Rhythm Master
Final Report

Yangyang Zhang (yz4843), Junfeng Zou (jz3850), Hongrui Huang (hh3084)

CSEE4840 Embedded Systems Columbia University, Spring 2025

Table of Contents

1. Project Introduction

This document outlines the design of Rhythm Master, a rhythm-based music game where players synchronize key presses with falling notes. The system integrates both hardware and software components to achieve real-time gameplay, audio synchronization, and user interaction. Key design decisions include:

Audio (FPGA + HPS): Audio playback is achieved through collaboration between the FPGA and HPS. The FPGA hosts three IP cores — Audio PLL, Audio and Video Config, and Audio Core — which handle sample timing, codec initialization, and audio output via the WM8731 codec using I²S. However, the audio data itself is streamed from the HPS to the FPGA in real time.

Display (FPGA + HPS): The VGA framebuffer is managed cooperatively. The HPS maintains both current_frame and next_frame buffers. After rendering a new frame in next_frame, the HPS compares it with current_frame and sends only the differences to the FPGA via the Avalon interface, updating the relevant indices in on-chip memory. The FPGA module continuously reads these indices, retrieves the actual color from a LUT, and outputs the video signal.

Game Logic & USB Keyboard Input (HPS): All gameplay logic, including note movement, hit detection, scoring, and menu flow, runs on the HPS. USB keyboard inputs are processed in software to control gameplay. During gameplay, audio timing serves as the reference for synchronizing game logic and visual output, ensuring audio-visual alignment.

Communication: The HPS streams 16-bit PCM audio data to the FPGA and transmits only the changed VGA pixel data for efficient, low-latency updates. This architecture emphasizes real-time audio playback and visual synchronization for a seamless rhythm game experience.

2. Design Overview

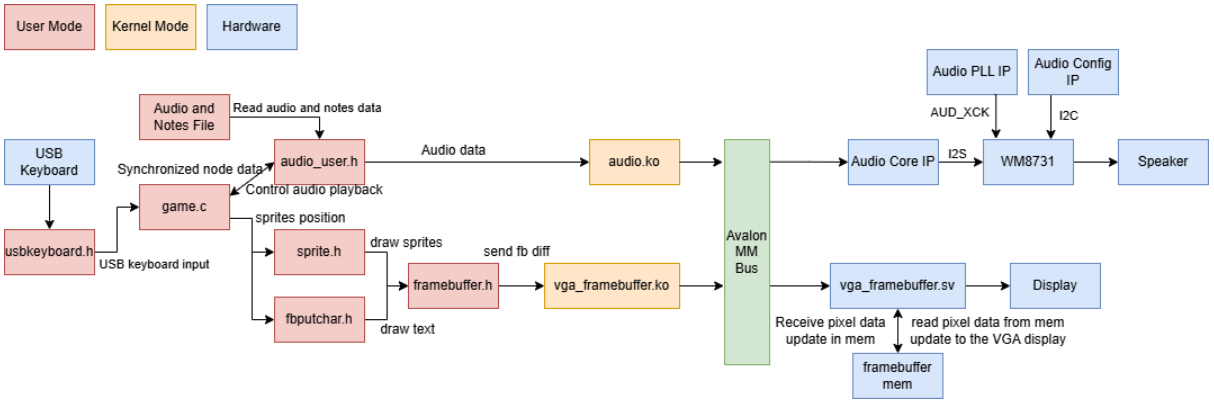


Figure 2.1. Block Diagram for System Design

The architecture is split into functional components as follows:

Hardware (FPGA) Components

1. Audio Pipeline:
    - Audio PLL IP: Generates required clock signals.
    - Audio and Video Config IP: Configures WM8731 codec.
    - Audio Core IP: Streams audio data to the codec using I²S protocol.
2. VGA Display Pipeline (vga_framebuffer.sv):
    - Receives pixel data via Avalon-MM from HPS
    - Stores pixel values in a framebuffer located in FPGA on-chip memory
    - Continuously scans framebuffer and generates VGA signals for monitor output

Software (HPS) Components

1. Sprite Renderer: Renders all graphical elements (background, notes, score, etc.) into a next_frame buffer in HPS memory.
2. Frame Sender: After rendering, the content of next_frame is compared with current_frame, and only the differences are sent to the FPGA via Avalon-MM interface, updating the relevant indices in on-chip memory.
3. Game Logic: Manages note generation, timing synchronization with audio, grading, and level control.
4. Input Handler: Handles USB keyboard input and maps them to in-game controls.
5. Audio Data Streamer: Transfers audio sample buffers to the FPGA through Avalon-MM interface.

Communication Protocols

1. Audio Streaming: 16-bit PCM samples are pushed from the HPS to the FPGA via Avalon-MM writes to the Audio Core IP's memory-mapped FIFO interface. The software alternates between left and right channel FIFO writes.

2. Video Streaming: The HPS renders each frame into a local next_frame buffer. After rendering, the HPS compares it with current_frame and sends only the differences to the FPGA via Avalon-MM interface, updating the relevant indices in on-chip memory. The FPGA module continuously reads these indices, retrieves the actual color from a LUT, and outputs the video signal.
3. Game Events/Input: Entirely managed on HPS; no need for hardware interrupts or polling from the FPGA side.

3. Hardware Design

3.1 Audio Hardware Control

To achieve high-quality audio playback with precise timing control, we utilized the audio IP cores provided by Intel's Platform Designer. The Audio Core IP serves as the central component of our audio subsystem, implementing the I²S protocol for digital audio transmission. This IP core features a memory-mapped interface that allows the HPS to stream audio data directly to the FPGA. The core maintains separate FIFO buffers for left and right channels, ensuring proper stereo audio output. When the HPS writes 16-bit PCM samples to the Audio Core's memory-mapped interface, the core automatically manages the timing and transmission of these samples to the WM8731 codec through the I²S interface (Figure 3.1).

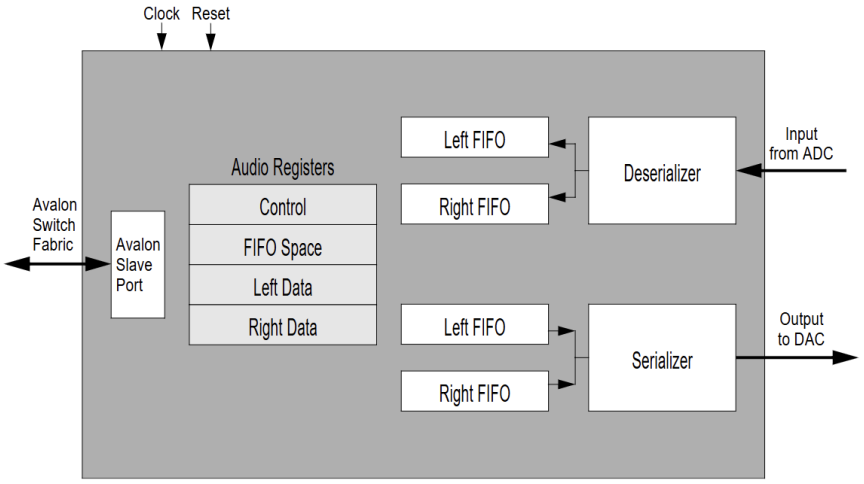


Figure 3.1 Block Diagram for Audio Core with Memory Mapped Interface

For the Audio Core to function properly, it requires two additional IP cores: the Audio PLL and Audio and Video Config. The Audio PLL generates the necessary 12.288 MHz clock signal required by the WM8731 codec, while the Audio and Video Config IP handles the initialization and configuration of the codec through the I²C protocol. The Audio Config IP sets up critical parameters such as the sample rate (48 kHz), bit depth (16-bit), and input/output routing. These three IP cores work together to create a complete audio pipeline, where the HPS streams audio data through the Avalon-MM interface to the Audio Core, which then transmits the data to the codec for conversion to analog audio output. The interconnection of these components in the Platform Designer is shown in Figure 3.2.

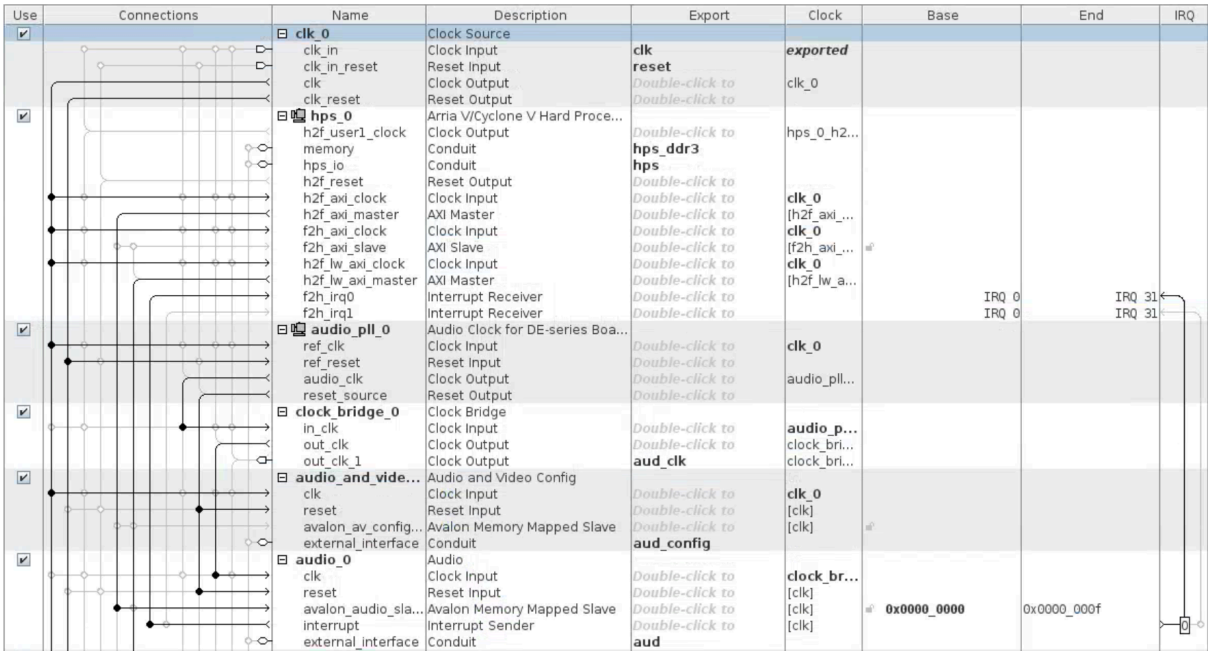


Figure 3.2 Audio Hardware Platform Designer Configuration

3.2 VGA Hardware Control

The VGA display system is implemented using a custom SystemVerilog module (vga_framebuffer.sv) that handles video signal generation and frame buffer management. This module generates standard VGA timing signals (HSync, VSync) and manages the pixel clock to ensure proper display synchronization. The core of the VGA system is its efficient frame buffer implementation, which uses FPGA on-chip memory to store pixel data. Instead of storing raw color values, the system uses a color lookup table (LUT) approach, where the frame buffer stores indices that reference actual color values in the LUT. This design significantly reduces memory bandwidth requirements while maintaining full color capability (Figure 3.3).

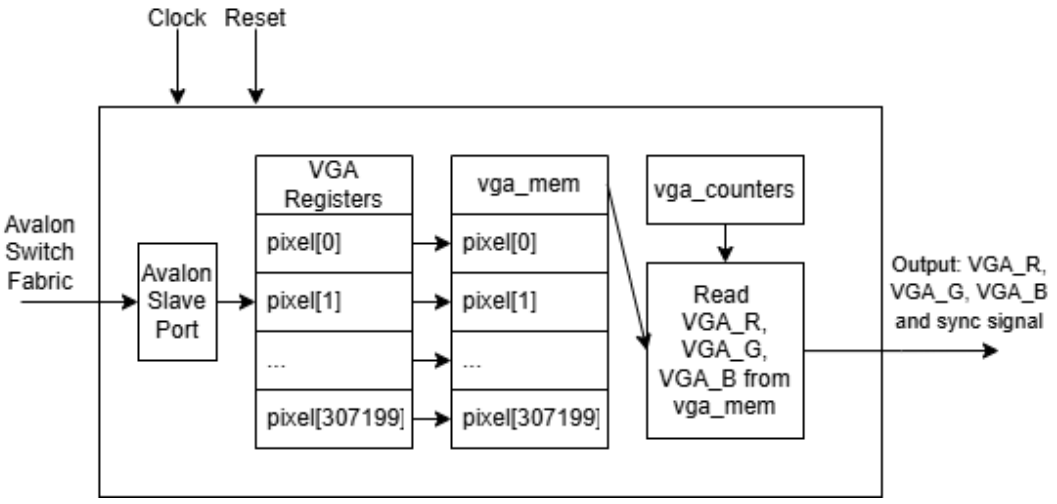


Figure 3.3 Block Diagram for VGA Frame Buffer SystemVerilog Module

The VGA controller implements a sophisticated update mechanism that only transfers changed pixels between the HPS and FPGA. When the HPS renders a new frame in its next_frame buffer, it compares this with the current_frame and sends only the differences to the FPGA through the Avalon-MM interface. The FPGA then updates the corresponding indices in its frame buffer. The VGA controller continuously reads these indices, looks up the actual colors from the LUT, and generates the appropriate video signals for display output. The complete VGA subsystem, including the interconnection between the HPS, VGA controller, and memory components, is configured in the Platform Designer as shown in Figure 3.4.

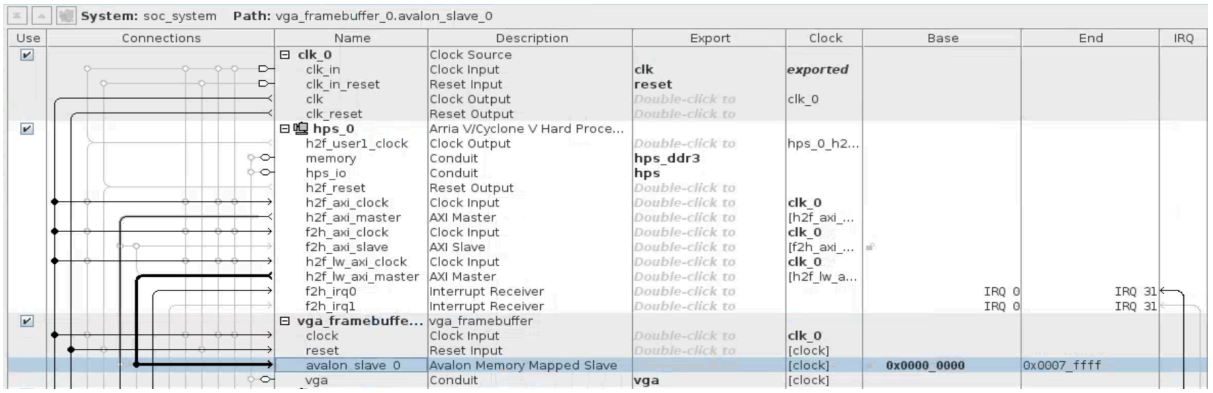


Figure 3.4 VGA Hardware Platform Designer Configuration

4. HW/SW Interface

All hardware-software interaction is based on memory-mapped I/O (MMIO). The system uses standard IPs for audio, and a custom VGA display pipeline using a software-driven framebuffer communication mechanism. No Avalon streaming or SDRAM buffers are used.

4.1 Audio Interface

The audio path is implemented using the Audio Core IP, which interfaces with the WM8731 codec via I²S. Only the playback (sound output) path is enabled. The following interface description is directly quoted from the "Audio core for Intel® DE-Series Boards" documentation:

4.1.1 Register Map

Software can control and communicate with the Audio core through four 32-bit registers when using the Memory-Mapped Avalon Type. By writing or reading these registers, data can be fetched from the CODEC's Analog-Digital Converter (ADC) or sent to the Digital-Analog Converter (DAC). Table 1 shows the format of the registers.

Table 1. Audio core register map

Offset in bytes	Register Name	R/W	Bit Description									
			31…24	23…16	15…10	9	8	7…4	3	2	1	0
0	control	RW	*(1)*			WI	RI	*(1)*	CW	CR	WE	RE
4	fifospace	R	WS LC	WS RC	RA LC			RA RC				
8	leftdata	RW *(2)*	Left Data									
12	rightdata	RW *(2)*	Right Data									

Notes on Table 1:
(1) Reserved. Read values are undefined. Write zero.
(2) Only reads incoming audio data and writes outgoing audio data.

4.1.2 Control Register

Table 2. Control register bits

Bit number	Bit name	Read/Write	Description
0	RE	R/W	Interrupt-enable bit for read interrupts. If the RE bit is set to 1 and both the left and right channel read FIFOs contain data, the Audio core generates an interrupt request (IRQ).
1	WE	R/W	Interrupt-enable bit for write interrupts. If the WE bit is set to 1 and both the left and right channel write FIFOs have space available for more data, the Audio core generates an interrupt request (IRQ).
2	CR	R/W	Clears the Audio core's Input FIFOs, when the bit is 1. Clear remains active until specifically set to zero.
3	CW	R/W	Clears the Audio core's Output FIFOs, when the bit is 1. Clear remains active until specifically set to zero.
8	RI	R	Indicates that a read interrupt is pending.
9	WI	R	Indicates that a write interrupt is pending.

4.1.3 Fifospace Register

The *fifospace* register fields WSLC (b31−24) and WSRC (b23−16) indicate the number of words available (i.e., the amount of empty space) for outgoing data in the left and right channel FIFOs, respectively, while RALC (b15−8) and RARC (b7−0) indicate the number of words of incoming audio data in the left and right channel FIFOs, respectively. When all of the outgoing and incoming FIFOs are empty, the fifospace register will hold WSLC = WSRC = 128, and RALC = RARC = 0.

4.1.4 Leftdata Register

The *leftdata* register is readable only for Audio In and writable only for Audio Out. It stores the data coming from or going to the left channel. The data is always flush right, i.e., the LSB is b0 of the *leftdata* register.

4.1.5 Rightdata Register

The *rightdata* register is readable only for Audio In and writable only for Audio Out. It stores the data coming from or going to the right channel. The data is always flush right, i.e., the LSB is b0 of the *rightdata* register.

4.2 VGA Frame Buffer Interface

The video output is implemented using a custom System Verilog module vga_framebuffer.sv, which maintains a framebuffer in FPGA on-chip memory and drives VGA signals in real time. The interface is designed to be simple and efficient, using a direct memory-mapped approach where each pixel's color index can be written directly to its corresponding memory location.

The framebuffer interface consists of a linear array of 307,200 (640×480) 8-bit registers, each representing a color index for a specific pixel location. The register map is as follows:

Offset	Register name	R/W	Description
0	pixel[0]	W	8-bit color index at pixel[0]
1	pixel[1]	W	8-bit color index at pixel[1]
2	pixel[2]	W	8-bit color index at pixel[2]
3	pixel[2]	W	8-bit color index at pixel[2]
…	pixel[...]	W	8-bit color index at pixel[...]
307198	pixel[307198]	W	8-bit color index at pixel[307198]
307199	pixel[307199]	W	8-bit color index at pixel[307199]

Each register is write-only and contains an 8-bit color index that maps to the actual color value through a color lookup table (LUT) implemented in the FPGA. The HPS can update any pixel's color by writing to its corresponding register address. The VGA controller continuously reads these color indices, looks up the actual colors from the LUT, and generates the appropriate video signals for display output.

5. Software Design

5.1 Audio Playback

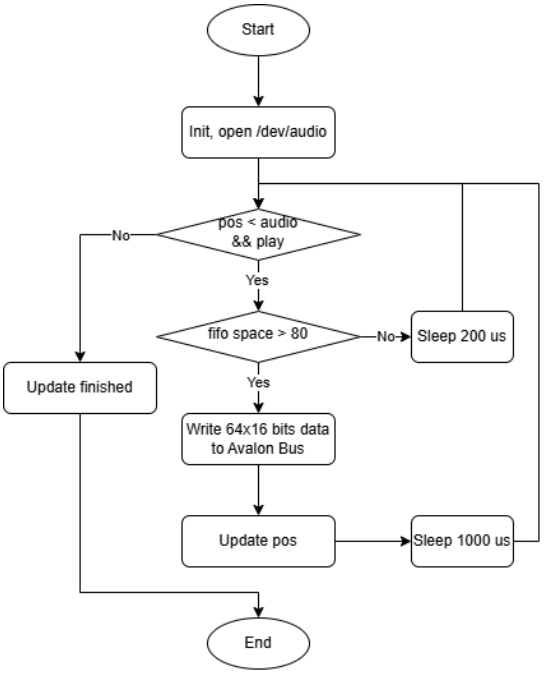


Figure 5.1 Audio Playback Thread Flowchart

The audio playback software is implemented as a dedicated thread to ensure continuous audio streaming without blocking the main game logic. This thread is responsible for streaming audio data to the FPGA and maintaining synchronization with the game logic. The implementation uses a buffered streaming mechanism that sends audio data in batches of 64 samples per channel to the Audio Core's memory-mapped interface. This batch processing approach ensures efficient use of the Avalon-MM interface while maintaining continuous audio playback.

