CROSSING THE BRIDGE: TAKING AUDIO DSP FROM THE TEXTBOOK TO THE DSP DESIGN ENGINEER'S BENCH

Robert C. Maher

Department of Electrical and Computer Engineering, Montana State University, Bozeman MT 59717
rob.maher@montana.edu

ABSTRACT

A set of seven audio digital signal processing (DSP) lab assignments have been developed for intermediate and advanced students in electrical engineering. The laboratory modules are designed to bridge the gap between a solid understanding of the mathematical principles of DSP and the practical issues that arise when implementing algorithms for real time audio processing. The audio DSP lab assignments assume a prerequisite of a one semester course in discrete-time signals and systems. The exercises include basic delay lines, recirculating delay lines for audio effects (echo, flanging, phasing, artificial reverberation), variable and parametric IIR equalizer filters, real time signal synthesis techniques (pseudo-random noise, wave table lookup methods, and direct computation), FFT and short-time Fourier transform (STFT) methods, and nonlinear and adaptive audio processes.

Index Terms— DSP education, DSP laboratory, audio DSP

1. INTRODUCTION

Most undergraduate signal processing courses include laboratory experience and hands-on demonstrations. Contemporary personal computers are now easily able to process signals for near-real time display and playback. Exercises using Matlab® are particularly popular and appropriate for understanding the sampling theorem, frequency response, digital filter principles, and the discrete Fourier transform. It is clear that the quality of undergraduate DSP education has been enhanced considerably by the use of computer-based laboratory exercises.

Many students enter electrical engineering because of an interest in—or at least a keen awareness of—the excitement and glamour of the audio business. Therefore it is quite suitable to use audio engineering examples to demonstrate DSP concepts and to motivate student learning. Moreover, several successful DSP textbooks, such as by *A DSP Primer: With Applications to Audio and Computer Music*, by Steiglitz (1996) [1], *Introduction to Signal Processing*, by Orfanidis (1995) [2], and *DSP First: A Multimedia Approach*, by McClellan, Schafer, and Yoder (1998) [3], have used audio examples to help students learn DSP principles.

Once the basic DSP concepts are learned in an introductory course, it is important for students who intend to develop DSP products in their engineering careers to see how the symbolic mathematics of the classroom are ultimately implemented in a real time embedded DSP system. For example, students who understand the finite-length vectors and file I/O concepts from Matlab may still need to learn how data is processed in the sustained, piecemeal world of real time input/output routines and rigid scheduling constraints: rather than having the entire data record available from the start, the real-time system must "throttle" itself based on the availability of data in its input and output buffers. Similarly, students must concern themselves with the timing constraints and priorities of the low-level processing steps so that all the calculations can be guaranteed to fit in the available inter-sample or inter-frame time period while still meeting the overall system latency requirements.

2. PRACTICAL REAL TIME AUDIO DSP

Practical digital signal processing considerations routinely involve numerical performance (e.g., how many bits are available to represent data, coefficients, and intermediate results), proper scaling to minimize the likelihood of clipping while maintaining adequate signal to quantization noise ratio, filter topology choices and tradeoffs, and data resampling and timing issues in multirate audio systems. Furthermore, students at the intermediate level have generally studied Direct Form I and II filters and the transpose forms, but they often have not had a chance to consider the computation, storage, and noise performance of the different structures in a practical digital audio system.

The need for a practical focus appropriate for intermediate and advanced DSP education led to the development of a course at the University of Colorado-Boulder, *ECEN4002: Digital Signal Processing Lab*. The course was first conceived by Prof. Tom Mullis of CU and

Dr. Jeff Barish, then of EuPhonics, Inc., in the mid-1990s. At that time innovation in the audio industry was increasingly based on improvements in DSP microprocessor performance, so it was natural to adopt a real time focus for the course.

In 2001, the author was asked to update and revise the course to reflect the changing environment for real time audio: faster processors, more memory, better software development tools, and the emergence of reconfigurable (RAM-based) processing systems. The revised lab experiments have now been used for several years at the Univ. of Colorado and by intermediate DSP students at Montana State University-Bozeman [4]. Versions of the audio DSP lab exercises have been prepared for the Freescale 563xx, Analog Devices 219x, and other similar fixed-point processors.

