RHYTHM MASTER

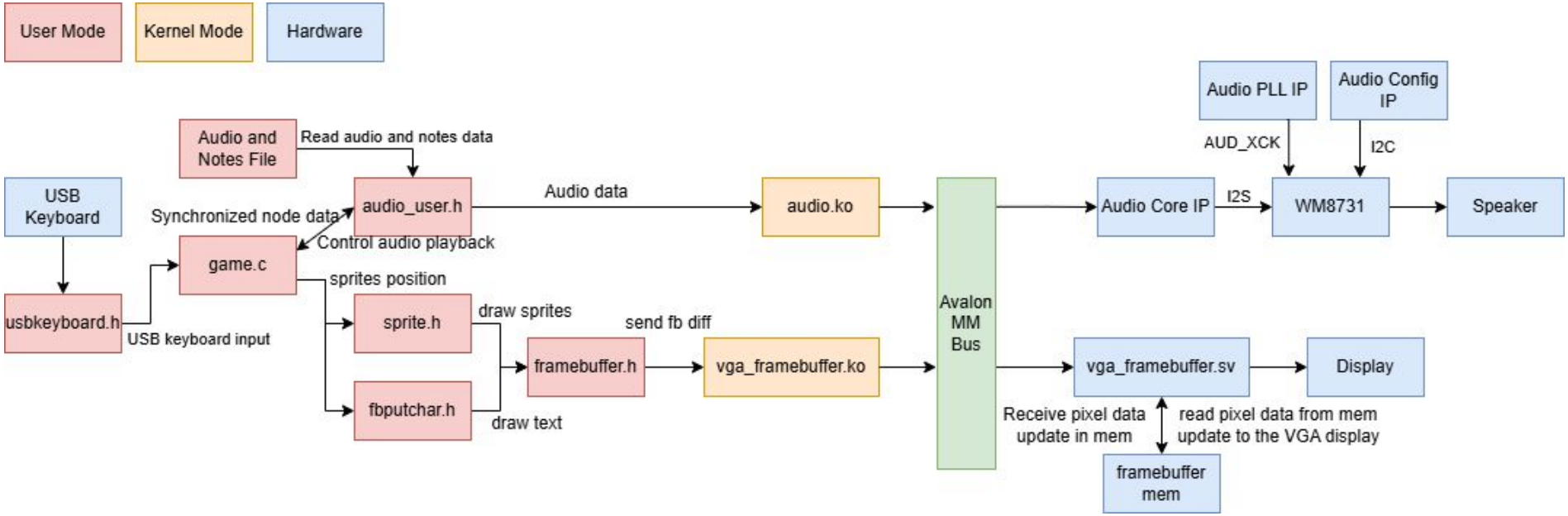
CSEE4840 Embedded System

Presenters:
Yangyang Zhang
Junfeng Zou
Hongrui Huang

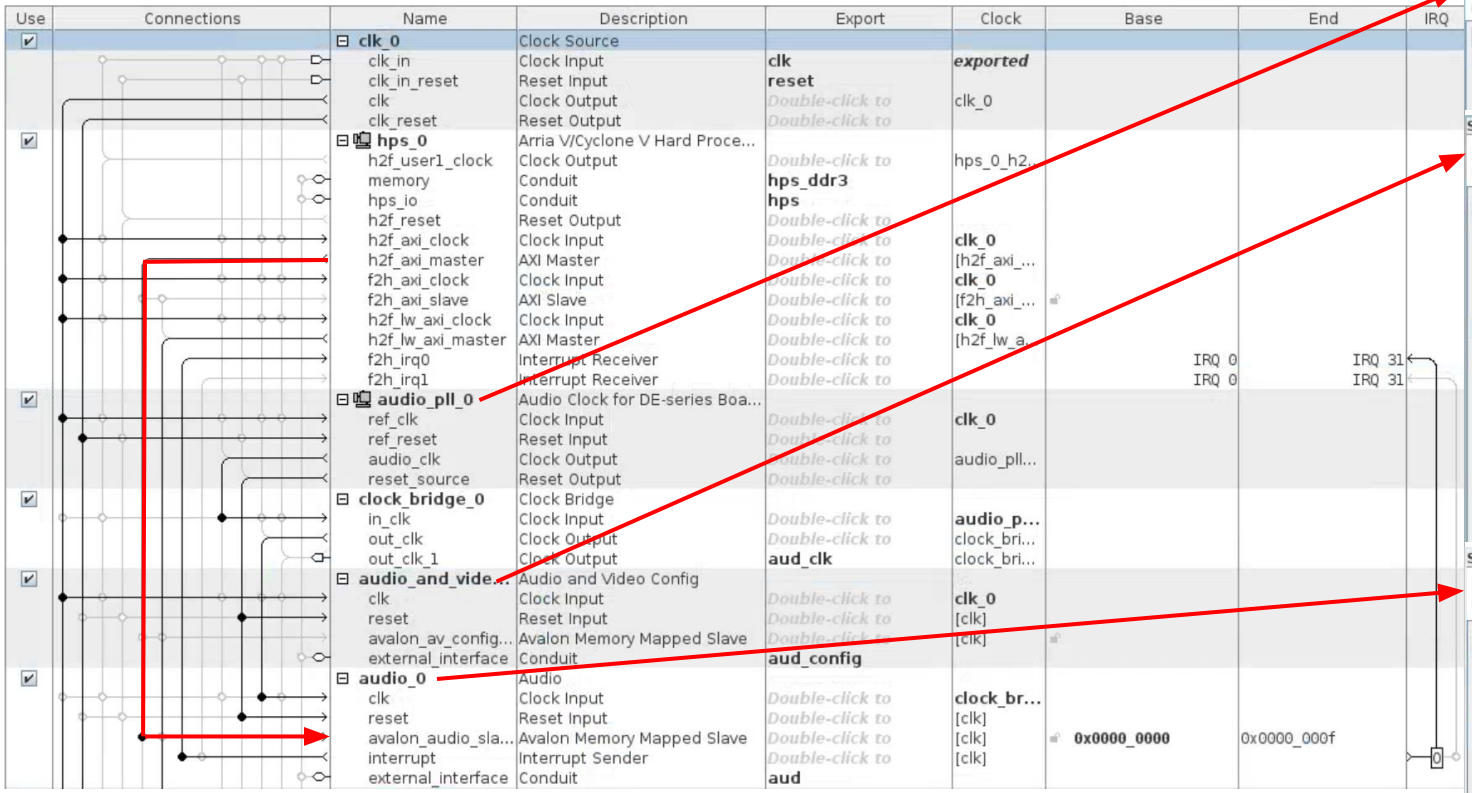
May 13, 2025

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Design Overview



