Musical Stimulus Visualization

ECSEE 4840 Embedded System Design Project Report

Guanxuan Li gl2619, Hongyu Zou hz2552, Shanglin Guo sg3640, Yiqi Sun ys3127
Columbia University

Abstract

Inspired by the patterns shown on the old generation MP3 screen that corresponds to the pitch and frequency of the music being played, we have developed a music visualization system based on the DE1-SoC development platform that responds to the real-time musical stimulus. One of the most difficult technical challenges that we have encountered is getting the audio input from the microphone and decode and sample the input signal appropriately. Through extended testing, we were able to achieve the real-time signal processing from the USB microphone and visualize the generated pattern on a VGA monitor.

1. Overview

1.1 Motivation

Inspired by the patterns shown on the old generation MP3 screen that corresponds to the pitch and volume of the music being played, we came by with the idea to visualize the music in a way that not only looks nice but is also computationally expensive. Therefore, using the FPGA programmable fabric can accelerate the computation of audio information and achieve real-time processing as well as the video output generation.

1.2 Achievement

A USB microphone is utilized to collect the sound signal and transferred into frequency & amplitude information by using FFT. After that, 16 digits data are passed to shift register in FPGA board, then add the size and decrease the color till the circles fade into the background.

For different frequency intervals, lower frequency denotes red/yellow circles, medium frequency demotes green and purple circles and higher frequency denotes blue circles.

The larger the volume is, the larger the initial size is. Similarly, the larger the frequency is, the brighter the color of the circles is.

2. Description

In this section, the objectives of our project will be discussed, followed by several technical challenges that we have encountered and our design decision to solve these challenges. Finally, a system flowchart will be shown to describe the whole system working process.

2.1. Objectives and Technical Challenges

The objective is to present four sets of rotating circles that denote different intervals of frequency and different input volume. When the user input signal of a specific frequency, the corresponding set of circles may change according to the volume.

During the implementation, we have encountered several technical challenges along the way, which includes:

- Getting the frequency of the input signal in a real-time way. Because the sample rate is relatively high, algorithms need to be implemented in order to get the representative frequency.
- Transferring the data from software to the hardware at an appropriate rate. Because in order to display the result smoothly, the transfer rate needs to match the register changing rate.
- Displaying all the circles simultaneously and keep the previous circles when new circles feed in. Shift registers are used to store historical data and a trigger signal is set to control shift

2.2. Problem Formulation and Design

The audio input is collected from a USB microphone and handled by the C code running on the hard processor system (HPS). The audio input is sampled at a constant interval (~ 0.4s). After FFT and noise suppression, the resulting frequency and amplitude information is passed to the hardware through Avalon Bus.

The hardware component read the data from software and store it in SIPO shift registers, then the parallel output from shift registers is used for visualization, and the result is displayed on a 640x480 VGA monitor.

The basic flow chart of the system is shown in Fig. 1.