A critical component of the audio system is the playback time tracking mechanism. The software maintains a timer that tracks the current playback position, which serves as the synchronization reference for the entire game system. This timing information is used to coordinate note rendering, hit detection, and scoring with the audio playback. To prevent race conditions between the audio thread and the main game thread, atomic variables are used to safely share timing information and control flags.

The software implements a double-buffering strategy to prevent audio underruns. It monitors the FIFO space through the fifospace register and ensures that new audio data is always available before the current buffer is exhausted. Figure 5.1 shows the audio playback thread flowchart, illustrating the thread's role in managing audio data flow and synchronization.

5.2 Sprite Rendering

The sprite rendering system is implemented entirely in software, designed to work efficiently with our color index-based display system. The original sprite assets are provided as PNG images with transparency layers, which need to be processed to fit our 16-color palette design. To achieve this, we developed a Python-based preprocessing pipeline that handles sprite conversion and color optimization.

The preprocessing pipeline first categorizes sprites into groups based on their visual characteristics and usage in the game. For each group, a Python script processes the PNG images to create an optimized 16-color palette. The script analyzes the color distribution in the original images and selects the most representative 16 colors for each sprite group. These colors are then used to create a new palette, and the images are converted to use only these colors while maintaining their transparency information. Finally, the processed images are saved back as 32-bit PNGs with transparency layers, ensuring that each sprite group uses only its designated 16 colors.

The sprite rendering software implementation, defined in sprite.h and sprite.c, provides a framework for managing and rendering these processed sprites. The system maintains a sprite registry that stores information about each sprite, including its dimensions and pixel data. When rendering, the software determines which sprites need to be drawn based on the current game state and audio playback position. For each sprite, it applies the appropriate transformations (position and scale) and renders it into the next_frame buffer using the sprite's designated color palette.

The rendering process takes into account sprite layering and transparency, ensuring that sprites are drawn in the correct order and that transparent pixels are properly handled. The system also implements sprite animation by managing frame sequences and timing, allowing for smooth transitions between different sprite states. This animation system is particularly important for the note sprites, which need to smoothly transition between different visual states as they move down the screen.

5.3 Video Rendering

The video rendering system implements an efficient diff-based update mechanism to minimize data transfer between the HPS and FPGA. The system maintains two frame buffers in HPS memory: now_buffer and next_buffer. All rendering operations, including sprite drawing, text rendering via fbputchar, UI elements, and game state visualization, are performed on the next_buffer. This approach allows for complete frame composition without affecting the currently displayed frame.

When a new frame is ready to be displayed, the software calls the framebuffer_swap function, which compares the next_buffer with now_buffer to identify changed pixels. This comparison is performed efficiently by iterating through each pixel address and checking for differences. The system then sends only the changed pixels to the FPGA via the Avalon-MM interface, using the kernel module framebuffer.ko to handle the memory-mapped I/O operations.

The diff-based update mechanism significantly reduces the amount of data that needs to be transferred between the HPS and FPGA, as only the changed pixels are sent rather than the entire frame. This is particularly important given the resource constraints imposed by audio streaming, which limits the available bandwidth for video updates. The system achieves a stable refresh rate of approximately 10Hz, which is sufficient for the rhythm game's visual requirements.

The video rendering system works in conjunction with the sprite rendering system to ensure that all visual elements are properly composed and updated. The sprite system handles the drawing of individual game elements, while the video rendering system manages the overall frame composition and transmission to the FPGA. This separation of concerns allows for efficient and maintainable code organization.

5.4 Game Logic

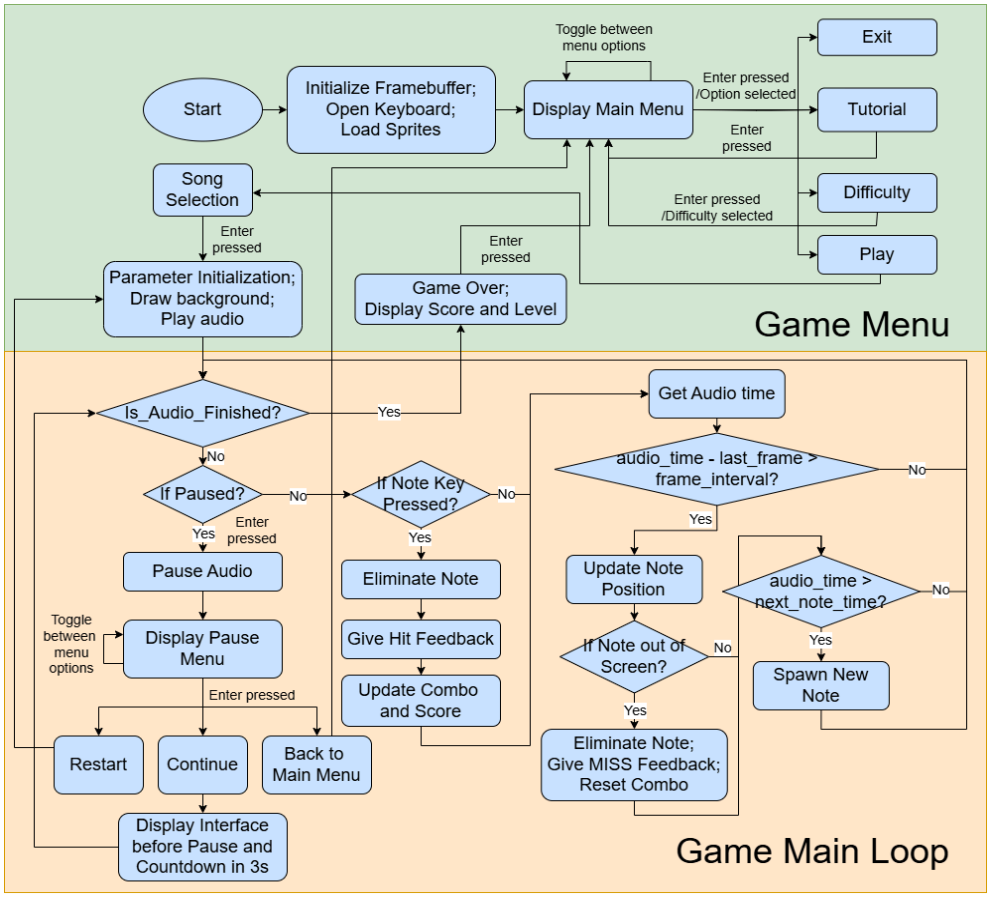


Figure 5.2 Game Logic Flowchart

The game logic is tightly synchronized with the audio playback timing to ensure accurate note timing and scoring. The system uses pre-processed note arrays of different songs to determine when and which track to appear on screen. Each note's position is updated based on the current audio playback time, ensuring perfect synchronization between audio and visual elements. The x,y coordinates as well as the width and height of the notes are accelerated to achieve a visual proximity effect from far away to near.

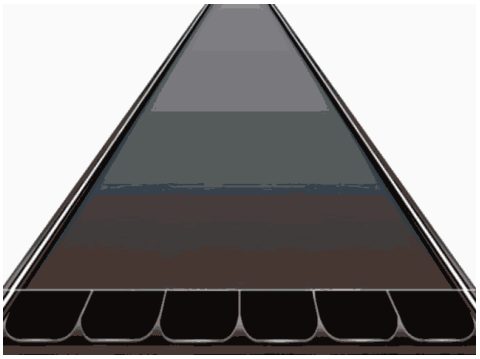
A determination area at the bottom of the screen serves as the hit detection zone. For each note, the system checks if it is the bottom note block on its track and if it is pressed or goes below the determination region. Hit feedback is given according to the note's position when its corresponding key is pressed: Perfect when most part of the note is in the determination region, Good when part of the note is in the determination region, and Miss when the key is not pressed.

The scoring system maintains a counter incremented for combo hits, which resets on miss. The final score is calculated as Hit Score (Perfect × 50; Good × 30) + Combo Bonus (combo × 5 but no greater than 50). When a whole song is completed, the system gives a level classification (S, A, B, C, D) according to the final score. The maximum possible score is calculated based on the total number of notes in the song, and the level classification is determined by the ratio of the player's score to the maximum score: S for 90% or higher, A for 80% or higher, B for 60% or higher, C for 40% or higher, and D for less than 40%.

Figure 5.2 shows the overall game logic flowchart, illustrating the flow of the game from start to finish, including note generation, hit detection, scoring, and level classification.

6. Resources

6.1 Sprites (Stored on HPS)

Category	Graphics	Size(pixel)
Logo		480×320
Background		640×480
Number		25×28
Note		80×32
Combo&Score		48×16
Hit Feedback		85×28; 102×28; 137×28
Level		76×73

6.2 VGA Frame Buffer Memory (Stored on FPGA BRAM)

The Cyclone V SoC FPGA provides a total of **4,450 Kbits (approximately 556 KB)** of on-chip embedded memory. This includes all available block RAM that must be shared across the entire design. If we were to store a full VGA frame (640×480 resolution) with 24-bit RGB color per pixel directly in memory, it would require 640×480×24 = 7,372,800 bits (approximately 902 KB), which exceeds the total available on-chip memory.

To address this limitation, we store only an 8-bit color index for each pixel in the vga_mem, reducing the required memory to **640×480×8 = 2,457,600 bits (approximately 307 KB)** — well within the available BRAM. This efficient format allows us to maintain full-resolution VGA graphics without exceeding the FPGA's memory resources.

6.3 Color Lookup Table (Stored on FPGA BRAM)

To convert color indices to actual 24-bit RGB values for display, we use a color lookup table stored as a small ROM. This table contains **256 entries, each 24 bits wide, for a total of 6,144 bits (768 bytes)**. The lookup table is implemented using distributed or block RAM and consumes a negligible amount of resources compared to a full framebuffer.

This architecture balances memory efficiency and display quality, enabling a rich visual output within the tight memory constraints of the Cyclone V FPGA.

6.4 Audio Core IP FIFO (Stored on FPGA BRAM)

Since the internal implementation of the Intel-provided Audio Core IP is not accessible, we make a reasonable assumption regarding its resource usage based on its documented functionality and our usage pattern. The Audio Core IP includes separate FIFO buffers for the left and right audio channels to support continuous stereo playback at 48 kHz using the I²S interface.

In our design, we only use 16-bit audio samples. If the Audio Core IP is optimized to match this configuration, it should only require **2 FIFOs × 128 samples × 16 bits, totaling 4,096 bits (512 bytes)** of BRAM. However, if the IP is not optimized and instead defaults to its maximum capacity with **4 FIFOs × 128 samples × 32 bits**, it would require **16,384 bits (2,048 bytes)**.

Even in the worst-case scenario, the total on-chip BRAM usage remains within the limits of the Cyclone V SoC’s 4,450 Kbits (556 KB) of embedded memory. Therefore, the Audio Core’s FIFO buffers do not pose a resource constraint in our system.

7. Task Allocation

Yangyang Zhang (yz4843):

- Implemented and configured the audio pipeline on FPGA using IP cores (Audio PLL, Audio and Video Config, Audio Core).
- Designed and debugged the I2S-based audio streaming mechanism from HPS to the WM8731 codec.
- Built the VGA framebuffer display logic in SystemVerilog, including memory-mapped interface and scanout logic.
- Developed the FPGA side of the Avalon-MM interface for both audio and video data.

Junfeng Zou (jz3850):

- Implemented game logic on HPS: note scheduling, movement, scoring, grading, pause menu, and endgame evaluation.
  Managed USB keyboard input via device file I/O and mapped key presses to in-game actions.
- Integrated timing logic to synchronize note updates and hit detection with audio playback.
  Handled transitions between gameplay states and implemented the main game loop.

Hongrui Huang (hh3084):

- Developed the sprite rendering functions in software, drawing notes, background, score, combo, and rating onto the next_frame buffer.
- Implemented the differential frame comparison system between current_frame and next_frame and designed the logic for sending only changed pixels to the FPGA via Avalon-MM for efficient video updates.
- Coordinated color LUT and graphics formatting to ensure correct on-screen display via the FPGA module.

8. Lessons Learned

Throughout the development of Rhythm Master, we gained valuable insights into hardware-software co-design and real-time system integration. A key takeaway was the importance of clear modular separation and communication between the FPGA and HPS. By offloading real-time audio playback and pixel updates to the FPGA while centralizing game logic and input handling in the HPS, we achieved both responsiveness and design clarity. Additionally, implementing differential frame updates — where only changed pixels are transmitted from the HPS to the FPGA — significantly reduced data transfer overhead and improved visual performance. This highlighted the effectiveness of optimizing bandwidth in embedded systems with limited shared memory access. Furthermore, synchronizing audio playback with visual updates using audio timing as the global reference taught us the importance of time-domain alignment in rhythm-based applications. Overall, the project deepened our understanding of Avalon-MM interfacing, I²S audio protocols, and real-time rendering strategies, reinforcing the critical role of careful timing, buffering, and task partitioning in achieving seamless multimedia experiences.

9. Code

9.1 Hardware

vga_framebuffer.sv

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module vga_framebuffer (
    input  logic         clk,
    input  logic         reset,
    input  logic [7:0]   writedata,    // 8-bit color index
    input  logic         write,
    input  logic         chipselect,
    input  logic [18:0]  address,      // 19-bit pixel address

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

  logic [10:0] hcount;
  logic [ 9:0] vcount;
  logic [ 8:0] pixel_y;
  logic [ 9:0] pixel_x;
  logic [18:0] read_addr;
  logic [18:0] read_addr_reg;
  logic [18:0] write_addr;

  logic [7:0]  pixel_data;
  logic [7:0]  write_data;
  logic        write_mem;

