EC39006 – Digital Signal Processing (DSP) Laboratory

Lab Report

Name: *Rekha Lokesh*

Roll No: *19EC10052*

Group: *3*

Expt3 – DTMF (Dual Tone Multifrequency or TouchTone) coder/decoder

---

Aim: In this experiment, we are going to discuss about:

- Using ***DTMF technique to encode the keys*** {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D}.
- Design a ***bandpass FIR filter bank*** and ***decode the keys*** from the signal sent and analyze the noise performance and filtering.

Theory:

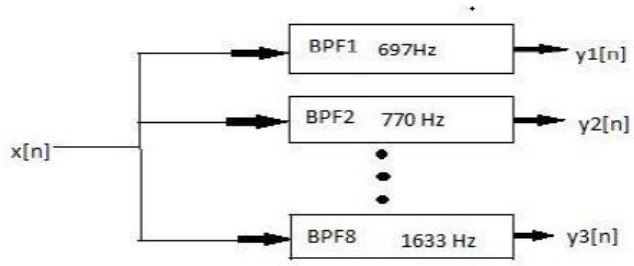
- Dual-tone multi-frequency signalling (***DTMF***) is a ***telecommunication signalling system*** using the voice-frequency band over telephone lines between telephone equipment and other communications devices and switching centres.
- The landline communication channel used in ***primitive telephone communication***, was ***analog*** in nature. For establishing a proper communication between the caller and the callee, the telephone exchange office had ***to decode the dialled digital number*** from analog information. Thus DTMF (Dual to Multi-Frequency) was used. If any key is dialled, a dual tone sinusoidal signal containing ***two distinct frequencies below and above 1000Hz*** is generated and transmitted over the telephone channel.
- The ***DTMF Encoding Scheme*** used for the keypad:

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

- Thus, based on the sequence of keys pressed, ***sinusoidal signals with frequencies corresponding to the keys pressed***, length of each bit or time segment being noted is ***generated***.

Decoding the DTMF Signals:

- ***FIR filter bank*** is used to ***decode DTMF signals***. The filter bank consists of ***8 BPFs***, each of which are designed to pass only one frequency component among the DTMF frequencies. ***The input signal for all filters is the same DTMF signal***.
- For a particular time-length L (which is priorly known in this case), when input to the filter bank is DTMF signal, ***two outputs will have relatively higher magnitude of frequencies from all the BPFs*** or in case of noise corrupted signals, two cases have dominant rms values of the output from the filter bank and ***those frequencies*** are ***used as row & column pointers*** to determine the ***key from DTMF code***. A filter bank consisting of 8 BPFs is shown below:



- ***Bandpass Filter Design:***

  The L-point average filter is a low pass filter with bandwidth inversely proportional to L (length of the filter). The BPF with L-points, gain in pass band and centre frequency w_c is defined by:

$$h[n] = \beta \cos(\omega_c n) \quad \text{where } 0 \le n < L$$

  β is chosen such that maximum value of the frequency response magnitude will be one.

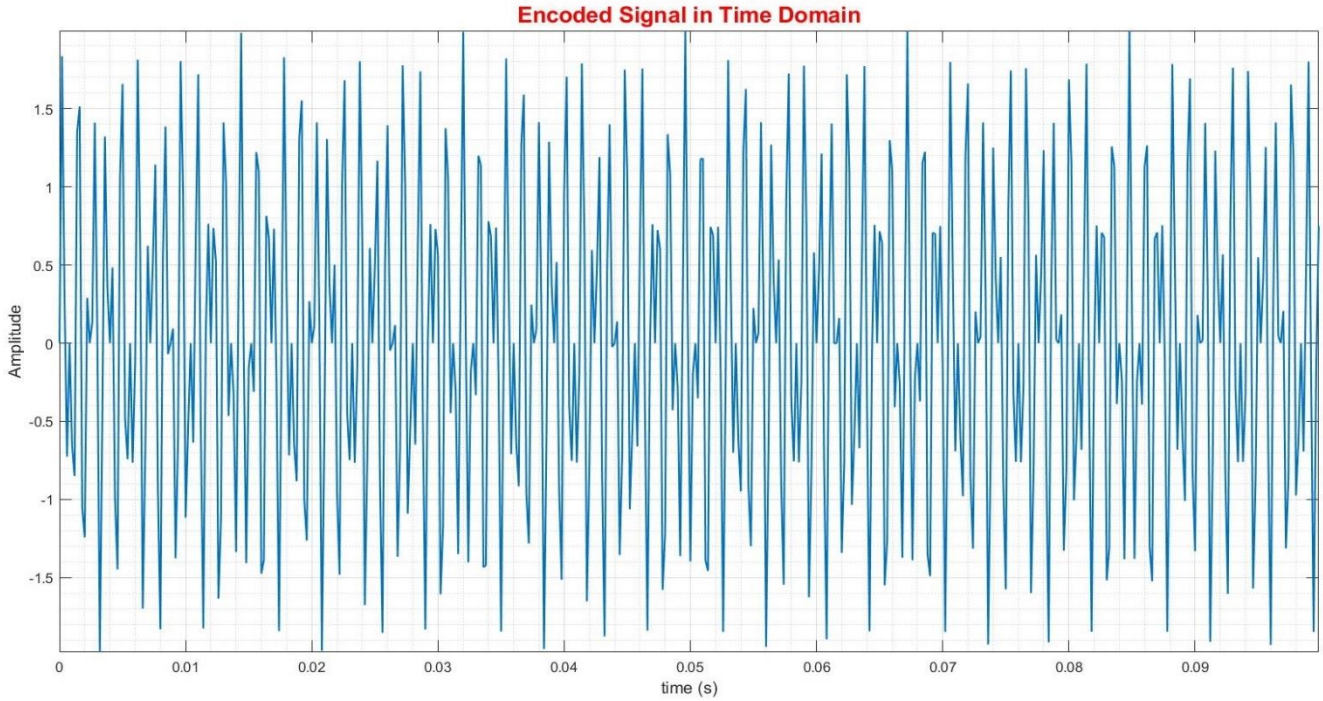
Observations:

Encoding:

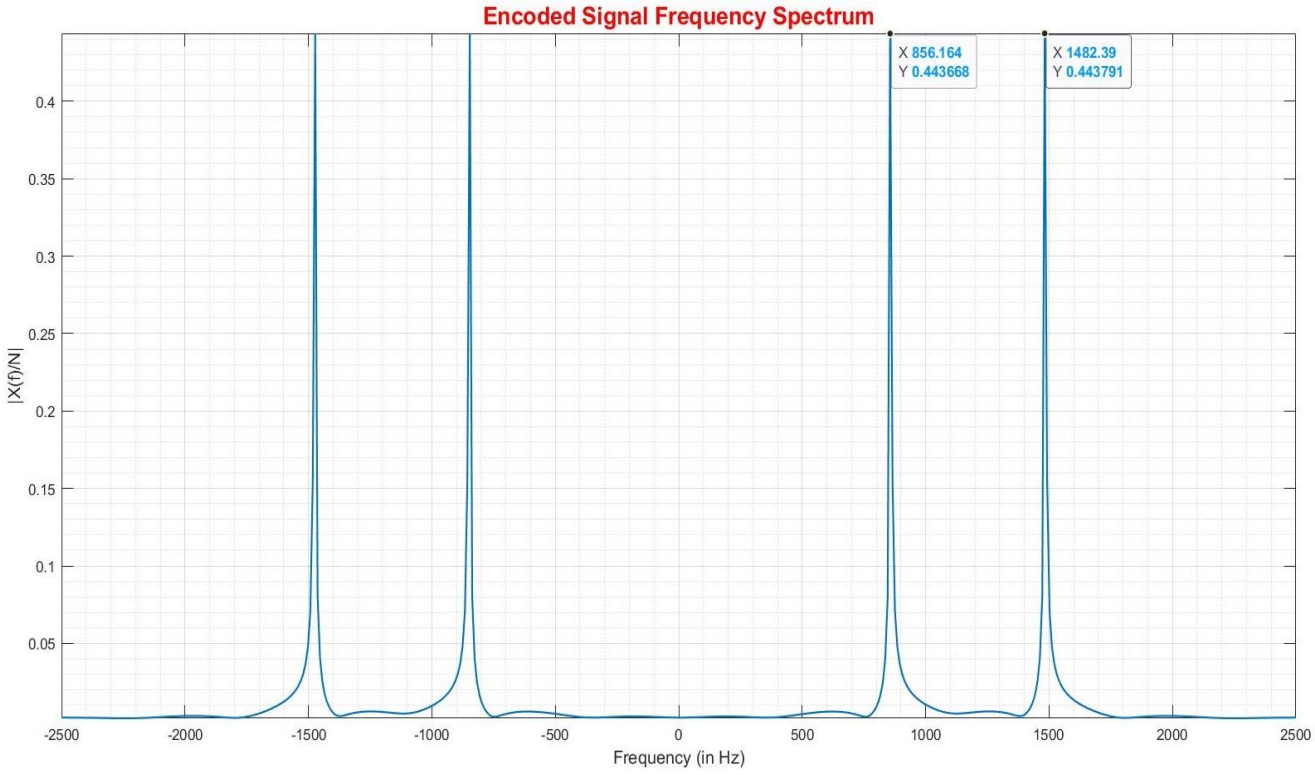
- In this experiment, input is taken from the user. For a ***single key***, ***A list dialog box*** command is used to take input from the user to choose the key from a keypad. Since the list dialog box only takes single input, ***console input is used to take a sequence of keys as input***. The key pressed has a ***combination to two unique frequencies using which a pure sinusoidal signal is formed***. This is done using ***map containers*** in MATLAB in which each key is associated with a row frequency and column frequency. Thus, using the input from the user, a corresponding unique signal is generated at the source end (***encoding***).

Single Key Encoding of Key '9':

