Jared Nixon

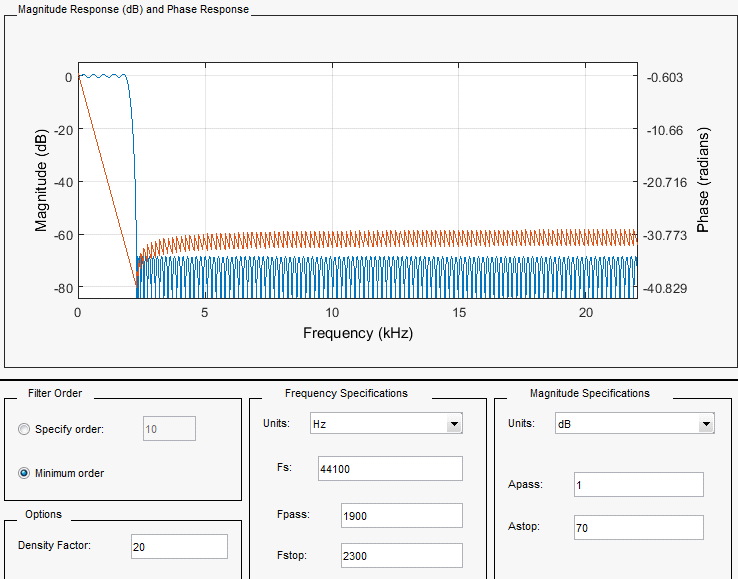
Project Phase 2 Report

April 11, 2016

1. **Description of Problem:** The task of Phase Two was to implement C/C++ code that is designed to eliminate the sine wave that was added to the WAV file from Phase One. This wave is approximately a quarter amplitude of the original file and has a frequency of 2500 Hz. The sine wave was directly embedded within the samples data values of the original WAV file, so due to the nature of the problem, the sine wave must be eliminated from the left and the right channels because the file is stereo and not mono. The program must also support two sampling frequencies, my case being 22050 Hz and 44100 Hz, otherwise have an error outputted.

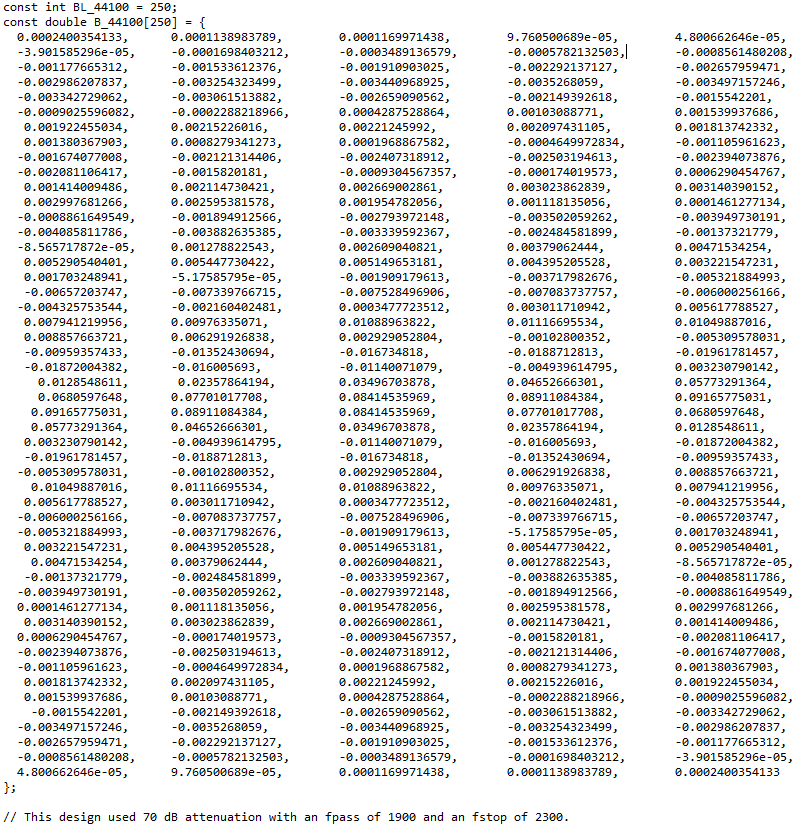
**Proposed Solution:** In order to achieve the task at hand, a low pass filter needs to be implemented to kill the sine wave at 2500 Hz with attuenation of a minimum 60 dB. This can be done in Matlab, then the filter coefficients can be directly exported to a C header file, which can then be run with the program. This program will process the left and the right channels separately by multiplying the filter coefficients with the sample values. Keeping a history of the samples is a requirement of this program, because there must always be X amount of samples if there are X amount of coefficients, and they must all be summed together to calculate the output value.