3. AUDIO DSP LAB EXPERIMENTS

The sequence of experiments begins with a "pass-through" program using interrupt service routines to pass unaltered samples from an analog-to-digital converter to a digital-to-analog converter. Using the pass-through program as a building block, the subsequent experiments add digital delays, recirculating delay lines, FIR and IIR filters, audio signal synthesis, and various FFT-based processing algorithms. Along the way the students learn the essential architectural features of real time signal processing systems.

The seven lab exercises are:

- **Pass-Through Program** (understanding interrupts, gain, quantization, sample rate and aliasing)
- **Delay Lines** (delay lines and modulo addressing; implementing FIR filters)
- **Recirculating Delay Lines and Artificial Reverb** (echo, flanging, phasing, reverb simulation)
- **IIR Filters and Variable Equalizers** (direct form filters, gain and scaling, parametric/variable filters)
- **Signal Synthesis** (pseudo random sequences; wavetable lookup synthesis; simple music players)
- **FFT and STFT** (FFT algorithms; block based processing; implementation and testing)
- **Nonlinear and Adaptive Processing** (dynamic range compression; LMS adaptive interference cancellation)

Each lab experiment is intended to require two weeks to accomplish (two 2-hour lab sessions). The first three experiments tend to build on one another, but it is possible to treat the other experiments *à la carte* to suit the students' needs and interests.

3.1. Experiment 1: The pass program

The introductory lab experiment is often the first time many of the students have actually had to download and run their own code on a real time system. Thus, the exercises are presented in a step-by-step, cookbook fashion so that the students get comfortable with the repetitive "edit, assemble, download, test" cycle.

The students start with a pre-existing code fragment that initializes the processor and the A/D and D/A system using interrupt service routines (ISRs). The A/D interrupt service routine is triggered at the system sample rate, and simply copies the left and right audio samples from the A/D system registers into the processor's memory. Similarly, the D/A ISR is triggered synchronously and simply copies the left and right audio samples from memory to the D/A system registers. With no data modification, the output signal is a reconstruction of the bandlimited and sampled input signal. The students are asked to perform a frequency response test using an analog function generator and an oscilloscope. They also examine the overall delay properties of the system and account for any signal buffers, A/D sample queues, etc., that shift the reconstructed output signal with respect to the input signal.

Once the pass-through system is up and running, the students write three simple alterations to the pass program. First, they must multiply each sample from the left channel input stream by a gain reduction value, such as 0.5. All that is needed are a few lines of code in the left channel interrupt service routine to do the multiplication while leaving the right channel pass-through unchanged. Even though this is a seemingly trivial exercise, it is intended to help the students gain confidence with the software tools, the processor's arithmetic architecture, and the minimum-delay DSP concept: as long as the processor completes its assigned tasks during the time interval between the arrival of an input sample in the A/D ISR and the corresponding output D/A ISR, the result is properly computed.

The final exercises in Experiment 1 are a first look (and listen) involving amplitude quantization and aliasing. The quantization exercise has the software mask off (zero out) some number of least significant bits between the A/D and the D/A to simulate fewer bits in the quantizer, while the aliasing demonstration involves decimating the sample sequence from the A/D by setting N-1 out of N samples equal to zero prior to the D/A. In keeping with the practical emphasis of the course, these exercises are designed to produce audible effects that the students can hear and remember when debugging real time programs later in the course.

3.2. Experiment 2: Delay lines and FIR filters

Experiment 2 starts with the simple pass-through program used in Experiment 1. In order to run in real time, the

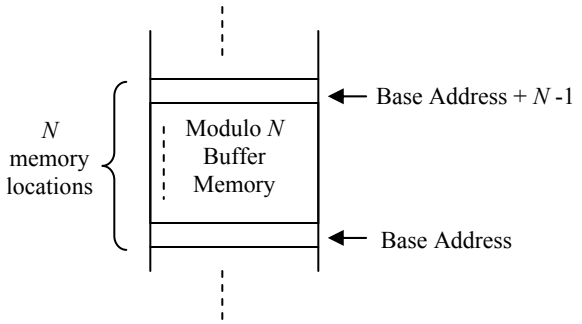

Figure 1: *Modulo addressing concept*

software must complete its processing of each sample at an average period that does not exceed the sample period ($1/f_s$). The students start by estimating the number of processor cycles that can be completed in one sample period, then alter the pass-through code to include that number of dummy instructions, such as NOPs (no operation). Incrementally increasing the processing time by adding more NOP instructions results in audible defects as the system is unable to keep up with the sample rate.

