Guitars, Drums, and Comb Filters 6

Electrical Engineering 20N
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Hsin-I Liu, Jonathan Kotker, Andrew Lee, Howard Lei, and Babak Ayazifar

1 Introduction

In this lab, we will explore further applications of the filters that we have seen so far in lectures, discussion sections, and previous lab sessions. In particular, we will use a comb filter to create the sound of a guitar string being plucked, and we will use other filters to make this sound as realistic as possible. Also, with a small modification, we can use the same model to create drum sounds. Along the way, we will explore different features of the impulse response and the frequency response of these filters. The methods discussed in much of this lab were formulated by Karplus and Strong [1].

1.1 Lab Goals

- Implement further practical applications of discrete-time filters in LabVIEW.
- Explore relationships between sampling frequency, fundamental frequency, and delays.
- Get acquainted further with subVIs and LabVIEW LLBs to make block diagrams cleaner and clearer.

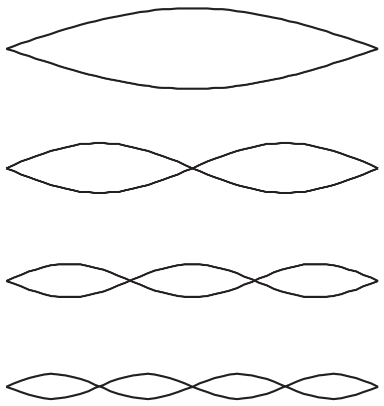
1.2 Checkoff Points

2 Pre-Lab Section

2.1 Sound Mechanics of String Instruments: Theory

String instruments, such as a guitar string, create vibrations that are similar to the simple sine wave model, but have more than one **mode of vibration**, as illustrated in Figure 1.

Figure 1 Vibration Modes of a Guitar String.



Each of these modes of vibration produces a different frequency. The first mode of vibration in the figure produces the lowest frequency, called the **fundamental frequency** (or the **first harmonic**), which is typically the frequency of the note being played; in the case of the A-440 sound, for instance, the fundamental frequency is 440 Hz. The next mode produces a component at *twice* that frequency, 880 Hz; this component is called the **second harmonic**. The third produces three times the fundamental frequency, 1320 Hz, while the fourth produces four times the fundamental frequency, 1760 Hz; these components are the **third harmonic** and the **fourth harmonic**, respectively. This pattern continues, and in general, the n^{th} harmonic produces a frequency that is n times the fundamental frequency.

If the guitar string is undamped, and the fundamental frequency is f_0 Hz, then the combined sound is a linear combination of harmonics. This sound can be written as a continuous-time function y, where,

$$\forall t \in \mathbb{R}, \qquad y(t) = \sum_{k=1}^{N} c_k \sin(2\pi f_k t), \tag{1}$$

where N is the number of harmonics and f_k is the frequency of the kth harmonic. The values of c_k are the relative weights of these harmonics. These values will depend on how the guitar was constructed and how it is played, and will affect the **timbre** of the sound.

2.2 Phase Response of Comb Filters

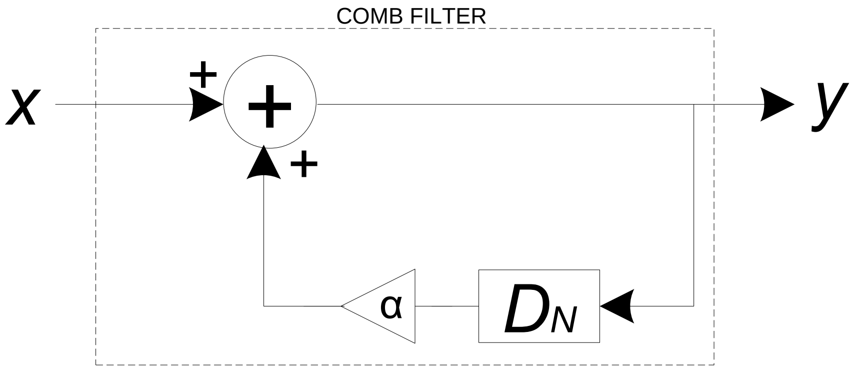
In lab 05 we have focused on the magnitude response of filters when performing the lab activities. In this lab, we will shift our focus to the phase response of a comb filter, and determine how its properties are useful for our purposes.

