CSEE 4840 Spring 2023

Project Final Report

Rhythm Game

Lu Zhang (lz2879), Qiao Zhang (qz2486), Jinke Zhao (jz3581),

Xinyuan Fu (xf2247), Zerui Li (zl3194)

Introduction

Our project is a 2D rhythm arcade game that requires a VGA display and speaker to provide an immersive gaming experience. Players use a customized controller to play the game and follow the rhythm of the music to hit notes by pressing the correct key as they reach the baseline. The more players hit the notes, the higher their score will be. Gameplay is shown below in Figure 1.

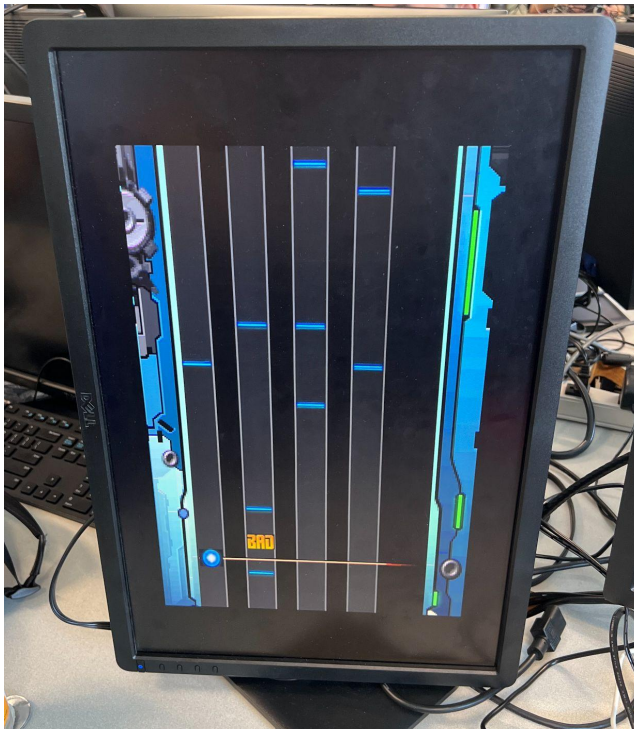


Figure 1. Gameplay screenshot

Our hardware implements VGA graphic display, audio, and user control input. The main device is the DE1-SoC Board. It receives input from the controller and runs software under its HPS. It also sent out the display or audio output signal.

Our software manages everything for the game, it can acquire the input from the user, control the game logic, calculate the game score, and send necessary data to the hardware.

List of topics

1. **System Overview**
2. **Hardware**
    a. **Sprite Graphy**
    b. **Customized Controller**
3. **Software**
    a. **Input translates**
    b. **Initialization**
    c. **Game logic**
4. **Audio Processing**
5. **Hardware Resource Utilization**
    a. **Memory**
    b. **Register**
6. **Work Distribution & Future Work**
7. **Code Appendix**
    a. **Hardware**
    b. **Software**
    c. **Audio**

1. System Overview

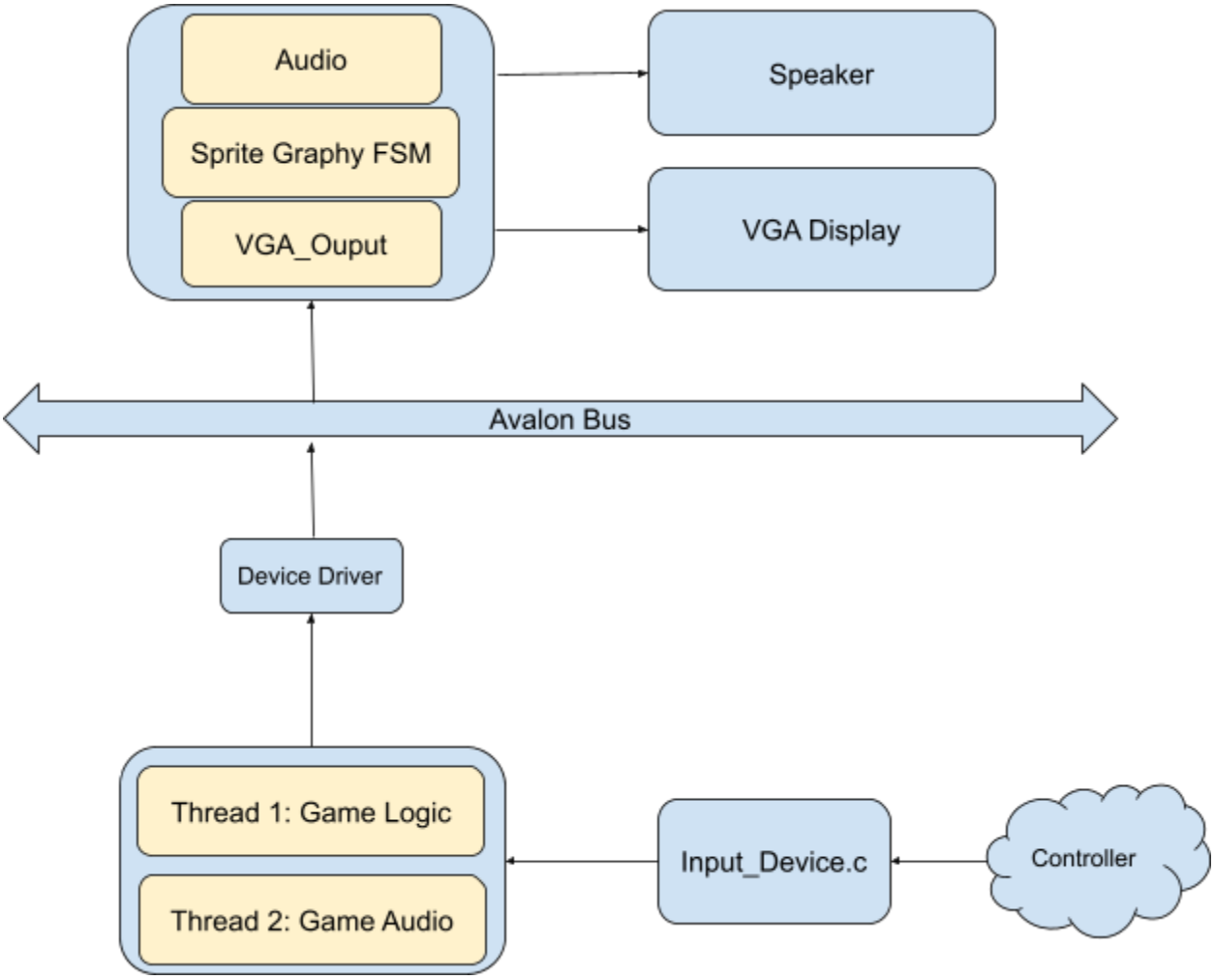


Figure 2. Abstract System block diagram

Our systems are mainly divided into hardware and software sections, but we implement them from graphics and audio two perspectives. Therefore, we have two threads in the software; one for the Audio processing, and the other for handling all game logic, such as input detection and graphics. For our hardware side. We have multiple implementations for different components, including all ROM units for all images.

2. Hardware Implementation

There are three essential input-output devices, a monitor, a speaker, and a custom controller.

Monitor: The Monitor is used to show the main GUI at first. Once the game starts by pressing the start button, notes will drop from the top of the screen. If the user hits a note correctly according to the software logic, an indicator will show up and let the user know if that input is a hit. In the opposite way, When the user misses a note, a miss indicator will also pop up. We used a finite state machine to implement this display algorithm. The algorithm is shown below:

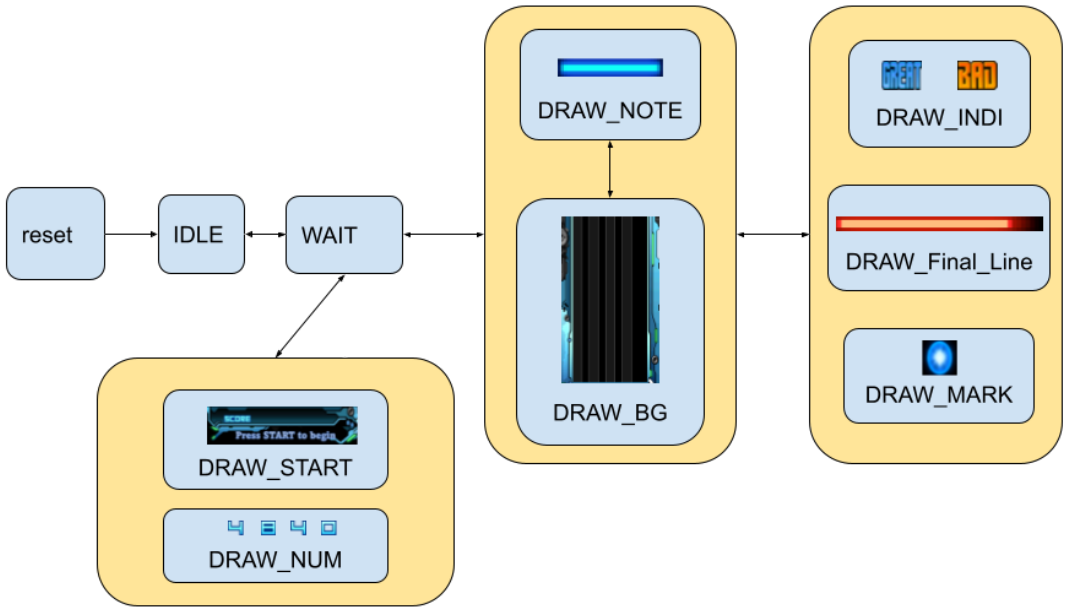


Figure 3. Display Graphic Algorithm

The screen uses a “horizontal line-based” algorithm to print the pixel. We have 2 variables “vcount” and “hcount” to represent the vertical and horizontal position of the pointer. The

pointer will start from the (0,0) of the screen(top left of the screen) move from left to right (horizontal increase)and then shift to the next row(vertical increase).

We have two pages displayed, one is the “start menu”, containing a “press START button to start” and 4 bits decimal number to represent the score, at the beginning of the game, no score is received, so the score will be shown as default “4840”. If the start button is pressed at the start menu, the display will switch to another page which is the gameplay background, all the notes, hit marks, and indicators will only be displayed on the gameplay background. There is a single-bit variable named “page” sent from the software to indicate which page will be displayed, 0 means “start page” while 1 means “ gameplay page”.

Each time the pointer comes to the end of the display it will shift to the next row, and the “line_end” signal will be set up high to inform the finite state machine. If next row does not have any pixel that needs to be shown, the “line_active” will keep at low, and the state machine will stay at IDLE; If the next line contains pixels to be shown, the “line_active” will go high which leads the FSM jumps to “WAIT” state. In “WAIT” state, the pointer will go through a row until it reaches a point with a pixel to be shown, and it will also transfer the state to “xxx_ACTIVE” depending on which element owns this pixel. The State will go back to the Wait once the pointer moves out of the drawing region of the image in that specific row.

For the “Gameplay page”, we have 16 blue notes in total which are the notes we need to hit. These notes are distributed through 4 tracks, each getting 4 notes. The tracks have their vertical coordinate fixed, because we rotate the monitor to display vertically. As a result, we only need to

update x coordinates for all the notes from the software. Each blue note has one dedicated register to receive coordinate updates from the software. In our vga_ball.sv file, we assigned 5 bits address for registers, which counts up to 32 different registers, each register will store a two-byte halfword value. The registers with the address from 0 to 15 are used to store x coordinates for all 16 blue notes; from 16 to 19 are used to store the track status of all tracks. This track status is a 2-bit integer used to inform the screen to display "GREAT", "BAD", hit mark, or nothing. Register 20 is used to store a single bit that controls whether the start menu or gameplay page is displayed. Register 21 is acquired by the score that will be shown on the start menu. Since the score is represented as a 4 digital number, we could use every 4 bits of the 16 bits "writedata" to represent one decimal number. Details of binary number manipulation can be found in Code Appendix.

Input Device (Controller): The input device is a custom controller shown in Figure 4.

Figure 4. Custom Controller

It will supply an operating platform for the user's input. It mainly consists of a couple of buttons, including the "hit" for each track and other operations like "confirm", and "end". We are able to find our custom controller by looking into "*bInterfaceClass*" and "*iInterface*". The class code for the interface of the custom controller is "Human Interface Device" (HID) and the *iInterface* is listed as "4". With all the information from above, we can query the stream of keycode.

3. Software Implementation

a. Input

In this game, we would use the controller as the input tool.This controller will function as a USB keyboard and can be connected to the FPGA board via its wired USB port. It has seven buttons and two knobs. In our game, we only use the seven buttons as the input. The function of the each buttons is shown in table

Button	Function
Start	Start to play the game, Restart
left button	Click and hit the note from left track
second left button	Click and hit the note from second left track
Right button	Click and hit the note from right track
Second right button	Click and hit the note from second right track
Bottom-left button	Cheat button(click all track at same time)
Bottom-right button	Cheat button(click all track at same time)

We can obtain data from our controller via the USB interface during each interrupt. The keycode format is shown below:

Button	Value 1	Value 2	Value 3
Start	0x64a3	0x0001	0x7c78
left button	0x64a3	0x0002	0x7c78
second left button	0x64a3	0x0004	0x7c78
Right button	0x64a3	0x0008	0x7c78
Second right button	0x64a3	0x0010	0x7c78
Bottom-left button	0x64a3	0x0020	0x7c78
Bottom-right button	0x64a3	0x0040	0x7c78

With the data, we can determine which button(s) is pressed. The keycode will be the sum of pressed buttons. For example, if the user pushes the left button and second left button at the same time, the keycode will show as 0x0006.

b. Initialization

Before starting the whole game,we need to initialize the game. First,We are using Python to process music and have obtained approximate beat timings accurate to 0.01 seconds. These timings have been randomly assigned to four arrays to represent the initial values at which each note on each track should be hit. We are using pixels to calculate the fall time, which is 1105 divided by 2 multiplied by 0.005 seconds（for each loop clock）. We are then subtracting the fall time from the hit time to determine the exact time at which each note should appear..At initialization, give the initial interface a score of 4840 and prepare the audiopart for playback in advance . The initial hit time is define here

```
float track_1[163] = {7.19, 7.72, 8.25, 8.78, 9.64, 10.69, 14.06,
    86.27, 86.67, 87.99, 88.38, 88.78, 89.83, 90.63, 90.89, 91.95,
    161.78, 162.31, 163.5, 164.02, 166.0, 166.66, 166.93, 170.23,
float track_2[216] = {6.47, 9.5, 9.77, 12.8, 13.53, 13.79, 14.06,
    71.88, 73.53, 73.93, 74.78, 75.84, 76.63, 77.42, 77.95, 78.48,
    124.29, 124.55, 126.07, 126.86, 128.91, 129.7, 130.23, 130.62,
    180.53, 180.92, 182.77, 183.56, 183.83, 184.35, 184.88, 185.15
float track_3[216] = {6.47, 9.11, 9.24, 10.36, 11.42, 12.01, 12.47
    69.77, 70.3, 70.82, 74.19, 74.78, 75.84, 76.63, 77.42, 77.95,
    128.91, 129.7, 130.23, 130.62, 130.96, 132.67, 132.94, 134.45,
    179.6, 180.66, 181.25, 181.65, 182.64, 183.03, 184.35, 185.54,
float track_4[167] = {7.19, 7.72, 8.25, 8.78, 10.36, 11.42, 12.47,
    84.49, 85.87, 86.27, 86.67, 87.72, 88.38, 88.78, 90.1, 90.63,
    161.38, 161.91, 162.7, 163.89, 164.42, 164.82, 165.61, 166.14,
```

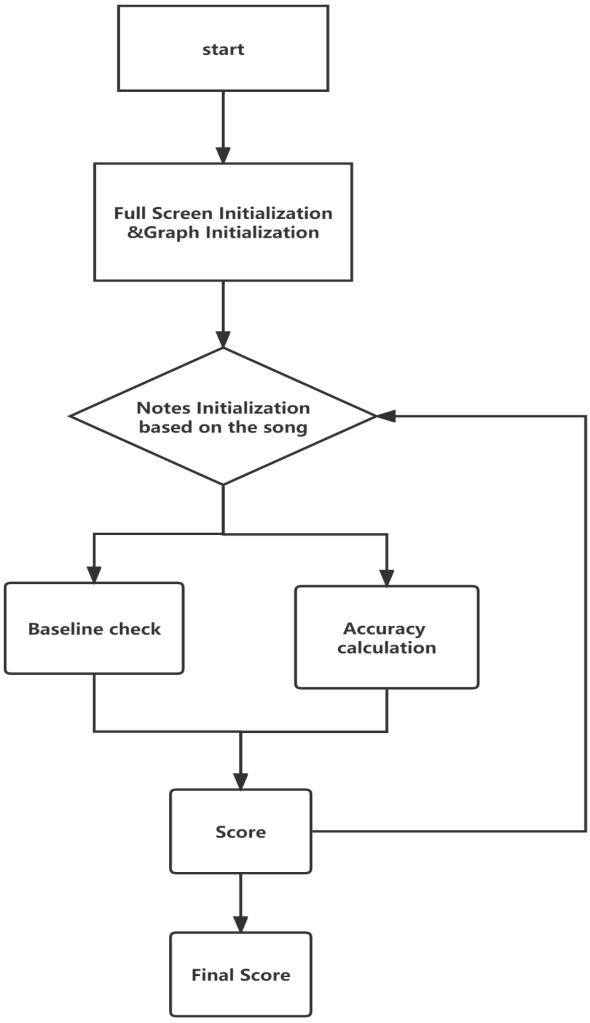
c. Translate input logic

According to the table, we can see that only value2 has changed and the values range from 1 to 64 in decimal, with increments of 2. We determine that pressing 32 and 64 simultaneously is the "cheat button". However, if two or more different buttons are pressed simultaneously, the value of value2 is the sum of the values of those buttons. Based on this principle, we can design an algorithm as follows:

When the value of value2 is greater than 32, all buttons are considered to be pressed and all four track hit marks are lit up.

