Design Document: Audio Visualizer
CSEE 4840

Manas Pange (mmp2248)

Yaagna Modi (ykm2110)

Gaurav Agarwal (gsa2131)

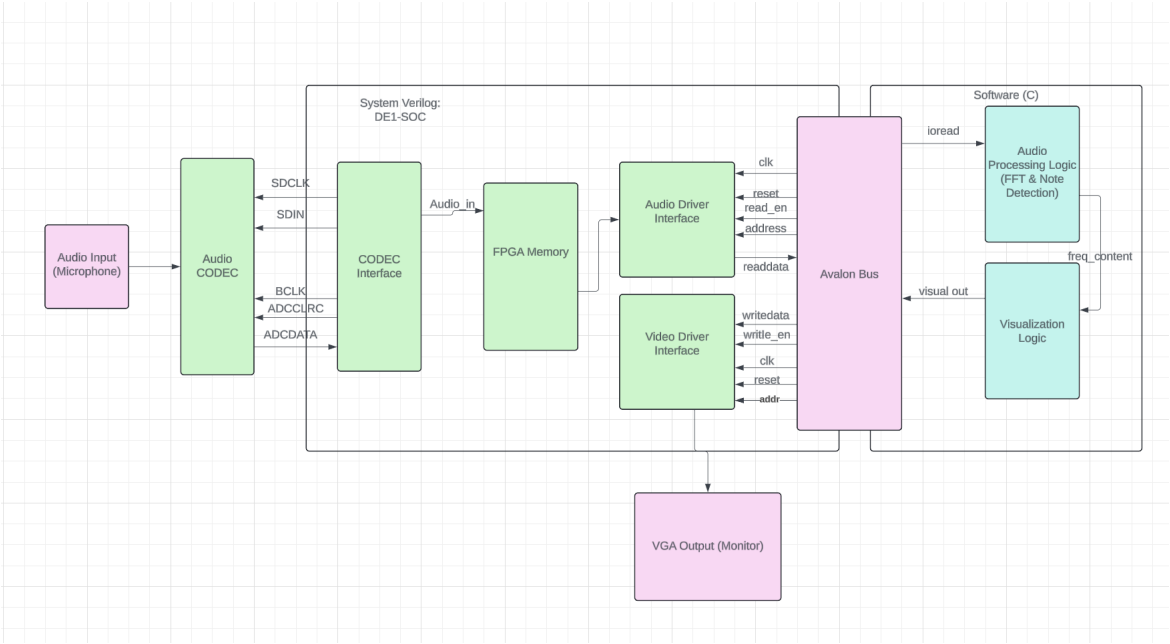
Max Lavey (mjl2274)

1. Introduction

This Design Document outlines the development of a custom System Verilog-based Audio Visualizer, utilizing Fast Fourier Transforms (FFTs) implemented on Field-Programmable Gate Arrays (FPGAs) for high-frequency sampling. The core of this venture is to create a visual representation of the frequency spectrum of audio music signals.

The heart of our system is the FFT Module, meticulously designed to sample 8192 instances and conduct FFTs on these samples, effectively translating the input analog signal into a discernible frequency spectrum. The visual output is displayed through a VGA Display, which will display sprites that correspond to the Frequency and Piano note of the sound played. This project not only aims to showcase technical prowess in FPGA programming and FFT analysis but also to enrich the audio experience with a display.

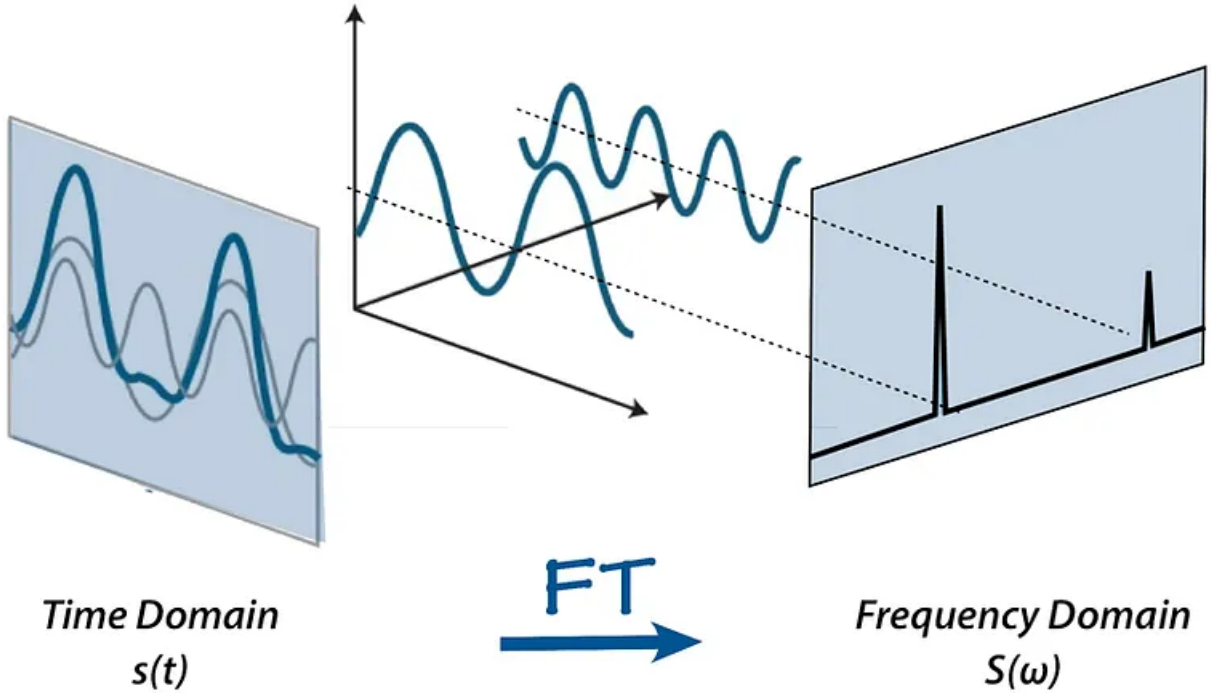
2. Block Diagram



First, we will take an audio input from an external microphone that is plugged into the DE1-SOC. This will be a Hiyanco microphone that we purchased through Amazon. This will go into an Audio CODEC to be processed and put through an ADC. The CODEC will then be interfaced so that we can filter and store it in the FPGA memory to be read out sequentially by the Audio Driver interface. This driver interface communicates through the Avalon Bus to transfer data to the Software. The software has two main components: Audio Processing Logic, and Visualization Logic. For the Audio Processing Logic, we take the ADC output and process it with FFT to analyze the key frequencies present in the audio-time sample. Based on the largest frequencies read, we will send different information to the Visualization logic, which will create specific visualizations for different frequencies. These will be packaged and sent back through the Avalon Bus. This bus will interface with another VGA driver similar to that of Lab 3, where we can create custom sprites and visualizations for the individual music components to be broadcast to the external VGA Monitor.

