EE 367 Final Lab

Design and Analysis of a DTMF Decoding System

Report by:
Ethan Vosburg (evosburg@calpoly.edu)
Isaac Lake (ilake@calpoly.edu)

March 19, 2024

Table of Contents

1 Documentation

Summary of our results: We prepared to implement all of our filters to complete the project when we realized we could skip that step and just run peak detectors. Our final ISI error was 7.1%. Because of our solution, we solved the need for filters thus we required 0 bits for them.

1. High-Level block diagram

   Our solution to the project reads the sample creates a buffer, takes the DFT of that buffer, and uses peak detectors to find the peak on the upper and lower half of the frequency chart, sends those peaks into a simple logic brick that will find and output the corresponding symbol.

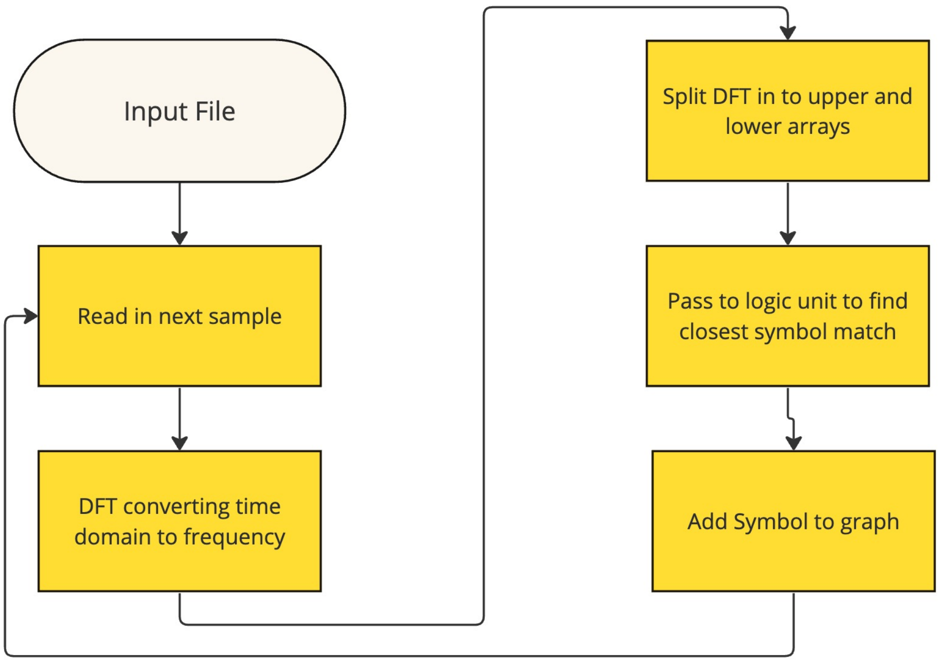

Figure 1: High-level block diagram

2. Python source code used to implement your system.

Listing 1: Python DTMF

```
#!/usr/bin/python

import sys
import time

import base64
import random as random

import datetime
import time
import math

import matplotlib.pyplot as plt
import numpy as np

from cpe367_wav import cpe367_wav
from cpe367_sig_analyzer import cpe367_sig_analyzer

from my_fifo import my_fifo
from tqdm import tqdm
from LogicUnit import LogicUnit


############################################
############################################
# define routine for detecting DTMF tones
def process_wav(fpath_sig_in):


   ##################################
   # define list of signals to be displayed by and analyzer
   #  note that the signal analyzer already includes: 'symbol_val','symbol_det','error'
   more_sig_list = ['sig_1','sig_2']

   # sample rate is 4kHz
   fs = 4000
   # 30 is best
   fifoSize = 30
   multiRatio = fs // fifoSize

   # Create fifo
   buffer = my_fifo(fifoSize)

   # instantiate signal analyzer and load data
   s2 = cpe367_sig_analyzer(more_sig_list,fs)
   s2.load(fpath_sig_in)
   s2.print_desc()

   ########################
   # students: setup filters

   # process input
   xin = 0
   for n_curr in tqdm(range(s2.get_len()), desc="Processing..."):

      # read next input sample from the signal analyzer
      xin = s2.get('xin',n_curr)

      buffer.update(xin)
      magnitudeArray = []

      for i in range(buffer.get_size()):
         magnitudeArray.append(buffer.get(i))

      dftList = dft(magnitudeArray)
```

```python
        lowerMax = np.argmax(dftList[:len(dftList)//2])
        upperMax = np.argmax(dftList[len(dftList)//2:]) + len(dftList)//2
        # print(buffer.get_size())
        # print(len(dftList))
        # print(dftList)

        symbol_val_det = LogicUnit(lowerMax * multiRatio, upperMax * multiRatio).symbol()

        # save intermediate signals as needed, for plotting
        #  add signals, as desired!
        s2.set('sig_1',n_curr,xin)
        s2.set('sig_2',n_curr,2 * xin)

        # save detected symbol
