

Laboratory Exercise #3

Recirculating Delay Lines and Artificial Reverberation

Introduction

In the last lab a delay line was implemented using the modulo addressing capability of the 56300. But digital filters are not the only application of delay lines. There is a variety of useful and interesting applications involving delayed signals. In this lab we will investigate several digital audio effects that involve variable delay lines, recirculating delays, and comb and all-pass reverberators.

When a signal is added to a delayed version of itself, the result is a boost of spectral components that have a period ($1/f$) equal to an integer multiple of the delay length (delayed signal added *in phase*), and attenuation of components for which the delay is an odd multiple of half the period (180° out of phase). These delay properties are very useful in audio signal processing because many acoustical systems have delays and resonant properties that are modeled well by simple delay lines. For example, musical instruments generally have an air column, string, or bar that has a “tuned” vibration corresponding to the size of the vibrating element and the speed of sound in that medium. Similarly, sound propagation in rooms and concert halls involves the direct arrival of sound waves from the source to the listeners’ ears, followed by the delayed arrival of the sound that reflects off the floor, walls, and other surfaces in the room. There are many technical and creative reasons that digital processing to mimic acoustical and electro-acoustical systems is both practical and useful.

In this experiment you will develop some basic audio effects that employ fixed and variable delay lines. The exercise descriptions are somewhat less “cookbook” than in the previous experiments because it is expected that you have become acquainted with the essential tools in Labs #1 and #2. Keep in mind that it may be helpful to debug portions of your code using the 56300 Simulator and the non-real time file I/O method of the Debugger. And, as always, develop your code incrementally so that the debugging task is simplified.

Time-Varying Delay: The “Flanger”

A simple audio effect can be created using a *feed-forward* delay line structure. The input signal is sent directly to the output and also into a delay line. The output is the sum of the input signal and the delayed signal. This structure is shown below.

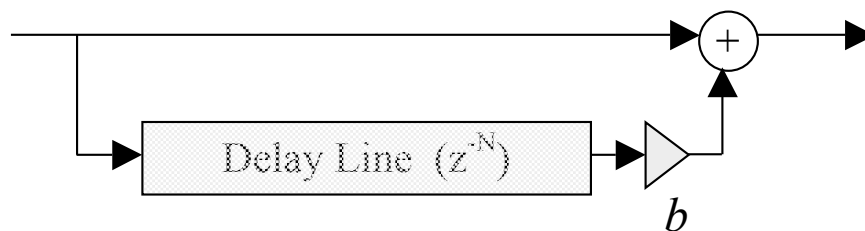


Figure 1: Feed-Forward delay structure

Note that the feed-forward structure has the transfer function

$$H(z) = 1 + bz^{-N}$$

which implements N zeros located equally spaced on a circle with radius $b^{1/N}$. The corresponding frequency response is a sequence of notches. This type of response is sometimes called a *comb* filter, since the sequence of notches looks a bit like a comb.

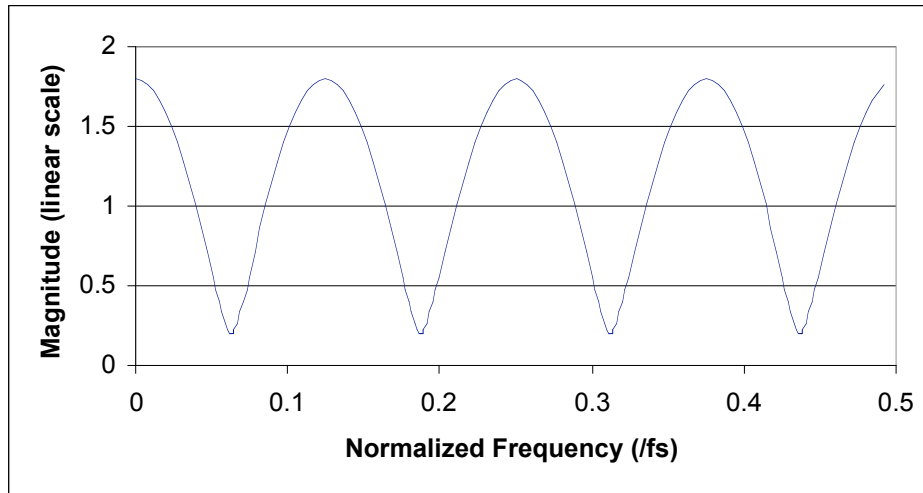


Figure 2: Frequency response of structure in Figure 1 (with $b=0.8$ and $N=8$).

