Digital Signal Processing Course Project Generation and Detection of DTMF Signals

Zexi Huang*
Yingcai Honors College

December 25, 2016

Abstract

Dual-tone multi-frequency (DTMF) signaling technology has reshaped the way of dialing and soon became the industry standard for landline and mobile service after it was invented. Here, we simulate the process of DTMF signal generation and detection based on methods implemented in hardware DSP modules with MATLAB. In addition to implementation of algorithms, a GUI is designed for better illustration.

1 Introduction

In telephone equipped with a keypad instead of a rotary dial, pressing of each buttons generates a unique set of two-tone signals, i.e., DTMF signals, which are further processed at the telephone central office to identify the key pressed by extracting the two frequency components that compose them. These frequencies are choosen from two sets, the low-band frequency group and the high-band one, and are standardized by ITU-T Recommendation Q.23 [1], which is illustrated in Figure 1.

Here, we simulate the process of DTMF singal generation and detection with MATLAB and also provide a GUI for better illustration of the rationale. The remainder of this report is organized as follows: In the following section, we introduce the algorithms of DTMF generation and detection in an hardware DSP module pespective. Section 3 presents the numerical experiments and the GUI implementation. Finally, we give a short discussion and conclude in Section 4.

2 Problem Formulation and Algorithms

As introduced before, each DTMF signal is represented as a sum of two frequency components, that is

$$g(t) = [\cos(2\pi f_L t) + \cos(2\pi f_H t)]u(t) \tag{1}$$

*Correspondence should be addressed to Z. Huang. E-mail: Eitima@163.com. Student No. 2014030302014

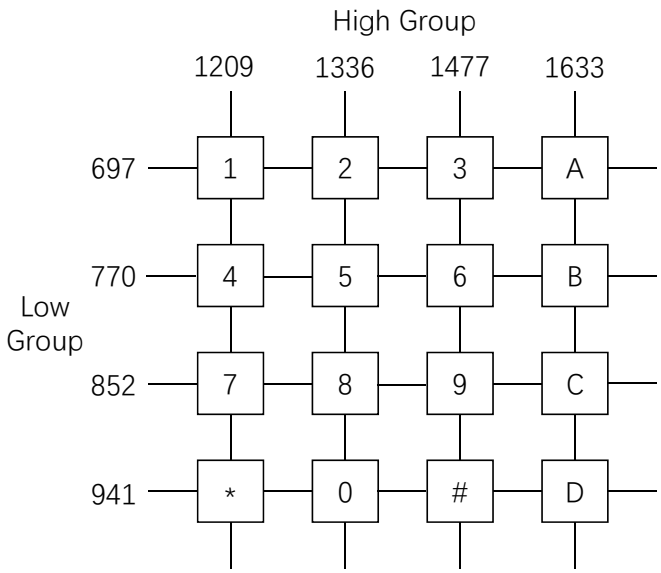

Figure 1: The tone frequency assignments for DTMF signals.

and the sampled discrete signal could be expressed as

$$g[n] = [\cos(\frac{2\pi f_L}{f_s}n) + \cos(\frac{2\pi f_H}{f_s}n)]u[n] \tag{2}$$

where f_L and f_H are frequencies from the low-band group and high-band respectively and f_s is the sampling frequency.

Our goals are:

- generate $g[n]$ (samples of $g(t)$) given a specified key is pressed (f_H and f_L are determined) and
- identify the key pressed (determine f_H and f_L) given the singal $g[n]$.

In addition, these objectives should be accomplished in a highly efficient manner so that hardware DSP modules with limited resources can implement. The detailed algorithms for implementation are illustrated in following subsections.

2.1 DTMF Signals Generation

To generate a DTMF signal, the fist step is to generate any cosine sequence given its frequency. For a DSP module, a common and efficient method is to implement an IIR filter with one of its poles located exactly in the unit circle, resulting an oscillation. The rationale is explained as follows.

Consider a causal cosine sequence to be generated:

$$h[n] = \cos(\omega_0 n)u[n] \tag{3}$$

Its z-transform $H(z)$ is

$$H(z) = \frac{1 - [\cos \omega_0] z^{-1}}{1 - [2 \cos \omega_0] z^{-1} + z^{-2}} \tag{4}$$

Given ω_0, the above expression specifies a second order IIR filter shown in Figure 2.

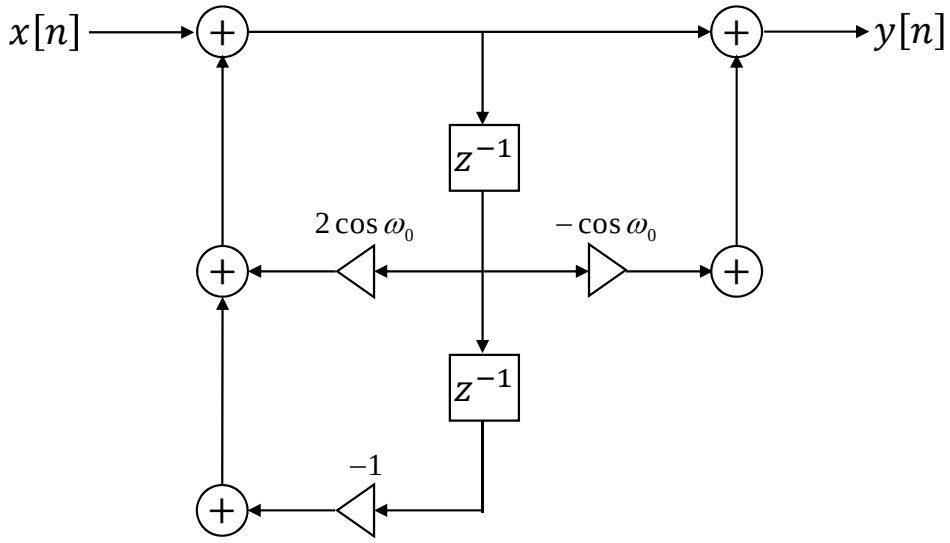

Figure 2: Block diagram of the digital oscillator.

