GEORGIA INSTITUTE OF TECHNOLOGY

SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2026 Spring 2018
Lab #10: Detecting DTMF Signals

Date: 2–5 April 2018

Pre-Lab: You should read the Pre-Lab section of the lab and do all the exercises in the Pre-Lab section ***before your assigned lab time.***

Verification: The Exercise section of each lab must be completed **during your assigned Lab time** and the steps marked *Instructor Verification* must also be signed off **during the lab time**. One of the laboratory instructors must verify the appropriate steps by signing on the **Instructor Verification** line. When you have completed a step that requires verification, simply raise your hand and demonstrate the step to the TA or instructor. Turn in the completed verification sheet to your TA when you leave the lab.

Lab Report Questions: The Lab-Report Sheet has a few lab related questions that can be answered at your own pace. The completed Lab-HW sheet is due at the beginning of the next lab.

Forgeries and plagiarism are a violation of the honor code and will be referred to the Dean of Students for disciplinary action. You are allowed to discuss lab exercises with other students, but you cannot give or receive any written material or electronic files. In addition, you are not allowed to use or copy material from old lab reports from previous semesters. Your submitted work must be your own original work.

1 Introduction

Please read through the information below prior to attending your lab.

1.1 Objective

The goal of this lab is to learn study the Dual-Tone Multi-Frequency (DTMF) signals used in Touch-Tone phone dialing.[1]

1. *Short Sinusoids:* The DTMF signals are two short sinusoids summed together.
2. *Windowing:* The concept of windowing will be applicable when dealing with these finite-length signals.
3. *Bandpass Filtering:* Narrowband BPFs will be needed to isolate individual sinusoids within the DTMF signal
4. *Design Methods:* FIR filters designed via computer optimization (firpm) will be used.
5. *Average Power* must be computed to give a detectable value that is a positive number.

1.2 Review: Telephone Touch Tone Dialing

Telephone touch-tone keypads generate *dual tone multiple frequency* (DTMF) signals to represent digits in a phone number when dialing a telephone. When any key is pressed, the sinusoids of the corresponding row

FREQS	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**

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. Keys A-D (in the fourth column) are not implemented on commercial and household telephone sets, but might be used in some special signaling applications, e.g., military communications.

and column frequencies (see Fig. 1) are generated and summed, hence dual tone. As an example, pressing the **5** key generates a signal containing the sum of the two tones at 770 Hz and 1336 Hz together.

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.[2] This makes it easier to detect exactly which tones are present in the dialed signal in the presence of non-linear line distortions.[3]

1.3 Dual Tone Signals

For the DTMF synthesis each key-press generates a signal that is the sum of two sinusoids. For example, when the key **7** is pressed, the two frequencies are 852 Hz and 1209 Hz, so the generated signal is the sum of two sinusoids which could be created in MATLAB via

```
Ts = 0.3e-3;      %- Sampling period = 0.3 msec
fsamp = 1/Ts;     %- Sampling rate
tt = 0:1/fsamp:0.3;
DTMFsig = cos(2*pi*852*tt+rand(1)) + cos(2*pi*1209*tt+rand(1));  %- Use random phases
xx = zeros(1,round(2/Ts));  %- pre-allocate vector to hold DTMF signals
n1 = round(0.6/Ts);   n2 = n1+length(DTMFsig)-1;
xx(n1:n2) = xx(n1:n2) + DTMFsig;
%-- soundsc(xx,fsamp);  %- Optional: Listen to a single DTMF signal
plotspec(xx,fsamp); grid on %- View its spectrogram
```

2 Pre-Lab

2.1 DTMF Decoding

There are several steps to decoding a DTMF signal:

1. Filter the signal to separate the possible frequency components into eight bandpass channels.

2. Ideally, the output of each BPF is either zero, or a sinusoid of a known frequency.

[1]Touch Tone is a registered trademark

[2]More information can be found at: `http://www.genave.com/dtmf.htm`, or search for "DTMF" on the internet.

