Design Document: Audio Visualizer
CSEE 4840

Manas Pange (mmp2248)

Yaagna Modi (ykm2110)

Gaurav Agarwal (gsa2131)

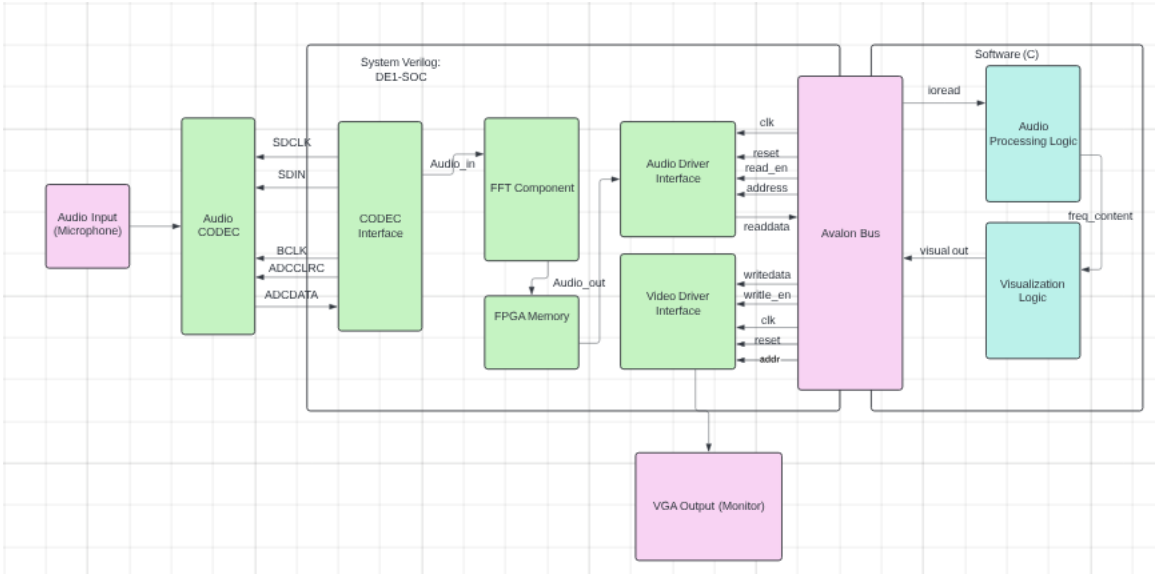
Max Lavey (mjl2274)

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 64 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, offering five distinct display modes. These modes are ingeniously manipulated by toggling LEDs in varying positions across each column, allowing for a range of customizable patterns easily modified or created by the user. Mode selection is facilitated by a push button, enabling users to cycle through display patterns with each press, ultimately reverting to the default setting. This push button is linked to a digital input, which is monitored at each cycle of display refresh, ensuring a dynamic and interactive user experience. This project not only aims to showcase technical prowess in FPGA programming and FFT analysis but also to enrich the audio experience with a visually stimulating spectrum display.