When the value of value2 is less than 32, we check if it is greater than 16. If it is, we produce a new value2 by adding the value of the right button, as the sum of the remaining button values is less than 16. If it is less than 16, we do not change value2 as the right button has not been pressed. This method resolves the issue with values 8, 4, 2, and 1, and we store the status of each button as 0 or 1 in an array, where 0 represents not pressed and 1 represents pressed. We then pass this array to the hardware. Here is example to notice the track four press

```
if (last_status != packet.b){
    if(packet.b - 16 >= 0){
        remain_v = packet.b - 16;
        if(status_arr[3] != 1){
            status_arr[3] = 1;
```

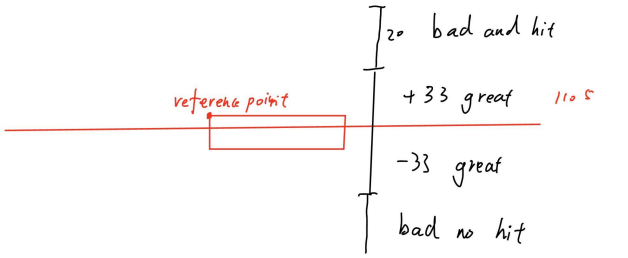


d. Movement logic and hit logic,

Figure 5. Software Algorithm

In our game, The logic of the entire game is based on Figure5 ,each note appears at a different time but falls at the same speed of 2 pixels per 0.005 seconds. For the hit logic, we have set up a hit line at the position of 1105 pixels on the player's screen, which has a length of approximately 1294 pixels. Based on the falling speed of the notes and the normal reaction time of the players in music games, we define the area within a range of plus or minus 33 pixels (1072-1138) from the top left corner of the note as the "great" section. If a player hits a note within this section, it

immediately disappears and scores are obtained. The section between 33-53 pixels (1052-1072) below the great section is defined as "bad". If a player hits a note within this section, it immediately disappears, but the player still gets points as hitting the note. Anything beyond the range of negative 33 pixels (>1138) is considered a miss and is also classified as bad.



And here is hit logic

```
for (int i = 0; i<4; i++){
    if(location_4[i]>=  1072&& location_4[i]<= 1138){

        status_arr[3] = 2;
        location_4[i] = 1500;
        total_score += note_score;
    }
    if ((location_4[i]>= 1052 && location_4[i]<1072)){
        status_arr[3] = 3;
        location_4[i] = 1500;
    }
```

e. Player logic

Start and restart

We have a "Start" button to begin the game and switch screens. Once the game music finishes playing, the first round of the game is complete. Upon completing the first round, the interface

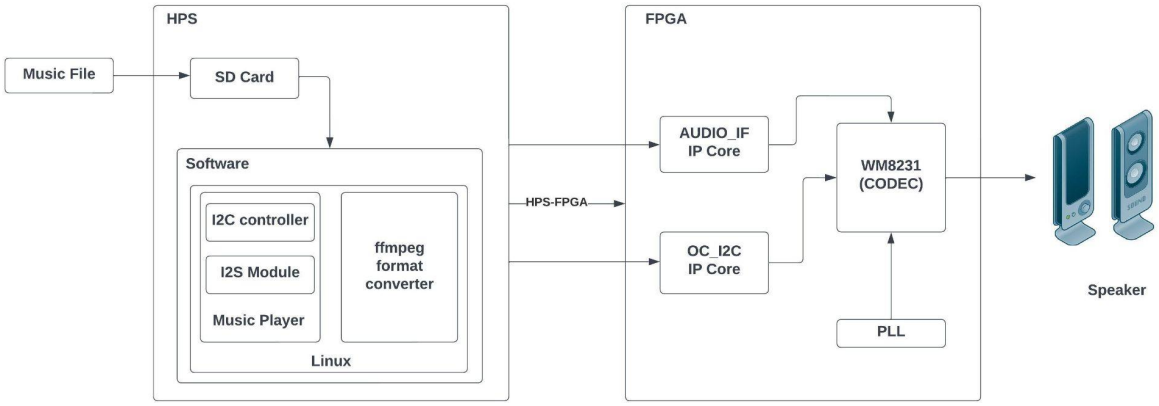
automatically jumps to the initialization screen and returns the score. Pressing the "Start" button at this point will allow the game to be restarted.

Score

Our game has a maximum score of 1000 points. Based on the number of notes, each hit is worth approximately 1.7 points. After completing the game, the final total score is obtained (which can be less than or equal to 1000).

4. Audio Processing

a. Audio Architecture



The audio processing component of our system is derived from music player design from Terasic DE1-soc sample code and can be divided into two main parts: the software part running on HPS and the hardware part designed on FPGA. The software part is responsible for various tasks, including reading the digital audio data file (.mp3), converting it to .pcm format, initializing and controlling the CODEC, configuring registers, and outputting the audio data to the FIFO. On the other hand, the hardware part handles buffering of audio data from HPS into the FIFO, loading data from the FIFO to the CODEC, generating a unique clock frequency for the CODEC chip, and configuring the CODEC chip via I2C.

b. Audio Hardware

To facilitate the operation of the CODEC audio chip, we have utilized two major IPs: AUDIO_IF IP from terasic and OC_I2C from wishbone. The OP_I2C IP contains an I2C controller responsible for managing the configuration data that needs to be written to the CODEC. In the AUDIO_IF IP, we have implemented a FIFO IP and its wrapper. The FIFO acts as a buffer for the audio data, allowing writing via the Avalon bus with a system clock of 50MHz and reading with the CODEC-specific clock of 18.432MHz. This asynchronous data transfer between the two clocks ensures efficient data flow.

c. Audio Software

File name	Description
main.c	Process the hw address with virtual base address read the .mp3 file can call ffmpeg to convert the .mp3 to .pcm format.
pcm.c/.h	processing the .pcm file data and call Audio fifo function to read and load the data into fifo.
audio_control.c/.h	Audio related control functions.
hps_0.h	address header file.

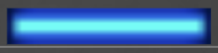
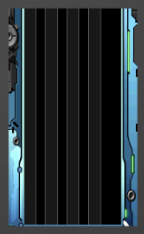
The file structure of the Audio software is derived from the sample design of the CODEC application provided by Terasic VIP sample code. We have made modifications to enable the selection of different .mp3 files. Additionally, we have optimized the code by utilizing an open-source program called ffmpeg in Linux. Ffmpeg is an audio data processing program that simplifies audio file conversion between different formats, including the conversion to .pcm format with customizable options such as the number of channels and sample rate. By leveraging ffmpeg, we have eliminated a significant portion of the original source code related to audio processing and seamlessly integrated the ffmpeg command using the system() function.

Since the code is compiled using the cross-compiler arm-linux-gnueabihf-gcc, which is a cross-toolchain package for the armhf architecture and includes hardware libraries created for arm architecture processor using in Altera, it is unlikely that we can directly compile it with our game logic software. Instead of

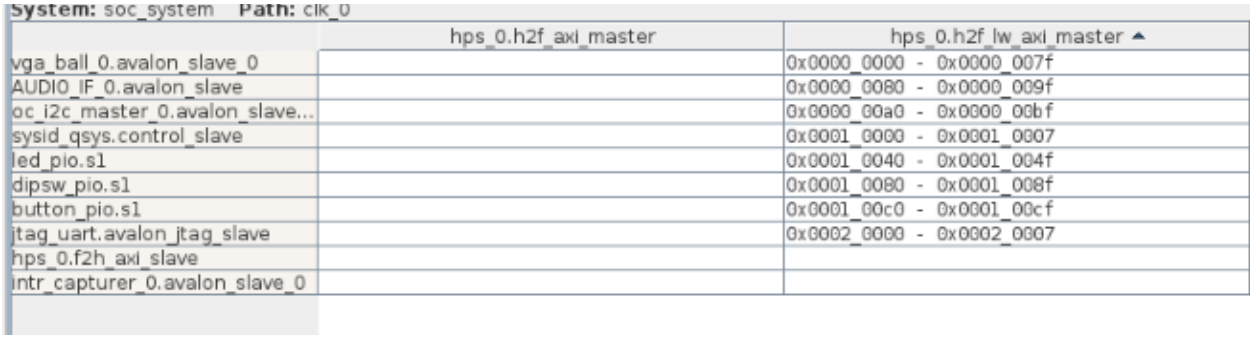
compiling everything together, we have employed a similar method as with ffmpeg by calling the executable function using system(). This approach allows us to interface the compiled audio processing code with the game logic software effectively.

5. Graphic Memory Budget

a. Memory(ROM) Usage:

Name	Graphics	Pixels#	Usage(bits)
note_blue		34*6	4896
baseline_red		286*4	27456
background		191*323	1480632
hit_mark		30*30	21600
hit_indicater		35*25	21000
miss_indicater		35*25	21000
nums	 etc.	15*13*10	46800
start		191*46	210864

b. Register Usage:



System: soc_system Path: clk_0

	hps_0.h2f_axi_master	hps_0.h2f_lw_axi_master ▲
vga_ball_0.avalon_slave_0		0x0000_0000 - 0x0000_007f
AUDIO_IF_0.avalon_slave		0x0000_0080 - 0x0000_009f
oc_i2c_master_0.avalon_slave...		0x0000_00a0 - 0x0000_00bf
sysid_qsys.control_slave		0x0001_0000 - 0x0001_0007
led_pio.s1		0x0001_0040 - 0x0001_004f
dipsw_pio.s1		0x0001_0080 - 0x0001_008f
button_pio.s1		0x0001_00c0 - 0x0001_00cf
jtag_uart.avalon_jtag_slave		0x0002_0000 - 0x0002_0007
hps_0.f2h_axi_slave		
intr_capturer_0.avalon_slave_0		

Work Distribution & Future Work

Lu Zhang (lz2879): Game Design & Cooperator & Sprite Graphy Assistant

Qiao Zhang (qz2486): Audio Processing & System Integration

Jinke Zhao (jz3581): Game Logic Software & System Integration

Xinyuan Fu (xf2247): Sprite Graphic Hardware

Zerui Li (zl3194): Custom Controller Input & System Integration

Our future goals are increasing the current precision of the note placement from .01 up to .001. We also need to increase the note capacity of each track from current 4 to 8, or more. This would allow our game to be more complex and have more choices on note placement. We also want to improve the clocking of both Audio and Game Logic, so we could have better synchronization between each other, and ultimately improve the gameplay experience.

Code Appendix

a. Hardware

VGAball:

```
module vga_ball(input logic        clk,
                input logic      reset,
                input logic [15:0]  writedata,
                input logic      write,
                input        chipselect,
                input logic [5:0]  address,   //2:0

                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 [7:0]     background_r, background_g, background_b;



  logic [10:0]     bg_x, note_x_1, note_x_2, note_x_3, note_x_4, note_x_5, note_x_6, note_x_7, note_x_8;
  logic [10:0]     note_x_9, note_x_10, note_x_11, note_x_12, note_x_13, note_x_14, note_x_15, note_x_16;
  logic [9:0]      bg_y, note_y_0;

  logic [7:0]  ball_num;

  //logic [10:0]   bg_x =
  //logic [9:0]   bg_y = 10;
  logic drawing, page;
  logic [10:0] address_sprite1, address_sprite2, address_sprite3, indicater1, indicater2, indicater3, indicater4;
  logic [15:0] address_bg, score;
  logic [23:0] pix_out;
  //logic enable_sprite;
  logic [23:0] out_sprite1, interlogic1, out_sprite2, interlogic2, out_sprite3, interlogic3, interbg;
  vga_counters counters(.clk50(clk), .*);
```

```
sprite_rom_7 sprite1(.clk(clk), .rst(reset), .vcount(vcount), .hcount(hcount),
                     .note_x_1(note_x_1), .note_x_2(note_x_2), .note_x_3(note_x_3), .note_x_4(note_x_4),
                     .note_x_5(note_x_5), .note_x_6(note_x_6), .note_x_7(note_x_7), .note_x_8(note_x_8),
                     .note_x_9(note_x_9), .note_x_10(note_x_10), .note_x_11(note_x_11), .note_x_12(note_x_12),
                     .note_x_13(note_x_13), .note_x_14(note_x_14), .note_x_15(note_x_15), .note_x_16(note_x_16),
                     .indicater1(indicater1), .indicater2(indicater2), .indicater3(indicater3), .indicater4(indicater4), .score(score), .page(page),
                     .drawing(drawing), .pix_out(pix_out));

 always_ff @(posedge clk) begin
   if (reset) begin
     background_r <= 8'h00;
     background_g <= 8'h00;
     background_b <= 8'h00;
     bg_x <= 1;
     bg_y <= 1;
     note_x_1  <= 160;
     note_x_2  <= 160;
     note_x_3  <= 160;
     note_x_4  <= 160;
     note_x_5  <= 260;
     note_x_6  <= 260;
     note_x_7  <= 260;
     note_x_8  <= 260;
     note_x_9  <= 360;
     note_x_10 <= 360;
     note_x_11 <= 360;
     note_x_12 <= 360;
     note_x_13 <= 460;
     note_x_14 <= 460;
     note_x_15 <= 460;
     note_x_16 <= 460;
     //note_y_0 <= 30;
     indicater1 <= 2;
     indicater2 <= 2;
     indicater3 <= 2;
     indicater4 <= 2;
     page <= 0;
```

```
    score <= 0;


  end else if (chipselect && write)
    case (address)
    //0`15: note, 16:19 hitmark, 20:23 good or bad;
      6'h0  : note_x_1  <= writedata[15:0];
      6'h1  : note_x_2  <= writedata[15:0];
      6'h2  : note_x_3  <= writedata[15:0];
      6'h3  : note_x_4  <= writedata[15:0];
      6'h4  : note_x_5  <= writedata[15:0];
      6'h5  : note_x_6  <= writedata[15:0];
      6'h6  : note_x_7  <= writedata[15:0];
      6'h7  : note_x_8  <= writedata[15:0];
      6'h8  : note_x_9  <= writedata[15:0];
      6'h9  : note_x_10 <= writedata[15:0];
      6'ha : note_x_11 <= writedata[15:0];
      6'hb : note_x_12 <= writedata[15:0];
      6'hc : note_x_13 <= writedata[15:0];
      6'hd : note_x_14 <= writedata[15:0];
      6'he : note_x_15 <= writedata[15:0];
      6'hf : note_x_16 <= writedata[15:0];
      6'h10: indicater1 <= writedata[15:0];
      6'h11: indicater2 <= writedata[15:0];
      6'h12: indicater3 <= writedata[15:0];
      6'h13: indicater4 <= writedata[15:0];
      6'h14: page <= writedata[0];
      6'h15: score <= writedata[15:0];
    endcase

end
```

```
    always_comb begin
      {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
      if (VGA_BLANK_n )
        if (drawing)begin
            {VGA_R, VGA_G, VGA_B} = pix_out;
        end
        else begin
            //enable_sprite = 0;
          {VGA_R, VGA_G, VGA_B} ={background_r, background_g, background_b};
          end
    end
endmodule  //vga_ball
```