3. Algorithms

Fast Fourier Transform



We are implementing a Fast Fourier Transform (FFT) to process the audio input from the microphone attached to the FPGA. We will be taking our audio input and transforming it to frequency domain by computing the Discrete Fourier Transform (DFT) of the audio sample. This will allow us to break it into discrete frequencies and determine the best way to visualise the sound. This will be done in hardware which will take the input from the ADC and CODEC and place it in memory sequentially, where it will then be sent to the audio driver to be processed by the software:

1. **Input Signal:** First we sample the audio input from the microphone, which is a signal in the time domain. This will be processed by the hardware and converted to a signal in frequency domain.

2. **Divide and Conquer:** The FFT is based on a divide and conquer approach. It will recursively breakdown the Discrete Fourier transform of size N into $\frac{N}{2}$, $\frac{N}{4}$, $\frac{N}{8}$... until reaching a base case of size one, which for us will be after bits. This recursive splitting has a complexity of $O(NlogN)$, which is much better than $O(N^2)$.

3. **Factor Multiplication:** During this decomposition, FFT multiplies the values by certain roots, which are precomputed and stored (thus can be resused during the computation). This allows us to combine the smaller DFT's to compute the larger one.

4. **Combine Results:** After computing each smaller DFT, the algorithm combines them to create a complete final DFT. All of these operations are linear and can be done efficiently.

5. **Output:** The output of the FFT algorithm is a sequence of numbers representing the frequency-domain representation of the input signal. These numbers indicate the magnitude and phase of each frequency component present in the signal. These will be passed through the memory to be processed by the driver, avalon bus, and eventually the software.

   FFT Hardware Issues: Motivation for Software When we set out to begin this project we initially wanted to implement FFT in Hardware as the focus of our project. This was proven to be exceedingly difficult for a number of reasons:

   (a) **Butterfly Operations:** Completing the butterfly operations of the Cooley Tukey Algorithm, was not only challenging syntactically in System Verilog, but it also required extensive memory mapping and differential timing that we struggled with when trying to implement FFT in Hardware by hand. The trickiest part was the memory addressing for multiplying the twiddle factors.

   (b) **Fixed Point Arithmetic:** Performing fixed point FFT proved to be extremely difficult both in *C* for our preliminary testing, and in System Verilog when trying to implement in hardware. When Max implemented it in *C* there was also a loss of precision that could cause inaccurate results. This was also abundantly clear in System Verilog and one of the main motivators for moving FFT to software. We wanted to ensure we had precise FFT so we could pick out as many of the 88 keys of the piano as possible.

   (c) **Campus Shutdown:** While everyone in the class this semester experienced this barrier, it still proved to be difficult. We left most of our supplies in the lab and were forced to halt productivity for almost a week due to the protests.

   Due to the above reasons, we really struggled to complete FFT in hardware in time, which is why we shifted our focus to completing it in software and spending time creating sprites and graphics for the visualization portion of our project.

4. Resource Budgets

The FPGA has 512 KB of on chip memory, we will be using roughly 384 KB of this sequentially to store and relay the input audio data to the audio driver, avalon bus, and eventually the Software to be processed. The FFT will sample at 48 kHz, we will have one channel, and we will have 4 bytes (32 bits) per sample. This is because we will be using our 32 bit input modified input signal buffer.

By utilizing the Avalon audio interface, we have the capability to adjust the sample rate and bit depth of the data transmitted from the Audio CODEC to the FPGA. For instance, with a sample rate of 48 kHz, a bit depth of 32 bits, and a mono channel spanning 2 seconds, the calculations unfold as follows:

Total Sample Count = Sample Rate $\times$ Duration = 48,000 $\times$ 2 = 96,000 samples

Bytes per Sample = Bit Depth $\div$ 8 bytes = 32 / 8 = 4 bytes per sample

Total Memory Required = Total Sample Count $\times$ Bytes per Sample = 96,000 $\times$ 4 = 384,000 bytes = Approximately 384 KB

Given that the audio file adheres to our FPGA memory limit of 512 KB, we did not encounter memory constraints or overlook space for additional overhead.

5. Hardware/Software Interface

Hardware

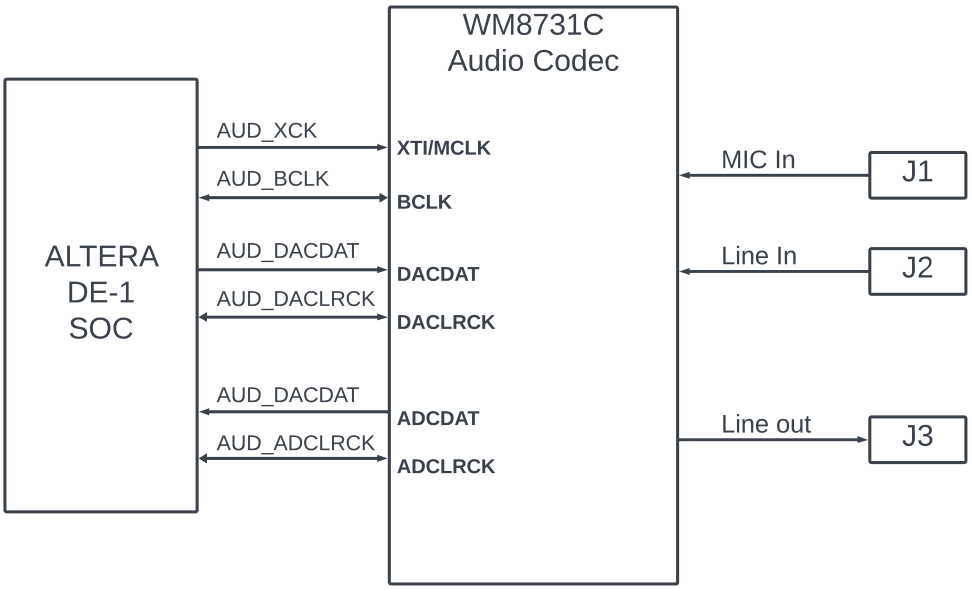
The Altera DE1 SOC features the WM8731 audio codec, known for its low power consumption and integrated headphone driver, tailored for portable MP3 audio and speech players as well as recorders. It's well-suited for applications such as MD, CD-RW machines, and DAT recorders. An external, passive microphone will be linked to this port to serve as our audio source. This CODEC possesses various configurations like:

(a) Master mode

(b) Baud rate set to 48 KHz

(c) Input from a microphone

(d) Output to a speaker



To achieve this setup, the I2C (Inter-Integrated Circuit) protocol will be utilized for programming the Audio CODEC. Analysis reveals a total of 11 distinct registers (R0 – R9, R15). Upon initial power-up, the registers reset to default values, which may not align with the desired configuration.