        s2.set('symbol_det',n_curr,symbol_val_det)

        # get correct symbol (provided within the signal analyzer)
        symbol_val = s2.get('symbol_val',n_curr)

        # compare detected signal to correct signal
        symbol_val_err = 0
        if symbol_val != symbol_val_det: symbol_val_err = 1

        # save error signal
        s2.set('error',n_curr,symbol_val_err)


    # display mean of error signal
    err_mean = s2.get_mean('error')
    print('mean error = '+str( round(100 * err_mean,1) )+'%')

    # define which signals should be plotted
    plot_sig_list = ['sig_1','sig_2','symbol_val','symbol_det','error']

    # plot results
    s2.plot(plot_sig_list)

    return True

############################################
############################################
# define DFT function
def dft(sampleValues):
    # Initialize variables
    dftList = []
    n = 0

    for sample in range(len(sampleValues)//2):
        # Initialize resetting variables
        dftCalcArray = []
        imag = 0
        real = 0

        # Perform DFT calculation
        for k in range(len(sampleValues) - 1) :
            imag += sampleValues[k] * np.sin(-2 * math.pi * k * n / len(sampleValues))
            real += sampleValues[k] * np.cos(-2 * math.pi * k * n / len(sampleValues))
        dftList.append(np.sqrt(imag**2 + real**2) / (len(sampleValues))//2)
        # print(f"Sample {sample}: {dftList[sample]} and {sampleValues[sample]}")
        n+=1

    return dftList




############################################
############################################
# define main program
def main():
```

```
    # check python version!
    major_version = int(sys.version[0])
    if major_version < 3:
        print('Sorry! must be run using python3.')
        print('Current version: ')
        print(sys.version)
        return False

    # assign file name
    fpath_sig_in = 'Final Lab/source/input/DTMF Signals Slow.txt'
    # fpath_sig_in = 'Final Lab/source/input/DTMF Signals Fast.txt'


    # let's do it!
    return process_wav(fpath_sig_in)




############################################
############################################
# call main function
if __name__ == '__main__':

    main()
    quit()
```

Listing 2: Logic Unit for Getting Symbols from Frequency

```
import numpy as np

class LogicUnit:

    def __init__(self, lowerMax, upperMax):
        self.lowerMax = lowerMax
        self.upperMax = upperMax

    def symbol(self):
        lowerArray = [697, 770, 852, 941]
        self.lowerDiffs = np.abs(lowerArray - self.lowerMax)
        upperArray = [1209, 1336, 1477, 1633]
        self.upperDiffs = np.abs(upperArray - self.upperMax)

        two_d_array = [
            [1, 2, 3, 'a'],
            [4, 5, 6, 'b'],
            [7, 8, 9, 'c'],
            ['*', 0, '#', 'd']
        ]
        return two_d_array[np.argmin(self.lowerDiffs)][np.argmin(self.upperDiffs)]
```