Recall that a comb filter is a discrete-time LTI system given by the following LCCDE:

$$\forall n \in \mathbb{Z}, \qquad y(n) - \alpha y(n-N) = x(n). \tag{2}$$

The block diagram representation of this system is shown in Figure 2. We assume that the system is initially at rest.

Figure 2 A basic discrete-time comb filter.



This system is commonly used to model echo effects in sound signals. The parameter $\alpha \in \mathbb{R}$ is used to model the attenuation or amplification of the sound signal with each echo. The parameter N is a positive integer used to model the amount of time delay in every echo. As you have seen in lab 05, the corresponding frequency response F_C is given by

$$\forall \omega \in \mathbb{R}, \quad F_C(\omega) = \frac{e^{i\omega N}}{e^{i\omega N} - \alpha}. \tag{3}$$

Let us explore this frequency response, specifically its phase, more closely.

1. Plot the phase response of a comb filter for different values of N; specifically, for values $N = 1, 2, 5$. How will the phase plot change as N changes (increases or decreases)?

2. N was restricted to be a positive integer. Based on your observations in the previous step, what do you predict will happen to the plot if we do not restrict N only to the positive integers, but allow N fractional values as well?

While a fractional delay block does not translate to an intuitive example in a discrete-time system (why?), we can still create a system that implements a fractional delay by cascading other filters with the comb filter to manipulate the phase response of the net filter. This is one concept that we will exploit in this lab.

2.3 Moving between Frequency Units

In lab 05 we had to move between frequencies represented in different units. Please read "The Many Faces of Frequency" for more detailed explanations. You can find it here: `http://ptolemy.eecs.berkeley.edu/eecs20/labs/LabVIEW_Labs/Lab06_old/TheManyFacesofFrequency.pdf`
In a nutshell, if we had a signal with a continuous-time frequency of f_c cycles per second, and the signal was sampled with a sampling frequency of f_s samples per second[1], then the discrete-time frequency of the **sampled** signal, Ω_D radians per sample, is given by

$$\Omega_D = \frac{2\pi f_c}{f_s}. \tag{4}$$

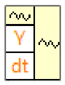
2.4 Checkoff Exercises

1. We will begin by adapting the `CF Time Domain` VI that you made during the post-lab exercises for lab 05. Make a copy of this VI under the name `CF Impulse Response.vi`. Delete everything except for the `For Loop` that represents the comb filter.

2. Feed an impulse signal of **13250 samples** as input. In other words, feed, as input to the comb filter, a signal that has a value of 1 at time 0 and a value of 0 elsewhere. Given a sampling frequency of 26.5 kHz, **how long does this signal last (in time)?**

3. Plot the impulse signal. When doing so, ensure that the x-axis, which represents time, *correctly* represents the time at which each sample is generated.

4. Now, we create a waveform based on the output of the system. Use the `Build Waveform` block under `Programming` $\rightarrow$ `Waveform`. Set `dt` to be `1/26500` (why?). Feed the output signal of the comb filter to the `Y` input.



5. Connect the output of the `Build Waveform` block to a `Waveform Graph`, available on the front panel under `Modern` $\rightarrow$ `Graph`. Change the plot type to a stem plot.

6. On the front panel, right-click on the graph and select `Visible Items` $\rightarrow$ `Graph Palette`. This makes visible a palette of graph tools that enable us with different ways of viewing the graph, as shown in Figure 3. Explore the various options available, and use the zoom tools to zoom into different areas of interest.

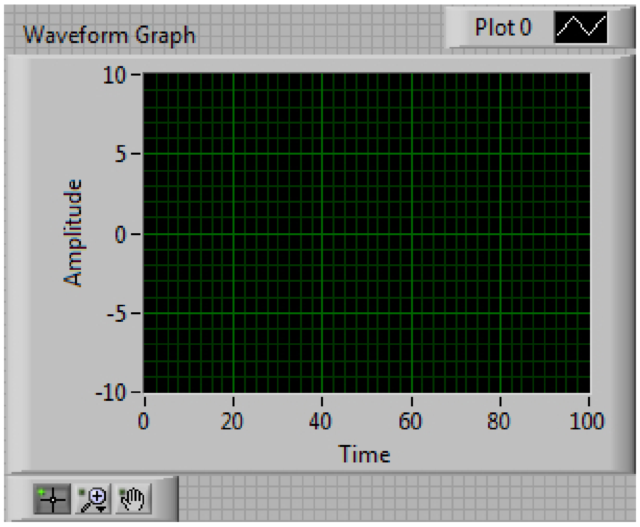
7. Use $N = 100$ and $\alpha = 0.99$. Run the virtual instrument, and observe the impulse response of the comb filter. Vary N to explore the effect of N on the impulse response.