Sprite Finite State Machine:

```
module sprite_rom (
    input  wire logic clk,
    input  wire logic rst,
    input  logic [9:0]     vcount,
    input  logic [10:0]    hcount, note_x_1, note_x_2, note_x_3, note_x_4,
    input  logic [10:0]    note_x_5, note_x_6, note_x_7, note_x_8,
    input  logic [10:0]    note_x_9, note_x_10, note_x_11, note_x_12,
    input  logic [10:0]    note_x_13, note_x_14, note_x_15, note_x_16, indicater1, indicater2, indicater3, indicater4,
    input  logic [15:0]    score,
    input  logic page,
    output logic drawing,
    output logic  [23:0]   pix_out
    );

    parameter a = 382, AA = 1292, b = 34, BB = 12, bg_x = 1, bg_y = 51;
    parameter start_x = 550, start_y = 51, STA = 92, st = 191;
    parameter note_y_1 = 161, note_y_2 = 225, note_y_3 = 289, note_y_4 = 353;
    parameter note_y_5 = 161, note_y_6 = 225, note_y_7 = 289, note_y_8 = 353;
    parameter note_y_9 = 161, note_y_10 = 225, note_y_11 = 289, note_y_12 = 353;
    parameter note_y_13 = 161, note_y_14 = 225, note_y_15 = 289, note_y_16 = 353;
    parameter hm = 30, HMM = 60, ind = 35, INDD = 50;
    parameter hm_x = 1075, hm_y_1 = 162, hm_y_2 =226, hm_y_3 = 290, hm_y_4 = 354;
    parameter indi_x = 1020, indi_y_1 = 160, indi_y_2 = 224, indi_y_3 = 288, indi_y_4 = 352;
    parameter finalline_x = 1101, finalline_y = 99, FLL = 8, fl = 286;
    parameter num_x = 580;
    parameter num_y_3 = 258, num_y_2 = 223, num_y_1 = 188, num_y_0 = 153;
    parameter num_height = 13, num_width = 15 ;

    logic line_active, line_end, bg_active, note_active, frame_finish;
    logic indicater_active, mark_active, finalline_active, start_active, num_active;
    logic [7:0]  address_num;
    logic [10:0] address_note, address_hitind, address_missind, address_hitmark;
    logic [11:0] address_finalline;
    logic [15:0] address_bg, address_start;
    logic [23:0] out_bg, out_note, interbuffer, out_hitind, out_hitmark, out_missind, out_finalline, out_start;
    logic [23:0] out_num0, out_num1, out_num2, out_num3, out_num4, out_num5, out_num6, out_num7, out_num8, out_num9;
```

```
enum {
    IDLE,
    WAIT,
    DRAW_BG,
    DRAW_NOTE,
    RESET,
    INDICATER,
    DRAW_FINAL,
    MARK,
    DRAW_START,
    DRAW_NUM
} state;

note_blue tile1(.address(address_note),.clock(clk),.q(out_note));
background bg(.address(address_bg),.clock(clk),.q(out_bg));
hit_indicater hitind(.address(address_hitind),.clock(clk),.q(out_hitind));
hit_mark hitmark(.address(address_hitmark),.clock(clk),.q(out_hitmark));
miss_indicater missind(.address(address_missind),.clock(clk),.q(out_missind));
note_red line(.address(address_finalline),.clock(clk),.q(out_finalline));
start start(.address(address_start),.clock(clk),.q(out_start));
num_0 num0(.address(address_num),.clock(clk),.q(out_num0));
num_1 num1(.address(address_num),.clock(clk),.q(out_num1));
num_2 num2(.address(address_num),.clock(clk),.q(out_num2));
num_3 num3(.address(address_num),.clock(clk),.q(out_num3));
num_4 num4(.address(address_num),.clock(clk),.q(out_num4));
num_5 num5(.address(address_num),.clock(clk),.q(out_num5));
num_6 num6(.address(address_num),.clock(clk),.q(out_num6));
num_7 num7(.address(address_num),.clock(clk),.q(out_num7));
num_8 num8(.address(address_num),.clock(clk),.q(out_num8));
num_9 num9(.address(address_num),.clock(clk),.q(out_num9));
```

```
always_comb begin
    line_active = ((((vcount - note_y_1 >= 0)&&(vcount - note_y_1 <= b-1)) ||
                   ((vcount - note_y_2 >= 0)&&(vcount - note_y_2 <= b-1)) ||
                   ((vcount - note_y_3 >= 0)&&(vcount - note_y_3 <= b-1)) ||
                   ((vcount - note_y_4 >= 0)&&(vcount - note_y_4 <= b-1)) ||
                   ((vcount - note_y_5 >= 0)&&(vcount - note_y_5 <= b-1)) ||
                   ((vcount - note_y_6 >= 0)&&(vcount - note_y_6 <= b-1)) ||
                   ((vcount - note_y_7 >= 0)&&(vcount - note_y_7 <= b-1)) ||
                   ((vcount - note_y_8 >= 0)&&(vcount - note_y_8 <= b-1)) ||
                   ((vcount - note_y_9 >= 0)&&(vcount - note_y_9 <= b-1)) ||
                   ((vcount - note_y_10 >= 0)&&(vcount - note_y_10 <= b-1)) ||
                   ((vcount - note_y_11 >= 0)&&(vcount - note_y_11 <= b-1)) ||
                   ((vcount - note_y_12 >= 0)&&(vcount - note_y_12 <= b-1)) ||
                   ((vcount - note_y_13 >= 0)&&(vcount - note_y_13 <= b-1)) ||
                   ((vcount - note_y_14 >= 0)&&(vcount - note_y_14 <= b-1)) ||
                   ((vcount - note_y_15 >= 0)&&(vcount - note_y_15 <= b-1)) ||
                   ((vcount - note_y_16 >= 0)&&(vcount - note_y_16 <= b-1)) ||
                   ((vcount - bg_y >= 0)&&(vcount - bg_y <= a-1))) && page)
                    ||
                    ((((vcount - start_y >= 0)&&(vcount - start_y <= 2*st-1)) ||
                    ((vcount - num_y_0 >= 0)&&(vcount - num_y_0 <= num_width - 1))||
                    ((vcount - num_y_1 >= 0)&&(vcount - num_y_1 <= num_width - 1))||
                    ((vcount - num_y_2 >= 0)&&(vcount - num_y_2 <= num_width - 1))||
                    ((vcount - num_y_3 >= 0)&&(vcount - num_y_3 <= num_width - 1)) )&& !page);
    line_end = hcount >= 1260;
    bg_active  = (hcount >= bg_x && hcount <= bg_x + AA - 1) && page;
```

```
note_active  =  ((hcount >= note_x_1 && hcount <= note_x_1 + BB - 1 && vcount - note_y_1 >= 0 && vcount - note_y_1 <= b-1) ||
                (hcount >= note_x_2 && hcount <= note_x_2 + BB - 1 && vcount - note_y_2 >= 0 && vcount - note_y_2 <= b-1) ||
                (hcount >= note_x_3 && hcount <= note_x_3 + BB - 1 && vcount - note_y_3 >= 0 && vcount - note_y_3 <= b-1) ||
                (hcount >= note_x_4 && hcount <= note_x_4 + BB - 1 && vcount - note_y_4 >= 0 && vcount - note_y_4 <= b-1) ||
                (hcount >= note_x_5 && hcount <= note_x_5 + BB - 1 && vcount - note_y_5 >= 0 && vcount - note_y_5 <= b-1) ||
                (hcount >= note_x_6 && hcount <= note_x_6 + BB - 1 && vcount - note_y_6 >= 0 && vcount - note_y_6 <= b-1) ||
                (hcount >= note_x_7 && hcount <= note_x_7 + BB - 1 && vcount - note_y_7 >= 0 && vcount - note_y_7 <= b-1) ||
                (hcount >= note_x_8 && hcount <= note_x_8 + BB - 1 && vcount - note_y_8 >= 0 && vcount - note_y_8 <= b-1) ||
                (hcount >= note_x_9 && hcount <= note_x_9 + BB - 1 && vcount - note_y_9 >= 0 && vcount - note_y_9 <= b-1) ||
                (hcount >= note_x_10 && hcount <= note_x_10 + BB - 1 && vcount - note_y_10 >= 0 && vcount - note_y_10 <= b-1) ||
                (hcount >= note_x_11 && hcount <= note_x_11 + BB - 1 && vcount - note_y_11 >= 0 && vcount - note_y_11 <= b-1) ||
                (hcount >= note_x_12 && hcount <= note_x_12 + BB - 1 && vcount - note_y_12 >= 0 && vcount - note_y_12 <= b-1) ||
                (hcount >= note_x_13 && hcount <= note_x_13 + BB - 1 && vcount - note_y_13 >= 0 && vcount - note_y_13 <= b-1) ||
                (hcount >= note_x_14 && hcount <= note_x_14 + BB - 1 && vcount - note_y_14 >= 0 && vcount - note_y_14 <= b-1) ||
                (hcount >= note_x_15 && hcount <= note_x_15 + BB - 1 && vcount - note_y_15 >= 0 && vcount - note_y_15 <= b-1) ||
                (hcount >= note_x_16 && hcount <= note_x_16 + BB - 1 && vcount - note_y_16 >= 0 && vcount - note_y_16 <= b-1)) && page;
frame_finish = hcount == 1255 && vcount == 475;
indicater_active = (((hcount >= indi_x && hcount <= indi_x + INDD - 1 && vcount - indi_y_1 >= 0 && vcount - indi_y_1 <= ind-1)&&
                   (indicater1==2||indicater1==3||indicater1==4))||
                   ((hcount >= indi_x && hcount <= indi_x + INDD - 1 && vcount - indi_y_2 >= 0 && vcount - indi_y_2 <= ind-1)&&
                   (indicater2==2||indicater2==3||indicater2==4))||
                   ((hcount >= indi_x && hcount <= indi_x + INDD - 1 && vcount - indi_y_3 >= 0 && vcount - indi_y_3 <= ind-1)&&
                   (indicater3==2||indicater3==3||indicater3==4))||
                   ((hcount >= indi_x && hcount <= indi_x + INDD - 1 && vcount - indi_y_4 >= 0 && vcount - indi_y_4 <= ind-1)&&
                   (indicater4==2||indicater4==3||indicater4==4)) && page;

mark_active = (((hcount >= hm_x && hcount <= hm_x + HMM - 1 && vcount - hm_y_1 >= 0 && vcount - hm_y_1 <= hm-1)&&(indicater1==1||indicater1==2||indicater1==3))||
              ((hcount >= hm_x && hcount <= hm_x + HMM - 1 && vcount - hm_y_2 >= 0 && vcount - hm_y_2 <= hm-1)&&(indicater2==1||indicater2==2||indicater2==3))||
              ((hcount >= hm_x && hcount <= hm_x + HMM - 1 && vcount - hm_y_3 >= 0 && vcount - hm_y_3 <= hm-1)&&(indicater3==1||indicater3==2||indicater3==3))||
              ((hcount >= hm_x && hcount <= hm_x + HMM - 1 && vcount - hm_y_4 >= 0 && vcount - hm_y_4 <= hm-1)&&(indicater4==1||indicater4==2||indicater4==3)) && page

finalline_active = (hcount >= finalline_x && hcount <= finalline_x + FLL - 1 && vcount - finalline_y >= 0 && vcount - finalline_y <= fl-1) && page;

start_active = (hcount >= start_x && hcount <= start_x + 2*STA - 1) && !page;
```

```
    num_active = ((hcount >= num_x && hcount <= num_x + 2*num_height - 1)&&((vcount - num_y_0 >= 0 && vcount - num_y_0 <= num_width - 1)||
                 (vcount - num_y_1 >= 0 && vcount - num_y_1 <= num_width - 1)||(vcount - num_y_2 >= 0 && vcount - num_y_2 <= num_width - 1)||
                 (vcount - num_y_3 >= 0 && vcount - num_y_3 <= num_width - 1)))&& !page;
end

always_ff @(posedge clk) begin
    if (rst) begin
        state <= IDLE;
        pix_out <= 0;
        drawing <= 0;
        address_bg <= 0;
        address_note <= 0;
        address_start <= 0;
        address_finalline <= 0;
        address_hitind <= 0;
        address_hitmark <= 0;
        address_missind <= 0;
        address_num <= 0;
    end else begin
        case (state)
            IDLE:begin
                if(line_active)begin
                    state <= WAIT;
                end else begin
                    state <= IDLE;
                end
            end
```

```
RESET: begin
    state <= IDLE;
    pix_out <= 0;
    drawing <= 0;
    address_bg <= 0;
    address_note <= 0;
    address_start <= 0;
    address_finalline <= 0;
    address_hitind <= 0;
    address_hitmark <= 0;
    address_missind <= 0;
    address_num <= 0;
end
WAIT:begin
    if(frame_finish) state <= RESET;
    else begin
        if(line_end) state <= IDLE;
        else begin
            if(note_active) begin
                state <= DRAW_NOTE;
            end
            else if(bg_active)begin
                state <= DRAW_BG;
            end
            if(start_active) begin
                state <= DRAW_START;
            end

        end
    end
    pix_out <= 0;
    drawing <= 0;
end
```

```
DRAW_START:begin
    if (!start_active && !num_active) state <= WAIT;
    else if(num_active) state <= DRAW_NUM;
    else state <= DRAW_START;
    address_start = (st - 1 - ((vcount - start_y)>>1)) + (((hcount - start_x)>>2)*st);
    drawing = 1;
    pix_out = out_start;

end
DRAW_NUM:begin
    if(!num_active) state <= DRAW_START;
    else state <= DRAW_NUM;
    if(vcount - num_y_0 >= 0 && vcount - num_y_0 <= num_width - 1 )begin
    case(score & 16'h000f)
     0: begin
        address_num = (num_width - 1 - (vcount - num_y_0)) + (((hcount - num_x)>>1)*num_width);
        pix_out = out_num0;
        end
     1: begin
        address_num = (num_width - 1 - (vcount - num_y_0)) + (((hcount - num_x)>>1)*num_width);
        pix_out = out_num1;
        end
     2: begin
        address_num = (num_width - 1 - (vcount - num_y_0)) + (((hcount - num_x)>>1)*num_width);
        pix_out = out_num2;
        end
     3: begin
        address_num = (num_width - 1 - (vcount - num_y_0)) + (((hcount - num_x)>>1)*num_width);
        pix_out = out_num3;
        end
     4: begin
        address_num = (num_width - 1 - (vcount - num_y_0)) + (((hcount - num_x)>>1)*num_width);
        pix_out = out_num4;
        end
```

```
 5: begin
    address_num = (num_width - 1 - (vcount - num_y_0)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num5;
    end
 6: begin
    address_num = (num_width - 1 - (vcount - num_y_0)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num6;
    end
 7: begin
    address_num = (num_width - 1 - (vcount - num_y_0)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num7;
    end
 8: begin
    address_num = (num_width - 1 - (vcount - num_y_0)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num8;
    end
 9: begin
    address_num = (num_width - 1 - (vcount - num_y_0)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num9;
    end
endcase
end
else if(vcount - num_y_1 >= 0 && vcount - num_y_1 <= num_width - 1)begin
case((score&16'h00f0)>>4)
 0: begin
    address_num = (num_width - 1 - (vcount - num_y_1)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num0;
    end
 1: begin
    address_num = (num_width - 1 - (vcount - num_y_1)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num1;
    end
 2: begin
    address_num = (num_width - 1 - (vcount - num_y_1)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num2;
    end
```

```
 3: begin
    address_num = (num_width - 1 - (vcount - num_y_1)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num3;
    end
 4: begin
    address_num = (num_width - 1 - (vcount - num_y_1)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num4;
    end
 5: begin
    address_num = (num_width - 1 - (vcount - num_y_1)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num5;
    end
 6: begin
    address_num = (num_width - 1 - (vcount - num_y_1)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num6;
    end
 7: begin
    address_num = (num_width - 1 - (vcount - num_y_1)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num7;
    end
 8: begin
    address_num = (num_width - 1 - (vcount - num_y_1)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num8;
    end
 9: begin
    address_num = (num_width - 1 - (vcount - num_y_1)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num9;
    end
endcase
end
else if(vcount - num_y_2 >= 0 && vcount - num_y_2 <= num_width - 1)begin
case((score&16'h0f00)>>8)
 0: begin
    address_num = (num_width - 1 - (vcount - num_y_2)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num0;
    end
```

```
1: begin
   address_num = (num_width - 1 - (vcount - num_y_2)) + (((hcount - num_x)>>1)*num_width);
   pix_out = out_num1;
   end
2: begin
   address_num = (num_width - 1 - (vcount - num_y_2)) + (((hcount - num_x)>>1)*num_width);
   pix_out = out_num2;
   end
3: begin
   address_num = (num_width - 1 - (vcount - num_y_2)) + (((hcount - num_x)>>1)*num_width);
   pix_out = out_num3;
   end
4: begin
   address_num = (num_width - 1 - (vcount - num_y_2)) + (((hcount - num_x)>>1)*num_width);
   pix_out = out_num4;
   end
5: begin
   address_num = (num_width - 1 - (vcount - num_y_2)) + (((hcount - num_x)>>1)*num_width);
   pix_out = out_num5;
   end
6: begin
   address_num = (num_width - 1 - (vcount - num_y_2)) + (((hcount - num_x)>>1)*num_width);
   pix_out = out_num6;
   end
7: begin
   address_num = (num_width - 1 - (vcount - num_y_2)) + (((hcount - num_x)>>1)*num_width);
   pix_out = out_num7;
   end
8: begin
   address_num = (num_width - 1 - (vcount - num_y_2)) + (((hcount - num_x)>>1)*num_width);
   pix_out = out_num8;
   end
9: begin
   address_num = (num_width - 1 - (vcount - num_y_2)) + (((hcount - num_x)>>1)*num_width);
   pix_out = out_num9;
   end
endcase
```

```
end
else if(vcount - num_y_3 >= 0 && vcount - num_y_3 <= num_width - 1)begin
case((score&16'hf000)>>12)
 0: begin
    address_num = (num_width - 1 - (vcount - num_y_3)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num0;
    end
 1: begin
    address_num = (num_width - 1 - (vcount - num_y_3)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num1;
    end
 2: begin
    address_num = (num_width - 1 - (vcount - num_y_3)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num2;
    end
 3: begin
    address_num = (num_width - 1 - (vcount - num_y_3)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num3;
    end
 4: begin
    address_num = (num_width - 1 - (vcount - num_y_3)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num4;
    end
 5: begin
    address_num = (num_width - 1 - (vcount - num_y_3)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num5;
    end
 6: begin
    address_num = (num_width - 1 - (vcount - num_y_3)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num6;
    end
 7: begin
    address_num = (num_width - 1 - (vcount - num_y_3)) + (((hcount - num_x)>>1)*num_width);
    pix_out = out_num7;
    end
```

```
        8: begin
            address_num = (num_width - 1 - (vcount - num_y_3)) + (((hcount - num_x)>>1)*num_width);
            pix_out = out_num8;
            end
        9: begin
            address_num = (num_width - 1 - (vcount - num_y_3)) + (((hcount - num_x)>>1)*num_width);
            pix_out = out_num9;
            end
        endcase
        end
        drawing <= 1;
    end
    DRAW_BG:begin
        if(finalline_active && !note_active && !mark_active)begin
            state <= DRAW_FINAL;
        end
        else begin
            if(!indicater_active && !mark_active)begin
                if(note_active) state <= DRAW_NOTE;
                else  begin
                    if (!bg_active) state <= WAIT;
                    else state <= DRAW_BG;
                end
            end
            else if(indicater_active && !mark_active)begin
                    state <= INDICATER;
            end
            else if(!indicater_active && mark_active)begin
                state <= MARK;
            end
        end
        address_bg = (190 - ((vcount - bg_y)>>1)) + (((hcount - bg_x)>>2)*191);
        drawing = 1;
        pix_out = out_bg;
    end
```

```
DRAW_NOTE:begin
    if(!indicater_active && !mark_active)begin
        if(!note_active && !finalline_active)begin
            state <= bg_active ? DRAW_BG : WAIT;
        end
        else if(!note_active && finalline_active)begin
            state <= DRAW_FINAL;
        end
        else begin
            state <= DRAW_NOTE;
        end
    end
    else if(indicater_active && !mark_active)begin
        state <= INDICATER;
    end
    else if(!indicater_active && mark_active)begin
        state <= MARK;
    end
    if(hcount >= note_x_1 && hcount <= note_x_1 + BB - 1 && vcount - note_y_1 >= 0 && vcount - note_y_1 <= b-1)begin
        address_note = (b - 1 - (vcount - note_y_1)) + (((hcount - note_x_1)>>1)*b);
        pix_out = out_note;
    end
    else if(hcount >= note_x_2 && hcount <= note_x_2 + BB - 1 && vcount - note_y_2 >= 0 && vcount - note_y_2 <= b-1)begin
        address_note = (b - 1 - (vcount - note_y_2)) + (((hcount - note_x_2)>>1)*b);
        pix_out = out_note;
    end
    else if(hcount >= note_x_3 && hcount <= note_x_3 + BB - 1 && vcount - note_y_3 >= 0 && vcount - note_y_3 <= b-1)begin
        address_note = (b - 1 - (vcount - note_y_3)) + (((hcount - note_x_3)>>1)*b);
        pix_out = out_note;
    end
    else if(hcount >= note_x_4 && hcount <= note_x_4 + BB - 1 && vcount - note_y_4 >= 0 && vcount - note_y_4 <= b-1)begin
        address_note = (b - 1 - (vcount - note_y_4)) + (((hcount - note_x_4)>>1)*b);
        pix_out = out_note;
    end
```

```
else if(hcount >= note_x_5 && hcount <= note_x_5 + BB - 1 && vcount - note_y_5 >= 0 && vcount - note_y_5 <= b-1)begin
      address_note = (b - 1 - (vcount - note_y_5)) + (((hcount - note_x_5)>>1)*b);
      pix_out = out_note;
end
else if(hcount >= note_x_6 && hcount <= note_x_6 + BB - 1 && vcount - note_y_6 >= 0 && vcount - note_y_6 <= b-1)begin
      address_note = (b - 1 - (vcount - note_y_6)) + (((hcount - note_x_6)>>1)*b);
      pix_out = out_note;
end
else if(hcount >= note_x_7 && hcount <= note_x_7 + BB - 1 && vcount - note_y_7 >= 0 && vcount - note_y_7 <= b-1)begin
      address_note = (b - 1 - (vcount - note_y_7)) + (((hcount - note_x_7)>>1)*b);
      pix_out = out_note;
end
else if(hcount >= note_x_8 && hcount <= note_x_8 + BB - 1 && vcount - note_y_8 >= 0 && vcount - note_y_8 <= b-1)begin
      address_note = (b - 1 - (vcount - note_y_8)) + (((hcount - note_x_8)>>1)*b);
      pix_out = out_note;
end
else if(hcount >= note_x_9 && hcount <= note_x_9 + BB - 1 && vcount - note_y_9 >= 0 && vcount - note_y_9 <= b-1)begin
      address_note = (b - 1 - (vcount - note_y_9)) + (((hcount - note_x_9)>>1)*b);
      pix_out = out_note;
end
else if(hcount >= note_x_10 && hcount <= note_x_10 + BB - 1 && vcount - note_y_10 >= 0 && vcount - note_y_10 <= b-1)begin
      address_note = (b - 1 - (vcount - note_y_10)) + (((hcount - note_x_10)>>1)*b);
      pix_out = out_note;
end
else if(hcount >= note_x_11 && hcount <= note_x_11 + BB - 1 && vcount - note_y_11 >= 0 && vcount - note_y_11 <= b-1)begin
      address_note = (b - 1 - (vcount - note_y_11)) + (((hcount - note_x_11)>>1)*b);
      pix_out = out_note;
end
else if(hcount >= note_x_12 && hcount <= note_x_12 + BB - 1 && vcount - note_y_12 >= 0 && vcount - note_y_12 <= b-1)begin
      address_note = (b - 1 - (vcount - note_y_12)) + (((hcount - note_x_12)>>1)*b);
      pix_out = out_note;
end
else if(hcount >= note_x_13 && hcount <= note_x_13 + BB - 1 && vcount - note_y_13 >= 0 && vcount - note_y_13 <= b-1)begin
      address_note = (b - 1 - (vcount - note_y_13)) + (((hcount - note_x_13)>>1)*b);
      pix_out = out_note;
```

```
      end
      else if(hcount >= note_x_14 && hcount <= note_x_14 + BB - 1 && vcount - note_y_14 >= 0 && vcount - note_y_14 <= b-1)begin
            address_note = (b - 1 - (vcount - note_y_14)) + (((hcount - note_x_14)>>1)*b);
            pix_out = out_note;
      end
      else if(hcount >= note_x_15 && hcount <= note_x_15 + BB - 1 && vcount - note_y_15 >= 0 && vcount - note_y_15 <= b-1)begin
            address_note = (b - 1 - (vcount - note_y_15)) + (((hcount - note_x_15)>>1)*b);
            pix_out = out_note;
      end
      else if(hcount >= note_x_16 && hcount <= note_x_16 + BB - 1 && vcount - note_y_16 >= 0 && vcount - note_y_16 <= b-1)begin
            address_note = (b - 1 - (vcount - note_y_16)) + (((hcount - note_x_16)>>1)*b);
            pix_out = out_note;
      end
      drawing = 1;
   end
   INDICATER:begin
      if(indicater_active)begin
         state <= INDICATER;
      end
      else begin
         if(note_active)begin
            state <= DRAW_NOTE;
         end
         else state <= DRAW_BG;
      end
      if(hcount >= indi_x && hcount <= indi_x + INDD - 1 && vcount - indi_y_1 >= 0 && vcount - indi_y_1 <= ind-1)begin
         if(indicater1==2)begin
            address_hitind = (ind - 1 - (vcount - indi_y_1)) + (((hcount - indi_x)>>1)*ind);
            pix_out = out_hitind;
         end
         else begin
            address_missind = (ind - 1 - (vcount - indi_y_1)) + (((hcount - indi_x)>>1)*ind);
            pix_out = out_missind;
         end
      end
```

```
    else if(hcount >= indi_x && hcount <= indi_x + INDD - 1 && vcount - indi_y_2 >= 0 && vcount - indi_y_2 <= ind-1)begin
        if(indicater2==2)begin
            address_hitind = (ind - 1 - (vcount - indi_y_2)) + (((hcount - indi_x)>>1)*ind);
            pix_out = out_hitind;
        end
        else begin
            address_missind = (ind - 1 - (vcount - indi_y_2)) + (((hcount - indi_x)>>1)*ind);
            pix_out = out_missind;
        end
    end
    else if(hcount >= indi_x && hcount <= indi_x + INDD - 1 && vcount - indi_y_3 >= 0 && vcount - indi_y_3 <= ind-1)begin
        if(indicater3==2)begin
            address_hitind = (ind - 1 - (vcount - indi_y_3)) + (((hcount - indi_x)>>1)*ind);
            pix_out = out_hitind;
        end
        else begin
            address_missind = (ind - 1 - (vcount - indi_y_3)) + (((hcount - indi_x)>>1)*ind);
            pix_out = out_missind;
        end
    end
    else if(hcount >= indi_x && hcount <= indi_x + INDD - 1 && vcount - indi_y_4 >= 0 && vcount - indi_y_4 <= ind-1)begin
        if(indicater4==2)begin
            address_hitind = (ind - 1 - (vcount - indi_y_4)) + (((hcount - indi_x)>>1)*ind);
            pix_out = out_hitind;
        end
        else begin
            address_missind = (ind - 1 - (vcount - indi_y_4)) + (((hcount - indi_x)>>1)*ind);
            pix_out = out_missind;
        end
    end
    drawing = 1;
end
```

```
MARK:begin
    if(mark_active)begin
        state <= MARK;
    end
    else begin
        if(note_active)begin
            state <= DRAW_NOTE;
        end
        else if(!note_active && finalline_active)begin
            state <= DRAW_FINAL;
        end
        else state <= DRAW_BG;
    end
    if(hcount >= hm_x && hcount <= hm_x + HMM - 1 && vcount - hm_y_1 >= 0 && vcount - hm_y_1 <= hm-1)begin
            address_hitmark = (hm - 1 - (vcount - hm_y_1)) + (((hcount - hm_x)>>1)*hm);
            pix_out = out_hitmark;
    end
    else if(hcount >= hm_x && hcount <= hm_x + HMM - 1 && vcount - hm_y_2 >= 0 && vcount - hm_y_2 <= hm-1)begin
            address_hitmark = (hm - 1 - (vcount - hm_y_2)) + (((hcount - hm_x)>>1)*hm);
            pix_out = out_hitmark;
    end
    else if(hcount >= hm_x && hcount <= hm_x + HMM - 1 && vcount - hm_y_3 >= 0 && vcount - hm_y_3 <= hm-1)begin
            address_hitmark = (hm - 1 - (vcount - hm_y_3)) + (((hcount - hm_x)>>1)*hm);
            pix_out = out_hitmark;
    end
    else if(hcount >= hm_x && hcount <= hm_x + HMM - 1 && vcount - hm_y_4 >= 0 && vcount - hm_y_4 <= hm-1)begin
            address_hitmark = (hm - 1 - (vcount - hm_y_4)) + (((hcount - hm_x)>>1)*hm);
            pix_out = out_hitmark;
    end
    drawing = 1;
end
```

```
                DRAW_FINAL:begin
                    if(!note_active && !mark_active && finalline_active)begin
                        state <= DRAW_FINAL;
                    end
                    else if(note_active && !mark_active)begin
                        state <= DRAW_NOTE;
                    end
                    else if(mark_active)begin
                        state <= MARK;
                    end
                    else state <= DRAW_BG;
                    address_finalline = (fl - 1 - (vcount - finalline_y)) + (((hcount - finalline_x)>>1)*fl);
                    drawing = 1;
                    pix_out = out_finalline;
                end
                default: state <= IDLE;
            endcase
        end
    end

endmodule
```

