Air-Raid Project

Jakob Stiens, Yongmao Luo, Zhaomeng Wang, Tristan Saidi

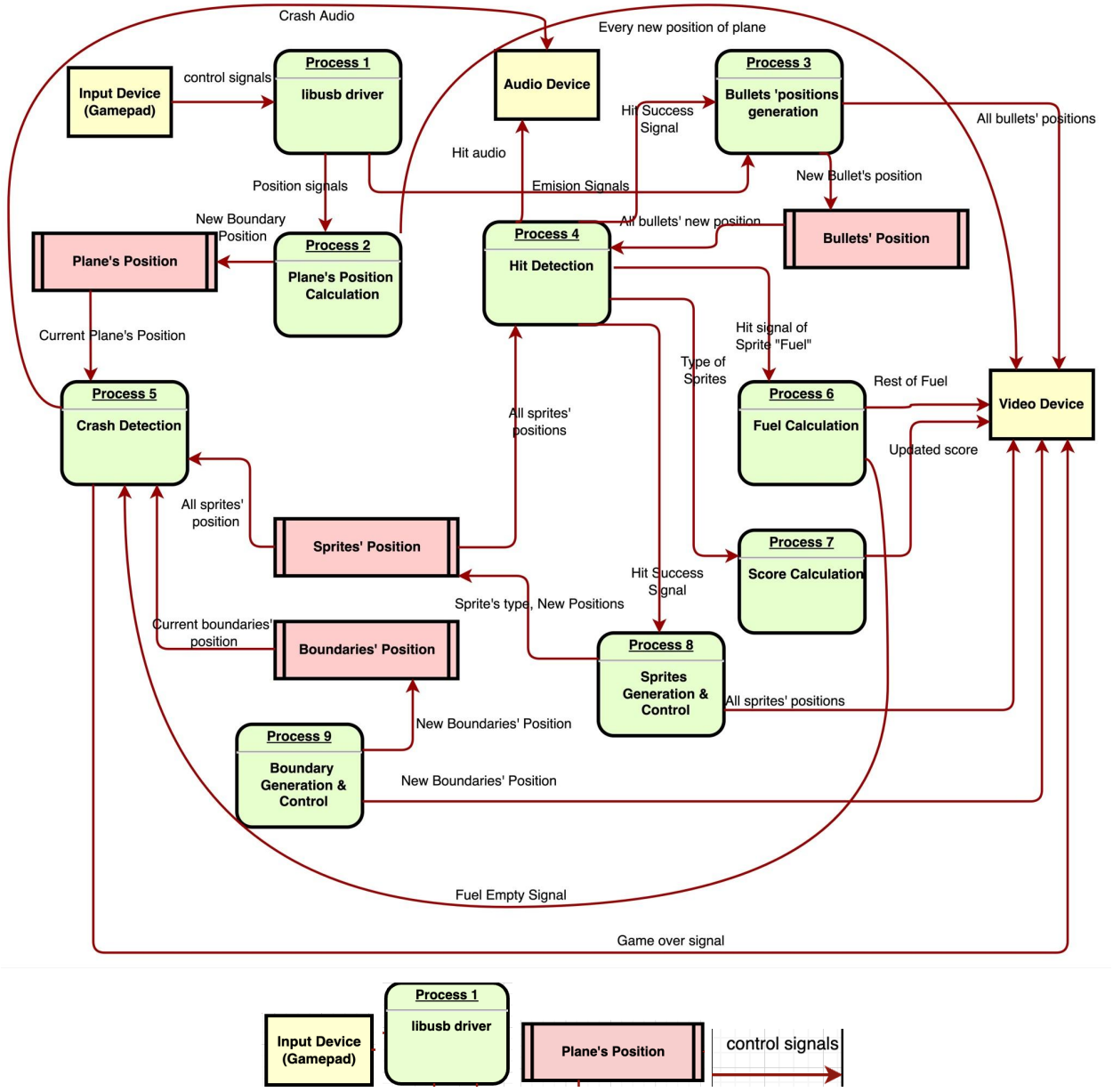
Contents

1. Introduction

We present Air-Raid, a variation of the Atari game “River Raid”. The premise of the game is that you control a plane flying above a river. The river is inbounds and flying over the land is out of bounds and will cause the plane to crash. There are also obstacles on the river (such as battleships and enemy airships) that cause the plane to crash if it runs into them. To defend itself, the plane can shoot and destroy obstacles on the river. The plane also has a fuel gauge that slowly decreases over time - if this gauge hits zero, the plane will crash. There are fuel sprites that randomly appear on the river which the plane can pick up to regenerate fuel. For every obstacle shot and the amount of time the player survives for, the score will increase. The goal of the game is to get as high of a score as possible before crashing.

To compile this project, firstly you need to put all software files in the original hierarchy. Run “cmake ./” in the directory containing “CMakeList.txt”. Then, run “make” in the same directory, which will generate the executable file in the “bin” directory. To compile the drivers, go to the directory “XboxController” and “VideoController” and do “make” in each folder separately. Then, install each module with the names “xpad.ko” and “water_video.ko” (with insmod). For compilation, run quartus and qsys inside of the hardware folder. Then generate the HDL from qsys and run quartus to generate the .dtb file. Then run “make rbf” to generate the .rbf file.
As a note, to run the hardware files, firstly you need to mount the “/dev/mmcblk0p1” to “/mnt”, and you will need to scp the .rbf and .dtb files to the /mnt directory of the board and reboot.

2. Game Logic Explanation



KEY: Yellow Block: External Entity. Green Block: Process. Red Block: Data Store. Arrow: Data Flow

Figure 2-1 Data Flow Diagram of the Air-Raid project

Process 1: Control Input & Driver:

Users can control the horizontal position of the plane with an Xbox Controller. The plane is coded to move left when the “X” button is pressed, and right when the “B” button is pressed. Users can also shoot bullets by pressing the “Y” button of the controller and the “A” button is used to start the game.

Since the Xbox Controller is not an open source device, it is hard for us to directly use it through generic USB interfaces like the interface for the keyboard (lsusb). Thankfully, the project [1] by Pavel has provided us with full support for all kinds of different Xbox controllers on Linux. We used this driver as a base and developed on top of it. For all kinds of Xbox Controllers, this driver can create an event device for each controller and the path is “/dev/input/event*”. Since in this project we just use one controller, the path should be “/dev/input/event0”. The event device will output data each time there is an event coming from the outside devices, and the format of the data is also fixed and shown as follows:

```
typedef struct {
    struct timeval time;
    unsigned short type;
    unsigned short code;
    unsigned int value;
}inputEvent;
```

Figure 2-2 the data structure of event device in Linux

Each button and joystick on the Xbox controller have a different unique code. As a result, if we press the button, the value for that code will become a 1. If we release the button, that value will become 0. Since the original driver outputs all kinds of data from the controller which may delay the arrival of data of the button we used for this project, we have commented some lines (line 972 - 981) to block reading the data of the controller's sticks.

In addition to the driver for the Xbox Controller, we also have the driver for our video device and audio device. The way we implement these drivers is to use “ioctl” to realize the functionality of writing data to hardware registers. Since all variables are no more than 16 bits, we make use of a “writedata” field and an “address” field. The addresses are predefined in the driver file called “water_video.c” in the “VideoDriver” folder. These addresses are shown in Figure 2-3. Sometimes we do not use the macro for addressing, since for some variables they have some numerical relationship that we can make use of (such as the one shown in Figure 2-4).

```
/* Device registers */
#define BOUNDARY0(x) (x)
#define BOUNDARY1(x) ((x)+2)
#define BOUNDARY2(x) ((x)+4)
#define BOUNDARY3(x) ((x)+6)
#define SHIFT(x) ((x)+8)
#define SCOREBOARDX(x) ((x) + 64)
#define SCOREBOARDY(x) ((x) + 66)
#define DIGIT1X(x) ((x) + 68)
#define DIGIT1Y(x) ((x) + 70)
#define DIGIT1IMG(x) ((x) + 72)
#define DIGIT2X(x) ((x) + 74)
#define DIGIT2Y(x) ((x) + 76)
#define DIGIT2IMG(x) ((x) + 78)
#define DIGIT3X(x) ((x) + 80)
#define DIGIT3Y(x) ((x) + 82)
#define DIGIT3IMG(x) ((x) + 84)
#define FUELGAUGEX(x) ((x) + 86)
#define FUELGAUGEY(x) ((x) + 88)
#define INDICATORX(x) ((x) + 90)
#define INDICATORY(x) ((x) + 92)
#define SHOOTAUDIO(x) ((x)+94)
#define HITAUDIO(x) ((x)+96)
#define EXPLODEAUDIO(x) ((x)+98)
```

Figure 2-3 Addresses of different hardware variables

```
static void writePosition(water_video_arg_position *arg)
{
    // index of sprites starting from x+10
    iowrite16(arg->pos.x, dev.virtbase+10+arg->index*6 );
    iowrite16(arg->pos.y, dev.virtbase+10+arg->index*6+2 );
    iowrite16(arg->type, dev.virtbase+10+arg->index*6+4 );

    dev.argPosition = *arg;
}
```

Figure 2-4 A function in the driver using the numerical relationship between different sprites

The code of the Video and Audio Controller is as follows:

Process 2: Plane's Position Calculation:

The plane is designed to maintain the same vertical position and have the background scroll to give the appearance of moving forwards just like the original game. At the beginning of the game, the plane will fly into the scene from the bottom of the background until it reaches the 300th vertical line from the top. After receiving the Xbox Controller's input from the user, we need to calculate the plane's position. The button "X" and "B" of the Xbox Controller control the movement of the plane. Each time when pressing these buttons, the horizontal position of the plane will be changed. If pressing "X", the position will go right, and pressing "B" will go left. Holding each button will continuously move towards the corresponding direction with speed related to the update frequency of the background. What's more, during the process, this module continuously sends the position of the plane to our dedicated video device through the video driver so we can display the plane on the screen.

Process 3: Bullets' Position Generation:

Bullets are shot from the plane when the user pushes the shoot button. When receiving such a signal, we generate a new bullet from the current plane's position, and let it move automatically towards the top of the screen. The shape of the bullet is 10*2 pixels. In this game, we limited the user to only shoot three bullets once which means you can only shoot three bullets and must wait for them to hit something or fly off screen before you can shoot again. We are treating this as the "reload" time. Also, when receiving the hit signal from the Hit Detection module, it means that there are some bullets hitting some sprites. We then set the positions of those bullets and sprites out of the screen so that they disappear. And when the program finds that the bullet flew out of the screen it will automatically make the bullet disappear. This module should continuously send the position of all the bullets to the video device so they can display the bullets on the screen.

Process 4: Hit Detection:

When shooting bullets, we need to determine if the bullets hit any sprites. We are going to implement this part by comparing the positions of the bullets relative to the sprites. This means that the module should read the position information stored for both the sprites and the bullets. If sprites are hit, this module will make the sprites disappear and add scores. Also, it will send the play audio signal to the audio device to initiate the sound effect for the explosion .

Process 5: Crash Detection:

In this game, the plane is easy to crash. If the plane hits the boundaries of the river, enemy sprites, or runs out of fuel, the plane crashes. Thus, every iteration we need to compare the plane's horizontal position with the current row's boundaries and all sprites' positions to see if the player has crashed or not. Meanwhile, in each iteration, we will query the fuel of the plane to see if it has run out to make the plane crash. If the plane crashes, the software will send a signal to hardware so that the video can display the "explosion" image at the position of the plane and make the plane disappear. Also, it will send the audio signals to the audio device to generate the sound effect of exploding.

Process 6: Fuel Calculation:

For the plane, when its fuel tank is full, it can contain 70 units of fuel. Each second the plane will consume one unit of fuel. When the player collides with the fuel tank sprite, the player will receive additional 10 units of fuel as a reward. However, if the player shoots the fuel tank sprite, it will not gain any additional fuel. The current remaining fuel amount will be shown at the top of the game area as an indicator. If the indicator is pointing to "F", meaning that the fuel tank is full. If the indicator is pointing to "E", it indicates the fuel tank is empty and the plane crashes.

Process 7: Score Calculation:

The score for the game will increase whenever the player destroys (hits and reduces its HP to 0) any enemy plane and battleships. Different types of targets will have different scores, for the enemy airplane which only has 1 HP, the score is 1 as well; while the battleship has 2 HP so its score is 2. The current score will be shown at the bottom of the game area, so everytime when the score is updated, the software will send a new integer to each score bit. Each bit will range from 0 to 9 and will control the sprite depicting that number in the score.

Process 8: Sprites Generation & Control:

There are three kinds of sprites in our game system: enemy airplanes (which include two kinds of images: helicopter and balloon), battleships and fuel tanks. Except for fuel tanks, all the targets should be shot while playing and have different score values (the fuel tank can be hit and destroyed but will not provide any points). All three kinds of sprite shape are 32*32 pixels. The generation and control of the sprites are done in software. During the process of the game, enemy airplanes and battleships will be generated randomly in the upper game area, following a specific frequency (35 game cycle). The software will send all the positions of all sprites to the video device each round to update their location. And in our design, the maximum number of sprites per showing screen is five in consideration of the game difficulty.

Process 9: Boundary Generation & Control:

For our game we are assuming the resolution our game is 640*480. In our game system, the game scenario (a.k.a. game background) is represented by 4 boundary variables. This allows for the river to have up to two branches at a time. There are 2 types of terrain in the game: ground and river. The boundary between the ground and river is represented by pairs of variables which is the horizontal coordinate. For example, variables with values 10, 100, 200 and 400 means: in that pixel row, the pixels located in 0-9, 100-199 and 400-639 are ground and in 10-99 and 200-399 are rivers. Also, variables with value 100, 200, 0 and 0 means: the pixels located in 0-99 and 200-639 are ground and in 100-199 is the river. If the third and fourth boundary variables are both set to 0, there will only be a single river branch. As time goes on, and the game scenario goes back (scroll down) at a rate of 60 pixel rows per second. The software will send all the variables of new boundaries to the video device each round to update the boundaries.

3. Interfaces & Software Implementation:

Interface between SW and Video Device (explained further in the Hardware Section):

Since we are using a 16 bits wide interface for writing data to the hardware, the address should add up by 2 for each one, since one address points to a block of memory with 8 bits. The address table is defined as follows:

1. 4 registers for boundaries for each row (with address 0 to 6)
2. A shift register which is used to signify the hardware to update the background (address 8)
3. Variables for sprites. We can at most show 9 sprites at the same time on the screen, including the plane, bullets, fuel tanks and all kinds of enemy sprites. For each sprite, there are 3 variables: the first two variables are positions, and the third variable is the index of image, since in hardware we use different indexes to show different images. (address 10-62)
4. Variables for score sprites. We will display a scoreboard on the lower right corner with 3 digits. There are 2 variables for the position of the scoreboard (address 64-66), and for each digit we have 3 variables (address 68-84).
5. Variables for fuel sprites. We have a sprite indicating the fuel gauge, with 2 variables defining its position (address 86-88). We also have an indicator of the fuel gauge, indicating the amount of fuel the plane has. We also have 2 variables to define the position of the indicator(address 90-92).
6. Variables for audio. We have 3 variables, indicating 3 different kinds of audio effect, the shooting audio, hitting audio and explosion audio (address 94-98).

Object-Oriented Implementation

Figure 3-1 UML Class Diagram of River-Raid Project

In this section, we will fully discuss the system by using an object-oriented approach. The UML diagram is shown above in Figure 3-1, which illustrates the detailed design of the system. The whole system can be divided into several classes, and synthesis these classes in the main.cpp file.

Main.cpp

The main game logic is located in the main.cpp file. Since we need to update the rows in 60Hz, we use a counter and only when the time of the counter has reached 1/60 second, the game logic is executed. For each iteration, we will scroll down the background by one row, and read position changing and shooting signals from the Xbox Controller. Meanwhile, we reduce the plane's fuel each second and add the fuel by 10 if it bumps into the fuel tank. What's more, we also need to examine the collision between the plane and the boundary, between the plane and all enemy sprites as well as between the bullets and all sprites except for the plane. If the plane is crashed, we need to make the sprite of the plane disappear and show

an explosion sprite at the same location. If the user shoots a bullet, we need to create a bullet sprite. If a sprite is hit, we need to remove this sprite and add a corresponding score. Each round the bullet should move upwards while the sprites except for planes should move down one row just the same as the background.

GameScenario Class

The GameScenario class contains a main attribute “boundaries” which represents the boundary of each vertical line. Since it is a 640*480 screen, the variable “boundaries” is an array with length 480. The function “updateBackground” is designed to update the game scenario, making the game area scroll down. The function maintains a state machine to record if the game scenario has one or two rivers, while the width of the river to decide how to generate the next boundaries of the game scenario. The selection of different states is chosen randomly to make the game more interesting. The function “initBackground” initializes the background each time when a new game starts. It will set the river as a single river with left and right boundaries 220 and 420.

The relationship between this class and the Airplane class is an 1-to-1 relation which means there is only one player controlled airplane in the game. The relationship between this class and the Sprite class is a 1-to-n relationship, which means there are more than one sprite in the game.
The codes are shown as follows:

Airplane Class

The Airplane class is one of the most important classes, it is responsible for the player controlled airplane through the whole game. The attribute Type shows which type of the object it is, for example type 0 means it is the Airplane, type 1 means it is the Bullet and so on. The attribute Position is the coordinates of the current position of the plane in the game area. The attribute “fuel” is the current fuel amount of the plane. The attribute Score is the current score the player gained in the game. The function “fire” is triggered when the player wants to shoot the bullet. The function “receivePos” is used to receive control signals from the Xbox Controller and control the airplane’s position(left or right). Function “isCrashed " is used to check the airplane to see if it crashed or not, and gives a bool return value. The airplane can shoot many bullets, so the relationship between Airplane and Bullet is a 1-to-n relationship.

The Bullet class implements the bullet the airplane shot in the game. The attribute Type shows which type of the object it is, Bullet has type value of 1. The attribute Position is the coordinates of the current position of the bullet in the game area. The function fly() is used to set the position of the bullet, it provides the track of the bullet, at any given time the position can be obtained by using fly().

The Sprite class is a parent class which is inherited by 3 child classes. The attributes type show what object it is, type 2 means it is the EnemyPlane; type 3 means the Battleship and type 4 means the FuelTank. The attribute hitPoint shows how many hits it needs to be destroyed and get its corresponding rewarding score. Because there are more than one of each kind of sprite, we need attribute ID to identify each individual object. The usage of attribute position is the same with the others class. The attribute width and length represents the width and length of the pixel of each object. The attribute isHit is used to show if the object is hitted by the bullet or not. The function generate() and disappear() is used to generate the object randomly within the predefined proper region or make it vanish when the plane fly passes and the game scenario scrolls down. The function move() is the same as the counterpart function in airplane class. And the function checkIfHit() is used to check if the object is hitted or not and gives the bool return value.

In those three child classes, EnemyPlane, Battleship and FuelTank, they all have the attributes and functions in the parent class Sprite. Besides, the EnemyPlane and Battleship have attributes of score which means the rewarding score it has when destroyed by the player. The class FuelTank has the attribute of fuelVolume which is the amount of fuel that should be added if the plane hits the Fuel Tank sprite.

Sprite Class

In this class, we defined the parent class of those three kinds of sprites: Enemy Plane, Enemy Battleship and Fuel Tank. Specifically, we set those following member variables: type, hitPoint, shape, left, canMove, index, isDestroy, and Pos. The "left" indicates the moving direction and the "canMove" flag indicates if the instance can move (because we want to set several still instances which can not move in order to make the game more versatile). And the "index" means the position of the instance in the interface between hardware and software. "isDestroy" shows if the instance is destroyed and "Pos" is the abbreviation of "Position" which means the position of the instance in the screen.

Besides "setter" and "getter" , the most important part of this class is the functions "move", "generate" and "disappear", which are used to make the sprites move in the game scenario or randomly generate new sprites in the game scenario or make a sprite disappear. The function "move" takes two parameters which are the boundary of the river in the row where spirits exist and the minimum width of the river in the row where spirits exist. That is because we limit the battleship and fuel tank must stay within the river area, so we need to know the boundary and the minimum width of the river in order to make sure that those kinds of sprites are in the river area. Except the boundary, the function "generate" takes another parameter "y" which is the vertical coordinates of the sprite in the screen and then it will generate the sprite in a random position in horizontal (random "x" value). The function is very simple, which just sets the "y" value indicating the vertical coordinates of the sprite to 0.

The code of those three functions are shown in the appendix

EnemyPlane Class/Battleship Class/FuelTank Class

All those three classes inherit from the SpriteClass, therefore have all the aforementioned member variables and functions in SpriteClass. The EnemyPlane and Battleship class are almost the same, they all have a variable score which indicates the score should be added when the player shoots it down. An important function called "checkIfHit" is used to detect whether the instance is hitted by any bullet or not, and if after detecting the hitting it will reduce the HP or make the instance disappear and add the score to the player if its HP comes to zero.

The function "checkIfHit" is shown in the appendix.

The FuelTank class is a little bit different from those aforementioned classes, which have a fuelVolume variable instead of score, meaning that the fuelVolume of that instance. When the player bumps into that instance, it will add the related fuel volume to the player.

Bullet Class

The bullet class has “type”, “sp”, “isCrashed”, “Pos” and “index” member variables, which indicated the type(bullet in this case), shape, position and index of the instance and “isCrashed” is the flag that help us to determine whether the bullet has flew out of the game scenario or hitted any sprites. The very simple but essential function is “fly” which makes the bullets fly against the game scenario vertically.

The code of the “fly” function is shown in the appendix.

