CARLETON UNIVERSITY
Department of Systems and Computer Engineering
SYSC 4405
Lab #3: DTMF Decoding – Part I: Bandpass Filter Design

1 Introduction

This lab introduces a practical application where sinusoidal signals are used to transmit information: a touch-tone dialer. One approach to decode touch-tone sounds is to use bandpass filters to extract the information encoded in the waveforms. The goal of this lab is to design and implement bandpass filters in MATLAB, and then in the next lab to use these designed filters and implement them on the c6713 DSK to perform the touch-tone decoding. *Be sure to keep your filter designs and* MATLAB *code from this lab for the next lab.* In the experiments of this lab, you will use `filter()` to implement filters and `freqz()` to obtain the filter's frequency response. As a result, you should learn how to characterize a filter by knowing how it reacts to different frequency components in the input.
The lab itself starts with background information in Sec. 2 and deals with synthesizing and decoding *dual tone multi-frequency* signals used in dialing a touch tone telephone. You should read through the entire lab before your scheduled lab time so that you know what to expect and can move quickly through the lab manual. If you cannot complete all of the lab during the scheduled time, you should show your results to the TAs at the beginning of your next lab session.

2 Lab Background

2.1 Telephone Touch Tone[1] Dialing

Telephone touch pads generate *dual tone multi-frequency* (DTMF) signals to dial a telephone. When any key is pressed, the tones of the corresponding column and row (in Fig. 1) are generated and summed, hence the name dual tone. As an example, pressing the **5** key generates a signal containing the sum of two tones at 770 Hz and 1336 Hz.
The frequencies in Fig. 1 were chosen (by the design engineers) to avoid harmonics. No frequency is an integer multiple of another, the difference between any two frequencies does not equal any of the frequencies, and the sum of any two frequencies does not equal any of the frequencies. This makes it easier to detect exactly which tones are present in the dial signal in the presence of non-linear line distortions.

2.2 DTMF Decoding

There are several steps to decoding a DTMF signal:
1. Divide the signal into shorter time segments representing individual key presses.
2. Filter the individual segments to extract the possible frequency components. Bandpass filters can be used to isolate sinusoidal components.
3. Determine which two frequency components are present in each time segment by measuring the level of the output signal from all of the bandpass filters.

1. [1]Touch Tone is a registered trademark

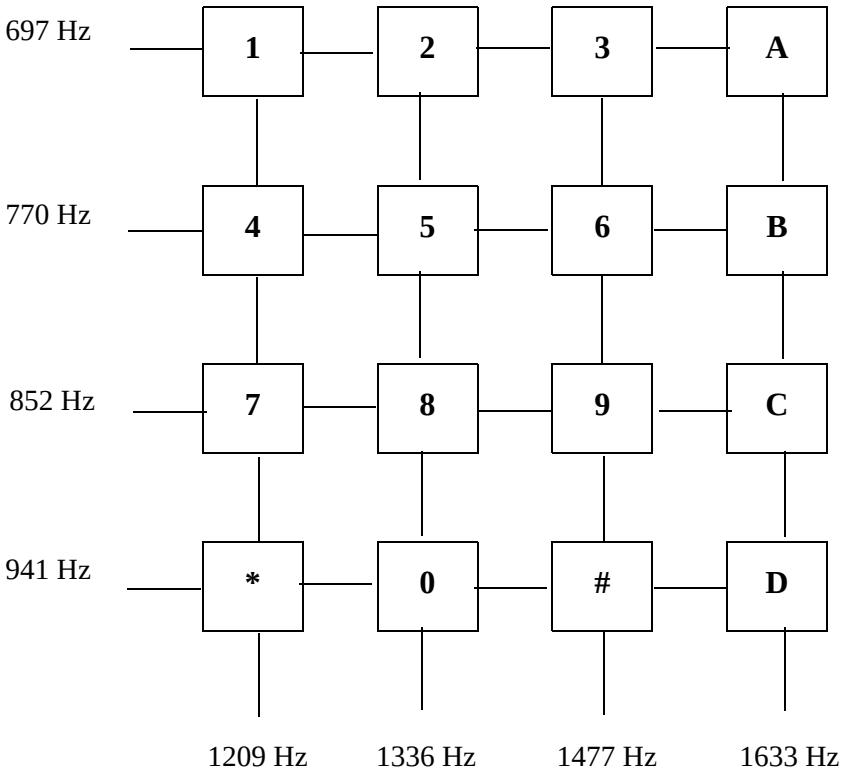

Figure 1: Extended DTMF encoding table for Touch Tone dialing. When any key is pressed the tones of the corresponding column and row are generated and summed. Common household phones do not have the fourth column.

