PROJECT REPORT
2006 - 2007

IMPLEMENTATION OF A VOICE - BASED BIOMETRIC SYSTEM

Submitted in the partial fulfillment for the
Award of Degree of
BACHELOR OF ENGINEERING
in
ELECTRONICS AND COMMUNICATION ENGINEERING
By the **Visveswaraya Technological University, Belgaum**
By

ADARSH K.P. (1RV03EC007)
A. R. DEEPAK (1RV03EC001)
DIWAKAR R. (1RV03EC037)
KARTHIK R. (1RV03EC138)

UNDER THE GUIDANCE OF

Mrs. M. Uttara Kumari
Assistant Professor
Dept. of Electronics and Communication
R.V.College of Engineering, Bangalore

Department of Electronics and Communication Engineering,
Rashtreeya Vidyalaya College of Engineering
R.V Vidyaniketan Post, Mysore Road, Bangalore - 560028

R.V.COLLEGE OF ENGINEERING
Mysore Road, Bangalore– 560 059
Department of Electronics and Communication Engineering

<u>CERTIFICATE</u>

This is to certify that the Project Titled "**Implementation of a Voice-Based Biometric System**" is a bonafide work carried out by:

ADARSH K.P. (1RV03EC007)
A. R. DEEPAK (1RV03EC001)
DIWAKAR R. (1RV03EC037)
KARTHIK R. (1RV03EC138)

in partial fulfillment for the award of the degree of Bachelor of Engineering in Electronics and Communication Engineering of the Visveswaraya Technological University, Belagum during the year 2006-2007. It is certified that all corrections /suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library. The project has been approved as it satisfies the academic requirements of the Project work prescribed for the Bachelor of engineering Degree.

Signature of the Guide Dept of E&C	Signature of the HOD Dept of E&C	Signature of Principal RVCE

Name of Examiners	Signature with date
1.	
2.	

ACKNOWLEDGEMENT

We would like to express our gratitude to all those who guided us in the completion of the project.

We are thankful to **Dr. S.C Sharma, Principal R.V.C.E** for encouraging us in the successful completion of the project.

We would like to thank **Prof S. Jagannathan, Head of Department, Electronics and Communication Engineering,** for providing constant support and encouragement during the course of our Project.

We are very grateful to **Mrs. M. Uttarakumari, Assistant Professor, Dept. of ECE**, our guide and teacher for her immense support, constant encouragement and exemplary guidance at every stage of the project. She has always emphasized in thoroughness and clarity in our approach and we are thankful to her, for helping us put our project in perspective.

We would like to express our gratitude to the teaching and non teaching staff of the Department of Electronics and Communication Engineering for their constant help and support throughout this project. We express our sincere thanks to **Mr. G. Harisha, Mr. C. Jayachandra and Mr. Ramesh Babu** who has been very helpful and cooperative in the use of the facilities at our department’s lab.

SYNOPSIS

Speech Recognition is the process of automatically recognizing a certain word spoken by a particular speaker based on individual information included in speech waves. This technique makes it possible to use the speaker's voice to verify his/her identity and provide controlled access to services like voice based biometrics, database access services, voice based dialing, voice mail and remote access to computers.

Signal processing front end for extracting the feature set is an important stage in any speech recognition system. The optimum feature set is still not yet decided though the vast efforts of researchers. There are many types of features, which are derived differently and have good impact on the recognition rate. This project presents one of the techniques to extract the feature set from a speech signal, which can be used in speech recognition systems.

The key is to convert the speech waveform to some type of parametric representation (at a considerably lower information rate) for further analysis and processing. This is often referred as the *signal-processing front end.* A wide range of possibilities exist for parametrically representing the speech signal for the speaker recognition task, such as *Linear Prediction Coding (LPC), Mel-Frequency Cepstrum Coefficients (MFCC),* and others. MFCC is perhaps the best known and most popular, and these will be used in this project. MFCCs are based on the known variation of the human ear's critical bandwidths with frequency filters spaced linearly at low frequencies and logarithmically at high frequencies have been used to capture the phonetically important characteristics of speech. However, another key characteristic of speech is quasi-stationarity, i.e. it is short time stationary which is studied and analyzed using short time, frequency domain analysis.

In this project, we propose to build a simple yet complete and representative automatic speaker recognition system, as applied to a voice based biometric system i.e. a voice based access control system. To achieve this, we have first made a comparative study of the MFCC approach with the Time domain approach for recognition by simulating both these techniques using MATLAB 7.0 and analyzing the consistency of recognition using both the techniques. This is followed by the implementation of the time domain approach on the TMS 320 C6713 DSK.

The voice based biometric system is based on isolated or single word recognition. A particular speaker utters the password once in the training session so as to train and store the features of the access word. Later in the testing session the user utters the password again in order to achieve recognition if there is a match. The feature vectors unique to that speaker are obtained in the training phase and this is made use of later on to grant authentication to the same speaker who once again utters the same word in the testing phase. At this stage an intruder can also test the system to test the inherent security feature by uttering the same word.

CONTENTS

1. INTRODUCTION

1.1 INTRODUCTION

The concept of a machine than can recognize the human voice has long been an accepted feature in Science Fiction. From ‘Star Trek’ to George Orwell’s ‘1984’ - *“Actually he was not used to writing by hand. Apart from very short notes, it was usual to dictate everything into the speakwriter.”* - It has been commonly assumed that one day it will be possible to converse naturally with an advanced computer-based system. Indeed in his book ‘The Road Ahead’, Bill Gates (co-founder of Microsoft Corp.) hails Automatic Speaker Recognition (ASR) as one of the most important innovations for future computer operating systems.

From a technological perspective it is possible to distinguish between two broad types of ASR: ‘direct voice input’ (DVI) and ‘large vocabulary continuous speech recognition’ (LVCSR). DVI devices are primarily aimed at voice command-and-control, whereas LVCSR systems are used for form filling or voice-based document creation. In both cases the underlying technology is more or less the same. DVI systems are typically configured for small to medium sized vocabularies (up to several thousand words) and might employ word or phrase spotting techniques. Also, DVI systems are usually required to respond immediately to a voice command. LVCSR systems involve vocabularies of perhaps hundreds of thousands of words, and are typically configured to transcribe continuous speech. Also, LVCSR need not be performed in real-time - for example, at least one vendor has offered a telephone-based dictation service in which the transcribed document is e-mailed back to the user.

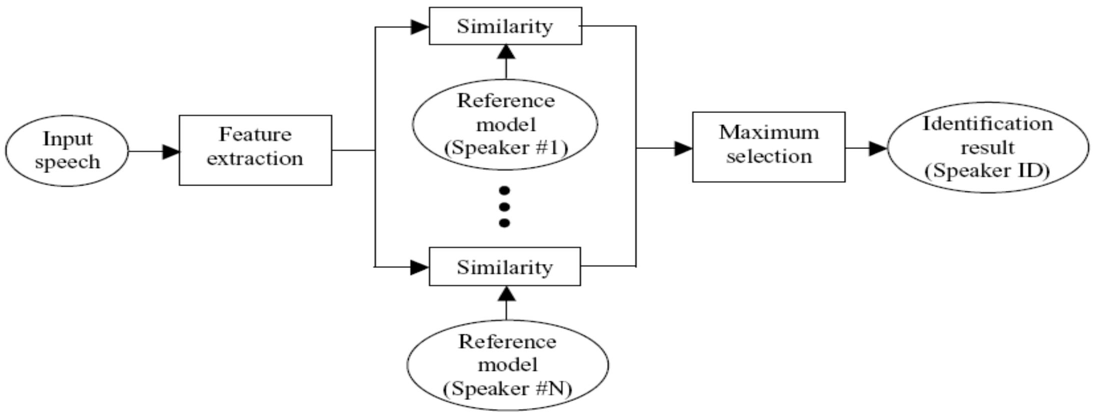
From an application viewpoint, the benefits of using ASR derive from providing an extra communication channel in hands-busy eyes-busy human-machine interaction (HMI), or simply from the fact that talking can be faster than typing. Also, whilst speaking to a machine cannot be described as natural, it can nevertheless be considered intuitive; as one ASR advertisement declared *“you have been learning since birth the only skill needed to use our system”.*

1.2 MOTIVATION

The motivation for ASR is simple; it is man's principle means of communication and is, therefore, a convenient and desirable mode of communication with machines. Speech communication has evolved to be efficient and robust and it is clear that the route to computer based speech recognition is the modeling of the human system. Unfortunately from pattern recognition point of view, human recognizes speech through a very complex interaction between many levels of processing; using syntactic and semantic information as well very powerful low level pattern classification and processing. Powerful classification algorithms and sophisticated front ends are, in the final analysis, not enough; many other forms of knowledge, e.g. linguistic, semantic and pragmatic, must be built into the recognizer. Nor, even at a lower level of sophistication, is it sufficient merely to generate "a good" representation of speech (i.e. a good set of features to be used in a pattern classifier); the classifier itself must have a considerable degree of sophistication. It is the case, however, it do not effectively discriminate between classes and, further, that the better the features the easier is the classification task.

Automatic speech recognition is therefore an engineering compromise between the ideal, i.e. a complete model of the human, and the practical, i.e. the tools that science and technology provide and that costs allow.

At the highest level, all speaker recognition systems contain two main modules (refer to Fig 1.1): *feature extraction* and *feature matching*. Feature extraction is the process that extracts a small amount of data from the voice signal that can later be used to represent each speaker. Feature matching involves the actual procedure to identify the unknown speaker by comparing extracted features from his/her voice input with the ones from a set of known speakers. We will discuss each module in detail in later sections.



Input speech
Feature extraction
Similarity
Reference model (Speaker #1)
Similarity
Reference model (Speaker #N)
Maximum selection
Identification result (Speaker ID)

Figure 1.1: Speaker Identification (Training)

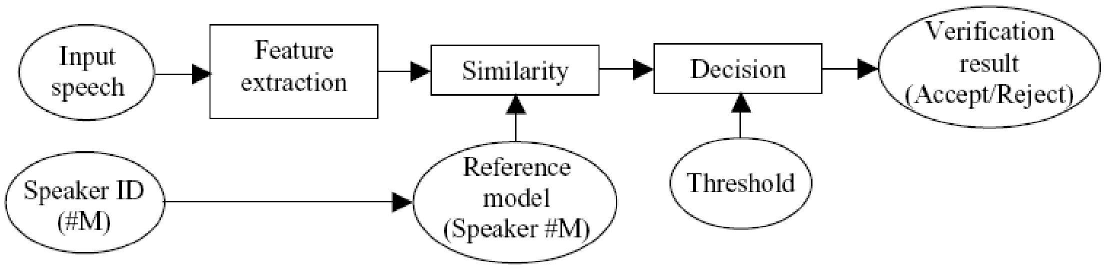


Figure 1.2: Speaker verification (Testing)

All Recognition systems have to serve two different phases. The first one is referred to the *enrollment sessions or training phase* while the second one is referred to as the *operation sessions or testing phase*. In the *training phase*, each registered speaker has to provide samples of their speech so that the system can build or train a reference model for that speaker. In case of speaker verification systems, in addition, a speaker-specific threshold is also computed from the training samples. During the *testing (operational) phase* (see Figure 1.2), the input speech is matched with stored reference model(s) and recognition decision is made.

Speech recognition is a difficult task and it is still an active research area. Automatic speech recognition works based on the premise that a person's speech exhibits characteristics that are unique to the speaker. However this task has been challenged by the highly *variant* of input speech signals. The principle source of variance is the speaker himself. Speech signals in training and testing sessions can be greatly different due to many facts such as people voice change with time, health conditions (e.g. the speaker has a cold), speaking rates, etc. There are also other factors, beyond speaker variability, that present a challenge to speech recognition technology.

Examples of these are acoustical noise and variations in recording environments (e.g. speaker uses different telephone handsets). The challenge would be make the system "**Robust**".

So what characterizes a "***Robust System***"? When people use an automatic speech recognition (ASR) system in real environment, they always hope it can achieve as good recognition performance as human's ears do which can constantly adapt to the environment characteristics such as the speaker, the background noise and the transmission channels. Unfortunately, at present, the capacities of adapting to unknown conditions on machines are greatly poorer than that of ours. In fact, the performance of speech recognition systems trained with clean speech may degrade significantly in the real world because of the mismatch between the training and testing environments. If the recognition accuracy does not degrade very much under mismatch conditions, the system is called "***Robust***".

1.3 ORGANIZATION OF THE REPORT

The report first introduces the reader to speech processing by giving a theoretical overview; with a stress on speech recognition to build a foundation for our project. In Chapter 3, we describe the process of feature extraction, which outlines the steps involved in extracting the feature vectors required to suitably represent the speech uttered.

In Chapter 4, we discuss in detail the algorithms which we have simulated and tested using MATLAB, with a comparative study of both. This is followed by the architectural details of the TMS320C6713 digital signal processor in Chapter 5, which we have used for the implementation of our project. In the next chapter, we have discussed some optimization steps to be followed while implementing an ASR system on the DSK.

The source code in MATLAB and C, for the simulation and the implementation of our speech recognition algorithm is provided in Chapter 7. The report ends with conclusion and scope for future work in Chapter 8.

2. ASR SYSTEMS

2.1 INTRODUCTION

Speech processing is the study of speech signals and the processing methods of these signals. The signals are usually processed in a digital representation whereby speech processing can be seen as the interaction of digital signal processing and natural language processing.

Natural language processing is a subfield of artificial intelligence and linguistics. It studies the problems of automated generation and understanding of natural human languages. Natural language generation systems convert information from computer databases into normal-sounding human language, and natural language understanding systems convert samples of human language into more formal representations that are easier for computer programs to manipulate.

Speech coding:

It is the compression of speech (into a code) for transmission with speech codecs that use audio signal processing and speech processing techniques. The techniques used are similar to that in audio data compression and audio coding where knowledge in psychoacoustics is used to transmit only data that is relevant to the human auditory system. For example, in narrow band speech coding, only information in the frequency band of 400 Hz to 3500 Hz is transmitted but the reconstructed signal is still adequate for intelligibility.

However, speech coding differs from audio coding in that there is a lot more statistical information available about the properties of speech. In addition, some auditory information which is relevant in audio coding can be unnecessary in the speech coding context. In speech coding, the most important criterion is preservation of intelligibility and "pleasantness" of speech, with a constrained amount of transmitted data.

It should be emphasized that the intelligibility of speech includes, besides the actual literal content, also speaker identity, emotions, intonation, timbre etc. that are all important for perfect intelligibility. The more abstract concept of pleasantness of degraded speech is a different property than intelligibility, since it is possible that degraded speech is completely intelligible, but subjectively annoying to the listener.

Speech synthesis:

Speech synthesis is the artificial production of human speech. A text-to-speech (TTS) system converts normal language text into speech; other systems render symbolic linguistic representations like phonetic transcriptions into speech.

Synthesized speech can also be created by concatenating pieces of recorded speech that are stored in a database. Systems differ in the size of the stored speech units; a system that stores phones or diphones provides the largest output range, but may lack clarity. For specific usage domains, the storage of entire words or sentences allows for high-quality output. Alternatively, a synthesizer can incorporate a model of the vocal tract and other human voice characteristics to create a completely "synthetic" voice output.

The quality of a speech synthesizer is judged by its similarity to the human voice, and by its ability to be understood. An intelligible text-to-speech program allows people with visual impairments or reading disabilities to listen to written works on a home computer. Many computer operating systems have included speech synthesizers since the early 1980s.

Voice analysis:

Voice problems that require voice analysis most commonly originate from the vocal cords since it is the sound source and is thus most actively subject to tiring. However, analysis of the vocal cords is physically difficult. The location of the vocal cords effectively prohibits direct measurement of movement. Imaging methods such as x-rays or ultrasounds do not work because the vocal cords are surrounded by cartilage which distorts image quality. Movements in the vocal cords are rapid, fundamental frequencies are usually between 80 and 300 Hz, thus preventing usage of ordinary video. High-speed videos provide an option but in order to see the vocal cords the camera has to be positioned in the throat which makes speaking rather difficult.

Most important indirect methods are inverse filtering of sound recordings and electroglottographs (EGG). In inverse filtering methods, the speech sound is recorded outside the mouth and then filtered by a mathematical method to remove the effects of the vocal tract. This method produces an estimate of the waveform of the pressure pulse which again inversely indicates the movements of the vocal cords. The other kind of inverse indication is the electroglottographs, which operates with electrodes attached to the subject’s throat close to the vocal cords. Changes in conductivity of the throat indicate inversely how large a portion of the vocal cords are touching each other. It thus yields one-dimensional information of the contact area. Neither inverse filtering nor EGG is thus sufficient to completely describe the glottal movement and provide only indirect evidence of that movement.

Speech recognition:

Speech recognition is the process by which a computer (or other type of machine) identifies spoken words. Basically, it means talking to your computer, and having it correctly recognize what you are saying. This is the key to any speech related application.

As shall be explained later, there are a number ways to do this but the basic principle is to somehow extract certain key features from the uttered speech and then treat those features as the key to recognizing the word when it is uttered again.

2.2 SPEECH RECOGNITION BASICS

Utterance

An utterance is the vocalization (speaking) of a word or words that represent a single meaning to the computer. Utterances can be a single word, a few words, a sentence, or even multiple sentences.

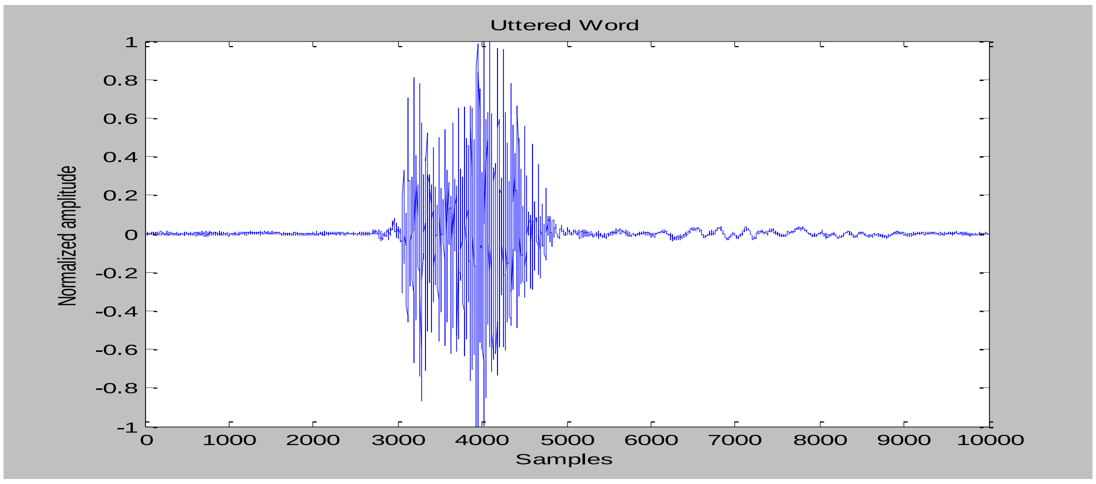


Figure 2.1: Utterance of "HELLO"

Speaker Dependence

Speaker dependent systems are designed around a specific speaker. They generally are more accurate for the correct speaker, but much less accurate for other speakers. They assume the speaker will speak in a consistent voice and tempo. Speaker independent systems are designed for a variety of speakers. Adaptive systems usually start as speaker independent systems and utilize training techniques to adapt to the speaker to increase their recognition accuracy.

Vocabularies

Vocabularies (or dictionaries) are lists of words or utterances that can be recognized by the SR system. Generally, smaller vocabularies are easier for a computer to recognize, while larger vocabularies are more difficult. Unlike normal dictionaries, each entry doesn't have to be a single word. They can be as long as a sentence or two. Smaller vocabularies can have as few as 1 or 2 recognized utterances (e.g.” Wake Up"), while very large vocabularies can have a hundred thousand or more!

Accuracy

The ability of a recognizer can be examined by measuring its accuracy - or how well it recognizes utterances. This includes not only correctly identifying an utterance but also identifying if the spoken utterance is not in its vocabulary. Good ASR systems have an accuracy of 98% or more! The acceptable accuracy of a system really depends on the application.

Training

Some speech recognizers have the ability to adapt to a speaker. When the system has this ability, it may allow training to take place. An ASR system is trained by having the speaker repeat standard or common phrases and adjusting its comparison algorithms to match that particular speaker. Training a recognizer usually improves its accuracy.

Training can also be used by speakers that have difficulty speaking, or pronouncing certain words. As long as the speaker can consistently repeat an utterance, ASR systems with training should be able to adapt.

2.3 CLASSIFICATION OF ASR SYSTEMS

A speech recognition system can operate in many different conditions such as speaker dependent/independent, isolated/continuous speech recognition, for small/large vocabulary. Speech recognition systems can be separated in several different classes by describing what types of utterances they have the ability to recognize. These classes are based on the fact that one of the difficulties of ASR is the ability to determine when a speaker starts and finishes an utterance. Most packages can fit into more than one class, depending on which mode they're using.

