Phase Vocoder

EE 443 Final Project Proposal

Jisoo Jung and Jake Garrison

Motivation

Digital audio processing is a part of everyone's lives whether it is in the form of noise filtering on a cell phone, voice amplification at a crowded political speech, pitch correction in the latest pop single, or synthetic speech found in a futuristic film. Many of these effects typically rely on expensive computer software with a steep learning curve. With the advent of efficient algorithms like FFT and modern microprocessor breakthroughs, many of these effects can now be packaged in the form of low power hardware and still operate with minimal latency.

We are interested in implementing a phase vocoder, which is a type of vocoder which can scale both the frequency and time domains of audio signals by using phase information. With this method, we hope to achieve realtime time stretching and pitch shifting effects on audio input. This method relies on the short-time Fourier transform (STFT), which converts a time domain representation of sound into a time-frequency representation, allowing modifications to the amplitudes or phases of specific frequency components of the sound. The advantage of this method is that pitch shifting does not affect the input vector length, which means that the play time of an audio file remains the same after the modification.

The pitch/time stretching effect dates back to the analog days when records could be played slower or faster, thus changing the pitch. The issue with these transformations is that time is also affected. It was not possible to change the pitch without affecting the time domain, or conversely changing the pitch without affecting time. With digital sound, this is now possible, and it unlocks many new, powerful applications.

Applications of this effect span from fun to serious purposes. Pitch shifting and time stretching is used all the time in music to allow an artist to explore multiple personalities with different pitched voices. Pitch correction is often used in the studio to fit vocals to prerecorded instruments. These applications are often hard to replicate in a live performance due to the latency and hardware complications. We hope by implementing on a FPGA, we can enable these effects in a real time performance setting. To create science fiction characters, hollywood studios will often layer several different pitched versions of the same sample to create robotic, synthetic speech. Our phase vocoder will be able to do this as well. If these multiple voices are tuned to a key, a choir can be generated from a single source (Such as in Sam Smith's Stay With Me). Karaoke can also benefit from pitch shifting so that a different key can be selected that may better fit a singer's vocal range.

There are other applications beyond entertainment. In the interconnected world we live in, privacy is crucial. Soon, identities will be revealed simply through speech. In order to protect privacy, the phase vocoder can be utilized to mask a voice. Shift parameters could theoretically be encoded and decoded with a private key pair so that, while the speech is understandable, only those with the key can reveal the identity. Pitch and time shifting can also be a usability feature to some. Youtube recently enabled a feature to change the speed of playback. Many people with disabilities or language barriers find this useful for understanding speech. Finally, text narrators found on personal operating systems rely on phase vocoders to generate synthetic speech as well.

Many of the applications mentioned are not new to the world. Several different methods for time and pitch shifting have been proposed in the past. The simplest way to change pitch is to simply resample.

Unfortunately, the frequencies in the sample are always scaled at the same rate as the speed, so you can't change pitch without affecting speed. More advanced methods exist that utilize either the time or frequency domain. In the time domain, the synchronized overlap-add method (SOLA or improved PSOLA) algorithm attempts to find the fundamental frequency of a given section of the wave using some pitch detection algorithm, and then crossfades one period into another. Several different pitch detection algorithms exist. SOLA performs somewhat faster than the phase vocoder on slower machines but fails when there are complicated harmonics (autocorrelation misses fundamental frequencies) [3]. In the frequency domain, the most popular method is the phase vocoder. The method is explained in detail in the Problem Formulation section. The phase vocoder technique has improved greatly over time and can be used to perform pitch shifting, chorusing, timbre manipulation, harmonizing, and other unusual modifications. We are utilizing phase vocoding because it maintains good quality for speech and can be done on an embedded device [4].

Problem Formulation

The main challenge in the phase vocoder design is to achieve a high quality sound output with minimal latency in the process. This way human speech is understandable and not delayed from when the speaker speaks.

The program should perform time-stretching without affecting the pitch and pitch-shifting without affecting the audio's playing time. In order to obtain high quality audio after phase vocoding the FFT conversion size should be 4096 or 8192, with higher number of samples resulting better sound quality. With a sampling frequency of 44100 Hz and conversion size of 4096, the expected latency in real-time processing is around 100 ms [1]. Since we are focused on speech, we will most likely to use a sampling rate of 8k or 16k Hz to further improve the latency at the cost of some quality. Filters to cancel noise may also be added.