4. Determine which key was pressed, **0–9**, *****, **#**, or **A–D** for extended DTMF, by converting frequency pairs back into key names according to Fig. 1.
It is possible to decode DTMF signals using a simple FIR filter bank. The filter bank illustrated in Fig. 2 consists of eight bandpass filters where each passes only one of the eight possible DTMF frequencies. The input signal for all the filters is the same DTMF signal.

Here is how the system should work: When the input to the filter bank is a DTMF signal, the outputs from two of the bandpass filters (BPFs) should be significantly larger than the rest. If we detect (or measure) which two are the large ones, then we know the two corresponding frequencies. These frequencies are then used to determine the DTMF code.

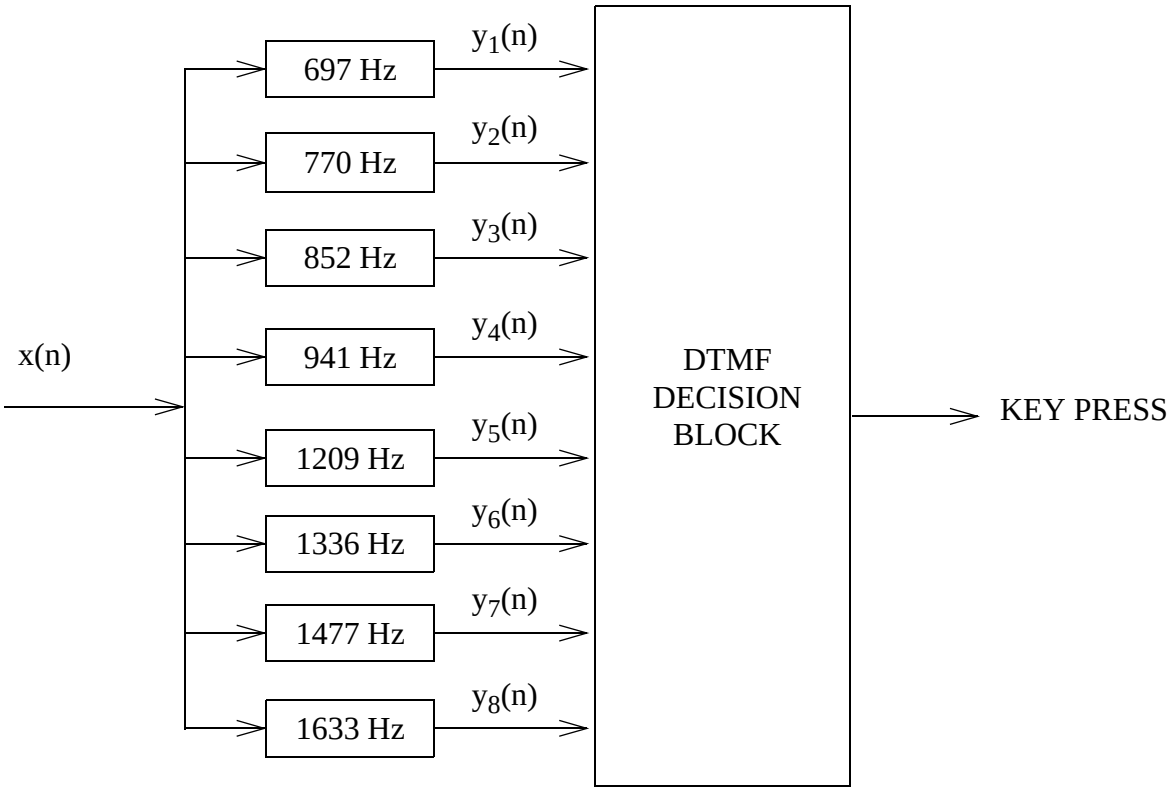

Figure 2: DTMF decoder consisting of a filter bank of bandpass filters which pass frequencies corresponding to the eight DTMF component frequencies listed in Fig. 1.

3 DTMF Synthesis

3.1 Signal Concatenation

When two signals are played one after the other, the composite signal is created by the operation of *concatenation.* In MATLAB, this can be done by making each signal a row vector, and then using the matrix building notation as follows:

```
>>xx = [ xx, xxnew ];
```

where `xxnew` is the sub-signal being appended. The length of the new signal is equal to the sum of the lengths of the two signals `xx` and `xxnew`. A third signal could be added later on by concatenating it to `xx`.