  assign read_addr = read_addr_reg;
```

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

  vga_mem mem (
      .clk(clk),
      .ra(read_addr),
      .wa(write_addr),
      .write(write_mem),
      .wd(write_data),
      .rd(pixel_data)
  );

  // Write logic: on Avalon write, capture address and data
  always_ff @(posedge clk or posedge reset) begin
    if (reset) begin
      write_addr <= 19'd0;
      write_data <= 8'd0;
      write_mem  <= 1'b0;
    end else begin
      write_mem <= 1'b0;
      if (chipselect && write) begin
        write_addr <= address;
        write_data <= writedata;
        write_mem  <= 1'b1;
      end
    end
  end


always_ff @(posedge clk or posedge reset) begin
  if (reset) begin
    // pixel_x <= 10'd0;
    // pixel_y <= 9'd0;
    read_addr_reg <= 19'd0;
  end else begin
    if (VGA_BLANK_n) begin
      if ((hcount[10:1] < 10'd640) && (vcount < 10'd480)) begin
        // pixel_x <= hcount[10:1];
        // pixel_y <= vcount[8:0];
        read_addr_reg <= vcount[8:0] * 640 + hcount[10:1];
```

```
      end
    end
  end
end

  // Pixel color mapping: mock placeholder
  always_comb begin
    {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
    if (VGA_BLANK_n) begin
      case (pixel_data)
        8'd0: {VGA_R, VGA_G, VGA_B} = 24'h000000;
        8'd1: {VGA_R, VGA_G, VGA_B} = 24'hffffff;
        8'd2: {VGA_R, VGA_G, VGA_B} = 24'hffff00;
        8'd3: {VGA_R, VGA_G, VGA_B} = 24'h9e4f71;
        8'd4: {VGA_R, VGA_G, VGA_B} = 24'hd66f34;
        8'd5: {VGA_R, VGA_G, VGA_B} = 24'hf755aa;
        8'd6: {VGA_R, VGA_G, VGA_B} = 24'he5790d;
        8'd7: {VGA_R, VGA_G, VGA_B} = 24'h5d695a;
        8'd8: {VGA_R, VGA_G, VGA_B} = 24'hfcfcf9;
        8'd9: {VGA_R, VGA_G, VGA_B} = 24'hb0a392;
        8'd10: {VGA_R, VGA_G, VGA_B} = 24'h8f2655;
        8'd11: {VGA_R, VGA_G, VGA_B} = 24'hcc9ba0;
        8'd12: {VGA_R, VGA_G, VGA_B} = 24'hc5e773;
        8'd13: {VGA_R, VGA_G, VGA_B} = 24'h9f9d9c;
        8'd14: {VGA_R, VGA_G, VGA_B} = 24'hedeceb;
        8'd15: {VGA_R, VGA_G, VGA_B} = 24'ha15524;
        8'd16: {VGA_R, VGA_G, VGA_B} = 24'h27d1cd;
        8'd17: {VGA_R, VGA_G, VGA_B} = 24'h657751;
        8'd18: {VGA_R, VGA_G, VGA_B} = 24'h53a058;
        8'd19: {VGA_R, VGA_G, VGA_B} = 24'hb6b3a9;
        8'd20: {VGA_R, VGA_G, VGA_B} = 24'he0c76b;
        8'd21: {VGA_R, VGA_G, VGA_B} = 24'hc1bcbb;
        8'd22: {VGA_R, VGA_G, VGA_B} = 24'hfb6dcd;
        8'd23: {VGA_R, VGA_G, VGA_B} = 24'h9c6b93;
        8'd24: {VGA_R, VGA_G, VGA_B} = 24'h9ca999;
        8'd25: {VGA_R, VGA_G, VGA_B} = 24'habcf99;
        8'd26: {VGA_R, VGA_G, VGA_B} = 24'h22b8c2;
        8'd27: {VGA_R, VGA_G, VGA_B} = 24'ha5a2a0;
        8'd28: {VGA_R, VGA_G, VGA_B} = 24'ha69260;
        8'd29: {VGA_R, VGA_G, VGA_B} = 24'h787c81;
        8'd30: {VGA_R, VGA_G, VGA_B} = 24'h76c5c5;
        8'd31: {VGA_R, VGA_G, VGA_B} = 24'he09e6c;
        8'd32: {VGA_R, VGA_G, VGA_B} = 24'hf8c58c;
```

```
8'd33: {VGA_R, VGA_G, VGA_B} = 24'h84ae76;
8'd34: {VGA_R, VGA_G, VGA_B} = 24'hb57954;
8'd35: {VGA_R, VGA_G, VGA_B} = 24'h666254;
8'd36: {VGA_R, VGA_G, VGA_B} = 24'hdfb68b;
8'd37: {VGA_R, VGA_G, VGA_B} = 24'h148968;
8'd38: {VGA_R, VGA_G, VGA_B} = 24'h55c561;
8'd39: {VGA_R, VGA_G, VGA_B} = 24'hddd8cc;
8'd40: {VGA_R, VGA_G, VGA_B} = 24'h52b726;
8'd41: {VGA_R, VGA_G, VGA_B} = 24'h256f76;
8'd42: {VGA_R, VGA_G, VGA_B} = 24'hd39772;
8'd43: {VGA_R, VGA_G, VGA_B} = 24'hbf9306;
8'd44: {VGA_R, VGA_G, VGA_B} = 24'h705926;
8'd45: {VGA_R, VGA_G, VGA_B} = 24'h656362;
8'd46: {VGA_R, VGA_G, VGA_B} = 24'h6c2d15;
8'd47: {VGA_R, VGA_G, VGA_B} = 24'h847978;
8'd48: {VGA_R, VGA_G, VGA_B} = 24'hb38870;
8'd49: {VGA_R, VGA_G, VGA_B} = 24'h91722f;
8'd50: {VGA_R, VGA_G, VGA_B} = 24'h226013;
8'd51: {VGA_R, VGA_G, VGA_B} = 24'h727271;
8'd52: {VGA_R, VGA_G, VGA_B} = 24'he8902a;
8'd53: {VGA_R, VGA_G, VGA_B} = 24'h778d5f;
8'd54: {VGA_R, VGA_G, VGA_B} = 24'h9a690e;
8'd55: {VGA_R, VGA_G, VGA_B} = 24'hd5c4bb;
8'd56: {VGA_R, VGA_G, VGA_B} = 24'h3a9f54;
8'd57: {VGA_R, VGA_G, VGA_B} = 24'h7a4c1d;
8'd58: {VGA_R, VGA_G, VGA_B} = 24'h17a294;
8'd59: {VGA_R, VGA_G, VGA_B} = 24'he3e2d8;
8'd60: {VGA_R, VGA_G, VGA_B} = 24'h8a6936;
8'd61: {VGA_R, VGA_G, VGA_B} = 24'he48f07;
8'd62: {VGA_R, VGA_G, VGA_B} = 24'h602b09;
8'd63: {VGA_R, VGA_G, VGA_B} = 24'h6ca158;
8'd64: {VGA_R, VGA_G, VGA_B} = 24'h6f5f4e;
8'd65: {VGA_R, VGA_G, VGA_B} = 24'h9fbdc2;
8'd66: {VGA_R, VGA_G, VGA_B} = 24'h996a5c;
8'd67: {VGA_R, VGA_G, VGA_B} = 24'hf83ec8;
8'd68: {VGA_R, VGA_G, VGA_B} = 24'h919192;
8'd69: {VGA_R, VGA_G, VGA_B} = 24'hc4b5a0;
8'd70: {VGA_R, VGA_G, VGA_B} = 24'hd8ad26;
8'd71: {VGA_R, VGA_G, VGA_B} = 24'hf0a670;
8'd72: {VGA_R, VGA_G, VGA_B} = 24'h754d27;
8'd73: {VGA_R, VGA_G, VGA_B} = 24'hf6ca70;
8'd74: {VGA_R, VGA_G, VGA_B} = 24'hecebea;
8'd75: {VGA_R, VGA_G, VGA_B} = 24'hf8a012;
```

```
8'd76: {VGA_R, VGA_G, VGA_B} = 24'h1e7a83;
8'd77: {VGA_R, VGA_G, VGA_B} = 24'h3a444b;
8'd78: {VGA_R, VGA_G, VGA_B} = 24'hb5673a;
8'd79: {VGA_R, VGA_G, VGA_B} = 24'hedefed;
8'd80: {VGA_R, VGA_G, VGA_B} = 24'hbec0c0;
8'd81: {VGA_R, VGA_G, VGA_B} = 24'h7b726a;
8'd82: {VGA_R, VGA_G, VGA_B} = 24'h8b7550;
8'd83: {VGA_R, VGA_G, VGA_B} = 24'h463a35;
8'd84: {VGA_R, VGA_G, VGA_B} = 24'h6d2d15;
8'd85: {VGA_R, VGA_G, VGA_B} = 24'h64c52f;
8'd86: {VGA_R, VGA_G, VGA_B} = 24'h45403c;
8'd87: {VGA_R, VGA_G, VGA_B} = 24'h7d8185;
8'd88: {VGA_R, VGA_G, VGA_B} = 24'hcf950f;
8'd89: {VGA_R, VGA_G, VGA_B} = 24'hefab60;
8'd90: {VGA_R, VGA_G, VGA_B} = 24'h656058;
8'd91: {VGA_R, VGA_G, VGA_B} = 24'h5c9fa0;
8'd92: {VGA_R, VGA_G, VGA_B} = 24'he6a310;
8'd93: {VGA_R, VGA_G, VGA_B} = 24'hb99b89;
8'd94: {VGA_R, VGA_G, VGA_B} = 24'hfdfdfd;
8'd95: {VGA_R, VGA_G, VGA_B} = 24'h82bc2e;
8'd96: {VGA_R, VGA_G, VGA_B} = 24'hf1d7ab;
8'd97: {VGA_R, VGA_G, VGA_B} = 24'h67a41f;
8'd98: {VGA_R, VGA_G, VGA_B} = 24'h473e40;
8'd99: {VGA_R, VGA_G, VGA_B} = 24'h9ed35a;
8'd100: {VGA_R, VGA_G, VGA_B} = 24'hfdd130;
8'd101: {VGA_R, VGA_G, VGA_B} = 24'hdfd6cf;
8'd102: {VGA_R, VGA_G, VGA_B} = 24'hf7338e;
8'd103: {VGA_R, VGA_G, VGA_B} = 24'h92d227;
8'd104: {VGA_R, VGA_G, VGA_B} = 24'he4cf9d;
8'd105: {VGA_R, VGA_G, VGA_B} = 24'h923901;
8'd106: {VGA_R, VGA_G, VGA_B} = 24'hab8770;
8'd107: {VGA_R, VGA_G, VGA_B} = 24'hb6907a;
8'd108: {VGA_R, VGA_G, VGA_B} = 24'h99d2a3;
8'd109: {VGA_R, VGA_G, VGA_B} = 24'hebdaa2;
8'd110: {VGA_R, VGA_G, VGA_B} = 24'h595a5c;
8'd111: {VGA_R, VGA_G, VGA_B} = 24'heae0a2;
8'd112: {VGA_R, VGA_G, VGA_B} = 24'heee8e8;
8'd113: {VGA_R, VGA_G, VGA_B} = 24'hd86492;
8'd114: {VGA_R, VGA_G, VGA_B} = 24'hd499b4;
8'd115: {VGA_R, VGA_G, VGA_B} = 24'hb29992;
8'd116: {VGA_R, VGA_G, VGA_B} = 24'hede8ec;
8'd117: {VGA_R, VGA_G, VGA_B} = 24'h18a293;
8'd118: {VGA_R, VGA_G, VGA_B} = 24'h1dc3bc;
```

```
8'd119: {VGA_R, VGA_G, VGA_B} = 24'h91286a;
8'd120: {VGA_R, VGA_G, VGA_B} = 24'hae6f8a;
8'd121: {VGA_R, VGA_G, VGA_B} = 24'ha19a9b;
8'd122: {VGA_R, VGA_G, VGA_B} = 24'h2e7154;
8'd123: {VGA_R, VGA_G, VGA_B} = 24'hac8a2c;
8'd124: {VGA_R, VGA_G, VGA_B} = 24'h91612a;
8'd125: {VGA_R, VGA_G, VGA_B} = 24'hce7947;
8'd126: {VGA_R, VGA_G, VGA_B} = 24'hdecf77;
8'd127: {VGA_R, VGA_G, VGA_B} = 24'h88817d;
8'd128: {VGA_R, VGA_G, VGA_B} = 24'he5d49b;
8'd129: {VGA_R, VGA_G, VGA_B} = 24'h0f0c0c;
8'd130: {VGA_R, VGA_G, VGA_B} = 24'h4d7c80;
8'd131: {VGA_R, VGA_G, VGA_B} = 24'hcbae65;
8'd132: {VGA_R, VGA_G, VGA_B} = 24'hd6cc9d;
8'd133: {VGA_R, VGA_G, VGA_B} = 24'he7a410;
8'd134: {VGA_R, VGA_G, VGA_B} = 24'hc8b795;
8'd135: {VGA_R, VGA_G, VGA_B} = 24'h6d6a5c;
8'd136: {VGA_R, VGA_G, VGA_B} = 24'hb2ca87;
8'd137: {VGA_R, VGA_G, VGA_B} = 24'hfefefe;
8'd138: {VGA_R, VGA_G, VGA_B} = 24'he57e45;
8'd139: {VGA_R, VGA_G, VGA_B} = 24'h369236;
8'd140: {VGA_R, VGA_G, VGA_B} = 24'hd8d4d4;
8'd141: {VGA_R, VGA_G, VGA_B} = 24'h333c43;
8'd142: {VGA_R, VGA_G, VGA_B} = 24'h686768;
8'd143: {VGA_R, VGA_G, VGA_B} = 24'h7a5d4a;
8'd144: {VGA_R, VGA_G, VGA_B} = 24'ha7d0cf;
8'd145: {VGA_R, VGA_G, VGA_B} = 24'h957251;
8'd146: {VGA_R, VGA_G, VGA_B} = 24'h9d6a0d;
8'd147: {VGA_R, VGA_G, VGA_B} = 24'h686b68;
8'd148: {VGA_R, VGA_G, VGA_B} = 24'h79cc9d;
8'd149: {VGA_R, VGA_G, VGA_B} = 24'hf5f2e9;
8'd150: {VGA_R, VGA_G, VGA_B} = 24'hb78b74;
8'd151: {VGA_R, VGA_G, VGA_B} = 24'hfbfbf9;
8'd152: {VGA_R, VGA_G, VGA_B} = 24'hfdcc16;
8'd153: {VGA_R, VGA_G, VGA_B} = 24'hfac76d;
8'd154: {VGA_R, VGA_G, VGA_B} = 24'hc9a536;
8'd155: {VGA_R, VGA_G, VGA_B} = 24'heac9a7;
8'd156: {VGA_R, VGA_G, VGA_B} = 24'h23631d;
8'd157: {VGA_R, VGA_G, VGA_B} = 24'hd6b197;
8'd158: {VGA_R, VGA_G, VGA_B} = 24'ha87150;
8'd159: {VGA_R, VGA_G, VGA_B} = 24'hb2872f;
8'd160: {VGA_R, VGA_G, VGA_B} = 24'hffffec;
8'd161: {VGA_R, VGA_G, VGA_B} = 24'hc1c0be;
```

```
8'd162: {VGA_R, VGA_G, VGA_B} = 24'he3e7d8;
8'd163: {VGA_R, VGA_G, VGA_B} = 24'h81033a;
8'd164: {VGA_R, VGA_G, VGA_B} = 24'ha47354;
8'd165: {VGA_R, VGA_G, VGA_B} = 24'he8eae8;
8'd166: {VGA_R, VGA_G, VGA_B} = 24'h060805;
8'd167: {VGA_R, VGA_G, VGA_B} = 24'h90b35f;
8'd168: {VGA_R, VGA_G, VGA_B} = 24'haf968b;
8'd169: {VGA_R, VGA_G, VGA_B} = 24'hf47c03;
8'd170: {VGA_R, VGA_G, VGA_B} = 24'h706971;
8'd171: {VGA_R, VGA_G, VGA_B} = 24'h020702;
8'd172: {VGA_R, VGA_G, VGA_B} = 24'ha6a29b;
8'd173: {VGA_R, VGA_G, VGA_B} = 24'h74c54c;
8'd174: {VGA_R, VGA_G, VGA_B} = 24'hf3efe5;
8'd175: {VGA_R, VGA_G, VGA_B} = 24'h7ac520;
8'd176: {VGA_R, VGA_G, VGA_B} = 24'h95602b;
8'd177: {VGA_R, VGA_G, VGA_B} = 24'ha4a99e;
8'd178: {VGA_R, VGA_G, VGA_B} = 24'h35140a;
8'd179: {VGA_R, VGA_G, VGA_B} = 24'hcea168;
8'd180: {VGA_R, VGA_G, VGA_B} = 24'he9ddd3;
8'd181: {VGA_R, VGA_G, VGA_B} = 24'h9e1852;
8'd182: {VGA_R, VGA_G, VGA_B} = 24'heae9e4;
8'd183: {VGA_R, VGA_G, VGA_B} = 24'h6b6759;
8'd184: {VGA_R, VGA_G, VGA_B} = 24'h661e5e;
8'd185: {VGA_R, VGA_G, VGA_B} = 24'hf0ebe0;
8'd186: {VGA_R, VGA_G, VGA_B} = 24'h15994a;
8'd187: {VGA_R, VGA_G, VGA_B} = 24'h502b15;
8'd188: {VGA_R, VGA_G, VGA_B} = 24'h9ba1a0;
8'd189: {VGA_R, VGA_G, VGA_B} = 24'hd4ae9c;
8'd190: {VGA_R, VGA_G, VGA_B} = 24'h8e8a7b;
8'd191: {VGA_R, VGA_G, VGA_B} = 24'hefa2d8;
8'd192: {VGA_R, VGA_G, VGA_B} = 24'h487175;
8'd193: {VGA_R, VGA_G, VGA_B} = 24'hd5eba6;
8'd194: {VGA_R, VGA_G, VGA_B} = 24'haca99f;
8'd195: {VGA_R, VGA_G, VGA_B} = 24'h68512e;
8'd196: {VGA_R, VGA_G, VGA_B} = 24'hae6e8a;
8'd197: {VGA_R, VGA_G, VGA_B} = 24'hc0bfc0;
8'd198: {VGA_R, VGA_G, VGA_B} = 24'ha59a9f;
8'd199: {VGA_R, VGA_G, VGA_B} = 24'h693048;
8'd200: {VGA_R, VGA_G, VGA_B} = 24'ha0e1dd;
8'd201: {VGA_R, VGA_G, VGA_B} = 24'hd08d69;
8'd202: {VGA_R, VGA_G, VGA_B} = 24'h61d0d3;
8'd203: {VGA_R, VGA_G, VGA_B} = 24'h877e81;
8'd204: {VGA_R, VGA_G, VGA_B} = 24'hdef2f0;
```

```
            8'd205: {VGA_R, VGA_G, VGA_B} = 24'hac7b67;
            8'd206: {VGA_R, VGA_G, VGA_B} = 24'h4bb7c6;
            8'd207: {VGA_R, VGA_G, VGA_B} = 24'hefba91;
            8'd208: {VGA_R, VGA_G, VGA_B} = 24'hcc1d60;
            8'd209: {VGA_R, VGA_G, VGA_B} = 24'h14994c;
            8'd210: {VGA_R, VGA_G, VGA_B} = 24'had9163;
            8'd211: {VGA_R, VGA_G, VGA_B} = 24'h53a3a6;
            8'd212: {VGA_R, VGA_G, VGA_B} = 24'hcbb262;
            8'd213: {VGA_R, VGA_G, VGA_B} = 24'h80bdc3;
            8'd214: {VGA_R, VGA_G, VGA_B} = 24'h6e6489;
            8'd215: {VGA_R, VGA_G, VGA_B} = 24'he8cf6f;
            8'd216: {VGA_R, VGA_G, VGA_B} = 24'hfcf9f6;
            8'd217: {VGA_R, VGA_G, VGA_B} = 24'hac9161;
            8'd218: {VGA_R, VGA_G, VGA_B} = 24'hd99ba5;
            8'd219: {VGA_R, VGA_G, VGA_B} = 24'h710132;
            8'd220: {VGA_R, VGA_G, VGA_B} = 24'ha59060;
            8'd221: {VGA_R, VGA_G, VGA_B} = 24'hac833d;
            8'd222: {VGA_R, VGA_G, VGA_B} = 24'h0b0805;
            8'd223: {VGA_R, VGA_G, VGA_B} = 24'ha19f98;
            8'd224: {VGA_R, VGA_G, VGA_B} = 24'h936d52;
            8'd225: {VGA_R, VGA_G, VGA_B} = 24'h4b7918;
            8'd226: {VGA_R, VGA_G, VGA_B} = 24'he1ad68;
            8'd227: {VGA_R, VGA_G, VGA_B} = 24'hafa8a3;
            8'd228: {VGA_R, VGA_G, VGA_B} = 24'h8fcc5e;
            8'd229: {VGA_R, VGA_G, VGA_B} = 24'h99670c;
            8'd230: {VGA_R, VGA_G, VGA_B} = 24'h947946;
            8'd231: {VGA_R, VGA_G, VGA_B} = 24'h6e1e2c;
            8'd232: {VGA_R, VGA_G, VGA_B} = 24'he1ae97;
            8'd233: {VGA_R, VGA_G, VGA_B} = 24'hb08c5e;
            8'd234: {VGA_R, VGA_G, VGA_B} = 24'h3c6a0f;
            8'd235: {VGA_R, VGA_G, VGA_B} = 24'hcda96c;
            8'd236: {VGA_R, VGA_G, VGA_B} = 24'hc3dbb5;
            8'd237: {VGA_R, VGA_G, VGA_B} = 24'h090406;
            8'd238: {VGA_R, VGA_G, VGA_B} = 24'h1d9ca6;
            8'd239: {VGA_R, VGA_G, VGA_B} = 24'ha4acac;
            8'd240: {VGA_R, VGA_G, VGA_B} = 24'hcf6269;
            8'd241: {VGA_R, VGA_G, VGA_B} = 24'h996126;
            8'd242: {VGA_R, VGA_G, VGA_B} = 24'hece8e3;
            8'd243: {VGA_R, VGA_G, VGA_B} = 24'h427137;
            default: {VGA_R, VGA_G, VGA_B} = 24'h000000;  // Default to
black
        endcase
    end
```

```
  end

endmodule

module vga_mem (
  input logic clk,
  input logic [18:0] ra,  // 19-bit address for 640*480 pixels
  input logic [18:0] wa,                      // 19-bit address for
write
  input logic write,
  input logic [7:0] wd,   // 8-bit data index
  output logic [7:0] rd   // 8-bit data index
);

  logic [7:0] data[307199:0];  // 640 * 480 = 307200 pixels
  always_ff @(posedge clk) begin
    if (write) data[wa] <= wd;
    rd <= data[ra];
  end
endmodule

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

  /*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0             1279       1599 0
 *             _______________              ________
 * ___________|    Video      |____________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *       _____________________      _____________
```

```
 * |____|       VGA_HS          |____|
 */
  // Parameters for hcount
  parameter HACTIVE      = 11'd 1280,
            HFRONT_PORCH = 11'd 32,
            HSYNC        = 11'd 192,
            HBACK_PORCH  = 11'd 96,
            HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                           HBACK_PORCH; // 1600

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

  logic endOfLine;

  always_ff @(posedge clk50 or posedge reset)
    if (reset) hcount <= 0;
    else if (endOfLine) hcount <= 0;
    else hcount <= hcount + 11'd1;

  assign endOfLine = hcount == HTOTAL - 1;

  logic endOfField;

  always_ff @(posedge clk50 or posedge reset)
    if (reset) vcount <= 0;
    else if (endOfLine)
      if (endOfField) vcount <= 0;
      else vcount <= vcount + 10'd1;

  assign endOfField = vcount == VTOTAL - 1;

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

```
  assign VGA_SYNC_n = 1'b0;  // For putting sync on the green signal;
unused

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

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

endmodule
```

soc_system_top.sv

```
// ====================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ====================================================================
//
// Modified 2019 by Stephen A. Edwards
//
// 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
module soc_system_top(

 ///////// ADC /////////
 inout           ADC_CS_N,
 output         ADC_DIN,
 input           ADC_DOUT,
 output         ADC_SCLK,

 ///////// AUD /////////
 input           AUD_ADCDAT,
 inout           AUD_ADCLRCK,
 inout           AUD_BCLK,
 output         AUD_DACDAT,
 inout           AUD_DACLRCK,
 output         AUD_XCK,

 ///////// CLOCK2 /////////
 input           CLOCK2_50,

 ///////// CLOCK3 /////////
 input           CLOCK3_50,

 ///////// CLOCK4 /////////
 input           CLOCK4_50,

 ///////// CLOCK /////////
 input           CLOCK_50,

 ///////// DRAM /////////
 output [12:0] DRAM_ADDR,
 output [1:0]  DRAM_BA,
```

```
output        DRAM_CAS_N,
output        DRAM_CKE,
output        DRAM_CLK,
output        DRAM_CS_N,
inout [15:0]  DRAM_DQ,
output        DRAM_LDQM,
output        DRAM_RAS_N,
output        DRAM_UDQM,
output        DRAM_WE_N,

///////// FAN /////////
output        FAN_CTRL,

///////// FPGA /////////
output        FPGA_I2C_SCLK,
inout          FPGA_I2C_SDAT,

///////// GPIO /////////
inout [35:0]  GPIO_0,
inout [35:0]  GPIO_1,

///////// HEX0 /////////
output [6:0]  HEX0,

///////// HEX1 /////////
output [6:0]  HEX1,

///////// HEX2 /////////
output [6:0]  HEX2,

///////// HEX3 /////////
output [6:0]  HEX3,

///////// HEX4 /////////
output [6:0]  HEX4,

///////// HEX5 /////////
output [6:0]  HEX5,

///////// HPS /////////
inout          HPS_CONV_USB_N,
output [14:0] HPS_DDR3_ADDR,
output [2:0]  HPS_DDR3_BA,
```

```
output        HPS_DDR3_CAS_N,
output        HPS_DDR3_CKE,
output        HPS_DDR3_CK_N,
output        HPS_DDR3_CK_P,
output        HPS_DDR3_CS_N,
output [3:0]  HPS_DDR3_DM,
inout [31:0]  HPS_DDR3_DQ,
inout [3:0]   HPS_DDR3_DQS_N,
inout [3:0]   HPS_DDR3_DQS_P,
output        HPS_DDR3_ODT,
output        HPS_DDR3_RAS_N,
output        HPS_DDR3_RESET_N,
input          HPS_DDR3_RZQ,
output        HPS_DDR3_WE_N,
output        HPS_ENET_GTX_CLK,
inout          HPS_ENET_INT_N,
output        HPS_ENET_MDC,
inout          HPS_ENET_MDIO,
input          HPS_ENET_RX_CLK,
input [3:0]   HPS_ENET_RX_DATA,
input          HPS_ENET_RX_DV,
output [3:0]  HPS_ENET_TX_DATA,
output        HPS_ENET_TX_EN,
inout          HPS_GSENSOR_INT,
inout          HPS_I2C1_SCLK,
inout          HPS_I2C1_SDAT,
inout          HPS_I2C2_SCLK,
inout          HPS_I2C2_SDAT,
inout          HPS_I2C_CONTROL,
inout          HPS_KEY,
inout          HPS_LED,
inout          HPS_LTC_GPIO,
output        HPS_SD_CLK,
inout          HPS_SD_CMD,
inout [3:0]   HPS_SD_DATA,
output        HPS_SPIM_CLK,
input          HPS_SPIM_MISO,
output        HPS_SPIM_MOSI,
inout          HPS_SPIM_SS,
input          HPS_UART_RX,
output        HPS_UART_TX,
input          HPS_USB_CLKOUT,
inout [7:0]   HPS_USB_DATA,
```