Isolated Words

Isolated word recognizers usually require each utterance to have quiet (lack of an audio signal) on BOTH sides of the sample window. It doesn't mean that it accepts single words, but does require a single utterance at a time. Often, these systems have "Listen/Not-Listen" states, where they require the speaker to wait between utterances (usually doing processing during the pauses). Isolated Utterance might be a better name for this class.

Connected Words

Connect word systems (or more correctly 'connected utterances') are similar to Isolated words, but allow separate utterances to be 'run-together' with a minimal pause between them.

Continuous Speech

Continuous recognition is the next step. Recognizers with continuous speech capabilities are some of the most difficult to create because they must utilize special methods to determine utterance boundaries. Continuous speech recognizers allow users to speak almost naturally, while the computer determines the content. Basically, it's computer dictation.

Spontaneous Speech

There appears to be a variety of definitions for what spontaneous speech actually is. At a basic level, it can be thought of as speech that is natural sounding and not rehearsed. An ASR system with spontaneous speech ability should be able to handle a variety of natural speech features such as words being run together, "ums" and "ahs", and even slight stutters.

Speaker Dependence

ASR engines can be classified as speaker dependent and speaker independent. Speaker Dependent systems are trained with one speaker and recognition is done only for that speaker. Speaker Independent systems are trained with one set of speakers. This is obviously much more complex than speaker dependent recognition. A problem of intermediate complexity would be to train with a group of speakers and recognize speech of a speaker within that group. We could call this speaker group dependent recognition.

2.4 WHY IS AUTOMATIC SPEECH RECOGNITION HARD?

There are a few problems in speech recognition that haven't yet been discovered. However there are a number of problems that have been identified over the past few decades most of which still remain unsolved. Some of the main problems in ASR are:

Determining word boundaries

Speech is usually continuous in nature and word boundaries are not clearly defined. One of the common errors in continuous speech recognition is the missing out of a minuscule gap between words. This happens when the speaker is speaking at a high speed.

Varying Accents

People from different parts of the world pronounce words differently. This leads to errors in ASR. However this is one problem that is not restricted to ASR but which plagues human listeners too.

Large vocabularies

When the number of words in the database is large, similar sounding words tend to cause a high amount of error i.e. there is a good probability that one word is recognized as the other.

Changing Room Acoustics

Noise is a major factor in ASR. In fact it is in noisy conditions or in changing room acoustic that the limitations of present day ASR engines become prominent.

Temporal Variance

Different speakers speak at different speeds. Present day ASR engines just cannot adapt to that.

2.5 SPEECH ANALYZER

Speech analysis, also referred to as front-end analysis or feature extraction, is the first step in an automatic speech recognition system. This process aims to extract acoustic features from the speech waveform. The output of front-end analysis is a compact, efficient set of parameters that represent the acoustic properties observed from input speech signals, for subsequent utilization by acoustic modeling.

There are three major types of front-end processing techniques, namely linear predictive coding (LPC), mel-frequency cepstral coefficients (MFCC), and perceptual linear prediction (PLP), where the latter two are most commonly used in state-of-the-art ASR systems.

Linear predictive coding

LPC starts with the assumption that a speech signal is produced by a buzzer at the end of a tube (voiced sounds), with occasional added hissing and popping sounds. Although apparently crude, this model is actually a close approximation to the reality of speech production. The glottis (the space between the vocal cords) produces the buzz, which is characterized by its intensity (loudness) and frequency (pitch). The vocal tract (the throat and mouth) forms the tube, which is characterized by its resonances, which are called formants. Hisses and pops are generated by the action of the tongue, lips and throat during sibilants and plosives.

LPC analyzes the speech signal by estimating the formants, removing their effects from the speech signal, and estimating the intensity and frequency of the remaining buzz. The process of removing the formants is called inverse filtering, and the remaining signal after the subtraction of the filtered modeled signal is called the residue.

The numbers which describe the intensity and frequency of the buzz, the formants, and the residue signal, can be stored or transmitted somewhere else. LPC synthesizes the speech signal by reversing the process: use the buzz parameters and the residue to create a source signal, use the formants to create a filter (which represents the tube), and run the source through the filter, resulting in speech.

Because speech signals vary with time, this process is done on short chunks of the speech signal, which are called frames; generally 30 to 50 frames per second give intelligible speech with good compression.

Mel Frequency Cepstrum Coefficients

These are derived from a type of cepstral representation of the audio clip (a "spectrum-of-a-spectrum"). The difference between the cepstrum and the Mel-frequency cepstrum is that in the MFC, the frequency bands are positioned logarithmically (on the mel scale) which approximates the human auditory system's response more closely than the linearly-spaced frequency bands obtained directly from the FFT or DCT. This can allow for better processing of data, for example, in audio compression. However, unlike the sonogram, MFCCs lack an outer ear model and, hence, cannot represent perceived loudness accurately.

MFCCs are commonly derived as follows:

1. Take the Fourier transform of (a windowed excerpt of) a signal
2. Map the log amplitudes of the spectrum obtained above onto the Mel scale, using triangular overlapping windows.
3. Take the Discrete Cosine Transform of the list of Mel log-amplitudes, as if it were a signal.
4. The MFCCs are the amplitudes of the resulting spectrum.

Perceptual Linear Prediction

Perceptual linear prediction, similar to LPC analysis, is based on the short-term spectrum of speech. In contrast to pure linear predictive analysis of speech, perceptual linear prediction (PLP) modifies the short-term spectrum of the speech by several psychophysically based transformations

This technique uses three concepts from the psychophysics of hearing to derive an estimate of the auditory spectrum:

(1) The critical-band spectral resolution,

(2) The equal-loudness curve, and

(3) The intensity-loudness power law.

The auditory spectrum is then approximated by an autoregressive all-pole model. In comparison with conventional linear predictive (LP) analysis, PLP analysis is more consistent with human hearing.

2.6 SPEECH CLASSIFIER

The problem of ASR belongs to a much broader topic in scientific and engineering so called *pattern recognition*. The goal of pattern recognition is to classify objects of interest into one of a number of categories or classes. The objects of interest are generically called *patterns* and in our case are sequences of acoustic vectors that are extracted from an input speech using the techniques described in the previous section. The classes here refer to individual speakers. Since the classification procedure in our case is applied on extracted features, it can be also referred to as *feature matching*.

The state-of-the-art in feature matching techniques used in speaker recognition includes Dynamic Time Warping (DTW), Hidden Markov Modeling (HMM), and Vector Quantization (VQ).

Dynamic Time Warping

Dynamic time warping is an algorithm for measuring similarity between two sequences which may vary in time or speed. For instance, similarities in walking patterns would be detected, even if in one video the person was walking slowly and if in another they were walking more quickly, or even if there were accelerations and decelerations during the course of one observation. DTW has been applied to video, audio, and graphics -indeed, any data which can be turned into a linear representation can be analyzed with DTW.

A well known application has been automatic speech recognition, to cope with different speaking speeds. In general, it is a method that allows a computer to find an optimal match between two given sequences (e.g. time series) with certain restrictions, i.e. the sequences are "warped" non-linearly to match each other. This sequence alignment method is often used in the context of hidden Markov models.

Hidden Markov Model

The basic principle here is to characterize words into probabilistic models wherein the various phonemes which contribute to the word represent the states of the HMM while the transition probabilities would be the probability of the next phoneme being uttered (ideally 1.0). Models for the words which are part of the vocabulary are created in the training phase.

Now, in the recognition phase when the user utters a word it is split up into phonemes as done before and it's HMM is created. After the utterance of a particular phoneme, the most probable phoneme to follow is found from the models which had been created by comparing it with the newly formed model. This chain from one phoneme to another continues and finally at some point we have the most probable word out of the stored words which the user would have uttered and thus recognition is brought about in a finite vocabulary system. Such a probabilistic system would be more efficient than just cepstral analysis as these is some amount of flexibility in terms of how the words are uttered by the users.

Vector Quantization

VQ is a process of mapping vectors from a large vector space to a finite number of regions in that space. Each region is called a cluster and can be represented by its center called a centroid. The collection of all codewords is called a codebook.

Fig 2.2 shows a conceptual diagram to illustrate this recognition process. In the figure, only two speakers and two dimensions of the acoustic space are shown. The circles refer to the acoustic vectors from the speaker 1 while the triangles are from the speaker 2. In the training phase, a speaker-specific VQ codebook is generated for each known speaker by clustering his/her training acoustic vectors. The result codewords (centroids) are shown in Figure 5 by black circles and black triangles for speaker 1 and 2, respectively. The distance from a vector to the closest codeword of a codebook is called a VQ-distortion. In the recognition phase, an input utterance of an unknown voice is "vector-quantized" using each trained codebook and the total VQ distortion is computed. The speaker corresponding to the VQ codebook with smallest total distortion is identified.

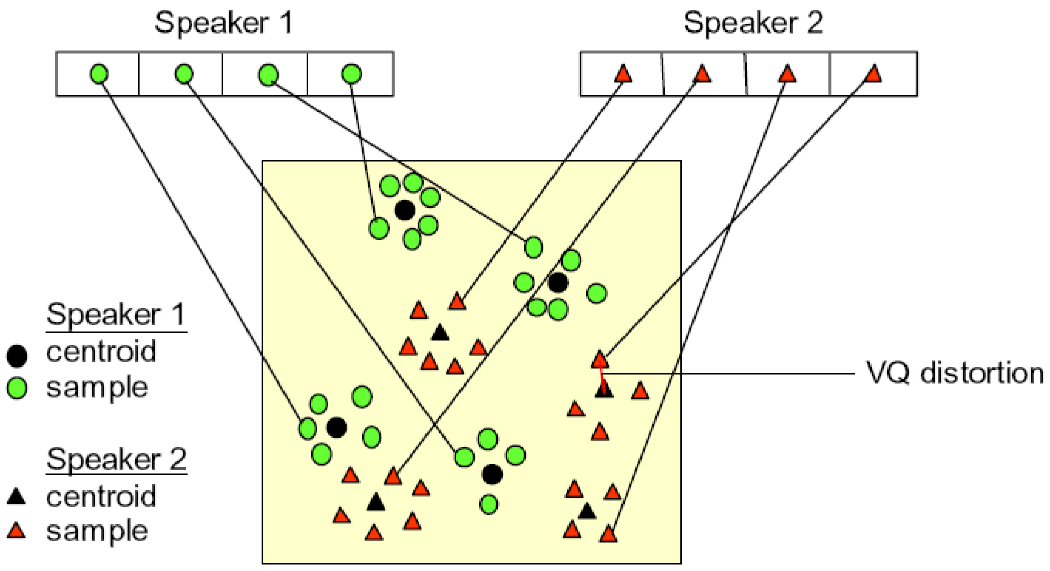


Figure 2.2: Conceptual diagram illustrating vector quantization codebook formation.

One speaker can be discriminated from another based of the location of centroids.

3. FEATURE EXTRACTION

3.1 PREPROCESSING

Firstly the input speech signal is pre-emphasized to artificially amplify the high frequencies. Speech signals are non-stationary signals, meaning to say the transfer function of the vocal tract; which generates it, changes with time, though it changes gradually. It is safe to assume that it is piecewise stationary. Thus the speech signal is framed with 30ms to 32ms frames with an overlap of 20ms. The need for the overlap is that information may be lost at the frame boundaries, so frame boundaries need to be within another frame. Mathematically, framing is equivalent to multiplying the signal with a series of sliding rectangular windows. The problem with rectangular windows is that the power contained in the side lobes is significantly high and therefore may give rise to spectral leakage. In order to avoid this we use a Hamming window given by.

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right), \quad 0 \le n \le N-1$$

3.2 MEL FREQUENCY CEPSTRAL COEFFICIENTS

MFCCs are typically computed by using a bank of triangular-shaped filters, with the center frequency of the filter spaced linearly for frequencies less than 1000 Hz and logarithmically above 1000 Hz. The bandwidth of each filter is determined by the center frequencies of the two adjacent filters and is dependent on the frequency range of the filter bank and number of filters chosen for design. But for the human auditory system it is estimated that the filters have a bandwidth that is related to the center frequency of the filter. Further it has been shown that there is no evidence of two regions (linear and logarithmic) in the experimentally determined Mel frequency scale.

Silence detection

This is a very important step in any front end speaker recognition system. When the user says out any word the system will never have control over the instant the word is uttered. It is for this reason that we need a silence detection step which basically determines when the user has actually started uttering the word any thereby translates the frame of reference to that instant.

Windowing

The next step in the processing is to window each individual frame so as to minimize the signal discontinuities at the beginning and end of the frame. The concept here is to minimize the spectral distortion by using the window to taper the signal to zero at the beginning and end of the frame. If we define the window as,

$$w(n),\ 0 \le n \le N-1,$$

where *N* is the number of samples in each frame, then the result of windowing is the signal

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

$$y_l(n) = x_l(n)w(n), \quad 0 \le n \le N-1$$

Typically the *Hamming* window is used, which has the form:

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right), \quad 0 \le n \le N-1$$

□□□□□□□□□□□□□□□□□□□□□□□□ **Fast Fourier Transform (FFT)**

The next processing step is the Fast Fourier Transform, which converts each frame of *N* samples from the time domain into the frequency domain. The FFT is a fast algorithm to implement the Discrete Fourier Transform (DFT) which is defined on the set of *N* samples { X_n }, as follows:

$$X_n = \sum_{k=0}^{N-1} x_k e^{-2\pi jkn/N}, \qquad n = 0,1,2,\ldots,N-1$$

Note that we use *j* here to denote the imaginary unit. In general X_n's are complex numbers. The resulting sequence {X_n} is interpreted as follows: the zero frequency corresponds to *n* = 0, positive frequencies: $0 < f < Fs/2$ correspond to values $1 \le n \le N/2-1$ while negative frequencies $-Fs/2 < f < 0$ correspond to $N/2+1 \le n \le N-1$.

Here, *Fs* denotes the sampling frequency. The result obtained after this step is often referred to as signal's *spectrum* or *periodogram.*

Mel-frequency Warping

As mentioned above, psychophysical studies have shown that human perception of the frequency contents of sounds for speech signals does not follow a linear scale. Thus for each tone with an actual frequency, *f*, measured in Hz, a subjective pitch is measured on a scale called the 'mel' scale. The *mel-frequency* scale is a linear frequency spacing below 1000 Hz and a logarithmic spacing above 1000 Hz. As a reference point, the pitch of a 1 kHz tone, 40 dB above the perceptual hearing threshold, is defined as 1000 mels. Therefore we can use the following approximate formula to compute the mels for a given frequency *f* in Hz:

$$mel(f) = 2595 * \log_{10}(1 + f / 700)$$

One approach to simulating the subjective spectrum is to use a filter bank, one filter for each desired mel-frequency component. That filter bank has a triangular bandpass frequency response, and the spacing as well as the bandwidth is determined by a constant mel-frequency interval. The modified spectrum of $S(\omega)$ thus consists of the output power of these filters when $S(\omega)$ is the input. The number of mel cepstral coefficients, *K*, is typically chosen as 20.

Note that this filter bank is applied in the frequency domain; therefore it simply amounts to taking those triangle-shape windows in the Fig 3.2.1 on the spectrum. A useful way of thinking about this mel-warped filter bank is to view each filter as a histogram bin (where bins have overlap) in the frequency domain. A useful and efficient way of implementing this is to consider these triangular filters in the Mel scale where they would in effect be equally spaced filters.

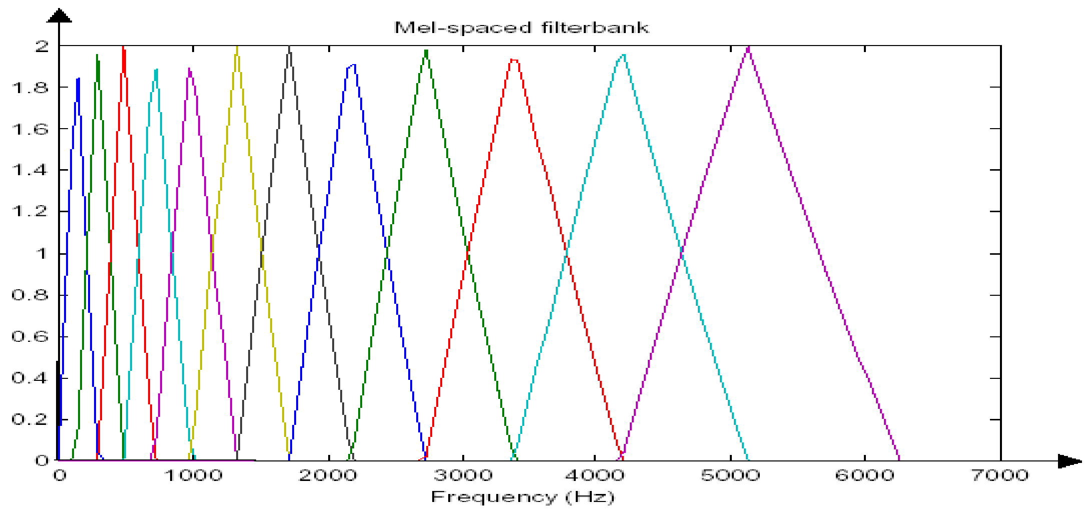


Figure 3.2.1: Filter Bank in linear frequency scale

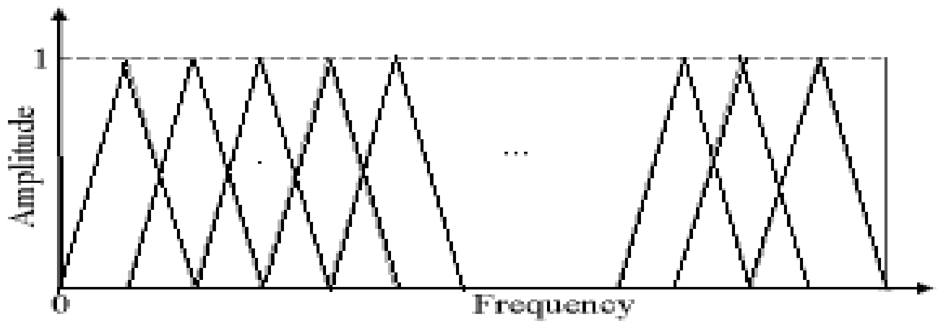


Figure 3.2.2: Filter Bank in Mel frequency scale

Cepstrum

In this final step, we convert the log mel spectrum back to time. The result is called the mel frequency cepstral coefficients (MFCC). The cepstral representation of the speech spectrum provides a good representation of the local spectral properties of the signal for the given frame analysis. Because the mel spectrum coefficients (and so their logarithm) are real numbers, we can convert them to the time domain using the Discrete Cosine Transform (DCT). Therefore if we denote those mel power spectrum coefficients that are the result of the last step are Sk , where *k* =1, 2,…..,K. Calculate the MFCC's, cn as,

$$\tilde{c}_n = \sum_{k=1}^{K} (\log \tilde{S}_k) \cos\left[n\left(k - \frac{1}{2}\right)\frac{\pi}{K}\right], \qquad n = 1,2,\ldots,K$$

Note that we exclude the first component C_0 from the DCT since it represents the mean value of the input signal which carried little speaker specific information.

Once we have the MFCCs, these characterize the particular word so during the training phase the coefficients for the word are determined and stored. During the recognition phase, the coefficients are again determined for the uttered word and recognition is carried out by analyzing the MSE with respect to the stored coefficients and defining an appropriate threshold depending on the level of security required for the application.

3.3 TIME DOMAIN ANALYSIS

Although speech is non-stationary in a true sense, study has shown that it can be assumed to be *quasi-stationary* i.e. slowly time varying. When examined over a sufficiently short period of time (between 20 and 40 msec), its characteristics are fairly stationary. The implication of this is that when we consider such small segments of speech they have a dominant frequency within the windowed segment. Therefore, this can be processed through a short-time frequency analysis. The figure below shows how we can use overlapping time domain windows so as to cater for the short time Fourier analysis.

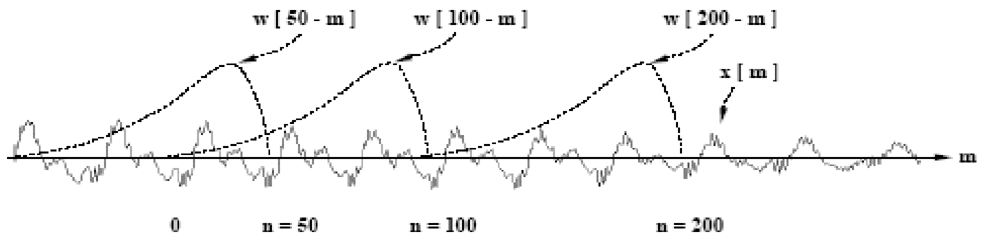


Figure 3.3: Overlapping windows in time domain

The short-time Fourier transform (STFT) of a speech signal *s*(*t*) is given by

$$S(f,t) = \int_{-\infty}^{\infty} s(\tau)w(t-\tau)e^{-j2\pi f\tau}d\tau$$

where, w(*t*) is a window function of duration *Tw*. In speech processing, the Hamming window function is typically used and its width *Tw* is normally 20 - 40 msec. We can decompose *S*(*f, t*) as follows:

$$S(f,t) = |S(f,t)|e^{j\psi(f,t)}$$

where $|S(f, t)|$ is the short-time magnitude spectrum and $\psi(f, t) = \angle S(f, t)$ is the short-time phase spectrum. Our project focuses on the short time magnitude spectrum as applied to the dominant frequency within each overlapping segment.