If you send music through this structure and listen to the output of this system compared to the unaltered input, you may hear a spectral shift or coloration that is peculiar to the comb filter response. As the frequency content of the music varies, the components that are near the notches of the comb get attenuated and the signal sounds somewhat “hollow” or “nasal” in character.

Now imagine what happens if we somehow make the delay length vary with time. Since the number and location of the spectral notches is determined by the delay, if we gradually change the delay length the notches will sweep up or down in frequency corresponding to the instantaneous delay time. If we arrange the delay to increase and decrease steadily and periodically, the result is a repetitively sweeping effect that can be a useful for interesting studio and live performance modifications. The varying feed-forward delay structure is commonly called a *flanger*, since the original method was created by a recording engineer using two duplicate tape recordings. By starting the duplicate tapes on two different tape decks, then pressing on the flange of one tape spool in order to slow it down slightly, the summed output of the two decks contained one signal and a slightly delayed version of the same signal. Varying the relative playback speeds produced the distinctive changing comb filter “whoosh” effect.

⇒ Exercise A: Flanger effect

To simulate the tape deck flange effect using digital techniques you will implement the feed-forward structure of Figure 1. Start again with the simple delay line code you wrote for Lab #2, but modify it as follows:

- (1) Begin by making a monophonic signal: add the **a** and **b** accumulators together and divide the sum by two. *Note* that a divide-by-2 can be accomplished with a one-bit arithmetic right shift. See the 56300 instruction set for how to do a right shift.
- (2) Allow a maximum delay of 15 milliseconds (720 samples with a 48kHz sample rate).
- (3) Choose a base address for your delay buffer that is valid for the 720 sample buffer length.
- (4) Place the output in both the **a** and **b** accumulators (mono output).
- (5) Implement the delay by having an address register point at the “head” of the delay line, and a separate pointer (or indexed offset) that locates the “tail” of the delay line. The number of samples separating the head and tail is the instantaneous delay, and this will vary in real time while the flanger is running.

First test your code using several different fixed delay amounts. Design your code so that you can use the Debugger to set and change the delay length.

Next, modify your code again so that the delay amount varies automatically from near zero to 720 and back to near zero repeatedly with a period of a few seconds. Try to design your code so that the repetition rate is adjustable using the Debugger. It will probably make sense to have a variable in fractional form that is incremented at each sample time, then multiply the integer 720 by this fractional number. Then truncate the product to use just the integer part as the required delay. This may take some thought to figure out where the integer radix point will be after the multiply.

Listen to the output using various music, speech, and test signals. Include a description in your report, along with any measurements you feel are appropriate to verify the behavior of your design.

One problem with the simple flanger is that the delay is restricted to be an integer number of samples, so switching from one delay amount to the next may result in an audible “click” or discontinuity in the output. A high-quality flanging algorithm requires an *interpolated* delay line so that the instantaneous delay can be a fractional number of samples. This means that the output of the delay line is not simply one of the samples out of the modulo buffer, but instead is a value interpolated from the delay line. For example, if the instantaneous delay needs to be 20.75 samples and we choose to use linear interpolation, we fetch the 20th and the 21st delayed samples and compute a new sample that is 0.75 of the amplitude difference between sample 20 and 21. Feel free to try this if you have time. We will do more with interpolation in future lab exercises.

Recirculating Delay Lines and Echo Generation

Consider left and right channel audio signals. If one of the two stereo channels is delayed and the delay line is made long enough (more than a quarter second or so) the sound coming out of the delay line is audibly distinct from the un-delayed signal: in other words, it sounds like an echo. If the delay is short, however, the effect of the delayed channel may be perceived as a spectral change or shift of the stereo image.

Now consider the effect of a *recirculating* delay line. In this case the output of the delay line is fed back to the input, so the signal circulates and recirculates through the delay line over and over again.

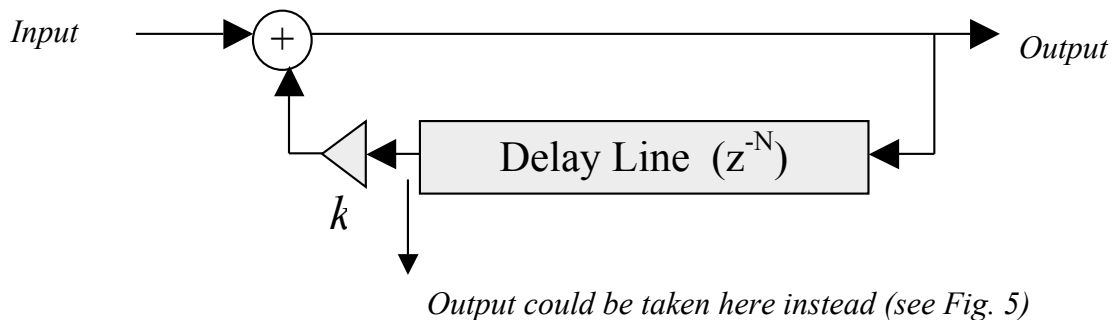


Figure 3: Recirculating (feedback) delay line.