OC_I2C master.v

```
module oc_i2c_master (
    inout scl_pad_io ,
    inout sda_pad_io ,

    output wb_ack_o ,
    input [2:0] wb_adr_i  ,
    input wb_clk_i ,
    //input wb_cyc_i ,
    input [31:0] wb_dat_i ,
    output [31:0] wb_dat_o ,
    //wb_err_o ,
    input wb_rst_i ,
```

```
  input wb_stb_i,
  input wb_we_i,
  output wb_inta_o
);


  wire scl_pad_i ;
  wire scl_pad_o ;
  wire scl_padoen_o ;
  wire sda_pad_i ;
  wire sda_pad_o ;
  wire sda_padoen_o ;
  wire arst_i ;
  wire [2:0] temp_wb_adr_i ;


assign temp_wb_adr_i = wb_adr_i;


i2c_master_top i2c_top_inst (
            .wb_clk_i  (wb_clk_i),
            .wb_rst_i  (wb_rst_i),
            .arst_i    (arst_i),
            .wb_adr_i  (temp_wb_adr_i),
```

```
        .wb_dat_i  (wb_dat_i),
        .wb_dat_o  (wb_dat_o),
        .wb_we_i   (wb_we_i),
        .wb_stb_i  (wb_stb_i),
        .wb_cyc_i  (1),
        .wb_ack_o  (wb_ack_o),
        .wb_inta_o (wb_inta_o),

        // i2c lines
        .scl_pad_i     (scl_pad_i),
        .scl_pad_o     (scl_pad_o),
        .scl_padoen_o  (scl_padoen_o),
        .sda_pad_i     (sda_pad_i),
        .sda_pad_o     (sda_pad_o),
        .sda_padoen_o  (sda_padoen_o)
);

assign arst_i = 1'd1;
assign scl_pad_io  =  ((scl_padoen_o) != 1'b0) ? 1'bZ : scl_pad_o;
assign sda_pad_io  =  ((sda_padoen_o) != 1'b0) ? 1'bZ : sda_pad_o;
assign scl_pad_i = scl_pad_io;
assign sda_pad_i = sda_pad_io;
```

endmodule

AUDIO_IF.V

```
//
=============================================================================
=======
// Copyright (c) 2012 by Terasic Technologies Inc.
//
=============================================================================
=======
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use
//   in synthesis for all Terasic Development Boards and Altera Development
//   Kits made by Terasic.  Other use of this code, including the selling
//   ,duplication, or modification of any portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design reference
//   which illustrates how these types of functions can be implemented.
//   It is the user's responsibility to verify their design for
```

```
//   consistency and functionality through the use of formal
//   verification methods.  Terasic provides no warranty regarding the use
//   or functionality of this code.
//
//
//////////////////////////////////////////////////////////////////////////////
//
//  Terasic Technologies Inc
//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
//
//
//
//                     web: http://www.terasic.com/
//                     email: support@terasic.com
//
//
//////////////////////////////////////////////////////////////////////////////

/*

Function:
```

WOLFSON WM8731 controller

I2C Configuration Requirements:

Master Mode

I2S, 16-bits

Clock:

18.432MHz to XTI/MCLK pin of WM8731

Revision:

1.0, 10/22/2007, Init by Richard

Compatibility:

Quartus 7.2

*/

//`include "./AUDIO_ADC.v"

//`include "./AUDIO_DAC.v"