8. Notice that the output is not purely periodic. However, we can interpret the output signal differently: also notice that it can be modeled by an impulse train (which *is* periodic) but *modulated* by a decaying exponential.

9. Notice that the comb filter generates the impulses in the impulse train every N samples. With the help of the LCCDE of the comb filter, **explain why this is the case**. In more rigorous terms, why is $f_C(n) = 0 \,\forall\, n \neq kN, k \in \mathbb{Z}, k \geq 0$?

[1]One subtlety regarding the units: one Hertz is defined to mean *per second*. f_c has units of cycles *per second* and f_s has units of samples *per second*, but "cycles" and "samples" are not conventional units. Thus, it is not uncommon to see either f_c or f_s expressed in just Hertz. Context determines whether or not we are talking about cycles, samples, radians, or other quantities.

Figure 3 A waveform graph with the graph palette visible.



10. Since the impulse train is (almost) periodic, it has a continuous-time *fundamental frequency*, the lowest frequency with which the signal repeats itself. **Determine the fundamental frequency (in Hertz) of the impulse train, as a function of *only* the sampling frequency f_s and N.**

    We can derive the formula as follows: We know that the nonzero samples of the discrete-time impulse response are separated by N samples. We also know that, if the discrete-time impulse response was obtained by sampling a continuous-time signal, then adjacent samples are separated by T_s seconds per sample, where T_s is the sampling period. With this in hand, we determine how many seconds separate two adjacent *nonzero* samples of the impulse response. Finally, we invert this relationship to determine how many adjacent *nonzero* samples of the impulse response occur per second—the fundamental frequency of the impulse train.

11. **Determine the numerical value of the fundamental frequency (in Hertz) of the impulse train that you generated in step 7.**

2.5 Submission Rules

1. Submit your files *no later than* 10 minutes after the beginning of your next lab session, during the week of **April 4, 2011**.
2. Late submissions will *not* be accepted, except under unusual circumstances.
3. If the pre-lab exercises are not performed, you will get an immediate zero for the entire lab.
4. These exercises should be done *individually.*
5. Keep your work safe for further usage in the in-lab sections.

2.6 Submission Instructions

1. Log on to bSpace and click on the `Assignments` tab.
2. Locate the assignment for `Lab 6 Pre-Lab`.
3. Attach the following files to the assignment:

(a) A text document containing your responses to the emboldened questions in section 2.4.
(b) The VI `CF Impulse Response`.

3 In-Lab Section

Throughout the in-lab sections, the guide asks you to consider a few conceptual questions. Try your hand at these questions and keep your answers handy. If you find that a question does not make sense, or if you need help in answering a question, feel free to ask your lab TA.

3.1 Guitar Hero: A Good Guitar Simulation

The goal of this portion of the lab session is to show how the impulse response of a comb filter can generate a sound wave similar to that produced by a musical instrument, specifically a guitar. We will start off with a simple model and develop the model as we progress through the lab.

Assume that the sampling frequency used is the same as that used in the pre-lab sections: 26.5 kHz.