4. Hardware

a. Hardware Overview

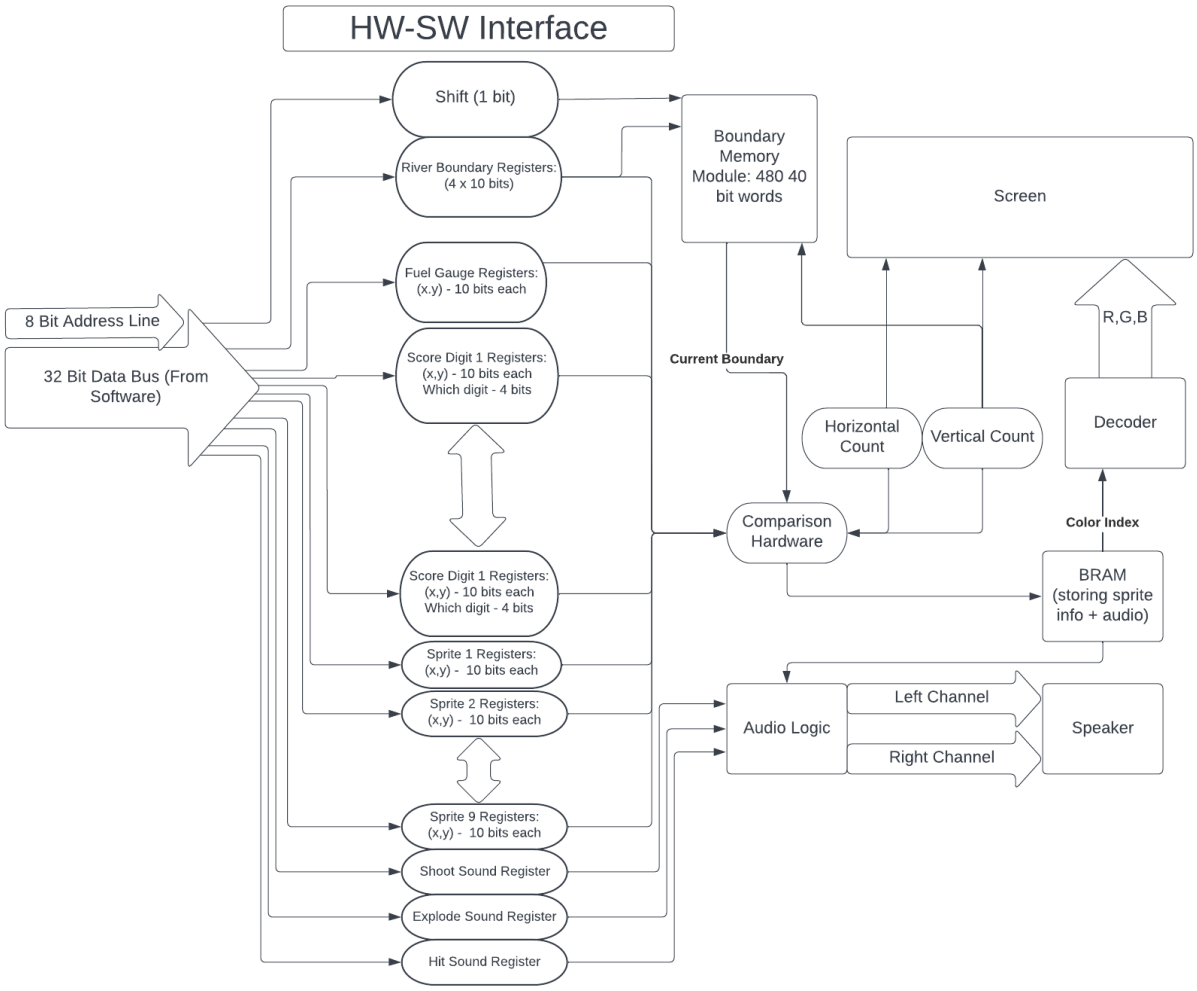


Figure: Hardware System design

Ultimately, the hardware was handled by displaying information from software on the VGA monitor as we initially intended. The software and hardware interface via a 16 bit data bus wired to about 50 registers, all which control some aspect of what needs to be displayed. Some examples of these registers include sprite_i's x and y positions, what image should be pulled from memory for sprite_i, and next river boundary locations. A full list of these registers is seen in the figure below. Similar to Lab 3, the register being written to is indicated by an address signal, which in our case has a width of 6 bits.

```
always_ff @(posedge clk) begin
  if (chipselect && write)
    case (address)

      6'd0 : boundary_1_IN          <= writedata[9:0];
      6'd1 : boundary_2_IN          <= writedata[9:0];
      6'd2 : boundary_3_IN          <= writedata[9:0];
      6'd3 : boundary_4_IN          <= writedata[9:0];
      6'd4 : shift                  <= writedata[0];
      6'd5 : sprite1_x              <= writedata[9:0];
      6'd6 : sprite1_y              <= writedata[9:0];
      6'd7 : sprite1_img            <= writedata[4:0];
      6'd8 : sprite2_x              <= writedata[9:0];
      6'd9 : sprite2_y              <= writedata[9:0];
      6'd10 : sprite2_img           <= writedata[4:0];
      6'd11 : sprite3_x             <= writedata[9:0];
      6'd12 : sprite3_y             <= writedata[9:0];
      6'd13 : sprite3_img           <= writedata[4:0];
      6'd14 : sprite4_x             <= writedata[9:0];
      6'd15 : sprite4_y             <= writedata[9:0];
      6'd16 : sprite4_img           <= writedata[4:0];
      6'd17 : sprite5_x             <= writedata[9:0];
      6'd18 : sprite5_y             <= writedata[9:0];
      6'd19 : sprite5_img           <= writedata[4:0];
      6'd20 : sprite6_x             <= writedata[9:0];
      6'd21 : sprite6_y             <= writedata[9:0];
      6'd22 : sprite6_img           <= writedata[4:0];
      6'd23 : sprite7_x             <= writedata[9:0];
      6'd24 : sprite7_y             <= writedata[9:0];
      6'd25 : sprite7_img           <= writedata[4:0];
      6'd26 : sprite8_x             <= writedata[9:0];
      6'd27 : sprite8_y             <= writedata[9:0];
      6'd28 : sprite8_img           <= writedata[4:0];
      6'd29 : sprite9_x             <= writedata[9:0];
      6'd30 : sprite9_y             <= writedata[9:0];
      6'd31 : sprite9_img           <= writedata[4:0];
      6'd32 : scoreboard_x          <= writedata[9:0];
      6'd33 : scoreboard_y          <= writedata[9:0];
      6'd34 : digit1_x              <= writedata[9:0];
      6'd35 : digit1_y              <= writedata[9:0];
      6'd36 : digit1_img            <= writedata[3:0];
      6'd37 : digit2_x              <= writedata[9:0];
      6'd38 : digit2_y              <= writedata[9:0];
      6'd39 : digit2_img            <= writedata[3:0];
      6'd40 : digit3_x              <= writedata[9:0];
      6'd41 : digit3_y              <= writedata[9:0];
      6'd42 : digit3_img            <= writedata[3:0];
      6'd43 : fuelgauge_x           <= writedata[9:0];
      6'd44 : fuelgauge_y           <= writedata[9:0];
      6'd45 : indicator_x           <= writedata[9:0];
      6'd46 : indicator_y           <= writedata[9:0];
      6'd47 : shootRegister         <= writedata[0];
      6'd48 : hitRegister           <= writedata[0];
      6'd49 : explodeRegister       <= writedata[0];


    endcase
```

Figure: HW-SW interface boundary registers

There are 50 registers that software can access via the writedata and address registers. boundary_1_IN through boundary_4_IN are used to pass the 4 new boundary positions for each new row. The shift signal is used to shift each row down one by 1 when the signal goes high. sprite1_x and sprite1_y are used to specify the first sprite's x and y position at any time. This is what software can change to specify to hardware where the sprite should move to. This is done for another 9 sprites. We also have scoreboard_x and scoreboard_y which store the x and y position of the center of the scoreboard as well as digit1_x and digit1_y which store the x and y position of the first digit. We have a total of 3 digits like this for the game. We chose to limit the score to 3 digits as reaching a fourth digit would require a long time, and seemed unrealistic to waste resources on this. We also have 2 registers to specify the center

of the fuel gauge, and another 2 registers to specify the red indicator on the fuel gauge. Finally, we have 3 registers that software can assert high to play the 3 types of audio. All registers and their sizes are listed in the *Register Description* section below.

The hardware accesses data from many of these registers as it writes to the VGA monitor. The screen generation works by sweeping through horizontally and vertically all 640 x 480 pixels; if it gets within range of any sprite (it will check all sprites for each pixel) it will start drawing it. The pixel information for each sprite is stored in BRAM. Instead of storing the RGB values we decided to map each color to a corresponding 4 bit binary number. This color palette approach ended up saving us a significant amount of BRAM space. The hardware also draws the map using information sent from software. The map itself is rather simple - just a river overlaid on top of a green background. We decided to therefore generate the map in software by deciding where the boundaries of the river are for each row, and sending that data over to hardware. These boundaries will be stored in 4 registers per each vertical row. The hardware will do a simple comparison of the three values with *Horizontal Count* and decide whether the current pixel is blue or green. The hardware also stores all previous 480 boundaries in two port SRAM in order to draw the background for the rest of the screen. The screen shift is implemented simply by decrementing the base read and base write addresses of the dual port SRAM (and looping back to address 512 when they hit address 0), and writing the current values in the four boundary registers to that updated address (described in more detail in section c).

b. Sprites

Data for sprite images are stored in individual ROM units. Most sprites are 32 x 32 pixels, or 1024 entries. This meant that each instantiation of a ROM unit came out to having 1024 4 bit words. As mentioned, these 4 bits get mapped (in hardware) to corresponding RBG values according to the table below:

```
1: {VGA_R, VGA_G, VGA_B}        <= {8'h00, 8'hff, 8'h00}; //Green
2: {VGA_R, VGA_G, VGA_B}        <= {8'h00, 8'h00, 8'hff}; //Blue
3: {VGA_R, VGA_G, VGA_B}        <= {8'hff, 8'h00, 8'h00}; //Red
4: {VGA_R, VGA_G, VGA_B}        <= {8'hff, 8'hff, 8'h00}; //Yellow
5: {VGA_R, VGA_G, VGA_B}        <= {8'h00, 8'hff, 8'hff}; //Cyan
6: {VGA_R, VGA_G, VGA_B}        <= {8'hff, 8'h00, 8'hff}; //Magenta
7: {VGA_R, VGA_G, VGA_B}        <= {8'h80, 8'h80, 8'h80}; //Gray
8: {VGA_R, VGA_G, VGA_B}        <= {8'h00, 8'h00, 8'h00}; //Black
9: {VGA_R, VGA_G, VGA_B}        <= {8'hff, 8'hff, 8'h00}; //White
```

Figure: System Verilog RGB color encodings

A value of zero retrieved from ROM indicates that the sprite image is transparent at that location, and the color of the pixel should be obtained from other sprites in the area or the background. This is where the priority encoding of the sprites comes in; sprites are checked in the order of their enumeration (i.e. sprite

1 has the highest priority, while sprite 9 has the lowest). If a zero is pulled from ROM, the hardware checks each of the other sprites (in order from 1 to 9) to see if any of them have a nonzero color at the current pixel. If no sprite has a nonzero color at the current pixel, the pixel reverts to the color of the background. In implementation, if several sprites overlap, sprites with lower enumerations (ie sprite 1) will appear on the top while sprites with higher enumerations will appear on the bottom.

Though it has not been explicitly mentioned, it might be apparent that sprites themselves are not associated with a specific image. Rather, a single sprite can be assigned an image through the `spritex_img` register. Sprite addresses are also computed in an always_comb block based on the relative location of each pixel from each sprite's center (according to the sprite's x and y position). Note that each sprite image is 32 x 32 pixels, which explains the SystemVerilog code shown below.

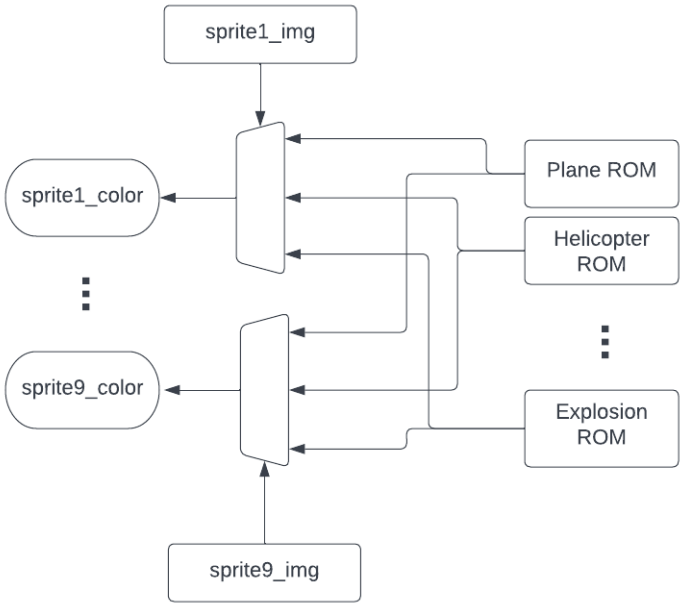


Figure: Block diagram of sprite color assignment

```
sprite1_address = ((vcount - (sprite1_y[9:1]-16)) << 5) + (hcount[10:1] - (sprite1_x-16));
sprite2_address = ((vcount - (sprite2_y[9:1]-16)) << 5) + (hcount[10:1] - (sprite2_x-16));
sprite3_address = ((vcount - (sprite3_y[9:1]-16)) << 5) + (hcount[10:1] - (sprite3_x-16));
sprite4_address = ((vcount - (sprite4_y[9:1]-16)) << 5) + (hcount[10:1] - (sprite4_x-16));
sprite5_address = ((vcount - (sprite5_y[9:1]-16)) << 5) + (hcount[10:1] - (sprite5_x-16));
sprite6_address = ((vcount - (sprite6_y[9:1]-16)) << 5) + (hcount[10:1] - (sprite6_x-16));
sprite7_address = ((vcount - (sprite7_y[9:1]-16)) << 5) + (hcount[10:1] - (sprite7_x-16));
sprite8_address = ((vcount - (sprite8_y[9:1]-16)) << 5) + (hcount[10:1] - (sprite8_x-16));
sprite9_address = ((vcount - (sprite9_y[9:1]-16)) << 5) + (hcount[10:1] - (sprite9_x-16));
```

Figure: Sprite ROM address calculation in SystemVerilog

The `isSpritex` register, which indicates whether the current pixel is in range of sprite x (this is computed combinationally) ultimately determines whether the ROM associated with `spritex_img`

should be assigned the address of sprite x, and whether the output should be assigned to `spritex_color`. This architecture allows us to have several sprites display the same image so long as they aren't near each other (in which case multiple values would be driving the addresses of the image's ROM).

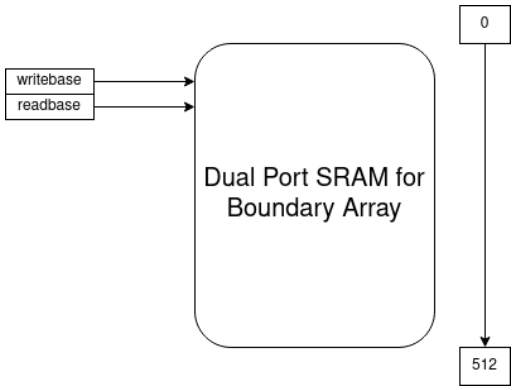
```
if(isSprite1_LATCHED) begin
        case(sprite1_img)
                0: begin
                        plane_address = sprite1_address;
                        sprite1_color = plane_out; //maybe latch the sprite colors
                   end
                1: begin
                        chopper_address = sprite1_address;
                        sprite1_color = chopper_out;
                   end
                2: begin
                        battleship_address = sprite1_address;
                        sprite1_color = battleship_out;
                   end
                3: begin
                        fuel_address = sprite1_address;
                        sprite1_color = fuel_out;
                   end
                4: begin
                        shoot_address = sprite1_address;
                        sprite1_color = shoot_out;
                   end
                5: begin
                        explosion_address = sprite1_address;
                        sprite1_color = explosion_out; //maybe latch the sprite colors
                   end
                6: begin
                        hotairballoon_address = sprite1_address;
                        sprite1_color = hotairballoon_out; //maybe latch the sprite colors
                   end
                7: begin
                        planeleft_address = sprite1_address;
                        sprite1_color = planeleft_out; //maybe latch the sprite colors
                   end
                8: begin
                        planeright_address = sprite1_address;
                        sprite1_color = planeright_out; //maybe latch the sprite colors
                   end
                9: begin
                        battleshipreverse_address = sprite1_address;
                        sprite1_color = battleshipreverse_out; //maybe latch the sprite colors
                   end
                10: begin
                        helicopterreverse_address = sprite1_address;
                        sprite1_color = helicopterreverse_out; //maybe latch the sprite colors
                   end

        endcase
end
```

Figure: Sprite address assignment and ROM output assignment

c. Boundary Memory

As mentioned, the indexing of boundary memory allows for the screen shift seen in the RiverRaid gameplay. The boundary memory is an instantiation of dual port SRAM where one port is used to read, and the other port is used to write. The word size is 40 bits (4 boundaries * 10 bits each) with a depth of 512. Registers `readbase` and `writebase` are initialized to values 9'd1 and 9'd0 respectively. The actual read address is



computed as an offset of `vcount` from `readbase`. Whenever the `shift` signal is asserted, both `readbase` and `writebase` are decremented (and wraparound to address 512 if they drop below 0). The values in registers `boundary_1_IN`, `boundary_2_IN`, `boundary_3_IN`, and `boundary_4_IN` (sent in from software) are concatenated and written to the new `writebase` address.

d. Audio

In addition to video, the hardware can also play audio with a pair of external speakers. The audio makes use of the built in audio codec. We ended up trying two different approaches to generating the audio. We tried both a software controlled as well as a hardware controlled approach. For the software approach, we looked into having software on the HPS using the LWAXI and AXI bridges to communicate with the FPGA and audio codec. We found that the VIP demo source code from the terasic DE1-SOC page to be a good starting point for this. However, after some testing this ended up being slower than we had initially hoped for. Instead we continued with a hardware implementation. To send signals to the speakers, we have access to two FIFOs one for the left audio, and one for the right. We also have the ability to assert when we are ready to push the audio (a valid signal), as well as receive a signal back from the codec when it is ready to receive audio (a ready signal). It is important that we only send audio after the ready signal has been returned for both audio channels, and we have set the valid signal high.

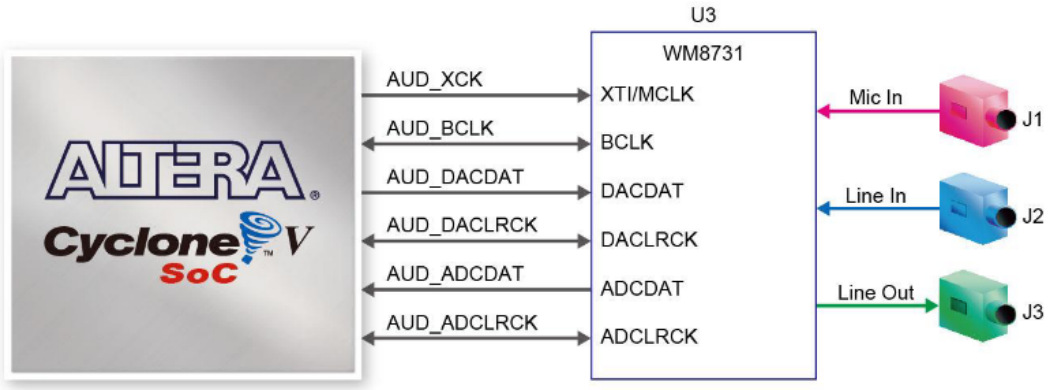


Figure: Audio Codec connections

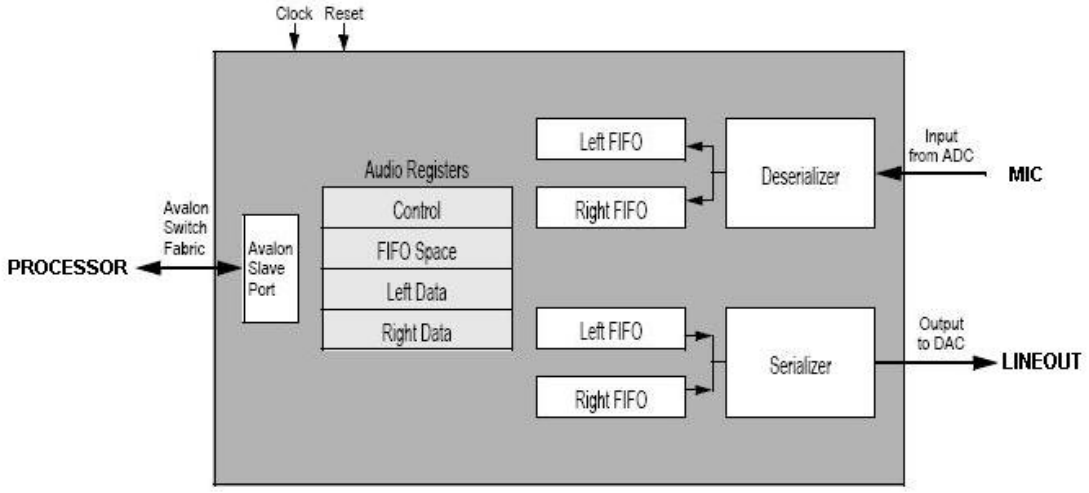


Figure: Audio Codec hardware

In order to generate the audio that we pass to the audio codec's FIFOs, we generated memory instantiation files (.mif). These were created by finding 16 bit .wav files. We then created a python script that read in the values of the .wav file, and reformatted the data into the .mif file format (such as seen in the figure below).

```
DEPTH = 2048;
WIDTH = 16;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT
BEGIN

0000 : 0000;
0001 : 47FA;
0002 : 47FA;
0003 : 47FA;
0004 : 47FA;
0005 : 47FC;
0006 : 47F8;
0007 : 47FF;
0008 : 47F8;
0009 : 47FC;
000a : 47FC;
000b : 47F9;
```

Figure: .mif file format

Once we had the .mif files, we were able to generate 1 port ROM files with the quartus megawizard. We choose the 1 port ROM as we do not need to write to the audio values. These are instantiated inside of the top level hardware module (vga_ball.sv). As a note, during the megawizard function we disabled the "'q' output port" option (discussed more in the lessons learned section) as seen in the figure below.

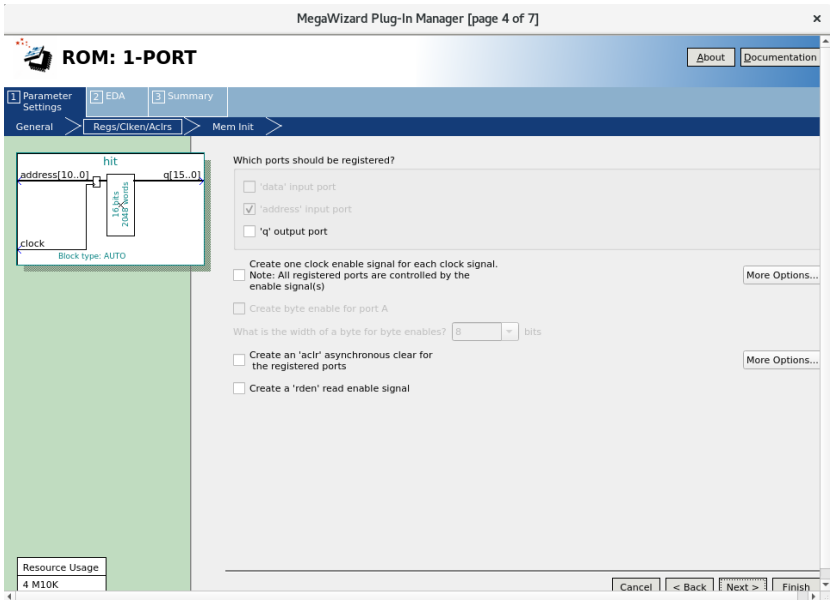


Figure: 1 port ROM initialization

Each ROM file has 3 inputs, an address, a clock, and an output. In our main audio loop, we check if the left and right audio channels from the audio codec are ready, if they are, then we begin iterating the address of the related sound effect. We then feed the combination of all three sound effect outputs into the respective FIFO on the codec. After the sound effect plays, we then set the address back to 0. This allows software to assert one of the three sound effect registers and have the entire sound effect played without having to worry after the initial register assertion.

We have specifically made the first value in each .mif file 0. This means that we can hold the address of 0 on all of the sound effects with no output on the speakers. To turn off the audio, we set the address to 0 and disable iteration of the respective address.