Audio HW Control



Audio Control & HW/SW Interface

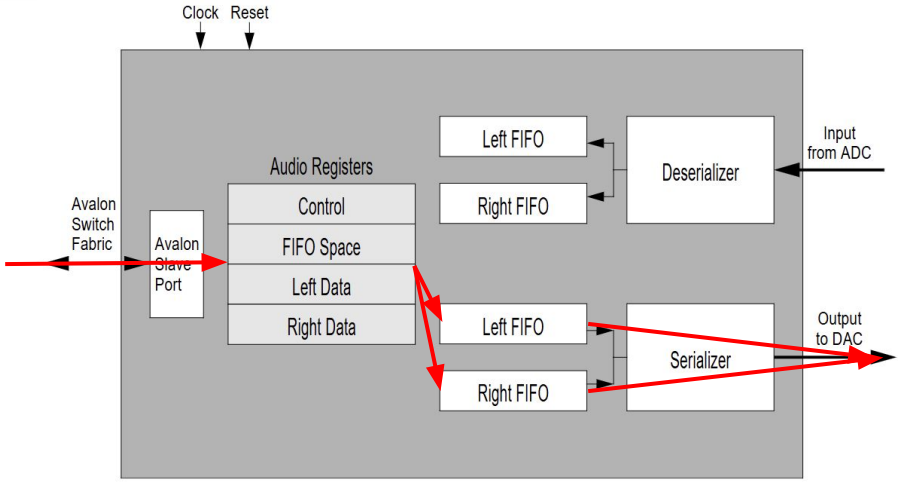


Figure 1. Block diagram for Audio core with Memory-Mapped Interface

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.