By setting the input sequence $x[n] = \delta[n]$, the output $y[n] = h[n]$ can be specified by the following difference equation

$$h[n] - [2 \cos \omega_0] h[n-1] + h[n-2] = \delta[n] - [\cos \omega_0] \delta[n-1] \tag{5}$$

with its initial condition

$$h[-2] = h[-1] = 0 \tag{6}$$

Now, after specifying ω_0, $h[n]$ can generated by recursively evaluating Equation 5.

The final DTMF singal $g[n]$ is then computed by summing $h_L[n]$ and $h_H[n]$, the generated cosine sequences when $\omega_0 = \frac{2\pi f_L}{f_s}$ and $\omega_0 = \frac{2\pi f_H}{f_s}$.

2.2 DTMF Signals Detection

For a received DTMF signal $g[n]$ as specified in Equation 2, the method to identify its composed frequency component is, to analyze its frequency domain $G(e^{j\omega})$,

$$G(e^{j\omega}) = \frac{1 - [\cos \frac{2\pi f_L}{f_s}] e^{-j\omega}}{1 - [2 \cos \frac{2\pi f_L}{f_s}] e^{-j\omega} + e^{-2j\omega}} + \frac{1 - [\cos \frac{2\pi f_H}{f_s}] e^{-j\omega}}{1 - [2 \cos \frac{2\pi f_H}{f_s}] e^{-j\omega} + e^{-2j\omega}} \tag{7}$$

the poles of which in $[0, 2\pi]$ are

$$\omega = \frac{2\pi f_L}{f_s}, \frac{2\pi f_H}{f_s}, 2\pi - \frac{2\pi f_L}{f_s}, 2\pi - \frac{2\pi f_H}{f_s} \tag{8}$$

By finding these corresponding frequencies with peak values of $G(e^{j\omega})$, the required frequencies are determined and the key pressed is identified. To implement such an algorithm in a DSP module, discrete Fourier transform $G[k] = G(e^{j\omega})|_{\omega=2\pi k/N}$ is computed instead of continuous function $G(e^{j\omega})$. Although fast algorithms for computing DFT are readily available in DSP, such as FFT, Goertzel's algorithm [2] is often more attractive in this case since only a few samples of DFT are required while FFT always computes all the DFT samples. Thus here we use Goertzel's algorithm to compute the required DFT of the DTMF signals. The details are explained as follows.

Consider the DFT of the sequence $g[n]$,

$$G[k] = \sum_{l=0}^{N-1} g[l] e^{-j\frac{2\pi}{N}kl} \tag{9}$$

Note

$$e^{j\frac{2\pi}{N}kN} = 1 \tag{10}$$

we can rewrite $G[k]$ as

$$G[k] = e^{j\frac{2\pi}{N}kN} \sum_{l=0}^{N-1} g[l] e^{-j\frac{2\pi}{N}kl} = \sum_{l=0}^{N-1} g[l] e^{j\frac{2\pi}{N}k(N-l)} \tag{11}$$

Replacing N with n in the above expression, we could define a new sequence $p[n]$ that is in the form of a convolution,

$$p_k[n] = \sum_{l=0}^{n-1} g[l] e^{j\frac{2\pi}{N}k(n-l)} = g_e[n] * q_k[n] \tag{12}$$

where

$$g_e[n] = \begin{cases} g[n], 0 \leqslant n \leqslant N-1 \\ 0, \text{otherwise} \end{cases} \tag{13}$$

and

$$q_k[n] = e^{j\frac{2\pi}{N}kn} u[n] \tag{14}$$

Then

$$G[k] = p_k[n]|_{n=N} \tag{15}$$

Here, we note that Equation 12 also specifies a LTI system where the input is the extended version of DTFT signal $g[n]$ and the output at $n = N$ is exactly the required DFT $G[k]$, with system transfer function

$$Q_k(z) = \frac{1}{1 - e^{j\frac{2\pi}{N}k} z^{-1}} \tag{16}$$

or equivalently

$$Q_k(z) = \frac{1 - e^{-j\frac{2\pi}{N}k} z^{-1}}{1 - 2\cos(\frac{2\pi}{N}k) z^{-1} + z^{-2}} \tag{17}$$

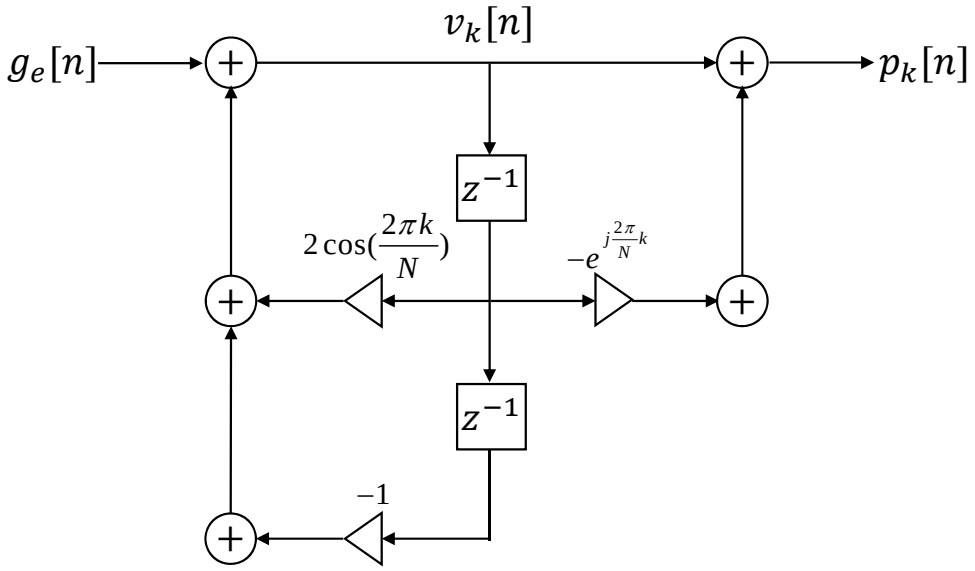

Figure 3: Block diagram of the system implementing Goertzel's algorithm.