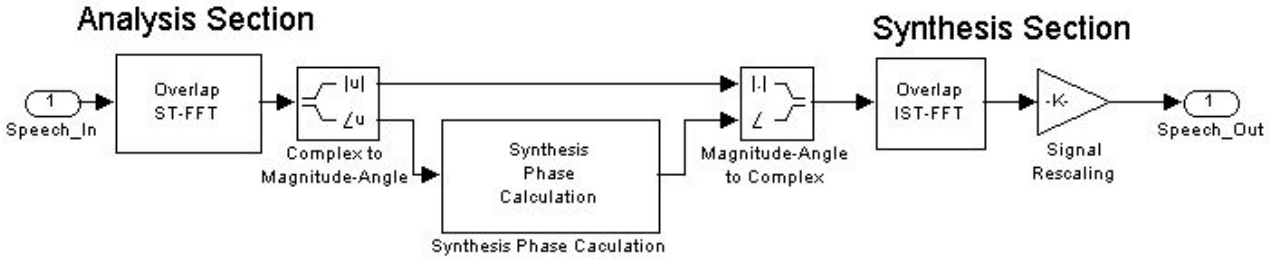

Figure 1. Block Diagram of Phase Vocoder [2]

The figure above shows the overall procedures in phase vocoding and it contains the following steps:

1. Perform STFT on the input samples in time domain to convert into time-frequency domain.
2. Apply desired processing to the Fourier transform magnitudes and phases. This stage is where most of the effects will occur.
3. Perform inverse STFT (ISTFT) on the processed phase and magnitudes to reconstruct the signal back to time domain.

Data Collection and Tasks

Phase vocoders rely on a few key functions, as explained above. To validate our design and algorithms, a Matlab model has been started and will be completed before any C development is done. Currently the Matlab model supports reading an audio sample, performing STFT and ISTFT, and applying the phase vocoder to

adjust the speed or pitch of the input. See the plots below. Many of these functions are borrowed from other academic researchers [5].

Once the expected behavior of our Matlab prototype is achieved, and the code is optimized, a C program will be developed to match the Matlab program's behavior. Having the Matlab ahead of time allows us to validate our C code against a working model. The C functions will be tested in software, then migrated to the FPGA over time. To ensure optimal behavior, the samples computed on the FPGA will be stored in a buffer, which will then be sent to Matlab program via UART for plotting. Visual representation will help us to better understand its performance.

Next, various tasks will be mapped to FPGA's switches and buttons. Each switch or button will be responsible for a task such as decreasing/increasing the pitch or stretching/compressing the play time. For accurate testing, the processed data will be sent to Matlab program for further analysis, but quick and easy testing is utilizing both left and right channel of AIC23 daughter board. One channel will output the original signal while the other channel outputs processed signal. This test is fast and the user can instantly hear the difference after changing a parameter. If time permits, additional, advanced processing may be added in the form of a Matlab GUI.

