ECEn 487 Digital Signal Processing Laboratory

Lab 3
FFT-based Spectrum Analyzer

Due Dates

This is a three week lab. All TA check off must be completed by Friday, March 14, at 3 PM or the lab will be marked late.

Lab write-up submission, beginning of lab class Friday, March 14.

Objectives

The purpose of this lab is for each student to build a working audio spectrum analyzer and spectrogram using the built-in PC sound card and MATLAB. Principles of in-place computation FFT architecture, bit reversal addressing, and oscilloscope display synchronization will be studied. The spectrum analyzer will operate in continuous mode, real time, in the audio range of 20 Hz to 20 kHz.

Reading Assignment

1. Oppenheim and Schafer chapters 8 and 9.
2. MATLAB online help and documentation on function "fft."

Introduction

One of the most useful signal analysis tools is the spectrum analyzer. It allows us to directly look at the frequency content of a signal as it evolves over time. I believe that the frequency domain representation is the more natural view for human interpretation because of our familiarity with pitch, and tonal quality. If an oscilloscope gives a graphical representation of the time-domain version of a signal, then a spectrum analyzer can be considered a "frequency domain oscilloscope." It allows us to view the signal's frequency content. It generally takes much more sophisticated electronics or processing to build a spectrum analyzer than an oscilloscope. We will use the PC (with MATLAB) to transform our signal into the frequency domain using an FFT algorithm. You will use the efficient built-in MATLAB FFT function, and will also code your own FFT algorithm. You will compare performance of the two algorithms.

<u>Experiment 1</u> Write your own in-place radix 2 FFT routine and evaluate its performance

You will now write an in-place code which is the same structure used by high speed dedicated DSP processors. For the calling sequence definitions we will use as a template the optimized assembly language fft code, "cfftr2.asm" used for the Texas Instruments TMS320C6701 DSP processor. The FFT architecture will correspond to figure 9.15 in your textbook. (In our experience, this portion of the lab may take more than one week to code – it is the most conceptually difficult part of the lab and will take some serious time to think about).

Procedure

1. Carefully read the prolog of the file "cfftr2.asm" to get the calling sequence and data structures right. Note particularly that you must a) pre-compute and provide as an input parameter the sine-cosine table, w, (i.e. the W_N^k FFT kernel values), b) perform bit reversal addressing to de-scramble the output order, and c) put the sine-cosine table, w, in bit reversed order. You will need to pre-compute a bit reversed addressing look up table, so that bit reversed offsets do not need to be computed on the fly in real time.
2. Write a new MATLAB subroutine that functions identically, and has the same calling sequence as the FFT function in cfftr2.asm. This should be an in-place radix two code. Be careful about the difference in array index bases in MATLAB and the mathematical notation (zero based in math and C, one based in MATLAB).
3. Compare performance for your fft and the built-in fft using 1024 point FFTs, a fixed number of loops, and "tic" and "toc" (as in Experiment 1, step 4). Explain to yourself why one would be slower than the other.
4. Now, rebuild your FFT using a "for" loop implementation of the direct DFT sum (this can be written in about four lines of code). Using $N = 1024$, "tic", and "toc" measure performance for this non-fft implementation and compare with your in-place FFT. Explain in why one would be slower than the other.
5. <u>Demonstrate your in-place FFT program to the TA for sign off.</u>

Hint: This MATLAB subroutine does not have to be long. If you get the butterfly branch indexing right, it consists of just 3 nested "for" loops.

<u>Experiment 2</u> Design and implement a spectrum analyzer using the built-in MATLAB FFT function.

Procedure

1. Build a working real-time spectrum analyzer code based on your digital oscilloscope program from lab 1 and the MATLAB fft function. This fft implementation is a decimation-in-time,

in place, very efficient Cooley-Tukey factored N implementation. It uses radix 4 decomposition if possible. Bit reversal data reordering is handled internally.

2. Use only the left input channel data.
3. Using the MATLAB plot function, display the FFT *magnitude squared* (not the full complex number or real or imaginary part) on the computer monitor. In MATLAB don't use abs(X).^2 to compute magnitude squared, use real(X).^2 + imag(X).^2. This is faster in a real DSP processor implementation since a square root is not computed, and closer to how this would really be implemented in a DSP processor. Update the display at lease once a second.