```
 input           HPS_USB_DIR,
 input           HPS_USB_NXT,
 output         HPS_USB_STP,

 ///////// IRDA /////////
 input           IRDA_RXD,
 output         IRDA_TXD,

 ///////// KEY /////////
 input [3:0]    KEY,

 ///////// LEDR /////////
 output [9:0]   LEDR,

 ///////// PS2 /////////
 inout           PS2_CLK,
 inout           PS2_CLK2,
 inout           PS2_DAT,
 inout           PS2_DAT2,

 ///////// SW /////////
 input [9:0]    SW,

 ///////// TD /////////
 input           TD_CLK27,
 input [7:0]    TD_DATA,
 input           TD_HS,
 output         TD_RESET_N,
 input           TD_VS,


 ///////// VGA /////////
 output [7:0]   VGA_B,
 output         VGA_BLANK_N,
 output         VGA_CLK,
 output [7:0]   VGA_G,
 output         VGA_HS,
 output [7:0]   VGA_R,
 output         VGA_SYNC_N,
 output         VGA_VS
);

   soc_system soc_system0(
```

```
.clk_clk                       ( CLOCK_50 ),
.reset_reset_n                 ( 1'b1 ),

.hps_ddr3_mem_a                ( HPS_DDR3_ADDR ),
.hps_ddr3_mem_ba               ( HPS_DDR3_BA ),
.hps_ddr3_mem_ck               ( HPS_DDR3_CK_P ),
.hps_ddr3_mem_ck_n             ( HPS_DDR3_CK_N ),
.hps_ddr3_mem_cke              ( HPS_DDR3_CKE ),
.hps_ddr3_mem_cs_n             ( HPS_DDR3_CS_N ),
.hps_ddr3_mem_ras_n            ( HPS_DDR3_RAS_N ),
.hps_ddr3_mem_cas_n            ( HPS_DDR3_CAS_N ),
.hps_ddr3_mem_we_n             ( HPS_DDR3_WE_N ),
.hps_ddr3_mem_reset_n          ( HPS_DDR3_RESET_N ),
.hps_ddr3_mem_dq               ( HPS_DDR3_DQ ),
.hps_ddr3_mem_dqs              ( HPS_DDR3_DQS_P ),
.hps_ddr3_mem_dqs_n            ( HPS_DDR3_DQS_N ),
.hps_ddr3_mem_odt              ( HPS_DDR3_ODT ),
.hps_ddr3_mem_dm               ( HPS_DDR3_DM ),
.hps_ddr3_oct_rzqin            ( HPS_DDR3_RZQ ),

.hps_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK ),
.hps_hps_io_emac1_inst_TXD0   ( HPS_ENET_TX_DATA[0] ),
.hps_hps_io_emac1_inst_TXD1   ( HPS_ENET_TX_DATA[1] ),
.hps_hps_io_emac1_inst_TXD2   ( HPS_ENET_TX_DATA[2] ),
.hps_hps_io_emac1_inst_TXD3   ( HPS_ENET_TX_DATA[3] ),
.hps_hps_io_emac1_inst_RXD0   ( HPS_ENET_RX_DATA[0] ),
.hps_hps_io_emac1_inst_MDIO   ( HPS_ENET_MDIO  ),
.hps_hps_io_emac1_inst_MDC    ( HPS_ENET_MDC   ),
.hps_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV ),
.hps_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN ),
.hps_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK ),
.hps_hps_io_emac1_inst_RXD1   ( HPS_ENET_RX_DATA[1]  ),
.hps_hps_io_emac1_inst_RXD2   ( HPS_ENET_RX_DATA[2]  ),
.hps_hps_io_emac1_inst_RXD3   ( HPS_ENET_RX_DATA[3]  ),

.hps_hps_io_sdio_inst_CMD     ( HPS_SD_CMD          ),
.hps_hps_io_sdio_inst_D0      ( HPS_SD_DATA[0]      ),
.hps_hps_io_sdio_inst_D1      ( HPS_SD_DATA[1]      ),
.hps_hps_io_sdio_inst_CLK     ( HPS_SD_CLK          ),
.hps_hps_io_sdio_inst_D2      ( HPS_SD_DATA[2]      ),
.hps_hps_io_sdio_inst_D3      ( HPS_SD_DATA[3]      ),

.hps_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0]     ),
```

```
.hps_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1]     ),
.hps_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2]     ),
.hps_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3]     ),
.hps_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4]     ),
.hps_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5]     ),
.hps_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6]     ),
.hps_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7]     ),
.hps_hps_io_usb1_inst_CLK     ( HPS_USB_CLKOUT      ),
.hps_hps_io_usb1_inst_STP     ( HPS_USB_STP         ),
.hps_hps_io_usb1_inst_DIR     ( HPS_USB_DIR         ),
.hps_hps_io_usb1_inst_NXT     ( HPS_USB_NXT         ),

.hps_hps_io_spim1_inst_CLK    ( HPS_SPIM_CLK  ),
.hps_hps_io_spim1_inst_MOSI   ( HPS_SPIM_MOSI ),
.hps_hps_io_spim1_inst_MISO   ( HPS_SPIM_MISO ),
.hps_hps_io_spim1_inst_SS0    ( HPS_SPIM_SS   ),

.hps_hps_io_uart0_inst_RX     ( HPS_UART_RX     ),
.hps_hps_io_uart0_inst_TX     ( HPS_UART_TX     ),

.hps_hps_io_i2c0_inst_SDA     ( HPS_I2C1_SDAT     ),
.hps_hps_io_i2c0_inst_SCL     ( HPS_I2C1_SCLK     ),

.hps_hps_io_i2c1_inst_SDA     ( HPS_I2C2_SDAT     ),
.hps_hps_io_i2c1_inst_SCL     ( HPS_I2C2_SCLK     ),

.hps_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N ),
.hps_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N ),
.hps_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO ),

.hps_hps_io_gpio_inst_GPIO48  ( HPS_I2C_CONTROL ),
.hps_hps_io_gpio_inst_GPIO53  ( HPS_LED ),
.hps_hps_io_gpio_inst_GPIO54  ( HPS_KEY ),
.hps_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT ),

.aud_BCLK                     ( AUD_BCLK ),
.aud_DACDAT                   ( AUD_DACDAT ),
.aud_DACLRCK                  ( AUD_DACLRCK ),
.aud_clk_clk                  ( AUD_XCK ),

.aud_config_SDAT              ( FPGA_I2C_SDAT ),
.aud_config_SCLK              ( FPGA_I2C_SCLK ),
```

```
    .vga_r (VGA_R),
    .vga_g (VGA_G),
    .vga_b (VGA_B),
    .vga_clk (VGA_CLK),
    .vga_hs (VGA_HS),
    .vga_vs (VGA_VS),
    .vga_blank_n (VGA_BLANK_N),
    .vga_sync_n (VGA_SYNC_N)
);

 // The following quiet the "no driver" warnings for output
 // pins and should be removed if you use any of these peripherals

 assign ADC_CS_N = SW[1] ? SW[0] : 1'bZ;
 assign ADC_DIN = SW[0];
 assign ADC_SCLK = SW[0];

 // assign AUD_ADCLRCK = SW[1] ? SW[0] : 1'bZ;
 // assign AUD_BCLK = SW[1] ? SW[0] : 1'bZ;
 // assign AUD_DACDAT = SW[0];
 // assign AUD_DACLRCK = SW[1] ? SW[0] : 1'bZ;
 // assign AUD_XCK = SW[0];

 assign DRAM_ADDR = { 13{ SW[0] } };
 assign DRAM_BA = { 2{ SW[0] } };
 assign DRAM_DQ = SW[1] ? { 16{ SW[0] } } : { 16{ 1'bZ } };
 assign {DRAM_CAS_N, DRAM_CKE, DRAM_CLK, DRAM_CS_N,
    DRAM_LDQM, DRAM_RAS_N, DRAM_UDQM, DRAM_WE_N} = { 8{SW[0]} };

 assign FAN_CTRL = SW[0];

 // assign FPGA_I2C_SCLK = SW[0];
 // assign FPGA_I2C_SDAT = SW[1] ? SW[0] : 1'bZ;

 assign GPIO_0 = SW[1] ? { 36{ SW[0] } } : { 36{ 1'bZ } };
 assign GPIO_1 = SW[1] ? { 36{ SW[0] } } : { 36{ 1'bZ } };

 assign HEX0 = { 7{ SW[1] } };
 assign HEX1 = { 7{ SW[2] } };
 assign HEX2 = { 7{ SW[3] } };
 assign HEX3 = { 7{ SW[4] } };
 assign HEX4 = { 7{ SW[5] } };
 assign HEX5 = { 7{ SW[6] } };
```

```
    assign IRDA_TXD = SW[0];

    assign LEDR = { 10{SW[7]} };

    assign PS2_CLK = SW[1] ? SW[0] : 1'bZ;
    assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
    assign PS2_DAT = SW[1] ? SW[0] : 1'bZ;
    assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;

    assign TD_RESET_N = SW[0];

    // assign {VGA_R, VGA_G, VGA_B} = { 24{ SW[0] } };
    // assign {VGA_BLANK_N, VGA_CLK, VGA_HS, VGA_SYNC_N, VGA_VS} = { 5{
SW[0] } };


endmodule
```

9.2 Software

audio.c

```
#include <linux/module.h>
#include <linux/platform_device.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/miscdevice.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_device.h>
#include <linux/delay.h>
#include <linux/slab.h>
#include "audio.h"

#define DRIVER_NAME "audio"

/* Register Offsets */
#define AUDIO_CONTROL     0x00
#define AUDIO_FIFOSPACE   0x04
#define AUDIO_LEFTDATA    0x08
#define AUDIO_RIGHTDATA   0x0C
```

```
/* Control Bits */
#define CONTROL_CW (1 << 3)
#define CONTROL_WE (1 << 1)

static void __iomem *audio_base;

static long audio_ioctl(struct file *f, unsigned int cmd, unsigned long
arg)
{
    audio_sample_t smp;

    switch (cmd) {
    case AUDIO_WRITE_SAMPLE:
        if (copy_from_user(&smp, (void __user *)arg, sizeof(smp)))
            return -EFAULT;

        uint32_t fifospace = ioread32(audio_base + AUDIO_FIFOSPACE);
        uint8_t ws_l = (fifospace >> 24) & 0xFF;
        uint8_t ws_r = (fifospace >> 16) & 0xFF;
        if (ws_l > 0 && ws_r > 0) {
            iowrite32((uint32_t)(uint16_t)smp.left, audio_base +
AUDIO_LEFTDATA);
            iowrite32((uint32_t)(uint16_t)smp.right, audio_base +
AUDIO_RIGHTDATA);
        }

        break;

    case AUDIO_READ_STATUS: {
        int status = ioread32(audio_base + AUDIO_FIFOSPACE);
        if (copy_to_user((void __user *)arg, &status, sizeof(int)))
            return -EFAULT;
        break;
    }

    default:
        return -EINVAL;
    }

    return 0;
}
```

```
static ssize_t audio_write(struct file *file, const char __user *buf,
size_t count, loff_t *ppos)
{
    size_t i;
    uint8_t sample[4];

    for (i = 0; i + 3 < count; i += 4) {
        if (copy_from_user(sample, buf + i, 4))
            return -EFAULT;

        int16_t left  = sample[0] | (sample[1] << 8);
        int16_t right = sample[2] | (sample[3] << 8);

        // wait until FIFO has space
        int retries = 1000;
        while (retries--) {
            uint32_t fifospace = ioread32(audio_base +
AUDIO_FIFOSPACE);
            uint8_t ws_l = (fifospace >> 24) & 0xFF;
            uint8_t ws_r = (fifospace >> 16) & 0xFF;

            if (ws_l > 0 && ws_r > 0) {
                iowrite32((uint32_t)(uint16_t)left, audio_base +
AUDIO_LEFTDATA);
                iowrite32((uint32_t)(uint16_t)right, audio_base +
AUDIO_RIGHTDATA);
                break;
            }

            udelay(10);
        }

        // still no spcae when out of time, exit
        if (retries <= 0)
            break;
    }

    return i;  // return byte number written in
}


static const struct file_operations audio_fops = {
    .owner = THIS_MODULE,
```

```
    .unlocked_ioctl = audio_ioctl,
    .write = audio_write,
};

static struct miscdevice audio_misc_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = "audio",
    .fops = &audio_fops,
};

static int audio_probe(struct platform_device *pdev)
{
    struct resource *res;
    int ret;

    pr_info("audio: probe()\n");

    res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
    if (!res) {
        pr_err("audio: failed to get mem resource\n");
        return -ENODEV;
    }

    audio_base = devm_ioremap_resource(&pdev->dev, res);
    if (IS_ERR(audio_base)) {
        pr_err("audio: ioremap failed\n");
        return PTR_ERR(audio_base);
    }

    // Reset FIFO
    iowrite32(CONTROL_CW | CONTROL_WE, audio_base + AUDIO_CONTROL);
    udelay(10);
    iowrite32(CONTROL_WE, audio_base + AUDIO_CONTROL);

    ret = misc_register(&audio_misc_device);
    if (ret) {
        pr_err("audio: misc_register failed\n");
        return ret;
    }

    pr_info("audio: driver loaded, base = %p\n", audio_base);
    return 0;
}
```

```c
static int audio_remove(struct platform_device *pdev)
{
    misc_deregister(&audio_misc_device);
    pr_info("audio: removed\n");
    return 0;
}

static const struct of_device_id audio_of_match[] = {
    { .compatible = "altr,audio-0.18" },
    {},
};
MODULE_DEVICE_TABLE(of, audio_of_match);

static struct platform_driver audio_driver = {
    .probe = audio_probe,
    .remove = audio_remove,
    .driver = {
        .name = DRIVER_NAME,
        .of_match_table = audio_of_match,
    },
};
module_platform_driver(audio_driver);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Yangyang Zhang, Columbia University");
MODULE_DESCRIPTION("Minimal audio driver with FIFO write for DE1-SoC");
```

audio.h

```c
#ifndef _AUDIO_H
#define _AUDIO_H

#ifndef __KERNEL__
    #include <stdint.h>
#endif

#include <linux/ioctl.h>

#define AUDIO_MAGIC 'A'

typedef struct __attribute__((packed)){
    int16_t left;
    int16_t right;
```

```
} audio_sample_t;

#define AUDIO_WRITE_SAMPLE _IOW(AUDIO_MAGIC, 1, audio_sample_t)
#define AUDIO_READ_STATUS  _IOR(AUDIO_MAGIC, 2, int)

#endif
```

audio_user.c

```
#include "audio_user.h"
#include "audio.h"
#include <stdint.h>     // for int16_t, uint8_t
#include <stdio.h>      // for FILE, fopen, fread, fclose, perror
#include <stdlib.h>     // for malloc, free
#include <string.h>     // for memset if needed
#include <unistd.h>     // for usleep, close, etc.
#include <fcntl.h>      // for open, O_WRONLY
#include <sys/ioctl.h>  // for ioctl
#include <pthread.h>    // for pthreads
#include <stdatomic.h>  // for atomic variables

#define DEVICE       "/dev/audio"
#define SAMPLE_RATE  48000
#define FRAME_BYTES  4  // 2 bytes for left + 2 bytes for right
#define CHUNK_SIZE   512

typedef struct {
    uint8_t *data;
    size_t size;
    size_t offset;
    size_t sample_rate;
    size_t frame_bytes;
} audio_buffer_t;

static audio_buffer_t audio_buf = {0};

static pthread_t audio_thread;
static atomic_int playing = 0;              // Atomic flag: whether
audio is playing
static atomic_size_t play_start_ms = 0;     // Start time in ms
static atomic_size_t play_current_ms = 0;   // Current playback
position in ms
static atomic_int audio_finished = 0;
```

```c
int load_audio(const char *filename) {
    // printf("Loading audio file: %s\n", filename);
    FILE *f = fopen(filename, "rb");
    if (!f) {
        perror("Failed to open audio file");
        return -1;
    }

    fseek(f, 0, SEEK_END);
    audio_buf.size = ftell(f);
    rewind(f);

    audio_buf.data = malloc(audio_buf.size);
    if (!audio_buf.data) {
        perror("Memory allocation failed");
        fclose(f);
        return -1;
    }

    fread(audio_buf.data, 1, audio_buf.size, f);
    fclose(f);

    audio_buf.offset = 0;
    audio_buf.sample_rate = SAMPLE_RATE;
    audio_buf.frame_bytes = FRAME_BYTES;

    return 0;
}

void* play_thread_func(void *arg) {
    size_t offset_ms = atomic_load(&play_start_ms);
    size_t offset_bytes = (offset_ms * SAMPLE_RATE / 1000) *
FRAME_BYTES;

    int fd_audio = open(DEVICE, O_WRONLY);
    if (fd_audio < 0) {
        perror("Failed to open /dev/audio");
        return NULL;
    }

    double ms_acc = offset_ms;
    size_t pos = offset_bytes;
```

```
    const size_t chunk_samples = 64; // 64 stereo samples (128 FIFO
words)
    const size_t chunk_bytes = chunk_samples * FRAME_BYTES;

    while (pos < audio_buf.size && atomic_load(&playing)) {
        // Check if FIFO is ready
        int status;
        if (ioctl(fd_audio, AUDIO_READ_STATUS, &status) < 0) {
            perror("AUDIO_READ_STATUS failed");
            break;
        }

        int ws_l = (status >> 24) & 0xFF;
        int ws_r = (status >> 16) & 0xFF;
        int available_pairs = ws_l < ws_r ? ws_l : ws_r;

        if (available_pairs < 80) {
            usleep(200); // Wait a bit before trying again (non-busy
wait)
            continue;
        }

        size_t remaining = audio_buf.size - pos;
        size_t to_write = remaining < chunk_bytes ? remaining :
chunk_bytes;

        ssize_t written = write(fd_audio, audio_buf.data + pos,
to_write);
        if (written <= 0) {
            perror("Audio write failed");
            break;
        }

        pos += written;

        double ms_written = (1000.0 * written) / (SAMPLE_RATE *
FRAME_BYTES);
        ms_acc += ms_written;
        atomic_store(&play_current_ms, (size_t)ms_acc);

        usleep(1000);  // Control stream frequency: check roughly every
1ms
    }
```

```
    close(fd_audio);
    atomic_store(&playing, 0);

    if (pos >= audio_buf.size) {
        atomic_store(&audio_finished, 1);
    } else {
        atomic_store(&audio_finished, 0);
    }
    return NULL;
}

int play_audio(int time_ms) {
    if (!audio_buf.data) return -1;

    atomic_store(&play_start_ms, time_ms);
    atomic_store(&play_current_ms, time_ms);
    atomic_store(&audio_finished, 0);
    atomic_store(&playing, 1);

    if (pthread_create(&audio_thread, NULL, play_thread_func, NULL) !=
0) {
        perror("Failed to create playback thread");
        return -1;
    }

    pthread_detach(audio_thread);  // Automatically release resources
    return 0;
}

int pulse_audio(void) {
    atomic_store(&audio_finished, 0);
    atomic_store(&playing, 0);
    return atomic_load(&play_current_ms);
}

int reset_audio(void) {
    atomic_store(&play_start_ms, 0);
    atomic_store(&play_current_ms, 0);
    atomic_store(&audio_finished, 0);
    return 0;
}
```

```c
int audio_time(void) {
    return atomic_load(&play_current_ms);
}

int is_audio_finished(void) {
    return atomic_load(&audio_finished);
}

void free_audio(void) {
    atomic_store(&playing, 0);
    atomic_store(&audio_finished, 0);
    atomic_store(&play_start_ms, 0);
    atomic_store(&play_current_ms, 0);
    if (audio_buf.data) {
        free(audio_buf.data);
        audio_buf.data = NULL;
        audio_buf.size = 0;
    }
}
```

audio_user.h

```c
#ifndef AUDIO_USER_H
#define AUDIO_USER_H

int load_audio(const char *filename);
int play_audio(int time_ms);
int pulse_audio(void);
int reset_audio(void);
int audio_time(void);
int is_audio_finished(void);
void free_audio(void);

#endif
```

color.c

```c
#include "color.h"

const uint8_t color_palette[256][3] = {
  {0, 0, 0},
  {255, 255, 255},
  {255, 255, 0},
  {158, 79, 113},
```

```
{214, 111, 52},
{247, 85, 170},
{229, 121, 13},
{93, 105, 90},
{252, 252, 249},
{176, 163, 146},
{143, 38, 85},
{204, 155, 160},
{197, 231, 115},
{159, 157, 156},
{237, 236, 235},
{161, 85, 36},
{39, 209, 205},
{101, 119, 81},
{83, 160, 88},
{182, 179, 169},
{224, 199, 107},
{193, 188, 187},
{251, 109, 205},
{156, 107, 147},
{156, 169, 153},
{171, 207, 153},
{34, 184, 194},
{165, 162, 160},
{166, 146, 96},
{120, 124, 129},
{118, 197, 197},
{224, 158, 108},
{248, 197, 140},
{132, 174, 118},
{181, 121, 84},
{102, 98, 84},
{223, 182, 139},
{20, 137, 104},
{85, 197, 97},
{221, 216, 204},
{82, 183, 38},
{37, 111, 118},
{211, 151, 114},
{191, 147, 6},
{112, 89, 38},
{101, 99, 98},
{108, 45, 21},
```