[3]A recent paper on a DSP implementation of the DTMF decoder, "A low complexity ITU-compliant dual tone multiple frequency detector", by Dosthali, McCaslin and Evans, in *IEEE Trans. Signal Processing*, March, 2000, contains a short discussion of the DTMF signaling system. You can get this paper on-line from the GT library, and you can also get it at `http://www.ece.utexas.edu/~bevans/papers/2000/dtmf/index.html`.

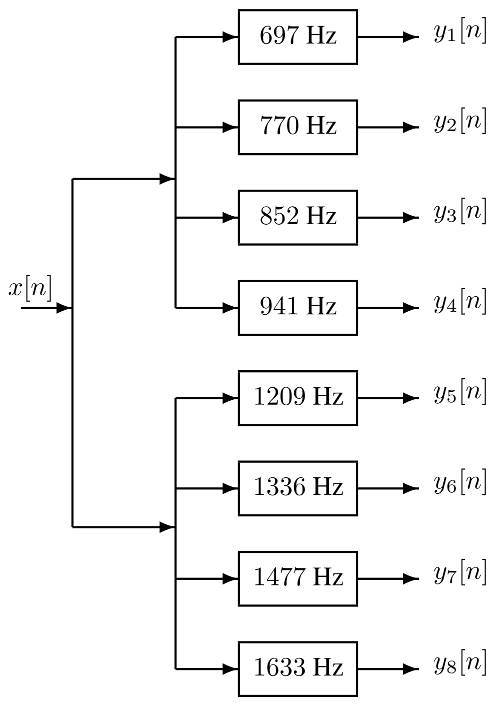

Figure 2: Filter bank consisting of bandpass filters to separate the dual-tone signals to perform frequency identification of the frequencies corresponding to the individual sinusoidal components of the DTMF signal as listed in Fig. 1. The eight filters are identified as Filter #1 to #8, from the top to the bottom in increasing frequency order.

3. Therefore, each BPF would be followed by further processing to estimate the amplitude, or average power, of the BPF output.

4. Decoding relies on the fact that only one row filter and one column filter should have a nonzero output at the same time. Determine which two frequency components are present in a specific time interval by measuring the size of the output signal from all of the bandpass filters during that time. Even when there is noise added to the signals, one row BPF output will be much larger than the other three; likewise, for the column BPFs.

5. It is necessary to isolate the signals from individual key presses. There must be short gaps of silence between separate key presses, and these short time intervals must be detected to find the beginning and end of the *distinct* key presses.

6. The final step is decoding each row-column frequency pair back into key names according to Fig. 1. The output is a list of keys that were pressed, selected from **0–9**, **A–D**, *****, or **#**.

2.2 Bandpass Filter Design

You will need a bandpass filter design function for this lab. The MATLAB functions `firpmord` and `firpm` can be used to design bandpass filters. The specifications for the band edges and ripples of the BPFs can be derived from the list of DTMF frequencies. The typical case will be to locate the passband of the BPF at one of the DTMF frequencies, and then define stopbands so that the other seven DTMF frequencies are attenuated by at least 40 dB.

2.2.1 Recall Filter Specifications

The specification of a LPF in terms of ripples, bandedges, and transition width can be summarized with the tolerance scheme shown in Fig. 3. The filter design process is to approximate the ideal frequency response very closely. Once we specify the desired ripples and bandedges, we can draw a template around the ideal frequency response. An acceptable filter design would be an FIR filter whose magnitude response lies entirely within the template. The length-23 FIR filter shown in Fig. 3 meets the specs, but if you designed a length-19 filter it would have a transition width that is greater than $\Delta\hat{\omega} = 0.08\pi$.

