California Polytechnic State University

CPE 367

Digital Signals and Systems

Final Project: CPE 367: Touch Tone Filter

Author:
Brian Mere, Thomas Choboter
Date:
December 2, 2023

CAL POLY

Abstract

We used a common Goertzel Algorithmic design to create a touch-tone analyzer, to predict what tone is being pressed within a given signal of a variety of frequencies. We detail our design below:

Results

Overall, we got an ISI error of `16.1%` on the fastest signal, and got an even better `6.5%` error on the slower signal. The following
For our filter coefficients, we used a $b = 20$ value for our bits as a good guideline. Increasing the number of bits did very little past this point, as it only increased the resolution of our data points, and not necessarily the correctness of them.
What follows is our runs on both `dtmf_signals_slow.txt` (see Figure 1) and `dtmf_signals_fast.txt` (see Figure 2) each:

High-Level Design

Our filters use an algorithm known as **Goertzel's Algorithm**. It essentially boils down to:

1. An IIR Bandpass filter centered at some frequency ω_0.
2. An FIR filter to extract the DFT from the incoming data.

The IIR Bandpass filter is described as follows via $s[n]$:

$$s[n] = x[n] + 2\cos(\omega_0)s[n-1] - s[n-2]$$

Notice that this is just our $H(z)$ Bandpass filter, with values of ω_0 for the center frequency, as well as using $r = 1$.
The FIR filter converts our $s[n]$ into $y[n]$ so that we can get the DFT at our ω_0 by calculating $y[N]$ for some input signal of length N. It's defined as:

$$y[n] = s[n] - e^{-j\omega_0}s[n-1]$$

The math behind how this caluclates the DFT is as follows. Notice that we can find the Z-transforms for both S and Y as follows:

