Water Raid

Tristan Saidi, Yongmao Luo, Jakob Stiens and Zhaomeng Wang

Overview

- Based off of the 1980’s Atari game River Raid released by Activision

- The goal is to survive as long as possible without getting shot, crashing, flying off the river, or running out of fuel.

- Score is given for staying alive, and shooting down enemy vehicles

Gameplay Overview

- Just like the original game, the player can only move left and right

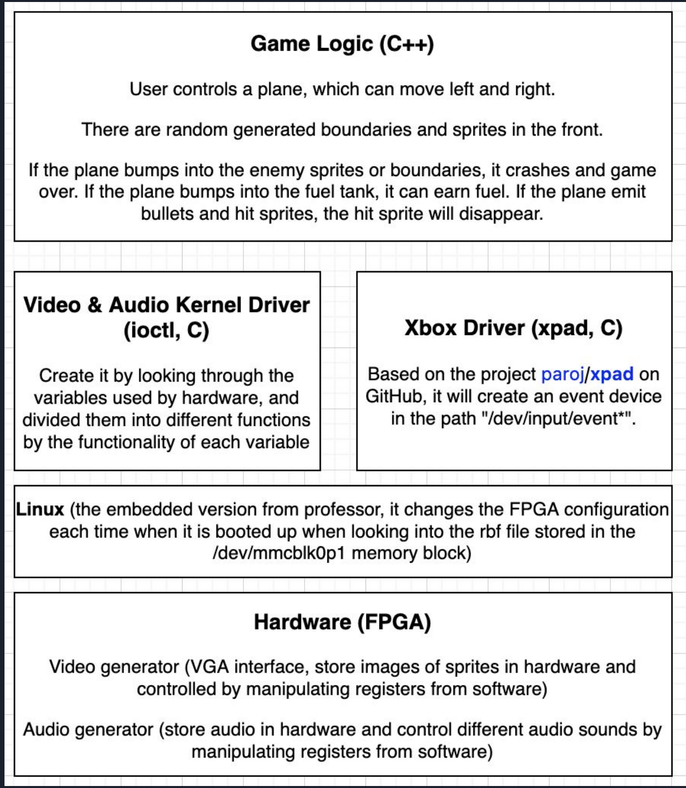
- Forward movement is simulated with the scrolling background and sprites

- We have chosen to have an xbox controller as the input with the following controls
    - “X” - left
    - “B” - right
    - “Y” - shoot
    - “A” - start

