A Speech and Music Detector Project for a DSP Class

Christopher J. Vondrachek, Joseph P. Hoffbeck

University of Portland

Abstract

A project is described in this paper which is designed to monitor a radio station and detect commercials and talking, which would allow the radio to be muted so the listener would not be distracted by obnoxious radio ads and D.J.'s. The project is designed to be an interesting application of a very simple pattern recognition system and requires little more than a low pass filter, high-pass filter, and a threshold scheme. The approach was to attempt to classify the signal as music or speech, and if the signal was found to be speech, the device could mute the radio (advertisements with music would not be eliminated). The device was based on the TMS320C31 DSK, which is an inexpensive digital signal processor (DSP) demonstration board available from Texas Instruments that contains an A/D converter, D/A converter, and a computer interface. The DSP on the demonstration board was programmed to take samples from the A/D converter, pass them through a low-pass and high-pass filter, and to compute the ratio of the average energy at the output of the LPF filter to the average energy of the output of the high pass filter. Since, most of the time, speech has less energy at high frequencies than music does, a high ratio usually indicates speech. The DSP compares the ratio to a threshold value, which was determined from recorded speech and music, and, if the ratio is higher than the threshold, generates a signal that could be used to mute the speech. The technique was found to work well with the test signals used to find the threshold, but did not generalize very well to new music and speech signals.

I. Introduction

The challenge in many courses is not in deciding how best to teach a topic, but in deciding how best to motivate students so that they want to learn the material. The project described in this paper is designed to show an interesting and useful application of DSP that requires little theory, and can be used to capture the interest of the students and motivate them to learn DSP theory. It shows how a low pass filter, a high pass filter, and a thresholding scheme can be used to determine if an audio signal is music or speech so that a radio could automatically mute commercials and other talking. Using the inexpensive TMS320C31 DSK, this simple speech/music recognition algorithm can be run and demonstrated in real time, with real music and speech signals.

The project can be used in courses in several different ways. For example, the students could analyze speech and music samples to determine the filter coefficients and the value of the

threshold for accurately distinguishing between music and speech, and then implement their values using the TMS320C31 DSK. In this case, the DSP program can be given to the students, and they only need to modify the filter coefficients and the threshold value. Alternately, the project can be used as a programming exercise, where the appropriate values of the filter coefficients and the threshold are given to the students, and they write the DSP code to implement the algorithm. In either case, the project is designed to be a simple, yet useful, system that requires little theory to understand.

Although this project is intended to only mute radio commercials, there are many other applications using the same concept, such as a radio that could change stations when a commercial comes on, or could play music from a CD while the radio station was playing commercials.

II. Recognition Algorithm

Human speech is composed almost entirely of tones in the 500-2500 Hz range. Music however, is composed of a relatively wide range of tones from 20Hz to well over 15kHz. This suggests that the spectrum of a signal could be used to determine if it contained music or speech. Furthermore, because the spectrum of speech is dominated by frequencies below 3kHz, it may be possible to identify the sample by just comparing the relative energy of the signal above and below a certain frequency. Most speech will have a higher ratio of low frequency energy to high frequency energy than most kinds of music. The ratio of low frequency energy to high frequency energy was used because the amplitude of the audio signal will not affect this ratio.

With the correct cutoff frequencies and ratio threshold, such an algorithm should be capable of identifying most types of music from most types of speech. By modeling the algorithm in MATLAB, the correct threshold can be determined by analyzing many different samples of music and voice at various cutoff frequencies. The correct cutoff frequency is the one that yields the largest difference between the energy ratios of a given set of music and speech samples. Using the method described above with the values from MATLAB simulations, a simple yet effective DSP algorithm can be developed.

III. Implementation

A Texas Instruments TMS320C31 DSP Starter Kit (DSK) was selected to implement this algorithm. The DSK hardware includes a TMS320C31 (‘C31) DSP running at 50 MHz, and TLC32040 Analog Interface Chip (AIC) that features 14 bit a DAC and ADC sampling at up to 19.2 kHz. Two RCA jacks provide the analog input and output interfaces, and a DB25 connector allows communication with a host IBM compatible PC via the parallel port. The TI DSK also included all the necessary software tools to begin code development using ‘C31 assembly[1].

The development of the recognition algorithm was split into several segments that could be individually coded and debugged. This would be especially useful for students with no

experience with ‘C31 assembly language. It also should be noted that the individual code segments could represent assignments or lab exercises in a DSP programming class.

The first code segment was a simple program that simply echoed an analog signal from the DSK’s analog input RCA jack to the analog output RCA jack. This step required that the TLC32040 AIC be initialized to both sample data and output a signal while communicating with the ‘C31 DSP via a high-speed serial interface[2]. When completed, the code was run and its operation was verified with a signal generator and an oscilloscope. It was noted that there was a small phase shift from the input to the output. This was an expected delay between the time when the AIC takes the sample and when the DSP sends it back to the AIC.

