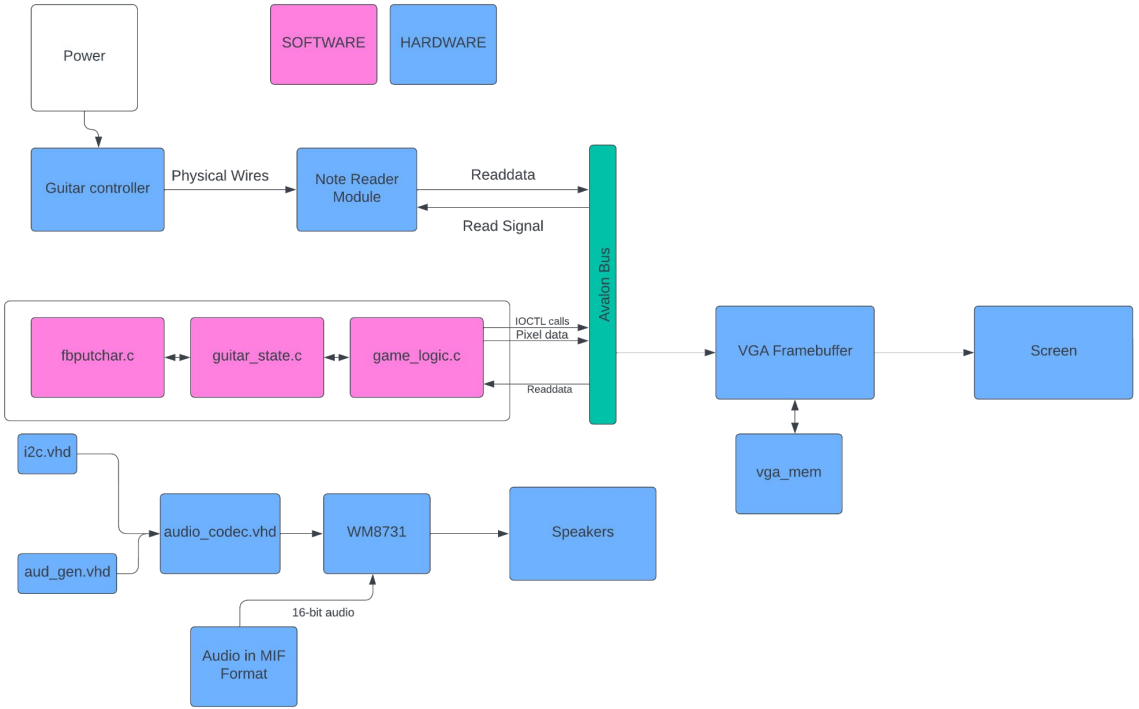
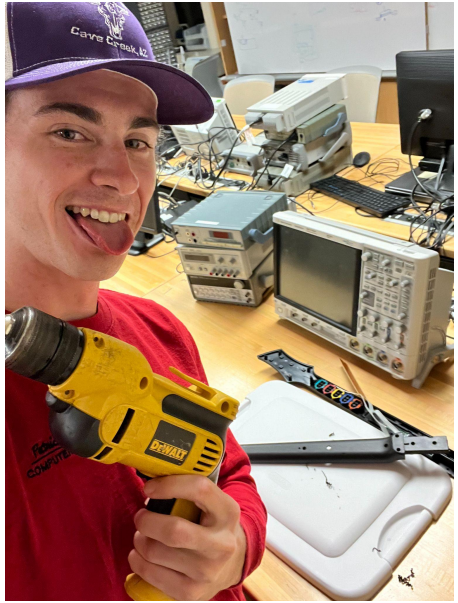