2. **A. Filter Characteristics for Magnitude and Phase Reponses:**



**B. Filter Choice:** The filter that was selected to implement was a low pass FIR filter. The FIR filter was used because it has a very simple implementation process, because the output is strictly based only on current input, whereas the IIR filter is based on current input, as well as previously calculated output values. If this filter were going to be designed for a microcontroller, the IIR filter would have been a much better choice due to the fact that it is composed of significantly less coefficients. In the FIR implementation, at 60 dB attenuation with Fpass = 2100 Hz and Fstop = 2400 Hz, the sine wave could still be heard faintly. In order to take that into account, the attenuation was bumped up to 70 dB, and the transition band was widened from Fpass = 1900 Hz to Fstop = 2300 Hz, so that the number of coefficients was still close to the same. This is the ultimate trade off for these FIR filters. When a transition band is steep (closer to ideal), a lot of coefficients are needed, and when the attenuation is high, a lot of coefficients are needed. A steep transition band was not necessary for this implementation, nor was a very high attenuation. The attenuation is set just enough to kill the sine wave, but not have a muffled voice, and the transition band is set as a more gradual approach in order to limit the number of coefficients required.

**C. Filter Coefficients:**



**D. Basic Organization of the C/C++ program:**

Initialize all arrays to all 0s;

read and write header file;

read first sample for left and right channels;

while(!feof(inFile))

{

store sample in [0] array spot for left and right input arrays;

pass to FIR function(see below) by pointers to left and right arrays as parameters;

write filtered left and right values to filtered WAV file;

shift left and right input arrays to hold the next "NOW" sample in [0]th position;

read in next samples left and right channel;

}

FIR(\*left, \*right)

{

double tempLeft = 0;

double tempRight = 0;

for(int i = 0; i < BL\_44100; i++)

{

tempLeft =+ **left[i] \* B\_44100[i];** // Basic FIR computation.

}

for(int i = 0; i < BL\_44100; i++)

{

tempRight =+ **right[i] \* B\_44100[i];** // Basic FIR computation.

