Piano Heroes Final Report

CSEE 4840: Embedded Systems
Spring 2025

Anita Bui-Martinez (adb2221), Michael John Flynn (mf3657),
Zakiy Tywon Manigo (ztm2106), Robel Wondwossen (rw3043)

Contents

1 Introduction

1.1 Overview

Piano Heroes is an FPGA-based video game inspired by the mobile game *Piano Tiles* created by Hu Wen Zeng in 2014. The goal of this project was to replicate the fast-paced, reflex-driven gameplay of Piano Tiles while integrating a real piano interface through a USB MIDI keyboard. Our version of the game expands on the original concept by supporting up to two octaves of notes, creating a more challenging and musically accurate experience.

Players must press the correct keys at the right time with the falling notes displayed on a VGA monitor. Each note corresponds to a specific piano key, and players must match the timing and order of the notes to continue playing. If they miss a note or press a note at the wrong time, they will lose and the game will end. The game requires precision and fast reflexes from the player. The project aims to replicate the engaging, rhythm-based challenge of the original game, while leveraging the performance and real-time capabilities of FPGA hardware.

1.2 Project Goals

- Implement real-time, responsive note detection using FPGA logic.
- Provide accurate audio feedback for each key press using high-quality WAV samples.
- Design a visually appealing VGA interface for the falling note tiles and real-time score display.
- Achieve low latency between key press, note hit detection, and audio playback.

1.3 System Design

The overall system architecture for Piano Heroes consists of three main components:

- **Hardware Design (FPGA Logic)**

Handles real-time note detection, hit logic, and VGA video output.

Uses SystemVerilog for high-speed processing of MIDI data.

Integrates with the ARM for more complex logic and control.

- **Software Design (Linux Side)**

  Interfaces with the USB MIDI keyboard for real-time note input.

  Handles game state, scoring, and timing logic.

  Uses C code for communication with the FPGA.

- **Hardware-Software Interface**

  Connects the ARM to the FPGA fabric via the Avalon bus.

  Uses memory-mapped I/O and interrupt-driven communication for low-latency data transfer.

  Processes MIDI messages and timestamps to synchronize note hit detection.

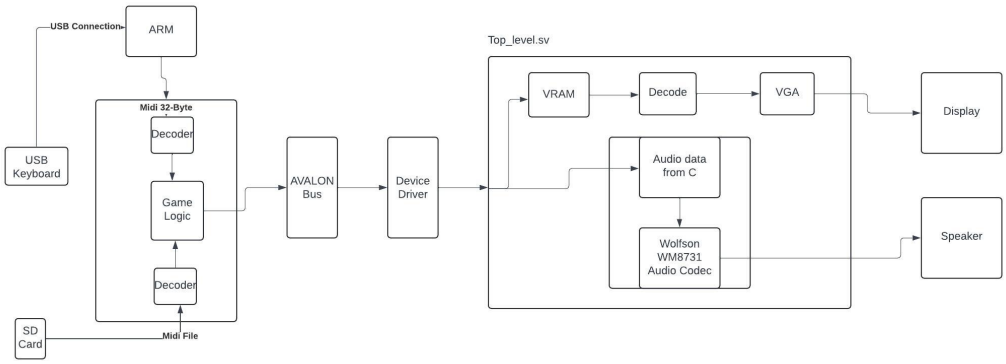

Figure 1: System Block Diagram

2 Project Design

2.1 Hardware

Our hardware contains a few key modules that run in parallel to ensure real-time performance:

- **Top-Level System** – This wires together the DE1-SoC's HPS interface, our custom RTL, the VGA controller, and the audio codec IP. It defines the memory-mapped base address and spans used by our custom peripherals.
- **MIDI Capture** - This small custom IP sits at offset 0x1000 within our peripheral space. It latches incoming 32-bit MIDI words and timestamps, raises an signal when new data arrives, and buffers them in a small FIFO.
- **Game Engine** - Reads from the MIDI FIFO, spawns "tiles" into a circular buffer (column, spawn timestamp, speed, color), computes each tile's y-position every VGA frame, and drives the VGA signal. It also contains the hit-line comparator and score counter.
- **VGA Controller** - A standard timing generator IP. The RTL overlays the playfield graphics on its sync signals, forcing specific RGB values when tiles or the hit line are active.
- **Audio Output** - Uses the Altera Audio + Audio PLL + Audio Config IPs to drive the Wolfson WM8731 at 48 kHz. When a Note On arrives, the engine pitches one of three stored WAV samples via a phase-accumulator block, mixing up to eight voices polyphonically.

We will now go into detail about a few aspects of the hardware.

2.1.1 VGA Output Display

The FPGA handles the generation of the VGA signal, including pixel timing and tile rendering. It uses a 2D array to represent the piano keys, with 3-bit color codes for different note states (e.g., white for unpressed, dark blue for active notes).
Key tasks include managing the hit line, rendering falling notes, and maintaining frame synchronization.

Position Calculation: The vertical speed of each tile is determined by the BPM and frame rate using the formula:

$$v = \frac{\text{pixels per beat}}{f \times \left(\frac{60}{\text{BPM}}\right)}$$

$$\text{pixels per beat} = \frac{\text{playfield\_height}}{\text{beats\_of\_lead\_time}}$$

$$y = (t_{\text{now}} - \text{spawn time}) \times v$$

where f is the frame rate and BPM is the beats per minute.
This would result in a caculation using numeric values like the following:

$$\text{pixels\_per\_beat} = \frac{\text{playfield\_height}}{\text{beats\_of\_lead\_time}} = \frac{480 \text{ px}}{4 \text{ beats}} = 120 \text{ px/beat}.$$

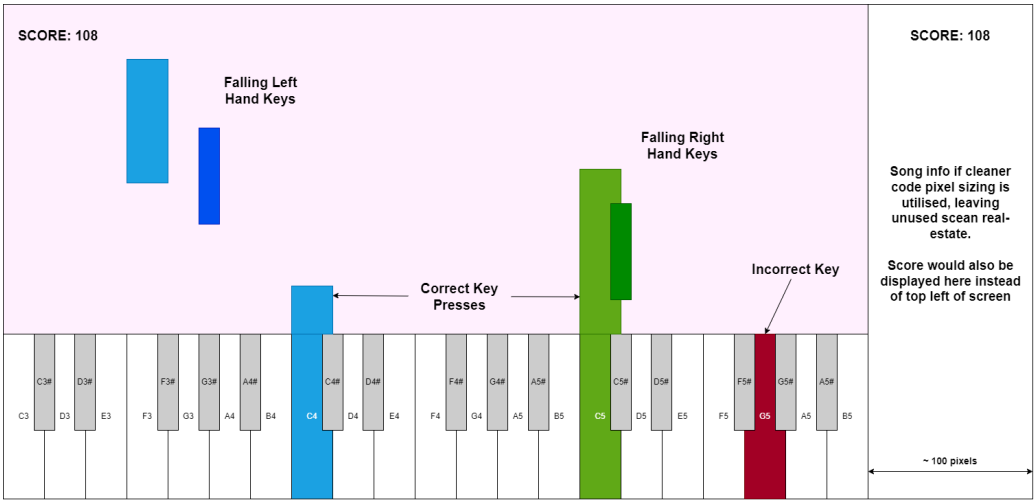

Figure 2: Display Map

2.1.2 MIDI Input Handling

The FPGA reads MIDI messages from the USB keyboard via physical memory mapped to addresses like 0xFF201000 for note data and 0xFF201004 for timestamps. The incoming data is processed to identify "Note On" messages, which trigger tile rendering and audio playback. This data is then passed to the game logic for hit detection.

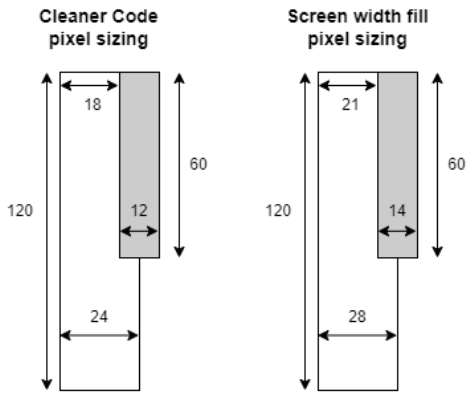

Figure 3: Pixel Sizing

2.1.3 Tile Data Structure

Active notes are managed in a circular buffer, with each tile represented by its:

- Column (corresponding to the piano key)
- Spawn time (when it enters the screen)
- Speed (based on the song's BPM)
- Color (indicating note state)

2.1.4 Audio Production Flow

Our audio hardware pipeline is designed for low-latency, polyphonic sound synthesis using minimal on-chip RAM. It consists of three major stages:

1. **Polyphonic Driver Peripheral**
    - Key Registers: Holds up to eight simultaneous "note-on" events, each with its own frequency and velocity.
    - Phase Accumulators: For each active voice, a phase accumulator steps through a stored waveform sample at a rate proportional to the note's frequency. By changing the phase increment, we pitch-shift a single base sample up or down.

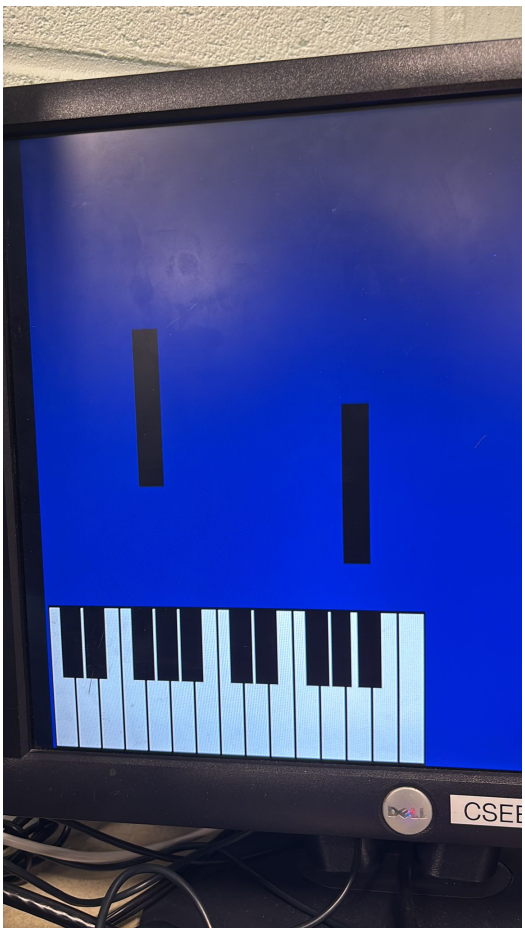

Figure 4: VGA Display

- Gain Multiplier and Mixer: Each voice's accumulator output is multiplied by a velocity-derived gain factor, then all voices are summed together in a lightweight mixer, producing a single digital audio stream.
- Audio RAM Controller: Fetches 48 kHz $\times$ 16 bit samples from on-chip RAM (2×1 s buffers for C3 and C4) and feeds them into the accumulators. This "two-sample" approach drastically cuts RAM usage compared to storing every note's waveform.