//`include "./audio_fifo.v"

module AUDIO_IF(

```
        avs_s1_clk,
        avs_s1_reset,
        avs_s1_address,
        avs_s1_read,
        avs_s1_readdata,
        avs_s1_write,
        avs_s1_writedata,
        //
        avs_s1_export_BCLK,
        avs_s1_export_DACLRC,
        avs_s1_export_DACDAT,
        avs_s1_export_ADCLRC,
        avs_s1_export_ADCDAT,
        avs_s1_export_XCK
);

/*****************************************************************************
 *                   Constant Declarations                   *

*****************************************************************************/
`define DAC_LFIFO_ADDR 0
`define DAC_RFIFO_ADDR 1
`define ADC_LFIFO_ADDR 2
```

```
`define ADC_RFIFO_ADDR 3
`define CMD_ADDR       4
`define STATUS_ADDR    5


/*****************************************************************************
 *                      Port Declarations                    *
*****************************************************************************/
input                                   avs_s1_clk;
input                                   avs_s1_reset;
input           [2:0]                   avs_s1_address;
input                                   avs_s1_read;
output          [15:0]                  avs_s1_readdata;
input                                   avs_s1_write;
input           [15:0]                  avs_s1_writedata;
        //
input                                   avs_s1_export_BCLK;
input                                   avs_s1_export_DACLRC;
output                                  avs_s1_export_DACDAT;
input                                   avs_s1_export_ADCLRC;
input                                   avs_s1_export_ADCDAT;
output                                  avs_s1_export_XCK;
```

```
/*****************************************************************************
 *              Internal wires and registers Declarations                    *

*****************************************************************************/
// host
reg             [15:0]                  reg_readdata;
reg                                     fifo_clear;

// dac
wire                                    dacfifo_full;
reg                                     dacfifo_write;
reg     [31:0]                  dacfifo_writedata;

// adc
wire                                    adcfifo_empty;
reg                                     adcfifo_read;
wire    [31:0]                  adcfifo_readdata;
reg             [31:0]                  data32_from_adcfifo;
reg             [31:0]                  data32_from_adcfifo_2;
```

```
/*****************************************************************************
 *                    Sequential logic                                       *
*****************************************************************************/

////////// fifo clear
always @ (posedge avs_s1_clk)
begin
        if (avs_s1_reset)
                fifo_clear <= 1'b0;
        else if (avs_s1_write && (avs_s1_address == `CMD_ADDR))
                fifo_clear <= avs_s1_writedata[0];
        else if (fifo_clear)
                fifo_clear <= 1'b0;
end



////////// write audio data(left&right) to dac-fifo
always @ (posedge avs_s1_clk)
begin
        if (avs_s1_reset || fifo_clear)
        begin
```

```
            dacfifo_write <= 1'b0;
        end

        else if (avs_s1_write && (avs_s1_address == `DAC_LFIFO_ADDR))
        begin
            dacfifo_writedata[31:16] <= avs_s1_writedata;
            dacfifo_write <= 1'b0;
        end
        else if (avs_s1_write && (avs_s1_address == `DAC_RFIFO_ADDR))
        begin
            dacfifo_writedata[15:0] <= avs_s1_writedata;
            dacfifo_write <= 1'b1;
        end
        else
            dacfifo_write <= 1'b0;
end


////////// response data to avalon-mm
always @ (negedge avs_s1_clk)
begin
        if (avs_s1_reset || fifo_clear)
            data32_from_adcfifo = 0;
```

```
        else if (avs_s1_read && (avs_s1_address == `STATUS_ADDR))
                reg_readdata <= {adcfifo_empty, dacfifo_full};
        else if (avs_s1_read && (avs_s1_address == `ADC_LFIFO_ADDR))
                reg_readdata <= data32_from_adcfifo[31:16];
        else if (avs_s1_read && (avs_s1_address == `ADC_RFIFO_ADDR))
        begin
                reg_readdata <= data32_from_adcfifo[15:0];
                data32_from_adcfifo <= data32_from_adcfifo_2;
        end
end

////////// read audio data from adc fifo
always @ (negedge avs_s1_clk)
begin
        if (avs_s1_reset)
        begin
                adcfifo_read <= 1'b0;
                data32_from_adcfifo_2 <= 0;
        end
        else if ((avs_s1_address == `ADC_LFIFO_ADDR) & avs_s1_read & ~adcfifo_empty)
        begin
                adcfifo_read <= 1'b1;
        end
```

```
        else if (adcfifo_read)
        begin
                data32_from_adcfifo_2 = adcfifo_readdata;
                adcfifo_read <= 1'b0;
        end
end

/******************************************************************************
 *                      Combinational logic                       *
 ******************************************************************************/
assign  avs_s1_readdata = reg_readdata;
assign  avs_s1_export_XCK = avs_s1_clk;

/******************************************************************************
 *                      Internal Modules                          *
 ******************************************************************************/
```

```
AUDIO_DAC DAC_Instance(
        // host
        .clk(avs_s1_clk),
        .reset(avs_s1_reset),
        .write(dacfifo_write),
        .writedata(dacfifo_writedata),
        .full(dacfifo_full),
        .clear(fifo_clear),
        // dac
        .bclk(avs_s1_export_BCLK),
        .daclrc(avs_s1_export_DACLRC),
        .dacdat(avs_s1_export_DACDAT)
);


AUDIO_ADC ADC_Instance(
        // host
        .clk(avs_s1_clk),
        .reset(avs_s1_reset),
        .read(adcfifo_read),
        .readdata(adcfifo_readdata),
        .empty(adcfifo_empty),
        .clear(fifo_clear),
```

```
        // dac
        .bclk(avs_s1_export_BCLK),
        .adclrc(avs_s1_export_ADCLRC),
        .adcdat(avs_s1_export_ADCDAT)
);


defparam
        DAC_Instance.DATA_WIDTH = 32;

defparam
        ADC_Instance.DATA_WIDTH = 32;


endmodule
```

b. Software

```
#include <stdio.h>
#include <memory.h>
#include <unistd.h>
#include "fbputchar.h"
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
```

```c
#include <arpa/inet.h>
#include <unistd.h>
#include "usbkeyboard.h"
#include <pthread.h>
#include <time.h>
#include "vga_ball.h"
#include <sys/mman.h>
#include <dirent.h>
#include <inttypes.h>
#include <stdbool.h>
#include <sys/ioctl.h>
#include <math.h>

/*
#include "hwlib.h"
#include "socal/hps.h"
#include "socal/socal.h"
#include "hps_0.h"
#include "audio_control.h"
#include "pcm.h"*/


#define SERVER_HOST "128.59.19.114"
#define SERVER_PORT 42000
#define BUFFER_SIZE 128

#define HW_REGS_BASE (ALT_STM_OFST)
#define HW_REGS_SPAN (0x04000000)
#define HW_REGS_MASK (HW_REGS_SPAN - 1)

//base addr
static volatile unsigned long *h2p_lw_axi_addr = NULL;

volatile unsigned long *oc_i2c_audio_addr = NULL;
volatile unsigned long *audio_addr = NULL;

int vga_ball_fd;
int sockfd; /* Socket file descriptor */
struct libusb_device_handle *keyboard;
uint8_t endpoint_address;
pthread_t network_thread;
void *network_thread_f(void *);
```

```c
clock_t start, end;
double cpu_time_used;


void update_array(int* arr, int size){
   for (int i = 0; i < size; i++) {
       if (arr[i] < 1500){
           arr[i]++;
           arr[i]++;
       }
       if (arr[i] == 1294 || arr[i] == 1293){
           arr[i] = 1500;
       }
       //printf("%d ", arr[i]);
   }
}

void set_ball_position(const note_position_t *c)
//------------------add-----------------
{
 note_pos_t bpvla;
 bpvla.position = *c;
 if (ioctl(vga_ball_fd, VGA_BALL_WRITE_POSITION, &bpvla)) {
     perror("ioctl(VGA_BALL_WRITE_POSITION) failed");
     return;
 }
}

void set_ball_status(const node_status_t *c)
//------------------add-----------------
{
 node_sta_t status_new;
 status_new.status = *c;
 if (ioctl(vga_ball_fd, VGA_BALL_WRITE_STATUS, &status_new)) {
     perror("ioctl(VGA_BALL_WRITE_STATUS) failed");
     return;
 }
}

void set_background(const background_t *c)  //------------------add-----------------
{
 back_t background_new;
```

```
 background_new.background = *c;
 if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &background_new)) {
     perror("ioctl(VGA_BALL_WRITE_BACKGROUND) failed");
     return;
 }
}

void set_ball_score(const total_score_t *c)  //------------------add-----------------
{
 total_sc_t score_new;
 score_new.total_score = *c;
 if (ioctl(vga_ball_fd, VGA_BALL_WRITE_SCORE, &score_new)) {
     perror("ioctl(VGA_BALL_WRITE_SCORE) failed");
     return;
 }
}



int main(){


   static const char filename[] = "/dev/vga_ball";
   printf("VGA ball Userspace program started\n");
   if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
       fprintf(stderr, "could not open %s\n", filename);
       return -1;
   }
   printf("initial state: ");


   if ( (keyboard = openkeyboard(&endpoint_address)) == NULL ) {
           fprintf(stderr, "Did not find a keyboard\n");
           exit(1);
       }

   float track_1[163] = {7.19, 7.72, 8.25, 8.78, 9.64, 10.69, 14.06, 17.75, 18.28,
19.87, 23.5, 23.89, 24.36, 25.87, 27.99, 30.43, 30.96, 33.13, 35.18, 36.44, 36.57,
36.96, 39.41, 42.05, 43.5, 44.03, 45.21, 45.74, 47.72, 48.38, 48.65, 51.95, 52.47,
52.74, 53.66, 54.19, 54.72, 54.98, 56.17, 57.56, 58.15, 59.47, 59.73, 60.0, 61.25,
61.52, 61.85, 63.17, 65.81, 66.07, 66.34, 67.26, 68.12, 68.45, 70.03, 70.56, 71.55,
73.27, 73.73, 75.58, 76.17, 78.75, 79.21, 80.53, 81.19, 82.44, 82.7, 82.97, 85.61,
86.27, 86.67, 87.99, 88.38, 88.78, 89.83, 90.63, 90.89, 91.95, 92.67, 94.26, 94.65,
```

```
95.05, 95.91, 96.17, 98.02, 98.28, 98.55, 100.39, 101.12, 102.24, 102.38, 102.51,
102.64, 103.56, 105.28, 107.0, 107.85, 109.04, 110.69, 113.2, 114.92, 117.29, 118.02,
122.24, 124.16, 124.42, 124.69, 126.07, 126.86, 128.38, 128.65, 129.31, 129.57,
133.33, 133.6, 133.86, 136.76, 137.03, 137.82, 143.37, 145.28, 145.54, 145.81, 146.93,
150.23, 150.49, 150.76, 153.46, 154.72, 154.85, 155.25, 157.69, 160.33, 161.78,
162.31, 163.5, 164.02, 166.0, 166.66, 166.93, 170.23, 170.76, 171.09, 171.94, 172.47,
174.45, 175.77, 176.5, 178.68, 179.21, 180.39, 180.92, 182.9, 183.56, 183.83, 187.13,
187.66, 187.99, 188.83, 189.37, 191.35, 192.67, 193.33};
   float track_2[216] = {6.47, 9.5, 9.77, 12.8, 13.53, 13.79, 14.06, 14.92, 16.7,
17.29, 20.07, 23.3, 25.61, 27.0, 29.83, 31.29, 31.95, 33.27, 34.19, 34.32, 34.45,
34.59, 35.18, 36.7, 37.23, 37.49, 37.82, 38.55, 39.41, 40.46, 40.73, 40.99, 41.25,
41.78, 42.31, 43.37, 44.03, 45.35, 45.74, 47.59, 48.38, 48.65, 49.17, 49.7, 49.97,
50.36, 51.81, 52.47, 52.74, 53.79, 54.19, 54.72, 54.98, 56.04, 56.83, 57.09, 58.41,
58.94, 60.59, 61.52, 63.17, 65.02, 65.28, 65.54, 67.13, 67.52, 70.03, 70.56, 71.35,
71.88, 73.53, 73.93, 74.78, 75.84, 76.63, 77.42, 77.95, 78.48, 78.94, 80.26, 80.92,
82.44, 82.7, 82.97, 85.48, 85.74, 87.46, 87.99, 89.7, 89.97, 91.35, 91.95, 92.67,
94.26, 94.65, 95.05, 96.04, 96.3, 98.15, 98.41, 99.01, 99.27, 100.39, 101.12, 101.78,
101.98, 102.77, 103.56, 104.88, 105.28, 107.0, 107.85, 108.38, 108.91, 109.57, 110.82,
111.22, 111.62, 111.95, 113.07, 113.33, 113.86, 114.12, 115.05, 115.44, 115.97,
116.24, 117.56, 118.61, 118.88, 119.14, 119.4, 120.72, 121.25, 122.24, 124.29, 124.55,
126.07, 126.86, 128.91, 129.7, 130.23, 130.62, 130.96, 133.46, 133.73, 134.45, 134.85,
135.18, 137.29, 137.56, 143.1, 143.63, 145.28, 145.54, 145.81, 146.93, 147.39, 147.99,
150.23, 150.49, 150.76, 151.81, 153.46, 154.98, 155.51, 155.77, 156.04, 156.83,
157.69, 158.74, 159.01, 159.27, 159.54, 160.06, 160.59, 161.65, 162.31, 163.63,
164.02, 165.87, 166.66, 166.93, 167.46, 168.65, 170.1, 170.76, 171.09, 172.08, 172.47,
174.32, 175.11, 175.38, 176.7, 177.23, 178.55, 179.21, 180.53, 180.92, 182.77, 183.56,
183.83, 184.35, 184.88, 185.15, 185.54, 187.0, 187.66, 187.99, 188.97, 189.37, 191.22,
192.01, 192.28, 193.59, 194.06};
   float track_3[216] = {6.47, 9.11, 9.24, 10.36, 11.42, 12.01, 12.47, 13.66, 14.92,
15.78, 16.7, 17.29, 18.88, 19.54, 22.97, 25.35, 26.2, 26.73, 27.19, 27.72, 28.51,
29.17, 29.31, 29.64, 31.42, 32.21, 32.47, 32.74, 33.6, 34.19, 34.32, 34.45, 34.59,
35.18, 36.7, 37.23, 37.49, 38.02, 38.35, 38.94, 39.87, 40.13, 41.78, 42.31, 43.23,
43.63, 44.42, 45.48, 46.14, 46.53, 47.46, 47.85, 49.17, 50.36, 51.68, 52.01, 53.93,
55.91, 56.3, 56.83, 57.09, 58.41, 58.94, 60.59, 60.86, 62.9, 63.43, 69.24, 69.5,
69.77, 70.3, 70.82, 74.19, 74.78, 75.84, 76.63, 77.42, 77.95, 78.48, 80.06, 83.23,
83.5, 84.22, 85.34, 85.87, 87.59, 87.85, 89.57, 90.1, 91.35, 92.28, 93.86, 94.12,
96.43, 96.96, 97.23, 98.15, 98.41, 99.01, 99.27, 100.72, 101.78, 101.98, 102.77,
103.17, 104.88, 105.61, 106.47, 106.6, 106.73, 106.86, 107.59, 108.38, 110.96, 111.22,
111.62, 111.95, 112.94, 113.33, 113.86, 114.12, 115.18, 115.44, 115.97, 116.24,
118.61, 118.88, 119.14, 119.4, 120.06, 120.33, 120.99, 121.85, 122.64, 126.4, 128.91,
129.7, 130.23, 130.62, 130.96, 132.67, 132.94, 134.45, 134.85, 135.18, 138.88, 139.4,
141.78, 142.04, 144.16, 144.69, 146.07, 146.34, 146.66, 147.72, 149.44, 149.7, 149.97,
```

```
151.55, 152.08, 152.6, 153.46, 154.98, 155.51, 155.77, 156.3, 156.63, 157.16, 158.08,
158.35, 160.06, 160.59, 161.52, 161.91, 162.7, 163.76, 164.42, 164.82, 165.74, 166.14,
167.46, 167.99, 168.25, 168.65, 169.97, 170.36, 172.21, 173.0, 173.26, 174.19, 174.59,
175.11, 175.38, 176.7, 177.23, 178.41, 178.81, 179.6, 180.66, 181.25, 181.65, 182.64,
183.03, 184.35, 185.54, 186.86, 187.26, 189.11, 189.83, 190.1, 191.09, 191.48, 192.01,
192.28, 193.59, 194.06};
   float track_4[167] = {7.19, 7.72, 8.25, 8.78, 10.36, 11.42, 12.47, 17.75, 18.28,
19.01, 19.27, 20.33, 20.86, 21.39, 21.91, 22.44, 22.77, 24.62, 25.08, 26.2, 26.73,
27.46, 28.32, 30.43, 30.96, 31.68, 33.77, 35.18, 36.44, 36.57, 36.96, 39.73, 40.0,
40.26, 42.05, 43.1, 43.63, 44.42, 45.61, 46.14, 46.53, 47.33, 47.85, 51.55, 52.01,
54.06, 55.77, 56.3, 57.56, 58.15, 59.47, 59.73, 60.0, 60.99, 62.9, 63.43, 69.24, 69.5,
69.77, 70.3, 70.82, 73.27, 74.45, 75.58, 76.17, 79.73, 83.23, 83.5, 84.03, 84.49,
85.87, 86.27, 86.67, 87.72, 88.38, 88.78, 90.1, 90.63, 90.89, 92.28, 93.73, 93.99,
96.43, 96.96, 97.23, 98.02, 98.28, 98.55, 100.72, 103.17, 104.36, 104.49, 104.62,
104.75, 105.61, 107.59, 111.09, 112.8, 115.31, 119.93, 120.2, 120.46, 121.85, 122.64,
126.4, 128.51, 128.78, 129.17, 129.44, 132.54, 132.8, 133.07, 139.14, 140.99, 141.25,
141.52, 142.31, 144.42, 146.07, 146.34, 146.66, 149.44, 149.7, 149.97, 152.6, 153.46,
154.72, 154.85, 155.25, 157.95, 158.22, 158.48, 160.33, 161.38, 161.91, 162.7, 163.89,
164.42, 164.82, 165.61, 166.14, 169.83, 170.36, 172.34, 173.0, 173.26, 174.06, 174.59,
175.77, 176.5, 178.28, 178.81, 179.6, 180.79, 181.25, 181.65, 182.51, 183.03, 186.73,
187.26, 189.24, 189.83, 190.1, 190.96, 191.48, 192.67, 193.33};

   int score = 18496;

   while(true){

   int location_1[4] ={1500,1500,1500,1500};
   int location_2[4] ={1500,1500,1500,1500};
   int location_3[4] ={1500,1500,1500,1500};
   int location_4[4] ={1500,1500,1500,1500};

   int size_1 = sizeof(track_1)/sizeof(track_1[0]);
   int size_2 = sizeof(track_2)/sizeof(track_2[0]);
   int size_3 = sizeof(track_3)/sizeof(track_3[0]);
   int size_4 = sizeof(track_4)/sizeof(track_4[0]);

   int last_status = 0;
   //keyboard varibale
   struct usb_keyboard_packet packet;
   int transferred;
   char keystate[12];
   int time_arr[4] = {0,0,0,0};
```

```
   int status_arr[4] = {0,0,0,0};

   float total_score = 0.0;

   float note_score = 1000/762;

   total_score_t d ={score};
   set_ball_score(&d);



   start = clock(); // get the start time
   int initial = 0;

   for(int i = 0; i < 600000; i++){
       //location part
       while (true){
           libusb_interrupt_transfer(keyboard, endpoint_address,
                 (unsigned char *) &packet, 5,
                 &transferred, 0);
           if (transferred == 5) {
               sprintf(keystate, "%04x %04x %04x %04x %04x", packet.a, packet.b,
packet.c, packet.d, packet.e);
               //printf("%s\n", keystate);
           }
           background_t c ={initial};
           set_background(&c);
           if(initial == 1){
               break;
           }

           if(packet.b == 1){
               initial = 1;
           }
           usleep(50000);
           //input inital to hardware
       }

       if(i == 0){
           char PCMName[128 + 4];
           //if audio need delay add usleep here
           pthread_create(&network_thread, NULL, network_thread_f, (void *) PCMName);
```

```
        }
        if(i == 0){
            //if logic need delay add usleep here
        }
        float timer = (float)i;
        float true_time = (timer+ 395.0)/200;
        if (i % 200 == 0){
        }

        //printf("%f\n",true_time);
        int remain_v = 0;
        for (int i1 = 0; i1 < size_1; i1++){
            if (true_time == track_1[i1]){
                for(int j1 = 0; j1 <4; j1++){
                    if(location_1[j1] == 1500){
                        location_1[j1] = 0;
                        break;
                    }
                }
            }
        }

        for (int i2 = 0; i2 < size_2; i2++){
            if (true_time == track_2[i2]){
                for(int j2 = 0; j2 <4; j2++){
                    if(location_2[j2] == 1500){
                        location_2[j2] = 0;
                        break;
                    }
                }
            }
        }

        for (int i3 = 0; i3 < size_3; i3++){
            if (true_time == track_3[i3]){
                for(int j3 = 0; j3 <4; j3++){
                    if(location_3[j3] == 1500){
                        location_3[j3] = 0;
                        break;
                    }
                }
            }
```

```
    }

    for (int i4 = 0; i4 < size_4; i4++){
        if (true_time == track_4[i4]){
            for(int j4 = 0; j4 <4; j4++){
                if(location_4[j4] == 1500){
                    location_4[j4] = 0;
                    break;
                }
            }
        }
    }



    // boom
    if (last_status != packet.b){
        if(packet.b - 16 >= 0){
            remain_v = packet.b - 16;
            if(status_arr[3] != 1){
                status_arr[3] = 1;
                for (int i = 0; i<4; i++){
                    if(location_4[i]>=  1072&& location_4[i]<= 1138){

                        status_arr[3] = 2;
                        location_4[i] = 1500;
                        total_score += note_score;
                    }
                    if ((location_4[i]>= 1052 && location_4[i]<1072)){
                        status_arr[3] = 3;
                        location_4[i] = 1500;
                    }

              }
          }
      }
      else{
            remain_v = packet.b;
            status_arr[3] = 0;

      }
      if(remain_v - 8 >= 0){
          remain_v = remain_v - 8;
```

```
            if(status_arr[2] != 1){
                status_arr[2] = 1;
                for (int i = 0; i<4; i++){
                    if(location_3[i]>= 1072 && location_3[i]<= 1138){

                        status_arr[2] = 2;
                        location_3[i] = 1500;
                        total_score += note_score;
                    }
                    if ((location_3[i]>=1052 && location_3[i]<1072)){
                        status_arr[2] = 3;
                        location_3[i] = 1500;
                    }
                }
            }
        }
        else{
          status_arr[2] = 0;
        }



        if(remain_v - 4 >= 0){
            remain_v = remain_v - 4;
            if(status_arr[1] != 1){
                status_arr[1] = 1;
                for (int i = 0; i<4; i++){
                    if(location_2[i]>=1072 && location_2[i]<=1138){
                        status_arr[1] = 2;
                        location_2[i] = 1500;
                        total_score += note_score;
                    }
                    if ((location_2[i]>=1052 && location_2[i]<1072)){

                        status_arr[1] = 3;
                        location_2[i] = 1500;



                    }
                }
            }
        }
```

```
        else{
          status_arr[1] = 0;
        }
        if(remain_v - 2 >= 0){
            remain_v = remain_v - 8;
            if(status_arr[0] != 1){
                status_arr[0] = 1;
                for (int i = 0; i<4; i++){
                    if(location_1[i]>=1072 && location_1[i]<=1138){

                        status_arr[0] = 2;
                        location_1[i] = 1500;
                        total_score += note_score;
                    }
                    if ((location_1[i]>=1052 && location_1[i]<1072)){

                        status_arr[0] = 3;
                        location_1[i] = 1500;
                    }
                }
            }
        }
        else{
          status_arr[0] = 0;
        }
      }

      for (int i = 0; i<4; i++){
          if (location_1[i] >= 1138 && location_1[i] <= 1154 && status_arr[0] == 0){
              status_arr[0] = 4;
              time_arr[0] = 1;
          }
          if (location_2[i] >= 1138 && location_2[i] <= 1154 && status_arr[1] == 0){
              status_arr[1] = 4;
              time_arr[1] = 1;
          }

          if (location_3[i] >= 1138 && location_3[i] <= 1154 && status_arr[2] == 0){
              status_arr[2] = 4;
              time_arr[2] = 1;
          }
```

```
            if (location_4[i] >= 1138 && location_4[i] <= 1154 && status_arr[3] == 0){
                status_arr[3] = 4;
                time_arr[3] = 1;
            }
        }
        //printf("%d %d %d %d\n",
status_arr[0],status_arr[1],status_arr[2],status_arr[3]);

        last_status = packet.b;

        for(int i =0 ; i <4; i++){
            if (time_arr[i] == 20){
                time_arr[i] = 0;
                /*if (status_arr[i] == 4){
                   status_arr[i] = 0;
                }*/
                status_arr[i] = 0;
            }
            if (time_arr[i] > 0 ){
                time_arr[i]++;
            }
        }

        note_position_t a ={location_4[0],location_3[0],location_2[0],location_1[0],\

location_4[1],location_3[1],location_2[1],location_1[1],\

location_4[2],location_3[2],location_2[2],location_1[2],\

location_4[3],location_3[3],location_2[3],location_1[3]};

        node_status_t b ={status_arr[0],status_arr[1],status_arr[2],status_arr[3]};


        set_ball_position(&a);
        set_ball_status(&b);



        //printf("%d\n",i);
        update_array(location_1,4);
        //printf("=======");
```

```
        update_array(location_2,4);
        //printf("=======");
        update_array(location_3,4);
        //printf("========");
        update_array(location_4,4);
        //printf("\n");

         end = clock(); // get the end time
        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

        if (cpu_time_used < 0.005) {
            // wait for the remaining time
            double remaining_time = 0.00421 - cpu_time_used;


            usleep(remaining_time * 1000000);
        }

        start = clock(); // update the start time for the next iteration
        if(i > 42200){
            pthread_cancel(network_thread);
            //pthread_join(network_thread,NULL);


            break;
        }

    }
    pthread_join(network_thread,NULL);
    score = (int)roundf(total_score);
    printf("%d\n",score);
    int first_decimal = score / 1000 << 12;
    int second_decimal = (score % 1000 ) / 100  << 8;
    int third_decimal = (score % 100 ) / 10 <<4;
    int fourth_decimal =(score % 10);
    score = first_decimal + second_decimal + third_decimal + fourth_decimal;
    printf("%d\n",score);
    //usleep(10000000);
    }
    return 0;
}
```

```
void *network_thread_f(void *arg)
{
 /*char *PCMName = (char *) arg;
 printf("[INFO] Now playing ...\n");
 play_PCM(PCMName);
 printf("[INFO] Music over, quitting...\n");*/
 char cmd[512];
 sprintf(cmd, "./MyPlayer nggyu.MP3");
 if(system(cmd) !=0 )
 {
   return 1;
 }
 for(int i = 0; i <10000; i++){
   if (i %200 == 0){
       printf("%d",i);
   }
 }

 return NULL;
}
```