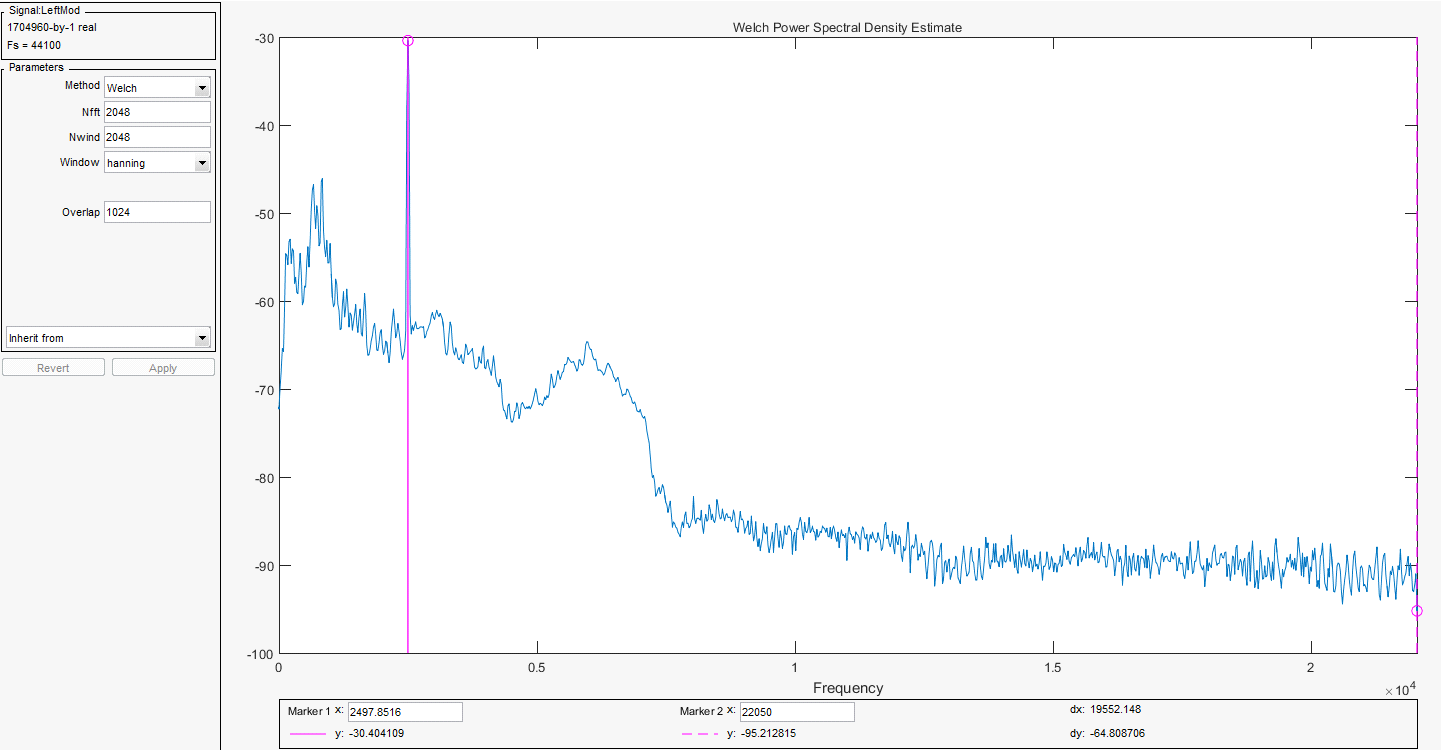
}

outLeft = (signed short)tempLeft;