This structure is recursive (IIR—infinite impulse response), and implements a function with N equally spaced poles located in a circle around the origin of the z -plane. Note that if the value of k is made greater than or equal to unity, the recirculating signal will be amplified each time around the loop, resulting in an unstable system (poles outside unit circle). The output of the filter for $N=8$ and $k=0.8$ is shown in Figure 4.

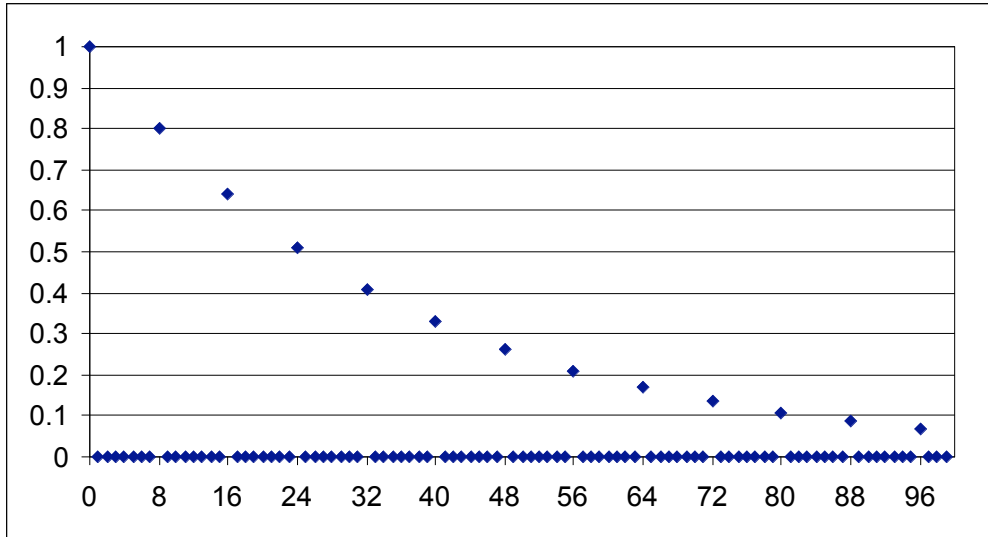


Figure 4: Unit sample response for $N=8, k=0.8$.

⇒ Exercise B: Make a recirculating delay line

Part 1: Find the z-transform expression for the system in Figure 3. Then using MATLAB, determine the complex pole locations, unit sample response, and the frequency response magnitude for a recirculating delay system with $N=7$ and $k=0.7$. Repeat with several different values of N and k . Include your results in your report.

Part 2: Starting with the “maximum” delay line code you wrote for Lab #2 Exercise C, make a modification that performs the recirculation as shown in Figure 3. Alter the feedback gain using the Debugger and listen to the result. Start by using a short input signal (“blip”) so that you can hear each echo without overlap. Confirm the echo behavior, and consider what happens if the recirculating signal level is high enough to cause overflow (saturation) in the output signal. Can you place a scaling gain parameter into the structure to keep the echo decay time unchanged while limiting the likelihood of overflow?

⇒ Exercise C: Long delay line using external SRAM

Now modify your program to use an even longer delay line. The ‘307 EVM boards have a 64k word external static RAM (SRAM) that can be used by the DSP chip. Enabling the SRAM requires a small modification to the initialization code in your `pass2.asm` program. Find the line near the START label that programs the chip’s BCR register (`x:M_BCR`). Add a line before that line to enable the external bus interface. The instructions will then look like this:

```

    movep    #$040821,x:M_AAR0        ;Compare 8 most significant bits
                                           ;Look for a match with address
                                           ;Y:0000 0100 xxxx xxxx xxxx xxxx
                                           ;No pack, no mux, Y enabled
                                           ;P and X disabled
                                           ;AAR0 pin active low
    movep    #$012421,x:M_BCR        ;One ext. wait state

```

See the *56300 Family Manual* for a complete description of the AAR0 register.

