A Project Report

On

SPEAKER RECOGINITION SYSTEM USING TMS320C6713DSK

At

Dharmsinh Desai University, Nadiad

by

Nisha Desai

EC-23, ID - 15ECUBS120

B.Tech. SEM. VIII

Faculty Supervisor
Dr. Vinay M. Thumar

Guide
Dr. Hardip K. Shah

In Partial Fulfillment of Requirement of Bachelor of Technology

Degree of Electronics & Communication Course

Submitted To

Department of Electronics & Communication Engineering

Faculty of Technology,

Dharmsinh Desai University, Nadiad-387001.

(April 2020)

CERTIFICATE

This is to certify that Miss. **Nisha Desai** (ID NO: 15ECUBS120) Roll number EC-23 studying in B.Tech. Sem. VIII (Electronics & Communication Engineering) of Faculty of Technology, D. D. University has satisfactorily delivered seminars and has satisfactorily completed term work in the subject of Industrial Training / Project. (titled: **speaker recognition system using TMS320C6713DSK**).

Date: 28th march 2020

Dr. Hardip K. Shah	Dr. Vinay M. Thumar	Dr. Nikhil Kothari
(Guide)	(Faculty Supervisor)	(Head of Department)

ACKNOWLEDGMENT

I would like to take this wonderful opportunity of doing B.Tech Project on "Speaker Recognition on TMS320C6713" which has given me a lot of learning and practical exposure.

I would like to thank all the people, involved directly or indirectly in this training session, who encouraged me and appreciated my decision to undertake training at Dharmsinh Desai University, Nadiad

I would like to thank my guide **Dr. Hardip K. Shah** for helping me to have a complete detailed study of the speaker recognition system and DSP kit(DSK6713). It was due to their constant guidance and supervision that i was able to understand all the concepts thoroughly and complete the mini projects in the digital signal processing module undertaken during the study.

I would like to express my sincere thanks to my internal guide **Dr. Vinay M. Thumar** for encouraging and appreciating my study project. I would also like to express my sincere thanks to **Dr. Nikhil Kothari**, Head of the Department of Electronics & Communication for giving me opportunity to work as a trainee at Dharmsinh Desai University.

Finally, I would like to thank my family, all my colleagues at the training center for being helpful at each and every phase of the training and guiding me in the right direction.

Nisha Desai
(EC-23, ID-15ECUBS120)

SYNOPSIS

Speaker Recognition is the process of automatically recognizing a certain word spoken by a particular speaker based on information included in speech waves. This technique makes it possible to use the speakers 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. Different features of speech signal such as Linear Prediction Coding (LPC), Mel-Frequency Cepstrum Coefficients (MFCC), and others. MFCC is perhaps the best known and most popular due to MFCC computation is a replication of the human hearing system intending to artificially implement the ear's working principle with the assumption that the human ear is a reliable speaker recognizer Therefore, this project implements MFCC based technique to extract the feature set from a speech signal over MATLAB and DSK6713.

In this project, we implemented 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 studied the MFCC approach with the Frequency domain approach for recognition by simulating technique using MATLAB R15b and analyzing the consistency of recognition by this technique. To implement this technique on hardware we have studied processor architecture of a DSP processor TMS320C6713 and the peripherals of DSK6713. We have also studied the development platform Code Composer Studio (CCS).

Our aim is to develop software to recognize the speech samples from different users so as to restrict access to a predefined set of users. For this purpose, we form a database of different speech samples. The MFCCs for a particular Speech signal is unique for every individual. Therefore every such signal will generate different MFCCs. These are then compared with the previously stored MFCCs of signals to check if any match is found. For real time processing of Speech signal, fast processors like Digital Signal Processors are required.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER – 1

SPEAKER RECOGNITION SYSTEM

1.1 Problem definition & Design specifications

Suppose we are having audio data which is recorded every day for several years. Out of these data we want to find out which one corresponds to a particular speaker. This can be done by using a speaker recognition system. Consider a case of video conferencing, where we want to focus a camera on a particular person. This can be easily implemented if a speaker recognition system exists. Automatic speaker verification and automatic speaker identification systems are the most economical solution to the problem of unauthorized computer access and other digital equipments hacking involving communications.

➢ Design specifications

1. TMS320C6713 DSK board

- Clock frequency 225 MHz
- Floating point DSP
- EDMA Controller
- AIC23 Stereo Codec
    - 16 bit ADC
    - 16 bit DAC
- 4 user accessible LEDs and DIPs switches
- VLIW architecture
- 192Kbytes Internal memory
- 16 Mbytes of synchronous DRAM
- 4Kb program /data cache
- 2 Timers
- 2 McBSPs (Multichannel buffer serial ports)

2. Software:

- MATLAB 2015b
- Code Composer Studio v5.3

1.2 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 speak writer. 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.

Speaker recognition is the process of automatically recognizing who is speaking on the basis of the information contained in the speech signal. Speaker recognition is carried out in two phases. First phase comprises the training phase whereas, the second phase comprises the testing phase. In the training phase the speaker has to utter something so as to provide speech samples. These samples enable the system to build a model of that speaker. In the testing phase, the input speech is matched with the stored reference models and a decision is made. Speaker recognition can be said as a biometric system which validates a person's claim to an identity based on the features extracted from the speech samples. The training and testing phase of speaker recognition system makes it possible to use the speaker's voice to verify his/her identity and control access to services such as voicedialing, telephone banking, telephone shopping, voice mail and other computer access services that are voice activated.

1.3 Principle of Speaker Recognition:

Speaker recognition methods can be divided into text-independent and textdependent methods. In a text-independent system, speaker models capture characteristics of somebody's speech which show up irrespective of what one is saying. In a textdependent system, on the other hand, the recognition of the speaker's identity is based on history her speaking one or more specific phrases, like passwords, card numbers, PIN codes, etc. Every technology of speaker recognition, identification and verification, whether text-independent and text dependent, each has its own advantages and disadvantages and may require different

treatments and techniques. The choice of which technology to use is application-specific. At the highest level, all speaker recognition systems contain two main modules feature extraction and feature matching.

- Speaker recognition can be classified into two categories:

1.) Speaker Identification: In speaker identification we find out which speaker has uttered the given speech.

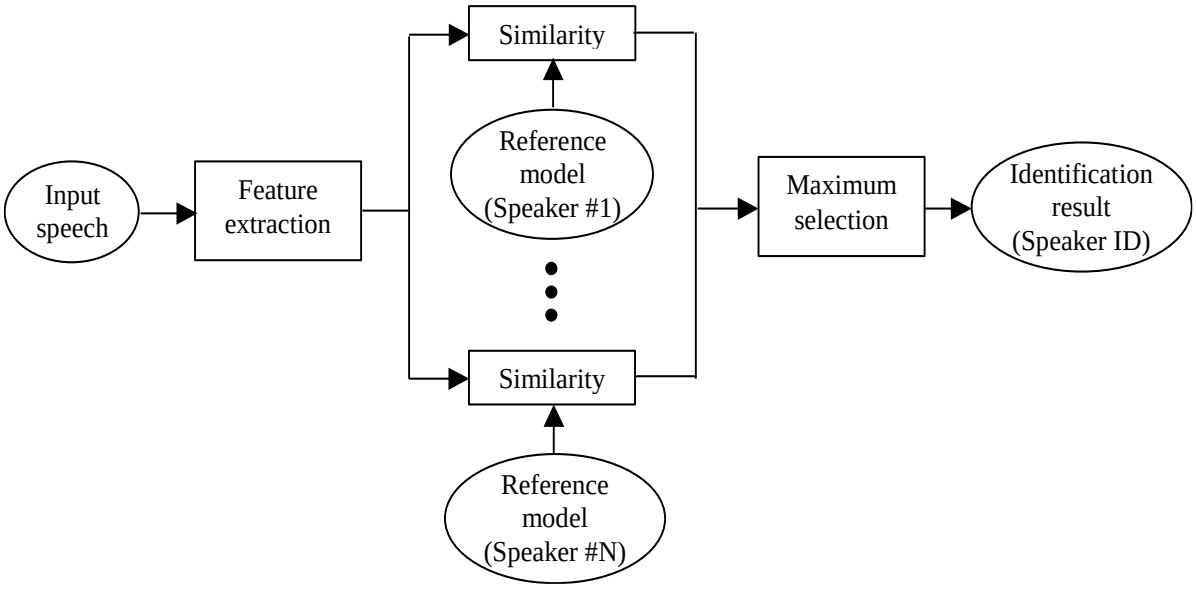


Fig 1.1 Speaker Identification

2.) Speaker verification: We determine if the speaker who is claiming a particular identity is telling truth or not.

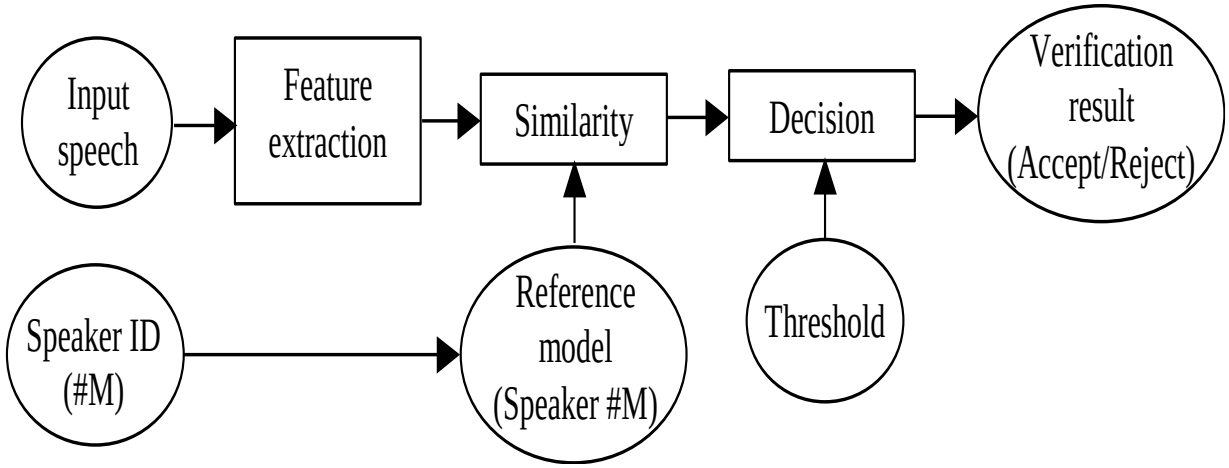


Fig 1.2 Speaker Verification

1.4 LITERATURE SURVEY

A lot of work on speaker recognition has already been done in industry, technological labs and educational universities. For example AT&T labs have synthesized speaker recognition systems. The NIPPON Telephone and Telegraph Company have their own speaker recognition systems. MIT of USA and National Tsing Hua university of Taiwan, Texas Instruments also have conducted testing of various speaker recognition systems. Automatic speaker identification systems applications include access control, telephone banking and telephone credit cards.ITT, Lernout & Hauspie, TNetix etc. are known for their automatic speaker verification systems.

It is estimated that cyber crimes involving 3-4 million dollars occur in banks in USA every year. Imagine how much can we save by using speaker recognition systems to check these fraudulent transactions. Products like SPRINTS's Foncard which uses TI's speaker verification engine are used to check the occurrence of these kinds of scams.

Furthermore, when choosing a processor, a fundamental question to answer is whether the application can be best addressed using a fixed-point or a floating-point processor. The Texas Instruments C6000 series of DSPs are available in both fixed- and floating-point varieties. For instance, in the C6201 and C6203, all eight functional units are fixed-point. In the C6701, six of the eight units are floating point. Because of their lower cost and power, fixed-point processors are best suited for high volume, heavily embedded applications. For fixed-point processors, the additional code complexity required for scaling may be offset by the lower cost of the silicon. Floating-point processors are best for applications that require extensive floating- point arithmetic, or in custom applications where the code is likely to change and the user can exploit the faster development efforts.[4]

CHAPTER – 2

HARDWARE COMPOSITION OF THE KIT

Before moving on to the actual application of the DSP processor it is necessary to understand the DSP processor's block diagram and the function of each component. Therefore let us familiarize ourselves with the TMS320C6713.

2.1 About DSK C6713

The C6713 DSK builds on TI's industry –leading line of cost easy to use DSP Starter Kit (DSK) development boards. The performance board features the TMS320C6713 floating point DSP. Capable of performing 1350 million floating –point operations per second (MFLOPS), the C6713 DSP makes the most powerful DSK development board. The DSK also serves as a hardware reference design for the TMS320C6713 DSP. Schematics, logic equations and application notes are available to ease hardware development and reduce time to market. [5]

The DSK starter kit includes the following hardware items:

TMS320C6713 DSK	TMS320C6713 DSK development board
Other hardware	External 5V DC power supply,
	IEEE 1284 compliant male-to-female cable
CD-ROM	Code Composer Studio DSK tools

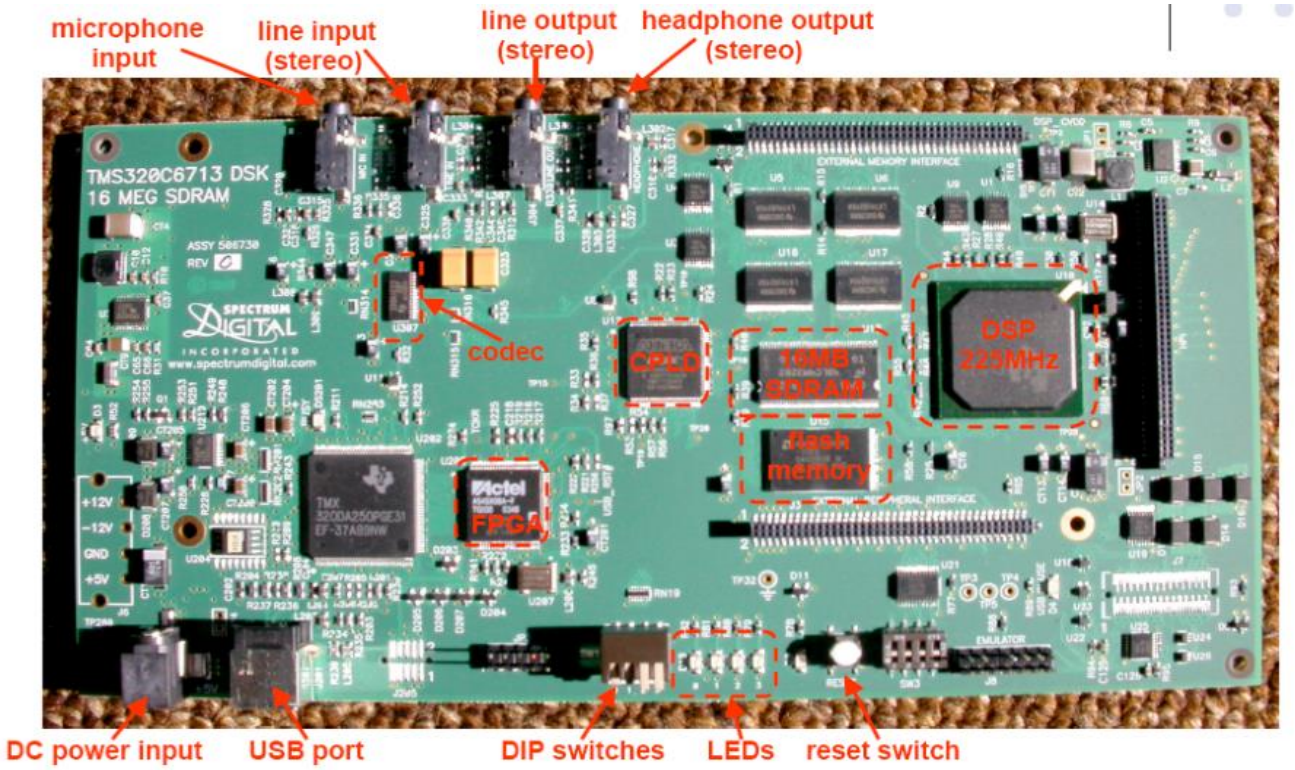


Fig 2.1 System Layout of TMS320C6713

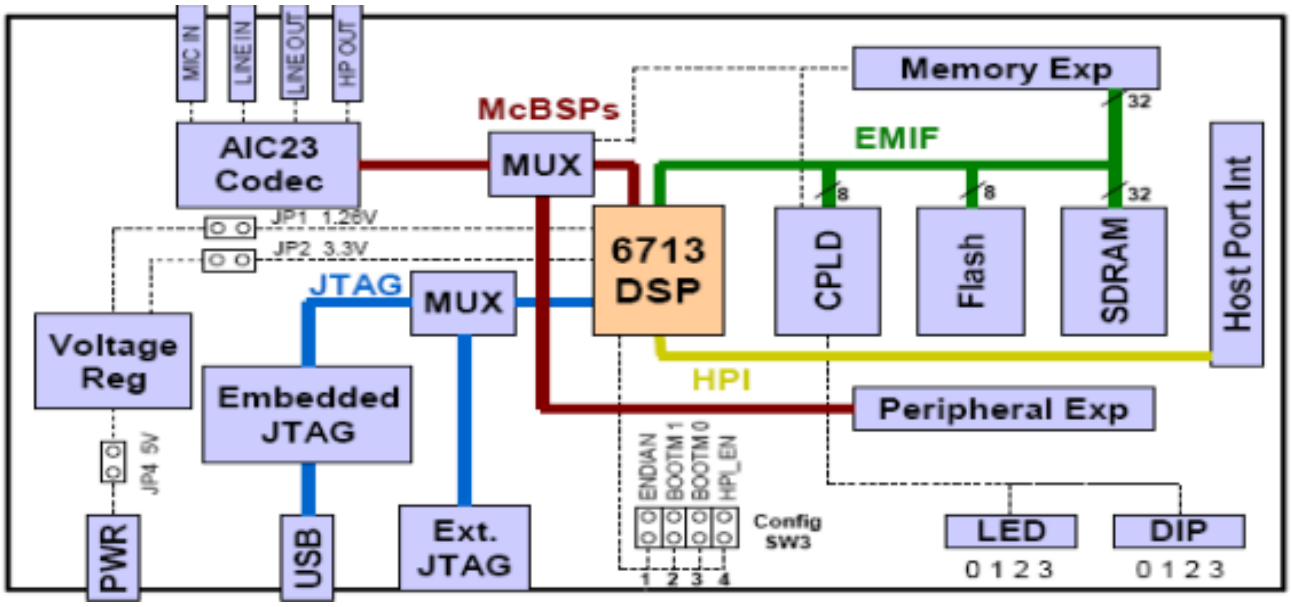


Fig 2.2 Block Diagram of DSK6713

2.2 Features of DSK C6713

The DSK comes with a full complement of on-board devices that suit a wide variety of application environments. Key features include:

- **A Texas Instruments TMS320C6713 DSP**

The kit has Highest Performance Floating Signal Processor (DSP) which executes Eight 32-bit Instructions/cycle operating at 225 MHz. It has rich peripheral set which is optimized for Audio. It supports programming languages like C/C++.

2.2.1An AIC23 stereo codec

The DSP interfaces to analog audio signals through an on-board AIC23 codec and four 3.5 mm audio jacks (microphone input, line input, line output, and headphone output). The codec can select the microphone or the line input as the active input. The analog output is driven to both the line out (fixed gain) and headphone (adjustable gain) connectors. McBSP0 is used to send commands to the codec control interface while McBSP1 is used for digital audio data. McBSP0 and McBSP1 can be re-routed to the expansion connectors in software.

The DSK uses a Texas Instruments AIC23 (part #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, one to control the codec's internal configuration registers and one to send and receive digital audio samples. McBSP0 is used as the unidirectional control channel. It should be programmed to send a **16-bit** control word to the AIC23 in SPI format. The top 7 bits of the control word should specify the register to be modified and the lower 9 should contain the register value. The control channel is only used when configuring the codec, it is generally idle when audio data is being transmitted.[5]

McBSP1 is used 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. The DSK examples generally use a **16-bit** sample width with the codec in master mode so it generates the frame sync and bit clocks at the correct sample rate without effort on the DSP side. The preferred serial format is DSP mode which is designed specifically to operate with the McBSP ports on TI DSPs.[5]

The codec has a **12MHz** system clock. The 12MHz system clock corresponds to USB sample rate mode, named because many USB systems use a 12MHz clock and can use the same clock for both the codec and USB controller. The internal sample rate generate subdivides the 12MHz clock to generate common frequencies such as 48KHz, 44.1KHz and 8KHz. The sample rate is set by the codec's SAMPLERATE register. The figure below shows the codec interface on the C6713 DSK.[5]

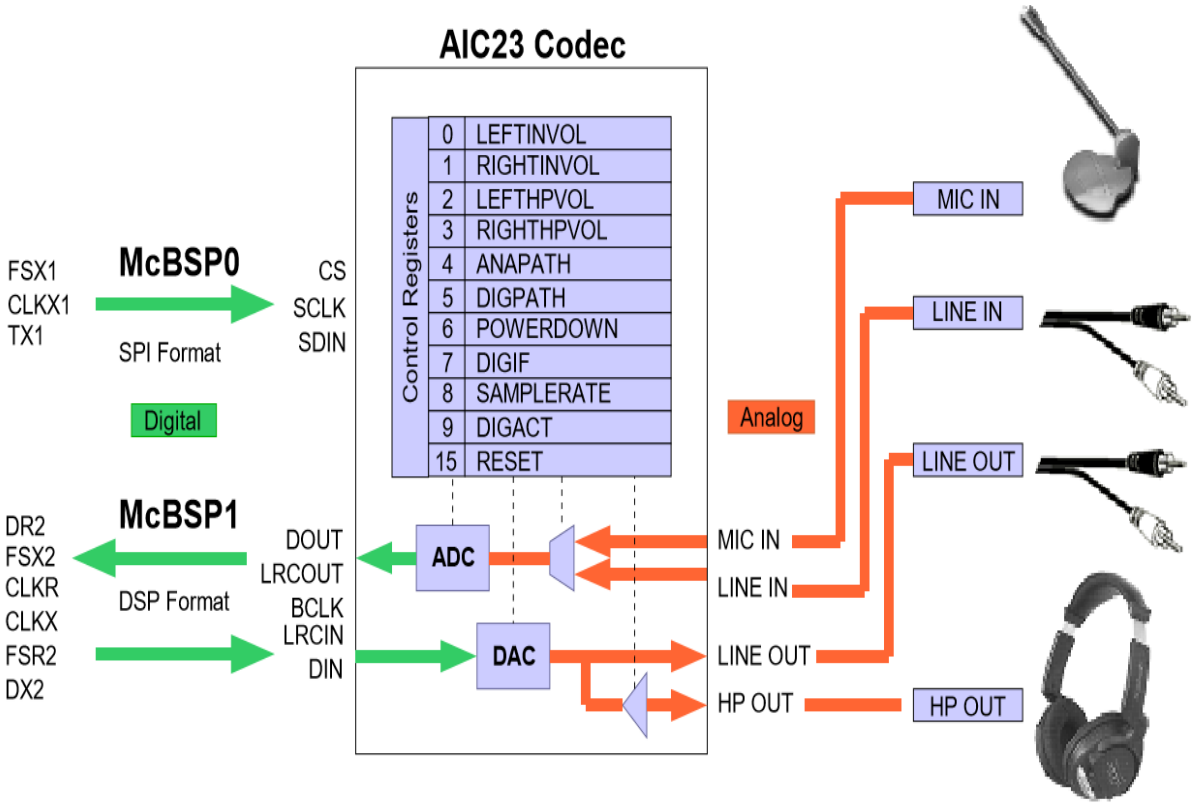


Fig 2.3 AIC23 Codec

- **Control Registers:**

Register	Specification
LEFTINVOL	Left line input channel volume
RIGHTINVOL	Right line input channel volume
LEFTHPVOL	Left channel headphone volume
RIGHTHPVOL	Right channel headphone volume
ANAPATH	Analog audio path control
DIGPATH	Digital audio path control
POWERDOWN	Power down control
DIGIF	Digital audio interface format
SAMPLERATE	Sample rate control
DIGACT	Digital interface activation

Table 2.1 Control Register

2.2.2 Study of DIP switches

The DSK has 4 configuration switches that allow users to control the operational state of the DSP when it is released from reset. The configuration switch block is labeled SW3 on the DSK board, next to the reset switch.

Configuration switch 1 controls the endianness of the DSP while switches 2 and 3 configure the boot mode that will be used when the DSP starts executing. Configuration switch 4 controls the on-chip multiplexing of HPI and McASP signals brought out to the HPI expansion connector. By default all switches are off which corresponds to EMIF boot (out of 8-bit Flash) in little endian mode and HPI signals on the HPI expansion connector.

- Reset Switch

  There are three resets on the TMS320C6713 DSK. The first reset is the power on reset. This circuit waits until power is within the specified range before releasing the power on reset pin to the TMS320C6713.

  External sources which control the reset are push button SW2, and the on board embedded USB JTAG emulator.

2.2.3 Study of on board LEDs

TheTMS320C6713 DSK has four system light emitting diodes (LEDs). These LEDs indicate various conditions on the DSK. These function of each LED is shown in the table below.

Reference Designator	Color	Function
D4	Green	USB Emulation in use. When External JTAG Emulator is used this LED is off.
D3	Green	+5 Volt present
D6	Orange	RESET Active
DS201	Green	USB Active, Blinks during USB data transfer

Table 2.2 study of on board LEDs

The DSK includes 4 software accessible LEDs (D7-D10) and DIP switches (SW1) that provide the user a simple form of input/output. Both are accessed through the CPLD USER_REG register.

2.2.4 CPLD Registers

A programmable logic device called a CPLD is used to implement glue logic that ties the board components together. The CPLD has a register based user interface that lets the user configure the board by reading and writing to its registers.