We connect to the audio codec via the following registers in our top level module. left_chan_ready and right_chan_ready are returned signals from the audio codec when it is ready to receive new audio on the respective channels, the sample_valid_l and sample_valid_r are what we can assert high when there is valid data to be played in the FIFO. The 16 bit sample_data_r and sample_data_l registers are the registers in the top level module that we directly connect to the FIFOs on the audio codec.

```
input left_chan_ready,
input right_chan_ready,
output logic [15:0] sample_data_l,
output logic sample_valid_l,
output logic [15:0] sample_data_r,
output logic sample_valid_r,
```

Figure: Audio I/O for our high level module

In order to connect these registers to the respective registers on the audio codec we use the following qsys connections:

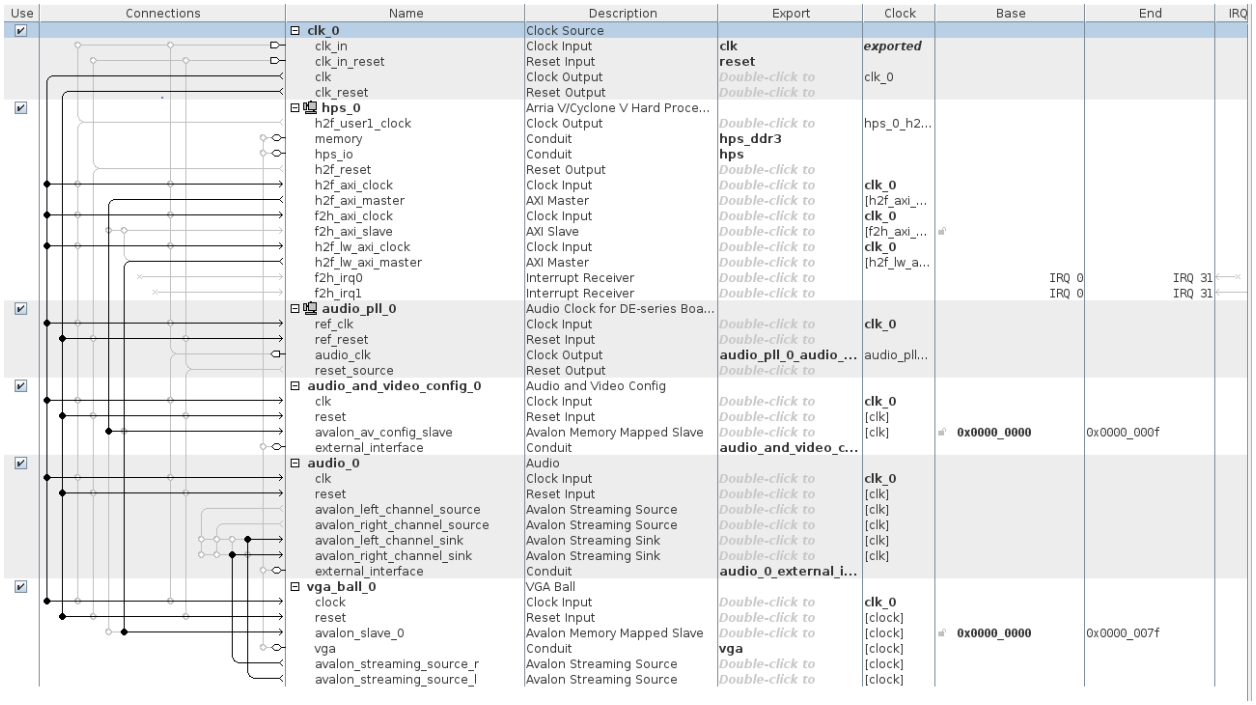


Figure: Qsys vga_ball connections to peripherals

These connections are as follows: a clk, the hps module, and the vga_ball_0 which we are using as the top level module for our game. For the audio specifically, we added both an audio_and_video_config_0 block as well as an audio_0 block. We also ended up adding an audio_pll_0 module for timing reasons.

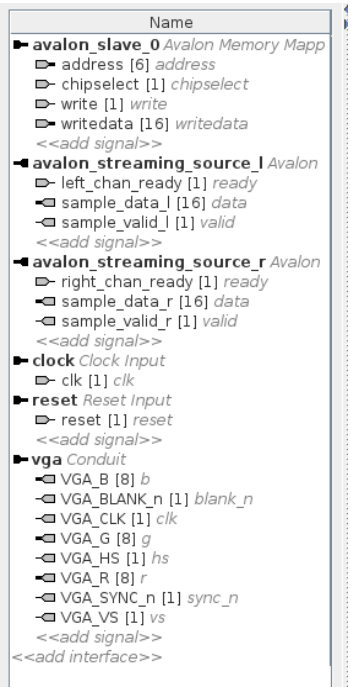


Figure: Qsys vga_ball top level connections

We then created an avalon_streaming_source for both the left and right channels in the vga_ball file and connected them to the related registers in the vga_ball module. These sources were then connected to the sink sources in the audio_0 module in qsys to allow our memory instantiated audio to be played on the speakers.

e. Register Descriptions

`boundary_1_IN`: 10 bit integer storing the left boundary of the river
`boundary_2_IN`: 10 bit integer storing the right boundary of the river
`boundary_3_IN`: 10 bit integer storing the left boundary of the river
`boundary_4_IN`: 10 bit integer storing the right boundary of the river

`shift:` 1 bit shift signal

`sprite1_x:` 10 bits storing the x position of sprite 1
`sprite1_y:` 10 bits storing the y position of sprite 1
`sprite1_img:` 5 bits indicating which image should be pulled from memory for this sprite

…

`sprite9_x:` 10 bits storing the x position of sprite 9

`sprite9_y`: 10 bits storing the y position of sprite 9
`sprite9_img`: 5 bits indicating which image should be pulled from memory for this sprite

`scoreboard_x`: 10 bits storing the x position of the scoreboard
`scoreboard_y`: 10 bits storing the y position of the scoreboard

`digit1_x`: 10 bits storing the x position of digit 1 of the score
`digit1_y`: 10 bits storing the y position of digit 1 of the score
`digit1_img`: 4 bits storing the y position of digit 1 of the score

`digit2_x`: 10 bits storing the x position of digit 2 of the score
`digit2_y`: 10 bits storing the y position of digit 2 of the score
`digit2_img`: 4 bits storing the y position of digit 2 of the score

`digit3_x`: 10 bits storing the x position of digit 3 of the score
`digit3_y`: 10 bits storing the y position of digit 3 of the score
`digit3_img`: 4 bits storing the y position of digit 3 of the score

`fuelgauge_x`: 10 bits storing the x position of the fuel gauge
`fuelgauge_y`: 10 bits storing the y position of the fuel gauge

`indicator_x`: 10 bits storing the x position of the fuel gauge indicator
`indicator_y`: 10 bits storing the y position of the fuel gauge indicator

5. Memory Budget

Graphics Memory

Category	Size (pixels)	Total Size (bits)
Plane	32*32	4096
Plane left tilt	32*32	4096
Plane right tilt	32*32	4096
Battleship	32*32	4096
Battleship mirrored	32*32	4096
Hot Air Balloon	32*32	4096
Helicopter	32*32	4096
Helicopter mirrored	32*32	4096
Score Board	40*32	5120
Number	20*32	2560 (x10 for 10 digits)
Fuel Gauge	80*40	12800
Fuel Gauge Indicator	32*32	4096
Bullet	32*32	4096
Explosion	32*32	4096
Fuel	32*32	4096

Total Memory Budget (bits): 92,672

Audio Memory

	Shoot	Hit	Explosion
Time (s)	0.5	0.5	0.5
F (KHz)	48	48	48
memory(bit)	4,000*16	4,000*16	4,000*16

Total Audio Budget (bits): 192,000

6. Contributions

a. Yongmao Luo

```
[yl4893@micro25 Water-Raid]$ ../gitinspector/gitinspector.py -x hardware -x ROM
-x SRAM -x SRAM_12
Statistical information for the repository 'Water-Raid' was gathered on
2022/05/12.
The following historical commit information, by author, was found:

Author                    Commits    Insertions      Deletions    % of changes
YongmaoLuo                     35          6690           2511           85.34
wangzhaomeng                   15          1077            355           13.28
yl4893                          1           148              0            1.37
```

Figure 6-1 running gitinspector[3] to get the commit statistics of the software part. Since hardware uploads a lot of auto-generated files, and in the repo the software part only contains the original code files, I only show the result for the software. "-x" means exclude files path or names. The above command gets the result of calculating the percentage of changes in terms of lines of code. The GitHub repo of this project is: https://github.com/YongmaoLuo/Water-Raid.

My involvement was designing the structure of the software part and implementing it, including the Game Logic, High Level Driver and Linux Kernel Drivers. We simulate the design of the Game "River-Raid", but all the logic written in software is created by ourselves from scratch. I wrote the algorithm to randomly generate the game background and control the airplane's operations, like the adding scores, changing amount of fuels, changing positions, shooting, judging the collision between plane and sprites as well as boundaries etc. I also added audio support in Game Logic. What's more, in the last week of the semester I helped debug the functionalities of sprites generation and movement. For example, in one of the previous versions, the sprites' generation is not in the right space, and movement is so fast that it is very difficult for users to pass the game. Also, I modified the software to improve the user experience, like adjusting the moving speed of enemy sprites and the plane, changing the image of sprites when changing the moving directions.

Meanwhile, I wrote all the kernel drivers for video and audio. The way I did it is based on the driver file in lab3, and add more functions based on the functionalities of different variables in hardware. Since the hardware for audio is synthesized with the hardware for video, I just need to use different functions to write corresponding variables in hardware.

What's more, I figured out how to use the Xbox Controller on Linux. At first, the driver did not provide a satisfying experience and sometimes the control is a bit laggy. After reading through the 2000 lines driver file, I understood the way it works and modified the driver to make the control experience much smoother.

b. Tristan Saidi

My involvement in this project was creating the hardware for graphics, screen shifting and some audio logic. I wrote all the code seen in vga_ball.sv - this undertaking required me to incrementally build off of Lab 3. I started by implementing the sprite graphics by preloading and instantiating ROM modules with the desired sprites. Some debugging and experimentation led me to a hardware architecture that allowed us to display multiple of the same image on the screen at the same time without having any glitching or buggy behavior. Once this was set up I moved on to implementing the screen shift. I set up the two port ROM and experimented with different ways of implementing the hardware; the final version is what is seen above.

In the last few weeks my work has been on creating all remaining graphics; this essentially required me to use Jakobs scripts to draw and generate .mif files for all graphics (fuel gauge, enemy sprites, bullet, score, etc.) and replicate the hardware that I had created for the initial sprites. This turned out to be very time consuming. My final hardware contribution was finishing the audio - Jakob had managed to figure out the qsys setup and ROM instantiation for the audio, and I spent a day in the lab describing and testing the audio logic seen at the bottom of the vga_ball module in vga_ball.sv.

c. Jakob Stiens

My contributions include creating Python scripts that read in .wav files and generated .mif files, Python scripts to read in arrays of values and outputting sprite .mif files. I also created game sprites and generated ROM files. I also created the original shift register verilog module for the background that was eventually replaced with the current RAM module. I also did all of the qsys work for the project. I learned about the audio codec, and figured out how to interface with it. This ended up taking quite a bit of time, as the first working implementation was a software connection via the LWAXI bridge to control the audio. This implementation was later ditched as the latency from starting the audio to when it was played on the speakers was relatively slow. I then worked on a hardware implementation. I figured out the qsys connections, vga_ball registers, generated all of the game's audio, as well as the original audio logic that Tristan later modified to remove a bug. I also did merging for combining Tristan's and my different local hardware files. The very end of the project I was working on assorted bug fixes and improvements in the hardware (such as adding in left and right turn sprites) and looking into having a background audio loop that was later ditched for space and distraction reasons.

d. Zhaomeng Wang

My work focuses on the software aspect of the project, specifically the game logic. I developed the Sprites, EnemyPlane, Battleship, FuelTank and bullet classes by using Object-Oriented Programming thinking. I wrote several important functions inside those classes, including the "move", "generate", "disappear" ,"checkIfHit " and so on. I coded the main.app file as well, which is the overall control of the game system. When the game restarts, I first clear all the sprites and bullets from the last game which is still shown on the screen. And nextly, in each game cycle, I make all the sprites and bullets fly or randomly move within the appropriate area. In the main.cpp I also developed the hit detection section, which detected the process of the bullet hitting any sprites, and after that detection adding the score and making the sprites disappear.

I also developed python scripts to convert an image which is designed to reshape and convert the cover image at the beginning of the game and an image of “game over” at the end of the game to certain .mif files. However, those images are not shown in our final game, because we run out of memory :(. I looked up some literature and tutorials to develop the software driver for audio.

7. Lessons Learned and Advice

This project presented unique challenges and obstacles that we worked hard as a group to overcome. As a result, we picked up a few valuable lessons concerning group projects. First and foremost, in the case of embedded systems, it is very important to have the hardware finished early.

Since the implementation of software is largely dependent on the implementation of the hardware, true testing on the software cannot begin until the hardware foundation is laid down. However, since we designed the system first, we have basic knowledge about how to organize the data structures, so even when hardware has not been done, we can construct some basic data structures, which can speed up the process. Also, since the structure of the whole system is organized rationally, we can develop software at a fast pace. Thus, the first lesson learned is correct system design beforehand is really important and helpful.

Focusing specifically on software, during its development, sometimes it is easy to change the temporary variable but you think you have changed the variable in the permenant data structure, which can cause problems. One way to deal with this problem is to always add a temporary variable to the permanent data structure at the end of the code block.

8. References

[1] Pavel R. paroj/xpad. (2022). GitHub Repository, https://github.com/paroj/xpad
[2] https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/optional-tonegen.html
[3] Ejwa. **ejwa/gitinspector.** (2022). GitHub Repository, https://github.com/ejwa/gitinspector