A second code segment was developed to implement a general-purpose digital output from the ‘C31 DSP to indicate if a signal contained speech or music. The ‘C31 DSK does not have any dedicated I/O pins that can be used for this purpose. An expansion bus is available, but significant hardware and software development is required to implement an expansion port. The DSP documentation indicated that the last used address is latched on the address bus until it is written again[1]. On the DSK board, the DSP’s address bus is only used for a short period of time at initialization, after that it remains idle. The result is 32 general-purpose output lines that can used to control external devices like speaker outputs.

The final segment consisted of implementing a finite impulse response digital filter using the ‘C31 DSP. This required the use of circular addressing support built in to the ‘C31 DSP to setup two circular buffers of variable length. One was for incoming sample data and the other for an array of coefficients that describe a FIR digital filter. By changing the coefficients and buffer lengths, a filter of any type and order can be implemented. However, filter order is limited to approximately 500 with the 2kB of SRAM onboard the ‘C31 DSP.

This final segment proved to be the most challenging and time consuming portion of the entire project. There are many special considerations when working with the circular addressing modes of the ‘C31 DSP. Once the code was finally completed, the result was a general FIR digital filter program that could implement virtually any type of filter. Changing the filter was a simple as running a quick script in MATLAB to generate new the coefficients and then copying them into the code[3].

Extensive testing was done on this portion of the project. Many types of filters were designed, implemented, and tested with lab equipment to verify their correct operation. As of this writing, the FIR code has implemented filters ranging from 1^{st} to 45^{th} order of all types; low pass, high pass, band pass, all pass, and comb. In all cases the operation was as expected and the transfer function matched MATLAB’s calculations.

With the three subsections of code completed and tested, it was then possible to glue them all together to make the final speech/music recognition algorithm (see attached code file “AD_KILL5.ASM”). Recall that this algorithm computes the ratio of low frequency energy to high frequency energy. As shown in Figure 1, the core of the algorithm consists of two finite impulse response (FIR) digital filters, each with the same audio signal input. One filter is a high

pass while the other is a low pass. For the purposes of this project, 14th order Hamming window filters both with a cutoff of 2kHz were found to be sufficient (based on analysis of audio samples). However, it is important to note that the nature of digital filters allows easy changes in filter order, type, and cutoff just by modifying the tap weights. In practice, the filters could be completely reconfigured in five minutes by changing values in the code, re-assembling, and loading the code into the DSP.

The output of each filter is squared to get the energy of the high and low spectrum before the samples are compressed (Figure 1). The samples are compressed by simply accumulating 31 samples in a buffer and averaging them. The result is placed into a secondary, 255 word circular buffer. The signal is then averaged further by taking the running average of this secondary buffer, every time a new 31 sample average is placed into it. The initial compression is necessary because the 'C31 DSP only has 2kWord of on board SRAM. With 1kWord allocated for code, and 512 words for samples, only about 125mS of samples can be stored at 19kHz. The compression extends this time to over 2 seconds! Finally, the averaging of the secondary buffer is necessary to smooth out the fluctuations in speech and music. Averaging over 1.5 seconds was found to be long enough to smooth both pauses in speech and rhythmic variations in music. Finally the ratio of the low spectrum to high spectrum energy is calculated and compared with a threshold. If the ratio is higher than the threshold the input is interpreted at speech, otherwise it is assumed to be music. At the time of the code development, the exact value for the threshold was not known. However, it was to be determined through training with known samples of music and speech.