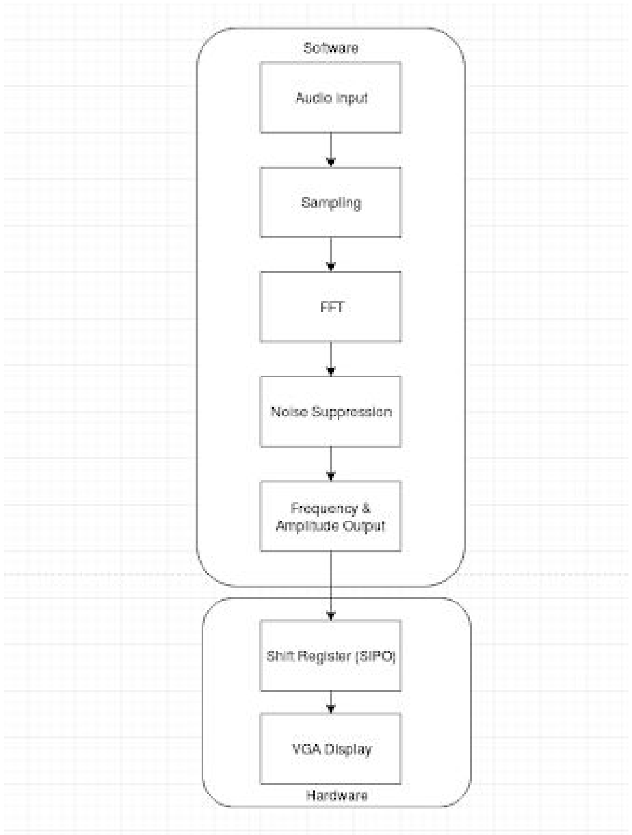

Fig. 1 System flow chart

3. Software Design

The software part includes audio input, data sampling, FFT operation, noise suppression, frequency & energy selection and circular movement algorithm.

3.1 Audio Input

In order to get the audio input from a USB microphone, supported drivers and software interfaces are required to be installed on the board. However, since the original Linux kernel on the board is a fully simplified one which only contains the core components, it cannot support the USB audio input directly. So the following steps are required to read the USB microphone input.

Recompile Kernel: A new kernel containing the USB sound drivers is implemented.

Install ALSA in the Linux system: Alsa is a software framework that provides an application programming interface (API) for sound card device drivers. It also provides audio and MIDI functionality to the Linux operating system.

With the supported USB sound drivers and application programming interfaces, the audio data can be read from the USB microphone.

Now that the audio input from the USB microphone is connected, we have planned several methods of reading and processing the audio input.

The first method that we have tried is using the Linux shell command "arecord" to record from the microphone to a ".wav" file and process the data from the ".wav" file. This methods involves two threads, recording and processing, and will involve a relatively large delay.

The second method is to read a pre-recorded ".wav" file to generate visualization output on the display and play the sound at the same time using SDL2. However, without real-time input, this method is not perfect.

Finally, we were able to implement a method that directly obtains the microphone data from a USB port using c program and do data processing on the input directly.

3.2 Data Sampling

To read the audio input from the USB microphone, the "alsa/asoundlib" C library has been used for real-time audio input reading. Several parameters of the audio input reading are set below:

- The sampling rate is set as 44100Hz;
- PCM_FORMAT is set as 'S16_LE' (signed 16 bits in little endian);
- Mono audio channel;

Input data decoding is crucial to generate the correct time-domain sound signal for future processing. After some research on the USB audio input, some of the data point features are shown below:

- Each sample is a signed 16 bits data point in a little-endian format consisting of higher 8 bits and lower 8 bits.
- All data are stored in the form of two's complement.

The decoding method act in the following way combining the 2 bytes data to a single sampled data point:

```
data = buffer[j+1];
data <<= 8;
data += buffer[j];
```

After decoding, a correct time-domain audio signal is generated.

3.3 FFT Operation

The frequency information of the audio input is required in this system. To extract information from the frequency domain, fast Fourier transform (FFT) is considered to be the most efficient way.

A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa.

To perform FFT operation in the C program, "fftw3" C library (provided by a team in MIT) is a good choice.

The FFT flow chart is shown in Fig. 2.

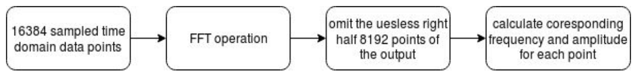
16384 sampled time domain data points
FFT operation
omit the uesless right half 8192 points of the output
calculate coresponding frequency and amplitude for each point

Fig. 2 FFT flow chart

Based on the sampling rate and the hardware display requirements, 16384 data points are sampled and performed FFT operation each time. According to the symmetrical pattern feature of the original FFT output, only the left half part (8192 output points) of the output is useful. Then the energy of each frequency component is calculated using the real and imaginary values.