GUITAR
HERO

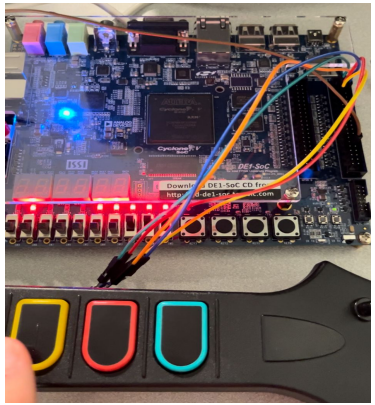
DESIGN

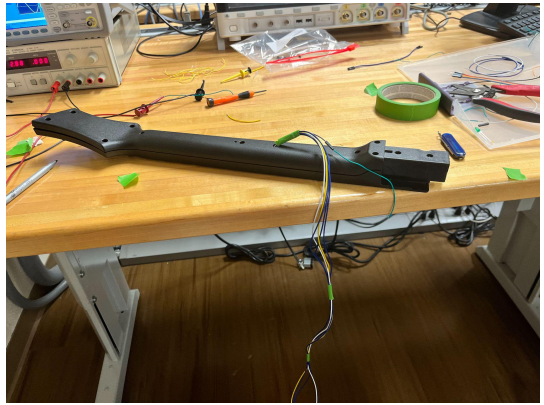

Power
SOFTWARE
HARDWARE
Guitar controller
Physical Wires
Note Reader Module
Readdata
Read Signal
Avalon Bus
fbputchar.c
guitar_state.c
game_logic.c
IOCTL calls
Pixel data
Readdata
VGA Framebuffer
Screen
vga_mem
i2c.vhd
audio_codec.vhd
WM8731
Speakers
aud_gen.vhd
16-bit audio
Audio in MIF Format

Controller

- Salvaged old controller
- Debounced then soldered into controller pcb circuit
- GPIO pinout





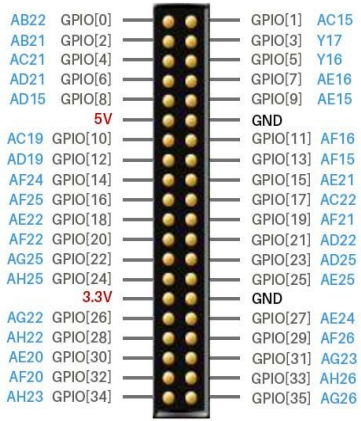


Hardware Module

Avalon Bus

8 bit readdata

Pin assignment




```
module note_reader(
    input logic clk,
    input logic reset,
    input chipselect,
    input logic [2:0] address,
    input logic [3:0] KEY,
    input logic [5:0] GPIO_1,

    output [7:0] LEDR,
    output logic [7:0] readdata,
    input logic read,
    output logic waitrequest
);
```

```
if (read && chipselect) begin
    readdata = 8'b00000000; // Initialize readdata to all zeros
    // Loop through each key
    for (int i = 0; i < 6; i = i + 1) begin
        // If the corresponding key is pressed, set the corresponding bit in readdata
        if (GPIO_1[i]) begin
            readdata = readdata | (1 << i);
        end
    end
```

Connecting hardware to software

```
/*
 * Reads the guitar state
 */
int read_guitar_state(void) { return ioread8(dev.virtbase); }

/*
```

```
char *read_note(int guitar_fd) {
  int arg;

  if (ioctl(fd: guitar_fd, request: GUITAR_READER_READ, &arg)) {
    perror(s: "ioctl(GUITAR_READER_READ) failed");
  }

  // Static buffer to hold the string (two characters + null terminator)
  static char result_string[3];

  // Convert the integer value to a two-digit hexadecimal string
  snprintf(s: result_string, maxlen: 3, format: "%02x", arg);

  return result_string;
}
```