In the STFT-based system, there are a number of design issues that must be addressed. First, what type of window function $w(t)$ should be used for computing the STFT. A tapered window function (Hamming is usually preferred in speech applications) has been used in all studies because with the gradual roll off of such a window, the effect of overlap is kept under check to make sure that each window gives prominence to an independent segment of speech while maintaining a smooth transition from one segment of speech to another. Second, what should be the duration t*w*. The modifier 'short-time' implies a finite-time window over which the properties of speech may be assumed stationary; it does not refer to the actual duration of the window. However, it is known that speech basically comprises of phonemes, and experiments conducted matched with our inferences that Tw could be between 20 msec to 50 msec.

4. ALGORITHMS

4.1 MFCC APPROACH:

A block diagram of the structure of an MFCC processor is as shown in Fig 4.1.1. The speech input is typically recorded at a sampling rate above 10000 Hz. This sampling frequency was chosen to minimize the effects of *aliasing* in the analog-to-digital conversion. These sampled signals can capture all frequencies up to 5 kHz, which cover most energy of sounds that are generated by humans. The main purpose of the MFCC processor is to mimic the behavior of the human ears. In addition, rather than the speech waveforms themselves, MFCC's are shown to be less susceptible to mentioned variations.

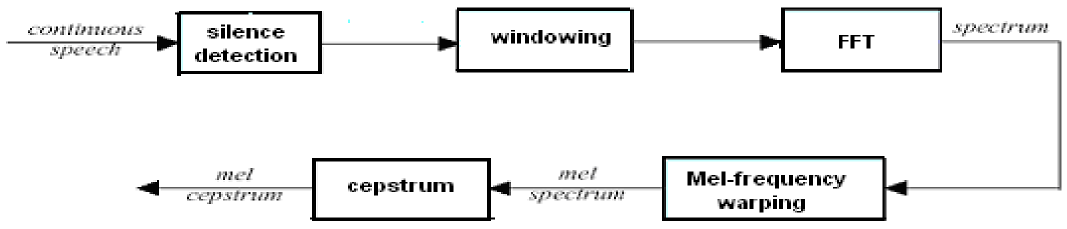


Figure 4.1.1

We first stored the speech signal as a 10000 sample vector. It was observed from our experiment that the actual uttered speech eliminating the static portions came up to about 2500 samples, so, by using a simple threshold technique we carried out the silence detection to extract the actual uttered speech.

It is clear that what we wanted to achieve was a voice based biometric system capable of recognizing isolated words. As our experiments revealed almost all the isolated words were uttered within 2500 samples. But, when we passed this speech signal through a MFCC processor, it spilt this up in the time domain by using overlapping windows each with about 250 samples. Thus when we convert this into the frequency domain we just have about 250 spectrum values under each window. This implied that converting it to the Mel scale would be redundant as the Mel scale is linear till 1000 Hz. So, we eliminated the block which did the Mel warping. We directly used the overlapping triangular windows in the frequency domain. We obtained the energy within each triangular window, followed by the DCT of their logarithms to achieve good compaction within a small number of coefficients as described by the MFCC approach.

This algorithm however, has a drawback. As explained earlier the key to this approach is using the energies within each triangular window, however, this may not be the best approach as was discovered. It was seen from the experiments that because of the prominence given to energy, this approach failed to recognize the same word uttered with different energy. Also, as this takes the summation of the energy within each triangular window it would essentially give the same value of energy irrespective of whether the spectrum peaks at one particular frequency and falls to lower values around it or whether it has an equal spread within the window. This is why we decided not to go ahead with the implementation of the MFCC approach.

The simulation was carried out in MATLAB. The various stages of the simulation have been represented in the form of the plots shown. The input continuous speech signal considered as an example for this project is the word "HELLO".

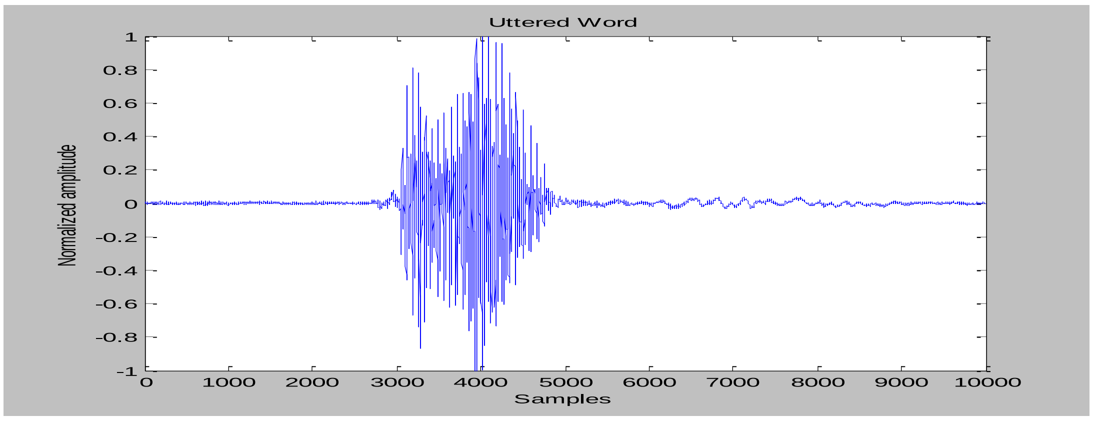


Figure 4.1.2: The word "Hello" taken for analysis

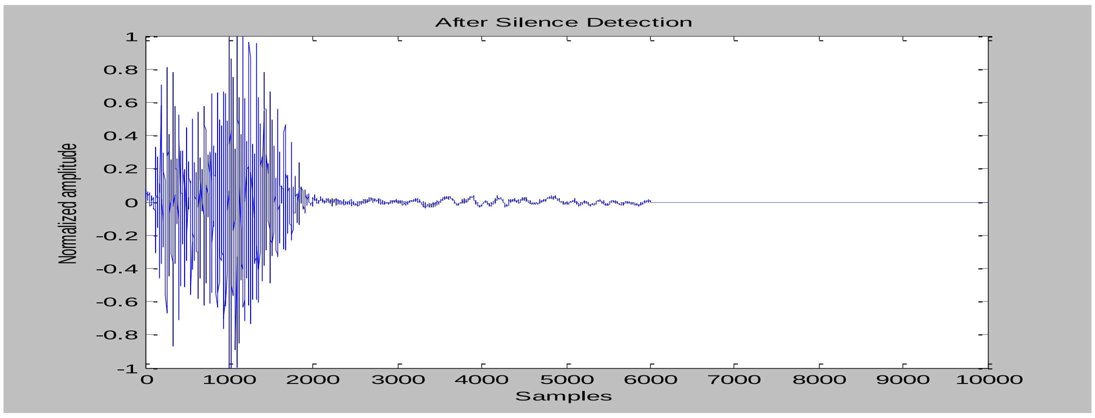


Figure 4.1.3: The word "Hello" after Silence Detection

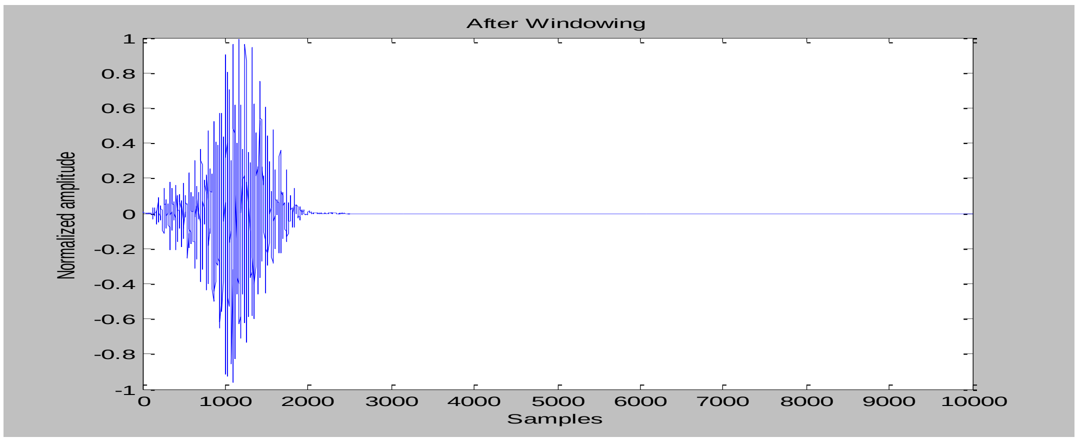


Figure 4.1.4: The word “Hello” after Windowing using a Hamming Window

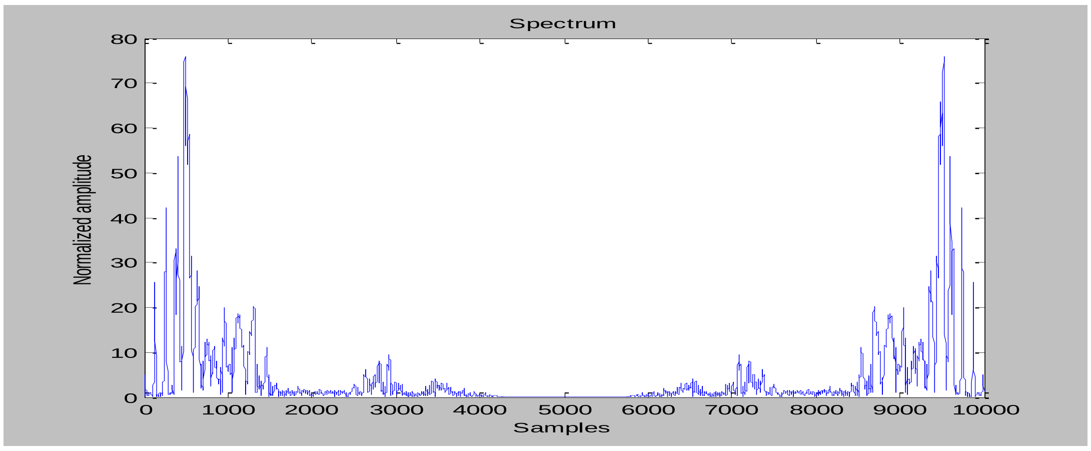


Figure 4.1.5: The Fourier Transform of the word “Hello” after pre processing

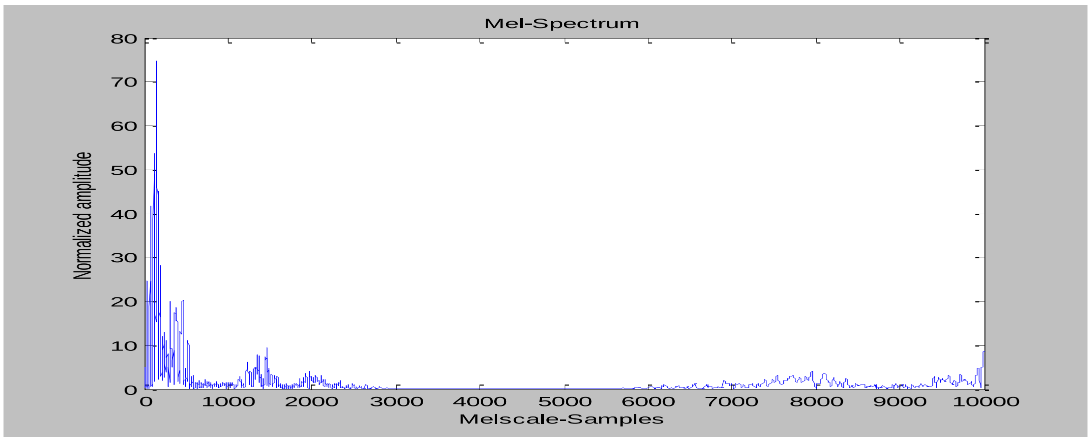
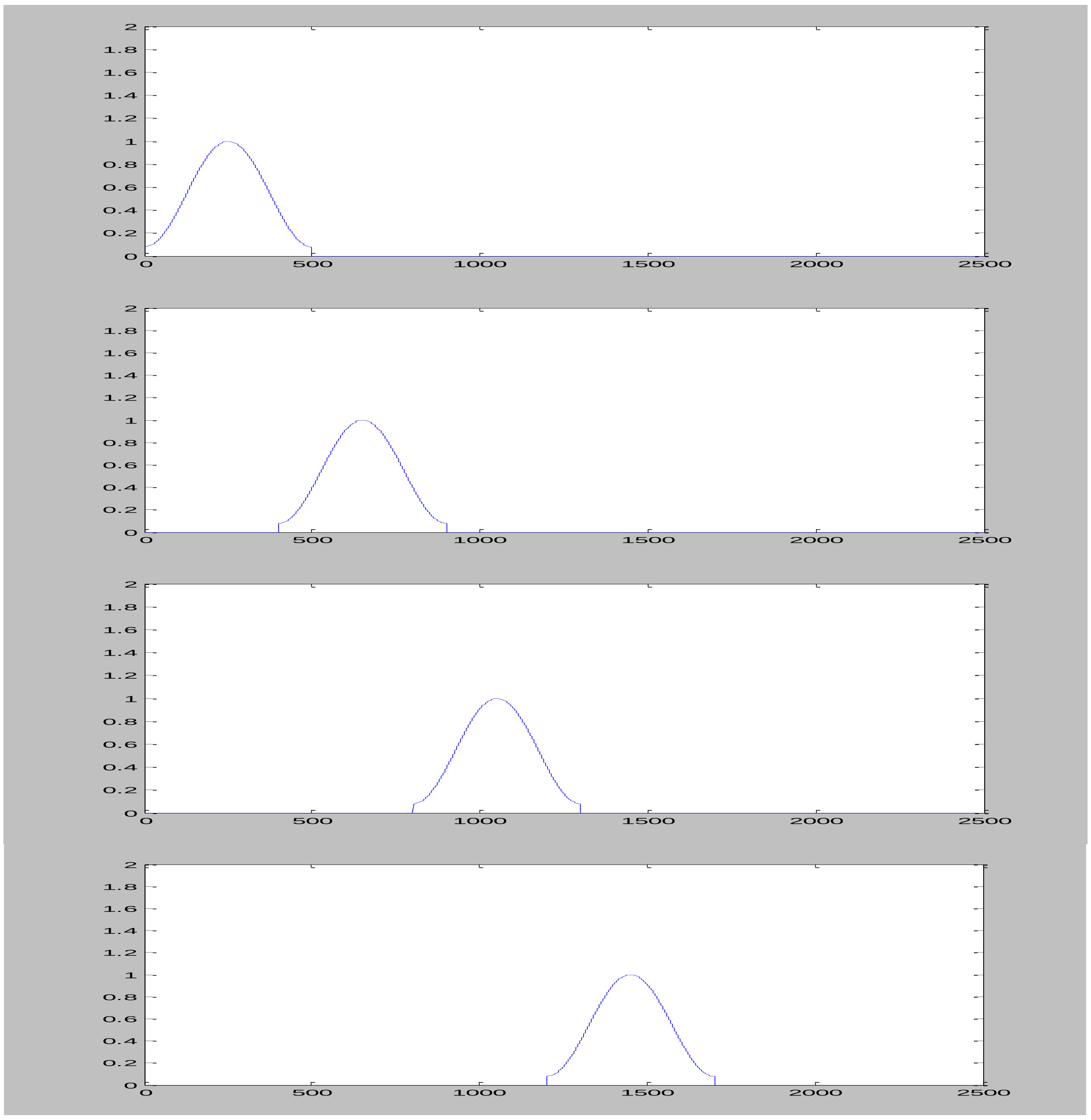


Figure 4.1.6: The spectrum of the word “Hello” after Mel – Warping

4.2 TIME DOMAIN APPROACH

The essence of STFT as explained is to extract the frequency spectrum within each window in the time domain to analyze the dominating frequencies within each window corresponding to various phonemes. In our project, instead of directly applying the STFT equation, we have used a hamming window to extract out segments of the speech and within each of these segments we used FFT to obtain the desired frequency spectrum.

The plots in Fig 4.2.1 show an example of how overlapping time domain hamming windows could be represented. The example considers hamming windows of width 500 samples which is about 50 ms and an overlap of 100 samples which would be around 10 msec. This has been modified in our project to achieve 25 msec window width and an overlap of 10 msec as required.



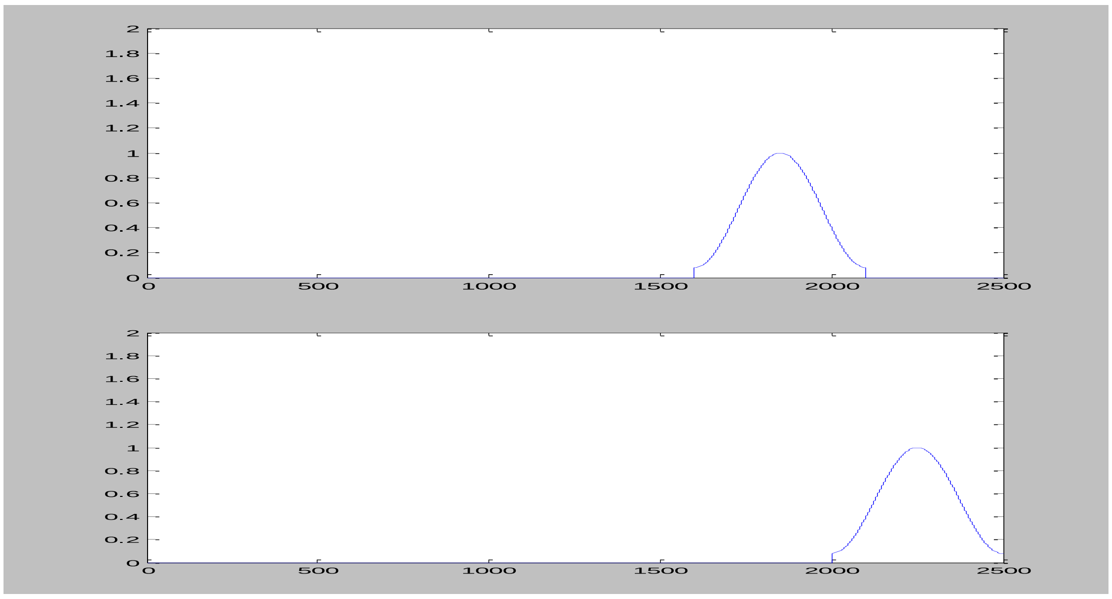


Figure 4.2.1: Plots of overlapping time domain hamming windows

The system we have simulated and finally implemented can be described using the block diagram shown in Fig 4.2.2.

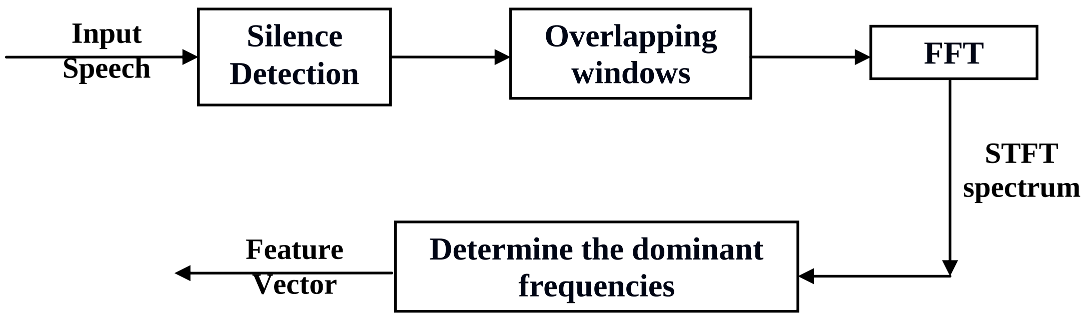


Figure 4.2.2: Block diagram of the time domain approach

We have considered a single word recognizer in our project and thus we have separated out about 2500 samples which represented the speech data. We tested the system with many different words and found that almost all the words were uttered within 2500 samples as explained earlier. By using a simple threshold algorithm as explained in the MFCC approach we separated out the required 2500 speech samples. Now, over these samples overlapping hamming windows were used to obtain the different segments. The plots below show an example of such windows but the scheme of windowing we used for the project consisted of 12 overlapping hamming windows, each of duration about 25 msec and an overlap of about 10 msec as required.