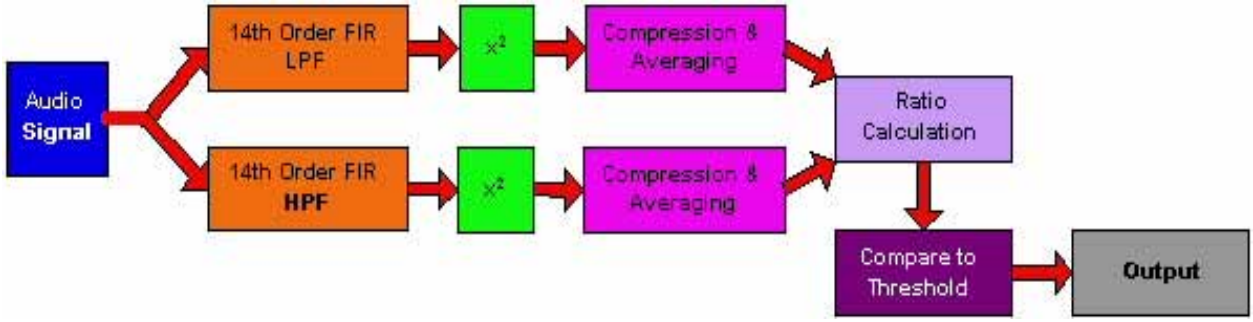

Figure 1: Music/Speech Recognition Algorithm Block Diagram

IV. Training

With a working algorithm the next step was to train it using samples of speech and music. The result of this task would be values for the cut-off frequency and threshold that would work for the known samples and hopefully others as well. Mathworks' MATLAB was selected to model the algorithm and analyze audio samples. MATLAB's DSP Toolbox includes several useful functions to accomplish these tasks. The **psd** function provides a graphical representation of an audio signal's power spectral density. This was useful in estimating where the cutoff frequencies of the filters should be set initially. The **fir1** function was used generate coefficients for finite impulse response (FIR) digital filters of all types and orders. Finally, the **freqz** function was used to plot the transfer function and phase response of a digital filter given only the coefficients. These plots were used to verify the response of the digital filters that were implemented with the 'C31.

To train the algorithm, a partial model of the algorithm was constructed in MATLAB's scripting language. The model was identical to the DSP algorithm with the exception that it did not attempt to determine if the sample was speech or music, instead it simply calculated the low frequency energy to high frequency energy ratio and logged it to a file. A complex script was written to run the model on various samples at various cut off frequencies and record the resulting ratio. By manually analyzing this data, two values could be extracted. The first was the cutoff frequency that provides the largest difference between the energy ratio of speech and music. The second value was the ratio threshold that best separates music from speech. For the training samples used in this project (approximately 10 minutes of mixed music and speech audio), the optimal cutoff frequency was found to be 2kHz and the threshold of low frequency energy to high frequency energy (above/below 2kHz) was found to be around 50-75. It should be noted that these values are dependent on the audio and speech data used to train the algorithm. Different types or amounts of music and speech samples will lead to different values.

V. Testing

The values obtained from the MATLAB simulations were placed into the DSP algorithm and it was tested with the known samples. The algorithm did not immediately perform as expected. Through experimentation, it was discovered that a threshold of 35 did produce desirable results. Then the algorithm correctly identified all of the sample music and speech clips. It was also discovered at this time that an averaging period of 1.5 seconds was not quite enough for some samples. Long pauses in speech or music caused the algorithm to behave unpredictably. When the averaging period was extended to 2 seconds this behavior ceased. Figure 2 and Figure 3 are graphs of the high and low frequency energy buffers in the 'C31 DSP while executing the detection algorithm. Figure 2 shows the energy during approximately 450ms of a music sample. Figure 3 shows the energy for a speech sample. Note how little high frequency energy is present in the speech sample. At the moment that this data was captured, the calculated ratios of low to high frequency energy were 237 for the speech sample and 7 for the music sample. Given the ratio threshold of 35, the algorithm correctly identified the two samples.

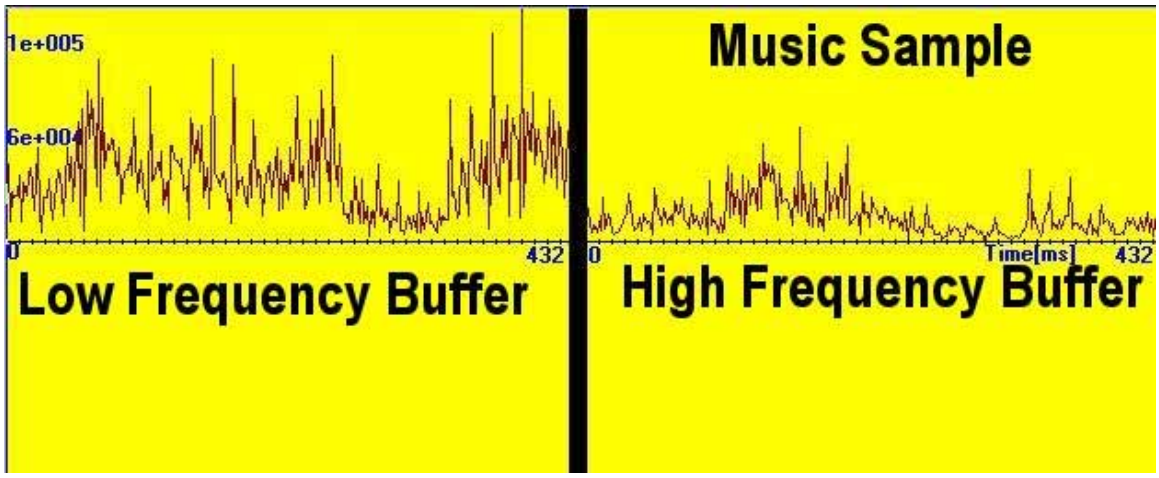

Figure 2: Energy buffers while algorithm is processing a music sample.

