b(car2_medium_ahead_b), .car1_valid(car2_medium_ahead_valid));

        Car2_Medium_Left_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car2_medium_left_x), .car1_y(car2_medium_left_y), .car1_r(car2_medium_left_r), .car1_g(car2_medium_left_g), .car1_b(car2_medium_left_b), .car1_valid(car2_medium_left_valid));

        Car2_Medium_Right_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car2_medium_right_x), .car1_y(car2_medium_right_y), .car1_r(car2_medium_right_r), .car1_g(car2_medium_right_g), .car1_b(car2_medium_right_b), .car1_valid(car2_medium_right_valid));


        Car3_Medium_Ahead_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car3_medium_ahead_x), .car1_y(car3_medium_ahead_y), .car1_r(car3_medium_ahead_r), .car1_g(car3_medium_ahead_g), .car1_b(car3_medium_ahead_b), .car1_valid(car3_medium_ahead_valid));

        Car3_Medium_Left_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car3_medium_left_x), .car1_y(car3_medium_left_y), .car1_r(car3_medium_left_r), .car1_g(car3_medium_left_g), .car1_b(car3_medium_left_b), .car1_valid(car3_medium_left_valid));

        Car3_Medium_Right_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car3_medium_right_x), .car1_y(car3_medium_right_y), .car1_r(car3_medium_right_r), .car1_g(car3_medium_right_g), .car1_b(car3_medium_right_b), .car1_valid(car3_medium_right_valid));

