# Lab 5: Audio Effects

Professor Deepa Kundur

## Objectives of this Lab

- This lab addresses the implementations of audio effects and their combination (a synthesizer).

## Prelab

Prior to beginning this lab,

- You must carefully read over this lab in order to plan how to implement the tasks assigned. Please highlight all the parts you need to show or answer for the TA, so that you do not miss any graded points during the in-lab component or for the report.
- For each of the five audio effects discussed in the "Testing in Simulink" section, you must design a block diagram discussing how you will use the various sub-blocks and operators discussed to create the effect. This must be included in the final report.

## Deliverables

- During the lab, you must show your TA the Simulink results discussed in the lab instructions; and
- After the lab, each student must submit a separate report showing your design for each special effect.

## Grading and Due Date

The lab will be graded on correctness, comprehensiveness and the insight you are able to provide when answering the questions. For full points, lab reports should be written in complete sentences with correct grammar.

Please note STRICT DEADLINE for report on the course web site.

# Lab 5: Audio Effects

TA: Mr. Lin Lai, Instructor: Dr. Deepa Kundur

## Introduction and Background

You're standing on stage in front of millions of Americans who are watching and waiting for your vocal floodgates to open and unleash mad talent on the one-millionth installment of American Idol. You're hoping the judges will say "You are the best thing since Christina Aguilera or Josh Groban," but when you open your mouth, only a squeak comes out, and they are not impressed. This may not be the nightmare you think it is because you are in luck! You have taken a real-time DSP course. It's not hard to imagine that there are devices out there that can enrich your voice. We are going to give your voice a total makeover through audio effects that you shall program on the DSP. We will look at five basic audio effects: reverberation, upsampling, downsampling, flanging effect, and chorus effect.

As discussed in the lectures, you can think of sound as a signal. The amplitude of the signal gives us the volume (loudness or softness) of the sound, and the frequency (how fast the signal is changing), gives us the pitch. Finally, the harmonics around the main frequency give you the timbre (what kind of instrument: voice, guitar, piano, etc.). The phase of the signal will give you relative timing information.

### Reverberation

So how do you create reverberation (an echo effect)? When you stand in the Grand Canyon and yell "I was here!" you hear the same statement echo back a few times softer each time without interfering with one another. If you stand in a smaller hall, the echo will bounce back much quicker, and may interfere with your speech before you're done with the entire sentence. However, the echoes always come back softer each time as it loses its energy via reflection on some surface. The simplest way to model reverberation is as follows. To create a single echo (called one "delay line"), the input signal is delayed by a certain amount and then added to the original (undelayed) signal. To create multiple echoes, multiple such "delay lines" are placed in parallel and then added together. To make it sound like the echoes gradually die off, an appropriately chosen gain can be inserted after each delay line before adding the parallel branches together.

### Downsampling (Decimation)

Say you want to modulate your voice to sound like Alvin and the Chipmunks. A simple way to do this is simply to downsample (decimate) the audio signal. Recall from DSP theory that decimation in time is equivalent to expanding the frequency spectrum by adding an appropriately located ``image''to represent high frequency components. This expansion gives you the higher frequencies you normally wouldn't have. Another intuitive, but less theoretical way of thinking of this, is if you get rid of every other sample point, a signal that may originally look smooth will now become a bit choppier – and a choppy signal has more high-pass components.

When downsampling, you want to make sure that the sampling theorem is still satisfied to avoid aliasing. Before downsampling a signal, you should send it through an *anti-aliasing lowpass filter* with a normalized cutoff frequency of $\pi/D$, where $D$ is an integer downsampling factor.

A cautionary note when conducting this lab: When implementing in Simulink, you may have some problems hearing the downsampling effect due to the Simulink block “To Wave Device” which can automatically change the sampling frequency of the obtained signal thus cancelling the effect created. To solve this problem, instead of reading the signal directly with the “To Wave Device” block, try saving the signal to the workspace and read it using the MATLAB functions sound(‘your signal’, Fs) or wavplay(‘your signal’, Fs) with ‘your signal’ being the variable corresponding to your saved signal and Fs being the sampling frequency of your original input.

## Upsampling

Now, say you want to sound like Optimus Prime or Darth Vader. A simple way to do this is to upsample the signal. DSP theory tells us that upsampling (adding zeros in every other element) is equivalent to frequency-compression in the frequency domain. By compressing your voice into the lower frequency ranges, it’ll sound pretty deep and ominous. Much like downsampling, upsampling requires a lowpass filter; however, this filter is placed *after* the upsampler as an interpolation filter. The normalized cutoff frequency, again, should be $\pi/K$, where $K$ is an integer downsampling factor.