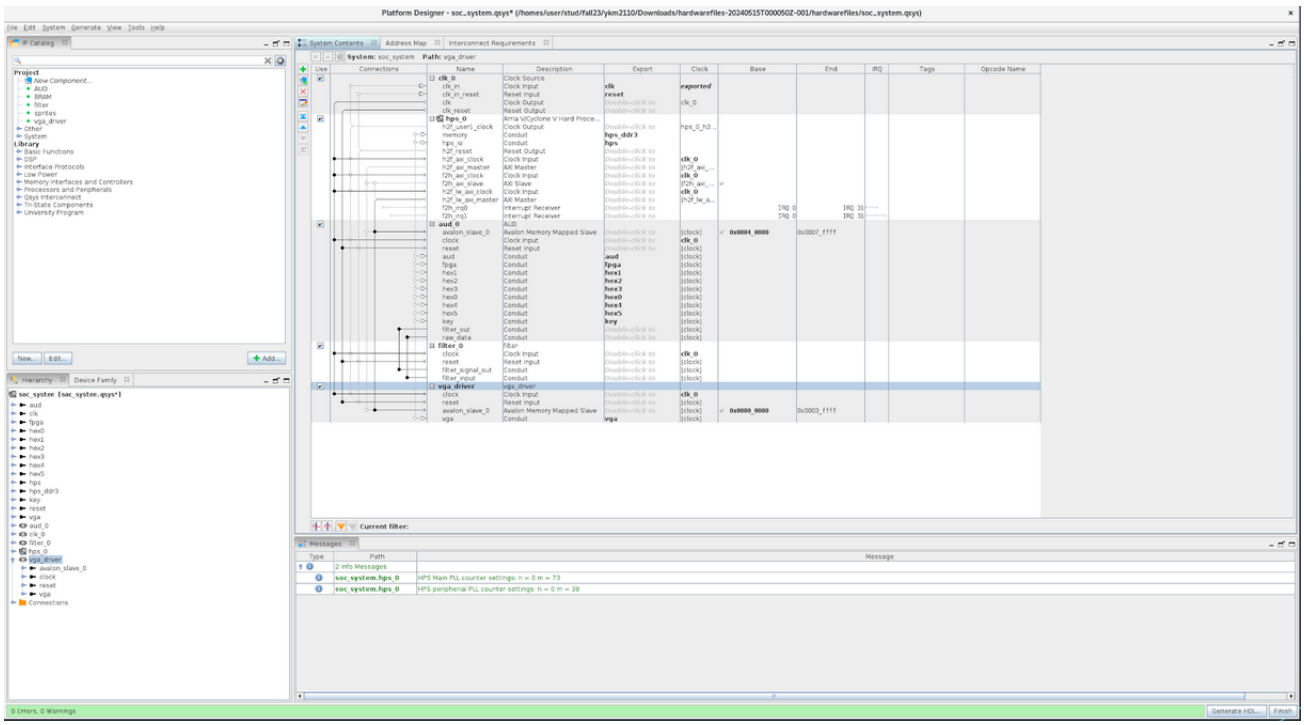
Subsequently, the creation of an I2C serial protocol interface is essential to load the values from the I2C registers into the internal registers of the audio CODEC.

REGISTER	B 15	B 14	B 13	B 12	B 11	B 10	B 9	B8	B7	B6	B5	B4	B3	B2	B1	B0
R0 (00h)	0	0	0	0	0	0	0	LRIN BOTH	LIN MUTE	0	0	LINVOL				
R1 (02h)	0	0	0	0	0	0	1	RLIN BOTH	RIN MUTE	0	0	RINVOL				
R2 (04h)	0	0	0	0	0	1	0	LRHP BOTH	LZCEN	LHPVOL						
R3 (06h)	0	0	0	0	0	1	1	RLHP BOTH	RZCEN	RHPVOL						
R4 (08h)	0	0	0	0	1	0	0	0	SIDEATT		SIDETONE	DAC SEL	BY PASS	INSEL	MUTE MIC	MIC BOOST
R5 (0Ah)	0	0	0	0	1	0	1	0	0	0	0	HPOR	DAC MU	DEEMPH		ADC HPD
R6 (0Ch)	0	0	0	0	1	1	0	0	PWR OFF	CLK OUTPD	OSCPD	OUTPD	DACPD	ADCPD	MICPD	LINEINPD
R7 (0Eh)	0	0	0	0	1	1	1	0	BCLK INV	MS	LR SWAP	LRP	IWL		FORMAT	
R8 (10h)	0	0	0	1	0	0	0	0	CLKO DIV2	CLKI DIV2	SR				BOSR	USB/ NORM
R9 (12h)	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	ACTIVE
R15(1Eh)	0	0	0	1	1	1	1	RESET								
	ADDRESS							DATA								

The objective of this code segment is to generate the required 12.288 MHz input for the AUD_XCK input on the audio CODEC chip which is the necessary input frequency for achieving the desired audio frequency of 48 KHz.

There is also a VGA driver that is enabled and houses all of the sprites produced to be able to display our identified notes and houses the ROM for each of the numbers and letters used to display the resulting note and frequency.

There will also be a Bandpass filter implemented in hardware to ensure that the frequencies received are all within a reasonable range for our instrument of choice (Piano). This filter will allow frequencies between 11Hz and 4200 Hz pass through the algorithm, ensuring we are only looking for and interested in Piano Notes.



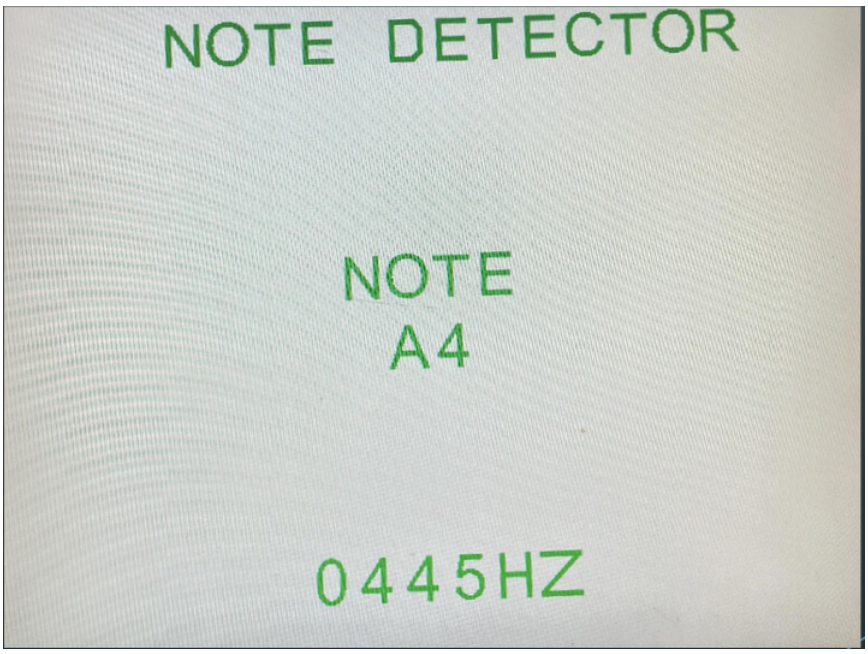
Software

The Driver Interface functions by fetching samples from the hardware through the audio ioctl. Synchronized with the ADC sampling rate of 48kHz, it triggers an irq (interrupt request) to notify the ARM Core that a sample is ready for retrieval. Subsequently, the device driver manages this interrupt by fetching data from the Driver Interface through the Avalon bus, which transmits 16 bits of data via the readdata line of the Avalon interface.

On the software side, the device driver receives data via an interrupt triggered by the driver interface over the Avalon bus, courtesy of our device driver. Each individual 16-bit sample, is transmitted across the Avalon bus. The device driver then transfers this data from kernel space to user space for further processing.