Our Game

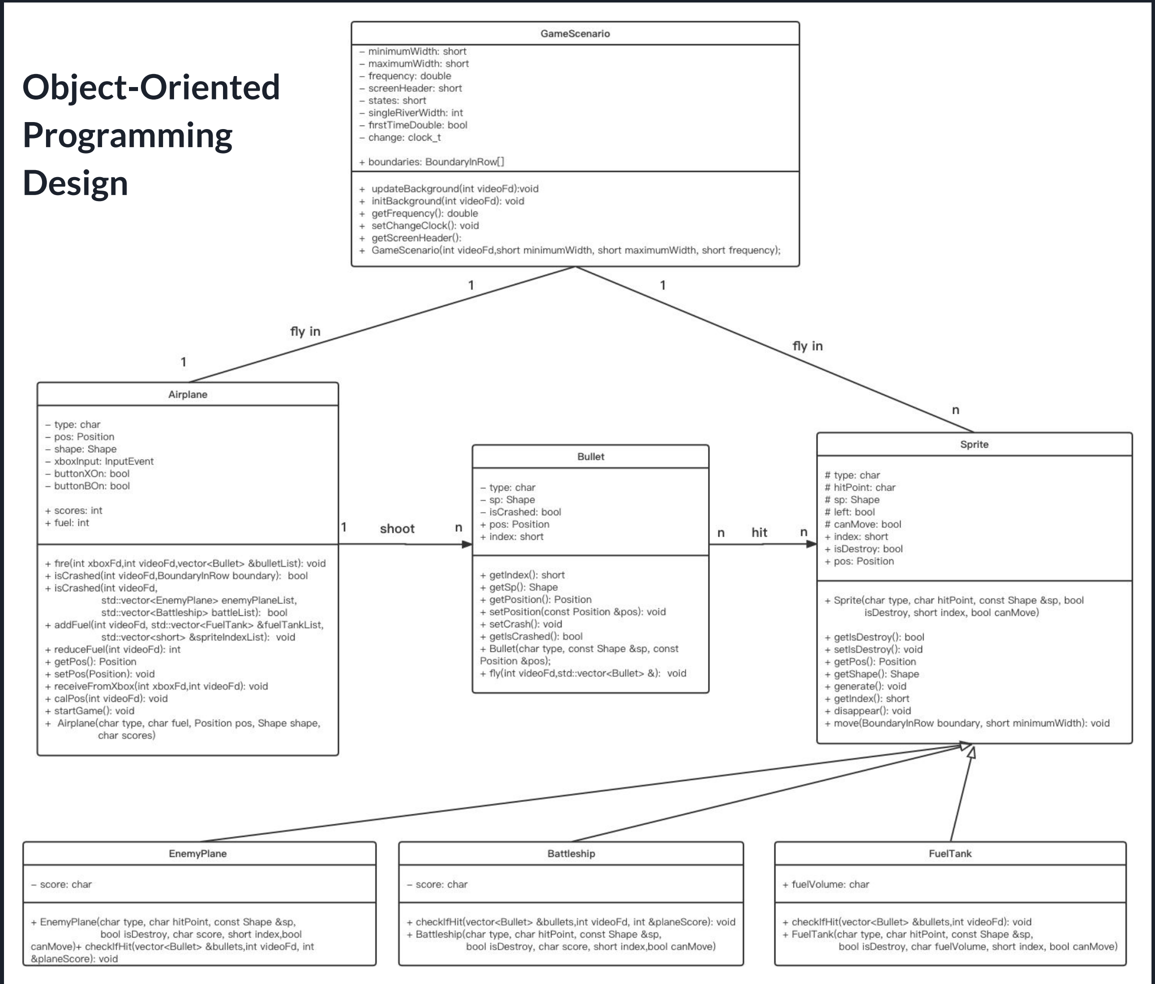
Original Game

Overview of the project



Game Logic

1. Main.cpp
2. Airplane
3. Game Scenario
4. Drivers
5. Sprites
    a. Fuel Tank
    b. Enemy Plane (helicopter, ballon)
    c. Battleship
6. Bullets



Common Data Structures

```
typedef struct{
   short x,y; // for y, we should put the coordinate at bit [9:1]
   //y[0] is the shift bit, y[0]=1 means disappear
}Position;

typedef struct{
   char width, length; // width is related to x coordinates
   // length is related to y coordinates
}Shape;

typedef struct{
   short river1_left,river1_right,river2_left, river2_right;
}BoundaryInRow;
```

Main.cpp

containing two nesting while loop with counter to execute logics in static frequency (60Hz)

Each iteration:

- Scroll down the background (randomly generating new boundaries for the background)
- Reduce and add fuels accordingly
- Examine if the plane crashes
- If fire, generate bullets
- Move all sprites except for the plane
- If hit, minus the HP for each hit sprite
- If HP has been consumed, delete the sprite

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;
```

Airplane

the class of Airplane. It has variable position, scores, fuel, etc. and functions to control the movement of the plane while updating fuel and scores.

```
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 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
```

Game Scenario

the class of Game Scenario. It has a one-dimensional boundaries array with length 480 to represent the background information. Each element of the array has four sub-elements, indicating the four boundaries of each row.