Once these 12 segments were obtained, we found the spectrum of each segment using the standard FFT algorithm. An important observation which helped us take up the Time domain approach for our project was the plots of the spectrum. As explained earlier, the windowing in time domain helps split the word to its phonemes and bring in the quasi – stationary characteristic of speech. This implies that the segments of speech under each window would have a particular dominant frequency due to the quasi - stationary property. This was clearly observed in all our tests i.e. the spectrum of a segment under a window for a particular word would always peak around the same frequency. This dominant frequency characteristic was the key to our project.

The dominant frequencies obtained for each of the 12 windows are stored in the training phase. In the testing phase, when the user again utters the word the process is repeated and the dominant frequency for each overlapping window is found. This is compared with the values stored in the training phase and if there are a certain number of close matches, access is granted. If not access is denied.

There are a number of classifiers which can be used once we have obtained the feature vectors i.e. once we have stored the vector during the training phase and we obtain a new vector in the testing phase and thus we need a method of classifying this as a match or not. Some of the classifiers are as described earlier like GMMs, HMMs. The key to this is what is called vector quantization.

A plot of the extracted speech samples and hence the output of an overlapping window showing various stages of the MATLAB simulation is shown below. Fig 4.2.3 shows the stored speech signal. In this experiment, the word “HELLO” was uttered. In Fig 4.2.4, the outcome of separating out the required 2500 speech samples is shown followed by, an output of one of the overlapping time domain hamming windows. In Fig 4.2.5 and hence in Fig 4.2.6, the effect of dominant frequency is clearly observed.

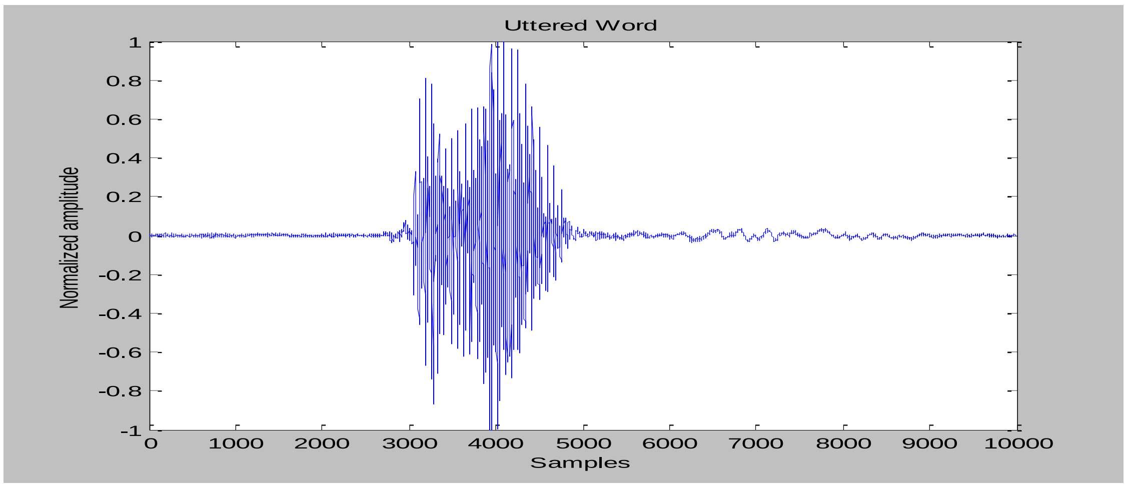


Figure 4.2.3: Input speech “HELLO”

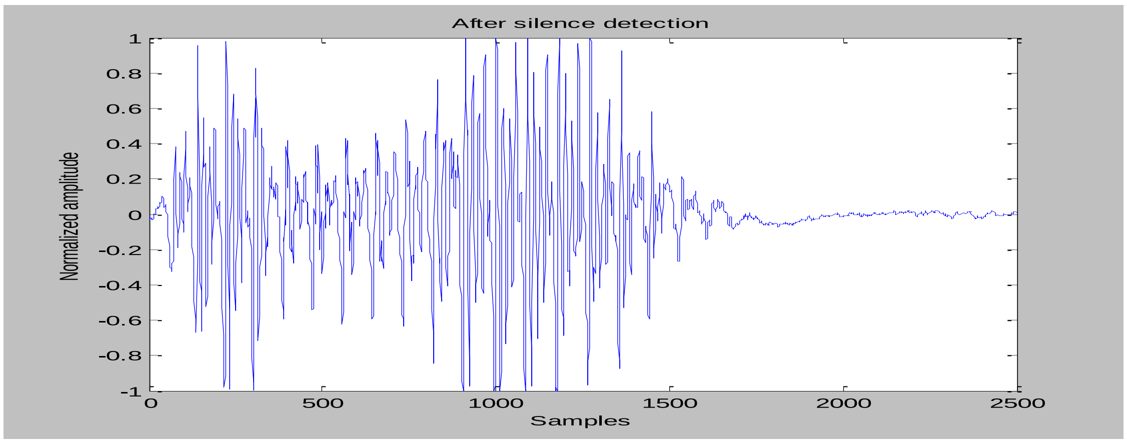


Figure 4.2.4: 2500 speech samples extracted

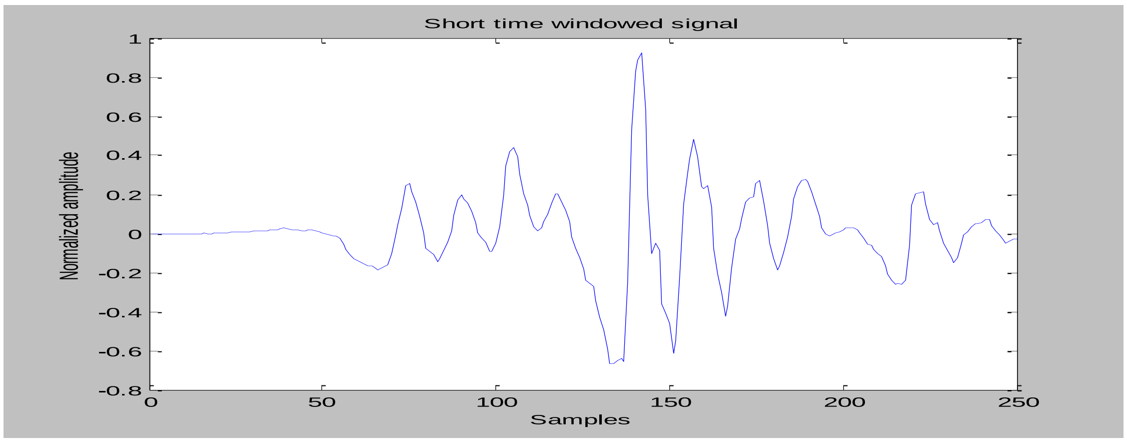


Figure 4.2.5: Output of one of the overlapping windows

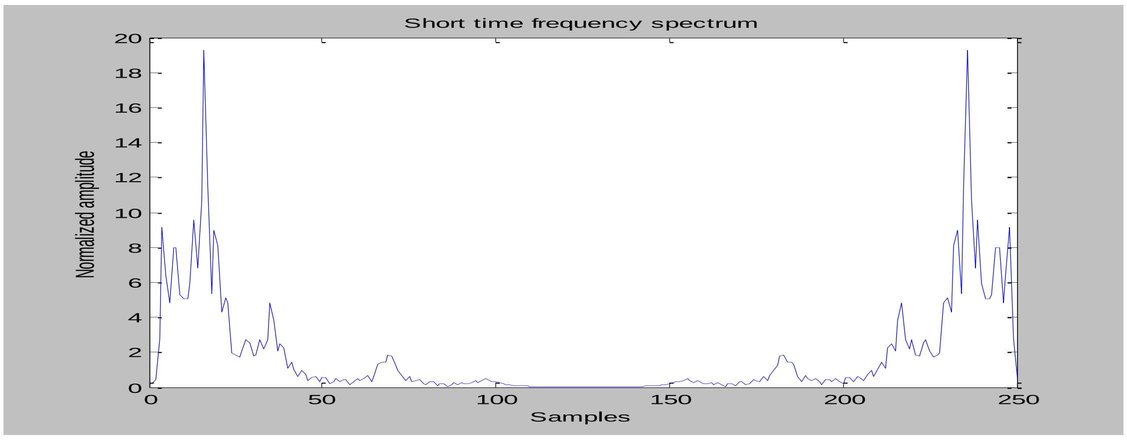


Figure 4.2.6: The frequency spectrum of the windowed signal showing the dominant frequency

As the time domain approach proved to be more effective and appropriate from our simulations using MATLAB, we have implemented the time domain approach on the DSK.

5. HARDWARE IMPLEMENTATION ON TMS320C6713

5.1 DIGITAL SIGNAL PROCESSORS

For embedded systems, the cost of speech recognition can be negligible comparing to the total system cost for additional speech module to the product. With the advances in semiconductor technologies, a number of mainstream Integrated Circuit (IC) products which use digital circuit techniques had emerged. The digital revolution not only increases circuit robustness, cost efficiency, system integration, and flexibility, but also enables a continuous stream of completely new consumer applications, increasingly based on embedded (re)programmable processor cores like Digital Signal Processors (DSPs), and Field-Programmable Gate Arrays (FPGA) chips and un-programmable processors like Application-specific ICs There are many principles such as flexibility, scalability, reusability, portability and short development times for selecting the right hardware for the embedded speech recognition engine.

The following is the comparison of some alternatives available for digital signal processing with DSP, which is the hardware applied in this project:

FPGA

Field-Programmable Gate Arrays have the capability of being reconfigurable within a system, offering reasonably fast time to market. However, FPGAs are significantly more expensive and typically have much higher power dissipation than DSPs with similar functionality. The ASIC alternative Application-specific ICs be used to perform specific functions extremely well, and can be made quite power efficient. However, since ASICs are not field reprogrammable, they can not be changed after development. Consequently, every new version of the product requires a redesign and has to be re-produced. This is a big impediment to rapid time-to-market.

Programmable DSPs

On the other hand, programmable DSPs can merely change the software program without changing the silicon, which greatly reduces the development cost and makes aftermarket feature enhancements possible with mere code downloads. Furthermore, more often than not, in real time signal processing applications, ASICs are typically employed as bus interfaces, glue logic, and functional accelerators for a programmable DSP-based system. According to the above description on various popular embedded systems nowadays, the reason of choosing DSP as the hardware platform for our research is quite obvious:

- Single-cycle multiply-accumulate operation
- Real-time performance, simulation and emulation
- Flexibility
- Reliability
- Increased system performance
- Reduced system cost.

Applications of DSP:

- Classic signal processing: digital filtering, adaptive filtering, FFT, spectrum analysis
- Speech processing: speech coding, speech synthesize, speech recognition, speech enhancement, speech mail, speech storage
- Image processing: image compressing/transferring, image recognition, animation, image enhancement
- Military electronics, Automatic control; Consumer electronics etc..

The general-purpose digital signal processor is dominated by applications in communications (cellular).Applications embedded digital signal processors are dominated by consumer products. They are found in cellular phones, fax/modems, disk drives, radio, printers, hearing aids, MP3 players, high-definition television (HDTV), digital cameras, and so on. These processors have become the products of choice for a number of consumer applications, since they have become very cost-effective. They can handle different tasks, since they can be reprogrammed readily for a different application.

DSP techniques have been very successful because of the development of low-cost software and hardware support. For example, modems and speech recognition can be less expensive using DSP techniques.

DSP processors are concerned primarily with real-time signal processing. Real-time processing requires the processing to keep pace with some external event, whereas non-real-time processing has no such timing constraint. The external event to keep pace with is usually the analog input whereas analog-based systems with discrete electronic components such as resistors can be more sensitive to temperature changes, DSP-based systems are less affected by environmental conditions. DSP processors enjoy the advantages of microprocessors. They are easy to use, flexible, and economical.

5.2 ARCHITECTURAL DETAILS OF TMS320C6713

5.2.1 TMS320C6713 Digital Signal Processor

The TMS320C6713 (C6713) is based on the VelociTI^{TM} , an advanced VLIW* architecture (Very Long Instruction Word), which is very well suited for numerically intensive algorithms. The TMS320C6x (C6x) family of processors are like fast special-purpose microprocessors with a specialized type of architecture and an instruction set appropriate for signal processing. The internal program memory is structured so that a total of eight instructions can be fetched every cycle. For example, with a clock rate of 225MHz, the C6713 is capable of fetching eight 32-bit instructions every 1/ (225 MHz) or 4.44 ns.

***Very Long Instruction Word** or **VLIW** refers to a CPU architectural approach to taking advantage of instruction level parallelism (ILP). A processor that executes every instruction one after the other (i.e. a non-pipelined scalar architecture) may use processor resources inefficiently, potentially leading to poor performance. The performance can be improved by executing different sub-steps of sequential instructions simultaneously (this is *pipelining*), or even executing multiple instructions entirely simultaneously as in superscalar architectures. Further improvement can be realized by executing instructions in an order different from the order they appear in the program; this is called out-of-order execution

These three techniques all come at a cost: increased hardware complexity. Before executing any operations in parallel, the processor must verify that the instructions do not have interdependencies. There are many types of interdependencies, but a simple example would be a program in which the first instruction's result is used as an input for the second instruction. They clearly cannot execute at the same time, and the second instruction can't be executed before the first. Modern out-of-order processors use significant resources in order to take advantage of these techniques, since the scheduling of instructions must be determined dynamically as a program executes based on dependencies.

The VLIW approach, on the other hand, executes operation in parallel based on a fixed schedule determined when programs are compiled. Since determining the order of execution of operations (including which operations can execute simultaneously) is handled by the compiler, the processor does not need the scheduling hardware that the three techniques described above require. As a result, VLIW CPUs offer significant computational power with less hardware complexity (but greater compiler complexity) than is associated with most superscalar CPUs

5.2.2 Architectural Details of the TMS320C6713

The features of the TMS320C6713 are

- Highest-Performance Floating-Point Digital Signal Processor (DSP): C6713
    - Eight 32-Bit Instructions/Cycle
    - 32/64-Bit Data Word
    - 225-, 200-MHz (GDP), and 200-, 167-MHz (PYP) Clock Rates
    - 4.4-, 5-, 6-Instruction Cycle Times
    - 1800/1350, 1600/12 00, and 1336/1000 MIPS /MFLOPS
    - Rich Peripheral Set, Optimized for Audio
    - Highly Optimized C/C++ Compiler
    - Extended Temperature Devices Available
- Advanced Very Long Instruction Word (VLIW) TMS320C67x™ DSP Core
    - Eight Independent Functional Units:
        - Two ALUs (Fixed-Point)
        - Four ALUs (Floating- and Fixed-Point)
        - Two Multipliers (Floating- and Fixed-Point)
    - Load-Store Architecture With 32 32-Bit General-Purpose Registers
    - Instruction Packing Reduces Code Size
    - All Instructions Conditional
- Instruction Set Features
    - Native Instructions for IEEE 754
        - Single- and Double-Precision
    - Byte-Addressable (8-, 16-, 32-Bit Data)
    - 8-Bit Overflow Protection
    - Saturation; Bit-Field Extract, Set, Clear; Bit-Counting; Normalization
- L1/L2 Memory Architecture
    - 4K-Byte L1P Program Cache (Direct-Mapped)

    - 4K-Byte L1D Data Cache (2-Way)
    - 256K-Byte L2 Memory Total: 64K-Byte L2 Unified Cache/Mapped RAM, and 192K-Byte Additional L2 Mapped RAM
- Device Configuration
    - Boot Mode: HPI, 8-, 16-, 32-Bit ROM Boot
    - Endianness: Little Endian, Big Endian
- 32-Bit External Memory Interface (EMIF)
    - Glueless Interface to SRAM, EPROM, Flash, SBSRAM, and SDRAM
    - 512M-Byte Total Addressable External Memory Space
- Enhanced Direct-Memory-Access (EDMA) Controller (16 Independent Channels)
- 16-Bit Host-Port Interface (HPI)
- Two Multichannel Audio Serial Ports (McASPs)
    - Two Independent Clock Zones Each (1 TX and 1 RX)
    - Eight Serial Data Pins Per Port: Individually Assignable to any of the Clock Zones
    - Each Clock Zone Includes:
        - Programmable Clock Generator
        - Programmable Frame Sync Generator
        - TDM Streams From 2-32 Time Slots
        - Support for Slot Size: 8, 12, 16, 20, 24, 28, 32 Bits
        - Data Formatter for Bit Manipulation
    - Wide Variety of I2S and Similar Bit Stream Formats
    - Integrated Digital Audio Interface Transmitter (DIT) Supports:
        - S/PDIF, IEC60958-1, AES-3, CP-430 Formats
        - Up to 16 transmit pins
        - Enhanced Channel Status/User Data
    - Extensive Error Checking and Recovery
- Two Inter-Integrated Circuit Bus (I2C Bus™) Multi-Master and Slave Interfaces
- Two Multichannel Buffered Serial Ports:
    - Serial-Peripheral-Interface (SPI)
    - High-Speed TDM Interface

    - AC97 Interface
- Two 32-Bit General-Purpose Timers
- Dedicated GPIO Module With 16 pins (External Interrupt Capable)
- Flexible Phase-Locked-Loop (PLL) Based Clock Generator Module
- IEEE-1149.1 (JTAG) Boundary-Scan-Compatible
- 208-Pin PowerPAD™ Plastic (Low-Profile) Quad Flatpack (PYP)
- 272-BGA Packages (GDP and ZDP)
- 0.13-µm/6-Level Copper Metal Process
    - CMOS Technology
- 3.3-V I/Os, 1.2 V Internal (GDP & PYP)

5.2.3 CPU (DSP core) description

The TMS320C6713/13B floating-point digital signal processor is based on the C67x CPU. The CPU fetches advanced very-long instruction words (VLIW) (256 bits wide) to supply up to eight 32-bit instructions to the eight functional units during every clock cycle. The VLIW architecture features controls by which all eight units do not have to be supplied with instructions if they are not ready to execute. The first bit of every 32-bit instruction determines if the next instruction belongs to the same execute packet as the previous instruction, or whether it should be executed in the following clock as a part of the next execute packet. Fetch packets are always 256 bits wide; however, the execute packets can vary in size. The variable-length execute packets are a key memory-saving feature, distinguishing the C67x CPU from other VLIW architectures.

The CPU features two sets of functional units. Each set contains four units and a register file. One set contains functional units, .L1, .S1, .M1, and .D1; the other set contains units .D2, .M2, .S2, and .L2. The two register files each contain 16 32-bit registers for a total of 32 general-purpose registers. The two sets of functional units, along with two register files, compose sides A and B of the CPU. The four functional units on each side of the CPU can freely share the 16 registers belonging to that side. Additionally, each side features a single data bus connected to all the registers on the other side, by which the two sets of functional units can access data from the register files on the opposite side. While register access by functional units on the same side of the CPU as the register file can service all the units in a single clock cycle, register access using the register file across the CPU supports one read and one write per cycle.

Another key feature of the C67x CPU is the load/store architecture, where all instructions operate on registers (as opposed to data in memory). Two sets of data-addressing units (.D1 and .D2) are responsible for all data transfers between the register files and the memory.