1. Open the `CF Impulse Response` VI that you created in the pre-lab section.
2. Change the type of the plot of the output waveform to be a continuous-time plot.
3. Since we have already converted the output of the comb filter into a waveform, we can use the handy `Play Waveform` block, found under `Programming` → `Graphics and Sound` → `Sound` → `Output`, to *listen* to it.
4. Set N to 101 and α to 0.99 and run the virtual instrument. Can you identify what you hear?
5. In pre-lab section 2.4 step 10, you derived a relationship between the fundamental frequency of the impulse response, the sampling frequency f_s, and N. Using this relationship, determine if there is a positive integer N that can generate the musical note A, whose frequency is 440 Hz. If so, what is its value; if no, why not?
6. If N were allowed to be a real number, however, would we be able to generate the A-440 signal? If so, what is its value; if no, why not?
7. Load the `CF Frequency Domain` VI that you created in the post-lab sections of lab 05, where you had plotted the magnitude and the phase of the frequency response of a comb filter.
8. Run the VI to plot the magnitude and the phase of the frequency response $F_C(\omega)$ with $\alpha = 0.99$, $N = 40$, and a step size of 0.00005 for your array of frequencies `w`. As a sanity check, recall from pre-lab section 2.2 that the number of repetitions in the frequency range $(-\pi, \pi]$ should be N. This periodic nature of the frequency response hints us towards the possibility of using the comb filter to simulate a guitar string, since we know that all the frequencies contained in the sound produced by an undamped guitar string are multiples of the fundamental frequency.
9. The *fundamental frequency* is the lowest positive nonzero frequency in the sound produced by an undamped guitar string, or in this case, the sound representation of the impulse response of the comb filter. Use the magnitude response plot for the comb filter, which you generated in step 8, to determine the approximate value of the fundamental frequency (in *Hertz*), with the help of the relationship expressed in section 2.3. (It is possible to get an exact value, due to the periodic nature of the frequency response.) Use the relationship you determined in step 10 of pre-lab section 2.4 to confirm your answer.
10. What is the value of the second harmonic? The third harmonic? The kth harmonic?

11. We recapitulate the equation of the continuous-time signal $y(t)$ representing the sound produced by an undamped guitar string here for convenience:

$$\forall t \in \mathbb{R}, \qquad y(t) = \sum_{k=1}^{N} c_k \sin(2\pi f_k t), \tag{1}$$

In the equation above, the coefficients c_k are real, as they must be if $y(t)$ is to be real. The values of the coefficients c_k in Equation 1 can be interpreted as **twice** the heights of the peaks in the magnitude response. Using the relationship above, explain why. (*Hint*: What does the magnitude of the frequency representation of one sinusoidal wave look like?)

12. What is the relationship between the coefficients c_k of the various harmonics: are they the same or are they different? Explain your answer.

3.2 That Can't Be Real (Get It?): A Better Guitar Simulation

Our goal in this lab session is to create a realistic guitar pluck sound. However, the guitar pluck sound that we created in section 3.1 sounds very mechanical. In this section, we will replace our earlier impulse input with an input containing a set of random values.

1. We will create another VI called `CF Random` using `CF Impulse Response` as a template.

2. Download the `Random Init` VI from the course lab page `http://ptolemy.eecs.berkeley.edu/eecs20/labs/`, and explore its block diagram. As structured, this VI will create an array of length `P`, the first `N` of which are random values between -1 and 1; verify this. The array produced by this VI will represent an input to the comb filter containing a set of random values. This input resembles a more realistic guitar plucking action.

3. In the VI `CF Random`, replace the impulse input with the output of the `Random Init` VI. Attach controls to inputs `N` and `P`. The value for the input `N` is the same as the delay N used by the filter.

   To import a subVI into another, we must right-click on a blank spot in the block diagram and choose `Select a VI...`.

4. Run the VI with α = 0.99, N = 58, and P = 13250. (With the P as given, how long, in time, is the input signal?) Listen to the output. Compare the sound produced earlier, when using the impulse as an input, to this new sound. Do they both have the same tone?

While we made the output less mechanical by using a different input, the output still has the same characteristics in terms of tonality. We will build off of this to further our model.

3.3 I Can't Believe It's Not a Guitar String: An Even Better Guitar Simulation

Real instrument sounds are more dynamic in their frequency structure. In other words, the frequency spectrum of the sound within the first few milliseconds of plucking the string is different from the spectrum a second or so later. Physically, this is because the high frequency vibrations of the string die out more rapidly than the low frequency vibrations.