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

```
        Car1_Small_Ahead_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car1_small_ahead_x), .car1_y(car1_s
mall_ahead_y), .car1_r(car1_small_ahead_r), .car1_g(car1_small_ahead_g), .car1_b(car1_small_ahead_
b), .car1_valid(car1_small_ahead_valid));

        Car1_Small_Left_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car1_small_left_x), .car1_y(car1_small_l
eft_y), .car1_r(car1_small_left_r), .car1_g(car1_small_left_g), .car1_b(car1_small_left_b), .car1_valid(car
1_small_left_valid));

        Car1_Small_Right_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car1_small_right_x), .car1_y(car1_sma
ll_right_y), .car1_r(car1_small_right_r), .car1_g(car1_small_right_g), .car1_b(car1_small_right_b), .car1_
valid(car1_small_right_valid));


        Car2_Small_Ahead_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car2_small_ahead_x), .car1_y(car2_s
mall_ahead_y), .car1_r(car2_small_ahead_r), .car1_g(car2_small_ahead_g), .car1_b(car2_small_ahead_
b), .car1_valid(car2_small_ahead_valid));

        Car2_Small_Left_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car2_small_left_x), .car1_y(car2_small_l
eft_y), .car1_r(car2_small_left_r), .car1_g(car2_small_left_g), .car1_b(car2_small_left_b), .car1_valid(car
2_small_left_valid));

        Car2_Small_Right_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car2_small_right_x), .car1_y(car2_sma
ll_right_y), .car1_r(car2_small_right_r), .car1_g(car2_small_right_g), .car1_b(car2_small_right_b), .car1_
valid(car2_small_right_valid));


        Car3_Small_Ahead_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car3_small_ahead_x), .car1_y(car3_s
mall_ahead_y), .car1_r(car3_small_ahead_r), .car1_g(car3_small_ahead_g), .car1_b(car3_small_ahead_
b), .car1_valid(car3_small_ahead_valid));

        Car3_Small_Left_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car3_small_left_x), .car1_y(car3_small_l
eft_y), .car1_r(car3_small_left_r), .car1_g(car3_small_left_g), .car1_b(car3_small_left_b), .car1_valid(car
3_small_left_valid));

        Car3_Small_Right_Sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(car3_small_right_x), .car1_y(car3_sma
ll_right_y), .car1_r(car3_small_right_r), .car1_g(car3_small_right_g), .car1_b(car3_small_right_b), .car1_
valid(car3_small_right_valid));

/////////////////////////

        Tree1_Sprite
tree1_sprite(.clk(clk), .hc(hc), .vc(vc), .tree1_x(tree1_x), .tree1_y(tree1_y), .tree2_x(tree4_x), .tree2_y(tr
ee4_y), .tree1_r(tree1_r), .tree1_g(tree1_g), .tree1_b(tree1_b), .tree_valid(tree1_valid));

        Tree_Medium_Sprite
tree2_sprite(.clk(clk), .hc(hc), .vc(vc), .tree1_x(tree2_x), .tree1_y(tree2_y), .tree2_x(tree5_x), .tree2_y(tr
ee5_y), .tree1_r(tree2_r), .tree1_g(tree2_g), .tree1_b(tree2_b), .tree_valid(tree2_valid));
```

```
	Tree_Small_Sprite
tree3_sprite(.clk(clk), .hc(hc), .vc(vc), .tree1_x(tree3_x), .tree1_y(tree3_y), .tree2_x(tree6_x), .tree2_y(tr
ee6_y), .tree1_r(tree3_r), .tree1_g(tree3_g), .tree1_b(tree3_b), .tree_valid(tree3_valid));


	Cloud_Big_Sprite
cloud_big_sprite(.clk(clk), .hc(hc), .vc(vc), .tree1_x(cloud1_x), .tree1_y(cloud1_y),.tree2_x(cloud3_x), .tr
ee2_y(cloud3_y), .tree1_r(cloud1_r), .tree1_g(cloud1_g), .tree1_b(cloud1_b), .tree_valid(cloud1_valid));

	Cloud_Small_Sprite
cloud_small_sprite(.clk(clk), .hc(hc), .vc(vc), .tree1_x(cloud2_x), .tree1_y(cloud2_y), .tree2_x(cloud4_x), .
tree2_y(cloud4_y), .tree1_r(cloud2_r), .tree1_g(cloud2_g), .tree1_b(cloud2_b), .tree_valid(cloud2_valid)
);


	Light_Sprite
light_sprite(.clk(clk), .hc(hc), .vc(vc), .tree1_x(light_x), .tree1_y(light_y), .light_signal(light_signal), .tree1
_r(light_r), .tree1_g(light_g), .tree1_b(light_b), .tree_valid(light_valid));


	Numbers_Sprite numbers_sprite(.*);


	Finish_Flag_Sprite
finish_flag_sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(flag_x), .car1_y(flag_y), .car1_r(flag_r), .car1_g(flag_g)
, .car1_b(flag_b), .car1_valid(flag_valid));

	Coin_Big_Sprite
coin_big_sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(coin1_x), .car1_y(coin1_y), .car1_r(coin1_r), .car1_g(coi
n1_g), .car1_b(coin1_b), .car1_valid(coin1_valid));

	Coin_Small_Sprite
coin_small_sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(coin2_x), .car1_y(coin2_y), .car1_r(coin2_r), .car1_g(c
oin2_g), .car1_b(coin2_b), .car1_valid(coin2_valid));

	Score_Label_Sprite
score_label_sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(score_label_x), .car1_y(score_label_y), .car1_r(score
_label_r), .car1_g(score_label_g), .car1_b(score_label_b), .car1_valid(score_label_valid));

	Mph_Label_Sprite
mph_label_sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(mph_label_x), .car1_y(mph_label_y), .car1_r(mph_la
bel_r), .car1_g(mph_label_g), .car1_b(mph_label_b), .car1_valid(mph_label_valid));

//////////////////////////////////

	Left_Arrow_Small_Sprite
left_arrow_small_sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(left_arrow_small_x), .car1_y(left_arrow_small_
y), .car1_r(left_arrow_small_r), .car1_g(left_arrow_small_g), .car1_b(left_arrow_small_b), .car1_valid(le
ft_arrow_small_valid));

            Left_Arrow_Big_Sprite
left_arrow_big_sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(left_arrow_big_x), .car1_y(left_arrow_big_y), .car
1_r(left_arrow_big_r), .car1_g(left_arrow_big_g), .car1_b(left_arrow_big_b), .car1_valid(left_arrow_big
_valid));

            Right_Arrow_Small_Sprite
right_arrow_small_sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(right_arrow_small_x), .car1_y(right_arrow_s
mall_y), .car1_r(right_arrow_small_r), .car1_g(right_arrow_small_g), .car1_b(right_arrow_small_b), .car
1_valid(right_arrow_small_valid));

            Right_Arrow_Big_Sprite
right_arrow_big_sprite(.clk(clk), .hc(hc), .vc(vc), .car1_x(right_arrow_big_x), .car1_y(right_arrow_big_y),
 .car1_r(right_arrow_big_r), .car1_g(right_arrow_big_g), .car1_b(right_arrow_big_b), .car1_valid(right_a
rrow_big_valid));




            always_ff @(posedge clk)

    if (reset) begin

/*                      VGA_B <= 8'hff;

                        VGA_R <= 8'h00;

                        VGA_G <= 8'h00;*/

                        SW <= 4'b 0000;

                        KEY <= 4'b 1111;


                        centerX <= 16'd 320;

                        centerY <= 16'd 240;

                        road_width <= 16'd 80;

//                      road_section <= 8'd 1;

                        car1_ahead_x <= 16'd 100;

                        car1_ahead_y <= 16'd 200;

                        car1_left_x <= 16'd 200;
```

```
car1_left_y <= 16'd 200;

car1_right_x <= 16'd 300;

car1_right_y <= 16'd 200;


car2_ahead_x <= 16'd 100;

car2_ahead_y <= 16'd 300;

car2_left_x <= 16'd 200;

car2_left_y <= 16'd 300;

car2_right_x <= 16'd 300;

car2_right_y <= 16'd 300;


car3_ahead_x <= 16'd 100;

car3_ahead_y <= 16'd 400;

car3_left_x <= 16'd 200;

car3_left_y <= 16'd 400;

car3_right_x <= 16'd 300;

car3_right_y <= 16'd 400;


tree1_x <= 16'd 100;

tree1_y <= 16'd 100;

tree2_x <= 16'd 200;

tree2_y <= 16'd 100;

tree3_x <= 16'd 300;

tree3_y <= 16'd 100;

tree4_x <= 16'd 100;

tree4_y <= 16'd 200;

tree5_x <= 16'd 300;

tree5_y <= 16'd 200;

tree6_x <= 16'd 400;
```

```
tree6_y <= 16'd 200;



cloud1_x <= 16'd 400;

cloud1_y <= 16'd 100;

cloud2_x <= 16'd 400;

cloud2_y <= 16'd 300;

cloud3_x <= 16'd 400;

cloud3_y <= 16'd 400;

cloud4_x <= 16'd 600;

cloud4_y <= 16'd 400;


light_x <= 16'd 200;

light_y <= 16'd 300;

light_signal <= 2'b 10;


pos <= 4'd 0;

pos_total <= 4'd 1;

min_ten <= 4'd 2;

min_uni <= 4'd 3;

sec_ten <= 4'd 4;

sec_uni <= 4'd 5;

speed_hund <= 4'd 6;

speed_ten <= 4'd 7;

speed_uni <= 4'd 8;

score_hund <= 4'd 9;

score_ten <= 4'd 0;

score_uni <= 4'd 1;
```

```
flag_x <= 16'd 300;
flag_y <= 16'd 300;
coin1_x <= 16'd 400;
coin1_y <= 16'd 300;
coin2_x <= 16'd 450;
coin2_y <= 16'd 300;
score_label_x <= 16'd 400;
score_label_y <= 16'd 500;
mph_label_x <= 16'd 630;
mph_label_y <= 16'd 440;

car1_medium_ahead_x <= 16'd 150;
car1_medium_ahead_y <= 16'd 200;
car1_medium_left_x <= 16'd 250;
car1_medium_left_y <= 16'd 200;
car1_medium_right_x <= 16'd 350;
car1_medium_right_y <= 16'd 200;

car2_medium_ahead_x <= 16'd 150;
car2_medium_ahead_y <= 16'd 300;
car2_medium_left_x <= 16'd 250;
car2_medium_left_y <= 16'd 300;
car2_medium_right_x <= 16'd 350;
car2_medium_right_y <= 16'd 300;

car3_medium_ahead_x <= 16'd 150;
car3_medium_ahead_y <= 16'd 400;
car3_medium_left_x <= 16'd 250;
car3_medium_left_y <= 16'd 400;
```

```
car3_medium_right_x <= 16'd 350;

car3_medium_right_y <= 16'd 400;


car1_small_ahead_x <= 16'd 50;

car1_small_ahead_y <= 16'd 200;

car1_small_left_x <= 16'd 150;

car1_small_left_y <= 16'd 200;

car1_small_right_x <= 16'd 250;

car1_small_right_y <= 16'd 200;


car2_small_ahead_x <= 16'd 50;

car2_small_ahead_y <= 16'd 300;

car2_small_left_x <= 16'd 150;

car2_small_left_y <= 16'd 300;

car2_small_right_x <= 16'd 250;

car2_small_right_y <= 16'd 300;


car3_small_ahead_x <= 16'd 50;

car3_small_ahead_y <= 16'd 400;

car3_small_left_x <= 16'd 150;

car3_small_left_y <= 16'd 400;

car3_small_right_x <= 16'd 250;

car3_small_right_y <= 16'd 400;


left_arrow_small_x <= 16'd 50;

left_arrow_small_y <= 16'd 420;


left_arrow_big_x <= 16'd 200;

left_arrow_big_y <= 16'd 420;
```

```
                        right_arrow_small_x <= 16'd 50;
                        right_arrow_small_y <= 16'd 200;

                        right_arrow_big_x <= 16'd 200;
                        right_arrow_big_y <= 16'd 200;

//                      radius <= 16'd 30;
    end else if (chipselect && write) begin
//                              SW <= 4'b 0000;
//                              KEY <= 4'b 1111;
      case (address)
                                8'd 0 : centerX[7:0] <= writedata;
                                8'd 1 : centerX[15:8] <= writedata;
                                8'd 2 : centerY[7:0] <= writedata;
                                8'd 3 : centerY[15:8] <= writedata;

                                8'd 4 : car1_ahead_x[7:0] <= writedata;
                                8'd 5 : car1_ahead_x[15:8] <= writedata;
                                8'd 6 : car1_ahead_y[7:0] <= writedata;
                                8'd 7 : car1_ahead_y[15:8] <= writedata;

                                8'd 8 : car1_left_x[7:0] <= writedata;
                                8'd 9 : car1_left_x[15:8] <= writedata;
                                8'd 10 : car1_left_y[7:0] <= writedata;
                                8'd 11 : car1_left_y[15:8] <= writedata;

                                8'd 12 : car1_right_x[7:0] <= writedata;
                                8'd 13 : car1_right_x[15:8] <= writedata;
```

```
8'd 14 : car1_right_y[7:0] <= writedata;
8'd 15 : car1_right_y[15:8] <= writedata;

8'd 16 : car2_ahead_x[7:0] <= writedata;
8'd 17 : car2_ahead_x[15:8] <= writedata;
8'd 18 : car2_ahead_y[7:0] <= writedata;
8'd 19 : car2_ahead_y[15:8] <= writedata;

8'd 20 : car2_left_x[7:0] <= writedata;
8'd 21 : car2_left_x[15:8] <= writedata;
8'd 22 : car2_left_y[7:0] <= writedata;
8'd 23 : car2_left_y[15:8] <= writedata;

8'd 24 : car2_right_x[7:0] <= writedata;
8'd 25 : car2_right_x[15:8] <= writedata;
8'd 26 : car2_right_y[7:0] <= writedata;
8'd 27 : car2_right_y[15:8] <= writedata;

8'd 28 : car3_ahead_x[7:0] <= writedata;
8'd 29 : car3_ahead_x[15:8] <= writedata;
8'd 30 : car3_ahead_y[7:0] <= writedata;
8'd 31 : car3_ahead_y[15:8] <= writedata;

8'd 32 : car3_left_x[7:0] <= writedata;
8'd 33 : car3_left_x[15:8] <= writedata;
8'd 34 : car3_left_y[7:0] <= writedata;
8'd 35 : car3_left_y[15:8] <= writedata;

8'd 36 : car3_right_x[7:0] <= writedata;
```

```
8'd 37 : car3_right_x[15:8] <= writedata;
8'd 38 : car3_right_y[7:0] <= writedata;
8'd 39 : car3_right_y[15:8] <= writedata;

8'd 40 : tree1_x[7:0] <= writedata;
8'd 41 : tree1_x[15:8] <= writedata;
8'd 42 : tree1_y[7:0] <= writedata;
8'd 43 : tree1_y[15:8] <= writedata;

8'd 44 : tree2_x[7:0] <= writedata;
8'd 45 : tree2_x[15:8] <= writedata;
8'd 46 : tree2_y[7:0] <= writedata;
8'd 47 : tree2_y[15:8] <= writedata;

8'd 48 : tree3_x[7:0] <= writedata;
8'd 49 : tree3_x[15:8] <= writedata;
8'd 50 : tree3_y[7:0] <= writedata;
8'd 51 : tree3_y[15:8] <= writedata;

8'd 52 : tree4_x[7:0] <= writedata;
8'd 53 : tree4_x[15:8] <= writedata;
8'd 54 : tree4_y[7:0] <= writedata;
8'd 55 : tree4_y[15:8] <= writedata;

8'd 56 : tree5_x[7:0] <= writedata;
8'd 57 : tree5_x[15:8] <= writedata;
8'd 58 : tree5_y[7:0] <= writedata;
8'd 59 : tree5_y[15:8] <= writedata;
```

```
8'd 60 : tree6_x[7:0] <= writedata;

8'd 61 : tree6_x[15:8] <= writedata;

8'd 62 : tree6_y[7:0] <= writedata;

8'd 63 : tree6_y[15:8] <= writedata;


8'd 64 : cloud1_x[7:0] <= writedata;

8'd 65 : cloud1_x[15:8] <= writedata;

8'd 66 : cloud1_y[7:0] <= writedata;

8'd 67 : cloud1_y[15:8] <= writedata;


8'd 68 : cloud2_x[7:0] <= writedata;

8'd 69 : cloud2_x[15:8] <= writedata;

8'd 70 : cloud2_y[7:0] <= writedata;

8'd 71 : cloud2_y[15:8] <= writedata;


8'd 72 : cloud3_x[7:0] <= writedata;

8'd 73 : cloud3_x[15:8] <= writedata;

8'd 74 : cloud3_y[7:0] <= writedata;

8'd 75 : cloud3_y[15:8] <= writedata;


8'd 76 : cloud4_x[7:0] <= writedata;

8'd 77 : cloud4_x[15:8] <= writedata;

8'd 78 : cloud4_y[7:0] <= writedata;

8'd 79 : cloud4_y[15:8] <= writedata;


8'd 80 : light_x[7:0] <= writedata;

8'd 81 : light_x[15:8] <= writedata;

8'd 82 : light_y[7:0] <= writedata;

8'd 83 : light_y[15:8] <= writedata;
```

```
8'd 84 : light_signal <= writedata;


8'd 85 : pos[3:0] <= writedata;

8'd 86 : pos_total[3:0] <= writedata;

8'd 87 : min_ten[3:0] <= writedata;

8'd 88 : min_uni[3:0] <= writedata;

8'd 89 : sec_ten[3:0] <= writedata;

8'd 90 : sec_uni[3:0] <= writedata;

8'd 91 : speed_hund[3:0] <= writedata;

8'd 92 : speed_ten[3:0] <= writedata;

8'd 93 : speed_uni[3:0] <= writedata;

8'd 94 : score_hund[3:0] <= writedata;

8'd 95 : score_ten[3:0] <= writedata;

8'd 96 : score_uni[3:0] <= writedata;


8'd 97 : flag_x[7:0] <= writedata;

8'd 98 : flag_x[15:8] <= writedata;

8'd 99 : flag_y[7:0] <= writedata;

8'd 100 : flag_y[15:8] <= writedata;

8'd 101 : coin1_x[7:0] <= writedata;

8'd 102 : coin1_x[15:8] <= writedata;

8'd 103 : coin1_y[7:0] <= writedata;

8'd 104 : coin1_y[15:8] <= writedata;

8'd 105 : coin2_x[7:0] <= writedata;

8'd 106 : coin2_x[15:8] <= writedata;

8'd 107 : coin2_y[7:0] <= writedata;

8'd 108 : coin2_y[15:8] <= writedata;

8'd 109 : score_label_x[7:0] <= writedata;

8'd 110 : score_label_x[15:8] <= writedata;
```

```
                              8'd 111 : score_label_y[7:0] <= writedata;
                              8'd 112 : score_label_y[15:8] <= writedata;
                              8'd 113 : mph_label_x[7:0] <= writedata;
                              8'd 114 : mph_label_x[15:8] <= writedata;
                              8'd 115 : mph_label_y[7:0] <= writedata;
                              8'd 116 : mph_label_y[15:8] <= writedata;
/////////////////////////////////////////////////////////////////////////////
                              8'd 117 : car1_medium_ahead_x[7:0] <= writedata;
                              8'd 118 : car1_medium_ahead_x[15:8] <= writedata;
                              8'd 119 : car1_medium_ahead_y[7:0] <= writedata;
                              8'd 120 : car1_medium_ahead_y[15:8] <= writedata;

                              8'd 121 : car1_medium_left_x[7:0] <= writedata;
                              8'd 122 : car1_medium_left_x[15:8] <= writedata;
                              8'd 123 : car1_medium_left_y[7:0] <= writedata;
                              8'd 124 : car1_medium_left_y[15:8] <= writedata;

                              8'd 125 : car1_medium_right_x[7:0] <= writedata;
                              8'd 126 : car1_medium_right_x[15:8] <= writedata;
                              8'd 127 : car1_medium_right_y[7:0] <= writedata;
                              8'd 128 : car1_medium_right_y[15:8] <= writedata;

                              8'd 129 : car2_medium_ahead_x[7:0] <= writedata;
                              8'd 130 : car2_medium_ahead_x[15:8] <= writedata;
                              8'd 131 : car2_medium_ahead_y[7:0] <= writedata;
                              8'd 132 : car2_medium_ahead_y[15:8] <= writedata;

                              8'd 133 : car2_medium_left_x[7:0] <= writedata;
                              8'd 134 : car2_medium_left_x[15:8] <= writedata;
```

```
                    8'd 135 : car2_medium_left_y[7:0] <= writedata;
                    8'd 136 : car2_medium_left_y[15:8] <= writedata;

                    8'd 137 : car2_medium_right_x[7:0] <= writedata;
                    8'd 138 : car2_medium_right_x[15:8] <= writedata;
                    8'd 139 : car2_medium_right_y[7:0] <= writedata;
                    8'd 140 : car2_medium_right_y[15:8] <= writedata;

                    8'd 141 : car3_medium_ahead_x[7:0] <= writedata;
                    8'd 142 : car3_medium_ahead_x[15:8] <= writedata;
                    8'd 143 : car3_medium_ahead_y[7:0] <= writedata;
                    8'd 144 : car3_medium_ahead_y[15:8] <= writedata;

                    8'd 145 : car3_medium_left_x[7:0] <= writedata;
                    8'd 146 : car3_medium_left_x[15:8] <= writedata;
                    8'd 147 : car3_medium_left_y[7:0] <= writedata;
                    8'd 148 : car3_medium_left_y[15:8] <= writedata;

                    8'd 149 : car3_medium_right_x[7:0] <= writedata;
                    8'd 150 : car3_medium_right_x[15:8] <= writedata;
                    8'd 151 : car3_medium_right_y[7:0] <= writedata;
                    8'd 152 : car3_medium_right_y[15:8] <= writedata;

/////////////////////////////////////////////////////////////////////////////
                    8'd 153 : car1_small_ahead_x[7:0] <= writedata;
                    8'd 154 : car1_small_ahead_x[15:8] <= writedata;
                    8'd 155 : car1_small_ahead_y[7:0] <= writedata;
                    8'd 156 : car1_small_ahead_y[15:8] <= writedata;
```

```
8'd 157 : car1_small_left_x[7:0] <= writedata;
8'd 158 : car1_small_left_x[15:8] <= writedata;
8'd 159 : car1_small_left_y[7:0] <= writedata;
8'd 160 : car1_small_left_y[15:8] <= writedata;

8'd 161 : car1_small_right_x[7:0] <= writedata;
8'd 162 : car1_small_right_x[15:8] <= writedata;
8'd 163 : car1_small_right_y[7:0] <= writedata;
8'd 164 : car1_small_right_y[15:8] <= writedata;

8'd 165 : car2_small_ahead_x[7:0] <= writedata;
8'd 166 : car2_small_ahead_x[15:8] <= writedata;
8'd 167 : car2_small_ahead_y[7:0] <= writedata;
8'd 168 : car2_small_ahead_y[15:8] <= writedata;

8'd 169 : car2_small_left_x[7:0] <= writedata;
8'd 170 : car2_small_left_x[15:8] <= writedata;
8'd 171 : car2_small_left_y[7:0] <= writedata;
8'd 172 : car2_small_left_y[15:8] <= writedata;

8'd 173 : car2_small_right_x[7:0] <= writedata;
8'd 174 : car2_small_right_x[15:8] <= writedata;
8'd 175 : car2_small_right_y[7:0] <= writedata;
8'd 176 : car2_small_right_y[15:8] <= writedata;

8'd 177 : car3_small_ahead_x[7:0] <= writedata;
8'd 178 : car3_small_ahead_x[15:8] <= writedata;
8'd 179 : car3_small_ahead_y[7:0] <= writedata;
8'd 180 : car3_small_ahead_y[15:8] <= writedata;
```

```
8'd 181 : car3_small_left_x[7:0] <= writedata;

8'd 182 : car3_small_left_x[15:8] <= writedata;

8'd 183 : car3_small_left_y[7:0] <= writedata;

8'd 184 : car3_small_left_y[15:8] <= writedata;


8'd 185 : car3_small_right_x[7:0] <= writedata;

8'd 186 : car3_small_right_x[15:8] <= writedata;

8'd 187 : car3_small_right_y[7:0] <= writedata;

8'd 188 : car3_small_right_y[15:8] <= writedata;


8'd 189 : road_width[7:0] <= writedata;

8'd 190 : road_width[15:8] <= writedata;


8'd 191 : left_arrow_small_x[7:0] <= writedata;

8'd 192 : left_arrow_small_x[15:8] <= writedata;

8'd 193 : left_arrow_small_y[7:0] <= writedata;

8'd 194 : left_arrow_small_y[15:8] <= writedata;


8'd 195 : left_arrow_big_x[7:0] <= writedata;

8'd 196 : left_arrow_big_x[15:8] <= writedata;

8'd 197 : left_arrow_big_y[7:0] <= writedata;

8'd 198 : left_arrow_big_y[15:8] <= writedata;


8'd 199 : right_arrow_small_x[7:0] <= writedata;

8'd 200 : right_arrow_small_x[15:8] <= writedata;

8'd 201 : right_arrow_small_y[7:0] <= writedata;

8'd 202 : right_arrow_small_y[15:8] <= writedata;
```

```
                        8'd 203 : right_arrow_big_x[7:0] <= writedata;
                        8'd 204 : right_arrow_big_x[15:8] <= writedata;
                        8'd 205 : right_arrow_big_y[7:0] <= writedata;
                        8'd 206 : right_arrow_big_y[15:8] <= writedata;

                        8'd 207 : SW <= writedata;




////////////////////////////////////////////////////////////////////////////////
  endcase
     end

//       assign x = centerX;
//       assign y = centerY;
         assign hc = posX;
         assign vc = posY;

         assign is_road1 = (480-vc)*(centerX-road_width/2);
         assign is_road2 = (480-centerY)*hc;
         assign is_road3 = (480-vc)*(centerX-road_width/2-30);
         assign is_road4 = (480-centerY)*(hc-40);

         assign is_road5 = (480-vc)*(610-(centerX+road_width/2));
         assign is_road6 = (480-centerY)*(600-hc);
         assign is_road7 = (480-vc)*(640-(centerX+road_width/2));
         assign is_road8 = (480-centerY)*(640-hc);
```

```
//	assign road_flag = ((vc - centerY)>>5) & 1'b1;


//	logic [7:0] road_section;
	always begin
		if ((vc>=240) && (vc<= 250))
			is_red = color_flag;
		if ((vc>250) && (vc<=270))
			is_red = ~color_flag;
		if ((vc>270) && (vc<=300))
			is_red = color_flag;
		if ((vc>300) && (vc<=340))
			is_red = ~color_flag;
		if ((vc>340) && (vc<=390))
			is_red = color_flag;
		if ((vc>390) && (vc<=450))
			is_red = ~color_flag;
		if ((vc>450) && (vc<=480))
			is_red = color_flag;
	end


	always_ff @(posedge clk) begin
		if (tree3_y[5])
			color_flag <= 1'b 1;
		else
			color_flag <= 1'b 0;
	end
/*
	always_comb begin
		if (road_flag)
```

```
				is_red = color_flag;
			else
				is_red = ~color_flag;
		end
*/


		always_comb begin
			if (numbers_valid && (numbers_r != 8'hff) && (numbers_g != 8'hff) && (numbers_b != 
8'hff)) begin
						VGA_R = numbers_r;
						VGA_G = numbers_g;
						VGA_B = numbers_b;
			end
			else if (flag_valid && (flag_r != 8'hff) && (flag_g != 8'hff) && (flag_b != 8'hff)) begin
						VGA_R = flag_r;
						VGA_G = flag_g;
						VGA_B = flag_b;
			end
			else if (score_label_valid && (score_label_r != 8'hff) && (score_label_g != 8'hff) && 
(score_label_b != 8'hff)) begin
						VGA_R = score_label_r;
						VGA_G = score_label_g;
						VGA_B = score_label_b;
			end
			else if (mph_label_valid && (mph_label_r != 8'hff) && (mph_label_g != 8'hff) && 
(mph_label_b != 8'hff)) begin
						VGA_R = mph_label_r;
						VGA_G = mph_label_g;
						VGA_B = mph_label_b;
			end
			else if (light_valid && (light_r != 8'h00) && (light_g != 8'h00) && (light_b != 8'h00)) begin
							VGA_R = light_r;
							VGA_G = light_g;
							VGA_B = light_b;
			end
			else if (car1_ahead_valid && (car1_ahead_r != 8'hff) && (car1_ahead_g != 8'hff) &&
(car1_ahead_b != 8'hff)) begin
							VGA_R = car1_ahead_r;
							VGA_G = car1_ahead_g;
							VGA_B = car1_ahead_b;
					end
			else if (car1_left_valid && (car1_left_r != 8'hff) && (car1_left_g != 8'hff) &&
(car1_left_b != 8'hff)) begin
							VGA_R = car1_left_r;
							VGA_G = car1_left_g;
							VGA_B = car1_left_b;
					end
			else if (car1_right_valid && (car1_right_r != 8'hff) && (car1_right_g != 8'hff) &&
(car1_right_b != 8'hff)) begin
							VGA_R = car1_right_r;
							VGA_G = car1_right_g;
							VGA_B = car1_right_b;
					end

//////////////////////////////////////////////////////////////////
			else if (car2_ahead_valid && (car2_ahead_r != 8'hff) && (car2_ahead_g != 8'hff) &&
(car2_ahead_b != 8'hff)) begin
							VGA_R = car2_ahead_r;
							VGA_G = car2_ahead_g;
							VGA_B = car2_ahead_b;
```

```
				end
			else if (car2_left_valid && (car2_left_r != 8'hff) && (car2_left_g != 8'hff) &&
(car2_left_b != 8'hff)) begin
						VGA_R = car2_left_r;
						VGA_G = car2_left_g;
						VGA_B = car2_left_b;
				end
			else if (car2_right_valid && (car2_right_r != 8'hff) && (car2_right_g != 8'hff) &&
(car2_right_b != 8'hff)) begin
						VGA_R = car2_right_r;
						VGA_G = car2_right_g;
						VGA_B = car2_right_b;
				end

//////////////////////////////////////////////////////////////////////////
			else if (car3_ahead_valid && (car3_ahead_r != 8'hff) && (car3_ahead_g != 8'hff) &&
(car3_ahead_b != 8'hff)) begin
						VGA_R = car3_ahead_r;
						VGA_G = car3_ahead_g;
						VGA_B = car3_ahead_b;
				end
			else if (car3_left_valid && (car3_left_r != 8'hff) && (car3_left_g != 8'hff) &&
(car3_left_b != 8'hff)) begin
						VGA_R = car3_left_r;
						VGA_G = car3_left_g;
						VGA_B = car3_left_b;
				end
			else if (car3_right_valid && (car3_right_r != 8'hff) && (car3_right_g != 8'hff) &&
(car3_right_b != 8'hff)) begin
						VGA_R = car3_right_r;
```

```
                                        VGA_G = car3_right_g;

                                        VGA_B = car3_right_b;

                            end

///////////medium/////////////////////

                    else if (car1_medium_ahead_valid && (car1_medium_ahead_r != 8'hff) &&
(car1_medium_ahead_g != 8'hff) && (car1_medium_ahead_b != 8'hff)) begin

                                        VGA_R = car1_medium_ahead_r;

                                        VGA_G = car1_medium_ahead_g;

                                        VGA_B = car1_medium_ahead_b;

                            end

                    else if (car1_medium_left_valid && (car1_medium_left_r != 8'hff) &&
(car1_medium_left_g != 8'hff) && (car1_medium_left_b != 8'hff)) begin

                                        VGA_R = car1_medium_left_r;

                                        VGA_G = car1_medium_left_g;

                                        VGA_B = car1_medium_left_b;

                            end

                    else if (car1_medium_right_valid && (car1_medium_right_r != 8'hff) &&
(car1_medium_right_g != 8'hff) && (car1_medium_right_b != 8'hff)) begin

                                        VGA_R = car1_medium_right_r;

                                        VGA_G = car1_medium_right_g;

                                        VGA_B = car1_medium_right_b;

                            end


                    else if (car2_medium_ahead_valid && (car2_medium_ahead_r != 8'hff) &&
(car2_medium_ahead_g != 8'hff) && (car2_medium_ahead_b != 8'hff)) begin

                                        VGA_R = car2_medium_ahead_r;

                                        VGA_G = car2_medium_ahead_g;

                                        VGA_B = car2_medium_ahead_b;

                            end
```

```
			else if (car2_medium_left_valid && (car2_medium_left_r != 8'hff) &&
(car2_medium_left_g != 8'hff) && (car2_medium_left_b != 8'hff)) begin
						VGA_R = car2_medium_left_r;
						VGA_G = car2_medium_left_g;
						VGA_B = car2_medium_left_b;
				end
			else if (car2_medium_right_valid && (car2_medium_right_r != 8'hff) &&
(car2_medium_right_g != 8'hff) && (car2_medium_right_b != 8'hff)) begin
						VGA_R = car2_medium_right_r;
						VGA_G = car2_medium_right_g;
						VGA_B = car2_medium_right_b;
				end

			else if (car3_medium_ahead_valid && (car3_medium_ahead_r != 8'hff) &&
(car3_medium_ahead_g != 8'hff) && (car3_medium_ahead_b != 8'hff)) begin
						VGA_R = car3_medium_ahead_r;
						VGA_G = car3_medium_ahead_g;
						VGA_B = car3_medium_ahead_b;
				end
			else if (car3_medium_left_valid && (car3_medium_left_r != 8'hff) &&
(car3_medium_left_g != 8'hff) && (car3_medium_left_b != 8'hff)) begin
						VGA_R = car3_medium_left_r;
						VGA_G = car3_medium_left_g;
						VGA_B = car3_medium_left_b;
				end
			else if (car3_medium_right_valid && (car3_medium_right_r != 8'hff) &&
(car3_medium_right_g != 8'hff) && (car3_medium_right_b != 8'hff)) begin
						VGA_R = car3_medium_right_r;
						VGA_G = car3_medium_right_g;
						VGA_B = car3_medium_right_b;
```

```
                    end

/////////////medium////////////////

//////////////////car small////////////////////
            else if (car1_small_ahead_valid && (car1_small_ahead_r != 8'hff) &&
(car1_small_ahead_g != 8'hff) && (car1_small_ahead_b != 8'hff)) begin
                            VGA_R = car1_small_ahead_r;
                            VGA_G = car1_small_ahead_g;
                            VGA_B = car1_small_ahead_b;
                    end
            else if (car1_small_left_valid && (car1_small_left_r != 8'hff) && (car1_small_left_g !=
8'hff) && (car1_small_left_b != 8'hff)) begin
                            VGA_R = car1_small_left_r;
                            VGA_G = car1_small_left_g;
                            VGA_B = car1_small_left_b;
                    end
            else if (car1_small_right_valid && (car1_small_right_r != 8'hff) &&
(car1_small_right_g != 8'hff) && (car1_small_right_b != 8'hff)) begin
                            VGA_R = car1_small_right_r;
                            VGA_G = car1_small_right_g;
                            VGA_B = car1_small_right_b;
                    end

/////////////////////////////////////////////////////////////////
            else if (car2_small_ahead_valid && (car2_small_ahead_r != 8'hff) &&
(car2_small_ahead_g != 8'hff) && (car2_small_ahead_b != 8'hff)) begin
                            VGA_R = car2_small_ahead_r;
                            VGA_G = car2_small_ahead_g;
```

```
					VGA_B = car2_small_ahead_b;
				end
			else if (car2_small_left_valid && (car2_small_left_r != 8'hff) && (car2_small_left_g != 
8'hff) && (car2_small_left_b != 8'hff)) begin
					VGA_R = car2_small_left_r;
					VGA_G = car2_small_left_g;
					VGA_B = car2_small_left_b;
				end
			else if (car2_small_right_valid && (car2_small_right_r != 8'hff) && 
(car2_small_right_g != 8'hff) && (car2_small_right_b != 8'hff)) begin
					VGA_R = car2_small_right_r;
					VGA_G = car2_small_right_g;
					VGA_B = car2_small_right_b;
				end

//////////////////////////////////////////////////////////////////////
			else if (car3_small_ahead_valid && (car3_small_ahead_r != 8'hff) && 
(car3_small_ahead_g != 8'hff) && (car3_small_ahead_b != 8'hff)) begin
					VGA_R = car3_small_ahead_r;
					VGA_G = car3_small_ahead_g;
					VGA_B = car3_small_ahead_b;
				end
			else if (car3_small_left_valid && (car3_small_left_r != 8'hff) && (car3_small_left_g != 
8'hff) && (car3_small_left_b != 8'hff)) begin
					VGA_R = car3_small_left_r;
					VGA_G = car3_small_left_g;
					VGA_B = car3_small_left_b;
				end
			else if (car3_small_right_valid && (car3_small_right_r != 8'hff) && 
(car3_small_right_g != 8'hff) && (car3_small_right_b != 8'hff)) begin
```

```
                                        VGA_R = car3_small_right_r;

                                        VGA_G = car3_small_right_g;

                                        VGA_B = car3_small_right_b;

                              end

//////////////////car small////////////////////////

                    else if (coin1_valid && (coin1_r != 8'hff) && (coin1_g != 8'hff) && (coin1_b != 8'hff))
begin

                                        VGA_R = coin1_r;

                                        VGA_G = coin1_g;

                                        VGA_B = coin1_b;

                    end

                    else if (coin2_valid && (coin2_r != 8'hff) && (coin2_g != 8'hff) && (coin2_b != 8'hff))
begin

                                        VGA_R = coin2_r;

                                        VGA_G = coin2_g;

                                        VGA_B = coin2_b;

                    end

///////////////////////////medium/////////////////////////////////////////////////////



///////////////////////////medium////////////////////////////////////////////////////

                    else if (tree1_valid && (tree1_r != 8'hff) && (tree1_g != 8'hff) && (tree1_b != 8'hff))
begin

                         VGA_R = tree1_r;

                         VGA_G = tree1_g;

                         VGA_B = tree1_b;

                    end

                    else if (tree2_valid && (tree2_r != 8'hff) && (tree2_g != 8'hff) && (tree2_b != 8'hff))
begin

                         VGA_R = tree2_r;
```

```
                VGA_G = tree2_g;

                VGA_B = tree2_b;

            end

            else if (tree3_valid && (tree3_r != 8'hff) && (tree3_g != 8'hff) && (tree3_b != 8'hff))
begin

                VGA_R = tree3_r;

                VGA_G = tree3_g;

                VGA_B = tree3_b;

            end

////////////////////////////////////////////////////

            else if (cloud1_valid && (cloud1_r != 8'h00) && (cloud1_g != 8'h00) && (cloud1_b !=
8'h00)) begin

                VGA_R = cloud1_r;

                VGA_G = cloud1_g;

                VGA_B = cloud1_b;

            end

            else if (cloud2_valid && (cloud2_r != 8'h00) && (cloud2_g != 8'h00) && (cloud2_b !=
8'h00)) begin

                VGA_R = cloud2_r;

                VGA_G = cloud2_g;

                VGA_B = cloud2_b;

            end


            else if (left_arrow_big_valid && (left_arrow_big_r != 8'h00) && (left_arrow_big_g !=
8'h00) && (left_arrow_big_b != 8'h00)) begin

                VGA_R = left_arrow_big_r;

                VGA_G = left_arrow_big_g;

                VGA_B = left_arrow_big_b;

            end
```

```
                    else if (left_arrow_small_valid && (left_arrow_small_r != 8'h00) &&
(left_arrow_small_g != 8'h00) && (left_arrow_small_b != 8'h00)) begin

                              VGA_R = left_arrow_small_r;

                              VGA_G = left_arrow_small_g;

                              VGA_B = left_arrow_small_b;

                    end


                    else if (right_arrow_big_valid && (right_arrow_big_r != 8'h00) && (right_arrow_big_g !=
8'h00) && (right_arrow_big_b != 8'h00)) begin

                              VGA_R = right_arrow_big_r;

                              VGA_G = right_arrow_big_g;

                              VGA_B = right_arrow_big_b;

                    end


                    else if (right_arrow_small_valid && (right_arrow_small_r != 8'h00) &&
(right_arrow_small_g != 8'h00) && (right_arrow_small_b != 8'h00)) begin

                              VGA_R = right_arrow_small_r;

                              VGA_G = right_arrow_small_g;

                              VGA_B = right_arrow_small_b;

                    end
///////////////////////////////////////////////////
                    else if (posY <= centerY) begin

                                        VGA_R = 8'h5c;

                                        VGA_G = 8'hac;

                                        VGA_B = 8'hee;

                                        end

                    else if (is_road1>is_road2) begin

                                                  VGA_R = 8'hff;

                                                  VGA_G = 8'hcc;

                                                  VGA_B = 8'h99;
```

```
                end
else if (is_road3>is_road4) begin
                        if (is_red) begin
                                VGA_R = 8'hff;
                                VGA_G = 8'h00;
                                VGA_B = 8'h00;
                        end
                        else begin
                                VGA_R = 8'hff;
                                VGA_G = 8'hff;
                                VGA_B = 8'hff;

                        end
                end
else if (is_road5<is_road6) begin
                VGA_R = 8'he0;
                VGA_G = 8'he0;
                VGA_B = 8'he0;
                end
else if (is_road7<is_road8) begin
                        if (is_red) begin
                                VGA_R = 8'hff;
                                VGA_G = 8'h00;
                                VGA_B = 8'h00;
                        end
                        else begin
                                VGA_R = 8'hff;
                                VGA_G = 8'hff;
```

```
                                    VGA_B = 8'hff;

                                end
                            end
                        else begin
                            VGA_R = 8'hff;
                            VGA_G = 8'hcc;
                            VGA_B = 8'h99;
                        end
/*                  else if (posY <= centerY) begin
                            VGA_R = 8'h5c;
                            VGA_G = 8'hac;
                            VGA_B = 8'hee;
                            end

                    else if ((is_road1>is_road2)&&(is_road3>is_road4)) begin
                            VGA_R = 8'he0;
                            VGA_G = 8'he0;
                            VGA_B = 8'he0;
                            end
                            else begin
                                VGA_R = 8'hff;
                                VGA_G = 8'hcc;
                                VGA_B = 8'h99;
                            end
*/
/*                  else if ((is_road1>is_road2)) begin
                            VGA_R = 8'he0;
                            VGA_G = 8'he0;
```

```
                                        VGA_B = 8'he0;
                                        end
                                        else begin
                                                VGA_R = 8'hff;
                                                VGA_G = 8'hcc;
                                                VGA_B = 8'h99;
                                        end
*/
        end



endmodule
```

