Real Time Spectral Processing in RackAFX

Andrew Jackson Stockton X

Music Engineering, University of Miami
Miami, FL

stocktonx@me.com

***Abstract*— This AppNote will cover the basis for real time signal processing in the frequency domain. A known method for efficient frequency domain processing is the implementation of the Short Time Fourier Transform (STFT). The STFT algorithm is very popular in many fields such as: speech, communications, vocoding, compression, etc.**

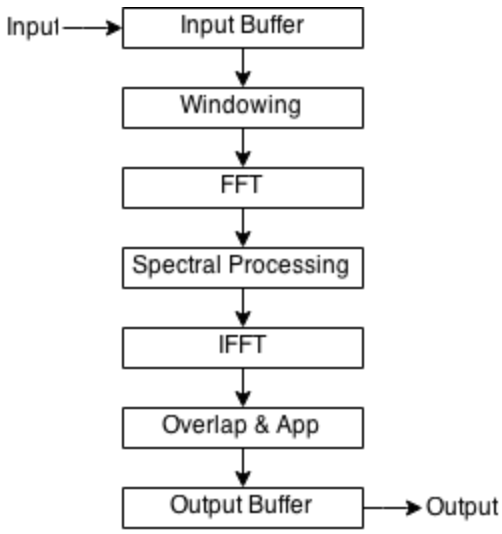
***Keywords*— STFT, Spectral Processing, Windowing, Magnitude, Phase, FFT**

I. Introduction

The Fast Fourier Transform is an efficient algorithm for calculating the magnitude and phase of a given signal. Although efficient, it is very CPU intensive to perform the FFT for every new sample we receive. This paper will cover the use of the Short Time Fourier Transform, which performs an FFT and IFFT on windowed buffers instead of individual frames.

II. Block Diagram

The STFT can be broken up according to the block diagram below:



III. STFT Implementation

A. Input Buffer

The input is read into an Input Buffer, and continually shifted through. This is essentially a delay block.

B. Windowing

A counter is set to loop from 0 to the length of our window. Everytime the counter resets, we will take the data from the Input Buffer, perform a windowing operation, and store it in our Window buffer. The window design I chose is the Hanning window, of which we pre-calculate the values into an array for efficiency. The Hanning window was chosen because with 50% overlap, we will achieve unity gain.

Multiplication in the time-domain is convolution in the frequency domain. If we didn't include a windowing function, we are essentially multiplying by a rectangle. This rectangular pulse transforms into the sinc function in the frequency domain. Convolution with a sinc function will produce an oscillation and smearing in the frequency domain. Windowing can reduce this "ringing" and smearing.

C. Fast Fourier Transform

The next step involves performing an FFT on our windowed buffer. In my project, you can use Will Pirkle's FFT function or my optimized FFT function. The FFT function will output the real and imaginary components of our input data. The next step involves calculating the magnitude and phase from the real and imaginary components. All of these calculations can be viewed in the attached code. Note: The default window size is 512, and the FFT length is 1024. The windowed data is zero-padded to perform the FFT. Also, the minimum length of the FFT should be window length if perfect reconstruction is desired.

D. Spectral Processing

Once we obtain the Magnitude and Phase, which are stored in buffers, spectral processing can be performed! The first two options in my STFT project are a Low Frequency Shelf and a High Frequency Shelf. The basis of operation is very simple. If the frequency bin number is below and above a specific cutoff frequency, we will attenuate the magnitude in that bin. Another option I included is a very crude pitch shifting algorithm, which stretches out the magnitude and phase bins. It uses linear interpolation, but the resulting output is very distorted. The last algorithm I included is a simple swapping of the magnitude and phase bins. All new magnitude and phases are stored in m_pMagShift and m_pPhaseShift.

E. Inverse Fast Fourier Transform

After performing operations in the frequency domain, we must perform the inverse fast fourier transform to convert our signal back into the time domain. Will's FFT function will allow us to calculate the iFFT.

F. Overlap-and-Add

Overlap-and-Add is the last essential step for reconstructing our signal. Since we used the Hanning Window, we will use an overlap-and-add ratio of 50%. The final result is stored in an Output Buffer.

G. Output Buffer

The Output Buffer is read out backwards due to our housekeeping.

IV. Results

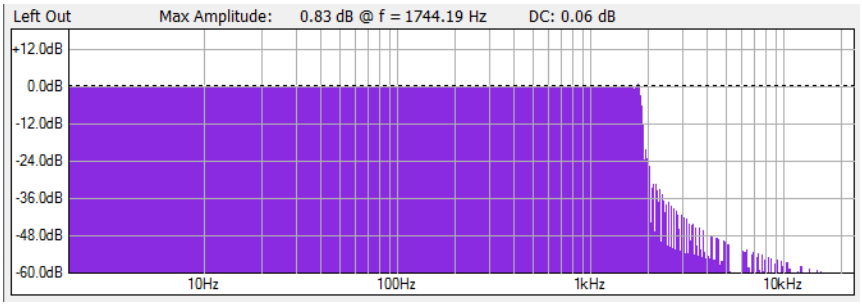


Fig. 1 Frequency Response of our High Shelving Filter at maximum cut

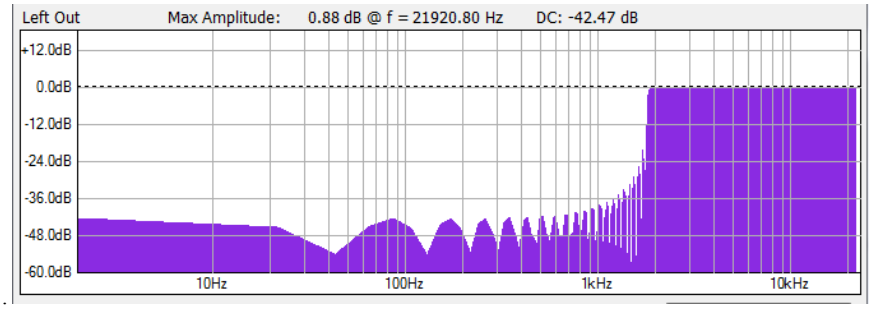


Fig. 2 Frequency Response of our Low Shelving Filter at maximum cut

V. Conclusions

In conclusion, perfect reconstruction of the input signal was achieved, and implementation of a few basic algorithms was successful. Further improvements can be done in the following fields: stereo implementation, different windowing functions, optimal balance of the FFT lengths and the window lengths, new-and-improved spectral processing algorithms, the use of circular buffers, and code optimization.

References

[1] Will Pirkle, *Designing Audio Effect Plug-Ins in C++,* Focal Press 2013

[2] D. G. Manolakis and V. K. Ingle, *Applied Digital Signal Processing,* Cambridge University Press 2011

[3] http://research.cs.tamu.edu/prism/lectures/sp/l6.pdf

[4] http://sethares.engr.wisc.edu/vocoders/phasevocoder.html