An Implementation of Real Time DTMF Signal Decoder

Ayan Shafqat

Abstract – This document explains the use of Goertzel's algorithm in order to decode DTMF signal very efficiently in real time. Also, discusses the importance of noise gate based DFT triggering for efficient utilization of embedded processors.

EL6183: Semester Project
Polytechnic Institute of New York University
Spring 2011
6 Metrotech Center
Brooklyn, NY-11201

Introduction

Dual Tone Multiple Frequency or DTMF scheme was invented by Bell Labs in the 1960's for replacing rotatory telephone dials. Rotary telephones needed more wait time since it used more pulses as digits got higher, costing telephone companies more money. Therefore, by using a couple of base frequencies, DTMF can be used to signal multiple signals for a short time. Although this system was used for telephone switching system, it is rarely used now due to new digital protocols and VOIP. But, DTMF is still used in menu driven telephone operators for many companies and government agencies.

DTMF, as the name implies, uses two tones per symbol. The symbols are composed of the digits 0 to 9, and some special characters such as # (called a "hash") and * (called a "star"), and the letters A to D. The letter symbols were dropped from a standard telephone set, but are still used for military signaling for encrypted/private telephone calls. Each symbol of a DTMF scheme uses two sinusoid frequencies. The lower frequency is called the low band, and the higher frequency is usually called the high band. By correlating the two bands, DTMF signals can be decoded to which symbol was sent. The frequencies of a DTMF system scheme are shown in the table below.

DTMF Symbols and Corresponding Frequencies

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

The way to interpret this table is to consider the DTMF signal as the sum of the low band frequency and the high band frequency. For example, the signal for the symbol * (denoted as $x_*(t)$) is shown below:

$$x_*(t) = \frac{1}{2}(\sin(2\pi(941\text{ Hz})t) + \sin(2\pi(1209\text{ Hz})t))$$

Besides having the frequency tables for symbols, DTMF standard implements frequencies for special events consisting of dial tone, busy signal, and ring back tone. It is not standardized globally, but each country uses their own tones to describe these events. The application of these tones are a cell phone hanging up after call is finished, or automatic callers calling back when the line is not busy. The frequencies of these special events are shown in the table below.

Event	**Low Band**	**High Band**
Busy (US)	480 Hz	620 Hz
Ring Back (US)	440 Hz	480 Hz
Dial Tone (US)	350 Hz	440 Hz

Geortzel Algorithm For Calculating DFT

Geortzel's algorithm is very useful for calculating DFT in real time, especially for embedded platforms. FFT is one of the most popular forms of calculating Discrete Fourier Transform, but the process is non-causal. In order to calculate a N point FFT, all N points have to gathered before executing FFT algorithm. The disadvantage of this is being the fact that between each samples, there is a small window of time available for the processor to calculate the value (for a sample rate of 8 kHz, we have 125µS time window). Also, another advantage of using Geortzel's algorithm over FFT is that, we don't have to calculate all the frequency bins. We can just calculate the bins we are interested in, such as the DTMF frequencies.

Goertzel algorithm computes a sequence, $s(n)$, given an input sequence, $x(n)$:

$$s(n) = x(n) + 2cos(\omega)s(n-1) - s(n-2)$$

Where $s(-2) = s(-1) = 0$ and ω is some frequency of interest, which should be less than $1/2$. This effectively implements a second-order Infinite Impulse Response Filter with poles at $e^{+j\omega}$ and $e^{-j\omega}$, and requires only one multiplication with a constant c, which is 2cos(ω). This then requires one addition and one subtraction per input sample, thus limiting computational complexity. Furthermore, for a given real signal, these operations are real as well.

The Z Transform of Geortzel's algorithm is:

$$\frac{S(z)}{X(z)} = \frac{1}{1 - 2\cos(\omega)} = \frac{1}{(1 - \mathrm{e}^{+j\omega}z^{-1})(1 - \mathrm{e}^{-j\omega}z^{-1})}$$

Applying an additional, FIR, transform of the form

$$\frac{Y(z)}{S(z)} = 1 - \mathrm{e}^{-j\omega}z^{-1}$$

This will give a transfer function of

$$H(z) = \frac{Y(z)}{X(z)} = \frac{S(z)Y(z)}{X(z)S(z)} = \frac{1}{1 - \mathrm{e}^{j\omega}z^{-1}}$$