Several tests have been done to verify the accuracy of the FFT operation. Fig. 3-a and Fig. 3-b refer to the signal in the time domain and frequency domain respectively of environmental noise. Fig. 4-a and Fig. 4-b refer to the signal in the time domain and frequency domain respectively of a test sound of 1KHz. Fig. 5-a and Fig. 5-b refer to the signal in the time domain and frequency domain respectively of a test sound of 10KHz. Finally, Fig. 6-a and Fig. 6-b refer to the signal in the time domain and frequency domain respectively of the human sound.

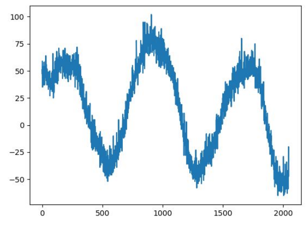

Fig. 3-a

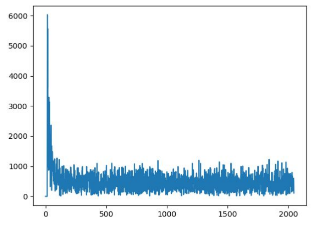

Fig. 3-b

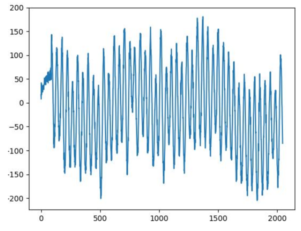

Fig. 4-a

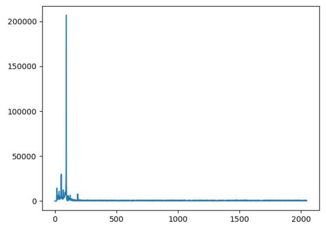

Fig. 4-b

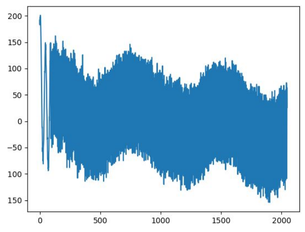

Fig. 5-a

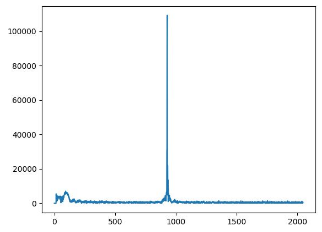

Fig. 5-b

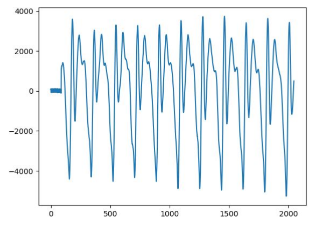

Fig. 6-a

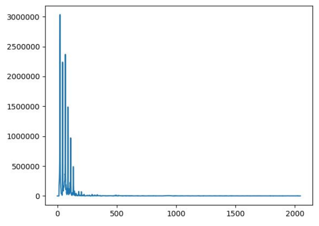

Fig. 6-b

From the results of the tests above, the accuracy of the FFT operation can be demonstrated by the tests of the two standard test sound (1KHz and 10KHz). Furthermore, the frequency of the environmental noise and human sound can be seen from the graph, both of which have relatively low values.

3.4 Noise Suppression

As environment noise will have a negative impact on the FFT result of useful data, noise suppression is required. A straightforward way to suppress the noise influence is to do simple spectral subtraction.

First, sample the noise signal and do FFT operation for it. Then, figure out the main distribution of noise energy in the frequency domain. Finally, in the frequency domain, just subtract the energy of mixed signal with the noise energy in each frequency component, omitting the useless noise in the frequency domain.

Fig. 7-a and Fig. 7-b show the noise signal in the time domain and frequency domain respectively. The frequency domain result is used for spectral subtraction.