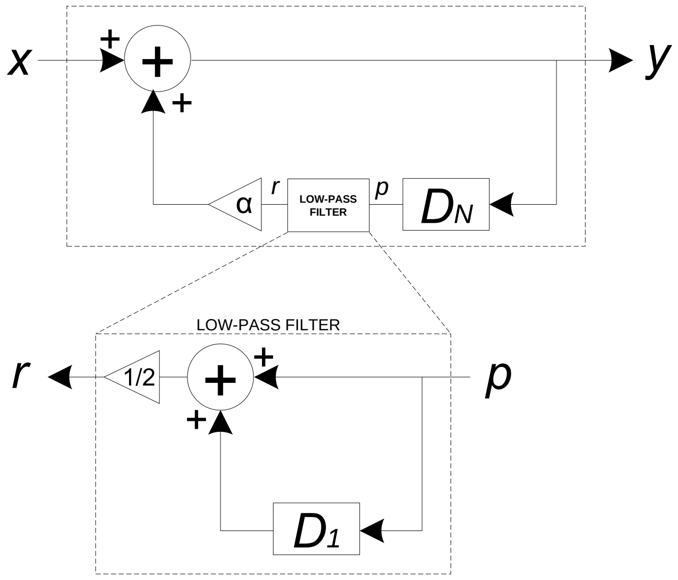
We can approximate this effect by modifying our VI and inserting a low-pass filter into the feedback loop, in order to filter out the high frequency components of the output signal. The resulting block diagram is shown in Figure 4, where the low-pass filter used is governed by the following LCCDE:

$$r(n) = \frac{1}{2}\left(p(n) + p(n-1)\right).$$

1. Determine the LCCDE that describes the block diagram of the filter shown in Figure 4. Your final answer should involve only the input signal $x(n)$ and the output signal $y(n)$. You may find the intermediate signals $p(n)$ and $r(n)$ useful in obtaining your answer.

2. As we progress further with the lab, we will be splicing other filters into the comb filter, and this process could make the block diagram of our VI messy and unintelligible. To overcome this, we will place every new filter into its own sub-VI, and insert this sub-VI into the current VI containing the comb filter. The following steps will guide us through this process. Save a copy of the `CF Random` VI under the name `Guitar String`, in a new folder also called `Guitar String`.

Figure 4 Low-Pass Filter embedded into a Comb Filter.



3.4 Creating a SubVI

One important feature of LabVIEW is its **modularity**: it is possible to encase different portions of a VI into smaller modules and to connect these modules to obtain a VI of similar functionality, but a VI that is easier to debug and neater to look at. In other words, it is possible to use one VI as a block in another; the former VI becomes a **subVI** of the latter. We have already seen a few of these: most of the non-trivial blocks, such as the `Y[i] = X[i - n] PtByPt` block, are actually VIs; you can double-click on any one of them to view the corresponding VI. SUBVI

In this section, we will encase our low-pass filter in a subVI to accomplish two things: we can abstract away how the low-pass filter works, and we can also keep a VI with a neat block diagram, preventing us from getting drowned in a lot of wires and blocks.

1. Create a new VI called `LPF for CF`, in the folder `Guitar String`: this VI will be a sub-VI for the comb filter, and will contain the low-pass filter that is embedded into the feedback loop of the comb filter, as shown in Figure 4.

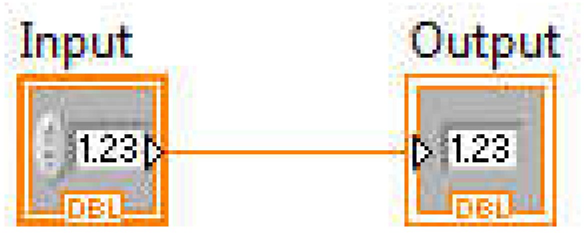
2. Notice that the input and output are *scalars*. This is because your subVI will be placed inside the `For Loop` representing the comb filter, and inside the comb filter, we only have access to one *sample* of the input signal, *not the entire signal* as one.

3. We will start by making `LPF for CF` a very simple subVI. It won't do anything particularly interesting at the moment, but in doing so, you will be familiarized with the things that go into making a subVI.

4. In order to create any subVI, we will need to create **controls** on all of our intended **inputs** and **indicators** on all of our intended **outputs**.

5. In the case of your `LPF for CF` VI, create one numeric control and one numeric indicator. In the block diagram, wire up the control directly to the indicator. Your block diagram should look like that of Figure 5.

Figure 5 Version 1 of the Low-Pass Filter.