Within the C program, audio samples are accumulated and stored in a time series array. They are then put though the FFT in software with a sampling frequency of 48kHz with 8192 bins. From this FFT output array, we are able to search the array for the index of the largest magnitude. Each bin is associated with a different frequency, with a separation of $\frac{48{,}000}{8192} = 5.185$Hz. The index of the largest magnitude in the array can be paired with a bin of a specific frequency. This frequency is then plugged into a lookup table, where the corresponding note being played is understood. This particular note will then be displayed in the terminal and sent to the VGA portion of software.

The VGA portion of the software, takes the identified note and determines the associated sprite, and note frequency to be sent back to the hardware through a second VGA ioctl. This IOCTL will take the write packet from the hardware and convert it to a note and a frequency that are to be displayed on the VGA Monitor.


NOTE DETECTOR
NOTE
A4
0445HZ

6. Group Breakdown

- **Manas:** Manas worked hard to implement the graphics in hardware and worked hard to find references in other projects and online resources. He also managed the Github and spent a lot of time testing and debugging the software-hardware interface ensuring the audio data made it through the avalon bus to the kernel space in C. He also setup a lot of the testing with known audio samples from online sources to ensure we were getting the proper output from known inputs.
- **Gaurav:** Gaurav spent his time working on the IOCTLs and ensuring the data was transferred successfully across the avalon bus for both the audio data and the visualization data. He also spent a lot of time debugging the visualization logic and fixing up transfer of data across to the avalon bus to display the correct sprites. Lastly, he wrote the main testing function in C to run our project and correctly detect a piano note frequency.
- **Yaagna:** Yaagna worked on the hardware implementation for the ADC CODEC. He spent most of his time working with the FPGA Memory ensuring that we had allocated enough space for the audio data and the sprites. He spent a lot of his time working with Quartus and ensuring the registers were set up properly so we could communicate with the Avalon Bus. He also worked on a Matlab simulation to directly verify the CODEC data was indeed the audio sample being played to our Microphone.
- **Max:** Max wrote the C code for the note detection algorithm and also implemented FFT in C to ensure our group obtained the same results from the same audio. He completed preliminary testing on .wav files to prepare the software for the hardware when it was ready. He also studied the FFT to aid the group in finding the right sampling frequency and number of samples for processing the digital audio data from the CODEC. Lastly, he also created some of the sprites for the VGA display portion of the project in Hardware.

7. Milestones

(a) Preliminary Research: Finalize project scope and objectives. Conduct a thorough literature review on FFTs, FPGA programming, and System Verilog. Acquire necessary hardware components (FPGA, ADC, Microphone, VGA Display).

(b) Design Phase: Develop a detailed design document outlining the architecture of the Audio Visualizer, including the integration of the FFT Module, ADC, and VGA Display. Create a simulation model for the FFT algorithm to ensure accuracy in frequency spectrum analysis.

(c) Development of FFT Module: Implement the FFT algorithm in System Verilog to process analog signals. Test the FFT Module with simulated input signals to validate its functionality.

(d) Integration of ADC with FPGA: Configure the ADC to mix left and right audio channels and feed into the FPGA. Implement and test the signal acquisition process to ensure accurate capture of audio signals.

(e) VGA Display Interface Development: Design and implement the VGA display logic in C for visual output. Also develop the display drivers in system verilog so that we may test our visualizations Develop and test five distinct display modes for visualizing the frequency spectrum.

(f) Testing and Performance Enhancement: Allow time to test each individual component as well as all of them working together to ensure we meet the project deadline. We also will be spending time improving performance until it meets our satisfaction. Once we ensure that our project is working, we can move on to finish the project

(g) Project Closure and Presentation: Prepare a final presentation summarizing the project development process, challenges encountered, solutions implemented, and demonstrations of the Audio Visualizer in action. Reflect on project outcomes, lessons learned, and potential future developments.

8. File Contents

- This is the C code used to initialize and test the entire project.