9. Appendix

vga_ball.sv

```
`include "../ROM/plane_ROM.v"
`include "../ROM/zero_ROM.v"
`include "../ROM/one_ROM.v"
`include "../ROM/two_ROM.v"
`include "../ROM/three_ROM.v"
`include "../ROM/four_ROM.v"
`include "../ROM/five_ROM.v"
`include "../ROM/six_ROM.v"
`include "../ROM/seven_ROM.v"
`include "../ROM/eight_ROM.v"
`include "../ROM/nine_ROM.v"
`include "../ROM/fuel_ROM.v"
`include "../ROM/chopper_ROM.v"
`include "../ROM/battleship_ROM.v"
`include "../ROM/scoreboard_ROM.v"
`include "../ROM/fuelgauge_ROM.v"
`include "../ROM/indicator_ROM.v"
`include "../ROM/shoot_ROM.v"
`include "../ROM/explosion_ROM.v"
`include "../ROM/hotairballoon_ROM.v"
`include "../ROM/planeleft_ROM.v"
`include "../ROM/planeright_ROM.v"
`include "../ROM/battleshipreverse_ROM.v"
`include "../ROM/helicopterreverse_ROM.v"
`include "../BoundaryMemory/boundary_mem.sv"
`include "shoot.v"
`include "bomb.v"
`include "hit.v"



module vga_ball(input logic         clk,
            input logic       reset,
       input logic [15:0] writedata,
       input logic      write,
       input            chipselect,
       input logic [5:0]  address,
```

```
    input left_chan_ready,
    input right_chan_ready,
    output logic [15:0] sample_data_l,
    output logic sample_valid_l,
    output logic [15:0] sample_data_r,
    output logic sample_valid_r,

    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);

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

logic [9:0]    boundary_1_IN;
logic [9:0]    boundary_2_IN;
logic [9:0]    boundary_3_IN;
logic [9:0]    boundary_4_IN;

logic [9:0]    boundary_1;
logic [9:0]    boundary_2;
logic [9:0]    boundary_3;
logic [9:0]    boundary_4;

logic [9:0]    boundary_1_LATCHED;
logic [9:0]    boundary_2_LATCHED;
logic [9:0]    boundary_3_LATCHED;
logic [9:0]    boundary_4_LATCHED;

logic [39:0]   boundary_out;

assign boundary_1 = boundary_out[39:30];
assign boundary_2 = boundary_out[29:20];
assign boundary_3 = boundary_out[19:10];
assign boundary_4 = boundary_out[9:0];

logic [3:0]    current_color; //for sprites
logic [3:0]          current_color_NONSPRITE;
logic [3:0]    current_background; //for background
```

```
    logic [3:0]          current_color_LATCHED;
    logic [3:0]          current_color_NONSPRITE_LATCHED;
    logic [3:0]    current_background_LATCHED;

    logic [3:0]          sprite1_color;
    logic [3:0]    sprite2_color;
    logic [3:0]    sprite3_color;
    logic [3:0]    sprite4_color;
    logic [3:0]    sprite5_color;
    logic [3:0]          sprite6_color;
    logic [3:0]          sprite7_color;
    logic [3:0]          sprite8_color;
    logic [3:0]          sprite9_color;
    logic [3:0]    digit1_color;
    logic [3:0]    digit2_color;
    logic [3:0]    digit3_color;

    logic [3:0]          sprite1_color_LATCHED;
    logic [3:0]    sprite2_color_LATCHED;
    logic [3:0]    sprite3_color_LATCHED;
    logic [3:0]    sprite4_color_LATCHED;
    logic [3:0]    sprite5_color_LATCHED;
    logic [3:0]          sprite6_color_LATCHED;
    logic [3:0]          sprite7_color_LATCHED;
    logic [3:0]          sprite8_color_LATCHED;
    logic [3:0]          sprite9_color_LATCHED;
    logic [3:0]    digit1_color_LATCHED;
    logic [3:0]    digit2_color_LATCHED;
    logic [3:0]    digit3_color_LATCHED;
    // last bit of y positions indicates whether sprite is onscreen
    logic    shift;
    // logic to determine whether or not to pull reset high

    logic [9:0]    sprite1_x;
    logic [9:0]    sprite1_y;
    logic [4:0]    sprite1_img; //which sprite is this?

    logic [9:0]    sprite2_x;
    logic [9:0]    sprite2_y;
    logic [4:0]    sprite2_img; //which sprite is this?

    logic [9:0]    sprite3_x;
    logic [9:0]    sprite3_y;
```

```
    logic [4:0]    sprite3_img; //which sprite is this?

    logic [9:0]    sprite4_x;
    logic [9:0]    sprite4_y;
    logic [4:0]    sprite4_img; //which sprite is this?

    logic [9:0]    sprite5_x;
    logic [9:0]    sprite5_y;
    logic [4:0]    sprite5_img;

    logic [9:0]    sprite6_x;
    logic [9:0]    sprite6_y;
    logic [4:0]    sprite6_img; //which sprite i.19.0'


    logic [9:0]    sprite7_x;
    logic [9:0]    sprite7_y;
    logic [4:0]    sprite7_img; //which sprite is this?

    logic [9:0]    sprite8_x;
    logic [9:0]    sprite8_y;
    logic [4:0]    sprite8_img; //which sprite is this?

    logic [9:0]    sprite9_x;
    logic [9:0]    sprite9_y;
    logic [4:0]    sprite9_img; //which sprite is this?

    logic [9:0]    digit1_x;
    logic [9:0]    digit1_y;
    logic [3:0]    digit1_img;

    logic [9:0]    digit2_x;
    logic [9:0]    digit2_y;
    logic [3:0]    digit2_img;

    logic [9:0]    digit3_x;
    logic [9:0]    digit3_y;
    logic [3:0]    digit3_img;


    logic [9:0]    scoreboard_x;
    logic [9:0]    scoreboard_y;
    logic [9:0]    fuelgauge_x;
```

```
    logic [9:0]    fuelgauge_y;
    logic [9:0]    indicator_x;
    logic [9:0]    indicator_y;

    logic    isSprite;
    logic    isMusic;  //remove
    logic [1:0]    whichClip;

//audio
    logic shootRegister;
    logic hitRegister;
    logic explodeRegister;


    boundary_mem boundary_mem(
     .clk(clk),
     .shift(shift),
     .reset(reset),
     .readaddress(vcount[8:0]),
     .datain({boundary_1_IN, boundary_2_IN, boundary_3_IN, boundary_4_IN}),
     .dataout(boundary_out)
    );



    vga_counters counters(.clk50(clk), .*);

    //ROM Wires

    logic [9:0]    sprite1_address;
    logic [9:0]    sprite2_address;
    logic [9:0]    sprite3_address;
    logic [9:0]    sprite4_address;
    logic [9:0]          sprite5_address;
    logic [9:0]    sprite6_address;
    logic [9:0]    sprite7_address;
    logic [9:0]    sprite8_address;
    logic [9:0]    sprite9_address;
    logic [9:0]    digit1_address;
    logic [9:0]    digit2_address;
    logic [9:0]    digit3_address;

    logic    isSprite1;
```

```
    logic    isSprite2;
    logic          isSprite3;
    logic    isSprite4;
    logic    isSprite5;
    logic    isSprite6;
    logic    isSprite7;
    logic    isSprite8;
    logic    isSprite9;
    logic    isScoreboard;
    logic    isFuelgauge;
    logic    isIndicator;
    logic    isDigit1;
    logic    isDigit2;
    logic    isDigit3;

    logic    isSprite1_LATCHED;
    logic    isSprite2_LATCHED;
    logic          isSprite3_LATCHED;
    logic    isSprite4_LATCHED;
    logic    isSprite5_LATCHED;
    logic    isSprite6_LATCHED;
    logic    isSprite7_LATCHED;
    logic    isSprite8_LATCHED;
    logic    isSprite9_LATCHED;
    logic    isScoreboard_LATCHED;
    logic    isFuelgauge_LATCHED;
    logic    isIndicator_LATCHED;
    logic    isDigit1_LATCHED;
    logic    isDigit2_LATCHED;
    logic    isDigit3_LATCHED;

    logic [3:0]    plane_out;
    logic [9:0]    plane_address;
    logic [9:0]    plane_address_LATCHED;

    logic [3:0]    chopper_out;
    logic [9:0]          chopper_address;
    logic [9:0]    chopper_address_LATCHED;

    logic [3:0]    battleship_out;
    logic [9:0]    battleship_address;
    logic [9:0]    battleship_address_LATCHED;
```

```
    logic [3:0]    fuel_out;
    logic [9:0]    fuel_address;
    logic [9:0]    fuel_address_LATCHED;

    logic [3:0]    shoot_out;
    logic [9:0]    shoot_address;
    logic [9:0]    shoot_address_LATCHED;

    logic [3:0]    explosion_out;
    logic [9:0]    explosion_address;
    logic [9:0]    explosion_address_LATCHED;

    logic [3:0]    hotairballoon_out;
    logic [9:0]    hotairballoon_address;
    logic [9:0]    hotairballoon_address_LATCHED;

    logic [3:0]    planeleft_out;
    logic [9:0]    planeleft_address;
    logic [9:0]    planeleft_address_LATCHED;

    logic [3:0]    planeright_out;
    logic [9:0]    planeright_address;
    logic [9:0]    planeright_address_LATCHED;

    logic [3:0]    battleshipreverse_out;
    logic [9:0]    battleshipreverse_address;
    logic [9:0]    battleshipreverse_address_LATCHED;

    logic [3:0]    helicopterreverse_out;
    logic [9:0]    helicopterreverse_address;
    logic [9:0]    helicopterreverse_address_LATCHED;

    logic [3:0]    scoreboard_out;
    logic [9:0]    scoreboard_address;
    logic [9:0]    scoreboard_address_LATCHED;

    logic [3:0]    fuelgauge_out;
    logic [11:0]   fuelgauge_address;
    logic [11:0]   fuelgauge_address_LATCHED;

    logic [3:0]    indicator_out;
    logic [9:0]    indicator_address;
    logic [9:0]    indicator_address_LATCHED;
```

```
   plane_ROM       plane_ROM(.address(plane_address_LATCHED),
.clock(clk),.q(plane_out));
   chopper_ROM
chopper_ROM(.address(chopper_address_LATCHED),.clock(clk),.q(chopper_out));
   battleship_ROM     batteship_ROM(.address(battleship_address_LATCHED),
.clock(clk), .q(battleship_out));
   fuel_ROM        fuel_ROM(.address(fuel_address_LATCHED), .clock(clk),
.q(fuel_out));
   scoreboard_ROM     scoreboard_ROM(.address(scoreboard_address_LATCHED),
.clock(clk), .q(scoreboard_out));
   fuelgauge_ROM
fuelgauge_ROM(.address(fuelgauge_address_LATCHED),.clock(clk),.q(fuelgauge_
out));
   indicator_ROM
indicator_ROM(.address(indicator_address_LATCHED),.clock(clk),.q(indicator_
out));
   shoot_ROM       shoot_ROM(.address(shoot_address_LATCHED),
.clock(clk),.q(shoot_out));
   explosion_ROM     explosion_ROM(.address(explosion_address_LATCHED),
.clock(clk),.q(explosion_out));
   hotairballoon_ROM
hotairballoon_ROM(.address(hotairballoon_address_LATCHED),
.clock(clk),.q(hotairballoon_out));
   planeleft_ROM     planeleft_ROM(.address(planeleft_address_LATCHED),
.clock(clk),.q(planeleft_out));
   planeright_ROM     planeright_ROM(.address(planeright_address_LATCHED),
.clock(clk),.q(planeright_out));
   battleshipreverse_ROM
battleshipreverse_ROM(.address(battleshipreverse_address_LATCHED),
.clock(clk),.q(battleshipreverse_out));
   helicopterreverse_ROM
helicopterreverse_ROM(.address(helicopterreverse_address_LATCHED),
.clock(clk),.q(helicopterreverse_out));


   logic [3:0]    zero_out;
   logic [9:0]    zero_address;
   logic [9:0]          zero_address_LATCHED;

   zero_ROM        zero_ROM(.address(zero_address_LATCHED),.clock(clk),
.q(zero_out));
```

```
   logic [3:0]    one_out;
   logic [9:0]    one_address;
   logic [9:0]          one_address_LATCHED;

   one_ROM         one_ROM(.address(one_address_LATCHED),.clock(clk),
.q(one_out));

   logic [3:0]    two_out;
   logic [9:0]    two_address;
   logic [9:0]          two_address_LATCHED;

   two_ROM         two_ROM(.address(two_address_LATCHED),.clock(clk),
.q(two_out));

   logic [3:0]    three_out;
   logic [9:0]    three_address;
   logic [9:0]          three_address_LATCHED;

   three_ROM
three_ROM(.address(three_address_LATCHED),.clock(clk),.q(three_out));

   logic [3:0]    four_out;
   logic [9:0]    four_address;
   logic [9:0]          four_address_LATCHED;

   four_ROM        four_ROM(.address(four_address_LATCHED),.clock(clk),
.q(four_out));

   logic [3:0]    five_out;
   logic [9:0]    five_address;
   logic [9:0]          five_address_LATCHED;

   five_ROM        five_ROM(.address(five_address_LATCHED),.clock(clk),
.q(five_out));

   logic [3:0]    six_out;
   logic [9:0]    six_address;
   logic [9:0]          six_address_LATCHED;

   six_ROM         six_ROM(.address(six_address_LATCHED),.clock(clk),
.q(six_out));

   logic [3:0]    seven_out;
```

```
   logic [9:0]    seven_address;
   logic [9:0]          seven_address_LATCHED;

   seven_ROM       seven_ROM(.address(seven_address_LATCHED),.clock(clk),
.q(seven_out));

   logic [3:0]    eight_out;
   logic [9:0]    eight_address;
   logic [9:0]          eight_address_LATCHED;

   eight_ROM       eight_ROM(.address(eight_address_LATCHED),.clock(clk),
.q(eight_out));

   logic [3:0]    nine_out;
   logic [9:0]    nine_address;
   logic [9:0]          nine_address_LATCHED;

   nine_ROM        nine_ROM(.address(nine_address_LATCHED),.clock(clk),
.q(nine_out));


   assign isSprite = isSprite1_LATCHED || isSprite2_LATCHED ||
isSprite3_LATCHED || isSprite4_LATCHED || isSprite5_LATCHED ||
isSprite6_LATCHED || isSprite7_LATCHED || isSprite8_LATCHED
||isSprite9_LATCHED ||isScoreboard_LATCHED || isFuelgauge_LATCHED ||
isIndicator_LATCHED || isDigit1_LATCHED || isDigit2_LATCHED ||
isDigit3_LATCHED;

   always_ff @(posedge clk) begin
      if (chipselect && write)
      case (address)

     6'd0 : boundary_1_IN      <= writedata[9:0];
     6'd1 : boundary_2_IN      <= writedata[9:0];
     6'd2 : boundary_3_IN      <= writedata[9:0];
     6'd3 : boundary_4_IN      <= writedata[9:0];
     6'd4 : shift              <= writedata[0];
     6'd5 : sprite1_x          <= writedata[9:0];
     6'd6 : sprite1_y          <= writedata[9:0];
     6'd7 : sprite1_img        <= writedata[4:0];
     6'd8 : sprite2_x          <= writedata[9:0];
     6'd9 : sprite2_y          <= writedata[9:0];
     6'd10 : sprite2_img       <= writedata[4:0];
```

```
    6'd11 : sprite3_x         <= writedata[9:0];
    6'd12 : sprite3_y         <= writedata[9:0];
    6'd13 : sprite3_img       <= writedata[4:0];
    6'd14 : sprite4_x         <= writedata[9:0];
    6'd15 : sprite4_y         <= writedata[9:0];
    6'd16 : sprite4_img       <= writedata[4:0];
    6'd17 : sprite5_x         <= writedata[9:0];
    6'd18 : sprite5_y         <= writedata[9:0];
    6'd19 : sprite5_img       <= writedata[4:0];
    6'd20 : sprite6_x         <= writedata[9:0];
    6'd21 : sprite6_y         <= writedata[9:0];
    6'd22 : sprite6_img       <= writedata[4:0];
    6'd23 : sprite7_x         <= writedata[9:0];
    6'd24 : sprite7_y         <= writedata[9:0];
    6'd25 : sprite7_img       <= writedata[4:0];
    6'd26 : sprite8_x         <= writedata[9:0];
    6'd27 : sprite8_y         <= writedata[9:0];
    6'd28 : sprite8_img       <= writedata[4:0];
    6'd29 : sprite9_x         <= writedata[9:0];
    6'd30 : sprite9_y         <= writedata[9:0];
    6'd31 : sprite9_img       <= writedata[4:0];
    6'd32 : scoreboard_x      <= writedata[9:0];
    6'd33 : scoreboard_y      <= writedata[9:0];
    6'd34 : digit1_x          <= writedata[9:0];
    6'd35 : digit1_y          <= writedata[9:0];
    6'd36 : digit1_img        <= writedata[3:0];
    6'd37 : digit2_x          <= writedata[9:0];
    6'd38 : digit2_y          <= writedata[9:0];
    6'd39 : digit2_img        <= writedata[3:0];
    6'd40 : digit3_x          <= writedata[9:0];
    6'd41 : digit3_y          <= writedata[9:0];
    6'd42 : digit3_img        <= writedata[3:0];
    6'd43 : fuelgauge_x       <= writedata[9:0];
    6'd44 : fuelgauge_y       <= writedata[9:0];
    6'd45 : indicator_x       <= writedata[9:0];
    6'd46 : indicator_y       <= writedata[9:0];
    6'd47 : shootRegister     <= writedata[0];
    6'd48 : hitRegister       <= writedata[0];
    6'd49 : explodeRegister    <= writedata[0];


    endcase
  end
```

```
   always_ff @(posedge clk) begin
    if (reset) begin
      {VGA_R, VGA_G, VGA_B} <= {8'h00, 8'h00, 8'h00}; //Black
      end
      else if(isSprite && current_color_LATCHED != 0 && vcount >= 60) begin
       case(current_color_LATCHED)

            0: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff}; //White
            1: {VGA_R, VGA_G, VGA_B}     <= {8'h00, 8'hff, 8'h00}; //Green
            2: {VGA_R, VGA_G, VGA_B}     <= {8'h00, 8'h00, 8'hff}; //Blue
            3: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'h00, 8'h00}; //Red
            4: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'h00};
//Yellow
            5: {VGA_R, VGA_G, VGA_B}     <= {8'h00, 8'hff, 8'hff}; //Cyan
            6: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'h00, 8'hff};
//Magenta
            7: {VGA_R, VGA_G, VGA_B}     <= {8'h80, 8'h80, 8'h80}; //Gray
            8: {VGA_R, VGA_G, VGA_B}     <= {8'h00, 8'h00, 8'h00}; //Black
            9: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'h00}; //White
            10: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White
            11: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White
            12: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White
            13: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White
            14: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White
            15: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White

       endcase
    end
    else if((isScoreboard_LATCHED || isFuelgauge_LATCHED ||
isIndicator_LATCHED || isDigit1_LATCHED || isDigit2_LATCHED ||
isDigit3_LATCHED) && current_color_NONSPRITE_LATCHED != 0) begin

       case(current_color_NONSPRITE_LATCHED)

            0: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff}; //White
            1: {VGA_R, VGA_G, VGA_B}     <= {8'h00, 8'hff, 8'h00}; //Green
```

```
            2: {VGA_R, VGA_G, VGA_B}     <= {8'h00, 8'h00, 8'hff}; //Blue
            3: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'h00, 8'h00}; //Red
            4: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'h00};
//Yellow
            5: {VGA_R, VGA_G, VGA_B}     <= {8'h00, 8'hff, 8'hff}; //Cyan
            6: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'h00, 8'hff};
//Magenta
            7: {VGA_R, VGA_G, VGA_B}     <= {8'h80, 8'h80, 8'h80}; //Gray
            8: {VGA_R, VGA_G, VGA_B}     <= {8'h00, 8'h00, 8'h00}; //Black
            9: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'h00}; //White
            10: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White
            11: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White
            12: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White
            13: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White
            14: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White
            15: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White

       endcase


    end
    else begin
       case(current_background_LATCHED)

            0: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff}; //White
            1: {VGA_R, VGA_G, VGA_B}     <= {8'h00, 8'hff, 8'h00}; //Green
            2: {VGA_R, VGA_G, VGA_B}     <= {8'h00, 8'h00, 8'hff}; //Blue
            3: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'h00, 8'h00}; //Red
            4: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'h00};
//Yellow
            5: {VGA_R, VGA_G, VGA_B}     <= {8'h00, 8'hff, 8'hff}; //Cyan
            6: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'h00, 8'hff};
//Magenta
            7: {VGA_R, VGA_G, VGA_B}     <= {8'h80, 8'h80, 8'h80}; //Gray
            8: {VGA_R, VGA_G, VGA_B}     <= {8'h00, 8'h00, 8'h00}; //Black
            9: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'h00}; //White
            10: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
```

```
//White
            11: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White
            12: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White
            13: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White
            14: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White
            15: {VGA_R, VGA_G, VGA_B}     <= {8'hff, 8'hff, 8'hff};
//White

       endcase
    end
   end

   always @(posedge clk) begin //latching of some values
    current_color_LATCHED      <= current_color;
    current_background_LATCHED     <= current_background;
    current_color_NONSPRITE_LATCHED <= current_color_NONSPRITE;
    boundary_1_LATCHED         <= boundary_1;
    boundary_2_LATCHED         <= boundary_2;
    boundary_3_LATCHED         <= boundary_3;
    boundary_4_LATCHED         <= boundary_4;
    plane_address_LATCHED      <= plane_address;
    planeleft_address_LATCHED        <= planeleft_address;
    planeright_address_LATCHED       <= planeright_address;
    battleshipreverse_address_LATCHED      <= battleshipreverse_address;
    helicopterreverse_address_LATCHED      <= helicopterreverse_address;
    shoot_address_LATCHED      <= shoot_address;
    explosion_address_LATCHED    <= explosion_address;
    hotairballoon_address_LATCHED    <= hotairballoon_address;
    chopper_address_LATCHED          <= chopper_address;
    battleship_address_LATCHED    <= battleship_address;
    fuel_address_LATCHED       <= fuel_address;
    scoreboard_address_LATCHED    <= scoreboard_address;
    fuelgauge_address_LATCHED    <= fuelgauge_address;
    indicator_address_LATCHED    <= indicator_address;
    zero_address_LATCHED       <= zero_address;
    one_address_LATCHED        <= one_address;
    two_address_LATCHED        <= two_address;
    three_address_LATCHED      <= three_address;
    four_address_LATCHED       <= four_address;
```

```
    five_address_LATCHED       <= five_address;
    six_address_LATCHED        <= six_address;
    seven_address_LATCHED      <= seven_address;
    eight_address_LATCHED      <= eight_address;
    nine_address_LATCHED       <= nine_address;
       sprite1_color_LATCHED         <= sprite1_color;
       sprite2_color_LATCHED         <= sprite2_color;
             sprite3_color_LATCHED         <= sprite3_color;
             sprite4_color_LATCHED         <= sprite4_color;
    sprite5_color_LATCHED      <= sprite5_color;
       sprite6_color_LATCHED         <= sprite6_color;
       sprite7_color_LATCHED         <= sprite7_color;
       sprite8_color_LATCHED         <= sprite8_color;
       sprite9_color_LATCHED         <= sprite9_color;
    digit1_color_LATCHED       <= digit1_color;
    digit2_color_LATCHED       <= digit2_color;
    digit3_color_LATCHED       <= digit3_color;
       isSprite1_LATCHED       <= isSprite1;
       isSprite2_LATCHED       <= isSprite2;
       isSprite3_LATCHED       <= isSprite3;
    isSprite4_LATCHED          <= isSprite4;
    isSprite5_LATCHED          <= isSprite5;
       isSprite6_LATCHED       <= isSprite6;
       isSprite7_LATCHED       <= isSprite7;
       isSprite8_LATCHED       <= isSprite8;
       isSprite9_LATCHED       <= isSprite9;
    isDigit1_LATCHED     <= isDigit1;
    isDigit2_LATCHED     <= isDigit2;
    isDigit3_LATCHED     <= isDigit3;
    isScoreboard_LATCHED       <= isScoreboard;
    isFuelgauge_LATCHED        <= isFuelgauge;
    isIndicator_LATCHED        <= isIndicator;
  end

  always begin

     isSprite1 = 0;
     isSprite2 = 0;
     isSprite3 = 0;
     isSprite4 = 0;
     isSprite5 = 0;
     isSprite6 = 0;
     isSprite7 = 0;
```

```
        isSprite8 = 0;
        isSprite9 = 0;
        isScoreboard = 0;
        isFuelgauge = 0;
        isIndicator = 0;
        isDigit1 = 0;
        isDigit2 = 0;
        isDigit3 = 0;

        zero_address = 10'd0;
        one_address = 10'd0;
        two_address = 10'd0;
        three_address = 10'd0;
        four_address = 10'd0;
        five_address = 10'd0;
        six_address = 10'd0;
        seven_address = 10'd0;
        eight_address = 10'd0;
        nine_address = 10'd0;
        plane_address = 10'd0;
        planeleft_address = 10'd0;
        planeright_address = 10'd0;
        battleshipreverse_address = 10'd0;
        helicopterreverse_address = 10'd0;
        explosion_address = 10'd0;
        hotairballoon_address = 10'd0;
        shoot_address = 10'd0;
        chopper_address = 10'd0;
        battleship_address = 10'd0;
        fuel_address = 10'd0;

        digit1_color = 4'b0;
        digit2_color = 4'b0;
        digit3_color = 4'b0;
        sprite1_color = 4'b0;
        sprite2_color = 4'b0;
        sprite3_color = 4'b0;
        sprite4_color = 4'b0;
        sprite5_color = 4'b0;
        sprite6_color = 4'b0;
        sprite7_color = 4'b0;
        sprite8_color = 4'b0;
        sprite9_color = 4'b0;
```

```
      sprite1_address = ((vcount - (sprite1_y[9:1]-16)) << 5) +
(hcount[10:1] - (sprite1_x-16));
      sprite2_address = ((vcount - (sprite2_y[9:1]-16)) << 5) +
(hcount[10:1] - (sprite2_x-16));
      sprite3_address = ((vcount - (sprite3_y[9:1]-16)) << 5) +
(hcount[10:1] - (sprite3_x-16));
      sprite4_address = ((vcount - (sprite4_y[9:1]-16)) << 5) +
(hcount[10:1] - (sprite4_x-16));
      sprite5_address = ((vcount - (sprite5_y[9:1]-16)) << 5) +
(hcount[10:1] - (sprite5_x-16));
      sprite6_address = ((vcount - (sprite6_y[9:1]-16)) << 5) +
(hcount[10:1] - (sprite6_x-16));
      sprite7_address = ((vcount - (sprite7_y[9:1]-16)) << 5) +
(hcount[10:1] - (sprite7_x-16));
      sprite8_address = ((vcount - (sprite8_y[9:1]-16)) << 5) +
(hcount[10:1] - (sprite8_x-16));
      sprite9_address = ((vcount - (sprite9_y[9:1]-16)) << 5) +
(hcount[10:1] - (sprite9_x-16));
      scoreboard_address = ((vcount - (scoreboard_y[9:1]-16)) * 40) +
(hcount[10:1] - (scoreboard_x-20));
      fuelgauge_address = (({2'b00,vcount} - ({3'b000,
fuelgauge_y[9:1]}-20)) * 80) + ({2'b00,hcount[10:1]} -
({2'b00,fuelgauge_x}-40));
      indicator_address = ((vcount - (indicator_y[9:1]-16)) << 5) +
(hcount[10:1] - (indicator_x - 16));
      digit1_address = ((vcount - (digit1_y[9:1]-16)) * 20) + (hcount[10:1]
- (digit1_x-10));
      digit2_address = ((vcount - (digit2_y[9:1]-16)) * 20) + (hcount[10:1]
- (digit2_x-10));
      digit3_address = ((vcount - (digit3_y[9:1]-16)) * 20) + (hcount[10:1]
- (digit3_x-10));

      //assuming none of the images will be the same
      if(isSprite1_LATCHED) begin
            case(sprite1_img)
             0: begin
                   plane_address = sprite1_address;
                   sprite1_color = plane_out; //maybe latch the sprite
colors
                 end
             1: begin
                   chopper_address = sprite1_address;
```

```
                    sprite1_color = chopper_out;
                  end
              2: begin
                    battleship_address = sprite1_address;
                    sprite1_color = battleship_out;
                         end
              3: begin
                    fuel_address = sprite1_address;
                    sprite1_color = fuel_out;
                  end
              4: begin
                    shoot_address = sprite1_address;
                    sprite1_color = shoot_out;
                  end
              5: begin
                    explosion_address = sprite1_address;
                    sprite1_color = explosion_out; //maybe latch the sprite
colors
                  end
              6: begin
                    hotairballoon_address = sprite1_address;
                    sprite1_color = hotairballoon_out; //maybe latch the
sprite colors
                  end
              7: begin
                    planeleft_address = sprite1_address;
                    sprite1_color = planeleft_out; //maybe latch the sprite
colors
                  end
              8: begin
                    planeright_address = sprite1_address;
                    sprite1_color = planeright_out; //maybe latch the sprite
colors
                  end
              9: begin
                    battleshipreverse_address = sprite1_address;
                    sprite1_color = battleshipreverse_out; //maybe latch the
sprite colors
                  end
              10: begin
                    helicopterreverse_address = sprite1_address;
                    sprite1_color = helicopterreverse_out; //maybe latch the
sprite colors
```

```
                end

            endcase
      end

      if(isSprite2_LATCHED) begin
            case(sprite2_img)
             0: begin
                   plane_address = sprite2_address;
                   sprite2_color = plane_out;
                  end
             1: begin
                   chopper_address = sprite2_address;
                   sprite2_color = chopper_out;
                  end
             2: begin
                   battleship_address = sprite2_address;
                   sprite2_color = battleship_out;
                  end
             3: begin
                   fuel_address = sprite2_address;
                   sprite2_color = fuel_out;
                  end
             4: begin
                   shoot_address = sprite2_address;
                   sprite2_color = shoot_out;
                  end
             5: begin
                   explosion_address = sprite2_address;
                   sprite2_color = explosion_out; //maybe latch the sprite
colors
                  end
             6: begin
                   hotairballoon_address = sprite2_address;
                   sprite2_color = hotairballoon_out; //maybe latch the
sprite colors
                  end
             7: begin
                   planeleft_address = sprite2_address;
                   sprite2_color = planeleft_out; //maybe latch the sprite
colors
                  end
             8: begin
```

```
                planeright_address = sprite2_address;
                sprite2_color = planeright_out; //maybe latch the sprite
colors
               end
            9: begin
                battleshipreverse_address = sprite2_address;
                sprite2_color = battleshipreverse_out; //maybe latch the
sprite colors
               end
            10: begin
                helicopterreverse_address = sprite2_address;
                sprite2_color = helicopterreverse_out; //maybe latch the
sprite colors
               end

         endcase
   end

   if(isSprite3_LATCHED) begin
         case(sprite3_img)
          0: begin
                plane_address = sprite3_address;
                sprite3_color = plane_out;
               end
          1: begin
                chopper_address = sprite3_address;
                sprite3_color = chopper_out;
               end
          2: begin
                battleship_address = sprite3_address;
                sprite3_color = battleship_out;
               end
          3: begin
                fuel_address = sprite3_address;
                sprite3_color = fuel_out;
               end
          4: begin
                shoot_address = sprite3_address;
                sprite3_color = shoot_out;
               end
          5: begin
                explosion_address = sprite3_address;
                sprite3_color = explosion_out; //maybe latch the sprite
```

```
colors
                end
            6: begin
                hotairballoon_address = sprite3_address;
                sprite3_color = hotairballoon_out; //maybe latch the
sprite colors
                end
            7: begin
                planeleft_address = sprite3_address;
                sprite3_color = planeleft_out; //maybe latch the sprite
colors
                end
            8: begin
                planeright_address = sprite3_address;
                sprite3_color = planeright_out; //maybe latch the sprite
colors
                end
            9: begin
                battleshipreverse_address = sprite3_address;
                sprite3_color = battleshipreverse_out; //maybe latch the
sprite colors
                end
            10: begin
                helicopterreverse_address = sprite3_address;
                sprite3_color = helicopterreverse_out; //maybe latch the
sprite colors
                end

        endcase
    end

    if(isSprite4_LATCHED) begin
        case(sprite4_img)
            0: begin
                plane_address = sprite4_address;
                sprite4_color = plane_out;
                end
            1: begin
                chopper_address = sprite4_address;
                sprite4_color = chopper_out;
                end
            2: begin
                battleship_address = sprite4_address;
```

```
                    sprite4_color = battleship_out;
                end
            3: begin
                    fuel_address = sprite4_address;
                    sprite4_color = fuel_out;
                end
            4: begin
                    shoot_address = sprite4_address;
                    sprite4_color = shoot_out;
                end
            5: begin
                    explosion_address = sprite4_address;
                    sprite4_color = explosion_out; //maybe latch the sprite
colors
                end
            6: begin
                    hotairballoon_address = sprite4_address;
                    sprite4_color = hotairballoon_out; //maybe latch the
sprite colors
                end
            7: begin
                    planeleft_address = sprite4_address;
                    sprite4_color = planeleft_out; //maybe latch the sprite
colors
                end
            8: begin
                    planeright_address = sprite4_address;
                    sprite4_color = planeright_out; //maybe latch the sprite
colors
                end
            9: begin
                    battleshipreverse_address = sprite4_address;
                    sprite4_color = battleshipreverse_out; //maybe latch the
sprite colors
                end
            10: begin
                    helicopterreverse_address = sprite4_address;
                    sprite4_color = helicopterreverse_out; //maybe latch the
sprite colors
                end

        endcase
    end
```

```
      if(isSprite5_LATCHED) begin
            case(sprite5_img)
             0: begin
                   plane_address = sprite5_address;
                   sprite5_color = plane_out;
                  end
             1: begin
                   chopper_address = sprite5_address;
                   sprite5_color = chopper_out;
                  end
             2: begin
                   battleship_address = sprite5_address;
                   sprite5_color = battleship_out;
                  end
             3: begin
                   fuel_address = sprite5_address;
                   sprite5_color = fuel_out;
                  end
             4: begin
                   shoot_address = sprite5_address;
                   sprite5_color = shoot_out;
                  end
             5: begin
                   explosion_address = sprite5_address;
                   sprite5_color = explosion_out; //maybe latch the sprite
colors
                  end
             6: begin
                   hotairballoon_address = sprite5_address;
                   sprite5_color = hotairballoon_out; //maybe latch the
sprite colors
                  end
             7: begin
                   planeleft_address = sprite5_address;
                   sprite5_color = planeleft_out; //maybe latch the sprite
colors
                  end
             8: begin
                   planeright_address = sprite5_address;
                   sprite5_color = planeright_out; //maybe latch the sprite
colors
                  end
```

```
            9: begin
                  battleshipreverse_address = sprite5_address;
                  sprite5_color = battleshipreverse_out; //maybe latch the
sprite colors
                end
            10: begin
                  helicopterreverse_address = sprite5_address;
                  sprite5_color = helicopterreverse_out; //maybe latch the
sprite colors
                end

          endcase
      end

      if(isSprite6_LATCHED) begin
            case(sprite6_img)
             0: begin
                  plane_address = sprite6_address;
                  sprite6_color = plane_out; //maybe latch the sprite
colors
                end
             1: begin
                  chopper_address = sprite6_address;
                  sprite6_color = chopper_out;
                end
             2: begin
                  battleship_address = sprite6_address;
                  sprite6_color = battleship_out;
                       end
             3: begin
                  fuel_address = sprite6_address;
                  sprite6_color = fuel_out;
                end
             4: begin
                  shoot_address = sprite6_address;
                  sprite6_color = shoot_out;
                end
             5: begin
                  explosion_address = sprite6_address;
                  sprite6_color = explosion_out; //maybe latch the sprite
colors
                end
             6: begin
```

```
                  hotairballoon_address = sprite6_address;
                  sprite6_color = hotairballoon_out; //maybe latch the
sprite colors
                end
            7: begin
                  planeleft_address = sprite6_address;
                  sprite6_color = planeleft_out; //maybe latch the sprite
colors
                end
            8: begin
                  planeright_address = sprite6_address;
                  sprite6_color = planeright_out; //maybe latch the sprite
colors
                end
            9: begin
                  battleshipreverse_address = sprite6_address;
                  sprite6_color = battleshipreverse_out; //maybe latch the
sprite colors
                end
            10: begin
                  helicopterreverse_address = sprite6_address;
                  sprite6_color = helicopterreverse_out; //maybe latch the
sprite colors
                end

          endcase
      end

      if(isSprite7_LATCHED) begin
            case(sprite7_img)
             0: begin
                  plane_address = sprite7_address;
                  sprite7_color = plane_out; //maybe latch the sprite
colors
                end
            1: begin
                  chopper_address = sprite7_address;
                  sprite7_color = chopper_out;
                end
            2: begin
                  battleship_address = sprite7_address;
                  sprite7_color = battleship_out;
                       end
```

```
            3: begin
                  fuel_address = sprite7_address;
                  sprite7_color = fuel_out;
                 end
            4: begin
                  shoot_address = sprite7_address;
                  sprite7_color = shoot_out;
                 end
            5: begin
                  explosion_address = sprite7_address;
                  sprite7_color = explosion_out; //maybe latch the sprite
colors
                 end
            6: begin
                  hotairballoon_address = sprite7_address;
                  sprite7_color = hotairballoon_out; //maybe latch the
sprite colors
                 end
            7: begin
                  planeleft_address = sprite7_address;
                  sprite7_color = planeleft_out; //maybe latch the sprite
colors
                 end
            8: begin
                  planeright_address = sprite7_address;
                  sprite7_color = planeright_out; //maybe latch the sprite
colors
                 end
            9: begin
                  battleshipreverse_address = sprite7_address;
                  sprite7_color = battleshipreverse_out; //maybe latch the
sprite colors
                 end
            10: begin
                  helicopterreverse_address = sprite7_address;
                  sprite7_color = helicopterreverse_out; //maybe latch the
sprite colors
                 end

          endcase
      end

      if(isSprite8_LATCHED) begin
```

```
            case(sprite8_img)
             0: begin
                   plane_address = sprite8_address;
                   sprite8_color = plane_out; //maybe latch the sprite
colors
                  end
             1: begin
                   chopper_address = sprite8_address;
                   sprite8_color = chopper_out;
                  end
             2: begin
                   battleship_address = sprite8_address;
                   sprite8_color = battleship_out;
                        end
             3: begin
                   fuel_address = sprite8_address;
                   sprite8_color = fuel_out;
                  end
             4: begin
                   shoot_address = sprite8_address;
                   sprite8_color = shoot_out;
                  end
             5: begin
                   explosion_address = sprite8_address;
                   sprite8_color = explosion_out; //maybe latch the sprite
colors
                  end
             6: begin
                   hotairballoon_address = sprite8_address;
                   sprite8_color = hotairballoon_out; //maybe latch the
sprite colors
                  end
             7: begin
                   planeleft_address = sprite8_address;
                   sprite8_color = planeleft_out; //maybe latch the sprite
colors
                  end
             8: begin
                   planeright_address = sprite8_address;
                   sprite8_color = planeright_out; //maybe latch the sprite
colors
                  end
             9: begin
```

```
                  battleshipreverse_address = sprite8_address;
                  sprite8_color = battleshipreverse_out; //maybe latch the
sprite colors
                end
             10: begin
                  helicopterreverse_address = sprite8_address;
                  sprite8_color = helicopterreverse_out; //maybe latch the
sprite colors
                end

            endcase
      end

      if(isSprite9_LATCHED) begin
            case(sprite9_img)
             0: begin
                  plane_address = sprite9_address;
                  sprite9_color = plane_out; //maybe latch the sprite
colors
                end
             1: begin
                  chopper_address = sprite9_address;
                  sprite9_color = chopper_out;
                end
             2: begin
                  battleship_address = sprite9_address;
                  sprite9_color = battleship_out;
                       end
             3: begin
                  fuel_address = sprite9_address;
                  sprite9_color = fuel_out;
                end
             4: begin
                  shoot_address = sprite9_address;
                  sprite9_color = shoot_out;
                end
             5: begin
                  explosion_address = sprite9_address;
                  sprite9_color = explosion_out; //maybe latch the sprite
colors
                end
             6: begin
                  hotairballoon_address = sprite9_address;
```

```
                  sprite9_color = hotairballoon_out; //maybe latch the
sprite colors
                end
            7: begin
                  planeleft_address = sprite9_address;
                  sprite9_color = planeleft_out; //maybe latch the sprite
colors
                end
            8: begin
                  planeright_address = sprite9_address;
                  sprite9_color = planeright_out; //maybe latch the sprite
colors
                end
            9: begin
                  battleshipreverse_address = sprite9_address;
                  sprite9_color = battleshipreverse_out; //maybe latch the
sprite colors
                end
            10: begin
                  helicopterreverse_address = sprite9_address;
                  sprite9_color = helicopterreverse_out; //maybe latch the
sprite colors
                end

          endcase
      end

      if(isDigit1_LATCHED) begin
            case(digit1_img)
             0: begin
                  zero_address = digit1_address;
                  digit1_color = zero_out;
             end
             1: begin
                  one_address = digit1_address;
                  digit1_color = one_out;
             end
             2: begin
                  two_address = digit1_address;
                  digit1_color = two_out;
             end
             3: begin
                  three_address = digit1_address;
```

```
                digit1_color = three_out;
            end
            4: begin
                four_address = digit1_address;
                digit1_color = four_out;
            end
            5: begin
                five_address = digit1_address;
                digit1_color = five_out;
            end
            6: begin
                six_address = digit1_address;
                digit1_color = six_out;
            end
            7: begin
                seven_address = digit1_address;
                digit1_color = seven_out;
            end
            8: begin
                eight_address = digit1_address;
                digit1_color = eight_out;
            end
            9: begin
                nine_address = digit1_address;
                digit1_color = nine_out;
            end
          endcase
    end

    if (isDigit2_LATCHED) begin
          case(digit2_img)
            0: begin
                zero_address = digit2_address;
                digit2_color = zero_out;
            end
            1: begin
                one_address = digit2_address;
                digit2_color = one_out;
            end
            2: begin
                two_address = digit2_address;
                digit2_color = two_out;
            end
```

```
            3: begin
                  three_address = digit2_address;
                  digit2_color = three_out;
            end
            4: begin
                  four_address = digit2_address;
                  digit2_color = four_out;
            end
            5: begin
                  five_address = digit2_address;
                  digit2_color = five_out;
            end
            6: begin
                  six_address = digit2_address;
                  digit2_color = six_out;
            end
            7: begin
                  seven_address = digit2_address;
                  digit2_color = seven_out;
            end
            8: begin
                  eight_address = digit2_address;
                  digit2_color = eight_out;
            end
            9: begin
                  nine_address = digit2_address;
                  digit2_color = nine_out;
            end
          endcase
      end

      if(isDigit3_LATCHED) begin
            case(digit3_img)
            0: begin
                  zero_address = digit3_address;
                  digit3_color = zero_out;
            end
            1: begin
                  one_address = digit3_address;
                  digit3_color = one_out;
            end
            2: begin
                  two_address = digit3_address;
```

```
                digit3_color = two_out;
            end
            3: begin
                three_address = digit3_address;
                digit3_color = three_out;
            end
            4: begin
                four_address = digit3_address;
                digit3_color = four_out;
            end
            5: begin
                five_address = digit3_address;
                digit3_color = five_out;
            end
            6: begin
                six_address = digit3_address;
                digit3_color = six_out;
            end
            7: begin
                seven_address = digit3_address;
                digit3_color = seven_out;
            end
            8: begin
                eight_address = digit3_address;
                digit3_color = eight_out;
            end
            9: begin
                nine_address = digit3_address;
                digit3_color = nine_out;
            end
          endcase
      end

      if(sprite1_y[0]) begin
            if((hcount[10:1] < sprite1_x + 16) && (hcount[10:1] >=
sprite1_x - 16) && (vcount < sprite1_y[9:1]+16) && (vcount  >=
sprite1_y[9:1]-16)) begin // check sprite1
             //pull its contents from memory
             isSprite1 = 1;
            end
      end

      if(sprite2_y[0]) begin
```

```
            if((hcount[10:1] < sprite2_x + 16) && (hcount[10:1] >=
sprite2_x - 16) && (vcount < sprite2_y[9:1]+16) && (vcount  >=
sprite2_y[9:1]-16)) begin // check sprite2
             //pull its contents from memory
             isSprite2 = 1;
            end
      end

      if(sprite3_y[0]) begin
            if((hcount[10:1] < sprite3_x + 16) && (hcount[10:1] >=
sprite3_x - 16) && (vcount < sprite3_y[9:1] + 16) && (vcount  >=
sprite3_y[9:1] - 16)) begin // check sprite3
             //pull its contents from memory
             isSprite3 = 1;
            end
      end

      if(sprite4_y[0]) begin
            if((hcount[10:1] < sprite4_x + 16) && (hcount[10:1] >=
sprite4_x - 16) && (vcount < sprite4_y[9:1] + 16) && (vcount  >=
sprite4_y[9:1] - 16)) begin // check sprite4
             //pull its contents from memory
             isSprite4 = 1;
            end
      end

      if(sprite5_y[0]) begin
            if((hcount[10:1] < sprite5_x + 16) && (hcount[10:1] >=
sprite5_x - 16) && (vcount < sprite5_y[9:1] + 16) && (vcount  >=
sprite5_y[9:1] - 16)) begin // check sprite5
             //pull its contents from memory
             isSprite5 = 1;
            end
      end

      if(sprite6_y[0]) begin
            if((hcount[10:1] < sprite6_x + 16) && (hcount[10:1] >=
sprite6_x - 16) && (vcount < sprite6_y[9:1]+16) && (vcount  >=
sprite6_y[9:1]-16)) begin // check sprite6
             //pull its contents from memory
             isSprite6 = 1;
            end
      end
```

```
      if(sprite7_y[0]) begin
            if((hcount[10:1] < sprite7_x + 16) && (hcount[10:1] >=
sprite7_x - 16) && (vcount < sprite7_y[9:1]+16) && (vcount  >=
sprite7_y[9:1]-16)) begin // check sprite7
             //pull its contents from memory
             isSprite7 = 1;
            end
      end

      if(sprite8_y[0]) begin
            if((hcount[10:1] < sprite8_x + 16) && (hcount[10:1] >=
sprite8_x - 16) && (vcount < sprite8_y[9:1]+16) && (vcount  >=
sprite8_y[9:1]-16)) begin // check sprite8
             //pull its contents from memory
             isSprite8 = 1;
            end
      end

      if(sprite9_y[0]) begin
            if((hcount[10:1] < sprite9_x + 16) && (hcount[10:1] >=
sprite9_x - 16) && (vcount < sprite9_y[9:1]+16) && (vcount  >=
sprite9_y[9:1]-16)) begin // check sprite9
             //pull its contents from memory
             isSprite9 = 1;
            end
      end

      if(scoreboard_y[0]) begin
            if((hcount[10:1] < scoreboard_x + 20) && (hcount[10:1] >=
scoreboard_x - 20) && (vcount < scoreboard_y[9:1] + 16) && (vcount  >=
scoreboard_y[9:1] - 16)) begin // check scoreboard
             //pull its contents from memory
             isScoreboard = 1;
            end
      end

      if(fuelgauge_y[0]) begin
            if((hcount[10:1] < fuelgauge_x + 40) && (hcount[10:1] >=
fuelgauge_x - 40) && (vcount < fuelgauge_y[9:1] + 20) && (vcount  >=
fuelgauge_y[9:1] - 20)) begin // check fuelgauge
             //pull its contents from memory
             isFuelgauge = 1;
```

```
            end
      end

      if(indicator_y[0]) begin
            if((hcount[10:1] < indicator_x + 16) && (hcount[10:1] >=
indicator_x - 16) && (vcount < indicator_y[9:1] + 16) && (vcount  >=
indicator_y[9:1] - 16)) begin // check indicator
             //pull its contents from memory
             isIndicator = 1;
            end
      end

      if(digit1_y[0]) begin
            if((hcount[10:1] < digit1_x + 10) && (hcount[10:1] >= digit1_x
- 10) && (vcount < digit1_y[9:1] + 16) && (vcount  >= digit1_y[9:1] - 16))
begin // check digit1
             //pull its contents from memory
             isDigit1 = 1;
            end
      end

      if(digit2_y[0]) begin
            if((hcount[10:1] < digit2_x + 10) && (hcount[10:1] >= digit2_x
- 10) && (vcount < digit2_y[9:1] + 16) && (vcount  >= digit2_y[9:1] - 16))
begin // check digit2
             //pull its contents from memory
             isDigit2 = 1;
            end
      end

      if(digit3_y[0]) begin
            if((hcount[10:1] < digit3_x + 10) && (hcount[10:1] >= digit3_x
- 10) && (vcount < digit3_y[9:1] + 16) && (vcount  >= digit3_y[9:1] - 16))
begin // check digit3
             //pull its contents from memory
             isDigit3 = 1;
            end
      end

      if(vcount < 60) begin
            current_background = 7; //gray
      end
      else if (boundary_3_LATCHED == 0 && boundary_4_LATCHED == 0) begin //
```

```
1 River
      if  (hcount[10:1] < boundary_1_LATCHED) begin
            current_background = 1; // green
     end
      else if (hcount[10:1] < boundary_2_LATCHED) begin
            current_background = 2; // blue
      end
     else begin
      current_background = 1; // green
     end
      end
      else begin // 2 Rivers
      if  (hcount[10:1] < boundary_1_LATCHED) begin
            current_background = 1; // green
     end
      else if (hcount[10:1] < boundary_2_LATCHED) begin
            current_background = 2; // blue
     end
      else if (hcount[10:1] < boundary_3_LATCHED) begin
            current_background = 1; // green
     end
      else if (hcount[10:1] < boundary_4_LATCHED) begin
            current_background = 2; // blue
     end
     else begin
      current_background = 1; // green
     end
      end

      //priority encoding of sprites
      if (isScoreboard) begin
       current_color_NONSPRITE = scoreboard_out;
      end
      else if (isFuelgauge && fuelgauge_out != 0) begin
       current_color_NONSPRITE = fuelgauge_out;
      end
      else if (isIndicator && indicator_out != 0) begin
       current_color_NONSPRITE = indicator_out;
      end
      else if (isDigit1 && digit1_color_LATCHED != 0) begin
       current_color_NONSPRITE = digit1_color_LATCHED;
      end
      else if (isDigit2 && digit2_color_LATCHED != 0) begin
```

```
      current_color_NONSPRITE = digit2_color_LATCHED;
    end
    else if (isDigit3 && digit3_color_LATCHED != 0) begin
      current_color_NONSPRITE = digit3_color_LATCHED;
    end
    else begin
      current_color_NONSPRITE = 0;
    end


    if (isSprite1 && sprite1_color_LATCHED != 0) begin
      current_color = sprite1_color_LATCHED;
    end
    else if(isSprite2 && sprite2_color_LATCHED != 0) begin
      current_color = sprite2_color_LATCHED;
    end
    else if(isSprite3 && sprite3_color_LATCHED != 0) begin
      current_color = sprite3_color_LATCHED;
    end
    else if(isSprite4 && sprite4_color_LATCHED != 0) begin
      current_color = sprite4_color_LATCHED;
    end
    else if(isSprite5 && sprite5_color_LATCHED != 0) begin
      current_color = sprite5_color_LATCHED;
    end
    else if(isSprite6 && sprite6_color_LATCHED != 0) begin
      current_color = sprite6_color_LATCHED;
    end
    else if(isSprite7 && sprite7_color_LATCHED != 0) begin
      current_color = sprite7_color_LATCHED;
    end
    else if(isSprite8 && sprite8_color_LATCHED != 0) begin
      current_color = sprite8_color_LATCHED;
    end
    else if(isSprite9 && sprite9_color_LATCHED != 0) begin
      current_color = sprite9_color_LATCHED;
    end
    else begin
      current_color = 0;
    end
  end
```

```
    reg [11:0] counter;

    reg    [12:0]  address1;
    wire    [15:0]  q1;

    shoot audio1(.address(address1), .clock(clk), .q(q1)); //1653


    reg    [12:0]  address2;
    wire    [15:0]  q2;
    hit  audio2(.address(address2), .clock(clk), .q(q2));//1235

    reg    [12:0]  address3;
    wire    [15:0]  q3;
    bomb audio3(.address(address3), .clock(clk), .q(q3));//5832


    logic shootRegisterPrev;
    logic hitRegisterPrev;
    logic explodeRegisterPrev;

    logic playShoot;
    logic playHit;
    logic playExplode;


       logic    [15:0]  q1_intermediate;
       logic    [15:0]  q2_intermediate;
       logic    [15:0]  q3_intermediate;

    always_ff @(posedge clk) begin

       sample_data_l = q1_intermediate + q2_intermediate + q3_intermediate;
       sample_data_r = q1_intermediate + q2_intermediate + q3_intermediate;

       if(reset) begin
             sample_valid_l <= 0;
             sample_valid_r <= 0;

             address3 <= 0;
             address2 <= 0;
             address1 <= 0;
```

```
            shootRegisterPrev <= 0;
            hitRegisterPrev <= 0;
            explodeRegisterPrev <= 0;

      end

      else if(left_chan_ready == 1 && right_chan_ready == 1) begin

            if(!shootRegister) shootRegisterPrev <= 0;
            if(!hitRegister) hitRegisterPrev <= 0;
            if(!explodeRegister) explodeRegisterPrev <= 0;

            //shoot logic
            if(shootRegister == 1 && shootRegisterPrev == 0) begin
                  playShoot <= 1;
                  shootRegisterPrev <= 1;
            end

            if(playShoot) begin
                  address1 <= address1 + 1;
                  sample_valid_l <= 1;
                  sample_valid_r <= 1;
                  q1_intermediate <= q1;
            end

            if(address1 >= 1650) begin
                  address1 <= 0;
                  playShoot <= 0;
            end


            //hit logic
            if(hitRegister == 1 && hitRegisterPrev == 0) begin
                  playHit <= 1;
                  hitRegisterPrev <= 1;
            end

            if(playHit) begin
                  address2 <= address2 + 1;
                  sample_valid_l <= 1;
                  sample_valid_r <= 1;
                  q2_intermediate <= q2;
            end
```

```
            if(address2 >= 1230) begin
                  address2 <= 0;
                  playHit <= 0;
            end

            //explode logic
            if(explodeRegister == 1 && explodeRegisterPrev == 0) begin
                  playExplode <= 1;
                  explodeRegisterPrev <= 1;
            end

            if(playExplode) begin
                  address3 <= address3 + 1;
                  sample_valid_l <= 1;
                  sample_valid_r <= 1;
                  q3_intermediate <= q3;
            end

            if(address3 >= 5830) begin
                  address3 <= 0;
                  playExplode <= 0;
            end

      end
      else begin
            sample_valid_l <= 0;
            sample_valid_r <= 0;
      end
   end

endmodule



module vga_counters(
 input logic            clk50, reset,
 output logic [10:0] hcount,  // hcount[10:1] is pixel column
 output logic [9:0]  vcount,  // vcount[9:0] is pixel row
 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 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;

  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'b0; // For putting sync on the green signal;
unused

   // 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: rising edge sensitive

endmodule
```