2. **On-Chip Audio RAM** We store exactly two 1s waveforms—one for each base note (C3/C4)—at 48 kHz, 16 bit. Pitch shifting is achieved in hardware via the phase accumulators rather than by storing separate waveforms for each note, keeping RAM utilization below 50 percent of the FPGA's available memory.

3. **Wolfson WM8731 Audio Codec** The mixed digital stream is sent over the Avalon Streaming interface into the Altera Audio IP (with Audio PLL and Config blocks) at 48 kHz, 24 bit. The WM8731 DAC converts this data into analog left/right outputs, driving headphones or speakers with sub-millisecond latency.

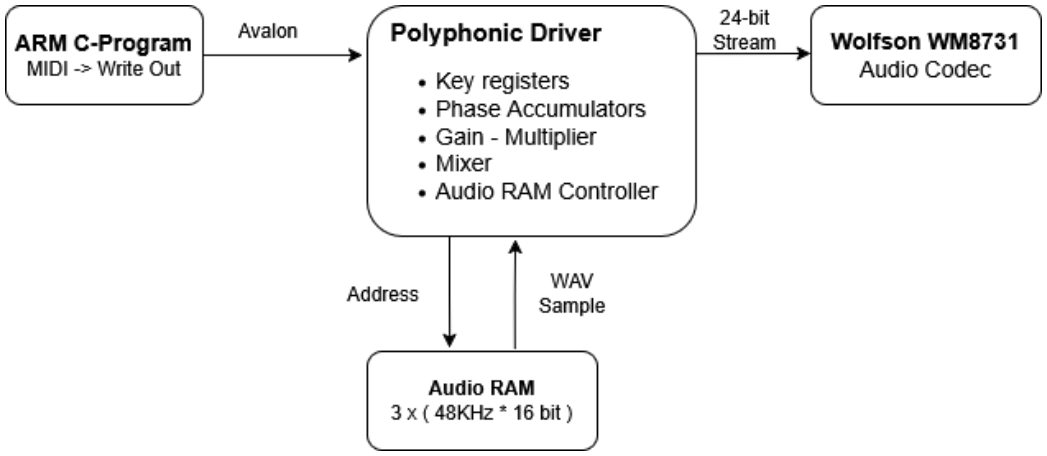

Figure 5: Audio Flow

2.2 Software

On the ARM Cortex-A9 running Linux, our software handles higher-level game control and raw MIDI ingestion:

- **MIDI Reader** - A userspace thread polls the USB endpoint on the Launchkey Mini, decodes USB MIDI "Note On" messages, and timestamps them with gettimeofday().
- **Memory Mapped I/O** - The same thread writes 32-bit words to specific addresses for the data and timestamp. It then signals the FPGA via an IRQ line.
- **Game State Manager** - A second thread reads back tile-hit interrupts from the FPGA, maintains the player's score and combo, and handles end-of-game conditions.
- **Audio Driver** - The same C program responds to key-press events by issuing small transfers of the appropriate WAV buffers to the Audio IP via its streaming interface, minimizing latency with direct writes to the FIFO.

We will now go into specifics about different parts of the software:

2.2.1 Input Handling

The main software component is a C program running on the ARM processor, responsible for capturing raw MIDI data. It uses libraries like libusb for low-level USB communication and sys/mman.h for memory-mapped I/O with the FPGA.

2.2.2 Game Logic

Tracks the state of each active note and manages the game lifecycle, including scoring and hit detection.

2.2.3 Audio Playback

MIDI messages are translated into corresponding WAV samples, which are pitch-shifted for accuracy without consuming excessive RAM. The audio subsystem is optimized to reduce latency and improve real-time response.

2.3 Hardware-Software Interface

Unfortunately, there we ran into some issues and were not able to fully implement the interfacing. However this is what was attempted.

2.3.1 AVALON Bus Communications

- The FPGA is connected to the ARM processor via the Avalon bus, using memory-mapped I/O for fast data transfer.
- The software writes MIDI data and timestamps to specific addresses within the FPGA's address space.

Register	Offset	Address
Peripheral base	—	0xFF20_0000
MIDI Data Register	0x1000	0xFF20_1000
Timestamp Register	0x1004	0xFF20_1004
Control / Status (e.g.)	0x1008	0xFF20_1008

Table 1: Memory map for the custom MIDI peripheral

2.3.2 Interrupt Handling

- The FPGA asserts interrupts to notify the ARM processor when new MIDI data is available or when a tile reaches the hit line.
- This ensures low-latency response times for both audio playback and tile collision detection.

3 Takeaways

3.1 Team Members

Anita Bui-Martinez, Michael John Flynn, Zakiy Tywon Manigo, Robel Wondwossen

3.2 Lessons Learned and Advice for Future Projects

- Give your team hard deadlines along the way to make sure you are on track.
- Work together along the way instead of having a hard division of labor and trying to put it together at the end. It makes things go more smoothly and makes disconnects between different parts of the project easier to mitigate.