Listing 1: test2.cpp

```
#define SHOW_SPRITES
#include <stdio.h>
#include "interfaces.h"
#include "vga_audio.h"
#include "aud.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <vector>
#include <math.h>
#define SAMPLING_RATE 30000
#define X_MAX 639
#define Y_MAX 479
#define N 8192
int vga_audio_fd;
int aud_fd;
float frequency;
int done = 0;
typedef struct {
    double real;
    double imag;
} Complex;


const float piano_notes[] = {
        27.50, 30.87, 32.70, 36.71, 41.20, 43.65, 48.99,
        55.00, 61.74, 65.41, 73.42, 82.40, 87.31, 97.99,
        110.00, 123.47, 130.81, 146.83, 164.81, 174.61, 195.99,
        220.00, 246.94, 261.63, 293.66, 329.62, 349.23, 391.99,
        440.00, 493.88, 523.25, 587.33, 659.25, 698.46, 783.99,
        880.00, 987.77, 1046.50, 1174.66, 1318.51, 1396.91, 1567.98,
        1760.00, 1975.53, 2093.00, 2349.32, 2637.02, 2793.83, 3135.96,
        3520.00, 3951.07, 4186.01
    };

const char* note_names[] = {
        "A0", "B0", "C1", "D1", "E1", "F1", "G1",
        "A1", "B1", "C2", "D2", "E2", "F2", "G2",
        "A2", "B2", "C3", "D3", "E3", "F3", "G3",
        "A3", "B3", "C4", "D4", "E4", "F4", "G4",
        "A4", "B4", "C5", "D5", "E5", "F5", "G5",
```

```
        "A5", "B5", "C6", "D6", "E6", "F6", "G6",
        "A6", "B6", "C7", "D7", "E7", "F7", "G7",
        "A7", "B7", "C8"
    };

const float midpoints[] = {
    29.185, 31.785, 34.705, 37.80, 41.27, 44.95, 49.08, 53.455, 58.37,
        63.575,
    69.415, 75.60, 82.545, 89.905, 98.165, 106.915, 116.735, 127.14,
       138.82,
    151.195, 165.085, 179.805, 196.325, 213.825, 233.47, 254.285,
       277.645,
    302.395, 330.18, 359.61, 392.645, 427.65, 466.94, 508.565, 555.29,
        604.79,
    660.355, 741.225, 807.30, 855.305, 933.885, 1017.135, 1110.58,
       1209.585,
    1320.71, 1482.445, 1614.60, 1710.61, 1867.765, 2034.265, 2221.16,
       2419.17,
    2641.425, 2964.895, 3229.20, 3421.22, 3735.535, 4068.54
};

// Function to find index of the nearest piano note to a given
   frequency
int find_nearest_note_index(float frequency) {
    int index = 0;
    float min_diff = fabs(frequency - piano_notes[0]);
    for (int i = 1; i < sizeof(piano_notes) / sizeof(piano_notes[0]);
       i++) {
        float diff = fabs(frequency - piano_notes[i]);
        if (diff < min_diff) {
            min_diff = diff;
            index = i;
        }
    }
    return index;
}

// Function to perform FFT
void FFT(Complex* input, int n) {
    if (n <= 1)
        return;

    // Divide
    Complex even[n/ 2];
    Complex odd[n/ 2];

    for (int i = 0; i < n / 2; ++i) {
        even[i] = input[2 * i];
        odd[i] = input[2 * i + 1];
    }

    // Conquer
    FFT(even, n/ 2);
    FFT(odd, n/ 2);
```

```
    // Combine
    for (int k = 0; k < n / 2; ++k) {
        double theta = -2 * M_PI * k / n;
        Complex twiddle = {cos(theta), sin(theta)};
        Complex temp = {0}; // Temporary variable for calculation

        // Apply twiddle factor to odd part
        temp.real = odd[k].real * twiddle.real - odd[k].imag * twiddle
            .imag;
        temp.imag = odd[k].real * twiddle.imag + odd[k].imag * twiddle
            .real;

        // Combine even and odd parts
        input[k].real = even[k].real + temp.real;
        input[k].imag = even[k].imag + temp.imag;
        input[k + n / 2].real = even[k].real - temp.real;
        input[k + n / 2].imag = even[k].imag - temp.imag;
    }
}


void updateBall(sprite *obj) {
        obj->x += obj->dx;
        obj->y += obj->dy;
        if (obj->x < 0 || obj->x >= X_MAX) {
                obj->y = 380;
                obj->x = 700;
                obj->id = 0;
                obj->dy = 0;
                obj->dx = 0;
    }
        if (obj->y < 0 || obj->y > Y_MAX) {
                // obj->dy = -obj->dy;
                obj->id = 0;
                obj->dy = 0;
        }
        // if () {
        //      obj->dy = -obj->dy;
        //      obj->id = 0;
        // }
}

void scorecombosetup(sprite *sprites) {
        //index 0 acts strangly
        //'SCORE'
        sprites[1].id = 19; //N
        sprites[2].id = 20; //O
        sprites[3].id = 21; //T
        sprites[4].id = 15; //E
        for (int i = 1; i < 5; i++) {
                sprites[i].x = 250+32*(i-1);
                sprites[i].y = 190;
                sprites[i].dx = 0;
```

```
                sprites[i].dy = 0;
                sprites[i].hit = 1;
                sprites[i].index = i;
        }
        sprites[5].id = 11;
        sprites[6].id = 10;
        for (int i = 5; i < 7; i++) {
                sprites[i].x = 250+32+32*(i-5);
                sprites[i].y = 240;
                sprites[i].dx = 0;
                sprites[i].dy = 0;
                sprites[i].hit = 1;
                sprites[i].index = i;
        }

        sprites[7].id = 10;
        sprites[8].id = 10;
        sprites[9].id = 10;
        sprites[10].id = 10;
        sprites[11].id = 22;     //H
        sprites[12].id = 23;     //Z
        for (int i = 7; i < 13; i++) {
                sprites[i].x = 255+32+32*(i-9);
                sprites[i].y = 400;
                sprites[i].dx = 0;
                sprites[i].dy = 0;
                sprites[i].hit = 1;
                sprites[i].index = i;
        }




        sprites[13].id = 19; //N
        sprites[14].id = 20; //O
        sprites[15].id = 21; //T
        sprites[16].id = 15; //E

        sprites[18].id = 14; //D
        sprites[19].id = 15; //E
        sprites[20].id = 21; //T
        sprites[21].id = 15; //E
        sprites[22].id = 13; //C
        sprites[23].id = 21; //T
        sprites[24].id = 20; //O
        sprites[25].id = 28; //R

        for (int i = 13; i < 26; i++) {
                sprites[i].x = 32*(i-9);
                sprites[i].y = 20;
                sprites[i].dx = 0;
                sprites[i].dy = 0;
                sprites[i].hit = 1;
                sprites[i].index = i;
```

```
        }
}

void update_combo(sprite *sprites, const int combo) {
        int thous = (int)combo/1000;
        int huds = (int)(combo - thous*1000)/100;
        int tens = (int)(combo - huds*100- thous*1000)/10;
        int ones = combo - huds*100 - tens*10 - thous*1000;
        if (thous == 0) thous = 10;
        if (huds == 0) huds = 10;
        if (tens == 0) tens = 10;
        if (ones == 0) ones = 10;
        sprites[7].id = thous;
        sprites[8].id = huds; //100s
        sprites[9].id = tens; //10s
        sprites[10].id = ones; //1s
        return;
}

void update_score(sprite *sprites, const int score) {
        int note = score %7;
        int num =  score/7;
        sprites[5].id = note+11; //note
        if(num == 0){
                sprites[6].id = 10;
        }else{
                sprites[6].id = num;
        }
        // int huds = (int)score/100;
        // int tens = (int)(score - huds*100)/10;
        // int ones = score - huds*100 - tens*10;
        // if (huds == 0) huds = 10;
        // if (tens == 0) tens = 10;
        // if (ones == 0) ones = 10;
        // sprites[6].id = huds; //100s
        // sprites[7].id = tens; //10s
        // sprites[8].id = ones; //1s
        return;
}

void update_max(sprite *sprites, const int max) {
        int huds = (int)max/100;
        int tens = (int)(max - huds*100)/10;
        int ones = max - huds*100 - tens*10;
        if (huds == 0) huds = 10;
        if (tens == 0) tens = 10;
        if (ones == 0) ones = 10;
        sprites[20].id = huds; //100s
        sprites[21].id = tens; //10s
        sprites[22].id = ones; //1s
        return;
}
// dedicate all sprites below
```

```
// spawns a block with sprite.id depending on note
void spawnnote(sprite* sprites, int note) {
        //scans sprite array for empty sprite
        if (note == 0) return;
        int i, j;
        for (i = 23; i < SIZE; i++) {
            if (sprites[i].id == 0) break;
        }
        for (j = i+1; j < SIZE; j++) {
            if (sprites[j].id == 0) break;
        }
        // change sprite information to match note
        sprites[i].x = 28 + 120*(note-1);
        sprites[i].y = 0;
        sprites[i].dx = 0;
        sprites[i].dy = 1;
        sprites[i].id = note*2 + 16;
        sprites[i].hit = 0;
        sprites[i].index = i;
        sprites[j].x = 60 + 120*(note-1);
        sprites[j].y = 0;
        sprites[j].dx = 0;
        sprites[j].dy = 1;
        sprites[j].id = note*2 + 17;
        sprites[j].hit = 0;
        sprites[j].index = j;
        return;
}


// return sprite id in region, -1 if none
int check_valid_region(sprite* sprites, int start) {
    int i;
    //int cf = *combo_flag;
    for (i = start; i < SIZE; i++) {
            if ((sprites[i].y > 330) && (sprites[i].y < 410) && (
               sprites[i].id != 0) && (sprites[i].hit == 0)) {
                //if (sprites[i].y == 480) cf = 0;
                return i;
            }
        }
        return -1;
}

void screen_refresh(sprite* sprites) {
    for (int i = 1; i < SIZE; i++) {
                sprites[i].x = 630;
                sprites[i].y = 470;
                sprites[i].dx = 0;
                sprites[i].dy = 0;
                sprites[i].id = 0;
                sprites[i].hit = 1;
                sprites[i].index = i;
        }
```

```
        return;
}
void addToBuffer(std::vector<int>& buffer, const int data) {

        // Check if buffer is not empty and the last element is equal
           to the new data
    if ((!buffer.empty()) && (buffer.back() == data)) {
        // Data is same as the last data entered, don't add it
                // std::cout << "Buffer empty: " <<buffer.empty() <<
                   std::endl;
                // std::cout << "Buffer: " <<buffer.back() << std::
                   endl;
                // std::cout << "Data: " <<data << std::endl;
        //std::cout << "Data is same as the last data entered, not
           added.\n" << std::endl;
        return;
    }
        buffer.push_back(data);
   // Check if buffer size exceeds the maximum size
    if (buffer.size() > N) {
        // Remove the least recent data from the beginning
        buffer.clear();
                buffer.push_back(data);

    }

}
// simple game of hitting random falling notes when they reach the
   green zone
int main()
{

        vga_audio_arg_t vzat;
        std::vector<int> buffer;
        aud_arg_t aat;
        aud_mem_t amt;

        srand(time(NULL));
        float fft_output[N];
        static const char filename1[] = "/dev/vga_audio";
        static const char filename2[] = "/dev/aud";

        printf("VGA␣audio␣test␣Userspace␣program␣started\n");
        printf("%d\n", sizeof(int));
        printf("%d\n", sizeof(short));

        if ((vga_audio_fd = open(filename1, O_RDWR)) == -1) {
                fprintf(stderr, "could␣not␣open␣%s\n", filename1);
                return -1;
        }
        if ((aud_fd = open(filename2, O_RDWR)) == -1) {
                fprintf(stderr, "could␣not␣open␣%s\n", filename2);
                return -1;
        }
```

```
        //FILE *fp = fopen("test.txt", "w");
        //if (fp == NULL)       return -1;

        sprite *sprites = NULL;
        // sprites = calloc(SIZE, sizeof(*sprites));

        sprites = (sprite*)calloc(SIZE,sizeof(*sprites));

        screen_refresh(sprites);
        scorecombosetup(sprites);
        Complex signal[N];

        int score = 0;
        int combo = 0;
        //packet of sprite data to send
        vga_audio_data_t vzdt;

        int combo_flag = 1;
        int counter = 0;
        int gamecounter = 0;
    int validleft, validright;
    int max = 0;
    int hitcount = 0;
        int noteCount = 0;
        int MAX_NOTE_COUNT = 100;
        FILE *fptr;
        fptr = fopen("sound.txt", "w");
        if (fptr == NULL) {
                        printf("Unable to open file.\n");
                        return 1;
        }

        while (1) {

                amt.data = get_aud_data(aud_fd);

                //printf("AUD DATA: %d\n", amt.data);
                done = 0;
                addToBuffer(buffer, amt.data);
                int score = 0;
                if(buffer.size() == N){
                        //printf("buffer: %d\n", buffer[0]);
                        for (int i = 0; i < N; i++){
                                signal[i].real = buffer[i];
                        signal[i].imag = 0;
                        }
                        FFT(signal, N);
                        for (int i = 0; i < N; i++) {
                                fft_output[i] = sqrt(signal[i].real *
                                   signal[i].real + signal[i].imag *
                                   signal[i].imag);
                        }

                        int max_value = 0;
```

```
                        int peak_index = 0;
                        for (int i = 0; i < 700; i++) {
                                if (fft_output[i] > max_value) {
                                        max_value = fft_output[i];
                                        peak_index = i;
                                }
                        }
                        //printf("Max Value: %d\n", max_value);
                        float sample_rate = 48000; // Sample rate,
                            change accordingly if your sample rate is
                            different
                        frequency = (float)peak_index * sample_rate /
                            N;
                        score = find_nearest_note_index(frequency);
                        // Find the index of the nearest piano note
                        //int note_index = find_nearest_note_index(
                            frequency);
                        // printf("The note played is: %.2f Hz, which
                            is approximately %dth note on a piano.\n",
                            frequency);
                        printf("%.2f Hz: \tNote: %d\n", frequency,
                            score);
                        update_combo(sprites, frequency);
                        update_score(sprites, score);
                        for (int i = 0; i < SIZE; i++) {
                                vzdt.data[i] = (sprites[i].index<<26)
                                    + (sprites[i].id<<20) + (sprites[i
                                    ].y<<10) + (sprites[i].x<<0);
                        }
                        send_sprite_positions(&vzdt, vga_audio_fd);

                }

        }

        return 0;
}
```