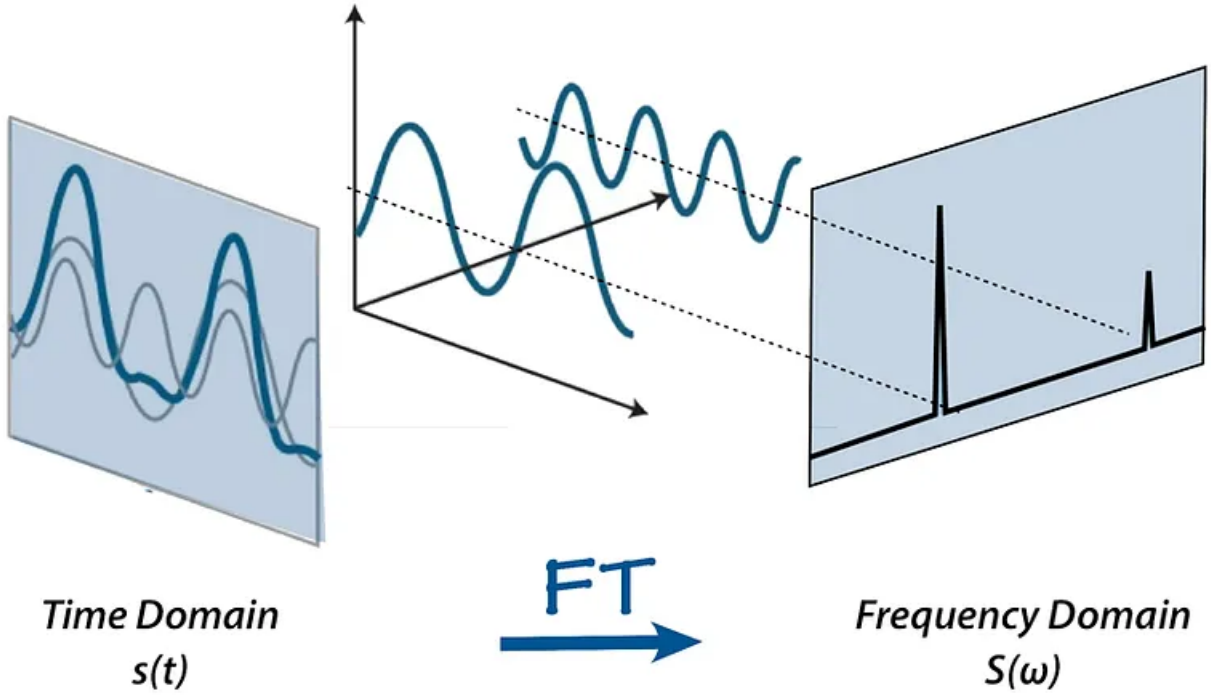
2. Block Diagram



First, we will take an audio input from an external microphone that is plugged into the DE1-SOC. This will be a USB microphone that is yet to be purchased. 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 perform Fast Fourier Transform in the hardware. When the audio signal is finished with its FFT, it will be placed in the FPGA memory to be read out sequentially by the Audio Driver interface. This driver interface will go through the Avalon Bus to be processed by the Software. The software has two main components: Audio Processing Logic, and Visualization Logic. For the Audio Processing Logic, we are going to take the FFT output and process it to analyze its frequencies. Based on the 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 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.

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 32 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 32 kHz, a bit depth of 32 bits, and a mono channel spanning 3 seconds, the calculations unfold as follows:

Total Sample Count = Sample Rate $\times$ Duration = 32,000 $\times$ 3 = 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 can rest assured that we won't encounter memory constraints or overlook space for additional overhead.

5. Hardware/Software Interface

Hardware

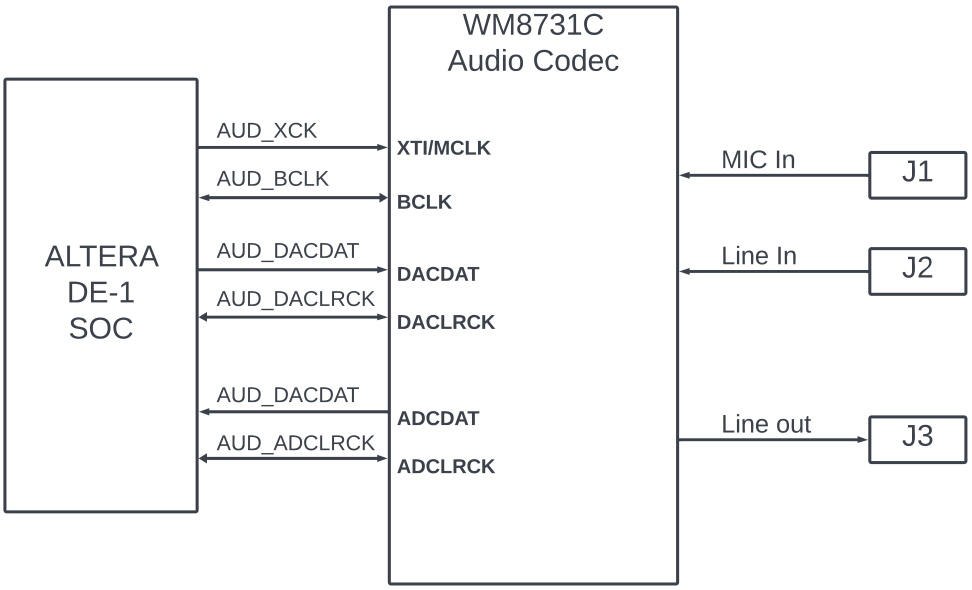
The Altera DE1 SOC features the WM8731 audio codec, known for its low power consumption and integrated headphone driver, tailored for portable MP3 audio and speech players as well as recorders. It's well-suited for applications such as MD, CD-RW machines, and DAT recorders. An external, passive microphone 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.

Software

The Driver Interface functions by fetching samples from the hardware. 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 64 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 32-bit sample, adjusted for octave shifting, 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. The FFT algorithm executed in the hardware identifies the highest frequency, from which the corresponding note being played is extracted. Utilizing the frequency content of the signal, we generate sprites and patterns to display the notes via a VGA interface.

6. 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.