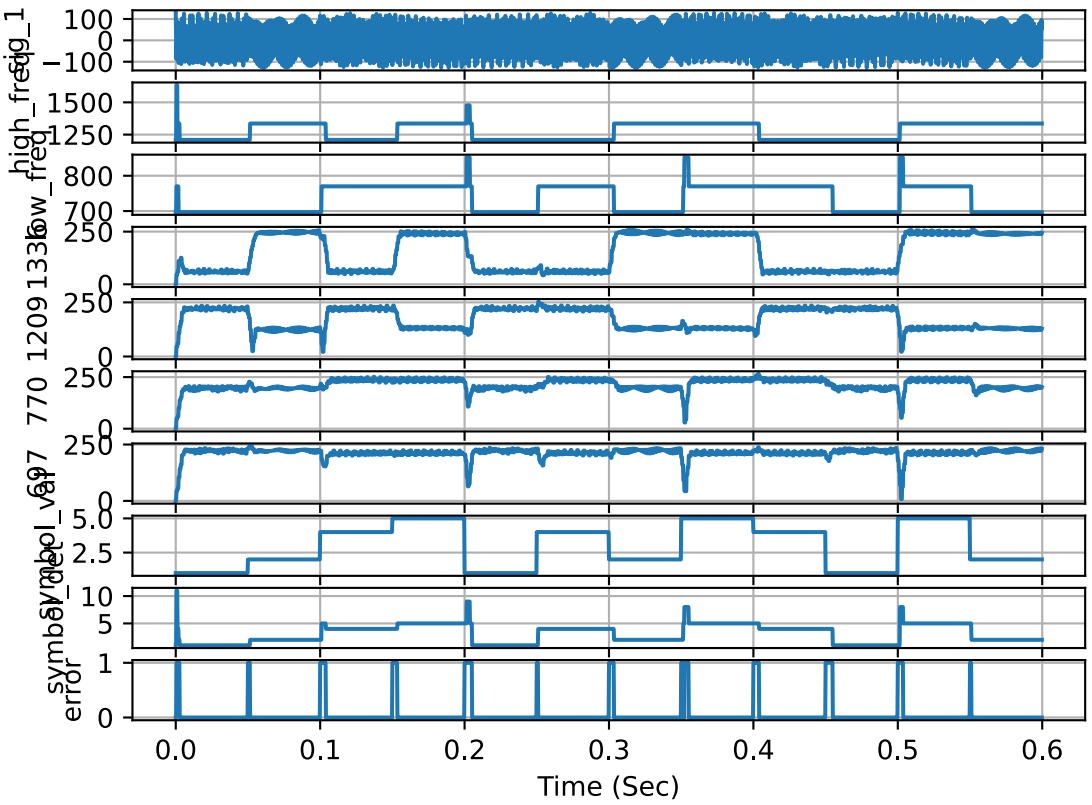

Figure 1: Output from `dtmf_signals_slow.txt`

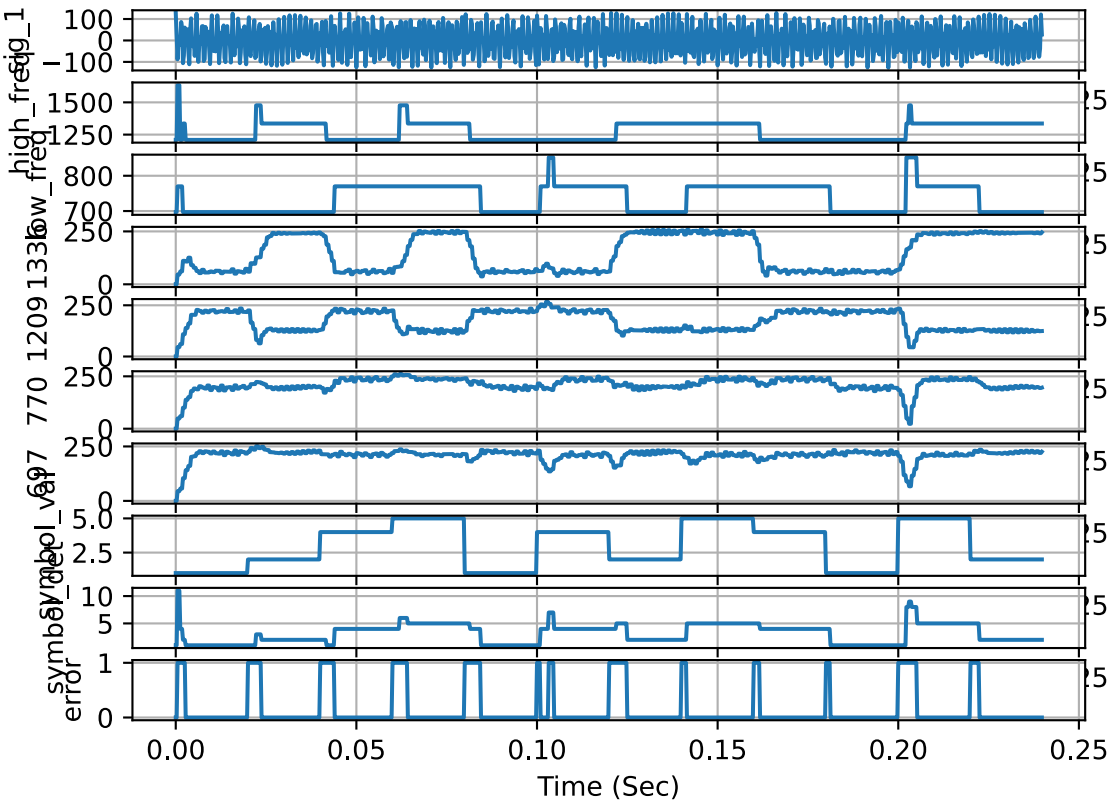

Figure 2: Output from `dtmf_signals_fast.txt`

$$\begin{aligned}\frac{S(z)}{X(z)} &= \frac{1}{1 - 2\cos(\omega_0)z^{-1} + z^{-2}} \\ &= \frac{1}{(1 - e^{j\omega_0}z^{-1})(1 - e^{-j\omega_0}z^{-1})}\end{aligned}$$

$$\frac{Y(z)}{S(z)} = 1 - e^{-j\omega_0}z^{-1}$$

Combining the two filters gives:

$$\begin{aligned}\frac{S(z)}{X(z)}\frac{Y(z)}{S(z)} &= \frac{Y(z)}{X(z)} \\ &= H(z) \\ &= \frac{1 - e^{-j\omega_0}z^{-1}}{(1 - e^{j\omega_0}z^{-1})(1 - e^{-j\omega_0}z^{-1})} \\ &= \frac{1}{1 - e^{+j\omega_0}z^{-1}}\end{aligned}$$

Which can be transformed back into:

$$\begin{aligned}y[n] &= x[n] + e^{+j\omega_0}y[n-1] \\ &= \sum_{k=-\infty}^{n} x[k]e^{+j\omega_0(n-k)}\end{aligned}$$