Software - Input

Notes stored in a CSV, exported to a TXT

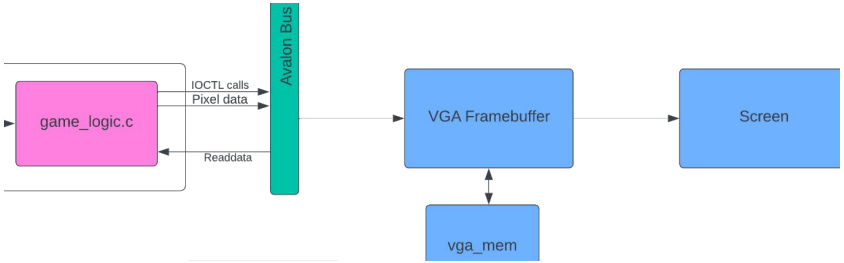
Output rows of notes based on stored note list

A thread continuously samples the guitar peripheral

```
typedef struct {
    int green;     // Re
    int red;       // Re
    int yellow;    // Re
    int blue;      // Re
    int orange;    // Re
    int strum;     // Re
    // Add any other ne
} NoteState;
```

```
00000001
00000001
00000001
00000001
00000001
00000001
00000001
00000001
00000001
00000001
00000001
00000001
00000001
00000001
00000001
00000001
00000001
00000010
```

VGA Framebuffer




```
module vga_framebuffer (
    input logic clk,
    input logic reset,
    input logic [31:0] writedata,
    input logic write,
    input chipselect,
    input logic [1:0] address,  /

    output logic [7:0] VGA_R, VGA
    output logic       VGA_CLK, V
    output logic       VGA_SYNC_n
);
```

```
module vga_mem (
    input logic clk,
    input logic [16:0] ra,
    wa,
    input logic write,
    input logic [5:0] wd,
    output logic [5:0] rd
);
```

Representing Colors

colors.c

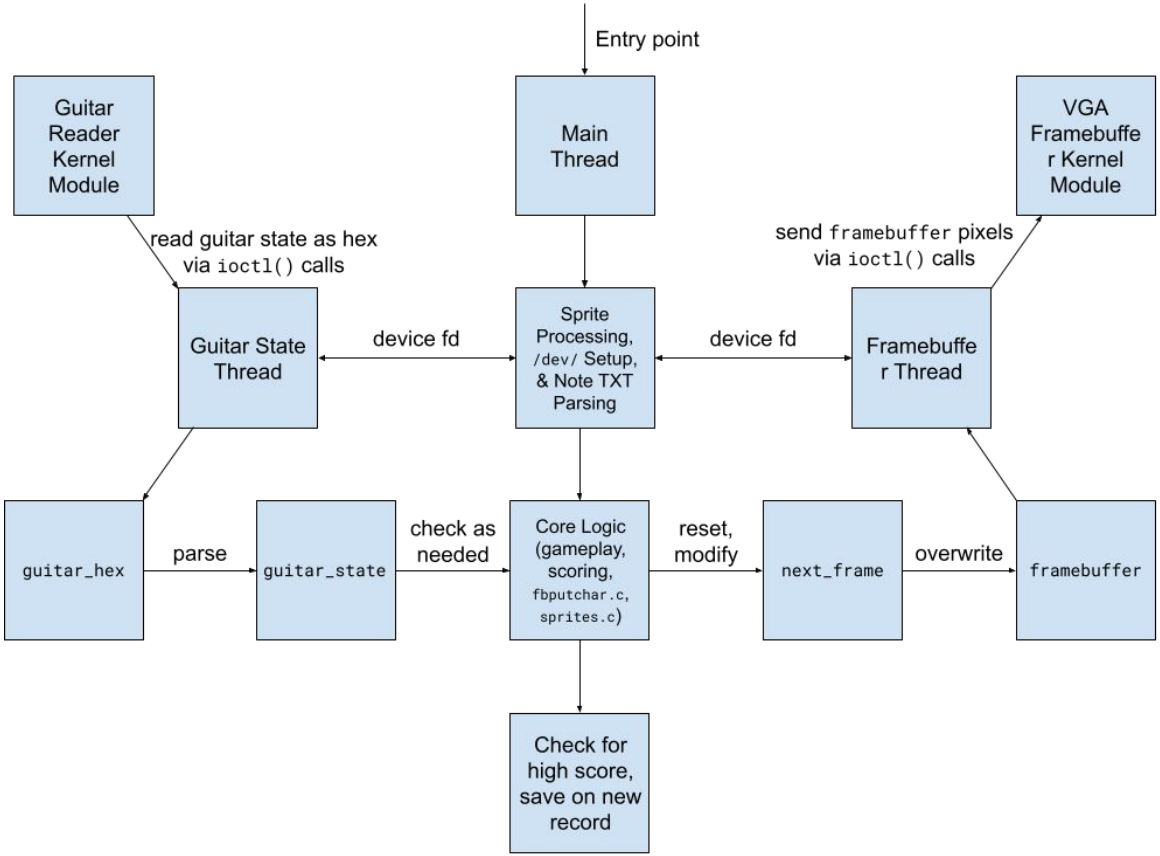
```
RGB palette[COLOR_COUNT] = {[BLACK] = {0, 0, 0},
                            [WHITE] = {255, 255, 255},
                            [RED] = {255, 0, 0},
                            [GREEN] = {0, 255, 0},
                            [BLUE] = {0, 0, 255},
                            [LIGHT_GREEN] = {20, 211, 69},
                            [MIDDLE_GREEN] = {17, 161, 50},
                            [DARK_GREEN] = {16, 162, 55},
                            [LIGHT_RED] = {211, 54, 47},
                            [MIDDLE_RED] = {154, 42, 38},
                            [DARK_RED] = {155, 41, 41},
                            [LIGHT_YELLOW] = {254, 243, 53},
                            [MIDDLE_YELLOW] = {197, 189, 26},
                            [DARK_YELLOW] = {207, 189, 61},
                            [LIGHT_BLUE] = {83, 117, 224},
                            [MIDDLE_BLUE] = {59, 89, 175},
                            [DARK_BLUE] = {64, 89, 171},
                            [LIGHT_ORANGE] = {218, 86, 43},
                            [MIDDLE_ORANGE] = {139, 53, 24},
                            [DARK_ORANGE] = {143, 55, 25}};
```