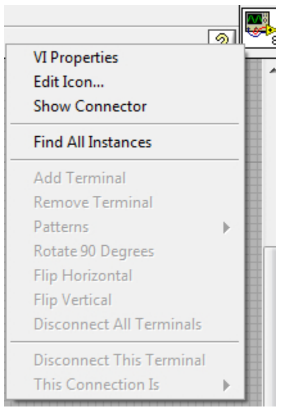
6. As you might expect, this subVI will simply pass its input to its output unchanged. In other words, it'll just act as a wire. Thus, once you plug it into the feedback loop as shown in Figure 4, the result should sound identical as before. We can take advantage of this fact to make sure that the subVI is working correctly.

7. Now, to make this subVI usable in the `Guitar String` VI, we have to "expose" the control and indicator to the outside world. We do this by hooking them up to the VI's **connector terminals**.

CONNECTOR TERMINALS

8. On the top right corner of your LabVIEW window, on the front panel, right-click on the **VI Icon** and select **Show Connector**, as shown in Figure 6. You will now see the **connector pane**, which describes the inputs and outputs of your VI.

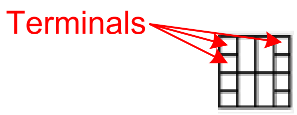
Figure 6 Showing the connector pane.



9. The connector pane, shown in Figure 7, is a pictorial representation of the terminals available to your VI; whether each terminal is an input or an output depends on what it is connected to. However, we recommend the convention that input terminals are on the left, while output terminals are on the right. You can change the pattern by right-clicking on the VI Icon again, going to the **Patterns**

submenu, and selecting a new pattern. Connect the numeric control to one input terminal, and the indicator to one output terminal: this connection is done by first clicking on the terminal, and then clicking on the relevant control or indicator on the front panel. If done correctly, the terminal will be colored in to signify that a connection has been made.

Figure 7 Connector pane with terminals (only a few are labeled).



10. One last thing: let us make our subVI easily identifiable. On the front panel, right-click on the icon for the VI near the top right and select `Edit Icon`, as shown in Figure 8. You will obtain the `Icon Editor`, as shown in Figure 9. Use the `Icon Editor` to create an icon for your VI: preferably one containing the words `Low-Pass Filter`. Perform your editions for `256 Colors`, and duplicate your creation into the other color schemes. Do not, however, spend a lot of time on this step; a simple descriptive icon will suffice.

Figure 8 Invoking the `Icon Editor`.

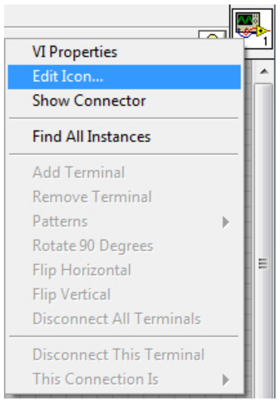
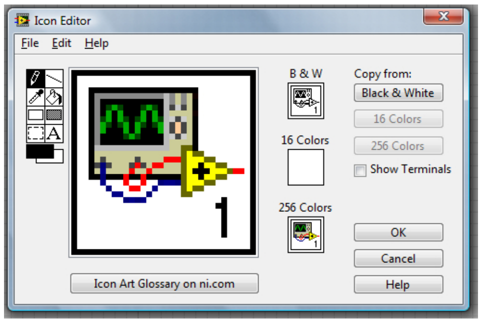


Figure 9 The `Icon Editor`.



Don't forget to save your work!

11. Congratulations! You now have a fully functional subVI that can be imported into any other VI. To use it, go to the block diagram of the `Guitar String` VI. Right-click any empty region in the block

diagram and click `Select a VI...`. Select the `LPF for CF` VI you made, and presto! Your subVI will appear as a usable block with the icon you made.

12. At this point, you should be able to hover your cursor over your subVI's connector terminals and see the input and output you have created for the subVI. Wire it into the feedback loop of your block diagram as the low-pass filter (I know it's not actually a low-pass filter yet!) as depicted in Figure 4.

13. Run your `Guitar String` VI with your subVI plugged in. As mentioned earlier, it should sound exactly as it did before. If it does not, look over the previous steps again.

14. If everything looks (or sounds) great so far, it's time to upgrade the subVI from being a simple wire to being a working low-pass filter. As you know, to implement a basic discrete-time low-pass filter, we have to hold on the last value of the signal to average the current value with. In past labs, we would often use a shift register to do this. However, we are now working inside a subVI, so using a shift register will not work cleanly (why?). Instead, let us understand a block in our arsenal that you may have encountered from time to time: the **Feedback Node**.