```
{132, 121, 120},
{179, 136, 112},
{145, 114, 47},
{34, 96, 19},
{114, 114, 113},
{232, 144, 42},
{119, 141, 95},
{154, 105, 14},
{213, 196, 187},
{58, 159, 84},
{122, 76, 29},
{23, 162, 148},
{227, 226, 216},
{138, 105, 54},
{228, 143, 7},
{96, 43, 9},
{108, 161, 88},
{111, 95, 78},
{159, 189, 194},
{153, 106, 92},
{248, 62, 200},
{145, 145, 146},
{196, 181, 160},
{216, 173, 38},
{240, 166, 112},
{117, 77, 39},
{246, 202, 112},
{236, 235, 234},
{248, 160, 18},
{30, 122, 131},
{58, 68, 75},
{181, 103, 58},
{237, 239, 237},
{190, 192, 192},
{123, 114, 106},
{139, 117, 80},
{70, 58, 53},
{109, 45, 21},
{100, 197, 47},
{69, 64, 60},
{125, 129, 133},
{207, 149, 15},
{239, 171, 96},
```

```
{101, 96, 88},
{92, 159, 160},
{230, 163, 16},
{185, 155, 137},
{253, 253, 253},
{130, 188, 46},
{241, 215, 171},
{103, 164, 31},
{71, 62, 64},
{158, 211, 90},
{253, 209, 48},
{223, 214, 207},
{247, 51, 142},
{146, 210, 39},
{228, 207, 157},
{146, 57, 1},
{171, 135, 112},
{182, 144, 122},
{153, 210, 163},
{235, 218, 162},
{89, 90, 92},
{234, 224, 162},
{238, 232, 232},
{216, 100, 146},
{212, 153, 180},
{178, 153, 146},
{237, 232, 236},
{24, 162, 147},
{29, 195, 188},
{145, 40, 106},
{174, 111, 138},
{161, 154, 155},
{46, 113, 84},
{172, 138, 44},
{145, 97, 42},
{206, 121, 71},
{222, 207, 119},
{136, 129, 125},
{229, 212, 155},
{15, 12, 12},
{77, 124, 128},
{203, 174, 101},
{214, 204, 157},
```

```
{231, 164, 16},
{200, 183, 149},
{109, 106, 92},
{178, 202, 135},
{254, 254, 254},
{229, 126, 69},
{54, 146, 54},
{216, 212, 212},
{51, 60, 67},
{104, 103, 104},
{122, 93, 74},
{167, 208, 207},
{149, 114, 81},
{157, 106, 13},
{104, 107, 104},
{121, 204, 157},
{245, 242, 233},
{183, 139, 116},
{251, 251, 249},
{253, 204, 22},
{250, 199, 109},
{201, 165, 54},
{234, 201, 167},
{35, 99, 29},
{214, 177, 151},
{168, 113, 80},
{178, 135, 47},
{255, 255, 236},
{193, 192, 190},
{227, 231, 216},
{129, 3, 58},
{164, 115, 84},
{232, 234, 232},
{6, 8, 5},
{144, 179, 95},
{175, 150, 139},
{244, 124, 3},
{112, 105, 113},
{2, 7, 2},
{166, 162, 155},
{116, 197, 76},
{243, 239, 229},
{122, 197, 32},
```

```
{149, 96, 43},
{164, 169, 158},
{53, 20, 10},
{206, 161, 104},
{233, 221, 211},
{158, 24, 82},
{234, 233, 228},
{107, 103, 89},
{102, 30, 94},
{240, 235, 224},
{21, 153, 74},
{80, 43, 21},
{155, 161, 160},
{212, 174, 156},
{142, 138, 123},
{239, 162, 216},
{72, 113, 117},
{213, 235, 166},
{172, 169, 159},
{104, 81, 46},
{174, 110, 138},
{192, 191, 192},
{165, 154, 159},
{105, 48, 72},
{160, 225, 221},
{208, 141, 105},
{97, 208, 211},
{135, 126, 129},
{222, 242, 240},
{172, 123, 103},
{75, 183, 198},
{239, 186, 145},
{204, 29, 96},
{20, 153, 76},
{173, 145, 99},
{83, 163, 166},
{203, 178, 98},
{128, 189, 195},
{110, 100, 137},
{232, 207, 111},
{252, 249, 246},
{172, 145, 97},
{217, 155, 165},
```

```
  {113, 1, 50},
  {165, 144, 96},
  {172, 131, 61},
  {11, 8, 5},
  {161, 159, 152},
  {147, 109, 82},
  {75, 121, 24},
  {225, 173, 104},
  {175, 168, 163},
  {143, 204, 94},
  {153, 103, 12},
  {148, 121, 70},
  {110, 30, 44},
  {225, 174, 151},
  {176, 140, 94},
  {60, 106, 15},
  {205, 169, 108},
  {195, 219, 181},
  {9, 4, 6},
  {29, 156, 166},
  {164, 172, 172},
  {207, 98, 105},
  {153, 97, 38},
  {236, 232, 227},
  {66, 113, 55}
};

int color_find_index(uint8_t r, uint8_t g, uint8_t b) {
    for (int i = 0; i < 256; i++) {
        if (color_palette[i][0] == r &&
            color_palette[i][1] == g &&
            color_palette[i][2] == b) {
            return i;
        }
    }
    return 255;
}
```

color.h

```
#include "color.h"

const uint8_t color_palette[256][3] = {
  {0, 0, 0},
```

```
{255, 255, 255},
{255, 255, 0},
{158, 79, 113},
{214, 111, 52},
{247, 85, 170},
{229, 121, 13},
{93, 105, 90},
{252, 252, 249},
{176, 163, 146},
{143, 38, 85},
{204, 155, 160},
{197, 231, 115},
{159, 157, 156},
{237, 236, 235},
{161, 85, 36},
{39, 209, 205},
{101, 119, 81},
{83, 160, 88},
{182, 179, 169},
{224, 199, 107},
{193, 188, 187},
{251, 109, 205},
{156, 107, 147},
{156, 169, 153},
{171, 207, 153},
{34, 184, 194},
{165, 162, 160},
{166, 146, 96},
{120, 124, 129},
{118, 197, 197},
{224, 158, 108},
{248, 197, 140},
{132, 174, 118},
{181, 121, 84},
{102, 98, 84},
{223, 182, 139},
{20, 137, 104},
{85, 197, 97},
{221, 216, 204},
{82, 183, 38},
{37, 111, 118},
{211, 151, 114},
{191, 147, 6},
```

```
{112, 89, 38},
{101, 99, 98},
{108, 45, 21},
{132, 121, 120},
{179, 136, 112},
{145, 114, 47},
{34, 96, 19},
{114, 114, 113},
{232, 144, 42},
{119, 141, 95},
{154, 105, 14},
{213, 196, 187},
{58, 159, 84},
{122, 76, 29},
{23, 162, 148},
{227, 226, 216},
{138, 105, 54},
{228, 143, 7},
{96, 43, 9},
{108, 161, 88},
{111, 95, 78},
{159, 189, 194},
{153, 106, 92},
{248, 62, 200},
{145, 145, 146},
{196, 181, 160},
{216, 173, 38},
{240, 166, 112},
{117, 77, 39},
{246, 202, 112},
{236, 235, 234},
{248, 160, 18},
{30, 122, 131},
{58, 68, 75},
{181, 103, 58},
{237, 239, 237},
{190, 192, 192},
{123, 114, 106},
{139, 117, 80},
{70, 58, 53},
{109, 45, 21},
{100, 197, 47},
{69, 64, 60},
```

```
{125, 129, 133},
{207, 149, 15},
{239, 171, 96},
{101, 96, 88},
{92, 159, 160},
{230, 163, 16},
{185, 155, 137},
{253, 253, 253},
{130, 188, 46},
{241, 215, 171},
{103, 164, 31},
{71, 62, 64},
{158, 211, 90},
{253, 209, 48},
{223, 214, 207},
{247, 51, 142},
{146, 210, 39},
{228, 207, 157},
{146, 57, 1},
{171, 135, 112},
{182, 144, 122},
{153, 210, 163},
{235, 218, 162},
{89, 90, 92},
{234, 224, 162},
{238, 232, 232},
{216, 100, 146},
{212, 153, 180},
{178, 153, 146},
{237, 232, 236},
{24, 162, 147},
{29, 195, 188},
{145, 40, 106},
{174, 111, 138},
{161, 154, 155},
{46, 113, 84},
{172, 138, 44},
{145, 97, 42},
{206, 121, 71},
{222, 207, 119},
{136, 129, 125},
{229, 212, 155},
{15, 12, 12},
```

```
{77, 124, 128},
{203, 174, 101},
{214, 204, 157},
{231, 164, 16},
{200, 183, 149},
{109, 106, 92},
{178, 202, 135},
{254, 254, 254},
{229, 126, 69},
{54, 146, 54},
{216, 212, 212},
{51, 60, 67},
{104, 103, 104},
{122, 93, 74},
{167, 208, 207},
{149, 114, 81},
{157, 106, 13},
{104, 107, 104},
{121, 204, 157},
{245, 242, 233},
{183, 139, 116},
{251, 251, 249},
{253, 204, 22},
{250, 199, 109},
{201, 165, 54},
{234, 201, 167},
{35, 99, 29},
{214, 177, 151},
{168, 113, 80},
{178, 135, 47},
{255, 255, 236},
{193, 192, 190},
{227, 231, 216},
{129, 3, 58},
{164, 115, 84},
{232, 234, 232},
{6, 8, 5},
{144, 179, 95},
{175, 150, 139},
{244, 124, 3},
{112, 105, 113},
{2, 7, 2},
{166, 162, 155},
```

```
{116, 197, 76},
{243, 239, 229},
{122, 197, 32},
{149, 96, 43},
{164, 169, 158},
{53, 20, 10},
{206, 161, 104},
{233, 221, 211},
{158, 24, 82},
{234, 233, 228},
{107, 103, 89},
{102, 30, 94},
{240, 235, 224},
{21, 153, 74},
{80, 43, 21},
{155, 161, 160},
{212, 174, 156},
{142, 138, 123},
{239, 162, 216},
{72, 113, 117},
{213, 235, 166},
{172, 169, 159},
{104, 81, 46},
{174, 110, 138},
{192, 191, 192},
{165, 154, 159},
{105, 48, 72},
{160, 225, 221},
{208, 141, 105},
{97, 208, 211},
{135, 126, 129},
{222, 242, 240},
{172, 123, 103},
{75, 183, 198},
{239, 186, 145},
{204, 29, 96},
{20, 153, 76},
{173, 145, 99},
{83, 163, 166},
{203, 178, 98},
{128, 189, 195},
{110, 100, 137},
{232, 207, 111},
```

```
  {252, 249, 246},
  {172, 145, 97},
  {217, 155, 165},
  {113, 1, 50},
  {165, 144, 96},
  {172, 131, 61},
  {11, 8, 5},
  {161, 159, 152},
  {147, 109, 82},
  {75, 121, 24},
  {225, 173, 104},
  {175, 168, 163},
  {143, 204, 94},
  {153, 103, 12},
  {148, 121, 70},
  {110, 30, 44},
  {225, 174, 151},
  {176, 140, 94},
  {60, 106, 15},
  {205, 169, 108},
  {195, 219, 181},
  {9, 4, 6},
  {29, 156, 166},
  {164, 172, 172},
  {207, 98, 105},
  {153, 97, 38},
  {236, 232, 227},
  {66, 113, 55}
};

int color_find_index(uint8_t r, uint8_t g, uint8_t b) {
    for (int i = 0; i < 256; i++) {
        if (color_palette[i][0] == r &&
            color_palette[i][1] == g &&
            color_palette[i][2] == b) {
            return i;
        }
    }
    return 255;
}
```

fbputchar.c

```
/* fbputchar.c: Framebuffer character generator

screen /dev/ttyUSB0 115200
rm -r ./project/
scp -r
jz3850@micro18.ee.columbia.edu:/user/stud/fall24/jz3850/4840/project ./
insmod vga_framebuffer.ko
rmmod vga_framebuffer

 */

#include "fbputchar.h"
#include "framebuffer.h"
#include <stdint.h>
#include <stdio.h>
#include <unistd.h>

#define FONT_WIDTH 8
#define FONT_HEIGHT 16
#define BITS_PER_PIXEL 32
#define LINE_LENGTH WINDOW_WIDTH * 4

/* 8 X 16 console font from /lib/kbd/consolefonts/lat0-16.psfu.gz
od --address-radix=n --width=16 -v -t x1 -j 4 -N 2048 lat0-16.psfu
*/

static unsigned char font[2048] = {
    0x00, 0x00, 0x7e, 0xc3, 0x99, 0x99, 0xf3, 0xe7, 0xe7, 0xff, 0xe7,
0xe7,
    0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x76, 0xdc,
0x00,
    0x76, 0xdc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x6e,
0xf8,
    0xd8, 0xd8, 0xdc, 0xd8, 0xd8, 0xd8, 0xf8, 0x6e, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x6e, 0xdb, 0xdb, 0xdf, 0xd8, 0xdb,
0x6e,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x38, 0x7c,
0xfe,
    0x7c, 0x38, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x88, 0x88,
0xf8,
```

```
    0x88, 0x88, 0x00, 0x3e, 0x08, 0x08, 0x08, 0x08, 0x00, 0x00, 0x00,
0x00,
    0x00, 0xf8, 0x80, 0xe0, 0x80, 0x80, 0x00, 0x3e, 0x20, 0x38, 0x20,
0x20,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x70, 0x88, 0x80, 0x88, 0x70, 0x00,
0x3c,
    0x22, 0x3c, 0x24, 0x22, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80,
0x80,
    0x80, 0xf8, 0x00, 0x3e, 0x20, 0x38, 0x20, 0x20, 0x00, 0x00, 0x00,
0x00,
    0x11, 0x44, 0x11, 0x44, 0x11, 0x44, 0x11, 0x44, 0x11, 0x44, 0x11,
0x44,
    0x11, 0x44, 0x11, 0x44, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55,
0xaa,
    0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0xdd, 0x77, 0xdd,
0x77,
    0xdd, 0x77, 0xdd, 0x77, 0xdd, 0x77, 0xdd, 0x77, 0xdd, 0x77, 0xdd,
0x77,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff,
    0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff,
    0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0xf0, 0xf0, 0xf0, 0xf0, 0xf0, 0xf0, 0xf0, 0xf0, 0xf0, 0xf0, 0xf0,
0xf0,
    0xf0, 0xf0, 0xf0, 0xf0, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f,
0x0f,
    0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x00, 0x88, 0xc8,
0xa8,
    0x98, 0x88, 0x00, 0x20, 0x20, 0x20, 0x20, 0x3e, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x88, 0x88, 0x50, 0x50, 0x20, 0x00, 0x3e, 0x08, 0x08, 0x08,
0x08,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0e, 0x38, 0xe0,
0x38,
    0x0e, 0x00, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0xe0, 0x38, 0x0e, 0x38, 0xe0, 0x00, 0xfe, 0x00, 0x00, 0x00, 0x00,
0x00,
```

```
    0x00, 0x00, 0x00, 0x06, 0x0c, 0xfe, 0x18, 0x30, 0xfe, 0x60, 0xc0,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x1e, 0x7e,
0xfe,
    0x7e, 0x1e, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0xc0, 0xf0, 0xfc, 0xfe, 0xfc, 0xf0, 0xc0, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x18, 0x3c, 0x7e, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18,
    0x18, 0x7e, 0x3c, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x18, 0x0c, 0xfe, 0x0c, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x60, 0xfe, 0x60, 0x30, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x3c, 0x7e, 0x18, 0x18,
0x18,
    0x18, 0x7e, 0x3c, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x28, 0x6c, 0xfe, 0x6c, 0x28, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x06, 0x36, 0x66, 0xfe, 0x60, 0x30, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xfe,
0x6e,
    0x6c, 0x6c, 0x6c, 0x6c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x18, 0x3c, 0x3c, 0x3c, 0x18, 0x18, 0x18, 0x00, 0x18,
0x18,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x66, 0x66, 0x66, 0x24, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x6c,
    0x6c, 0xfe, 0x6c, 0x6c, 0x6c, 0xfe, 0x6c, 0x6c, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x10, 0x10, 0x7c, 0xd6, 0xd0, 0xd0, 0x7c, 0x16, 0x16, 0xd6,
0x7c,
```

```
    0x10, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc2, 0xc6, 0x0c,
0x18,
    0x30, 0x60, 0xc6, 0x86, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x38,
0x6c,
    0x6c, 0x38, 0x76, 0xdc, 0xcc, 0xcc, 0xcc, 0x76, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x18, 0x18, 0x18, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0c, 0x18, 0x30, 0x30, 0x30,
0x30,
    0x30, 0x30, 0x18, 0x0c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30,
0x18,
    0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x18, 0x30, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x66, 0x3c, 0xff, 0x3c, 0x66, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18,
0x7e,
    0x18, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x18, 0x30, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfe, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x06, 0x0c, 0x18, 0x30, 0x60, 0xc0, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x7c, 0xc6, 0xce, 0xce, 0xd6, 0xd6, 0xe6, 0xe6, 0xc6,
0x7c,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x38, 0x78, 0x18, 0x18,
0x18,
    0x18, 0x18, 0x18, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7c,
0xc6,
    0x06, 0x0c, 0x18, 0x30, 0x60, 0xc0, 0xc6, 0xfe, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x7c, 0xc6, 0x06, 0x06, 0x3c, 0x06, 0x06, 0x06, 0xc6,
0x7c,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0c, 0x1c, 0x3c, 0x6c, 0xcc,
0xfe,
```

```
    0x0c, 0x0c, 0x0c, 0x1e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfe,
0xc0,
    0xc0, 0xc0, 0xfc, 0x06, 0x06, 0x06, 0xc6, 0x7c, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x38, 0x60, 0xc0, 0xc0, 0xfc, 0xc6, 0xc6, 0xc6, 0xc6,
0x7c,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfe, 0xc6, 0x06, 0x06, 0x0c,
0x18,
    0x30, 0x30, 0x30, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7c,
0xc6,
    0xc6, 0xc6, 0x7c, 0xc6, 0xc6, 0xc6, 0xc6, 0x7c, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0x7e, 0x06, 0x06, 0x06, 0x0c,
0x78,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x00,
0x00,
    0x00, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x18, 0x18, 0x00, 0x00, 0x00, 0x18, 0x18, 0x30, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x06, 0x0c, 0x18, 0x30, 0x60, 0x30, 0x18, 0x0c,
0x06,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfe, 0x00,
0x00,
    0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x60,
    0x30, 0x18, 0x0c, 0x06, 0x0c, 0x18, 0x30, 0x60, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x7c, 0xc6, 0xc6, 0x0c, 0x18, 0x18, 0x18, 0x00, 0x18,
0x18,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0xde,
0xde,
    0xde, 0xdc, 0xc0, 0x7c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
0x38,
    0x6c, 0xc6, 0xc6, 0xfe, 0xc6, 0xc6, 0xc6, 0xc6, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0xfc, 0x66, 0x66, 0x66, 0x7c, 0x66, 0x66, 0x66, 0x66,
0xfc,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3c, 0x66, 0xc2, 0xc0, 0xc0,
0xc0,
    0xc0, 0xc2, 0x66, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf8,
0x6c,
```

```
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x6c, 0xf8, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0xfe, 0x66, 0x62, 0x68, 0x78, 0x68, 0x60, 0x62, 0x66,
0xfe,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfe, 0x66, 0x62, 0x68, 0x78,
0x68,
    0x60, 0x60, 0x60, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3c,
0x66,
    0xc2, 0xc0, 0xc0, 0xde, 0xc6, 0xc6, 0x66, 0x3a, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0xc6, 0xc6, 0xc6, 0xc6, 0xfe, 0xc6, 0xc6, 0xc6, 0xc6,
0xc6,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3c, 0x18, 0x18, 0x18, 0x18,
0x18,
    0x18, 0x18, 0x18, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1e,
0x0c,
    0x0c, 0x0c, 0x0c, 0x0c, 0xcc, 0xcc, 0xcc, 0x78, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0xe6, 0x66, 0x66, 0x6c, 0x78, 0x78, 0x6c, 0x66, 0x66,
0xe6,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf0, 0x60, 0x60, 0x60, 0x60,
0x60,
    0x60, 0x62, 0x66, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc6,
0xee,
    0xfe, 0xfe, 0xd6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0xc6, 0xe6, 0xf6, 0xfe, 0xde, 0xce, 0xc6, 0xc6, 0xc6,
0xc6,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0xc6,
0xc6,
    0xc6, 0xc6, 0xc6, 0x7c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc,
0x66,
    0x66, 0x66, 0x7c, 0x60, 0x60, 0x60, 0x60, 0xf0, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xd6, 0xde,
0x7c,
    0x0c, 0x0e, 0x00, 0x00, 0x00, 0x00, 0xfc, 0x66, 0x66, 0x66, 0x7c,
0x6c,
    0x66, 0x66, 0x66, 0xe6, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7c,
0xc6,
    0xc6, 0x60, 0x38, 0x0c, 0x06, 0xc6, 0xc6, 0x7c, 0x00, 0x00, 0x00,
0x00,
```