Codes for custom controller:

usbkeyboard.c:

```
#include "usbkeyboard.h"

#include <stdio.h>
#include <stdlib.h>

/* References on libusb 1.0 and the USB HID/keyboard protocol
*
* http://libusb.org
* http://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb-10/
* http://www.usb.org/developers/devclass_docs/HID1_11.pdf
* http://www.usb.org/developers/devclass_docs/Hut1_11.pdf
*/
```

```
/*
* Find and return a USB keyboard device or NULL if not found
* The argument con
*
*/
struct libusb_device_handle *openkeyboard(uint8_t *endpoint_address) {
 libusb_device **devs;
 struct libusb_device_handle *keyboard = NULL;
 struct libusb_device_descriptor desc;
 ssize_t num_devs, d;
 uint8_t i, k;
  /* Start the library */
 if ( libusb_init(NULL) < 0 ) {
   fprintf(stderr, "Error: libusb_init failed\n");
   exit(1);
 }

 /* Enumerate all the attached USB devices */
 if ( (num_devs = libusb_get_device_list(NULL, &devs)) < 0 ) {
   fprintf(stderr, "Error: libusb_get_device_list failed\n");
   exit(1);
 }

 /* Look at each device, remembering the first HID device that speaks
    the keyboard protocol */

 for (d = 0 ; d < num_devs ; d++) {
   libusb_device *dev = devs[d];
   if ( libusb_get_device_descriptor(dev, &desc) < 0 ) {
     fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
     exit(1);
   }

   if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
     struct libusb_config_descriptor *config;
     libusb_get_config_descriptor(dev, 0, &config);

     for (i = 0 ; i < config->bNumInterfaces ; i++)
       for ( k = 0 ; k < config->interface[i].num_altsetting ; k++ ) {
         const struct libusb_interface_descriptor *inter =
config->interface[i].altsetting + k ;
         printf("%6d\n", inter->bInterfaceProtocol);
```

```
        printf("%6d class\n", inter->bInterfaceClass);
        if ( inter->bInterfaceClass == LIBUSB_CLASS_HID &&
             inter->iInterface == 4) {
             printf("ergtdg\n");
          int r;
          if ((r = libusb_open(dev, &keyboard)) != 0) {
            fprintf(stderr, "Error: libusb_open failed: %d\n", r);
            exit(1);
          }

          if (libusb_kernel_driver_active(keyboard,i))
            libusb_detach_kernel_driver(keyboard, i);
          libusb_set_auto_detach_kernel_driver(keyboard, i);

          if ((r = libusb_claim_interface(keyboard, i)) != 0) {
            fprintf(stderr, "Error: libusb_claim_interface failed: %d\n", r);
            exit(1);
          }
          *endpoint_address = inter->endpoint[0].bEndpointAddress;
          goto found;
        }
      }
   }
 }

found:
 libusb_free_device_list(devs, 1);

 return keyboard;
}
```

usbkeyboard.h:

```
#ifndef _USBKEYBOARD_H
#define _USBKEYBOARD_H

#include <libusb-1.0/libusb.h>

#define USB_HID_KEYBOARD_PROTOCOL 1
```

```
/* Modifier bits */
#define USB_LCTRL  (1 << 0)
#define USB_LSHIFT (1 << 1)
#define USB_LALT   (1 << 2)
#define USB_LGUI   (1 << 3)
#define USB_RCTRL  (1 << 4)
#define USB_RSHIFT (1 << 5)
#define USB_RALT   (1 << 6)
#define USB_RGUI   (1 << 7)

struct usb_keyboard_packet {
 uint16_t a;
 uint16_t b;
 uint16_t c;
 uint16_t d;
 uint16_t e;

};

/* Find and open a USB keyboard device.  Argument should point to
  space to store an endpoint address.  Returns NULL if no keyboard
  device was found. */
extern struct libusb_device_handle *openkeyboard(uint8_t *);

#endif
```