4 Code

4.1 fpga_intf.sv

```
/*
 * Avalon memory-mapped peripheral that generates VGA and l/R Audio
 *
 * Team Piano Heros
 * Columbia University
 *
 * Register map:
 *   0x00 { VGA background red (8-bit)
 *   0x04 { VGA background green (8-bit)
 *   0x08 { VGA background blue (8-bit)
 *   0x0C { Audio note command (32-bit)
 *   0x10 { Audio sample1 value (32-bit)
 *   0x14 { Audio sample2 value (32-bit)
 *   0x18 - Midi input packets (64-bit)
 */
module fpga_intf (
    input  logic         clk,
    input  logic         reset,
    input  logic [31:0]  writedata,
    input  logic         write,
    input  logic         chipselect,
    input  logic [2:0]   address,
    input  logic         advance,
```

```
// Audio Outputs
output logic [23:0]  leftSample,
output logic [23:0]  rightSample,

// VGA Outputs
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
);

// VGA Counters
logic [10:0] hcount;
logic [9:0]  vcount;

// VGA Color Registers
logic [7:0] background_r, background_g, background_b;

// Audio Driver Control
logic [1:0]  audio_addr;
logic        audio_we;
logic        audio_cs;
logic [31:0] audio_data;

 // --- MIDI Input Tracking ---
logic [127:0] active_notes;
logic [63:0]  midi_input_packet;
logic [2:0]   midi_byte_count;
logic [7:0]   status, note, velocity;

// --- Falling Tile Simulation ---
logic [9:0] tile_y;
logic [3:0] tile_column;
logic [23:0] frame_counter;
```

```
// VGA Timing Module
vga_counters counters (
    .clk50      (clk),
    .reset      (reset),
    .hcount     (hcount),
    .vcount     (vcount),
    .VGA_CLK    (VGA_CLK),
    .VGA_HS     (VGA_HS),
    .VGA_VS     (VGA_VS),
    .VGA_BLANK_n(VGA_BLANK_n),
    .VGA_SYNC_n (VGA_SYNC_n)
);

// Polyphonic Audio Driver Instantiation
polyphonicDriver #(
    .SAMPLE_LEN(48000),
    .NUM_VOICES(8)
) poly_driver (
    .clk        (clk),
    .reset      (reset),
    .advance    (advance),
    .address    (audio_addr),
    .write      (audio_we),
    .chipselect (audio_cs),
    .writedata  (audio_data),
    .leftSample (leftSample),
    .rightSample(rightSample)
);

// === COMBINATIONAL VGA DRAWING ===
always_comb begin
    {VGA_R, VGA_G, VGA_B} = {background_r, background_g, background_b};

    if (VGA_BLANK_n) begin
        int key_top    = 360;
        int key_bottom = 480;
```

```
int border     = 2;
int white_key_width = 42;
int black_key_width = 38;
int black_key_height = 60;
int base_note = 60; // MIDI note C4
logic [7:0] local_note = 0;
int x1 = 0;
int x0 = 0;
int col = 0;
int tile_width;
int tile_x_start;
int tile_x_end;

tile_width = 42;
tile_x_start = 5 + tile_column * tile_width;
tile_x_end = tile_x_start + tile_width;

// === WHITE KEYS ===
if (vcount >= key_top && vcount < key_bottom) begin
    for (int i = 0; i < 15; i++) begin
        x0 = 5 + i * white_key_width;
        x1 = x0 + white_key_width;
        local_note = base_note + i;

        if (hcount >= x0 && hcount < x1) begin
            if ((hcount - x0 < border) || (x1 - hcount <= border) ||
                (vcount - key_top < border) || (key_bottom - vcount <=
                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00}; // boro
            end else if (active_notes[local_note]) begin
                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h00}; // gree
            end else begin
                {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF}; // whit
            end
        end
    end
end

// === BLACK KEYS ===
```

```
        if (vcount >= key_top && vcount < key_top + black_key_height) begin
            int black_key_cols[10]  = '{0, 1, 3, 4, 5, 7, 8, 10, 11, 12};
            int black_key_notes[10] = '{61, 63, 66, 68, 70, 73, 75, 78, 80, 82

            for (int j = 0; j < 10; j++) begin
                col = black_key_cols[j];
                local_note = black_key_notes[j];
                x0 = 5 + col * white_key_width + 21;
                x1 = x0 + black_key_width;

                if (hcount >= x0 && hcount < x1) begin
                    if (active_notes[local_note]) begin
                        {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h00}; // gree
                    end else begin
                        {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00}; // blac
                    end
                end
            end

            // === FALLING TILE DEMO ===
            if (vcount >= tile_y && vcount < tile_y + 120) begin
                if (hcount >= tile_x_start && hcount < tile_x_end) begin
                    {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hFF}; // cyan
                end
            end
        end
    end

    // Write decode logic (VGAs and audio all together)
    always_ff @(posedge clk) begin
        if (reset) begin
            background_r <= 8'h00;
            background_g <= 8'h00;
            background_b <= 8'h80;
            tile_y <= 0;
            tile_column <= 4'd4;
            frame_counter <= 0;
```

```
    midi_byte_count <= 0;
    midi_input_packet <= 0;
    active_notes <= 128'd0;
end else if (chipselect && write) begin
    case (address)
        3'd0: begin
            background_r <= writedata[7:0];
            audio_we     <= 1'b0;
            audio_cs     <= 1'b0;
        end
        3'd1: begin
            background_g <= writedata[7:0];
            audio_we     <= 1'b0;
            audio_cs     <= 1'b0;
        end
        3'd2: begin
            background_b <= writedata[7:0];
            audio_we     <= 1'b0;
            audio_cs     <= 1'b0;
        end
        3'd3: begin // Audio note command
            audio_we     <= 1'b1;
            audio_cs     <= 1'b1;
            audio_addr   <= 2'd0;
            audio_data   <= writedata;
        end
        3'd4: begin // Audio sample 1
            audio_we     <= 1'b1;
            audio_cs     <= 1'b1;
            audio_addr   <= 2'd1;
            audio_data   <= writedata;
        end
        3'd5: begin // Audio sample 2
            audio_we     <= 1'b1;
            audio_cs     <= 1'b1;
            audio_addr   <= 2'd2;
            audio_data   <= writedata;
        end
```

```
                3'd6: begin
                        midi_input_packet <= {midi_input_packet[55:0], writedata};
                        midi_byte_count <= midi_byte_count + 1;

                        if (midi_byte_count == 7) begin
                            status   <= midi_input_packet[39:32];
                            note     <= midi_input_packet[31:24];
                            velocity <= midi_input_packet[23:16];

                            if ((midi_input_packet[39:32] & 8'hF0) == 8'h90 && mid
                                active_notes[midi_input_packet[31:24]] <= 1'b1;
                            else if ((midi_input_packet[39:32] & 8'hF0) == 8'h80 |
                                     ((midi_input_packet[39:32] & 8'hF0) == 8'h90
                                active_notes[midi_input_packet[31:24]] <= 1'b0;

                            midi_byte_count <= 0;
                        end
                    end
                default: begin
                    audio_we     <= 1'b0;
                    audio_cs     <= 1'b0;
                end
            endcase
        end else begin
            audio_we <= 1'b0;
            audio_cs <= 1'b0;
        end
            tile_y <= 0;
            tile_column <= 4'd4;
            frame_counter <= 0;
            midi_byte_count <= 0;
            midi_input_packet <= 0;
            active_notes <= 128'd0;
    end


endmodule
```

```
//=============================================================================
// Polyphonic Module
//=============================================================================


module polyphonicDriver #(
    parameter SAMPLE_LEN = 48000,
    parameter NUM_VOICES = 8
)(
    input  logic        clk,
    input  logic        reset,

    // Avalon-MM interface
    input  logic [1:0]  address,
    input  logic        write,
    input  logic        chipselect,
    input  logic [31:0] writedata,

    // Audio interface
    input  logic        advance,
    output logic [23:0] leftSample,
    output logic [23:0] rightSample
);

    logic [7:0] note;
    logic cmd;

    //=====================================================
    // Sample Memories (sample1 = C3, sample2 = C4)
    //=====================================================
    logic [15:0] sample1_mem [0:SAMPLE_LEN-1];
    logic [15:0] sample2_mem [0:SAMPLE_LEN-1];
    logic [15:0] sample1_write_ptr, sample2_write_ptr;

    //=====================================================
    // Voice State
```

```
//=====================================================
typedef struct packed {
    logic        active;
    logic [7:0]  note;
    logic        sample_sel;
    logic [31:0] index;
    logic [31:0] step;
} voice_t;

voice_t voices[NUM_VOICES];
logic signed [23:0] voice_out[NUM_VOICES];

//=====================================================
// Pitch Step Lookup
//=====================================================
function automatic logic [31:0] pitch_step(input int semitone);
    case (semitone)
        0:   return 32'h00010000;
        1:   return 32'h00010F3B;
        2:   return 32'h00011F5C;
        3:   return 32'h0001306F;
        4:   return 32'h00014289;
        5:   return 32'h000155B5;
        6:   return 32'h00016A09;
        7:   return 32'h00017F91;
        8:   return 32'h00019660;
        9:   return 32'h0001AE8A;
        10:  return 32'h0001C824;
        11:  return 32'h0001E3C3;
        default: return 32'h00010000;
    endcase
endfunction

logic signed [23:0] mix;
int idx0;
int idx1;
int idx2;
int idx3;
```

```
int idx4;
int idx5;
int idx6;
int idx7;

logic [15:0] s0;
logic [15:0] s1;
logic [15:0] s2;
logic [15:0] s3;
logic [15:0] s4;
logic [15:0] s5;
logic [15:0] s6;
logic [15:0] s7;

//=====================================================
// Command Handling
//=====================================================
always_ff @(posedge clk) begin
if (reset) begin
    leftSample <= 24'd0;
    rightSample <= 24'd0;
    sample1_write_ptr <= 0;
    sample2_write_ptr <= 0;
    voices[0].active <= 0;
    voices[1].active <= 0;
    voices[2].active <= 0;
    voices[3].active <= 0;
    voices[4].active <= 0;
    voices[5].active <= 0;
    voices[6].active <= 0;
    voices[7].active <= 0;
end else if (write && chipselect) begin
    case (address)
        2'd0: begin // Note Command
            note <= writedata[7:0];
            cmd  <= writedata[8];
            if (cmd) begin // Note On
                if (!voices[0].active) begin
```

```
            voices[0].active     <= 1;
            voices[0].note       <= note;
            voices[0].sample_sel <= (note < 60) ? 0 : 1;
            voices[0].index      <= 0;
            voices[0].step       <= pitch_step((note < 60) ? (note - 4
        end else if (!voices[1].active) begin
            voices[1].active     <= 1;
            voices[1].note       <= note;
            voices[1].sample_sel <= (note < 60) ? 0 : 1;
            voices[1].index      <= 0;
            voices[1].step       <= pitch_step((note < 60) ? (note - 4
        end else if (!voices[2].active) begin
            voices[2].active     <= 1;
            voices[2].note       <= note;
            voices[2].sample_sel <= (note < 60) ? 0 : 1;
            voices[2].index      <= 0;
            voices[2].step       <= pitch_step((note < 60) ? (note - 4
        end else if (!voices[3].active) begin
            voices[3].active     <= 1;
            voices[3].note       <= note;
            voices[3].sample_sel <= (note < 60) ? 0 : 1;
            voices[3].index      <= 0;
            voices[3].step       <= pitch_step((note < 60) ? (note - 4
        end else if (!voices[4].active) begin
            voices[4].active     <= 1;
            voices[4].note       <= note;
            voices[4].sample_sel <= (note < 60) ? 0 : 1;
            voices[4].index      <= 0;
            voices[4].step       <= pitch_step((note < 60) ? (note - 4
        end else if (!voices[5].active) begin
            voices[5].active     <= 1;
            voices[5].note       <= note;
            voices[5].sample_sel <= (note < 60) ? 0 : 1;
            voices[5].index      <= 0;
            voices[5].step       <= pitch_step((note < 60) ? (note - 4
        end else if (!voices[6].active) begin
            voices[6].active     <= 1;
            voices[6].note       <= note;
```

```
                    voices[6].sample_sel <= (note < 60) ? 0 : 1;
                    voices[6].index      <= 0;
                    voices[6].step       <= pitch_step((note < 60) ? (note - 4
                end else if (!voices[7].active) begin
                    voices[7].active     <= 1;
                    voices[7].note       <= note;
                    voices[7].sample_sel <= (note < 60) ? 0 : 1;
                    voices[7].index      <= 0;
                    voices[7].step       <= pitch_step((note < 60) ? (note - 4
                end
            end else begin // Note Off
                if (voices[0].active && voices[0].note == note) voices[0].acti
                if (voices[1].active && voices[1].note == note) voices[1].acti
                if (voices[2].active && voices[2].note == note) voices[2].acti
                if (voices[3].active && voices[3].note == note) voices[3].acti
                if (voices[4].active && voices[4].note == note) voices[4].acti
                if (voices[5].active && voices[5].note == note) voices[5].acti
                if (voices[6].active && voices[6].note == note) voices[6].acti
                if (voices[7].active && voices[7].note == note) voices[7].acti
            end
        end
        2'd1: begin // Load Sample1
            sample1_mem[sample1_write_ptr] <= writedata[15:0];
            sample1_write_ptr <= sample1_write_ptr + 1;
        end
        2'd2: begin // Load Sample2
            sample2_mem[sample2_write_ptr] <= writedata[15:0];
            sample2_write_ptr <= sample2_write_ptr + 1;
        end
    endcase
end
//=====================================================
// Voice Playback and Mixing
//=====================================================
else if (advance) begin

        mix = 24'sd0;
   // ---------------- Voice 0 ----------------
```

```
if (voices[0].active) begin
    idx0 <= voices[0].index[31:16];
    if (idx0 >= SAMPLE_LEN) begin
        voices[0].active <= 1'b0;
        voice_out[0]     <= 24'sd0;
    end
    else begin
        s0              <= voices[0].sample_sel ? sample2_mem[idx0] : samp
        voice_out[0]    <= {{8{s0[15]}}, s0};
        voices[0].index <= voices[0].index + voices[0].step;
    end
end
else begin
    voice_out[0] <= 24'sd0;
end
mix += voice_out[0];

// ---------------- Voice 1 ----------------
if (voices[1].active) begin
    idx1 <= voices[1].index[31:16];
    if (idx1 >= SAMPLE_LEN) begin
        voices[1].active <= 1'b0;
        voice_out[1]     <= 24'sd0;
    end
    else begin
        s1               <= voices[1].sample_sel ? sample2_mem[idx1] : sam
        voice_out[1]    <= {{8{s1[15]}}, s1};
        voices[1].index <= voices[1].index + voices[1].step;
    end
end
else begin
    voice_out[1] <= 24'sd0;
end
mix += voice_out[1];

// ---------------- Voice 2 ----------------
if (voices[2].active) begin
    idx2 <= voices[2].index[31:16];
```

```
    if (idx2 >= SAMPLE_LEN) begin
        voices[2].active <= 1'b0;
        voice_out[2]     <= 24'sd0;
    end
    else begin
        s2               <= voices[2].sample_sel ? sample2_mem[idx2] : sam
        voice_out[2]    <= {{8{s2[15]}}, s2};
        voices[2].index <= voices[2].index + voices[2].step;
    end
end
else begin
    voice_out[2] <= 24'sd0;
end
mix += voice_out[2];

// ---------------- Voice 3 ----------------
if (voices[3].active) begin
    idx3 <= voices[3].index[31:16];
    if (idx3 >= SAMPLE_LEN) begin
        voices[3].active <= 1'b0;
        voice_out[3]     <= 24'sd0;
    end
    else begin
        s3               <= voices[3].sample_sel ? sample2_mem[idx3] : sam
        voice_out[3]    <= {{8{s3[15]}}, s3};
        voices[3].index <= voices[3].index + voices[3].step;
    end
end
else begin
    voice_out[3] <= 24'sd0;
end
mix += voice_out[3];

// ---------------- Voice 4 ----------------
if (voices[4].active) begin
    idx4 <= voices[4].index[31:16];
    if (idx4 >= SAMPLE_LEN) begin
        voices[4].active <= 1'b0;
```

```
        voice_out[4]     <= 24'sd0;
    end
    else begin
        s4               <= voices[4].sample_sel ? sample2_mem[idx4] : sam
        voice_out[4]    <= {{8{s4[15]}}, s4};
        voices[4].index <= voices[4].index + voices[4].step;
    end
end
else begin
    voice_out[4] <= 24'sd0;
end
mix += voice_out[4];

// ---------------- Voice 5 ----------------
if (voices[5].active) begin
    idx5 <= voices[5].index[31:16];
    if (idx5 >= SAMPLE_LEN) begin
        voices[5].active <= 1'b0;
        voice_out[5]     <= 24'sd0;
    end
    else begin
        s5               <= voices[5].sample_sel ? sample2_mem[idx5] : sam
        voice_out[5]    <= {{8{s5[15]}}, s5};
        voices[5].index <= voices[5].index + voices[5].step;
    end
end
else begin
    voice_out[5] <= 24'sd0;
end
mix += voice_out[5];

// ---------------- Voice 6 ----------------
if (voices[6].active) begin
    idx6 <= voices[6].index[31:16];
    if (idx6 >= SAMPLE_LEN) begin
        voices[6].active <= 1'b0;
        voice_out[6]     <= 24'sd0;
    end
```

```
            else begin
                s6               <= voices[6].sample_sel ? sample2_mem[idx6] : sam
                voice_out[6]    <= {{8{s6[15]}}, s6};
                voices[6].index <= voices[6].index + voices[6].step;
            end
        end
        else begin
            voice_out[6] <= 24'sd0;
        end
        mix += voice_out[6];

        // ---------------- Voice 7 ----------------
        if (voices[7].active) begin
            idx7 <= voices[7].index[31:16];
            if (idx7 >= SAMPLE_LEN) begin
                voices[7].active <= 1'b0;
                voice_out[7]     <= 24'sd0;
            end
            else begin
                s7               <= voices[7].sample_sel ? sample2_mem[idx7] : sam
                voice_out[7]    <= {{8{s7[15]}}, s7};
                voices[7].index <= voices[7].index + voices[7].step;
            end
        end
        else begin
            voice_out[7] <= 24'sd0;
        end
        mix += voice_out[7];

        // ----------------------------------------
        // Output mixed sample to left & right
        // ----------------------------------------
        leftSample  <= mix;
        rightSample <= mix;
    end
end
endmodule
```

```
//=============================================================================
//VGA MODULE
//=============================================================================

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

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0             1279       1599 0
 *             ---------------             --------
 * ___________|    Video      |___________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *       ------------------------      -------------
 * |____|       VGA_HS          |____|
 */
   // Parameters for hcount
   parameter HACTIVE      = 11'd 1280,
             HFRONT_PORCH = 11'd 32,
             HSYNC        = 11'd 192,
             HBACK_PORCH  = 11'd 96,
             HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                            HBACK_PORCH; // 1600

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