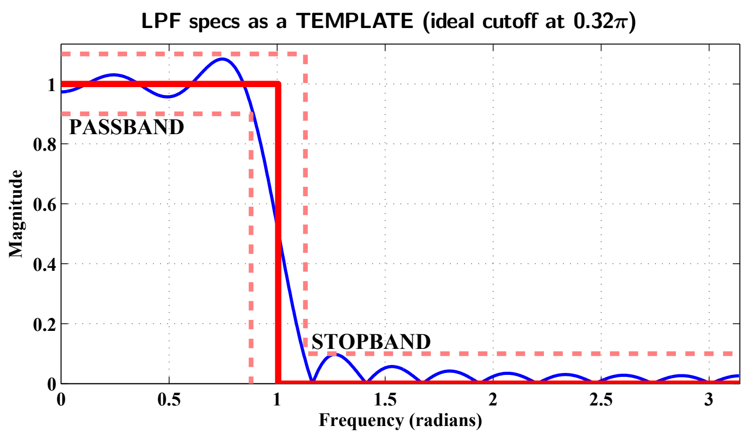

Figure 3: Tolerance scheme drawn around an ideal LPF with a cutoff frequency of $\hat{\omega}_c = 0.32\pi$. Dashed lines indicate the maximum allowable deviation from the ideal LPF. The template uses $\hat{\omega}_p = 0.28\pi$, $\hat{\omega}_s = 0.36\pi$, and $\delta_p = \delta_s = 0.1$. The actual FIR filter shown is the length-23 FIR filter that just barely meets these specs.

2.2.2 Filter Design via Optimization

Many different methods have been developed for filter design via mathematical optimization. One of the widely used methods is `firpm` in MATLAB. For designing a LPF, it uses the following two step process:

1. Use the desired specifications for $\hat{\omega}_p$, $\hat{\omega}_s$, δ_p, and δ_s to estimate the filter order (M) that will be needed. This is done with the MATLAB function `firpmord`.

2. Use the outputs from `firpmord` as inputs to the function `firpm` to run the optimization and obtain the FIR filter coefficients that should meet the specs on δ_p and δ_s. In effect, the inputs to `firpm` are $\hat{\omega}_p$, $\hat{\omega}_s$, M, and the ratio δ_p/δ_s.

3. If the ripple specs are not met with the predicted order, then increase the order by one and try again. A higher order such as $M + 1$ or $M + 2$ should meet the specs.

4. FIR filters designed by this method will have linear phase in their frequency response. The slope of the phase vs. frequency $(\hat{\omega})$ is the delay in the time domain. This is a consequence of the delay property of the DTFT:

$$y[n] = x[n - n_d] \quad \longleftrightarrow \quad Y(e^{j\omega}) = e^{-j\hat{\omega}n_d}X(e^{j\omega})$$

For the calling arguments of these functions, do `help firpmord` and `help firpm`.

2.3 Synthesizing Long Signals

Long signals can be created by joining together many sinusoids. When two signals are played one after the other, the composite signal could be 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.

2.3.1 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 the vector xx every time a new sub-signal is appended via concatenation. If the length of 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 signal length is known ahead of time.

2.4 Review: Encoding from Frequency Vectors

Explain how the following program uses frequency information stored in two vectors to generate a long signal. *Note: this code will not synthesize a correct DTMF signal.*

From the frequency information in the vectors f1 and f2 and the pairs in the keys array, determine the frequencies played. Then determine the total length of the signal played by the soundsc function. How many samples and how many seconds?

```
f1 = [11,13,14,17]*70
f2 = [2,3,5,7,8]*85
fs = 10000/3;
xx = [ ];
keys = [1,1; 3,4; 2,5; 3,3; 1,5; 4,2]
xx = zeros(1, 1200*size(keys,1));  %- pre-allocate
disp('--- Here we go through the Loop ---')
n1 = 1;
for ii = 1:size(keys,1)
   n2 = n1+299;
   xx(n1:n2) = xx(n1:n2) + zeros(1,300);  %- precede each key with silence
   n1 = n1+300;
   n2 = n1+899;
   k1 = keys(ii,1);   k2 = keys(ii,2);
   xx(n1:n2) = xx(n1:n2) + cos(2*pi*(f1(k1)+f2(k2))*(0:899)/fs);  %-- NOT a DTMF signal
   n1 = n1+900;
end
%-- soundsc(xx,fs);  %- OPTIONAL
plotspec(xx,fs); grid on
```

2.5 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 the 7-point averager, created from

```
HH = freqz((1/7)*ones(1,7),1,ww)
```

Make sure that the horizontal frequency axis extends from $-\pi$ to $+\pi$.