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.

4.1.2 *Fifospace* Register

The `fifospace` register fields WSLC (b_{31-24}) and WSRC (b_{23-16}) 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 (b_{15-8}) and RARC (b_{7-0}) 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.

4.1.3 *Leftdata* Register

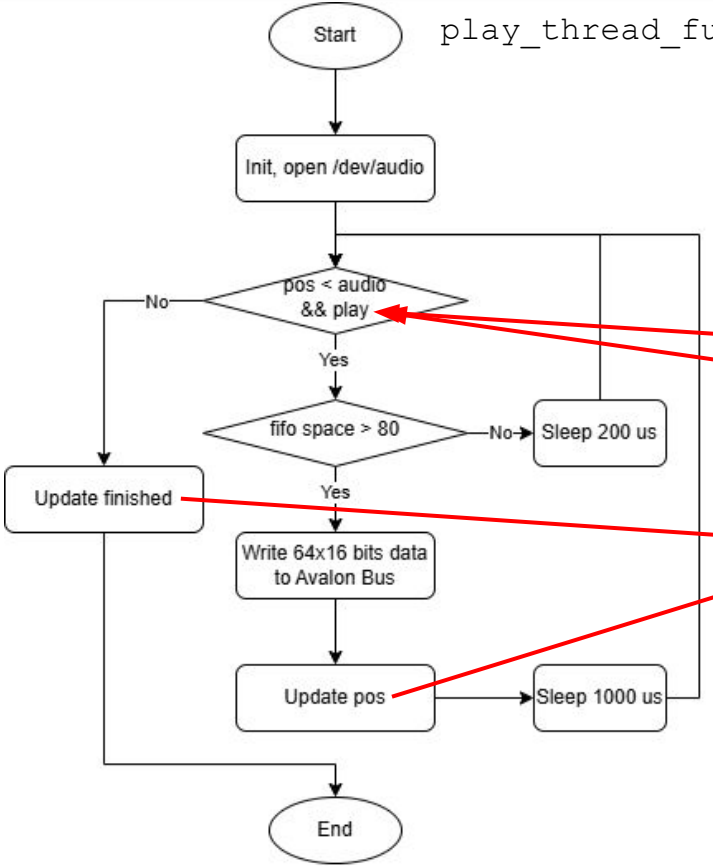
Each FIFO can store up to 128 32-bit words.
2 * 128 * 32 = 8192 bits

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 b_0 of the `leftdata` register.

4.1.4 *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 b_0 of the `rightdata` register.

Audio SW Control




```
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);
```

VGA HW Control

vga_framebuffer_dev → ioctl(VGA_FRAMEBUFFER_WRITE_PIXEL) → iowrite8(pixel.color, dev.virtbase + pixel.addr) →