When this signal is transformed in the time domain, it becomes:

$$y(n) = x(n) + y(n-1)\mathrm{e}^{+j\omega} = \sum_{k=-\infty}^{n} x(k)\mathrm{e}^{j\omega(n-k)} = e^{+j\omega n}\sum_{k=-\infty}^{n} x(k)\mathrm{e}^{-j\omega k}$$

And for a causal system, this equation becomes:

$$y(n) = e^{j\omega n}\sum_{k=0}^{n} x(k)\mathrm{e}^{-j\omega k}$$

Revising back to the formula for a DFT, the above equation actually is:

$$X(k) = \sum_{n=0}^{N-1} x(n)\mathrm{e}^{-j\frac{(2\pi k)}{N}n}$$

$$X(\omega) = \sum_{n=0}^{N-1} x(n)\mathrm{e}^{-j\omega n} = y(N-1)\mathrm{e}^{-j\omega(N-1)}$$

Then the Fourier Transform can be written as:

$$X(\omega) = y(N-1)\mathrm{e}^{-j\omega(N-1)} = \left[s(N-1) - s(N-2)\,\mathrm{e}^{-j\omega}\right]\mathrm{e}^{-j\omega(N-1)}$$

When $\omega N = k$, for some integer, k, where k is in an integer representing a DFT bin:

$$X(\omega) = \left[s(N-1) - s(N-2)e^{-j\omega}\right]e^{j\omega} = s(N-1)e^{j\omega} - s(N-2)$$

If we are only interested at the magnitude, the power of the DFT at k^{th} bin can found by:

$$X(\omega)X^*(\omega) = s^2(N-2) + s^2(N-1) - 2cos(\omega)s(N-2)s(N-1)$$

Summary of the algorithm for calculating DFT

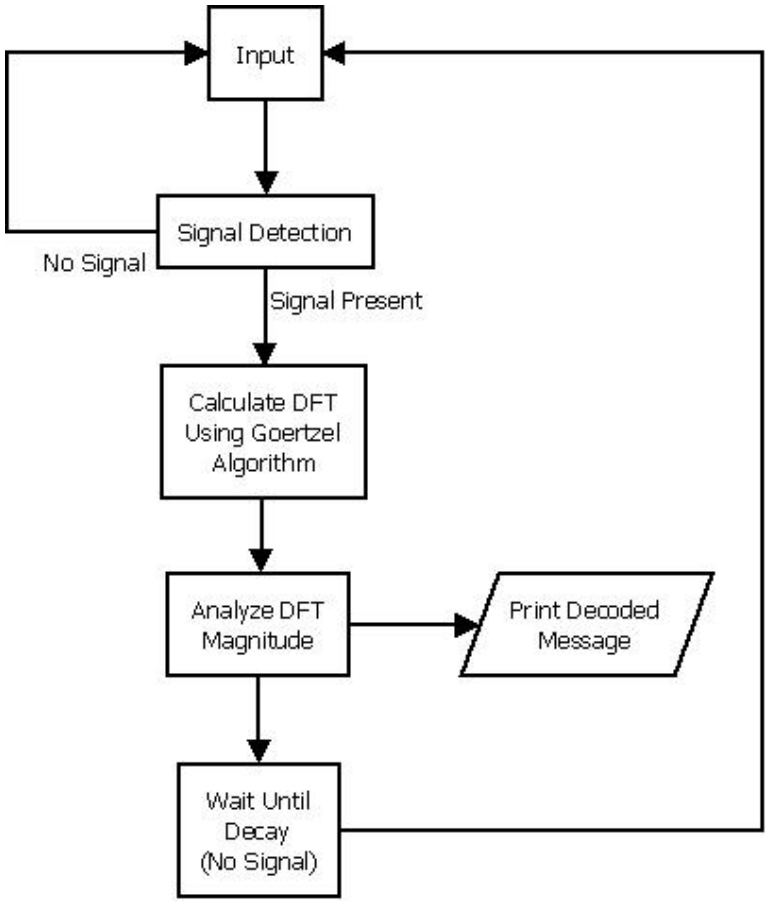
```
S_k[-1] = 0
S_k[-2] = 0
f_N = F/F_s
c_k = 2*cos(2*π*f_N);
for each sample, x[n],
  s_k[n] = x[n] + c*s_k[n - 1] - s_k[n - 2];
end
power = s_k^2[n - 2] + s_k^2[n - 1] - c*s_k[n - 1]*s_k[n - 2];
```