```
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 */
```

Game Scenario

Most important functionalities

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

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
```

Sprite

```
protected:
    char type;
    char hitPoint;
    Shape sp;
    bool left = true;
    bool canMove;

public:
    short index;
    bool isDestroy;

    Position pos;


    void generate(BoundaryInRow boundary, short y);

    void disappear();

    //Make the sprites randomly moved within a certain range
    void move(BoundaryInRow boundary, short minimumWidth);
```

EnemyPlane, Battleship and FuelTank

```
//EnemyPlane
private:
    char score;

public:

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

//Battleship
private:
    char score;

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

//FuelTank
private:
    char fuelVolume;

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

Bullets

```
private:
    char type;
    Shape sp;
    bool isCrashed;


public:
    Position pos;
    short index;

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

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

Driver in GameLogic

```
// 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
```

Linux Kernel Driver – Video & Audio Driver

According to the functionality of each variable in hardware

Write different functions, each can realize part of functionality to the whole project

Reduce amount of data transferring from software to hardware compared to single function implementation

```
#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 *)
```

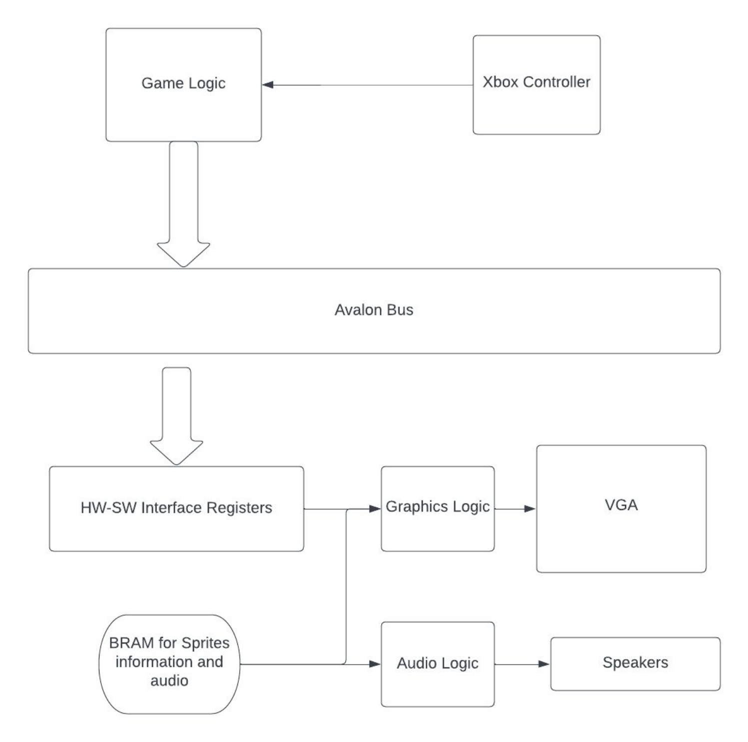
Linux Kernel Driver – Xbox Controller

The **paroj/xpad project**

```
root@de1-soc:~/Water-Raid# ls /dev/input
by-id  by-path  event0
```

```c
        if (!(xpad->mapping & MAP_STICKS_TO_NULL)) {
            /* left stick */
//          input_report_abs(dev, ABS_X,
//                  (__s16) le16_to_cpup((__le16 *)(data + 10)));
//          input_report_abs(dev, ABS_Y,
//                  ~(__s16) le16_to_cpup((__le16 *)(data + 12)));
//
//          /* right stick */
//          input_report_abs(dev, ABS_RX,
//                  (__s16) le16_to_cpup((__le16 *)(data + 14)));
//          input_report_abs(dev, ABS_RY,
//                  ~(__s16) le16_to_cpup((__le16 *)(data + 16)));
        }
```