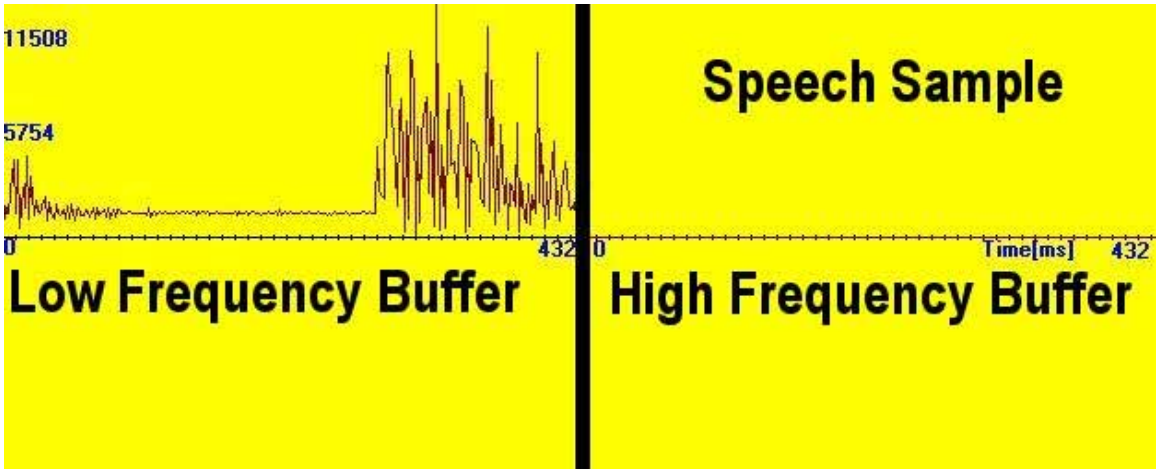

Figure 3: Energy buffers while algorithm is processing a speech sample.

The algorithm was also tested with speech and music samples that were not used to train the algorithm to see if it could correctly identify samples that were not in the set used to train it. The algorithm correctly identified nearly all the music samples, however speech samples with high-pitched voices or background music were often identified incorrectly as music.

VI. Discussion

In short, this detection algorithm was able to correctly identify music and speech samples used to train it once the correct threshold was determined. The MATLAB modeling proved to be most valuable in finding the cut-off frequency. However, the threshold that it calculated did not work as well as expected. There are several possible reasons for this, most of which concern the model itself. When constructing the model, the AC coupled input characteristics of the AIC were not taken into account. This meant that the MATLAB model was being affected by the low frequency components of the samples, while the DSP algorithm was not. This would explain why MATLAB predicted a slightly higher ratio of low spectrum to high spectrum energy. Another possible source for this discrepancy is the averaging period used in the MATLAB model. While the actual implementation was averaging over 1.5-2 seconds, the MATLAB model was constructed to average over the entire sample. This would cause the model to report a ratio that was correct for the entire sample, but incorrect for certain sections. The end result would be the running algorithm would identify some parts of a sample correctly, and some parts incorrectly. This behavior was noted in early testing.

VII. Conclusion

A project to monitor a radio station and automatically mute commercials and talking was presented based on the Texas Instruments TMS320C31 DSK. The algorithm is a simple recognition scheme based on a low pass filter, a high pass filter, and a thresholding scheme that can be understood with little DSP theory. The project is a useful and interesting application that can be used to help motivate students to learn DSP material.

VIII. Bibliography

1. Texas Instruments. TMS320C3x DSP Starter Kit User's Guide. Texas Instruments, 1996.
2. Chassaing, R. Digital Signal Processing-Laboratory Experiments Using C and the TMS320C31 DSK. J. Wiley, 1999.
3. B.P. Lathi. Signal Processing and Linear Systems. Berkeley-Cambridge Press, 1998.

CHRISTOPHER J. VONDRACHEK
Christopher J. Vondrachek is currently a hardware design engineer at Intel Corporation in Hillsboro, Oregon. He earned his B.S.E.E. from the University of Portland in 2000 and currently works as a printed circuit board designer within Intel's desktop products division. His technical interests include high speed PCB design, embedded and control systems, automobile engine management and DSP. Email: chris.vondrachek@intel.com

JOSEPH P. HOFFBECK
Joseph P. Hoffbeck is currently an Assistant Professor of Electrical Engineering at the University of Portland, Oregon. He is a member of the IEEE and the ASEE, and his technical interests include communication systems, digital signal processing, and remote sensing. Email: hoffbeck@up.edu