The core of the **TMS320C6713** is shown in Fig 5.2.3

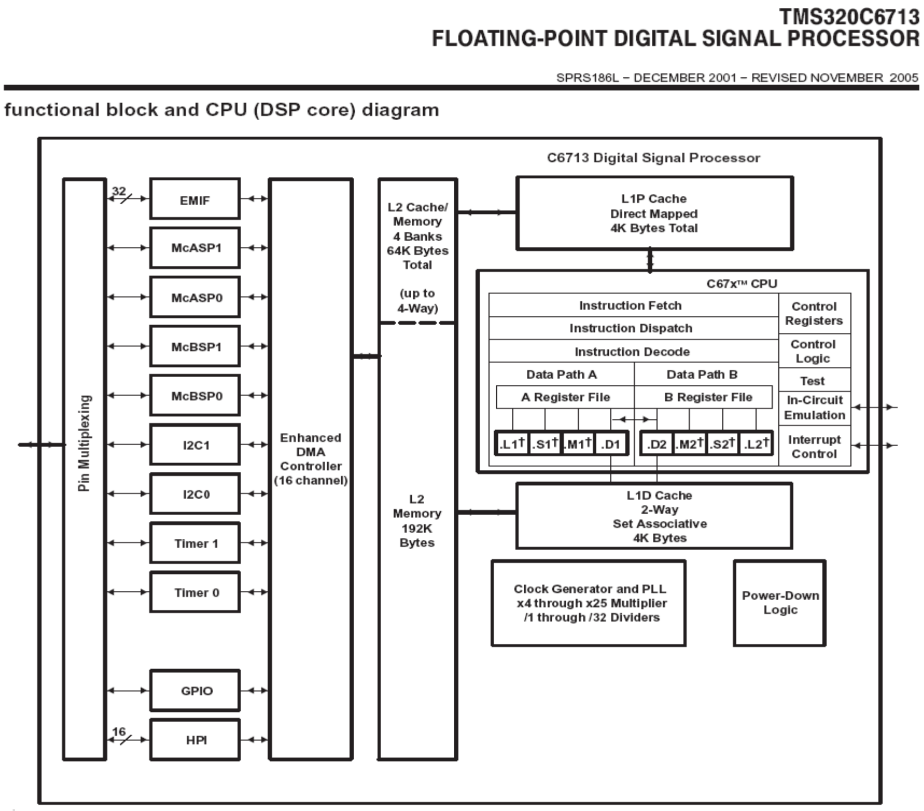

TMS320C6713
FLOATING-POINT DIGITAL SIGNAL PROCESSOR
SPRS186L – DECEMBER 2001 – REVISED NOVEMBER 2005
functional block and CPU (DSP core) diagram
C6713 Digital Signal Processor
Pin Multiplexing
32
EMIF
McASP1
McASP0
McBSP1
McBSP0
I2C1
I2C0
Timer 1
Timer 0
GPIO
16
HPI
Enhanced DMA Controller (16 channel)
L2 Cache/ Memory 4 Banks 64K Bytes Total (up to 4-Way)
L2 Memory 192K Bytes
L1P Cache Direct Mapped 4K Bytes Total
C67x™ CPU
Instruction Fetch
Instruction Dispatch
Instruction Decode
Data Path A
Data Path B
A Register File
B Register File
.L1† .S1† .M1† .D1
.D2 .M2† .S2† .L2†
Control Registers
Control Logic
Test
In-Circuit Emulation
Interrupt Control
L1D Cache 2-Way Set Associative 4K Bytes
Clock Generator and PLL x4 through x25 Multiplier /1 through /32 Dividers
Power-Down Logic

Figure 5.2.3: TMS320C6713 core

5.3 THE 6713 DSK INTERFACE

The implementation of signal processing algorithms typically follows the flow given below

Simulation using tools like MATLAB, Octave Lab view

$\Downarrow$

Development of a script in C/C++

$\Downarrow$

Use of an Integrated Development Environment (IDE)
like Code Composer Studio (CCS) to compile and link

$\Downarrow$

The assembly code (.asm) for the C code is built using CCS and
this is downloaded onto the target processor

DSP Development System

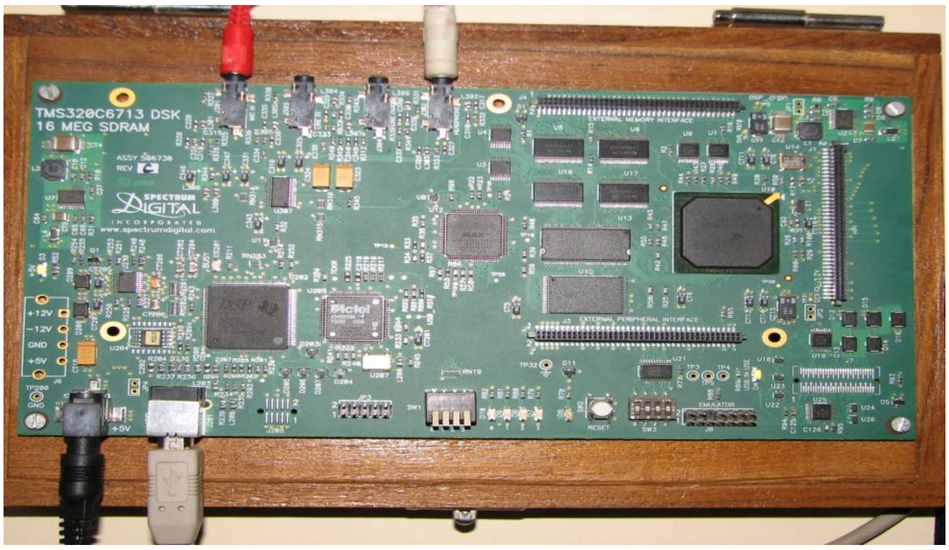

TMS320C6713 DSK
16 MEG SDRAM
SPECTRUM DIGITAL INCORPORATED
www.spectrumdigital.com
EXTERNAL MEMORY INTERFACE
EXTERNAL PERIPHERAL INTERFACE
+12V
-12V
GND
+5V
RESET
EMULATOR

Figure 5.3.1: DSP Development System

The DSP development system shown in Fig 2(a) contains the TMS320C6713 (C6713) floating-point digital signal processor as well as a 32-bit stereo codec for input and output (I/O) support, a universal synchronous bus (USB) cable that connects the

DSK board to a PC, 5V power supply for the DSK board and an *IBM-compatible PC*. The DSK board connects to the USB port of the PC through the USB cable included with the DSK package.

5.3.1 DSK Development Board

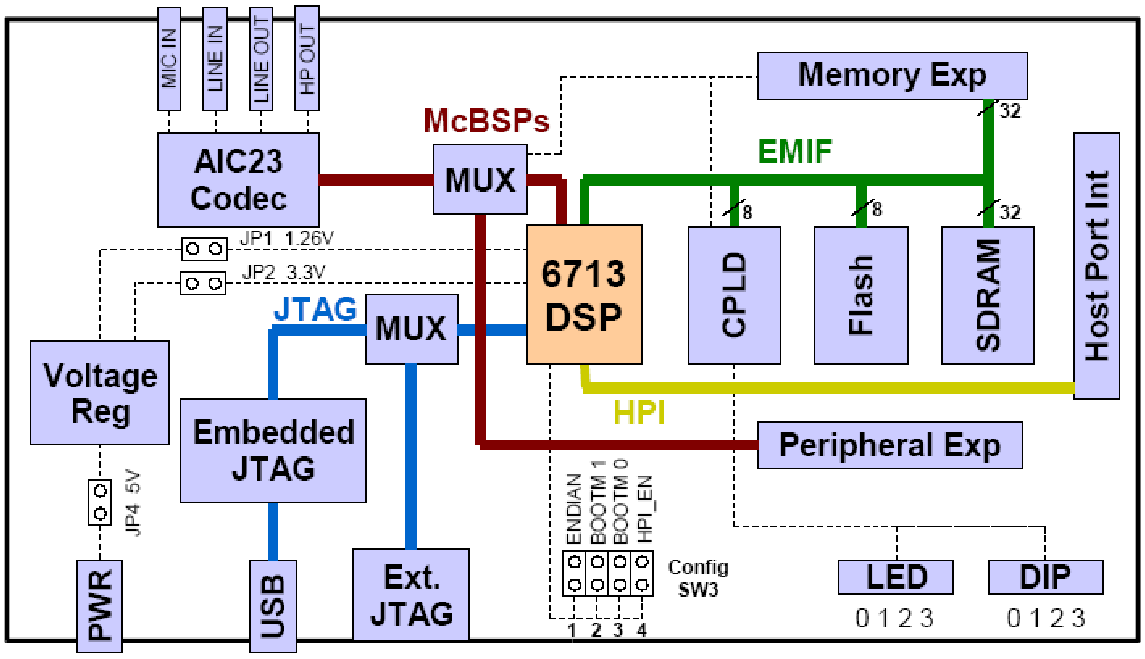


Figure 5.3.2: TMS320C6713-based DSK board (Courtesy of Texas Instruments)

The DSK package contains hardware and software support tools for real-time signal processing. It is a complete DSP system. The DSK board, with an approximate includes the C6713 floating-point digital signal processor and a 32-bit stereo codec TLV320AIC23 (AIC23) for input and output. The onboard codec AIC23 [37] uses a sigma–delta technology that provides ADC and DAC. It connects to a 12-MHz system clock. Variable sampling rates from 8 to96 kHz can be set readily. A daughter card expansion is also provided on the DSK board. Two 80-pin connectors provide for external peripheral and external memory interfaces.

The DSK board includes 16MB (megabytes) of synchronous dynamic random access memory (SDRAM) and 256kB (kilobytes) of flash memory. Four connectors on the board provide input and output: MIC IN for microphone input, LINE IN for line input, LINE OUT for line output, and HEADPHONE for a headphone output (multiplexed with line output). The status of the four user dip switches on the DSK board can be read from a program and provides the user with a feedback control interface. The DSK operates at 225MHz. Also onboard the DSK are voltage regulators that provide 1.26 V for the C6713 core and 3.3V for its memory and peripherals.

5.3.2 AIC23 Codec

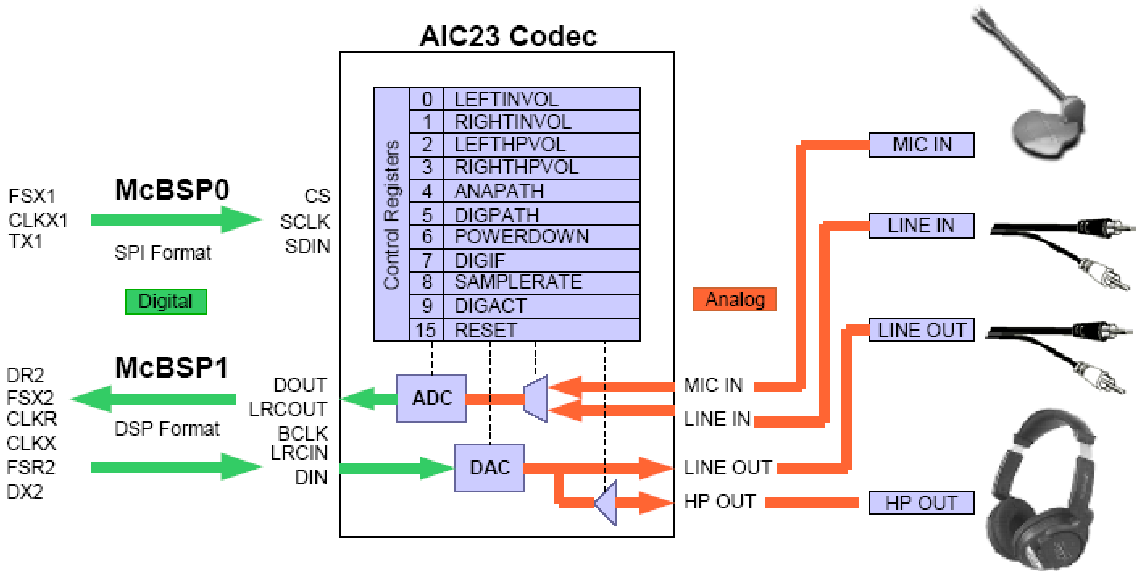


Figure 5.3.3 TMS320C6713 DSK CODEC INTERFACE

The DSK uses a Texas Instruments AIC23 (part no. TLV320AIC23) stereo codec for input and output of audio signals. The codec samples analog signals on the microphone or line inputs and converts them into digital data so it can be processed by the DSP. When the DSP is finished with the data it uses the codec to convert the samples back into analog signals on the line and headphone outputs so the user can hear the output. The codec communicates using two serial channels- McBSP0 & McBSP1,

- McBSP0 is used to control the codec's internal configuration registers

The SPI format is used to specify the data (Top 7 bits to select the configuration register and last 9 bits to specify the register contents).This Channel is idle when audio data is transmitted.

- McBSP1 is used to send and receive digital audio samples.

Acts as the bi-directional data channel. All audio data flows through the data channel. Many data formats are supported based on the three variables of sample width, clock signal source and serial data format. Preferred serial format is DSP mode which is designed specifically to operate with the McBSP ports on TI DSPs.

The codec has a 12 MHz system clock. The 12 MHz system clock corresponds to USB sample rate mode, named because many USB systems use a 12 MHz clock and can use the same clock for both the codec and USB controller. The internal sample rate generate subdivides the 12 MHz clock to generate common frequencies such as 48 KHz, 44.1 KHz and 8 KHz. The sample rate is set by the codec's SAMPLERATE register.

5.3.3 Development Tools

Code Composer Studio

CCS provides an IDE to incorporate the software tools. CCS includes tools for code generation, such as a C compiler, an assembler, and a linker. It has graphical capabilities and supports real-time debugging. It provides an easy-to-use software tool to build and debug programs. The C compiler compiles a C source program with extension .c to produce an assembly source file with extension.*asm*. The assembler assembles an.*asm* source file to produce a machine language object file with extension.*obj*.The linker combines object files and object libraries as input to produce an executable file with extension. *out*. This executable file represents a linked common object file format (COFF), popular in Unix-based systems and adopted by several makers of digital signal processors. This executable file can be loaded and run directly on the C6713 processor.

Real-time analysis can be performed using real-time data exchange (RTDX). RTDX allows for data exchange between the host PC and the target DSK, as well as analysis in real time without stopping the target. Key statistics and performance can be monitored in real time. Through the joint team action group (JTAG), communication with on-chip emulation support occurs to control and monitor program execution. The C6713 DSK board includes a JTAG interface through the USB port.

The different files typically encountered in a CCS environment while building the system are

1. file.pjt: to create and build a project named file

2. file.c: C source program

3. file.asm: assembly source program created by the user, by the C compiler, or by the linear optimizer

4. file.sa: linear assembly source program. The linear optimizer uses *file.sa* as input to produce an assembly program *file.asm*

5. file.h: header support file

6. file.lib: library file, such as the run-time support library file rts6700.lib

7. file.cmd: linker command file that maps sections to memory

8. file.obj: object file created by the assembler

9. file.out: executable file created by the linker to be loaded and run on the C6713 processor

10. file.cdb: configuration file when using DSP/BIOS

Procedure to work on Code Composer Studio for DSK6713:

1. To create New Project

   Project -> New (file name eg: test)

2. To create a source file

   File -> New -> Type the code (Save and give a file name eg:sum.c)

3. To add source files to project

   Project -> Add files to project -> sum.c

4. To add library files

   Project -> Add files to project -> dsk6713bsl.lib

   Note: Library files path - c:\ccstudio\c6000\dsk6713bsl.lib

   Select Object and Library in Type of files(*.o and *.l files)

5. To add Configuration Database file

   File -> New -> DSP/BIOS configuration

   Select DSK6713.cdb -> Open

   Save as X.cdb in the project folder (X depends on the the first include file in the main program)

   Project -> Add files to project -> Add X.cdb from the project folder

6. To include the files

   Open c:\ccstudio\c6000\dsk6713\include

   Copy the files

   dsk6713.h &

   dsk6713_aic23.h into the project folder

   Project -> Add files to project -> Add the two files just copied

7. Project -> Add files to project -> Xcfg.h

8. To connect

   Debug -> Connect

9. To Compile

   Project -> Compile

10. To Rebuild

   Project -> Rebuild

11. To Load program

   File -> Load Program -> Select *.out

12. To Execute the program

   Build -> Run

5.3.4 Memory Map

The C67xx family of DSPs has a large byte addressable address space. Program code and data can be placed anywhere in the unified address space. Addresses are always 32-bits wide. The memory map shows the address space of a generic 6713 processor on the left with specific details of how each region is used on the right. By default, the internal memory sits at the beginning of the address space. Portions of the internal memory can be reconfigured in software as L2 cache rather than fixed RAM.

The IRAM memory segment is selected by default. The choice of IRAM/SDRAM is left to the programmer and is usually dictated by the storage requirements of the application. Real Time applications like Speech Processing, Image Processing usually have high storage requirements and hence the space constraints of the IRAM necessitate the use of the SDRAM. In our project we have made judicious use of IRAM/SDRAM to handle the storage requirements. Typically the parameters like Input samples, windowed sample FFT computed values are dumped in the SDRAM. A more detailed explanation on the way memory has been handled for our application will be explained in the later sections.

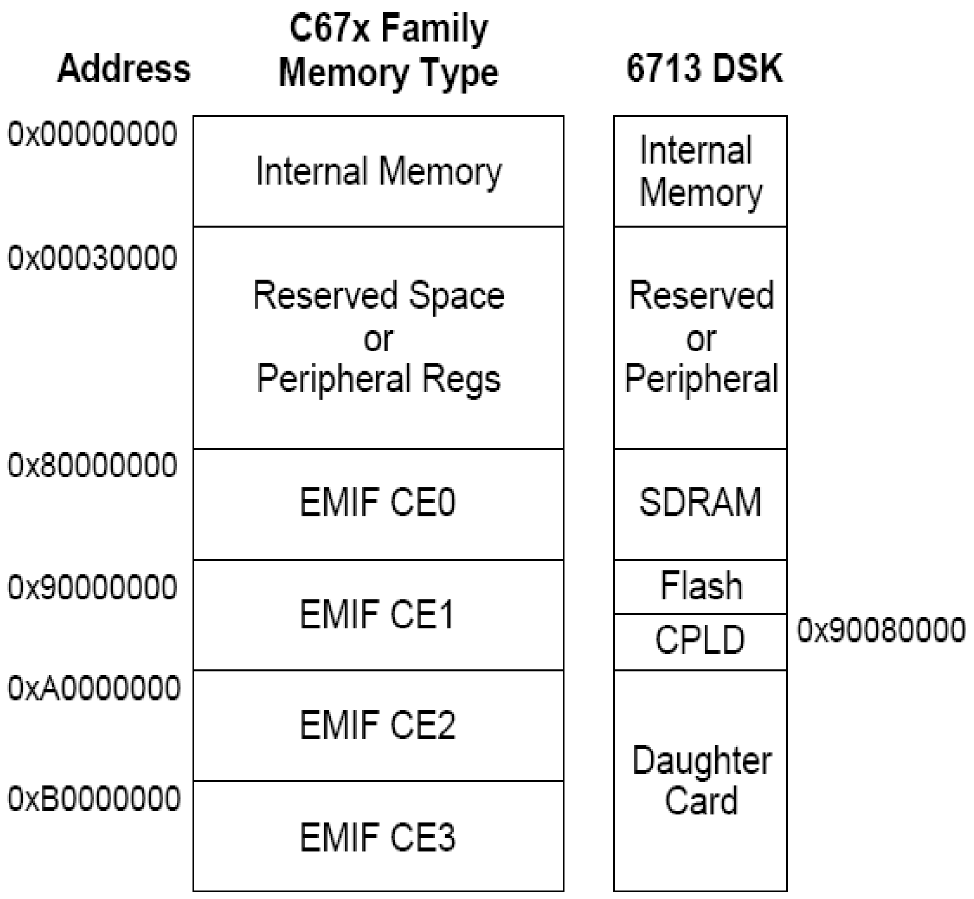


Figure 5.3.4: Memory Map

6. ASR ALGORITHM OPTIMIZATION

6.1 INTRODUCTION

Despite the continual growth of the computing power of DSP, direct implementation of ASR algorithms can't over real-time processing. When porting speech recognition algorithms to embedded platforms, the essential procedures will be included:

- Analyze the algorithm and identify the computationally most critical parts.
- Simplify the computation by, e.g. using look-up table, avoiding complex function, etc.
- Avoid unnecessarily repetitive computation.
- Pre-compute frequently used parameters as pre-stored constant values whenever possible.
- Use low-level language if appropriate.

In this work, we have used a set of optimizations for ASR algorithms. By analyzing each computation part in ASR, the most effective optimization methods are applied.

Feature extraction techniques

Feature extraction is computationally intensive because it involves many signal processing steps such as windowing, Fast Fourier Transform, Mel-scale filter banks, Discrete Cosine transform and so on. For each of these steps, different approximation techniques have been considered.

Windowing

In this work, we use the Hamming window .For a frame with fixed-length N, the window shape is fixed and it is not needed to calculate the window function $h\ (n)$ for each incoming frame. Instead, $h\ (n)\ ;\ 0 <n < N$ is pre-stored in the RAM. In this way, we save N cosine operations, N multiplications and N additions which require a lot of CPU cycles. We will need N 16-bit words of RAM to store the window function which is not overburden compared to the execution time it costs.

FFT Implementation

Although the Fast Fourier Transform (FFT) is much more efficient than the original Discrete Fourier Transform (DFT), it still requires the computation of many exponential or trigonometric functions and multiplications that would take a long execution time on DSPs. In our work, we have used a 256 point DIF –Radix 2 FFT. Since speech is a real-valued signal, each input frame contains 256 real elements .The result contains the first half i.e. 128 complex elements as the FFT output. This output still provides the full information because an FFT of a real sequence has even symmetry around the center or Nyquist point (*N/2*).

6.2 MEMORY REQUIREMENTS CONSIDERATION

Embedded DSP system has limited memory resource. For the front-end part, the major memory requirement comes from storage of look-up table for DCT and gains of filterbanks, and some pre-computed parameters such as the weights of Hamming windows, the center frequencies of each filter bank and cepstral weights.

In our work, memory requirements arose from the fact that the speech samples (10000 samples /sec) need to be initially stored, windowed and suitably processed. The default space selected by CCS corresponds to IRAM which doesn’t satisfy the memory storage requirements. So the SDRAM space was made use to store the parameters. Suitable heap structures were defined in SDRAM and the parameters dumped into these. Typical these included,

- Input speech samples >SDRAM
- Hamming window points > Implemented as a lookup table and stored in IRAM
  FFT and Magnitude of FFT > IRAM
- In this case only the Windowed signal > SDRAM

It is not advisable to evaluate FFT using the twiddle factors and the data in the SDRAM, so, we created a buffer space in the IRAM where the input samples were fetched from external memory, stored and the FFT was evaluated. This avoids multiple external memory accesses and speeds up the process.