- Configurable boot options
- Standard expansion connectors for daughter card use
- **JTAG emulation through on-board JTAG emulator with USB host interface or external emulator.**

  Code Composer communicates with the DSK through an embedded JTAG emulator with a USB host interface. The DSK can also be used with an external emulator through the external JTAG connector.
- **Single voltage power supply (+5V)**

An included 5V external power supply is used to power the board. On-board switching voltage regulators provide the +1.26V DSP core voltage and +3.3V I/O supplies. The board is held in reset until these supplies are within operating specifications. [5]

2.2.5 PROCESSOR DESCRIPTION:

- 512 Kbytes of non-volatile Flash memory (256 Kbytes usable in default configuration)
- bytes of synchronous DRAM
- Processor executes eight 32-bit Instructions/cycle operating at 225 MHz
- The TMS320C6713B 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 CPU features two sets of functional units. Each set contains four units and a register file
- The two register files each contain 16 32-bit registers for a total of 32 general-purpose register.
- The C67x CPU supports a variety of indirect addressing modes using either linear- or circular-addressing modes with 5- or 15-bit offsets
- All instructions are conditional, and most can access any one of the 32 registers
- All load and store instructions are byte-, half-word, or word-addressable

2.2.6 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 dataaddressing units (.D1 and .D2) are responsible for all data transfers between the register files and the memory.[5]

functional block and CPU (DSP core) diagram

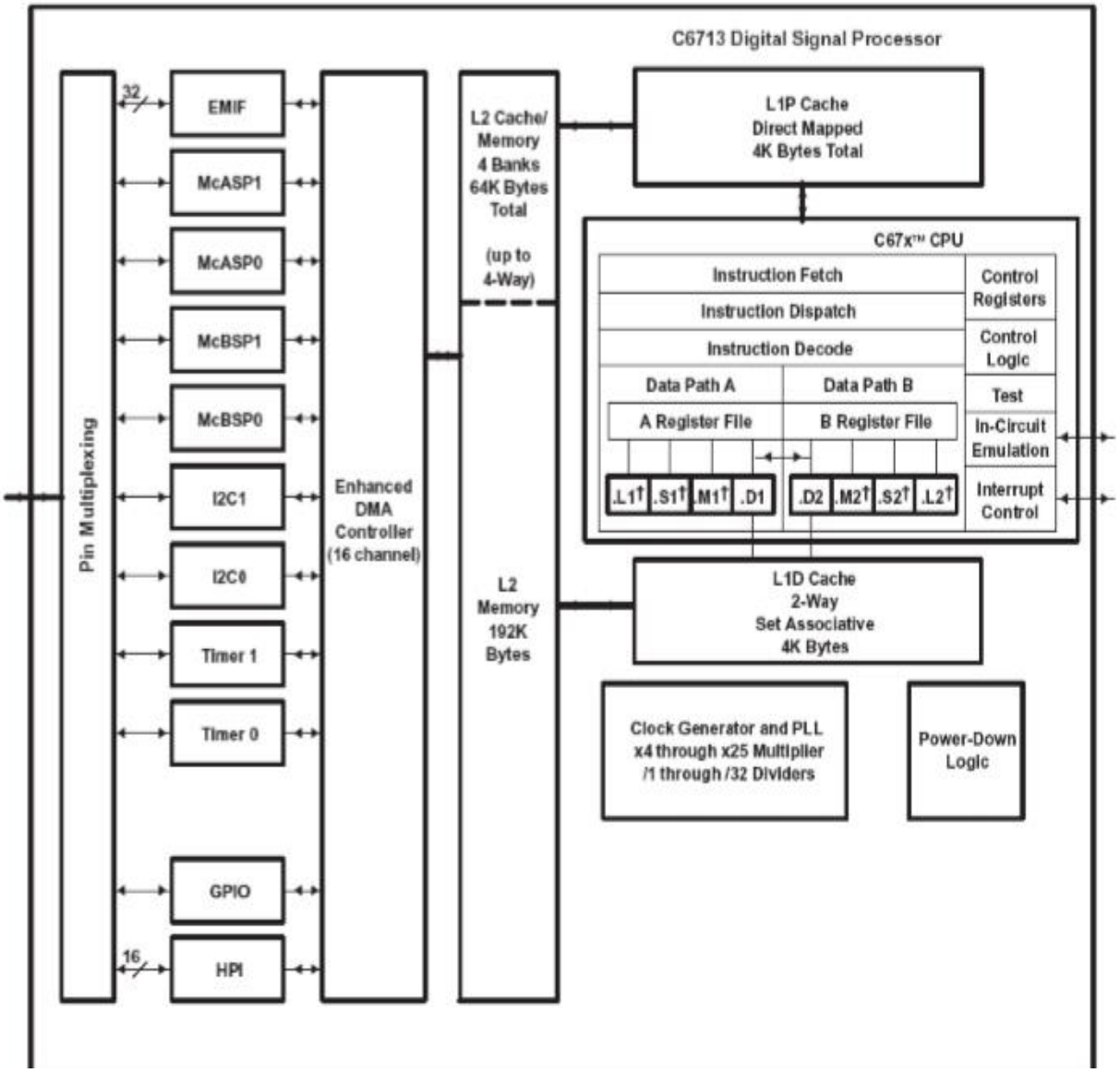


Fig 3.4 Functional Block & CPU Diagram

CHAPTER – 3

SOFTWARE DESCRIPTION

In order to communicate with the DSK, we use a software ambiance called the Code Composer Studio.

3.1 OVERVIEW OF CODE COMPOSER 5.3.0

Code Composer Studio (CCS) allows us to write a program in C language that can be used to initialize the DSK. Through CCS, we can initialize various ports and registers of the DSK. Code Composer provides a rich debugging environment that allows stepping through the code, set breakpoints, and examining the registers as the code is getting executed.

The Code Composer Studio (CCS) application provides an integrated environment with the capabilities like Integrated development environment with an editor, debugger, project manager, and profiler, C/C++ compiler, assembly optimizer and linker, Simulator, Real-time operating system (DSP/BIOS™), Real-Time Data Exchange (RTDX™) between the Host and the Target, and Real-time analysis and data visualization.

CCS studio integrated development environment includes host tools and target software that slashes development time and optimizes the performance for all real-time embedded DSP applications. Some of the Code Composer Studio's host side tools include TMS320 DSPs and OMAP Code, Drag and Drop CCS studio setup utility, Component manager support for multiple versions of DSP/BIOS and code generation tools within the IDE, Source Code Debugger common interface for both simulator and emulator targets, Connect/Disconnect; robust, resilient host to target connection, Application Code Tuning Dashboard, RTDX ™ data transfer for real time data exchange between host and target, Data Converter Plug-in to auto configure support for Texas Instruments Mixed Signal products, Quick Start tutorials and Help.

Code Composer Studio's target software includes DSP/BIOS ™ Kernel for the TMS320 DSPs, TMS320 DSP Algorithm Standard to enable software reuse, Chip Support Libraries to simplify device configuration, and DSP Libraries for optimum DSP functionality.

3.1.1 Installation of Code Composer Studio v5.3.0

1. Run the ccs_setup_5.3.0.00090.exe from the CCS5.3.0.00090_win32 folder

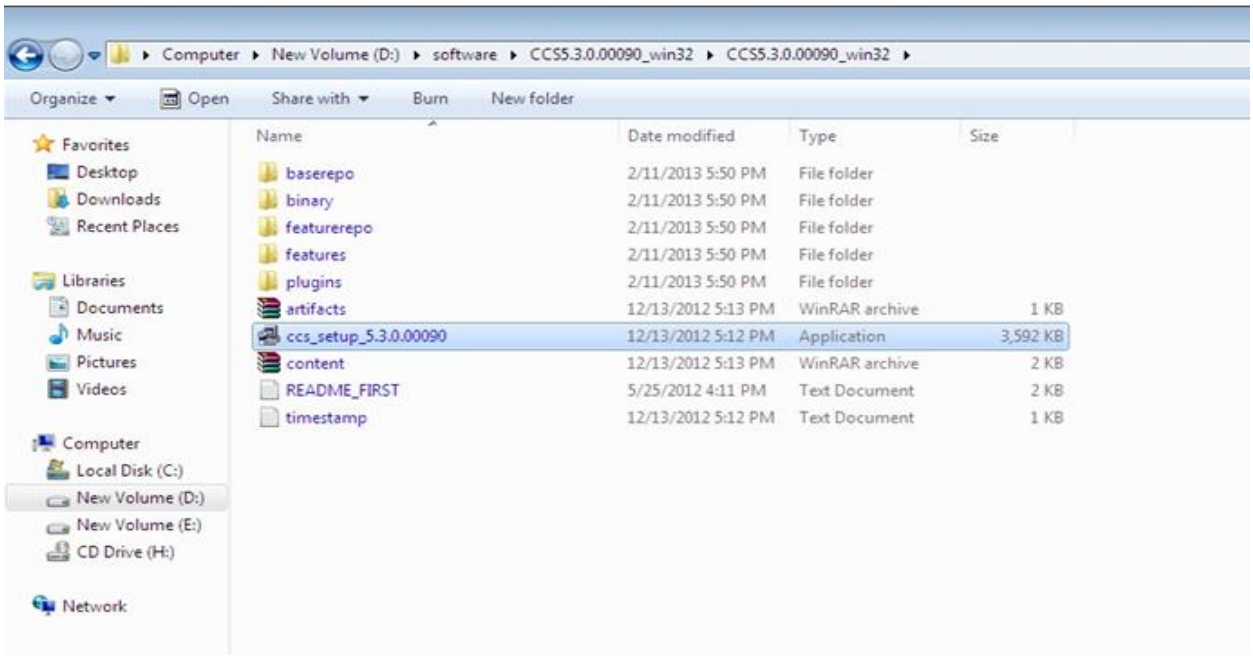


Fig. 3.1 Run the ccs_setup_5.3.0.00090.exe

2. Following screen will appear select “I accept the terms of the License agreement” and click on Next.

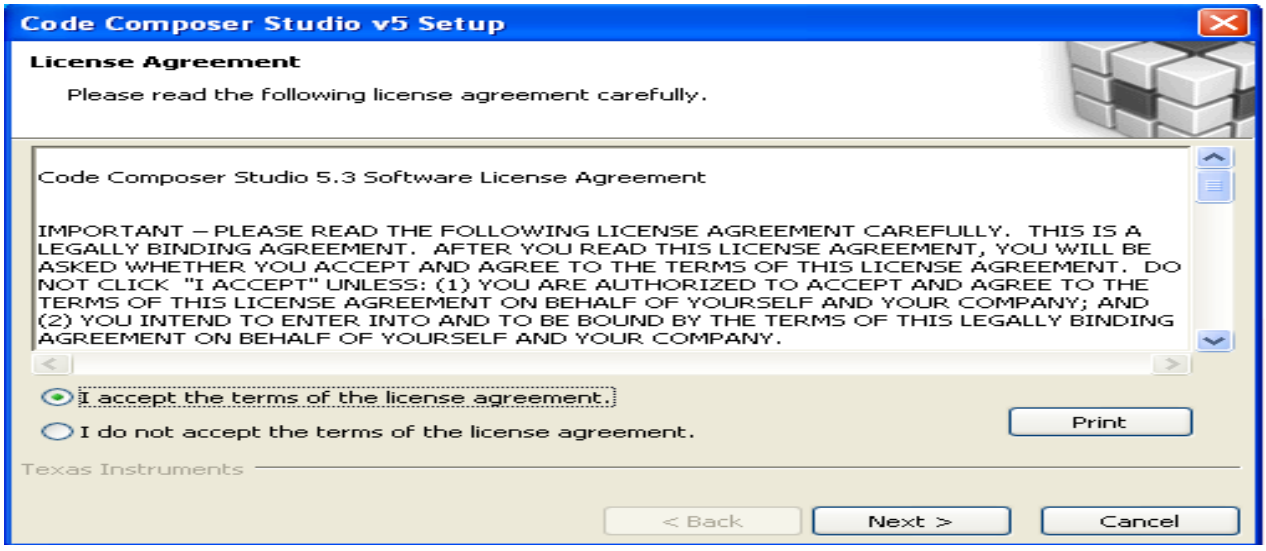


Fig 3.2 License Agreement Window

3. Keep the installation path as “C:/ti” and click on Next

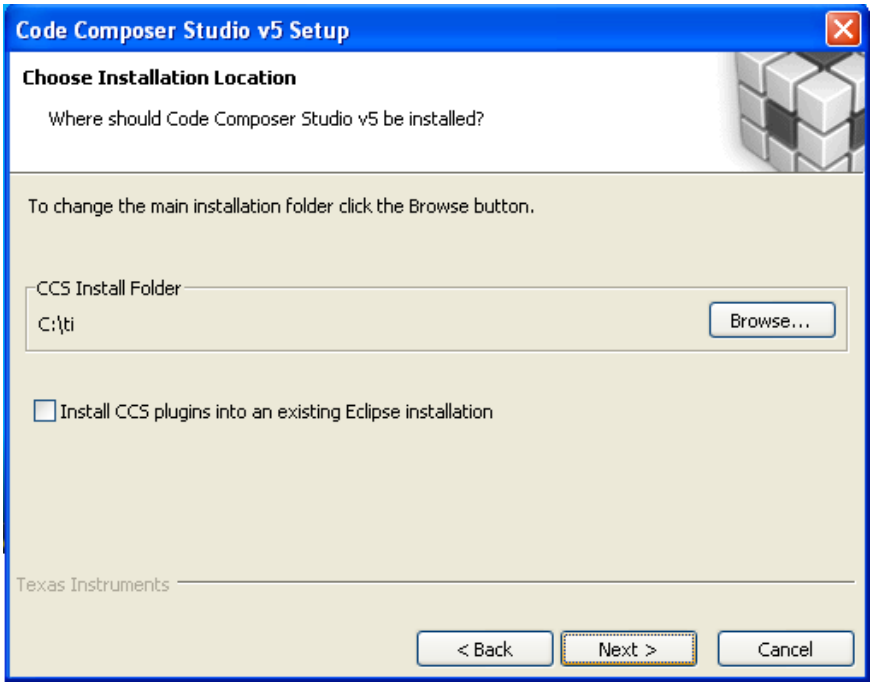


Fig 3.3 Installation Location

4. Select the Setup type as “Complete feature set” and click on Next

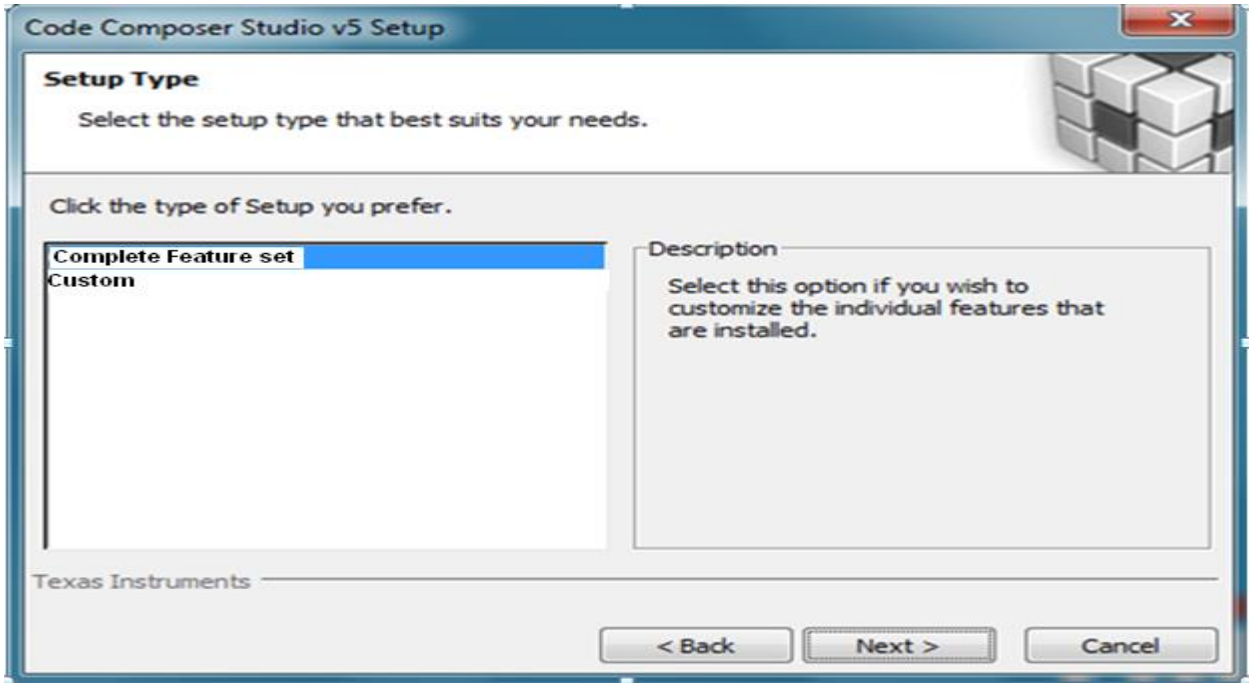


Fig 3.4 Setup Type Window

5. Select the Emulators as default as shown below and click on Next

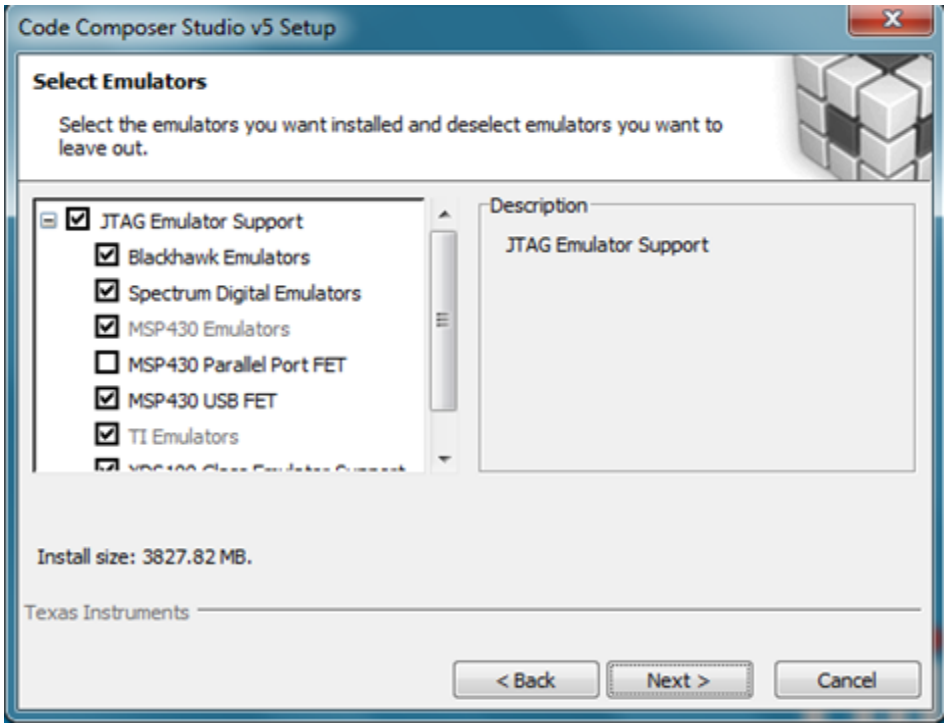


Fig 3.5 Select Emulator Window

6. Click Next. The installation Process will start as shown below. The installation will take some time.

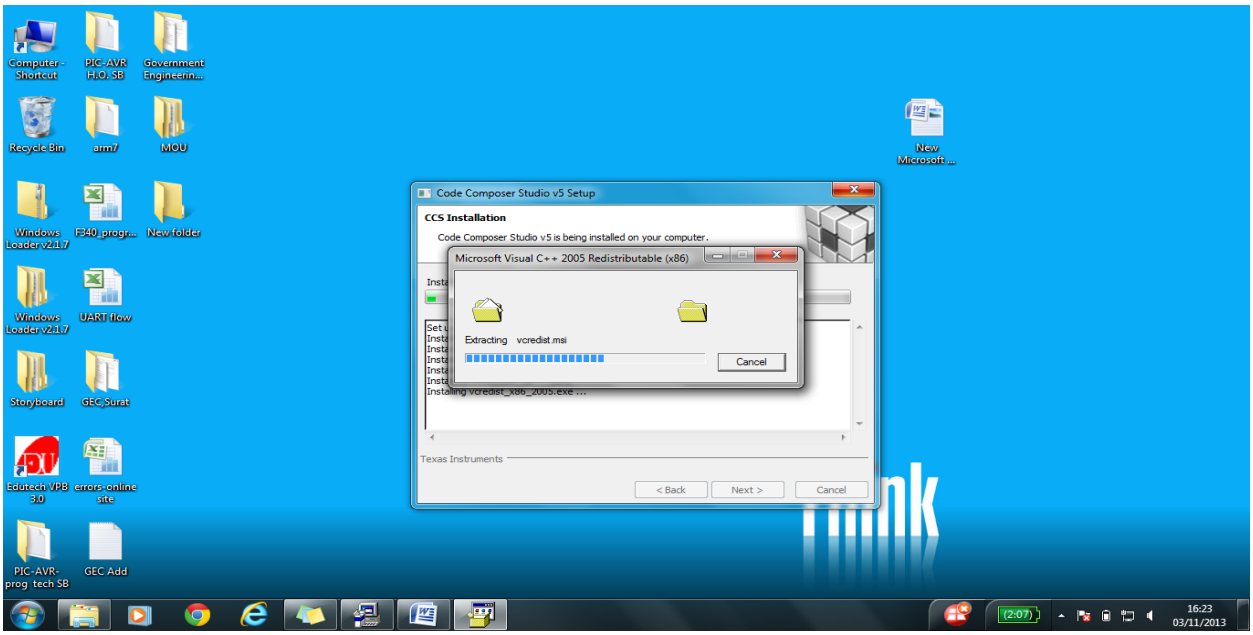


Fig 3.6 Installation Process

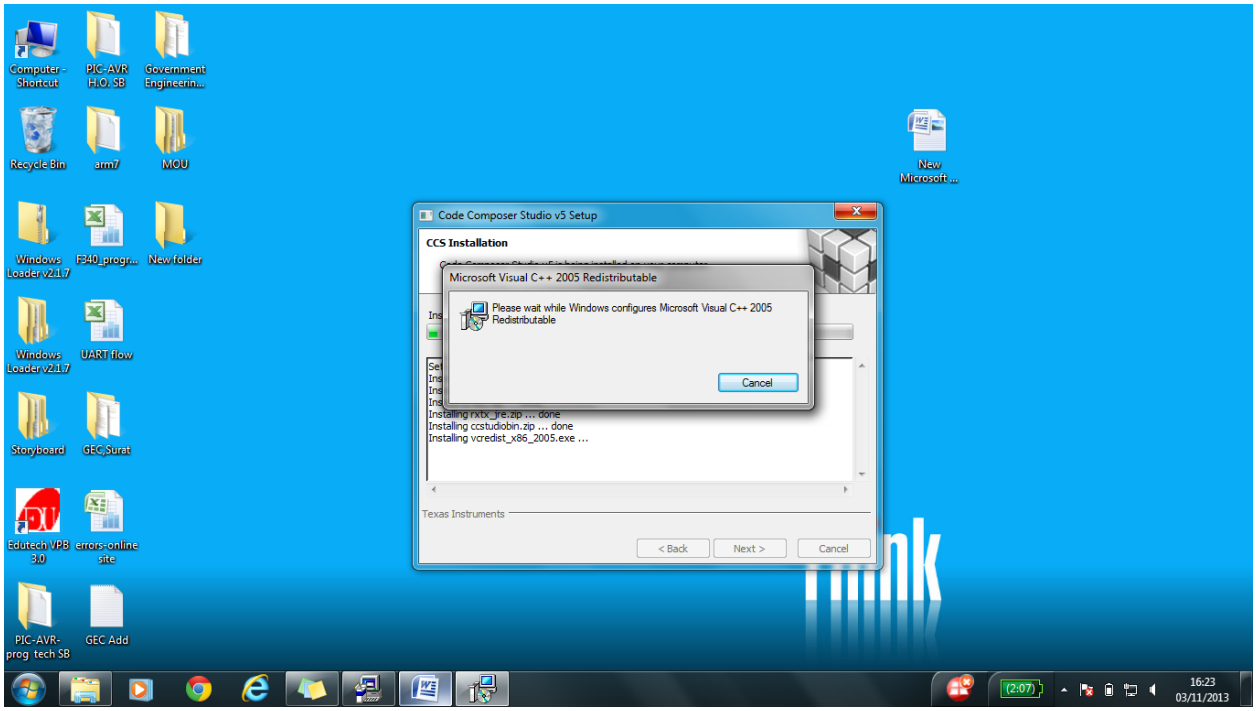


Fig 3.7 Installation Completed

Meanwhile installation it will ask for the different tool's installation permissions as pop up window, click OK or Yes for permission grant.

Once installation finish, click on Finish.

3.1.2 Updating Installation

1. Install CSL library by executing **C6000.exe** setup from *"\EPB_6713\Setup\C6000 CSL\sprc090"* and while installing **change its installation directory** as "**C:\C6xCSL**"
2. rename "**lib_2x**" by "**lib**" name at "*C:\C6xCSL*" location.
3. Copy **"workspace_v5_3_6713_DSP"** folder to "**C:/**" drive
4. Open CCS5.3 with workspace as "C:\workspace_v5_3_6713_DSP" and rebuild/execute all examples

3.1.3 Steps for creating new project:

1. Open *CCS V5.3* from desktop shortcut

It will open default *CCS V5* screen.