Time Domain Representation of the Encoded Signal sampled at 5kHz

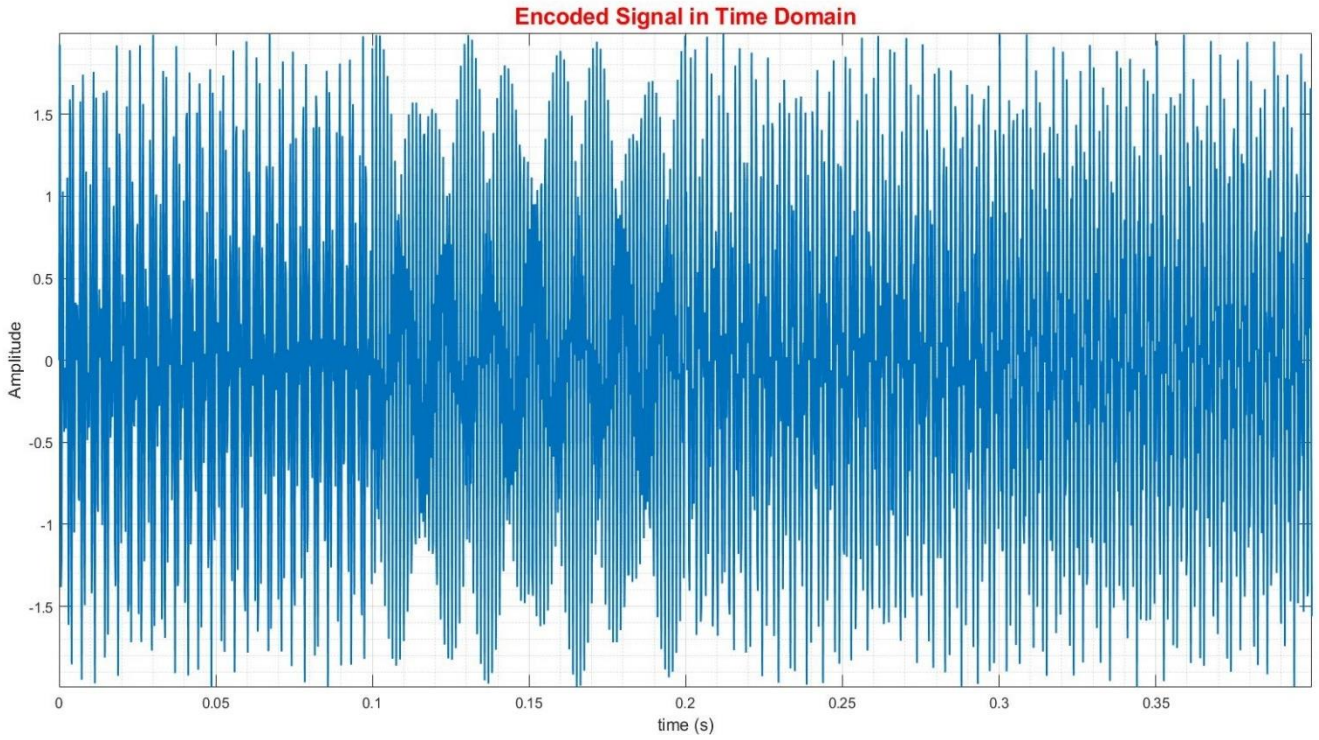


Frequency Domain Representation of the Encoded Signal sampled at 5kHz

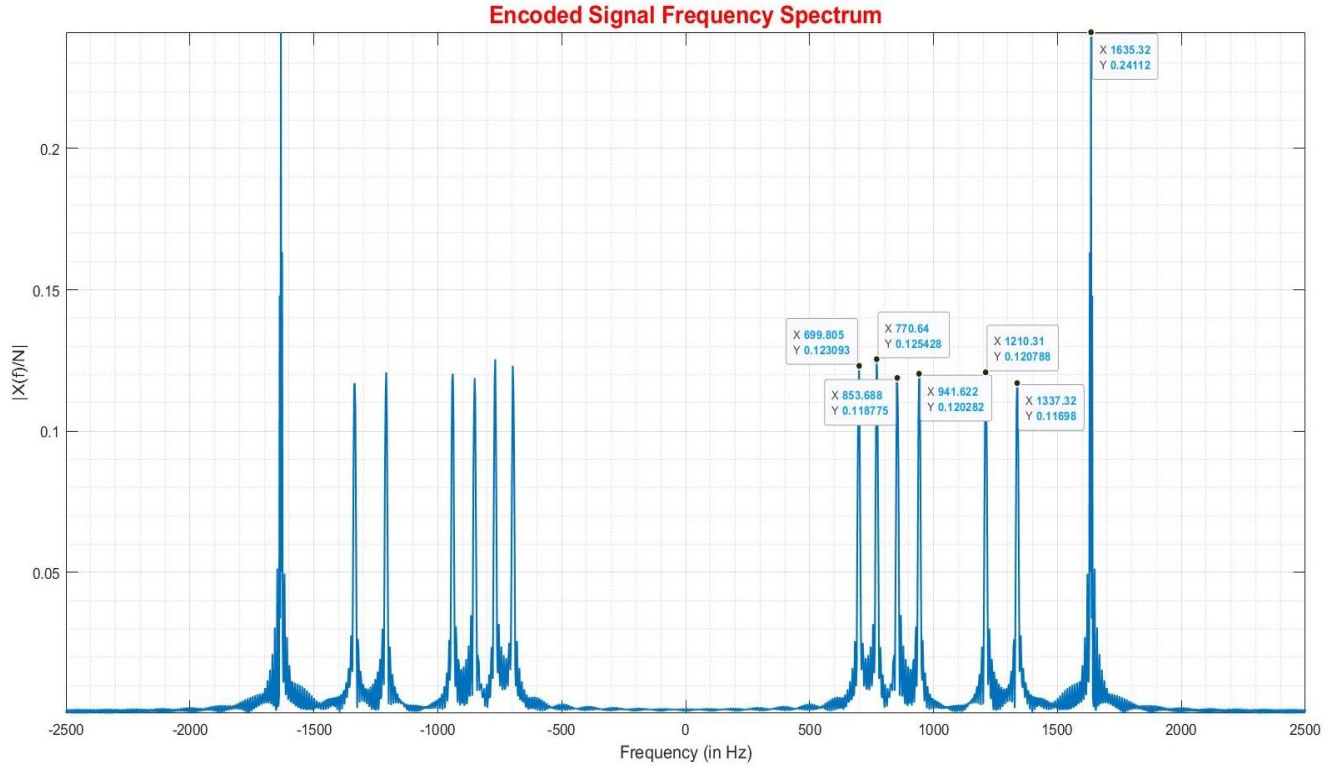


*Multiple Key Encoding of Key '*C5A':*

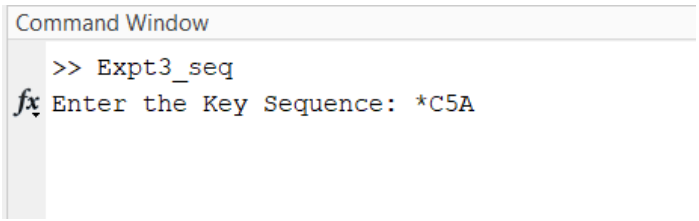
Time Domain Representation of the Encoded Signal sampled at 5kHz



Frequency Domain Representation of the Encoded Signal sampled at 5kHz



- In the above ***encoding scheme***, for each key pressed, a signal constituting of its corresponding frequencies is generated which had a ***time length of L = 0.1 seconds***. The ***sampling frequency*** chosen is ***5 kHz*** and number of ***sample points per bit (key) are 512***.
- In the Single Key Encoding, the frequency corresponding to the keys can be observed from its ***frequency spectrum***.
- The ***Sampling frequency*** is chosen such that it is ***greater than the twice of the highest frequency of the encoding scheme*** and it has a trade off to computational time in its higher end. Since frequency axis in frequency spectrum is linspace (-0.5, 0.5, N) × Fs.
- Thus, in the multiple sequence case, *C5A is given as input



MATLAB Code:

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

keyval = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '*', '0', '#',
'A', 'B', 'C', 'D'};
rowval = [697, 697, 697, 770, 770, 770, 852, 852, 852, 941, 941, 941,
697, 770, 852, 941];
colval = [1209, 1336, 1477, 1209, 1336, 1477, 1209, 1336, 1477, 1209,
1336, 1477, 1633, 1633, 1633, 1633];

row_freq = containers.Map(keyval, rowval);     %key to row freq map
col_freq = containers.Map(keyval, colval);     %key to col freq map

% inp = takekey(); %Taking the key input and the user must enter a single
key only

inp = input("Enter the Key Sequence: ",'s');     %For multiple key input

% disp(inp);

Fs = 5e3; %sampling frequency
L = 0.1;     %Time Duration of individual bit in seconds
N = 512; %Sample Points
noise_amp = 0.5; %Noise amplitude

x_t = [];
for i=1:length(inp)
    f1 = row_freq(inp(i));
    f2 = col_freq(inp(i));

    t = ((i-1)*L):(1/Fs):((i*L)-(1/Fs));
```

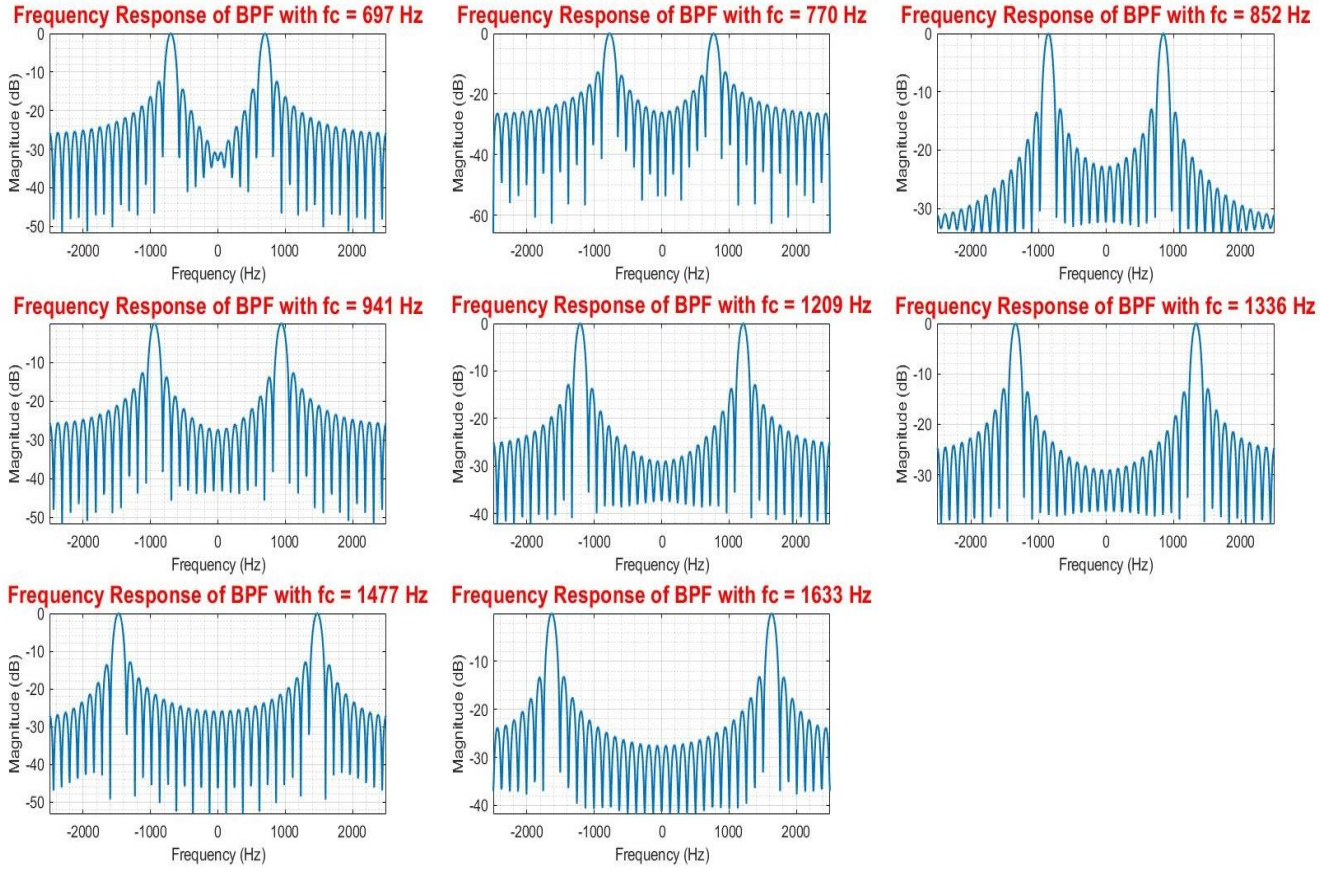
```
    x1_t = sin(2*pi*f1*t) + sin(2*pi*f2*t);
    x_t = cat(2, x_t, x1_t);  %Input signal
end

N1 = N*length(inp);
X_w = fftshift(fft(x_t, N1))/N1; %fft sprectrum shifting and
normalization
f = linspace(-1/2,1/2,N1)*(Fs);

%Time Domain Response of input signal
figure("Name", "Encoded signal in time domain");
plot((0:(1/Fs):((length(inp)*L)-1/Fs)), x_t(1:(length(inp)*L*Fs)),
"LineWidth", 1.25);
hold on;
xlabel('time (s)', "fontsize", 12);
ylabel('Amplitude', "fontsize", 12);
title("Encoded Signal in Time Domain", "fontsize", 16, "Color", 'r');
% legend("Encoded Signal");
axis tight;
grid on;
grid minor;
hold off;

%     %Freq Spectrum of Input signal
figure("Name", "Encoded signal in frequency domain");
plot(f, abs(X_w), "LineWidth", 1.25);
xlabel('Frequency (in Hz)', "fontsize", 12);
ylabel('|X(f)/N|', "fontsize", 12);
title("Encoded Signal Frequency Spectrum","fontsize", 16, "Color", 'r');
% legend("Encoded Signal");
hold on;
axis tight;
grid on;
grid minor;

res = decoded_key(x_t, L, freqvec, Fs, N, noise_amp);

if(res==inp)    disp("CORRECT! Your chosen key is found to be "+res);
else disp("INCORRECT! Your chosen key is "+inp+" and the res is "+res);
end

