ECE 514E – RADAR & SATELLITE ENGINEERING
AUDIO EFFECTS & REAL-TIME PROCESSING
Thursday, 27 October 2022

OBJECTIVE

To understand and implement a variety of standard audio effects including echo reverberation and filtering.

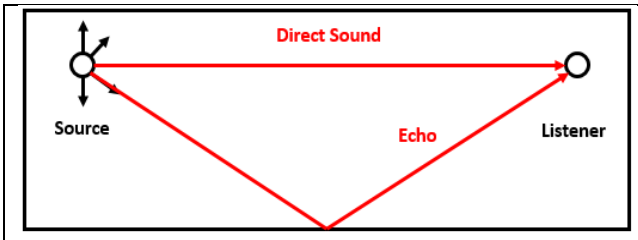
GENERAL THEORY

Audio effects processing is typically done using either time- or frequency-domain algorithms that process a stream of audio vectors. An echo effect, for example, can be easily implemented by creating a buffer of sample memory to delay a sound and play it back later, mixing it in with the original. Extremely short delays (of one or two samples) can be used to implement digital filters, which attenuate or boost different frequency ranges in the sound. Slightly longer delays create resonation points called comb filters that form an important building block in simulating the short echoes in room reverberation. A variable-delay comb filter creates the resonant swooshing effect called flanging. Longer delays are used to create a variety of echo, reverberation, and looping systems and can also be used to create pitch shifters (by varying the playback speed of a slightly delayed sound).

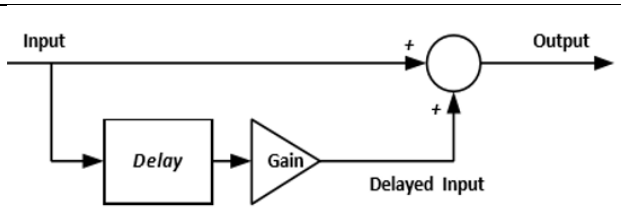
LAB-01 – CREATING A SIMPLE ECHO AUDIO EFFECT

Introduction

Echo arising from reflection from physical objects is audible because the speed of sound is relatively slow, about 343 meters per second. For example, a reflection from an object which is 30m away is delayed by 175ms which our ears can discern. The figure below shows the principle of echo generation and the corresponding model.



(a) Echo due to reflection of sound



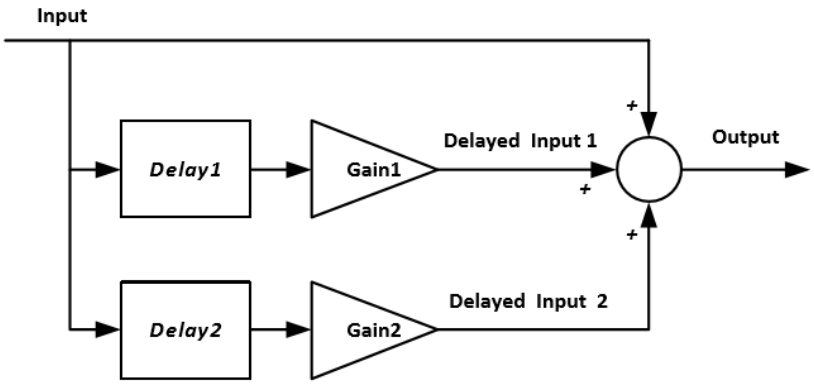
(b) A model of echo generation

If we consider only one echo path as shown above then an echo can be simulated using the following equation:

$$\text{Output} = \text{Input} + \text{Delayed Input} \times \text{Gain}$$

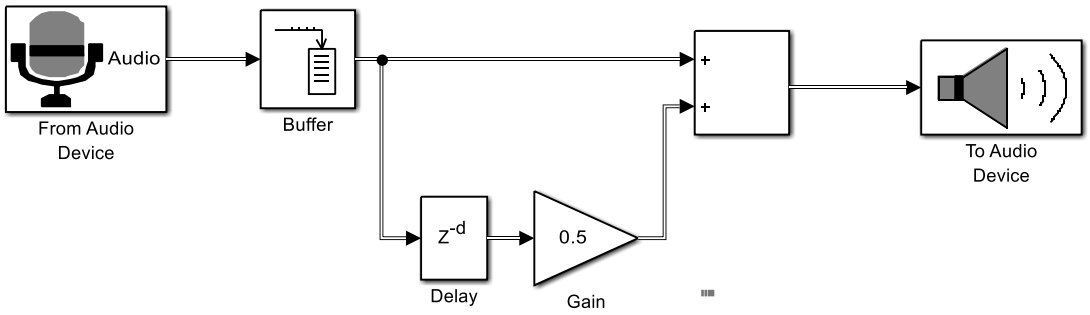
Where Gain < 1, due to losses in the echo path.