Algorithm (DTMF Decoder)

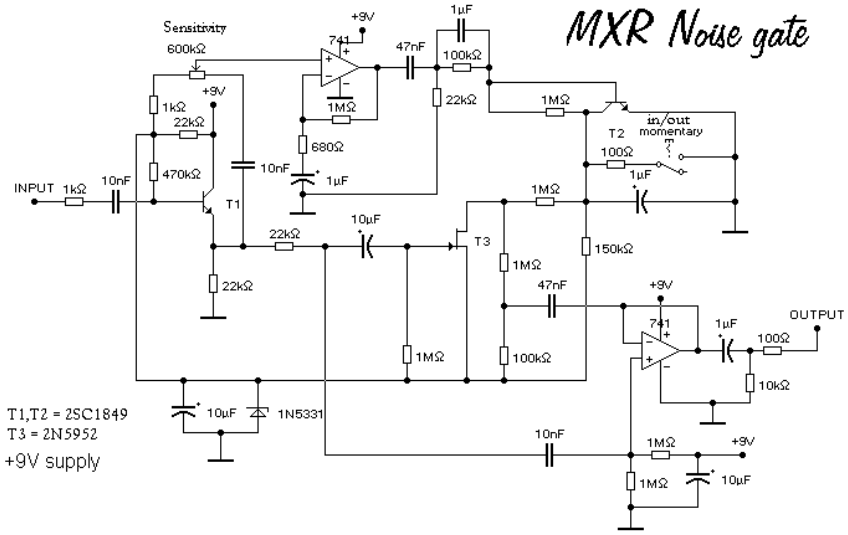
Decoding DTMF signals has to do with more than just calculating the DFT of the signal and analyze them. First of all, a user may push a button at different rates from others, and telephone modems can dial a phone number a lot faster than a user can dial (approximately 20-50mS per symbol). There are no fixed rates for how many symbols can be pressed at a certain window of time. Thus, calculating DFT of the signal every N samples would be pointless to decode a DTMF message. Also, realizing when a user has done pressing the button is also crucial, so the decoder can get ready for decoding the next symbol. Therefore, a trigger mechanism will be used to detect when a signal is present, which will trigger on the analysis algorithm. The analysis algorithm will compute the DFT of the signal and look for magnitude peaks at frequency bins. Since there are two tones per symbol, we only need to consider two peaks. And from those bin indexes, the decoder can make a decision on what symbol was actually pressed. After the symbol has been declared, the trigger mechanism has to wait until there are no signals present. The reason behind this due to the fact that tone from a symbol still might be present after calculating DFT and analysis.

For the triggering mechanism, I have used a simple noise gate using a rectifier (absolute value function) with a 1st order low pass filter. Through experimenting, I have found having a time constant around 4mS was suitable for triggering without decaying very slowly. There has to be a slight balance between having as less rectified ripples as possible, and decay time. That is where I had to make the most compromise. Having a higher time constant would allow me to get a clean trigger signal, but the noise gate would attack very slowly, almost missing an entire symbol tone.

In order to understand the algorithm better, a flowchart representation of the algorithm is indicated below:



Also, I looked at an analog noise gate schematic to understand how to implement noise gate for DTMF decoder. The schematic below is a noise gate used for removing pickup noise from electric guitars. Note the rectifier at lower left corner, comparator on upper middle, and trigger on lower right.



The code below shows the code for DTMF decoder.
(For full source, look into the source folder)

```
/*****************************************************************************
/* DTMF Decoder                                                              *
/*****************************************************************************/
void DTMF_Decode(dtmf_dft G, noise_gate gate, short x) {
	// Calculate DFT as reading samples
	int lowB, hiB;
	if(NoiseGateTriggerOn(gate, x)) {
		if(G->counter < G->N) {
			Goertzel_DTMF_DFT_Calc(G, x);
		}
		else {
			Find_Peak(G, &lowB, &hiB);
			if(G->print_flag) {
				#ifdef DTMF_DEBUG
					printf("Successfuly Decoded: ");
				#endif
					 if(lowB == 7 && hiB == 9 ) printf("0");
				else if(lowB == 4 && hiB == 8 ) printf("1");
				else if(lowB == 4 && hiB == 9 ) printf("2");
				else if(lowB == 4 && hiB == 10) printf("3");
				else if(lowB == 5 && hiB == 8 ) printf("4");
				else if(lowB == 5 && hiB == 9 ) printf("5");
				else if(lowB == 5 && hiB == 10) printf("6");
				else if(lowB == 6 && hiB == 8 ) printf("7");
				else if(lowB == 6 && hiB == 9 ) printf("8");
				else if(lowB == 6 && hiB == 10) printf("9");
				else if(lowB == 4 && hiB == 11) printf("A");
				else if(lowB == 5 && hiB == 11) printf("B");
				else if(lowB == 6 && hiB == 11) printf("C");
				else if(lowB == 7 && hiB == 11) printf("D");
				else if(lowB == 0 && hiB == 1 ) printf("T");
				else if(lowB == 1 && hiB == 2 ) printf("R");
				else if(lowB == 2 && hiB == 3 ) printf("U");
				else                            printf(" ");
				#ifdef DTMF_DEBUG
					printf("\n");
				#endif
				G->print_flag = 0;
			}
		}
	} else {
		Reset_DTMF_DFT(G);
		NoiseGateReset(gate);
	}
	return;
}
```