7.1.3 VGA_Sprite_Emulator.sv

```
module VGA_BALL_Emulator(
 input logic            clk50, reset,
// input logic [7:0]  hex0, hex1, hex2, hex3, hex4, hex5, hex6, hex7,
// input logic [15:0]  centerX,
// input logic [15:0]  centerY,
// input logic [15:0]  radius,
// input logic [7:0] VGA_R_TEMP, VGA_G_TEMP, VGA_B_TEMP,
// input logic       write,
// input                         chipselect,
// input logic [7:0] VGA_R, VGA_G, VGA_B,
 output logic [9:0] posX,
 output logic [8:0] posY,
// output logic [7:0] VGA_R, VGA_G, VGA_B,
output logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0          1279      1599 0
 *          _______________          ________
 * ___________|   Video     |____________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *     _______________________     _____________
 * |____|      VGA_HS        |____|
 */
  // Parameters for hcount
  parameter HACTIVE     = 11'd 1280,
            HFRONT_PORCH = 11'd 32,
            HSYNC       = 11'd 192,
            HBACK_PORCH  = 11'd 96,
            HTOTAL      = HACTIVE + HFRONT_PORCH + HSYNC + HBACK_PORCH; // 1600

  // Parameters for vcount
  parameter VACTIVE     = 10'd 480,
            VFRONT_PORCH = 10'd 10,
            VSYNC       = 10'd 2,
            VBACK_PORCH  = 10'd 33,
            VTOTAL      = VACTIVE + VFRONT_PORCH + VSYNC + VBACK_PORCH; // 525

  logic [10:0]                         hcount; // Horizontal counter
```

```
                                    // Hcount[10:1] indicates pixel column (0-639)
logic                               endOfLine;

always_ff @(posedge clk50 or posedge reset)
  if (reset)          hcount <= 0;
  else if (endOfLine) hcount <= 0;
  else            hcount <= hcount + 11'd 1;

assign endOfLine = hcount == HTOTAL - 1;

// Vertical counter
logic [9:0]                              vcount;
logic                               endOfField;

always_ff @(posedge clk50 or posedge reset)
  if (reset)          vcount <= 0;
  else if (endOfLine)
    if (endOfField)   vcount <= 0;
    else              vcount <= vcount + 10'd 1;

assign endOfField = vcount == VTOTAL - 1;

// Horizontal sync: from 0x520 to 0x5DF (0x57F)
// 101 0010 0000 to 101 1101 1111
assign VGA_HS = !( (hcount[10:8] == 3'b101) & !(hcount[7:5] == 3'b111));
assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

assign VGA_SYNC_n = 1; // For adding sync to video signals; not used for VGA
```

```
   // Horizontal active: 0 to 1279     Vertical active: 0 to 479
   // 101 0000 0000  1280                 01 1110 0000  480
   // 110 0011 1111  1599                 10 0000 1100  524
   assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                                  !( vcount[9] | (vcount[8:5] == 4'b1111) );

   /* VGA_CLK is 25 MHz
    *          __   __   __
    * clk50   __|  |__|  |__|
    *
    *          _____     __
    * hcount[0]__|     |_____|
    */
   assign VGA_CLK = hcount[0]; // 25 MHz clock: pixel latched on rising edge

//         logic [15:0] posX, posY;
//         logic [31:0] distance;
//         logic [31:0] radius_square;



//         assign centerx = 640/2;
//         assign centery = 480/2;

//         assign centerx = HACTIVE/2/2;
//         assign centery = VACTIVE/2;
           assign posX = hcount[10:1];
           assign posY = vcount[8:0];

//         assign r = 9'd 900;
```

```
//	assign radius_square = radius * radius;
//	assign distance = (posX - centerX)*(posX - centerX) + (posY - centerY)*(posY - centerY);
/*

	always_comb begin
    if (distance <= radius_square)
			{VGA_R, VGA_G, VGA_B} = {VGA_R_TEMP, VGA_G_TEMP, VGA_B_TEMP};
			else
			{VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0}; // Black
  end
*/
endmodule // VGA_LED_Emulator
```