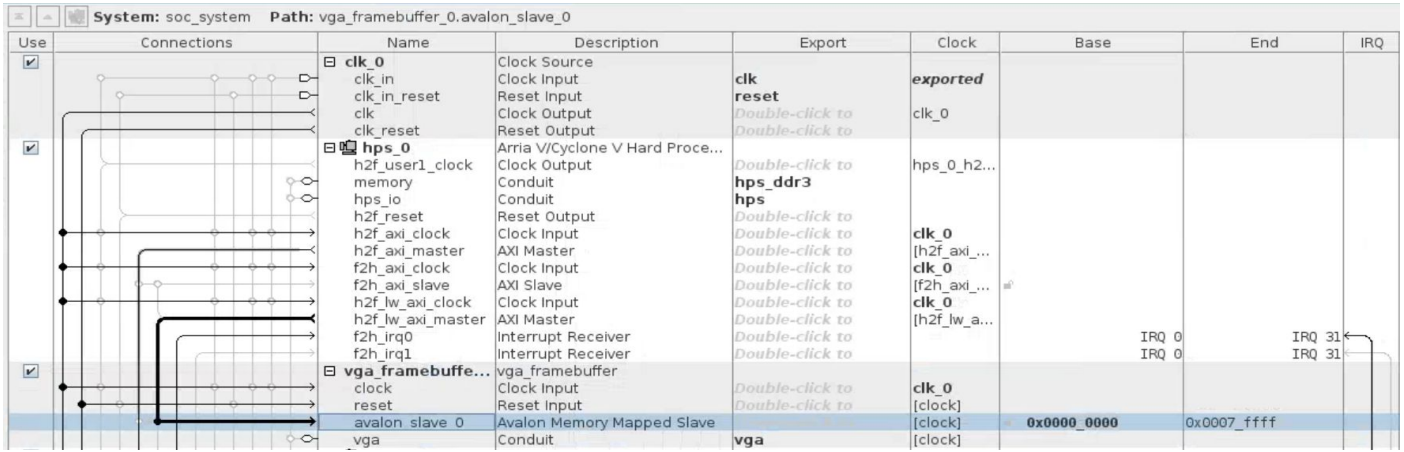
Propagate signals via Avalon MM →

vga_framebuffer.sv

Store colors → vga_mem

Calculate locations → vga_counters

vga_mem, vga_counters → Output: VGA_R, VGA_G, VGA_B and sync signal

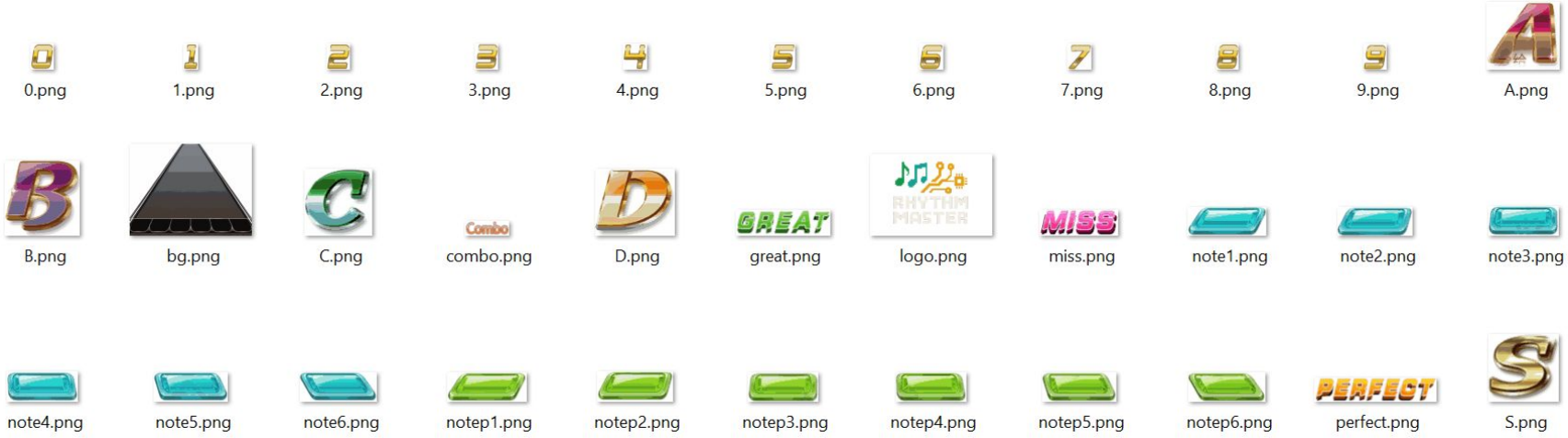


Resources used:
vga_mem
307200 * 8 bits = 2457600 bits

VGA HW/SW Interface

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]