Hardware-Software Interface System Design



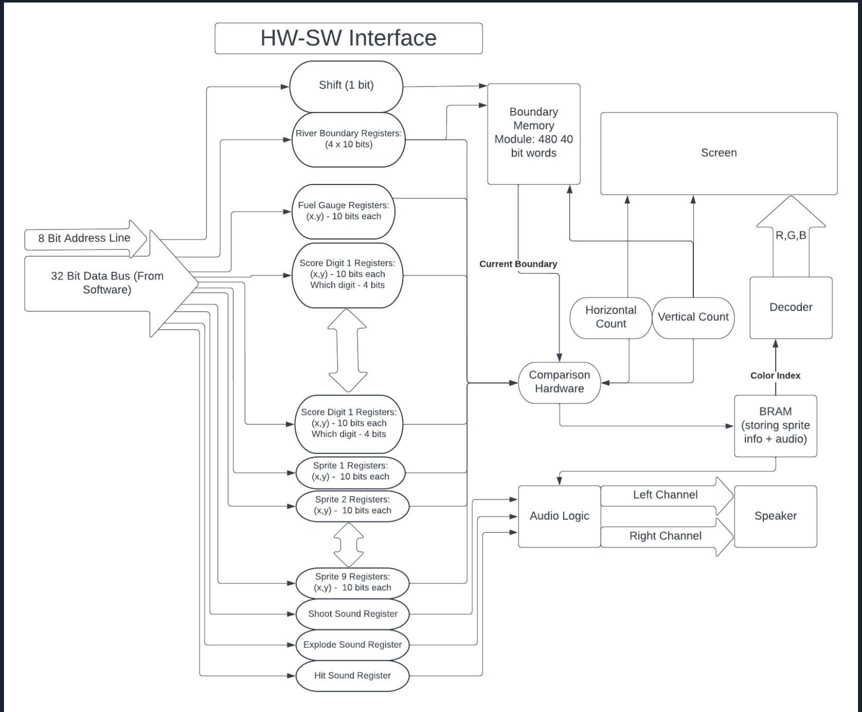
The Hardware and Software interface via the avalon bus. A series of registers controlling information about graphics and audio can be set from software through the avalon bus. Hardware then pulls from these registers and memory for the graphics logic and audio logic.

VGA signals are asserted based on the values read from the interface registers and based on sprite data stored in the ROM modules. A similar phenomenon is true for the audio, where the audio signal is read from ROM and pushed to the audio CODEC.

Hardware - Overview

The hardware is set up such that software can control the location and "image" of all sprites on screen via writes through the avalon bus. Essentially software dictates where the sprite is, and which ROM that specific sprite should pull from (dictating which image is displayed).

Background generation is done on the software side - the software generates four "boundary" values, each corresponding to the right/left side of a branch of the river. If the last two boundary values are zero, this indicates that there is only one river branch. To accomplish the background shifting, the hardware has a single "shift" signal; upon being toggled, the hardware shifts the entire screen down one pixel, and loads a new set of boundaries in from software. The details of how this is accomplished will be explained on subsequent slides.



Hardware - Sprite Generation

- most sprites are 32 x 32 pixels (except for fuel and scoreboard)

- generated from color palette value arrays

    - 0 means invisible background

- instantiated as 1 Port ROM modules generated from .mif files.

- have a total of 15 sprites on screen at once (including score and fuel gauge)

```python
def Plane(filename, debug):
    arrayOfValues = []
    filename = "PlaneSprite.mif"
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4])
    arrayOfValues.append([ 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4])
    arrayOfValues.append([ 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4])
    arrayOfValues.append([ 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4])
    arrayOfValues.append([ 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4])
    arrayOfValues.append([ 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4])
    arrayOfValues.append([ 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4])
    arrayOfValues.append([ 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4])
    arrayOfValues.append([ 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 4, 4, 4, 4, 4, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 4, 4, 4, 4, 4, 0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0])
    arrayOfValues.append([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    if (debug):
        print(arrayOfValues)
    return arrayOfValues, filename
```

Hardware - Sprite Display

- whenever a row is being displayed, hardware checks if a sprite overlaps

```
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));
```