The feature vectors evaluated after the training phase would be required for the testing phase. It was seen that the data in the IRAM was not overwritten provided we declared a proper section for all our required data; so, we stored the feature vector in the predefined section in the IRAM thus retaining the data even for the testing phase.

Note: Keep track of the memory segments defined in the C codes at the end of the report

7. SOURCE CODE

7.1 MATLAB CODE

7.1.1 MFCC Approach

7.1.1.1 Training Phase

```
fs = 10000;            % Sampling Frequency
t = hamming(4000); % Hamming window to smooth the speech signal
w = [t ; zeros(6000,1)];
f = (1:10000);
mel(f) = 2595 * log(1 + f / 700); % Linear to Mel frequency scale conversion
tri = triang(100);
win1 = [tri ; zeros(9900,1)]; % Defining overlapping triangular windows for
win2 = [zeros(50,1) ; tri ; zeros(9850,1)]; % frequency domain analysis
win3 = [zeros(100,1) ; tri ; zeros(9800,1)];
win4 = [zeros(150,1) ; tri ; zeros(9750,1)];
win5 = [zeros(200,1) ; tri ; zeros(9700,1)];
win6 = [zeros(250,1) ; tri ; zeros(9650,1)];
win7 = [zeros(300,1) ; tri ; zeros(9600,1)];
win8 = [zeros(350,1) ; tri ; zeros(9550,1)];
win9 = [zeros(400,1) ; tri ; zeros(9500,1)];
win10 = [zeros(450,1) ; tri ; zeros(9450,1)];
win11 = [zeros(500,1) ; tri ; zeros(9400,1)];
win12 = [zeros(550,1) ; tri ; zeros(9350,1)];
win13 = [zeros(600,1) ; tri ; zeros(9300,1)];
win14 = [zeros(650,1) ; tri ; zeros(9250,1)];
win15 = [zeros(700,1) ; tri ; zeros(9200,1)];
win16 = [zeros(750,1) ; tri ; zeros(9150,1)];
win17 = [zeros(800,1) ; tri ; zeros(9100,1)];
win18 = [zeros(850,1) ; tri ; zeros(9050,1)];
win19 = [zeros(900,1) ; tri ; zeros(9000,1)];
win20 = [zeros(950,1) ; tri ; zeros(8950,1)];
x = wavrecord(1 * fs, fs, 'double'); % Record and store the uttered speech
plot(x);
wavplay(x);
i = 1;
while abs(x(i)) < 0.05                % Silence detection
    i = i + 1;
end
x(1 : i) = [];
x(6000 : 10000) = 0;
x1 = x. * w;
mx = fft(x1);                  % Transform to frequency domain
nx = abs(mx(floor(mel(f)))); % Mel warping
nx = nx. / max(nx);
nx1 = nx. * win1;
nx2 = nx. * win2;
nx3 = nx. * win3;
nx4 = nx. * win4;
```

```
nx5 = nx. * win5;
nx6 = nx. * win6;
nx7 = nx. * win7;
nx8 = nx. * win8;
nx9 = nx. * win9;
nx10 = nx. * win10;
nx11 = nx. * win11;
nx12 = nx. *win12;
nx13 = nx. * win13;
nx14 = nx. * win14;
nx15 = nx. * win15;
nx16 = nx. * win16;
nx17 = nx. * win17;
nx18 = nx. * win18;
nx19 = nx. * win19;
nx20 = nx. * win20;
sx1 = sum(nx1. ^ 2); % Determine the energy of the signal within each window
sx2 = sum(nx2. ^ 2); % by summing square of the magnitude of the spectrum
sx3 = sum(nx3. ^ 2);
sx4 = sum(nx4. ^ 2);
sx5 = sum(nx5. ^ 2);
sx6 = sum(nx6. ^ 2);
sx7 = sum(nx7. ^ 2);
sx8 = sum(nx8. ^ 2);
sx9 = sum(nx9. ^ 2);
sx10 = sum(nx10. ^ 2);
sx11 = sum(nx11. ^ 2);
sx12 = sum(nx12. ^ 2);
sx13 = sum(nx13. ^ 2);
sx14 = sum(nx14. ^ 2);
sx15 = sum(nx15. ^ 2);
sx16 = sum(nx16. ^ 2);
sx17 = sum(nx17. ^ 2);
sx18 = sum(nx18. ^ 2);
sx19 = sum(nx19. ^ 2);
sx20 = sum(nx20. ^ 2);
sx = [sx1, sx2, sx3, sx4, sx5, sx6, sx7, sx8, sx9, sx10, sx11, sx12, sx13, sx14,
        sx15, sx16, sx17, sx18, sx19, sx20];
sx = log(sx);
dx = dct(sx);              % Determine DCT of Log of the spectrum energies
fid = fopen('sample.dat', 'w');
fwrite(fid, dx, 'real*8'); % Store this feature vector as a .dat file
fclose(fid);
```

7.1.1.2 Testing Phase

```
fs = 10000;              % Sampling Frequency
t = hamming(4000); % Hamming window to smooth the speech signal
w = [t ; zeros(6000,1)];
f = (1:10000);
mel(f) = 2595 * log(1 + f / 700); % Linear to Mel frequency scale conversion
tri = triang(100);
win1 = [tri ; zeros(9900,1)]; % Defining overlapping triangular windows for
win2 = [zeros(50,1) ; tri ; zeros(9850,1)]; % frequency domain analysis
win3 = [zeros(100,1) ; tri ; zeros(9800,1)];
win4 = [zeros(150,1) ; tri ; zeros(9750,1)];
win5 = [zeros(200,1) ; tri ; zeros(9700,1)];
win6 = [zeros(250,1) ; tri ; zeros(9650,1)];
win7 = [zeros(300,1) ; tri ; zeros(9600,1)];
win8 = [zeros(350,1) ; tri ; zeros(9550,1)];
win9 = [zeros(400,1) ; tri ; zeros(9500,1)];
win10 = [zeros(450,1) ; tri ; zeros(9450,1)];
win11 = [zeros(500,1) ; tri ; zeros(9400,1)];
win12 = [zeros(550,1) ; tri ; zeros(9350,1)];
win13 = [zeros(600,1) ; tri ; zeros(9300,1)];
win14 = [zeros(650,1) ; tri ; zeros(9250,1)];
win15 = [zeros(700,1) ; tri ; zeros(9200,1)];
win16 = [zeros(750,1) ; tri ; zeros(9150,1)];
win17 = [zeros(800,1) ; tri ; zeros(9100,1)];
win18 = [zeros(850,1) ; tri ; zeros(9050,1)];
win19 = [zeros(900,1) ; tri ; zeros(9000,1)];
win20 = [zeros(950,1) ; tri ; zeros(8950,1)];
y = wavrecord(1 * fs, fs, 'double'); %Store the uttered password for authentication
i = 1;
while abs(y(i)) < 0.05            % Silence Detection
    i = i + 1;
end
y(1 : i) = [];
y(6000 : 10000) = 0;
y1 = y. * w;
my = fft(y1);                    % Transform to frequency domain
ny = abs(my(floor(mel(f)))); % Mel warping
ny = ny. / max(ny);
ny1 = ny. * win1;
ny2 = ny. * win2;
ny3 = ny. * win3;
ny4 = ny. * win4;
ny5 = ny. * win5;
ny6 = ny. * win6;
ny7 = ny. * win7;
ny8 = ny. * win8;
ny9 = ny. * win9;
ny10 = ny. * win10;
ny11 = ny. * win11;
```

```
ny12 = ny. * win12;
ny13 = ny. * win13;
ny14 = ny. * win14;
ny15 = ny. * win15;
ny16 = ny. * win16;
ny17 = ny. * win17;
ny18 = ny. * win18;
ny19 = ny. * win19;
ny20 = ny. * win20;
sy1 = sum(ny1. ^ 2);
sy2 = sum(ny2. ^ 2);
sy3 = sum(ny3. ^ 2);
sy4 = sum(ny4. ^ 2);
sy5 = sum(ny5. ^ 2);
sy6 = sum(ny6. ^ 2);
sy7 = sum(ny7. ^ 2);
sy8 = sum(ny8. ^ 2);
sy9 = sum(ny9. ^ 2);
sy10 = sum(ny10. ^ 2); % Determine the energy of the signal within each window
sy11 = sum(ny11. ^ 2); % by summing square of the magnitude of the spectrum
sy12 = sum(ny12. ^ 2);
sy13 = sum(ny13. ^ 2);
sy14 = sum(ny14. ^ 2);
sy15 = sum(ny15. ^ 2);
sy16 = sum(ny16. ^ 2);
sy17 = sum(ny17. ^ 2);
sy18 = sum(ny18. ^ 2);
sy19 = sum(ny19. ^ 2);
sy20 = sum(ny20. ^ 2);
sy = [sy1, sy2, sy3, sy4, sy5, sy6, sy7, sy8, sy9, sy10, sy11, sy12, sy13, sy14,
        sy15, sy16, sy17, sy18, sy19, sy20];
sy = log(sy);
dy = dct(sy);              % Determine DCT of Log of the spectrum energies
fid = fopen('sample.dat','r');
dx = fread(fid, 20, 'real*8');          % Obtain the feature vector for the password
fclose(fid);                             % evaluated in the training phase
dx = dx.';
MSE=(sum((dx - dy). ^ 2)) / 20;     % Determine the Mean squared error
if MSE<1
   fprintf('\n\nACCESS GRANTED\n\n');
   Grant=wavread('Grant.wav');  % "Access Granted" is output if within threshold
   wavplay(Grant);
else
   fprintf('\n\nACCESS DENIED\n\n');
   Deny=wavread('Deny.wav');    % "Access Denied" is output in case of a failure
   wavplay(Deny);
end
```

7.1.2 Time Domain Approach

7.1.2.1 Training Phase

```
fs = 10000;          % Sampling Frequency
w = hamming(2500);    % Hamming window to smoothen speech frame
h = hamming(256);     % Hamming window to find STFT
win1 = [h ; zeros(2244,1)];   % Twelve windows in Time-domain to compute STFT
win2 = [zeros(200,1) ; h ; zeros(2044,1)];
win3 = [zeros(400,1) ; h ; zeros(1844,1)];
win4 = [zeros(600,1) ; h ; zeros(1644,1)];
win5 = [zeros(800,1) ; h ; zeros(1444,1)];
win6 = [zeros(1000,1) ; h ; zeros(1244,1)];
win7 = [zeros(1200,1) ; h ; zeros(1044,1)];
win8 = [zeros(1400,1) ; h ; zeros(844,1)];
win9 = [zeros(1600,1) ; h ; zeros(644,1)];
win10 = [zeros(1800,1) ; h ; zeros(444,1)];
win11 = [zeros(2000,1) ; h ; zeros(244,1)];
win12 = [zeros(2244,1) ; h];

fprintf('\nHit enter and store the password');
pause;
x=wavrecord(1 * fs,fs,'double');          % Record Training signal
figure; plot(x);
wavplay(x);

%Silence Detection
i = 1;
while abs(x(i)) < 0.05             % Detect Speech signal Position
    i = i + 1;
end
x(1 : i) = [];
x(6000 : 10000) = 0;               % Extract Only 2500 samples of Actual Speech
x(2501 : 10000) = [];
x1 = x;
figure; plot(x1);
nx= x1;

nx1 = nx. * win1;       % Window the Speech Frame with a First Short-time
Window
i = 1;
while abs(nx1(i)) == 0
    i = i + 1;
end
nx1(1 : i) = [];
nx1(6000 : 10000) = 0;
nx1(501 : 10000) = [];
X1 = abs(fft(nx1));            % Finds the position of Dominant Frequency
Component
```

```
while X1(i) ~= max(X1)
     i = i + 1;
end
d1 = i

nx2 = nx. * win2;    % Window the Speech Frame with a Second Short-time
Window
i = 1;
while abs(nx2(i)) == 0
   i = i + 1;
end
nx2(1 : i) = [];
nx2(6000 : 10000) = 0;
nx2(501 : 10000) = [];
X2 = abs(fft(nx2));              % Finds the position of Dominant Frequency
Component
i = 1;
while X2(i) ~= max(X2)
      i = i + 1;
end
d2 = i

nx3 = nx. * win3;       % Window the Speech Frame with a Third Short-time
Window
i = 1;
while abs(nx3(i)) == 0
    i = i + 1;
end
nx3(1 : i) = [];
nx3(6000 : 10000) = 0;
nx3(501 : 10000) = [];
X3 = abs(fft(nx3));              % Finds the position of Dominant Frequency
Component
i = 1;
while X3(i) ~= max(X3)
    i = i + 1;
end
d3 = i

nx4 = nx. * win4;     % Window the Speech Frame with a Fourth Short-time
Window
i = 1;
while abs(nx4(i)) == 0
     i = i + 1;
end
nx4(1 : i) = [];
nx4(6000 : 10000) = 0;
nx4(501 : 10000) = [];
X4 = abs(fft(nx4));              % Finds the position of Dominant Frequency
Component
```

```
i = 1;
while X4(i) ~= max(X4)
    i = i + 1;
end
d4 = i

nx5 = nx. * win5;      % Window the Speech Frame with a Fifth Short-time Window
i = 1;
while abs(nx5(i)) == 0
    i = i + 1;
end
nx5(1 : i) = [];
nx5(6000 : 10000) = 0;
nx5(501 : 10000) = [];
X5 = abs(fft(nx5));         % Finds the position of Dominant Frequency Component
i = 1;
while X5(i ) ~= max(X5)
    i = i + 1;
end
d5 = i

nx6 = nx. * win6;       % Window the Speech Frame with a Sixth Short-time
Window
i = 1;
while abs(nx6(i)) == 0
    i = i + 1;
end
nx6(1 : i) = [];
nx6(6000 : 10000) = 0;
nx6(501 : 10000) = [];
X6 = abs(fft(nx6));              % Finds the position of Dominant Frequency
Component
i = 1;
while X6(i) ~= max(X6)
    i = i + 1;
end
d6 = i

nx7 = nx. * win7; % Window the Speech Frame with a Seventh Short-time Window
i = 1;
while abs(nx7(i)) == 0
    i = i + 1;
end
nx7(1 : i) = [];
nx7(6000 : 10000) = 0;
nx7(501 : 10000) = [];
X7 = abs(fft(nx7));              % Finds the position of Dominant Frequency
Component
i = 1;
while X7(i) ~= max(X7)
```

```
    i  = i  + 1;
end
d7 = i


nx8 = nx. * win8;   % Window the Speech Frame with a Eighth Short-time Window
i = 1;
while abs(nx8(i)) == 0
    i = i + 1;
end
nx8(1 : i) = [];
nx8(6000 : 10000) = 0;
nx8(501 : 10000) = [];
X8 = abs(fft(nx8));              % Finds the position of Dominant Frequency
Component
i = 1;
while X8(i) ~= max(X8)
    i = i + 1;
end
d8 = i


nx9 = nx. * win9;   % Window the Speech Frame with a Nineth Short-time Window
i = 1;
while abs(nx9(i)) == 0
   i = i + 1;
end
nx9(1 : i) = [];
nx9(6000 : 10000) = 0;
nx9(501 : 10000) = [];
X9 = abs(fft(nx9));            % Finds the position of Dominant Frequency Component
i = 1;
while X9(i) ~= max(X9)
    i = i + 1;
end
d9 = i


nx10 = nx.*win10;     % Window the Speech Frame with a Tenth Short-time
Window
i = 1;
while abs(nx10(i)) == 0
    i = i + 1;
end
nx10(1 : i) = [];
nx10(6000 : 10000) = 0;
nx10(501 : 10000) = [];
X10 = abs(fft(nx10));        % Finds the position of Dominant Frequency Component
i = 1;
while X10(i) ~= max(X10)
```

```
    i = i + 1;
end
d10 = i


nx11 = nx. * win11;% Window the Speech Frame with a Eleventh Window
i = 1;
while abs(nx11(i)) == 0
    i = i + 1;
end
nx11(1 : i) = [];
nx11(6000 : 10000) = 0;
nx11(501 : 10000) = [];
X11 = abs(fft(nx11));      % Finds the position of Dominant Frequency Component
i = 1;
while X11(i) ~= max(X11)
    i = i + 1;
end
d11 = i


nx12 = nx. *win12; % Window the Speech Frame with a Twelfth Window
i = 1;
while abs(nx12(i)) == 0
    i = i + 1;
end
nx12(1 : i) = [];
nx12(6000 : 10000) = 0;
nx12(501 : 10000) = [];
X12 = abs(fft(nx12));    % Finds the position of Dominant Frequency Component
i = 1;
while X12(i) ~= max(X12)
    i = i + 1;
end
d12 = i


dx = [d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12];
fid = fopen('feature.dat', 'w');      % Save the Feature Vector as a .dat file for
fwrite(fid, dx, 'real*8');             % comparison in the testing phase
fclose(fid);
```

7.1.2.2 Testing Phase

```
fs = 10000;          % Sampling Frequency
w = hamming(2500);   % Hamming window to smoothen speech frame
h = hamming(256);    % Hamming window to find STFT
win1 = [h ; zeros(2244,1)];   % Twelve windows in Time-domain to compute STFT
win2 = [zeros(200,1) ; h ; zeros(2044,1)];
win3 = [zeros(400,1) ; h ; zeros(1844,1)];
win4 = [zeros(600,1) ; h ; zeros(1644,1)];
win5 = [zeros(800,1) ; h ; zeros(1444,1)];
win6 = [zeros(1000,1) ; h ; zeros(1244,1)];
win7 = [zeros(1200,1) ; h ; zeros(1044,1)];
win8 = [zeros(1400,1) ; h ; zeros(844,1)];
win9 = [zeros(1600,1) ; h ; zeros(644,1)];
win10 = [zeros(1800,1) ; h ; zeros(444,1)];
win11 = [zeros(2000,1) ; h ; zeros(244,1)];
win12 = [zeros(2244,1) ; h];

fprintf('\nHit enter and store the password');
pause;
x=wavrecord(1 * fs,fs,'double');         % Record Training signal
figure; plot(x);
wavplay(x);

%Silence Detection
i = 1;
while abs(x(i)) < 0.05           % Detect Speech signal Position
    i = i + 1;
end
x(1 : i) = [];
x(6000 : 10000) = 0;             % Extract Only 2500 samples of Actual Speech
x(2501 : 10000) = [];
x1 = x;
figure; plot(x1);
nx= x1;

nx1 = nx. * win1;     % Window the Speech Frame with a First Short-time Window
i = 1;
while abs(nx1(i)) == 0
    i = i + 1;
end
nx1(1 : i) = [];
nx1(6000 : 10000) = 0;
nx1(501 : 10000) = [];
X1 = abs(fft(nx1));          % Finds the position of Dominant Frequency Component
while X1(i) ~= max(X1)
    i = i + 1;
end
d1 = i
```

```
nx2 = nx. * win2;  % Window the Speech Frame with a Second Short-time Window
i = 1;
while abs(nx2(i)) == 0
   i = i + 1;
end
nx2(1 : i) = [];
nx2(6000 : 10000) = 0;
nx2(501 : 10000) = [];
X2 = abs(fft(nx2));           % Finds the position of Dominant Frequency Component
i = 1;
while X2(i) ~= max(X2)
      i = i + 1;
end
d2 = i

nx3 = nx. * win3;    % Window the Speech Frame with a Third Short-time Window
i = 1;
while abs(nx3(i)) == 0
    i = i + 1;
end
nx3(1 : i) = [];
nx3(6000 : 10000) = 0;
nx3(501 : 10000) = [];
X3 = abs(fft(nx3));          % Finds the position of Dominant Frequency Component
i = 1;
while X3(i) ~= max(X3)
    i = i + 1;
end
d3 = i

nx4 = nx. * win4;   % Window the Speech Frame with a Fourth Short-time Window
i = 1;
while abs(nx4(i)) == 0
     i = i + 1;
end
nx4(1 : i) = [];
nx4(6000 : 10000) = 0;
nx4(501 : 10000) = [];
X4 = abs(fft(nx4));           % Finds the position of Dominant Frequency Component
i = 1;
while X4(i) ~= max(X4)
     i = i + 1;
end
d4 = i

nx5 = nx. * win5;     % Window the Speech Frame with a Fifth Short-time Window
i = 1;
while abs(nx5(i)) == 0
     i = i + 1;
end
```

```
nx5(1 : i) = [];
nx5(6000 : 10000) = 0;
nx5(501 : 10000) = [];
X5 = abs(fft(nx5));           % Finds the position of Dominant Frequency Component
i = 1;
while X5(i ) ~= max(X5)
    i = i + 1;
end
d5 = i

nx6 = nx. * win6;    % Window the Speech Frame with a Sixth Short-time Window
i = 1;
while abs(nx6(i)) == 0
    i = i + 1;
end
nx6(1 : i) = [];
nx6(6000 : 10000) = 0;
nx6(501 : 10000) = [];
X6 = abs(fft(nx6));          % Finds the position of Dominant Frequency Component
i = 1;
while X6(i) ~= max(X6)
    i = i + 1;
end
d6 = i

nx7 = nx. * win7; % Window the Speech Frame with a Seventh Short-time Window
i = 1;
while abs(nx7(i)) == 0
    i = i + 1;
end
nx7(1 : i) = [];
nx7(6000 : 10000) = 0;
nx7(501 : 10000) = [];
X7 = abs(fft(nx7));          % Finds the position of Dominant Frequency Component
i = 1;
while X7(i) ~= max(X7)
     i  = i  + 1;
end
d7 = i

nx8 = nx. * win8;   % Window the Speech Frame with a Eighth Short-time Window
i = 1;
while abs(nx8(i)) == 0
     i = i + 1;
end
nx8(1 : i) = [];
nx8(6000 : 10000) = 0;
nx8(501 : 10000) = [];
X8 = abs(fft(nx8));         % Finds the position of Dominant Frequency Component
i = 1;
```

```
while X8(i) ~= max(X8)
     i = i + 1;
end
d8 = i

nx9 = nx. * win9;   % Window the Speech Frame with a Ninth Short-time Window
i = 1;
while abs(nx9(i)) == 0
    i = i + 1;
end
nx9(1 : i) = [];
nx9(6000 : 10000) = 0;
nx9(501 : 10000) = [];
X9 = abs(fft(nx9));           % Finds the position of Dominant Frequency Component
i = 1;
while X9(i) ~= max(X9)
     i = i + 1;
end
d9 = i

nx10 = nx.*win10;  % Window the Speech Frame with a Tenth Short-time Window
i = 1;
while abs(nx10(i)) == 0
     i = i + 1;
end
nx10(1 : i) = [];
nx10(6000 : 10000) = 0;
nx10(501 : 10000) = [];
X10 = abs(fft(nx10));       % Finds the position of Dominant Frequency Component
i = 1;
while X10(i) ~= max(X10)
     i = i + 1;
end
d10 = i

nx11 = nx. * win11;   %Window the Speech Frame with a Eleventh Window
i = 1;
while abs(nx11(i)) == 0
     i = i + 1;
end
nx11(1 : i) = [];
nx11(6000 : 10000) = 0;
nx11(501 : 10000) = [];
X11 = abs(fft(nx11));           % Finds the position of Dominant Frequency
Component
i = 1;
while X11(i) ~= max(X11)
     i = i + 1;
end
d11 = i
```

```
nx12 = nx. *win12; % Window the Speech Frame with a Twelfth Window
i = 1;
while abs(nx12(i)) == 0
     i = i + 1;
end
nx12(1 : i) = [];
nx12(6000 : 10000) = 0;
nx12(501 : 10000) = [];
X12 = abs(fft(nx12));      % Finds the position of Dominant Frequency Component
i = 1;
while X12(i) ~= max(X12)
     i = i + 1;
end
d12 = i

dy = [d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12];

fid = fopen('feature.dat', 'r');    %Retrieve the Feature Vector from Training Phase
dx = fread(fid, 12, 'real*8');
fclose(fid);
dx = dx.';

i = 1;
j = 0;
while(i < 13)
   if(abs(dy(i) - dx(i)) < 5)   % Check the deviation between dominant
       j = j + 1;                % spectral Components in each window
   end
i = i + 1;
end
if j > 7                              % Check for desired number of matches
   fprintf('\n\nACCESS GRANTED\n\n');
   Grant = wavread('Grant.wav');  % "Access Granted" previously stored as .dat
   wavplay(Grant);
else
   fprintf('\n\nACCESS DENIED\n\n');
   Deny = wavread('Deny.wav');  % "Access Denied" previously stored as .dat
   wavplay(Deny);
end
```