```
logic endOfLine;

always_ff @(posedge clk50 or posedge reset)
  if (reset)          hcount <= 0;
  else if (endOfLine) hcount <= 0;
  else            hcount <= hcount + 11'd 1;

assign endOfLine = hcount == HTOTAL - 1;

logic endOfField;

always_ff @(posedge clk50 or posedge reset)
  if (reset)          vcount <= 0;
  else if (endOfLine)
    if (endOfField)   vcount <= 0;
    else              vcount <= vcount + 10'd 1;

assign endOfField = vcount == VTOTAL - 1;

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

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

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

/* VGA_CLK is 25 MHz
 *             __    __    __
 * clk50    __|  |__|  |__|
 *
```

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

endmodule
```

4.2 fpga_intf.c

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

#define DRIVER_NAME "fpga_intf"

/* Register offsets (byte-aligned addresses) for FPGA registers */
#define BG_RED(x)      ((x) + 0x0)   /* Background red register address */
#define BG_GREEN(x)    ((x) + 0x4)   /* Background green */
#define BG_BLUE(x)     ((x) + 0x8)   /* Background blue */
#define NOTE_REG(x)    ((x) + 0xC)   /* Audio note command */
#define SAMPLE1_REG(x) ((x) + 0x10)  /* Audio sample1 */
#define SAMPLE2_REG(x) ((x) + 0x14)  /* Audio sample2 */

/* Device-specific structure to hold mapped resource and last values */
struct fpga_intf_dev {
    struct resource res;      /* memory resource for registers */
    void __iomem *virtbase;   /* virtual base address for FPGA regs */
```

```
    fpga_intf_color_t background; /* last written background color */
    fpga_intf_note_t note;        /* last written note value */
    fpga_intf_sample_t sample1;     /* last written sample1 value */
    fpga_intf_sample_t sample2;     /* last written sample2 value */
} dev;

/* Write VGA background color components to FPGA registers */
static void set_background(fpga_intf_color_t *bg)
{
    /* Write each color component to its aligned register (32-bit write) */
    iowrite8(bg->red,   BG_RED(dev.virtbase));
    iowrite8(bg->green, BG_GREEN(dev.virtbase));
    iowrite8(bg->blue,  BG_BLUE(dev.virtbase));
    dev.background = *bg;
}

/* Write audio note command to FPGA register */
static void set_note(fpga_intf_note_t *note)
{
    iowrite32(note->note, NOTE_REG(dev.virtbase));
    dev.note = *note;
}

/* Write first audio sample value to FPGA register */
static void set_sample1(fpga_intf_sample_t *samp)
{
    iowrite32(samp->sample, SAMPLE1_REG(dev.virtbase));
    dev.sample1 = *samp;
}

/* Write second audio sample value to FPGA register */
static void set_sample2(fpga_intf_sample_t *samp)
{
    iowrite32(samp->sample, SAMPLE2_REG(dev.virtbase));
    dev.sample2 = *samp;
}

/* ioctl handler for /dev/fpga_intf */
```

```
static long fpga_intf_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
    fpga_intf_color_t color_arg;
    fpga_intf_note_t  note_arg;
    fpga_intf_sample_t sample_arg;

    switch (cmd) {
    case FPGA_INTF_SET_BACKGROUND:
        if (copy_from_user(&color_arg, (fpga_intf_color_t __user *)arg, sizeof(col
            return -EACCES;
        set_background(&color_arg);
        break;

    case FPGA_INTF_GET_BACKGROUND:
        color_arg = dev.background;
        if (copy_to_user((fpga_intf_color_t __user *)arg, &color_arg, sizeof(color
            return -EACCES;
        break;

    case FPGA_INTF_SET_NOTE:
        if (copy_from_user(&note_arg, (fpga_intf_note_t __user *)arg, sizeof(note_
            return -EACCES;
        set_note(&note_arg);
        break;

    case FPGA_INTF_SET_SAMPLE1:
        if (copy_from_user(&sample_arg, (fpga_intf_sample_t __user *)arg, sizeof(s
            return -EACCES;
        set_sample1(&sample_arg);
        break;

    case FPGA_INTF_SET_SAMPLE2:
        if (copy_from_user(&sample_arg, (fpga_intf_sample_t __user *)arg, sizeof(s
            return -EACCES;
        set_sample2(&sample_arg);
        break;

    default:
```

```
        return -EINVAL;
    }

    return 0;
}

/* File operations structure */
static const struct file_operations fpga_intf_fops = {
    .owner            = THIS_MODULE,
    .unlocked_ioctl   = fpga_intf_ioctl,
};

/* Misc device for /dev/fpga_intf */
static struct miscdevice fpga_intf_misc_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name  = DRIVER_NAME,
    .fops  = &fpga_intf_fops,
};

/* Driver probe function - called when device is initialized */
static int __init fpga_intf_probe(struct platform_device *pdev)
{
    fpga_intf_color_t init_color = { .red=0xF9, .green=0xE4, .blue=0xB7, .pad=0x00
    fpga_intf_note_t init_note = { .note = 0 };
    fpga_intf_sample_t init_sample = { .sample = 0 };
    int ret;

    /* Register the misc device (/dev/fpga_intf) */
    ret = misc_register(&fpga_intf_misc_device);
    if (ret) {
        dev_err(&pdev->dev, "Failed to register misc device\n");
        return ret;
    }

    /* Get the physical address of the registers from device tree */
    ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
    if (ret) {
        misc_deregister(&fpga_intf_misc_device);
```

```
        return -ENOENT;
    }
    /* Request and ioremap the memory region for FPGA registers */
    if (request_mem_region(dev.res.start, resource_size(&dev.res), DRIVER_NAME) ==
        misc_deregister(&fpga_intf_misc_device);
        return -EBUSY;
    }
    dev.virtbase = of_iomap(pdev->dev.of_node, 0);
    if (dev.virtbase == NULL) {
        release_mem_region(dev.res.start, resource_size(&dev.res));
        misc_deregister(&fpga_intf_misc_device);
        return -ENOMEM;
    }

    /* Initialize FPGA registers with default values */
    set_background(&init_color);    // set initial background color
    set_note(&init_note);           // initialize note register to 0
    set_sample1(&init_sample);      // initialize audio sample1 to 0
    set_sample2(&init_sample);      // initialize audio sample2 to 0

    pr_info(DRIVER_NAME ": probe successful, mapped regs at 0x%pa\n", &dev.res.sta
    return 0;
}

/* Driver remove function */
static int fpga_intf_remove(struct platform_device *pdev)
{
    iounmap(dev.virtbase);
    release_mem_region(dev.res.start, resource_size(&dev.res));
    misc_deregister(&fpga_intf_misc_device);
    return 0;
}

#ifdef CONFIG_OF
static const struct of_device_id fpga_intf_of_match[] = {
    { .compatible = "csee4840,fpga_intf-1.0" },
    {},
};
```

```
MODULE_DEVICE_TABLE(of, fpga_intf_of_match);
#endif

static struct platform_driver fpga_intf_driver = {
    .driver = {
        .name  = DRIVER_NAME,
        .owner = THIS_MODULE,
        .of_match_table = of_match_ptr(fpga_intf_of_match),
    },
    .probe  = fpga_intf_probe,
    .remove = __exit_p(fpga_intf_remove),
};

/* Module init and exit */
static int __init fpga_intf_init(void)
{
    pr_info(DRIVER_NAME ": init\n");
    return platform_driver_register(&fpga_intf_driver);
}
static void __exit fpga_intf_exit(void)
{
    platform_driver_unregister(&fpga_intf_driver);
    pr_info(DRIVER_NAME ": exit\n");
}

module_init(fpga_intf_init);
module_exit(fpga_intf_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Team Piano Heros");
MODULE_DESCRIPTION("FPGA interface driver (VGA background and audio control)");
```

4.3 fpga_intf.h

```
#ifndef _FPGA_INTF_H
#define _FPGA_INTF_H

#include <linux/ioctl.h>
```

```
/* Structure for VGA background color (3 bytes color + padding for alignment) */
typedef struct {
    unsigned char red;
    unsigned char green;
    unsigned char blue;
} fpga_intf_color_t;

/* Structure for audio note command */
typedef struct {
    unsigned int note;
} fpga_intf_note_t;

/* Structure for audio sample value */
typedef struct {
    unsigned int sample;
} fpga_intf_sample_t;

/* Magic number for ioctl commands */
#define FPGA_INTF_MAGIC  'F'

/* ioctl command codes */
#define FPGA_INTF_SET_BACKGROUND  _IOW(FPGA_INTF_MAGIC, 1, fpga_intf_color_t)
#define FPGA_INTF_GET_BACKGROUND  _IOR(FPGA_INTF_MAGIC, 2, fpga_intf_color_t)
#define FPGA_INTF_SET_NOTE        _IOW(FPGA_INTF_MAGIC, 3, fpga_intf_note_t)
#define FPGA_INTF_SET_SAMPLE1     _IOW(FPGA_INTF_MAGIC, 4, fpga_intf_sample_t)
#define FPGA_INTF_SET_SAMPLE2     _IOW(FPGA_INTF_MAGIC, 5, fpga_intf_sample_t)

#endif /* _FPGA_INTF_H */
```

4.4 fpga_ioctl.h

```
// fpga_ioctl.h
#ifndef FPGA_IOCTL_H
#define FPGA_IOCTL_H

#include <linux/ioctl.h>
```

```
#define IOCTL_SEND_MIDI_EVENT _IOW('M', 3, uint64_t)