7.1.4 Car1_Medium_Right_Sprite.sv

```
module Car1_Medium_Right_Sprite(
 input logic clk,
 input logic [15:0] hc, vc,
 input logic [15:0] car1_x, car1_y,
 output logic [7:0] car1_r, car1_g, car1_b,
 output logic car1_valid);

  // Parameters for width and height
  parameter HEIGHT      = 16'd 48,
	    WIDTH   = 16'd 48;

 logic [15:0] address;
 logic [23:0] rgb;
 logic car1_valid1, car1_valid2;
 logic [15:0] index_x, index_y;
```

```
CAR1_MEDIUM_RIGHT car1_medium_right_rom(.address(address), .clock(clk), .q(rgb));

always begin
        if (car1_y < (HEIGHT/2)) begin
                if (vc < car1_y + HEIGHT/2) begin
                        car1_valid1 = 1'b 1;
                        index_y = vc + (HEIGHT/2 - car1_y);
                end
                else begin
                        car1_valid1 = 1'b 0;
          end
        end
        else if ((vc >= car1_y - HEIGHT/2) && (vc < car1_y + HEIGHT/2)) begin
                        car1_valid1 = 1'b 1;
                  index_y = vc - (car1_y - HEIGHT/2);
                 end
                 else begin
                        car1_valid1 = 1'b 0;
                 end
end

always begin
        if (car1_x < WIDTH/2) begin
                if (hc < car1_x + WIDTH/2) begin
                        car1_valid2 = 1'b 1;
                        index_x = hc + (WIDTH/2 - car1_x);
                end
                else
```

```
                                    car1_valid2 = 1'b 0;
                end
                else begin if ((hc >= car1_x - WIDTH/2) && (hc < car1_x + WIDTH/2)) begin
                                    car1_valid2 = 1'b 1;
                          index_x = hc - (car1_x - WIDTH/2);
                         end
                         else
                                    car1_valid2 = 1'b 0;
                end
        end
       /*
        always_comb begin
                if (~(car1_valid1 & car1_valid2))
                        car1_valid = 1'b0;
                else if (rgb == 24'hffffff)
                                        car1_valid = 1'b0;
                         else
                                        car1_valid = 1'b1;


        end


       */
        /*
        always_comb begin
                if (rgb == 24'hffffff)
                        car1_valid3 = 1'b 0;
                else
                        car1_valid3 = 1'b 1;
        end
```

```
*/

assign car1_valid = car1_valid1 & car1_valid2;

always_comb begin
 if (car1_valid)
                                        address = index_x + index_y * WIDTH;
 else
                                        address = 16'h0000;
end

assign {car1_r, car1_g, car1_b} = rgb;




endmodule
```