(b) Use the `stem` function to place vertical markers at the zeros of the frequency response.

```
hold on, stem(2*pi/7*[-3,-2,-1,1,2,3],0.3*ones(1,6),'r.'), hold off
```

2.6 Plotting Multiple Signals

The MATLAB function `strips` is a good way to plot several signals at once, e.g., the eight outputs from the BPFs. Observe the plot(s) made by `strips(cos(2*pi*linspace(0,1,201)'*(4:10)));`
Alternatively, in the *SP-First* toolbox, the function `striplot` can be used to plot multiple signals contained in the columns of a matrix via: `striplot(xmat,fs,size(xmat,1));`

3 In-Lab Exercise: DTMF Filtering

The objective of the lab exercise is to synthesize DTMF signals from a phone number, and then filter the signal with two bandpass filters.

3.1 Review: Touch-Tone Dial Function (from Lab03 Report)

Write a function, `DTMFdial.m`, to implement a Touch-Tone dialer based on the frequency table defined in Fig. 1. A skeleton of `DTMFdial.m` is given in Fig. 4.
In this exercise, you must complete the dialing code so that it implements the following:

1. The input to the function is a vector of characters, each one being equal to one of the key names on the telephone. The n-th character is `keyNames(n)`. The MATLAB array called `TTkeys` containing the key names is a 4×4 matrix that corresponds exactly to the keyboard layout in Fig. 1. To convert any key name to its corresponding row-column indices, consider the following example:

   ```
   [jrow,jcol] = find('3'==TTkeys)
   ```

2. The output should be a vector of signal samples (at $T_s = 0.3\,\text{ms}$) containing the DTMF sinusoids—each key being the sum of two sinusoids. Remember that each DTMF signal is the sum of two (equal amplitude) sinusoidal signals. The duration of each tone pair should be exactly 180 ms, and a gap of silence, exactly 48 ms long, should separate the DTMF tone pairs. These times can be declared as fixed variables in the code for `DTMFdial`, i.e., there is no need to pass the durations as input variables.

3. The frequency information is given as two 4-element vectors (`TTcolTones` and `TTrowTones`): one contains the column frequencies, the other has the row frequencies. You can translate a key such as the **6** key into the correct location in these vectors by using MATLAB's `find` function. For example, the **6** key is in row 2 and column 3, so we would generate sinusoids with frequencies equal to `TTrowTones(2)` and `TTcolTones(3)`.

   Also, consult the MATLAB code in Section 2.4 for hints about writing `DTMFdial.m`.

```
function xx = DTMFdial(keyNames,fs)
%DTMFDIAL  Create a signal vector of tones that will dial
%           a DTMF (Touch Tone) telephone system.
%
% usage:  xx = DTMFdial(keyNames,fs)
%  keyNames = vector of CHARACTERS containing valid key names
%        fs = sampling frequency
%   xx = signal vector that is the concatenation of DTMF tones.
%
TTkeys =  ['1','2','3','A';
            '4','5','6','B';
            '7','8','9','C';
            '*','0','#','D'];
TTcolTones = [1209,1336,1477,1633];  %-- in Hz
TTrowTones = [697,770,852,941];
numKeys = length(keyNames);
durDualTone = ?  %-- in seconds
LenDualTone = ?
durSilence = ?
LenSilence = ?
xx = ....  %- initialize xx to be long enough to hold the entire output
n1 = 1;
for kk=1:numKeys
   [jrow,jcol] = find(....  %- which key?
   ... more code to make the dual-tone signals
   ... precede each dual-tone signal with a short interval of silence
end
```

Figure 4: Skeleton of `DTMFdial.m`, a Touch-Tone phone dialer. Complete this function by adding more lines of code to generate the dual-tone sinusoids. The vector of characters needed for the input `keyNames` is actually a string, e.g., `'9785551234ABCD'`.

4. You could implement error checking so that an illegitimate key name is rejected.

Your function should create the appropriate tone sequence to dial an arbitrary phone number. In fact, when played through a speaker into a conventional telephone handset, the output of your function will be able to dial the phone.[4] For your own verification, please use `plotspec` to show the time-frequency analysis of the generated signal for the key sequence `'159D*286A'` when the sampling period is $T_s = 0.3$ ms.