% function str = takekey()
% %     liststr = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '*',
'#', 'A', 'B', 'C', 'D'};
% %     [indx, tf] = listdlg('ListString', liststr, "PromptString",
"Enter the Keys: ", 'OKString','Apply', 'ListSize',[150,250],
'Name','KEYPAD', 'CancelString','No Selection');
% % %     if (tf == 0)  str = takekey(); %Take input again since the user
pressed 'cancel' or 'esc'
% % %     else
% %         disp(indx);
% %         str = {};
% %         for i=1:length(indx)
% %             str{i} = liststr{indx(i)};
% %         end
% % %     end
% %     disp(str);
% end
```

Band Pass Filter Design:

- The Band Pass filter is designed according to the filter response:

$$h[n] = \beta \cos(\omega_c n) \quad \text{where } 0 \le n < L$$

- The filter bank is a group of filters with each band pass filter having central frequency corresponding to each frequency in the DTMF encoding scheme.



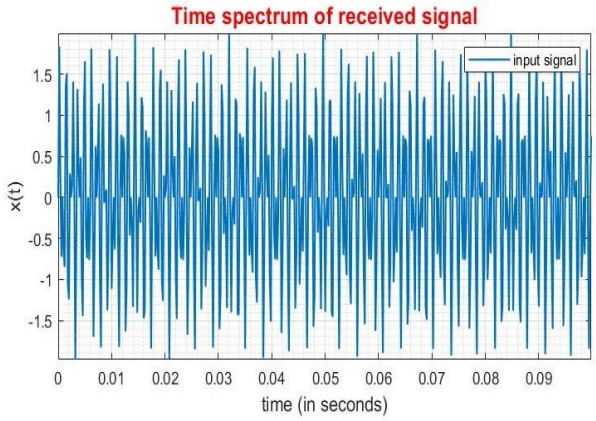
- Here the ***length of the filter*** is taken as ***40 i.e., (40/Fs) seconds***.
- For ***L < 5***, most of the ***decoding sequences are coming out to be wrong***.
- As the ***filter length increases***, the ***filter bandwidth decreases***, ***increasing the filter resolution*** and ***decoding accuracy***. With lower filter length, the bandwidth will be larger which allows nearby frequencies to pass increasing the rms value or higher frequency response magnitude which may lead to incorrect decoding. In practical scenarios, ***having large L increases resolution*** requiring ***more resources*** and hence apparatus cost may increase.
- The value of **β** is chosen such that ***peak frequency spectrum magnitude of the filter is 1***. This is because ***pass band filter gain has to be 1 (or constant) for all the filter***s such that when an input signal is filtered, the ***peak frequency components*** can be ***identified by comparing the output peak magnitudes***. Thus, it helps in efficient decoding.

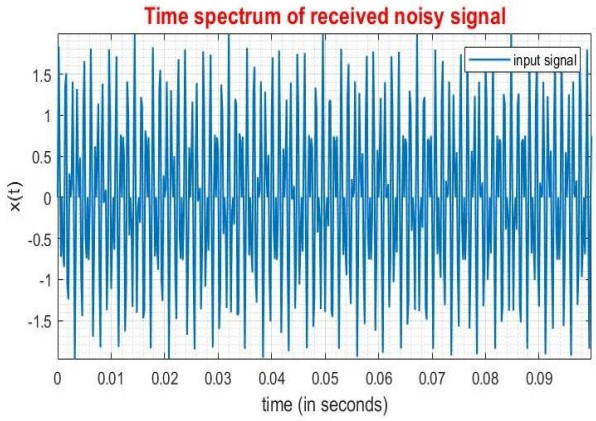
Decoding:

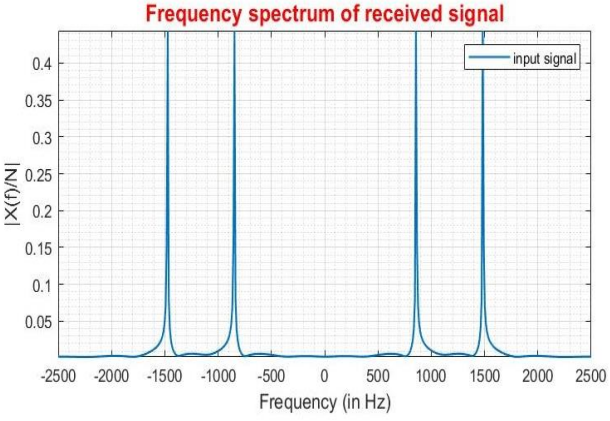
- The Encoded Signal passes through channel and reaches to the decoding side. The signal might be containing various keys encoded. In our experiment, the ***time segment length of each bit*** (or key) is ***priorly known*** on the decoding side.