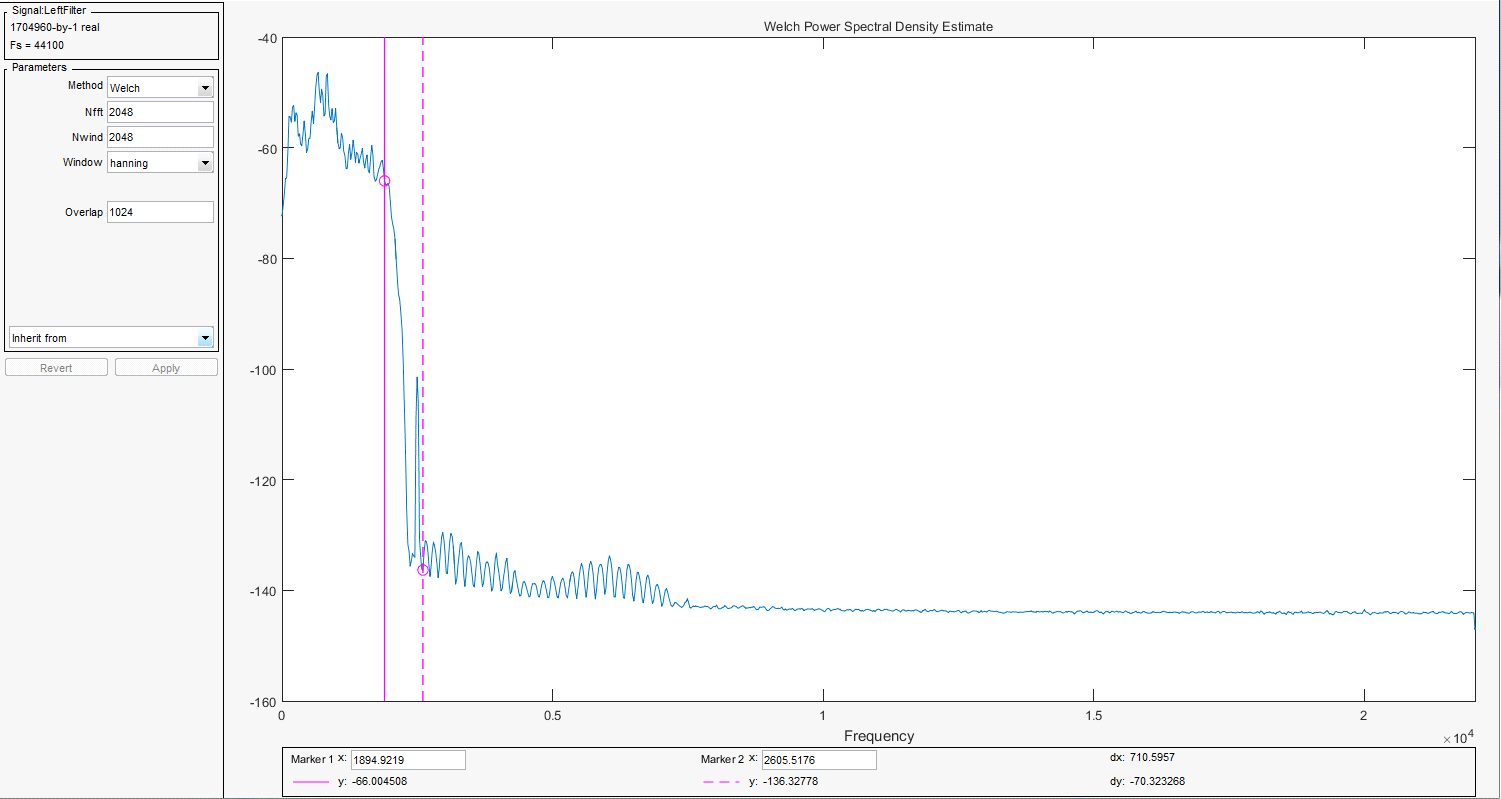
outRight = (signed short) tempRight;

}

3. **Spectrum of the input (modified) WAV file:**



**Spectrum of the output (filtered) WAV file:**



As seen in the above screenshot, in the modified spectrum, the 2500 Hz sine wave is clearly the dominant spectrum of the entire file. Now looking at the filtered spectrum, the 2500 Hz sine wave is significantly reduced by the FIR filter. Taking note of the values at the bottom (where the pink trackers are located), it can be seen that the Fpass = 1900 Hz and Fstop = 2300 Hz is where the transition band is located on the filtered spectrum, so this verifies that the implementation of our code is working as it should. It is important to note that this filter still is not perfect. There are still frequencies after our stop band. The fact of the matter is the quality of the filter is directly associated with the transition band and the attenuation (dB). More coefficients (aka steeper transition and higher attenuation) may kill these frequencies more efficiently, but that would come with a possible cost of 700-100 coefficients, whereas the implementation used here is only 250, not to even mention how long the processing would increase if 700-1000 coefficients were used. Only the left channel spectrum of the filtered signal is shown above, but the right channel is almost identical to the left, so this is sufficient.