#endif
```

4.5 hardware_defs.h

```
#ifndef HARDWARE_DEFS_H
#define HARDWARE_DEFS_H

// FPGA memory mapping
#define HW_REGS_BASE        0xFF200000  // Base physical address of lightweight br
#define HW_REGS_SPAN        0x00200000  // 2MB span

// FPGA component offsets (relative to base)
#define SONG_LOADER_OFFSET  0x00002000  // Address to send parsed MIDI song notes
#define MIDI_INPUT_OFFSET   0x00002008  // Address to send real-time keyboard inpu
#define SONG_CTRL_OFFSET    0x00003000  // Address for song control block (3 x 32-
                                        // [0] = song_index       (read)
                                        // [1] = load_song_trigger (read)
                                        // [2] = song_loaded_done  (write)

#endif // HARDWARE_DEFS_H
```

4.6 hw_writer.h

```
#ifndef HW_WRITER_H
#define HW_WRITER_H

#include <stdint.h>
#include "midi_common.h"

uint64_t pack_midi_event(MidiEvent e);
uint64_t pack_midi_input(uint8_t status, uint8_t note, uint8_t velocity, uint64_t

#endif // HW_WRITER_H
```

4.7 midi_common.h

```
#ifndef MIDI_COMMON_H
#define MIDI_COMMON_H

#include <stdint.h>
#include <stddef.h>

// Represents a note event (either from a MIDI file or live input)
typedef struct {
    uint8_t note;
    uint8_t velocity;
    uint32_t timestamp_us;
    uint32_t duration_us;
    int active; // 1 = note on, 0 = rest
} MidiEvent;

// Basic MIDI file metadata
typedef struct {
    uint8_t *data;
    size_t size;
    uint16_t division;
    uint32_t tempo_us_per_quarter;
} MidiFile;

uint16_t read16(const uint8_t *data);
uint32_t read_variable_length(const uint8_t **data_ptr);
int transpose_note(int note);

#endif // MIDI_COMMON_H
```

4.8 src/logger/midi_logger.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <sys/time.h>
```

```
#include <libusb-1.0/libusb.h>

#include "midi_common.h"  // For future reuse of transpose_note(), etc.

#define VENDOR_ID        0x1235   // Focusrite-Novation
#define PRODUCT_ID       0x0102   // Launchkey Mini
#define INTERFACE_NUMBER 1        // MIDI Streaming Interface
#define ENDPOINT_IN      0x81     // MIDI IN endpoint (Bulk IN)

// Prints raw and decoded MIDI event with timestamp
void log_midi_packet(const unsigned char *packet, uint64_t timestamp_us) {
    uint8_t status = packet[1];
    uint8_t note   = packet[2];
    uint8_t velocity = packet[3];

    printf("[%llu us] Raw: %02X %02X %02X %02X  ",
           timestamp_us, packet[0], status, note, velocity);

    if ((status & 0xF0) == 0x90 && velocity > 0) {
        printf("Note On - Note: %d, Velocity: %d\n", note, velocity);
    } else if ((status & 0xF0) == 0x80 || ((status & 0xF0) == 0x90 && velocity ==
        printf("Note Off - Note: %d\n", note);
    } else {
        printf("Unhandled MIDI event\n");
    }
}

int main() {
    libusb_context *ctx = NULL;
    libusb_device_handle *handle = NULL;
    unsigned char buffer[64];
    int transferred, result;

    printf(" Launchkey MIDI Logger Starting...\n");

    if (libusb_init(&ctx) < 0) {
        fprintf(stderr, " libusb initialization failed.\n");
        return EXIT_FAILURE;
```

```
}

handle = libusb_open_device_with_vid_pid(ctx, VENDOR_ID, PRODUCT_ID);
if (!handle) {
    fprintf(stderr, " Could not find Launchkey Mini (VID: 0x1235, PID: 0x0102)
    libusb_exit(ctx);
    return EXIT_FAILURE;
}

libusb_set_auto_detach_kernel_driver(handle, 1);
libusb_detach_kernel_driver(handle, INTERFACE_NUMBER);

if (libusb_claim_interface(handle, INTERFACE_NUMBER) != 0) {
    fprintf(stderr, " Failed to claim interface %d.\n", INTERFACE_NUMBER);
    libusb_close(handle);
    libusb_exit(ctx);
    return EXIT_FAILURE;
}

printf(" MIDI interface claimed. Listening for MIDI events...\n");

while (1) {
    result = libusb_bulk_transfer(handle, ENDPOINT_IN, buffer, sizeof(buffer),
    if (result == 0 && transferred > 0) {
        struct timeval tv;
        gettimeofday(&tv, NULL);
        uint64_t timestamp_us = (uint64_t)tv.tv_sec * 1000000 + tv.tv_usec;

        for (int i = 0; i < transferred; i += 4) {
            if (i + 3 >= transferred) break;
            log_midi_packet(&buffer[i], timestamp_us);
        }
    } else if (result == LIBUSB_ERROR_TIMEOUT) {
        continue;
    } else {
        fprintf(stderr, " USB Transfer Error: %s\n", libusb_error_name(result)
        break;
    }
```

```
    }

    libusb_release_interface(handle, INTERFACE_NUMBER);
    libusb_close(handle);
    libusb_exit(ctx);

    return EXIT_SUCCESS;
}
```

4.9 src/logger/midi_logger_hw.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <sys/time.h>
#include <libusb-1.0/libusb.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>

#include "midi_common.h"
#include "hw_writer.h"
#include "fpga_ioctl.h"  // Include IOCTL command macros

#define VENDOR_ID        0x1235
#define PRODUCT_ID       0x0102
#define INTERFACE_NUMBER 1
#define ENDPOINT_IN      0x81
#define FPGA_DEVICE      "/dev/fpga_intf"

int main() {
    libusb_context *ctx = NULL;
    libusb_device_handle *handle = NULL;
    unsigned char buffer[64];
    int transferred, result;

    printf("Launchkey MIDI Logger with Kernel Driver Starting...\n");
```

```
// Open FPGA kernel device
int fd = open(FPGA_DEVICE, O_RDWR);
if (fd < 0) {
    perror(" Failed to open /dev/fpga_intf");
    return EXIT_FAILURE;
}

// Initialize USB MIDI connection
if (libusb_init(&ctx) < 0) {
    fprintf(stderr, "Failed to initialize libusb.\n");
    return EXIT_FAILURE;
}

handle = libusb_open_device_with_vid_pid(ctx, VENDOR_ID, PRODUCT_ID);
if (!handle) {
    fprintf(stderr, "Could not find Launchkey Mini.\n");
    libusb_exit(ctx);
    return EXIT_FAILURE;
}

libusb_set_auto_detach_kernel_driver(handle, 1);
libusb_detach_kernel_driver(handle, INTERFACE_NUMBER);

if (libusb_claim_interface(handle, INTERFACE_NUMBER) != 0) {
    fprintf(stderr, "Failed to claim MIDI interface.\n");
    libusb_close(handle);
    libusb_exit(ctx);
    return EXIT_FAILURE;
}

printf("MIDI interface claimed. Listening for input...\n");

while (1) {
    result = libusb_bulk_transfer(handle, ENDPOINT_IN, buffer, sizeof(buffer),
    if (result == 0 && transferred > 0) {
        struct timeval tv;
        gettimeofday(&tv, NULL);
```

```
            uint64_t timestamp_us = (uint64_t)tv.tv_sec * 1000000 + tv.tv_usec;

            for (int i = 0; i < transferred; i += 4) {
                if (i + 3 >= transferred) break;

                uint8_t status   = buffer[i + 1];
                uint8_t note     = buffer[i + 2];
                uint8_t velocity = buffer[i + 3];

                if ((status & 0xF0) == 0x90 && velocity > 0) {
                    uint64_t packet = pack_midi_input(status, note, velocity, time

                    printf("[%llu us] Note On: Note = %d, Velocity = %d\n",
                           timestamp_us, note, velocity);

                    if (ioctl(fd, IOCTL_SEND_MIDI_EVENT, &packet) < 0) {
                        perror("ioctl failed to send MIDI packet");
                    }

                    usleep(100);  // throttle
                }
            }
        } else if (result == LIBUSB_ERROR_TIMEOUT) {
            continue;
        } else {
            fprintf(stderr, " Transfer error: %s\n", libusb_error_name(result));
            break;
        }
    }

    libusb_release_interface(handle, INTERFACE_NUMBER);
    libusb_close(handle);
    libusb_exit(ctx);
    close(fd);

    return EXIT_SUCCESS;
}
```

4.10 src/song_loader/midi_song_loader.c

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

#include "midi_common.h"  // Shared definitions and functions

#define INPUT_MIDI_FILE   "songs_in_midi/harmony.mid"
#define OUTPUT_TEXT_FILE  "parsed_midi_output.txt"

void write_event_to_file(FILE *out, MidiEvent event) {
    uint8_t duration_ms = (event.duration_us / 1000) & 0xFF;
    uint32_t midi_word = (event.note << 24) | (event.velocity << 16) | (duration_m

    if (event.active) {
        fprintf(out, "MidiWord: 0x%08X  | Timestamp: %u us | Note: %d | Velocity:
                midi_word, event.timestamp_us, event.note, event.velocity, duratio
    } else {
        fprintf(out, "REST     : --------  | Timestamp: %u us | Duration: %d ms\n'
                event.timestamp_us, duration_ms);
    }
}

int load_midi_file(const char *filename, MidiFile *midi) {
    FILE *f = fopen(filename, "rb");
    if (!f) {
        perror("Cannot open MIDI file");
        return -1;
    }

    fseek(f, 0, SEEK_END);
    midi->size = ftell(f);
    fseek(f, 0, SEEK_SET);

    midi->data = (uint8_t *)malloc(midi->size);
    if (!midi->data) {
```

```
        fclose(f);
        fprintf(stderr, "Failed to allocate memory for MIDI file.\n");
        return -1;
    }

    fread(midi->data, 1, midi->size, f);
    fclose(f);

    if (memcmp(midi->data, "MThd", 4) != 0) {
        fprintf(stderr, "Not a valid MIDI file\n");
        return -1;
    }

    midi->division = read16(midi->data + 12);
    midi->tempo_us_per_quarter = 500000;  // Default 120 BPM
    return 0;
}

int main() {
    MidiFile midi;
    if (load_midi_file(INPUT_MIDI_FILE, &midi) != 0) return -1;

    FILE *out = fopen(OUTPUT_TEXT_FILE, "w");
    if (!out) {
        perror("Cannot create output file");
        free(midi.data);
        return -1;
    }

    const uint8_t *ptr = midi.data + 14;
    if (memcmp(ptr, "MTrk", 4) != 0) {
        fprintf(stderr, "No MTrk chunk found\n");
        fclose(out);
        free(midi.data);
        return -1;
    }
    ptr += 8;  // Skip 'MTrk' and length
```

```
uint32_t current_ticks = 0;
uint32_t last_event_us = 0;
uint8_t last_status = 0;
double micros_per_tick = (double)midi.tempo_us_per_quarter / midi.division;

while (ptr < midi.data + midi.size) {
    uint32_t delta_ticks = read_variable_length(&ptr);
    current_ticks += delta_ticks;
    uint32_t current_us = (uint32_t)(current_ticks * micros_per_tick);

    uint8_t status = *ptr;
    if (status < 0x80) {
        status = last_status;  // Running status
    } else {
        ptr++;
        last_status = status;
    }

    if ((status & 0xF0) == 0x90 && ptr[1] > 0) {
        // Insert REST if there's a time gap
        if (current_us > last_event_us) {
            uint32_t rest_duration = current_us - last_event_us;
            if (rest_duration > 0) {
                MidiEvent rest = {
                    .note = 0,
                    .velocity = 0,
                    .timestamp_us = last_event_us,
                    .duration_us = rest_duration,
                    .active = 0
                };
                write_event_to_file(out, rest);
            }
        }

        // NOTE ON: Search ahead for matching NOTE OFF
        uint8_t note = ptr[0];
        uint8_t velocity = ptr[1];
        uint32_t note_on_ticks = current_ticks;
```

```
            const uint8_t *search_ptr = ptr + 2;
            uint32_t search_ticks = current_ticks;
            uint8_t search_status = last_status;

            while (search_ptr < midi.data + midi.size) {
                uint32_t search_delta = read_variable_length(&search_ptr);
                search_ticks += search_delta;

                uint8_t s = *search_ptr;
                if (s < 0x80) s = search_status;
                else {
                    search_ptr++;
                    search_status = s;
                }

                if (((s & 0xF0) == 0x80 || ((s & 0xF0) == 0x90 && search_ptr[1] ==
                    uint32_t duration_ticks = search_ticks - note_on_ticks;
                    uint32_t duration_us = (uint32_t)(duration_ticks * micros_per_

                    MidiEvent event = {
                        .note = transpose_note(note),
                        .velocity = velocity,
                        .timestamp_us = current_us,
                        .duration_us = duration_us,
                        .active = 1
                    };
                    write_event_to_file(out, event);
                    last_event_us = current_us + duration_us;
                    break;
                }
                search_ptr += 2;  // Skip note and velocity
            }
        }

        ptr += 2;  // Advance to next event
    }

    fclose(out);
```

```
    free(midi.data);

    printf("Parsed MIDI saved to: %s\n", OUTPUT_TEXT_FILE);
    return 0;
}
```