A real room will have multiple echo paths, as illustrated below. The key point to note is that the echo is derived solely from the input.



Procedure

1. Select **<Sum>** block in the section **<Simulink ><Commonly Used Blocks>.**
2. Select the **<Delay>** block in the section **<Simulink ><Commonly Used Blocks>**.
3. Select the **<Gain>** block in the section **<Simulink > <Commonly Used Blocks>**.
4. Link the blocks as shown in the figure below.

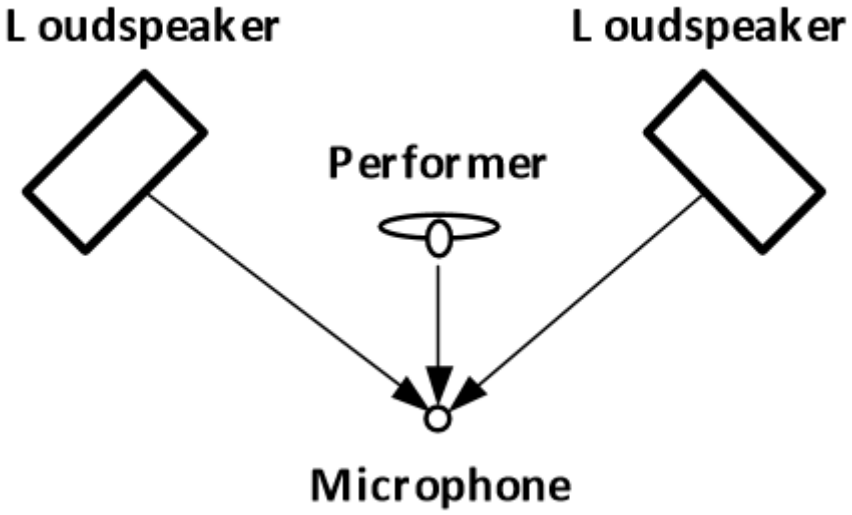


5. Double-click on the **<Gain>** block to change its parameter to **"0.5".**
6. Double-click on the **<Delay>** block to change **[Delay length] to "44100"** and **[Input processing] to "Columns as channels".**
7. Connect headphones to avoid feedback from the speaker to the microphone.
8. Run the simulation. If you talk into the input, you should hear the input signal combined with an echo delayed by 1 second.
9. For real-time audio and effects, it is desirable to have the minimal latency possible. In this simulation, you can adjust the latency by changing the **[Output buffer size]** of the **<Buffer>** block.
10. Save the model with the name **'echo_1'**
10. Determine the smallest buffer size that works on your system without buffer underflow (e.g., buffer underflow causes gaps in white noise at the signal output).

LAB 02 - REVERBERATION

2.0 Introduction

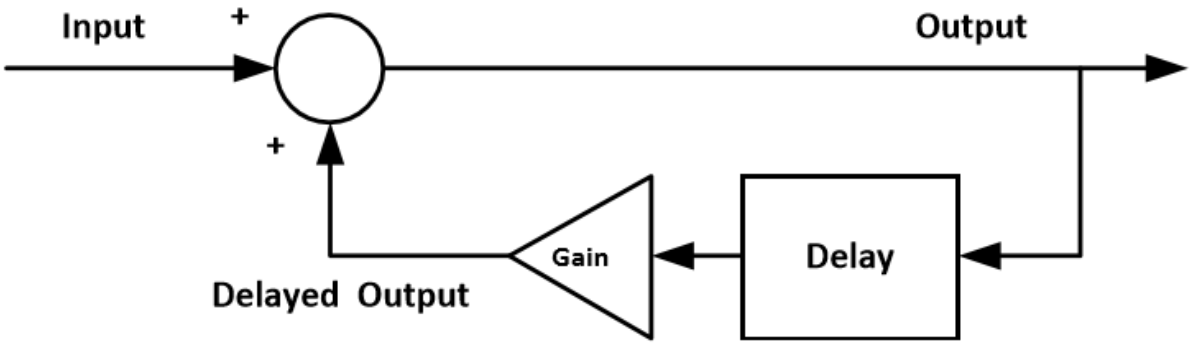
Reverberation is very similar to echo, except that the additional sound sources have been processed in some way, e.g. where the extra sound sources are amplified versions of the original sound source, as shown below.