Fig 3.8 CCS V5 Screen

2. Then it will ask for workspace path Select path "***C:\workspace_v5_3_6713_DSP***" for windows

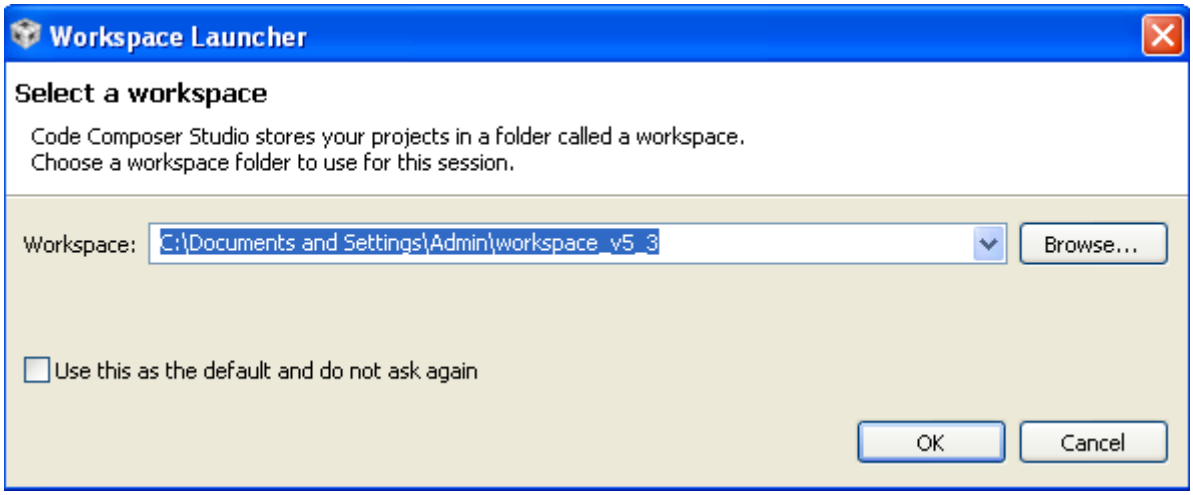


Fig 3.9 Workspace Select path

Then it will open Default CCS5 screen as shown below

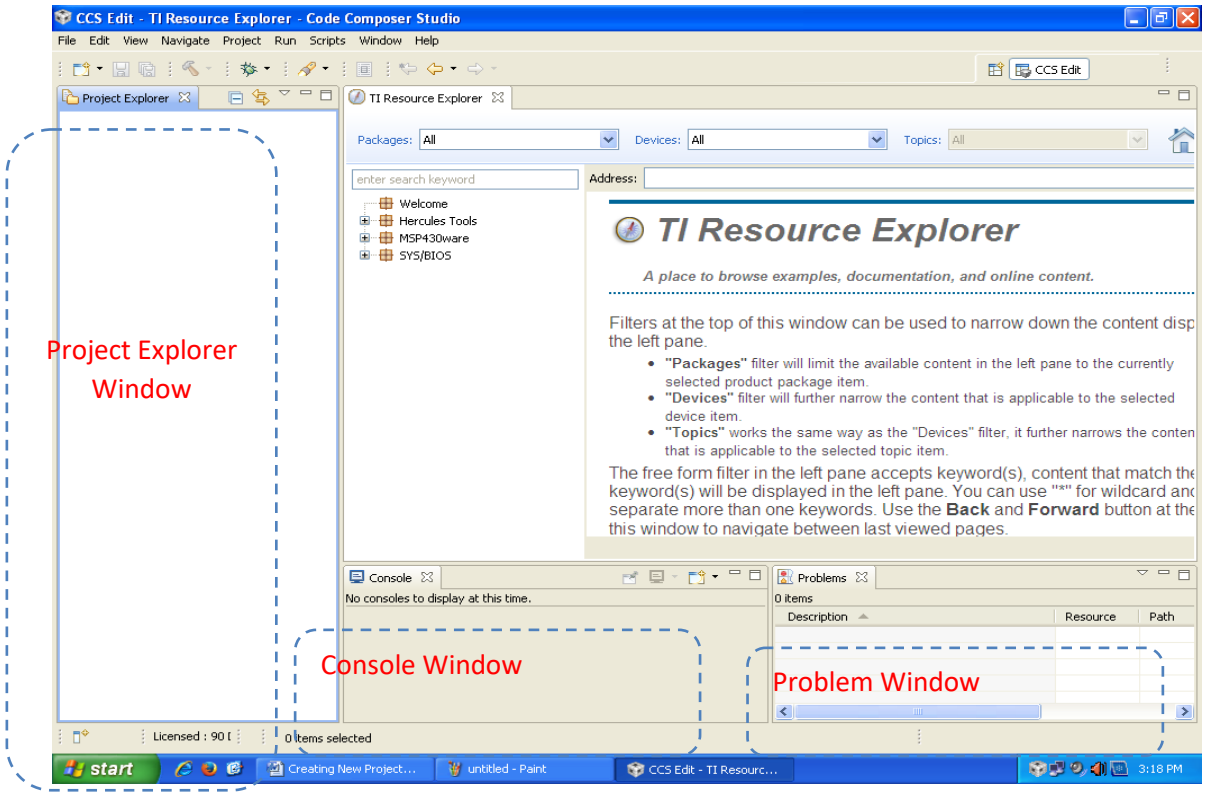


Fig 3.10 Default CCS Screen

2. Click "*Project -> New CCS Project*" menu.

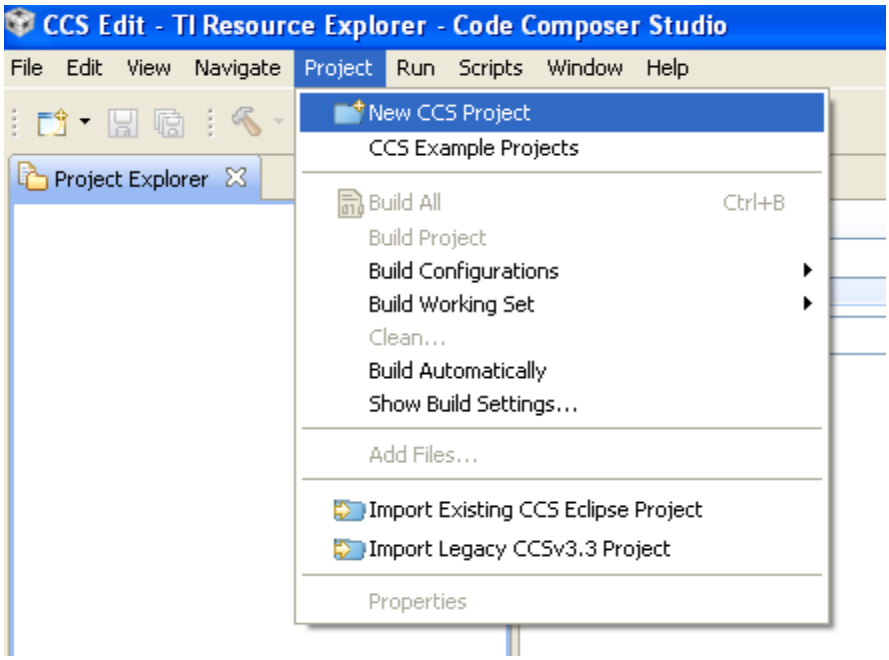


Fig 3.11 New CCS Project

3. It will open following screen

Project name as desired, - e.g "Eg7_Sine_Sweep8000"

Output type: *Executable* as in figure.

And keep selected "***use default location*** ". So that project will be created in workspace with project name typed

Select family: *C6700*,

Variant: TMS320C*C67xx*

Processor:*DSK6713*

And use **connection** type as Spectrum Digital DSK-EVM-eZdsp onboard USB emulator

Then at last select "*Empty project*" example from "*Empty Projects*" location in **Project Templates and example** tab. And **Finish**

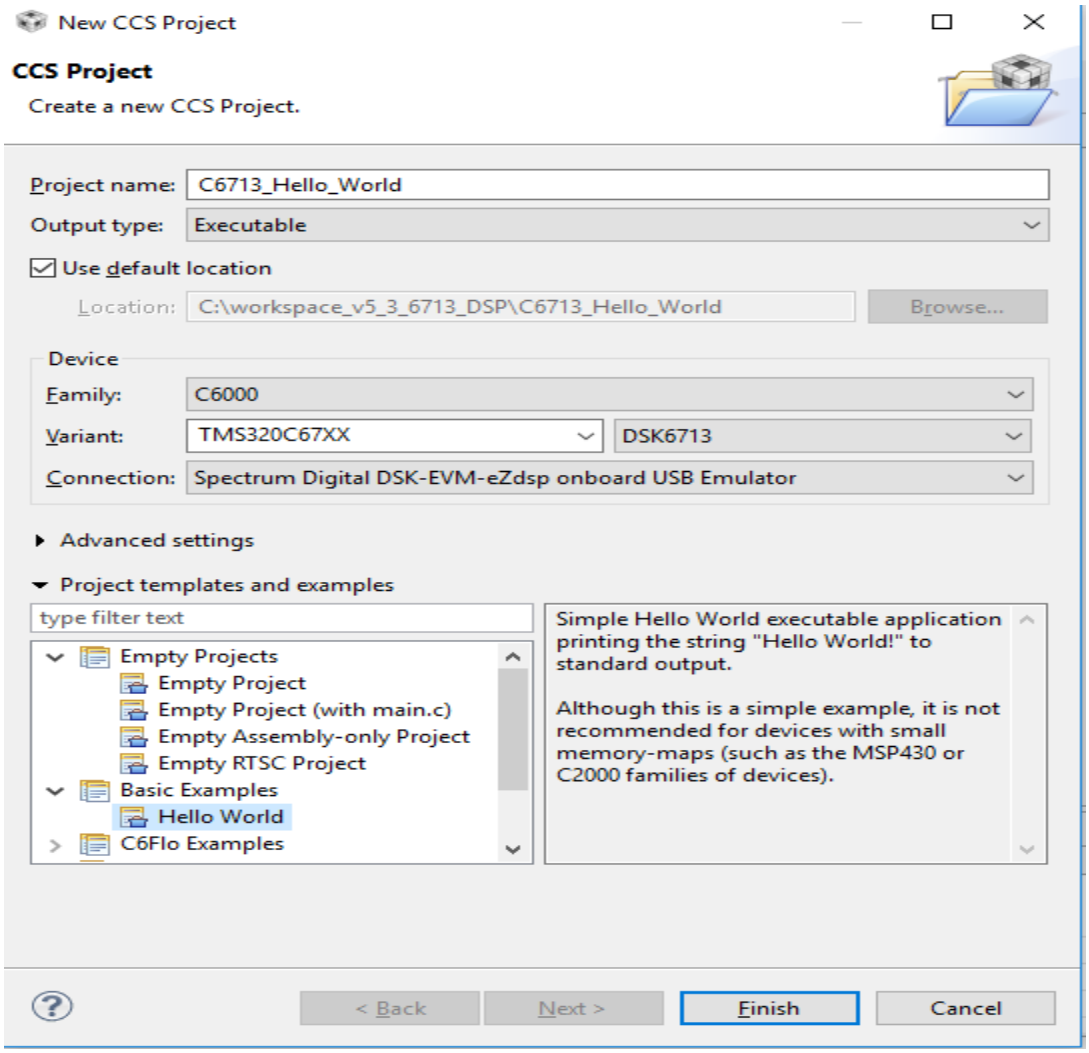


Fig 3.12 Create New Project

4. Then open “project property” by “*right click-> properties*” and at “Build->C6000 Linker->Basic Options” set the ***stack size*** as shown below in screenshot image.

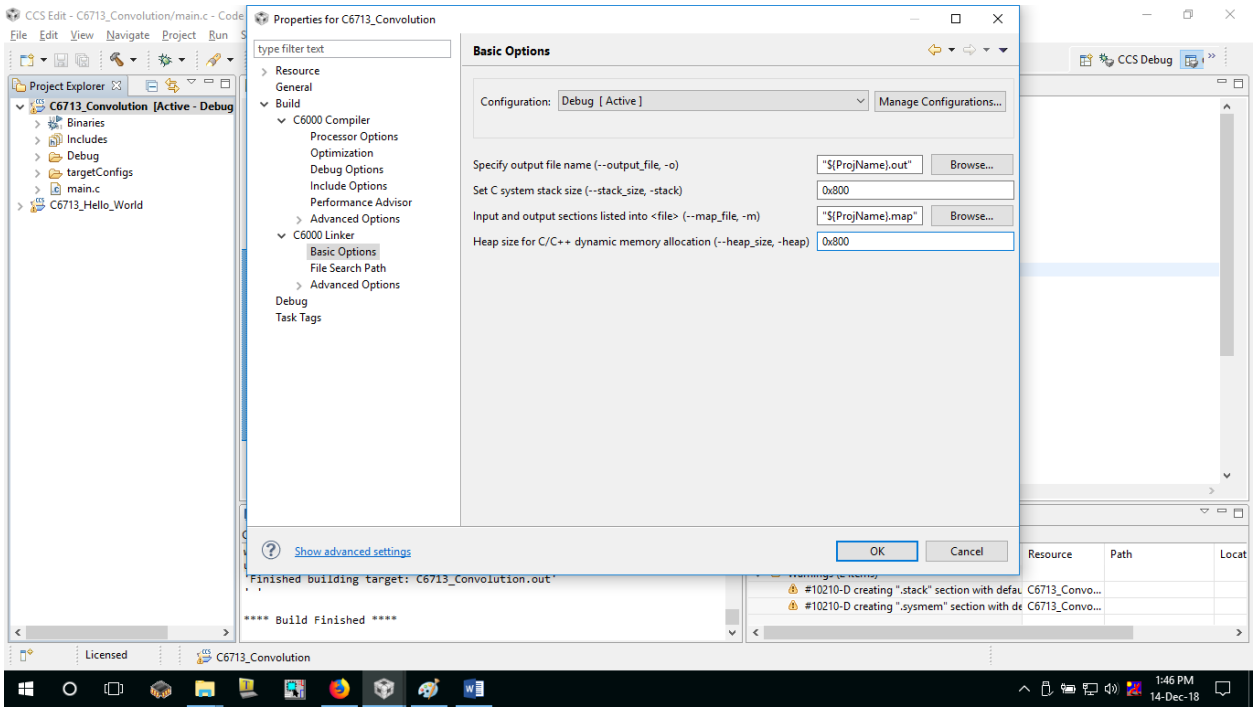


Fig 3.13 C6000 Linker (basic path)

5. Then open “project property” by “*right click-> properties*” and at “Build->C6000 Linker->File Search Path” set the Include Library file or command file as input (--library, l)” **and** Add libraries as shown below from respective path

   Files:

   C:\C6xCSL\lib_3x\csl6713.lib

   C:\workspace_v5_3_6713_DSP\Support\dsk6713bsl.lib

   C:\ti\ccsv5\tools\compiler\c6000_7.4.1\lib\rts6200.lib

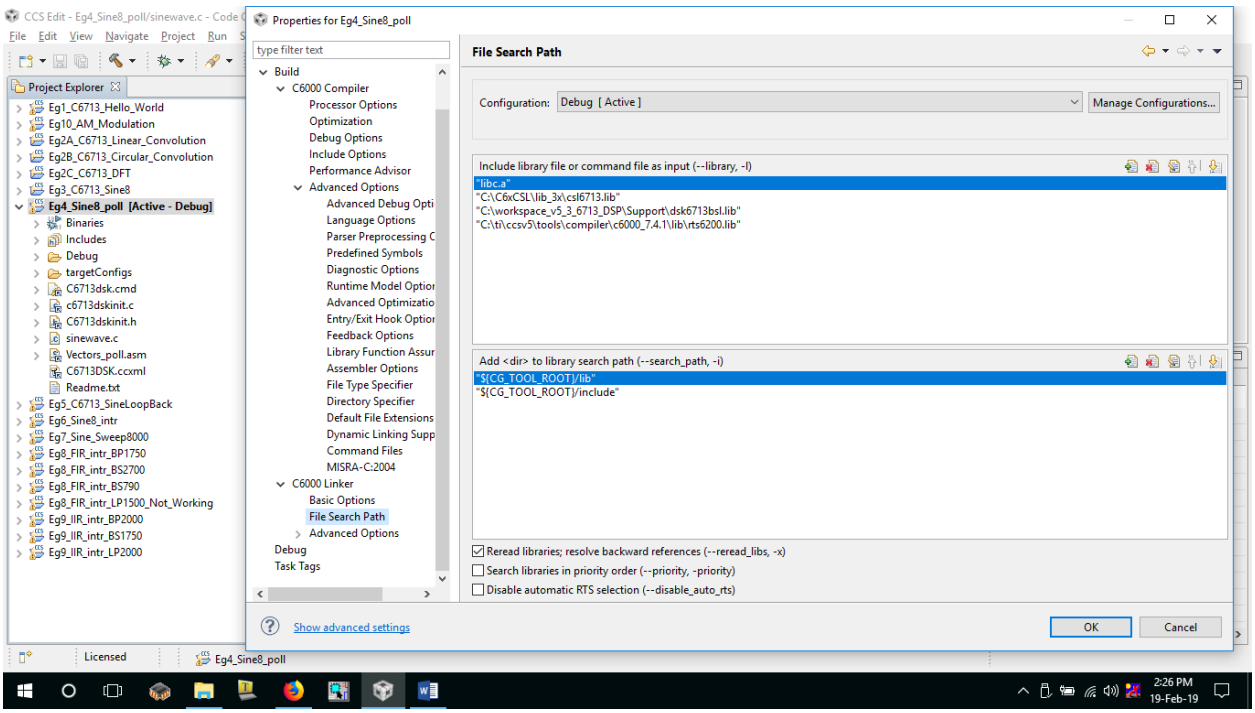


Fig 3.14 C6000 Linker (file search path)

6. Then open "project property" by "*right click-> properties*" and at"**C6000 Compiler->include Option** ". Add the header file path as shown here.

   Files:

   C:\C6xCSL\include

   C:\workspace_v5_3_6713_DSP\Support

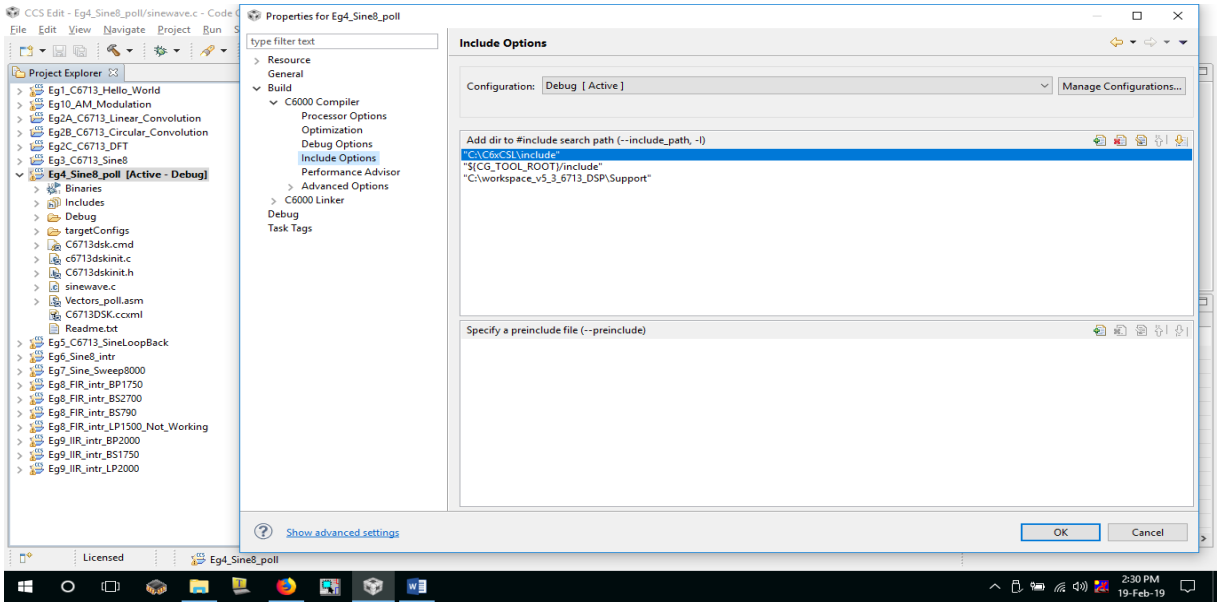


Fig 3.15 C6000 Compiler (include option)

7. Then open "project property" by "*right click-> properties*" and at"**C6000 Compiler->Predefined Symbol**". Add the pre-defined name CHIP-6713 as shown.

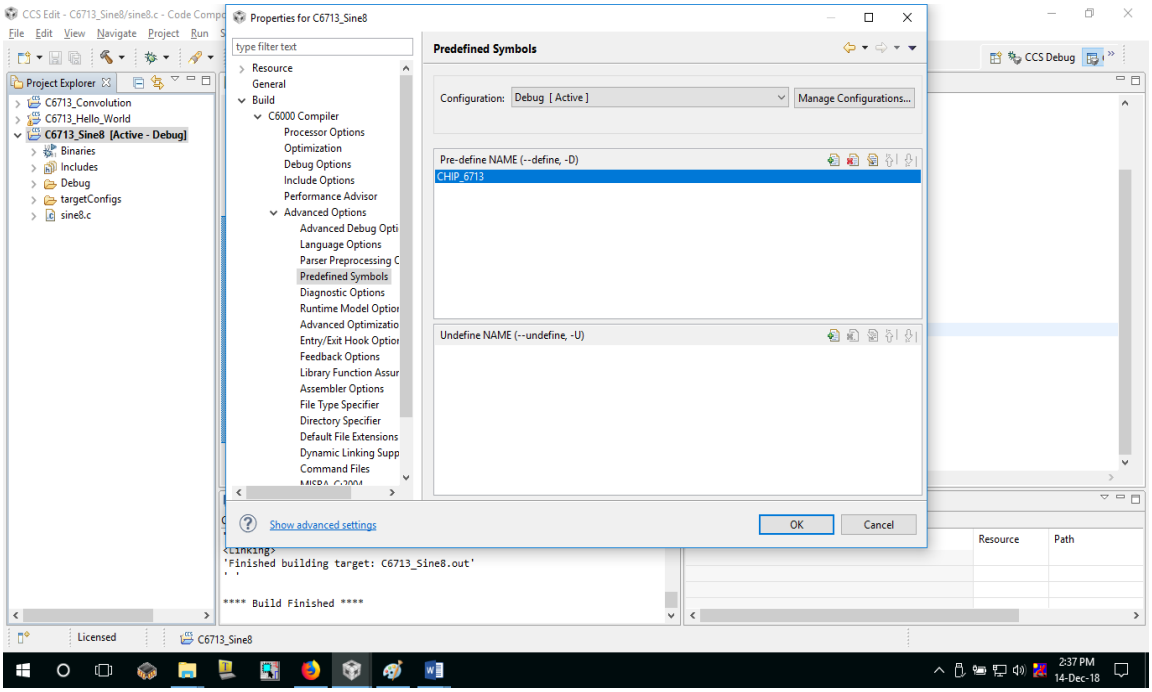


Fig 3.16 C6000 Compiler(predefined symbol)

❖ **Steps to Build the project:**

Compile the program by "*right click-> build project*" or "*right click-> rebuild project*" as shown.

It will generate Eg7_Sine_Sweep8000.out file in "*debug*" folder.

❖ **Steps to run/debug program:**

Now to debug the program click "**debug**" as shown in the screen from home screen icon **OR** from "**run->debug**" menu. It will configure/connect DSK_C6713 kit with the CCSV5. It will be done automatically. Once Configuration is over, it will start loading program into the CPU.

3.1.4 Connection of dsk6713 with CCS

1. Open DSK6713.ccxml file from targetConfigs.
2. Select connection “Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator.
3. Select Device “DSK6713”.

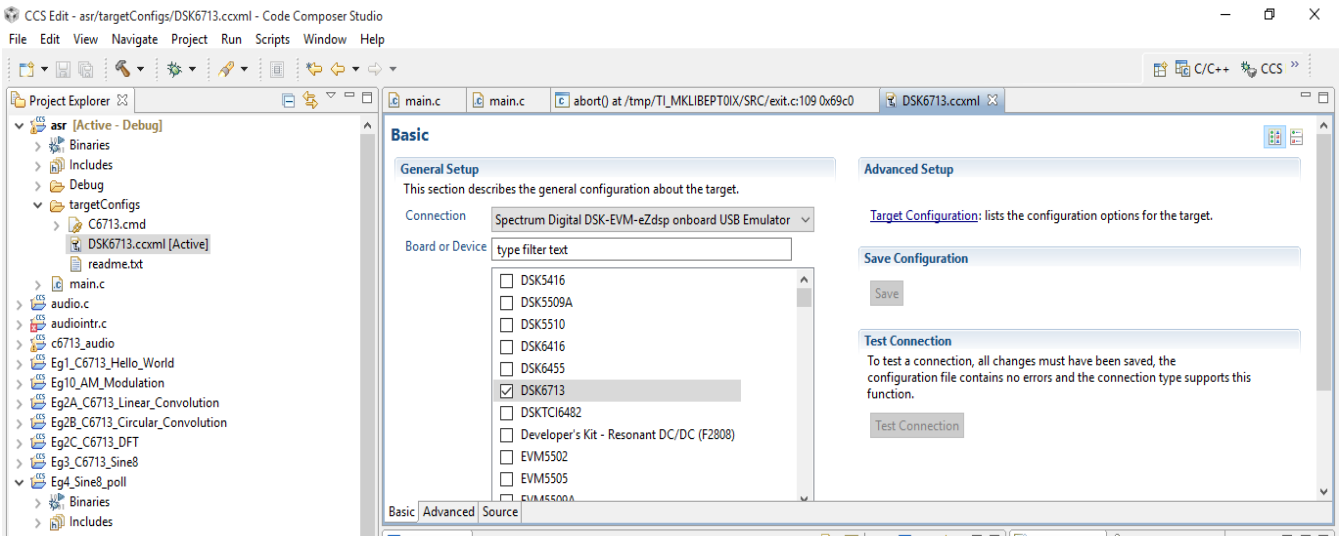


Fig.3.17 DSK6713.ccxml file from TargetConfigs

4. In advanced setting option one can view saved setting of project.

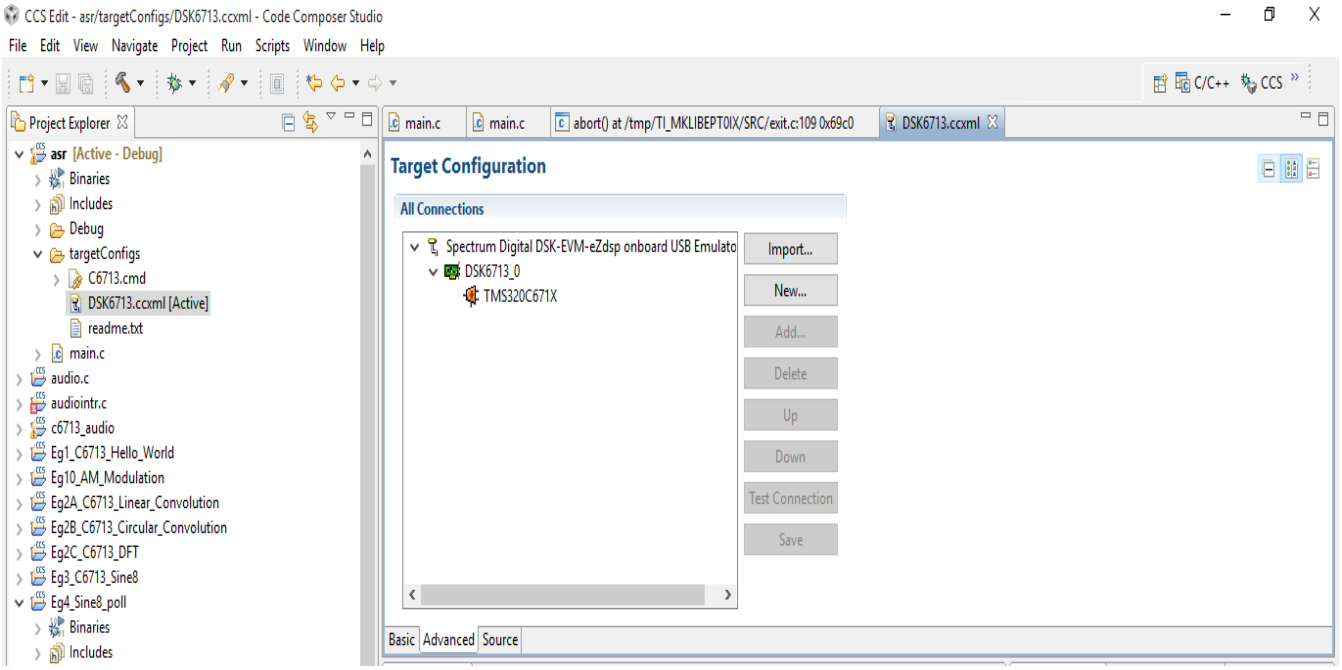


Fig.3.18 Setting Option

3.2 Audacity

- Matlab has a function audioread to read the audio file. This function can only read files which has .wav format. Audio file that I have used is in .mp3 format hence to convert .mp3 to .wav file I have used audacity application.
- This application converts audio files into .wav format.