   Display only the positive frequency terms. Label the horizontal axis in Hz, calculated based on your sample frequency. The user should be able to specify the sample rate, and FFT length. Also, the vertical axis should be in either linear scale, or dB, under user control. The plot should be updated continuously, with as fast a refresh rate as possible.
4. Modify your code so the display vertical axis is in dBm scale (This is decibels relative to 1 milliwatt. You will need to experimentally calibrate your ADC input so you know what the digital count is corresponding to a 1V input signal level. Let *A* be the integer number you get with a 1V dc input signal. Assume the impedance for your analog input to the ADC is 600 Ohms. Then a 1V d.c. input signal would be $(1V)^2/600$ Watts, or 1.667 mW. Thus A^2 corresponds to 1.667 mW = 2.22 dBm. With this information you can figure out how to normalize your display to be in dBm. Whatever your *A* is, that integer count must correspond to 2.22 dBm. Let *X*[0] be the complex output of your *N* point FFT in the dc frequency bin. $10\log_{10}\left\{|X[0]|^2 / \text{ref}\right\}$ = 2.22 dBm, so solve for ref. For a 1Vdc input, $|X[0]|^2 = N^2A^2$. So $10\log_{10}\left\{N^2A^2 / \text{ref}\right\}$ = 2.22 dB. Thus ref = $N^2A^2 / (10^{0.222})$. Now, some sound cards will not pass a dc voltage, and the function generator will not output a dc signal, so you will need to use a sinusoidal signal which has an RMS level of 1V.

   Comment on why the dBm display mode might be a good idea for a power spectrum analyzer. Note that you may need to add a small constant to the FFT magnitude before taking the logarithm to avoid very large negative values when the ADC output is zero.
5. Evaluate and compare the computational speed performance for FFT lengths of 1024 points and 1021 points. This can be done by modifying our code to run for a fixed number of plot updates (say 100) and timing the full run using the "tic" and "toc" functions. Explain any performance differences you observe between the two FFT lengths.
6. Demonstrate your running program to the TA using two analog input signals of your choosing which have different frequency content.

<u>Experiment 3</u> Perform signal analysis with your power spectrum analyzer, and improve the design.

Procedure

1. Verify proper operation by using a sine wave input from the function generator and observing proper positioning of the spectrum peak as you seep the frequency. Verify that the highest frequency you can display, at the far right of your oscilloscope display, corresponds to half of the sample frequency. Record your observations.
2. Using a square wave signal with fundamental frequency of about 200 Hz, use your spectrum analyzer to verify that this signal has the correct continuous-time Fourier series coefficients as described in your ECEN 380 textbook. Record your observations and have the TA verify them.
3. Analyze the spectrum of some CD music passages. What conclusions do you draw regarding the time variation of the spectrum, or the relationship between specific instruments, voice, and their corresponding spectra?
4. Analyze the spectrum of the Morse Code recording. Record your estimates of the center frequencies for each signal present.
5. Add an exponential decaying time average feature to your spectrum analyzer. If $|X[k,r]|$ is the k^{th} frequency bin magnitude for the r^{th} data window, then the averaged magnitude, $A[k,r]$, is computed as: $A[k,r] = (1-\alpha)|X[k,r]| + \alpha A[k,r-1]$, where $0 \le \alpha < 1$. $\alpha=0$ implies no averaging, while larger values of α imply a longer averaging window over time.
6. Using exponentially averaging, analyze the spectra of the Morse code signal and some CD music, and your speech, for several of values of α (e.g. 0, 0.7, 0.9, 0.99), and comment on what advantages or disadvantages this method has. For the speech signal use the microphones and pre-amplifiers provided in the lab.
7. <u>Demonstrate the exponential averaging spectrum analyzer to the TA.</u>

<u>Experiment 4</u> Develop a Working Spectrogram Tool in MATLAB

Procedure

1. Write a simple MATLAB code using the "spectrum" and "imagesc" functions to display frequency content with respect to time for a sampled audio signal. You must select window length, window overlap percentage, and window shape for good performance. Due to the delay time between sampled window blocks using the built-in PC ADC, it will not be possible to make this work continuously, but you should collect several seconds of data from the ADC and display the computed spectrogram in a single image plot.
2. Use your spectrogram tool to analyze speech. You will need to use a preamplifier and microphone to drive the PC's ADC input. Study the difference between your own spoken

vowel sounds and fricatives ("s", "sh", "f", etc.), and plosives ("p", "t", "d", etc.). Make sure you have designed the spectrogram so you can see the fundamental and harmonic structure in the vowel sounds. Comment in your lab book on what you have leaned and observed regarding the structure of human speech.

3. Use your spectrogram tool to decode a message .wav file provided to you by the TA. You will have to change your code to accept a .wav file as an input rather than using the ADC input. Demonstrate your working spectrogram to the TA, and let him know what message you decoded in order to pass off this experiment.

Conclusions

Discuss any additional implications of what you observed. What other applications can you see for using a spectrum analyzer? What improvements could be made to the basic design to make it more usable? Describe what you feel are the important principals demonstrated in this lab, and note anything that you learned unexpectedly. What debug and redesign procedures did you need to perform to get it to work?

Laboratory 3 TA Check-off Page (to be submitted with laboratory report)

Student Name: _____

_____ Task 1.5 – Demonstrate your working fft to the TA.

_____ Task 2.6 – Demonstrate your working spectrum analyzer to the TA.

_____ Task 3.7 – Demonstrate your spectrum analyzer with exponential decay to the TA.

_____ Task 4.3 – Demonstrate your working spectrogram to the TA.