3. Generated Analysis Data

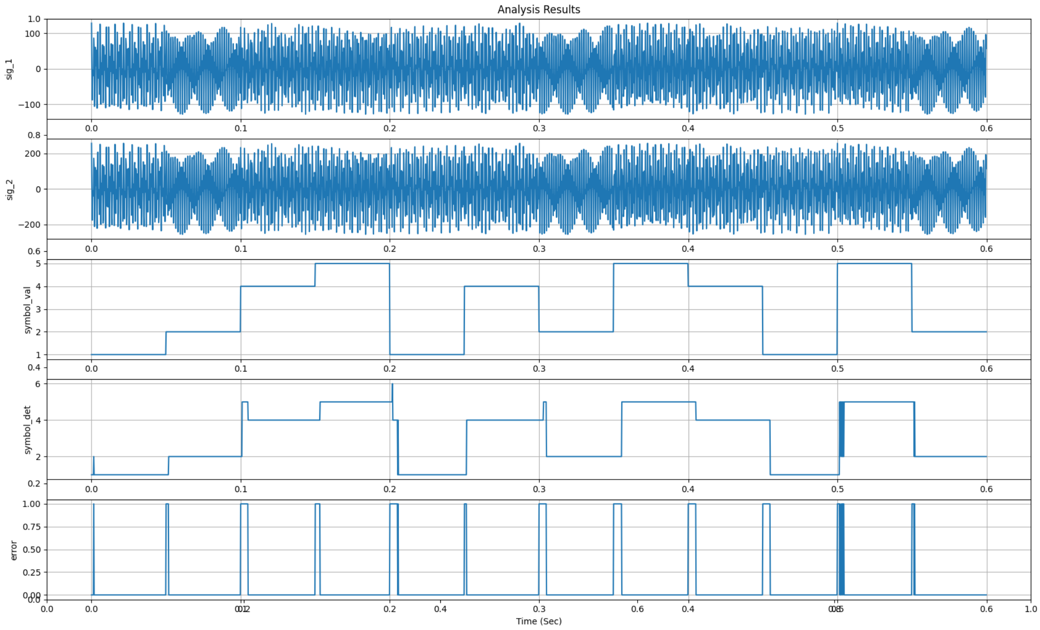

Figure 2: Analysis Results

4. Filters (While we did not end up using them these are the filters we made)

Listing 3: 697HZ BP

```
function y = BP697(x)
%DOFILTER Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 23.2 and DSP System Toolbox 23.2.
% Generated on: 05-Mar-2024 13:08:57

%#codegen

% To generate C/C++ code from this function use the codegen command.
% Type 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fstop1 = 620;   % First Stopband Frequency
    % Fpass1 = 680;   % First Passband Frequency
    % Fpass2 = 710;   % Second Passband Frequency
    % Fstop2 = 780;   % Second Stopband Frequency
    % Astop1 = 6;     % First Stopband Attenuation (dB)
    % Apass  = 1;     % Passband Ripple (dB)
    % Astop2 = 6;     % Second Stopband Attenuation (dB)
    % Fs     = 4000;  % Sampling Frequency
    %
    % h = fdesign.bandpass('fst1,fp1,fp2,fst2,ast1,ap,ast2', Fstop1, Fpass1, ...
    %                      Fpass2, Fstop2, Astop1, Apass, Astop2, Fs);
```

```
    %
    % Hd = design(h, 'butter', ...
    %     'MatchExactly', 'passband', ...
    %     'SystemObject', true,...
    %      UseLegacyBiquadFilter=true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 0 -1 1 -0.881385021996587 0.911473662795519], ...
        'ScaleValues', [0.0442631686022406; 1]);
end

s = double(x);
y = step(Hd,s);
```

Listing 4: 770HZ BP

```
function y = BP770(x)
%DOFILTER Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 23.2 and DSP System Toolbox 23.2.
% Generated on: 05-Mar-2024 13:07:06

%#codegen

% To generate C/C++ code from this function use the codegen command.
% Type 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fstop1 = 700;   % First Stopband Frequency
    % Fpass1 = 750;   % First Passband Frequency
    % Fpass2 = 790;   % Second Passband Frequency
    % Fstop2 = 840;   % Second Stopband Frequency
    % Astop1 = 6;     % First Stopband Attenuation (dB)
    % Apass  = 1;     % Passband Ripple (dB)
    % Astop2 = 6;     % Second Stopband Attenuation (dB)
    % Fs     = 4000;  % Sampling Frequency
    %
    % h = fdesign.bandpass('fst1,fp1,fp2,fst2,ast1,ap,ast2', Fstop1, Fpass1, ...
    %                       Fpass2, Fstop2, Astop1, Apass, Astop2, Fs);
    %
    % Hd = design(h, 'butter', ...
    %     'MatchExactly', 'passband', ...
    %     'SystemObject', true,...
    %      UseLegacyBiquadFilter=true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 0 -1 1 -0.666157014109844 0.883665323160146], ...
        'ScaleValues', [0.0581673384199269; 1]);
end

s = double(x);
y = step(Hd,s);
```

Listing 5: 852HZ BP

```
function y = BP852(x)
%DOFILTER Filters input x and returns output y.

```

```
% MATLAB Code
% Generated by MATLAB(R) 23.2 and DSP System Toolbox 23.2.
% Generated on: 05-Mar-2024 13:10:47

%#codegen

% To generate C/C++ code from this function use the codegen command.
% Type 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fstop1 = 790;   % First Stopband Frequency
    % Fpass1 = 840;   % First Passband Frequency
    % Fpass2 = 870;   % Second Passband Frequency
    % Fstop2 = 920;   % Second Stopband Frequency
    % Astop1 = 6;     % First Stopband Attenuation (dB)
    % Apass  = 1;     % Passband Ripple (dB)
    % Astop2 = 6;     % Second Stopband Attenuation (dB)
    % Fs     = 4000;  % Sampling Frequency
    %
    % h = fdesign.bandpass('fst1,fp1,fp2,fst2,ast1,ap,ast2', Fstop1, Fpass1, ...
    %                      Fpass2, Fstop2, Astop1, Apass, Astop2, Fs);
    %
    % Hd = design(h, 'butter', ...
    %     'MatchExactly', 'passband', ...
    %     'SystemObject', true,...
    %      UseLegacyBiquadFilter=true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 0 -1 1 -0.431733010683812 0.911473662795518], ...
        'ScaleValues', [0.044263168602241; 1]);
end

s = double(x);
y = step(Hd,s);
```