- Hence, the ***received signal is divided into segments of a particular time length*** and each time segment is analyzed to find out the frequencies present in it using the filter bank discussed above and the ***top two frequencies dominantly present*** are used as ***row and column pointers to decode the key***.
- Since sometimes the signal while passing through the channel gets corrupted with noise, so ***noise analysis is also done on the decoding scheme***. Random noise with some noise amplitude is added on the decoding side in one case.

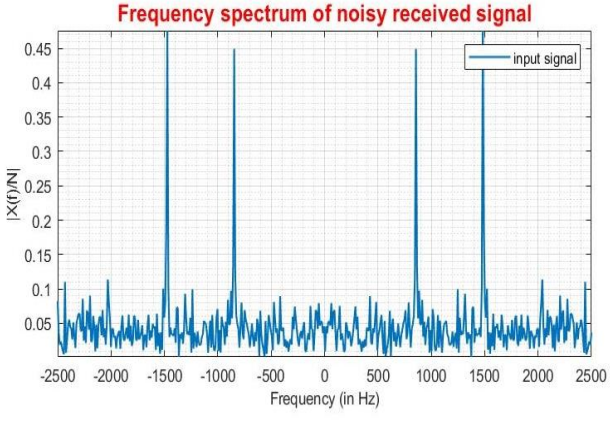
Single Key Decoding:

Received Signals at Decoding Side

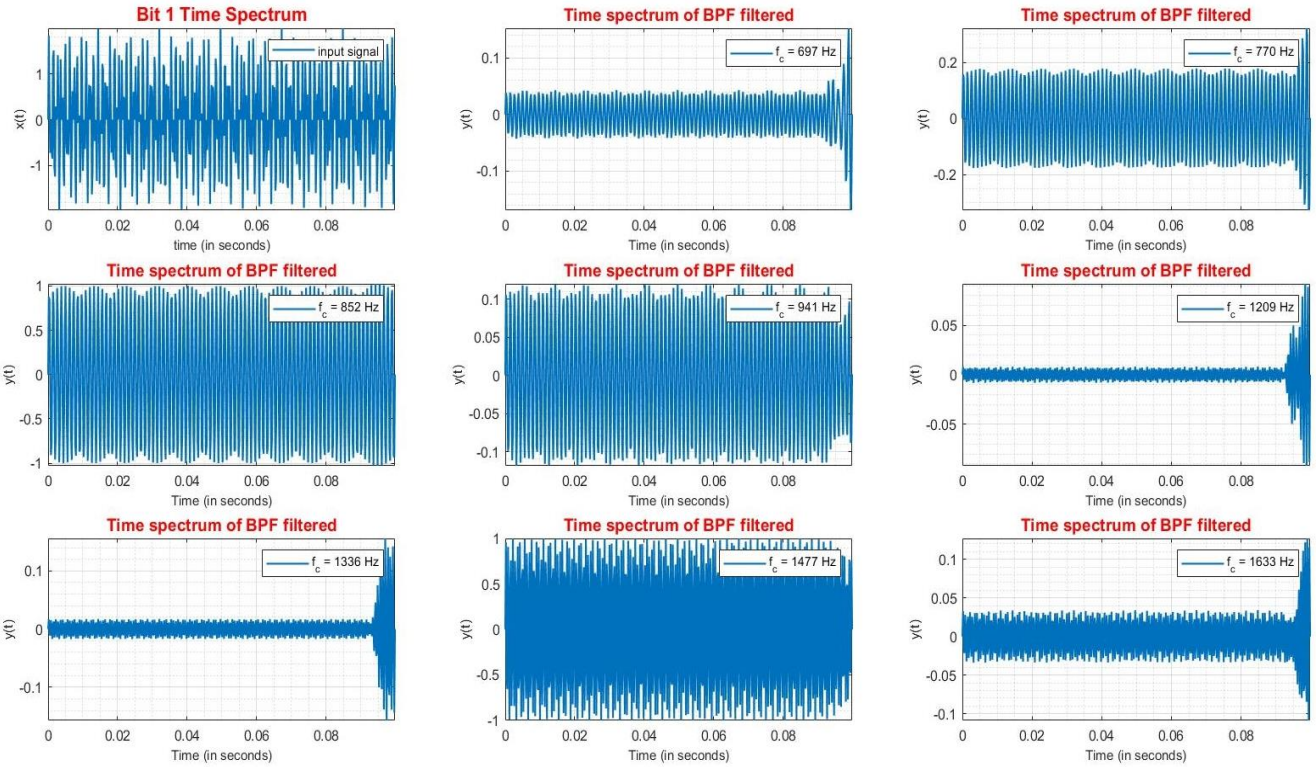




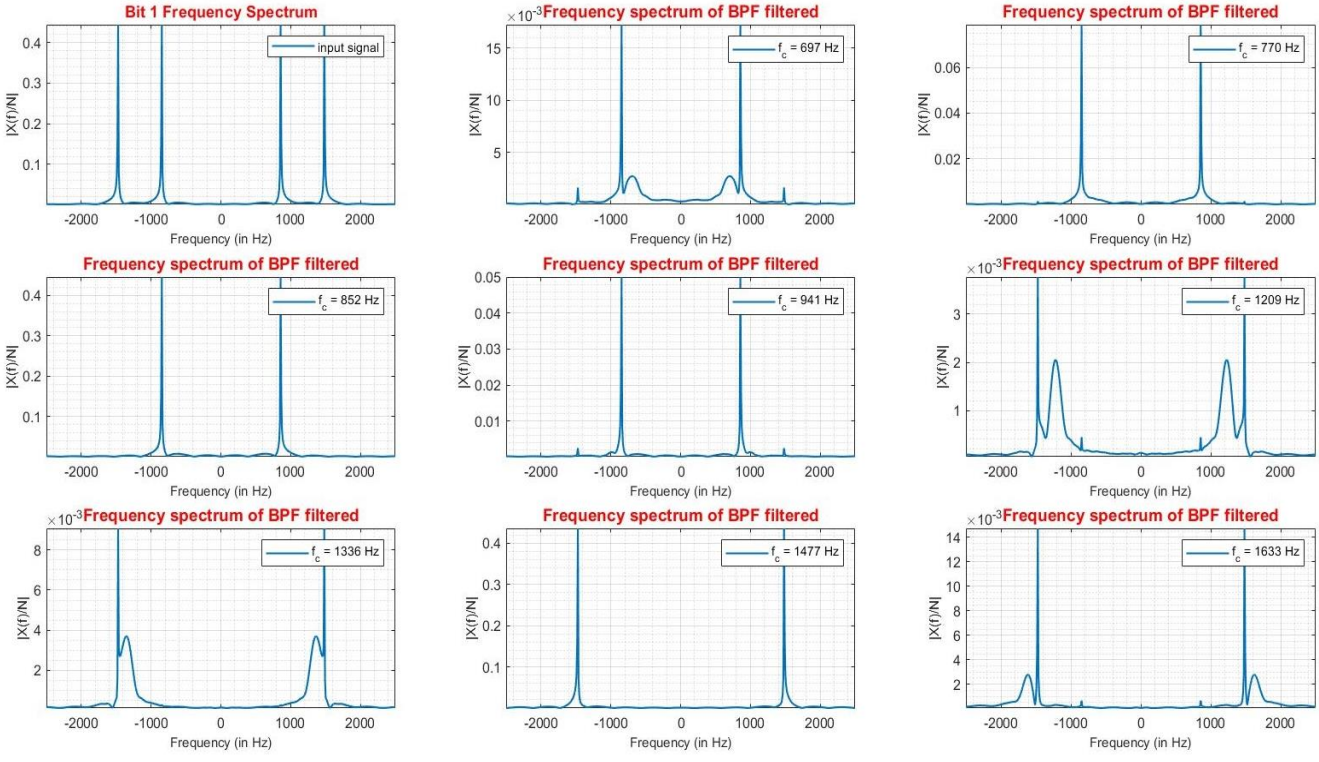




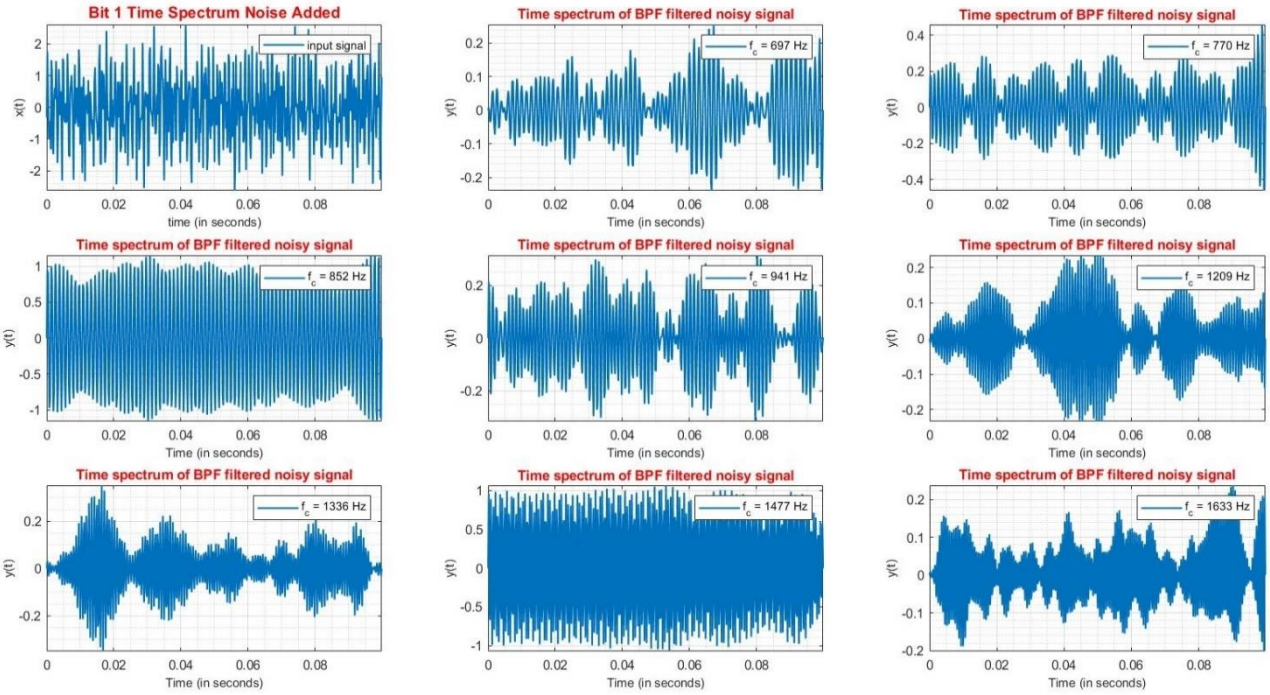
Time Domain Decoding of Bit 1



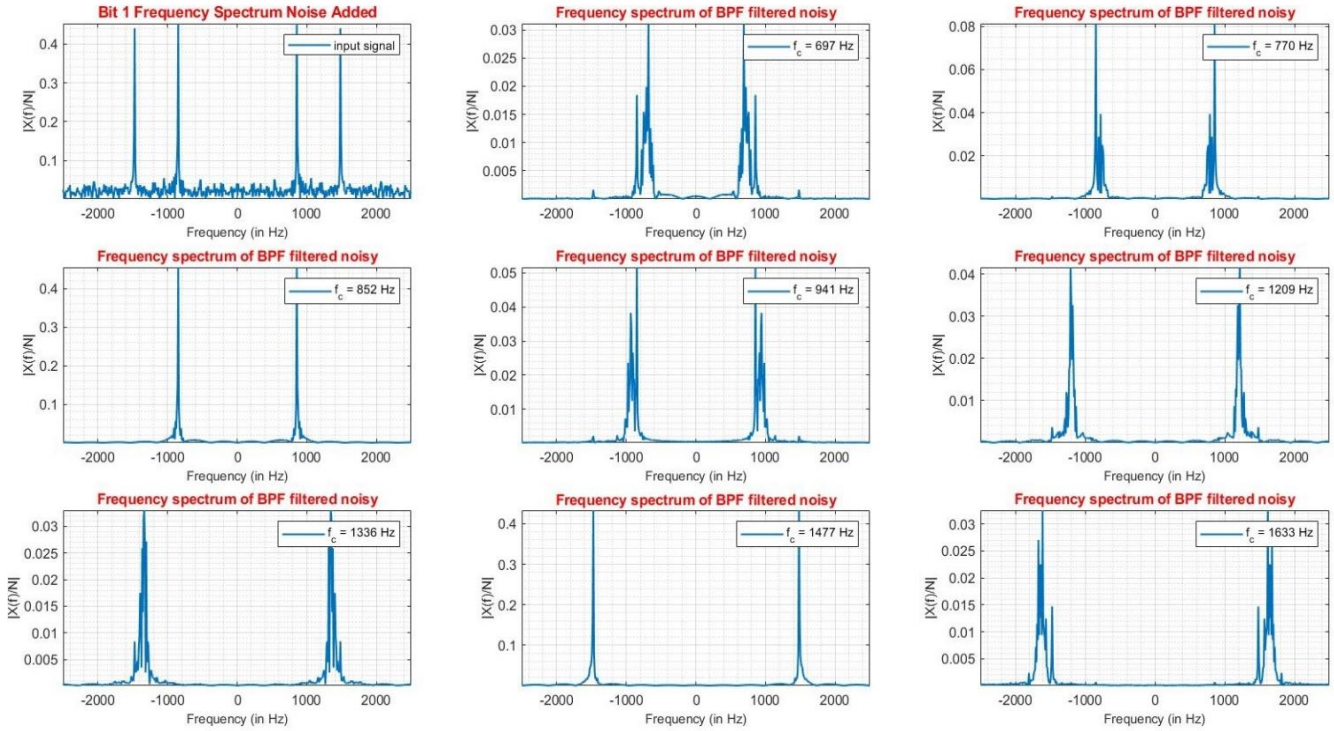
Frequency Domain Decoding of Bit 1



Time Domain Decoding of Noise Corrupted Bit 1



Frequency Domain Noisy Decoding of Bit 1

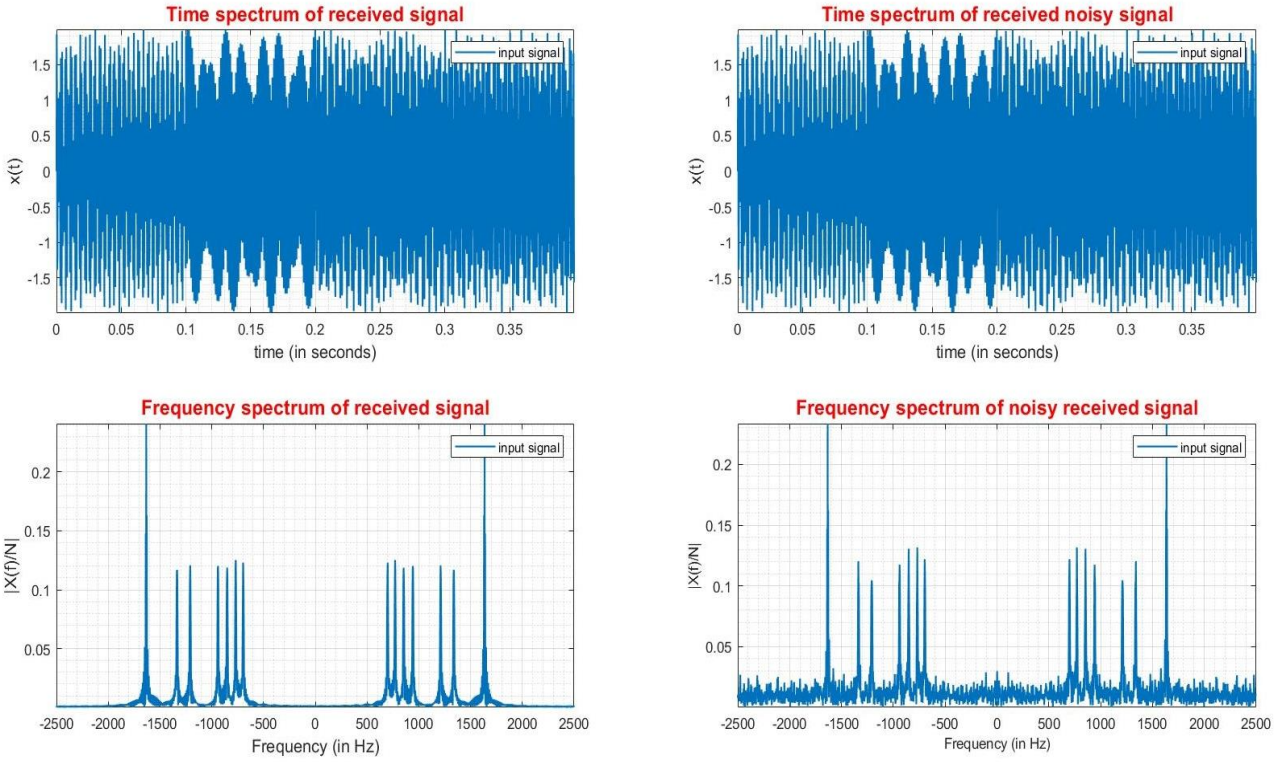


- While ***decoding a single key***, it can be observed that from the output of the filter banks, with the assumed values of filter length, signal time segment length and adequate sampling, ***frequency spectrum magnitude peak*** is highest for ***f_c = 852 Hz and f_c = 1477 Hz*** which correspond to key ***9*** which was the ***actual encoded key***.

Multiple Key Decoding:

- This is like multiple cases of single key encoding wherein the received signal is divided into segments of priorly known time length and each segment is analyzed for single key decoding and the finally the outcome of all the segments is merged to output the decoded result.
- For every segment, the above four graphs as in single key decoding is plotted. The input sequence here is ****C5A*** and the ***corresponding graphs*** are plotted below.

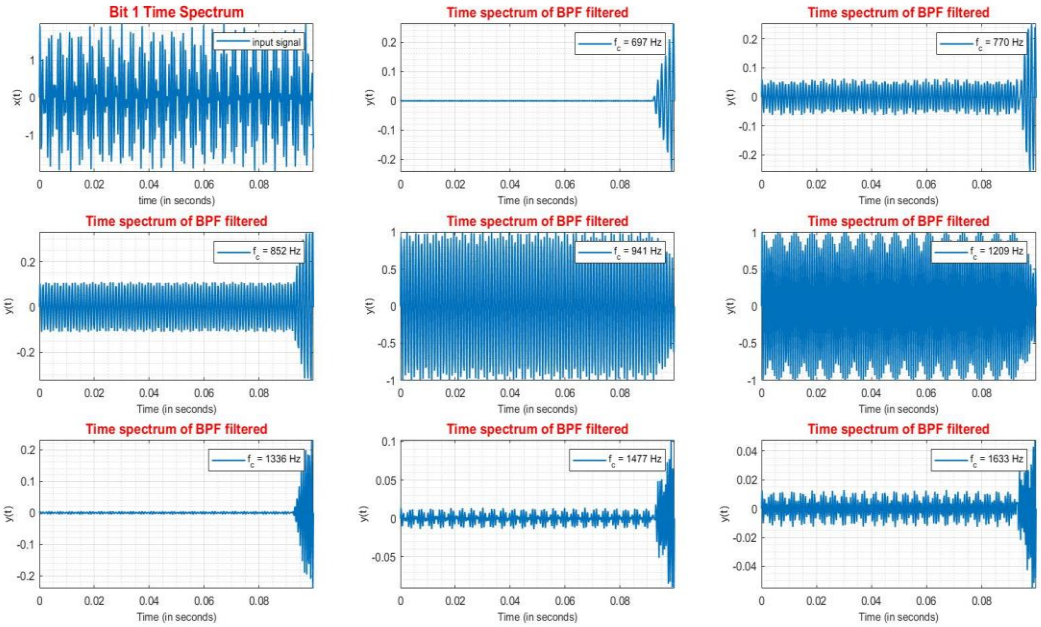
Received Signals at Decoding Side



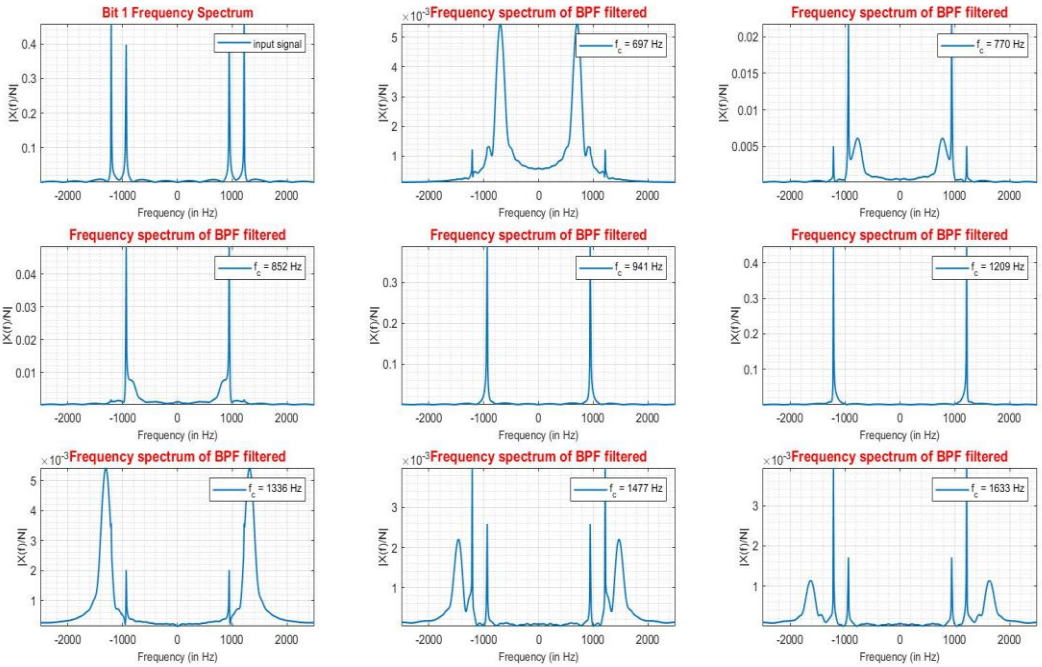
- In the received signals at decoding side graph for multiple key input, ***the 4 different waveforms*** (time signals in each time length segment) ***corresponding to 4 different keys*** can be viewed in the time domain response and their ***constituting frequencies*** (697 Hz, 770 Hz, 852 Hz, 941 Hz, 1209 Hz, 1336 Hz, 1633 Hz ---- 7 different frequencies in case of *C5A) can be seen in the ***frequency spectrums***.
- Now, the time domain signal is split into 4 segments and analyzed for decoding a single key. Each segment is of length L = 0.1 seconds in this case.