We impose that when $k < 0$ that $x[k] = 0$, which is usually the case, so then:

$$\begin{aligned}y[n] &= \sum_{k=0}^{n} x[k]e^{+j\omega_0(n-k)} \\ &= e^{+j\omega_0}\sum_{k=0}^{n} x[k]e^{-j\omega_0 k}\end{aligned}$$

Where the right side is just the DFT! The power of this algorithm is that we want to just calculate $y[N]$, but notice that we can substitute the definition of $s[n]$ to get:

$$\begin{aligned} y[N] &= s[N] - e^{-j\omega_0} s[N-1] \\ &= (2\cos(\omega_0) s[N-1] - s[N-2]) - e^{-j\omega_0} s[N-1] \\ &= e^{+j\omega_0} s[N-1] - s[N-2] \end{aligned}$$

Thus, the algorithm essentially boils down to:

1. Calculate, using $x[n]$ all $s[0] \to s[N-1]$ terms using the definition of $s[n]$.
2. Use the shortcut described above to get $y[N]$
3. Return $|y[N]|$ as your DFT magnitude for ω_0.

We do note that $\omega_0 = 2\pi\frac{k}{N}$, so the frequency you want to measure must satisfy that k is an integer in the equation. Usually, even if some f we want to measure isn't an integer multiple, we take the nearest k as a good approximation.
As such, a high-level diagram of our design is as follows:

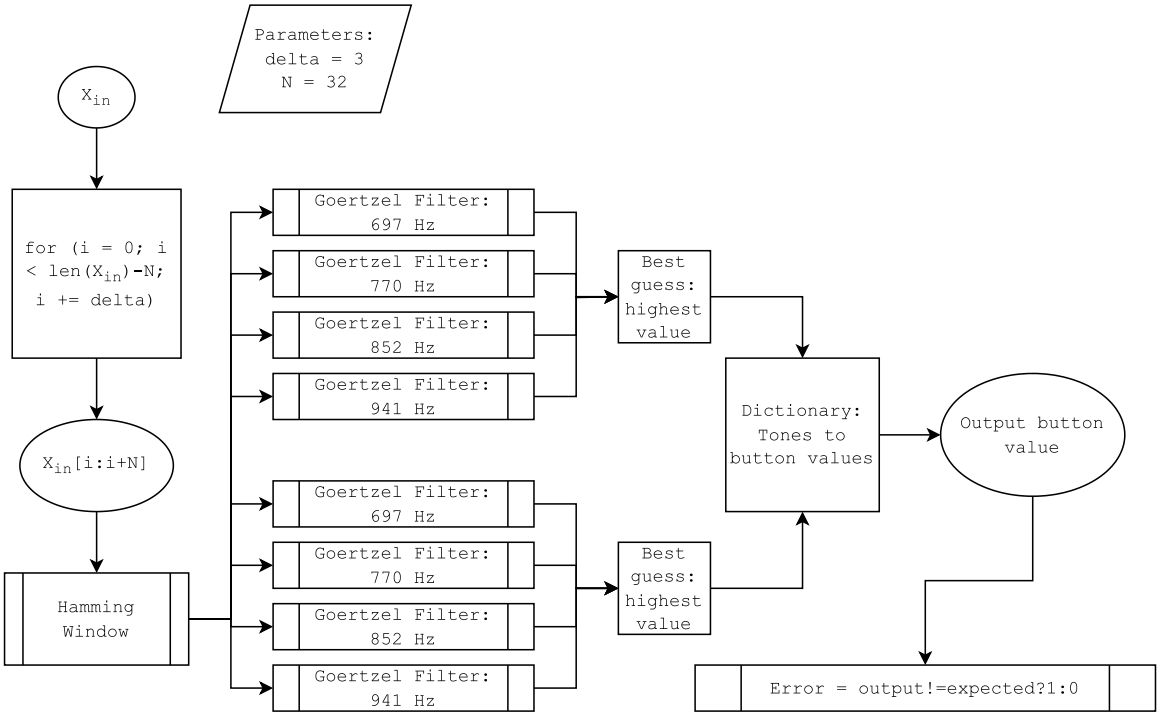

Figure 3: High Level Diagram

We define each "group" of frequencies we want to analyze and choose a "best" values as a `GoertzelComb`. Here, in this case the frequencies `697, 770, 852, 941` form a `GoertzelComb`.
As such, as long as `GoertzelFilter` works, we get a best guess for each group of frequencies, giving us the tone that we detect.