The experiment then introduces the debugging support features provided by the DSP microprocessor's development software. For example, many systems provide non-real time input/output simulation using computer files instead of the A/D and D/A hardware. This can be handy for development and verification purposes, so it is important for the students to be aware of the typical debugging concepts and features.

Next, the students learn how to use the modulo addressing hardware (Fig. 1) typically found in DSP microprocessors, and implement first-in first-out (FIFO) delay line queues of various lengths. In systems with sufficient RAM, the input/output delay through the FIFO can be made audible. Otherwise, the delay can be observed

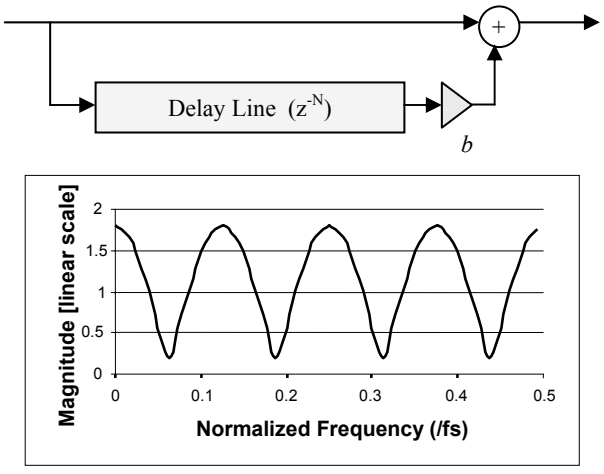

Figure 2: *Feed-forward delay structure and frequency response (N=8, b=0.8)*

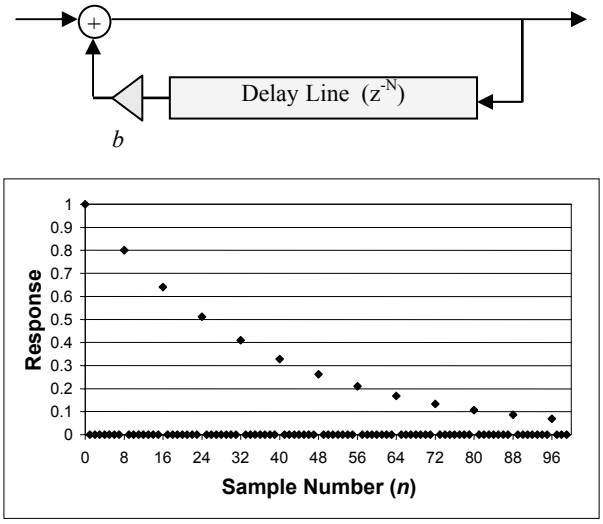

Figure 3: *Feedback (recirculating delay) structure and unit sample response (N=8, b=0.8)*

using a two-channel oscilloscope to compare the system input and output simultaneously.

Experiment 2 concludes with a set of exercises involving finite impulse response (FIR) filter structures. The key practical insight is for the students to understand the multiply-accumulate (MAC) framework embodied in the hardware. In particular, the students learn about the simultaneous coefficient and data fetch operations provided by the classic Harvard architecture of DSP microprocessors.

3.3. Experiment 3: Recirculating delay lines and artificial reverberation

At this point, the students have become acquainted with the fundamental architectural features of the DSP microprocessor and the program development process. Experiment 3 combines the basic concepts into several useful audio DSP effects.