**Matlab code used to plot spectrums:**

clear

[y, fs] = audioread('c:\Users\xJBiRDx\My Documents\Visual Studio 2015\Projects\Project2\Project2\Nixon\_44\_Mod.wav');

left = y(:,1); // Breaks the WAV file into left and right channel.

right = y(:,2); // Breaks the WAV file into left and right channel.

sptool

clear

[x,hs] = audioread('c:\Users\xJBiRDx\My Documents\Visual Studio 2015\Projects\Project2\Project2\Nixon\_44\_Filter.wav');

left = x(:,1); // Breaks the WAV file into left and right channel.

right = x(:,2); // Breaks the WAV file into left and right channel.

sptool

4. **Performance and real time:** I believe this program can run in real time. The fact of the matter is, the file is 38 seconds long, but completely processed and filtered in about 4.5 seconds. This is extremely fast and should be fast enough to influence a real time system by making the output feed right back into the input.

**Summary text file of program:**

CPE381 project - Processing WAV File - Jared Nixon

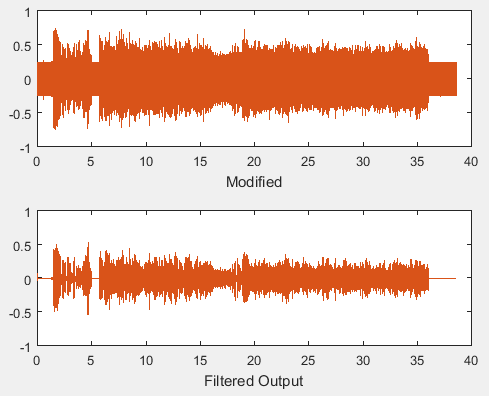
The sampling frequency of this WAV file is 44100 Hz.

The length of this file in seconds (calculated from the number of samples) is 38 seconds.

This program completed the processing of the WAV file in **4.521** seconds.

5. **Experience and Lessons Learned:** The implementation of Phase One and Phase Two is without a doubt the best project experience I have had thus far at UAH. Because of the instant results upon compiling and listening to the output WAV file, the drive is high to keep pushing to write and debug code until successful results are heard. The major lesson learned is how to implement FIR/IIR filters in C/C++, which is something that will forever be burned in the brain. Regardless of only implementing the FIR, research was done upon both of them in order to decide which one to implement. The educational programming experience behind this project was enormous and learning how to use Matlab to see our results visually was a much added bonus. For a personal case, I learned that when I believe my code definitely should be working, to always check my input file, because that was the corruption of my entire program. Upon rerecording a WAV file through Audacity (instead of Windows 7 sound recorder), my program worked like a charm. Another major lesson learned was to pass arrays by pointers when passing them to a function, so that there is no risk of overwriting them on the stack and losing important data, because they will be stored directly in memory. That is just a good programming technique learned in general, not just specific to this program.

6.  **Matlab plots of modified and filtered signals:**



The graphs above also indicate that the implementation of the FIR filter was correct. As seen, the noise in the modified signal is significantly reduced in regards to the filtered output. This ensures that the coefficients generated by Matlab were correct. This does not mean, however, that the filter is perfect, as stated above. There are always trade offs to improve it.

**Matlab code used to plot signals:**

[y,fs] = audioread('c:\Users\xJBiRDx\My Documents\Visual Studio 2015\Projects\Project2\Project2\Nixon\_44\_Mod.wav');

t = [1/fs:1/fs:length(y)/fs];

[x,hs] = audioread('c:\Users\xJBiRDx\My Documents\Visual Studio 2015\Projects\Project2\Project2\Nixon\_44\_Filter.wav');

z = [1/hs:1/hs:length(x)/hs];

subplot(2,1,1);

plot(t,y);

xlabel('Modified');

subplot(2,1,2);

plot(z,x);

xlabel('Filtered Output');

**Notes:**

* All of the data recorded above was for my 44100 Hz sampling frequency input files.
* Can supply documentation and analysis for left and right channels at 44100 Hz and 22050 Hz sampling frequencies if needed, i.e, spectrums, codes, coefficients.