Fig 3.19 Open Audacity

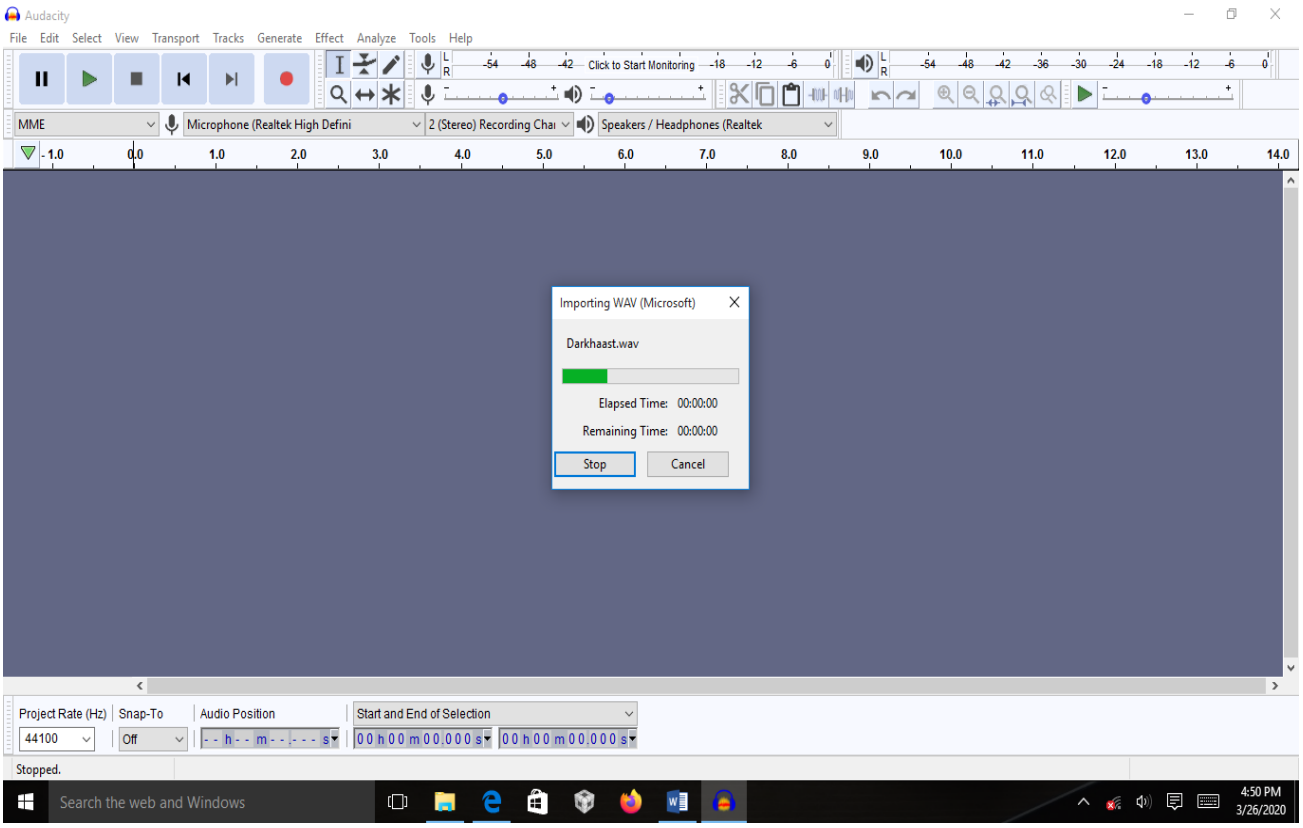


Fig 3.20 Start Importing Audio File

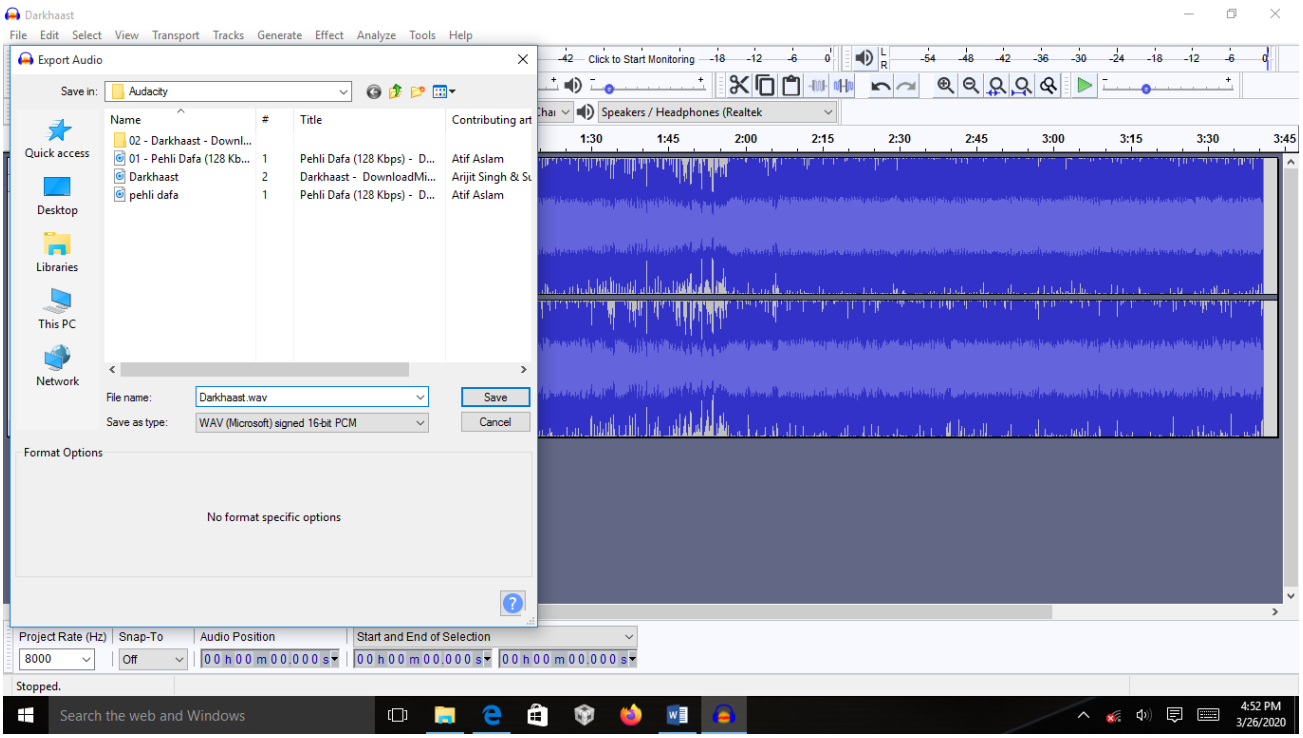


Fig 3.21 Export Audio into Wav file

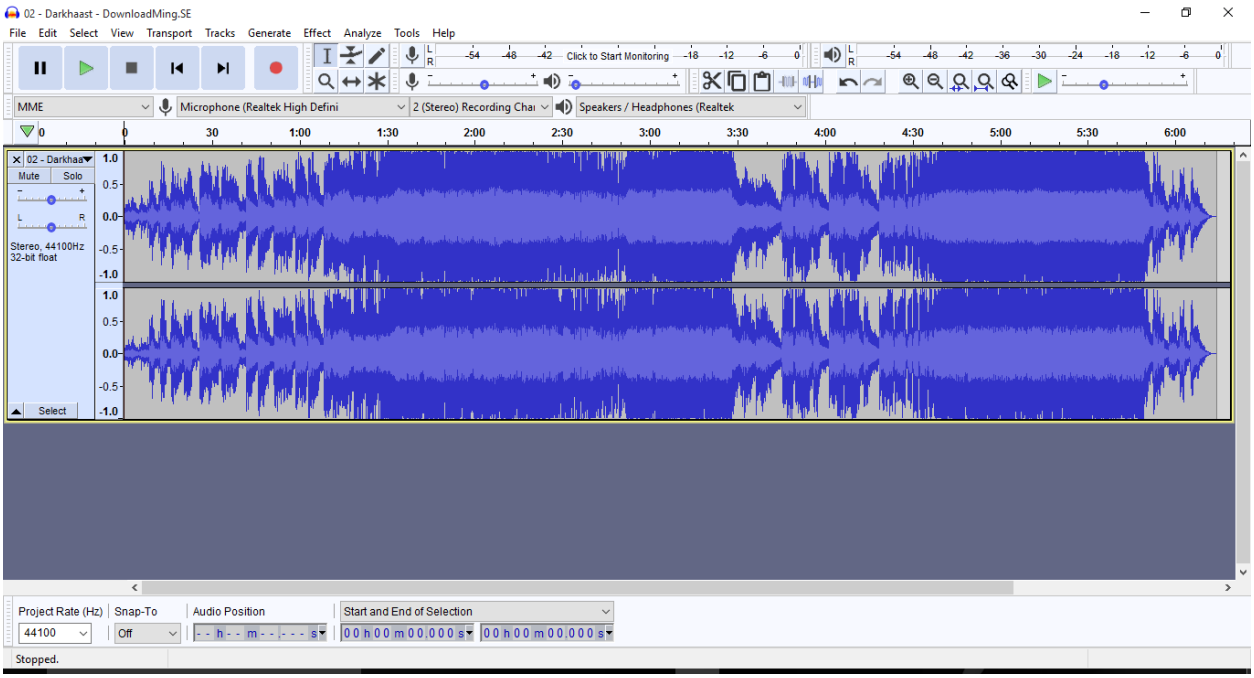


Fig 3.22 Audio File Convert into Wav File

CHAPTER – 4

SPEECH FEATURE EXTRACTION PROCESS

4.1 INTRODUCTION OF SPEECH FEATURE EXTRACTION:

The purpose of this module 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. The speech signal is a slowly timed varying signal (it is called quasi-stationary). An example of speech signal is shown below.

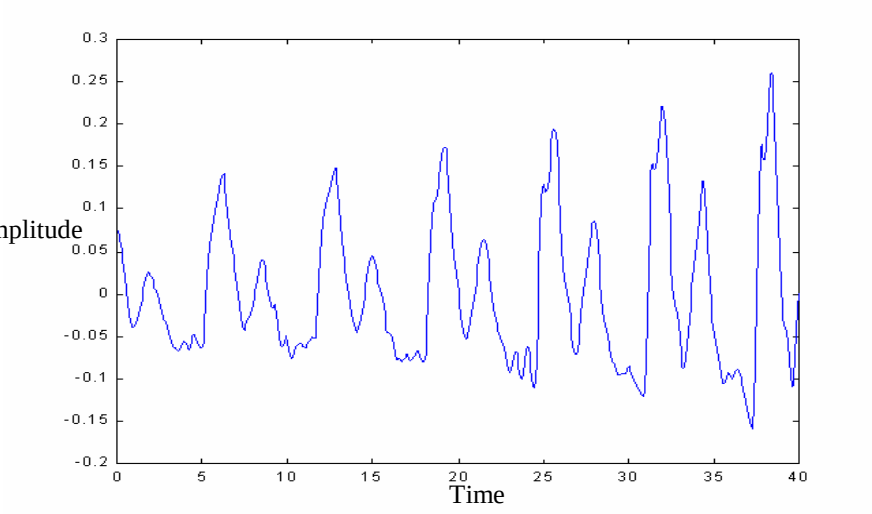


Fig 4.1 Example of Speech Signal

When examined over a sufficiently short period of time (between 5 and 100 msec), its characteristics are fairly stationary. However, over long periods of time (on the order of 1/5 seconds or more) the signal characteristic change to reflect the different speech sounds being spoken. Therefore, short-time spectral analysis is the most common way to characterize the speech signal.[11]

4.2 THE "MFCC" PROCESSOR

MFCC's 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. This is expressed in the mel-frequency scale, which is a linear frequency spacing below 1000 Hz and a logarithmic spacing above 1000 Hz.

A block diagram of the structure of an MFCC processor is given in figure below.

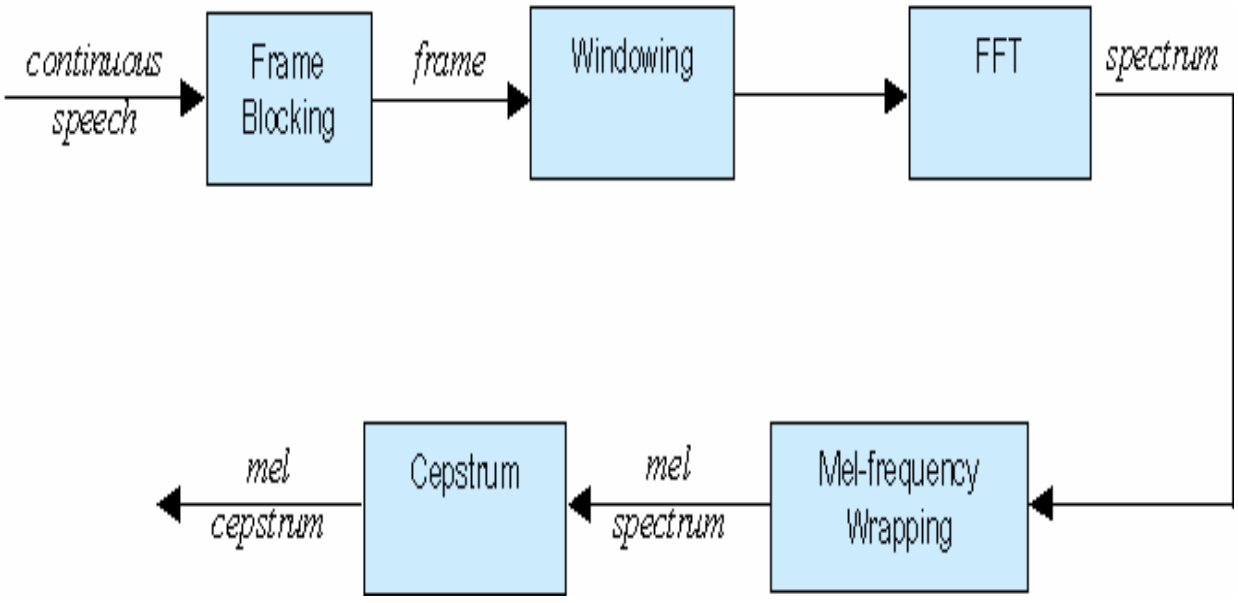


Fig 4.2 Block Diagram of the MFCC Processor

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-todigital conversion. These sampled signals can capture all frequencies up to 5 kHz, which cover most energy of sounds that are generated by humans.[12]

The main purpose of the MFCC processor is to mimic the behavior of the human ears. In addition, rather than the speech waveforms themselves, MFFC's are shown to be less susceptible to mentioned variations.

4.2.1 Framing

The sound signal is sampled and the sampled data is stored in an array. The size of the buffer varies depending upon the time for which the input sound signal is taken. Hence the length of the real-time sound signal is variable. So, before performing FFT on this data, it is necessary to split the data into uniform frames on which the FFT could be performed.

In this step the continuous speech signal is blocked into frames of N samples, with adjacent frames being separated by M (M < N). The first frame consists of the first N samples. This process continues until all the speech is accounted for within one or more frames. Typical values for N and M are N = 256 (which is equivalent to ~ 30 msec windowing and facilitate the fast radix-2 FFT) and M = 100.

4.2.2 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 each 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 each frame. Once the data is framed, it is necessary to pass it through a window so as to reduce all spectral leakage.

4.2.2.1 Spectral leakage

The frequency spectrum of a 1000 Hz sine (or cosine) wave consists of a single sharp line. However, sine waves of other frequencies do not in general have such "clean" spectra.

This spreading out of spectral energy across several frequency "channels" is called spectral leakage. Spectral leakage affects any frequency component of a signal which does not exactly coincide with a frequency channel. Since the frequency components of an arbitrary signal are unlikely to satisfy this requirement, spectral leakage is more likely to occur than not with real-life sampled signals

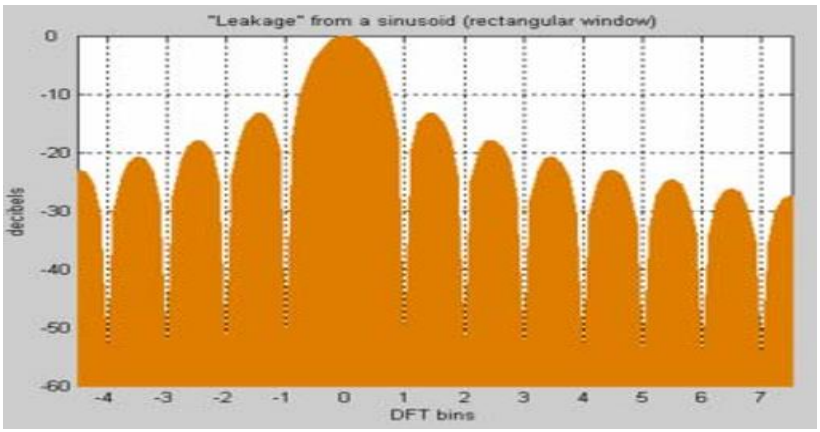


Fig 4.3 Leakage in the Sinusoid

.

4.2.2.2 Cause of spectral leakage

Spectral leakage occurs when a frequency component of a signal does not slot exactly into one of the frequency channels in the spectrum computed using the discrete Fourier transform. These "frequency channels", the frequencies represented by lines in the spectrum, are exact integer multiples (harmonics) of the fundamental frequency 1/Nh. A sine wave with a frequency coinciding with one of these frequency channels has the property that you can fit an exact integer number of sine wave cycles into the complete sample length of the sampled signal. (The number of cycles is just the harmonic number).

If there is a mismatch, the sudden jump or discontinuity created by the pattern mismatch gives rise to the spurious components in the spectrum of the signal, causing a particular frequency component of the signal to appear not as a single sharp line but as a spread of frequencies, roughly centered around where the frequency component should be located, somewhere between the two nearest frequency channels either side.

The "real life" signals are not simple sine waves. Likewise, a speech waveform contains many components of different frequencies, and it is extremely unlikely that there will be a smooth match at the beginning and end of the sampled signal. Spectral leakage is therefore almost certainly going to affect the spectrum of any signal of practical interest.[13]

4.2.2.3 Reducing spectral leakage

The only way to avoid such leakage entirely would be to arrange that all the frequency components of the signal being examined coincide exactly with frequency channels in the computed spectrum. This, however, is impractical for an arbitrary signal containing many (usually unknown) frequency components.

While spectral leakage cannot in general be eliminated completely, its effects can be reduced. This is done by applying a window function to the sampled signal. The sampled values of the signal are multiplied by a function which tapers toward zero at either end, so that the sampled signal, rather than starting and stopping abruptly, "fades" in and out like some music CD tracks. This reduces the effect of the discontinuities where the mismatched sections of the signal join up and hence also the amount of leakage.[13]

- **Hammimg Window:**

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 each 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 each frame. Once the data is framed, it is necessary to pass it through a window so as to reduce all spectral leakage.

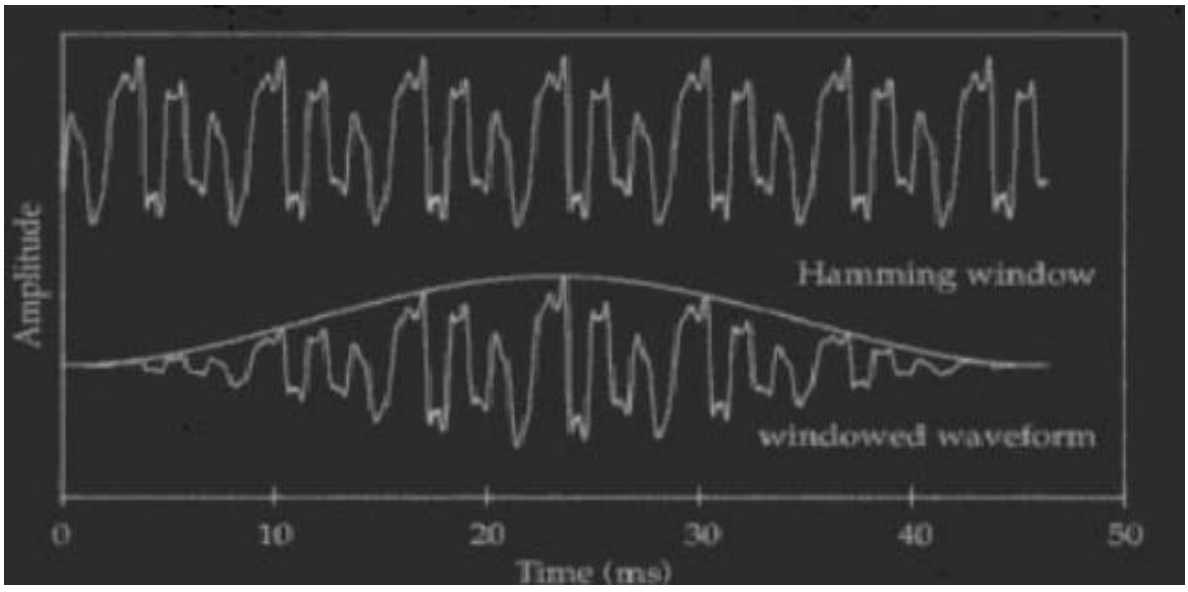


Fig.4.4 Hamming Window

The Hamming window reduces the amplitudes of the samples near the edges of the waveform chunk. The Hamming window should be in the conjunction with FFT analysis. The signal Typically the Hamming window is used, which has the form:

$$W(n) = 0.54 – 0.46 \cos(2\pi n/(N-1); \quad 0 \leq n \leq N-1$$

4.2.3 Fast Fourier Transform

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 Fast Fourier transform (FFT) is a discrete Fourier transform algorithm which reduces the number of computations needed for points from 2N2 to 2NlgN, where lg is the base-2 logarithm. The FFT is a fast algorithm to implement the Discrete Fourier Transform (DFT) which is defined on the set of N samples {xn}, as follow:

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

4.2.4 Power Spectrum of the Signal

Speech is a real signal, but its FFT has both real and imaginary components. The power of the frequency domain is calculated by summing the square of the real and imaginary components of the signal to yield a real signal. The second half of the samples in the frame are ignored since they are symmetric to the first half. (the speech signal being real)

For a given signal, the power spectrum gives a plot of the portion of a signal's power (energy per unit time) falling within given frequency bins. [14] "Power Spectra" answers the question "which frequencies contain the signal´s power?" The answer is in the form of a distribution of power values as a function of frequency, where "power" is considered to be the average of the signal. In the frequency domain, this is the square of FFT´s magnitude.

Power spectra can be computed for the entire signal at once (a "periodogram") or periodograms of segments of the time signal can be averaged together to form the "power spectral density".[15]

4.2.5 Mel-Frequency wrapping

Triangular filters are designed using the Mel frequency scale with a bank of filters to approximate the human ear. The power signal is then applied to this bank of filters to determine the frequency content across each filter. Twenty filters are chosen, uniformly spaced in the Mel-frequency scale between 0 and 4 kHz. The Mel-frequency spectrum is computed by multiplying the signal spectrum with a set of triangular filters designed using the Mel scale. For a given frequency f, the Mel of the frequency is given by

$$B(f) = [1125 \ln (1+f /700)] \text{ mels}$$

If m is the Mel, then the corresponding frequency is

$$B\text{-}1(m) = [700 \exp (2595) - 700] \text{ Hz}$$

The frequency edge of each filter is computed by substituting the corresponding Mel. Once the edge frequencies and the center frequencies of the filter are found, boundary points are computed to determine the transfer function of the filter. [16]

4.2.6 Conversion to Decibels

After calculating the Mel Frequency Coefficients, we scale them using the logarithmic scale. A logarithmic scale is a scale of measurement that uses the logarithm of a physical quantity instead of the quantity itself.

A reason for using the decibel is that different sound signals together produce very large range of sound pressures. Because the power in a sound wave is proportional to the square of the pressure, the ratio of the maximum power to the minimum power is in (short scale) trillions. We need to plot the power spectrum of these signals where we need to deal with such a range, so we choose this conversion to decibels.

After finding out the power spectrum, the log Mel spectrum has to be converted back to time. The result is called the Mel frequency cepstrum coefficients (MFCCs). 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 are real numbers (and so are their logarithms), they may be converted to the time domain using the Discrete Cosine Transform (DCT).

4.2.7 Discrete Cosine Transform

The final stage in extracting MFCC feature vectors is to apply a discrete cosine transform (DCT). The DCT serves two purposes. First, the DCT performs the final part of a cepstral transformation which separates the slowly varying spectral envelope (or vocal tract) information from the faster varying speech excitation. Lower order coefficients represent the slowly varying vocal tract while higher order coefficients contain excitation information. For speech recognition, vocal tract information is more useful for classification than excitation information. Therefore, to create the final MFCC vector, the output vector from the DCT is truncated to retain only the lower order coefficients. The second purpose of DCT is to decorrelate the elements of the feature vector making it suitable for diagonal covariance matrix statistical classifiers.

4.2.8 MEL FREQUENCY FILTER BANK

Mel-frequency analysis of speech is based on human perception experiments. It has been proved that human ears are more sensitive and have higher resolution to low frequency compared to high frequency. Hence, the filter bank is designed to emphasize the low frequency over the high frequency. Also the voice signal does not follow the linear frequency scale used in FFT. Hence, a perceptual scale of pitches equal in distance, namely Mel scale is used for feature extraction. Mel scale frequency is proportional to the logarithm of the linear frequency, reflecting the human perception. We use log because our ears work in decibels.

$$mel(f) = 2595 * \log(1 + f / 700)$$

Triangular band pass filters are used to extract the spectral envelope, which is constituted by dominant frequency components in the speech signal. Thus, Mel-frequency filters are triangular band pass filters non-uniformly spaced on the linear frequency axis and uniformly spaced on the Mel frequency axis, with more number of filters in the low frequency region and less number of filters in the high frequency region.