In reverberation, the output is derived from both the input and the previous output, i.e

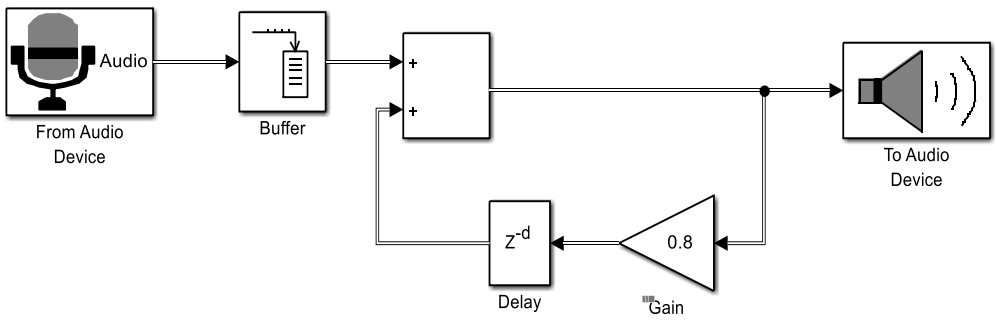
```
Output =  Input + Delayed Output × Gain
```

The simplified block diagram of reverberation is as shown below.



Procedure

1. Open the file '**echo_1'** and save it as **'reverb_1'**
2. Amend the model to be as shown below.
3. set the gain to be 0.8
4. Alter the gain in step as follow 0.8, 0.7, 0.6, …, 0.1
5. Alter the delay as follow 44000,22000, 11,000, 6,600, 3,300



6. Why does Reverberation require a shorter delay time than echo?
7. What effect does the "Gain" block have on the stability of the reverberation system?

LAB-03 – FILTERING NOISE IN AUDIO SIGNAL

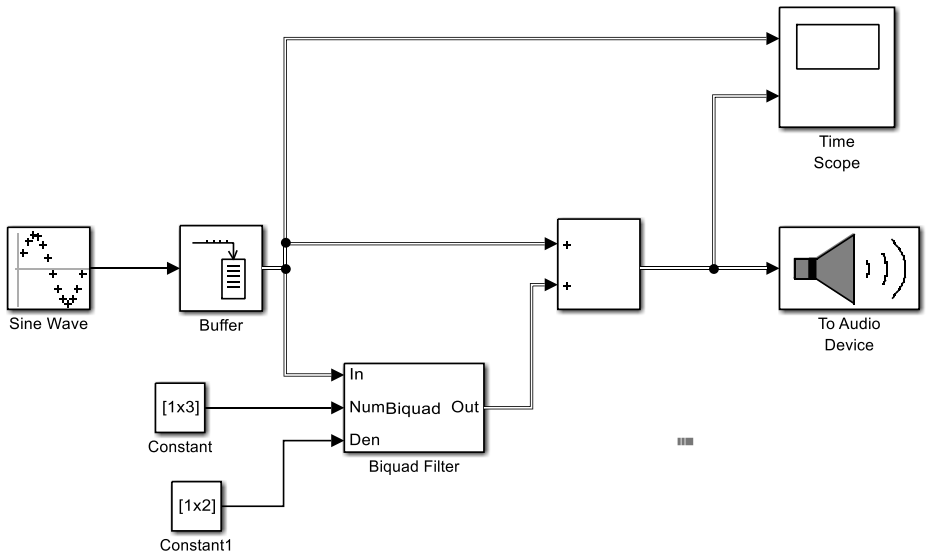
3.1 Introduction

This model implements a first-order IIR filter using the <Biquad> block with constant coefficients input through the <Constant> blocks. The <TimeScope> block can be double-clicked before running the model, so that its inputs are graphed in real-time and displayed after starting the simulation. In this case, you should see two sinewaves with different amplitudes. Remember that it is standard to use column vectors for vector-valued block inputs.