boundary_mem.sv

```
`include "../SRAM/SRAM_twoport.v"


module boundary_mem(
    input          clk,
    input          shift,
    input          reset,
    input          [8:0] readaddress,
    input          [39:0] datain,
    output         [39:0] dataout
    );
```

```
    logic          [8:0] readbase = 9'd1;
    logic          [8:0] writebase = 9'd0;
    logic          [39:0] q_b; //non-necessary

    logic          shift_prev = 0;

    logic          [8:0] readaddress_mem;

    assign readaddress_mem = readaddress + readbase;

    SRAM_twoport     SRAM_twoport( //port a will be read, port b will be
write
       .address_a(readaddress_mem),
       .address_b(writebase),
       .clock(clk),
       .data_a(39'bX),
       .data_b(datain),
       .wren_a(1'b0),
       .wren_b(1'b1),
       .q_a(dataout),
       .q_b(q_b)
    );

    always_ff @(negedge clk) begin

       if(reset) begin
             readbase <= 9'd1;
             writebase <= 9'd0;
             shift_prev <= shift;
       end
       if (shift != shift_prev) begin
             readbase <= readbase-1;
             writebase <= writebase-1;
             shift_prev <= shift;
       end
    end


endmodule
```

driver.h

```
//
// Created by Yongmao Luo on 4/8/22.
//

#ifndef WATER_RAID_DRIVER_H
#define WATER_RAID_DRIVER_H

#include "common_data_structure.h"
#include "../VideoDriver/water_video.h"

class WaterDriver{
public:
   // video
   static void initBackground(int videoFd); // set up the fuel gauge and
scoreboard
   static void writeBoundary(int videoFd, BoundaryInRow boundary); // write
boundary for each row
   static void writePosition(int videoFd,Position position,int type, int
index); // write position for each sprite
   static void writeFuel(int videoFd,int fuel); // adjust the indicator of
the fuel gauge
   static void writeScore(int videoFd,int score); // change the scores in
the scoreboard
   // audio
   static void playAudio(int audioFd,int index); // play audio of different
sound effect

};

#endif //WATER_RAID_DRIVER_H
```

driver.cpp

```
//
// Created by Yongmao Luo on 4/29/22.
//

#include "driver.h"
#include "../VideoDriver/water_video.h"
#include <unistd.h>
#include <sys/ioctl.h>
```

```
#include <stdio.h>

int shift=0;

void WaterDriver::writeBoundary(int videoFd, BoundaryInRow boundary) {
   water_video_arg_boundary arg;
   arg.boundary=boundary;
   shift=1-shift;
   arg.shift=shift;

   if (ioctl(videoFd, WATER_VIDEO_WRITE_BOUNDARY, &arg)) {
       perror("ioctl(WATER_VIDEO_WRITE_BOUNDARY) failed");
       return;
   }
}

void WaterDriver::writePosition(int videoFd,Position position,int type, int
index) {
   water_video_arg_position arg;
   arg.pos=position;
   arg.type=type;
   arg.index=index;

   if (ioctl(videoFd, WATER_VIDEO_WRITE_POSITION, &arg)) {
       perror("ioctl(WATER_VIDEO_WRITE_POSITION) failed");
       return;
   }

}

void WaterDriver::writeFuel(int videoFd, int fuel) {
   water_video_arg_fuel arg;
   arg.fuel=fuel;

   if (ioctl(videoFd, WATER_VIDEO_WRITE_FUEL, &arg)) {
       perror("ioctl(WATER_VIDEO_WRITE_FUEL) failed");
       return;
   }
}

void WaterDriver::writeScore(int videoFd, int score) {
   water_video_arg_score arg;
   arg.score=score;
```

```
    if (ioctl(videoFd, WATER_VIDEO_WRITE_SCORE, &arg)) {
        perror("ioctl(WATER_VIDEO_SET_SCORE) failed");
        return;
    }
}

void WaterDriver::initBackground(int videoFd) {
    water_video_arg_init arg;
    arg.scorePos.x=480;
    arg.scorePos.y=(30 << 1) + 1;
    arg.digit1Pos.x=525;
    arg.digit1Pos.y=(30 << 1) + 1;
    arg.digit2Pos.x=550;
    arg.digit2Pos.y=(30 << 1) + 1;
    arg.digit3Pos.x=575;
    arg.digit3Pos.y=(30 << 1) + 1;
    arg.fuelPos.x=320;
    arg.fuelPos.y=(30 << 1) + 1;
    arg.indicatorPos.x=320;
    arg.indicatorPos.y=(28 << 1) + 1;
    if (ioctl(videoFd, WATER_VIDEO_INIT,&arg)) {
        perror("ioctl(WATER_VIDEO_INIT) failed");
        return;
    }
}

void WaterDriver::playAudio(int audioFd, int index) {
    water_audio_arg arg;
    arg.index=index;

    if (ioctl(audioFd, WATER_AUDIO_PLAY, &arg)) {
        perror("ioctl(WATER_VIDEO_SET_SCORE) failed");
        return;
    }

}
```