Implementation of Goertzel's Algorithm for DTMF Signal Decoding

We know, we don't have to calculate the DFT for all the frequency bins. Therefore, we must list the frequencies that are being used by DTMF protocol. The following table used N = 250, and F_s = 8000. Note the frequency bins must be an integer.

$$k = \frac{f}{f_s} N$$

k has to be an integer, therefore we use the nearest-integer-round function.

$$\hat{k} = \lfloor k \rceil$$

The filter coefficient is then calculated by:

$$c_k = 2\cos\left(\frac{2\pi\hat{k}}{N}\right)$$

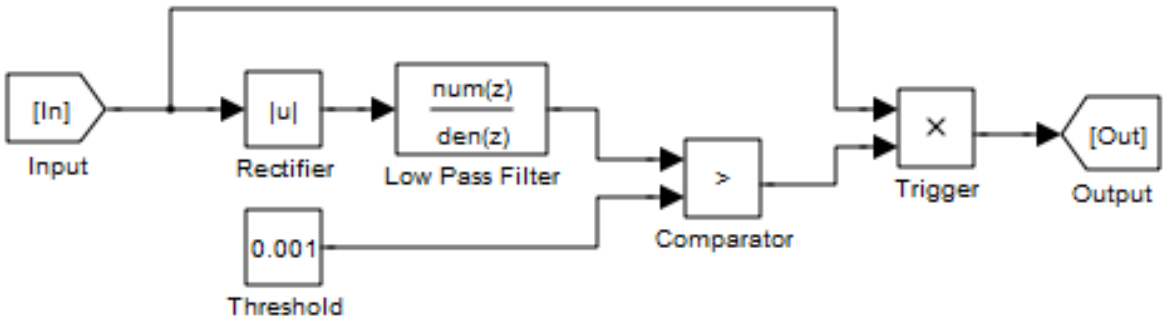
The following table shows the Goertzel's coefficients used in this project.

Frequency Index	DTMF Frequency (Hz)	DFT Bin (k)	$\hat{k}$	Ck
0	350	10.938	11	1.9241
1	440	13.750	14	1.8775
2	480	15.000	15	1.8596
3	620	19.375	19	1.7763
4	697	21.781	22	1.7020
5	770	24.063	24	1.6471
6	852	26.625	27	1.5569
7	941	29.406	29	1.4919
8	1209	37.781	38	1.1551
9	1336	41.750	42	0.9855
10	1477	46.156	46	0.8058
11	1633	51.031	51	0.5700