This modification causes the 64k (65,536 words) SRAM to be located in the memory space between `Y:$040000 – Y:$04FFFF`. Now modify your recirculating delay program so that the right channel uses the entire SRAM as its delay line. *NOTE* that there is a limitation in the valid modulo length for the m registers

of the address generation unit. Figure out a way to use the entire 64k delay. Listen to the output and observe the delay time. Verify the performance of your software.

Artificial Reverberation

Reverberation is characterized by a gradual decay of sound energy over time. The decay rate is usually characterized by the “reverberation time,” which is the time required for the sound pressure to decay 60 dB from its initial value. The reverberation time of a full-size concert hall is usually in the range of 2-3 seconds. Therefore, reverberation filters must involve long time delays. In principle, one could apply a tapped delay line (FIR filter) to create artificial reverberation. The tap weights of the delay line would be the sampled impulse response of the concert hall. However, there are usually thousands of reflections during the reverberation time, so due to memory and computation constraints the processor is usually only capable of synthesizing perhaps a few hundred reflections. The computation required for the FIR approach to reverberation may therefore be unreasonable—at least in the real time case.

As observed above, an IIR recirculating filter provides a computationally efficient approach to creating delayed “echoes” for reverberation. The infinitely long, gradually decaying characteristic of reverberation can be modeled by the impulse response of a suitable IIR recirculating delay filter. Unfortunately, the impulse response of this IIR is too regular. We get the sequence of gradually decaying, regularly spaced impulses as shown in Figure 4. This regularity results in temporal and spectral patterns that are readily audible and sound unnatural. Real reverberation has an echo density that increases with the square of time; the echo density of the simple IIR reverberator is constant. To use an IIR filter for reverberation, we need to find a way to increase the density of its impulse response.

A feedback “unit reverberator” is shown in Figure 5. An example frequency response of this reverberator is shown in Figure 6.

A common approach to increasing the complexity of the impulse response is to combine several unit reverberators in parallel as shown in Figure 7. Here, the unit reverberators are feeding two all-pass filters in series. The parallel unit reverberators have differing delay times, which are incommensurate so that the individual response resonances do not coincide. The all-pass filter is almost the same as the unit reverberator, but it has a feed-forward signal path in addition to the feedback path (see Figure 8). This adds N zeroes to the response, and each zero is located at the same angle as one of the poles but with a radius that is the reciprocal of the pole radius. This makes the steady-state frequency response magnitude uniform over frequency (by definition). The purpose of the all-pass filters is to increase echo density without interfering with the regular unit reverberators. By simply combining these basic building blocks imaginatively, we can produce electronic reverberation with little coloration of the signal.