4.11 src/song_loader/midi_song_loader_hw.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <time.h>

#include "midi_common.h"
#include "hw_writer.h"
#include "fpga_ioctl.h"  //  Add this header to access IOCTL_SEND_MIDI_EVENT

#define MAX_ACTIVE_NOTES 128
#define FPGA_DEVICE "/dev/fpga_intf"

typedef struct {
    uint8_t note;
    uint8_t velocity;
    uint32_t start_ticks;
    uint32_t start_us;
    int active;
} ActiveNote;

int load_midi(const char *filename, MidiFile *midi) {
    FILE *f = fopen(filename, "rb");
    if (!f) {
        perror("Cannot open MIDI file");
        return -1;
```

```
    }

    fseek(f, 0, SEEK_END);
    midi->size = ftell(f);
    fseek(f, 0, SEEK_SET);
    midi->data = (uint8_t *)malloc(midi->size);
    fread(midi->data, 1, midi->size, f);
    fclose(f);

    if (memcmp(midi->data, "MThd", 4) != 0) {
        fprintf(stderr, "Not a valid MIDI file\n");
        return -1;
    }

    midi->division = read16(midi->data + 12);
    midi->tempo_us_per_quarter = 500000;
    return 0;
}

void parse_and_send_midi(MidiFile *midi, int fd) {
    const uint8_t *ptr = midi->data + 14;
    if (memcmp(ptr, "MTrk", 4) != 0) {
        fprintf(stderr, "No MTrk chunk found\n");
        return;
    }
    ptr += 8;

    ActiveNote active_notes[MAX_ACTIVE_NOTES] = {0};
    uint32_t current_ticks = 0;
    uint8_t last_status = 0;
    double micros_per_tick = (double)midi->tempo_us_per_quarter / midi->division;

    while (ptr < midi->data + midi->size) {
        uint32_t delta_ticks = read_variable_length(&ptr);
        current_ticks += delta_ticks;
        uint32_t current_us = (uint32_t)(current_ticks * micros_per_tick);

        uint8_t status = *ptr;
```

```
if (status < 0x80) {
    status = last_status;
} else {
    ptr++;
    last_status = status;
}

if (status == 0xFF && ptr[0] == 0x51 && ptr[1] == 0x03) {
    ptr += 2;
    uint32_t new_tempo = (ptr[0] << 16) | (ptr[1] << 8) | ptr[2];
    midi->tempo_us_per_quarter = new_tempo;
    micros_per_tick = (double)new_tempo / midi->division;
    printf(" Tempo Change: %u us/quarter → %.2f µs/tick\n", new_tempo, mic
    ptr += 3;
    continue;
}

if ((status & 0xF0) == 0x90 && ptr[1] > 0) {
    uint8_t note = transpose_note(ptr[0]);
    uint8_t velocity = ptr[1];

    active_notes[note].note = note;
    active_notes[note].velocity = velocity;
    active_notes[note].start_ticks = current_ticks;
    active_notes[note].start_us = current_us;
    active_notes[note].active = 1;
    ptr += 2;
} else if ((status & 0xF0) == 0x80 || ((status & 0xF0) == 0x90 && ptr[1] =
    uint8_t note = transpose_note(ptr[0]);

    if (active_notes[note].active) {
        uint32_t note_on_us = active_notes[note].start_us;
        uint32_t duration_us = current_us - note_on_us;

        MidiEvent e = {
            .note = note,
            .velocity = active_notes[note].velocity,
            .timestamp_us = note_on_us,
```

```
                    .duration_us = duration_us,
                    .active = 1
                };

                uint64_t packet = pack_midi_event(e);

                // Send packet to kernel driver via ioctl
                if (ioctl(fd, IOCTL_SEND_MIDI_EVENT, &packet) < 0) {
                    perror("ioctl failed to send MIDI packet");
                }

                printf(" Packet Sent - Note: %d | Velocity: %d | Duration: %d ms |
                       e.note, e.velocity, e.duration_us / 1000, e.timestamp_us);

                active_notes[note].active = 0;
            }
            ptr += 2;
        } else {
            ptr += 2;
        }
    }
}

int main() {
    printf("Press KEY1 to load and send song1.mid...\n");

    int fd = open(FPGA_DEVICE, O_RDWR);
    if (fd < 0) {
        perror("Failed to open FPGA device");
        return 1;
    }

    while (1) {
        printf(" Checking KEY1...\n");
        sleep(1);  // Simulate polling

        printf("Detected KEY1 press. Loading song1.mid\n");
```

```
        MidiFile midi;
        if (load_midi("songs/song1.mid", &midi) == 0) {
            parse_and_send_midi(&midi, fd);
            free(midi.data);
            printf("song1.mid loaded and packets sent.\n");
        } else {
            fprintf(stderr, "Failed to load song1.mid\n");
        }

        sleep(1);
    }

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

4.12 src/utils/hw_writer.c

```
#include <stdint.h>
#include "midi_common.h"

// Pack a MidiEvent (used by midi_song_loader_hw)
uint64_t pack_midi_event(MidiEvent e) {
    uint16_t duration_ms = (e.duration_us / 1000) & 0xFFFF;

    uint64_t packet = 0;
    packet |= ((uint64_t)e.note << 56);
    packet |= ((uint64_t)e.velocity << 48);
    packet |= ((uint64_t)duration_ms << 32);
    packet |= (uint64_t)e.timestamp_us;

    return packet;
}

// Pack live keyboard input (used by midi_logger_hw)
uint64_t pack_midi_input(uint8_t status, uint8_t note, uint8_t velocity, uint64_t
    uint64_t packet = 0;
```

```
    packet |= ((uint64_t)status << 16);
    packet |= ((uint64_t)note << 8);
    packet |= (uint64_t)velocity;
    packet |= ((timestamp_us & 0xFFFFFFFF) << 24);  // use only lower 32 bits

    return packet;
}
```

4.13 src/utils/midi_parser.c

```
#include "midi_common.h"

uint16_t read16(const uint8_t *data) {
    return (data[0] << 8) | data[1];
}

uint32_t read_variable_length(const uint8_t **data_ptr) {
    uint32_t value = 0;
    const uint8_t *data = *data_ptr;
    uint8_t byte;
    do {
        byte = *data++;
        value = (value << 7) | (byte & 0x7F);
    } while (byte & 0x80);
    *data_ptr = data;
    return value;
}

int transpose_note(int note) {
    while (note < 60) note += 12;     // C5
    while (note > 83) note -= 12;     // B6
    return note;
}
```

4.14 Makefile

```
#  Compiler Config
CC = gcc
CFLAGS = -Wall -O2 -Iinclude
LIBUSB_CFLAGS = $(shell pkg-config --cflags libusb-1.0)
LIBUSB_LDFLAGS = $(shell pkg-config --libs libusb-1.0)

#  File Paths
SRC_DIR = src
UTILS = $(SRC_DIR)/utils/midi_parser.c $(SRC_DIR)/utils/hw_writer.c

BIN_DIR = build
BINARIES = $(BIN_DIR)/midi_logger \
           $(BIN_DIR)/midi_logger_hw \
           $(BIN_DIR)/midi_song_loader \
           $(BIN_DIR)/midi_song_loader_hw

KERNEL_NAME = fpga_intf
KERNEL_SRC = fpga_midi.c
KERNEL_OBJ = fpga_midi.ko

#  Default Target
all: $(BINARIES) $(KERNEL_OBJ)

#  User Binary Targets
$(BIN_DIR)/midi_logger: $(SRC_DIR)/logger/midi_logger.c $(UTILS)
 @mkdir -p $(BIN_DIR)
 $(CC) $(CFLAGS) $(LIBUSB_CFLAGS) -o $@ $^ $(LIBUSB_LDFLAGS)

$(BIN_DIR)/midi_logger_hw: $(SRC_DIR)/logger/midi_logger_hw.c $(UTILS)
 @mkdir -p $(BIN_DIR)
 $(CC) $(CFLAGS) $(LIBUSB_CFLAGS) -o $@ $^ $(LIBUSB_LDFLAGS)

$(BIN_DIR)/midi_song_loader: $(SRC_DIR)/song_loader/midi_song_loader.c $(UTILS)
 @mkdir -p $(BIN_DIR)
 $(CC) $(CFLAGS) -o $@ $^
```

```
$(BIN_DIR)/midi_song_loader_hw: $(SRC_DIR)/song_loader/midi_song_loader_hw.c $(UTI
 @mkdir -p $(BIN_DIR)
 $(CC) $(CFLAGS) -o $@ $^

#  Kernel Module Targets
obj-m := fpga_midi.o

$(KERNEL_OBJ): $(KERNEL_SRC)
 @echo " Building kernel module: $(KERNEL_OBJ)"
 $(MAKE) -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

kernel: $(KERNEL_OBJ)

load_kernel: $(KERNEL_OBJ)
 @echo "Loading kernel module..."
 insmod $(KERNEL_OBJ)
 @sleep 0.5
 @dmesg | tail -n 10

unload_kernel:
 @echo " Unloading kernel module..."
 rmmod $(KERNEL_NAME)
 @sleep 0.5
 @dmesg | tail -n 10

clean_kernel:
 $(MAKE) -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
 @rm -f *.ko *.mod.* *.o *.order *.symvers .*.cmd

#  Run Targets
run_logger: $(BIN_DIR)/midi_logger
 @echo "Running midi_logger..."
 ./$<

run_logger_hw: $(BIN_DIR)/midi_logger_hw
 @echo "Running midi_logger_hw (requires sudo)..."
 ./$<
```

```
run_song_loader: $(BIN_DIR)/midi_song_loader
 @echo "Running midi_song_loader..."
 ./$<

run_song_loader_hw: $(BIN_DIR)/midi_song_loader_hw
 @echo "Running midi_song_loader_hw (requires sudo)..."
 ./$<

#  Cleanup
clean:
 rm -rf $(BIN_DIR)

.PHONY: all clean kernel clean_kernel load_kernel unload_kernel \
 run run_logger run_logger_hw run_song_loader run_song_loader_hw
```