The code below shows the implementation of Geortzel's Algorithm for DFT

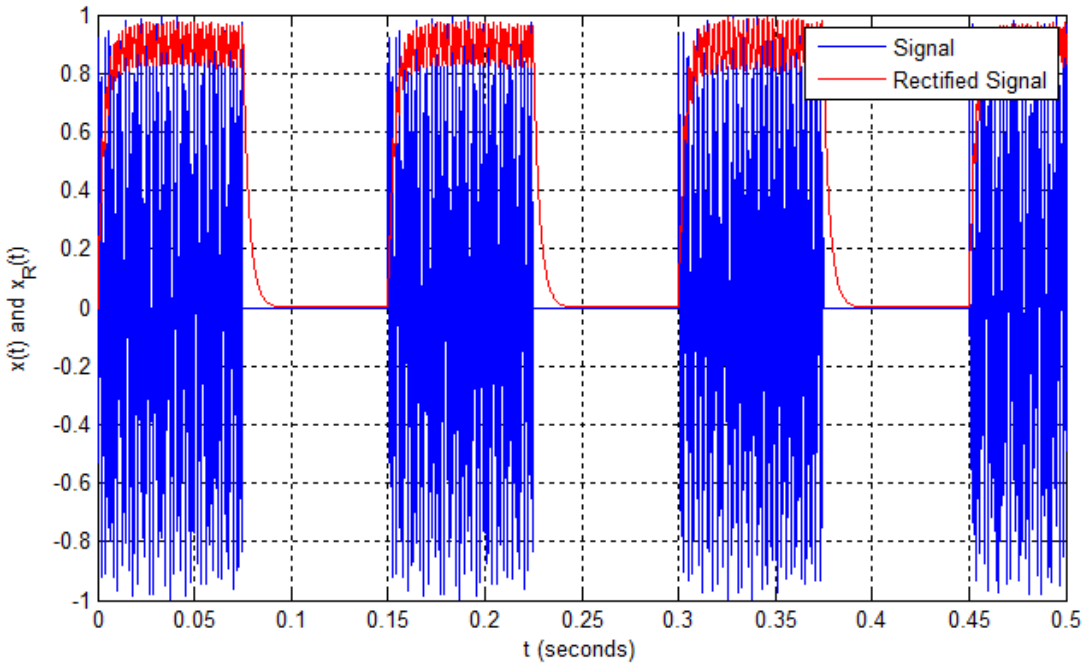
```
/*****************************************************************************
/* Geortzel Algorithm DFT                                                    *
/*****************************************************************************/
#define DFT_LENGTH 400
typedef struct {
        float* c; unsigned int counter; char print_flag; unsigned int dft_done; unsigned int N;
        float  s[NUM_DTMF_FREQ], s1[NUM_DTMF_FREQ], s2[NUM_DTMF_FREQ];
} DTMF_DFT;
typedef DTMF_DFT* dtmf_dft;
float* Goertzel_Coef(unsigned short N) {
        /* Calculates Goertzel coefficient */
        short n;
        static float C[NUM_DTMF_FREQ]; unsigned int k;
        #ifdef DTMF_DEBUG
                printf("Goertzel Coefficients\n");
                printf("Frequency\tk (bin)\tCoefficient\n");
                printf("-----------------------------------\n");
        #endif
        for(n = 0; n < NUM_DTMF_FREQ; n++) {
                k = round((N*FREQ_TABLE[n])/((float)DTMF_SAMPLE_RATE));
                C[n] = 2.0 * cos((TWOPI*(float)k)/(float)N);
                #ifdef DTMF_DEBUG
                printf("%5.2f Hz\t %d\t%f\n",FREQ_TABLE[n], k, C[n]);
                #endif
        }
        return C;
}
dtmf_dft Goertzel_DTMF_DFT_Init(unsigned int N) {
        int n;
        static DTMF_DFT newDFT;
        newDFT.N = N;
        newDFT.c = Goertzel_Coef(N);
        for(n = 0; n < NUM_DTMF_FREQ; n++) {
                newDFT.s[n] = 0; newDFT.s1[n] = 0; newDFT.s2[n] = 0;
        }
        newDFT.counter    = 0; newDFT.print_flag = 1;
        return (&newDFT);
}
void Goertzel_DTMF_DFT_Calc(dtmf_dft G, short x){
        unsigned int n;  if(G->counter < G->N) {
                for(n = 0; n < NUM_DTMF_FREQ; n++){
                        (G->s[n])  = Fix162Float(x) + (G->c[n])*(G->s1[n]) - (G->s2[n]);
                        (G->s2[n]) = (G->s1[n]); (G->s1[n]) = (G->s[n] );
                }
        } G->counter++;
}
void Reset_DTMF_DFT(dtmf_dft G) {
        int n; for(n = 0; n < NUM_DTMF_FREQ; n++) { G->s[n] = 0; G->s1[n] = 0; G->s2[n] = 0;}
        G->counter = 0; G->print_flag = 1;
}
void Find_Peak(dtmf_dft G, int *lowB, int *hiB) {
        int n; float mag[NUM_DTMF_FREQ], max;
        unsigned int   index = 0;
        for(n = 0; n < NUM_DTMF_FREQ; n++) {
                mag[n] = (G->s2[n]*G->s2[n]) + (G->s1[n]*G->s1[n]) - (G->c[n]*G->s1[n]*G->s2[n]);
                if(mag[n] > max) { max = mag[n]; index = n; }
        }
        (*lowB) = index; mag[index] = 0; index = 0; max = 0;
        for(n = 0; n < NUM_DTMF_FREQ; n++) {
                if(mag[n] > max) { max = mag[n]; index = n; }
        }
        (*hiB) = index;
        if( (*hiB) < (*lowB) ) {
                index   = (*lowB); (*lowB) = (*hiB); (*hiB)  = index;
        }
        return;
}
```

Noise Gate Algorithm for Triggering DFT.