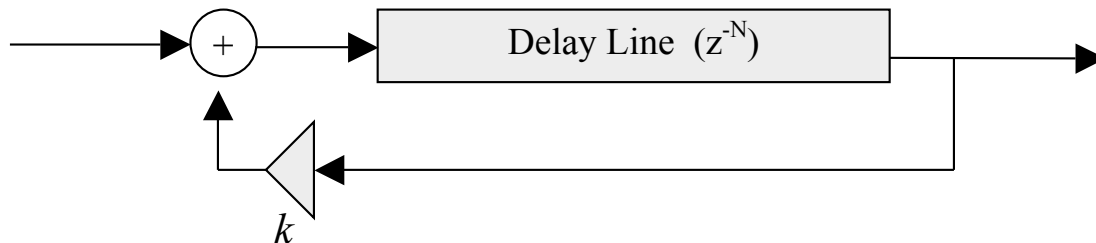


Figure 5: Unit comb reverberator

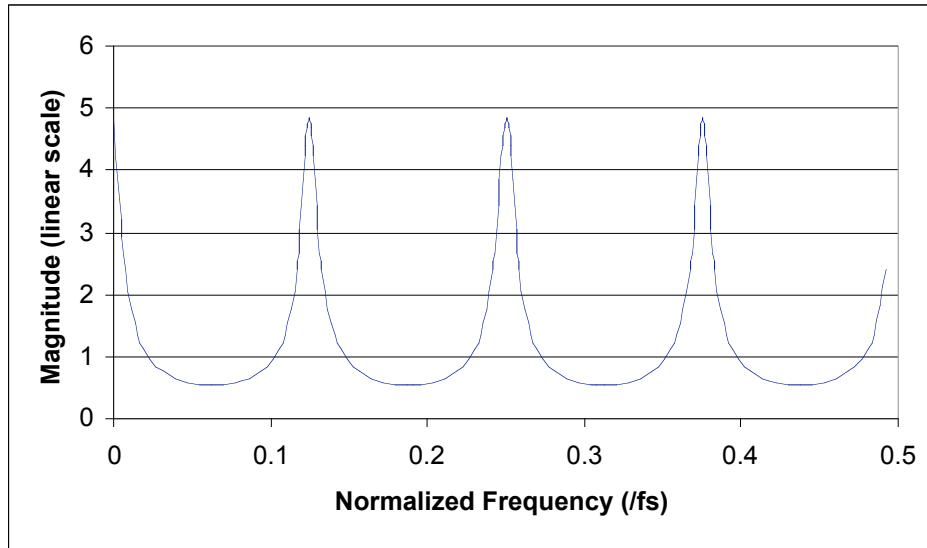


Figure 6: Frequency response of unit comb reverberator ($N=8, k=0.8$)

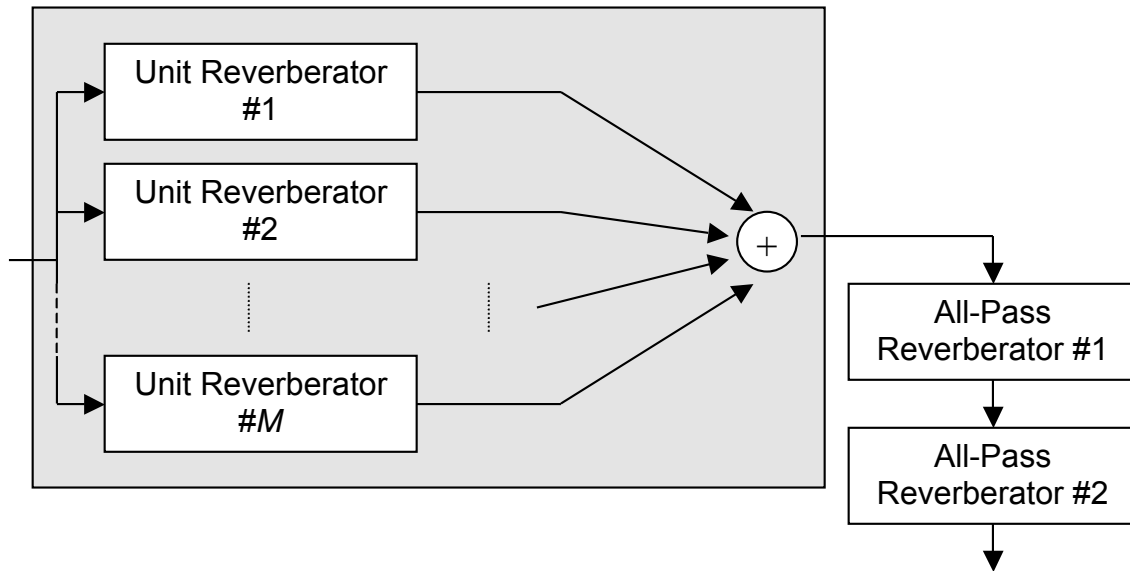


Figure 7: Multiple unit reverberator system

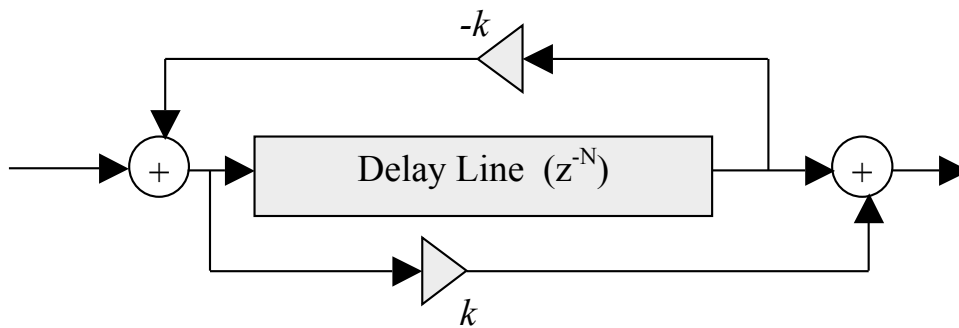


Figure 8: Unit all-pass reverberator

⇒ Exercise D: Four Parallel Unit Reverberators

Write DSP code to construct a simple reverberator comprising four parallel unit reverberators. Delay times should be in the following ratios:

```
delrat equ @POW(2.0,1.0/4.0)
delay1 equ @CVI(50.0*samp_rate/1000.0)      ; you can change "50.0"
                                           ; to scale the reverb time

delay2 equ @CVI(delay1*delrat+0.5)
delay3 equ @CVI(delay2*delrat+0.5)
delay4 equ @CVI(delay3*delrat+0.5)
```