3.2 FIR Filter Designs

In the following parts, you should use the `firpm` filter design method (Sect. 2.2.2) to create bandpass filters similar to what will be needed in the Touch-Tone decoder.

Filter #2 Specifications: Design a bandpass filter with a pass band from $(770 - \Delta f)$ Hz to $(770 + \Delta f)$ Hz with $\Delta f = 6$ Hz. The center frequency of 770 Hz will pass one of the DTMF frequencies. Choose the stopbands to reject all the rest of the DTMF frequencies. The easiest way to do this is to define two stopbands: one from 0 to 697 Hz, the other from 852 Hz to $\frac{1}{2}f_s$ Hz. Assume that the sampling interval is $T_s = 0.3$ ms. The passband ripple specification is $\pm 2\%$, i.e., 1 ± 0.02. The stopband ripple is defined by requiring that δ_s be less than -40 dB which is a factor of 100 lower than the passband value of one.

(a) A BPF has two transition zones. Determine the two transition widths in normalized frequency $\hat{\omega}$.

[4]In MATLAB the demo called `phone` also shows the waveforms and spectra generated in a Touch-Tone system.

(b) Use the `firpm` filter design method to create a bandpass filter that meets the specs above with the goal of minimizing the filter order M. Some trial and error with the order M might be needed to minimize the filter order M while meeting the specs. Use only even orders for M. Summarize the results in the table provided.

(c) Make a plot of the frequency response magnitude for the designed filter versus frequency in Hz, and show that the specifications on the passband and stopbands are met, i.e., correct bandedges and ripples. *Note:* recall the relationship between $\hat{\omega}$ and f in Hz.

(d) Generate a DTMF signal for the key sequence `'159D*286A'`. Then filter this signal with the designed BPF. In order to verify that the filter worked properly, plot spectrogram(s) of the input and output signals. There should be only one frequency present at a time. Explain why this is the case.

Instructor Verification (separate page)

(e) Design Filter #5, at 1209 Hz, and summarize you results in the table provided. Continue using $\Delta f = 6$ Hz. In this case, there is some flexibility in choosing the stopband cutoff frequency for the upper stopband. However, the two transition widths should be about the same size; otherwise, the filter design result might have an undesirable shape in the transition region which is an unconstrained region for the optimization. *Note: We will not ask you to design Filter #5 in this in-lab exercise, but you will use the same design rules as in designing Filter #2 above. You also apply the same rules designing other six bandpass Filters #3 to #7 in report.*

(f) Design Filter #8, which is a highpass filter for 1633 Hz, and summarize your results in the table provided. In this case, there is only one transition region, and there is no need for Δf.

Instructor Verification (separate page)

3.2.1 Delay Compensation

You need to have the same delay through all eight channels. The filters are linear-phase, so the delay through an M^{th} order filter is $\frac{1}{2}M$. In order to get the same overall delay, so any filters that are shorter must be cascaded with a pure delay to guarantee time-alignment of the channels. Rcall that the impulse response of a pure delay is $\delta[n - n_d]$, which can be implemented in MATLAB with `firfilt([0,0,...,0,1],sig)`. In other words, the filter coefficients are all zeros with a one at the position of the delay.

3.3 Design the Average Power Module

The output of each BPF is an oscillating signal because it is a signal that is concentrated at one frequency, i.e., at the center frequency of the channel. In order to perform detections, we must determine the "size" of the signal. One easy way is to compute the average power

$$\text{AVERAGE POWER} = p_{y_k}[n] = \frac{1}{L_p}\sum_{m=0}^{L_p-1}(y_k[n-m])^2$$

where $y_k[n]$ is the output signal from the BPF in the k^{th} channel. In this case, the average power is taken over L_p sample points.

An alternative viewpoint is that the average power operation is trying to estimate the DC value of the squared signal. To extract the DC value, we need a LPF with a very narrow passband, so another way to do the processing is to square the signal first, and then use the squared signal as the input to filter with a very narrowband LPF. A good narrowband filter can be designed with the Hamming window method. Recall that increasing the window length will make the passband narrower.