generate_verilog_colors.py

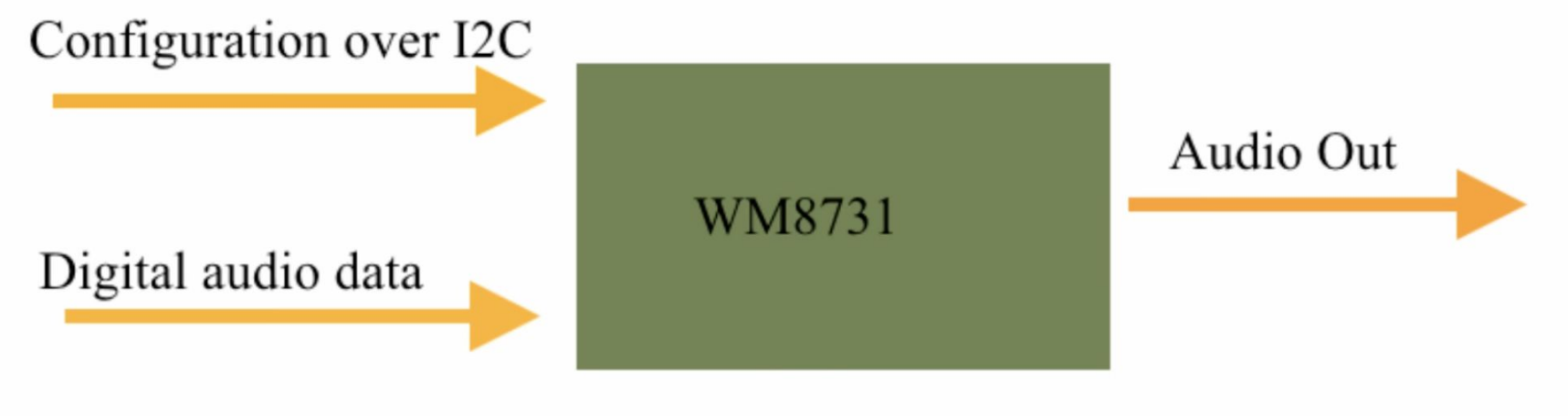
vga_framebuffer.sv

```
case (pixel_data)
  6'd0: {VGA_R, VGA_G, VGA_B} = 24'h000000;  // Black
  6'd1: {VGA_R, VGA_G, VGA_B} = 24'hffffff;  // White
  6'd2: {VGA_R, VGA_G, VGA_B} = 24'hff0000;  // Red
  6'd3: {VGA_R, VGA_G, VGA_B} = 24'h00ff00;  // Green
  6'd4: {VGA_R, VGA_G, VGA_B} = 24'h0000ff;  // Blue
  6'd5: {VGA_R, VGA_G, VGA_B} = 24'h14d345;  // Light_green
  6'd6: {VGA_R, VGA_G, VGA_B} = 24'h11a132;  // Middle_green
  6'd7: {VGA_R, VGA_G, VGA_B} = 24'h10a237;  // Dark_green
  6'd8: {VGA_R, VGA_G, VGA_B} = 24'hd3362f;  // Light_red
  6'd9: {VGA_R, VGA_G, VGA_B} = 24'h9a2a26;  // Middle_red
  6'd10: {VGA_R, VGA_G, VGA_B} = 24'h9b2929;  // Dark_red
  6'd11: {VGA_R, VGA_G, VGA_B} = 24'hfef335;  // Light_yellow
  6'd12: {VGA_R, VGA_G, VGA_B} = 24'hc5bd1a;  // Middle_yellow
  6'd13: {VGA_R, VGA_G, VGA_B} = 24'hcfbd3d;  // Dark_yellow
  6'd14: {VGA_R, VGA_G, VGA_B} = 24'h5375e0;  // Light_blue
  6'd15: {VGA_R, VGA_G, VGA_B} = 24'h3b59af;  // Middle_blue
  6'd16: {VGA_R, VGA_G, VGA_B} = 24'h4059ab;  // Dark_blue
  6'd17: {VGA_R, VGA_G, VGA_B} = 24'hda562b;  // Light_orange
  6'd18: {VGA_R, VGA_G, VGA_B} = 24'h8b3518;  // Middle_orange
  6'd19: {VGA_R, VGA_G, VGA_B} = 24'h8f3719;  // Dark_orange
  default: {VGA_R, VGA_G, VGA_B} = 24'hffffff;  // Default to white
endcase
```