resulting in the realization shown in Figure 3. The respective difference equation is

$$v_k[n] = g_e[n] + 2\cos(\frac{2\pi k}{N})v_k[n-1] - v_k[n-2] \tag{18}$$

with its initial condition

$$v_k[-2] = v_k[-1] = 0 \tag{19}$$

And the required DFT $G[k]$ is computed by

$$G[k] = p_k[N] = v_k[N] - e^{-j\frac{2\pi}{N}k}v_k[N-1] \tag{20}$$

For DTMF signal, the magnitude of $G[k]$ alone is enough to determine the frequencies with peak values, thus we could compute

$$|G[k]|^2 = v_k^2[N] + v_k^2[N-1] - 2\cos(\frac{2\pi k}{N})v_k[N]v_k[N-1] \tag{21}$$

considering both $g_e[n]$ and $v_k[n]$ are real sequences. Now we also see the reason why we adopt Equation 17 instead of Equation 16 to implement the system: it requires less complex operations and in the end, finding $|G[k]|^2$ by recursively computing Equation 20 and evaluating Equation 21 involves no complex operations at all.

Finally, f_H and f_L are determined by finding the respective frequencies where the peak values of $|G[k]|^2$ are reached, or for implementation by a computer program, identifying the local maxima of $|G[k]|^2$.

3 Numerical Experiments and GUI Interaction

3.1 Parameters and Enviroment

Some numerical parameters used in simulation are specified here. First, as is common in telephone services, the sampling frequency f_s is chosen to be $f_s = 8kHz$, and each DTMF signal lasts $t_0 = 50ms$ in a time slot of $100ms$. Therefore, $n_0 = t_0 \times f_s = 400$ samples are required for each DTMF signal. Second, for antialising in frequency domain, $N = n_0 = 400$ is used in the detection process.

The simulation environment for the whole process is MATLAB 9.0.0.341360 (R2016a), and all sources codes used in this simulation and listed in the appendix are of that syntax.

3.2 DTMF Generation Results

CosineGenerator.m is the digital oscillator-based implementation of generating a cosine sequence given its length and radian frequency. Then by calling it, *DTMFGenerator.m* generates a DTMF sequence given the key pressed based on the mapping relation indicated in Figure 1.

As an example, the time sequences generated when the input keys are '4' and '#' are shown in Figure 4 and Figure 5, respectively. However, it's hard to direct tell their expressions in time domain as in Equation 1 from these figures.

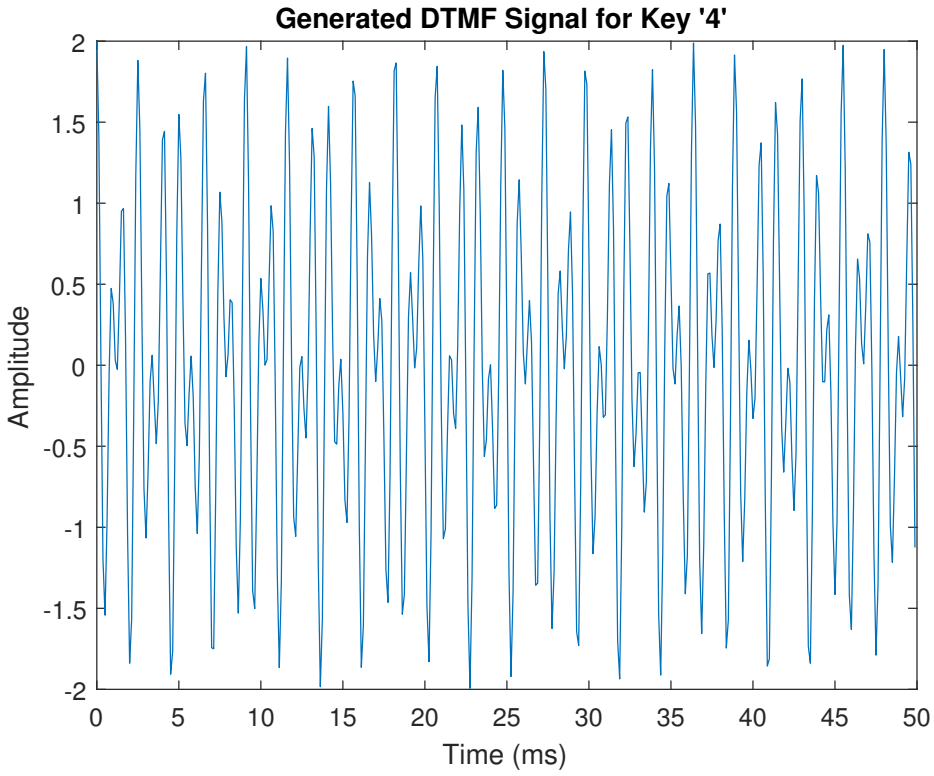

Figure 4: DTMF signal for key '4'.