The length of the LPF (or averager) should be long, but cannot be longer than the expected silence between the dual-tones. If we expect 48 ms of silence between the dual-tones, then at a sampling interval of 0.3 ms there are 160 samples in the silence region. This will be an upper bound on L_p. Other factors will come into play once we discuss a detection strategy, and *down-sampling.*

3.4 Down-Sampling the DTMF Average-Power Output

The average power output from the LPF varies slowly. Therefore, the output can be down-sampled which means that every R^{th} value will be taken

```
pDownSampled = avgPower(1:R:end)
```

This is discrete-time to discrete-time sampling, but the same issues of aliasing apply as in continuous to discrete sampling. If you plot the frequency response of the LPF designed with the Hamming window, the passband width will be about $4\pi/L_p$, if measured at the first zero crossing.. This is the highest frequency in the output signal for average power, so we must sample at twice that frequency, or $8\pi/L_p$. The down-sampling factor is 2π divided by $8\pi/L_p$, or $L_p/4$ (rounded to an integer).

(a) Design a Hamming window LPF with $L_p = 99$, and plot its frequency response.

(b) Determine the value of R that you will use for down-sampling.

Instructor Verification (separate page)

4 Lab Report: Design All Eight Filters for DTMF Detection

YOU HAVE TWO WEEKS TO DO THIS REPORT AND THE GRADE COUNTED TRIPLE BECAUSE WE HAVE NO RERPORTS FOR LABS #11 and #12. WE EXPECT YOU TO WORK HARD ON BAND-PASS FILTER DESIGN.

4.1 Completing FIR Filter Designs

Complete the design of all the filters that will be needed for the DTMF decoder which will be explored in Part (B) of this Lab. All but one of the filters should be BPFs; the highest DTMF frequency should be handled with a highpass filter (HPF).

1. Use the same specifications for the ripples δ_p and δ_s as in the Lab Exercise above.
2. For the band edges use the neighboring DTMF frequencies to pick the stopband edges, but keep the transition widths comparable.
3. For the passband edges, use $\pm\Delta f$ around the center frequency which is one of the DTMF frequencies.
4. Recall that, if the ripple specs are not met with the predicted order, then increase the order by one and try again. A higher order such as $M + 1$ or $M + 2$ should meet the specs.
5. For the DTMF filters, make sure that all the designed filters are even-order filters. If necessary, increase M by one to satisfy this constraint.

4.1.1 Delay Compensation

You need to have the same delay through all eight channels. The filters are linear-phase, so the delay through an M^{th} order filter is $\frac{1}{2}M$. In order to get the same overall delay, so any filters that are shorter must be cascaded with a pure delay to guarantee time-alignment of the channels. Rcall that the impulse response of a pure delay is $\delta[n-n_d]$, which can be implemented in MATLAB with `firfilt([0,0,...,0,1],sig)`. In other words, the filter coefficients are all zeros with a one at the position of the delay.

4.2 DTMF Detection Criterion

Once the average power outputs are available at a low sampling rate, the signals from the eight channels can be processed by a detector that determines the keys. A MATLAB function `decodeDTMF` will be provided for this task. The detection algorithm implemented in `decodeDTMF` has the following steps.

1. At each sample time, take the four row channels and determine if one of the channels is significantly higher than the other three. If so, declare that channel to be active with a tone that denotes the row number on the keyboard. If all the channel output are nearly equal, declare the output to be zero. This turns the 4 channel signals into a stream of integers for row indices (or zero for no signal, i.e., silence).
2. Repeat the previous step for the four column channels.
3. Use the keyboard mapping to decode the row-column indices obtained in the previous two steps. Use `'Z'` to indicate silence when one of the indices is zero.
4. At this point, there will be many repetitions because the sampling rate of the average power outputs is much greater than the signaling rate of pressing buttons. To remove the repetitions, there are 3 steps:
   (a) Look for "three-repeats." Remove any sequence that does not repeat at least three times, and also compress any long sequence of repeats into three. The assumption of this strategy is that any silence region will have at least three consecutive `'Z'` characters.
   (b) Compress all three-repeats into singletons.
   (c) There will be `'Z'` characters in every other position, so remove those to get the true phone number.