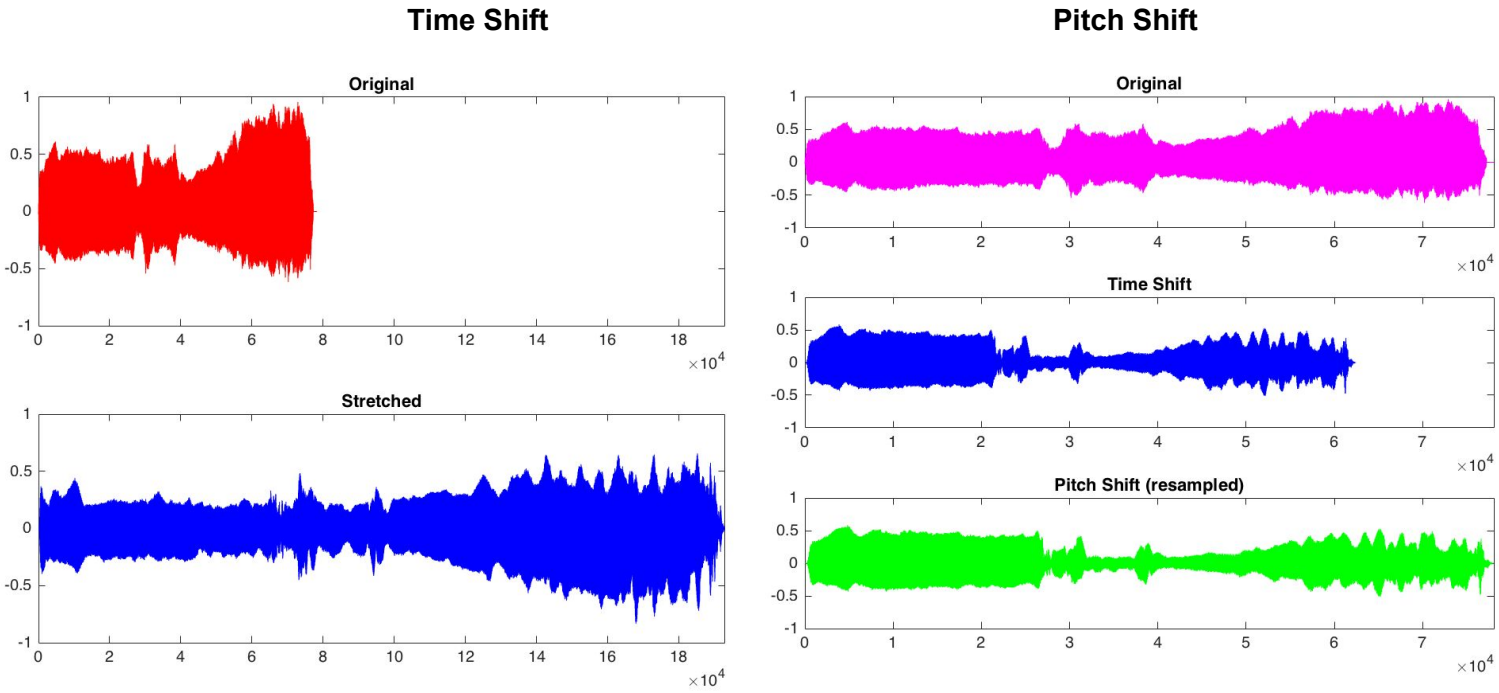

Figure 2. Matlab demo output. Notice how the pitch shifted sample has the same length as the original.

Timeline and Responsibilities

The task outlined in the previous section will mostly be handled collaboratively, however we do plan on delegating different focuses. Jake will be more responsible for prototyping and optimizing in Matlab and C as software. Jisoo, using his embedded experience, will be responsible for getting the code to work on the FPGA and hardware, including the interrupts, UART and various IO. The timeline in Figure 3 below highlights the milestones and tasks. Though most of the preliminary objectives are in series, many of the later tasks can be done in parallel.