- This is the SystemVerilog code for interfacing Wolfson Codec with the DE1-SoC board

Listing 2: aud.sv

```
`include "global_variables.sv"
`include "./AudioCodecDrivers/audio_driver.sv"


module audio_control(

        // 7-segment LED displays; HEX0 is rightmost
        output logic [6:0]          HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,

        //Audio pin assignments
        //Used because Professor Scott Hauck and Kyle Gagner
        output logic                FPGA_I2C_SCLK,
        inout                       FPGA_I2C_SDAT,
        output logic                AUD_XCK,
        input logic                 AUD_ADCLRCK,
        input logic                 AUD_DACLRCK,
        input logic                 AUD_BCLK,
        input logic                 AUD_ADCDAT,
        output logic                AUD_DACDAT,

        //Driver IO ports
        input logic                 clk,
        input logic                 reset,
        input logic [31:0]          writedata,
        input logic                 write,
        input logic                 read,
        input                       chipselect,
        input logic [15:0]          address,
        output logic [31:0]         readdata,



        //filter control
        output logic [15:0]         sink_real,
        input  logic [15:0]         source_real,
        input logic [15 : 0]        source_imag
        //button
        input logic [3:0]   KEY

        );
    //Audio Controller
    reg [15:0]      dac_left_in;
    reg [15:0]      dac_right_in;
    wire [15:0]     adc_left_out;
    wire [15:0]     adc_right_out;


    // wire advance;
```

```
    //Device drivers from Altera modified by Professor Scott Hauck and
        Kyle Gagner in Verilog
    audio_driver aDriver(
        .CLOCK_50(clk),
        .reset(reset),
        .dac_left(dac_left_in),
        .dac_right(dac_right_in),
        .adc_left(adc_left_out),
        .adc_right(adc_right_out),
        .advance(advance),
        .FPGA_I2C_SCLK(FPGA_I2C_SCLK),
        .FPGA_I2C_SDAT(FPGA_I2C_SDAT),
        .AUD_XCK(AUD_XCK),
        .AUD_DACLRCK(AUD_DACLRCK),
        .AUD_ADCLRCK(AUD_ADCLRCK),
        .AUD_BCLK(AUD_BCLK),
        .AUD_ADCDAT(AUD_ADCDAT),
        .AUD_DACDAT(AUD_DACDAT)
        );


  // Debounce variables
wire [15:0] buffer;
    always_comb begin
       // audioInMono = (adc_right_out>>1) + (adc_left_out>>1);


       buffer=adc_right_out;


    end

    //Determine when the driver is in the middle of pulling a sample
    //by default dont use the BRAM module

    logic [31:0]    driverReading = 31'd0;




    always_ff @(posedge clk) begin : IOcalls
       if (advance)begin
        raw_data <=buffer;
       end

        if (chipselect && read) begin
            case (address)
                16'h0000 : begin
                        if (raw_data[15] == 0) begin
                                readdata[31:0] <=  32'b0 | raw_data;
                        end
                        else begin
                        readdata[31:0] <=  {16'b1111111111111111,
```

```
                            raw_data};
                        end
                end
            endcase

        end

    end

    wire sampleBeingTaken;
    assign sampleBeingTaken = driverReading[0];

    //Map timer(Sample) counter output
    parameter readOutSize = 16'hffff;
    //Sample inputs/Audio passthrough

endmodule
```

- This is the Verilog top level module which instantiates all the necessary submodules like Audio, VGA, etc.