The fact that you want to get three consecutive `'Z'` characters means that you must limit the length of the averager L_p and the down-sampling factor R so that you will get at least three samples in the assumed silence region.

The function `phoneNumber = decodeDTMF(avgPower,fs,R)` will perform the steps described above. The input `avgPower` is a matrix with 8 columns. The length of the columns is the number of signal values after down-sampling. The ordering of the columns is by DTMF frequency, i.e., the first column is channel #1 with $f = 697\,\text{Hz}$. The input `fs` is the original sampling rate, so the rate after down-sampling is `fs/R`.

(a) Write a MATLAB program that will filter a signal through the 8 BPF channels. Display some spectrograms to show that the filtering is correct.

(b) Then, filter the output squared to get average power (for each channel), and downsample. Make a plot of all 8 output waveforms $p_k[n]$ together by using the MATLAB function `strips`. In this plot you should be able to visually decode the DTMF signal because exactly two channels will be active at any one time.

(c) Put those 8 waveforms into a matrix and call `decodeDTMF` to see if you get the correct answer.

Lab #10
ECE-2026 Spring-2018
INSTRUCTOR VERIFICATION SHEET

Turn this page in to your lab grading TA before the end of your scheduled Lab time.

Name: ____________________ gtLoginUserName: ____________ Date: ______

Part	Observations
3.2(a)	Transition widths from specifications of Filter #2 designed with `firpm`. (use $\hat{\omega}$)
3.2(b)	Design Filter #2 and summarize info in table below: the specs, the order M, and measure the actual stopband ripple δ_s:
3.2(c)	Frequency response of designed filter:
3.2(d)	Spectrograms before and after filtering:

Verified:____________________ Date/Time:__________

3.2(f)	Design Filter #8 (HPF) and summarize results in table below:

Verified:____________________ Date/Time:__________

Filter	M (even)	$\hat{\omega}_{s_1}$	$\hat{\omega}_{p_1}$	$\hat{\omega}_{p_2}$	$\hat{\omega}_{s_2}$	$\delta_{s_1}^{\text{MEAS}}$	Delay
#2							
#8				**X**	**X**		

Verified:____________________ Date/Time:__________

3.4(a)	Design Hamming window LPF
3.4(a)	Show Frequency Response of LPF
3.4(b)	Pick down-sample factor R. Explain.

Verified:____________________ Date/Time:__________

Lab #10
ECE-2026 Spring-2018
LAB REPORT QUESTION

Turn this page in to your lab grading TA together with your CODE at the very beginning of your scheduled Lab time during the week of 12-14 April, i.e., you have one EXTRA week to do this set of report, and the grade here will be counted TRIPLE because there is no scheduled reports for Labs #11 and #12.

Name: ______________________ gtLoginUserName: ______________ Date: _______

Part 4.1: Design all eight FIR filters for the DTMF filter bank. Using the optimization method in `firpm`.

(a) The specs are dictated by the eight DTMF frequencies. The passband ripple for all filters should be $\delta_p = 0.02$, and the stopband deviation (ripple) $\delta_s = 0.01$.

(b) In the passband, use $\Delta f = 6\,\text{Hz}$ to give the passbands a little bit of width. If you want to experiment with Δf, you can try reducing its value in the hope that you will get a lower order M for the filters.

(c) Fill in the table below.

Filter	M (even)	$\hat{\omega}_{s_1}$	$\hat{\omega}_{p_1}$	$\hat{\omega}_{p_2}$	$\hat{\omega}_{s_2}$	$\delta_{s_1}^{\text{MEAS}}$	Delay
#1							
#2							
#3							
#4							
#5							
#6							
#7							
#8				**X**	**X**		

Part 4.2: Test all eight FIR filters for DTMF decoding.

Part	Observations
4.2(a)	Process DTMF signal through 8 channels with BPFs. Show spectrogram of some outputs.
4.2(b)	Process outputs through average power filter. Show output waveforms after down-sampling.
4.2(c)	Send output signals to the detection function, and verify correct answer.