7.1.5 Car1_Medium_Ahead_Sprite.sv

```
module Car1_Medium_Ahead_Sprite(
 input logic clk,
 input logic [15:0] hc, vc,
 input logic [15:0] car1_x, car1_y,
 output logic [7:0] car1_r, car1_g, car1_b,
 output logic car1_valid);

   // Parameters for width and height
   parameter HEIGHT      = 16'd 48,
            WIDTH    = 16'd 48;

logic [15:0] address;

logic [23:0] rgb;

logic car1_valid1, car1_valid2;

logic [15:0] index_x, index_y;

CAR1_MEDIUM_AHEAD car1_medium_ahead_rom(.address(address), .clock(clk), .q(rgb));

always begin

            if (car1_y < (HEIGHT/2)) begin

                        if (vc < car1_y + HEIGHT/2) begin

                                    car1_valid1 = 1'b 1;

                                    index_y = vc + (HEIGHT/2 - car1_y);

                        end

                        else begin

                                    car1_valid1 = 1'b 0;

              end

            end

            else if ((vc >= car1_y - HEIGHT/2) && (vc < car1_y + HEIGHT/2)) begin

                                    car1_valid1 = 1'b 1;

                          index_y = vc - (car1_y - HEIGHT/2);

                         end

                         else begin

                                    car1_valid1 = 1'b 0;

                         end

end

always begin
```

```
            if (car1_x < WIDTH/2) begin

                        if (hc < car1_x + WIDTH/2) begin

                                    car1_valid2 = 1'b 1;

                                    index_x = hc + (WIDTH/2 - car1_x);

                        end

                        else

                                    car1_valid2 = 1'b 0;

            end

            else begin if ((hc >= car1_x - WIDTH/2) && (hc < car1_x + WIDTH/2)) begin

                                    car1_valid2 = 1'b 1;

                          index_x = hc - (car1_x - WIDTH/2);

                         end

                         else

                                    car1_valid2 = 1'b 0;

            end

 end

/*

 always_comb begin

            if (~(car1_valid1 & car1_valid2))

                        car1_valid = 1'b0;

            else if (rgb == 24'hffffff)

                                                car1_valid = 1'b0;

                         else

                                                car1_valid = 1'b1;


 end


*/

 /*
```

```
always_comb begin
        if (rgb == 24'hffffff)
                car1_valid3 = 1'b 0;
        else
                car1_valid3 = 1'b 1;
end
*/

assign car1_valid = car1_valid1 & car1_valid2;

always_comb begin
 if (car1_valid)
                        address = index_x + index_y * WIDTH;
 else
                        address = 16'h0000;
end

assign {car1_r, car1_g, car1_b} = rgb;




endmodule
```

7.1.6 Car1_Medium_Left_Sprite.sv

```
module Car1_Medium_Left_Sprite(
 input logic clk,
input logic [15:0] hc, vc,

input logic [15:0] car1_x, car1_y,

output logic [7:0] car1_r, car1_g, car1_b,

output logic car1_valid);


  // Parameters for width and height

  parameter HEIGHT      = 16'd 48,

            WIDTH   = 16'd 48;


logic [15:0] address;

logic [23:0] rgb;

logic car1_valid1, car1_valid2;

logic [15:0] index_x, index_y;


CAR1_MEDIUM_LEFT car1_medium_left_rom(.address(address), .clock(clk), .q(rgb));


always begin

        if (car1_y < (HEIGHT/2)) begin

                if (vc < car1_y + HEIGHT/2) begin

                        car1_valid1 = 1'b 1;

                        index_y = vc + (HEIGHT/2 - car1_y);

                end

                else begin

                        car1_valid1 = 1'b 0;

          end

        end

        else if ((vc >= car1_y - HEIGHT/2) && (vc < car1_y + HEIGHT/2)) begin

                        car1_valid1 = 1'b 1;

                  index_y = vc - (car1_y - HEIGHT/2);
```

```
                        end
                        else begin
                                car1_valid1 = 1'b 0;
                        end
end

always begin
        if (car1_x < WIDTH/2) begin
                if (hc < car1_x + WIDTH/2) begin
                        car1_valid2 = 1'b 1;
                        index_x = hc + (WIDTH/2 - car1_x);
                end
                else
                        car1_valid2 = 1'b 0;
        end
        else begin if ((hc >= car1_x - WIDTH/2) && (hc < car1_x + WIDTH/2)) begin
                        car1_valid2 = 1'b 1;
                 index_x = hc - (car1_x - WIDTH/2);
                end
                else
                        car1_valid2 = 1'b 0;
        end
end
/*
always_comb begin
        if (~(car1_valid1 & car1_valid2))
                car1_valid = 1'b0;
        else if (rgb == 24'hffffff)
                                car1_valid = 1'b0;
                        else
                                    car1_valid = 1'b1;

end

*/
/*
always_comb begin
        if (rgb == 24'hffffff)
                car1_valid3 = 1'b 0;
        else
                car1_valid3 = 1'b 1;
end
*/

assign car1_valid = car1_valid1 & car1_valid2;

always_comb begin
 if (car1_valid)
                        address = index_x + index_y * WIDTH;
 else
                        address = 16'h0000;
end

assign {car1_r, car1_g, car1_b} = rgb;
```

Endmodule

7.1.7 Generating “accelerate.mif”

```
%generating accelerate.mif
[y,Fs]=audioread('accelerate.wav');
y1=y(150000:170000);
y2=y(180000:220000);
y3=y(230000:250000);
y1=y1';
y2=y2';
y3=y3';

c=[y1;y2;y3];
c1=c(1000:3000);
c2=c(20000:23000);
c3=c(31000:33000);
c4=c(43000:52000);
c=[c1;c2;c3;c4];

c=c*65536;
z=typecast(int16(c),'uint16');

fileID = fopen('accelerate.mif', 'w');
str = 'WIDTH=16;\nDEPTH=%d;\n\nADDRESS_RADIX=HEX;\nDATA_RADIX=HEX;\n\n';
fprintf(fileID, str,length(z));

str = 'CONTENT BEGIN\n\n';
fprintf(fileID, str);
for i=1:length(z);
    str = '%04X :%X';
    fprintf(fileID,str,i-1);
    str=' %04X;\n%X';
    fprintf(fileID,str,z(i));
```

```
end
str = '\nEND;';
fprintf(fileID, str);
fclose(fileID);
```

7.1.8 audio_codec.sv

```
//Audio codec interface
module audio_codec (
   input  clk, //audio clock
   input  reset,
   output [1:0]  sample_end,   //end of sample
   output [1:0]  sample_req,   //request new sample
   input  [15:0] audio_output, //audio output sent to audio codec
   input  [1:0] channel_sel,   //select channel
   output AUD_ADCLRCK, //ADC channel clock
   input AUD_ADCDAT,
   output AUD_DACLRCK, //DAC channel clock
   output AUD_DACDAT,
   output AUD_BCLK //Bit clock
);

// divided by 256 clock for the LRC clock, one clock is oen audio frame
reg [7:0] lrck_divider;

// divided by 4 clock for the bit clock BCLK
reg [1:0] bclk_divider;

reg [15:0] shift_out;
reg [15:0] shift_temp;
```

```
wire lrck = !lrck_divider[7];

//assigning clocks from the clock divider
assign AUD_ADCLRCK = lrck;
assign AUD_DACLRCK = lrck;
assign AUD_BCLK = bclk_divider[1];

// assigning data as last bit of shift register
assign AUD_DACDAT = shift_out[15];


always @(posedge clk) begin
    if (reset) begin
        lrck_divider <= 8'hff;
        bclk_divider <= 2'b11;
    end else begin
        lrck_divider <= lrck_divider + 1'b1;
        bclk_divider <= bclk_divider + 1'b1;
    end
end

//first 16 bit sample sent after 16 bclks or 4*16=64 mclk
assign sample_end[1] = (lrck_divider == 8'h40);
//second 16 bit sample sent after 48 bclks or 4*48 = 192 mclk
assign sample_end[0] = (lrck_divider == 8'hc0);

// end of one lrc clk cycle (254 mclk cycles)
assign sample_req[1] = (lrck_divider == 8'hfe);
// end of half lrc clk cycle (126 mclk cycles) so request for next sample
assign sample_req[0] = (lrck_divider == 8'h7e);
```

```
wire clr_lrck = (lrck_divider == 8'h7f); // 127 mclk
wire set_lrck = (lrck_divider == 8'hff); // 255 mclk
// high right after bclk is set
wire set_bclk = (bclk_divider == 2'b10 && !lrck_divider[6]);
// high right before bclk is cleared
wire clr_bclk = (bclk_divider == 2'b11 && !lrck_divider[6]);

//implementing shift operation to send the audio samples
always @(posedge clk) begin
   if (reset) begin
      shift_out <= 16'h0;
      shift_temp <= 16'h0;
       end
       else if (set_lrck) begin
         shift_out <= audio_output;
         shift_temp <= audio_output;
   end
       else if (clr_lrck) begin
                  shift_out <= shift_temp;
   end else if (clr_bclk == 1) begin
      shift_out <= {shift_out[14:0], 1'b0};
   end
end

endmodule
```

7.1.9 audio_effects.sv

```
module audio_effects (
   input  logic clk, //audio clock
```

```
    input  logic sample_end, //sample ends
    input  logic sample_req, //request new sample
          //input logic [15:0] audio_sample, //get audio sample from audio codec interface, not
needed here
    output logic [15:0] audio_output, //sends audio sample to audio codec
    input logic [15:0] M_city,    //city sound ROM data
    output logic [14:0] addr_city,
          input logic [15:0] accelerate,   //acceleration ROM data
    output logic [14:0] addr_accelerate,
          input logic [15:0] crash,    //crash sound ROM data
    output logic [14:0] addr_crash,
          input logic [3:0] ctl  //control signal
);



//index through the sound ROM data for different sounds
reg  [15:0]  index_city = 15'd0;
reg  [15:0]  index_accelerate = 15'd0;
reg  [15:0]  index_crash = 15'd0;
reg  [15:0]  count = 15'd0;

reg [15:0] dat;

assign audio_output = dat;

//assign index to ROM addresses
always @(posedge clk) begin
    addr_city <= index_city;
          addr_accelerate <= index_accelerate;
          addr_crash <= index_crash;
```

```
end

//Play background (city) sound if ctl is 0
//Play acceleration sound effect if ctl is 1
//Play crash sound effect if ctl is 2
always @(posedge clk) begin

    if (sample_req) begin
            case(ctl)
                        4'b0000: begin
                                                dat<=M_city;
                                                if (index_city == 15'd22049)
                                                            index_city <= 15'd0;
                                                else
                                                            index_city <= index_city +1'b1;   //increment
city index
                                    end
                        4'b0001: begin
                                                dat<=accelerate;
                                                if (index_accelerate == 15'd16003)
                                                            index_accelerate <= 15'd0;
                                                else
                                                           index_accelerate <= index_accelerate +1'b1;
//increment accelerate index
                                    end
                        4'b0010: begin
                                                dat<=crash;
                                                if (index_crash == 15'd15999)
                                                            index_crash <= 15'd0;
                                                else
```

```
                                                                    index_crash <= index_crash +1'b1;
//increment crash index
                                                    end
        endcase
      end
    end
endmodule
```