Listing 3: soc_system_top.sv

```
// ====================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ====================================================================
//
// Modified 2019 by Stephen A. Edwards
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use
//   in synthesis for all Terasic Development Boards and Altera
//   Development Kits made by Terasic.  Other use of this code,
//   including the selling ,duplication, or modification of any
//   portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design
//   reference which illustrates how these types of functions can be
//   implemented.  It is the user's responsibility to verify their
//   design for consistency and functionality through the use of
//   formal verification methods.  Terasic provides no warranty
//   regarding the use or functionality of this code.
//
// ====================================================================
//
//  Terasic Technologies Inc

//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City,
   30070. Taiwan
//
//
//                     web: http://www.terasic.com/
//                     email: support@terasic.com
module soc_system_top(

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

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

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

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

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

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

//////// DRAM ////////
output [12:0] DRAM_ADDR,
output [1:0]  DRAM_BA,
output        DRAM_CAS_N,
output        DRAM_CKE,
output        DRAM_CLK,
output        DRAM_CS_N,
inout [15:0]  DRAM_DQ,
output        DRAM_LDQM,
output        DRAM_RAS_N,
output        DRAM_UDQM,
output        DRAM_WE_N,

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

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

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

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

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

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

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

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

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

```
//////// HPS ////////
inout          HPS_CONV_USB_N,
output [14:0]  HPS_DDR3_ADDR,
output [2:0]   HPS_DDR3_BA,
output         HPS_DDR3_CAS_N,
output         HPS_DDR3_CKE,
output         HPS_DDR3_CK_N,
output         HPS_DDR3_CK_P,
output         HPS_DDR3_CS_N,
output [3:0]   HPS_DDR3_DM,
inout [31:0]   HPS_DDR3_DQ,
inout [3:0]    HPS_DDR3_DQS_N,
inout [3:0]    HPS_DDR3_DQS_P,
output         HPS_DDR3_ODT,
output         HPS_DDR3_RAS_N,
output         HPS_DDR3_RESET_N,
input          HPS_DDR3_RZQ,
output         HPS_DDR3_WE_N,
output         HPS_ENET_GTX_CLK,
inout          HPS_ENET_INT_N,
output         HPS_ENET_MDC,
inout          HPS_ENET_MDIO,
input          HPS_ENET_RX_CLK,
input [3:0]    HPS_ENET_RX_DATA,
input          HPS_ENET_RX_DV,
output [3:0]   HPS_ENET_TX_DATA,
output         HPS_ENET_TX_EN,
inout          HPS_GSENSOR_INT,
inout          HPS_I2C1_SCLK,
inout          HPS_I2C1_SDAT,
inout          HPS_I2C2_SCLK,
inout          HPS_I2C2_SDAT,
inout          HPS_I2C_CONTROL,
inout          HPS_KEY,
inout          HPS_LED,
inout          HPS_LTC_GPIO,
output         HPS_SD_CLK,
inout          HPS_SD_CMD,
inout [3:0]    HPS_SD_DATA,
output         HPS_SPIM_CLK,
input          HPS_SPIM_MISO,
output         HPS_SPIM_MOSI,
inout          HPS_SPIM_SS,
input          HPS_UART_RX,
output         HPS_UART_TX,
input          HPS_USB_CLKOUT,
inout [7:0]    HPS_USB_DATA,
input          HPS_USB_DIR,
input          HPS_USB_NXT,
output         HPS_USB_STP,

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

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

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

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

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

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


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

  soc_system soc_system0(
    .clk_clk                       ( CLOCK_50 ),
    .reset_reset_n                 ( 1'b1 ),

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

```
    .hps_ddr3_oct_rzqin           ( HPS_DDR3_RZQ ),

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

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

    .hps_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0]     ),
    .hps_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1]     ),
    .hps_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2]     ),
    .hps_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3]     ),
    .hps_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4]     ),
    .hps_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5]     ),
    .hps_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6]     ),
    .hps_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7]     ),
    .hps_hps_io_usb1_inst_CLK     ( HPS_USB_CLKOUT      ),
    .hps_hps_io_usb1_inst_STP     ( HPS_USB_STP         ),
    .hps_hps_io_usb1_inst_DIR     ( HPS_USB_DIR         ),
    .hps_hps_io_usb1_inst_NXT     ( HPS_USB_NXT         ),

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

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

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

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

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

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

    .key_key(KEY),
    .hex0_hex0(HEX0),
    .hex1_hex1(HEX1),
    .hex2_hex2(HEX2),
    .hex3_hex3(HEX3),
    .hex4_hex4(HEX4),
    .hex5_hex5(HEX5),

    .fpga_i2c_sclk(FPGA_I2C_SCLK),
    .fpga_i2c_sdat(FPGA_I2C_SDAT),

    .aud_xck(AUD_XCK),
    .aud_daclrck(AUD_DACLRCK),
    .aud_adclrck(AUD_ADCLRCK),
    .aud_bclk(AUD_BCLK),
    .aud_adcdat(AUD_ADCDAT),
    .aud_dacdat(AUD_DACDAT),

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

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

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

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

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

```
    assign FAN_CTRL = SW[0];

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

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

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

    assign IRDA_TXD = SW[0];

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

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

    assign TD_RESET_N = SW[0];

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


endmodule
```

- This is the Verilog code to display texts on VGA by using pixel based images stored in ROM.

Listing 4: sprites.sv

```
/*
 * Sprite ROM seletor and color decoders as well as syncronous ROM
    module
 * Alex Yu
 * Columbia University
 */

module sprites(
        input logic [5:0]               n_sprite,
        input logic [9:0]               line,
        input logic [5:0]                       pixel,
        input logic                     clk,
        output logic [3:0]                      color_code);

        logic [9:0] spr_rom_addr ;

        assign spr_rom_addr = (line<<5) + pixel;

        logic [3:0] spr_rom_data [32:0];
        // sprites indevidually stored in roms

        // numbers
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/1.txt")
    ) num_1 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd1])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/2.txt")
    ) num_2 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd2])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/3.txt")
    ) num_3 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd3])
    );
```

```
    rom_sync #(
    .WIDTH(4),
    .WORDS(1024),
    .INIT_F("./sprites/Sprite_rom/4.txt")
) num_4 (
    .clk(clk),
    .addr(spr_rom_addr),
    .data(spr_rom_data[6'd4])
);
    rom_sync #(
    .WIDTH(4),
    .WORDS(1024),
    .INIT_F("./sprites/Sprite_rom/5.txt")
) num_5 (
    .clk(clk),
    .addr(spr_rom_addr),
    .data(spr_rom_data[6'd5])
);
    rom_sync #(
    .WIDTH(4),
    .WORDS(1024),
    .INIT_F("./sprites/Sprite_rom/6.txt")
) num_6 (
    .clk(clk),
    .addr(spr_rom_addr),
    .data(spr_rom_data[6'd6])
);
    rom_sync #(
    .WIDTH(4),
    .WORDS(1024),
    .INIT_F("./sprites/Sprite_rom/7.txt")
) num_7 (
    .clk(clk),
    .addr(spr_rom_addr),
    .data(spr_rom_data[6'd7])
);
    rom_sync #(
    .WIDTH(4),
    .WORDS(1024),
    .INIT_F("./sprites/Sprite_rom/8.txt")
) num_8 (
    .clk(clk),
    .addr(spr_rom_addr),
    .data(spr_rom_data[6'd8])
);
    rom_sync #(
    .WIDTH(4),
    .WORDS(1024),
    .INIT_F("./sprites/Sprite_rom/9.txt")
) num_9 (
    .clk(clk),
    .addr(spr_rom_addr),
    .data(spr_rom_data[6'd9])
);
```