```
    0x00, 0x00, 0x7e, 0x7e, 0x5a, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x3c,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6,
0xc6,
    0xc6, 0xc6, 0xc6, 0x7c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc6,
0xc6,
    0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0x6c, 0x38, 0x10, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0xc6, 0xc6, 0xc6, 0xc6, 0xd6, 0xd6, 0xd6, 0xfe, 0xee,
0x6c,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc6, 0xc6, 0x6c, 0x7c, 0x38,
0x38,
    0x7c, 0x6c, 0xc6, 0xc6, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x66,
0x66,
    0x66, 0x66, 0x3c, 0x18, 0x18, 0x18, 0x18, 0x3c, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0xfe, 0xc6, 0x86, 0x0c, 0x18, 0x30, 0x60, 0xc2, 0xc6,
0xfe,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3c, 0x30, 0x30, 0x30, 0x30,
0x30,
    0x30, 0x30, 0x30, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0xc0, 0x60, 0x30, 0x18, 0x0c, 0x06, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x3c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c,
0x3c,
    0x00, 0x00, 0x00, 0x00, 0x10, 0x38, 0x6c, 0xc6, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff,
0x00,
    0x00, 0x30, 0x30, 0x30, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x78, 0x0c,
0x7c,
    0xcc, 0xcc, 0xcc, 0x76, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xe0,
0x60,
    0x60, 0x78, 0x6c, 0x66, 0x66, 0x66, 0x66, 0x7c, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc0, 0xc0, 0xc0, 0xc6,
0x7c,
```

```
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1c, 0x0c, 0x0c, 0x3c, 0x6c,
0xcc,
    0xcc, 0xcc, 0xcc, 0x76, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x7c, 0xc6, 0xfe, 0xc0, 0xc0, 0xc6, 0x7c, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x38, 0x6c, 0x64, 0x60, 0xf0, 0x60, 0x60, 0x60, 0x60,
0xf0,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x76, 0xcc,
0xcc,
    0xcc, 0xcc, 0xcc, 0x7c, 0x0c, 0xcc, 0x78, 0x00, 0x00, 0x00, 0xe0,
0x60,
    0x60, 0x6c, 0x76, 0x66, 0x66, 0x66, 0x66, 0xe6, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x18, 0x18, 0x00, 0x38, 0x18, 0x18, 0x18, 0x18, 0x18,
0x3c,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x06, 0x00, 0x0e, 0x06,
0x06,
    0x06, 0x06, 0x06, 0x06, 0x66, 0x66, 0x3c, 0x00, 0x00, 0x00, 0xe0,
0x60,
    0x60, 0x66, 0x6c, 0x78, 0x78, 0x6c, 0x66, 0xe6, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x70, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x34,
0x18,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xec, 0xfe,
0xd6,
    0xd6, 0xd6, 0xd6, 0xc6, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0xdc, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6,
0x7c,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xdc, 0x66,
0x66,
    0x66, 0x66, 0x66, 0x7c, 0x60, 0x60, 0xf0, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x76, 0xcc, 0xcc, 0xcc, 0xcc, 0xcc, 0x7c, 0x0c, 0x0c, 0x1e,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0xdc, 0x76, 0x66, 0x60, 0x60, 0x60,
0xf0,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7c, 0xc6,
0x60,
```

```
    0x38, 0x0c, 0xc6, 0x7c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
0x30,
    0x30, 0xfc, 0x30, 0x30, 0x30, 0x30, 0x36, 0x1c, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0xcc, 0xcc, 0xcc, 0xcc, 0xcc, 0xcc,
0x76,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x66, 0x66,
0x66,
    0x66, 0x66, 0x3c, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0xc6, 0xc6, 0xd6, 0xd6, 0xd6, 0xfe, 0x6c, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0xc6, 0x6c, 0x38, 0x38, 0x38, 0x6c,
0xc6,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc6, 0xc6,
0xc6,
    0xc6, 0xc6, 0xc6, 0x7e, 0x06, 0x0c, 0xf8, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0xfe, 0xcc, 0x18, 0x30, 0x60, 0xc6, 0xfe, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x0e, 0x18, 0x18, 0x18, 0x70, 0x18, 0x18, 0x18, 0x18,
0x0e,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18,
    0x18, 0x18, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x70,
0x18,
    0x18, 0x18, 0x0e, 0x18, 0x18, 0x18, 0x18, 0x70, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x76, 0xdc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x66, 0x00, 0x66, 0x66, 0x66, 0x66,
0x3c,
    0x18, 0x18, 0x18, 0x3c, 0x00, 0x00, 0x00, 0x00,
};

/*
 * Draw the given character at the given row/column.
 * fbopen() must be called first.
 */
void fbputchar(int x0, int y0, char c, int color_index) {
  const uint8_t *glyph = font + c * FONT_HEIGHT;
  for (int row = 0; row < FONT_HEIGHT; row++) {
      uint8_t bits = glyph[row];
```

```
        for (int col = 0; col < FONT_WIDTH; col++) {
            if (bits & (1 << (7 - col))) {
                framebuffer_draw_pixel(x0 + col, y0 + row, color_index);
            }
        }
    }
}

/*
 * Draw the given string at the given row/column.
 * String must fit on a single line: wrap-around is not handled.
 */
void fbputs(int x0, int y0, const char *s, int color_index) {
    int cursor_x = x0;
    int cursor_y = y0;
    while (*s) {
        fbputchar(cursor_x, cursor_y, *s, color_index);
        cursor_x += FONT_WIDTH;
        s++;
    }
}
```

fbputchar.h

```
#ifndef _FBPUTCHAR_H
#define _FBPUTCHAR_H

void fbputchar(int x, int y, char c, int color_index);
void fbputs(int x, int y, const char *s, int color_index);

#endif
```

framebuffer.c

```
#include "framebuffer.h"
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include "vga_framebuffer.h"

static int fb_fd = -1;
```

```c
static uint8_t *next_buffer = NULL;
static uint8_t *now_buffer = NULL;

int framebuffer_init(const char *device) {
    fb_fd = open(device, O_RDWR);
    if (fb_fd < 0) {
        perror("open framebuffer");
        return -1;
    }

    next_buffer = malloc(SCREEN_SIZE);
    now_buffer = malloc(SCREEN_SIZE);
    if (!next_buffer || !now_buffer) {
        perror("malloc buffer");
        framebuffer_exit();
        return -1;
    }

    memset(next_buffer, 255, SCREEN_SIZE);
    memset(now_buffer, 255, SCREEN_SIZE);

    return 0;
}

void framebuffer_clear(uint8_t color_index) {
    if (next_buffer) {
        memset(next_buffer, color_index, SCREEN_SIZE);
    }
}

void framebuffer_draw_sprite(const uint8_t *sprite_pixels, int
sprite_width, int sprite_height, int dst_x, int dst_y) {
    if (!next_buffer) return;

    for (int y = 0; y < sprite_height; y++) {
        for (int x = 0; x < sprite_width; x++) {
            uint8_t color = sprite_pixels[y * sprite_width + x];
            if (color == 255) continue;

            int draw_x = dst_x + x;
            int draw_y = dst_y + y;
```

```
            if (draw_x < 0 || draw_x >= SCREEN_WIDTH || draw_y < 0 ||
draw_y >= SCREEN_HEIGHT)
                continue;

            next_buffer[draw_y * SCREEN_WIDTH + draw_x] = color;
        }
    }
}

void framebuffer_draw_pixel(int x, int y, uint8_t color) {
    if (!next_buffer) return;
    if (x < 0 || x >= SCREEN_WIDTH || y < 0 || y >= SCREEN_HEIGHT)
        return;
    next_buffer[y * SCREEN_WIDTH + x] = color;
}

void framebuffer_swap() {
    if (now_buffer && next_buffer) {
        for (int addr = 0; addr < SCREEN_WIDTH * SCREEN_HEIGHT; addr++)
{
            if (now_buffer[addr] != next_buffer[addr]) {
                vga_framebuffer_pixel_t pixel;
                pixel.addr = addr;
                pixel.color = next_buffer[addr];
                ioctl(fb_fd, VGA_FRAMEBUFFER_WRITE_PIXEL, &pixel);

                now_buffer[addr] = next_buffer[addr]; // now_buffer
            }
        }
    }
}

void framebuffer_exit() {
    if (next_buffer) free(next_buffer);
    if (now_buffer) free(now_buffer);
    if (fb_fd >= 0) close(fb_fd);

    next_buffer = NULL;
    now_buffer = NULL;
    fb_fd = -1;
}
```

framebuffer.h

```c
#ifndef _FRAMEBUFFER_H
#define _FRAMEBUFFER_H

#include <stdint.h>

#define SCREEN_WIDTH 640
#define SCREEN_HEIGHT 480
#define SCREEN_SIZE (SCREEN_WIDTH * SCREEN_HEIGHT)

int framebuffer_init(const char *device);
void framebuffer_clear(uint8_t color_index);
void framebuffer_draw_sprite(const uint8_t *sprite_pixels, int
sprite_width, int sprite_height, int dst_x, int dst_y);
void framebuffer_draw_pixel(int x, int y, uint8_t color);
void framebuffer_swap();  // swap next_buffer to now_buffer
void framebuffer_exit();

#endif
```

game.c

```c
#include "fbputchar.h"
#include "usbkeyboard.h"
#include "Note.h"
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include "sprite.h"
#include "framebuffer.h"
#include "audio_user.h"
#include <time.h>

// framebuffer_init("/dev/vga_framebuffer"); // initialize framebuffer
// sprite_init(); // initialize and load sprites

// sprite_exit(); // release sprites
// framebuffer_exit(); // release framebuffer

// framebuffer_clear(0);
// sprite_draw(sprite_0, 100, 100, 0, 0);
// fbputs(10, 20, "Hello, FPGA!", 1);
// fbputchar(10, 10, 'A', 1);
```

```
// framebuffer_swap();

// Keyboard
struct libusb_device_handle *keyboard;
uint8_t endpoint_address;
// Menu Items
#define MENU_ITEMS 4

//-----------------------------------------------------Menu---------------
------------------------------------------------------------
const char *menu_options[MENU_ITEMS] = {
    "Play",
    "Tutorial",
    "Difficulty",
    "Exit"};
void display_menu(int selected_index)
{
    framebuffer_clear(0);
    sprite_draw(sprite_logo, 80, 0, 0, 0);
    fbputs(200, 300, "Welcome to Rhythm Master!", 1);
    fbputs(200, 320, "Please select an option:", 1);

    // Display menu options
    for (int i = 0; i < MENU_ITEMS; i++)
    {
        if (i == selected_index)
        {
            fbputs(190, 360 + i * 20, "> ", 1);
            fbputs(200, 360 + i * 20, menu_options[i], 1);
        }
        else
        {
            fbputs(190, 360 + i * 20, "  ", 1);
            fbputs(200, 360 + i * 20, menu_options[i], 1);
        }
    }
    framebuffer_swap();
}

void tutorial()
{
    framebuffer_clear(0);
    fbputs(50, 70, "Welcome to Rhythm Master!", 1);
```

```
    fbputs(50, 100, "This is a rhythm-based music game.", 1);
    fbputs(50, 130, "You need to press keys in synchronous with falling
notes.", 1);
    fbputs(50, 160, "There are 6 paths, and you can eliminate the
falling notes by", 1);
    fbputs(50, 190, "pressing a s d j k l when they enter determination
area.", 1);
    fbputs(50, 220, "Early or late press results in loss of points.",
1);
    fbputs(50, 250, "Miss press earns no point.", 1);
    fbputs(50, 280, "Combo earns extra points.", 1);
    fbputs(50, 310, "While playing, you can pause the game by pressing
enter.", 1);
    fbputs(50, 340, "When game completed, you'll ge given a score and a
level.", 1);
    fbputs(50, 370, "Press Enter to return to the main menu...", 1);
    framebuffer_swap();
    struct usb_keyboard_packet packet;
    int transferred;
    while (1)
    {
        libusb_interrupt_transfer(keyboard, endpoint_address, (unsigned
char *)&packet, sizeof(packet), &transferred, 0);
        if (transferred == sizeof(packet))
        {
            if (packet.keycode[0] == 0x28)
            {           // Enter key
                break; // Exit the loop and return to the main menu
            }
        }
    }
}

#define DIFFICULTY_ITEMS 3
const char *difficulty_options[DIFFICULTY_ITEMS] = {
    "Easy",
    "Medium",
    "Hard"};
// int note_interval = 5;
int current_difficulty = 0;
void difficulty()
{
    int selected = current_difficulty;
```

```
    struct usb_keyboard_packet packet;
    int transferred;
    while (1)
    {
        framebuffer_clear(0);
        for (int i = 0; i < DIFFICULTY_ITEMS; i++)
        {
            if (i == selected)
            {
                fbputs(230, 200 + i * 30, "> ", 1);
            }
            else
            {
                fbputs(230, 200 + i * 30, "  ", 1);
            }
            fbputs(240, 200 + i * 30, difficulty_options[i], 1);
        }
        framebuffer_swap();
        libusb_interrupt_transfer(keyboard, endpoint_address, (unsigned
char *)&packet, sizeof(packet), &transferred, 0);
        if (transferred == sizeof(packet))
        {
            if (packet.keycode[0] == 0x52)
            { // Up
                selected = (selected - 1 + DIFFICULTY_ITEMS) %
DIFFICULTY_ITEMS;
            }
            else if (packet.keycode[0] == 0x51)
            { // Down
                selected = (selected + 1) % DIFFICULTY_ITEMS;
            }
            else if (packet.keycode[0] == 0x28)
            { // Enter
                current_difficulty = selected;
                break;
            }
        }
        usleep(100000);
    }
}

#define MUSIC_ITEMS 2
const char *music_options[MUSIC_ITEMS] = {
```

```
    "Littlestar",
    "CanonRock"};
int music_selected = 0;
void choose_music()
{
    int selected = music_selected;
    struct usb_keyboard_packet packet;
    int transferred;
    while (1)
    {
        framebuffer_clear(0);
        for (int i = 0; i < MUSIC_ITEMS; i++)
        {
            if (i == selected)
            {
                fbputs(230, 200 + i * 30, "> ", 1);
            }
            else
            {
                fbputs(230, 200 + i * 30, "  ", 1);
            }
            fbputs(240, 200 + i * 30, music_options[i], 1);
        }
        framebuffer_swap();
        libusb_interrupt_transfer(keyboard, endpoint_address, (unsigned
char *)&packet, sizeof(packet), &transferred, 0);
        if (transferred == sizeof(packet))
        {
            if (packet.keycode[0] == 0x52)
            { // Up
                selected = (selected - 1 + MUSIC_ITEMS) % MUSIC_ITEMS;
            }
            else if (packet.keycode[0] == 0x51)
            { // Down
                selected = (selected + 1) % MUSIC_ITEMS;
            }
            else if (packet.keycode[0] == 0x28)
            { // Enter
                music_selected = selected;
                break;
            }
        }
        usleep(100000);
```

```
    }
}
//-----------------------------------------------------Play---------------
-------------------------------------------------------------

int note_sequence[] = {1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 6, 5, 4, 3,
2, 1, 1, 1, 1, 1};

#define MAX_ACTIVE_NOTES 1000
#define PAUSE_MENU_ITEMS 3
const char *pause_menu_options[PAUSE_MENU_ITEMS] = {
    "Continue",
    "Restart",
    "Back to Main Menu"};

typedef struct
{
    int track;
    int frame;
    int active;
    // long created_sys_time;
    // int created_audio_time;
} Note;

typedef struct
{
    int show;
    int timer;
} TrackFeedback;

void draw_pause_menu(int selected_index, int combo, int score)
{

    fbputs(280, 170, "PAUSED", 1);

    char combo_str[32], score_str[32];
    sprintf(combo_str, "Combo: %d", combo);
    sprintf(score_str, "Score: %d", score);
    fbputs(180, 200, combo_str, 1);
    fbputs(180, 220, score_str, 1);

    for (int i = 0; i < PAUSE_MENU_ITEMS; i++)
    {
```

```
        if (i == selected_index)
        {
            fbputs(190, 250 + i * 30, "> ", 1);
            fbputs(200, 250 + i * 30, pause_menu_options[i], 1);
        }
        else
        {
            fbputs(190, 250 + i * 30, "  ", 1);
            fbputs(200, 250 + i * 30, pause_menu_options[i], 1);
        }
    }
}

int show_pause_menu(int *combo, int *score)
{
    int selected_index = 0;
    struct usb_keyboard_packet packet;
    int transferred;

    while (1)
    {
        framebuffer_clear(0);
        draw_pause_menu(selected_index, *combo, *score);
        framebuffer_swap();

        libusb_interrupt_transfer(keyboard, endpoint_address,
                                  (unsigned char *)&packet,
sizeof(packet),
                                  &transferred, 10);

        if (transferred == sizeof(packet))
        {
            if (packet.keycode[0] == 0x28)
            {                           // Enter
                return selected_index;
            }
            else if (packet.keycode[0] == 0x52)
            { // Up
                selected_index = (selected_index - 1 +
PAUSE_MENU_ITEMS) % PAUSE_MENU_ITEMS;
            }
            else if (packet.keycode[0] == 0x51)
            { // Down
```

```
                selected_index = (selected_index + 1) %
PAUSE_MENU_ITEMS;
            }
        }

        usleep(10000);
    }
}
void countdown_before_continue(int combo, int score, Note
*active_notes, int active_count,
                               NoteAnimation **track_animations)
{
    for (int i = 3; i > 0; i--)
    {
        char count_str[32];
        sprintf(count_str, "Resuming in %d...", i);

        framebuffer_clear(0);
        draw_bg(combo, score);

        // redraw all notes before pause
        for (int j = 0; j < active_count; j++)
        {
            if (active_notes[j].active)
            {
                Note *note = &active_notes[j];
                NoteAnimation *anim = track_animations[note->track];
                int *frame = anim->frames[note->frame];
                sprite_t *sprites[6] = {sprite_note1, sprite_note2,
sprite_note3, sprite_note4, sprite_note5, sprite_note6};
                sprite_draw(sprites[note->track], frame[0], frame[1],
frame[2], frame[3]);
            }
        }

        // countdown
        fbputs(240, 200, count_str, 1);
        framebuffer_swap();
        sleep(1);
    }
}

int play()
```

```
{
    framebuffer_clear(0);
    const char *loading_str = "Loading...";
    fbputs(240, 200, loading_str, 1);
    framebuffer_swap();
    Note active_notes[MAX_ACTIVE_NOTES] = {{0}};
    int active_count = 0;
    // int current_note_index = 0;
    // int note_generation_timer = 0;
    // int note_generation_interval = 100;
    int game_over = 0;
    TrackFeedback track_feedbacks[6] = {{0}};
    int combo = 0;
    int bonus = 0;
    int great_count = 0;
    int perfect_count = 0;
    int score = 0;

    NoteAnimation *track_animations[] = {&POS1, &POS2, &POS3, &POS4,
&POS5, &POS6};

    struct usb_keyboard_packet packet;
    int transferred;

    struct timespec last_tick, now;
    clock_gettime(CLOCK_MONOTONIC, &last_tick);
    int64_t pause_total_ms = 0;

    char audio_file[32];
    char note_file[32];

    snprintf(audio_file, sizeof(audio_file), "./music/%s.raw",
music_options[music_selected]);

    if (load_audio(audio_file) < 0)
    {
        fprintf(stderr, "Failed to loading audio.\n");
        return 0;
    }

    if (current_difficulty == 0)
    {
```

```
        snprintf(note_file, sizeof(note_file), "./music/%s_easy.map",
music_options[music_selected]);
    }
    else
    {
        snprintf(note_file, sizeof(note_file), "./music/%s.map",
music_options[music_selected]);
    }
    if (load_note_file(note_file) < 0)
    {
        fprintf(stderr, "Failed to loading notes.\n");
        return 0;
    }

    int current_event_index = 0;
    int music_started = 0;
    double difficulty_drop_time_ms = (current_difficulty == 2) ? 5000.0
: 8000.0;
    double frame_time_ms = difficulty_drop_time_ms / NOTE_ANIMATION;
    const int drop_time_ms = (int)(frame_time_ms * NOTE_ANIMATION +
0.5);
    struct timespec sys_start;
    clock_gettime(CLOCK_MONOTONIC, &sys_start);
    int64_t time_offset = 0;
    reset_audio();
    // play_audio(0);

    while (!game_over)
    {
        // int ms = audio_time();
        // printf("ms: %d\n", ms);
        // check pause
        libusb_interrupt_transfer(keyboard, endpoint_address,
                                  (unsigned char *)&packet,
sizeof(packet),
                                  &transferred, 3);

        if (transferred == sizeof(packet) && packet.keycode[0] == 0x28)
        { // press Enter to pause
            int ms = 0;
            struct timespec pause_start;
            if (music_started)
            {
```