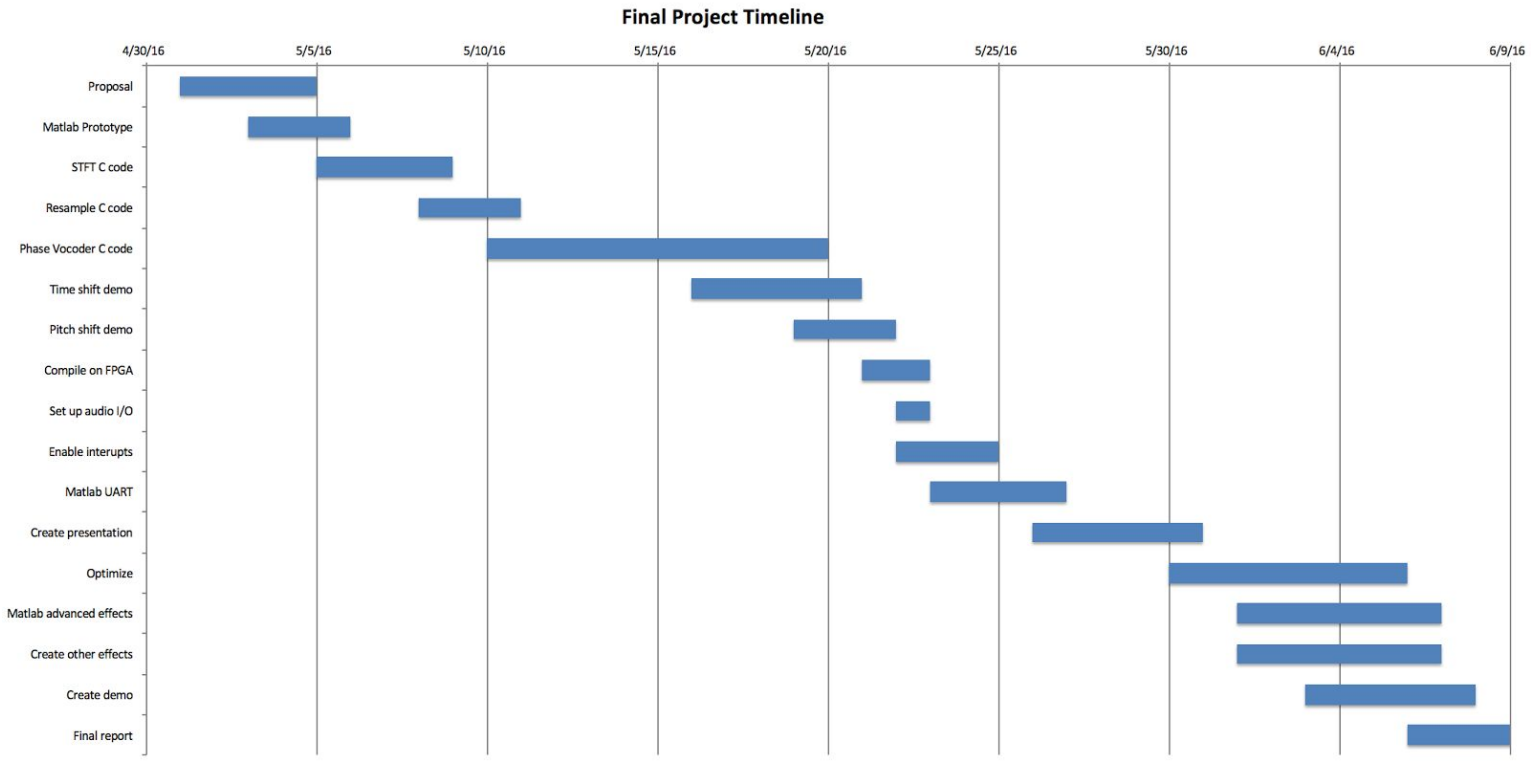

Figure 3. Timeline chart for project

Tasks include: Proposal, Matlab prototype, STFT C code, Resample C code, Phase vocoder C code, Time shift demo, Pitch shift demo, Compile on FPGA, Set up audio I/O, Enable interrupts, Matlab UART, Create presentation, Optimize, Matlab advanced effects, Create other effects, Create demo, Final report

References

[1] O. Parviainen, "article", *Surina.net*, 2016. [Online]. Available: http://www.surina.net/article/time-and-pitch-scaling.html. [Accessed: 05- May- 2016].

[2] "Pitch Shifting and Time Dilation Using a Phase Vocoder in MATLAB - MATLAB & Simulink Example", *Mathworks.com*, 2016. [Online]. Available: http://www.mathworks.com/help/audio/examples/pitch-shifting-and-time-dilation-using-a-phase-vocoder-in-matlab.html. [Accessed: 05- May- 2016].

[3] "Time Stretching And Pitch Shifting of Audio Signals – An Overview | Stephan Bernsee's Blog", *Blogs.zynaptiq.com*, 2007. [Online]. Available: http://blogs.zynaptiq.com/bernsee/time-pitch-overview/. [Accessed: 05- May- 2016].

[4] J. Laroche and M. Dolson, "NEW PHASE-VOCODER TECHNIQUES FOR PITCH-SHIFTING, HARMONIZING AND OTHER EXOTIC EFFECTS", 2016. [Online]. Available: http://www.ee.columbia.edu/~dpwe/papers/LaroD99-pvoc.pdf. [Accessed: 05- May- 2016].

[5] D. Ellis, "A Phase Vocoder in Matlab", *Labrosa.ee.columbia.edu*, 2016. [Online]. Available: http://labrosa.ee.columbia.edu/matlab/pvoc/. [Accessed: 05- May- 2016].