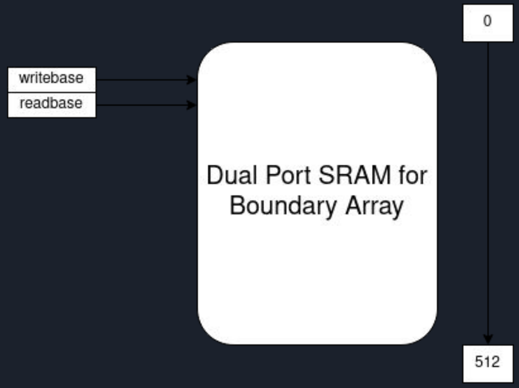
- If a sprite overlaps, the sprite color is chosen as the pixel color instead of the background

- Palette pixel color for sprites is chosen through a switch case

```
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
```

Hardware - Background

The hardware to handle the background shift required some thought. We settled on having a one bit shift signal that, whenever toggled, indicates to hardware that the screen should be shifted down. The background boundaries for all rows that are currently visible on the screen are stored in a two port SRAM unit (4 boundaries x 10 bits per boundary location = 40 bits per word). The hardware uses one port to read values from SRAM as it constantly cycles through and re-updates the screen. The other port is used to overwrite memory values with the most recent boundaries set by software. Everytime the shift signal is toggled, the base address for both the read and write are incremented - this enables the shift behavior that is seen. It is worth noting that the two port SRAM has 512 words despite the vertical depth of the screen only being 480 - this made it easier to circularly update the memory as the wraparound for a 9 bit counter replaces the need for modulo circuitry.



Hardware - Audio Overview

- Audio has 3 sound effects - shoot, hit, explode

- Each audio file has address 0 set to 0
    - Can be used to turn off the sound

- Have 3 registers that software can access

```
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
```

Hardware - Audio Main Loop

- When a register is set high, the address for that sound begins incrementing

- When the maximum address is reached, the incrementing stops

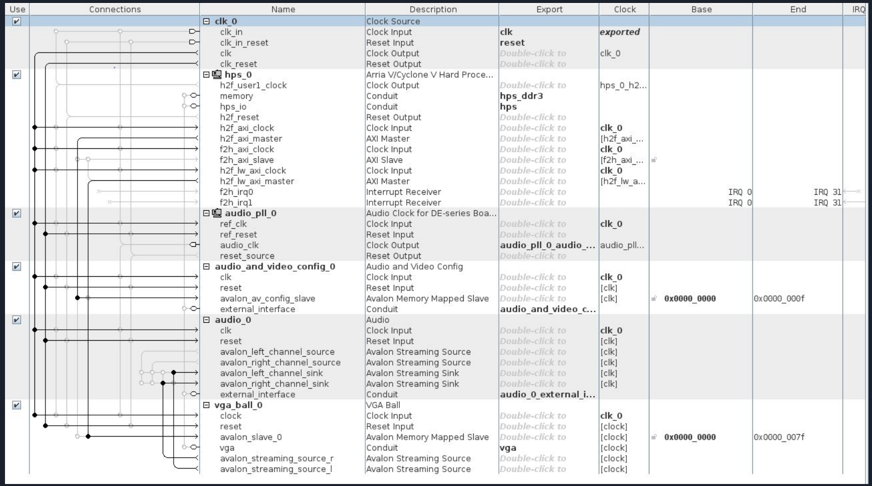
- Each audio file has address 0 set to 0
    - Can be used to turn off the sound

```
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
```

Hardware - Audio Connections

- Connect to the audio codec with the audio_0 and audio_and_video_config_0 modules

- Connect avalon streaming sources on the top level to avalon sinks connecting to audio codec



Hardware - Total Resources

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

	Shoot	Hit	Explosion
memory(bit)	23 Kb	16 Kb	82 Kb

Total Memory Usage: 368.3 Kb

Challenges and Lessons Learned

Challenges:

- Figuring out qsys configuration for audio
- Getting initial sprite graphics to work (ROM instantiation and reading out) without the ability to really look at waveforms
- Screen shift and two port memory
- Figure out the logic between different objects

Lessons Learned:

- Try to have hardware finished as early as possible
- Do correct system design for connecting the hardware and software

Demo!