First, the students create a real time variable delay structure known as a *flanger* [2]. The effect involves summing a direct feed of the input signal with a delayed and scaled version in a feed-forward arrangement (Fig. 2). The amount of delay is gradually varied from short to long and back to short in a cyclical manner. The effect is an audible comb filter with spectral notches (system zeros) sweeping up and down through the spectrum. The process gives an interesting spectral coloration that some describe as alternately "hollow" and "nasal."

A recirculating (feedback) arrangement with an embedded delay line is used in the next exercise (Fig. 3). For short delays the result is a comb filter with peaks corresponding to the system poles. For long delays the effect is a set of discrete echoes. An all pass filter structure is also used to provide greater echo density without introducing excessive spectral coloration (Fig. 4). The

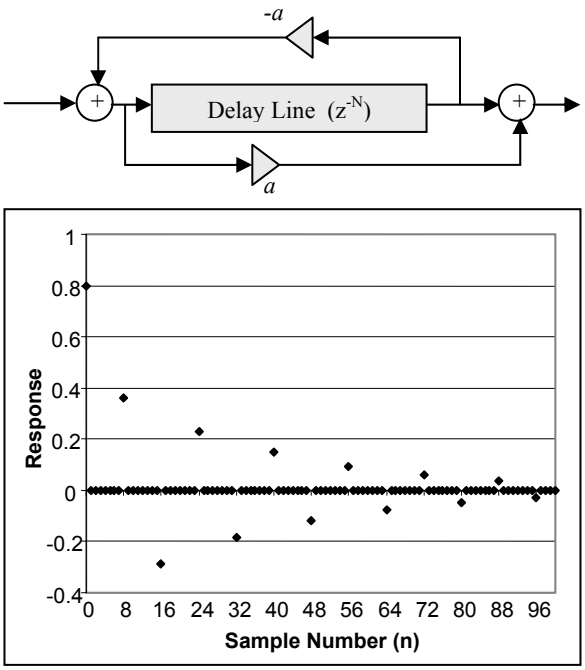

Figure 4: *All-pass recirculating filter for reverberation simulation (N=8, |a|=0.8)*

students combine sets of these recirculating delay blocks in parallel and in cascade to form a basic artificial reverberation system [2, 5]. As before, the students must use their ears to judge the sound quality.

3.4. Experiment 4: IIR Filters and variable equalizers

Continuing the practical tour of audio DSP, Experiment 4 introduces how infinite impulse response (IIR) digital filters are implemented using a DSP microprocessor. The importance of numerical overflow, coefficient quantization, and proper scaling are presented, using Matlab to calculate direct form filter coefficients and to predict the resulting frequency response.

Experiment 4 emphasizes second order IIR filter building blocks. This leads to several practical implementation considerations, particularly for fixed-point DSP microprocessors.

Parametric equalizer filters are considered in the second part of Experiment 4. Unlike conventional direct form filter topologies, parametric filters are designed so that individual coefficients independently control the filter's center frequency, bandwidth, and boost/cut (Fig. 5). Digital audio systems often use parametric filters so that the user can "turn a knob" to adjust the filter characteristics without having to recompute all the coefficients and the entire filter state. The students implement a second-order parametric filter with an embedded all pass digital filter element [5].

3.5. Experiment 5: Audio signal synthesis

Students logically think of audio DSP systems as needing an audio input and audio output, but many interesting DSP systems are only signal generators. Experiment 5 presents two audio synthesis methods: algorithmic synthesis and wavetable synthesis.

A pseudo-random sequence generator is used to demonstrate algorithmic synthesis. A computationally cheap method for generating a pseudo-random sequence is the *linear congruential* method. The computation is expressed as

$$y[n]=(a\cdot y[n\text{-}1] + c)_{\text{mod } M},$$

where a is a constant multiplier, c is a constant offset, and M is the modulus. The initial value to start the sequence, $y[0]$, is called the *seed* of the random generator. The choice of a, c, and M must be made with care to ensure that a maximal length output sequence is obtained for any choice of the seed, $y[0]$. It is common to choose M to be equal to the word length of the processor (e.g., 24 bits for the Freescale 563xx) so that the modulo operation is achieved simply by taking the appropriate portion of the accumulator.