c. Audio software**:**

audio_control.c

```
#include <stdio.h>

#include "audio_control.h"
#include "hwlib.h"
#include "socal/socal.h"
#include "socal/hps.h"
```

```c
#define AUDIO_DAC_LFIFO_PORT 0
#define AUDIO_DAC_RFIFO_PORT 1
#define AUDIO_ADC_LFIFO_PORT 2
#define AUDIO_ADC_RFIFO_PORT 3
#define AUDIO_CMD_PORT 4
#define AUDIO_STATUS_PORT 5

#define MASK_STATUS_DAC_FULL 0x01
#define MASK_STATUS_ADC_EMPTY 0x02

#define LINEOUT_DEFUALT_VOL 0x79 // 0 dB

extern volatile unsigned long *oc_i2c_audio_addr;
extern volatile unsigned long *audio_addr;

//i2c initial
void oc_i2c_audio_init(void)
{
    uint32_t this_data;
    //clock prescale register... set the frequency 100KHz for I2C (
50M/(5*100K) = 99)
    alt_write_word(oc_i2c_audio_addr, 0x16);
    alt_write_word(oc_i2c_audio_addr + 1, 0x00);
    //oc_i2c_audio_addr[1] = 0x00;

    //enable the I2C core, but disable the IRQ
    oc_i2c_audio_addr[2] = 0x80;

    //this_data = oc_i2c_audio_addr[0];
    this_data = alt_read_word(oc_i2c_audio_addr);
    if ((this_data & 0x00ff) == 0x19)
    {
        printf("[INFO] Prescale low byte set is success! \n");
        this_data = alt_read_word(oc_i2c_audio_addr + 1);
        if ((this_data & 0x00ff) == 0x00)
            printf("[INFO] Prescale high byte set is success! \n");
        else
            printf("[INFO] Prescale high byte set is NG! \n");
    }
    else
```

```
        printf("[INFO] Prescale low byte set is NG! \n");

    this_data = oc_i2c_audio_addr[2];
    if ((this_data & 0x00ff) == 0x80)
        printf("[INFO] I2C core is enabled! \n");
    else
        printf("[INFO] I2C core is not enabled! \n");
}

//audio i2c write reg
//if dev_addr 7bit+write
int oc_i2c_audio_wr_reg(int reg, int data)
{
    uint32_t this_data;
    uint32_t reg_data = 0x0;
    int bsucces = 0;

    //set the tx reg audio chip dev address with write bit
    //oc_i2c_audio_addr[3] = 0x34;
    alt_write_word(oc_i2c_audio_addr + 3, 0x34);
    //set STA and WR bits(bit7 and bit4)
    //oc_i2c_audio_addr[4] = 0x90;
    alt_write_word(oc_i2c_audio_addr + 4, 0x90);
    //oc_i2c_audio_addr[4] = 0x10;
    //wait TIP bit go to 0 to end Tx
    this_data = alt_read_word(oc_i2c_audio_addr + 4);
    while (this_data & 0x02)
    {
        this_data = alt_read_word(oc_i2c_audio_addr + 4);
    }
    //wait the rx ACK signal 0-valid
    this_data = alt_read_word(oc_i2c_audio_addr + 4);
    while (this_data & 0x80)
    {
        this_data = alt_read_word(oc_i2c_audio_addr + 4);
    }
    //   printf("\n receive ACK-device address! \n");

    //set the txr reg data with reg address + 1 data MSB
    reg_data = (reg << 1) & 0xFE;
```

```
    reg_data |= ((data >> 8) & 0x01);
    // oc_i2c_audio_addr[3] = (reg_data & 0xff);
    alt_write_word(oc_i2c_audio_addr + 3, reg_data & 0xff);
    //set WR bits(bit4)
    // oc_i2c_audio_addr[4] = 0x10;
    alt_write_word(oc_i2c_audio_addr + 4, 0x10);
    //wait TIP bit go to 0 to end Tx
    this_data = alt_read_word(oc_i2c_audio_addr + 4);
    while (this_data & 0x02)
    {
        this_data = alt_read_word(oc_i2c_audio_addr + 4);
    }
    //wait the rx ACK signal 0-valid
    this_data = alt_read_word(oc_i2c_audio_addr + 4);
    while (this_data & 0x80)
    {
        this_data = alt_read_word(oc_i2c_audio_addr + 4);
    }
    //   printf("\n receive ACK-reg address! \n");

    //set the txr reg data with the other data 8 bit LSB
    //oc_i2c_audio_addr[3] = (data & 0xff);
    alt_write_word(oc_i2c_audio_addr + 3, data & 0xff);
    //set STO and WR bits(bit7 and bit4)
    //oc_i2c_audio_addr[4] = 0x10;
    alt_write_word(oc_i2c_audio_addr + 4, 0x10);
    //wait TIP bit go to 0 to end Tx
    this_data = alt_read_word(oc_i2c_audio_addr + 4);
    while (this_data & 0x02)
    {
        this_data = alt_read_word(oc_i2c_audio_addr + 4);
    }
    //wait the rx ACK signal 0-valid
    this_data = alt_read_word(oc_i2c_audio_addr + 4);
    while (this_data & 0x80)
    {
        this_data = alt_read_word(oc_i2c_audio_addr + 4);
    }
    //oc_i2c_audio_addr[4] = 0x40;
    alt_write_word(oc_i2c_audio_addr + 4, 0x40);
```

```
    // printf("[INFO] Wr_reg receive ACK-data! \n");
    bsucces = 1;

    return bsucces;
}

int oc_i2c_audio_rd_reg(int reg)
{
    uint32_t this_data;
    uint32_t reg_data = 0x0;
    uint32_t data = 0x0;
    int bsucces = 0;

    //set the tx reg audio chip dev address with write bit
    // oc_i2c_audio_addr[3] = 0x34;
    alt_write_word(oc_i2c_audio_addr + 3, 0x34);

    //set STA and WR bits(bit7 and bit4)
    //oc_i2c_audio_addr[4] = 0x90;
    alt_write_word(oc_i2c_audio_addr + 4, 0x90);
    //oc_i2c_audio_addr[4] = 0x10;
    //wait TIP bit go to 0 to end Tx
    this_data = alt_read_word(oc_i2c_audio_addr + 4);
    while (this_data & 0x02)
    {
        this_data = alt_read_word(oc_i2c_audio_addr + 4);
    }
    //wait the rx ACK signal 0-valid
    this_data = alt_read_word(oc_i2c_audio_addr + 4);
    while (this_data & 0x80)
    {
        this_data = alt_read_word(oc_i2c_audio_addr + 4);
    }
    // printf("\n read receive ACK-device address! \n");

    //set the txr reg data with reg address + 0
    reg_data = (reg << 1) & 0xFE;
    oc_i2c_audio_addr[3] = (reg_data & 0xff);
    //set WR bits(bit4)
```

```
    //oc_i2c_audio_addr[4] = 0x10;
    alt_write_word(oc_i2c_audio_addr + 4, 0x10);
    //wait TIP bit go to 0 to end Tx
    this_data = alt_read_word(oc_i2c_audio_addr + 4);
    while (this_data & 0x02)
    {
        this_data = alt_read_word(oc_i2c_audio_addr + 4);
    }
    //wait the rx ACK signal 0-valid
    this_data = alt_read_word(oc_i2c_audio_addr + 4);
    while (this_data & 0x80)
    {
        this_data = alt_read_word(oc_i2c_audio_addr + 4);
    }
    //  printf("\n read receive ACK-reg address! \n");

    //read
    //set the tx reg audio chip dev address with read bit 1
    //oc_i2c_audio_addr[3] = 0x35;
    alt_write_word(oc_i2c_audio_addr + 3, 0x35);

    //set STA and WR bits(bit7 and bit4)
    //oc_i2c_audio_addr[4] = 0x90;
    alt_write_word(oc_i2c_audio_addr + 4, 0x90);
    // oc_i2c_audio_addr[4] = 0x10;
    //wait TIP bit go to 0 to end Tx
    this_data = alt_read_word(oc_i2c_audio_addr + 4);
    while (this_data & 0x02)
    {
        this_data = alt_read_word(oc_i2c_audio_addr + 4);
    }
    //wait the rx ACK signal 0-valid
    this_data = alt_read_word(oc_i2c_audio_addr + 4);
    while (this_data & 0x80)
    {
        this_data = alt_read_word(oc_i2c_audio_addr + 4);
    }
    //   printf("\n read receive ACK-device address(read)! \n");

    //set the RD and ACK bit(bit5 and bit3)
```

```
    //oc_i2c_audio_addr[4] = 0x20;
    alt_write_word(oc_i2c_audio_addr + 4, 0x20);
    //wait TIP bit go to 0 to end Tx
    this_data = alt_read_word(oc_i2c_audio_addr + 4);
    while (this_data & 0x02)
    {
        this_data = alt_read_word(oc_i2c_audio_addr + 4);
    }

    // printf("\n read receive ACK-device address(read)! \n");

    // oc_i2c_audio_addr[4] = 0x00;
    //read the rxr data
    data = alt_read_word(oc_i2c_audio_addr + 3) & 0xff;

    // set the STO ,RD and NACK bit(bit 6,bit5 and bit3)
    oc_i2c_audio_addr[4] = 0x28;
    // alt_write_word(oc_i2c_audio_addr+4, 0x20);

    //wait TIP bit go to 0 to end Tx
    this_data = alt_read_word(oc_i2c_audio_addr + 4);
    while (this_data & 0x02)
    {
        this_data = alt_read_word(oc_i2c_audio_addr + 4);
    }
    // printf("\n read receive ACK-device address(read)! \n");
    //oc_i2c_audio_addr[4] = 0x04;

    //alt_write_word(oc_i2c_audio_addr+4, 0x08);
    // alt_write_word(oc_i2c_audio_addr+4, 0x08);
    // alt_write_word(oc_i2c_audio_addr+4, 0x40);

    //oc_i2c_audio_addr[4] = 0x40;

    //read the rxr data
    this_data = (alt_read_word(oc_i2c_audio_addr + 3)) & 0xff;
    // printf("this data = %Xh\r\n", this_data);
    data |= ((this_data << 8) & 0x100);

    alt_write_word(oc_i2c_audio_addr + 4, 0x40);
```

```
    printf("[INFO] read audio reg[%02d] = %04Xh\r\n", reg, data);

    bsucces = 1;

    return bsucces;
}

int AUDIO_InterfaceActive(int bActive)
{
    int bSuccess;
    bSuccess = oc_i2c_audio_wr_reg(9, bActive ? 0x0001 : 0x0000);
    printf("[INFO] AUDIO_InterfaceActive... %s\r\n", bSuccess ? "success"
: "fail");
    return bSuccess;
}

int AUDIO_MicBoost(int bBoost)
{
    int bSuccess;
    uint32_t control;
    control = 0x0014;
    if (bBoost)
        control |= 0x0001;
    else
        control &= 0xFFFE;
    bSuccess = oc_i2c_audio_wr_reg(0b0000100, control); // Left Line In:
set left line in volume
    printf("[INFO] AUDIO_MicBoost... %s\r\n", bSuccess ? "success" :
"fail");
    return bSuccess;
}

int AUDIO_AdcEnableHighPassFilter(int bEnable)
{
    int bSuccess;
    uint32_t control;
    control = 0x0000;
    if (bEnable)
        control &= 0xFFFE;
```

```
    else
        control |= 0x0001;
    bSuccess = oc_i2c_audio_wr_reg(5, control); // Left Line In: set left
line in volume

    printf("[INFO] AUDIO_AdcEnableHighPassFilter... %s\r\n", bSuccess ?
"success" : "fail");
    return bSuccess;
}

int AUDIO_DacDeemphasisControl(int deemphasis_type)
{
    int bSuccess;
    uint32_t control;
    control = 0x0000;
    control &= 0xFFF9;
    switch (deemphasis_type)
    {
    case DEEMPHASIS_48K:
        control |= ((0x03) << 1);
        break;
    case DEEMPHASIS_44K1:
        control |= ((0x02) << 1);
        break;
    case DEEMPHASIS_32K:
        control |= ((0x01) << 1);
        break;
    }
    bSuccess = oc_i2c_audio_wr_reg(5, control); // Left Line In: set left
line in volume
    printf("[INFO] AUDIO_DacDeemphasisControl... %s\r\n", bSuccess ?
"success" : "fail");

    return bSuccess;
}

int AUDIO_DacEnableSoftMute(int bEnable)
{
    int bSuccess;
    uint32_t control;
```

```
    uint32_t mask;
    control = 0x0000;
    mask = 0x01 << 3;
    if (bEnable)
        control |= mask;
    else
        control &= ~mask;
    bSuccess = oc_i2c_audio_wr_reg(5, control); // Left Line In: set left
line in volume
    printf("[INFO] AUDIO_DacEnableSoftMute... %s\r\n", bSuccess ?
"success" : "fail");

    return bSuccess;
}

int AUDIO_MicMute(int bMute)
{
    int bSuccess;
    uint32_t control;
    uint32_t mask;
    control = 0x014;
    mask = 0x01 << 1;
    if (bMute)
        control |= mask;
    else
        control &= ~mask;
    bSuccess = oc_i2c_audio_wr_reg(4, control); // Left Line In: set left
line in volume
    printf("[INFO] AUDIO_MicMute... %s\r\n", bSuccess ? "success" :
"fail");
    return bSuccess;
}

int AUDIO_LineInMute(int bMute)
{
    int bSuccess;
    uint32_t control_l, control_r;
    uint32_t mask;
    control_l = 0x0017;
    control_r = 0x0017;
```

```
    mask = 0x01 << 7;
    if (bMute)
    {
        control_l |= mask;
        control_r |= mask;
    }
    else
    {
        control_l &= ~mask;
        control_r &= ~mask;
    }
    bSuccess = oc_i2c_audio_wr_reg(0, control_l); // Left Line In: set
left line in volume

    printf("[INFO] AUDIO_LineInMute...\r\n");

    if (bSuccess)
        bSuccess = oc_i2c_audio_wr_reg(1, control_r); // Left Line In: set
left line in volume

    printf("[INFO] AUDIO_LineInMute...%s\r\n", bSuccess ? "success" :
"fail");
    return bSuccess;
}

int AUDIO_SetInputSource(int InputSource)
{
    int bSuccess;
    uint32_t control;
    uint32_t mask;
    control = 0x0014;
    mask = 0x01 << 2;
    if (InputSource == SOURCE_MIC)
        control |= mask;
    else
        control &= ~mask;
    bSuccess = oc_i2c_audio_wr_reg(4, control); // Left Line In: set left
line in volume
    printf("[INFO] AUDIO_SetInputSource... %s\r\n", bSuccess ? "success" :
"fail");
```

```
    return bSuccess;
}
//audio controller related

// See datasheet page 39
int AUDIO_SetSampleRate(int SampleRate)
{
    int bSuccess;
    uint32_t control;
    control = 0;

    switch (SampleRate)
    {
        // MCLK = 18.432
    case RATE_ADC48K_DAC48K:
        control = (0x0) << 2;
        break;
    case RATE_ADC48K_DAC8K:
        control = (0x1) << 2;
        break;
    case RATE_ADC8K_DAC48K:
        control = (0x2) << 2;
        break;
    case RATE_ADC8K_DAC8K:
        control = (0x3) << 2;
        break;
    case RATE_ADC32K_DAC32K:
        control = (0x6) << 2;
        break;
    case RATE_ADC96K_DAC96K:
        control = (0x7) << 2;
        break;
    case RATE_ADC44K1_DAC44K1:
        control = (0x8) << 2;
        break;

        //  case RATE_ADC44K1_DAC8K: control = (0x9) << 2; break;
        //  case RATE_ADC8K_DAC44K1: control = (0xA) << 2; break;
    }
    control |= 0x02; // BOSR=1 (384fs = 384*48k = 18.432M)
```

```
    bSuccess = oc_i2c_audio_wr_reg(8, control); // Left Line In: set left
line in volume

    printf("[INFO] AUDIO_SetSampleRate... %s\r\n", bSuccess ? "success" :
"fail");
    return bSuccess;
}

int AUDIO_SetLineInVol(int l_vol, int r_vol)
{
    int bSuccess;
    uint32_t control;

    // left
    control = 0x0017;
    control &= 0xFFC0;
    control += l_vol & 0x3F;
    bSuccess = oc_i2c_audio_wr_reg(0, control);

    usleep(10 * 1000);

    if (bSuccess)
    {
        // right
        control = 0x0017;
        control &= 0xFFC0;
        control += r_vol & 0x3F;
        bSuccess = oc_i2c_audio_wr_reg(1, control);
    }

    printf("[INFO] set Line-In vol(%d,%d) %s\r\n", l_vol, r_vol, bSuccess
? "success" : "fail");
    return bSuccess;
}

int AUDIO_SetLineOutVol(int l_vol, int r_vol)
{
    int bSuccess;
    uint32_t control;
```

```
    // left
    control = 0x005B;
    control &= 0xFF80;
    control += l_vol & 0x7F;
    bSuccess = oc_i2c_audio_wr_reg(2, control);

    usleep(10 * 1000);

    if (bSuccess)
    {
        // right
        control = 0x005B;
        control &= 0xFF80;
        control += r_vol & 0x7F;
        bSuccess = oc_i2c_audio_wr_reg(3, control);
    }

    printf("[INFO] set Line-Out vol(%x,%x) %s\r\n", l_vol, r_vol, bSuccess
? "success" : "fail");

    return bSuccess;
}

int AUDIO_EnableByPass(int bEnable)
{
    int bSuccess;
    uint32_t control;
    uint32_t mask;
    control = 0x0014;
    mask = 0x01 << 3;
    if (bEnable)
        control |= mask;
    else
        control &= ~mask;
    bSuccess = oc_i2c_audio_wr_reg(4, control);
    return bSuccess;
}

int AUDIO_EnableSiteTone(int bEnable)
```

```
{
    int bSuccess;
    uint32_t control;
    uint32_t mask;
    control = 0x0014;
    mask = 0x01 << 5;
    if (bEnable)
        control |= mask;
    else
        control &= ~mask;
    bSuccess = oc_i2c_audio_wr_reg(4, control);
    return bSuccess;
}

// check whether the dac-fifo is full.
int AUDIO_DacFifoNotFull(void)
{
    int bReady;

    bReady = ((alt_read_word(audio_addr + AUDIO_STATUS_PORT) &
MASK_STATUS_DAC_FULL) ? 0x1 : 0x0) ? 0x0 : 0x1;
    return bReady;
}

// call AUDIO_PlayIsReady to make sure the fifo is not full before call
this function
void AUDIO_DacFifoSetData(int ch_left, int ch_right)
{

    alt_write_word(audio_addr + AUDIO_DAC_LFIFO_PORT, ch_left & 0xFFFF);
    alt_write_word(audio_addr + AUDIO_DAC_RFIFO_PORT, ch_right & 0xFFFF);

    // AUDIO_DAC_WRITE_L(ch_left);
    // AUDIO_DAC_WRITE_R(ch_right);
}

void AUDIO_FifoClear(void)
{
    alt_write_word(audio_addr + AUDIO_CMD_PORT, 0x01);
}
```

```
int init_audio(void)
{
    printf("[INFO] Setting up audio chip...");
    int bSuccess = 1;

    // setting up audio chip
    if (bSuccess)
        bSuccess = oc_i2c_audio_wr_reg(15, 0x0000); // reset
    usleep(10 * 1000);
    if (bSuccess)
        bSuccess = oc_i2c_audio_wr_reg(9, 0x0000); // inactive interface
    usleep(10 * 1000);
    if (bSuccess)
        bSuccess = oc_i2c_audio_wr_reg(0, 0x0017); // Left Line In: set
left line in volume
    usleep(10 * 1000);
    if (bSuccess)
        bSuccess = oc_i2c_audio_wr_reg(1, 0x0017); // Right Line In: set
right line in volume
    usleep(10 * 1000);
    if (bSuccess)
        bSuccess = oc_i2c_audio_wr_reg(2, 0x005B); // Left Headphone Out:
set left line out volume
    usleep(10 * 1000);
    if (bSuccess)
        bSuccess = oc_i2c_audio_wr_reg(3, 0x005B); // Right Headphone Out:
set right line out volume
    usleep(10 * 1000);
    if (bSuccess)
        bSuccess = oc_i2c_audio_wr_reg(4, 0x0015 | 0x20 | 0x08 | 0x01); //
Analogue Audio Path Control: set mic as input and enable dac
    usleep(10 * 1000);
    if (bSuccess)
        bSuccess = oc_i2c_audio_wr_reg(5, 0x0000); // Digital Audio Path
Control: disable soft mute
    usleep(10 * 1000);
    if (bSuccess)
        bSuccess = oc_i2c_audio_wr_reg(6, 0x0000); // power down control:
power on all
```

```
    usleep(10 * 1000);
    if (bSuccess)
        bSuccess = oc_i2c_audio_wr_reg(7, 0x0042); // I2S, iwl=16-bits,
Enable Master Mode
    usleep(10 * 1000);
    if (bSuccess)
        bSuccess = oc_i2c_audio_wr_reg(8, 0x0002); // Normal, Base
OVer-Sampleing Rate 384 fs (BOSR=1)
    usleep(10 * 1000);
    // if (bSuccess)
    //     bSuccess = oc_i2c_audio_wr_reg(16, 0x007B); //ALC CONTROL 1
    // usleep(10 * 1000);
    // if (bSuccess)
    //     bSuccess = oc_i2c_audio_wr_reg(17, 0x0032); //ALC CONTROL 2
    // usleep(10 * 1000);
    // if (bSuccess)
    //     bSuccess = oc_i2c_audio_wr_reg(18, 0x0000); //NOISE GATE
    // usleep(10 * 1000);
    if (bSuccess)
        bSuccess = oc_i2c_audio_wr_reg(9, 0x0001); // active interface

    printf("%s\n", bSuccess ? "success" : "fail");

    AUDIO_InterfaceActive(0x0);
    usleep(10 * 1000);

    AUDIO_DacEnableSoftMute(0x1);
    usleep(10 * 1000);
    AUDIO_MicMute(0x1);
    usleep(10 * 1000);
    AUDIO_LineInMute(0x1);
    usleep(10 * 1000);
    AUDIO_DacEnableSoftMute(0x0);
    usleep(10 * 1000);

    AUDIO_SetLineOutVol(LINEOUT_DEFUALT_VOL, LINEOUT_DEFUALT_VOL); // max
7F, min: 30, 0x79: 0 db
    usleep(10 * 1000);
    AUDIO_DacEnableSoftMute(0x0);
    usleep(10 * 1000);
```

```
    AUDIO_FifoClear();
    usleep(10 * 1000);

    AUDIO_SetSampleRate(RATE_ADC44K1_DAC44K1); // Default
    usleep(10 * 1000);

    AUDIO_InterfaceActive(0x1);

    usleep(10 * 1000);

    return bSuccess;
}
```