Time Segment 1 -> time (0 – 0.1s):

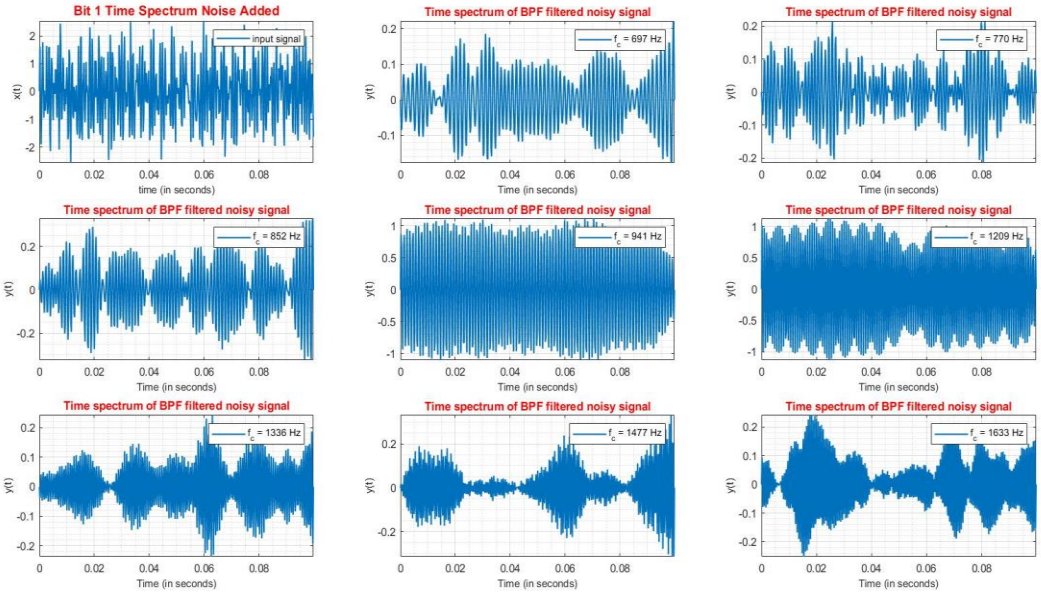
Time Domain Decoding of Bit 1



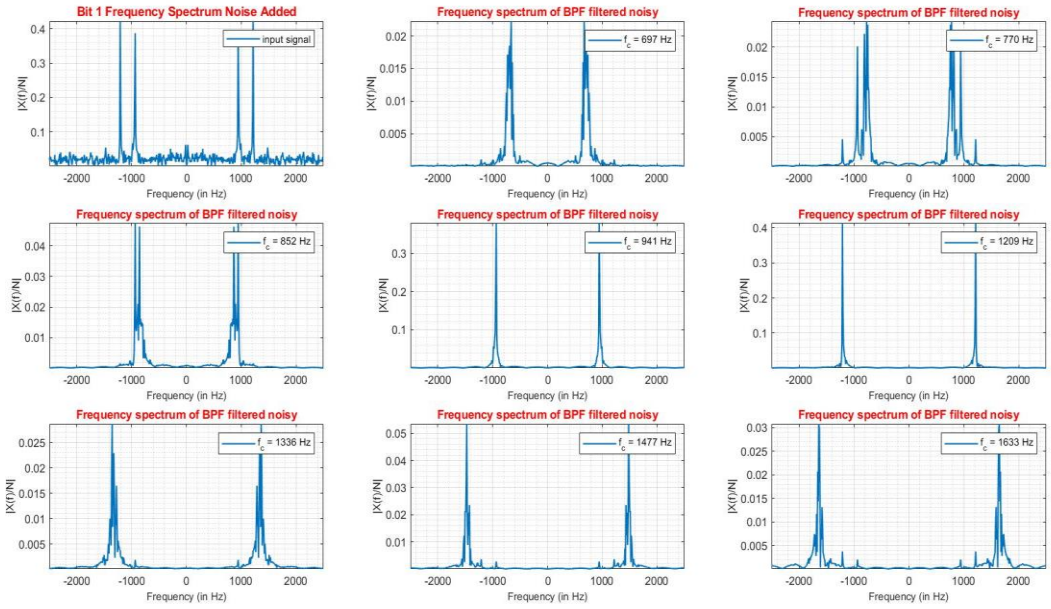
Frequency Domain Decoding of Bit 1



Time Domain Decoding of Noise Corrupted Bit 1



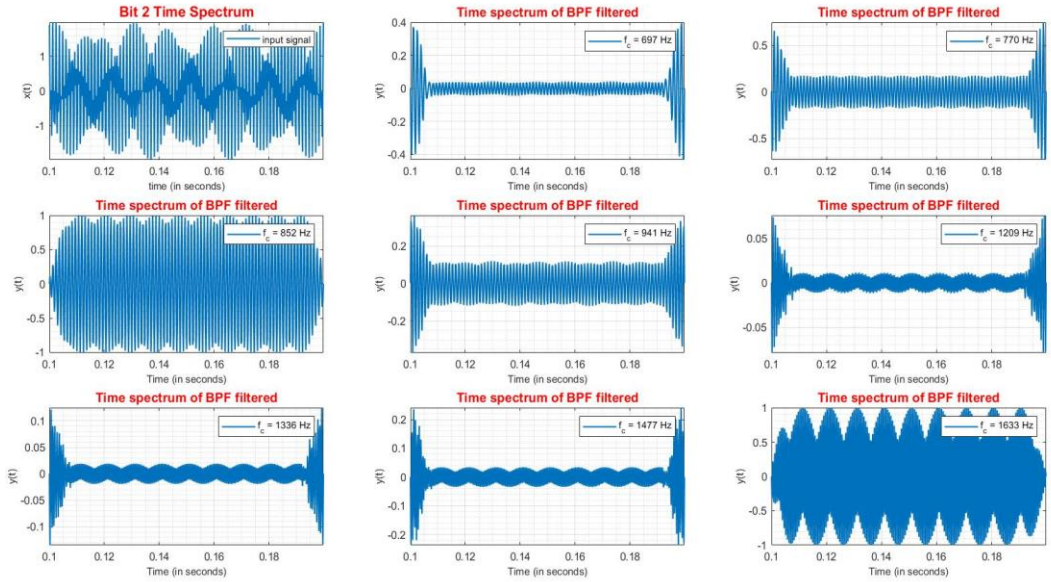
Frequency Domain Decoding of Noise Corrupted Bit 1



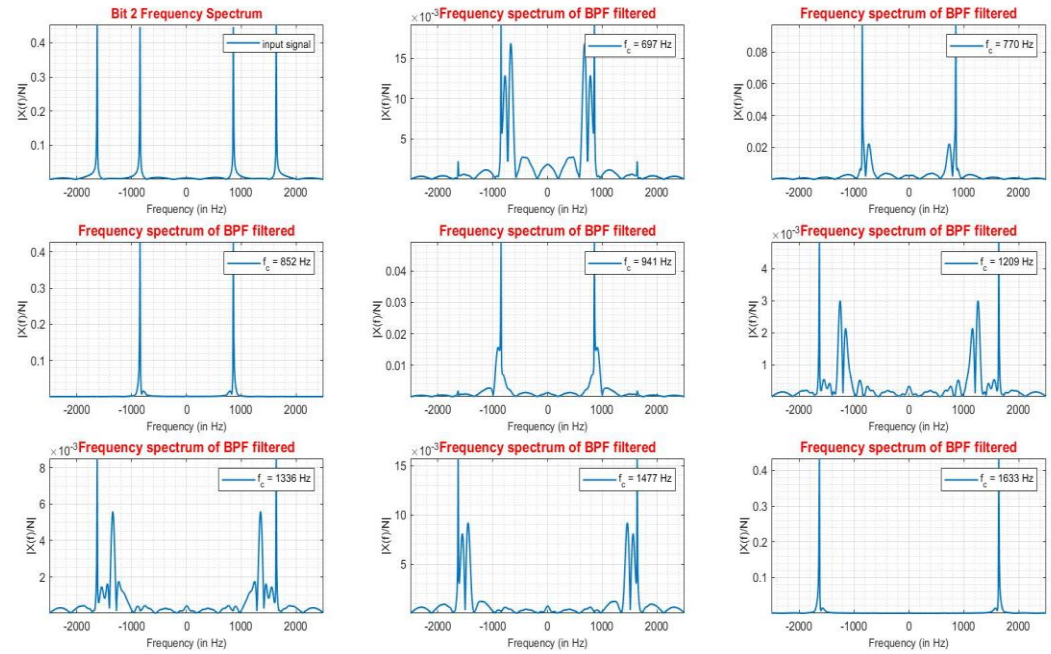
- For Segment 1, from the above plots, it can be seen that the ***amplitude*** is ***maximum for f_c = 941 Hz and 1209 Hz*** which correspond to the key '*' which is indeed the ***first key in the encoded sequence***.

Time Segment 2 -> time (0.1 – 0.2s):

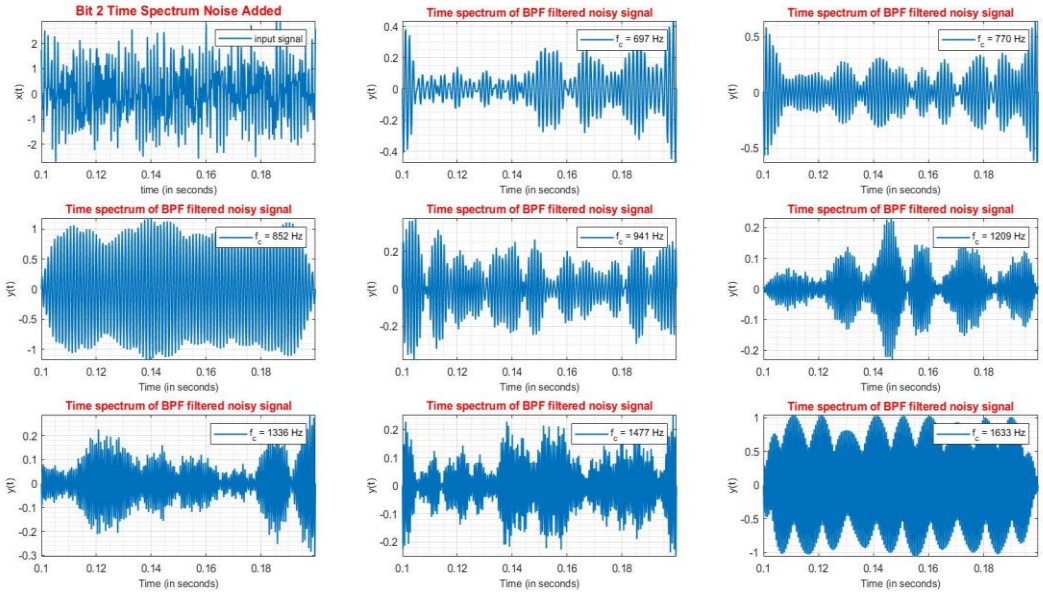
Time Domain Decoding of Bit 2



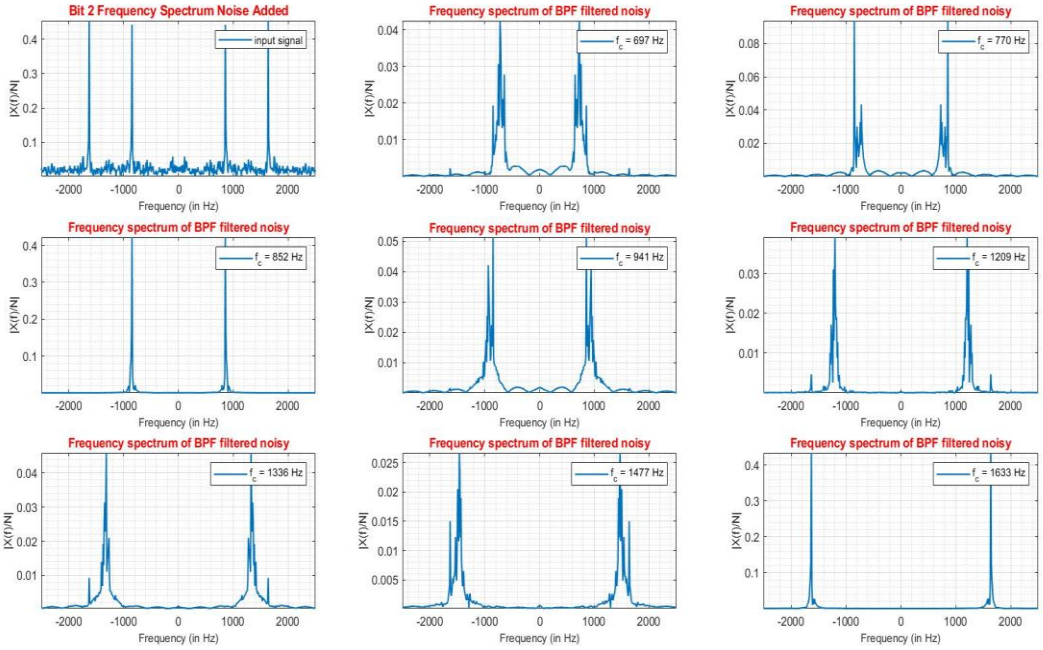
Frequency Domain Decoding of Bit 2



Time Domain Decoding of Noise Corrupted Bit 2



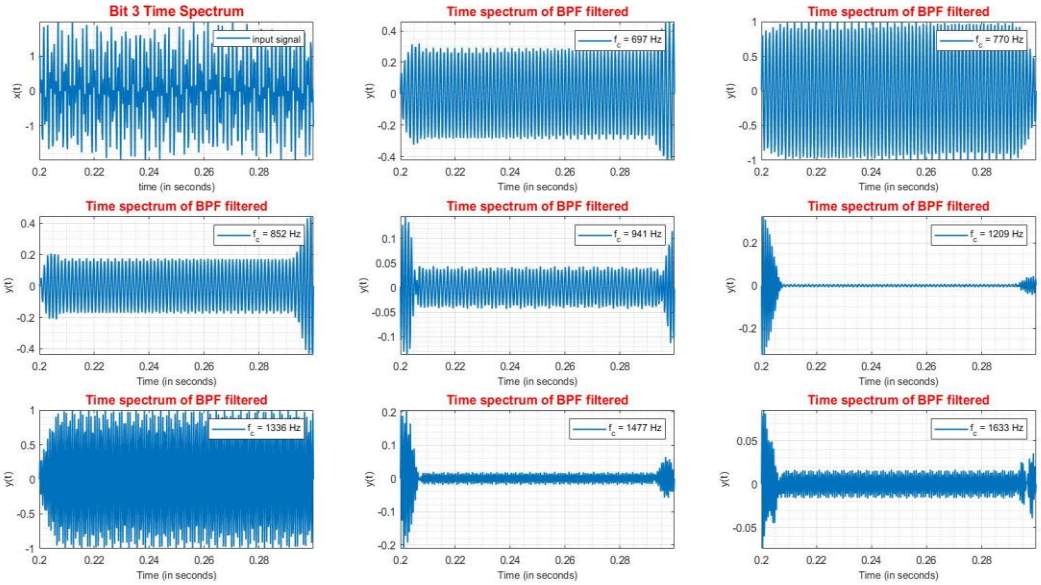
Frequency Domain Decoding of Noise Corrupted Bit 2



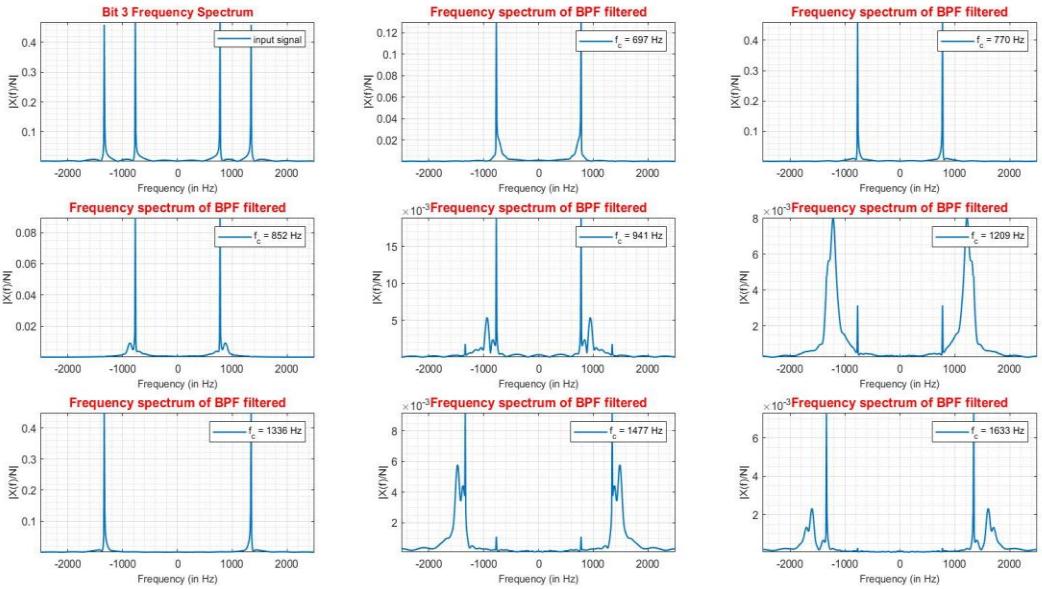
- For Segment 2, from the above plots, it can be seen that the ***amplitude*** is ***maximum for f_c = 852 Hz and 1633 Hz*** which correspond to the key 'C' which is indeed the ***second key in the encoded sequence***.

Time Segment 3 -> time (0.2 – 0.3s):

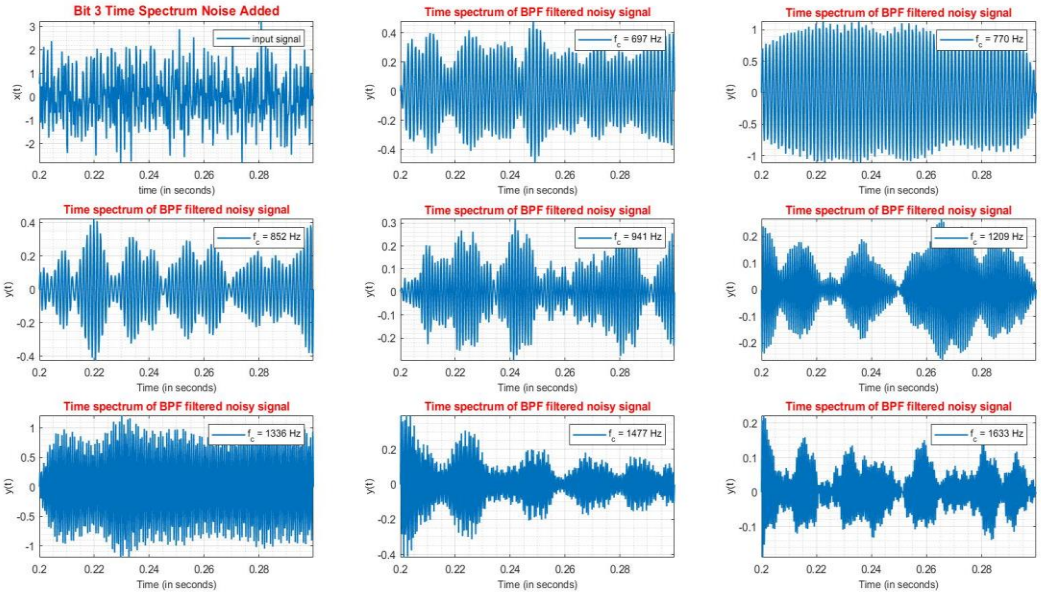
Time Domain Decoding of Bit 3



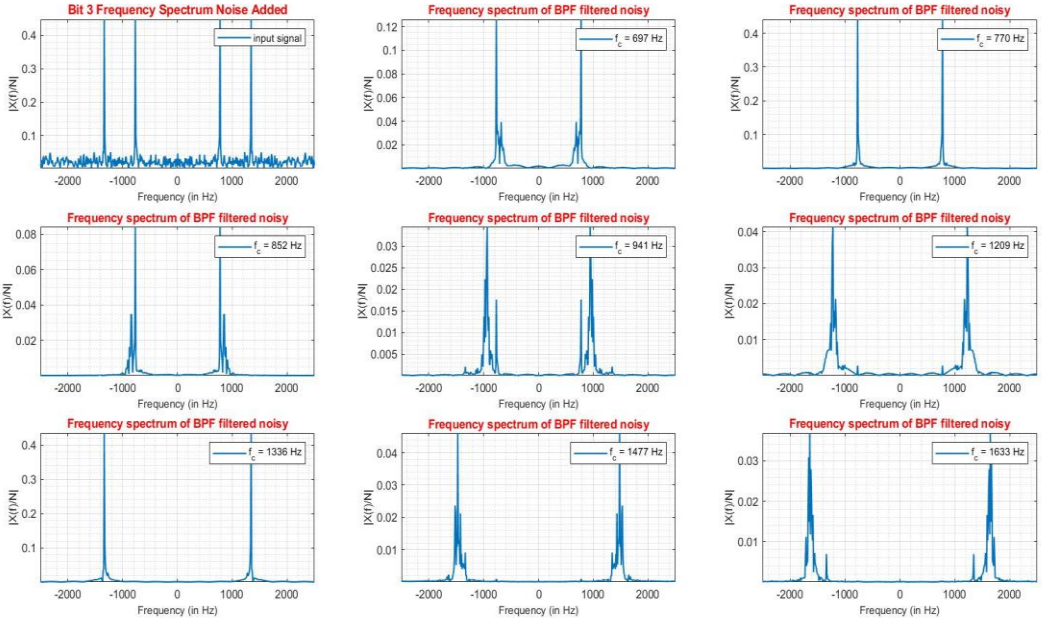
Frequency Domain Decoding of Bit 3



Time Domain Decoding of Noise Corrupted Bit 3



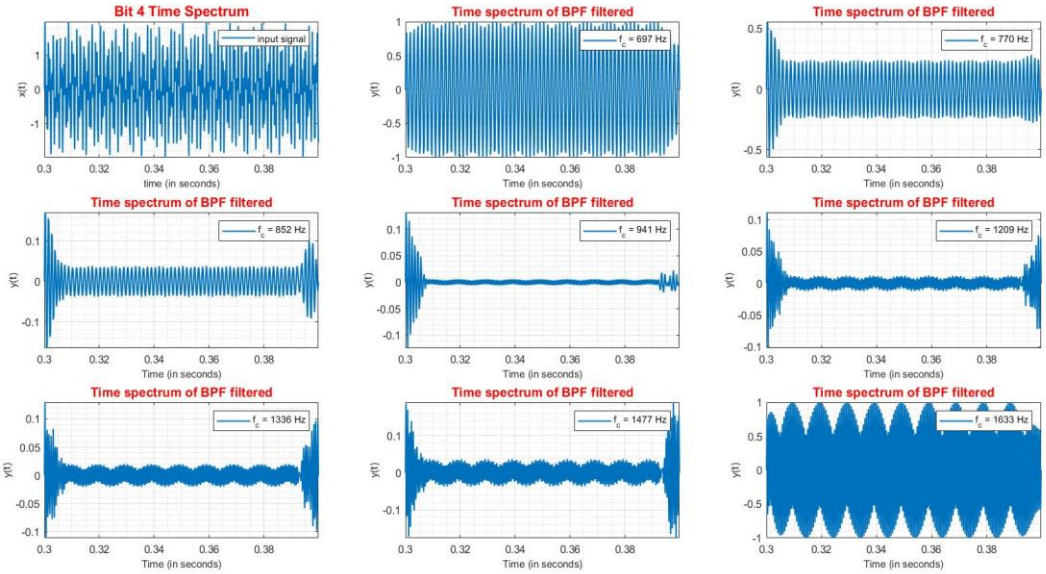
Frequency Domain Decoding of Noise Corrupted Bit 3



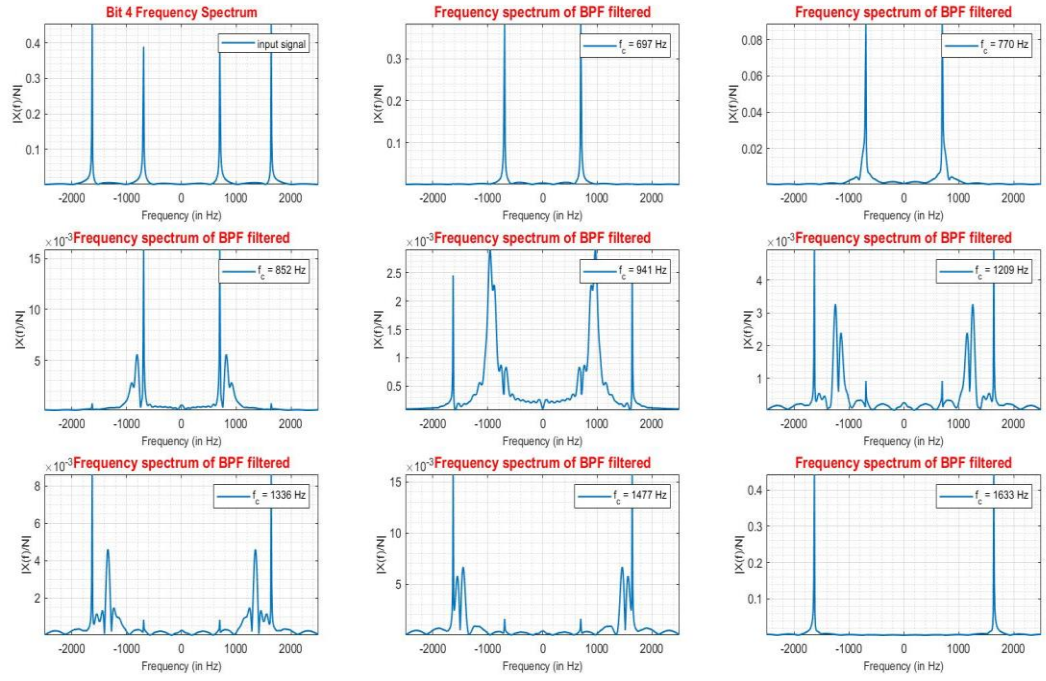
- For Segment 3, from the above plots, it can be seen that the ***amplitude*** is ***maximum for f_c = 770 Hz and 1336 Hz*** which correspond to the key '5' which is indeed the ***third key in the encoded sequence***.

Time Segment 4 -> time (0.3 – 0.4s):

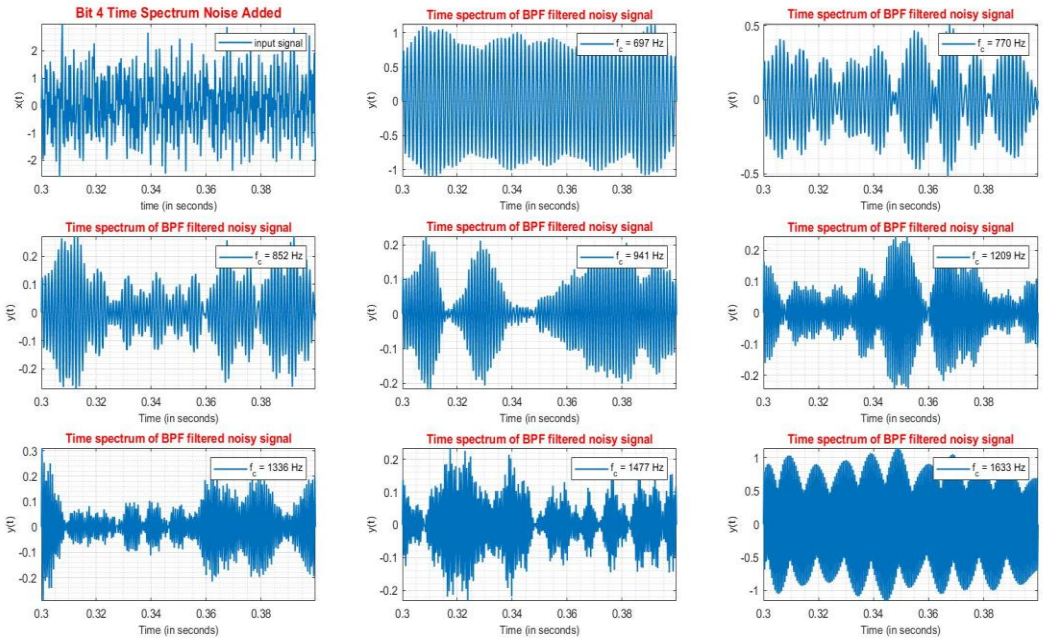
Time Domain Decoding of Bit 4



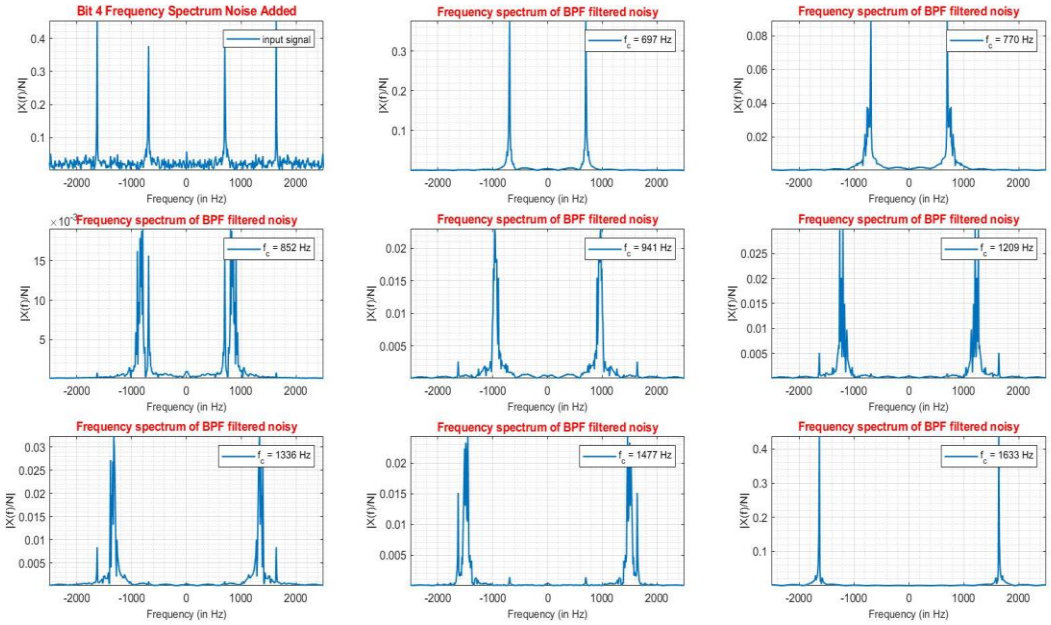
Frequency Domain Decoding of Bit 4



Time Domain Decoding of Noise Corrupted Bit 4



Frequency Domain Decoding of Noise Corrupted Bit 4



- For Segment 4, from the above plots, it can be seen that the ***amplitude*** is ***maximum for f_c = 697 Hz and 1633 Hz*** which correspond to the key 'A' which is indeed the ***fourth key in the encoded sequence***.

- Thus, for the multiple sequence case, as seen above

