## College of Electrical and Mechanical Engineering

## National University of Sciences and Technology

## 

## Digital Signal Processing

## Spring 2023

## Project

## (Complex Engineering Problem)

As part of OBE paradigm, in this course, you are expected to work on a complex engineering problem as your semester project. This complex engineering problem tests the students for their ***depth of knowledge required ( WP1)*, and *range of conflicting requirements (WP2)* and *depth of analysis required (WP3)*.**

**Objective:**

The objective of this years project is to design and apply the knowledge of Digital Signal Processing to design and implement a digital system on a DSP Processor.

**Background:**

You have already designed and implemented the audio equalizer in your labs. We have done several labs around this where we gave you the already designed filters, and in another lab, we asked you to design the filters your self according to the audio of your choice. In this project, you will implement the audio equalizer in the DSP processor kit that we have.

**Goals:**

Your main goal is to use the available DSP kits to implement audio equalizer. You will have to write your code in the code composer that requires you to write your code in the C programming language.

**Tasks:**

1. **Audio Input/ Output**

Working with audio input/output to a DSP processor is always a challenge. There are multiple options that you can take:

1. Store the input/output files and perform the read/write operation on that files
2. Take the real time input/output from the mic/ speaker.
3. **Filter Designing/Implmentation**

You have to apply different filters in the DSP processor to separate the audio into different frequency bands.

Get the input from the user OR fix the gains

Implement different filters. As the filters are implemented using the difference equations, they can be very easily implemented in C using for loops

1. **Audio Output**

You have to write the final output of the equalized audio into a file or send it to the audio output port of the DSP processor.

**Code:**

**#include** <stdio.h>

**#include** <stdlib.h>

**#define** MAX\_SAMPLES 1000

// Filter coefficients for different bands

**const** **float** lowPassCoefficients[3] = {0.25, 0.5, 0.25};

**const** **float** highPassCoefficients[3] = {-0.25, 0.5, -0.25};

// Number of bands

**#define** NUM\_BANDS 2

**int** **main**() {

FILE \*csvFile;

**float** inputSignal[MAX\_SAMPLES];

**float** filteredSignals[NUM\_BANDS][MAX\_SAMPLES];

**float** state[NUM\_BANDS][2] = {{0}};

// Open the CSV file for reading

csvFile = **fopen**("C:\\Users\\Surface\\Documents\\vv\\audio.csv", "r");

**if** (csvFile == NULL) {

**printf**("Failed to open the CSV file.\n");

**return** 1;

}

// Read the audio signal from the CSV file

**int** sampleCount = 0;

**float** sampleValue;

**while** (**fscanf**(csvFile, "%f", &sampleValue) != EOF && sampleCount < MAX\_SAMPLES) {

inputSignal[sampleCount] = sampleValue;

sampleCount++;

}

// Apply the filters to the input signal

**int** band;

**for** (band = 0; band < NUM\_BANDS; band++) {

**int** n;

**for** (n = 0; n < sampleCount; n++) {

**float** output = 0.0;

**int** k;

**for** (k = 0; k <= 2; k++) {

output += (band == 0 ? lowPassCoefficients[k] : highPassCoefficients[k]) \* inputSignal[n - k];

}

filteredSignals[band][n] = output;

}

}

// Print the filtered signals

// int band;

**for** (band = 0; band < NUM\_BANDS; band++) {

**printf**("Filtered Signal for Band %d:\n", band);

**int** n;

**for** (n = 0; n < sampleCount; n++) {

**printf**("%f\n", filteredSignals[band][n]);

}

**printf**("\n");

}

// Close the CSV file

**fclose**(csvFile);

**return** 0;

}

**Introduction**

The purpose of this project is to implement a simple audio signal filtering algorithm using a finite impulse response (FIR) filter. The code provided reads an input audio signal from a CSV file, applies two filters (low-pass and high-pass) to the input signal, and outputs the filtered signals. This report provides an overview of the project, explains the code structure, and discusses the implementation details.