5 Sample Outputs

5.1 Keyboard Midi Log

```
Launchkey MIDI Logger with Timestamp & Note Parsing Starting...
MIDI interface claimed. Listening for input...
[1744831296625841 us] Raw: 0F F8 00 00
[1744831297525909 us] Raw: 0F F8 00 00
[1744831297555736 us] Raw: 09 90 60 70  Note On - Note: 0x60 (96), Velocity: 0x70
[1744831297555910 us] Raw: 0F F8 00 00
[1744831297585760 us] Raw: 0F F8 00 00
[1744831297915924 us] Raw: 09 90 5F 76  Note On - Note: 0x5F (95), Velocity: 0x76
[1744831297915924 us] Raw: 0F F8 00 00
[1744831298305728 us] Raw: 09 90 5E 79  Note On - Note: 0x5E (94), Velocity: 0x79
[1744831298305914 us] Raw: 0F F8 00 00
[1744831298665730 us] Raw: 09 90 5D 71  Note On - Note: 0x5D (93), Velocity: 0x71
[1744831298665929 us] Raw: 0F F8 00 00
[1744831299025932 us] Raw: 09 90 5C 7F  Note On - Note: 0x5C (92), Velocity: 0x7F
[1744831299025932 us] Raw: 0F F8 00 00
[1744831299055758 us] Raw: 0F F8 00 00
[1744831299355937 us] Raw: 09 90 5B 6F  Note On - Note: 0x5B (91), Velocity: 0x6F
```

```
[1744831299355937 us] Raw: 0F F8 00 00
[1744831299385732 us] Raw: 0F F8 00 00
[1744831299685925 us] Raw: 09 90 5A 77  Note On - Note: 0x5A (90), Velocity: 0x77
[1744831299686050 us] Raw: 0F F8 00 00
[1744831299715758 us] Raw: 0F F8 00 00
[1744831300015928 us] Raw: 09 90 59 6D  Note On - Note: 0x59 (89), Velocity: 0x6D
[1744831300045758 us] Raw: 0F F8 00 00
[1744831300375942 us] Raw: 09 90 58 73  Note On - Note: 0x58 (88), Velocity: 0x73
[1744831300735818 us] Raw: 09 90 57 7F  Note On - Note: 0x57 (87), Velocity: 0x7F
[1744831301065819 us] Raw: 09 90 56 71  Note On - Note: 0x56 (86), Velocity: 0x71
[1744831301095729 us] Raw: 0F F8 00 00
[1744831301125728 us] Raw: 0F F8 00 00
[1744831301125809 us] Raw: 0F F8 00 00
[1744831301155729 us] Raw: 0F F8 00 00
[1744831301395952 us] Raw: 09 90 55 7F  Note On - Note: 0x55 (85), Velocity: 0x7F
[1744831301395952 us] Raw: 0F F8 00 00
[1744831301425763 us] Raw: 0F F8 00 00
[1744831301725732 us] Raw: 09 90 54 73  Note On - Note: 0x54 (84), Velocity: 0x73
[1744831301725816 us] Raw: 0F F8 00 00
[1744831301755730 us] Raw: 0F F8 00 00
[1744831302055946 us] Raw: 09 90 53 65  Note On - Note: 0x53 (83), Velocity: 0x65
[1744831302085745 us] Raw: 0F F8 00 00
[1744831302385730 us] Raw: 09 90 52 6A  Note On - Note: 0x52 (82), Velocity: 0x6A
[1744831302385815 us] Raw: 0F F8 00 00
[1744831302415755 us] Raw: 0F F8 00 00
[1744831302715729 us] Raw: 09 90 51 65  Note On - Note: 0x51 (81), Velocity: 0x65
[1744831302715822 us] Raw: 0F F8 00 00
[1744831303045838 us] Raw: 09 90 50 7F  Note On - Note: 0x50 (80), Velocity: 0x7F
[1744831303075750 us] Raw: 0F F8 00 00
[1744831303105729 us] Raw: 0F F8 00 00
[1744831303375965 us] Raw: 09 90 4F 71  Note On - Note: 0x4F (79), Velocity: 0x71
[1744831303375965 us] Raw: 0F F8 00 00
[1744831303405757 us] Raw: 0F F8 00 00
[1744831303735843 us] Raw: 09 90 4E 79  Note On - Note: 0x4E (78), Velocity: 0x79
[1744831303735843 us] Raw: 0F F8 00 00
[1744831304005844 us] Raw: 09 90 4D 68  Note On - Note: 0x4D (77), Velocity: 0x68
[1744831304035728 us] Raw: 0F F8 00 00
[1744831304065752 us] Raw: 0F F8 00 00
```

```
[1744831304305732 us] Raw: 0F F8 00 00
[1744831304335728 us] Raw: 09 90 4C 6C  Note On - Note: 0x4C (76), Velocity: 0x6C
[1744831304335975 us] Raw: 0F F8 00 00
[1744831304335975 us] Raw: 0F F8 00 00
[1744831304635833 us] Raw: 09 90 4B 6C  Note On - Note: 0x4B (75), Velocity: 0x6C
[1744831304665729 us] Raw: 0F F8 00 00
[1744831304995941 us] Raw: 09 90 4A 69  Note On - Note: 0x4A (74), Velocity: 0x69
[1744831304996095 us] Raw: 0F F8 00 00
[1744831305025735 us] Raw: 0F F8 00 00
[1744831305445729 us] Raw: 09 90 49 63  Note On - Note: 0x49 (73), Velocity: 0x63
[1744831305445843 us] Raw: 0F F8 00 00
[1744831305805847 us] Raw: 09 90 48 5B  Note On - Note: 0x48 (72), Velocity: 0x5B
[1744831305835755 us] Raw: 0F F8 00 00
```

5.2 Device Info Dump

```
Bus 001 Device 003: ID 1235:0102 Focusrite-Novation
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               2.00
  bDeviceClass          239 Miscellaneous Device
  bDeviceSubClass         2 ?
  bDeviceProtocol         1 Interface Association
  bMaxPacketSize0        64
  idVendor           0x1235 Focusrite-Novation
  idProduct          0x0102
  bcdDevice            2.00
  iManufacturer           1 Focusrite A.E
  iProduct                2 Launchkey Mini MK3
  iSerial                 3 365E34763038
  bNumConfigurations      1
  Configuration Descriptor:
    bLength                 9
    bDescriptorType         2
```

```
wTotalLength          142
bNumInterfaces          3
bConfigurationValue     1
iConfiguration          0
bmAttributes         0x80
  (Bus Powered)
MaxPower              200mA
Interface Association:
  bLength                8
  bDescriptorType       11
  bFirstInterface        0
  bInterfaceCount        2
  bFunctionClass         1 Audio
  bFunctionSubClass      1 Control Device
  bFunctionProtocol      0
  iFunction              0
Interface Descriptor:
  bLength                9
  bDescriptorType        4
  bInterfaceNumber       0
  bAlternateSetting      0
  bNumEndpoints          0
  bInterfaceClass        1 Audio
  bInterfaceSubClass     1 Control Device
  bInterfaceProtocol     0
  iInterface             0
  AudioControl Interface Descriptor:
    bLength                9
    bDescriptorType       36
    bDescriptorSubtype     1 (HEADER)
    bcdADC              1.00
    wTotalLength           9
    bInCollection          1
    baInterfaceNr( 0)      1
Interface Descriptor:
  bLength                9
  bDescriptorType        4
  bInterfaceNumber       1
```

```
bAlternateSetting       0
bNumEndpoints           2
bInterfaceClass         1 Audio
bInterfaceSubClass      3 MIDI Streaming
bInterfaceProtocol      0
iInterface              0
MIDIStreaming Interface Descriptor:
  bLength                 7
  bDescriptorType        36
  bDescriptorSubtype      1 (HEADER)
  bcdADC               1.00
  wTotalLength           67
MIDIStreaming Interface Descriptor:
  bLength                 6
  bDescriptorType        36
  bDescriptorSubtype      2 (MIDI_IN_JACK)
  bJackType               1 Embedded
  bJackID                 1
  iJack                  11 Launchkey Mini MK3 MIDI Port
MIDIStreaming Interface Descriptor:
  bLength                 6
  bDescriptorType        36
  bDescriptorSubtype      2 (MIDI_IN_JACK)
  bJackType               1 Embedded
  bJackID                 2
  iJack                  12 Launchkey Mini MK3 DAW Port
MIDIStreaming Interface Descriptor:
  bLength                 9
  bDescriptorType        36
  bDescriptorSubtype      3 (MIDI_OUT_JACK)
  bJackType               1 Embedded
  bJackID                 3
  bNrInputPins            1
  baSourceID( 0)          1
  BaSourcePin( 0)         1
  iJack                  18 Launchkey Mini MK3 MIDI Port
MIDIStreaming Interface Descriptor:
  bLength                 9
```

```
  bDescriptorType        36
  bDescriptorSubtype      3 (MIDI_OUT_JACK)
  bJackType               1 Embedded
  bJackID                 4
  bNrInputPins            1
  baSourceID( 0)          2
  BaSourcePin( 0)         1
  iJack                  19 Launchkey Mini MK3 DAW Port
Endpoint Descriptor:
  bLength                 9
  bDescriptorType         5
  bEndpointAddress     0x01  EP 1 OUT
  bmAttributes            2
    Transfer Type            Bulk
    Synch Type               None
    Usage Type               Data
  wMaxPacketSize     0x0040  1x 64 bytes
  bInterval               0
  bRefresh                0
  bSynchAddress           0
  MIDIStreaming Endpoint Descriptor:
    bLength                 6
    bDescriptorType        37
    bDescriptorSubtype      1 (GENERAL)
    bNumEmbMIDIJack         2
    baAssocJackID( 0)       1
    baAssocJackID( 1)       2
Endpoint Descriptor:
  bLength                 9
  bDescriptorType         5
  bEndpointAddress     0x81  EP 1 IN
  bmAttributes            2
    Transfer Type            Bulk
    Synch Type               None
    Usage Type               Data
  wMaxPacketSize     0x0040  1x 64 bytes
  bInterval               0
  bRefresh                0
```

```
    bSynchAddress           0
    MIDIStreaming Endpoint Descriptor:
      bLength                 6
      bDescriptorType        37
      bDescriptorSubtype      1 (GENERAL)
      bNumEmbMIDIJack         2
      baAssocJackID( 0)       3
      baAssocJackID( 1)       4
Interface Association:
  bLength                 8
  bDescriptorType        11
  bFirstInterface         2
  bInterfaceCount         1
  bFunctionClass          8 Mass Storage
  bFunctionSubClass       6 SCSI
  bFunctionProtocol      80 Bulk-Only
  iFunction               9 Novation MSD
Interface Descriptor:
  bLength                 9
  bDescriptorType         4
  bInterfaceNumber        2
  bAlternateSetting       0
  bNumEndpoints           2
  bInterfaceClass         8 Mass Storage
  bInterfaceSubClass      6 SCSI
  bInterfaceProtocol     80 Bulk-Only
  iInterface              9 Novation MSD
  Endpoint Descriptor:
    bLength                 7
    bDescriptorType         5
    bEndpointAddress     0x82  EP 2 IN
    bmAttributes            2
      Transfer Type            Bulk
      Synch Type               None
      Usage Type               Data
    wMaxPacketSize     0x0040  1x 64 bytes
    bInterval               1
  Endpoint Descriptor:
```

```
        bLength                 7
        bDescriptorType         5
        bEndpointAddress     0x02  EP 2 OUT
        bmAttributes            2
          Transfer Type            Bulk
          Synch Type               None
          Usage Type               Data
        wMaxPacketSize     0x0040  1x 64 bytes
        bInterval               1
Device Status:     0x0000
  (Bus Powered)
```

