# Electronics Club Summer Project - 2009

## DTMF Decoder

*By*

*DCODRS*

Team Members

Ankit Agrawal
Himanshu Singh
K.Venkata
*Mentor*-Anubhav Singla

# Acknowledgements

We would like to thank our mentor, Anubhav Singla, without whose help and guidance even starting our project would have been very difficult. Our topic involved a fair amount of theoretical background, and it wouldn’t have been possible without this help.
Special thanks to Arpit Mathur, Siddharth Garg and Ankit Gupta, whom we disturbed once in a while, both online and offline.

# Contents

# Introduction

DTMF stands for Dual Tone Multi Frequency.It is used in cell phones, landline phones etc. to identify the key pressed.
Corresponding to every row and column of our keypad, there is a frequency associated with it. When a key is pressed, a signal is sent, which is the superposition of sinusoids of the 2 frequencies associated with that key. This signal when decoded, gives us the key pressed.
The following are the frequencies used for the DTMF (dual-tone, multi-frequency) system, which is also referred to as tone dialing. The signal is encoded as a pair of sinusoidal (sine wave) tones from the table below which are mixed with each other.

| Symbol | | Tone B [Hz] | | | |
|---|---|---|---|---|---|
| | | 1209 | 1336 | 1477 | 1633 |
| Tone A [Hz] | 697 | 1 | 2 | 3 | A |
| | 770 | 4 | 5 | 6 | B |
| | 852 | 7 | 8 | 9 | C |
| | 941 | * | 0 | # | D |

In commercial DTMF decoders, analog filters are used. However, for our project, we have done it digitally on an MCU.

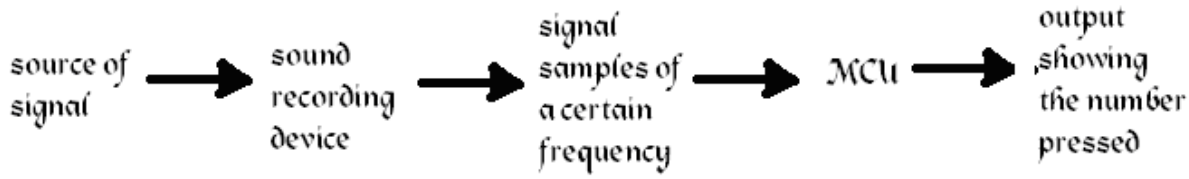
# Project Motivation

We were looking for a project which involved more of coding and less of hardware. Also, we wanted something which we could involve some new concept, something which we would have to learn. Our mentor

suggested this idea, and it fit in quite well, as far as our interests were concerned.

## Outline of project

The basic flowchart for our project is as follows:




In our case, the source of signal was a cell phone. We tried using a microphone. However, since it wasn't giving favorable results, we had to use a laptop for this purpose. We recorded the signal using Matlab, which then gave us the samples of required frequency. From this, we transferred the data to the MCU, which performed certain calculations and displayed the result on the LCD screen.

## Theory

Every signal can be written as a Fourier series i.e. as a sum of sines and cosines. Our signal is basically a superposition of 2 sinusoids of different frequency, plus the noise.
So, our signal $S=A\,(e^{iwt}+e^{-iwt}) + B\,(e^{iut}+e^{-iut})$ + noise.
Noise itself is composed of sinusoids, of much lesser frequency though.
Now suppose we perform the following integral:
$I=\int S^{*}eixt\,dt$. If $x=\pm w, u$; then, the integral can be as large as possible, provided we choose a sufficiently large time period.
However, in all other cases, our integral is bounded.

So this is the main concept involved. It's inspired from Fourier Transforms.
We carry out this integral for different values of x, and choosing an appropriate time period, we can get to know the frequencies present in the signal.

## Hardware implementations

Not much of hardware was involved in our project. We only had to connect the MCU to the LCD screen for output.

## Software implementations

Our task at hand is to perform the given integral.
Integral leads to summation, and it turns out that this method is good enough even after this approximation.
Had it been the usual High level programming languages, it would have been fairly easy, given that we have practically no limit on precision of decimals, size of numbers etc.
However, MCU does have its limitations.
One major problem is that of handling large numbers, decimal numbers etc.
Secondly, speed and restricted memory are also other small issues.
To some extent, we did solve these problems.

## Code

So we are given the signal in the form of a char array A [] (char because of memory restrictions).
We need to perform our operations on this array.
Now comes the summation part:
$a=\sum A[i]*\cos(2*pi*x*t)\ \Delta t.$ $b=\sum A[i]*\sin(2*pi*x*t)\ \Delta t.$
t=i*sampling interval.
The maximum values of $a^2+b^2$ gives us the 2 frequencies.

The only hitch in this entire thing is the presence of large integers and decimal numbers. To avoid loss of precision, we multiply everything with 1000.
So now we are left with only one problem: large integers.
For this, we use a kind of data structure.
Consider a 9 digit number a. It can be considered to be composed of 3 3 digit numbers a1 a2 a3.989343255 is composed of a1=989 a2=343 a3=255. Using this, we can store 9 digit numbers. We only need to implement addition, subtraction and multiplication on this, which is fairly easy, just the way we do it manually, using carry over etc.
One thing that needs to be made sure is that a2<1000 and a3<1000. Suppose at any point, say a3 exceeds 1000. Then, as we do manually, a2=a3/1000 and a3=a3%1000.
One minor issue: In order to avoid the hassles of negative numbers, we initially set a1=999 a2=999 and a3=999.

## Problems faced

One major problem faced was that with the microphone. It is still not fixed, and as a result, we had to use our laptop for this purpose.
Secondly, as mentioned before, we had problems with large numbers and decimal points, which were handled successfully.
Finally, the speed problem. Initially, the code was running very slow. However, this was later fixed by changing the MCU clock frequency to 8 MHz.

## Future scope

We could make this a complete stand alone device if we could fix the microphone problem. Moreover, this could be used for password checks etc.

# Conclusion

It was a great learning experience. This was the first time any of us had used MCU, worked with Fouriers etc. We also tried a few other approaches like using digital filters which gave us a good exposure to the field of signal processing.