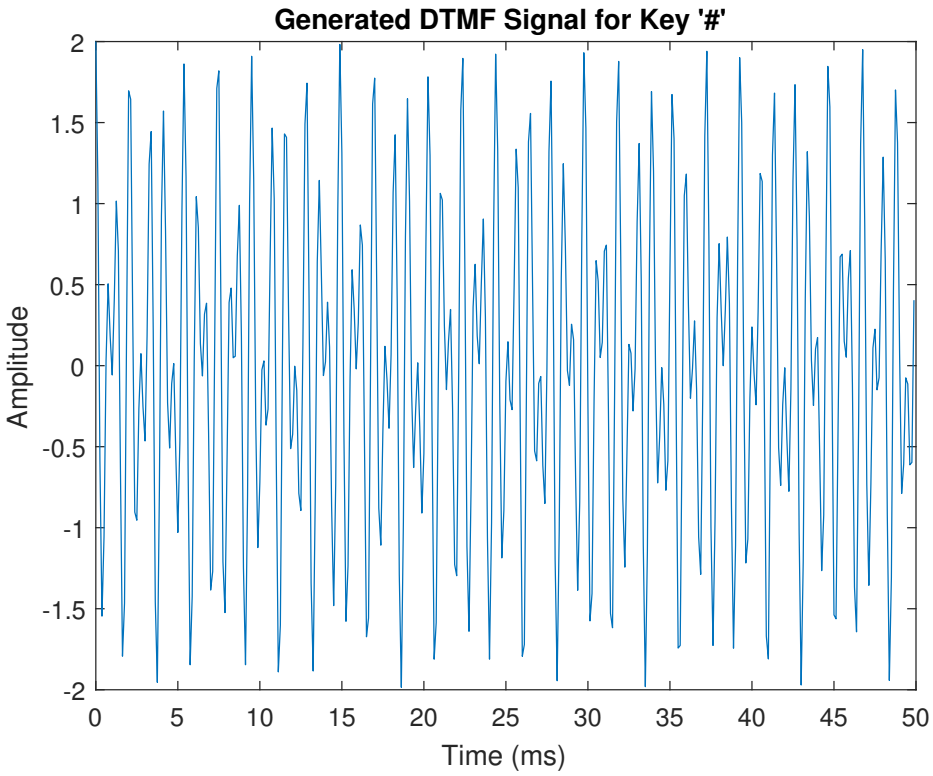

Figure 5: DTMF signal for key '#'.

3.3 DTMF Detection Results

Goertzel's algorithm is first implemented in *GoertzelDFT.m* to computing DTFT samples of the received DTFM signal. Then, *DTFTtoCTFT.m* transfers DTFT to CTFT where *LocalMaximaIdentifier.m* finds the local maxima of the sequence in frequency domain. By calling these funcitons above, *DTMFDetector.m* maps these maxima in frequencies back to the key pressed as indicated in Figure 1.

The intensity (square amplitude) of DTFT and CTFT for the received DTMF signals when '4' and '#' are pressed are shown in Figure 6 and Figure 7 respectively. As is consistent with Equation 8, four peaks exist in each DTFT figure in $[0, 2\pi]$. In CTFT figures, the corresponding frequencies of DTMF signals are easily recognized.

3.4 GUI Design

For better interaction with possible users, a GUI is designed based on MATLAB GUIDE. The main GUI function is *gui.m* and the function for GUI update is implemented in *update_gui.m*. The GUI layout is stored in *gui.fig* and is shown in Figure 8. When the user left-click any of these 16 buttons, the respective DTMF signal will be generated by calling *DTMFGenerator.m*. Then, its time-domain figure will be drawn in the figure box above. After that, *DTMFDetector.m* begins working on the generated signal, showing the key identified in the text box next to the 'Key Pressed' box and plotting the respective CTFT intensity in the figure box below. The results when the user click '2' and '7' are shown in Figure 9 and

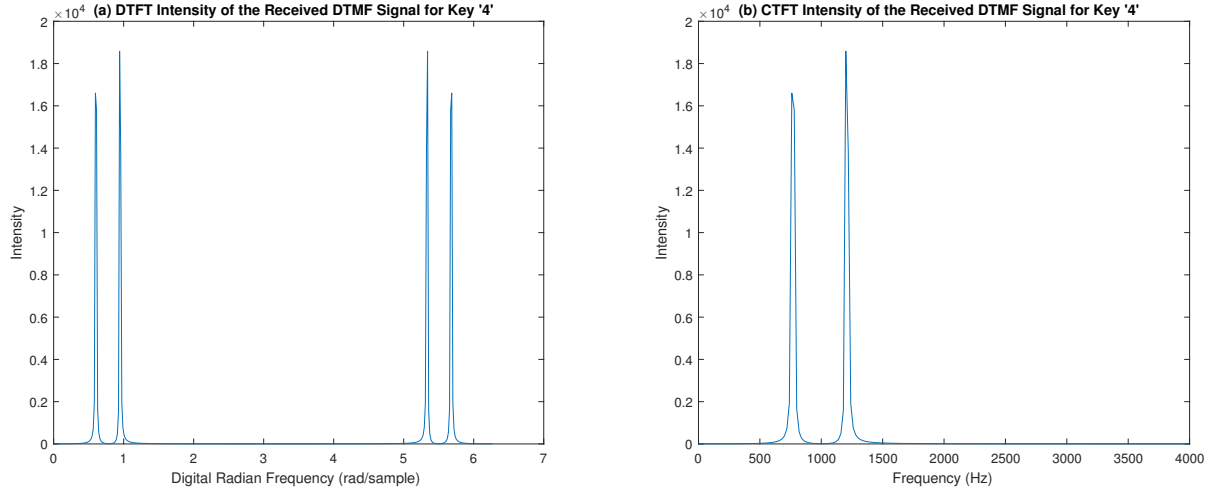

Figure 6: Intensity of **(a)** DTFT and **(b)** CTFT for the received DTMF signals when '4' is pressed.