Low-Level Design

Implementing the difference equations above, we get the following low-level diagram for our design:

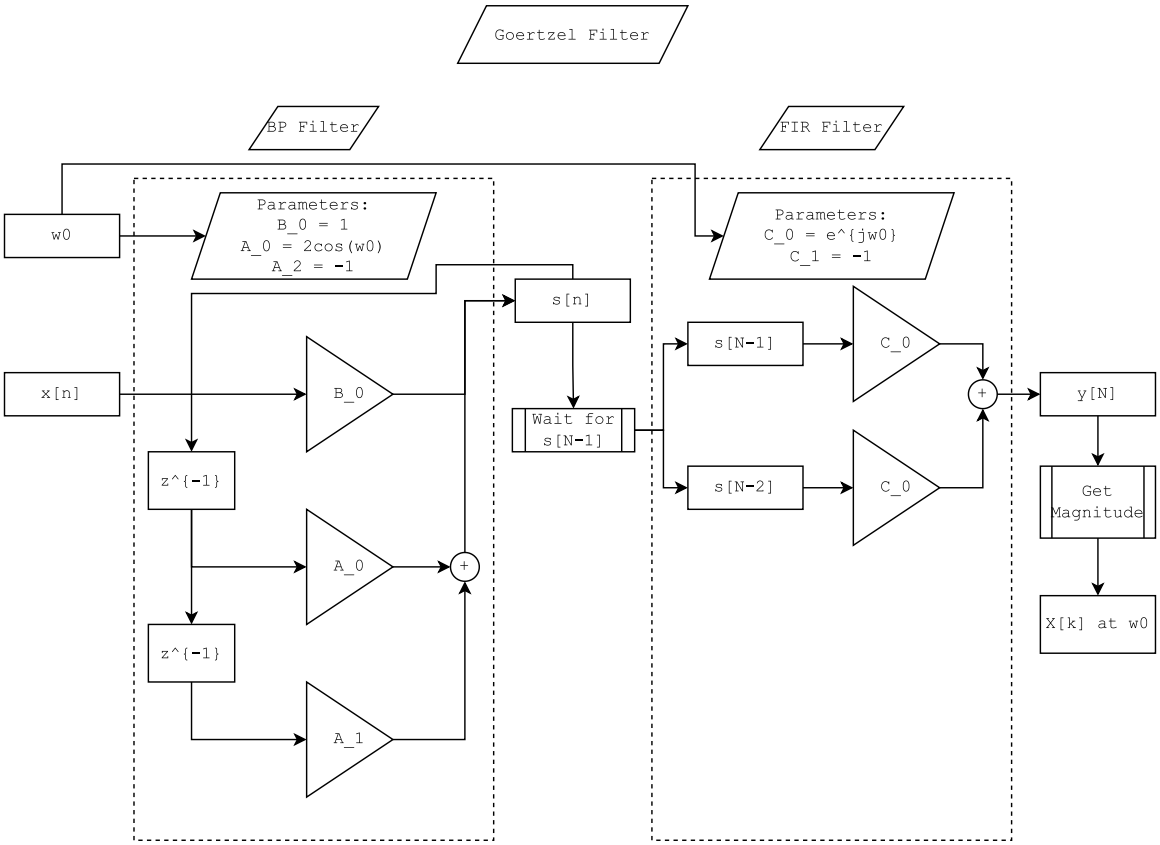

Figure 4: Low Level Diagram

Our `DSP` for using integer arithmetic is as follows:

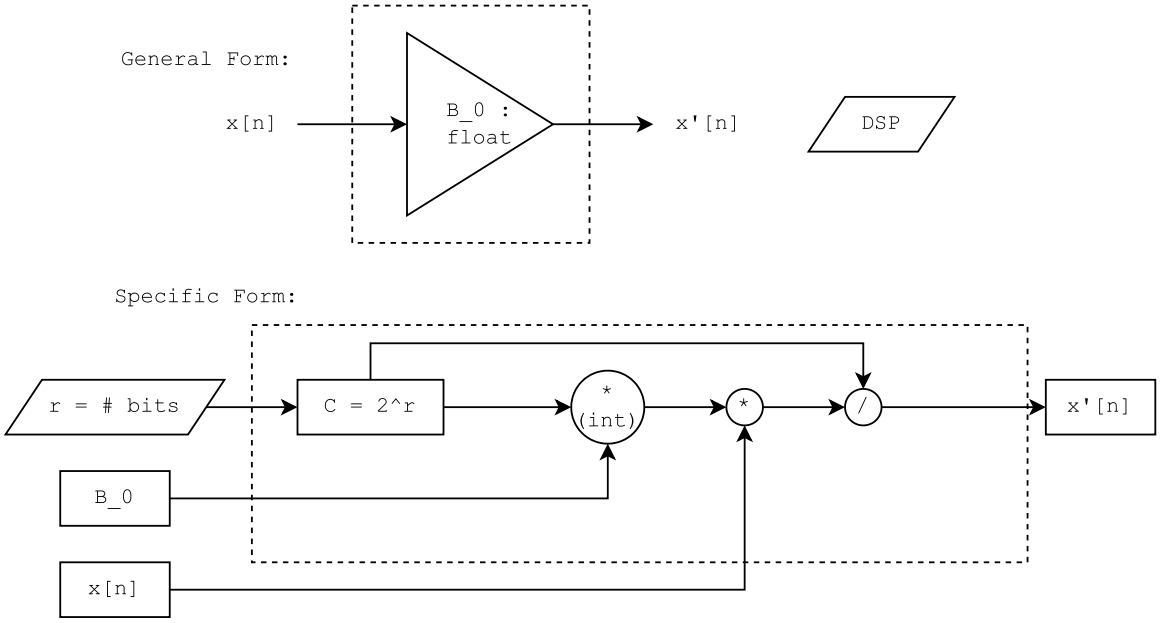

Figure 5: DSP Diagram

Coefficients Used

We note that we generalized a lot of the process of finding the coefficients used for most difference equations. As such, the equations are generated "on-the-fly" based on the input frequencies we need. The good thing is that all frequencies are constant, so we can generate a list of the constants used. As such, we used the following `float` constants, and used the `DSP` above to get the following coefficients:

```
Coefficients for ALL filters: (Using r = 20, N = 32)
-------------------------------------------
b0 = 1048576 -> b0(float) = 1.0
a1 = 1048576 -> a0(float) = 1.0

Coefficients for each filter by frequency:
===========================================

Low-Frequencies:
----------------
f = 697Hz  -> a0 = -802545, a0(float) = -0.7653668647301797
```

```
f = 770Hz  -> a0 = -802545, a0(float) = -0.7653668647301797
f = 852Hz  -> a0 = -409134, a0(float) = -0.39018064403225666
f = 941Hz  -> a0 = 0      , a0(float) = -1.2246467991473532e-16

High-Frequencies:
----------------
f = 1209Hz -> a0 = 802545 , a0(float) = 0.7653668647301795
f = 1336Hz -> a0 = 1165115, a0(float) = 1.111140466039204
f = 1447Hz -> a0 = 1482910, a0(float) = 1.414213562373095
f = 1633Hz -> a0 = 1743718, a0(float) = 1.6629392246050907
```

These come from the following `matlab` file to generate the coefficients and the following magnitude-phase plots:

```
function getBP(f_m)

N = 32;
f_s = 4000;

% Coefficients...
b0 = 1.0;

k = floor(0.5 + (N * f_m / f_s));
w = 2 * pi * k / N;
a0 = -2 * cos(w);

a1 = 1.0;

fvtool([b0], [1, a0, a1], "Fs", f_s);
disp(a0);

end
```

Which gives the following plots:

Codebase

We used a lot of files for this project, and don't want to fill up the contents of this report. Please check out **this link (click on me)** for a full list of current files used in our design (use the `main` branch).