7.1.10 Audio_Top.sv

```
module Audio_Top (
   input   OSC_50_B8A,
   inout   AUD_ADCLRCK, AUD_DACLRCK,AUD_BCLK, AUD_I2C_SDAT,
   input   AUD_ADCDAT,
   output  AUD_DACDAT, AUD_XCK, AUD_I2C_SCLK, AUD_MUTE,
       input [3:0]        KEY,
       input [3:0]        SW,
       output [3:0]       LED
);

wire reset = !KEY[0];
wire main_clk;
wire audio_clk;
//wire chipselect = 1;
wire [1:0] sample_end;
wire [1:0] sample_req;
wire [15:0] audio_output;
//wire [15:0] audio_sample;

//Sound samples from audio ROM blocks
wire [15:0] M_city;
wire [15:0] accelerate;
wire [15:0] crash;

//Audio ROM block addresses
wire [14:0] addr_city;
wire [14:0] addr_accelerate;
wire [14:0] addr_crash;

//Store sounds in memory ROM blocks
city c0 (.clock(OSC_50_B8A), .address(addr_city), .q(M_city));
acc a0 (.clock(OSC_50_B8A), .address(addr_accelerate), .q(accelerate));
crash cr0 (.clock(OSC_50_B8A), .address(addr_crash), .q(crash));

//generate audio clock
pll pllclock (
    .refclk (OSC_50_B8A),
    .rst (reset),
    .outclk_0 (audio_clk),
    .outclk_1 (main_clk)
);

//Configure registers of audio codec ssm2603
i2c_av_config av_config (
    .clk (main_clk),
    .reset (reset),
    .i2c_sclk (AUD_I2C_SCLK),
    .i2c_sdat (AUD_I2C_SDAT),
          .status(LED)
);

assign AUD_XCK = audio_clk;
assign AUD_MUTE = 1;



//Call Audio codec interface
audio_codec ac (
    .clk (audio_clk),
    .reset (reset),
    .sample_end (sample_end),
    .sample_req (sample_req),
    .audio_output (audio_output),
    .channel_sel (2'b10),

    .AUD_ADCLRCK (AUD_ADCLRCK),
    .AUD_ADCDAT (AUD_ADCDAT),
    .AUD_DACLRCK (AUD_DACLRCK),
    .AUD_DACDAT (AUD_DACDAT),
    .AUD_BCLK (AUD_BCLK)
);

//Fetch audio samples from these ROM blocks

audio_effects ae (
    .clk (audio_clk),
    .sample_end (sample_end[1]),
    .sample_req (sample_req[1]),
    .audio_output (audio_output),
    //.audio_sample  (audio_sample),
    .addr_city(addr_city),
    .M_city(M_city),
          .addr_accelerate(addr_accelerate),
    .accelerate(accelerate),
          .addr_crash(addr_crash),
    .crash(crash),
          .ctl(SW)
);
endmodule
```

7.1.11 i2c_av_config.sv

```
module i2c_av_config (
    input  clk,
    input  reset,

    output  i2c_sclk, //I2C clock
    inout   i2c_sdat,   // I2C data out
          output [3:0] status
);

reg [23:0] i2c_data;
reg [15:0] lut_data;
reg [3:0]  lut_index = 4'd0;

parameter LAST_INDEX = 4'ha;

reg  i2c_start = 1'b0;
wire i2c_done;
wire i2c_ack;

//Send data to I2C controller
i2c_controller control (
    .clk (clk),
    .i2c_sclk (i2c_sclk),
    .i2c_sdat (i2c_sdat),
    .i2c_data (i2c_data),
    .start (i2c_start),
    .done (i2c_done),
    .ack (i2c_ack)
);

//configure various registers of audio codec ssm 2603
always @(*) begin
    case (lut_index)
        4'h0: lut_data <= 16'h0c10; // power on everything except out
        4'h1: lut_data <= 16'h0017; // left input
        4'h2: lut_data <= 16'h0217; // right input
        4'h3: lut_data <= 16'h0479; // left output
        4'h4: lut_data <= 16'h0679; // right output
        4'h5: lut_data <= 16'h08d4; // analog path
        4'h6: lut_data <= 16'h0a04; // digital path
        4'h7: lut_data <= 16'h0e01; // digital IF
        4'h8: lut_data <= 16'h1020; // sampling rate
        4'h9: lut_data <= 16'h0c00; // power on everything
        4'ha: lut_data <= 16'h1201; // activate
        default: lut_data <= 16'h0000;
    endcase
end

reg [1:0] control_state = 2'b00;
assign status =lut_index;

always @(posedge clk) begin
```

```
    if (reset) begin
        lut_index <= 4'd0;
        i2c_start <= 1'b0;
        control_state <= 2'b00;
    end else begin
        case (control_state)
            2'b00: begin
                i2c_start <= 1'b1;
                i2c_data <= {8'h34, lut_data};
                control_state <= 2'b01;
            end
            2'b01: begin
                i2c_start <= 1'b0;
                control_state <= 2'b10;
            end
            2'b10: if (i2c_done) begin
                if (i2c_ack) begin
                    if (lut_index == LAST_INDEX)
                        control_state <= 2'b11;
                    else begin
                        lut_index <= lut_index + 1'b1;
                        control_state <= 2'b00;
                    end
                end else
                    control_state <= 2'b00;
            end
        endcase
    end
end
```

```
endmodule
```

7.1.12 i2c_controller.sv

```
module i2c_controller (
    input  clk,

    output i2c_sclk,    //i2c clock
    inout  i2c_sdat,    //i2c data  out

    input  start,
    output done,
    output ack,

    input [23:0] i2c_data
);

reg [23:0] data;

reg [4:0] stage;
reg [6:0] sclk_divider;
reg clock_en = 1'b0;

// don't toggle the clock unless we're sending data
// clock will also be kept high when sending START and STOP symbols
assign i2c_sclk = (!clock_en) || sclk_divider[6];
wire midlow = (sclk_divider == 7'h1f);

//reg sdat = 1'b1;
reg sdat = 1'b1;
// rely on pull-up resistor to set SDAT high
```

```
assign i2c_sdat = (sdat) ? 1'bz : 1'b0;

reg [2:0] acks;

parameter LAST_STAGE = 5'd29;

assign ack = (acks == 3'b000);
assign done = (stage == LAST_STAGE);


//implementing I2C protocol
always @(posedge clk) begin
    if (start) begin
        sclk_divider <= 7'd0;
        stage <= 5'd0;
        clock_en = 1'b0;
        sdat <= 1'b1;
        acks <= 3'b111;
        data <= i2c_data;
    end else begin
        if (sclk_divider == 7'd127) begin
            sclk_divider <= 7'd0;

            if (stage != LAST_STAGE)
                stage <= stage + 1'b1;

            case (stage)
                // after start
                5'd0:  clock_en <= 1'b1;
                // receive acks
                5'd9:  acks[0] <= i2c_sdat;
                5'd18: acks[1] <= i2c_sdat;
                5'd27: acks[2] <= i2c_sdat;
                // before stop
                5'd28: clock_en <= 1'b0;
            endcase
        end else
            sclk_divider <= sclk_divider + 1'b1;

        if (midlow) begin
            case (stage)
                // start
                5'd0:  sdat <= 1'b0;
                // byte 1
                5'd1:  sdat <= data[23];
                5'd2:  sdat <= data[22];
                5'd3:  sdat <= data[21];
                5'd4:  sdat <= data[20];
                5'd5:  sdat <= data[19];
                5'd6:  sdat <= data[18];
                5'd7:  sdat <= data[17];
                5'd8:  sdat <= data[16];
                // ack 1
                5'd9:  sdat <= 1'b1;
                // byte 2
                5'd10: sdat <= data[15];
                5'd11: sdat <= data[14];
                5'd12: sdat <= data[13];
                5'd13: sdat <= data[12];
                5'd14: sdat <= data[11];
```

```
                5'd15: sdat <= data[10];
                5'd16: sdat <= data[9];
                5'd17: sdat <= data[8];
                // ack 2
                5'd18: sdat <= 1'b1;
                // byte 3
                5'd19: sdat <= data[7];
                5'd20: sdat <= data[6];
                5'd21: sdat <= data[5];
                5'd22: sdat <= data[4];
                5'd23: sdat <= data[3];
                5'd24: sdat <= data[2];
                5'd25: sdat <= data[1];
                5'd26: sdat <= data[0];
                // ack 3
                5'd27: sdat <= 1'b1;
                // stop
                5'd28: sdat <= 1'b0;
                5'd29: sdat <= 1'b1;
            endcase
        end
    end
end

endmodule
```

7.2 Software Part

7.2.1 hello.c

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <pthread.h>
#include "vga_ball.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include "usbkeyboard.h"
#include <math.h>
#include <time.h>

struct libusb_device_handle *keyboard;
uint8_t endpoint_address;

pthread_t network_thread;
void *network_thread_f(void *);

#define HEIGHT 480
#define WIDTH  640
#define R_MAX 50
#define R_MIN  20

int vga_ball_fd;

/* Read and print the segment values */
void print_segment_info() {
  vga_ball_arg_t vla;
  int i;

  for (i = 0 ; i < VGA_BALL_DIGITS ; i++) {
    vla.digit = i;
    if (ioctl(vga_ball_fd, VGA_BALL_READ_DIGIT, &vla)) {
      perror("ioctl(VGA_LED_READ_DIGIT) failed");
      return;
    }
    printf("%02x ", vla.segments);
  }
  printf("\n");
}

/* Write the contents of the array to the display */
void write_segments(const unsigned char segs[12])
{
  vga_ball_arg_t vla;
  int i;
  for (i = 0 ; i < VGA_BALL_DIGITS ; i++) {
    vla.digit = i;
    vla.segments = segs[i];
    if (ioctl(vga_ball_fd, VGA_BALL_WRITE_DIGIT, &vla)) {
      perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
      return;
    }
  }
}

void write_register(unsigned char value, unsigned short address)
```

```
{
        vga_ball_arg_t vla;
        vla.digit = address;
        vla.segments = value;
        if (ioctl(vga_ball_fd, VGA_BALL_WRITE_DIGIT, &vla)) {
                perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
                return;
        }
}


void score_display (int car_score){
    int score_remainder;
    if (car_score < 10){
    write_register(car_score, 96);
        write_register(0, 95);
    }
    if (car_score >= 10 && car_score < 100){
    write_register(car_score%10, 96);
    write_register(car_score/10, 95);
        write_register(0, 94);
    }
    if (car_score >= 100 && car_score < 1000){
    score_remainder = car_score%100;
    write_register(score_remainder%10, 96);
    write_register(score_remainder/10, 95);
    write_register(car_score/100, 94);
    }
```

```
}

void time_display(int gametime){
        int time_reminder;
        if(gametime < 10){
                write_register(gametime, 90);
        }
        if(gametime >= 10 && gametime < 60){
                write_register(gametime%10, 90);
                write_register(gametime/10, 89);
        }
        if(gametime >=60 && gametime < 599){
                time_reminder = gametime % 60;
                write_register(gametime/60, 88);
                write_register(time_reminder%10, 90);
                write_register(time_reminder/10, 89);
        }
        if(gametime >= 599 && gametime < 5999){
                time_reminder = gametime % 600;
                write_register(gametime/600, 87);
                write_register(time_reminder/60, 88);
                write_register((time_reminder%60)%10, 90);
                write_register((time_reminder%60)/10, 89);
        }
}

void speed_display(int speed){
        int speed_remainder;
        if(speed < 10){
                write_register(speed, 93);
                write_register(0, 92);
                write_register(0, 91);
        }
        if(speed >= 10 && speed < 100){
                write_register(speed%10, 93);
                write_register(speed/10, 92);
                write_register(0, 91);
        }
        if(speed >= 100 && speed < 1000){
                speed_remainder = speed%100;
                write_register(speed_remainder%10, 93);
                write_register(speed_remainder/10, 92);
                write_register(speed/100, 91);
        }
}

//car1 x position
  int car1_x = 320;
  int car1_y = 400;

  int car1_x_left = 320;
  int car1_x_right = 320;

  int arrowl_x = 320;
  int arrowl_y = 260;
  int arrowr_x = 320;
  int arrowr_y = 260;
  int boost = 0;

  int boost_time = 0;

  int boost_num = 0;

  int car_score = 0;

  int left_right_key = 0;

  int corner = 0;

  int score_decre = 0; //signals if score decreases

  int collide_slowtime = 0;


  struct usb_keyboard_packet packet;

  int transferred;

  char keystate[12];

  int stop = 0;

  unsigned int randomnum;




  unsigned int random_number()

  {

  randomnum = rand();

  randomnum = randomnum%66 + 1;

  randomnum = randomnum +287;

  return randomnum;

  }


  void place_sprite(int x, int y, int addr){

        write_register(x, addr);

        write_register(x>>8, addr+1);

        write_register(y, addr+2);
```

```
            write_register(y>>8, addr+3);
 }



int main(){
 vga_ball_arg_t vla;
 int i;
 int j;

 float center_x, center_y, r, v_x, v_y;
 int rgb, dr;
 float smallrx, smallry, mediumrx, mediumry, largerx, largery;
 float smalllx, smallly, mediumlx, mediumly, largelx, largely;
 float smallrx_x, smallry_y, mediumrx_x, mediumry_y, largerx_x, largery_y;
 float smalllx_x, smallly_y, mediumlx_x, mediumly_y, largelx_x, largely_y;
 int cloud1x, cloud1y, cloud2x, cloud2y, cloud3x, cloud3y, cloud4x, cloud4y;
 int car2_x, car2_y;
 int car3_x, car3_y;
 int coin_small_x, coin_small_y, coin_large_x, coin_large_y;
 int count_car2;
 int ltree_smallx, ltree_smally, ltree_mediumx, ltree_mediumy, ltree_largex, ltree_largey;
 float lx_incre, ly_incre, rx_incre, ry_incre;
 int center_incre = 80;
 float tree_location_incre = 1;



 int count_coin;
 int score_uni, score_ten, score_hund;
 int twinkle = 0;
```

```
  int coin_count = 0;

  int starttimer = 0;//control the start of the game

  int gametimer = 0;

  int car1_speed = 1;

  int car2_speed = 2;

  int car3_speed = 3;

  int car1_dist = 0;

  int car2_dist = 0;

  int car3_dist = 0;

  int boost_dist = 0;


  int pass_car2 = 0;

  int pass_car3 = 0;

  int flagx = 320;

  int flagy = 240;

  int speed = 0;

  int speed_fast = 0;

  srand(time(NULL)); //initiate rand

/*


  struct usb_keyboard_packet packet;

  int transferred;

  char keystate[12];