7.2 C CODE FOR IMPLEMENTATION

7.2.1 Initialization

Note*: **All the initial declaration of the arrays and the various segments in the memory should be retained in all the programs. Failing to do so will create overlap in memory resulting in loss of important data.***

7.2.1.1 Storing the "Access Granted" Signal

```
/* "Access Granted" was uttered and stored so that in the testing phase when the system
validates the user it can send out this signal through the speakers */

#include"dsk6713.h"

#include"dsk6713_aic23.h"

#include"stdio.h"

#include"math.h"

#define FS 10000
#define PTS 256
#define PI 3.14159265358979

#pragma DATA_SECTION(spec,".temp") //.temp is a segment defined in IRAM
#pragma DATA_SECTION(x,".SDRAM$heap") // declaration to store in SDRAM
#pragma DATA_SECTION(nx1,".SDRAM$heap")
#pragma DATA_SECTION(nx12,".SDRAM$heap")
#pragma DATA_SECTION(win,".SDRAM$heap")
#pragma DATA_SECTION(win1,".SDRAM$heap")
#pragma DATA_SECTION(win12,".SDRAM$heap")
#pragma DATA_SECTION(w,".temp")
#pragma DATA_SECTION(samples,".temp")
#pragma DATA_SECTION(deny,".new") //.new is a segment in IRAM
#pragma DATA_SECTION(grant,".new")

typedef struct {float real, imag;} COMPLEX;
void FFT_init(float *); // Declaration of functions for calculating FFT
void FFT(COMPLEX *, int );
int maxim(float *, int ); // Function to evaluate dominant frequency

float win[256], win1[2500], win12[2500]; // Declaration of all the variables in use
float x[10000], deny[8000], grant[8000], spec[256];
float nx1[2500], nx12[2500];
COMPLEX w[PTS]; // Twiddle factors defined as objects of a complex structure
COMPLEX samples[PTS];

int i = 0,num = 0, j, temp, N, k, d[12];
```

```
/* The following section is the initialization of the codec, the main change which
was made by us is to make parameter 4 – 0x0015 to enable analog audio path
control, The first four parameters are varied till the input and output speech is
clearly audible */

DSK6713_AIC23_Config config = { \
   0x0017,  /* 0 DSK6713_AIC23_LEFTINVOL  Left line input channel volume */ \
   0x0017,  /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume */\
   0x00e0,  /* 2 DSK6713_AIC23_LEFTHPVOL  Left channel headphone volume */  \
   0x00e0,  /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume */ \
   0x0015,  /* 4 DSK6713_AIC23_ANAPATH    Analog audio path control */      \
   0x0000,  /* 5 DSK6713_AIC23_DIGPATH    Digital audio path control */     \
   0x0000,  /* 6 DSK6713_AIC23_POWERDOWN  Power down control */             \
   0x0043,  /* 7 DSK6713_AIC23_DIGIF      Digital audio interface format */ \
   0x0081,  /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */            \
   0x0001   /* 9 DSK6713_AIC23_DIGACT     Digital interface activation */   \
};

void main()
{
   DSK6713_AIC23_CodecHandle hCodec;

   Uint32 l_input, r_input,l_output, r_output;

   int i;
   /* Initialize the board support library, must be called first */
   DSK6713_init();

   /* Start the codec */
   hCodec = DSK6713_AIC23_openCodec(0, &config);

   DSK6713_AIC23_setFreq(hCodec, 1); // Sampling frequency : 1 = 8000 Hz

   for(i = 0 ; i < 8000 ; i++)
     {        /* Read a sample to the left channel */
                     while (!DSK6713_AIC23_read(hCodec, &l_input));

                     /* Read a sample to the right channel */
                     while (!DSK6713_AIC23_read(hCodec, &r_input));


                              grant[i]=l_input; // The uttered signal is sent out through
                              l_output=grant[i]; // the speakers so that the user can hear
                              r_output=grant[i]; // what is uttered


                     /* Send a sample to the left channel */
         while (!DSK6713_AIC23_write(hCodec, l_output));

         /* Send a sample to the right channel */
         while (!DSK6713_AIC23_write(hCodec, r_output));
```

```
        }

    /* Close the codec */
    DSK6713_AIC23_closeCodec(hCodec);

  }
```

7.2.1.2 Storing the “Access Denied” Signal

```
/* “Access Denied” was uttered and stored so that in the testing phase when the system
validates the user it can send out this signal through the speakers */

#include"dsk6713.h"
#include"dsk6713_aic23.h"

#include"stdio.h"

#include"math.h"

#define FS 10000
#define PTS 256
#define PI 3.14159265358979

#pragma DATA_SECTION(spec,".temp") //.temp is a segment defined in IRAM
#pragma DATA_SECTION(x,".SDRAM$heap") // declaration to store in SDRAM
#pragma DATA_SECTION(nx1,".SDRAM$heap")
#pragma DATA_SECTION(nx12,".SDRAM$heap")
#pragma DATA_SECTION(win,".SDRAM$heap")
#pragma DATA_SECTION(win1,".SDRAM$heap")
#pragma DATA_SECTION(win12,".SDRAM$heap")
#pragma DATA_SECTION(w,".temp")
#pragma DATA_SECTION(samples,".temp")
#pragma DATA_SECTION(deny,".new") //.new is a segment in IRAM
#pragma DATA_SECTION(grant,".new")

typedef struct {float real, imag;} COMPLEX;
void FFT_init(float *); // Declaration of functions for calculating FFT
void FFT(COMPLEX *, int );
int maxim(float *, int ); // Function to evaluate dominant frequency

float win[256], win1[2500], win12[2500]; // Declaration of all the variables in use
float x[10000], deny[8000], grant[8000], spec[256];
float nx1[2500], nx12[2500];
COMPLEX w[PTS]; // Twiddle factors defined as objects of a complex structure
COMPLEX samples[PTS];

int i = 0,num = 0, j, temp, N, k, d[12];

/* The following section is the initialization of the codec, the main change which
was made by us is to make parameter 4 – 0x0015 to enable analog audio path
```

control, The first four parameters are varied till the input and output speech is clearly audible */

```
DSK6713_AIC23_Config config = { \
   0x0017,  /* 0 DSK6713_AIC23_LEFTINVOL  Left line input channel volume */ \
   0x0017,  /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume */\
   0x00e0,  /* 2 DSK6713_AIC23_LEFTHPVOL  Left channel headphone volume */  \
   0x00e0,  /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume */ \
   0x0015,  /* 4 DSK6713_AIC23_ANAPATH    Analog audio path control */      \
   0x0000,  /* 5 DSK6713_AIC23_DIGPATH    Digital audio path control */     \
   0x0000,  /* 6 DSK6713_AIC23_POWERDOWN  Power down control */             \
   0x0043,  /* 7 DSK6713_AIC23_DIGIF      Digital audio interface format */ \
   0x0081,  /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */            \
   0x0001   /* 9 DSK6713_AIC23_DIGACT     Digital interface activation */   \
};


void main()
{
   DSK6713_AIC23_CodecHandle hCodec;

   Uint32 l_input, r_input,l_output, r_output;

   int i;
   /* Initialize the board support library, must be called first */
   DSK6713_init();

   /* Start the codec */
   hCodec = DSK6713_AIC23_openCodec(0, &config);

   DSK6713_AIC23_setFreq(hCodec, 1); // Sampling frequency : 1 = 8000 Hz

   for(i = 0 ; i < 8000 ; i++)
     {       /* Read a sample to the left channel */
                   while (!DSK6713_AIC23_read(hCodec, &l_input));

                   /* Read a sample to the right channel */
                   while (!DSK6713_AIC23_read(hCodec, &r_input));


                          deny [i]=l_input; // The uttered signal is sent out through
                          l_output= deny [i]; //the speakers so that the user can hear
                          r_output= deny [i]; // what is uttered


                   /* Send a sample to the left channel */
        while (!DSK6713_AIC23_write(hCodec, l_output));

        /* Send a sample to the right channel */
        while (!DSK6713_AIC23_write(hCodec, r_output));
      }
```

```
  /* Close the codec */
  DSK6713_AIC23_closeCodec(hCodec);

}
```

7.2.2 Training Phase

```
#include"spchcfg.h"

#include"dsk6713.h"

#include"dsk6713_aic23.h"

#include"stdio.h"

#include"math.h"

#define FS 10000
#define PTS 256
#define PI 3.14159265358979

#pragma DATA_SECTION(x,".SDRAM$heap") // Define all the array pointers
in #pragma DATA_SECTION(spec,".temp")        // the various segments of
memory
#pragma DATA_SECTION(nx1,".SDRAM$heap")
#pragma DATA_SECTION(nx12,".SDRAM$heap")
#pragma DATA_SECTION(win,".SDRAM$heap")
#pragma DATA_SECTION(win1,".SDRAM$heap")
#pragma DATA_SECTION(win12,".SDRAM$heap")
#pragma DATA_SECTION(w,".temp")// Define segment .temp in IRAM
#pragma DATA_SECTION(samples,".temp")
#pragma DATA_SECTION(deny,".new") // Define segment .new in SDRAM
#pragma DATA_SECTION(grant,".new")


typedef struct {float real, imag;} COMPLEX;
void FFT_init(float *);// Declaration of functions for calculating FFT
void FFT(COMPLEX *, int );
int maxim(float *, int ); // Function to evaluate dominant frequency

float win[256], win1[2500], win12[2500]; // Declaration of all the variables in
use
float x[10000], spec[256],deny[8000],grant[8000];
float nx1[2500],nx12[2500];
COMPLEX w[PTS]; ]; //Twiddle factors defined as objects of a structure
COMPLEX samples[PTS];

int i = 0, j, temp, N, k, d[12];
```

```
/* Initialization of the codec*/
DSK6713_AIC23_Config config = { \
   0x0017,  /* 0 DSK6713_AIC23_LEFTINVOL  Left line input channel volume */ \
   0x0017, /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume */\
   0x00e0, /* 2 DSK6713_AIC23_LEFTHPVOL  Left channel headphone volume */  \
   0x00e0, /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume*/
   0x0015,  /* 4 DSK6713_AIC23_ANAPATH    Analog audio path control */  \
   0x0000,  /* 5 DSK6713_AIC23_DIGPATH    Digital audio path control */  \
   0x0000,  /* 6 DSK6713_AIC23_POWERDOWN  Power down control */   \
   0x0043,  /* 7 DSK6713_AIC23_DIGIF      Digital audio interface format */ \
   0x0081,  /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */   \
   0x0001   /* 9 DSK6713_AIC23_DIGACT     Digital interface activation */   \
};


void main()
{
   DSK6713_AIC23_CodecHandle hCodec;

   Uint32 l_input, r_input,l_output, r_output;

   int i;
   /* Initialize the board support library, must be called first */
   DSK6713_init();

   /* Start the codec */
   hCodec = DSK6713_AIC23_openCodec(0, &config);

   DSK6713_AIC23_setFreq(hCodec, 1);//sampling frequency = 8000Hz
   for(i = 0 ; i < 10000 ; i++)
     {
                          /* Read a sample to the left channel */
                            while (!DSK6713_AIC23_read(hCodec, &l_input));

                            /* Read a sample to the right channel */
                            while (!DSK6713_AIC23_read(hCodec, &r_input));


                                     x[i]=l_input;
                                     l_output=x[i];
                                     r_output=x[i];


        /* Send a sample to the left channel */
        while (!DSK6713_AIC23_write(hCodec, l_output));

        /* Send a sample to the right channel */
        while (!DSK6713_AIC23_write(hCodec, r_output));
     }
```

```
/* Close the codec */
DSK6713_AIC23_closeCodec(hCodec);

 /* Silence detection to separate out just the speech signals */
       i = 0;
       while(abs(x[i]) < 0.12)
            i++;
       temp =  i;
       for(j = 0 ; j < 10000 - temp ; j++)
           x[j] = x[i++];
       for(j = 10000 - temp ; j < 10000 ; j++)
           x[j] = 0;

      /* Defining the hamming window */
       for(i = 0 ; i < 256 ; i++)
           win[i] = 0.53836 - 0.46164 * cos((2 * 3.14 * i)/(256));


      hCodec = DSK6713_AIC23_openCodec(0, &config);

     DSK6713_AIC23_setFreq(hCodec, 1);

     /* Output the separated out speech samples to make
          sure it is stored properly */
     for(i = 0 ; i < 2500 ; i++)
     {

                    l_output=x[i];
                    r_output=l_output;


       /* Send a sample to the left channel */
       while (!DSK6713_AIC23_write(hCodec, l_output));

       /* Send a sample to the right channel */
       while (!DSK6713_AIC23_write(hCodec, r_output));
    }

   /* Close the codec */
   DSK6713_AIC23_closeCodec(hCodec);

   /* Defining the overlapping 256 point hamming windows in time domain */
   for(k = 0 ; k < 11 ; k++)
   {
          for(j = 0 ; j < (200 * k) ; j++)
                 win1[j] = 0;
          for(j = (200 * k) ; j < ((200 * k) + 256) ; j++)
                 win1[j] = win[j - (200 * k)];
          for(j = ((200 * k) + 256) ; j < 2500 ; j++)
                 win1[j]=0;
```

```
                for(i = 0 ; i < 2500 ; i++)
                        nx1[i] = x[i] * win1[i];
                i = 0;
                while(nx1[i] == 0)
                       i++;
               temp = i;
               for(j = 0 ; j < 2500 - temp ; j++)
                     nx1[j] = nx1[i++];
               for(j = 2500 - temp ; j < 2500 ; j++)
                     nx1[j] = 0;
               FFT_init(nx1);
              //find magnitude of fft and put it in the array X
              //find position of max ampitude in the spectrum
             d[k] = maxim(spec, 256);

        }
             for(j = 0 ; j < 2244 ; j++)
                    win12[j]=0;
             for(j = 2244 ; j < 2500 ; j++)
                     win12[j] = win[j - 2244];

             for(i = 0 ; i < 2500;i++)
                      nx12[i] = x[i] * win12[i];
             i = 0;
             while(nx12[i] == 0)
                 i++;
             temp =  i;
             for(j = 0 ; j < 2500 - temp ; j++)
                  nx12[j] = nx12[i++];
             for(j = 2500 - temp ; j < 2500 ; j++)
                  nx12[j] = 0;
             FFT_init(nx12);
             //find magnitude of fft and put it in the array X
             //find position of max ampitude in the spectrum
             d[11] = maxim(spec, 256);

}

/* Function to evaluate the FFT */
void FFT_init(float *x1)
{
             for(i = 0 ; i < PTS ; i++)
             {
                  w[i].real = cos(2 * PI * i / (PTS * 2.0)); // Twiddle factors
                  w[i].imag = -sin(2 * PI * i / (PTS * 2.0));
             }
             for(i = 0 ; i < PTS ; i++)
             {
                  samples[i].real = 0;
```

```
            samples[i].imag = 0;
        }
        for(i = 0 ; i < PTS ; i++)
        {
           samples[i].real = x1[i];
        }
        for(i = 0 ; i < PTS ; i++)
           samples[i].imag = 0;
           FFT(samples, PTS);
        for(i = 0 ; i < PTS ; i++) // Evaluate and store the magnitude of the FFT
        {
          spec[i] = sqrt(samples[i].real * samples[i].real + samples[i].imag *
                       samples[i].imag);
        }
return;
}

void FFT(COMPLEX *Y, int N)
{
    COMPLEX temp1, temp2;
    int i, j, k;
    int upper_leg, lower_leg;
    int leg_diff;
    int num_stages = 0;
    int index, step;
    i = 1;

   do
   {
     num_stages += 1;
     i = i * 2;
   }while(i != N);
   leg_diff = N / 2;
   step = (PTS * 2) / N;
   for(i = 0; i < num_stages ; i++)
  {
     index = 0;
     for(j = 0 ; j < leg_diff ; j++)
    {
       for(upper_leg = j ; upper_leg < N ; upper_leg += (2 * leg_diff))
       {
           lower_leg = upper_leg + leg_diff;
           temp1.real = (Y[upper_leg]).real + (Y[lower_leg]).real;
           temp1.imag = (Y[upper_leg]).imag + (Y[lower_leg]).imag;
           temp2.real = (Y[upper_leg]).real - (Y[lower_leg]).real;
           temp2.imag = (Y[upper_leg]).imag - (Y[lower_leg]).imag;
           (Y[lower_leg]).real = temp2.real * (w[index]).real - temp2.imag *
                               (w[index]).imag;
           (Y[lower_leg]).imag = temp2.real * (w[index]).imag + temp2.imag *
                                (w[index]).real;
            (Y[upper_leg]).real = temp1.real;
            (Y[upper_leg]).imag = temp1.imag;
        }
   index += step;
   }
   leg_diff = leg_diff / 2;
   step *= 2;
 }
 j = 0;
 for(i = 0 ; i < (N-1) ; i++)
 {
     k = N / 2;
     while(k <= j)
     {
        j = j - k;
        k = k / 2;
     }
     j = j + k;

     if(i < j)
     {
         temp1.real = (Y[j]).real;
         temp1.imag = (Y[j]).imag;
         (Y[j]).real = (Y[i]).real;
         (Y[j]).imag = (Y[i]).imag;
         (Y[i]).real = temp1.real;
         (Y[i]).imag = temp1.imag;
     }
 }
return;
}

/*Function to evaluate the dominant frequency component in the spectrum */
int maxim(float *a, int length)
{
  float temp;
  int pos;
  temp = a[1];
  pos = 1;
  for(i = 2 ; i < length / 2 ; i++)
  {
    if(temp < a[i])
    {
       temp = a[i];
       pos = i;
    }
  }
  return pos;
}
```