Sprite SW Implementation




```
sprite_t *sprite_load(const char *filename);
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]);
void sprite_free(sprite_t *sprite);
```

Framebuffer SW Control

fbputchar.h

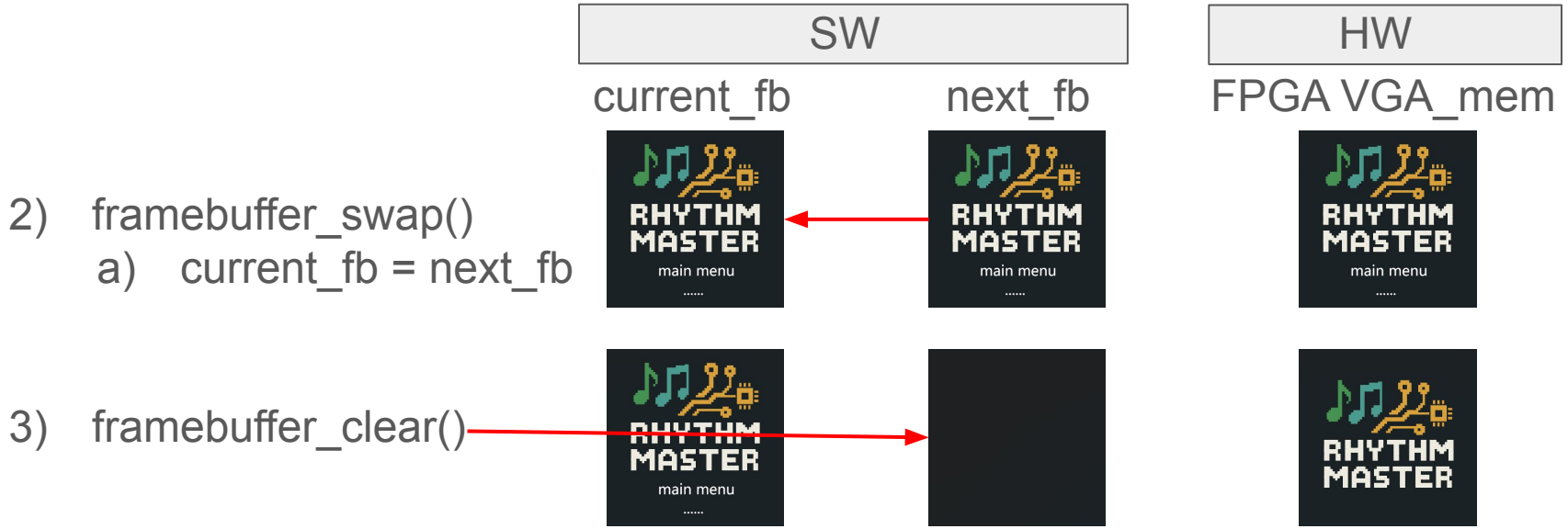
```
void fbputchar(int x, int y, char c, int color_index);
void fbputs(int x, int y, const char *s, int color_index);
```

sprite: logo 

1) sprite_draw(...)
   fbputs(...)

2) framebuffer_swap()
   a) Send diff via Avalon

Framebuffer SW Control



Ready for next frame …

Game Logic Key Functions

Game Menu

- Main Menu and Pause Menu
- Provide difficulty selection and song selection

Note Rendering

- Pre-processed beat maps
- Pre-processed note animation arrays

Hit Feedback

- Monitor keyboard for press detection
- Eliminate note and give feedback (MISS/GREAT/PERFECT)

Scoring

- Score = GREAT * 30 + PERFECT * 50 + combo bonus
- Level Feedback (S/A/B/C/D)