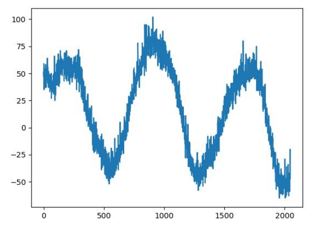

Fig. 7-a

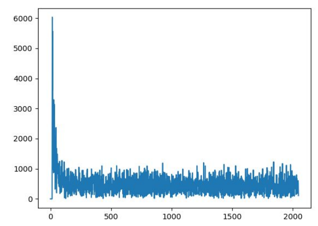

Fig. 7-b

3.5 Frequency & Energy Selection

Since there is a large amount of FFT output data, it is not realistic to transfer all of them to the hardware. So a selection algorithm should be implemented to select some valuable data which is able to filter out the most remarkable characteristics of the audio input so that our display can be distinguished among different kinds of audio.

To begin with, in our implementation, we get rid of the frequency that is greater than 11kHz, because normal audio input like human sound or music would not generate high energy under the frequency higher than 10kHz. Then, the whole frequency domain is divided into 4 parts, 150Hz to 1kHz, 1kHz to 2.5kHz, 2.5kHz to 5kHz and 5kHz to 11kHz. Our division is not even because we want to put more weight on the lower-frequency part, which is normally the frequency of human voice and most music. In each part, we choose one frequency having the largest energy as our data feed.

Moreover, as the raw frequency & energy values of the FTT output are too large, modulation should be done to adapt them to meet the hardware requirement which requires the frequency number is less than 765 and the energy number is less than 100 in our implementation. Therefore, we figure out an algorithm to map the data to this range as shown in the function below.

$$\text{power} = (\text{short})\left(\frac{\log(rawPower)}{\log(2)} * 6 - 80\right)$$

$$\text{frequency} = (\text{short})\left(\frac{rawFrequency}{22} + 450\right)$$

Besides, we also encode the data to short int, so that we can transfer the data through iowrite16. Then, four useful <frequency, energy> pairs are generated and transmitted to the hardware.

3.6 Circular Movement

In addition to the <frequency, energy> pair data, we also transfer 8 location data to the hardware, so that our pattern is able to move and look fancier.

In our display, 4 dynamic circles will move around the center of the screen. The circular movement algorithm is implemented in the software C program.

```
if(angle < 6.28)
        angle = angle + 0.05;
    else
        angle1 = 0;
x = (short)(640 / 2 + cos(angle) * r) *2;
y = (short)(480 / 2 + sin(angle) * r);
```

As shown in the pseudocode above, “x” and “y” are the position data we need to transfer and they are computed from the variable angle which denotes the radian of the circle. The if the statement shows that if the points are now within a circle, the radian will add 0.05 to itself in every cycle where 6.28 is equal to 2π denoting a full circle. Then, we can compute the “x” and “y” position data. Since the vga_clock signal is 25MHz and the system clock signal is 50MHz in the hardware, so “x” needs to be multiplied by 2.

Therefore, the positions (coordinate) of the 4 dynamic circles are generated in real-time and passed to the hardware.

4. Hardware Design

The hardware part includes the input of 4*16 digits frequency, 4*16 digits amplitude, and 8*16 digits position data (4 for x position and 4 for y position). Also, it includes 4*48 digits serial in parallel out shift registers.

The design logic is shown in Fig. 8.

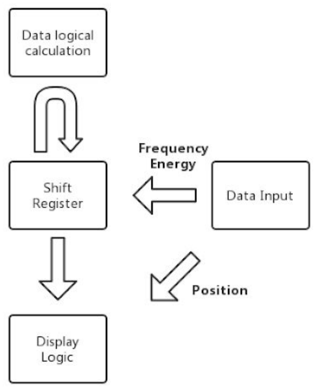

Fig. 8 Hardware Design Logic

4.1 Data Input

4 sets of 16 digits frequency and 4 sets of 16 digits energy data are transferred into hardware through 16 digits “writedata” logic input. As long as there is an input, either frequency or size, the enable signal would be set to

"1", which means there would be a new 16-digit value shift into the shift register and the oldest value is shifted out. Besides, we have a "tictoc" pulse signal denotes the enable of the increase or decrease of the value inside the shift register. As soon as the "tictoc" signal is "1", 0~15 digits, 16~31 digits and 32~47 digits of all the shift registers would change simultaneously and for size register it's "+1" and for frequency register it's "-2" to ensure when the circle is getting bigger the color of the circle is getting darker and finally fade into the background.