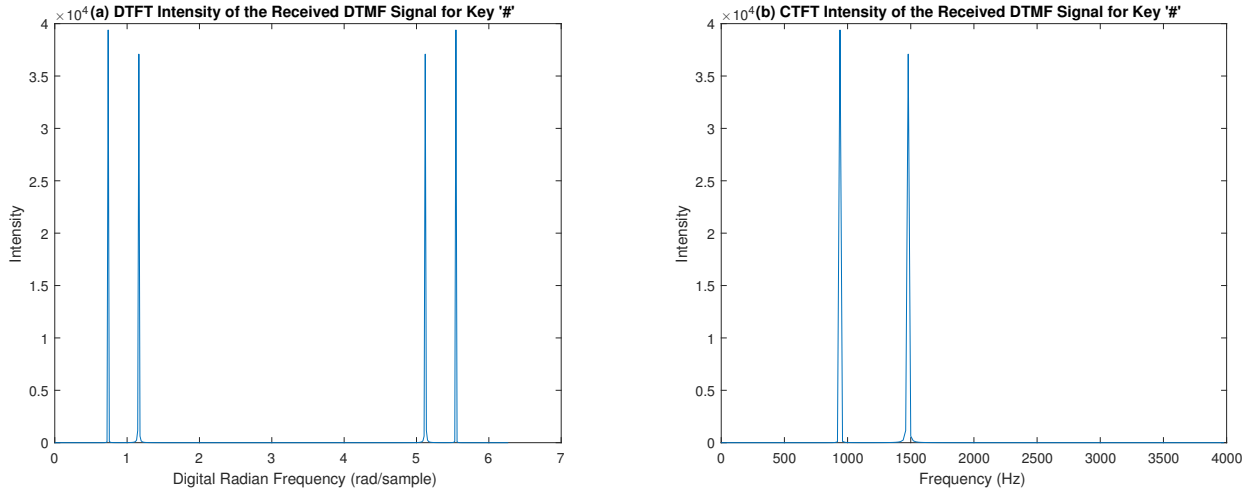

Figure 7: Intensity of **(a)** DTFT and **(b)** CTFT for the received DTMF signals when '#' is pressed.

Figure 10 respectively.

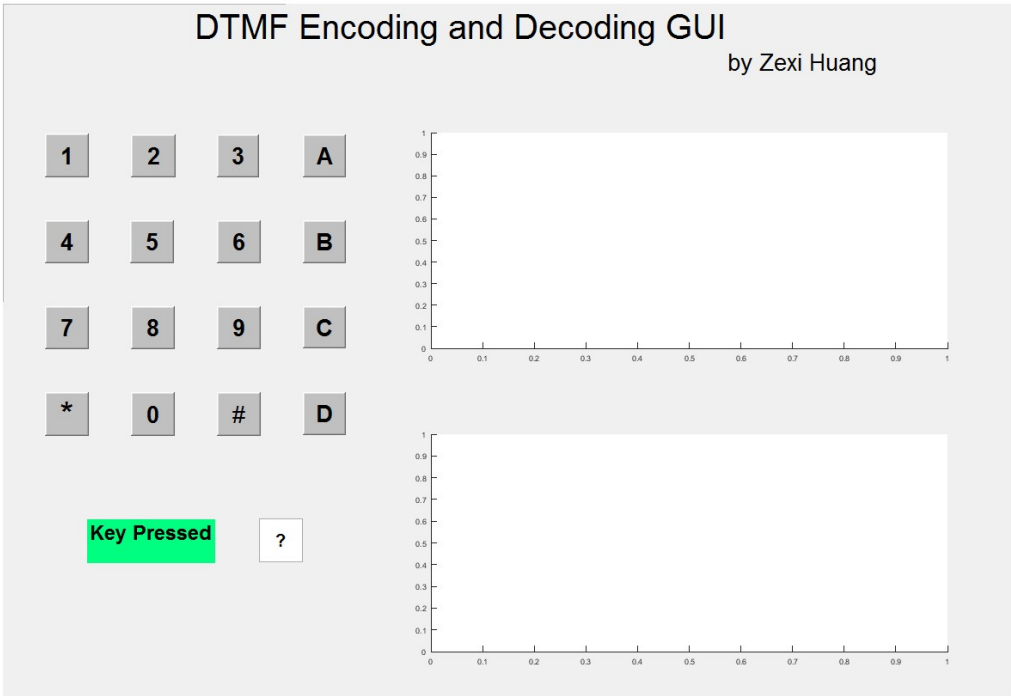

Figure 8: The GUI layout.

4 Concluding Remarks

In this report, we present generation and detection of DTMF signals using MATLAB. The generation process is based on the digital oscillator method while for detection, Goertzel's algorithm, a highly efficient algorithm in this case, is used instead of FFT to obtain DFT samples of the signal. In addition, a GUI presentation is designed to facilitate the illustration of DTMF signals.

References

[1] International Telecommunication Union. Technical features of push-button telephone sets, August 1988. Retrieved from http://www.itu.int/rec/T-REC-Q.23/en.

[2] Gerald Goertzel. An algorithm for the evaluation of finite trigonometric series. *The American Mathematical Monthly*, 65(1):34–35, 1958.

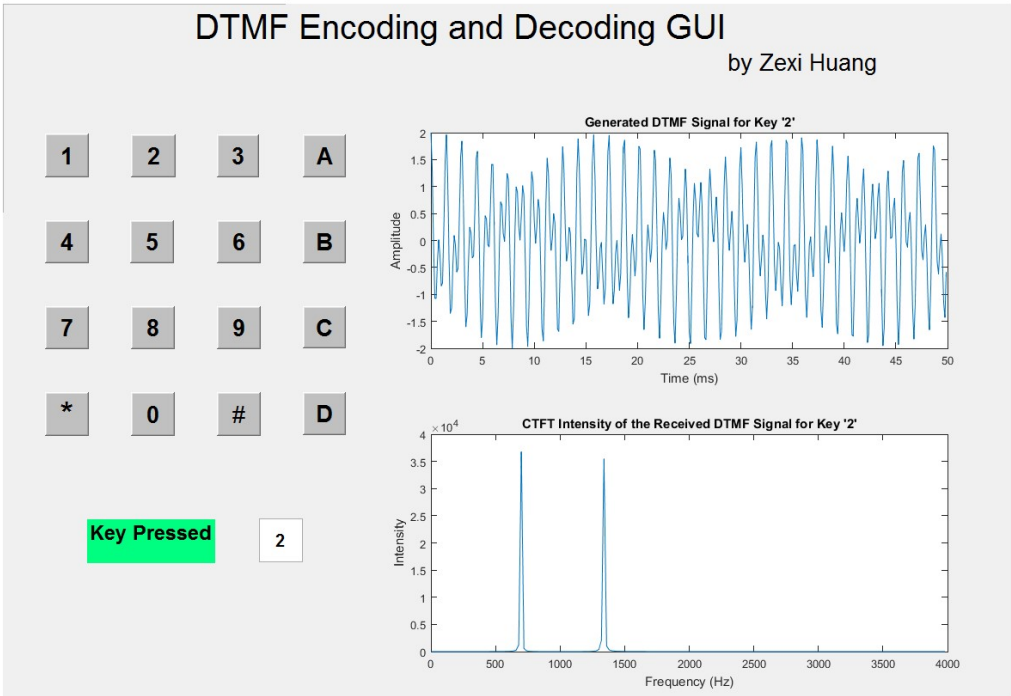

Figure 9: The GUI presentation when '2' is clicked.