The students also observe that a linear congruential generator is a deterministic algorithm, i.e., the sequence of output values is exactly determined from the seed and the constant parameters, so the "randomness" of individual bits (particularly the least significant bits) can be poor.

The basic building block for periodic real time audio signal synthesis is the *wavetable* synthesizer. The shape of a desired waveform is stored in memory (a *wavetable*) so that the samples in the table can be extracted sequentially and sent to the D/A to produce the desired sound. The wavetable can be filled with samples recorded from a natural sound (a

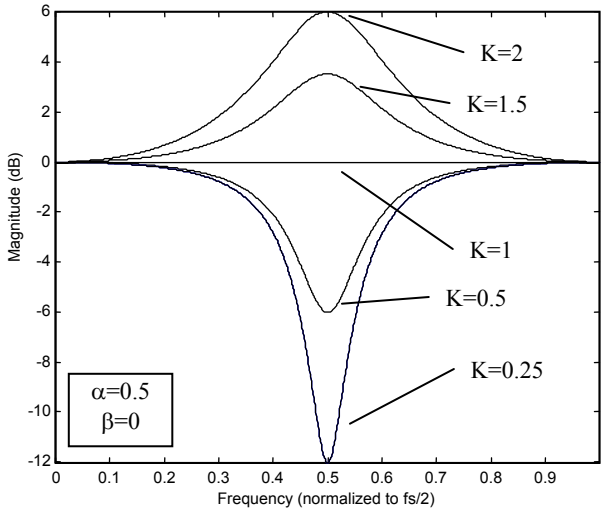

Figure 5: *Parametric filter response as a function of the single control coefficient, K.*

process known as "sampling" to electronic musicians) or with arbitrary pre-computed samples.

The students quickly discover the basic problem: the waveform in the wavetable corresponds to exactly one period, but an interesting signal generator must be able to vary its frequency and amplitude. To generate other repetition frequencies from the same wavetable data, a *fractional* phase look-up accumulator and a sample increment, both with an integer part and a fractional part, are used to calculate the address of the next sample to fetch from the wavetable. A nonzero fractional part of the look-up accumulator indicates a sample is needed in between two of the samples actually stored in the wavetable, and interpolation can be used to approximate the desired value.

Once the student has a working wavetable synthesizer with frequency control, the next step is to make a rudimentary music synthesizer with frequencies corresponding to musical pitches and a time-varying amplitude envelope for rhythm.

3.6. Experiment 6: The FFT and STFT

Many interesting and useful audio DSP techniques involve a spectral decomposition such as the discrete Fourier transform (DFT) or the discrete cosine transform (DCT). Thus, Experiment 6 introduces the use of a fast Fourier transform (FFT) algorithm to compute the DFT.

The exercises use existing FFT code provided by the processor manufacturer, but the students must prepare the input data, twiddle factors, proper scaling (assuming a fixed-point processor), and verify the computed results for both the forward and the inverse FFT.

Among the other real time issues for the FFT is the inherent throughput delay, or *latency*, that is necessary: the FFT computation cannot begin until a full buffer of input data is available. Thus, even if the FFT computation itself was instantaneous, the overall processing delay will be at least $N \cdot f_s$, where N is the number of audio samples processed by each FFT, and f_s is the sample rate.

The students create an FFT-based pass-through program that segments the input data stream into overlapping blocks, applies a data window and performs the FFT, then followed immediately by the IFFT and an overlap-add reconstruction (Fig. 6). Next, a bandpass filter and a simple spectral noise gate (suppressing spectral components below a predetermined power threshold) are implemented in software to modify the FFT output prior to the IFFT and reconstruction steps. The students are able to verify the overall delay and processing characteristics during the experiment.

3.7. Experiment 7: Nonlinear and adaptive processing

Although most introductory DSP labs concentrate on linear processing, there are many interesting and useful audio processing techniques that require data-dependent, nonlinear functions. The spectral noise gate in Experiment 6 is one example, and the students work on two additional exercises in Experiment 7.