The schematic diagram of the input and the shift registers is shown in Fig. 9.

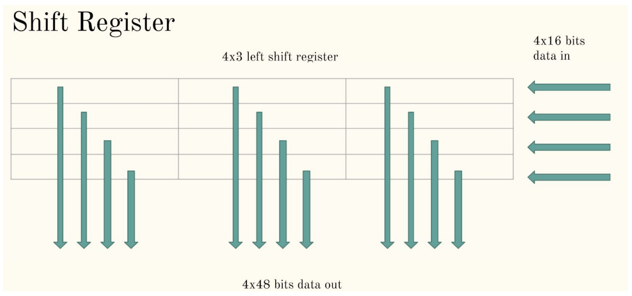

Fig. 9 Shift Register Logic

4.2 Display logic

There are four sets of circles displayed on the screen and each set has a different color range, which denotes a different range of frequency. The initial size of the circle is related to input volume and the brightness of the color is related to the input frequency.

The frequency is divided into four intervals and the red/yellow circles denote the lowest frequency, green and purple circles denote two medium frequencies. Blue circles denote the highest frequency. We also have a rotation logic. There are four x_position input and four y_position input represent different positions of the sets of circles.

The logic is to execute "+1" on size and execute "-2" on frequency in order to make the circles fade into the black background and increase the size at the same time. The logic of the circle is to show the pixel when the pixel is larger than the square of the size and smaller than the square of the size plus 1. We utilize multipliers to accomplish the square function. It's not time efficient but easy to implement.

5. Results

In order to show the result in different frequency intervals, we firstly use vocal input to test the output. Because the vocal signal mainly consists of low-frequency signals, the size of red/yellow circles would increase significantly as well as the green circles.

Then we used the sound signal which frequency continuous changes according to time. When the frequency is relatively low, the red/yellow circles are significantly larger than other circles. Along with the increase of the time, the size of green circles, purple circles, and the blue circles would increase one by one.

All of the sets of circles rotate around the center of the screen continuously and smoothly.

One of the displaying results is shown in Fig. 10.

.

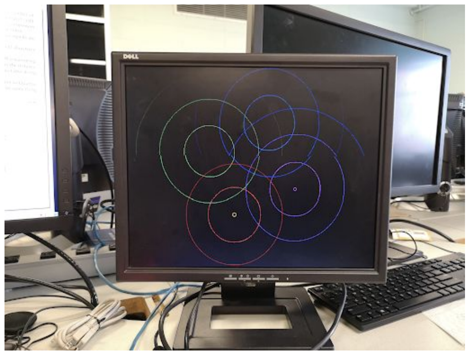

Fig. 10 Display Result

6. Conclusion

In conclusion, in this project, we successfully built a music stimulus data visualization system utilizing both the software and hardware resources. In other words, this is also a streaming data processing and hardware data visualization project. In the software part, we managed to gather data from a USB microphone and make use of the data by decoding it. Then, we implemented an FFT component for data processing in which we used many innovative algorithms and data processing techniques to make our result much more reliable. Moreover, we also implemented a pattern control logic to control the movement of our pattern to make the result fancier.

Then we transferred all the data feed through Avalon bus to the hardware. And we built a VGA monitor display logic in the hardware system, including several shift registers and some innovative technique to display a fancy pattern based on the data input.

This project is aiming to turn the theory we learned in class into practice and we made use of lots of knowledge we learned in class including device driver, Avalon bus, communication with peripherals, etc. to successfully establish the system.

7. Acknowledgments

We would like to express our special thanks of gratitude to our professor (Stephen A. Edwards) as well as our teaching assistants (John Hui, Martha Barker) who gave us the golden opportunity to do this wonderful project in the course of Embedded System Design, which also helped us in doing a lot of research. We learned many new knowledge and skills during this project and we are really thankful to them.