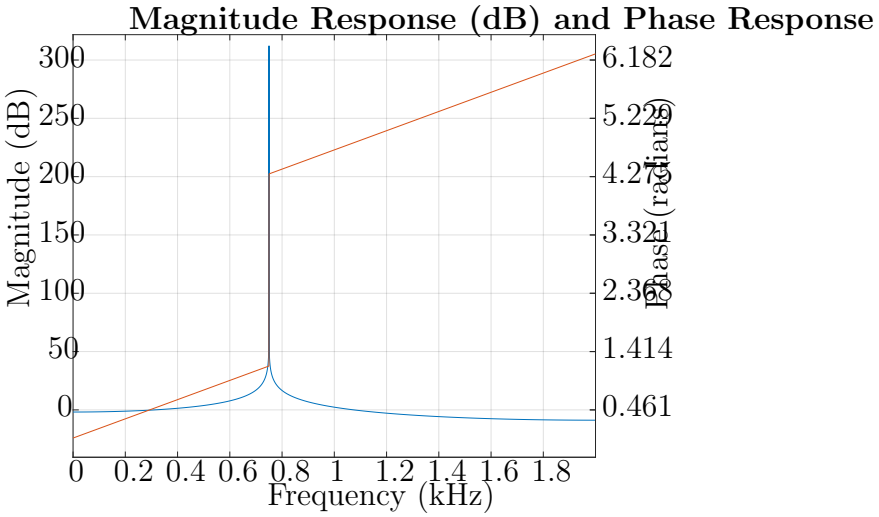

Figure 6: $f = 697$ Bandpass Filter Response

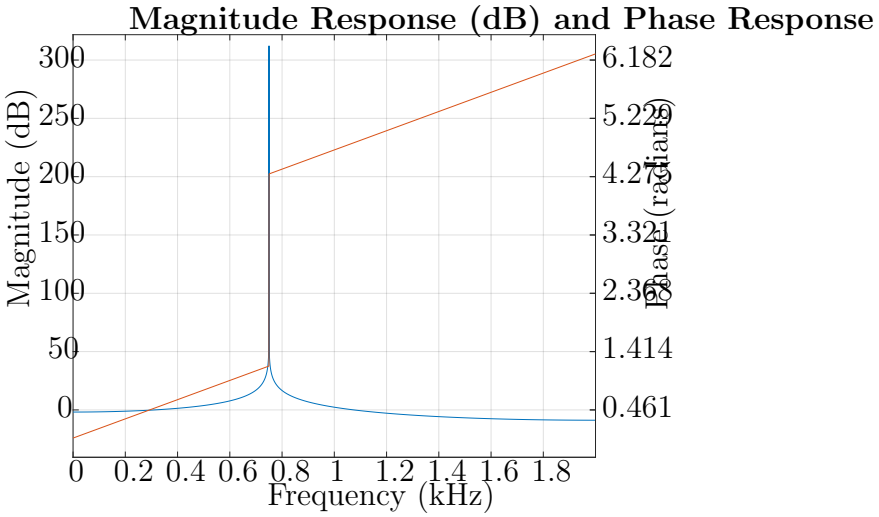

Figure 7: $f = 770$ Bandpass Filter Response

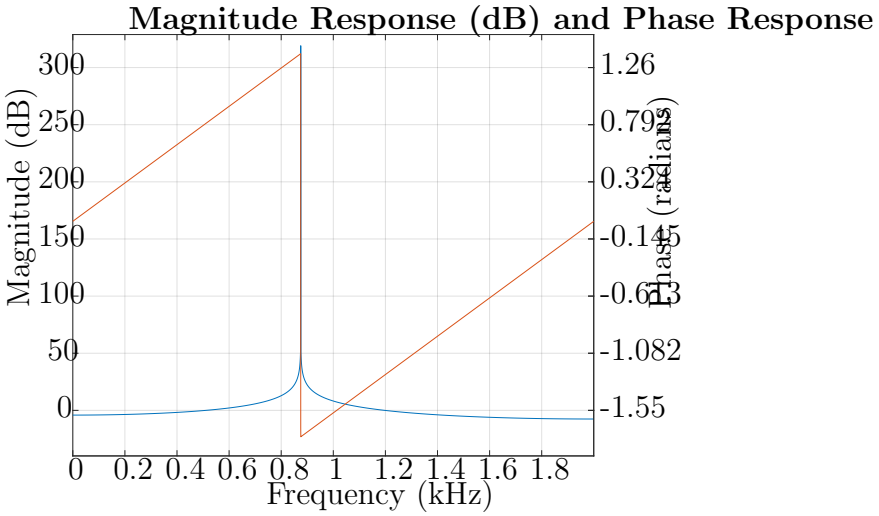

Figure 8: $f = 852$ Bandpass Filter Response

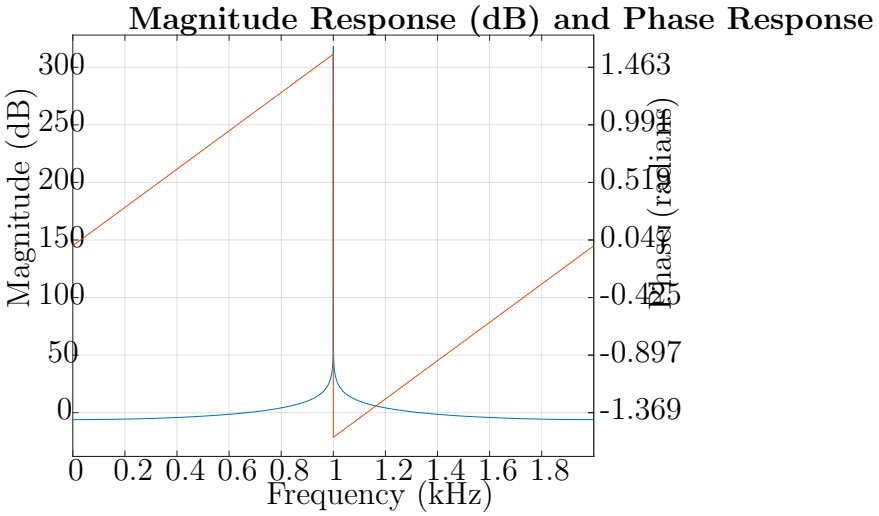

Figure 9: $f = 941$ Bandpass Filter Response

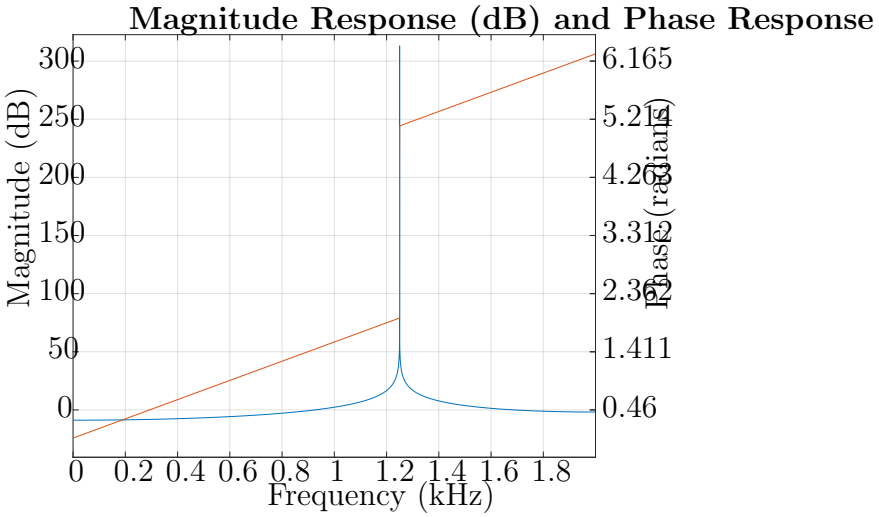

Figure 10: $f = 1209$ Bandpass Filter Response

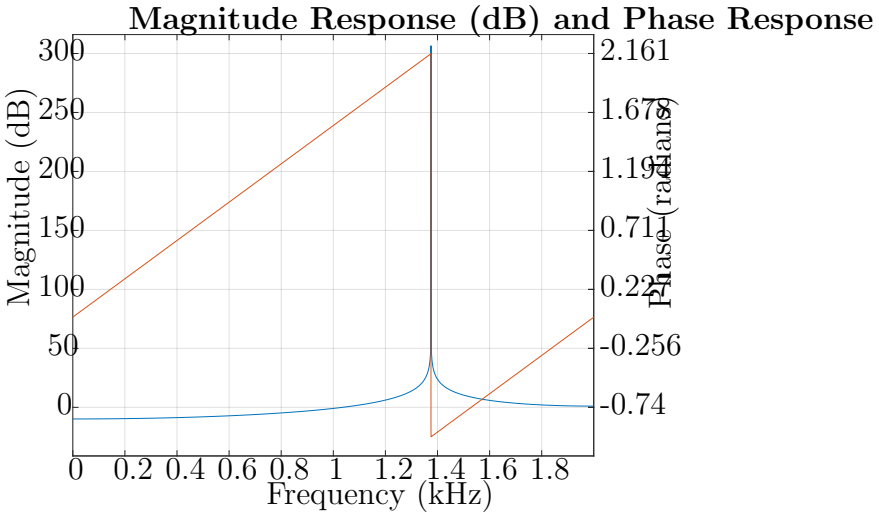

Figure 11: $f = 1336$ Bandpass Filter Response

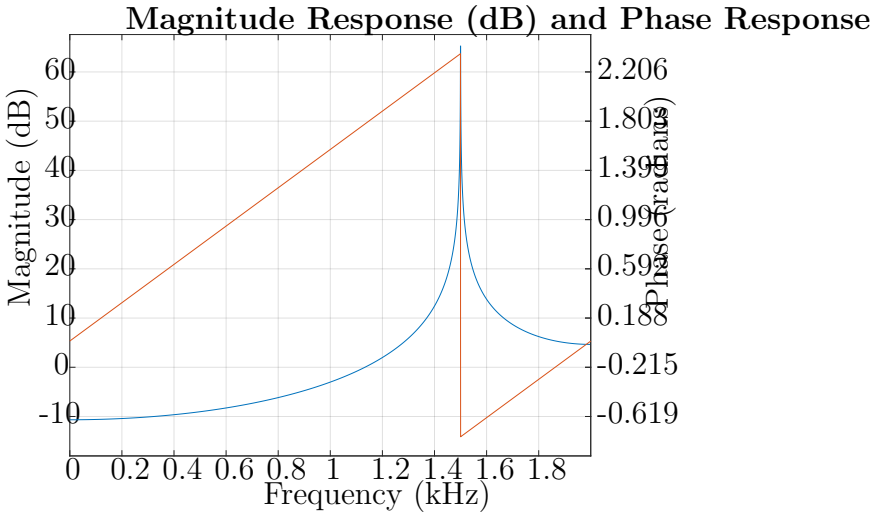

Figure 12: $f = 1477$ Bandpass Filter Response

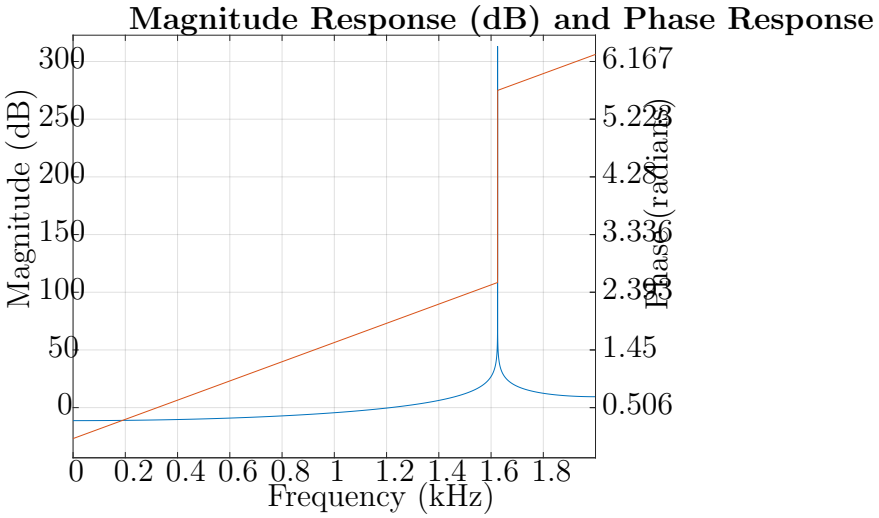

Figure 13: $f = 1633$ Bandpass Filter Response

Notice the following:

- The *true* center frequencies of the bandpass filters are *not* the values of f_m we pass in. Rather, due to the nature of the DFT, we have to use an integer multiple of $2\pi/N$, so the frequency must get rounded to the nearest value of ω_0 such that k is indeed an integer.

- The impulse repsonse itself is periodic, so it should *never* reach a stable state of decreasing by 50% or less by some unit of time.

Additional Discussion

Some things of interest that were covered during this project:

1. We had to really nail down using a correct Goertzel Algorithm. Many approaches *actually* don't use complex arithmetic for their calculations, but convert to separate real and imaginary numbers for the values of question.

2. We tried to before use a BP filter *before* sending it to `Goertzel`; however, it's better not to overfilter the signal as otherwise the input is very sensitive to deviations away from the target frequency. As such, we used the fact that `Goertzel` itself contains a BP filter as a justification of not using an external one into it.

3. For outputting no-symbol, we hypothesized that we could try to output a no-symbol signal *if* all of the $X[k]$ for each frequency in question is about the same. If that's the case, it's most likely that there is no symbol, as the frequencies we are detecting are likely just noise.

4. It's good that we used the DTMF standard values for frequencies to detect. This is because they're pretty relatively spread out, so the values of k for each frequency is, at the very least, unique (for some big enough N).

5. Some ways to improve this design includes trying to use different N values for the rows and columns of frequencies. We could use a smaller N value for higher frequencies (when trying to choose k), so to have a more up-to-date way of updating the high and low frequencies, we could offset these N values to have one update at faster times.