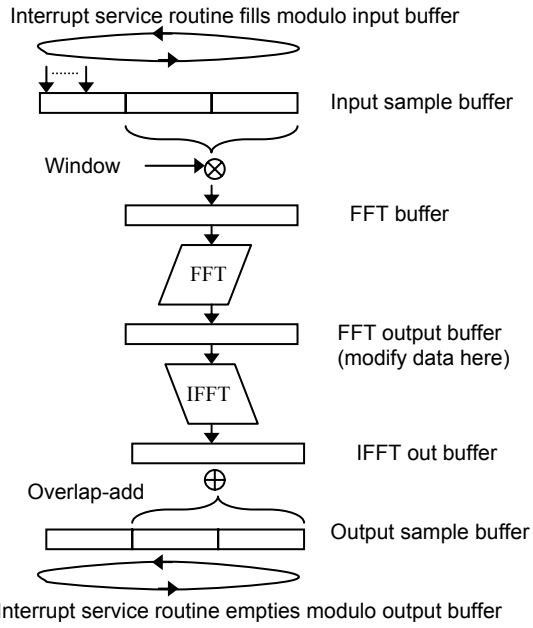

Figure 6: *Short-time Fourier transform real time processing framework*

First, the students write code for an automatic gain control and dynamic range compression scheme. A leaky peak detector is used as the envelope follower, and the output gain is altered accordingly, including variable attack and release time constants to reduce the audible artifacts of the gain changes [2].

The students also implement a two-channel FIR adaptive interference canceling system using the LMS algorithm [6]. One input is provided with a signal+noise sample (the desired signal, $d[n]$) while the other input receives a delayed and filtered version of the noise alone (the filter input, $x[n]$). The filter coefficients, b_i, are adapted using a small convergence parameter, K, to minimize the output discrepancy, $e[n]$, which is also the enhanced output signal.

The LMS algorithm steps can be summarized as follows.

1. Obtain next samples of $d[n]$ and $x[n]$
2. Run FIR filter on $x[n]$ to obtain $y[n]$
3. Subtract $y[n]$ from $d[n]$ to get $e[n]$
4. Apply the update expression
   $b_i[n+1] = b_i[n] + K\, e[n]\, x[n-i]$
   to get the filter coefficients for the next time step
5. Repeat

The students then experiment with the convergence parameter, signal levels, and other implementation details.

4. DISCUSSION AND CONCLUSIONS

Students generally find the real time audio DSP lab to be fun and interesting, but also quite challenging. Probably the most difficult issue for the students to overcome is the need to learn the low-level architecture and assembly language of the target processor. Even students who have studied microprocessor programming in earlier courses generally have not had to deal with multiply-accumulate pipelines, modulo addressing modes, numerical scaling, and real time processing constraints.

Intermediate-level students can become frustrated if they do not have sufficient lab assistance, particularly for the first few experiments. Thus, it is very desirable to have several experienced DSP assembly language programmers available in the lab to provide rapid feedback as the students develop their own programming and debugging skills. Upper division student volunteers can be of great assistance during open lab sessions for the first few weeks of the course.

Most DSP microprocessor manufacturers have a good selection of application notes and example programs available on their web sites. Students should be encouraged to seek out this sort of information on their own since this strategy will serve them well once they graduate. Nonetheless, it can be helpful to have a collection of key reference books, tutorials, and summary sheets available in the lab for student use.

5. REFERENCES

[1] Steiglitz, K., *A Digital Signal Processing Primer with Applications to Digital Audio and Computer Music*, Addison Wesley, 1996.

[2] Orfanidis, S.J., *Introduction to Signal Processing*, Prentice-Hall, 1996.

[3] McClellan, J.H., Schafer, R.W., and Yoder, M.A., *DSP First: A Multimedia Approach*, Prentice-Hall, 1998.

[4] Maher, R.C., *Real Time Digital Audio Signal Processing Lab*, http://www.coe.montana.edu/ee/rmaher/audio_dsp .

[5] Zölzer, U., *Digital Audio Signal Processing*, Wiley, 1997.

[6] Treichler, J.R., Johnson, C.R., Jr., Larimore, M.G., *Theory and Design of Adaptive Filters*, Prentice-Hall, 2001.