To make the Darth Vader effect a little better, you may need to upsample by a non-integer factor (i.e. by a fraction). To do this, you would actually upsample by an integer factor and then downsample by a different integer factor. For example, to upsample by 3/2, you could upsample by 3 and then downsample by 2. Cascading upsampling and then downsampling makes use of two lowpass filters, one for each process? As discussed in class, both are not needed. You can combine them into one lowpass filter (with the more restrictive – i.e., lower – cutoff frequency) placed between the upsampler and the downsampler. Usually, the upsampling factor $K$ is greater than the downsampling factor $D$, so the cutoff frequency is set to $\pi/K$ which is lower than $\pi/D$.

Once again, you might have some problems hearing the effect, which may be due to the “To Wave Device” Simulink block automatically changing the sampling frequency of the obtained signal thus cancelling your effect. To solve this problem, try saving the signal to the workspace and read it using the MATLAB functions sound(‘your signal’, Fs) or wavplay(‘your signal’, Fs) with ‘your signal’ being the variable corresponding to your saved signal and Fs being the sampling frequency of your original input.

## Flanging Effect

Flanging is usually applied to music and is most commonly described as a cool “whooshing” sound in the background. Figure 1 below shows a simple block diagram of how flanging is produced. First, the input signal is delayed by a time-varying amount. The amount of delay actually varies as a sine wave, called the Low Frequency Oscillator (LFO)[1]. Literally, the value of the sine wave at any point in time determines the amount by which the input at that moment is delayed. Being a sine wave, the LFO has several parameters. First, the offset of the LFO from zero is called the *delay* and generally is in the range of 1-10 milliseconds. Second, the amplitude of the sine wave is called the *depth* and is generally only a few milliseconds. Third, the frequency of the LFO is called the *rate* and is generally just a few Hertz or less (hence the name, LOW Frequency Oscillator). After the signal is delayed, it is multiplied by

[1] Some flangers use a triangle wave for their LFO instead of a sine wave. We’ll just stick with a sine wave because it’s easier to deal with in Simulink.

a constant scalar value, called the *mix*. Finally, the delayed signal is added to the original signal to produce the output.

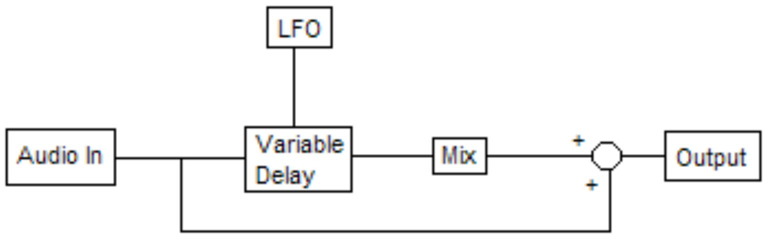

**Figure 1:** Flanging Effect

### Chorus Effect

The chorus effect is a technique used to make a single person's voice (or a single instrument) sound as if it is multiple voices (or instruments) talking (or playing) at once. Interestingly enough, the chorus effect is produced in the same as the flanging effect. The ONLY difference is found in the parameters for the LFO. Namely, instead of the delay being 1-10 ms, it is a little bit higher, in the range of 30-50 ms (if you make the delay too much longer than this, then the human ear will detect it as an echo). Thus, once you have a flanger working, it should be simple enough to make the step to the chorus effect.

## Design and Implementation

We've described 5 signal processing techniques to make you sound like you're in the Grand Canyon, conversing with Alvin and Chipmunks as well as robots and half-man-machine concoctions. Now it's time to design these puppies. We've purposely not given you any more details than you need, because we want you to think about what was described in the Introduction and turn these ideas in your mind to come up with simple designs.

### Testing in Simulink

First, you'll design your audio effects and test them in Simulink Normal mode. Build each audio effect listed below in a separate Simulink model. You can send the output of each system to either a "To Wave Device" block, which will make the output available to your computer's speakers, or a "To Wave File" block, which will allow you to write the output to a .wav audio file[2]. Both of these source blocks are available in Signal Processing Blockset → Signal Processing Sinks. We recommend that you set the output of whatever source you choose to be sample-based (frame size of 1), with the exception of the reverberation effect (see below). For the input to each system, you can use either a "From Wave File" block, which reads audio from a .wav file, or a "From Wave Device" block, which allows you to provide input to the system from your microphone. Both of these blocks are available in Signal Processing Blockset → Signal Processing Sources. Finally, you want to be sure to choose the stop time of your Simulation carefully. For example, if you're taking input from your microphone and sending the output

[2] After the .wav file is written, you can play it on your computer through any media player; the file is created in the MATLAB Current Directory, so be aware of what your Current Directory is set to.

to your speakers, you'll probably want the stop to be indefinite (i.e. infinity); otherwise, you'll probably want to set a finite stop time.