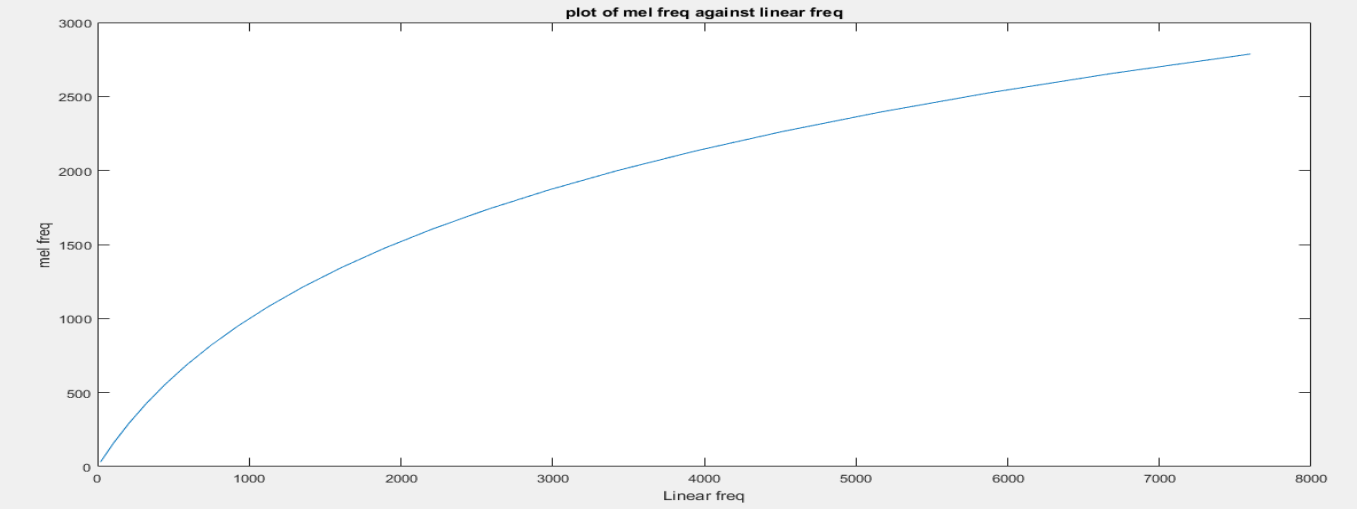


Fig 4.5 Plot Linear frequency vs Mel frequency

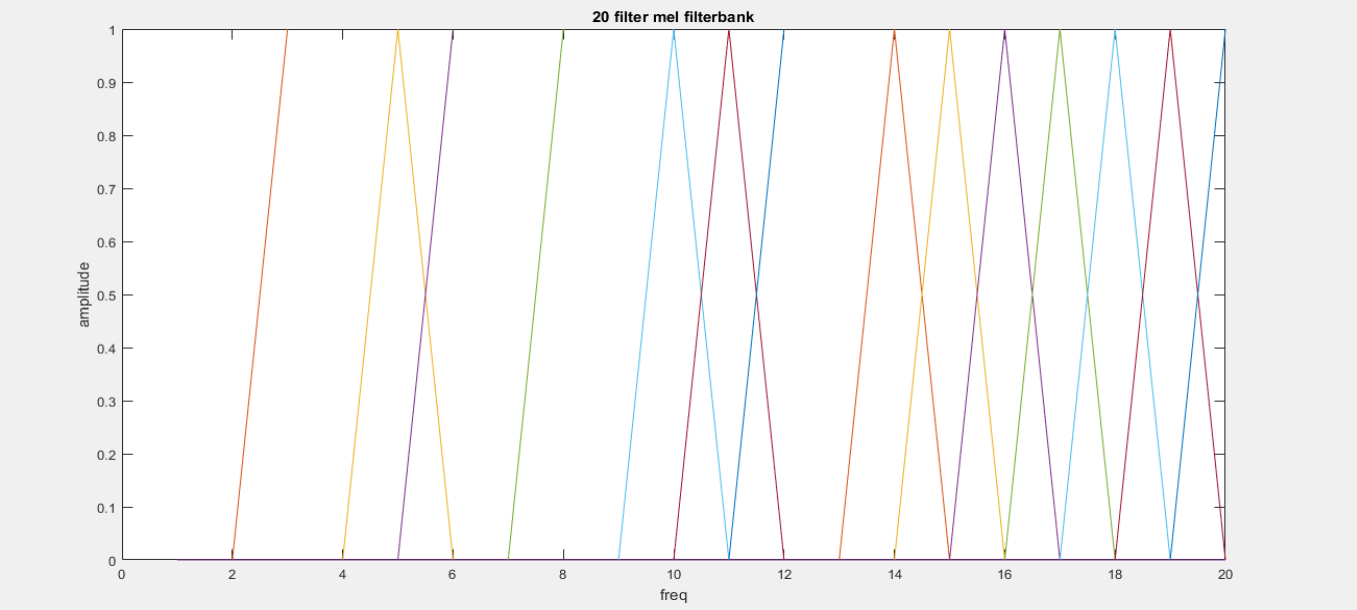


Fig 4.6 Mel filterbank

The number of Mel cepstrum coefficients, K, is typically chosen as 20. The first component is excluded from the DCT since it represents the mean value of the input signal which carries little speaker specific information. By applying the procedure described above, for each speech frame of about 30ms, a set of Mel-frequency cepstrum coefficients is computed. This set of coefficients is called an acoustic vector. These acoustic vectors can be used to represent and recognize the voice characteristic of the speaker. Therefore each input utterance is transformed into a sequence of acoustic vectors.

CHAPTER – 5

IMPLEMENTATION OF THE PROJECT

5.1 Overview of the Project

The aim of this project is to determine the identity of the speaker from the speech sample of the speaker and the trained vectors. Trained vectors are derived from the speech sample of the speaker at a different time.

First the input analog speech signal is digitized at 8 KHz sampling frequency using the on board ADC (Analog to Digital Converter). The Speech sample is stored in a one-dimensional array. The speech signal is split into frames. Each frame consists of 128 Samples of Speech signal. Speech sample in one frame is considered to be stationary.

After Framing, to prevent the spectral leakage we apply windowing. Here Hamming window with 128 coefficients is used.

Third step is to convert the Time domain speech Signal into Frequency Domain using Discrete Fourier Transform. Here Fast Fourier Transform is used. The resultant transformation will result in a signal being complex in nature. Speech is a real signal but its Fourier Transform will be a complex one (Signal having both real and imaginary).

The power of the signal in Frequency domain is calculated by summing the square of Real and Imaginary part of the signal in Frequency Domain. The power signal will be a real one.

Triangular filters are designed using Mel Frequency Scale. This bank of filters will approximate our ears. The power signal is then applied to this bank of filters to determine the frequency content across each filter. In our implementation we choose total number of filters to be 20. These 20 filters are uniformly spaced in Mel Frequency scale between 0-4KhZ. After computing the Mel-Frequency Spectrum, log of Mel-Frequency Spectrum is computed. Discrete Cosine Transform of the resulting signal will result in the computation of the Mel-Frequency Cepstral Co-efficient. Euclidean distance between the trained vectors and the

Mel-Frequency Cepstral Coefficients are computed for each trained vectors. The trained vector that produces the smallest distance will be identified as the speaker.

5.2 Flowchart of the program:

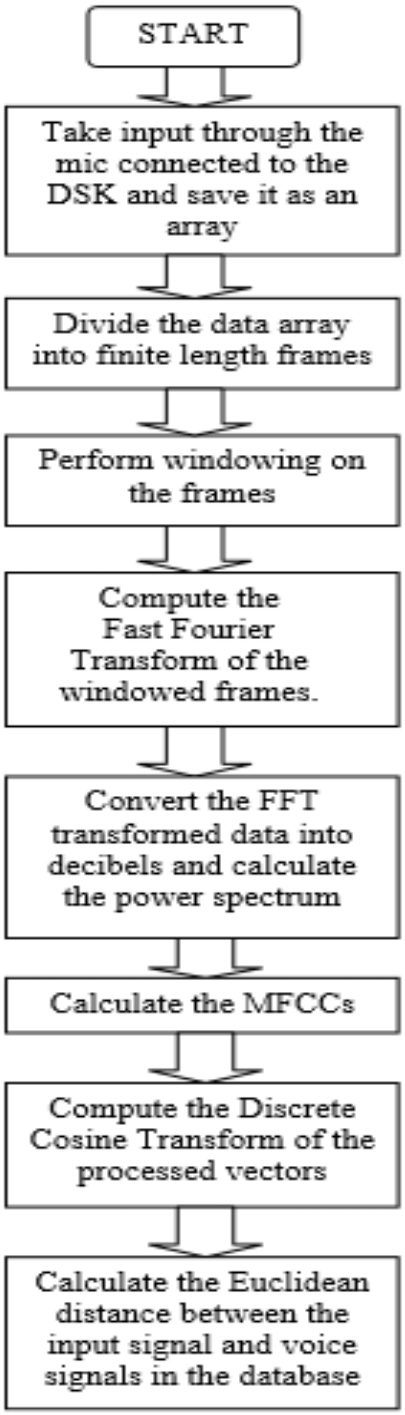


Fig 5.1flowchart of the program

CHAPTER - 6

RESULTS AND ANALYSIS

6.1 RESULTS

6.1.1 SPEAKER RECOGNITION RESULT OF MATLAB:

1) Input signal

- Fig 6.1 represents speech signal without any pre-processing.

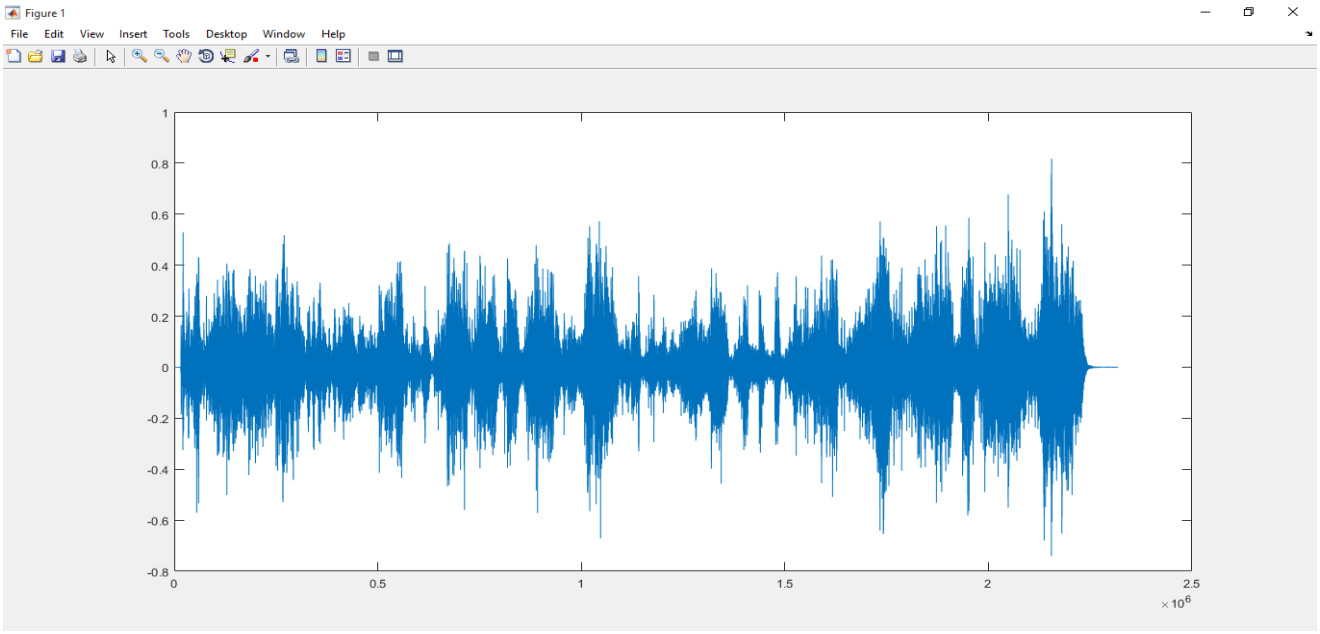


Fig 6.1 Speech Signal

2.) Framing

- MFCC algorithm's first step is framing. Fig 6.2 showing one frame with duration 0.002 sec and fig 6.3 showing frame having duration 0.20 sec.

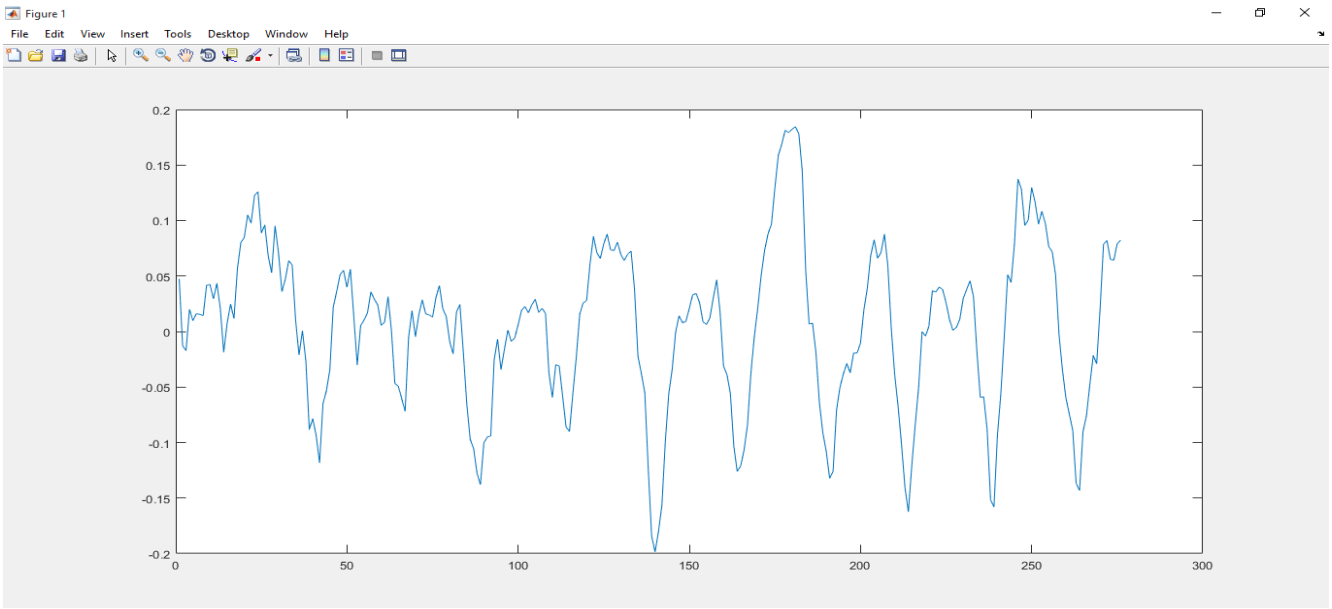


Fig 6.2 Frame1 of Speech signal (0.002 sec)

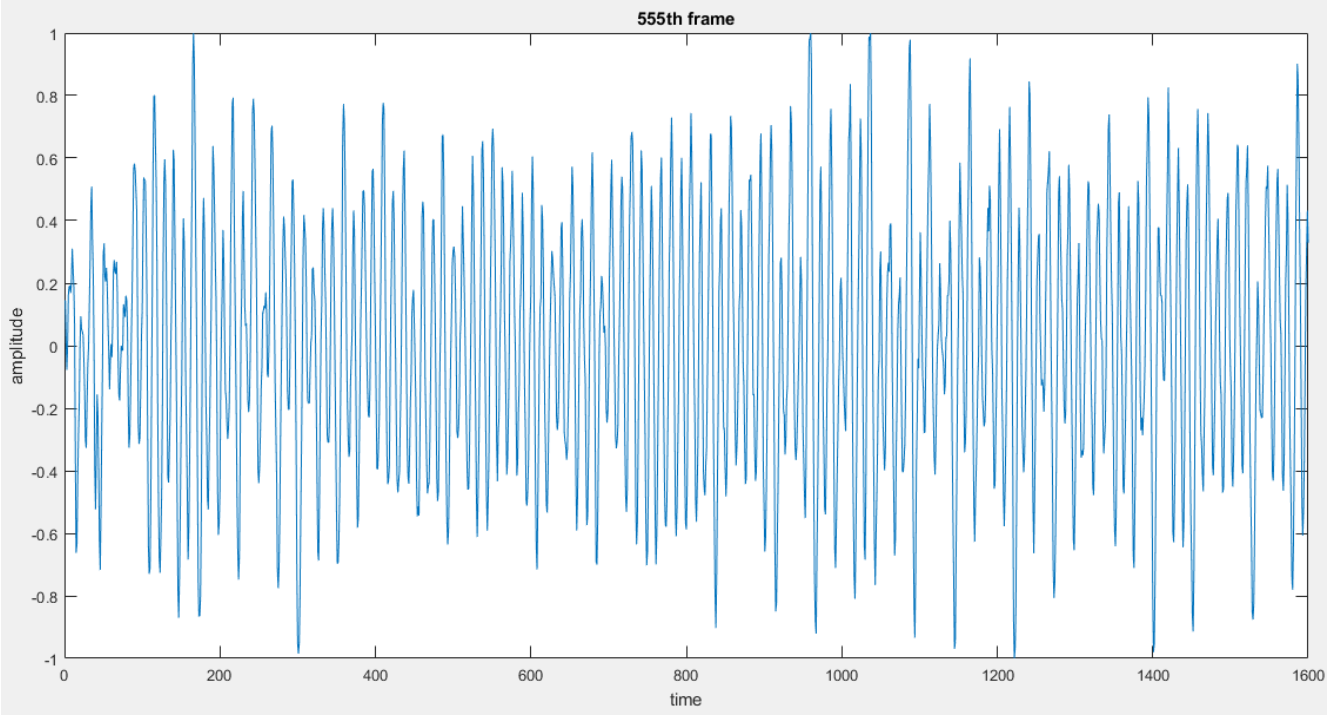


Fig 6.3 Frame2 of Speech signal(0.20 sec)

2) Windowing

- 2nd step of algorithm is Windowing. We have used Hamming window. We have to apply windowing on each frame of speech signal. Fig 6.4 showing frame after applying hamming window.

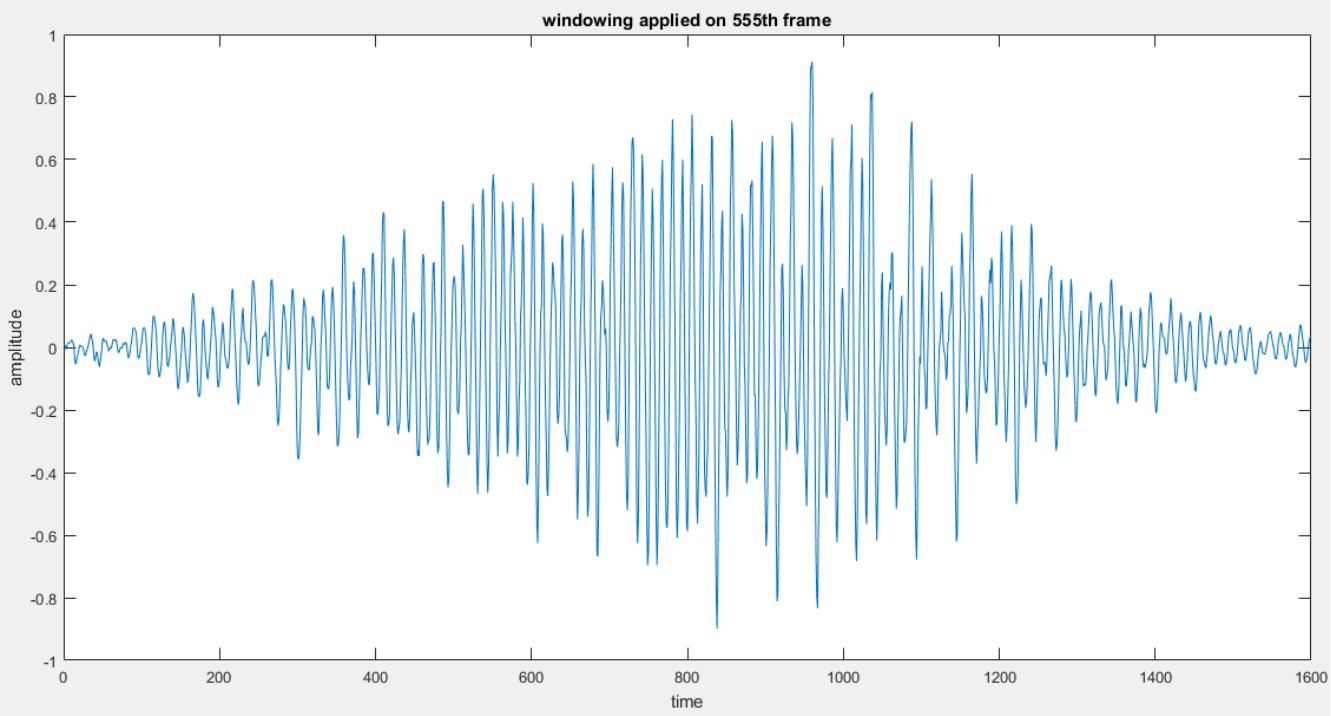


Fig 6.4 Hamming window applied on frame2

3) FFT

- 3rd step is Fast Fourier Transform. Fig 6.5 showing difference in results of fft ,without applying windowing & with applied windowing on frame

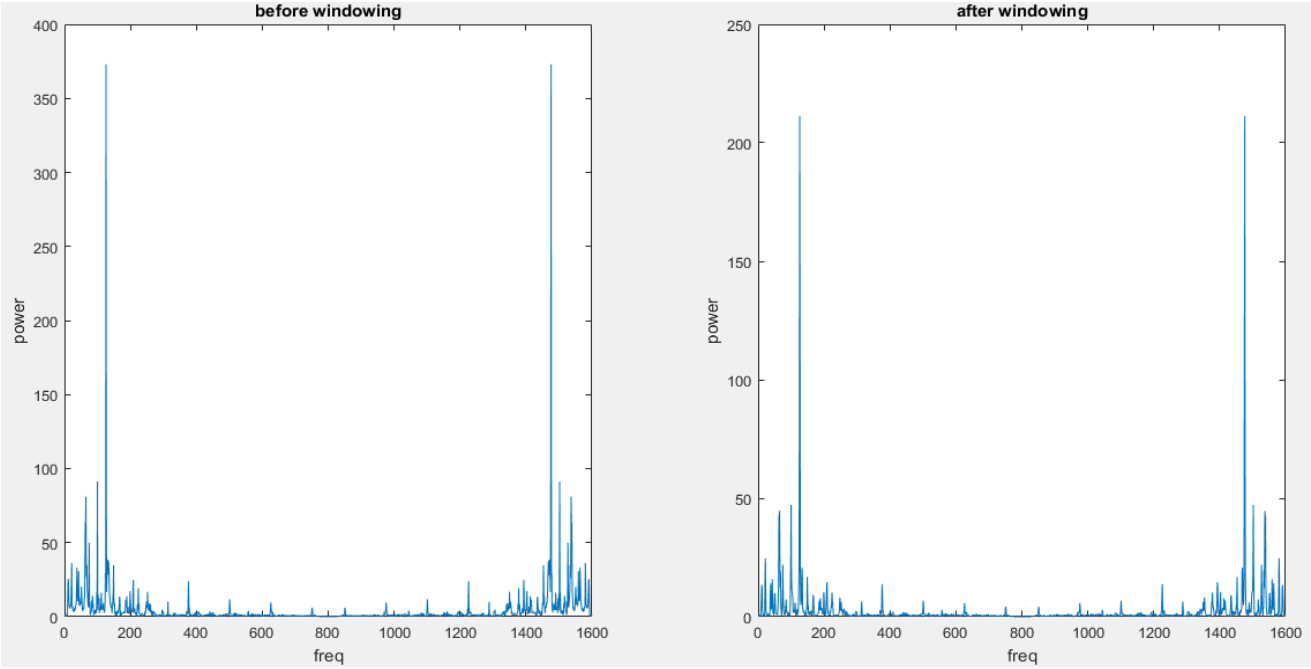


Fig 6.5 FFT of frame2 before applying window & after applying window

4) Power spectrum

- 4th step is power spectrum of fft result. Fig 6.6 showing power spectrum results.

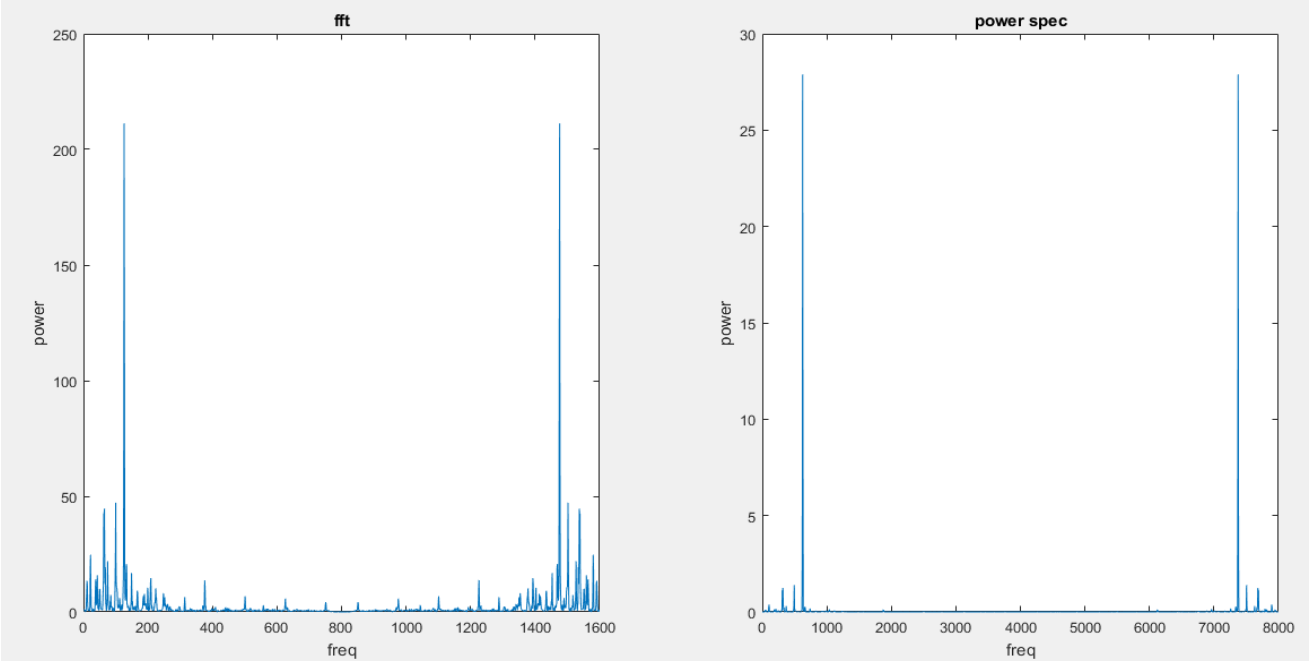


Fig 6.6 FFT &Power spectrum

5.) Mel frequency wrapping & mel-filterbank

- 5th step is converting linear scale into mel scale which is known as mel-frequency wrapping. Fig 6.7 is showing plot between linear scale & mel scale.
- We have used triangular bandpass filter in filterbank. Fig 6.8 is showing mel-filterbank having 20 co-efficients.