```
Command Window
>> Expt3_seq
Enter the Key Sequence: *C5A
CORRECT! Your chosen key is found to be *C5A
fx >>
```

- ***Analyzing*** by ***maximum frequency magnitude*** peak ***doesn't always*** yield correct results as when the signal gets ***corrupted with noise of high noise amplitude***, ***random spikes*** in other filter responses may turn out to be greater than of filtered with frequency components.
- Hence, ***in case of noise added signals***, ***analyzing by rms value*** of the ***output of the filters gave best results***.
- The ***max amplitude result from normal signals*** and ***rms amplitude from noisy signals*** are ***compared*** to give the ***final result***.
- In case of noise analysis, ***noise amplitude is chosen around 0.5*** such that the noise generated doesn't yield ***random spike magnitude greater than the signal amplitude levels***.

```
noise = noise_amp*randn(1,length(x_t)); %%white noise
x_noise = x_t + noise; %%noisy input
```

- Appropriate ***time segment length (L > (1/500))*** since lowest frequency component is 697 Hz), ***filter length (filt_len > 20), sampling frequency (Fs > 3.5 kHz*** since highest frequency component is 1633 Hz), ***number of sampling points (N around 512*** since from previous experiments, this yielded best sampling) and ***noise amplitude (noise_amp < 4)*** are required for efficient encoding and decoding.

MATLAB Code:

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

keyval = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '*', '0', '#',
'A', 'B', 'C', 'D'};
rowval = [697, 697, 697, 770, 770, 770, 852, 852, 852, 941, 941, 941,
697, 770, 852, 941];
colval = [1209, 1336, 1477, 1209, 1336, 1477, 1209, 1336, 1477, 1209,
1336, 1477, 1633, 1633, 1633, 1633];

row_freq = containers.Map(keyval, rowval);    %key to row freq map
col_freq = containers.Map(keyval, colval);    %key to col freq map
```

```
% inp = takekey(); %Taking the key input and the user must enter a single
key only

inp = input("Enter the Key Sequence: ",'s');    %For multiple key input

% disp(inp);

Fs = 5e3; %sampling frequency
L = 0.1;    %Time Duration of individual bit in seconds
N = 512; %Sample Points
noise_amp = 0.5; %Noise amplitude

x_t = [];
for i=1:length(inp)
    f1 = row_freq(inp(i));
    f2 = col_freq(inp(i));

    t = ((i-1)*L):(1/Fs):((i*L)-(1/Fs));
    x1_t = sin(2*pi*f1*t) + sin(2*pi*f2*t);
    x_t = cat(2, x_t, x1_t);  %Input signal
end
if(res==inp)    disp("CORRECT! Your chosen key is found to be "+res);
else disp("INCORRECT! Your chosen key is "+inp+" and the res is "+res);
end

%Decoding side starts
function rslt = decoded_key(x_t, len, freqvec, Fs, N, noise_amp)
    rslt = "";
    num_bit = length(x_t)/(len*Fs); %Number of bits in the input
    t = 0:1/Fs:(num_bit*len-1/Fs);    %len is individual bit length

    N1 = N*num_bit;

    f = linspace(-1/2,1/2,N1)*(Fs);

    noise = noise_amp*randn(1,length(x_t)); %%white noise
    x_noise = x_t + noise; %%noisy input

    %Time domain of Input signal
%     figure("Name", "Plotting received signals in time domain");
    figure("Name", "Plotting received signals");
    subplot(2,2,1);
    plot((0:(1/Fs):(num_bit*len)-1/Fs), x_t(1:(num_bit*len*Fs)), "LineWidth", 1.25);
    xlabel('time (in seconds)', "fontsize", 12);
    ylabel('x(t)', "fontsize", 12);
    title("Time spectrum of received signal", "fontsize", 14, "Color", 'r');
    legend("input signal");
%     legend(legend_strings_freq(1:1));
    axis tight;
    grid on;
    grid minor;

    %Time domain of Noise Input signal
%     figure("Name", "Plotting received noisy signals in time domain");
    subplot(2,2,2);
    plot((0:(1/Fs):(num_bit*len)-1/Fs), x_t(1:(num_bit*len*Fs)), "LineWidth", 1.25);
    xlabel('time (in seconds)', "fontsize", 12);
    ylabel('x(t)', "fontsize", 12);
    title("Time spectrum of received noisy signal", "fontsize", 14, "Color", 'r');
    legend("input signal");
%     legend(legend_strings_freq(1:1));
    axis tight;
    grid on;
```

```
    grid minor;


    %Freq Spectrum of Input signal
%     figure("Name", "Plotting received signals in frequency domain");
    subplot(2,2,3);
    plot(f, abs(fftshift(fft(x_t, N1))/N1), "LineWidth", 1.25);
    xlabel('Frequency (in Hz)', "fontsize", 12);
    ylabel('|X(f)/N|', "fontsize", 12);
    title("Frequency spectrum of received signal", "fontsize", 14, "Color", 'r');
    legend("input signal");
%     legend(legend_strings_freq(1:1));
    axis tight;
    grid on;
    grid minor;

    %Freq Spectrum of noisy Input signal
%     figure("Name", "Plotting received noisy signals in frequency domain");
    subplot(2,2,4);
    plot(f, abs(fftshift(fft(x_noise, N1))/N1), "LineWidth", 1.25);
    xlabel('Frequency (in Hz)', "fontsize", 10);
    ylabel('|X(f)/N|', "fontsize", 10);
    title("Frequency spectrum of noisy received signal", "fontsize", 14, "Color",
'r');
    legend("input signal");
%     legend(legend_strings_freq(1:1));
    axis tight;
    grid on;
    grid minor;


    for k = 1:num_bit
        maxmagval = zeros(1, length(freqvec));
        maxmagval_n = zeros(1, length(freqvec));
        res = "";
        x1_t = x_t(((k-1)*len*Fs+1):(k*len*Fs));
        x1_noise = x_noise(((k-1)*len*Fs+1):(k*len*Fs));
        t1 = ((k-1)*len):1/Fs:(k*len-1/Fs);
        f1 = linspace(-1/2,1/2,N)*Fs;

        %Time domain of Input signal
        f_time = figure("Name", "Bit "+int2str(k)+" Time Domain");
        subplot(3,3,1);
        plot(t1, x1_t, "LineWidth", 1.25);
        xlabel('time (in seconds)', "fontsize", 9);
        ylabel('x(t)', "fontsize", 9);
        title("Bit "+int2str(k)+" Time Spectrum", "fontsize", 12.5, "Color", 'r');
        legend("input signal");
        axis tight;
        grid on;
        grid minor;

        %Time domain of Noise Input signal
        f_time_noise = figure("Name", "Bit "+int2str(k)+" Time Domain Noisy");
        subplot(3,3,1);
        plot(t1, x1_noise, "LineWidth", 1.25);
        xlabel('time (in seconds)', "fontsize", 9);
        ylabel('x(t)', "fontsize", 9);
        title("Bit "+int2str(k)+" Time Spectrum Noise Added", "fontsize", 11.5,
"Color", 'r');
        legend("input signal");
        axis tight;
        grid on;
        grid minor;


        %Freq Spectrum of Input signal
```