**Project Overview**

The project aims to demonstrate the application of digital signal processing techniques to filter audio signals. The code implements a basic FIR filter using predefined filter coefficients for two different frequency bands: low-pass and high-pass. The input audio signal is read from a CSV file, and the filtered signals are computed and printed to the console.

**Code Structure**

The code is written in C programming language and consists of the following main sections:

**Header Files:** The necessary header files, including <stdio.h> and <stdlib.h>, are included to enable input/output and memory allocation functionalities.

**Constants and Definitions:** The code defines a constant MAX\_SAMPLES to determine the maximum number of audio samples to process. It also defines two sets of filter coefficients (lowPassCoefficients and highPassCoefficients) for the low-pass and high-pass filters, respectively. The number of frequency bands is defined as NUM\_BANDS.

**Main Function:** The main() function is the entry point of the program. It performs the following steps:

Opens the CSV file containing the input audio signal for reading.

Checks if the file was successfully opened; otherwise, it displays an error message and terminates the program.

Reads the audio samples from the CSV file and stores them in the inputSignal array until either the end of file or the maximum sample count is reached.

Applies the filters to the input signal by iterating over each frequency band and sample.

Stores the filtered signals in the filteredSignals array.

Prints the filtered signals for each frequency band to the console.

Closes the CSV file.

Implementation Details

Input Signal: The input audio signal is read from a CSV file named "audio.csv". Each line in the file represents a single audio sample, which is read and stored in the inputSignal array. The maximum number of samples to process is defined by MAX\_SAMPLES.

**Filter Coefficients:** Two sets of filter coefficients are defined for the low-pass and high-pass filters. The low-pass filter coefficients (lowPassCoefficients) represent a simple moving average, while the high-pass filter coefficients (highPassCoefficients) perform a differencing operation. These coefficients are used to compute the filtered output for each frequency band.

**Filtering Process:** The code applies the filters to the input signal using nested loops. For each frequency band, it iterates over each sample in the input signal and calculates the filtered output using the corresponding filter coefficients. The filtered output is stored in the filteredSignals array for further processing or analysis.

**Output:** The code prints the filtered signals for each frequency band to the console. It iterates over each frequency band and sample, displaying the filtered value. The output is organized by band, and each band's filtered signals are separated by an empty line.

**Conclusion**

In this project, a basic audio signal filtering algorithm using FIR filters has been implemented. The provided code reads an input audio signal from a CSV file, applies low-pass and high-pass filters to the signal, and outputs the filtered signals. The project demonstrates the application of digital signal processing techniques to modify audio signals based on different frequency bands.

Future improvements to the project could include additional filter types, user-defined

**Advantages of DSK6713 over DSK6416:**

The DSK6713 toolkit offers several advantages over the DSK6416 toolkit:

1. **Higher Processing Power:** The DSK6713 utilizes the TMS320C6713 DSP chip, which generally provides higher processing power and a higher clock speed compared to the TMS320C6416 DSP chip used in the DSK6416. This increased processing power allows for more complex and computationally intensive signal processing algorithms.
2. **More On-Chip Memory:** The DSK6713 typically has more on-chip memory available compared to the DSK6416. This additional memory capacity is advantageous when working with larger audio buffers or implementing algorithms that require more memory for intermediate calculations or data storage.
3. **Enhanced Peripheral Connectivity:** The DSK6713 often offers expanded peripheral connectivity options compared to the DSK6416. It may include additional serial ports, Ethernet, USB, or other interfaces, enabling easier integration with external devices or communication networks.
4. **Improved Development Tools and Support:** The DSK6713 benefits from the advancements in development tools, documentation, and community support compared to the DSK6416. It may have a more mature and feature-rich software development kit (SDK), which can facilitate faster development and debugging of DSP applications.
5. **Wider Availability:** Due to the DSK6713's newer technology and improved performance, it may be more widely available and supported by manufacturers and distributors. This can make it easier to source components, find compatible software and libraries, and access technical support when needed.