5.3 Parsed Midi Output

```
REST     : --------  | Timestamp: 0 us | Duration: 190 ms
MidiWord: 0x451EF400  | Timestamp: 7614583 us | Note: 69 | Velocity: 30 | Duration
MidiWord: 0x431EF400  | Timestamp: 8114583 us | Note: 67 | Velocity: 30 | Duration
MidiWord: 0x421EFA00  | Timestamp: 8614583 us | Note: 66 | Velocity: 30 | Duration
MidiWord: 0x3E1EFA00  | Timestamp: 8864583 us | Note: 62 | Velocity: 30 | Duration
MidiWord: 0x3D1EFA00  | Timestamp: 9114583 us | Note: 61 | Velocity: 30 | Duration
MidiWord: 0x3E1EFA00  | Timestamp: 9364583 us | Note: 62 | Velocity: 30 | Duration
MidiWord: 0x471EF400  | Timestamp: 9614583 us | Note: 71 | Velocity: 30 | Duration
MidiWord: 0x491EFA00  | Timestamp: 10114583 us | Note: 73 | Velocity: 30 | Duratio
MidiWord: 0x4A1EFA00  | Timestamp: 10364583 us | Note: 74 | Velocity: 30 | Duratio
MidiWord: 0x4A1EE800  | Timestamp: 10614583 us | Note: 74 | Velocity: 30 | Duratio
MidiWord: 0x4C1EF400  | Timestamp: 11614583 us | Note: 76 | Velocity: 30 | Duratio
MidiWord: 0x4A1EFA00  | Timestamp: 12114583 us | Note: 74 | Velocity: 30 | Duratio
MidiWord: 0x4C1EFA00  | Timestamp: 12364583 us | Note: 76 | Velocity: 30 | Duratio
MidiWord: 0x4E1EF400  | Timestamp: 12614583 us | Note: 78 | Velocity: 30 | Duratio
MidiWord: 0x4A1EF400  | Timestamp: 13114583 us | Note: 74 | Velocity: 30 | Duratio
MidiWord: 0x4F1EE800  | Timestamp: 13614583 us | Note: 79 | Velocity: 30 | Duratio
MidiWord: 0x451EF400  | Timestamp: 14614583 us | Note: 69 | Velocity: 30 | Duratio
MidiWord: 0x4A1EF400  | Timestamp: 15114583 us | Note: 74 | Velocity: 30 | Duratio
MidiWord: 0x4C1EE800  | Timestamp: 15614583 us | Note: 76 | Velocity: 30 | Duratio
MidiWord: 0x4A1ED000  | Timestamp: 16614583 us | Note: 74 | Velocity: 30 | Duratio
MidiWord: 0x4A1EE800  | Timestamp: 18614583 us | Note: 74 | Velocity: 30 | Duratio
MidiWord: 0x491EE800  | Timestamp: 19614583 us | Note: 73 | Velocity: 30 | Duratio
```

```
MidiWord: 0x4A1ED000  | Timestamp: 20614583 us | Note: 74 | Velocity: 30 | Duratic
REST     : --------  | Timestamp: 22614583 us | Duration: 184 ms
MidiWord: 0x421EF400  | Timestamp: 25614583 us | Note: 66 | Velocity: 30 | Duratic
MidiWord: 0x451EF400  | Timestamp: 26114583 us | Note: 69 | Velocity: 30 | Duratic
MidiWord: 0x471EF400  | Timestamp: 26614583 us | Note: 71 | Velocity: 30 | Duratic
MidiWord: 0x4A1EF400  | Timestamp: 27114583 us | Note: 74 | Velocity: 30 | Duratic
MidiWord: 0x4C1EFA00  | Timestamp: 27614583 us | Note: 76 | Velocity: 30 | Duratic
MidiWord: 0x4A1EFA00  | Timestamp: 27864583 us | Note: 74 | Velocity: 30 | Duratic
MidiWord: 0x491EF400  | Timestamp: 28114583 us | Note: 73 | Velocity: 30 | Duratic
MidiWord: 0x4A1EA000  | Timestamp: 28614583 us | Note: 74 | Velocity: 30 | Duratic
REST     : --------  | Timestamp: 32614583 us | Duration: 208 ms
MidiWord: 0x451ED000  | Timestamp: 34614583 us | Note: 69 | Velocity: 30 | Duratic
MidiWord: 0x461ED000  | Timestamp: 36614583 us | Note: 70 | Velocity: 30 | Duratic
REST     : --------  | Timestamp: 38614583 us | Duration: 244 ms
MidiWord: 0x4D1EF400  | Timestamp: 39114583 us | Note: 77 | Velocity: 30 | Duratic
MidiWord: 0x4C1EF400  | Timestamp: 39614583 us | Note: 76 | Velocity: 30 | Duratic
MidiWord: 0x4A1EFA00  | Timestamp: 40114583 us | Note: 74 | Velocity: 30 | Duratic
MidiWord: 0x4C1EFA00  | Timestamp: 40364583 us | Note: 76 | Velocity: 30 | Duratic
MidiWord: 0x4D1EB800  | Timestamp: 40614583 us | Note: 77 | Velocity: 30 | Duratic
MidiWord: 0x4C1EB800  | Timestamp: 43614583 us | Note: 76 | Velocity: 30 | Duratic
REST     : --------  | Timestamp: 46614583 us | Duration: 136 ms
MidiWord: 0x4F1EE800  | Timestamp: 51614583 us | Note: 79 | Velocity: 30 | Duratic
MidiWord: 0x4C1ED000  | Timestamp: 52614583 us | Note: 76 | Velocity: 30 | Duratic
MidiWord: 0x4D1ED000  | Timestamp: 54614583 us | Note: 77 | Velocity: 30 | Duratic
MidiWord: 0x4C1ED000  | Timestamp: 56614583 us | Note: 76 | Velocity: 30 | Duratic
REST     : --------  | Timestamp: 58614583 us | Duration: 244 ms
MidiWord: 0x481EF400  | Timestamp: 59114583 us | Note: 72 | Velocity: 30 | Duratic
MidiWord: 0x4A1EF400  | Timestamp: 59614583 us | Note: 74 | Velocity: 30 | Duratic
MidiWord: 0x471EF400  | Timestamp: 60114583 us | Note: 71 | Velocity: 30 | Duratic
MidiWord: 0x481ED000  | Timestamp: 60614583 us | Note: 72 | Velocity: 30 | Duratic
MidiWord: 0x451EDC00  | Timestamp: 62614583 us | Note: 69 | Velocity: 30 | Duratic
MidiWord: 0x481EE800  | Timestamp: 64114583 us | Note: 72 | Velocity: 30 | Duratic
MidiWord: 0x431EBE00  | Timestamp: 65114583 us | Note: 67 | Velocity: 30 | Duratic
REST     : --------  | Timestamp: 67864583 us | Duration: 172 ms
MidiWord: 0x451ED000  | Timestamp: 71364583 us | Note: 69 | Velocity: 30 | Duratic
MidiWord: 0x481E7D00  | Timestamp: 73364583 us | Note: 72 | Velocity: 30 | Duratic
MidiWord: 0x4A1E7D00  | Timestamp: 73489583 us | Note: 74 | Velocity: 30 | Duratic
MidiWord: 0x4C1EEE00  | Timestamp: 73614583 us | Note: 76 | Velocity: 30 | Duratic
```

```
MidiWord: 0x451EC400  | Timestamp: 74364583 us | Note: 69 | Velocity: 30 | Duratic
REST     : --------  | Timestamp: 76864583 us | Duration: 198 ms
MidiWord: 0x491EEE00  | Timestamp: 100614583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x491EEE00  | Timestamp: 101364583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x491EEE00  | Timestamp: 102114583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x491EEE00  | Timestamp: 102864583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x481EDC00  | Timestamp: 103614583 us | Note: 72 | Velocity: 30 | Durati
MidiWord: 0x491EEE00  | Timestamp: 105114583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x491EEE00  | Timestamp: 105864583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x491EEE00  | Timestamp: 106614583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x491EEE00  | Timestamp: 107364583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x481EEE00  | Timestamp: 108114583 us | Note: 72 | Velocity: 30 | Durati
MidiWord: 0x481EFA00  | Timestamp: 108864583 us | Note: 72 | Velocity: 30 | Durati
MidiWord: 0x491EFA00  | Timestamp: 109114583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x491EFA00  | Timestamp: 109364583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x491EEE00  | Timestamp: 109614583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x491EEE00  | Timestamp: 110364583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x481EEE00  | Timestamp: 111114583 us | Note: 72 | Velocity: 30 | Durati
MidiWord: 0x481EFA00  | Timestamp: 111864583 us | Note: 72 | Velocity: 30 | Durati
MidiWord: 0x491EFA00  | Timestamp: 112114583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x461EFA00  | Timestamp: 112364583 us | Note: 70 | Velocity: 30 | Durati
MidiWord: 0x491EEE00  | Timestamp: 112614583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x491EFA00  | Timestamp: 113364583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x491EFA00  | Timestamp: 113614583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x491EFA00  | Timestamp: 113864583 us | Note: 73 | Velocity: 30 | Durati
MidiWord: 0x481EFA00  | Timestamp: 114114583 us | Note: 72 | Velocity: 30 | Durati
REST     : --------  | Timestamp: 114364583 us | Duration: 125 ms
```