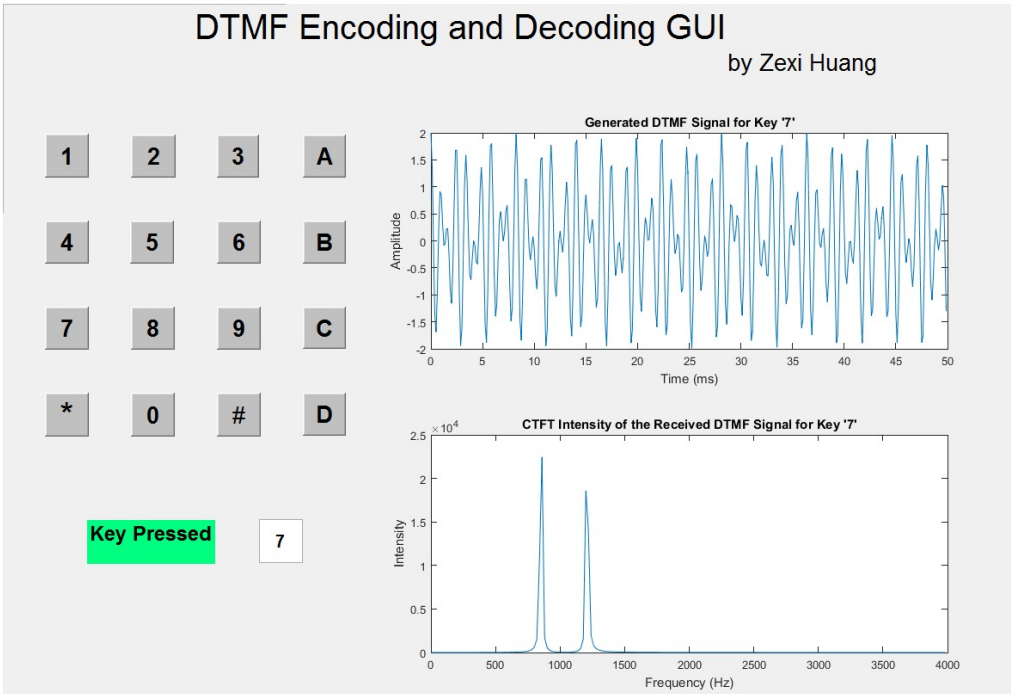

Figure 10: The GUI presentation when '7' is clicked.

A Appendix

A.1 Source Codes for DTMF Generation

- *CosineGenerator.m.*

```
%{
%Generating n0=400 samples of cosine sequence given digital radian
    frequency w.
%Zexi Huang
%Dec. 14 2016
%}

function h=CosineGenerator(w,n0)
%h: output, n0=400 samples of the required cosine sequence.
%p: digital radian frequency of the cosine sequence.


C=cos(w);
%Store the constant.
h=zeros(n0,1)';
%Initiate h.


h(1)=1;
%Note that it in fact computes h[0], that is, h[n]=h(n+1) since MATLAB
%don't allow zero index.
h(2)=C;
%Initial condition computed from h(-2)=h(-1)=0

for ii=3:1:n0
    h(ii)=2*C*h(ii-1)-h(ii-2);
end
%Loop implementation of recursive evaluation.

end
```

- *DTMFGenerator.m.*

```
%{
%Generating a DTMF sequence with specified key pressed based on digital
oscillator.
%Zexi Huang
%Dec. 14 2016
%}

function [x,t]=DTMFGenerator(key)
%key: input key.
%[x,t]: output respective DTMF sequence.

fs=8000;
n0=400;

%List required frequencies.
```

```
fl=[697, 770, 852, 941];
fh=[1209, 1336, 1477, 1633];

%Generate time stamps.
t=0:1/8:50-1/8;


%Establish mapping relations.
switch key
    case '1'
        wl=fl(1);
        wh=fh(1);
    case '2'
        wl=fl(1);
        wh=fh(2);
    case '3'
        wl=fl(1);
        wh=fh(3);
    case 'A'
        wl=fl(1);
        wh=fh(4);
    case '4'
        wl=fl(2);
        wh=fh(1);
    case '5'
        wl=fl(2);
        wh=fh(2);
    case '6'
        wl=fl(2);
        wh=fh(3);
    case 'B'
        wl=fl(2);
        wh=fh(4);
    case '7'
        wl=fl(3);
        wh=fh(1);
    case '8'
        wl=fl(3);
        wh=fh(2);
    case '9'
        wl=fl(3);
        wh=fh(3);
    case 'C'
        wl=fl(3);
        wh=fh(4);
    case '*'
        wl=fl(4);
        wh=fh(1);
    case '0'
        wl=fl(4);
        wh=fh(2);
    case '#'
        wl=fl(4);
        wh=fh(3);
```

```
    case 'D'
        wl=fl(4);
        wh=fh(4);
end

wl=2*pi*wl/fs;
wh=2*pi*wh/fs;

%Calling digital oscillator based cosine generator.
x=CosineGenerator(wl,n0)+CosineGenerator(wh,n0);

sound(x,fs);
end
```