1. Using only Integer Delay Blocks, adders, and multipliers, design a reverberation effect that consists of four separate delay lines. Note that you'll have to buffer the input into a frame-based signal that is at least as large as your largest delay block. You can do this by either changing the settings of your source block or by adding a Buffer block. First, choose the delays and gains to mimic a small-hall environment. Then, change the delays and gains to sound more like the Grand Canyon.
2. Use a downsampler and an appropriate lowpass FIR filter to create the helium-junky effect. Note that you may have to experiment with different downsampling factors depending on the sampling rate of your source. In general, the higher the sampling rate of your source, the higher the downsampling factor you will need to produce a noticeable effect. Make sure to change your filter every time you change your downsampling factor; you may want to use FDAT to quickly design and import new filters to your model.
3. Use an upsampler, a downsampler, and an appropriate lowpass FIR filter to create the Darth Vader effect with a non-integer upsampling factor. Experiment with different non-integer factors somewhere between 1 and 2. For this effect, use the microphone as the source (set the sampling rate to 8000) and the To Wave File block as the sink. Also, right before the To Wave File block, you should probably boost the signal by a gain of about 20 dB so it is loud enough (you can use the dB Gain from Signal Processing Blockset → Math Functions → Math Operations). Record a few seconds of speech and then play the resulting .wav file on your computer. This is probably the best way to hear this effect using Simulink only. Don't worry if it doesn't sound good right now; it should work a lot better when you implement it on the board.
4. Using a Variable Integer Delay block from Signal Processing Blockset → Signal Operations - as well as a multiplier, an adder, and a sine wave source - create a flanging effect. Note that if you use an analog sine source, you'll have to use a zero-order hold block to discretize it. CAUTION: your LFO will control the number of *samples* that are delayed; make sure this corresponds correctly to the *time* delays (in units of seconds) specified in the introduction and background section of this lab to get the intended effect. That is make use of the fact that a delay of $k_0$ samples is equal to a delay of $k_0$/Fs seconds and set $k_0$ for the intended temporal delay. Finally, note that the flanging effect is usually applied to music, so if you try to use it on speech only, you won't necessarily hear anything too noticeable. Regardless of the type of audio you use it on, just make sure your TA approves of your flanger.
5. Create a chorus effect. Again, make sure that your LFO parameters make sense based on the discussion in the introduction and background regarding temporal delay (convert this to sample delay using the Fs value). This effect should work better on speech than the flanging effect.

## Implementation on the Board

Now it's time to load your audio effects onto the board. For all five effects, replace the audio input and output blocks in your models with the usual ADC, DAC, and data type conversion blocks. Also, remember to add the DM6437 preferences block and to change your Configuration Parameters (if you don't remember, refer to lab 2). Build each model, load and run it on your board, and verify that each effect is working properly. Here are some points to consider as you proceed:

1. It's probably best to set the output of the ADC to be sample-based (frame size of 1). The only exception is the reverberation effect, for which you'll have to make the frame size at least as large as the largest delay block.
2. For the downsampling model, use a downsampling factor of 2. This should produce a perfectly reasonable effect. Remember to change your lowpass filter!
3. For the flanging and chorus effects, if you used a continuous-time sinusoidal source for the LFO, you'll have to scrap it here because the C6437 can't handle continuous-time from Simulink. You MUST use a digital sinusoid source (use the Sine Wave block from Signal Processing Blockset → Signal Processing Sources). Note that this block doesn't allow you to add an offset to the sinusoid (for the delay parameter). The easiest way to add an offset is to use a Bias block from Simulink → Math Operations.

Ensure that each effect works properly on the board. With all five effects working, add a switch to each effect and sum all of the outputs together to create a synthesizer. The goal here is to use the switches on the board to control which effects are heard at the output. The combination of certain effects with others can create an audio effect that each on its own could not. Just like in the previous lab though, try to keep the filter order of all filters within your effects low. If the total filter order is too high, there will be a noticeable amount of latency at the output.

Your Name:_____________________________________ UIN:__________________________

## Questions

Please answer the following questions to be included with your lab report.

1. *Echo Effect*. For a given sound signal (you may use the audio sample I have provided on the course website or make up/find your own), create a single echo effect by setting alpha to 0.25 and n0 to 10000. Describe qualitatively what the result sounds like. Plot your original sound signal and the signal with the echo on separate graphs. You can use MATLAB if you like.

2. *Reverberation.* Take your reverb filter from the lab (which is an FIR/all-zero filter) and add feedback to it to generate a Direct Form II IIR filter (with both poles and zeros) as discussed in class. You will have $M$=4 (from the lab specification) and let $N$=3 (which represent three feedback paths).

   a. Describe qualitatively what the result sounds like and what parameters give you a rich reverb.

   b. Plot your original sound signal and the reverb signal on separate graphs.

   c. Graph the impulse response of your reverb filter up to a point it seems to die out.

   d. Analytically provide the frequency response of the reverb filter.