3.1.1 Comment on Efficiency

In MATLAB the concatenation method, `xx = [ xx, xxnew ]`, would append the signal vector `xxnew` to the existing signal `xx`. However, this becomes an *inefficient* procedure if the signal length gets to be very large. The reason is that MATLAB must re-allocate the memory space for `xx` every time a new sub-signal is appended via concatenation. If the length `xx` were being extended from 400,000 to 401,000, then a clean section of memory consisting of 401,000 elements would have to be allocated followed by a copy of the existing 400,000 signal elements and finally the append would be done. This is clearly inefficient, but

would not be noticed for short signals.
An alternative is to pre-allocate storage for the complete signal vector, but this can only be done if the final length is known ahead of time.

3.2 Using the `find` Function

A useful function in MATLAB is the `find()` function that allows vectors and matrices to be searched for values that match a given criteria. For instance, the following MATLAB code searches the vector `q` for values equal to 4 and for values greater than 10. The `find` function returns the indices to the values within the vector that match the search criteria.

```
q = [1, 4, 91, 5, 62, 4, 10, 38];
r1 = find(q == 4)
r2 = find(q > 10)
```

We can also search within a matrix and have the row and column values for the matched elements to be returned. For instance, the following MATLAB code[1] searches through a matrix for values that are 2 or smaller.

```
a = magic(4)
[y, x] = find(a <= 2 )
```

3.3 Overlay Plotting

Sometimes it is convenient to overlay information onto an existing MATLAB plot. The MATLAB command *`hold on`* will inhibit the figure erase that is usually done just before a new plot. Demonstrate that you can do an overlay by following these instructions:
(a) Plot the magnitude response of a 5-point averager filter, created from

```
[H,W] = freqz( ones(1,5) / 5, 1, -pi:pi/300:pi);
plot(W, abs(H));
```

Make sure that the horizontal frequency axis extends from -π to +π.
(b) Use the `stem` function to place vertical markers at the zeros of the frequency response.

```
hold on
stem(2*pi/5*[-2,-1,1,2],0.3*ones(1,4),'r.')
hold off
```

3.4 DTMF Dial Function

Write a function, `dtmfdial()`, to implement a DTMF dialer. A skeleton of `dtmfdial.m` is provided below.

```
function dtmfsig = dtmfdial(nums,fs)
%DTMFDIAL Create a vector of tones which will dial
% a DTMF (Touch Tone) telephone system.
%
% usage: dtmfsig = dtmfdial(nums,fs)
% nums = vector of numbers to dial ranging from 0 to 15.
% These numbers have the following mapping:
% 1=1, 2=2, 3=3, 12=A
% 4=4, 5=5, 6=6, 13=B
% 7=7, 8=8, 9=9, 14=C
% 10=*, 0=0, 11=#, 15=D
```

1. Note: The `magic(N)` function simply returns an N × N matrix with equal sums along rows, columns, and diagonals.

```
% fs = sampling frequency
% dtmfsig = vector containing the corresponding tones.
%
% ex: To generate the tones to dial 555-1234, use
% fs = 8000;
% tones = dtmfdial([5 5 5 1 2 3 4], fs);
% soundsc(tones, fs);
%
keylayout = [ 1, 2, 3, 12; ... % 1 2 3 A
4, 5, 6, 13; ... % 4 5 6 B
7, 8, 9, 14; ... % 7 8 9 C
10, 0, 11, 15]; % * 0 # D
tone_rows = [ 697; 770; 852; 941];
tone_cols = [1209, 1336, 1477, 1633];
tone_dur = % <-- Fill in this line
tt = 0:1/fs:tone_dur;
silence_dur = % <-- Fill in this line
silence = zeros(1, round(silence_dur * fs));
dtmfsig = [];
for k = 1:length(nums)
[y, x] = find( keylayout == nums(k) );
tone = % <-- Fill in this line
dtmfsig = % <-- Fill in this line. Make sure you
% include the tone as well as silence
% between tones.
end
```

In this warm-up, you must complete the dialing code so that it implements the following:

1. The input to the function is a vector of numbers, each one being between 0 and 15, with 0–9 corresponding to the digits, 10 corresponds to the ***** key, 11 to the **#** key, and 12–15 are the keys **A–D**.

2. The output should be a vector containing the DTMF tones, sampled at f_s = 11025 Hz. The duration of each tone should be about 0.25 seconds, and a silence, about 0.1 seconds long, should separate the DTMF tones. These times can be hard-wired into `dtmfdial`. Remember that each DTMF tone for a key is the sum of a pair of (equal amplitude) sinusoidal signals.