where @POW, and @CVI are assembler functions. @POW(a,b) tells the assembler to calculate a^b , i.e., a to the b power. @CVI(val) takes the integer part of val, i.e., truncates a fractional number to an integer. Be sure to see the *56300 Assembler Manual* for the complete definition of these and other assemble-time functions!

Note that the delay times are related by the ratio delrat, which is an irrational number. That relationship assumes minimal coincidence of the resonances. Use the following gains:

```
comb_k equ 0.83
combk1 equ comb_k*(1-0.46)
combk2 equ comb_k*(1-0.48)
combk3 equ comb_k*(1-0.50)
combk4 equ comb_k*(1-0.52)
```

Since the code for each of the four unit reverberators will be essentially identical, you may choose to use a subroutine call (passing in the appropriate pointers, gains, and modulus), or perhaps even better would be a *macro* reference. Subroutine calls generally minimize the amount of program memory used, but they may be less efficient in computation due to the overhead of the subroutine setup, call, and return. A macro is interpreted by the assembler and inserts replicas of the specified code into program memory, so computation is fast—at the expense of program memory. For this exercise we probably won't be limited by computation or by program memory, so you can probably choose according to your own preference. Think about the changes and effort that would be required to make a small change to each unit reverberator, or to add a few more unit reverberators in parallel.

If you write your code as a sequence of similar instructions, you should try to keep a pointer for each unit reverberator in its own address register so that you don't have to move the pointer back and forth between the address registers and memory.

Begin with a single reverberator unit and listen to the sound with music for input. Compare this to the echo generator. This structure has a frequency response that resembles a pointy comb (see Figure 6, and also compare to the feed-forward (zeroes) response of Figure 2). With long delays, you will hear the impulse response of the comb filter. It corresponds roughly to the sound heard from the impulse response between two parallel blank walls. You should be able to hear the spectral modification, particularly with shorter delay times. Sometimes these effects are used as sound effects for music or film production.

Now using all four unit reverberators, play with their parameters. How does it sound? What parameter measurements can you make (reverb time, for example)? It is easy to make the reverberator sound bad, but it may be possible to make it sound better than it does with the initial choices for the parameters. Make some modifications and record a set of parameters that sounds good to you.

Additional exercise for Graduate Students:

Add a cascade of two all-pass unit reverberators to the output of the parallel bank of comb filters. Use the following specifications for the all-pass units:

ALLP1D	EQU	$6.4 * \text{SRATE} / 1000.0$; 6.4ms
ALLP1K	EQU	0.7	
ALLP2D	EQU	$6.7 * \text{SRATE} / 1000.0$; 6.7ms
ALLP2K	EQU	0.7	

Listen to the sound of one of the all-pass filters alone. Even though the filter has a constant magnitude at all frequencies (all-pass), it causes an audible coloration of the response. Why is that? What does the phase response of the filter look like?

Report and Grading Checklist**A: Flanger effect**

Code listing for flanger routine, with comments.
Measurements and data verifying design and implementation.
Written discussion of results.

B: Recirculating delay line

Part 1: z -transform expression and pole locations for several values of N and k .
Part 2: Code listing with comments for recirculating delay line.
Written discussion and comments on input scaling issues.

C: Long recirculating delay line using external SRAM

Code listing with comments for long external recirculating delay.
Written discussion and verification of design and implementation.

D: Unit reverberators and artificial reverberation

Code listing with comments for unit (comb) reverberators.
Written discussion and verification of design and implementation, including scaling issues, if any.
Suggested “improved” values of delay and feedback gain.
Grad students: code listing and discussion for 4 parallel comb plus 2-all-pass cascade reverberator.

Grading Guidelines (for each grade, you must also satisfy the requirements of all lower grades):

F	Anything less than what is necessary for a D.
D	Exercise A with results and discussion.
C	Exercise B with results and discussion.
C+	Exercise C with results and comments.
B	Exercise D with complete results and lucid comments.
A	Live demonstration (during office hours) of fully functional software. Demonstrate for Nathan Gilles (TA) or Prof. Maher.

Note: grad student grading also requires the additional all-pass code for exercise D.