Feedback Node

3.5 I Need Some Feedback

`Feedback Nodes` ( ) also hold values from one iteration to the next. In other words, `Feedback Nodes` delay their input by one iteration, just as a `Shift Register` does. Several `Feedback Nodes` in series (one after the other) will also achieve the effect of delaying the signal for as many iterations as there are nodes. LabVIEW inserts `Feedback Nodes` automatically when it detects that a signal is being combined with itself, but modified in some manner. Despite their similarities, `Feedback Nodes` and `Shift Registers` are **not identical in functionality**; however, for our limited usage, they can be used interchangeably. For this lab session in particular, `Feedback Nodes` have the visual advantage of resembling the delay element that delays its input by one sample.
Please note that since the subVI will be placed inside an external `For Loop` representing the comb filter, **do not initialize any** `Feedback Nodes` **or shift registers you may use!** Doing so will cause them to lose history, because they would repeatedly initialize every time the external `For Loop` representing the comb filter performs an iteration.

15. With `Feedback Nodes` under our belt, implement the low-pass filter subVI as illustrated in Figure 4. Again, make sure to save your work.

16. With your modified `LPF for CF` VI, run your `Guitar String` virtual instrument with α = 0.99, N = 58, and P = 13250 as before. Can you hear the difference?

3.6 Marching To A Different Drummer

Now that we have simulated the sound of a guitar string being plucked, we can use the same system, with a slight modification, to create the sounds of two kinds of drums[2].

1. Make a copy of the `Guitar String` VI and name it `Drum`, and save it in another folder also called `Drum`. Also, save a copy of the `LPF for CF` VI and the `Random Init` VI in the `Drum` folder.

2. Modify the block diagram of the `Drum` VI to implement the following LCCDE:

$$\forall n \in \mathbb{Z}, \quad y(n) = \begin{cases} x(n) + \frac{\alpha}{2}\left(y(n-N) + y(n-N-1)\right) & \text{with 50\% chance.} \\ x(n) - \frac{\alpha}{2}\left(y(n-N) + y(n-N-1)\right) & \text{with 50\% chance.} \end{cases}$$

   As a hint, how is the LCCDE you are given here different from the LCCDE you derived in step 1 of section 3.3? You may consider creating another sub-VI, which has no input terminals, but produces an output that is 1 with a fifty percent chance and -1 with the other fifty percent. If you do create such a sub-VI, do not forget to add it to the `Drum` folder.

3. Run your virtual instrument with $\alpha = 0.99$, $N = 200$, and $P = 13250$. You should hear the simulated sound of a snare drum. For comparison, there are two sound samples of a snare drum, one when the sound is unmuffled and one when the drum is played on its rim, available on bSpace as part of the resources for this lab.

4. Change N to 20 and run the virtual instrument again. Now, the sound is more similar to that of a tom-tom being brushed.

   Notice that, unlike the guitar sound, changing N will not change the frequency of the sound, but simply its duration. This is because the randomness we introduced destroys the fundamental frequency of the comb filter.

5. Convert the `Guitar String` folder and the `Drum` folder each into LLB files for checkoff. You may find the LLB conversion instructions in the lab 06 postlab useful.

4 Acknowledgments

Special thanks go out to the teaching assistants (TAs) of the Spring 2009 semester (Vinay Raj Hampapur, Miklos Christine, Sarah Wodin-Schwartz), of the Fall 2009 semester (David Carlton, Judy Hoffman, Mark Landry, Feng Pan, Changho Suh), and of the Spring 2010 semester (Xuan Fan, Brian Lambson, Kelvin So) for providing suggestions, ideas, and fixes to this lab guide. This lab guide was based on, although substantially modified from, the "Plucked string instrument" laboratory exercise as presented in the book `Structure and Interpretation of Signals and Systems`, written by Edward A. Lee and Pravin Varaiya (ISBN 0201745518). The drum sound samples were obtained from the `Drums` entry on Wikimedia Commons, under the Creative Commons license.

References

[1] K. Karplus and A. Strong. Digital Synthesis of Plucked-String and Drum Timbres. *Computer Music Journal*, 7(2):4355, Summer 1983.

[2] Yes, this model is *that* amazing.