Water_video.h

```
#ifndef _WATER_VIDEO_H
```

```c
#define _WATER_VIDEO_H

#include <linux/ioctl.h>
#include "../GameLogic/common_data_structure.h"

// typedef struct {
//     unsigned char red, green, blue;
// } vga_ball_color_t;

typedef struct{
   BoundaryInRow boundary;
   short shift;
}water_video_arg_boundary;

typedef struct{
   Position pos;
   short type;
   short index;
}water_video_arg_position;

typedef struct {
   short fuel;
}water_video_arg_fuel;

typedef struct {
   short score;
}water_video_arg_score;

typedef struct {
   Position fuelPos,scorePos,digit1Pos,digit2Pos,digit3Pos,indicatorPos;
}water_video_arg_init;

typedef struct {
   short index;
}water_audio_arg;

#define WATER_VIDEO_MAGIC 'q'

/* ioctls and their arguments */
#define WATER_VIDEO_WRITE_BOUNDARY _IOW(WATER_VIDEO_MAGIC, 1,
water_video_arg_boundary *)
```

```
#define WATER_VIDEO_WRITE_POSITION _IOR(WATER_VIDEO_MAGIC, 2,
water_video_arg_position *)
#define WATER_VIDEO_WRITE_FUEL _IOR(WATER_VIDEO_MAGIC, 3,
water_video_arg_fuel *)
#define WATER_VIDEO_WRITE_SCORE _IOR(WATER_VIDEO_MAGIC, 4,
water_video_arg_score *)
#define WATER_VIDEO_INIT _IOR(WATER_VIDEO_MAGIC, 5, water_video_arg_init
*)
#define WATER_AUDIO_PLAY _IOR(WATER_VIDEO_MAGIC, 6, water_audio_arg *)

#endif
```

Water_video.c

```
/* * Video and Audio driver for the Water Raid Project
*
* A Platform device implemented using the misc subsystem
*
* Yongmao Luo
* Columbia University
*
* References:
* Linux source: Documentation/driver-model/platform.txt
*               drivers/misc/arm-charlcd.c
* http://www.linuxforu.com/tag/linux-device-drivers/
* http://free-electrons.com/docs/
*
* "make" to build
* insmod water_video.ko
*
*/

#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
```

```c
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "water_video.h"

#define DRIVER_NAME "water_video"

/* Device registers */
#define BOUNDARY0(x) (x)
#define BOUNDARY1(x) ((x)+2)
#define BOUNDARY2(x) ((x)+4)
#define BOUNDARY3(x) ((x)+6)
#define SHIFT(x) ((x)+8)
#define SCOREBOARDX(x) ((x) + 64)
#define SCOREBOARDY(x) ((x) + 66)
#define DIGIT1X(x) ((x) + 68)
#define DIGIT1Y(x) ((x) + 70)
#define DIGIT1IMG(x) ((x) + 72)
#define DIGIT2X(x) ((x) + 74)
#define DIGIT2Y(x) ((x) + 76)
#define DIGIT2IMG(x) ((x) + 78)
#define DIGIT3X(x) ((x) + 80)
#define DIGIT3Y(x) ((x) + 82)
#define DIGIT3IMG(x) ((x) + 84)
#define FUELGAUGEX(x) ((x) + 86)
#define FUELGAUGEY(x) ((x) + 88)
#define INDICATORX(x) ((x) + 90)
#define INDICATORY(x) ((x) + 92)
#define SHOOTAUDIO(x) ((x)+94)
#define HITAUDIO(x) ((x)+96)
#define EXPLODEAUDIO(x) ((x)+98)

#define FUELGAUGEHALFLENGTH 40

/*
* Information about our device
*/
struct water_video_dev {
```

```
  struct resource res; /* Resource: our registers */
  void __iomem *virtbase; /* Where registers can be accessed in memory
*/
  water_video_arg_boundary argBoundary;
     water_video_arg_position argPosition;
     water_video_arg_fuel argFuel;
     water_video_arg_score argScore;
     water_video_arg_init argInit;
     water_audio_arg argAudio;
} dev;

/*
* Write segments of a single digit
* Assumes digit is in range and the device information has been set up
*/
// static void write_background(water_video_color_t *background)
// {
//     iowrite8(background->red, BG_RED(dev.virtbase) );
//     iowrite8(background->green, BG_GREEN(dev.virtbase) );
//     iowrite8(background->blue, BG_BLUE(dev.virtbase) );
//     dev.background = *background;
// }

static void writeBoundary(water_video_arg_boundary *arg)
{
   iowrite16(arg->boundary.river1_left, BOUNDARY0(dev.virtbase) );
   iowrite16(arg->boundary.river1_right, BOUNDARY1(dev.virtbase) );
   iowrite16(arg->boundary.river2_left, BOUNDARY2(dev.virtbase) );
   iowrite16(arg->boundary.river2_right, BOUNDARY3(dev.virtbase) );
   iowrite16(arg->shift, SHIFT(dev.virtbase));
   dev.argBoundary = *arg;
}

/*
* Write 16 bits for each location variable
* Assumes digit is in range and the device information has been set up
*/
static void writePosition(water_video_arg_position *arg)
{
   // index of sprites starting from x+10
  iowrite16(arg->pos.x, dev.virtbase+10+arg->index*6 );
```

```
  iowrite16(arg->pos.y, dev.virtbase+10+arg->index*6+2 );
   iowrite16(arg->type, dev.virtbase+10+arg->index*6+4 );

   dev.argPosition = *arg;
}

static void writeFuel(water_video_arg_fuel *arg)
{
   iowrite16(320-FUELGAUGEHALFLENGTH+arg->fuel, INDICATORX(dev.virtbase)
);
   dev.argFuel = *arg;
}

static void writeScore(water_video_arg_score *arg)
{
   short score=arg->score;
   int i;
   for(i=0;i<3;i++){
       iowrite16(score%10, DIGIT3IMG(dev.virtbase)-i*6 );
       score/=10;
   }
   dev.argScore = *arg;
}

static void initBackground(water_video_arg_init *arg){
   iowrite16(arg->scorePos.x, SCOREBOARDX(dev.virtbase));
   iowrite16(arg->scorePos.y, SCOREBOARDY(dev.virtbase));
   iowrite16(arg->digit1Pos.x, DIGIT1X(dev.virtbase));
   iowrite16(arg->digit1Pos.y, DIGIT1Y(dev.virtbase));
   iowrite16(0, DIGIT1IMG(dev.virtbase));
   iowrite16(arg->digit2Pos.x, DIGIT2X(dev.virtbase));
   iowrite16(arg->digit2Pos.y, DIGIT2Y(dev.virtbase));
   iowrite16(0, DIGIT2IMG(dev.virtbase));
   iowrite16(arg->digit3Pos.x, DIGIT3X(dev.virtbase));
   iowrite16(arg->digit3Pos.y, DIGIT3Y(dev.virtbase));
   iowrite16(0, DIGIT3IMG(dev.virtbase));
   iowrite16(arg->fuelPos.x, FUELGAUGEX(dev.virtbase));
   iowrite16(arg->fuelPos.y, FUELGAUGEY(dev.virtbase));
   iowrite16(arg->indicatorPos.x, INDICATORX(dev.virtbase));
   iowrite16(arg->indicatorPos.y, INDICATORY(dev.virtbase));
}
```

```
static void playAudio(water_audio_arg *arg){

   iowrite16(1, SHOOTAUDIO(dev.virtbase)+arg->index*2);
   iowrite16(0, SHOOTAUDIO(dev.virtbase)+arg->index*2);
}

/*
* Handle ioctl() calls from userspace:
* Read or write the segments on single digits.
* Note extensive error checking of arguments
*/
static long water_video_ioctl(struct file *f, unsigned int cmd, unsigned
long arg)
{
   water_video_arg_boundary argBoundary;
   water_video_arg_position argPosition;
   water_video_arg_fuel argFuel;
   water_video_arg_score argScore;
   water_video_arg_init argInit;
   water_audio_arg argAudio;

  switch (cmd) {
  // case water_video_WRITE_BACKGROUND:
  //     if (copy_from_user(&vla, (water_video_arg_t *) arg,
  //                sizeof(water_video_arg_t)))
  //        return -EACCES;
  //     write_background(&vla.background);
  //     break;

       case WATER_VIDEO_WRITE_BOUNDARY:
           if (copy_from_user(&argBoundary, (water_video_arg_boundary *)
arg,
                      sizeof(argBoundary)))
               return -EACCES;
           writeBoundary(&argBoundary);
           break;

       case WATER_VIDEO_WRITE_POSITION:
           if (copy_from_user(&argPosition, (water_video_arg_position *)
arg,
```

```
                            sizeof(argPosition)))
                return -EACCES;
                writePosition(&argPosition);
                break;
        case WATER_VIDEO_WRITE_FUEL:
            if (copy_from_user(&argFuel, (water_video_arg_fuel *) arg,
                            sizeof(argFuel)))
                return -EACCES;
                writeFuel(&argFuel);
                break;
        case WATER_VIDEO_WRITE_SCORE:
            if (copy_from_user(&argScore, (water_video_arg_score *) arg,
                            sizeof(argScore)))
                return -EACCES;
            writeScore(&argScore);
            break;
        case WATER_VIDEO_INIT:
            if (copy_from_user(&argInit, (water_video_arg_init *) arg,
                            sizeof(argInit)))
                return -EACCES;
            initBackground(&argInit);
            break;
        case WATER_AUDIO_PLAY:
            if (copy_from_user(&argAudio, (water_video_arg_init *) arg,
                            sizeof(argAudio)))
                return -EACCES;
            playAudio(&argAudio);
            break;


// case WATER_VIDEO_READ_BACKGROUND:
//     vla.background = dev.background;
//     if (copy_to_user((water_video_arg_t *) arg, &vla,
//             sizeof(water_video_arg_t)))
//         return -EACCES;
//     break;

default:
   return -EINVAL;
}
```

```
  return 0;
}

/* The operations our device knows how to do */
static const struct file_operations water_video_fops = {
  .owner    = THIS_MODULE,
  .unlocked_ioctl = water_video_ioctl,
};

/* Information about our device for the "misc" framework -- like a char
dev */
static struct miscdevice water_video_misc_device = {
  .minor    = MISC_DYNAMIC_MINOR,
  .name     = DRIVER_NAME,
  .fops     = &water_video_fops,
};

/*
* Initialization code: get resources (registers) and display
* a welcome message
*/
static int __init water_video_probe(struct platform_device *pdev)
{
  int ret;

  /* Register ourselves as a misc device: creates /dev/water_video */
  ret = misc_register(&water_video_misc_device);

  /* Get the address of our registers from the device tree */
  ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
  if (ret) {
     ret = -ENOENT;
     goto out_deregister;
  }

  /* Make sure we can use these registers */
  if (request_mem_region(dev.res.start, resource_size(&dev.res),
                DRIVER_NAME) == NULL) {
     ret = -EBUSY;
     goto out_deregister;
  }
```

```
  /* Arrange access to our registers */
  dev.virtbase = of_iomap(pdev->dev.of_node, 0);
  if (dev.virtbase == NULL) {
     ret = -ENOMEM;
     goto out_release_mem_region;
  }

  return 0;

out_release_mem_region:
  release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
  misc_deregister(&water_video_misc_device);
  return ret;
}

/* Clean-up code: release resources */
static int water_video_remove(struct platform_device *pdev)
{
  iounmap(dev.virtbase);
  release_mem_region(dev.res.start, resource_size(&dev.res));
  misc_deregister(&water_video_misc_device);
  return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id water_video_of_match[] = {
  { .compatible = "csee4840,vga_ball-1.0" },
  {},
};
MODULE_DEVICE_TABLE(of, water_video_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver water_video_driver = {
  .driver    = {
     .name  = DRIVER_NAME,
     .owner = THIS_MODULE,
     .of_match_table = of_match_ptr(water_video_of_match),
```

```
  },
  .remove     = __exit_p(water_video_remove),
};

/* Called when the module is loaded: set things up */
static int __init water_video_init(void)
{
  pr_info(DRIVER_NAME ": init\n");
  return platform_driver_probe(&water_video_driver, water_video_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit water_video_exit(void)
{
  platform_driver_unregister(&water_video_driver);
  pr_info(DRIVER_NAME ": exit\n");
}

module_init(water_video_init);
module_exit(water_video_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Yongmao Luo, Columbia University");
MODULE_DESCRIPTION("video driver of Water-Raid Project");
```