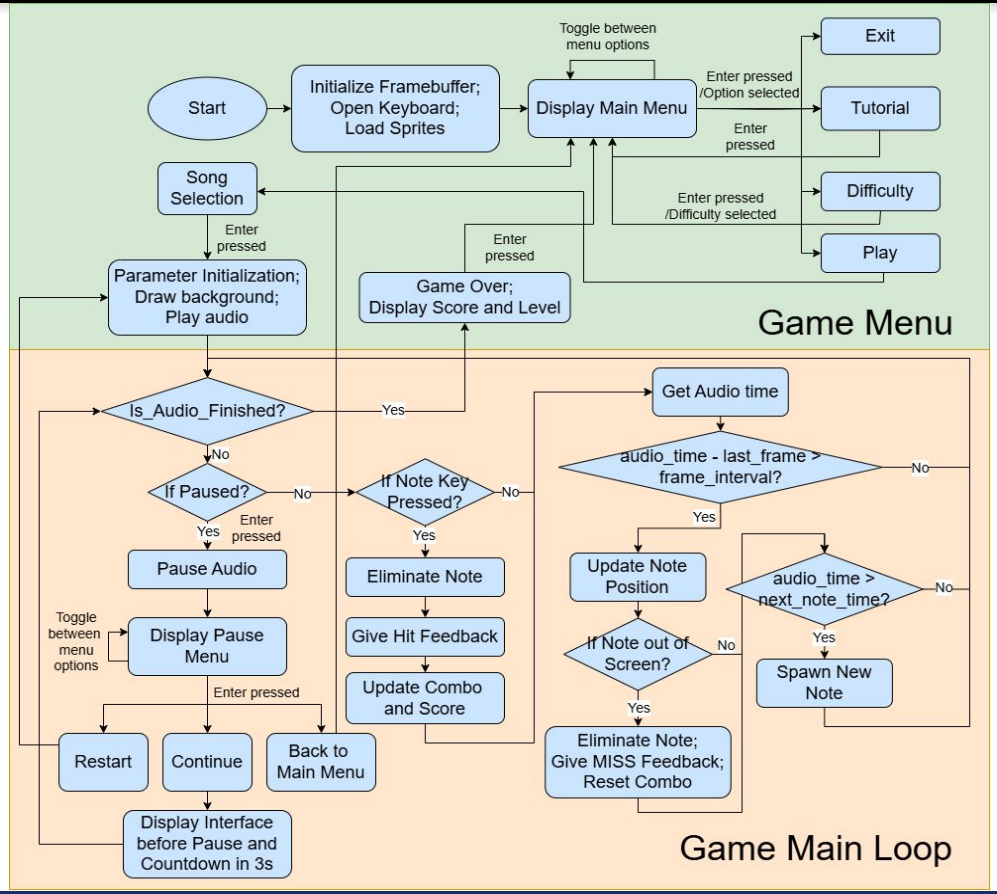
Game Controller

- USB Keyboard

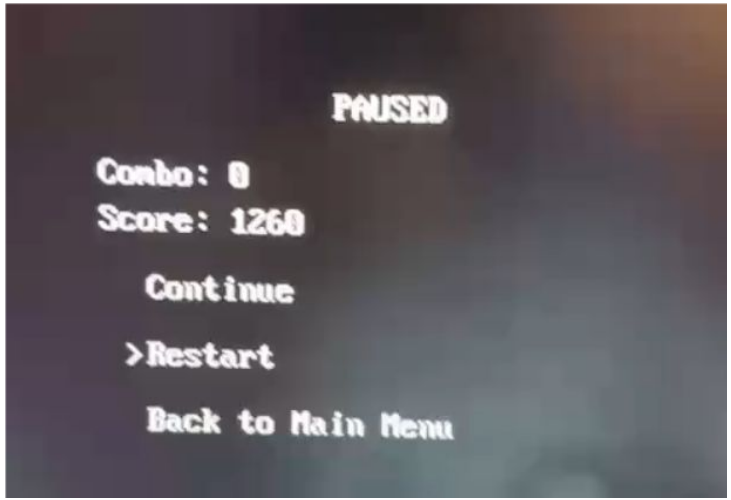
Confirm selection/Pause game

Move cursor up/down

Game Logic Flowchart



Demo



Main Menu, Song Selection Menu, and Pause Menu

Demo

Playing Interface and
Final Score Level