audio_control.h

```
#ifndef _AUDIO_CONTROL_H
#define _AUDIO_CONTROL_H
//audio controller related

typedef enum
{
    SOURCE_MIC = 0,
    SOURCE_LINEIN
} INPUT_SOURCE;

typedef enum
{
    DEEMPHASIS_NONE,
    DEEMPHASIS_48K,
    DEEMPHASIS_44K1,
    DEEMPHASIS_32K
} DEEMPHASIS_TYPE;

typedef enum
{
    // MCLK = 18.432 (BOSR==1)
    RATE_ADC48K_DAC48K,
```

```
    RATE_ADC48K_DAC8K,
    RATE_ADC8K_DAC48K,
    RATE_ADC8K_DAC8K,
    RATE_ADC32K_DAC32K,
    RATE_ADC96K_DAC96K,
    RATE_ADC44K1_DAC44K1
} AUDIO_SAMPLE_RATE;

#define TRUE 1
#define FALSE 0

#define MAX_TRY_CNT 1024
#define LINEOUT_DEFUALT_VOL 0x79 // 0 dB

void oc_i2c_audio_init(void);
int oc_i2c_audio_wr_reg(int reg, int data);
int oc_i2c_audio_rd_reg(int reg);
int AUDIO_Init(void);
int AUDIO_InterfaceActive(int bActive);
int AUDIO_MicBoost(int bBoost);
int AUDIO_AdcEnableHighPassFilter(int bEnable);
int AUDIO_DacDeemphasisControl(int deemphasis_type);
int AUDIO_DacEnableSoftMute(int bEnable);
int AUDIO_MicMute(int bMute);
int AUDIO_LineInMute(int bMute);
int AUDIO_SetInputSource(int InputSource);
int AUDIO_SetSampleRate(int SampleRate);
int AUDIO_SetLineInVol(int l_vol, int r_vol);
int AUDIO_SetLineOutVol(int l_vol, int r_vol);
int AUDIO_EnableByPass(int bEnable);
int AUDIO_EnableSiteTone(int bEnable);
int AUDIO_DacFifoNotFull(void);
void AUDIO_DacFifoSetData(int ch_left, int ch_right);
void AUDIO_FifoClear(void);
int init_audio(void);
#endif //_AUDIO_CONTROL_H
```

pcm.c

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

#include "pcm.h"
#include "audio_control.h"

int play_PCM(const char *filename)
{
    FILE *fp = fopen(filename, "rb+");
    if (fp == NULL)
    {
        fprintf(stderr, "[ERROR] Open %s failed.\n", filename);
        return -1;
    }

    char *sample = (char *)malloc(4);
    while (!feof(fp))
    {
        int try_cnt = 0;
        while (!AUDIO_DacFifoNotFull() && try_cnt < MAX_TRY_CNT)
            try_cnt++;

        if (try_cnt >= MAX_TRY_CNT)
            fprintf(stderr, "[ERROR] Audio chip error...\n");

        fread(sample, 1, 4, fp);
        int sample_l = (int)sample[0] | (((int)sample[1]) << 8);
        int sample_r = (int)sample[2] | (((int)sample[3]) << 8);

        AUDIO_DacFifoSetData(sample_l, sample_r);
    }

    free(sample);
    fclose(fp);
    return 0;
}
```

pcm.h

```
#ifndef _pcm_h_
#define _pcm_h_
int play_PCM(const char *filename);
#endif
```

Hps_0.h

```
#ifndef _ALTERA_HPS_0_H_
#define _ALTERA_HPS_0_H_

/*
 * This file was automatically generated by the swinfo2header utility.
 *
 * Created from SOPC Builder system 'soc_system' in
 * file '../../soc_system.sopcinfo'.
 */

/*
 * This file contains macros for module 'hps_0' and devices
 * connected to the following masters:
 *   h2f_axi_master
 *   h2f_lw_axi_master
 *
 * Do not include this header file and another header file created for a
 * different module or master group at the same time.
 * Doing so may result in duplicate macro names.
 * Instead, use the system header file which has macros with unique names.
 */

/*
 * Macros for device 'oc_i2c_master_0', class 'oc_i2c_master'
 * The macros are prefixed with 'OC_I2C_MASTER_0_'.
 * The prefix is the slave descriptor.
 */
#define OC_I2C_MASTER_0_COMPONENT_TYPE oc_i2c_master
#define OC_I2C_MASTER_0_COMPONENT_NAME oc_i2c_master_0
#define OC_I2C_MASTER_0_BASE 0x0
#define OC_I2C_MASTER_0_SPAN 32
#define OC_I2C_MASTER_0_END 0x1f
```

```
/*
 * Macros for device 'AUDIO_IF_0', class 'AUDIO_IF'
 * The macros are prefixed with 'AUDIO_IF_0_'.
 * The prefix is the slave descriptor.
 */
#define AUDIO_IF_0_COMPONENT_TYPE AUDIO_IF
#define AUDIO_IF_0_COMPONENT_NAME AUDIO_IF_0
#define AUDIO_IF_0_BASE 0x20
#define AUDIO_IF_0_SPAN 32
#define AUDIO_IF_0_END 0x3f

/*
 * Macros for device 'sysid_qsys', class 'altera_avalon_sysid_qsys'
 * The macros are prefixed with 'SYSID_QSYS_'.
 * The prefix is the slave descriptor.
 */
#define SYSID_QSYS_COMPONENT_TYPE altera_avalon_sysid_qsys
#define SYSID_QSYS_COMPONENT_NAME sysid_qsys
#define SYSID_QSYS_BASE 0x10000
#define SYSID_QSYS_SPAN 8
#define SYSID_QSYS_END 0x10007
#define SYSID_QSYS_ID 2899645186
#define SYSID_QSYS_TIMESTAMP 1638767327

/*
 * Macros for device 'led_pio', class 'altera_avalon_pio'
 * The macros are prefixed with 'LED_PIO_'.
 * The prefix is the slave descriptor.
 */
#define LED_PIO_COMPONENT_TYPE altera_avalon_pio
#define LED_PIO_COMPONENT_NAME led_pio
#define LED_PIO_BASE 0x10040
#define LED_PIO_SPAN 16
#define LED_PIO_END 0x1004f
#define LED_PIO_BIT_CLEARING_EDGE_REGISTER 0
#define LED_PIO_BIT_MODIFYING_OUTPUT_REGISTER 0
#define LED_PIO_CAPTURE 0
#define LED_PIO_DATA_WIDTH 10
#define LED_PIO_DO_TEST_BENCH_WIRING 0
```

```
#define LED_PIO_DRIVEN_SIM_VALUE 0
#define LED_PIO_EDGE_TYPE NONE
#define LED_PIO_FREQ 50000000
#define LED_PIO_HAS_IN 0
#define LED_PIO_HAS_OUT 1
#define LED_PIO_HAS_TRI 0
#define LED_PIO_IRQ_TYPE NONE
#define LED_PIO_RESET_VALUE 15

/*
 * Macros for device 'dipsw_pio', class 'altera_avalon_pio'
 * The macros are prefixed with 'DIPSW_PIO_'.
 * The prefix is the slave descriptor.
 */
#define DIPSW_PIO_COMPONENT_TYPE altera_avalon_pio
#define DIPSW_PIO_COMPONENT_NAME dipsw_pio
#define DIPSW_PIO_BASE 0x10080
#define DIPSW_PIO_SPAN 16
#define DIPSW_PIO_END 0x1008f
#define DIPSW_PIO_IRQ 0
#define DIPSW_PIO_BIT_CLEARING_EDGE_REGISTER 1
#define DIPSW_PIO_BIT_MODIFYING_OUTPUT_REGISTER 0
#define DIPSW_PIO_CAPTURE 1
#define DIPSW_PIO_DATA_WIDTH 10
#define DIPSW_PIO_DO_TEST_BENCH_WIRING 0
#define DIPSW_PIO_DRIVEN_SIM_VALUE 0
#define DIPSW_PIO_EDGE_TYPE ANY
#define DIPSW_PIO_FREQ 50000000
#define DIPSW_PIO_HAS_IN 1
#define DIPSW_PIO_HAS_OUT 0
#define DIPSW_PIO_HAS_TRI 0
#define DIPSW_PIO_IRQ_TYPE EDGE
#define DIPSW_PIO_RESET_VALUE 0

/*
 * Macros for device 'button_pio', class 'altera_avalon_pio'
 * The macros are prefixed with 'BUTTON_PIO_'.
 * The prefix is the slave descriptor.
 */
#define BUTTON_PIO_COMPONENT_TYPE altera_avalon_pio
```

```
#define BUTTON_PIO_COMPONENT_NAME button_pio
#define BUTTON_PIO_BASE 0x100c0
#define BUTTON_PIO_SPAN 16
#define BUTTON_PIO_END 0x100cf
#define BUTTON_PIO_IRQ 1
#define BUTTON_PIO_BIT_CLEARING_EDGE_REGISTER 1
#define BUTTON_PIO_BIT_MODIFYING_OUTPUT_REGISTER 0
#define BUTTON_PIO_CAPTURE 1
#define BUTTON_PIO_DATA_WIDTH 4
#define BUTTON_PIO_DO_TEST_BENCH_WIRING 0
#define BUTTON_PIO_DRIVEN_SIM_VALUE 0
#define BUTTON_PIO_EDGE_TYPE FALLING
#define BUTTON_PIO_FREQ 50000000
#define BUTTON_PIO_HAS_IN 1
#define BUTTON_PIO_HAS_OUT 0
#define BUTTON_PIO_HAS_TRI 0
#define BUTTON_PIO_IRQ_TYPE EDGE
#define BUTTON_PIO_RESET_VALUE 0

/*
 * Macros for device 'jtag_uart', class 'altera_avalon_jtag_uart'
 * The macros are prefixed with 'JTAG_UART_'.
 * The prefix is the slave descriptor.
 */
#define JTAG_UART_COMPONENT_TYPE altera_avalon_jtag_uart
#define JTAG_UART_COMPONENT_NAME jtag_uart
#define JTAG_UART_BASE 0x20000
#define JTAG_UART_SPAN 8
#define JTAG_UART_END 0x20007
#define JTAG_UART_IRQ 2
#define JTAG_UART_READ_DEPTH 64
#define JTAG_UART_READ_THRESHOLD 8
#define JTAG_UART_WRITE_DEPTH 64
#define JTAG_UART_WRITE_THRESHOLD 8


#endif /* _ALTERA_HPS_0_H_ */
```

main.c

```c
#include <stdio.h>
#include <memory.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <dirent.h>
#include <inttypes.h>
#include <stdbool.h>

#include "hwlib.h"
#include "socal/hps.h"
#include "socal/socal.h"
#include "hps_0.h"
#include "audio_control.h"
#include "pcm.h"

#define HW_REGS_BASE (ALT_STM_OFST)
#define HW_REGS_SPAN (0x04000000)
#define HW_REGS_MASK (HW_REGS_SPAN - 1)

//base addr
static volatile unsigned long *h2p_lw_axi_addr = NULL;

volatile unsigned long *oc_i2c_audio_addr = NULL;
volatile unsigned long *audio_addr = NULL;

int main(int argc, char **argv)
{
    void *virtual_base;
    int fd;

    if ((fd = open("/dev/mem", (O_RDWR | O_SYNC))) == -1)
    {
        fprintf(stderr, "[ERROR] Fail to open \"/dev/mem\"...\n");
        return 1;
    }
```

```
    virtual_base = mmap(NULL, HW_REGS_SPAN, (PROT_READ | PROT_WRITE),
MAP_SHARED, fd, HW_REGS_BASE);
    if (virtual_base == MAP_FAILED)
    {
        fprintf(stderr, "[ERROR] mmap() failed...\n");
        close(fd);
        return 1;
    }

    h2p_lw_axi_addr = virtual_base + ((unsigned long)(ALT_LWFPGASLVS_OFST)
& (unsigned long)(HW_REGS_MASK));
    oc_i2c_audio_addr = virtual_base + ((unsigned
long)(ALT_LWFPGASLVS_OFST + OC_I2C_MASTER_0_BASE) & (unsigned
long)(HW_REGS_MASK));
    audio_addr = virtual_base + ((unsigned long)(ALT_LWFPGASLVS_OFST +
AUDIO_IF_0_BASE) & (unsigned long)(HW_REGS_MASK));

    printf("[INFO] i2c_audio_addr:  %04Xh\n", (unsigned
int)oc_i2c_audio_addr);
    printf("[INFO] audio_addr:      %04Xh\n", (unsigned int)audio_addr);

    oc_i2c_audio_init();
    init_audio();

    usleep(500 * 1000); ///// !!!!!!! note. this delay is necessary

    char AudioName[128] = {"TheDawn.mp3"};
    if (argc == 2 && argv[1])
        strncpy(AudioName, argv[1], 128);

    //TODO: Check if .pcm exists.

    char PCMName[128 + 4];
    strncpy(PCMName, AudioName, 128);
    strncat(PCMName, ".pcm", 4);

    AUDIO_SetSampleRate(RATE_ADC32K_DAC32K);

    FILE *is_exist;
    if (is_exist = fopen(PCMName, "r"), is_exist == NULL)
```

```
    {
        printf("[INFO] Converting PCM...\n");
        char cmd[512];
        sprintf(cmd, "ffmpeg -i \"%s\" -f s16le -ar 32000 -ac 2 -acodec
pcm_s16le \"%s\" -y", AudioName, PCMName);
        if (system(cmd) != 0)
        {
            fprintf(stderr, "[ERROR] Convert PCM failed...");
            munmap(virtual_base, HW_REGS_SPAN);
            close(fd);
            return 1;
        }
    }

    printf("[INFO] Now playing %s...\n", AudioName);
    play_PCM(PCMName);
    printf("[INFO] Music over, quitting...\n");

    if (munmap(virtual_base, HW_REGS_SPAN) != 0)
    {
        fprintf(stderr, "[ERROR] munmap() failed...\n");
        close(fd);
        return 1;
    }

    close(fd);
    return 0;
}
```