A.2 Source Codes for DTMF Detection

- *GoertzelDFT.m.*

```
%{
%Goertzel's Algorithm to compute DFT for given sequence.
%Zexi Huang
%Dec. 14 2016
%}

function [X,w]=GoertzelDFT(x)
%[X,w]: output DTFT samples.
%x: input sequence.


%N: Number of samples.
N=400;

%Generate frequency stamps.
w=0:2*pi/N:2*pi*(1-1/N);

%Initiate X.
X=zeros(N,1)';

%Generate X[k].
for k=0:1:N-1
    %Store the constant.
    C=2*cos(2*pi*k/N);
    %Initiate vk.
    vk=zeros(N+1,1)';

    %vk[0], vk[1] computed from vk[-2], vk[-1].
    vk(1)=x(1);
    vk(2)=x(2)+C*vk(1);

    for ii=3:1:N
        vk(ii)=x(ii)+C*vk(ii-1)-vk(ii-2);
    end
    vk(N+1)=C*vk(N)-vk(N-1);
```

```
    %In fact X(k).
    X(k+1)=vk(N+1)^2+vk(N)^2-C*vk(N+1)*vk(N);
end
```

- *DTFTtoCTFT.m.*

```
%{
%Transfer DTFT back to CTFT samples for DTMF.
%Zexi Huang
%Dec. 14 2016
%}
function [X_CT,w_CT]=DTFTtoCTFT(X,w)
%[X,w]: DTFT of the DTMF signal.
%[X_CT,w_CT]: CTFT samples of the DTMF signal.

%Same amplitude domain.
X_CT=X(1:200);

%Mapping digital radian frequency into analog frequency.
w_CT=w(1:200);
w_CT=w_CT*8000/(2*pi);
```

- *LocalMaximaIdentifier.m.*

```
%{
%Finding local maxima for a given sequence.
%Zexi Huang
%Dec. 15 2016
%}
function [maxX,maxw]=LocalMaximaIdentifier(X,w)
%[X,w]: the sequence whose local maxima are to be identified.
%[maxX,maxw]: the output sets of local maximas.


N=length(X);
jj=1;

%Find local maxima accroding to its definition.
for ii=2:N-1
    if X(ii)>X(ii-1) &&  X(ii)>X(ii+1)
        maxX(jj)=X(ii);
        maxw(jj)=w(ii);
        jj=jj+1;
    end
end
```

- *DTMFDetector.m.*

```
%{
%Identify the respective key for a given DTMF signal.
%Zexi Huang
%Dec. 15 2016
%}

```

```
function key=DTMFDetector(x,t)
%[x,t]: input DTMF sequence.
%key: the key pressed identified.

%Computing DTFT of the sequence based on Goertzel's algorithm.
[X,w]=GoertzelDFT(x);
%Mapping DTFT to CTFT samples.
[X,w]=DTFTtoCTFT(X,w);
%Identify local maxima of the sequence.
[maxX,maxw]=LocalMaximaIdentifier(X,w);

%Identified fl and fh.
fl0=maxw(1);
fh0=maxw(2);

%fl and fh sets.
fl=[697, 770, 852, 941];
fh=[1209, 1336, 1477, 1633];

%Difference between identified fl, fh and fl, fh sets.
fl0=fl0*ones(1,4);
fh0=fh0*ones(1,4);
fl_diff=abs(fl-fl0);
fh_diff=abs(fh-fh0);

%Mapping identified frequency back to key pressed.
if min(fl_diff)==fl_diff(1)
    if min(fh_diff)==fh_diff(1)
        key='1';
    elseif min(fh_diff)==fh_diff(2)
        key='2';
    elseif min(fh_diff)==fh_diff(3)
        key='3';
    elseif min(fh_diff)==fh_diff(4)
        key='A';
    end
elseif min(fl_diff)==fl_diff(2)
    if min(fh_diff)==fh_diff(1)
        key='4';
    elseif min(fh_diff)==fh_diff(2)
        key='5';
    elseif min(fh_diff)==fh_diff(3)
        key='6';
    elseif min(fh_diff)==fh_diff(4)
        key='B';
    end
elseif min(fl_diff)==fl_diff(3)
    if min(fh_diff)==fh_diff(1)
        key='7';
    elseif min(fh_diff)==fh_diff(2)
        key='8';
    elseif min(fh_diff)==fh_diff(3)
        key='9';
    elseif min(fh_diff)==fh_diff(4)
```

```
        key='C';
    end
elseif min(fl_diff)==fl_diff(4)
    if min(fh_diff)==fh_diff(1)
        key='*';
    elseif min(fh_diff)==fh_diff(2)
        key='0';
    elseif min(fh_diff)==fh_diff(3)
        key='#';
    elseif min(fh_diff)==fh_diff(4)
        key='D';
    end
end
```