7.2.3 Testing Phase

```
#include"spchcfg.h"

#include"dsk6713.h"

#include"dsk6713_aic23.h"

#include"stdio.h"

#include"math.h"

#define FS 10000
#define PTS 256
#define PI 3.14159265358979

#pragma DATA_SECTION(x,".SDRAM$heap") // Define all the array pointers
#pragma DATA_SECTION(spec,".temp")     // in the various memory segments
#pragma DATA_SECTION(nx1,".SDRAM$heap")
#pragma DATA_SECTION(nx12,".SDRAM$heap")
#pragma DATA_SECTION(win,".SDRAM$heap")
#pragma DATA_SECTION(win1,".SDRAM$heap")
#pragma DATA_SECTION(win12,".SDRAM$heap")
#pragma DATA_SECTION(w,".temp")// Define segment .temp in IRAM
#pragma DATA_SECTION(samples,".temp")
#pragma DATA_SECTION(deny,".new") // Define segment .new in SDRAM
#pragma DATA_SECTION(grant,".new")

typedef struct {float real, imag;} COMPLEX;
void FFT_init(float *);// Declaration of functions for calculating FFT
void FFT(COMPLEX *, int );
int maxim(float *, int ); // Function to evaluate dominant frequency

float win[256], win1[2500], win12[2500]; // Declaration of all the variables in
use
float x[10000], spec[256],deny[8000],grant[8000];
float nx1[2500],nx12[2500];
COMPLEX w[PTS]; ]; //Twiddle factors defined as objects of a structure
COMPLEX samples[PTS];

int i = 0,num = 0, j, temp, N, k, d[12], dd[12];
int *ptr;
float *out;

/* Initialization of the codec*/
DSK6713_AIC23_Config config = { \
    0x0017,  /* 0 DSK6713_AIC23_LEFTINVOL  Left line input channel volume */ \
    0x0017,  /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume */\
    0x00e0,  /* 2 DSK6713_AIC23_LEFTHPVOL  Left channel headphone volume */  \
    0x00e0,  /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume*/
```

```
   0x0015,  /* 4 DSK6713_AIC23_ANAPATH    Analog audio path control */   \
   0x0000,  /* 5 DSK6713_AIC23_DIGPATH    Digital audio path control */   \
   0x0000,  /* 6 DSK6713_AIC23_POWERDOWN  Power down control */    \
   0x0043,  /* 7 DSK6713_AIC23_DIGIF      Digital audio interface format */ \
   0x0081,  /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */   \
   0x0001   /* 9 DSK6713_AIC23_DIGACT     Digital interface activation */   \
};

void main()
{
   DSK6713_AIC23_CodecHandle hCodec;

   Uint32 l_input, r_input,l_output, r_output;

   int i;
   /* Initialize the board support library, must be called first */
   DSK6713_init();

   /* Start the codec */
   hCodec = DSK6713_AIC23_openCodec(0, &config);

   DSK6713_AIC23_setFreq(hCodec, 1);
   for(i = 0 ; i < 10000 ; i++)
     {
              /* Read a sample to the left channel */
              while (!DSK6713_AIC23_read(hCodec, &l_input));

              /* Read a sample to the right channel */
              while (!DSK6713_AIC23_read(hCodec, &r_input));


                    x[i]=l_input;
                    l_output=x[i];
                    r_output=x[i];


              /* Send a sample to the left channel */
          while (!DSK6713_AIC23_write(hCodec, l_output));

          /* Send a sample to the right channel */
        while (!DSK6713_AIC23_write(hCodec, r_output));
     }

     /* Close the codec */
    DSK6713_AIC23_closeCodec(hCodec);

   /* Silence detection to separate out just the speech signals */
       i = 0;
       while(abs(x[i]) < 0.12)
          i++;
       temp =  i;
```

```
        for(j = 0 ; j < 10000 - temp ; j++)
            x[j] = x[i++];
        for(j = 10000 - temp ; j < 10000 ; j++)
            x[j] = 0;

/* Defining the hamming window */
        for(i = 0 ; i < 256 ; i++)
            win[i] = 0.53836 - 0.46164 * cos((2 * 3.14 * i)/(256));


      hCodec = DSK6713_AIC23_openCodec(0, &config);

      DSK6713_AIC23_setFreq(hCodec, 1);

      /* Output the separated out speech samples to make
             sure it is stored properly */
      for(i = 0 ; i < 2500 ; i++)
     {

                   l_output=x[i];
                   r_output=l_output;



          /* Send a sample to the left channel */
          while (!DSK6713_AIC23_write(hCodec, l_output));

          /* Send a sample to the right channel */
          while (!DSK6713_AIC23_write(hCodec, r_output));
     }

    /* Close the codec */
    DSK6713_AIC23_closeCodec(hCodec);

 /* Defining the overlapping 256 point hamming windows in time domain */
  for(k = 0 ; k < 11 ; k++)
 {
          for(j = 0 ; j < (200 * k) ; j++)
                 win1[j] = 0;
          for(j = (200 * k) ; j < ((200 * k) + 256) ; j++)
                 win1[j] = win[j - (200 * k)];
          for(j = ((200 * k) + 256) ; j < 2500 ; j++)
                 win1[j]=0;

          for(i = 0 ; i < 2500 ; i++)
                 nx1[i] = x[i] * win1[i];
          i = 0;
          while(nx1[i] == 0)
                i++;
         temp = i;
         for(j = 0 ; j < 2500 - temp ; j++)
```

```
           nx1[j] = nx1[i++];
        for(j = 2500 - temp ; j < 2500 ; j++)
            nx1[j] = 0;
        FFT_init(nx1);
       //find magnitude of fft and put it in the array X
       //find position of max ampitude in the spectrum
      d[k] = maxim(spec, 256);

}
      for(j = 0 ; j < 2244 ; j++)
            win12[j]=0;
      for(j = 2244 ; j < 2500 ; j++)
             win12[j] = win[j - 2244];

      for(i = 0 ; i < 2500;i++)
              nx12[i] = x[i] * win12[i];
      i = 0;
      while(nx12[i] == 0)
          i++;
      temp =  i;
      for(j = 0 ; j < 2500 - temp ; j++)
           nx12[j] = nx12[i++];
      for(j = 2500 - temp ; j < 2500 ; j++)
           nx12[j] = 0;
      FFT_init(nx12);
      //find magnitude of fft and put it in the array X
      //find position of max ampitude in the spectrum
      d[11] = maxim(spec, 256);


      ptr = (int *)0x0002cc04; /* pointer to location where the feature vector
                                             from training phase is stored */
      for(i = 0 ; i < 12 ; i++)
      {
           dd[i] = *ptr++; /* Fetch the feature vector from training and store it*/

      }
      for(i=0;i<12;i++)
      if(abs(d[i] - dd[i]) < 10) // Check the deviation between dominant
           num++;                    // frequency within each window
       if(num>7)
      {
           out = (float *)0x00007f00; // Store pointer to memory location where
      }                                          // the “Access Granted” signal was stored
      else
      {
           out = (float *)0x00000200; // Store pointer to memory location where
      }                                          // the “Access Denied” signal was stored

     hCodec = DSK6713_AIC23_openCodec(0, &config);
```

```
        DSK6713_AIC23_setFreq(hCodec, 1);

         /* Output "Access Granted" or "Access Denied" as an indication of
            correct recognition or a failure */
        for(i = 0 ; i < 8000 ; i++)
        {
                        l_output= *(out++);
                        r_output=l_output;


            /* Send a sample to the left channel */
            while (!DSK6713_AIC23_write(hCodec, l_output));

             /* Send a sample to the right channel */
            while (!DSK6713_AIC23_write(hCodec, r_output));
         }

        /* Close the codec */
        DSK6713_AIC23_closeCodec(hCodec);
 }

/* Function to evaluate the FFT */
void FFT_init(float *x1)
{
         for(i = 0 ; i < PTS ; i++)
         {
            w[i].real = cos(2 * PI * i / (PTS * 2.0));
            w[i].imag = -sin(2 * PI * i / (PTS * 2.0));
         }
         for(i = 0 ; i < PTS ; i++)
         {
            samples[i].real = 0;
            samples[i].imag = 0;
         }
         for(i = 0 ; i < PTS ; i++)
         {
           samples[i].real = x1[i];
         }
         for(i = 0 ; i < PTS ; i++)
           samples[i].imag = 0;
         FFT(samples,PTS);
         for(i = 0 ; i < PTS ; i++) // Evaluate and store the magnitude of the FFT
         {
           spec[i] = sqrt(samples[i].real * samples[i].real + samples[i].imag *
                  samples[i].imag);
         }
return;
}
```

```
void FFT(COMPLEX *Y, int N)
{
  COMPLEX temp1, temp2;
  int i, j, k;
  int upper_leg, lower_leg;
  int leg_diff;
  int num_stages = 0;
  int index, step;
  i = 1;

  do
  {
     num_stages += 1;
     i = i * 2;
  }while(i != N);
  leg_diff = N / 2;
  step = (PTS * 2) / N;
  for(i = 0; i < num_stages ; i++)
  {
     index = 0;
     for(j = 0 ; j < leg_diff ; j++)
    {
        for(upper_leg = j ; upper_leg < N ; upper_leg += (2 * leg_diff))
        {
           lower_leg = upper_leg + leg_diff;
           temp1.real = (Y[upper_leg]).real + (Y[lower_leg]).real;
           temp1.imag = (Y[upper_leg]).imag + (Y[lower_leg]).imag;
           temp2.real = (Y[upper_leg]).real - (Y[lower_leg]).real;
           temp2.imag = (Y[upper_leg]).imag - (Y[lower_leg]).imag;
           (Y[lower_leg]).real = temp2.real * (w[index]).real - temp2.imag *
                                   (w[index]).imag;
           (Y[lower_leg]).imag = temp2.real * (w[index]).imag + temp2.imag *
                                    (w[index]).real;
           (Y[upper_leg]).real = temp1.real;
           (Y[upper_leg]).imag = temp1.imag;
        }
   index += step;
   }
   leg_diff = leg_diff / 2;
   step *= 2;
 }
 j = 0;
 for(i = 0 ; i < (N-1) ; i++)
 {
    k = N / 2;
    while(k <= j)
    {
       j = j - k;
       k = k / 2;
    }
```

```
     j = j + k;

     if(i < j)
     {
          temp1.real = (Y[j]).real;
          temp1.imag = (Y[j]).imag;
          (Y[j]).real = (Y[i]).real;
          (Y[j]).imag = (Y[i]).imag;
          (Y[i]).real = temp1.real;
          (Y[i]).imag = temp1.imag;
     }
 }
return;
}

/*Function to evaluate the dominant frequency component in the spectrum */
int maxim(float *a, int length)
{
  float temp;
  int pos;
  temp = a[1];
  pos = 1;
  for(i = 2 ; i < length / 2 ; i++)
  {
    if(temp < a[i])
    {
       temp = a[i];
       pos = i;
    }
  }
  return pos;
}
```

8. RESULTS AND CONCLUSIONS

8.1 OVERVIEW

This project basically involved carrying out a research on the existing techniques of isolated word recognition, simulating the work using the available tools like MATLAB and finally carrying out the implementation on the TMS320C6713 DSP platform. The experiments were carried out by taking real time data from a microphone making the ASR system Real Time. The results have been quantified by carrying out the experiments under varying conditions and multitude of speakers, male/female.

An interesting aspect which came out of the experiments was that the MFCC approach failed to give favorable results with respect to a few test cases where the differing phenomes were very few as the difference in energy between two such words ("h-ello" and "f-ellow") showed couldn't be caught as the energy levels associated with the reminder of the phenomes constituting the word tend to average out the difference. The Time domain approach in our case proved to be more trustworthy. The implementation on the TMS320C6713 platform also threw up a number of challenges like making the algorithms compliant for an efficient implementation on the DSP with respect to aspects like instruction execution efficiency, Memory usage and we were able to incorporate some aspects which would help achieve this. The results proved to be quite satisfactory considering the amount of storage requirement needed. Hence, the basic aim towards which our research was directed i.e. designing a Real Time ASR system was primarily met but the prospect of making it robust always remains. The constraints before us made us leave this aspect for the future researchers.

When the robustness of a speech recognition system is under question, we need to realize that such a system it has got a few disadvantages. On one hand the human voice is not invariant in time therefore the biometric template must be adapted during progressing time. The human voice is also variable through temporal variations of the voice, caused by a cold, hoarseness, stress, emotional different states or puberty vocal change. On the other hand voice recognition systems have got a higher error rate compared to fingerprint recognition systems, because the human voice is not as "unique" as fingerprints. For the computation of the Fast Fourier Transformation the systems needs to have a co-processor and more processing power than for e.g fingerprint matching. Therefore speaker verification systems are not suitable for mobile applications / battery powered systems in the current state.

8.2 COMPARISON

We successfully simulated the MFCC approach and the Time domain approach using MATLAB. We concluded from our experiments that for an isolated word recognition system like the one we aimed to implement in our project, the time domain approach proved to be more effective. The reason for this is that, the MFCC approach has a drawback. As explained earlier the key to this approach is using the energies, however, this may not be the best approach as was discovered.

It was seen from the experiments that because of the prominence given to energy, this approach failed to recognize the same word uttered with different energy. Also, as this takes the summation of the energy within each triangular window it would essentially give the same value of energy irrespective of whether the spectrum peaks at one particular frequency and falls to lower values around it or whether it has an equal spread within the window.

However, as the time domain approach is not based on energy but purely based on the dominant frequencies within small segments of speech, it makes good use of the quasi – stationary property of speech. We thus concluded from our simulation results that the time domain approach is more suitable for the isolated work recognizer required for a voice based biometric system. A comparison between the algorithms we simulated is given in the tables below. Based on this conclusion, we decided to implement the time domain approach on the DSK.

MFCC Approach:

The following test results were obtained by simulating the MFCC based recognition approach on MATLAB. The entries indicate the MSE obtained for four users who tested the System. A threshold MSE of 1 was set for this experiment.

Diwakar	**Adarsh**	**Deepak**	**Karthik**
1.0859	0.3507	0.8546	0.9821
0.3354	0.3983	0.6148	1.7451
0.3288	0.4055	0.3465	0.8956
0.8407	0.3018	2.0014	0.7998
0.4480	0.2232	0.4269	0.9324

0.5672	1.4045	0.3458	1.6542
0.4181	0.3946	0.5982	0.5421
0.5288	0.7628	0.3991	0.9654
1.6195	0.5548	0.4651	0.6495
0.4471	0.4912	0.8651	0.5981
0.4887	0.6149	0.4438	0.8495
0.3610	0.3674	0.3215	0.2136
0.9020	0.6713	1.3201	0.9101
0.3661	0.2025	0.5231	0.2631
0.5645	0.3948	0.6723	0.6498
0.4213	0.4532	0.2132	2.2651
0.4656	1.2063	0.3135	0.4561
0.3263	0.2123	0.4136	0.6462
0.9262	0.5631	1.7512	0.7412
0.6298	0.3516	0.3212	0.6521

Time Domain Approach:

The following test results were obtained by simulating the Time Domain based recognition approach on MATLAB. The entries indicate the number of dominant frequency matches obtained for four users who tested the System. A threshold of 8 matches was set for this experiment.

Diwakar	**Adarsh**	**Deepak**	**Karthik**
11	10	10	11
11	12	10	8
8	9	9	10
10	11	11	10
12	7	9	9
9	10	6	5
10	8	9	11
8	11	10	12
12	12	10	11

9	9	11	9
11	8	7	10
6	10	8	9
11	11	10	7
10	10	12	9
11	12	11	12
10	8	9	11
8	11	10	10
12	12	12	11
10	9	11	9
11	8	9	10

Results of the simulation:

The following table shows the percentage successful speaker recognition for both the algorithms.

	Adarsh	**Deepak**	**Diwakar**	**Karthik**
MFCC Approach	**90%**	**85%**	**90%**	**85%**
Time Domain Approach	**95%**	**90%**	**95%**	**90%**

9. APPLICATIONS

After nearly sixty years of research, speech recognition technology has reached a relatively high level. However, most state-of-the-art ASR systems run on desktop with powerful microprocessors, ample memory and an ever-present power supply. In these years, with the rapid evolvement of hardware and software technologies, ASR has become more and more expedient as an alternative human-to-machine interface that is needed for the following application areas:

- Stand-alone consumer devices such as wrist watch, toys and hands-free mobile phone in car where people are unable to use other interfaces or big input platforms like keyboards are not available.

- Single purpose command and control system such as voice dialing for cellular, home, and office phones where multi-function computers (PCs) are redundant.

Some of the applications of speaker verification systems are:

- Time and Attendance Systems
- Access Control Systems
- Telephone-Banking/Broking
- Biometric Login to telephone aided shopping systems
- Information and Reservation Services
- Security control for confidential information
- Forensic purposes

Voice based Telephone dialing is one of the applications we simulated. The key focus of this application is to aid the physically challenged in executing a mundane task like telephone dialing. Here the user initially trains the system by uttering the digits from 0 to 9. Once the system has been trained, the system can recognize the digits uttered by the user who trained the system. This system can also add some inherent security as the system based on cepstral approach is speaker dependent. The algorithm is run on a particular speaker and the MFCC coefficients determined. Now the algorithm is applied to a different speaker and the mismatch was clearly observed. Thus the inherent security provided by the system was confirmed.

Presently systems have also been designed which incorporate Speech and Speaker Recognition. Typically a user has two levels of check. He/She has to initially speak the right password to gain access to a system. The system not only verifies if the correct password has been said but also focused on the authenticity of the speaker. The ultimate goal is do have a system which does a Speech, Iris, Fingerprint Recognition to implement access control.

10. SCOPE FOR FUTURE WORK

This project focused on “Isolated Word Recognition”. But we feel the idea can be extended to “Continuous Word Recognition” and ultimately create a Language Independent Recognition System based on algorithms which make these systems robust. The use of Statistical Models like HMMs, GMMs or learning models like Neural Networks and other associated aspects of Artificial Intelligence can also be incorporated in this direction to improve upon the present project. This would make the system much tolerant to variations like accent and extraneous conditions like noise and associated residues and hence make it less error prone. Some other aspects which can be looked into are:

- The end-point detection used in this work is only based on the frame energy which is not good for a noisy environment with low SNR. The error rate of determining the beginning and ending of speech segments will greatly increase which directly influence the recognition performance at the pattern recognition part. So, we should try to use some effective way to do end-point detection. One of these methods could be to use the statistical way to find a distribution which can separate the noise and speech from each other.
- The size of the training data i.e. the code book can be increased as it is clearly proven that the greater the size of the training data, the greater the recognition accuracy. This training data could incorporate aspects like the different ways viz the accents in which a word can be spoken, the same words spoken by male/female speakers and the word being spoken under different conditions say under conditions in which the speaker may have a sore throat etc.
- Although some methods have been proposed and used with respect the handling of the input and the processed samples, there maybe some other optimizations that we can apply before finally storing it in the available memory.
- The present work has been implemented on the TMS320C6713 platform which is a floating point DSP. The case where a fixed point implementation on a DSP (with suitable word size which maintains accuracy without causing truncation errors) is done will have to be studied and a comparison carried out so as to make a choice on the DSP to be used for such applications. This is necessary as Embedded Systems have “Response Time” requirements as one of their essential criteria.

REFERENCES

[1] Lawrence Rabiner, Biing-Hwang Juang – '*Fundamentals of Speech Recognition'*

[2] Wei Han, Cheong-Fat Chan, Chiu-Sing Choy and Kong-Pang Pun – '*An Efficient MFCC Extraction Method in Speech Recognition'*, Department of Electronic Engineering, The Chinese University of Hong Kong, Hong, IEEE – ISCAS, 2006

[3] Leigh D. Alsteris and Kuldip K. Paliwal – '*ASR on Speech Reconstructed from Short- time Fourier Phase Spectra',* School of Microelectronic Engineering Griffth University, Brisbane, Australia, ICLSP - 2004

[4] Waleed H. Abdulla – '*Auditory Based Feature Vectors for Speech Recognition Systems'*, Electrical & Electronic Engineering Department, The University of Auckland

[5] Pradeep Kumar P and Preeti Rao – '*A Study of Frequency-Scale Warping for Speaker Recognition'*, Dept of Electrical Engineering, IIT- Bombay, National Conference on Communications, NCC 2004, IISc Bangalore, Jan 30 -Feb 1, 2004

[6] Beth Logan – '*Mel Frequency Cepstral Coefficients for Music Modeling'*, Cambridge Research Laboratory, Compaq Computer Corporation

[7] MIT Open Courseware