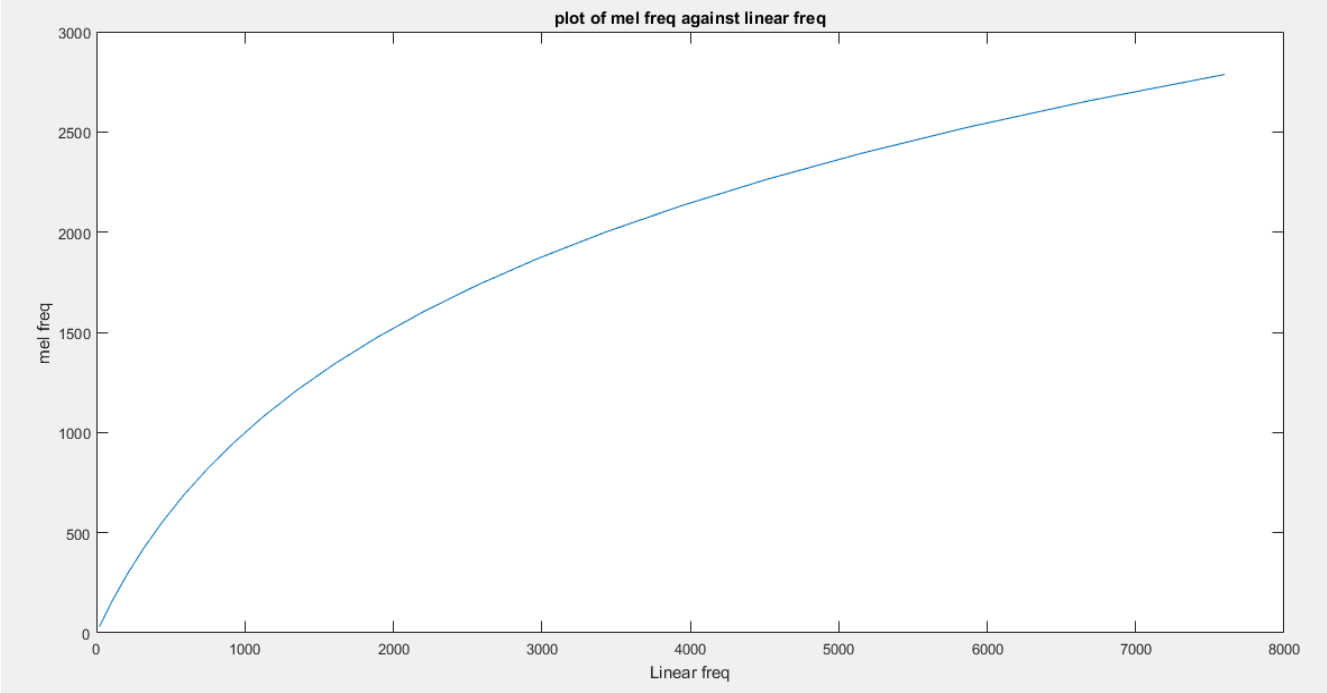


Fig 6.7 Plot linear frequency vs Mel frequency

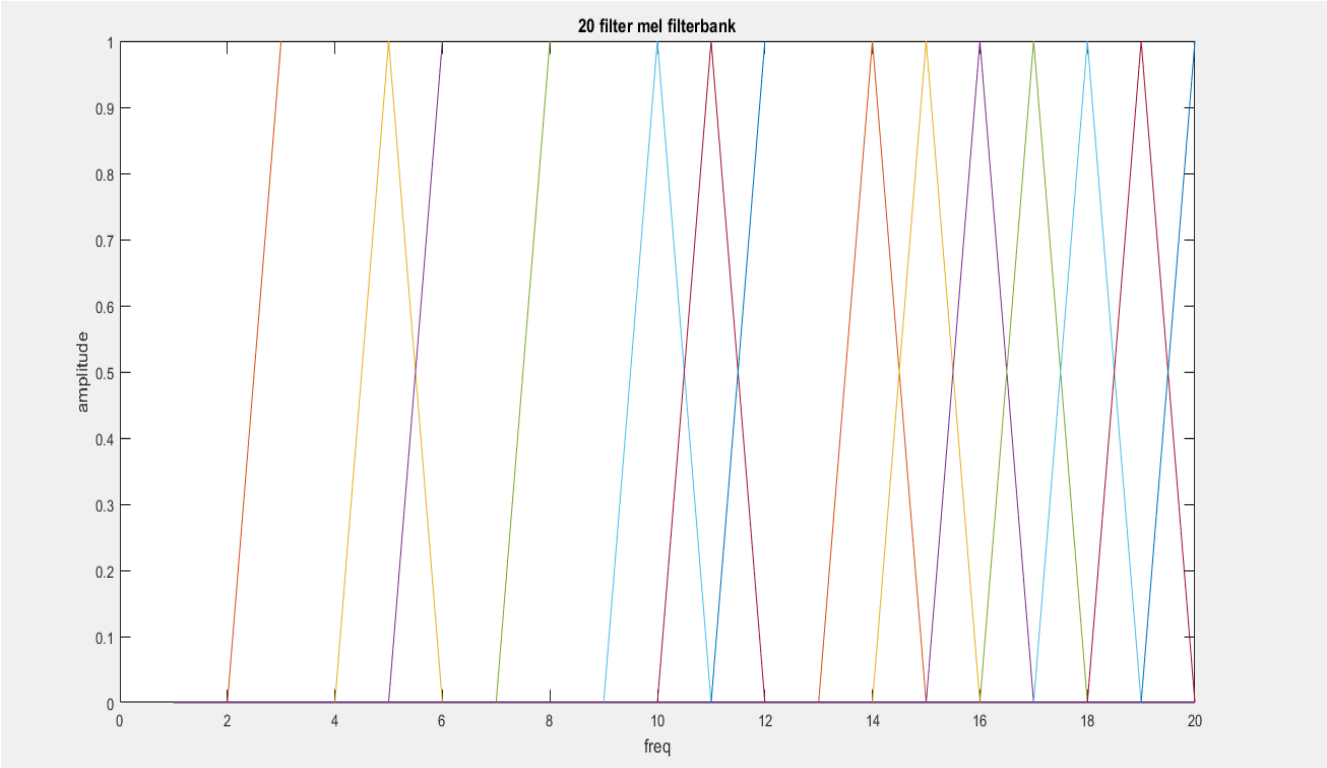
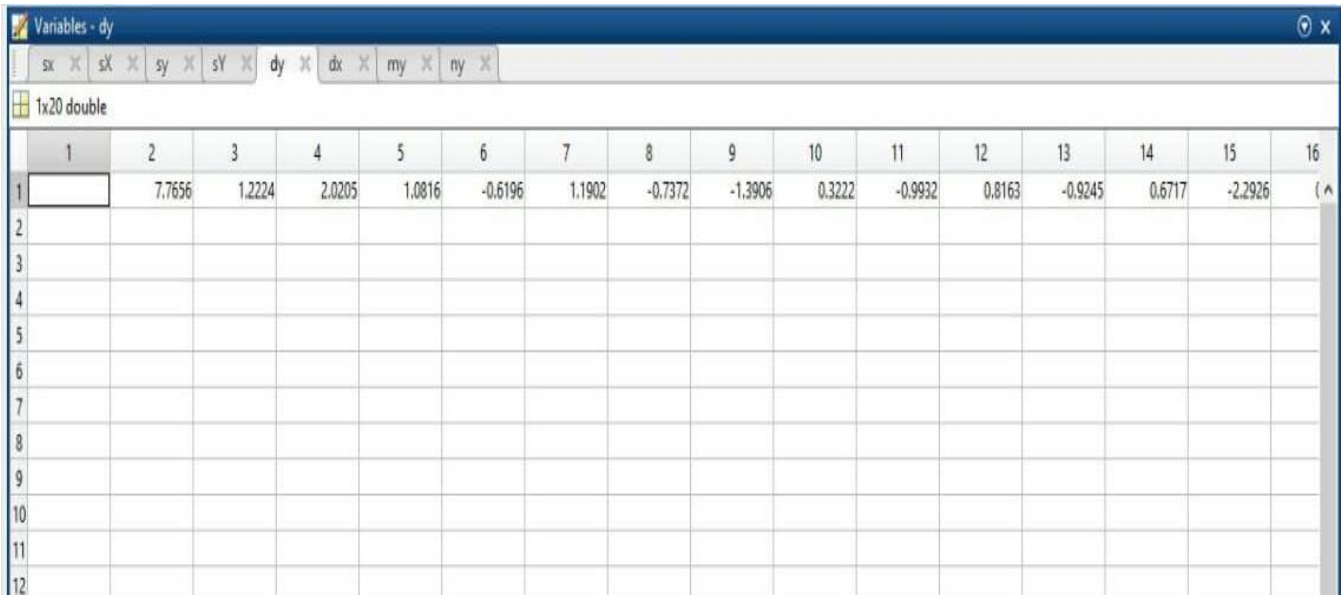


Fig 6.8 Mel filterbank

6.) DCT

- 6th & last of MFCC algorithm is DCT. Fig 6.9 is showing values of DCT. We called it acoustic vector.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		7.7656	1.2224	2.0205	1.0816	-0.6196	1.1902	-0.7372	-1.3906	0.3222	-0.9932	0.8163	-0.9245	0.6717	-2.2926	
2																
3																
4																
5																
6																
7																
8																
9																
10																
11																
12																

Fig 6.9 Acoustic vector

I. Single user

- Training phase: recording single user's speech signal & storing it into memory.

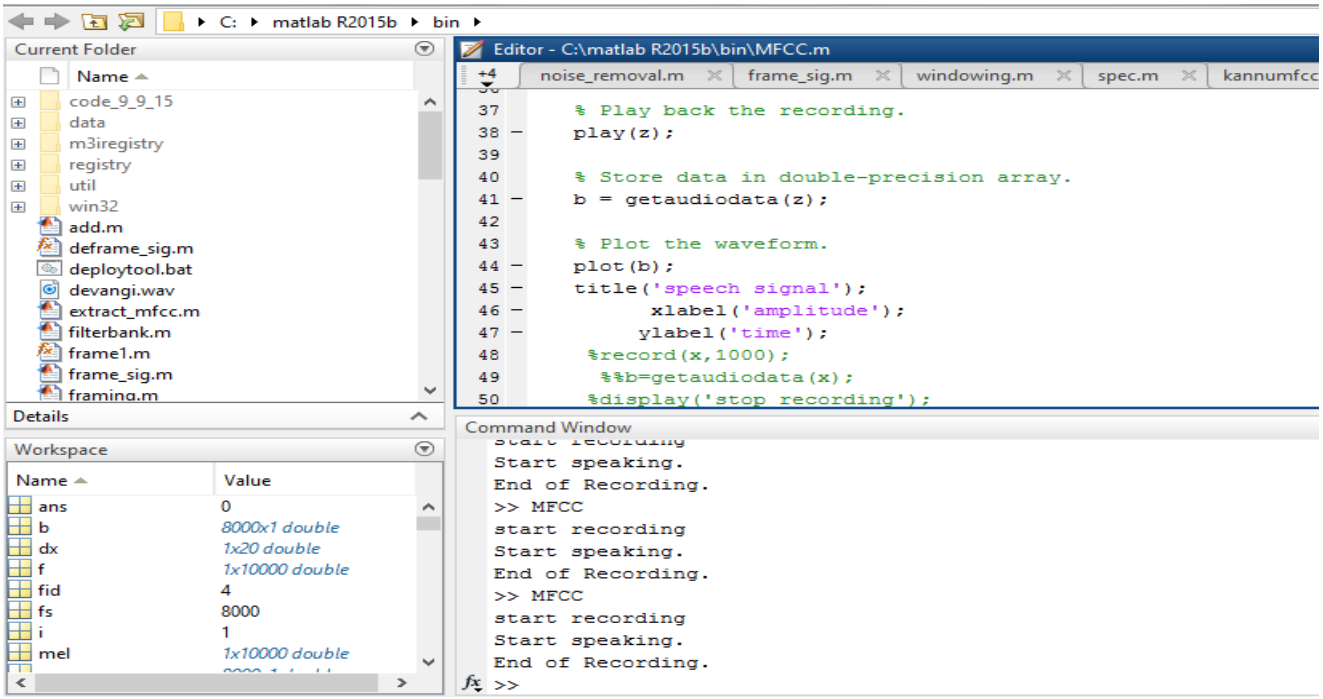


Fig 6.10 Training phase for single user

➢ Testing Phase : Comparing stored user's speech with same user & different user's speech

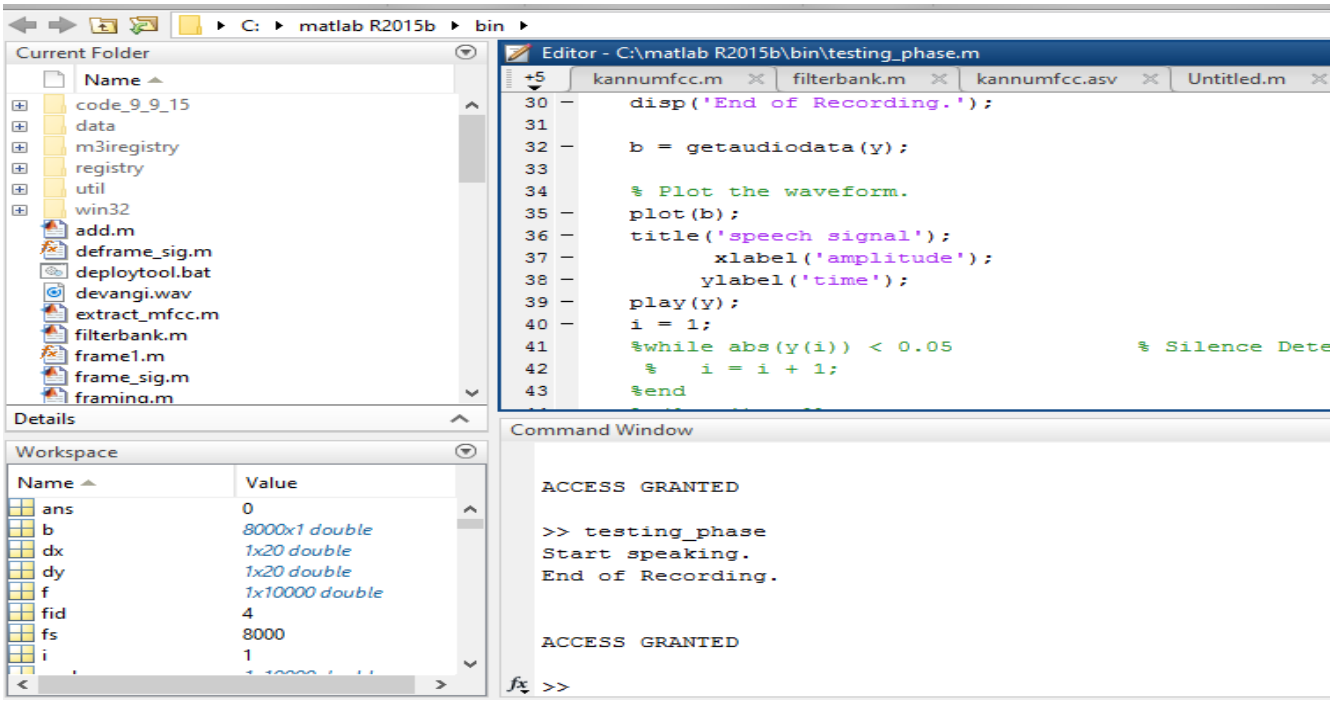


Fig 6.11 Testing phase(for same user)

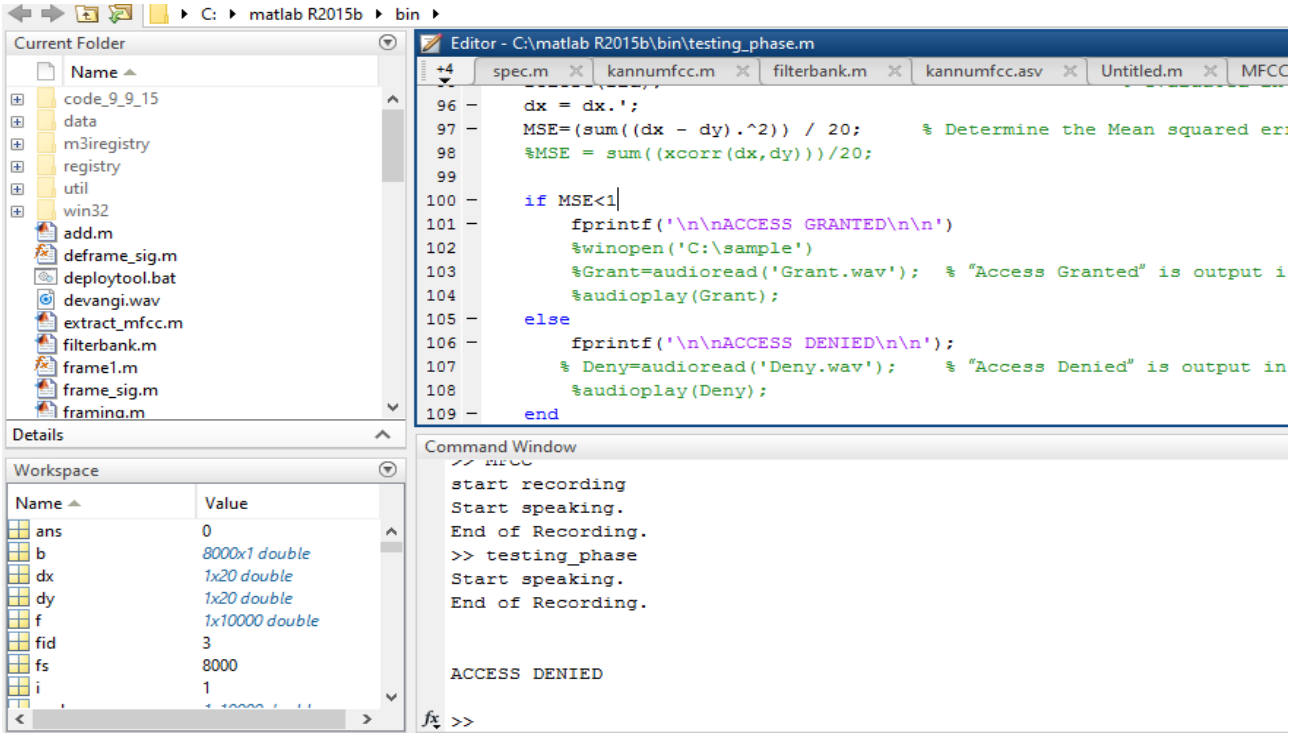


Fig 6.12 Testing phase(for different user)

II. Multiple users

- We have implemented this project with 4 users.
- This project has 2 phases.

1.) Training Phase : In training phase we are recording different users' speech signal to make a database.(Fig 6.13)

2.)Testing Phase : In testing phase we are identifying the user with reference to database which we have made in training phase.(Fig 6.14 , 6.15, 6.16, 6.17)