Main.cpp

```
//
// Created by Yongmao Luo on 4/1/22.
//

#include "bullet.h"

#include "game_scenario.h"
#include "airplane.h"

#include "battleship.h"
#include "enemy_plane.h"
#include "driver.h"
```

```
#include "common_data_structure.h"
#include "sprite.h"
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include<ctime>
#include<iostream>

// type of different sprite
#define SPRITE_PLANE 0
#define SPRITE_HELI 1
#define SPRITE_BATTLE 2
#define SPRITE_FUEL 3
#define SPRITE_BULLET 4
#define SPRITE_EXPLODE 5
#define SPRITE_BALLOON 6



// center coordinate related to upper left corner
#define SPRITE_X 12
#define SPRITE_Y 12
#define AIRPLANE_X 14
#define AIRPLANE_Y 14

#define MINIMUM_RIVER_WIDTH 50
#define MAXFUEL 75

#define SHOOT_AUDIO 0
#define HIT_AUDIO 1
#define EXPLODE_AUDIO 2

using namespace std;

// the struct used to store different kinds of sprites

int main() {
   vector<Bullet> bulletList;
   vector<Battleship> battleList;
   vector<EnemyPlane> enemyList;
   vector<short> spriteIndexList;
   vector<FuelTank> fuelTankList;
   Airplane airplane;
```

```
   static const char xbox[] = "/dev/input/event0";
   static const char waterVideo[]="/dev/water_video";
   int videoFd,xboxFd;

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

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

   GameScenario gameScenario(videoFd,60,640,60);
   double duration=1/gameScenario.getFrequency();
   clock_t execute=clock();
   clock_t reduceFuelClock=clock();
   clock_t moveClock=clock();
   clock_t counterSurvival;
   gameScenario.setChangeClock();
   bool timeToMove;
   while(1) {

       for (int i = 0; i < bulletList.size(); i++){
           bulletList[i].setCrash();
           WaterDriver::writePosition(videoFd, bulletList[i].getPosition(),
SPRITE_BULLET,
                                      bulletList[i].getIndex());
       }

       for (int i = 0; i < battleList.size(); i++){
           battleList[i].disappear();
           WaterDriver::writePosition(videoFd, battleList[i].getPos(),
SPRITE_BATTLE,
                                      battleList[i].getIndex());
       }

       for (int i = 0; i < enemyList.size(); i++){
           enemyList[i].disappear();
           WaterDriver::writePosition(videoFd, enemyList[i].getPos(),
SPRITE_HELI,
```

```
                                    enemyList[i].getIndex());
        }

        for (int i = 0; i < fuelTankList.size(); i++){
            fuelTankList[i].disappear();
            WaterDriver::writePosition(videoFd, fuelTankList[i].getPos(),
SPRITE_FUEL,
                                    fuelTankList[i].getIndex());
        }

        bulletList.clear();
        enemyList.clear();
        battleList.clear();
        fuelTankList.clear();

        gameScenario.initBackground(videoFd);
        WaterDriver::initBackground(videoFd);

        spriteIndexList = {4,5,6,7,8};
        counterSurvival=0;
        timeToMove= false;


        // wait the button on the xbox controller to be pressed
        /* wait to be finished */

        bool isCrashed;
        Position tempPos;
        tempPos.x = 320;
        tempPos.y = (512 << 1) + 1;
        Shape tempShape;
        tempShape.width = AIRPLANE_X;
        tempShape.length = AIRPLANE_Y;
        airplane = Airplane(SPRITE_PLANE, MAXFUEL, tempPos, tempShape, 0);
        WaterDriver::writeScore(videoFd,0);

        // the plane enter the screen
        while (airplane.getPos().y > (400 << 1) + 1) {
            WaterDriver::writePosition(videoFd, airplane.getPos(),
SPRITE_PLANE, 0);
            tempPos = airplane.getPos();
            tempPos.y -= 2;
            airplane.setPos(tempPos);
```

```
            usleep(5000);
        }

        // wait for user to press button A so we can start the game
        while(!airplane.startGame()){
            airplane.receiveFromXbox(xboxFd);
        }

        int iteration = 35;
        while (1) {
            if (double(clock() - execute) / CLOCKS_PER_SEC >= duration) {

                if(double(clock()-moveClock)/CLOCKS_PER_SEC>0.1){
                    moveClock=clock();
                    timeToMove=true;
                }
                execute = clock();
                gameScenario.updateBackground(videoFd);

                //receive control signal from xbox
                airplane.receiveFromXbox(xboxFd);
                airplane.calPos(videoFd);

                // determine if the plane has crashed
                // plane is always located at y=400
                BoundaryInRow boundaryAheadOfPlane;
                if(gameScenario.getScreenHeader() -400 + SPRITE_Y>=0)
                    boundaryAheadOfPlane = gameScenario.boundaries[
                        gameScenario.getScreenHeader() -400 + SPRITE_Y];
                else
                    boundaryAheadOfPlane = gameScenario.boundaries[
                            gameScenario.getScreenHeader() -400+480 +
SPRITE_Y];

                isCrashed = airplane.isCrashed(videoFd,
boundaryAheadOfPlane)|| airplane.isCrashed(videoFd,enemyList,battleList);
                if (isCrashed){
                    WaterDriver::playAudio(videoFd,EXPLODE_AUDIO);
                    break;
                }

/                // if the plane bumped into the fuel tank
                airplane.addFuel(videoFd,fuelTankList,spriteIndexList);
```

```
                // check if the airplane is about to emit a bullet
                airplane.fire(xboxFd,videoFd,bulletList);
                //printf("bullet list size: %d\n",bulletList.size());

                // reduce fuel
                if ((clock() - reduceFuelClock) / CLOCKS_PER_SEC >= 1) {
                    counterSurvival++;
                    reduceFuelClock = clock();
                    int temp = airplane.reduceFuel(videoFd);
                    if (temp == -1){
                        tempPos.y=0;

WaterDriver::writePosition(videoFd,tempPos,airplane.getType(),0);

WaterDriver::writePosition(videoFd,airplane.getPos(),SPRITE_EXPLODE,0);
                        break;
                    }
                }

                // add survival scores
                if(counterSurvival>=2){
                    counterSurvival=0;
                    airplane.scores+=1;
                    WaterDriver::writeScore(videoFd,airplane.scores);
                }

                //bullet fly
                Bullet::fly(videoFd,bulletList);


                //enemy plane move
                for (int i = 0; i < enemyList.size(); i++)
                {
                    enemyList[i].pos.y += 2;
                    if(enemyList[i].pos.y >= (512 << 1) + 1){
                        enemyList[i].disappear();
                        spriteIndexList.push_back(enemyList[i].getIndex());

WaterDriver::writePosition(videoFd,enemyList[i].pos,enemyList[i].getType(),
enemyList[i].getIndex());
                        enemyList.erase(enemyList.begin()+i);
                    } else if(timeToMove){
```

```
                    if(gameScenario.getScreenHeader() -
int((enemyList[i].getPos().y -1) >> 1)>=0){

enemyList[i].move(videoFd,gameScenario.boundaries[

gameScenario.getScreenHeader() - int((enemyList[i].getPos().y -1) >> 1)],
2);
                    }else

enemyList[i].move(videoFd,gameScenario.boundaries[

(gameScenario.getScreenHeader() - int((enemyList[i].getPos().y -1) >> 1) +
                                                480 )], 2);
                }
                WaterDriver::writePosition(videoFd,
enemyList[i].getPos(), enemyList[i].getType(),
                                                enemyList[i].getIndex());
            }

            //battleship move
            for (int i = 0; i < battleList.size(); i++)
            {
                battleList[i].pos.y += 2;
                if(battleList[i].pos.y >=  (512 << 1) + 1){
                    battleList[i].disappear();
                    spriteIndexList.push_back(battleList[i].getIndex());

WaterDriver::writePosition(videoFd,battleList[i].pos,battleList[i].getType(
),battleList[i].getIndex());
                    battleList.erase(battleList.begin()+i);
                } else if(timeToMove){
                    if((gameScenario.getScreenHeader() -
int((battleList[i].getPos().y - 1) >> 1) >=0))

battleList[i].move(videoFd,gameScenario.boundaries[(gameScenario.getScreenH
eader() - int((battleList[i].getPos().y - 1) >> 1) )],
                                                2);
                    else

battleList[i].move(videoFd,gameScenario.boundaries[(gameScenario.getScreenH
eader() - int((battleList[i].getPos().y - 1) >> 1) + 480) ],
                                                2);
```

```
                }
                WaterDriver::writePosition(videoFd,
battleList[i].getPos(), battleList[i].getType(),
                                           battleList[i].getIndex());
            }

            //fuel tank move
            for (int i = 0; i < fuelTankList.size(); i++)
            {
                fuelTankList[i].pos.y += 2;
                if(fuelTankList[i].pos.y >=  (512 << 1) + 1){
                    fuelTankList[i].disappear();

spriteIndexList.push_back(fuelTankList[i].getIndex());

WaterDriver::writePosition(videoFd,fuelTankList[i].pos,fuelTankList[i].getT
ype(),fuelTankList[i].getIndex());
                    fuelTankList.erase(fuelTankList.begin()+i);
                } else if(timeToMove){
                    if(gameScenario.getScreenHeader() -
int((fuelTankList[i].getPos().y - 1) >> 1)>=0)

fuelTankList[i].move(videoFd,gameScenario.boundaries[(gameScenario.getScree
nHeader() - int((fuelTankList[i].getPos().y - 1) >> 1)) ],
                                          2);
                    else

fuelTankList[i].move(videoFd,gameScenario.boundaries[(gameScenario.getScree
nHeader() - int((fuelTankList[i].getPos().y - 1) >> 1) + 480 )],
                                            2);
                }
                WaterDriver::writePosition(videoFd,
fuelTankList[i].getPos(), fuelTankList[i].getType(),
                                           fuelTankList[i].getIndex());
            }

            if(timeToMove){
                timeToMove= false;
            }

            //check the enemy plane to see if hit
            for (int i = 0; i < enemyList.size(); i++)
            {
```

```
                    enemyList[i].checkIfHit(bulletList,
videoFd,airplane.scores);
                }
                //destroy and release the index
                for (int i = 0; i < enemyList.size(); i++)
                {
                    if(enemyList[i].getIsDestroy()){
                        enemyList[i].disappear();
                        WaterDriver::writePosition(videoFd,
enemyList[i].getPos(), enemyList[i].getType(),
                                                    enemyList[i].getIndex());
                        spriteIndexList.push_back(enemyList[i].getIndex());
                        enemyList.erase(enemyList.begin()+i);
                    }
                }

                //check the battleship to see if hit
                for (int i = 0; i < battleList.size(); i++)
                {
                    battleList[i].checkIfHit(bulletList,videoFd,
airplane.scores);
                }
                //destroy and release the index
                for (int i = 0; i < battleList.size(); i++)
                {
                    if(battleList[i].getIsDestroy()){
                        battleList[i].disappear();
                        WaterDriver::writePosition(videoFd,
battleList[i].getPos(), SPRITE_BATTLE,

battleList[i].getIndex());
                        spriteIndexList.push_back(battleList[i].getIndex());
                        battleList.erase(battleList.begin()+i);
                    }
                }

                //check the fuel tank to see if hit
                for (int i = 0; i < fuelTankList.size(); i++)
                {
                    fuelTankList[i].checkIfHit(bulletList,videoFd);
                }
                //destroy and release the index
                for (int i = 0; i < fuelTankList.size(); i++)
```

```
            {
                if(fuelTankList[i].getIsDestroy()){
                    fuelTankList[i].disappear();
                    WaterDriver::writePosition(videoFd,
fuelTankList[i].getPos(), SPRITE_FUEL,

fuelTankList[i].getIndex());

spriteIndexList.push_back(fuelTankList[i].getIndex());
                    fuelTankList.erase(fuelTankList.begin()+i);
                }
            }

            // update bullet (for aimed)
            for (int i = 0; i < bulletList.size(); i++) {
                if (bulletList[i].getIsCrashed()) {
                    WaterDriver::writePosition(videoFd,
bulletList[i].getPosition(), SPRITE_BULLET,

bulletList[i].getIndex());
                    bulletList.erase(bulletList.begin() + i);
                }
            }
            iteration ++;

            //if possible, randomly generate sprite
            if (!spriteIndexList.empty() && iteration >=35) {
                iteration = 0;
                switch (rand() % 7) {
                case 0: case 6: {    //for generate enemy plane
                        Shape newShape;
                        newShape.width = SPRITE_X;
                        newShape.length = SPRITE_Y;
                        bool canMove= false;
                        if(rand()%5==0){
                            canMove= true;
                        }
                        EnemyPlane enemyPlane = EnemyPlane(SPRITE_HELI,
1, newShape, false, 2, spriteIndexList[0],
                                                                canMove);
                        spriteIndexList.erase(spriteIndexList.begin());
                        short randomRow = 0;
                        short rowIndex = (gameScenario.getScreenHeader()
```

```
- randomRow) % 480;
                            BoundaryInRow boundaryGenerateInRow =
gameScenario.boundaries[rowIndex];
                            enemyPlane.generate(boundaryGenerateInRow,
randomRow);
                            enemyList.push_back(enemyPlane);
                            WaterDriver::writePosition(videoFd,
enemyPlane.getPos(), SPRITE_HELI,

enemyPlane.getIndex());
                            break;
                        }
                        case 1: case 4: {   //for generate enemy battleship
                            Shape newShape;
                            newShape.width = SPRITE_X;
                            newShape.length = SPRITE_Y;
                            bool canMove= true;
                            if(rand()%5==0){
                                canMove= false;
                            }
                            Battleship battleship =
Battleship(SPRITE_BATTLE, 2, newShape, false, 3,
spriteIndexList[0],canMove);
                            spriteIndexList.erase(spriteIndexList.begin());
                            short randomRow = 0;
                            short rowIndex = (gameScenario.getScreenHeader()
- randomRow) % 480;
                            BoundaryInRow boundaryGenerateInRow =
gameScenario.boundaries[rowIndex];
                            battleship.generate(boundaryGenerateInRow,
randomRow);
                            battleList.push_back(battleship);
                            WaterDriver::writePosition(videoFd,
battleship.getPos(), SPRITE_BATTLE,

battleship.getIndex());
                            break;
                        }

                        case 2: {//for generate enemy fuel_tank
                            Shape newShape;
                            newShape.width = SPRITE_X;
                            newShape.length = SPRITE_Y;
```

```
                            bool canMove= false;
                            if(rand()%5==0){
                                canMove= true;
                            }
                            FuelTank fuelTank = FuelTank(SPRITE_FUEL, 1,
newShape, false, 2, spriteIndexList[0],canMove);
                            spriteIndexList.erase(spriteIndexList.begin());
                            short randomRow = 0;
                            short rowIndex = (gameScenario.getScreenHeader()
- randomRow ) % 480;
                            BoundaryInRow boundaryGenerateInRow =
gameScenario.boundaries[rowIndex];
                            fuelTank.generate(boundaryGenerateInRow,
randomRow);
                            fuelTankList.push_back(fuelTank);
                            WaterDriver::writePosition(videoFd,
fuelTank.getPos(), SPRITE_FUEL,
                                                                fuelTank.getIndex());
                            break;
                        }
                        case 3:  case 5:  {    //for generate enemy plane
                                Shape newShape;
                                newShape.width = SPRITE_X;
                                newShape.length = SPRITE_Y;
                                bool canMove= false;
                                if(rand()%5==0){
                                    canMove= true;
                                }
                                EnemyPlane enemyPlane =
EnemyPlane(SPRITE_BALLOON, 1, newShape, false, 2, spriteIndexList[0],
                                                                        canMove);

spriteIndexList.erase(spriteIndexList.begin());
                                short randomRow = 0;
                                short rowIndex =
(gameScenario.getScreenHeader() - randomRow) % 480;
                                BoundaryInRow boundaryGenerateInRow =
gameScenario.boundaries[rowIndex];
                                enemyPlane.generate(boundaryGenerateInRow,
randomRow);
                                enemyList.push_back(enemyPlane);
                                WaterDriver::writePosition(videoFd,
enemyPlane.getPos(), SPRITE_BALLOON,
```

```
enemyPlane.getIndex());
                                break;
                            }
                        }
                    }
                }
            }
    }
    return 0;
}
```

Gamescenario.h

```
//
// Created by Yongmao Luo on 4/1/22.
//

#ifndef WATER_RAID_GAME_Scenario_H
#define WATER_RAID_GAME_Scenario_H

#include "string.h"
#include "common_data_structure.h"
#include "time.h"
#include "stdlib.h"

#define INCREASE_WIDTH 0
#define DECREASE_WIDTH 1
#define DOUBLE_RIVER 2
#define SINGLE_RIVER 3

class GameScenario
{
private:
    short minimumWidth; // the minimum width of the river
    short maximumWidth; // the maximum with of the river
    double frequency; // how many lines the plane flies over per second
    short screenHeader; // the header of the circle queue
    short states; // the state of the state machine
    int singleRiverWidth; // when we double the river, we need to record the
former width of the river
```

```
   bool firstTimeDouble; // indicator for first time the state becomes
DOUBLE_RIVER
   clock_t change; // clock used to adjust the frequency of randomly select
new state

public:
   BoundaryInRow boundaries[480];/* background register */
   void updateBackground(int videoFd); // randomly generate new boundaries
by maintaining a state machine
   void initBackground(int videoFd); // at the start of each round of game,
flash the background to the same
   double getFrequency();
   void setChangeClock();
   int getScreenHeader();
   GameScenario(int videoFd,short minimumWidth, short maximumWidth, short
frequency);

   BoundaryInRow get_boundaries(){
       BoundaryInRow temp;
       temp.river1_left=boundaries[0].river1_left;
       temp.river1_right=boundaries[0].river1_right;
       temp.river2_left=boundaries[0].river2_left;
       temp.river2_right=boundaries[0].river2_right;
       return temp;
   }

};

#endif
```

Gamescenario.cpp

```
//
// Created by Yongmao Luo on 4/6/22.
//

#include "game_scenario.h"
#include "driver.h"
#include "stdio.h"
#include <unistd.h>

void GameScenario::initBackground(int videoFd) {
```

```
   for(int i=0;i<480;i++){
       boundaries[i].river1_left=220;
       boundaries[i].river1_right=420;
       boundaries[i].river2_left=0;
       boundaries[i].river2_right=0;
       WaterDriver::writeBoundary(videoFd,boundaries[i]);
       usleep(5000);
   }
   states=0;
   firstTimeDouble= true;
   screenHeader=0;
}

int doubleRiverCounter=0;

void GameScenario::updateBackground(int videoFd) {
   BoundaryInRow newBoundaries;
   srand (time(NULL));

           // a round is over, need to change the background
           if(double(clock()-change)/CLOCKS_PER_SEC>=2){
               firstTimeDouble= true;
               int temp=rand()%4;
               change=clock();
               states=temp;
           }


           switch (states) {
               case INCREASE_WIDTH:
                   if(boundaries[screenHeader].river2_left==0){

if(boundaries[screenHeader].river1_right-boundaries[screenHeader].river1_le
ft<=maximumWidth/2-2){

newBoundaries.river1_left=boundaries[screenHeader].river1_left-1;

newBoundaries.river1_right=boundaries[screenHeader].river1_right+1;

newBoundaries.river2_left=newBoundaries.river2_right=0;
                           // move down the boundary matrix
                           screenHeader=(screenHeader+1)%480;
                           boundaries[screenHeader]=newBoundaries;
```

```
WaterDriver::writeBoundary(videoFd,newBoundaries);
                        }else{

newBoundaries.river1_left=boundaries[screenHeader].river1_left;

newBoundaries.river1_right=boundaries[screenHeader].river1_right;

newBoundaries.river2_left=newBoundaries.river2_right=0;
                            // move down the boundary matrix
                            screenHeader=(screenHeader+1)%480;
                            boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                        }
                    }else{// ensure the middle of the two river is at least
4

if(boundaries[screenHeader].river1_right+5<boundaries[screenHeader].river2_
left &&
boundaries[screenHeader].river1_right-boundaries[screenHeader].river1_left<
=maximumWidth/2-2 &&
boundaries[screenHeader].river2_right-boundaries[screenHeader].river2_left<
=maximumWidth/2-2){

newBoundaries.river1_left=boundaries[screenHeader].river1_left-1;

newBoundaries.river1_right=boundaries[screenHeader].river1_right+1;

newBoundaries.river2_left=boundaries[screenHeader].river2_left-1;

newBoundaries.river2_right=boundaries[screenHeader].river2_right+1;
                            // move down the boundary matrix
                            screenHeader=(screenHeader+1)%480;
                            boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                        }else{

newBoundaries.river1_left=boundaries[screenHeader].river1_left;

newBoundaries.river1_right=boundaries[screenHeader].river1_right;
```

```
newBoundaries.river2_left=boundaries[screenHeader].river2_left;

newBoundaries.river2_right=boundaries[screenHeader].river2_right;
                        // move down the boundary matrix
                        screenHeader=(screenHeader+1)%480;
                        boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                    }
                }
                break;
            case DECREASE_WIDTH:
                if (boundaries[screenHeader].river2_left == 0) {
                    if (boundaries[screenHeader].river1_right -
boundaries[screenHeader].river1_left >= minimumWidth + 2) {
                        newBoundaries.river1_left =
boundaries[screenHeader].river1_left + 1;
                        newBoundaries.river1_right =
boundaries[screenHeader].river1_right - 1;
                        newBoundaries.river2_left =
newBoundaries.river2_right = 0;
                        // move down the boundary matrix
                        screenHeader=(screenHeader+1)%480;
                        boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                    }else{

newBoundaries.river1_left=boundaries[screenHeader].river1_left;

newBoundaries.river1_right=boundaries[screenHeader].river1_right;

newBoundaries.river2_left=newBoundaries.river2_right=0;
                        // move down the boundary matrix
                        screenHeader=(screenHeader+1)%480;
                        boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                    }
                } else {
                    if (boundaries[screenHeader].river1_right -
boundaries[screenHeader].river1_left >= minimumWidth + 2 &&
                        boundaries[screenHeader].river2_right -
```

```
boundaries[screenHeader].river2_left >= minimumWidth - 2) {
                        // ensure the middle of the two river is at
least 4
                        newBoundaries.river1_left =
boundaries[screenHeader].river1_left + 1;
                        newBoundaries.river1_right =
boundaries[screenHeader].river1_right - 1;
                        newBoundaries.river2_left =
boundaries[screenHeader].river2_left + 1;
                        newBoundaries.river2_right =
boundaries[screenHeader].river2_right - 1;
                        // move down the boundary matrix
                        screenHeader=(screenHeader+1)%480;
                        boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                    }else{

newBoundaries.river1_left=boundaries[screenHeader].river1_left;

newBoundaries.river1_right=boundaries[screenHeader].river1_right;

newBoundaries.river2_left=boundaries[screenHeader].river2_left;

newBoundaries.river2_right=boundaries[screenHeader].river2_right;
                        // move down the boundary matrix
                        screenHeader=(screenHeader+1)%480;
                        boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                    }
                }
                break;
            case DOUBLE_RIVER:
                    if(firstTimeDouble){

singleRiverWidth=boundaries[screenHeader].river1_right-boundaries[screenHea
der].river1_left;
                        firstTimeDouble= false;
                        //printf("single river width:
%d\n",singleRiverWidth);
                    }
```

```
if(boundaries[screenHeader].river2_left==0&&singleRiverWidth*2+20<=maximumW
idth){
                            // when the width is not enough to be divided
                            if
(boundaries[screenHeader].river1_right-boundaries[screenHeader].river1_left
<(singleRiverWidth)*2+20){

newBoundaries.river1_left=boundaries[screenHeader].river1_left-1;

newBoundaries.river1_right=boundaries[screenHeader].river1_right+1;

newBoundaries.river2_left=newBoundaries.river2_right=0;
                                // move down the boundary matrix
                                screenHeader=(screenHeader+1)%480;
                                boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                            }else{
                                    if(doubleRiverCounter>0){
                                        doubleRiverCounter--;

newBoundaries.river1_left=boundaries[screenHeader].river1_left;

newBoundaries.river1_right=boundaries[screenHeader].river1_right;

newBoundaries.river2_left=newBoundaries.river2_right=0;
                                            screenHeader=(screenHeader+1)%480;

boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                                    }else{

newBoundaries.river1_left=boundaries[screenHeader].river1_left;

newBoundaries.river2_right=boundaries[screenHeader].river1_right;

newBoundaries.river1_right=newBoundaries.river1_left+singleRiverWidth;

newBoundaries.river2_left=newBoundaries.river2_right-singleRiverWidth;
                                            screenHeader=(screenHeader+1)%480;

boundaries[screenHeader]=newBoundaries;
```

```
WaterDriver::writeBoundary(videoFd,newBoundaries);
                                    }
                            }


if(boundaries[screenHeader].river1_right-boundaries[screenHeader].river1_le
ft<(singleRiverWidth)*2+20&&

boundaries[screenHeader].river1_right-boundaries[screenHeader].river1_left+
2>=(singleRiverWidth)*2+20) {
                                // the first time to satisfy the width of
doubling the river
                                doubleRiverCounter=50;
                            }


                        }else
if(boundaries[screenHeader].river2_left==0&&(singleRiverWidth-20)/2>=minimu
mWidth&&singleRiverWidth+20<=maximumWidth){

newBoundaries.river1_left=boundaries[screenHeader].river1_left;

newBoundaries.river2_right=boundaries[screenHeader].river1_right;

newBoundaries.river1_right=newBoundaries.river1_left+(singleRiverWidth-20)/
2;

newBoundaries.river2_left=newBoundaries.river1_left+(singleRiverWidth-20)/2
+20;
                            screenHeader=(screenHeader+1)%480;
                            boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                        }
                        else{

newBoundaries.river1_left=boundaries[screenHeader].river1_left;

newBoundaries.river1_right=boundaries[screenHeader].river1_right;

newBoundaries.river2_left=boundaries[screenHeader].river2_left;
```

```
newBoundaries.river2_right=boundaries[screenHeader].river2_right;
                            // move down the boundary matrix
                            screenHeader=(screenHeader+1)%480;
                            boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                        }
                    break;
                case SINGLE_RIVER:
                    if(boundaries[screenHeader].river2_left!=0){ // now we
have two rivers
                            if
(boundaries[screenHeader].river2_left-boundaries[screenHeader].river1_right
>0){
                                // when there are still having 2 rivers

newBoundaries.river1_left=boundaries[screenHeader].river1_left;

newBoundaries.river1_right=boundaries[screenHeader].river1_right+1;

newBoundaries.river2_left=boundaries[screenHeader].river2_left-1;

newBoundaries.river2_right=boundaries[screenHeader].river2_right;
                                // move down the boundary matrix
                                screenHeader=(screenHeader+1)%480;
                                boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                            }else
if(boundaries[screenHeader].river2_left!=0){
                                // change it to single river

newBoundaries.river1_left=boundaries[screenHeader].river1_left;

newBoundaries.river1_right=boundaries[screenHeader].river2_right;

newBoundaries.river2_left=newBoundaries.river2_right=0;
                                screenHeader=(screenHeader+1)%480;
                                boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                            }else{
                                // single river
```

```
if(boundaries[screenHeader].river1_right=boundaries[screenHeader].river1_le
ft>=minimumWidth+2){

newBoundaries.river1_left=boundaries[screenHeader].river1_left+1;

newBoundaries.river1_right=boundaries[screenHeader].river2_right-1;

newBoundaries.river2_left=newBoundaries.river2_right=0;
                                screenHeader=(screenHeader+1)%480;
                                boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                            }else{

newBoundaries.river1_left=boundaries[screenHeader].river1_left;

newBoundaries.river1_right=boundaries[screenHeader].river1_right;

newBoundaries.river2_left=newBoundaries.river2_right=0;
                                // move down the boundary matrix
                                screenHeader=(screenHeader+1)%480;
                                boundaries[screenHeader]=newBoundaries;

WaterDriver::writeBoundary(videoFd,newBoundaries);
                            }

                        }

                }else{

newBoundaries.river1_left=boundaries[screenHeader].river1_left;

newBoundaries.river1_right=boundaries[screenHeader].river1_right;

newBoundaries.river2_left=newBoundaries.river2_right=0;
                    // move down the boundary matrix
                    screenHeader=(screenHeader+1)%480;
                    boundaries[screenHeader]=newBoundaries;
                    WaterDriver::writeBoundary(videoFd,newBoundaries);
                }
                break;
            }
```

```
}

GameScenario::GameScenario(int videoFd,short minimumWidth, short
maximumWidth, short frequency): minimumWidth(minimumWidth),
maximumWidth(maximumWidth),frequency(frequency){

}

double GameScenario::getFrequency(){
   return frequency;
}

void GameScenario::setChangeClock() {
   change=clock();
}

int GameScenario::getScreenHeader() {
   return screenHeader;
}
```

Airplane.h

```
//
// Created by Yongmao Luo on 4/1/22.
//

#ifndef WATER_RAID_AIRPLANE_H
#define WATER_RAID_AIRPLANE_H

#include "common_data_structure.h"
#include "bullet.h"
#include "enemy_plane.h"
#include "fuel_tank.h"
#include "battleship.h"
#include <vector>
#include <pthread.h>

typedef struct {
   struct timeval time;
   unsigned short type;
   unsigned short code;
   unsigned int value;
```

```
}InputEvent;

class Airplane{
private:
   char type; // what type of sprite it is
   Position pos; // the position of the plane
   Shape shape; // the shape of the sprite
   InputEvent xboxInput; // the input data from xbox
   bool buttonXOn,buttonBOn; // help to determine if the user keeps
pressing the two buttons
public:
   int scores,fuel;
   void fire(int xboxFd,int videoFd,vector<Bullet> &bulletList); // Fire a
bullet
   bool isCrashed(int videoFd,BoundaryInRow boundary); // if it crashes on
the boundary
   bool isCrashed(int videoFd,
                  std::vector<EnemyPlane> enemyPlaneList,
                  std::vector<Battleship> battleList); // if the plane
crashes on some enemy sprites
   void addScore(int videoFd,int score); // add scores to the plane
   void addFuel(int videoFd,std::vector<FuelTank> &fuelTankList,
std::vector<short> &spriteIndexList); // add fuel if the plane bumps into
the fuel tank
   int reduceFuel(int videoFd); // when time flies, the plane should
consume more fuels
   Position getPos(); // get the position of the plane
   void setPos(Position); // set the position of the plane
   void receiveFromXbox(int xboxFd); // receive control signals from the
Xbox
   void calPos(int videoFd); // calculate the new position based on the
received data
   bool startGame(); // If we press button A, the game starts
   Airplane(char type, char fuel, Position pos, Shape shape, char scores);

   char getType(){
       return this->type;
   }
   char getFuel(){
       return this->fuel;
   }
   char getScores(){
       return this->scores;
```

```
    }
    Airplane(){
        //mutexPos= PTHREAD_MUTEX_INITIALIZER;
        type=0;
        fuel=100;
        pos.x=320;
        pos.y=480;
        scores=0;
        shape.length=3;
        shape.width=5;
    }
};

#endif
```