```
                ms = pulse_audio();
            }
            else
            {
                clock_gettime(CLOCK_MONOTONIC, &pause_start);
            }
            int pause_result = show_pause_menu(&combo, &score);

            switch (pause_result)
            {
            case 0: // Continue
                countdown_before_continue(combo, score, active_notes,
active_count,
                                          track_animations);
                if (music_started)
                {
                    play_audio(ms); // recover audio
                }
                else
                {
                    struct timespec pause_end;
                    clock_gettime(CLOCK_MONOTONIC, &pause_end);

                    int64_t pause_delta_ms =
                        (pause_end.tv_sec - pause_start.tv_sec) * 1000
+
                        (pause_end.tv_nsec - pause_start.tv_nsec) /
1000000;

                    pause_total_ms += pause_delta_ms;
                }
                break;
            case 1: // Restart
                memset(active_notes, 0, sizeof(active_notes));
                active_count = 0;
                // current_note_index = 0;
                // note_generation_timer = 0;
                memset(track_feedbacks, 0, sizeof(track_feedbacks));
                combo = 0;
                great_count = 0;
                perfect_count = 0;
                score = 0;
                music_started = 0; // restart audio playing
```

```
            reset_audio();
            time_offset = 0;
            clock_gettime(CLOCK_MONOTONIC, &sys_start);
            current_event_index = 0;
            break;
        case 2:           // Back to Main Menu
            free_audio(); // stop audio
            unload_note_file();
            return 0;
        }
    }

    // game logic
    if (transferred == sizeof(packet))
    {
        for (int i = 0; i < 6; i++)
        { // check each key
            uint8_t key = packet.keycode[i];
            if (key == 0)
                continue;
            int track = -1;
            switch (key)
            {
            case 0x04:
                track = 0;
                break; // a
            case 0x16:
                track = 1;
                break; // s
            case 0x07:
                track = 2;
                break; // d
            case 0x0d:
                track = 3;
                break; // j
            case 0x0e:
                track = 4;
                break; // k
            case 0x0f:
                track = 5;
                break; // l
            default:
                break;
```

```
                }
                if (track != -1)
                {
                    // find the bottom note
                    int max_frame = -1;
                    int selected_idx = -1;
                    for (int j = 0; j < active_count; j++)
                    {
                        if (active_notes[j].active &&
active_notes[j].track == track)
                        {
                            if (active_notes[j].frame > max_frame)
                            {
                                max_frame = active_notes[j].frame;
                                selected_idx = j;
                            }
                        }
                    }

                    if (selected_idx != -1)
                    {
                        Note *note = &active_notes[selected_idx];
                        NoteAnimation *anim =
track_animations[note->track];
                        int y = anim->frames[note->frame][1];
                        if (y > 200)
                        {
                            note->active = 0; // eliminate note

                            if (y > 460)
                            { // Miss (>460)
                                combo = 0;
                                track_feedbacks[track].show = -1;
                            }
                            else if (y < 340)
                            { // Great (200-340)
                                combo++;
                                if (combo > 10)
                                    bonus = 10;
                                else
                                    bonus = combo;
                                great_count++;
                                track_feedbacks[track].show = 1;
```

```
                                score = score + bonus * 5 + 30;
                            }
                            else
                            { // Perfect (340-460)
                                combo++;
                                if (combo > 10)
                                    bonus = 10;
                                else
                                    bonus = combo;
                                perfect_count++;
                                track_feedbacks[track].show = 2;
                                score = score + bonus * 5 + 50;
                            }
                            track_feedbacks[track].timer = 10;
                        }
                    }
                }
            }
        }

        clock_gettime(CLOCK_MONOTONIC, &now);
        long elapsed_ms = (now.tv_sec - last_tick.tv_sec) * 1000 +
(now.tv_nsec - last_tick.tv_nsec) / 1000000;

        if (elapsed_ms < frame_time_ms)
            continue; // wait for next frame

        last_tick = now; // update time

        // Update Notes
        for (int i = 0; i < active_count; i++)
        { // Miss
            if (active_notes[i].active)
            {
                active_notes[i].frame++;

                if (active_notes[i].frame >= NOTE_ANIMATION)
                {
                    struct timespec t;
                    clock_gettime(CLOCK_MONOTONIC, &t);

                    combo = 0;
                    track_feedbacks[active_notes[i].track].show = -1;
```

```
                track_feedbacks[active_notes[i].track].timer = 10;
                active_notes[i].active = 0;
            }
        }
    }

    // spawn new notes
    struct timespec now;
    clock_gettime(CLOCK_MONOTONIC, &now);
    int64_t sys_elapsed =
        (now.tv_sec - sys_start.tv_sec) * 1000 +
        (now.tv_nsec - sys_start.tv_nsec) / 1000000 -
        pause_total_ms;

    int64_t ref_time;
    if (!music_started)
    {
        // audio not started, use system time
        time_offset = sys_elapsed;
        if (time_offset > drop_time_ms - (current_difficulty == 2 ?
-200 : 400))
        {
            play_audio(0);
            music_started = 1;
        }
    }

    ref_time = time_offset + audio_time();

    while (current_event_index < note_event_count &&
           ref_time >= note_events[current_event_index].time_ms)
    {
        int tr = note_events[current_event_index].track;
        if (tr >= 0 && tr < 6 && active_count < MAX_ACTIVE_NOTES)
        {
            active_notes[active_count++] = (Note){
                .track = tr,
                .frame = 0,
                .active = 1,
                // .created_sys_time   = sys_elapsed,
                // .created_audio_time = (int)ref_time
            };
```

```
                // printf("[NOTE #%d] Target time: %d ms, ref_time: %d
ms\n",
                //       current_event_index,
note_events[current_event_index].time_ms, (int)ref_time);
            }
            current_event_index++;
        }

        // update feedback display timer
        for (int i = 0; i < 6; i++)
        {
            if (track_feedbacks[i].timer > 0 &&
--track_feedbacks[i].timer == 0)
            {
                track_feedbacks[i].show = 0;
            }
        }

        // begin drawing
        framebuffer_clear(0);
        draw_bg(combo, score);

        // draw notes
        for (int i = 0; i < active_count; i++)
        {
            if (active_notes[i].active)
            {
                Note *note = &active_notes[i];
                NoteAnimation *anim = track_animations[note->track];
                int *frame = anim->frames[note->frame];
                sprite_t *sprites[6] = {sprite_note1, sprite_note2,
sprite_note3, sprite_note4, sprite_note5, sprite_note6};
                sprite_draw(sprites[note->track], frame[0], frame[1],
frame[2], frame[3]);
            }
        }

        // draw feedback
        for (int i = 0; i < 6; i++)
        {
            if (track_feedbacks[i].show)
            {
```

```
                sprite_t *sprite = track_feedbacks[i].show == -1 ?
sprite_miss : track_feedbacks[i].show == 1 ? sprite_great

: sprite_perfect;
                sprite_draw(sprite, i * 100, 430, 0, 0);
            }
        }

        framebuffer_swap();

        // check if game end
        if (is_audio_finished())
        {
            int all_inactive = 1;
            for (int i = 0; i < active_count; i++)
            {
                if (active_notes[i].active)
                {
                    all_inactive = 0;
                    break;
                }
            }
            game_over = all_inactive;
        }
    }

    // game over interface
    const int max_score = note_event_count * 100 - 225;
    float ratio = (float)score / max_score;
    framebuffer_clear(0);
    draw_score_final(score);
    if (ratio >= 0.90)
        sprite_draw(sprite_S, 240, 250, 0, 0);
    else if (ratio >= 0.80)
        sprite_draw(sprite_A, 250, 250, 0, 0);
    else if (ratio >= 0.60)
        sprite_draw(sprite_B, 250, 250, 0, 0);
    else if (ratio >= 0.40)
        sprite_draw(sprite_C, 250, 250, 0, 0);
    else
        sprite_draw(sprite_D, 250, 250, 0, 0);
    framebuffer_swap();
```

```
    // wait for Enter
    while (1)
    {
        libusb_interrupt_transfer(keyboard, endpoint_address, (unsigned
char *)&packet, sizeof(packet), &transferred, 0);
        if (transferred == sizeof(packet) && packet.keycode[0] == 0x28)
            break;
        usleep(10000);
    }

    free_audio();
    unload_note_file();

    return 0;
#undef MAX_ACTIVE_NOTES
}

//-------------------------------------------------------main-----------
-----------------------------------------------
int main()
{
    framebuffer_init("/dev/vga_framebuffer"); // initialize framebuffer
    sprite_init();                            // initialize and load
sprites

    struct usb_keyboard_packet packet;
    int transferred;
    // char keystate[12];

    // Open the keyboard
    if ((keyboard = openkeyboard(&endpoint_address)) == NULL)
    {
        fprintf(stderr, "Did not find a keyboard\n");
        exit(1);
    }
    // Variables for menu interaction
    int selected_index = 0; // Currently selected menu item
    int in_menu = 1;
    display_menu(selected_index);
    // Look for and handle keypresses
    for (;;)
    {
```

```
        libusb_interrupt_transfer(keyboard, endpoint_address, (unsigned
char *)&packet, sizeof(packet), &transferred, 0);
        if (!in_menu || transferred == sizeof(packet))
        {
            if (!in_menu || packet.keycode[0] == 0x28)
            { // Break from submenu and back to main || Enter key
                if (!in_menu)
                { // Enter the menu
                    in_menu = 1;
                    display_menu(selected_index);
                }
                else
                { // Confirm the selected option
                    in_menu = 0;
                    switch (selected_index)
                    {
                    case 0: // Play
                        choose_music();
                        sleep(1);
                        play();
                        break;
                    case 1: // Tutorial
                        tutorial();
                        break;
                    case 2: // Difficulty
                        difficulty();
                        break;
                    case 3: // Exit
                        framebuffer_clear(0);
                        fbputs(250, 200, "Exiting...", 1);
                        framebuffer_swap();
                        sleep(2);
                        framebuffer_clear(0);
                        framebuffer_swap();
                        return 0;
                    }
                }
            }
            else if (packet.keycode[0] == 0x52)
            { // Up arrow pressed (keycode 0x52)
                if (in_menu)
                {
```

```
                    selected_index = (selected_index - 1 + MENU_ITEMS)
% MENU_ITEMS;
                    display_menu(selected_index);
                }
            }
            else if (packet.keycode[0] == 0x51)
            { // Down arrow pressed (keycode 0x51)
                if (in_menu)
                {
                    selected_index = (selected_index + 1) % MENU_ITEMS;
                    display_menu(selected_index);
                }
            }
        }
    }
    sprite_exit();      // release sprites
    framebuffer_exit(); // release framebuffer
}
```

Note.c

```
#include "Note.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

// Global variables
NoteEvent *note_events = NULL;
int note_event_count = 0;

int load_note_file(const char *path) {
    FILE *f = fopen(path, "r");
    if (!f) return -1;

    // Count the number of valid lines
    int lines = 0;
    char buf[128];
    while (fgets(buf, sizeof(buf), f)) {
        if (buf[0] == '#' || buf[0] == '\n') continue;
        lines++;
    }
    if (lines == 0) {
        fclose(f);
        return -1;
```

```
    }

    // Allocate memory
    note_events = malloc(lines * sizeof(NoteEvent));
    if (!note_events) {
        fclose(f);
        return -1;
    }

    // Rewind and parse
    rewind(f);
    int idx = 0;
    while (fgets(buf, sizeof(buf), f)) {
        if (buf[0] == '#' || buf[0] == '\n') continue;
        int t, tr;
        if (sscanf(buf, "%d , %d", &t, &tr) == 2) {
            note_events[idx].time_ms = t;
            note_events[idx].track = tr;
            idx++;
        }
    }
    fclose(f);

    note_event_count = idx;
    return 0;
}

int load_note_file_bin(const char *path) {
    // printf("Loading binary note file: %s\n", path);
    FILE *f = fopen(path, "rb");
    if (!f) return -1;

    // Get file size
    fseek(f, 0, SEEK_END);
    long size = ftell(f);
    rewind(f);

    if (size % sizeof(NoteEvent) != 0) {
        fclose(f);
        return -1;  // Corrupt file or incorrect format
    }

    int count = size / sizeof(NoteEvent);
```

```
    note_events = malloc(size);
    if (!note_events) {
        fclose(f);
        return -1;
    }

    if (fread(note_events, sizeof(NoteEvent), count, f) != count) {
        free(note_events);
        fclose(f);
        return -1;
    }

    fclose(f);
    note_event_count = count;
    return 0;
}

void unload_note_file(void) {
    free(note_events);
    note_events = NULL;
    note_event_count = 0;
}

NoteAnimation POS1 = {
    .frames = {
        {255, 0, 15, 6},
        {255, 0, 15, 6},
        {255, 0, 15, 6},
        {255, 0, 15, 6},
        {255, 0, 15, 6},
        {255, 1, 15, 6},
        {254, 1, 15, 6},
        {254, 1, 15, 6},
        {254, 2, 15, 6},
        {253, 3, 16, 6},
        {253, 4, 16, 6},
        {252, 6, 16, 6},
        {251, 7, 16, 7},
        {250, 9, 17, 7},
        {248, 12, 17, 7},
        {247, 14, 18, 7},
        {245, 17, 19, 7},
        {243, 21, 19, 8},
```

```
        {241, 25, 20, 8},
        {238, 29, 21, 8},
        {235, 34, 22, 9},
        {232, 39, 23, 9},
        {229, 45, 24, 9},
        {225, 52, 26, 10},
        {221, 59, 27, 11},
        {217, 66, 29, 11},
        {212, 74, 30, 12},
        {207, 83, 32, 12},
        {201, 93, 34, 13},
        {195, 103, 36, 14},
        {189, 114, 38, 15},
        {182, 126, 41, 16},
        {175, 139, 43, 17},
        {167, 152, 46, 18},
        {158, 167, 49, 19},
        {150, 182, 52, 20},
        {140, 198, 55, 21},
        {131, 215, 59, 23},
        {120, 233, 63, 24},
        {109, 251, 66, 25},
        {98, 271, 70, 27},
        {86, 292, 75, 29},
        {73, 314, 79, 30},
        {60, 337, 84, 32},
        {46, 361, 89, 34},
        {31, 386, 94, 36},
        {16, 413, 99, 38},
        {0, 440, 105, 40},
    }
};

NoteAnimation POS2 = {
    .frames = {
        {270, 0, 15, 6},
        {270, 0, 15, 6},
        {270, 0, 15, 6},
        {270, 0, 15, 6},
        {270, 0, 15, 6},
        {270, 1, 15, 6},
        {270, 1, 15, 6},
        {269, 1, 15, 6},
```

```
        {269, 2, 15, 6},
        {269, 3, 16, 6},
        {268, 4, 16, 6},
        {268, 6, 16, 6},
        {267, 7, 16, 7},
        {266, 9, 17, 7},
        {266, 12, 17, 7},
        {264, 14, 18, 7},
        {263, 17, 18, 7},
        {262, 21, 19, 8},
        {260, 25, 20, 8},
        {259, 29, 21, 8},
        {257, 34, 22, 9},
        {255, 39, 23, 9},
        {253, 45, 24, 9},
        {250, 52, 25, 10},
        {247, 59, 26, 11},
        {244, 66, 28, 11},
        {241, 74, 29, 12},
        {238, 83, 31, 12},
        {234, 93, 33, 13},
        {230, 103, 35, 14},
        {226, 114, 37, 15},
        {221, 126, 39, 16},
        {216, 139, 42, 17},
        {211, 152, 44, 18},
        {206, 167, 47, 19},
        {200, 182, 50, 20},
        {194, 198, 53, 21},
        {187, 215, 56, 23},
        {180, 233, 60, 24},
        {173, 251, 64, 25},
        {165, 271, 67, 27},
        {157, 292, 71, 29},
        {149, 314, 76, 30},
        {140, 337, 80, 32},
        {131, 361, 85, 34},
        {121, 386, 90, 36},
        {111, 413, 95, 38},
        {100, 440, 100, 40},
    }
};
```

```
NoteAnimation POS3 = {
    .frames = {
        {285, 0, 15, 6},
        {285, 0, 15, 6},
        {285, 0, 15, 6},
        {285, 0, 15, 6},
        {285, 0, 15, 6},
        {285, 1, 15, 6},
        {285, 1, 15, 6},
        {285, 1, 15, 6},
        {285, 2, 15, 6},
        {284, 3, 16, 6},
        {284, 4, 16, 6},
        {284, 6, 16, 6},
        {284, 7, 16, 7},
        {283, 9, 17, 7},
        {283, 12, 17, 7},
        {282, 14, 18, 7},
        {282, 17, 18, 7},
        {281, 21, 19, 8},
        {280, 25, 19, 8},
        {279, 29, 20, 8},
        {278, 34, 21, 9},
        {277, 39, 22, 9},
        {276, 45, 23, 9},
        {275, 52, 24, 10},
        {274, 59, 26, 11},
        {272, 66, 27, 11},
        {271, 74, 29, 12},
        {269, 83, 30, 12},
        {267, 93, 32, 13},
        {265, 103, 34, 14},
        {263, 114, 36, 15},
        {261, 126, 38, 16},
        {258, 139, 40, 17},
        {256, 152, 43, 18},
        {253, 167, 45, 19},
        {250, 182, 48, 20},
        {247, 198, 51, 21},
        {244, 215, 54, 23},
        {240, 233, 57, 24},
        {236, 251, 61, 25},
        {233, 271, 64, 27},
```

```
        {229, 292, 68, 29},
        {224, 314, 72, 30},
        {220, 337, 76, 32},
        {215, 361, 81, 34},
        {210, 386, 85, 36},
        {205, 413, 90, 38},
        {200, 440, 95, 40},
    }
};

NoteAnimation POS4 = {
    .frames = {
        {300, 0, 15, 6},
        {300, 0, 15, 6},
        {300, 0, 15, 6},
        {300, 0, 15, 6},
        {300, 0, 15, 6},
        {300, 1, 15, 6},
        {300, 1, 15, 6},
        {300, 1, 15, 6},
        {300, 2, 15, 6},
        {300, 3, 16, 6},
        {300, 4, 16, 6},
        {300, 6, 16, 6},
        {300, 7, 16, 7},
        {300, 9, 17, 7},
        {300, 12, 17, 7},
        {300, 14, 18, 7},
        {300, 17, 18, 7},
        {300, 21, 19, 8},
        {300, 25, 19, 8},
        {300, 29, 20, 8},
        {300, 34, 21, 9},
        {300, 39, 22, 9},
        {300, 45, 23, 9},
        {300, 52, 24, 10},
        {300, 59, 26, 11},
        {300, 66, 27, 11},
        {300, 74, 29, 12},
        {300, 83, 30, 12},
        {300, 93, 32, 13},
        {300, 103, 34, 14},
        {300, 114, 36, 15},
```

```
        {300, 126, 38, 16},
        {300, 139, 40, 17},
        {300, 152, 43, 18},
        {300, 167, 45, 19},
        {300, 182, 48, 20},
        {300, 198, 51, 21},
        {300, 215, 54, 23},
        {300, 233, 57, 24},
        {300, 251, 61, 25},
        {300, 271, 64, 27},
        {300, 292, 68, 29},
        {300, 314, 72, 30},
        {300, 337, 76, 32},
        {300, 361, 81, 34},
        {300, 386, 85, 36},
        {300, 413, 90, 38},
        {300, 440, 95, 40},
    }
};

NoteAnimation POS5 = {
    .frames = {
        {315, 0, 15, 6},
        {315, 0, 15, 6},
        {315, 0, 15, 6},
        {315, 0, 15, 6},
        {315, 0, 15, 6},
        {315, 1, 15, 6},
        {315, 1, 15, 6},
        {315, 1, 15, 6},
        {315, 2, 15, 6},
        {316, 3, 16, 6},
        {316, 4, 16, 6},
        {316, 6, 16, 6},
        {316, 7, 16, 7},
        {317, 9, 17, 7},
        {317, 12, 17, 7},
        {318, 14, 18, 7},
        {318, 17, 18, 7},
        {319, 21, 19, 8},
        {320, 25, 20, 8},
        {321, 29, 21, 8},
        {322, 34, 22, 9},
```