➢ Training Phase

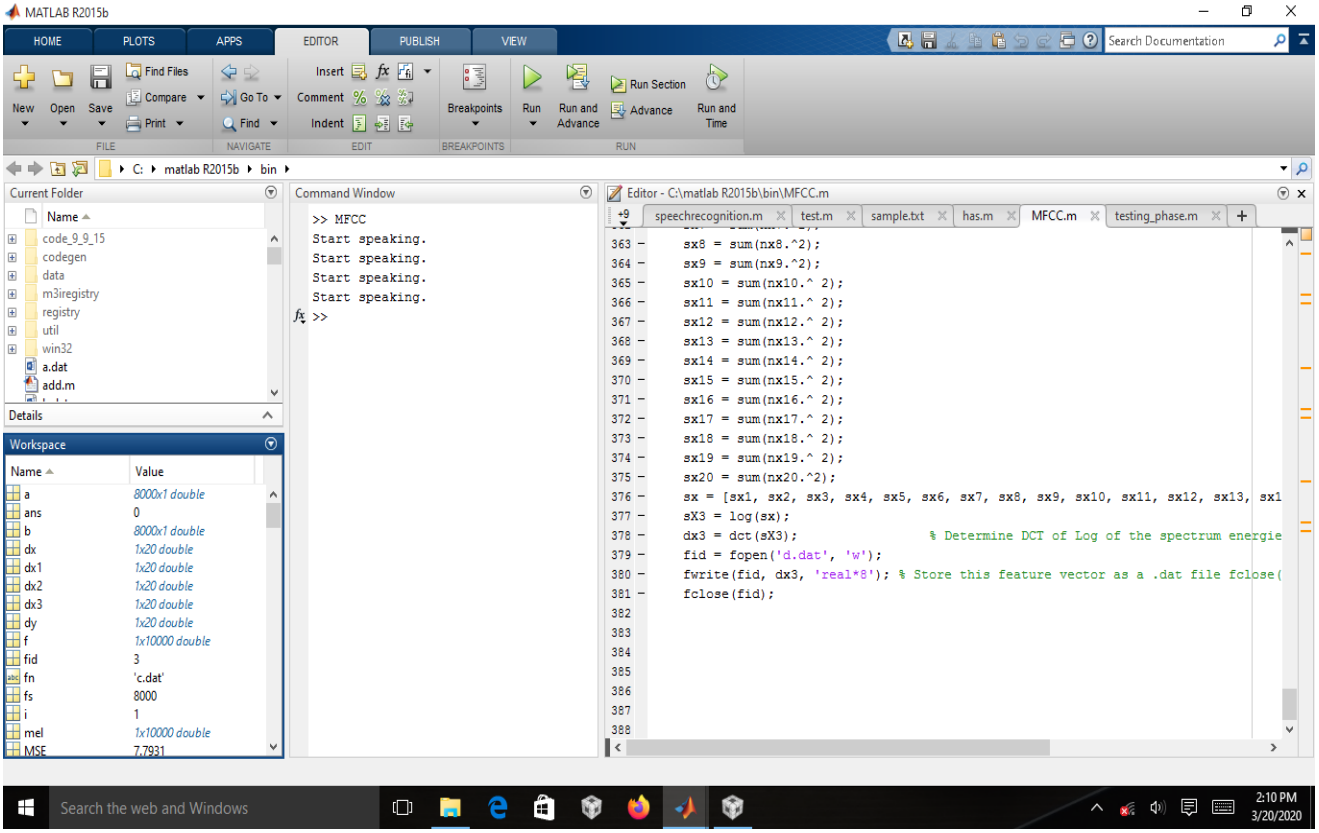


Fig.6.13 Training phase

➢ Testing phase

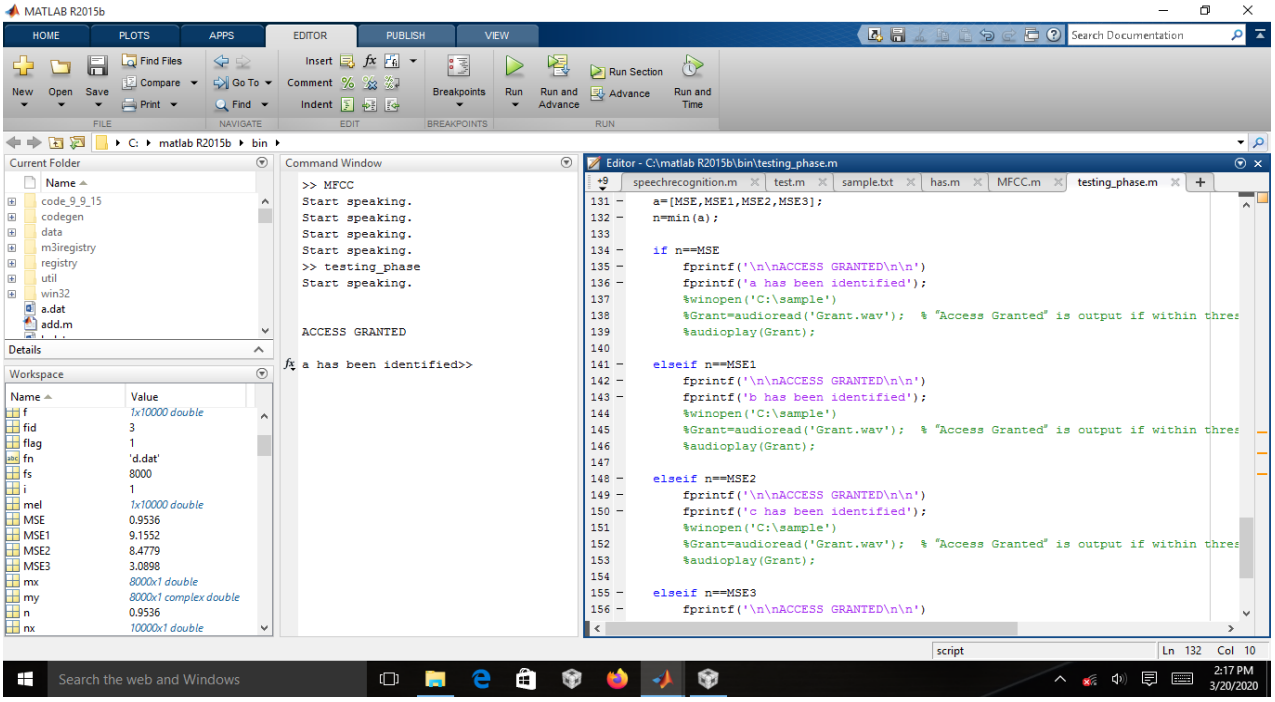


Fig.6.14 User1 identified

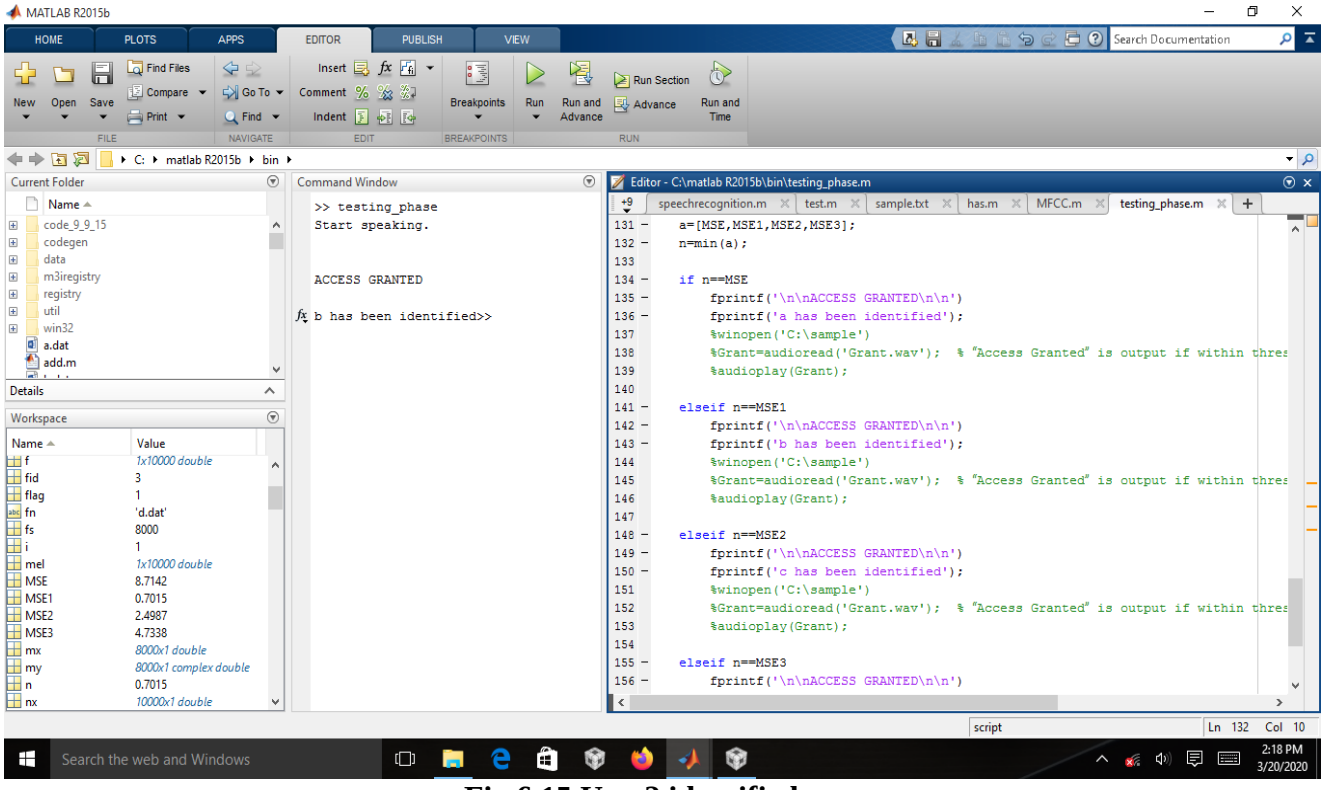


Fig.6.15 User2 identified

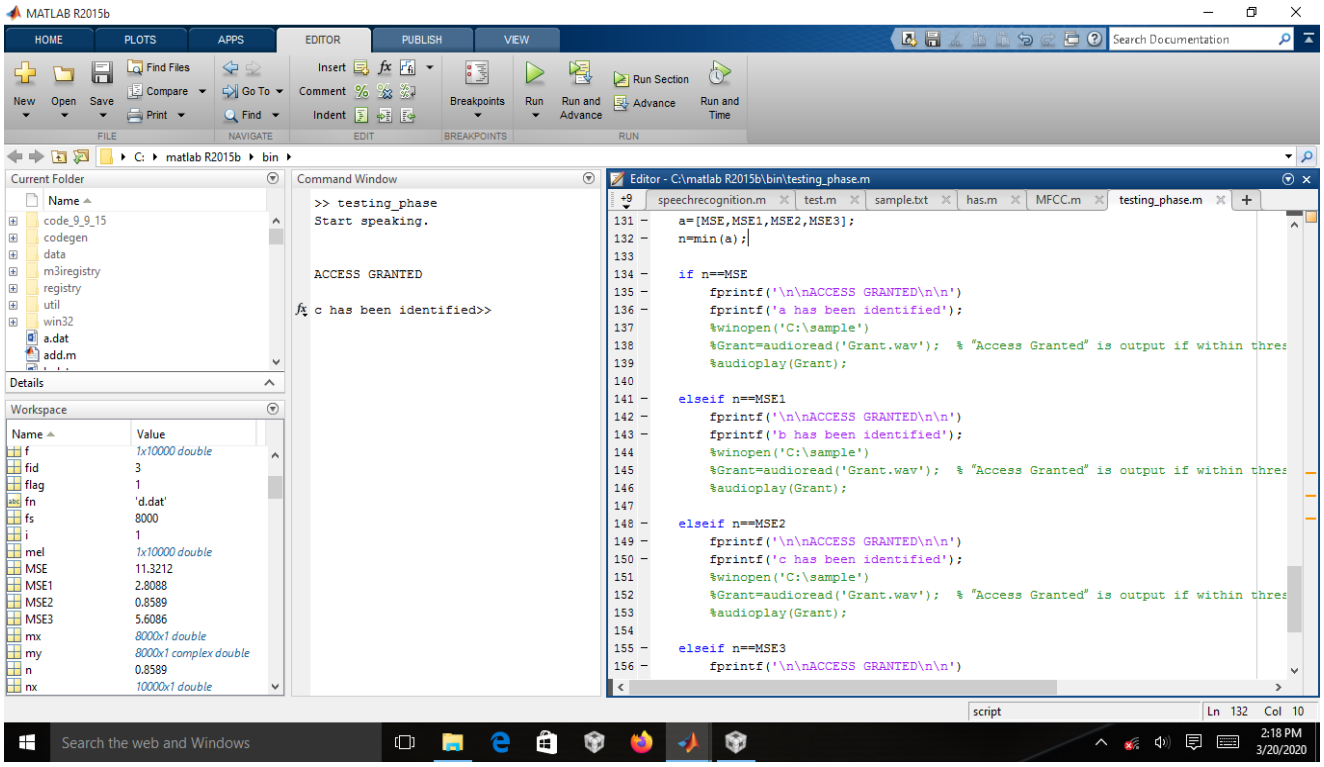


Fig.6.16 User3 identified

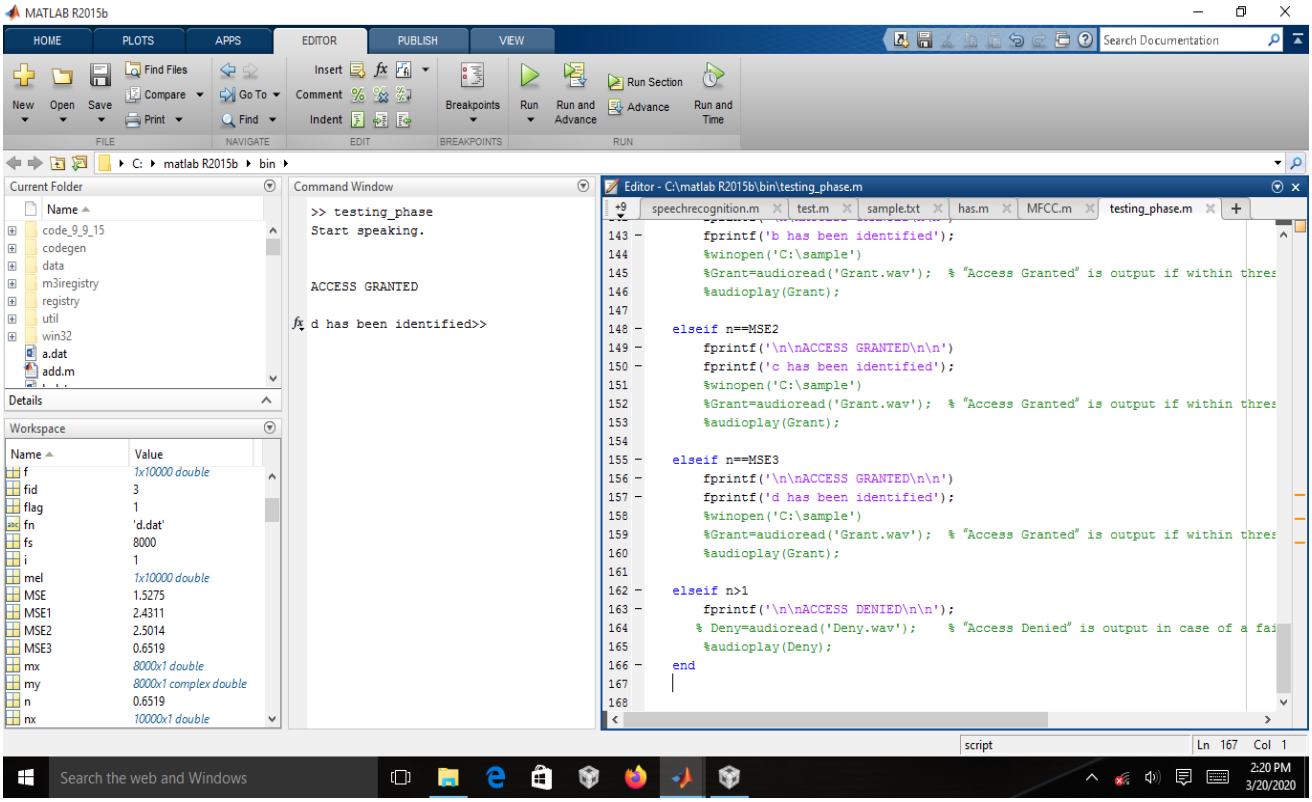


Fig 6.17 User4 identified

6.1.2 Results of ccs with hardware:

- **Audio loopback program**

- Simple audio loopback program in which we are giving speech/audio signal as a input and getting same speech signal as a output.
- Fig 6.18 showing that connection of dsk6713 board and ccs software has been completed.
- Fig 6.19 showing setup of giving audio signal in input through computer.

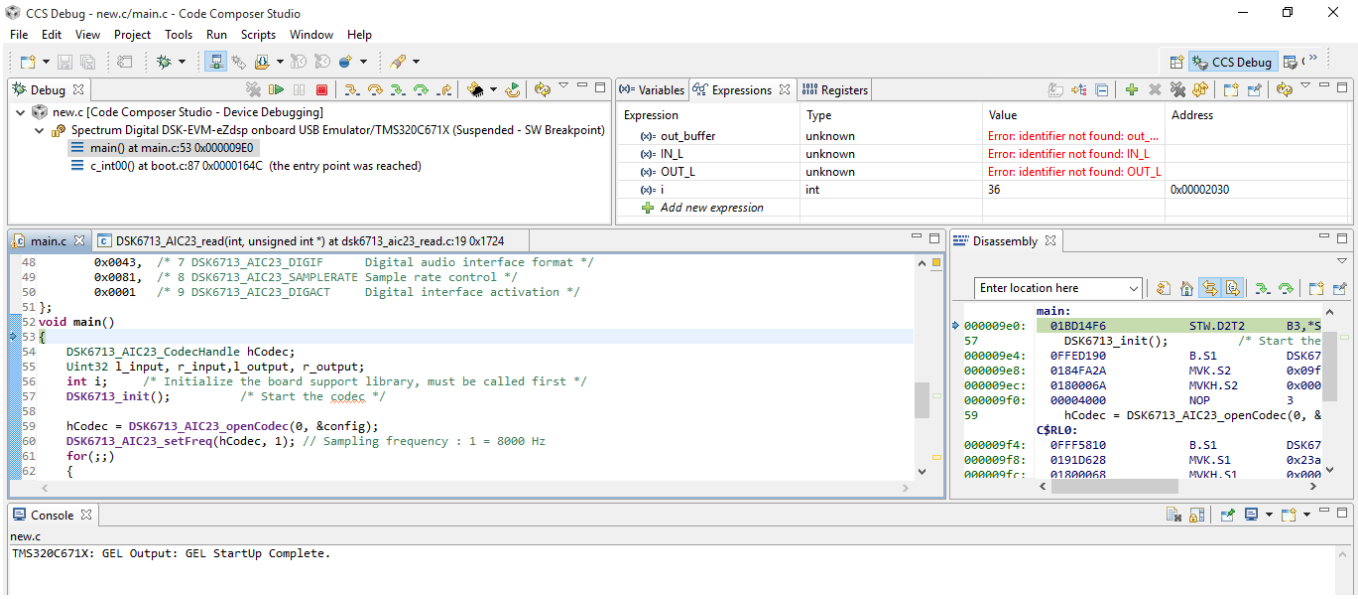


Fig 6.18 GEL startup complete

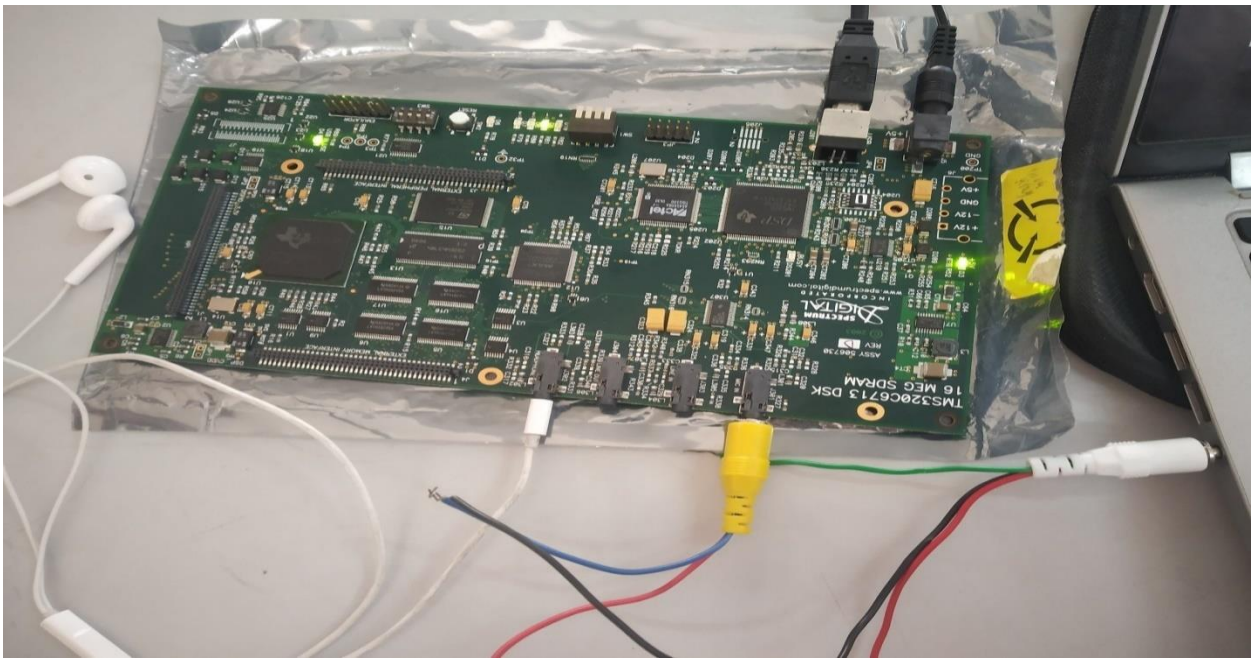


Fig.6.19 Output from Headphone pin by giving input through Computer

I. Observation of speech signal on DSO

- Audio/speech signal which we are getting in output in audio loop back program can be seen on DSO.
- Fig 6.20 showing speech/audio signal on DSO.

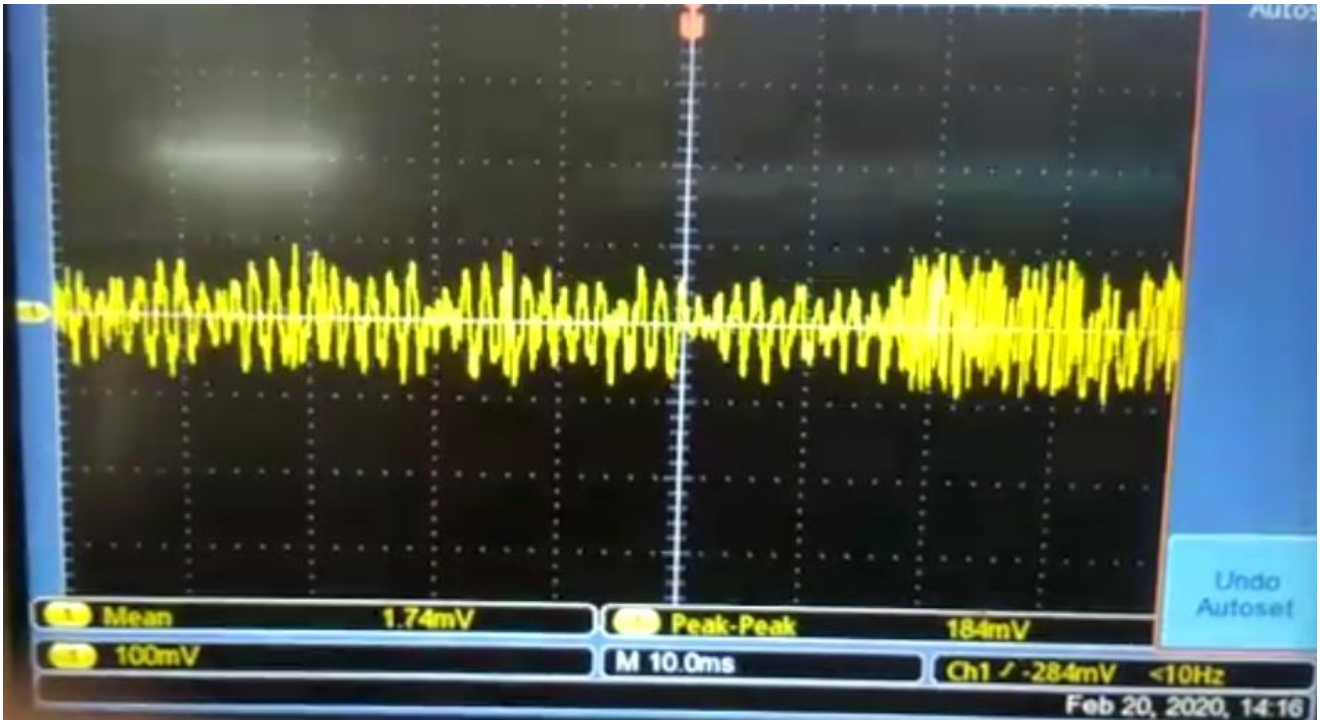


Fig 6.20 Observation of speech signal on DSO

6.1.3 Speaker recognition result of CCS

I. single user

➢ Training Phase

- Fig 6.21 showing speech signal has been recorded & MFCC applied on it.Result of MFCC which is Acoustic vector has been stored in array ‘d’.

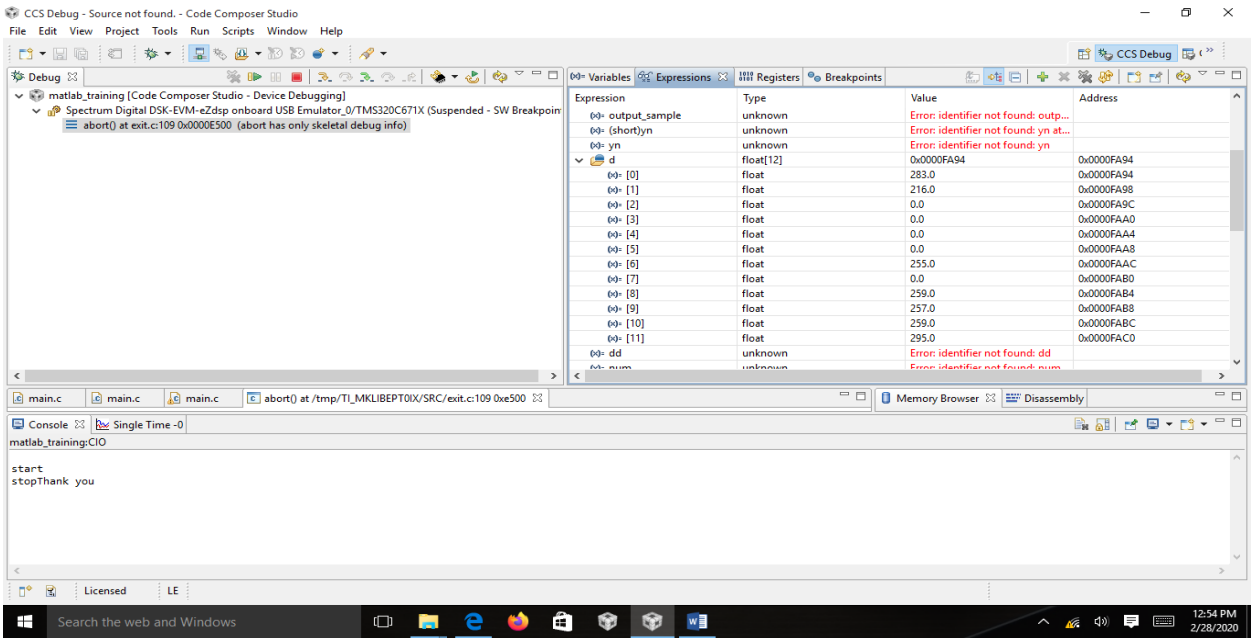


Fig 6.21 Training phase (acoustic vector stored in array 'd')

➢ Testing Phase

- Fig 6.22 showing testing phase has been performed. In testing phase we have to compare stored acoustic vector(array 'd') and input acoustic vector(array 'dd').

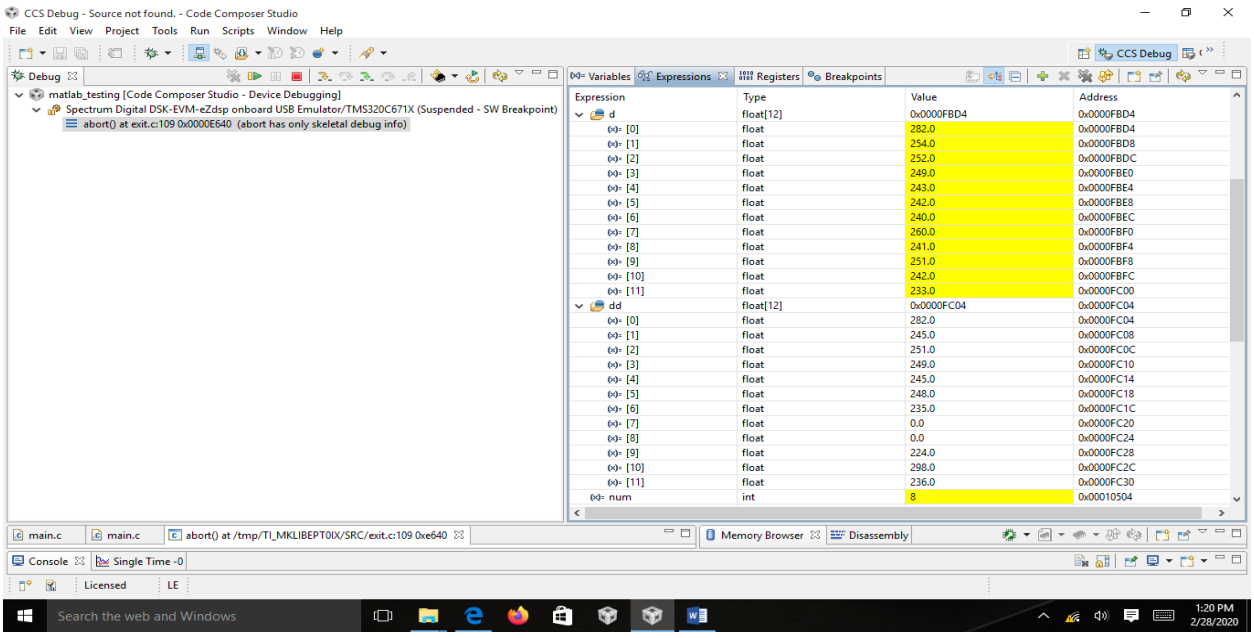


Fig 6.22 Comparison of both Vectors

- Fig 6.23 is showing result of testing phase for single user.
- Fig 6.24 is showing result of testing phase for input given from same user for 10 times. We are getting 8 out of 10 times correct user verified.

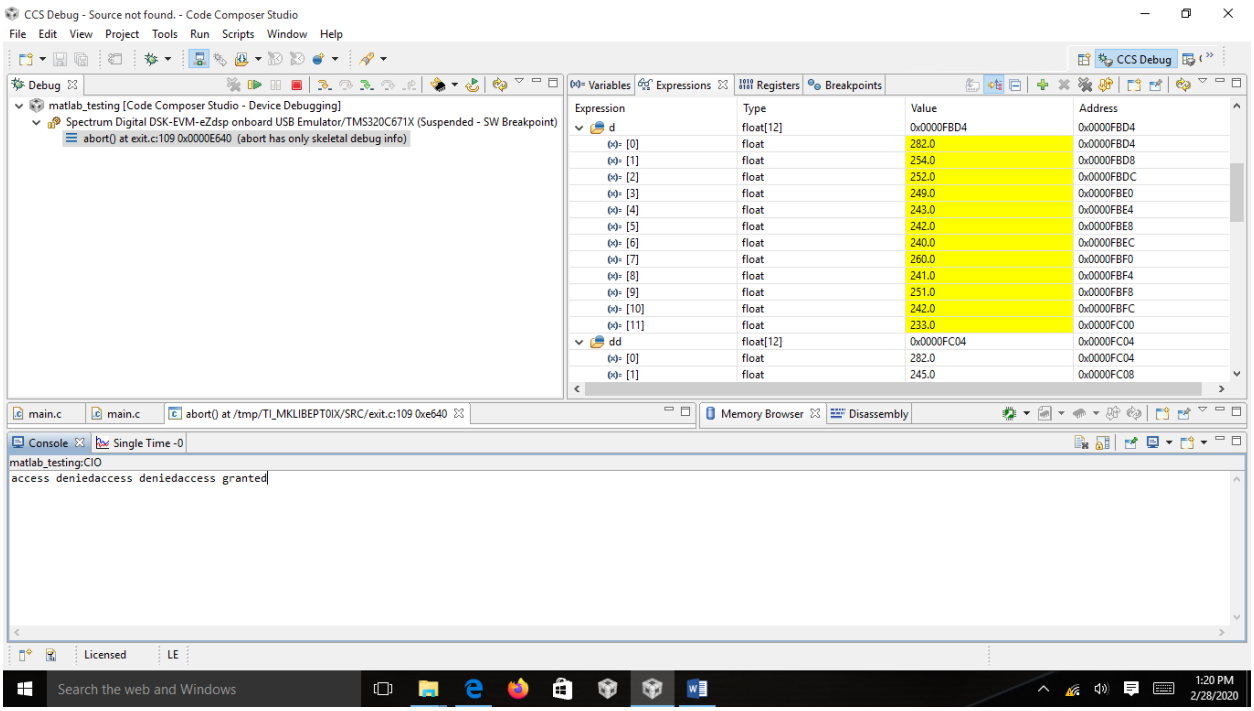


Fig 6.23 Result

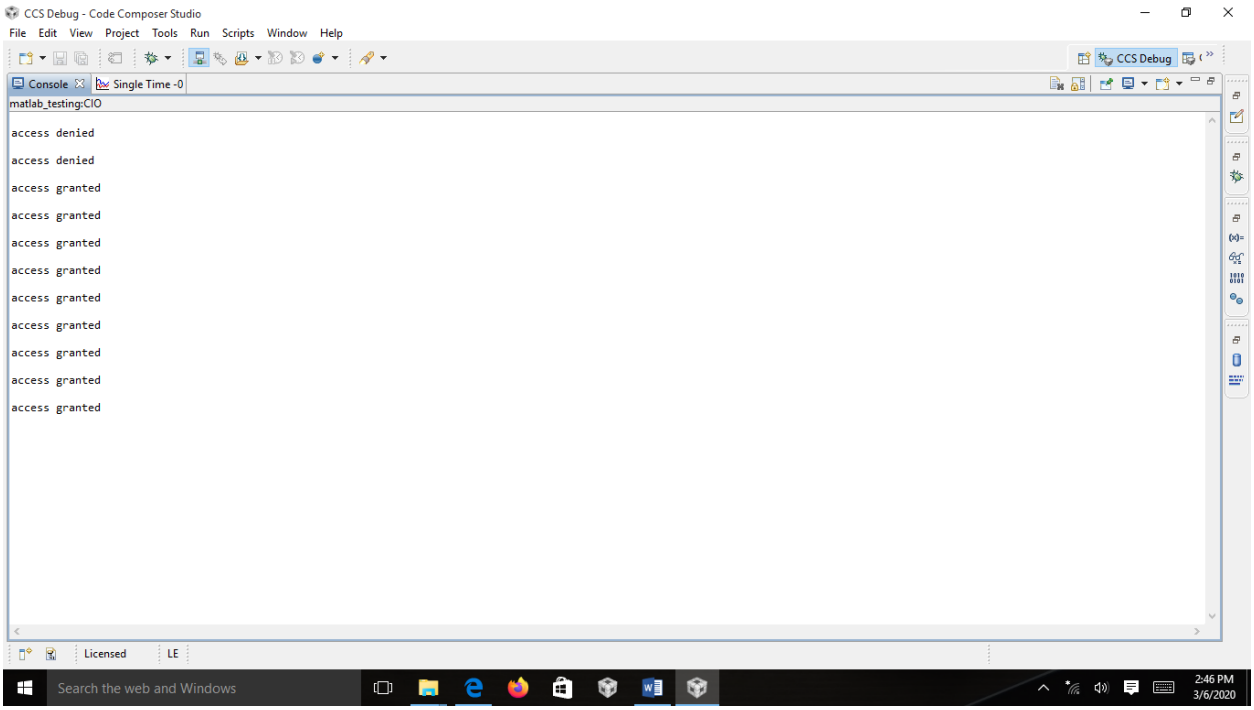


Fig 6.24 Result of testing phase for same user

II. Multiple users

➢ Training phase

- In training phase, we have recorded speech signal & stored acoustic vector into memory as a database file.(fig 6.25)
- Fig 6.26 is showing database file of stored user.