  /* Open the keyboard */


  if ( (keyboard = openkeyboard(&endpoint_address)) == NULL ) {

    fprintf(stderr, "Did not find a keyboard\n");
    exit(1);
  }

//  unsigned char R, G, B;

  center_x = WIDTH/2;
  center_y = HEIGHT/2;
  r = 1;
  dr = 1;
  v_x = 2;
  v_y = 4;

//score
  score_uni = 0;
  score_ten = 0;
  score_hund = 0;
//right trees initial coordinates
  smallrx = 400;
  smallry = 220;
  mediumrx = 480;
  mediumry = 264;
  largerx = 572;
  largery = 317;
//left trees initial coordinates
  smalllx = 240;
  smallly = 220;
  mediumlx = 160;
  mediumly = 264;
  largelx = 68;
  largely = 317;
//clouds
  cloud1x = 520;
  cloud1y = 110;
  cloud2x = 400;
  cloud2y = 150;
  cloud3x = 120;
  cloud3y = 110;
  cloud4x = 240;
  cloud4y = 150;

//left corner tree intia
  ltree_smallx = 60;
  ltree_smally = 220;
  ltree_mediumx = 305;
  ltree_mediumy = 290;
  ltree_largex = 545;
  ltree_largey = 360;

//car2 and car3 initialization
  //car2_x = 330;
  //car2_y = 265;
  car2_x = 430;
  car2_y = 400;
  car3_x = 210;
  car3_y = 400;
//  car2_medium_x = 370;
//  car2_medium_y = 330;
//  car2_large_x = 420;
//  car2_large_y = 400;


//coin_small initialization
  coin_small_x = 320;
  coin_small_y = 250;



  //car1_x = 320;



  rgb = 0xff0000;
  count_car2 = 1;
  count_coin = 1;
  static const char filename[] = "/dev/vga_ball";


//  static unsigned char message[8] = { 0x20, 0x01, 0xf0, 0x00, 0x1e, 0x00, 0xff, 0x00 };
/*  unsigned char message[256] = { center_x, center_x>>8, center_y, center_y>>8, center_x,
center_x>>8, center_y, center_y>>8,center_x, center_x>>8, center_y, center_y>>8};
*/
  printf("VGA LED Userspace program started\n");


  if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
    fprintf(stderr, "could not open %s\n", filename);
    return -1;
  }


  printf("initial state: ");
  print_segment_info();
  printf("current state: ");
  print_segment_info();



/*
  for (i = 0 ; i < 20 ; i++) {
//    unsigned char c0 = message[0];
//    memmove(message, message+1, VGA_BALL_DIGITS - 1);
//    message[VGA_BALL_DIGITS - 1] = c0;
    write_segments(message);
    usleep(400000);
  }
*/
  /* Start the network thread */
  pthread_create(&network_thread, NULL, network_thread_f, NULL);



      write_register(0, 96);
      write_register(0, 95);
      write_register(0, 94);

  while(1){

if(starttimer >= 0 && starttimer <400)
{
//background music
write_register(0,207);
            //define road width
            write_register(50, 189);
```

```
        write_register(50>>8, 190);
        starttimer++;
        usleep(10000);




//score label
place_sprite(580,50,109);

//initial speed
speed_display(0);

//initial time
write_register(0, 90);
write_register(0, 89);
write_register(0, 88);
write_register(0, 87);

//initial road center
place_sprite(320,240,0);
//mph sign
place_sprite(610,432,113);

//inital position
write_register(3, 85);
write_register(3, 86);

//small tree on the right
```

```
place_sprite(smallrx,smallry,60);

//medium tree on the right
place_sprite(mediumrx,mediumry,56);

//large tree on the right

place_sprite(largerx,largery,52);
//small tree on the left

place_sprite(smalllx,smallly,48);
//medium tree on the left

place_sprite(mediumlx,mediumly,44);
//large tree on the left

place_sprite(largelx,largely,40);
//clouds

place_sprite(520,110,64);
place_sprite(400,150,68);
place_sprite(120,110,72);
place_sprite(240,150,76);

//cars

place_sprite(320,400,4); //car1ahead
place_sprite(car2_x,car2_y,16);//car2ahead
place_sprite(car3_x,car3_y,28);//car3ahead
place_sprite(700,200,8); //car1left
place_sprite(700,200,12);//car1right
place_sprite(700,200,20);
place_sprite(700,200,24);
place_sprite(700,200,32);
place_sprite(700,200,36);
place_sprite(700,200,117);
place_sprite(700,200,121);
place_sprite(700,200,125);
place_sprite(700,200,129);
place_sprite(700,200,133);
place_sprite(700,200,137);
place_sprite(700,200,141);
place_sprite(700,200,145);
place_sprite(700,200,149);
place_sprite(700,200,153);
place_sprite(700,200,157);
place_sprite(700,200,161);
place_sprite(700,200,165);
place_sprite(700,200,169);
place_sprite(700,200,173);
place_sprite(700,200,177);
place_sprite(700,200,181);
place_sprite(700,200,185);


//coins disappear
place_sprite(700,240,101);
place_sprite(700,240,105);
```

```
//flag disappear

place_sprite(700,240,97);


//arrow disappear

place_sprite(800,200,191);

place_sprite(800,200,195);

place_sprite(800,200,199);

place_sprite(800,200,203);



if(starttimer >= 0 && starttimer < 200) {


        place_sprite(320,200,80);

        write_register(1,84);//light signal red


}

if(starttimer >= 200 && starttimer < 300) {


        place_sprite(320,220,80);

        write_register(2,84);//light signal yellow


}

if(starttimer >= 300 && starttimer < 400) {


        place_sprite(320,240,80);

        write_register(0,84);//light signal green

}

}
```

```
if(starttimer >=400){
//make light disapear


        place_sprite(700,240,80);

        write_register(0,84);





//gametimer starts to time



        car1_dist+=car1_speed ;

        car2_dist+=car2_speed ;

        car3_dist+=car3_speed ;



//time: slow, normal, speedup

    if(collide_slowtime == 0 && boost == 0)

        {

                //background music

                write_register(0,207);

        usleep(50000);

                gametimer+=2;

                time_display(gametimer/40);

                speed_display(80);

                speed_fast = 0;


        }

    else if (collide_slowtime == 0 && boost == 1 && boost_time < 150)
```

```
{
    usleep(25000);
    boost_time += 1;
            car1_speed = 8;
            car2_speed = 1;
            car3_speed = 2;
            gametimer += 1;
            time_display(gametimer/40);
            if (speed_fast<=100){
            speed_fast+=1;
            speed_display(80+speed_fast);
            }
            write_register(1,207); //speed up sound

}
else{
    usleep(100000);
            gametimer+=4;
            time_display(gametimer/40);
            car1_speed = 0;
            car2_speed = 2;
            car3_speed = 3;
            speed_display(0);
            write_register(0,207);
    }
if(boost_time == 150)
{
    boost = 0;
    boost_time = 0;
```

```
                        car1_speed = 1;

                        car2_speed = 2;

                        car3_speed = 3;

            }


    //printf("%i\n", car1_dist);

    //printf("%i %i\n", (car2_dist-car1_dist)/25, (car3_dist-car1_dist)/25);

    //player_car1

            if(collide_slowtime == 1)

                {

                        place_sprite(700,400,8);

                        place_sprite(700,400,12);

                twinkle++;

                }

            if(twinkle%10 < 5)

                        place_sprite(car1_x,400,4);


            else

                        place_sprite(700,400,4);




            if(twinkle == 50)

            {

                twinkle = 0;

                collide_slowtime = 0;

                boost = 0;

            }
```

```
//collide with other cars
        if((abs(car2_x - car1_x) < 45 && abs(car2_y - car1_y) < 45) || (abs(car3_x - car1_x) < 45 &&
abs(car3_y - car1_y) < 45))
        {
              collide_slowtime = 1;
              car1_x = 320;
                          write_register(2,207); //collide sound
        }

//Player car collide with road
              if (car1_x < ((center_x / (480 - center_y)) * 80 + 64) || car1_x > (640 - ((640 - center_x)/(480 -
center_y)) * 80 - 64)) {

            car1_x = 320;

            collide_slowtime = 1;
                write_register(2,207);  //collide sound

              }

//left arrow display
              if (gametimer > 160)
              {
                   place_sprite(arrowl_x,arrowl_y,191);
                   arrowl_y += ly_incre * 10;
             arrowl_x -= (lx_incre - 0.2) * 10;
             if (arrowl_y == 300)
             {
                          place_sprite(arrowl_x,arrowl_y,195);
               place_sprite(700,400,191);
             }
              }

//right arrow display
              if (gametimer > 560)
              {
                   place_sprite(arrowr_x,arrowr_y,199);
                   arrowr_y += ry_incre * 10;
             arrowr_x += (rx_incre - 0.2) * 10;
             if (arrowr_y == 300)
             {
                          place_sprite(arrowr_x,arrowr_y,203);
               place_sprite(700,400,199);
             }
              }




//corner - left and right

              place_sprite(center_x,center_y,0);

        if (gametimer > 200 && gametimer < 370)     //340
        {
                          car1_x += 6;
            if (gametimer % 22 == 0)
                {
```

```
        center_x -= center_incre;
         if(center_x == 0)
            center_incre = - center_incre;
         corner = 1;
            }

}
else if (gametimer > 600 && gametimer < 770)
        {
                    car1_x -= 6;
          if (gametimer % 22 == 0)
            {
        center_x -= center_incre;
         if(center_x == 640)
            center_incre = - center_incre;
         corner = 2;
            }

}
else
{
    center_x = WIDTH/2;
    corner = 0;
}
```

```
//Tree moving increment
            lx_incre = center_x / sqrt(center_x * center_x + (480 - center_y) * (480 - center_y));
       ly_incre = (480 - center_y) / sqrt(center_x * center_x + (480 - center_y) * (480 - center_y));
       rx_incre = (640 - center_x) / sqrt((640 - center_x) * (640 - center_x) + (480 - center_y) * (480 -
center_y));
       ry_incre = (480 - center_y) / sqrt((640 - center_x) * (640 - center_x) + (480 - center_y) * (480 -
center_y));


//right trees initial coordinates
            smallrx = 640 - ((640 - center_x)/(480 - center_y)) * (480 - 252) + 64;
            smallry = 220;                                  //252
            mediumrx = 640 - ((640 - center_x)/(480 - center_y)) * (480 - 312) + 64;
            mediumry = 264;                                 //312
            largerx = 640 - ((640 - center_x)/(480 - center_y)) * (480 - 381) + 64;
            largery = 317;                                  //381
//left trees initial coordinates
            smalllx = (center_x / (480 - center_y)) * 228 - 64;
            smallly = 220;
            mediumlx = (center_x/(480 - center_y)) * (480 - 312) - 64;
            mediumly = 264;
            largelx = (center_x/(480 - center_y)) * (480 - 381) - 64;
            largely = 317;



       //printf("%f\n",smallry_y);


            //Tree moving
       smallrx_x = smallrx + rx_incre * 10 * tree_location_incre;
```

```
        smallry_y = smallry + ry_incre * 10 * tree_location_incre;

    mediumrx_x = mediumrx + rx_incre * 10 * tree_location_incre;
    mediumry_y = mediumry + ry_incre * 10 * tree_location_incre;

        largerx_x = largerx + rx_incre * 10 * tree_location_incre;
        largery_y = largery + ry_incre * 10 * tree_location_incre;

    smalllx_x = smalllx - lx_incre * 10 * tree_location_incre;
        smallly_y = smallly + ly_incre * 10 * tree_location_incre;

    mediumlx_x = mediumlx - lx_incre * 10 * tree_location_incre;
    mediumly_y = mediumly + ly_incre * 10 * tree_location_incre;

        largelx_x = largelx - lx_incre * 10 * tree_location_incre;
        largely_y = largely + ly_incre * 10 * tree_location_incre;

//small tree on the right
        place_sprite(smallrx_x,smallry_y,60);