```
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/0.txt")
    ) num_10 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd10])
    );

        // Letters
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/A.txt")
    ) num_11 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd11])
    );

        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/B.txt")
    ) num_12 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd12])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/C.txt")
    ) num_13 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd13])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/D.txt")
    ) num_14 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd14])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/E.txt")
    ) num_15 (
        .clk(clk),
```

```
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd15])
    );

        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/F.txt")
    ) num_16 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd16])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/G.txt")
    ) num_17(
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd17])
    );
        // NOTE BLOCKS
        rom_sync #( //blue
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/flat.txt")
    ) num_18 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd18])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/N.txt")
    ) num_19 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd19])
    );
        rom_sync #( //orange
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/O.txt")
    ) num_20 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd20])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/T.txt")
```

```
    ) num_21 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd21])
    );
        rom_sync #( //pink
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/H.txt")
    ) num_22 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd22])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/Z.txt")
    ) num_23 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd23])
    );
        rom_sync #(//purple
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/dot.txt")
    ) num_24 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd24])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/N.txt")
    ) num_25 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd25])
    );
        rom_sync #(//purple
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/A.txt")
    ) num_26 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd26])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/X.txt")
```

```
    ) num_27 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd27])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/R.txt")
    ) num_28 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd28])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/A.txt")
    ) num_29 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd29])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/A.txt")
    ) num_30 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd30])
    );
        rom_sync #(
        .WIDTH(4),
        .WORDS(1024),
        .INIT_F("./sprites/Sprite_rom/A.txt")
    ) num_31 (
        .clk(clk),
        .addr(spr_rom_addr),
        .data(spr_rom_data[6'd31])
    );
        rom_sync #(
                .WIDTH(4),
                .WORDS(1024),
                .INIT_F("./sprites/Sprite_rom/A.txt")
            ) num_32 (
                .clk(clk),
                .addr(spr_rom_addr),
                .data(spr_rom_data[6'd32])
            );
        always_comb begin
                case (n_sprite)
                        // numbers
                        6'd1   : color_code = spr_rom_data[6'd1];   // 1
```

```
                        6'd2  : color_code = spr_rom_data[6'd2];  // 2
                        6'd3  : color_code = spr_rom_data[6'd3];  // 3
                        6'd4  : color_code = spr_rom_data[6'd4];  // 4
                        6'd5  : color_code = spr_rom_data[6'd5];  // 5
                        6'd6  : color_code = spr_rom_data[6'd6];  // 6
                        6'd7  : color_code = spr_rom_data[6'd7];  // 7
                        6'd8  : color_code = spr_rom_data[6'd8];  // 8
                        6'd9  : color_code = spr_rom_data[6'd9];  // 9
                        6'd10 : color_code = spr_rom_data[6'd10]; //
                           10
                        // letters
                        6'd11 : color_code = spr_rom_data[6'd11]; // B
                        6'd12 : color_code = spr_rom_data[6'd12]; // C
                        6'd13 : color_code = spr_rom_data[6'd13]; // E
                        6'd14 : color_code = spr_rom_data[6'd14]; // M
                        6'd15 : color_code = spr_rom_data[6'd15]; // O
                        6'd16 : color_code = spr_rom_data[6'd16]; // R
                        6'd17 : color_code = spr_rom_data[6'd17]; // S
                        // notes
                        6'd18 : color_code = spr_rom_data[6'd18]; // E
                            w
                        6'd19 : color_code = spr_rom_data[6'd19]; // U
                            w
                        6'd20 : color_code = spr_rom_data[6'd20]; // M
                            w
                        6'd21 : color_code = spr_rom_data[6'd21]; // V
                            w
                        6'd22 : color_code = spr_rom_data[6'd22]; // S
                            w
                        6'd23 : color_code = spr_rom_data[6'd23]; // T
                            w
                        6'd24 : color_code = spr_rom_data[6'd24]; // I
                            w
                        6'd25 : color_code = spr_rom_data[6'd25]; // N
                            w
                        // A/X added later
                        6'd26 : color_code = spr_rom_data[6'd26]; // A
                        6'd27 : color_code = spr_rom_data[6'd27]; // X
                        // 6'd28 : color_code = spr_rom_data[6'd28];
                        6'd28 : color_code = spr_rom_data[6'd28];
                        6'd29 : color_code = spr_rom_data[6'd29];
                        6'd30 : color_code = spr_rom_data[6'd30];
                        6'd31 : color_code = spr_rom_data[6'd31];
                        6'd32 : color_code = spr_rom_data[6'd32];
                        default : begin
                                color_code <= 4'h0;
                        end
                endcase
        end
endmodule

module sprite_color_pallete(
        input logic [3:0]       color_code_o,
        input logic [3:0]       color_code_e,
```

```
        input logic             select,
        output logic [23:0]     color
        );
        logic [3:0] color_code;
        assign color_code = (select) ? color_code_o : color_code_e;
        always_comb begin
                case(color_code)
                        //sprite colors
                        //bird
                        4'h0 : color = 24'hFFFFFF;
                        4'h1 : color = 24'hFFFFFF;
                        4'h2 : color = 24'h646361;
                        4'h3 : color = 24'h0a0808;
                        4'h4 : color = 24'hfdc603;
                        4'h5 : color = 24'h5f2a04;
                        4'h6 : color = 24'hea7e02;
                        4'h7 : color = 24'hdab6ff;
                        //pipe
                        4'h8 : color = 24'h101f06;
                        4'h9 : color = 24'h7ed012;
                        4'ha : color = 24'h0bad01;

                        4'hb : color = 24'h05ab00;
                        //background colors
                        4'hc : color = 24'hFFFFFF;
                        4'hd : color = 24'hc3e8d0;
                        4'he : color = 24'hc3e8d0;
                        4'hf : color = 24'h06170e;

                        default : color = 24'h000000;
                endcase
        end
endmodule

module rom_sync #(
    parameter WIDTH=4,
    parameter WORDS=1024,
    parameter INIT_F="",
    parameter ADDRW=10
    ) (
    input wire logic clk,
    input wire logic [ADDRW-1:0] addr,
    output     logic [WIDTH-1:0] data
    );

    logic [WIDTH-1:0] memory [WORDS];

    initial begin
        if (INIT_F != 0) begin
            $display("Creating␣rom_sync␣from␣init␣file␣'%s'.", INIT_F)
                ;
            $readmemh(INIT_F, memory);
        end
    end
```

```
    always_ff @(posedge clk) begin
        data <= memory[addr];
    end
endmodule
```