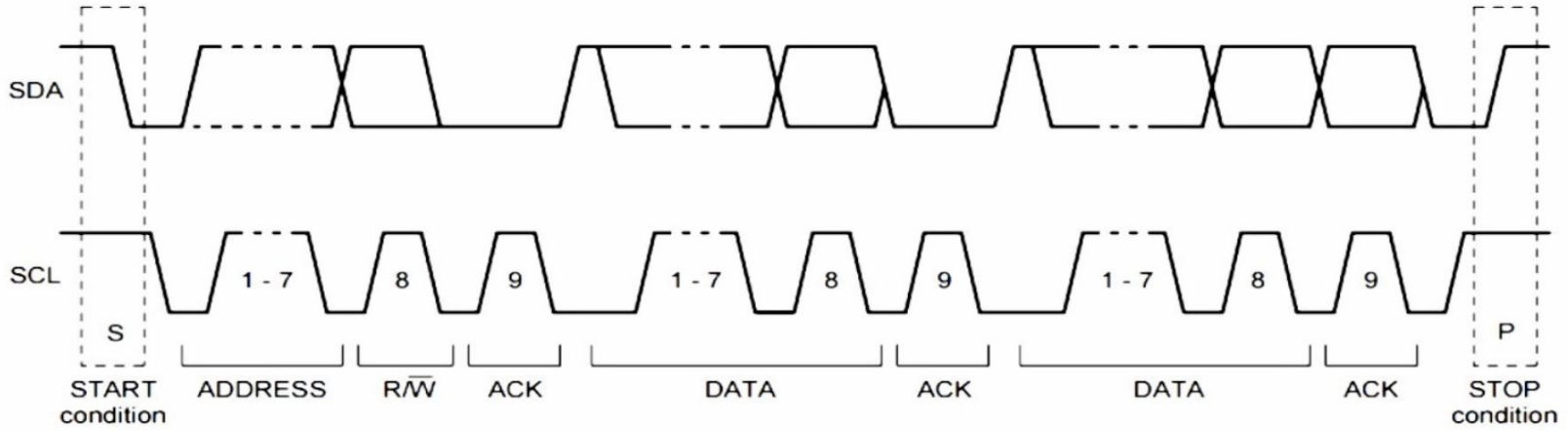
General Software Structure



Audio - CODEC Overview

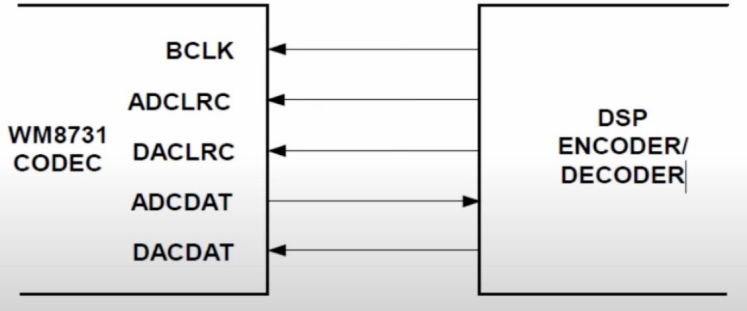


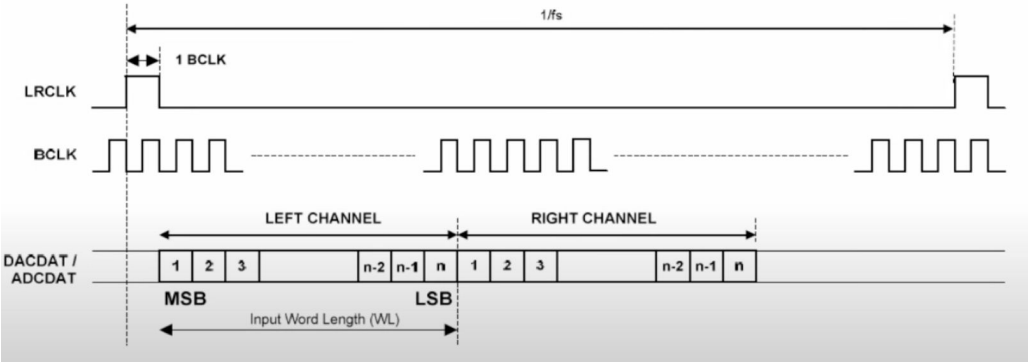
Audio - I2C


SDA
SCL
S
1 - 7
8
9
1 - 7
8
9
1 - 7
8
9
P
START condition
ADDRESS
R/W
ACK
DATA
ACK
DATA
ACK
STOP condition

Audio - CODEC Config

1. Slave Mode – the CODEC gets all the necessary signals from the FPGA
2. USB Mode – clk @ 12 Mhz, sample rate 48 kSps @ 16 Bit
3. Pulse-code modulation





Audio - Modules

Remote Development

- VGA Emulator (`vga_emulator.c`) w/ libsdl2-dev. Replaces the HW & drivers