Listing 6: 941HZ BP

```
function y = BP941(x)
%DOFILTER Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 23.2 and DSP System Toolbox 23.2.
% Generated on: 05-Mar-2024 13:11:50

%#codegen

% To generate C/C++ code from this function use the codegen command.
% Type 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fstop1 = 870;   % First Stopband Frequency
    % Fpass1 = 920;   % First Passband Frequency
    % Fpass2 = 960;   % Second Passband Frequency
    % Fstop2 = 1010;  % Second Stopband Frequency
    % Astop1 = 6;     % First Stopband Attenuation (dB)
    % Apass  = 1;     % Passband Ripple (dB)
    % Astop2 = 6;     % Second Stopband Attenuation (dB)
```

```
    % Fs     = 4000;  % Sampling Frequency
    %
    % h = fdesign.bandpass('fst1,fp1,fp2,fst2,ast1,ap,ast2', Fstop1, Fpass1, ...
    %                      Fpass2, Fstop2, Astop1, Apass, Astop2, Fs);
    %
    % Hd = design(h, 'butter', ...
    %     'MatchExactly', 'passband', ...
    %     'SystemObject', true,...
    %      UseLegacyBiquadFilter=true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 0 -1 1 -0.17735608093889 0.883665323160146], ...
        'ScaleValues', [0.058167338419927; 1]);
end

s = double(x);
y = step(Hd,s);
```

Listing 7: 1209HZ BP

```
function y = BP1209(x)
%DOFILTER Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 23.2 and DSP System Toolbox 23.2.
% Generated on: 05-Mar-2024 13:13:13

%#codegen

% To generate C/C++ code from this function use the codegen command.
% Type 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fstop1 = 1140;  % First Stopband Frequency
    % Fpass1 = 1190;  % First Passband Frequency
    % Fpass2 = 1230;  % Second Passband Frequency
    % Fstop2 = 1280;  % Second Stopband Frequency
    % Astop1 = 6;     % First Stopband Attenuation (dB)
    % Apass  = 1;     % Passband Ripple (dB)
    % Astop2 = 6;     % Second Stopband Attenuation (dB)
    % Fs     = 4000;  % Sampling Frequency
    %
    % h = fdesign.bandpass('fst1,fp1,fp2,fst2,ast1,ap,ast2', Fstop1, Fpass1, ...
    %                      Fpass2, Fstop2, Astop1, Apass, Astop2, Fs);
    %
    % Hd = design(h, 'butter', ...
    %     'MatchExactly', 'passband', ...
    %     'SystemObject', true,...
    %      UseLegacyBiquadFilter=true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 0 -1 1 0.610453230045989 0.883665323160146], ...
        'ScaleValues', [0.058167338419927; 1]);
end

s = double(x);
y = step(Hd,s);
```

Listing 8: 1336HZ BP

```
function y = BP1336(x)
%DOFILTER Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 23.2 and DSP System Toolbox 23.2.
% Generated on: 05-Mar-2024 13:14:08

%#codegen

% To generate C/C++ code from this function use the codegen command.
% Type 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fstop1 = 1270;  % First Stopband Frequency
    % Fpass1 = 1320;  % First Passband Frequency
    % Fpass2 = 1360;  % Second Passband Frequency
    % Fstop2 = 1410;  % Second Stopband Frequency
    % Astop1 = 6;     % First Stopband Attenuation (dB)
    % Apass  = 1;     % Passband Ripple (dB)
    % Astop2 = 6;     % Second Stopband Attenuation (dB)
    % Fs     = 4000;  % Sampling Frequency
    %
    % h = fdesign.bandpass('fst1,fp1,fp2,fst2,ast1,ap,ast2', Fstop1, Fpass1, ...
    %                      Fpass2, Fstop2, Astop1, Apass, Astop2, Fs);
    %
    % Hd = design(h, 'butter', ...
    %     'MatchExactly', 'passband', ...
    %     'SystemObject', true,...
    %      UseLegacyBiquadFilter=true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 0 -1 1 0.959337037818403 0.883665323160145], ...
        'ScaleValues', [0.0581673384199272; 1]);
end

s = double(x);
y = step(Hd,s);
```