3. The frequency information is given in two vectors: `tone rows` contains the row frequencies, and `tone cols` contains the column frequencies for the touch tone pad. You can translate a key press to the correct frequency by first searching for the key through the `keylayout` matrix with the `find` function to get the x and y coordinates of the key. Then perform a look-up in `tone rows` and `tone cols` to determine the corresponding frequencies for that row and column. You must then generate sinusoids with frequencies equal to the corresponding position in `tone rows` and `tone cols`.

Your function should create the appropriate tone sequence to dial an arbitrary phone number. When played through a telephone handset, the output of your function will be able to dial the phone. You should use the `specgram` function to check your generated tones {e.g. specgram(dtmfsig,512,11025)}.

Instructor Verification (separate page)

4 FIR Bandpass Filters

4.1 FIR Bandpass Filter Design

An ideal discrete-time lowpass filter with zero-phase distortion has an impulse response of the form

$$h_{FIR-LP}(n) = \frac{\omega_c}{\pi}\frac{\sin(\omega_c n)}{(\omega_c n)}$$

where ω_c is the cutoff frequency for the filter. We can form a causal FIR approximation of $h_{FIR-LP}(n)$ using

$$h_{FIR-LP}(n) = w_N(n) \times \left[\frac{\omega_c}{\pi}\frac{\sin\left(\omega_c\left(n-\frac{N-1}{2}\right)\right)}{\left(\omega_c\left(n-\frac{N-1}{2}\right)\right)}\right] \text{.....equn 1}$$

where N is the number of filter coefficients and $w_N[n]$ is an appropriate windowing function of length N. The causal lowpass filter $h_{FIR-LP}(n)$ can be transformed into a causal bandpass filter $h_{FIR-BP}(n)$ using the relationship

$$h_{FIR-BP}(n) = (h_{FIR-LP}(n)) \times 2\cos\left(\omega_0\left(n-\frac{N-1}{2}\right)\right) \text{.....equn 2}$$

where ω_0 is the centre frequency that defines the frequency location of the passband.

Design a MATLAB function `bpfdesign.m` to be able to design a bandpass filter based on Eq. 2 and Eq. 1. You can use the following MATLAB code as a template for this function.

```
function h = bpfdesign(w0, wc, window)
%BPFDESIGN: Design a bandpass filter based on a windowed FIR
% approximation of an ideal IIR filter.
%
% h = bpfdesign(w0, wc, window)
% Returns a vector with the FIR filter coefficients.
% w0 = centre frequency (in rads/sample)
% wc = cutoff frequency (approximate) on either side of w0
% window = vector of length N with window multipliers
% Note: wc would be half the bandwidth. Also, note that
% the bandwidth here is only approximate and may
% have to be tuned.
%
N = length(window);
window = reshape(window,1,N);
n = 0:(N-1);
h = % <--- Fill in this line
return;
```

With your working `bpfdesign()` function, perform the following filter designs and analysis.

(a) Generate a bandpass filter with 32 taps using a rectangular window where the filter has a centre frequency of $\omega_0 = 0.6\pi$ and with bandwidth of 0.4π. Make a plot of the frequency response magnitude and phase.

(b) The *passband* of a BPF filter is defined by the region of the frequency response where $|H(e^{j\omega})|^2$ is close to its maximum value of one. Typically, the passband width is defined as the width of the frequency region where $|H(e^{j\omega})|^2$ is greater than 0.5, which corresponds to the -3 dB bandwidth of the filter. You can use MATLAB's `find` function to locate those frequencies where the magnitude satisfies

$|H(e^{j\omega})|^2 \geq 1/\sqrt{2}$ as follows:

```
[H, W] = freqz(h, 1, 0:pi/1000:pi);
k= find( abs(H) > 1/sqrt(2) );
W(k(1))
W(k(end))
```

Use the plot of the frequency response for the length-32 bandpass filter from part (a) to determine the passband width. Make two other plots of BPFs for N = 16 and N = 64 coefficients, and measure the passband width in both. Then explain how the width of the passband is related to filter length (*i.e.*, what happens when the filter length is doubled or halved).

(c) Now try designing a few other N = 32 tap BFP filters using the Hamming and Blackman windows. Comment on the differences between using these windows compared to the rectangular window.

Instructor Verification (separate page)

4.2 DTMF Bandpass Filter Design: `dtmfdesign.m`

We now wish to implement the filter bank of eight bandpass filters as illustrated in Fig. 2. To gain some flexibility, we introduce a multiplicative factor β in Eq. 2 to give

$$h_{FIR-BP}(n) = \beta \times (h_{FIR-LP}(n)) \times 2\cos\left(\omega_0\left(n - \frac{N-1}{2}\right)\right) \quad \text{.....equn 3}$$

The constant β gives flexibility for scaling the filter's gain to meet a constraint, such as making the maximum value of the frequency response equal to one. In dealing with analog filter specifications, the centre frequency ω_0 can be expressed as

$$\omega_0 = 2\pi\frac{f_0}{f_s} \quad \text{.....equn 4}$$

where f_0 is the continuous-time centre frequency in Hertz and f_s is the sampling frequency for the system. The parameter f_0 defines the frequency location of the passband, e.g., we pick f_0 = 852 Hz if we want to isolate any 852 Hz sinusoidal component.