A.3 Source Codes for GUI Design

- *gui.m.*

```

%{
%GUI main function for the GUI design for DTMF.
%Zexi Huang
%Dec. 16 2016
%}
function varargout = gui(varargin)
% GUI M-file for gui.fig
%      GUI, by itself, creates a new GUI or raises the existing
%      singleton*.
%
%      H = GUI returns the handle to a new GUI or the handle to
%      the existing singleton*.
%
%      GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI.M with the given input arguments.
%
%      GUI('Property','Value',...) creates a new GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs
     are
%      applied to the GUI before gui_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
    application
%      stop.  All inputs are passed to gui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
    one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help gui


% Begin initialization code - DO NOT EDIT
```

```
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui_OpeningFcn, ...
                   'gui_OutputFcn',  @gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before gui is made visible.
function gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to gui (see VARARGIN)

% Choose default command line output for gui
handles.output = hObject;


% Update handles structure
guidata(hObject, handles);

% UIWAIT makes gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = gui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in b1.
function b1_Callback(hObject, eventdata, handles)
% hObject    handle to b1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
87 [x,t]=DTMFGenerator('1');
88 key='1';
89 update_gui
90
91 % --- Executes on button press in b2.
92 function b2_Callback(hObject, eventdata, handles)
93 % hObject    handle to b2 (see GCBO)
94 % eventdata  reserved - to be defined in a future version of MATLAB
95 % handles    structure with handles and user data (see GUIDATA)
96 [x,t]=DTMFGenerator('2');
97 key='2';
98 update_gui
99
100 % --- Executes on button press in b3.
101 function b3_Callback(hObject, eventdata, handles)
102 % hObject    handle to b3 (see GCBO)
103 % eventdata  reserved - to be defined in a future version of MATLAB
104 % handles    structure with handles and user data (see GUIDATA)
105 [x,t]=DTMFGenerator('3');
106 key='3';
107 update_gui
108 % --- Executes on button press in b4.
109 function b4_Callback(hObject, eventdata, handles)
110 % hObject    handle to b4 (see GCBO)
111 % eventdata  reserved - to be defined in a future version of MATLAB
112 % handles    structure with handles and user data (see GUIDATA)
113 [x,t]=DTMFGenerator('4');
114 key='4';
115 update_gui
116 % --- Executes on button press in b5.
117 function b5_Callback(hObject, eventdata, handles)
118 % hObject    handle to b5 (see GCBO)
119 % eventdata  reserved - to be defined in a future version of MATLAB
120 % handles    structure with handles and user data (see GUIDATA)
121 [x,t]=DTMFGenerator('5');
122 key='5';
123 update_gui
124 % --- Executes on button press in b6.
125 function b6_Callback(hObject, eventdata, handles)
126 % hObject    handle to b6 (see GCBO)
127 % eventdata  reserved - to be defined in a future version of MATLAB
128 % handles    structure with handles and user data (see GUIDATA)
129 [x,t]=DTMFGenerator('6');
130 key='6';
131 update_gui
132 %--- Executes on button press in b7.
133 function b7_Callback(hObject, eventdata, handles)
134 % hObject    handle to b7 (see GCBO)
135 % eventdata  reserved - to be defined in a future version of MATLAB
136 % handles    structure with handles and user data (see GUIDATA)
137 [x,t]=DTMFGenerator('7');
138 key='7';
139 update_gui
140
```

```
141 % --- Executes on button press in b8.
142 function b8_Callback(hObject, eventdata, handles)
143 % hObject    handle to b8 (see GCBO)
144 % eventdata  reserved - to be defined in a future version of MATLAB
145 % handles    structure with handles and user data (see GUIDATA)
146 [x,t]=DTMFGenerator('8');
147 key='8';
148 update_gui
149
150 % --- Executes on button press in b9.
151 function b9_Callback(hObject, eventdata, handles)
152 % hObject    handle to b9 (see GCBO)
153 % eventdata  reserved - to be defined in a future version of MATLAB
154 % handles    structure with handles and user data (see GUIDATA)
155 [x,t]=DTMFGenerator('9');
156 key='9';
157 update_gui
158
159 % --- Executes on button press in ba.
160 function bstar_Callback(hObject, eventdata, handles)
161 % hObject    handle to bstar (see GCBO)
162 % eventdata  reserved - to be defined in a future version of MATLAB
163 % handles    structure with handles and user data (see GUIDATA)
164 [x,t]=DTMFGenerator('*');
165 key='*';
166 update_gui
167
168 % --- Executes on button press in b0.
169 function b0_Callback(hObject, eventdata, handles)
170 % hObject    handle to b0 (see GCBO)
171 % eventdata  reserved - to be defined in a future version of MATLAB
172 % handles    structure with handles and user data (see GUIDATA)
173 [x,t]=DTMFGenerator('0');
174 key='0';
175 update_gui
176
177 % --- Executes on button press in bn.
178 function bcell_Callback(hObject, eventdata, handles)
179 % hObject    handle to bcell (see GCBO)
180 % eventdata  reserved - to be defined in a future version of MATLAB
181 % handles    structure with handles and user data (see GUIDATA)
182 [x,t]=DTMFGenerator('#');
183 key='#';
184 update_gui
185
186 function bA_Callback(hObject, eventdata, handles)
187 % hObject    handle to bcell (see GCBO)
188 % eventdata  reserved - to be defined in a future version of MATLAB
189 % handles    structure with handles and user data (see GUIDATA)
190 [x,t]=DTMFGenerator('A');
191 key='A';
192 update_gui
193
194 function bB_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to bcell (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[x,t]=DTMFGenerator('B');
key='B';
update_gui

function bC_Callback(hObject, eventdata, handles)
% hObject    handle to bcell (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[x,t]=DTMFGenerator('C');
key='C';
update_gui

function bD_Callback(hObject, eventdata, handles)
% hObject    handle to bcell (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[x,t]=DTMFGenerator('D');
key='D';
update_gui


% --- Executes on mouse press over axes background.
function fig1_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to fig1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)




% --- Executes during object creation, after setting all properties.
function EditPressed_CreateFcn(hObject, eventdata, handles)
% hObject    handle to EditPressed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'
        ));
end


function EditPressed_Callback(hObject, eventdata, handles)
% hObject    handle to EditPressed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
247 % handles    structure with handles and user data (see GUIDATA)
248
249 % Hints: get(hObject,'String') returns contents of EditPressed as text
250 %        str2double(get(hObject,'String')) returns contents of
    EditPressed as a double
```

- *update_gui.m.*

```
1
2 %{
3 %GUI update for the GUI design for DTMF.
4 %Zexi Huang
5 %Dec. 16 2016
6 %}
7
8
9
10
11 % For Figure1
12
13 axes(handles.fig1);
14 PlotTimeDomain(x,t,key);
15
16 %For key pressed window.
17 keyPressed=DTMFDetector(x,t);
18 set(handles.EditPressed,'String',keyPressed);
19
20 % For Figure2
21
22 axes(handles.fig2);
23 [X,w]=GoertzelDFT(x);
24 [X,w]=DTFTtoCTFT(X,w);
25
26 plot(w,X);
27 xlabel('Frequency (Hz)');
28 ylabel('Intensity');
29 title(['CTFT Intensity of the Received DTMF Signal for Key ''',keyPressed
    ,''''']);
```

- *gui.fig.* The generated *m*-file for the layout is to long (over 1000 lines) and thus is omitted here.