```
        {323, 39, 23, 9},
        {324, 45, 24, 9},
        {325, 52, 25, 10},
        {326, 59, 26, 11},
        {328, 66, 28, 11},
        {329, 74, 29, 12},
        {331, 83, 31, 12},
        {333, 93, 33, 13},
        {335, 103, 35, 14},
        {337, 114, 37, 15},
        {339, 126, 39, 16},
        {342, 139, 42, 17},
        {344, 152, 44, 18},
        {347, 167, 47, 19},
        {350, 182, 50, 20},
        {353, 198, 53, 21},
        {356, 215, 56, 23},
        {360, 233, 60, 24},
        {364, 251, 64, 25},
        {367, 271, 67, 27},
        {371, 292, 71, 29},
        {376, 314, 76, 30},
        {380, 337, 80, 32},
        {385, 361, 85, 34},
        {390, 386, 90, 36},
        {395, 413, 95, 38},
        {400, 440, 100, 40},
    }
};

NoteAnimation POS6 = {
    .frames = {
        {330, 0, 15, 6},
        {330, 0, 15, 6},
        {330, 0, 15, 6},
        {330, 0, 15, 6},
        {330, 0, 15, 6},
        {330, 1, 15, 6},
        {330, 1, 15, 6},
        {331, 1, 15, 6},
        {331, 2, 15, 6},
        {331, 3, 16, 6},
        {332, 4, 16, 6},
```

```
        {332, 6, 16, 6},
        {333, 7, 16, 7},
        {334, 9, 17, 7},
        {334, 12, 17, 7},
        {336, 14, 18, 7},
        {337, 17, 19, 7},
        {338, 21, 19, 8},
        {340, 25, 20, 8},
        {341, 29, 21, 8},
        {343, 34, 22, 9},
        {345, 39, 23, 9},
        {347, 45, 24, 9},
        {350, 52, 26, 10},
        {353, 59, 27, 11},
        {356, 66, 29, 11},
        {359, 74, 30, 12},
        {362, 83, 32, 12},
        {366, 93, 34, 13},
        {370, 103, 36, 14},
        {374, 114, 38, 15},
        {379, 126, 41, 16},
        {384, 139, 43, 17},
        {389, 152, 46, 18},
        {394, 167, 49, 19},
        {400, 182, 52, 20},
        {406, 198, 55, 21},
        {413, 215, 59, 23},
        {420, 233, 63, 24},
        {427, 251, 66, 25},
        {435, 271, 70, 27},
        {443, 292, 75, 29},
        {451, 314, 79, 30},
        {460, 337, 84, 32},
        {469, 361, 89, 34},
        {479, 386, 94, 36},
        {489, 413, 99, 38},
        {500, 440, 105, 40},
    }
};
```

Note.h

```
#ifndef _NOTE_H
#define _NOTE_H
```

```c
#define NOTE_COUNT 25
#define CHANNELS 6

typedef struct {
    int time_ms;   // timestamp
    int track;     // track number 0〜5
} NoteEvent;

extern NoteEvent *note_events;
extern int        note_event_count;

int load_note_file(const char *path);
int load_note_file_bin(const char *path);
void unload_note_file(void);

#define NOTE_ANIMATION 48
typedef struct {
    int frames[NOTE_ANIMATION][4];  // x, y, width, height
} NoteAnimation;

extern NoteAnimation POS1;
extern NoteAnimation POS2;
extern NoteAnimation POS3;
extern NoteAnimation POS4;
extern NoteAnimation POS5;
extern NoteAnimation POS6;

#endif
```

sprite.c

```c
#include "sprite.h"
#include "framebuffer.h"
#include "color.h"

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

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

// Define all sprites
```

```
sprite_t *sprite_0 = NULL;
sprite_t *sprite_1 = NULL;
sprite_t *sprite_2 = NULL;
sprite_t *sprite_3 = NULL;
sprite_t *sprite_4 = NULL;
sprite_t *sprite_5 = NULL;
sprite_t *sprite_6 = NULL;
sprite_t *sprite_7 = NULL;
sprite_t *sprite_8 = NULL;
sprite_t *sprite_9 = NULL;
sprite_t *sprite_A = NULL;
sprite_t *sprite_B = NULL;
sprite_t *sprite_bg = NULL;
sprite_t *sprite_C = NULL;
sprite_t *sprite_combo = NULL;
sprite_t *sprite_D = NULL;
sprite_t *sprite_great = NULL;
sprite_t *sprite_logo = NULL;
sprite_t *sprite_miss = NULL;
sprite_t *sprite_note1 = NULL;
sprite_t *sprite_note2 = NULL;
sprite_t *sprite_note3 = NULL;
sprite_t *sprite_note4 = NULL;
sprite_t *sprite_note5 = NULL;
sprite_t *sprite_note6 = NULL;
sprite_t *sprite_notep1 = NULL;
sprite_t *sprite_notep2 = NULL;
sprite_t *sprite_notep3 = NULL;
sprite_t *sprite_notep4 = NULL;
sprite_t *sprite_notep5 = NULL;
sprite_t *sprite_notep6 = NULL;
sprite_t *sprite_perfect = NULL;
sprite_t *sprite_S = NULL;
sprite_t *sprite_score = NULL;

sprite_t *sprite_load(const char *filename) {
    // Use stb_image to load PNG (lightweight and simpler than libpng)
    extern unsigned char *stbi_load(const char *filename, int *x, int
*y, int *channels_in_file, int desired_channels);
    extern void stbi_image_free(void *retval_from_stbi_load);

    int width, height, channels;
```

```
    unsigned char *img_data = stbi_load(filename, &width, &height,
&channels, 4); // Force convert to RGBA

    if (!img_data) {
        printf("Failed to load image: %s\n", filename);
        return NULL;
    }

    sprite_t *sprite = (sprite_t *)malloc(sizeof(sprite_t));
    sprite->width = width;
    sprite->height = height;
    sprite->pixels = (uint8_t *)malloc(width * height);

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int idx = (y * width + x) * 4;
            uint8_t r = img_data[idx + 0];
            uint8_t g = img_data[idx + 1];
            uint8_t b = img_data[idx + 2];
            uint8_t a = img_data[idx + 3];

            if (a == 0) {
                sprite->pixels[y * width + x] = 255;  // Transparent
pixel
            } else {
                sprite->pixels[y * width + x] = color_find_index(r, g,
b); // Map to palette color
            }
        }
    }

    stbi_image_free(img_data);
    return sprite;
}

void sprite_free(sprite_t *sprite) {
    if (sprite) {
        free(sprite->pixels);
        free(sprite);
    }
}

void sprite_init(void) {
```

```
    sprite_0 = sprite_load("./sprites/0.png");
    sprite_1 = sprite_load("./sprites/1.png");
    sprite_2 = sprite_load("./sprites/2.png");
    sprite_3 = sprite_load("./sprites/3.png");
    sprite_4 = sprite_load("./sprites/4.png");
    sprite_5 = sprite_load("./sprites/5.png");
    sprite_6 = sprite_load("./sprites/6.png");
    sprite_7 = sprite_load("./sprites/7.png");
    sprite_8 = sprite_load("./sprites/8.png");
    sprite_9 = sprite_load("./sprites/9.png");
    sprite_A = sprite_load("./sprites/A.png");
    sprite_B = sprite_load("./sprites/B.png");
    sprite_bg = sprite_load("./sprites/bg.png");
    sprite_C = sprite_load("./sprites/C.png");
    sprite_combo = sprite_load("./sprites/combo.png");
    sprite_D = sprite_load("./sprites/D.png");
    sprite_great = sprite_load("./sprites/great.png");
    sprite_logo = sprite_load("./sprites/logo.png");
    sprite_miss = sprite_load("./sprites/miss.png");
    sprite_note1 = sprite_load("./sprites/note1.png");
    sprite_note2 = sprite_load("./sprites/note2.png");
    sprite_note3 = sprite_load("./sprites/note3.png");
    sprite_note4 = sprite_load("./sprites/note4.png");
    sprite_note5 = sprite_load("./sprites/note5.png");
    sprite_note6 = sprite_load("./sprites/note6.png");
    sprite_notep1 = sprite_load("./sprites/notep1.png");
    sprite_notep2 = sprite_load("./sprites/notep2.png");
    sprite_notep3 = sprite_load("./sprites/notep3.png");
    sprite_notep4 = sprite_load("./sprites/notep4.png");
    sprite_notep5 = sprite_load("./sprites/notep5.png");
    sprite_notep6 = sprite_load("./sprites/notep6.png");
    sprite_perfect = sprite_load("./sprites/perfect.png");
    sprite_S = sprite_load("./sprites/S.png");
    sprite_score = sprite_load("./sprites/score.png");
}

void sprite_exit(void) {
    sprite_free(sprite_0);
    sprite_free(sprite_1);
    sprite_free(sprite_2);
    sprite_free(sprite_3);
    sprite_free(sprite_4);
    sprite_free(sprite_5);
```

```
    sprite_free(sprite_6);
    sprite_free(sprite_7);
    sprite_free(sprite_8);
    sprite_free(sprite_9);
    sprite_free(sprite_A);
    sprite_free(sprite_B);
    sprite_free(sprite_bg);
    sprite_free(sprite_C);
    sprite_free(sprite_combo);
    sprite_free(sprite_D);
    sprite_free(sprite_great);
    sprite_free(sprite_logo);
    sprite_free(sprite_miss);
    sprite_free(sprite_note1);
    sprite_free(sprite_note2);
    sprite_free(sprite_note3);
    sprite_free(sprite_note4);
    sprite_free(sprite_note5);
    sprite_free(sprite_note6);
    sprite_free(sprite_notep1);
    sprite_free(sprite_notep2);
    sprite_free(sprite_notep3);
    sprite_free(sprite_notep4);
    sprite_free(sprite_notep5);
    sprite_free(sprite_notep6);
    sprite_free(sprite_perfect);
    sprite_free(sprite_S);
    sprite_free(sprite_score);
}

void sprite_draw(sprite_t *sprite, int dst_x, int dst_y, int target_w,
int target_h) {
    if (!sprite) return;

    int src_w = sprite->width;
    int src_h = sprite->height;

    if (target_w <= 0) target_w = src_w;
    if (target_h <= 0) target_h = src_h;

    for (int y = 0; y < target_h; y++) {
        for (int x = 0; x < target_w; x++) {
            int src_x = x * src_w / target_w;
```

```
            int src_y = y * src_h / target_h;

            uint8_t color = sprite->pixels[src_y * src_w + src_x];
            if (color == 255) continue;

            int draw_x = dst_x + x;
            int draw_y = dst_y + y;

            if (draw_x >= 0 && draw_x < 640 && draw_y >= 0 && draw_y <
480) {
                framebuffer_draw_pixel(draw_x, draw_y, color);
            }
        }
    }
}

void split_digits(int num, int digits[5]) {
    if (num < 0) num = -num;
    digits[0] = (num / 10000) % 10;  // Ten-thousands place
    digits[1] = (num / 1000)  % 10;  // Thousands place
    digits[2] = (num / 100)   % 10;  // Hundreds place
    digits[3] = (num / 10)    % 10;  // Tens place
    digits[4] = num % 10;            // Units place
}

void draw_score(int score){
    sprite_draw(sprite_score, 380, 10, 80, 30);
    sprite_t *sprites[10] = {sprite_0, sprite_1, sprite_2, sprite_3,
sprite_4,
                             sprite_5, sprite_6, sprite_7, sprite_8,
sprite_9};
    int digits[5];
    split_digits(score, digits);
    for (int i = 0; i < 5; i++) {
        sprite_draw(sprites[digits[i]], 465 + i * 25, 10, 0, 0);
    }
}

void draw_score_final(int score){
    sprite_draw(sprite_score, 200, 200, 80, 30);
    sprite_t *sprites[10] = {sprite_0, sprite_1, sprite_2, sprite_3,
sprite_4,
```

```
                                sprite_5, sprite_6, sprite_7, sprite_8,
sprite_9};
    int digits[5];
    split_digits(score, digits);
    for (int i = 0; i < 5; i++) {
        sprite_draw(sprites[digits[i]], 290 + i * 25, 200, 0, 0);
    }
}

void draw_combo(int combo){
    sprite_draw(sprite_combo, 10, 10, 80, 30);
    sprite_t *sprites[10] = {sprite_0, sprite_1, sprite_2, sprite_3,
sprite_4,
                                sprite_5, sprite_6, sprite_7, sprite_8,
sprite_9};
    int digits[5];
    split_digits(combo, digits);
    for (int i = 0; i < 5; i++) {
        sprite_draw(sprites[digits[i]], 100 + i * 25, 10, 0, 0);
    }
}

void draw_bg (int combo, int score) {
    sprite_draw(sprite_bg, 0, 0, 600, 480);
    draw_combo(combo);
    draw_score(score);
}
```

sprite.h

```
#ifndef SPRITE_H
#define SPRITE_H

#include <stdint.h>

typedef struct sprite {
    int width;
    int height;
    uint8_t *pixels;
} sprite_t;

// Load a single sprite
sprite_t *sprite_load(const char *filename);
void sprite_free(sprite_t *sprite);
```

```
// Unified initialization and cleanup of all sprites
void sprite_init(void);
void sprite_exit(void);

// All sprites exposed for use in main
extern sprite_t *sprite_0;
extern sprite_t *sprite_1;
extern sprite_t *sprite_2;
extern sprite_t *sprite_3;
extern sprite_t *sprite_4;
extern sprite_t *sprite_5;
extern sprite_t *sprite_6;
extern sprite_t *sprite_7;
extern sprite_t *sprite_8;
extern sprite_t *sprite_9;
extern sprite_t *sprite_A;
extern sprite_t *sprite_B;
extern sprite_t *sprite_bg;
extern sprite_t *sprite_C;
extern sprite_t *sprite_combo;
extern sprite_t *sprite_D;
extern sprite_t *sprite_great;
extern sprite_t *sprite_logo;
extern sprite_t *sprite_miss;
extern sprite_t *sprite_note1;
extern sprite_t *sprite_note2;
extern sprite_t *sprite_note3;
extern sprite_t *sprite_note4;
extern sprite_t *sprite_note5;
extern sprite_t *sprite_note6;
extern sprite_t *sprite_notep1;
extern sprite_t *sprite_notep2;
extern sprite_t *sprite_notep3;
extern sprite_t *sprite_notep4;
extern sprite_t *sprite_notep5;
extern sprite_t *sprite_notep6;
extern sprite_t *sprite_perfect;
extern sprite_t *sprite_S;
extern sprite_t *sprite_score;

// Draw sprite, supports scaling
```

```
void sprite_draw(sprite_t *sprite, int dst_x, int dst_y, int target_w,
int target_h);
void draw_bg(int combo, int score);
void draw_combo(int combo);
void draw_score(int score);
void draw_score_final(int score);
void split_digits(int num, int digits[5]);

#endif // SPRITE_H
```

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
 *
https://web.archive.org/web/20210302095553/https://www.dreamincode.net/
forums/topic/148707-introduction-to-using-libusb-10/
 *
 * https://www.usb.org/sites/default/files/documents/hid1_11.pdf
 *
 * https://usb.org/sites/default/files/hut1_5.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 ;
    if ( inter->bInterfaceClass == LIBUSB_CLASS_HID &&
         inter->bInterfaceProtocol == USB_HID_KEYBOARD_PROTOCOL) {
      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 {
  uint8_t modifiers;
  uint8_t reserved;
  uint8_t keycode[6];
};

/* 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
```

vga_framebuffer.c

```
/* Device driver for the Guitar Hero Guitar
 *
 * Yangyang Zhang
 * Columbia University CSEE 4840 - Embedded Systems
 *
 * Adapted from code by Stephen A. Edwards, Columbia University
 * A Platform device implemented using the misc subsystem
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod vga_framebuffer.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree vga_framebuffer.c
 */

#include "vga_framebuffer.h"
#include <linux/errno.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/io.h>
#include <linux/kernel.h>
#include <linux/miscdevice.h>
#include <linux/module.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/platform_device.h>
#include <linux/slab.h>
#include <linux/uaccess.h>
#include <linux/version.h>

#define DRIVER_NAME "vga_framebuffer"

/*
 * Information about our device
 */
struct vga_framebuffer_dev {
    struct resource res; /* Resource: our registers */
```

```
    void __iomem *virtbase; /* Where registers can be accessed in
memory */
} dev;

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long vga_framebuffer_ioctl(struct file *f, unsigned int cmd,
                                  unsigned long arg) {
  vga_framebuffer_pixel_t pixel;

  switch (cmd) {
  case VGA_FRAMEBUFFER_WRITE_PIXEL:
      if (copy_from_user(&pixel, (vga_framebuffer_pixel_t *) arg,
      sizeof(vga_framebuffer_pixel_t)))
        return -EACCES;

      iowrite8(pixel.color, dev.virtbase + pixel.addr);
      break;

  default:
      return -EINVAL;
  }

  return 0;
}

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

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

```
/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_framebuffer_probe(struct platform_device *pdev)
{
  int ret;
    /* Register ourselves as a misc device: creates
/dev/vga_framebuffer */
    ret = misc_register(&vga_framebuffer_misc_device);

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

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

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

    return 0;

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

/* Clean-up code: release resources */
static int vga_framebuffer_remove(struct platform_device *pdev)
```

```
{
    iounmap(dev.virtbase);
    release_mem_region(dev.res.start, resource_size(&dev.res));
    misc_deregister(&vga_framebuffer_misc_device);
    return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_framebuffer_of_match[] = {
    { .compatible = "csee4840,vga_framebuffer-1.1" },
    {},
};
MODULE_DEVICE_TABLE(of, vga_framebuffer_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_framebuffer_driver = {
    .driver = {
        .name   = DRIVER_NAME,
        .owner  = THIS_MODULE,
        .of_match_table = of_match_ptr(vga_framebuffer_of_match),
    },
    .remove = __exit_p(vga_framebuffer_remove),
};

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

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

module_init(vga_framebuffer_init);
module_exit(vga_framebuffer_exit);
```

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Yangyang Zhang, Columbia University");
MODULE_DESCRIPTION("VGA framebuffer driver");
```

vga_framebuffer.h

```
/* Device driver for the Guitar Hero Guitar
 *
 * Yangyang Zhang
 * Columbia University CSEE 4840 - Embedded Systems
 *
 * Adapted from code by Stephen A. Edwards, Columbia University
 * A Platform device implemented using the misc subsystem
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod vga_framebuffer.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree vga_framebuffer.c
 */

#include "vga_framebuffer.h"
#include <linux/errno.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/io.h>
#include <linux/kernel.h>
#include <linux/miscdevice.h>
#include <linux/module.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/platform_device.h>
#include <linux/slab.h>
#include <linux/uaccess.h>
#include <linux/version.h>

#define DRIVER_NAME "vga_framebuffer"
```

```
/*
 * Information about our device
 */
struct vga_framebuffer_dev {
    struct resource res; /* Resource: our registers */
    void __iomem *virtbase; /* Where registers can be accessed in
memory */
} dev;

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long vga_framebuffer_ioctl(struct file *f, unsigned int cmd,
                                  unsigned long arg) {
  vga_framebuffer_pixel_t pixel;

  switch (cmd) {
  case VGA_FRAMEBUFFER_WRITE_PIXEL:
      if (copy_from_user(&pixel, (vga_framebuffer_pixel_t *) arg,
      sizeof(vga_framebuffer_pixel_t)))
        return -EACCES;

      iowrite8(pixel.color, dev.virtbase + pixel.addr);
      break;

  default:
      return -EINVAL;
  }

  return 0;
}

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

/* Information about our device for the "misc" framework -- like a char
dev */
```

```
static struct miscdevice vga_framebuffer_misc_device = {
    .minor      = MISC_DYNAMIC_MINOR,
    .name       = DRIVER_NAME,
    .fops       = &vga_framebuffer_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_framebuffer_probe(struct platform_device *pdev)
{
  int ret;
    /* Register ourselves as a misc device: creates
/dev/vga_framebuffer */
    ret = misc_register(&vga_framebuffer_misc_device);

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

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

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

    return 0;

out_release_mem_region:
    release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
```

```
    misc_deregister(&vga_framebuffer_misc_device);
    return ret;
}

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

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_framebuffer_of_match[] = {
    { .compatible = "csee4840,vga_framebuffer-1.1" },
    {},
};
MODULE_DEVICE_TABLE(of, vga_framebuffer_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_framebuffer_driver = {
    .driver = {
        .name   = DRIVER_NAME,
        .owner  = THIS_MODULE,
        .of_match_table = of_match_ptr(vga_framebuffer_of_match),
    },
    .remove = __exit_p(vga_framebuffer_remove),
};

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

/* Calball when the module is unloaded: release resources */
static void __exit vga_framebuffer_exit(void)
{
```

```
    platform_driver_unregister(&vga_framebuffer_driver);
    pr_info(DRIVER_NAME ": exit\n");
}

module_init(vga_framebuffer_init);
module_exit(vga_framebuffer_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Yangyang Zhang, Columbia University");
MODULE_DESCRIPTION("VGA framebuffer driver");
```