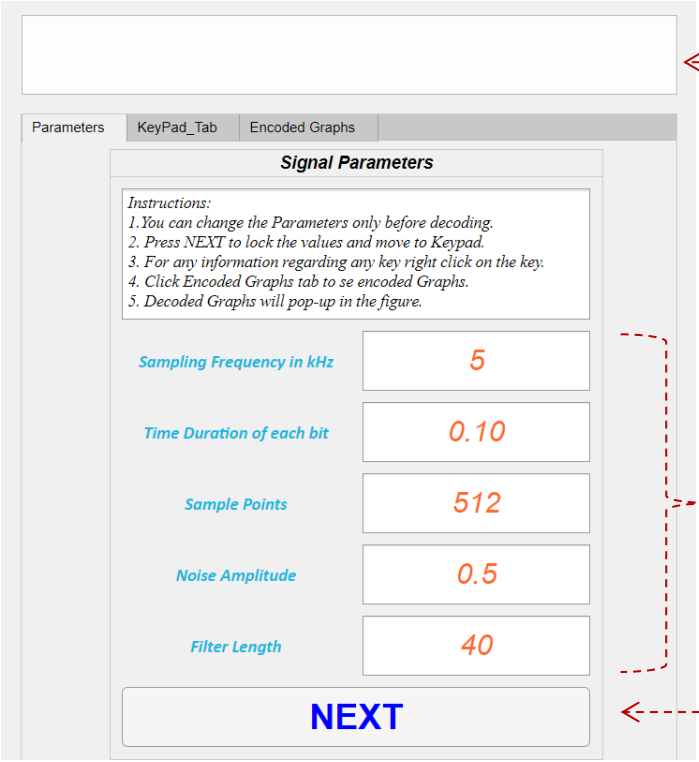
```
        f_freq = figure("Name", "Bit "+int2str(k)+" Freq Domain");
        subplot(3,3,1);
        plot(f1, abs(fftshift(fft(x1_t, N))/N), "LineWidth", 1.25);
        xlabel('Frequency (in Hz)', "fontsize", 9);
        ylabel('|X(f)/N|', "fontsize", 9);
        title("Bit "+int2str(k)+" Time Spectrum", "fontsize", 11, "Color", 'r');
        legend("input signal");
        axis tight;
        grid on;
        grid minor;

        %Freq Spectrum of noisy Input signal
        f_freq_noise = figure("Name", "Bit "+int2str(k)+" Freq Domain Noisy");
        subplot(3,3,1);
        plot(f1, abs(fftshift(fft(x1_noise, N))/N), "LineWidth", 1.25);
        xlabel('Frequency (in Hz)', "fontsize", 9);
        ylabel('|X(f)/N|', "fontsize", 9);
        title("Bit "+int2str(k)+" Frequency Spectrum Noise Added", "fontsize", 10.5,
"Color", 'r');
        legend("input signal");
        axis tight;
        grid on;
        grid minor;

%         f_freq_H = figure("Name", "Filter Response");

        for j=1:length(freqvec)
    %         fpass = [freqvec(i)-10, freqvec(i)+10];   %for using inbuilt filter
    %         y_n = bandpass(x_t, fpass, Fs);

            filt_len = 40;  %Length of filter

            h_n = BPFilt(freqvec(j), filt_len, Fs);

            y_n = filtfilt(h_n, 1, x1_t);    %Filtering the signal using filtfilt
command

            y_noise = filtfilt(h_n, 1, x1_noise);    %Filtering the noisy signal
using filtfilt command

            Y_w = fftshift(fft(y_n, N))/N; %fft sprectrum shifting and normalization
             Y_w_noise = fftshift(fft(y_noise, N))/N; %fft sprectrum shifting and
normalization
%             %         Plotting the frequency domain representation of the h_n
%             figure(f_freq_H);
%             subplot(3,3,j);
%             omega = linspace(-pi, pi, 750); %sweep of omega
%             H_f = freqz(h_n, 1, omega); % Frequency response of the filter
%
%             plot(omega/pi*(Fs/2), 20*log10(abs(H_f)), "LineWidth", 1.3);
%             xlabel("Frequency (Hz)", "fontsize", 10);
%             ylabel('Magnitude (dB)',"fontsize", 10);
%             title("Frequency Response of BPF with fc = "+freqvec(j)+" Hz",
"fontsize", 14, "Color", 'r');
%             axis tight;
%             grid on;
%             grid minor;

            %Time Spectrum of Filtered signal
            figure(f_time);
            subplot(3,3, j+1);
            plot(t1, y_n, "LineWidth", 1.25);   %Plotting for first 128 points
            xlabel('Time (in seconds)', "fontsize", 9);
            ylabel('y(t)', "fontsize", 9);
            title("Time spectrum of BPF filtered", "fontsize", 11.5, "Color", 'r');
            legend("f_c = "+int2str(freqvec(j))+ " Hz");
            axis tight;
```

```
            grid on;
            grid minor;

            %Time Spectrum of Noisy Filtered signal
            figure(f_time_noise);
            subplot(3,3, j+1);
            plot(t1, y_noise, "LineWidth", 1.25);   %Plotting for 2 sampling periods
            xlabel('Time (in seconds)', "fontsize", 9);
            ylabel('y(t)', "fontsize", 9);
            title("Time spectrum of BPF filtered noisy signal", "fontsize", 10,
"Color", 'r');
            legend("f_c = "+int2str(freqvec(j))+ " Hz");
            axis tight;
            grid on;
            grid minor;

            %Freq Spectrum of Filtered signal
            figure(f_freq);
            subplot(3,3, j+1);
            plot(f1, abs(Y_w), "LineWidth", 1.25);
            xlabel('Frequency (in Hz)', "fontsize", 9);
            ylabel('|X(f)/N|', "fontsize", 9);
            title("Frequency spectrum of BPF filtered","fontsize",11.5,"Color",'r');
            legend("f_c = "+int2str(freqvec(j))+ " Hz");
            axis tight;
            grid on;
            grid minor;

            %Freq Spectrum of Noisy Filtered signal
            figure(f_freq_noise);
            subplot(3,3, j+1);
            plot(f1, abs(Y_w_noise), "LineWidth", 1.25);
            xlabel('Frequency (in Hz)', "fontsize", 9);
            ylabel('|X(f)/N|', "fontsize", 9);
            title("Frequency spectrum of BPF filtered noisy","fontsize",10,"Color",
'r');
            legend("f_c = "+int2str(freqvec(j))+ " Hz");
            axis tight;
            grid on;
            grid minor;

            [temp, ind] = max(Y_w);%finding the max freq value and storing its
corresponding freq value
            maxmagval(j) = temp;

            [temp_n] = rms(y_noise);%finding the rms freq value and storing its
corresponding freq value
            maxmagval_n(j) = temp_n;
    %         disp("for h_"+int2str(i)+" filtered, max val freq is at:
"+int2str(abs(f(ind))));
        end

        [tmp, I] = maxk(maxmagval, 2); %Finding the max 2 of all the corresponding
values and storing the corresponding indices and from that indices finding the
corresponding frequecies from freqvec
        maxim = [freqvec(I(1)), freqvec(I(2))];
        res1 = findkey(maxim);

        %For noisy frequency response
        [tmp_n, I_n] = maxk(maxmagval_n, 2); %Finding the max 2 of all the
corresponding values and storing the corresponding indices and from that indices
finding the corresponding frequecies from freqvec
        maxim_n = [freqvec(I_n(1)), freqvec(I_n(2))];
        res2 = findkey(maxim_n);

        if(res1==res2)  res = res1;
        else res = 'W'; %Returns Incorrect result
```

```
        end
        rslt = rslt + res;
    end

end

function h_n = BPFilt(f_c, filt_len, Fs)
    %designing bandpass filter bank
    w_c = f_c*(pi/(Fs/2));  %Digitizing central frequency
    wid = filt_len; %length of filter
    h_n = zeros(1, wid+1);
    for n=1:(wid)
       h_n(n) = cos(w_c*(n-1));
    end
    omega = linspace(-pi, pi, 750); %sweep of omega
    H_f = freqz(h_n, 1, omega); % Frequency response of the filter
    beta = 1/max(abs(H_f)); %Choosing beta to make the maximum value of frequency
response magnitude as 1
    h_n = h_n*beta;
end

function res = findkey(maxim)
    if(ismember(697, maxim) && ismember(1209, maxim)) res = '1';
    elseif (ismember(697, maxim) && ismember(1336, maxim)) res = '2';
    elseif (ismember(697, maxim) && ismember(1477, maxim)) res = '3';
    elseif (ismember(697, maxim) && ismember(1633, maxim)) res = 'A';
    elseif (ismember(770, maxim) && ismember(1209, maxim)) res = '4';
    elseif (ismember(770, maxim) && ismember(1336, maxim)) res = '5';
    elseif (ismember(770, maxim) && ismember(1477, maxim)) res = '6';
    elseif (ismember(770, maxim) && ismember(1633, maxim)) res = 'B';%Freq Spectrum
of Filtered signal
    elseif (ismember(852, maxim) && ismember(1209, maxim)) res = '7';
    elseif (ismember(852, maxim) && ismember(1336, maxim)) res = '8';
    elseif (ismember(852, maxim) && ismember(1477, maxim)) res = '9';
    elseif (ismember(852, maxim) && ismember(1633, maxim)) res = 'C';
    elseif (ismember(941, maxim) && ismember(1209, maxim)) res = '*';
    elseif (ismember(941, maxim) && ismember(1336, maxim)) res = '0';
    elseif (ismember(941, maxim) && ismember(1477, maxim)) res = '#';
    elseif (ismember(941, maxim) && ismember(1633, maxim)) res = 'D';
    end
end
```