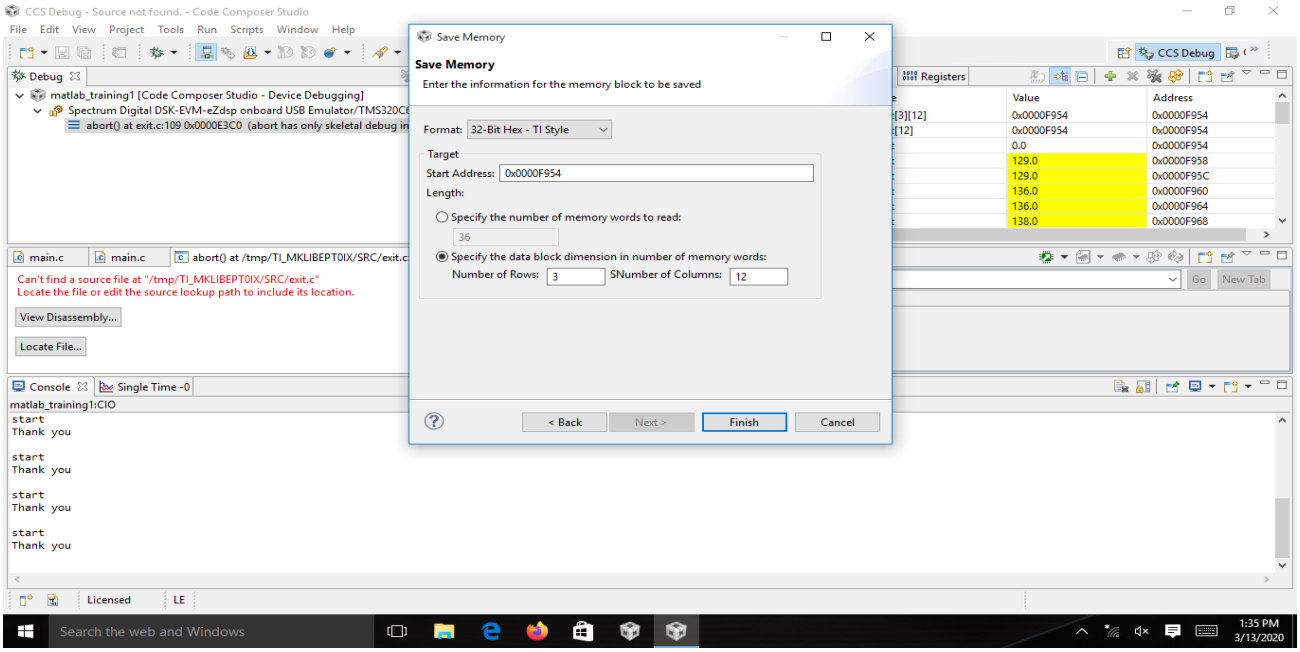


Fig 6.25 Saving Database file (training phase)

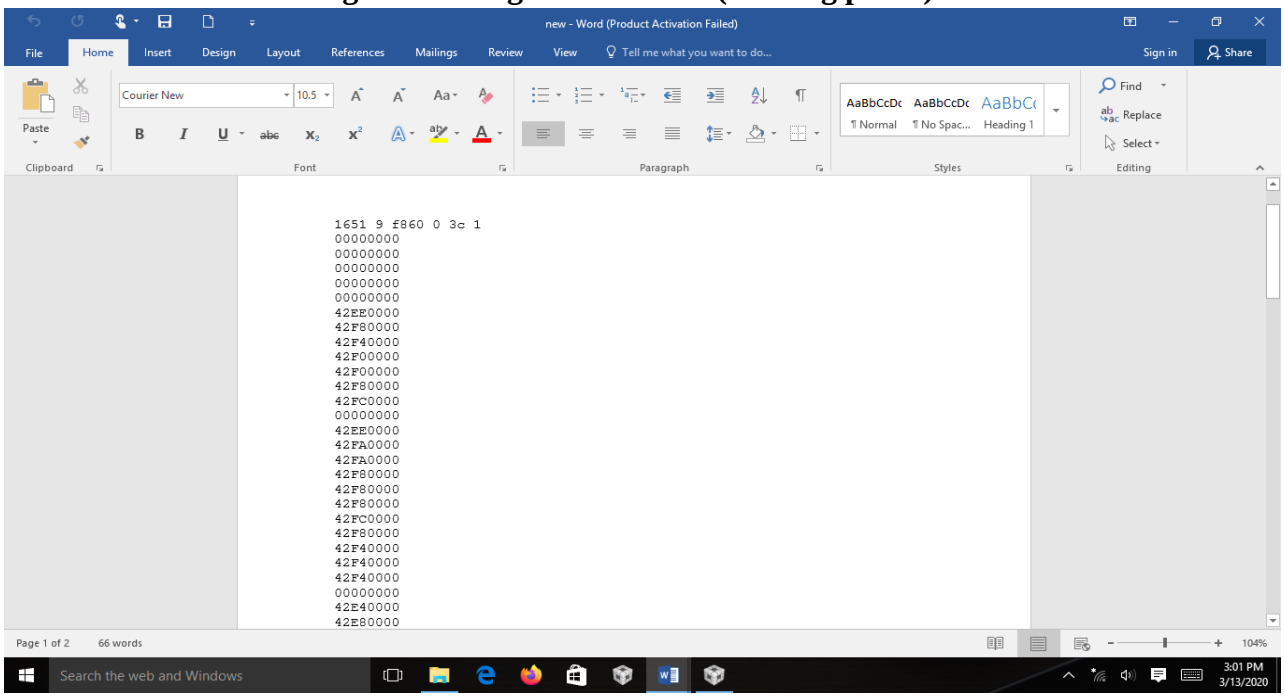


Fig 6.26 Database file

➢ Testing phase

- In testing phase, before running the program we need to load the database file into memory.

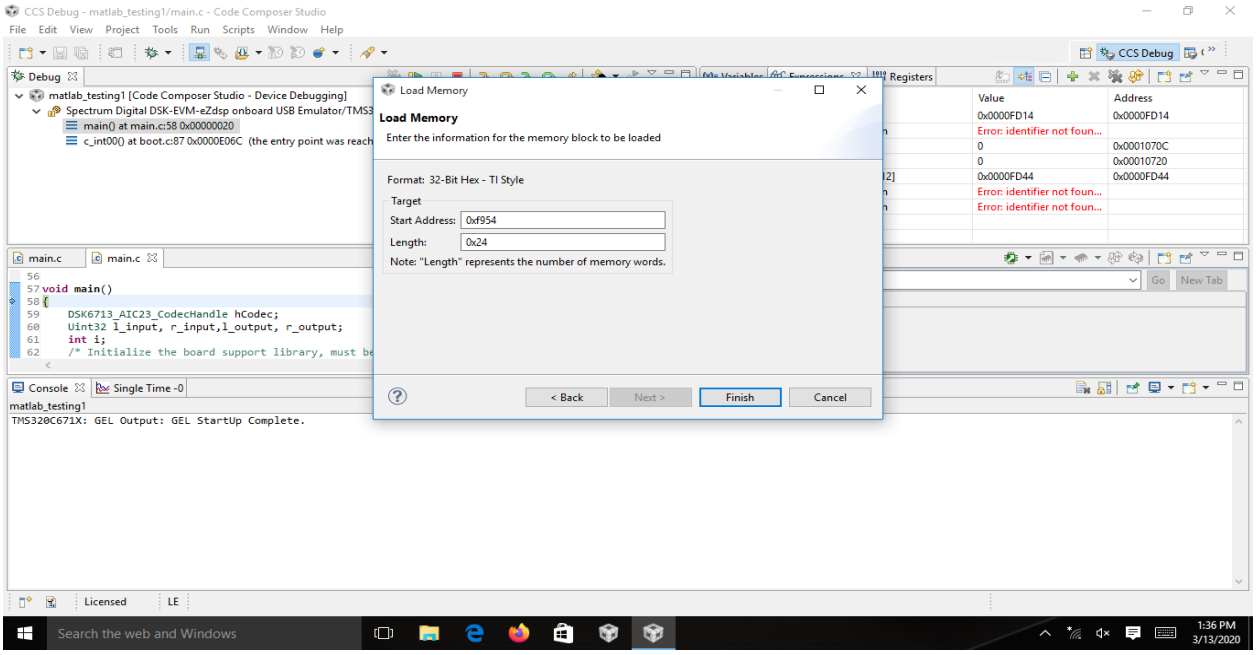


Fig 6.27 Loading Database file into memory (testing phase)

- Result1(6.28) and Result2 (6.29) are showing result of testing phase for multiple users. Name of identified user has been displayed.
- if unknown user's speech is given in input then "access denied" will be displayed, which can be seen in fig 6.29.

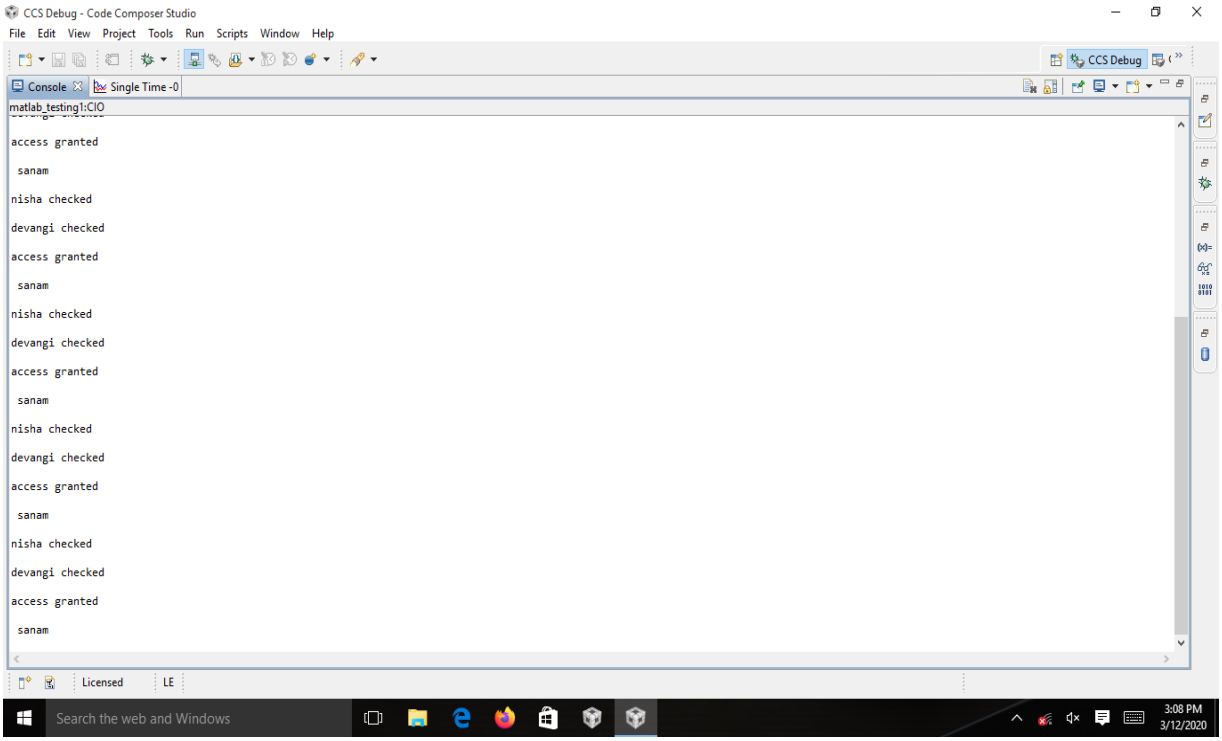


Fig 6.28 Result 1

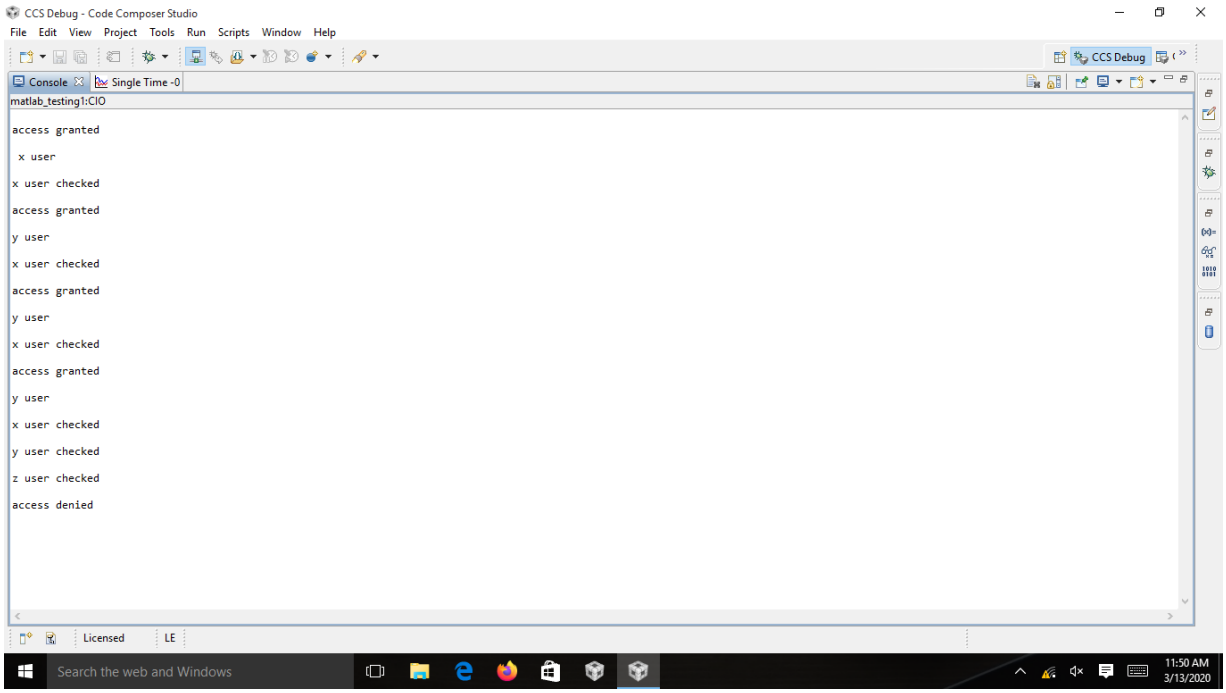


Fig 6.29 Result 2

6.2 ANALYSIS

For the speaker identification process we made a database of 4 users. Each user trained 4 times. Here there were three possibilities: Correct speaker is recognized, Incorrect Speaker is recognized or Access is denied because difference in the input sample and stored coefficients is very high. Each user was made to speak 10 times. The results are as shown below

User Number	Correctly Recognized	Incorrectly Recognized	Access Denied
User1	8	2	0
User2	9	1	0
User3	9	0	1
User4	7	2	1
Total	33	5	2

Table 6.1 Result of different users

Considering the access denied cases as also correct results because it might be due to noise in the surroundings of the user, we can compute efficiency as:

$$\text{EFFICIENCY} = \frac{\text{(Total Samples – (Incorrectly Recognized+ Access Denied))}}{\text{(Total Samples)}}$$

Therefore efficiency achieved is **82.5%**

6.3 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 phonemes 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 phonemes constituting the word tend to average out the difference. 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.

CHAPTER 7

APPLICATIONS AND FUTURE SCOPE

7.1 Applications

- The main application of speaker recognition is in security systems to identify a person. Thus access will granted only to the person who has permission granted by the administrator. The person's speech samples must first be included in the database by the administrator.
- Speaker recognition for access control can be extended to wide variety of applications ranging from voice dialing, banking by telephone, telephone shopping, database access services, Information services, voice mail, security control for confidential information areas, and remote access to computers.
- Recognition of speech can also be extended to speech to text converters. A widely used application of the recognition of spoken words is the voice tags application found in most new mobiles.

7.2 Future Scope

- 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.
- 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 i.e. 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 may be some other optimizations that we can apply before finally storing it in the available memory.

CHAPTER 8

CONCLUSION

We have thereby, effectively implemented speaker identification and speaker verification using MATLAB & DSK TMS320C6713 kit. For simulation purpose we have implemented this project in MATLAB and for real-time purpose we have used TMS320C6713 DSK. This speaker recognition module performs accurately for speech dependent system.

The results acquired by our system confirm that the use of Fourier Transform with MFCC parameterization is a very promising method in the Automatic Speaker Recognition field. For real time processing of Speech signal, fast processors like Digital Signal Processors are required. Therefore the TMS320C Digital Signal Processors provide an excellent platform for the development of speaker recognition modules due to involvement of complex Fourier analysis in their algorithm.

Feature extraction is the most relevant portion of speaker recognition.Mel-frequency cepstrum coefficients (MFCC) is the best known and very popular algorithm.Cepstral coefficients are said to be accurate in certain pattern recognition problems relating tohuman voice.Also, it is a perfect representation for sounds when the source characteristics are stable and consistent (music and speech). MFCC features are not exactly accurate inthe existence of background noise and might not be well suited for generalization. Due to that efficiency of this system tend to reduce when it comes to real- time scenario.

Even though much care is taken it is difficult to obtain an efficient speaker recognition system since this task has been challenged by the highly variant input speech signals. The principle source of this 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 speaker recognition technology. Because of all these difficulties this technology is still an active area of research.

REFERENCES

1. Steven W. Smith, Ph.D., ”The Scientist and Engineer's Guide to Digital Signal Processing”, http://www.dspguide.com/whatdsp.htm

2. Thomas Farley and Ken Schmidt , Privateline Telecomunication Expertise, http://www.privateline.com/PCS/history9.htm

3. Processor Comparison: TI C6000 DSP and Motorola G4 PowerPC , Pentek Inc. , www.pentek.com/deliver/TechDoc.cfm/ProcComp.pdf?Filename=ProcComp.pdf

4. TI Training Brochure 2005 (Rev. C) , SPRT294C , “ TI Education Events “ , 2005 , focus.ti.com.cn/cn/lit/an/sprt294c/sprt294c.pdf

5. TMS320C6713DSK, Technical Reference, 2003, Printed in 2003

6. TMS320C6713B Floating-Point Digital Signal Processor (Rev. B), SPRT294B, Revised June 2006 , focus.ti.com/lit/ds/symlink/tms320c6713b.pdf

7. DSP Starter Kit (DSK) for the TMS320C6713, Quick Start Installation Guide, 506206-4001B , www.spectrumdigital.com

8. Lawrence Rabiner and Biing-Hwang Juang, “Fundamental of Speech Recognition”, Prentice-Hall, Englewood Cliffs, N.J., 1993.

9. Zhong-Xuan, Yuan & Bo-Ling, Xu & Chong-Zhi, Yu. (1999). “Binary Quantization of Feature Vectors for Robust Text-Independent Speaker Identification” in IEEE Transactions on Speech and Audio Processing, Vol. 7, No. 1, January 1999. IEEE, New York, NY, U.S.A.

10. Speech recognition using DSP ,www.vgyan.com/seminar/download/

11. //mfcc processor Jr., J. D., Hansen, J., and Proakis, J. Discrete-Time Processing of Speech Signals, second ed. IEEE Press, New York, 2000

14. Press, William H. [et. al.], "Power Spectrum Estimation Using the FFT", sec. 13.4, Numerical Recipes in C, 2nd ed., Cambridge University Press, 1992.

15. Md. Rashidul Hasan, Mustafa Jamil, Md. Golam Rabbani Md. Saifur Rahman , 3rd International Conference on Electrical & Computer Engineering ICECE 2004, 28-30December2004,Dhaka,Bangladesh www.buet.ac.bd/eee/icece2004/P140.pdf

16. F. Soong, E. Rosenberg, B. Juang, and L. Rabiner, "A Vector Quantization Approach to Speaker Recognition", AT&T Technical Journal, vol. 66, March/April 1987.

APPENDIX

1.audio loopback program:

```
/*
 * main.c#include "c6713dskinitmic.h"
 */
//#include "DSK6713_loopcfg.h"
#include"dsk6713.h"
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_MIC;
#include"dsk6713_aic23.h"
//#include "dsk6713initmic.h"
#include"stdlib.h"
#include"math.h"
#include"C:\C6xCSL\include\csl.h"
int i;

// Codec configuration settings
DSK6713_AIC23_Config config = {
   0x0017,  /* 0 DSK6713_AIC23_LEFTINVOL  Left line input channel volume */\
   0x0017,  /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume */\
   0x01f9,  /* 2 DSK6713_AIC23_LEFTHPVOL  Left channel headphone volume */  \
   0x01f9,  /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume */ \
   0x0011,  /* 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 */ \
   0x0001,  /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */            \
   0x0001   /* 9 DSK6713_AIC23_DIGACT     Digital interface activation */   \
};

voidmain()
{
    DSK6713_AIC23_CodecHandle hCodec;
    Int16 OUT_L, OUT_R;
    Uint32 IN_L;

    // Initialize BSL
    DSK6713_init();

    //Start codec
    hCodec = DSK6713_AIC23_openCodec(1, &config);

    // Set  frequency to 48KHz
```

```
    DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_48KHZ);

    for(;;)
    {
            for(i=0;i<1000;i++)
            {

            // Read sample from the left channel
            while (!DSK6713_AIC23_read(hCodec, &IN_L));

            }

            // Feeding the input directly to output you can add effects here
            OUT_L = IN_L*100;
            OUT_R = IN_L*100;

            // Send sample, first left next right channel
    while (!DSK6713_AIC23_write(hCodec, OUT_L));
    while (!DSK6713_AIC23_write(hCodec, OUT_R));
    }

    //DSK6713_AIC23_closeCodec(hCodec); // Codec close is unreachable
}
```

2. matlab code:

- Training phase:

```
fs = 8000;              % 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)];
 z = audiorecorder(fs, 16 , 1); %Store the uttered password for authentication
disp('Start speaking.')
recordblocking(z,1);
%disp('End of Recording.');
a = getaudiodata(z);
% Plot the waveform.
%plot(a);
%title('speech signal');
%     xlabel('amplitude');
%   ylabel('time');
%play(z);
i = 1;

% while (x(i)) < 0.05                    % Silence detection
%    i = i + 1;
%end
%x(1 : i) = [];
%x(6000 : 10000) = 0;
 x1 = a .* hamming(length(a));
 mx = abs(fft(a));                      % 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('a.dat', 'w');
fwrite(fid, dx, 'real*8'); % Store this feature vector as a .dat file fclose(fid);
fclose(fid);
```

- Testing phase

```
fs = 8000;              % 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 = audiorecorder(fs, 16 , 1); %Store the uttered password for authentication
disp('Start speaking.')
recordblocking(y,1);
%disp('End of Recording.');
b = getaudiodata(y);
% Plot the waveform.
plot(b);
title('speech signal');
     xlabel('amplitude');
    ylabel('time');
play(y);
i = 1;
%while abs(y(i)) < 0.05            % Silence Detection
%   i = i + 1;
%end
%y(1 : i) = [];
%y(6000 : 10000) = 0;
y1 = b.* hamming(length(b));
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('a.dat','r');
dx = fread(fid, 20, 'real*8'); % Obtain the feature vector for the password
[fn,path]=uigetfile('a.dat','Select file to process');
%dx=importdata(fullfile(path,fn));
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');  % “Access Granted” is output if within threshold
else
fprintf('\n\nACCESS DENIED\n\n');  % “Access Denied” is output in case of a failure
end
```

3. C code for CCS

- 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")

typedefstruct {floatreal, imag;}
COMPLEX;
voidFFT_init(float *);// Declaration of functions for calculating FFT
voidFFT(COMPLEX *, int );
intmaxim(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,z,f,l;
    float   d[3][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 */   \

    };

voidmain()
    {
     for(z=0;z<3;z++)
      {
            printf("\nstart\n");
     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 < 25000 ; 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*50;
             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 */
//   printf("\nstop");
     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]*50;
             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[z][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[z][11] = maxim(spec, 256);
      //fptr = fopen("train.dat","w");
      // for ( i =0; i < 12; i++)
      //fprintf(fptr, "%f, ",d[i]);
      //fclose(fptr);
      printf("Thank you\n");
                }


}
/* Function to evaluate the FFT */
voidFFT_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;
 }
```

```
voidFFT(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 */
intmaxim(float *a, int length)
{   float pos;
    float temp;
     temp = a[1];

    for(i = 2 ; i < length / 2 ; i++)
     {
            if(temp < a[i])
             {
                    temp = a[i];
        pos=10*log(temp);
             }
     }
    return pos;
}
```

- 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")
typedefstruct {floatreal, imag;} COMPLEX;

voidFFT_init(float *);// Declaration of functions for calculating FFT
voidFFT(COMPLEX *, int );
intmaxim(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,flag=0,num1=0,num2=0;
    float   d[12]={0};
    float  aa[3][12]={0};
float *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 */   \
};

voidmain()
{
    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 < 25000 ; 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*50;
            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]*50;
      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 = (float *)0x0000F954 ; /* 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)
 {

  Printf("\n access granted\n")

}

 Else
  {
       Printf("\n access denied \n")
```

```
 }


/* Function to evaluate the FFT */
voidFFT_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;
}

voidFFT(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 */
intmaxim(float *a, int length)
{   float pos;
    float temp;
    temp = a[1];
    pos = 1;
    for(i = 2 ; i < length / 2 ; i++)
    {
            if(temp < a[i])
            {
                    temp = a[i];
                    pos=10*log(temp);

            }
    }
    return pos;

}
```