Airplane.cpp

```
//
// Created by Yongmao Luo on 4/6/22.
//

#include "airplane.h"

#include "driver.h"
#include <stdio.h>


#include <fcntl.h>
#include <unistd.h>
#include <string.h>

#define XBOX_BUTTON_B 305
#define XBOX_BUTTON_X 307
#define XBOX_BUTTON_A 304
#define XBOX_BUTTON_Y 308

#define SPRITE_BULLET 4
#define SPRITE_EXPLODE 5
#define SPRITE_FUEL 3
#define PLANE_LEFT 7
#define PLANE_RIGHT 8
```

```
#define MAXFUEL 75
#define MINFUEL 5

#define SHOOT_AUDIO 0
#define HIT_AUDIO 1
#define EXPLODE_AUDIO 2

bool Airplane::isCrashed(int videoFd, BoundaryInRow boundary) {
   // collide the boundary
   if(boundary.river2_left==0){
       if((boundary.river1_left>=pos.x-shape.width)||
            (boundary.river1_right<=pos.x+shape.width)){
           // plane disappear
           Position tempPos;
           tempPos.y=0;
           tempPos.x=0;
           WaterDriver::writePosition(videoFd,tempPos,type,0);
           // create explosion effect
           WaterDriver::writePosition(videoFd,pos,SPRITE_EXPLODE,0);
           pos=tempPos;
           return true;
       }
   }else{
       if((boundary.river2_left>pos.x&&
          boundary.river1_left>=pos.x-shape.width)||
            (boundary.river2_left>pos.x&&
          boundary.river1_right<=pos.x+shape.width)||
            (boundary.river2_left<pos.x&&
            boundary.river2_left>=pos.x-shape.width)||
            (boundary.river2_left<pos.x&&
            boundary.river2_right<=pos.x+shape.width)){
           printf("river1 left:%d",boundary.river1_left);
            printf("river1 right:%d",boundary.river1_right);
            printf("river2 left:%d",boundary.river2_left);
            printf("river2 right:%d",boundary.river2_right);
           // crashed when there are 2 rivers
           // plane disappear
           Position tempPos;
           tempPos.y=0;
           tempPos.x=0;
           WaterDriver::writePosition(videoFd,tempPos,type,0);
           // create explosion effect
```

```
            WaterDriver::writePosition(videoFd,pos,SPRITE_EXPLODE,0);
            pos=tempPos;
            return true;
        }
    }
    return false;
}

bool Airplane::isCrashed(int videoFd,
                                std::vector<EnemyPlane> enemyPlaneList,
                                std::vector<Battleship> battleList){

    // collide the enemy planes
    for(int i=0;i<enemyPlaneList.size();i++){

if(pos.y-shape.length<=enemyPlaneList[i].getPos().y+enemyPlaneList[i].getSh
ape().length&&

pos.y+shape.length>=enemyPlaneList[i].getPos().y-enemyPlaneList[i].getShape
().length&&

!(pos.x+shape.width<enemyPlaneList[i].getPos().x-enemyPlaneList[i].getShape
().width)&&

!(pos.x-shape.width>enemyPlaneList[i].getPos().x+enemyPlaneList[i].getShape
().width)){
            Position tempPos;
            tempPos.y=0;
            tempPos.x=0;
            WaterDriver::writePosition(videoFd,tempPos,type,0);
            // create explosion effect
            WaterDriver::writePosition(videoFd,pos,SPRITE_EXPLODE,0);
            pos=tempPos;
            return true;
        }
    }
    //collide the battleships
    for(int i=0;i<battleList.size();i++){

if(pos.y-shape.length<=battleList[i].getPos().y+battleList[i].getShape().le
ngth&&

pos.y+shape.length>=battleList[i].getPos().y-battleList[i].getShape().lengt
```

```
h&&

!(pos.x+shape.width<battleList[i].getPos().x-battleList[i].getShape().width
)&&

!(pos.x-shape.width>battleList[i].getPos().x+battleList[i].getShape().width
)){
            Position tempPos;
            tempPos.y=0;
            tempPos.x=0;
            WaterDriver::writePosition(videoFd,tempPos,type,0);
            // create explosion effect
            WaterDriver::writePosition(videoFd,pos,SPRITE_EXPLODE,0);
            pos=tempPos;
            return true;
        }
    }

    return false;
}

void Airplane::addFuel(int videoFd,std::vector<FuelTank> &fuelTankList,
std::vector<short> &spriteIndexList){
    for(int i=0;i<fuelTankList.size();i++){

if(pos.y-shape.length<=fuelTankList[i].getPos().y+fuelTankList[i].getShape(
).length&&

pos.y+shape.length>=fuelTankList[i].getPos().y-fuelTankList[i].getShape().l
ength&&

!(pos.x+shape.width<fuelTankList[i].getPos().x-fuelTankList[i].getShape().w
idth)&&

!(pos.x-shape.width>fuelTankList[i].getPos().x+fuelTankList[i].getShape().w
idth)){
            // collide with fuelTank
            if(fuel+10<=MAXFUEL)
                fuel+=10;
            else
                fuel=MAXFUEL;
            WaterDriver::writeFuel(videoFd,fuel);
            Position tempPos;tempPos.y=0;
```

```
            // remove the fuelTank

WaterDriver::writePosition(videoFd,tempPos,SPRITE_FUEL,fuelTankList[i].inde
x);
            spriteIndexList.push_back(fuelTankList[i].getIndex());
            fuelTankList.erase(fuelTankList.begin()+i);
            break;
        }
    }
    WaterDriver::writeFuel(videoFd,this->fuel);
}

int Airplane::reduceFuel(int videoFd) {
    if(fuel>MINFUEL){
        fuel-=1;
        WaterDriver::writeFuel(videoFd,fuel);
        return 0;
    }else{
        return -1;
    }

}

void Airplane::fire(int xboxFd,int videoFd,vector<Bullet> &bulletList){

    if (xboxInput.code == XBOX_BUTTON_Y && xboxInput.value==1) {
        // press the button to emit bullet
        int numOfBullets=bulletList.size();
        if(numOfBullets==3)
            return;
        WaterDriver::playAudio(videoFd,SHOOT_AUDIO);
        char temp[4]; // 1, 2, 3
        memset(temp,0,sizeof(temp));
        if(numOfBullets>0){
            for(vector<Bullet>::iterator
it=bulletList.begin();it!=bulletList.end();it++){
                temp[it->index]=1;
            }
        }


        for(int i=1;i<=3;i++){
            if(temp[i]==0){
```

```
                Shape sp;sp.width=1;sp.length=5;
                Position tempPos; tempPos.x=pos.x;
tempPos.y=pos.y-sp.length*2;
                Bullet bullet=Bullet(SPRITE_BULLET,sp,tempPos);
                bullet.index=i;
                bulletList.push_back(bullet);
                WaterDriver::writePosition(videoFd,tempPos,SPRITE_BULLET,i);
                break;
            }
        }
    }

}

Airplane::Airplane(char type, char fuel, Position pos, Shape shape, char
scores):
type(type), fuel(fuel), pos(pos),shape(shape),scores(scores){
    //mutexPos= PTHREAD_MUTEX_INITIALIZER;
    buttonBOn=buttonXOn= false;
}

Position Airplane::getPos() {
    //pthread_mutex_lock(&mutexPos);
    Position result=pos;
    //pthread_mutex_unlock(&mutexPos);
    return result;
}

void Airplane::setPos(Position change){
    //pthread_mutex_lock(&mutexPos);
    pos=change;
    //pthread_mutex_unlock(&mutexPos);
    return;
}



void Airplane::receiveFromXbox(int xboxFd) {

    int flags= fcntl(xboxFd,F_GETFL,0);
    fcntl(xboxFd,F_SETFL,flags|O_NONBLOCK);
    read(xboxFd, &xboxInput, 24);
```

```
}

void Airplane::calPos(int videoFd) {
    if (xboxInput.code == XBOX_BUTTON_X && xboxInput.value==1) {
        buttonXOn=true;
    } else if(xboxInput.code == XBOX_BUTTON_X && xboxInput.value==0){
        buttonXOn=false;
    }else if (xboxInput.code == XBOX_BUTTON_B && xboxInput.value==1) {
        buttonBOn=true;
    }else if(xboxInput.code == XBOX_BUTTON_B && xboxInput.value==0){
        buttonBOn=false;
    }

    if(buttonXOn){
        Position tempPos = getPos();
        tempPos.x -= 2;
        setPos(tempPos);
        WaterDriver::writePosition(videoFd,pos,PLANE_LEFT,0);
    }else if(buttonBOn){
        Position tempPos = getPos();
        tempPos.x += 2;
        setPos(tempPos);
        WaterDriver::writePosition(videoFd,pos,PLANE_RIGHT,0);
    }else{
        WaterDriver::writePosition(videoFd,pos,type,0);
    }
}

void Airplane::addScore(int videoFd,int score) {
    int tempScores=this->scores+score;
    if(tempScores>999){
        this->scores=999;
    }else{
        this->scores=tempScores;
    }
    WaterDriver::writeScore(videoFd, this->scores);
}

bool Airplane::startGame() {
    if(xboxInput.code==XBOX_BUTTON_A&&xboxInput.value==1){
        return true;
    }
    return false;
```

```
}
```

Sprites.h

```
//
// Created by Frank on 4/1/22.
//

#ifndef WATER_RAID_SPRITE_H
#define WATER_RAID_SPRITE_H

#define SPRITE_PLANE 0
#define SPRITE_HELI 1
#define SPRITE_BATTLE 2
#define SPRITE_FUEL 3
#define SPRITE_BULLET 4

#define SPRITE_X 14
#define SPRITE_Y 14

#include "common_data_structure.h"
#include "bullet.h"
#include "game_scenario.h"
#include "driver.h"
#include<vector>

using namespace std;

class Sprite
{
protected:
   char type;
   char hitPoint;
   Shape sp;
   bool left = true;
   bool canMove;

public:
   short index;
```

```
    bool isDestroy;

    Position pos;

    Sprite(char type, char hitPoint, const Shape &sp, bool isDestroy,
short index, bool canMove) : type(type),

hitPoint(hitPoint),

sp(sp),

isDestroy(isDestroy),

index(index),canMove(canMove){}



    bool getIsDestroy() const {
        return isDestroy;
    }

    void setIsDestroy(bool isDestroy) {
        Sprite::isDestroy = isDestroy;
    }

    const Position &getPos() const {
        return pos;
    }

    Shape getShape();

    void setPos(const Position &pos) {
        Sprite::pos = pos;
    }

    void generate(BoundaryInRow boundary, short y);

    short getIndex() const {
        return index;
```

```
    }

    //start from 4 to 8

    void disappear();

    void move(BoundaryInRow boundary, short minimumWidth);
};

#endif //WATER_RAID_SPRITE_H
```

Sprites.cpp

```
Sp//
// Created by Frank on 4/1/22.
//
#include "sprite.h"
#include "bullet.h"
#include<ctime>
#include<iostream>
#include<vector>

#define SPRITE_PLANE 0
#define SPRITE_HELI 1
#define SPRITE_BATTLE 2
#define SPRITE_FUEL 3
#define SPRITE_BALLOON 6

using namespace std;

void Sprite::generate(BoundaryInRow boundary, short y) {

    srand(time(0));
    if (this->type == SPRITE_FUEL || this->type == SPRITE_BATTLE) {
        if (boundary.river2_left == 0) {
            short randomNumber = rand() % (boundary.river1_right -
boundary.river1_left);
            short new_pos_x = randomNumber + boundary.river1_left;
            this->pos.x = new_pos_x;
```

```
        } else {
            if (rand() % 2) {
                short randomNumber = rand() % (boundary.river1_right -
boundary.river1_left);
                short new_pos_x = randomNumber + boundary.river1_left;
                this->pos.x = new_pos_x;
            } else {
                short randomNumber = rand() % (boundary.river2_right -
boundary.river2_left);
                short new_pos_x = randomNumber + boundary.river2_left;
                this->pos.x = new_pos_x;
            }
        }
    } else if(this->type==SPRITE_HELI) {
        short new_pos_x = rand() % 640;
        this->pos.x = new_pos_x;
    }

    this->pos.y = (y << 1) + 1;

}

void Sprite::move(BoundaryInRow boundary, short minimumWidth) {

    if(this->canMove== false){
        return;
    }

    srand(time(0));
    short forward = this->pos.x + this->sp.width + minimumWidth;
    short backward = this->pos.x - this->sp.width - minimumWidth;

    if (this->type == SPRITE_FUEL || this->type == SPRITE_BATTLE) {
        //Attention: minimumWidth is the minimum width of every branch
of the river

        if (boundary.river2_left == 0) {
            if(backward<=boundary.river1_left&&left){
                left=false;
```

```
            }else if(forward>=boundary.river1_right&&!left){
                left=true;
            }

            if(left){
                this->pos.x=backward;
            }else{
                this->pos.x=forward;
            }
        } else {
            if (this->getPos().x < boundary.river1_right) {
                // on left river
                if(backward<=boundary.river1_left&&left){
                    left=false;
                }else if(forward>=boundary.river1_right&&!left){
                    left=true;
                }

                if(left){
                    this->pos.x=backward;
                }else{
                    this->pos.x=forward;
                }
            } else {
                // on right river
                if(backward<=boundary.river2_left&&left){
                    left=false;
                }else if(forward>=boundary.river2_right&&!left){
                    left=true;
                }

                if(left){
                    this->pos.x=backward;
                }else{
                    this->pos.x=forward;
                }
            }
        }
    } else if(this->type==SPRITE_HELI) {
```

```
        // helicopter can fly through the whole screen
        if(forward>=640-sp.width&&!left){
            left= true;
        }else if(backward<=sp.width){
            left= false;
        }

        if(left){
            this->pos.x=backward;
        }else{
            this->pos.x=forward;
        }
    }
}

void Sprite::disappear() {
    this->pos.y = 0;
}

Shape Sprite::getShape() {
    return sp;
}
```

Fuel_tank.h

```
//
// Created by Frank on 4/1/22.
//

#ifndef WATER_RAID_FUEL_TANK_H
#define WATER_RAID_FUEL_TANK_H

#include "sprite.h"
#include <vector>

class FuelTank : public Sprite
{
```

```
private:
   char fuelVolume;

public:
   void checkIfHit(vector<Bullet> &bullets,int videoFd);

   FuelTank(char type, char hitPoint, const Shape &sp, bool
isDestroy, char fuelVolume, short index, bool canMove);

};

#endif //WATER_RAID_FUEL_TANK_H
```

```
//
// Created by Frank on 4/1/22.
//

#ifndef WATER_RAID_FUEL_TANK_H
#define WATER_RAID_FUEL_TANK_H

#include "sprite.h"
#include <vector>

class FuelTank : public Sprite
{

private:
   char fuelVolume;

public:
   void checkIfHit(vector<Bullet> &bullets,int videoFd);

   FuelTank(char type, char hitPoint, const Shape &sp, bool isDestroy, char
fuelVolume, short index, bool canMove);

};

#endif //WATER_RAID_FUEL_TANK_H
```

Fuel_tank.cpp

```
//
// Created by Frank on 4/4/22.
//
#include "fuel_tank.h"
#include "driver.h"

#define HIT_AUDIO 1

void FuelTank::checkIfHit(vector<Bullet> &bullets,int videoFd) {
   for (int i = 0; i < bullets.size(); i++) {
       if (bullets[i].getPosition().x >= (this->getPos().x -
this->sp.width) &&
           bullets[i].getPosition().x <= (this->getPos().x +
this->sp.width) && (bullets[i].getPosition().y -
bullets[i].getSp().length)<=

(this->getPos().y +

this->sp.length)){
           this->hitPoint--;
           WaterDriver::playAudio(videoFd,HIT_AUDIO);
           bullets[i].setCrash();
           if(this->hitPoint == 0){
               this->setIsDestroy(true);
           }
       }
   }
}

void FuelTank::movement(int videoFd, vector<FuelTank> &fuelTankList,
vector<short> &spriteIndexList,
                        GameScenario gameScenario) {
   for (int i = 0; i < fuelTankList.size(); i++)
   {
       fuelTankList[i].pos.y += 2;
       if(fuelTankList[i].pos.y ==  (480 << 1) + 1){
           fuelTankList[i].disappear();
           WaterDriver::writePosition(videoFd,
fuelTankList[i].getPos(), SPRITE_FUEL,
```

```
                                        fuelTankList[i].getIndex());
            spriteIndexList.push_back(fuelTankList[i].getIndex());
            fuelTankList.erase(fuelTankList.begin()+i);
        } else{

fuelTankList[i].move(gameScenario.boundaries[(gameScenario.getScreenH
eader() - fuelTankList[i].getPos().y + 480 + SPRITE_Y) % 480], 5);
            WaterDriver::writePosition(videoFd,
fuelTankList[i].getPos(), SPRITE_FUEL,
                                        fuelTankList[i].getIndex());
        }
    }
}

FuelTank::FuelTank(char type, char hitPoint, const Shape &sp, bool
isDestroy,
                   char fuelVolume, short index,bool canMove) :
Sprite(type, hitPoint, sp, isDestroy, index,canMove),
fuelVolume(fuelVolume) {}
```

Enemy_plane.h

```
//
// Created by Frank on 4/1/22.
//

#ifndef WATER_RAID_ENEMY_PLANE_H
#define WATER_RAID_ENEMY_PLANE_H

#include "sprite.h"
#include "bullet.h"
#include <vector>

class EnemyPlane: public Sprite
{
private:
    char score;

public:
```

```
   void checkIfHit(vector<Bullet> &bullets,int videoFd, int
&planeScore);

   EnemyPlane(char type, char hitPoint, const Shape &sp, bool
isDestroy, char score, short index,bool canMove);

};

#endif //WATER_RAID_ENEMY_PLANE_H
```

Enemy_plane.cpp

```
//
// Created by Frank on 4/4/22.
//

#include "enemy_plane.h"
#include "bullet.h"
#include "driver.h"

#define HIT_AUDIO 1


void EnemyPlane::checkIfHit(vector<Bullet> &bullets,int videoFd, int
&planeScore) {
   for (int i = 0; i < bullets.size(); i++) {
       if (bullets[i].getPosition().x >= (this->getPos().x -
this->sp.width)   &&
           bullets[i].getPosition().x <= (this->getPos().x +
this->sp.width) && (bullets[i].getPosition().y -
bullets[i].getSp().length) <=

(this->getPos().y +

this->sp.length)){
           this->hitPoint--;
           WaterDriver::playAudio(videoFd,HIT_AUDIO);
           bullets[i].setCrash();
```

```
            if(this->hitPoint == 0){
                this->setIsDestroy(true);
                planeScore+= score;
                WaterDriver::writeScore(videoFd,planeScore);
            }
        }
    }
}


EnemyPlane::EnemyPlane(char type, char hitPoint, const Shape &sp,
bool isDestroy,
                       char score, short index,bool canMove) :
Sprite(type, hitPoint, sp, isDestroy, index,canMove), score(score) {}
```

Bullet.h

```
//
// Created by Frank on 4/1/22.
//

#ifndef WATER_RAID_BULLET_H
#define WATER_RAID_BULLET_H

#include "common_data_structure.h"
#include <vector>

#define SPRITE_PLANE 0
#define SPRITE_HELI 1
#define SPRITE_BATTLE 2
#define SPRITE_FUEL 3
#define SPRITE_BULLET 4
#define SPRITE_BALLOON 6

class Bullet
{
private:
    char type;
    Shape sp;
```

```
    bool isCrashed;



public:
    Position pos;
    short index;

    short getIndex() const;

    const Shape &getSp() const {
        return sp;
    }

    void setSp(const Shape &sp) {
        Bullet::sp = sp;
    }

    Shape getShape(){
        return sp;
    }

    const Position &getPosition() const {
        return pos;
    }

    void setPosition(const Position &pos) {
        Bullet::pos = pos;
    }

    void setCrash(){
        isCrashed = true;
        this->pos.y = 0;
    }

    bool getIsCrashed() const;

    Bullet(char type, const Shape &sp, const Position &pos);
```

```
   static void fly(int videoFd,std::vector<Bullet> &);

};

#endif //WATER_RAID_BULLET_H
```

Bullet.cpp

```
//
// Created by Frank on 4/1/22.
//
#include "bullet.h"
#include "driver.h"
using namespace std;

void Bullet::fly(int videoFd,vector<Bullet> &bulletList) {
   for (int i = 0; i < bulletList.size(); i++)
   {
       Position tempPos=bulletList[i].getPosition();tempPos.y-=8;
       bulletList[i].setPosition(tempPos);

WaterDriver::writePosition(videoFd,bulletList[i].getPosition(),SPRITE
_BULLET,
                                  bulletList[i].getIndex());
       if (bulletList[i].pos.y <= (0 << 1) + 1){
           bulletList[i].setCrash();
           WaterDriver::writePosition(videoFd,
bulletList[i].getPosition(), SPRITE_BULLET,
                                      bulletList[i].getIndex());
           bulletList.erase(bulletList.begin()+i);
       }
   }

}

Bullet::Bullet(char type, const Shape &sp, const Position &pos) :
type(type), sp(sp),

pos(pos) {}
```

```
bool Bullet::getIsCrashed() const {
   return isCrashed;
}

short Bullet::getIndex() const {
   return index;
}
```

Battleship.h

```
//
// Created by Frank on 4/1/22.
//
#include "bullet.h"
#include "driver.h"
using namespace std;

void Bullet::fly(int videoFd,vector<Bullet> &bulletList) {
   for (int i = 0; i < bulletList.size(); i++)
   {
       Position tempPos=bulletList[i].getPosition();tempPos.y-=8;
       bulletList[i].setPosition(tempPos);

WaterDriver::writePosition(videoFd,bulletList[i].getPosition(),SPRITE
_BULLET,
                                  bulletList[i].getIndex());
       if (bulletList[i].pos.y <= (0 << 1) + 1){
           bulletList[i].setCrash();
           WaterDriver::writePosition(videoFd,
bulletList[i].getPosition(), SPRITE_BULLET,
                                      bulletList[i].getIndex());
           bulletList.erase(bulletList.begin()+i);
       }
   }

}

Bullet::Bullet(char type, const Shape &sp, const Position &pos) :
```

```
type(type), sp(sp),

pos(pos) {}

bool Bullet::getIsCrashed() const {
   return isCrashed;
}

short Bullet::getIndex() const {
   return index;
}
```

Battleship.cpp

```
//
// Created by Frank on 4/4/22.
//

#include "battleship.h"
#include "driver.h"

#define HIT_AUDIO 1

void Battleship::checkIfHit(vector<Bullet> &bullets,int videoFd, int
&planeScore) {
   for (int i = 0; i < bullets.size(); i++) {
       if (bullets[i].getPosition().x >= (this->getPos().x -
this->sp.width) &&
           bullets[i].getPosition().x <= (this->getPos().x +
this->sp.width) && (bullets[i].getPosition().y -
bullets[i].getSp().length) <=

(this->getPos().y +

this->sp.length)){
           this->hitPoint--;
           WaterDriver::playAudio(videoFd,HIT_AUDIO);
           bullets[i].setCrash();
           if(this->hitPoint == 0){
```

```
                this->setIsDestroy(true);
                planeScore+=score;
                WaterDriver::writeScore(videoFd,planeScore);
            }
        }
    }
}


Battleship::Battleship(char type, char hitPoint, const Shape &sp,
bool isDestroy,
                       char score, short index,bool canMove) :
Sprite(type, hitPoint, sp, isDestroy, index,canMove), score(score) {}
```