//medium tree on the right
        place_sprite(mediumrx_x,mediumry_y,56);

//large tree on the right
        place_sprite(largerx_x,largery_y,52);

//small tree on the left
        place_sprite(smalllx_x,smallly_y,48);
```

```
//medium tree on the left
            place_sprite(mediumlx_x,mediumly_y,44);


//large tree on the left
            place_sprite(largelx_x,largely_y,40);


            if (smallrx_x > mediumrx)
            {


//right trees initial coordinates
            smallrx_x = smallrx;
            smallry_y = smallry;                                          //252
            mediumrx_x = mediumrx;
            mediumry_y = mediumry;                                          //312
            largerx_x = largerx;
            largery_y = largery;                                          //381
//left trees initial coordinates
            smalllx_x = smalllx;
            smallly_y = smallly;
            mediumlx_x = mediumlx;
            mediumly_y = mediumly;
            largelx_x = largelx;
            largely_y = largely;


       tree_location_incre = 0;
            }


            tree_location_incre = tree_location_incre + 1.0;
```

```
//car head direction

            if (left_right_key == 1 && collide_slowtime == 0)
            {
                  place_sprite(700,400,4);
                  place_sprite(700,400,12);
          place_sprite(car1_x,400,8);

            }
        else if (left_right_key == 0 && collide_slowtime == 0)
        {
          place_sprite(car1_x,400,4);
                  place_sprite(700,400,8);
            }



            if (left_right_key == 2 && collide_slowtime == 0)
            {
                  place_sprite(700,400,4);
                  place_sprite(700,400,8);
          place_sprite(car1_x,400,12);

            }
        else if (left_right_key == 0 && collide_slowtime == 0)
        {
          place_sprite(car1_x,400,4);
                  place_sprite(700,400,12);
            }
```

```
                left_right_key = 0;

////////////////////////////////////////

        //place_sprite(700,400,12);

        place_sprite(700,400,20);

///////////////////////////////////////

//Cloud moving
        //cloud1x = cloud1x + 5;
        //cloud1y = cloud1y + 5;
        cloud2x = cloud2x + 5;
        cloud2y = cloud2y - 2;
        //cloud3x = cloud3x - 5;
        //cloud3y = cloud3y
        cloud4x = cloud4x - 5;
        cloud4y = cloud4y - 2;

//cloud 1 large
if (cloud2x >= 520){

        place_sprite(cloud2x,cloud2y,64);
        place_sprite(700,cloud2y,68);


        place_sprite(cloud4x,cloud4y,72);


        place_sprite(700,cloud4y,76);
        }
        if (cloud2x >= 640){


        place_sprite(750,cloud2y,64);


        place_sprite(750,cloud4y,72);
        }
//cloud 2 small
        place_sprite(cloud2x,cloud2y,68);


//cloud 3 large
/*if (cloud4x >= 120){
        write_register(cloud4x, 72);
        write_register(cloud4x>>8, 73);
        write_register(cloud4y, 74);
        write_register(cloud4y>>8, 75);


        write_register(700, 76);
        write_register(700>>8, 77);
        write_register(cloud4y, 78);
        write_register(cloud4y>>8, 79);
}
*/
```

```
//cloud 4 small
        place_sprite(cloud4x,cloud4y,76);


if (cloud2x == 640){
  // cloud1x = 520;
  //cloud1y = 110;
  cloud2x = 400;
  cloud2y = 150;
  //cloud3x = 120;
  //cloud3y = 110;
  cloud4x = 240;
  cloud4y = 150;
}


//At the beginning of the game
if(gametimer >= 0 && gametimer <= 300){


//speed
speed++;
if(speed<=80) speed_display(speed);




//car1 and car2
        place_sprite(car2_x,car2_y,16);
        if (car2_y >= 310 && car2_y < 385){


        place_sprite(700,700,16);


        place_sprite(car2_x,car2_y,129);
```

```
            }
            if (car2_y >= 265 && car2_y < 310){
            place_sprite(700,700,129);

            place_sprite(700,700,16);


            place_sprite(car2_x,car2_y,165);
            }
            if(car2_y <265){
            place_sprite(700,700,16);


            place_sprite(700,car2_y,165);
            }


                    car2_x = car2_x - 2;
            car2_y = car2_y - 3;

    //car1 and car3


            place_sprite(car3_x,car3_y,28);
            if (car3_y >= 310 && car3_y < 385){


            place_sprite(700,700,28);


            place_sprite(car3_x,car3_y,141);


            }
            if (car3_y >= 265 && car3_y < 310){
```

```
        place_sprite(700,700,141);


        place_sprite(700,700,28);


        place_sprite(car3_x,car3_y,177);
        }
        if(car3_y <265){
        place_sprite(700,700,28);


        place_sprite(700,car3_y,177);
        }


                car3_x = car3_x + 2;
        car3_y = car3_y - 3;




}


if(gametimer>310 && gametimer <=312){
//reset car2 and car3 position to top
  car2_x = -330;
  car2_y = 265;
  car3_x = -310;
  car3_y = 265;
}


if(gametimer >312){
```

```
//car1 and car2 position relation

if(((car2_dist - car1_dist)/25<=5 && (car2_dist - car1_dist)/25>=0)|| ((car1_dist-car2_dist)/25 <= 20 &&
(car1_dist-car2_dist)/25>=0)){


pass_car2 = 1;

}


if(pass_car2 == 1){

        if(car2_y<=265){

                place_sprite(700,car2_y,129);

                place_sprite(700,car2_y,16);

                place_sprite(700,car2_y,165);

        }

        if(car2_y>265 && car2_y < 310){

                place_sprite(car2_x,car2_y,165);

                place_sprite(700,car2_y,129);

                place_sprite(700,car2_y,16);

        }

        if(car2_y>=310 && car2_y < 385){

                place_sprite(car2_x,car2_y,129);

                place_sprite(700,car2_y,16);

                place_sprite(700,car2_y,165);


        }

        if(car2_y>=385 && car2_y < 530){
```

```
                        place_sprite(car2_x,car2_y,16);

                        place_sprite(700,car2_y,129);

                        place_sprite(700,car2_y,165);


                }

                if(car2_y>=530){

                        place_sprite(700,car2_y,129);

                        place_sprite(700,car2_y,16);

                        place_sprite(700,car2_y,165);

                }




        if(car2_speed > car1_speed){

                car2_x = abs(car2_x) - 2;

            car2_y = car2_y - 3;

        if(car2_x < 330) car2_x = -330;

        if(car2_y < 265) {

                car2_y = 265;

                pass_car2 = 0;

                }

        }

        if(car1_speed > car2_speed){

                car2_x = abs(car2_x) + 2;

            car2_y = car2_y + 3;

        if(car2_x > 495) car2_x = -495;

        if(car2_y > 512) {

                car2_y = 512;

                pass_car2 = 0;
                }
        }

        }
        //car1 and car3 position relation

        if(((car3_dist - car1_dist)/25<=20 && (car3_dist - car1_dist)/25>=0)|| ((car1_dist-car3_dist)/25 <= 5&&
        (car1_dist-car3_dist)/25>=0)){

        pass_car3 = 1;
        }

        if(pass_car3==1){
                if(car3_y<=265){
                        place_sprite(700,car3_y,141);
                        place_sprite(700,car3_y,28);
                        place_sprite(700,car3_y,177);
                }
                if(car3_y>265 && car3_y < 310){
                        place_sprite(car3_x,car3_y,177);
                        place_sprite(700,car3_y,141);
                        place_sprite(700,car3_y,28);
                }
                if(car3_y>=310 && car3_y < 385){
                        place_sprite(car3_x,car3_y,141);
                        place_sprite(700,car3_y,28);
                        place_sprite(700,car3_y,177);
```

```
            }
            if(car3_y>=385 && car3_y < 530){
                        place_sprite(car3_x,car3_y,28);
                        place_sprite(700,car3_y,141);
                        place_sprite(700,car3_y,177);


            }
            if(car3_y>=530){
                        place_sprite(700,car3_y,141);
                        place_sprite(700,car3_y,28);
                        place_sprite(700,car3_y,177);
            }




if(car3_speed > car1_speed){
            car3_x = abs(car3_x) + 2;
      car3_y = car3_y - 3;
if(car3_x > 310) car3_x = -310;
if(car3_y < 265) {
            car3_y = 265;
            pass_car3 = 0;
            }
}
if(car1_speed > car3_speed){
            car3_x = abs(car3_x) - 2;
      car3_y = car3_y + 3;
if(car3_x < 145) car3_x = -145;
if(car3_y > 512) {
		car3_y = 512;

		pass_car3 = 0;

		}

}

}

//Detecting player's position

if(car2_y > 400 && car3_y < 400) write_register(2, 85);

if(car2_y > 400 && car3_y > 400) write_register(1, 85);

if(car2_y < 400 && car3_y < 400) write_register(3, 85);

//coin

 coin_count = coin_count + 10;

int a = 0;

int coin_flag;

while(a<100000){

if(coin_count == a){

		coin_small_y = 250;

		coin_small_x = random_number();

coin_flag=1;

}

a = a+850;

}

if(coin_flag==1 && corner == 0){
    coin_small_y += 5;
    if(coin_small_x >= 310&&coin_small_x<=330){
    place_sprite(coin_small_x,coin_small_y,105);

    if (coin_small_y >= 310 ){

    place_sprite(700,700,105);

    place_sprite(coin_small_x,coin_small_y,101);
}
    }
    if(coin_small_x < 310){
    coin_small_x -= 4;
    place_sprite(coin_small_x,coin_small_y,105);

    if (coin_small_y >= 310 ){

    place_sprite(700,700,105);

    place_sprite(coin_small_x,coin_small_y,101);
}
    }
    if(coin_small_x > 330){
    coin_small_x += 4;
    place_sprite(coin_small_x,coin_small_y,105);

    if (coin_small_y >= 310 ){

    place_sprite(700,700,105);
```

```
            place_sprite(coin_small_x,coin_small_y,101);
        }
            }
            if (coin_small_y == 504){
            coin_flag = 0;
            }
}


//check if player car hits the coin

 if (abs(coin_small_x - car1_x) < 48 && abs(coin_small_y - car1_y) < 48){

            coin_small_y = 700;

            place_sprite(700,700,105);

            place_sprite(700,coin_small_y,101);
            car_score++;

}

 if ((abs(coin_small_x - car2_x) < 48 && abs(coin_small_y - car2_y) < 48) || (abs(coin_small_x - car3_x) <
48 && abs(coin_small_y - car3_y) < 48))
 {
            coin_small_y = 700;

            place_sprite(700,coin_small_y,105);
            place_sprite(700,coin_small_y,101);
 }

score_display(car_score);
boost_num = car_score / 5;

if(score_decre == 1){
             boost_num--;
             car_score = car_score - 5;
             score_decre = 0;
}




//car3smallright
            place_sprite(700,700,185);

//light signal
            place_sprite(700,700,80);
            write_register((j++)%2, 84);


//End of game

if(gametimer>2400){
```

```
flagy+=10;

place_sprite(flagx,flagy,97);


if(flagy == 450)

{

        write_register(0,207);

        break;

}


}




//ESC: Stop the game

    if(stop == 1)

      break;


}

}


 }






 /* Terminate the network thread */

 pthread_cancel(network_thread);


 /* Wait for the network thread to finish */

 pthread_join(network_thread, NULL);
```

```
  printf("VGA LED Userspace program terminating\n");
 // printf("%d\n",&());
  return 0;
}

void *network_thread_f(void *ignored)
{
 for (;;) {
      libusb_interrupt_transfer(keyboard, endpoint_address,
                                          (unsigned char *) &packet, sizeof(packet),
                                          &transferred, 50);

      if (packet.keycode[0] == 0x29) { /* ESC pressed? */
               stop = 1;
          break;
      }
//left_arrow

      if (packet.keycode[0] == 0x50) {
               car1_x = car1_x - 10;
             //  place_sprite(car1_x,400,8);
       //  place_sprite(700,400,4);
               left_right_key = 1;

      }

//right_arrow
      if (packet.keycode[0] == 0x4f) {
               car1_x = car1_x + 10;
```

```
            //  place_sprite(car1_x,400,12);
       //  place_sprite(700,400,4);
             left_right_key = 2;
     }

//up_arrow
     if (packet.keycode[0] == 0x52) {
       if(boost_num > 0 && corner == 0 && collide_slowtime == 0){
             boost = 1;
        boost_time = 0;
             score_decre = 1;
             }
     }
  }



//down arrow
     //if (packet.keycode[0] == 0x51) {

//ESC
 /*    if (packet.keycode[0] == 0x29) {
            //fbputs("Exit the Chat Room", display_row++, 0, 'R', 1);
            break;
     } */
  return NULL;
}
```