Write a MATLAB function `dtmfdesign()` that will design the eight required bandpass filters for detecting each of the eight DTMF frequencies. A skeleton of one possible design for the `dtmfdesign()` function is given below.

```
function hh = dtmfdesign(fcent, B, fs, window)
%DTMFDESIGN
% hh = dtmfdesign(fcent, B, fs, window)
% returns a matrix (length(fcent) x length(window))
% where each row is the impulse response of a BPF,
% one for each frequency in fcent.
% fcent = vector of centre frequencies (in Hertz)
% B = vector of bandwidths for each BPF (in Hertz)
% window = vector of length N with window multipliers
% fs = sampling frequency
%
%
% Example:
% hh = dtmfdesign([697 770 852], [40 40 40], 8000, box-
```

```
car(31));
%
N = length(window);
n = 0:(N-1);
window = reshape(window,1,N);
for k = 1:length(fcent)
wc = 2*pi*(B(k)*0.5)/fs*2; % Cutoff frequency
w0 = 2*pi*fcent(k)/fs; % Bandwidth
% Design the bandpass filter
h = % <--- Fill in this line
% Make sure the peak is at unity gain
[H,W] = freqz(h, 1, 0:0.0001:pi);
beta = 1 / max(abs(H));
% Save the impulse response for this BPF into the hh matrix
hh(k,1:N) = beta * h;
end
```

Notice that the given `dtmfdesign()` skeleton accepts a vector `fcent` which holds the centre frequencies for each of the bandpass filters. Also, the bandwidth, window for the FIR bandpass filter and the sampling frequency in `fs` are specified as arguments to `dtmfdesign()`. The `dtmfdesign()` function returns a matrix `hh` of size `length(fcent) x length(window)` where each row is an impulse response for one of the FIR bandpass filters.

One issue when we later use the bandpass filters is that the maximum magnitude for the frequency response should be one. The skeleton code for `dtmfdesign()` analyzes the frequency response of `h` using `freqz()` and changes the gain of `h` using β such that the maximum magnitude of the frequency response is 1.0.

When you have completed your DTMF filter design function `dtmfdesign()`, you should test it for different window lengths and types to get a good passband width and good stopband rejection of other frequencies.

Use your "dtmfdesign()" to generate the eight (scaled) bandpass filters with f_s = 8000 Hz. Plot the magnitude of the frequency responses all together on one plot (the range for positive frequencies is sufficient since the magnitude of the frequency response is even symmetric). Indicate the locations of each of the eight DTMF frequencies (697, 770, 852, 941, 1209, 1336, 1477, and 1633 Hz) on this plot to illustrate whether or not the passbands are narrow enough to separate the DTMF frequency components. *Hint: use the* `hold` *command and markers as you did in the warm-up.*

Instructor Verification (separate page)

==========

The following MATLAB code fragment may help you make the appropriate plots.

```
% overlay_play_dtmf
fs = % <--- Fill in this line
N = % <--- Fill in this line
window = % <--- Fill in this line
dtmf_freqs = % <--- Fill in this line
```

```
bandwidths = % <--- Fill in this line
hold off;
stem(dtmf_freqs, ones(1,length(dtmf_freqs)), 'r')
hh = dtmfdesign(dtmf_freqs, bandwidths, fs, window);
hold on;
for k = 1:size(hh,1)
[H,W] = freqz(hh(k,:), 1, 0:pi/1000:pi);
plot(W * fs / (2 * pi), abs(H));
end
hold off;
```

Lab #3
SYSC 4405
winter 2015
Instructor Verification Sheet

Hand this page to the lab instructor after it is completed.

Name: Student ID:
Name: Student ID:

Part 3.4: Completed the dialing function `dtmfdial()` and explained the spectrogram:
Verified: Date/Time:_____

Part 4.1: Generated a bandpass filter centred at $\omega_c = 0.6\,\pi$ with bandwidth of 0.4π, looked at N = 16, 32, and 64–point versions, and compared rectangular, Hamming, and Blackman windows.
Verified: Date/Time:_____

Part 4.2: Demonstrated a working `dtmfdesign()` function and that the eight filters work as required.:
Verified: Date/Time:_____