Listing 9: 1477HZ BP

```
function y = BP1477(x)
%DOFILTER Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 23.2 and DSP System Toolbox 23.2.
% Generated on: 05-Mar-2024 13:17:30

%#codegen

% To generate C/C++ code from this function use the codegen command.
% Type 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fstop1 = 1390;  % First Stopband Frequency
    % Fpass1 = 1450;  % First Passband Frequency
    % Fpass2 = 1490;  % Second Passband Frequency
```

```
    % Fstop2 = 1540;  % Second Stopband Frequency
    % Astop1 = 6;     % First Stopband Attenuation (dB)
    % Apass  = 1;     % Passband Ripple (dB)
    % Astop2 = 6;     % Second Stopband Attenuation (dB)
    % Fs     = 4000;  % Sampling Frequency
    %
    % h = fdesign.bandpass('fst1,fp1,fp2,fst2,ast1,ap,ast2', Fstop1, Fpass1, ...
    %                      Fpass2, Fstop2, Astop1, Apass, Astop2, Fs);
    %
    % Hd = design(h, 'butter', ...
    %     'MatchExactly', 'passband', ...
    %     'SystemObject', true,...
    %      UseLegacyBiquadFilter=true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 0 -1 1 1.2683561909662 0.883665323160146], ...
        'ScaleValues', [0.058167338419927; 1]);
end

s = double(x);
y = step(Hd,s);
```

Listing 10: 1633HZ BP

```
function y = BP1633(x)
%DOFILTER Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 23.2 and DSP System Toolbox 23.2.
% Generated on: 05-Mar-2024 13:18:44

%#codegen

% To generate C/C++ code from this function use the codegen command.
% Type 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fstop1 = 1560;  % First Stopband Frequency
    % Fpass1 = 1620;  % First Passband Frequency
    % Fpass2 = 1660;  % Second Passband Frequency
    % Fstop2 = 1710;  % Second Stopband Frequency
    % Astop1 = 6;     % First Stopband Attenuation (dB)
    % Apass  = 1;     % Passband Ripple (dB)
    % Astop2 = 6;     % Second Stopband Attenuation (dB)
    % Fs     = 4000;  % Sampling Frequency
    %
    % h = fdesign.bandpass('fst1,fp1,fp2,fst2,ast1,ap,ast2', Fstop1, Fpass1, ...
    %                      Fpass2, Fstop2, Astop1, Apass, Astop2, Fs);
    %
    % Hd = design(h, 'butter', ...
    %     'MatchExactly', 'passband', ...
    %     'SystemObject', true,...
    %      UseLegacyBiquadFilter=true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 0 -1 1 1.59121640388426 0.883665323160147], ...
        'ScaleValues', [0.0581673384199265; 1]);
end

s = double(x);
y = step(Hd,s);
```

2 Additional Question

Answers elaborating on our project.

1. Describe how your system design evolved throughout the project. Were there any significant changes to your approach?

   Originally we were following the advice of filtering for each frequency and then detecting the peak but eventually, we realized we could accomplish the same result with only peak detectors thus we pivoted our solution halfway through. This was also simpler to implement and allowed use to leverage the matrix operations available in numpy.
2. How would you modify your system design to provide a separate output signal indicating that no symbol is present?

   We could very easily put a check that if the peaks detected weren't high enough we send in a unique fail value to our logic unit that would be able to check against a threshold that we set and then from there fail if a symbols is not found or bad.
3. How might your system be improved, in terms of: ISI performance, computational requirements, or numerical representation?

   This can always be optimized by finding the perfect buffer size amount, shifting from 36 to 30 counts brought our error down to 7.1% from 15%. Changing the signal chain of the DFT would also help with the processing accuracy that we saw. Because we know the specific data that we are working with, we would also be able to tailor the solution more closely to our goal.
4. What are the benefits of using the dual tones of the DTMF standard?

   The benefits are that we only need $2\sqrt{(\text{\# of symbols})}$ to represent symbols, in our case, we only need to look at 6 frequencies as opposed to 9 individual frequencies, additionally we know where the frequencies we are looking for are and can ignore much of the noise with some clever peak detection.
5. Could a square wave be used? What are the pros and cons of using a square wave to generate the various DTMF signals?

   A square wave could be used instead of a sin wave to implement the DTMF however, there might be some problems with combining the square waves as they would sum differently than when using sin waves.