MATLAB Code for testing different parameters for filter coefficient:

```
filename = 'DTMF_TEST.WAV';
f_cut = 50;
[x fs] = wavread(filename);
l = length(x); h = 1/fs;
RC = 1/(2*pi*f_cut); a = h/(h+RC); % a = Filter Coefficient
y = zeros(l,1); y(1) = x(1);
for t = 2:l,
   y(t) = a*abs(x(t)) + (1-a)*y(t-1);
end;
t = (0:(length(x) -1 )) / fs;
plot(t, x, 'b-', t, y./max(y), 'r-');
grid on; axis([0 0.5 -1 1])
```



Code below shows implementation of noise gate

```
/*****************************************************************************
/* Noise Gate for Signal Detection                                           *
/*****************************************************************************/
typedef struct {
	short		threshold; // Signal threshold
	short		k, k2;	   // Filter Coefficients
	short		y_prev;    // Previous values
} NOISE_GATE;
typedef NOISE_GATE* noise_gate;
noise_gate NoiseGateInit(void) {
	static NOISE_GATE gate;
	unsigned int n;
	float a = 0.037786053325671L;
	// Converting float to fix16_t
	gate.k  = Float2Fix16(a);
	gate.k2 = Float2Fix16(1.0 - a);
	gate.y_prev = 0;
	gate.threshold = Float2Fix16(0.02);
	return (&gate);
}
short NoiseGateTriggerOn(noise_gate gate, short input) {
	int x, output; x = abs(input);
	output = FixMult(gate->k, x) + FixMult(gate->k2, gate->y_prev);
	gate->y_prev = output;
	if(output > gate->threshold) {
		#ifdef DTMF_DEBUG
			printf("Signal is ABOVE treshold (%f, %f)\n",
					Fix162Float(output), Fix162Float(gate->threshold));
			usleep(50000);
		#endif
		return 1;
	} else {
		#ifdef DTMF_DEBUG
			printf("Signal is below treshold (%f, %f)\n",
					Fix162Float(output), Fix162Float(gate->threshold));
			usleep(50000);
		#endif
		return 0;
	}
}
short NoiseGateTriggerOff(noise_gate gate, short input) {
	int x, y1, y2, output; // Rectified input variable
	// Rectifying
	x = abs(input);
	output = FixMult(gate->k, x) + FixMult(gate->k2, gate->y_prev);
	gate->y_prev = output;
	if(output <= gate->threshold) {
		return 1;
	}
	else return 0;
}
void NoiseGateReset(noise_gate gate) {
	gate->y_prev = 0;
}
```

Results

Since I didn't have a constant access to the DSK board, I had to write the program for both DSK and PC. The algorithm ran on the DSK board, without any modification whatsoever. For accomplishing this project, I have used libsndfile (http://www.mega-nerd.com/libsndfile/) and Sound eXchange plugin (SoX: http://sox.sourceforge.net/) to record DTMF tones and decoding them. I have also included a fully functional DTMF encoder in the demoExe folder as well as decoder.

How to use DTMF decoder application package:

The application package contains several application, (1) dtmf-enc: encodes a string of characters into DTMF message and saves it to DTMF_TEST.WAV file, (2) dtmf-enc-dec: encodes and decodes DTMF message internally in the memory, without using a wav file, (3) dtmf-enc-dec-snd: encodes a DTMF message, saves them in the wav file (DTMF_TEST.WAV) and plays back the audio and then decodes it, (4) dtmf-dec: Decodes a DTMF message from a given wav file (Note: the wave file needs to be at a sample rate of 8000Hz). It is very easy to use, no command line parameters required.