ADDENDUM

- This is a GUI designed to deal with input parameters, key sequence and output plots etc.



Main Screen

Displays the Final Result

When, the app or code is run, this tab appears. This is the parameters tab.

Parameters

These are the default values. You can edit the values before giving input or directly move to input

NEXT

Click to save the above values if changed before encoding

Clear, Back and Enter buttons convey their regular meanings. For any other info, right click on those buttons

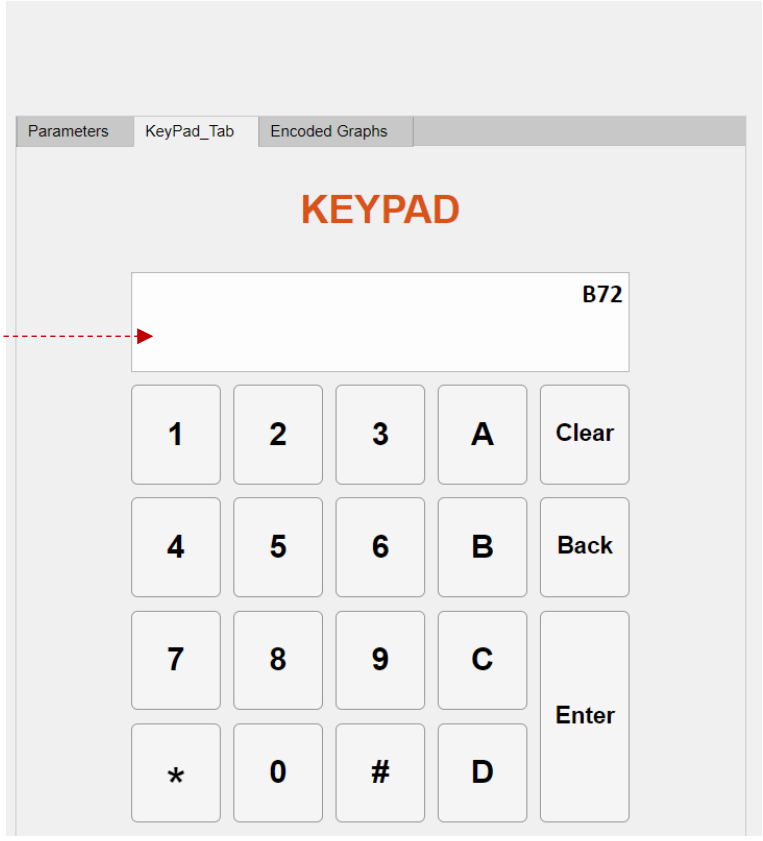
*Click **Enter** to start decoding at the other end*

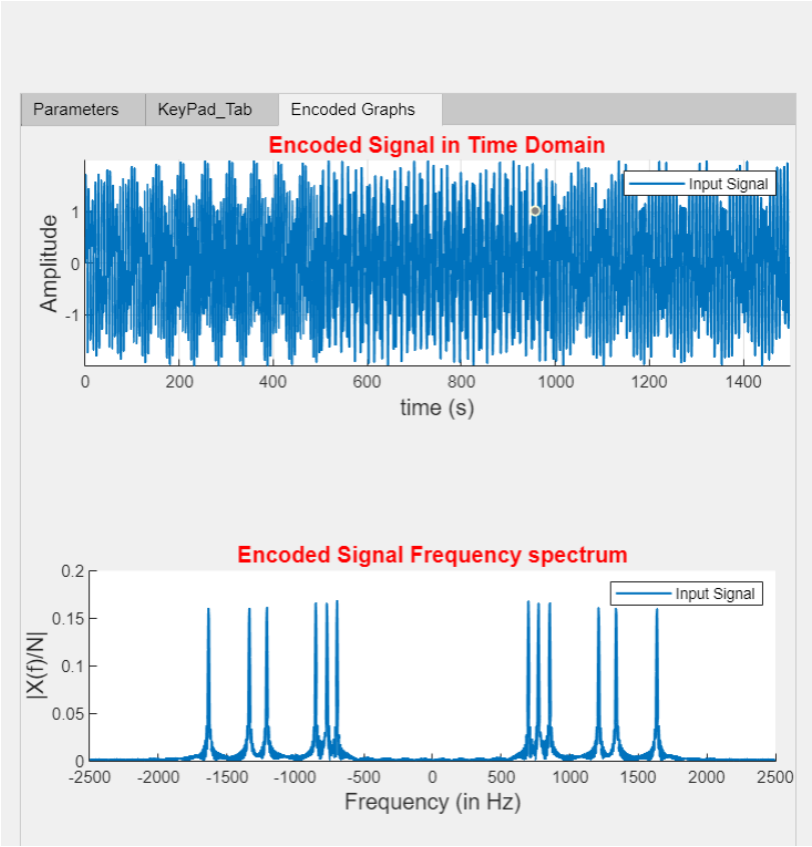
Keypad Screen

This is the KEYPAD tab. Used to give input

*The Digits 0-9, A-D, * and # are the keys to be pressed*

The entered keys are displayed on the keypad-screen simultaneously as pressed (with some time lag)





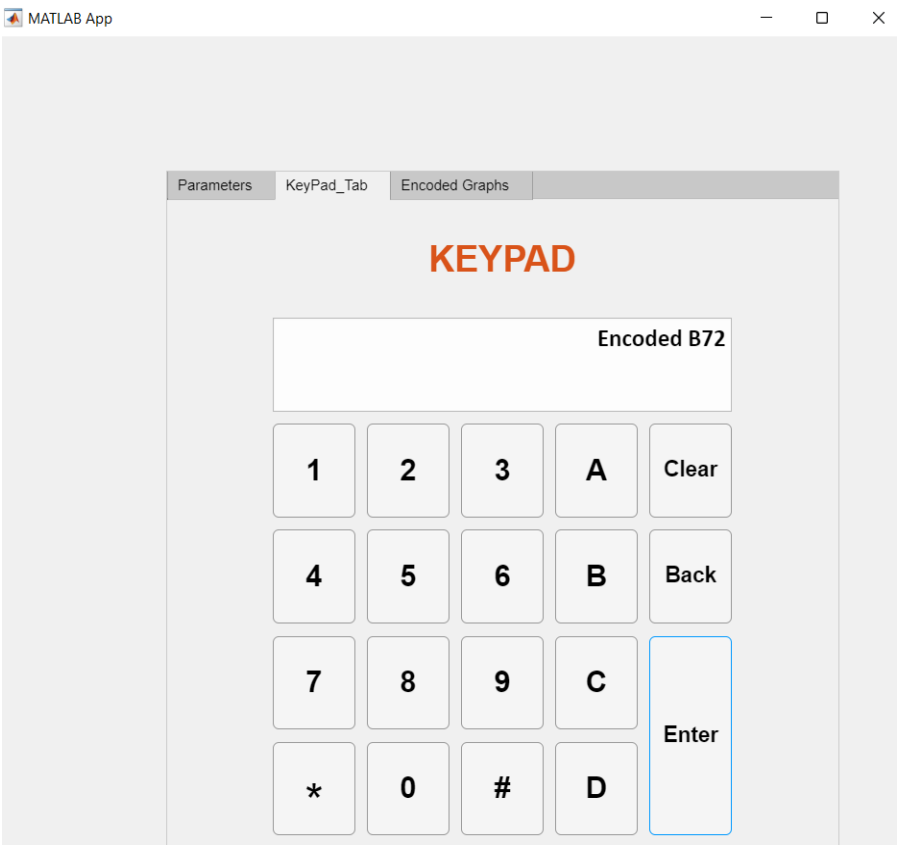
As the input is being given in the keypad tab, the corresponding encoded signals will be plotted here simultaneously. These change with every key pressed

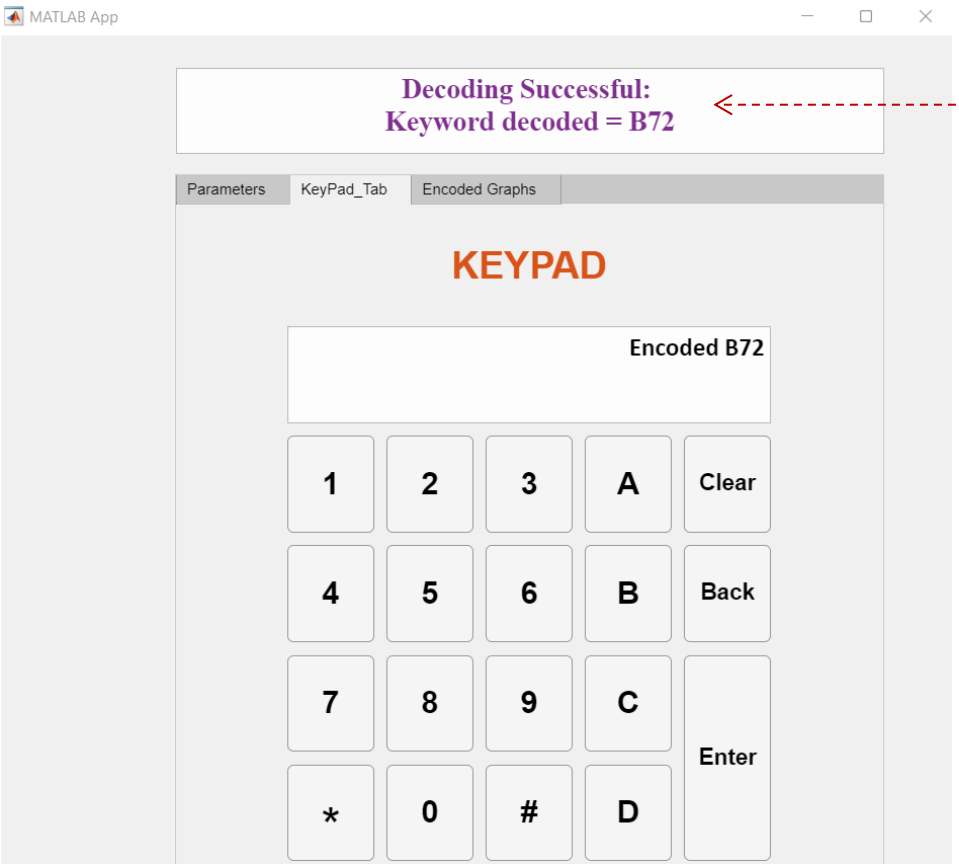
At any time, you can switch between the tabs

*When pressed **Enter,** the program goes to the decoding side, and it starts decoding key by key and the relevant plots will start popping up in the new page.*

***Note:** For a sequence having n keys, there will be (4×n+1) decoding related plot figures with each figure having 9 subplots*

For each key, there will be four graphs related to time, frequency, noise corrupted time domain and noise corrupted frequency domain and in each case there are 8 subplots: input signal segment and the 8 bandpass filter outputs





Final Result

After decoding completely, final result will be displayed on the Main Screen.

*To try again, Click **Clear** and start giving input.*

*The parameter values are locked when clicked **NEXT** and can be unlocked by clicking **Clear** after which parameter values can be changed*

- For the above entered sequence ***B72,*** the obtained plot figures are made available here due to space constraints.
- The above app can be run inside the MATLAB app section or run the corresponding MATLAB code (run like any .m script file) whose links are KEYPAD_APP and *KEYPAD_APP_exported respectively*.

Note: The above mentioned are the drive links to the files and plot figures and all the links to the files are made available in the *Reference* section at last.

Discussion:

- ***DTMF*** is a signalling system used for identifying the keys or the number dialled on a pushbutton or DTMF keypad. The early telephone systems used pulse dialling or loop disconnect signalling. This was replaced by multi frequency (MF) dialling. ***DTMF is a multi-frequency tone dialling system*** used by the push button keypads in telephone and mobile sets to convey the number or key dialled by the caller. DTMF has enabled the long-distance signalling of dialled numbers in voice frequency range over telephone lines. This has eliminated the need of telecom operator between the caller and the callee and evolved automated dialling in the telephone switching centres.
- ***DTMF*** (Dual tone multi frequency) as the name suggests uses a ***combination of two sine wave tones*** to represent a key. These tones are called row and column frequencies as they correspond to the layout of a telephone keypad.
- The frequencies used are 697 Hz, 770 Hz, 852 Hz, 941 Hz, 1209 Hz, 1336 Hz, 1477Hz, and 1633 Hz. The ***frequencies were carefully chosen*** in such a way as to ***prevent harmonics***. ***No frequency is a multiple of another*** and the ***difference or sum between any two frequencies is not equal*** to any other frequency. In practical systems, the higher frequencies are transmitted at 3 dB louder to compensate for any high frequency roll-off in the channel.
- The signal thus generated corresponding to the input sequence of keys is analyzed ***window by window (bit by bit or time segment by segment)*** where in for each segment, ***maximum frequency response and maximum rms value*** of the output of the filter banks is noted and the ***maximum two of those*** are ***detected to decode the key***.
- There are several advantages to this scheme. One is that by using this scheme ***we can encode 16 keys by only 8 frequencies***. If we had used a single frequency to encode each key, then we would have needed 16 frequencies and hence, ***16 FIR filters leading to large overheads and more resolution***. Also, the frequencies chosen for DTMF tones cannot be interpreted by the human ear but can be easily decoded by a phone system and computer. Thus, sensitive information can be isolated from the agents as well as from call recording systems.
- The ***FIR Band Pass*** filter design is ***derived from the FIR Ideal pass*** by ***multiplying it with cosine pulse*** and ***rectangular windowing***. Ideally, it should be of infinite length but due to practical constraints, it is truncated using a rectangular window of length filt_len. ***As filt_len decreases***, bandwidth increases which allows adjacent frequencies to pass and hence may lead to ***incorrect decoding***.

- In the given scheme, there are ***5 main parameters which can be varied***. They are ***time segment length (L), Sampling frequency (F_s), Number of Sample Points (N), Noise amplitude (noise_amp) and filter length(filt_len)*** whose optimal values and reasons are discussed in the above observations.
- In my design, it is better to choose ***values of L and F_s*** such that the ***value of $L \times F_s$ is an integer*** since its ***rounded off value is used as an index of a vector***.
- Choosing the ***number of sample points (N) as power of 2 yields faster results*** as ***computing fft will be faster*** in those cases. (Since radix 2 fft algos will be faster)
- In ***most practical*** cases or ***updated cases***, ***prior knowledge*** of the ***time segment length of each key*** will ***not*** be ***available*** at the ***decoding side***, or the length may keep changing. In those cases, a ***change in key*** is ***identified*** by ***detecting the white noise*** present between the ***time gap of*** pressing ***two keys*** where ***no DTMF frequency*** is present.

For Further Reference:

DRIVE LINK - *Link to my plots, figures, reports, script files and app related to the experiment.*

Additional DOC – *contains the MATLAB code written in doc*

MATLAB Code script – *MATLAB script file .m extension*