Outputs

DTMF Encoder

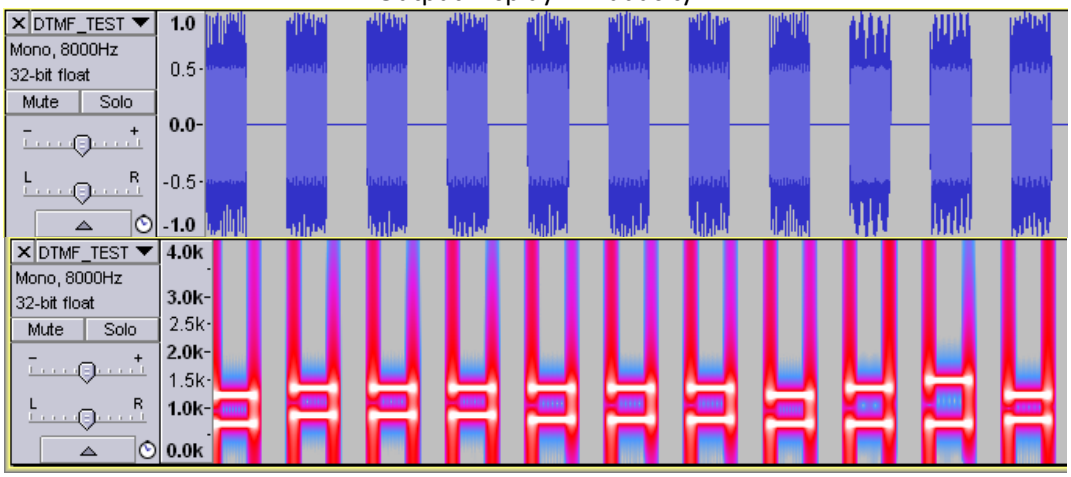
```
D:\EL6183 Final Project\DemoExe>dtmf-enc
DTMF Encoder
Author: Ayan Shafqat
(Tones: 0-9, A-D, U = Busy, R = Ringback, T = Dial Tone)
Enter DTMF Message (Up to 15 Symbols): 18885551234
Encoding message: 18885551234
DTMF Signal writing complete (DTMF_TEST.WAV)
--------------------------------------------------------------------------------
".\sox-14.3.2\play" DTMF_TEST.WAV

DTMF_TEST.WAV:

 File Size: 40.0k     Bit Rate: 128k
  Encoding: Signed PCM
  Channels: 1 @ 16-bit
Samplerate: 8000Hz
Replaygain: off
  Duration: 00:00:02.50

In:100%  00:00:02.50 [00:00:00.00] Out:20.0k [!=====|=====!]        Clip:0
Done.
--------------------------------------------------------------------------------
```

Output Display in Audacity



DTMF Decoder

```
D:\EL6183 Final Project\DemoExe>dtmf-dec DTMF_TEST.WAV
Decoding DTMF From file: DTMF_TEST.WAV
Decoded Message: 18885551234
Press any key to continue . . . _
```

DTMF Encoder and Decoder (No Sound)

```
D:\EL6183 Final Project\DemoExe>dtmf-enc-dec
DTMF Encoder
Author: Ayan Shafqat
(Tones: 0-9, A-D, U = Busy, R = Ringback, T = Dial Tone)
Enter DTMF Message (Up to 15 Symbols): 18005551234
Encoding message: 18005551234
DTMF signal generation complete
--------------------------------------------------------------------------------
DTMF Decoding
Decoded Message: 18005551234
Press any key to continue . . . _
```

DTMF Encoder and Decoder (With Sound)

```
D:\EL6183 Final Project\DemoExe>dtmf-enc-dec-snd
DTMF Encoder
Author: Ayan Shafqat
(Tones: 0-9, A-D, U = Busy, R = Ringback, T = Dial Tone)
Enter DTMF Message (Up to 15 Symbols): 18885551234
Encoding message: 18885551234
DTMF Signal writing complete (DTMF_TEST.WAV)
--------------------------------------------------------------------------------
".\sox-14.3.2\play" DTMF_TEST.WAV

DTMF_TEST.WAV:

 File Size: 40.0k     Bit Rate: 128k
  Encoding: Signed PCM
  Channels: 1 @ 16-bit
Samplerate: 8000Hz
Replaygain: off
  Duration: 00:00:02.50

In:100%  00:00:02.50 [00:00:00.00] Out:20.0k [!=====|=====!]        Clip:0
Done.
--------------------------------------------------------------------------------
DTMF Decoding (DTMF_TEST.WAV)
Decoded Message: 18885551234
Press any key to continue . . . _
```