3.2 Procedure

1. Select **<Sine Wave>** block and double click change parameters as follows: **<Bias>** to **"0", <Samples per period>** to **"150",** and **<Sample time>** to **"1/44100".**
2. Select **<Biquad>** block in the section **<Simulink > <DSP System Toolbox HDL Support > <Filtering>**. Double-click on the "Biquad" block to change its parameters. Set the **<Coefficient source>** option to **"Input port(s)",** the **<Scale values mode>** option to **"Assume all are unity and optimize"** and the **<Input processing>** option to **"Columns as channels (frame based)".**
3. Select two **<Constant>** blocks from the section **<Simulink > <Sources>.** Double-click on one of the "Constant" blocks and set the parameter "Constant Value" to **[1 -1 0]** (note that it is a column vector). Similarly set the same parameter to **[-0.9 0]** for the other "Constant" block. These are the filter coefficients corresponding to B(z) and A(z) respectively, where the where the leading "1" has been omitted for the A(z) vector as required by the "Biquad" block.

4. Select **<Time Scope>** block from the section **<Simulink > <DSP System Toolbox HDL Support > <Sinks>**. Right-click on the block, **select <Signals & Ports > <Number of Input Ports>** and click on **"2".**
5. Now complete the connections between these blocks as shown in the following figure.



Results and Review

6. Given the above filter coefficients and system sampling frequency, which frequencies are attenuated by the filter and which ones are passed almost as is? Demonstrate and verify your calculations by suitably varying the sine wave's frequency and observing the output of the **<Time Scope>** block.
7. Save the model as LB-02 Include this model file as part of your submission.

LAB-04 – FILTERING NOISE IN A SIGNAL USING A NOTCH FILTER

4.1 Introduction

In this experiment, we shall modify the model developed in LAB-02 and implement a notch filter whose coefficients are controlled by the sine wave's frequency. To realize this, we will introduce a "MATLAB Function" block in the model.

4.2 Procedure

Start with the above model, remove all the links and the "Time Scope" block. Save this new model as LAB_03.
1. Change the values in the two "Constant" blocks to "300" and "20", and rename them as "Sine Frequency" and "Q" respectively.
2. Add the block **<From Multimedia File>** from library **<Simulink> <Audio System Toolbox > <Sources>**. Double-click on it and give the full path of the file "guitar1.wav". Make sure that the **<Samples per audio channel>** parameter is set to **"1024".**
3. Add a **<Sum>** block from library **<Simulink> <Math Operations>**
4. Add the <MATLAB Function> block from the library **<Simulink> <User-Defined Functions>** section. Double-click it and a function file should open in your MATLAB Editor. Rename the function as "iirnotchS" with inputs "f" and "Q", corresponding to the sine wave's frequency and the notch filter's "Q" factor respectively. Also rename the outputs as "nr" and "dr", corresponding to the numerator and denominator coefficients of the notch filter respectively.

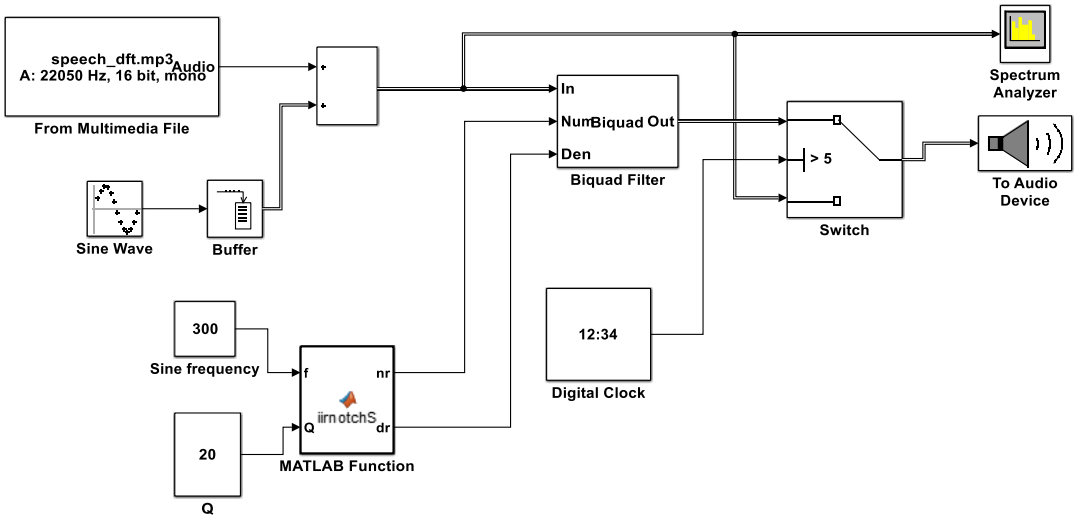
```
function [nr, dr] = iirnotchS(f,Q)
% IRR notch filter
lfo = f;
Fs = 44100; Wo = (2*lfo/Fs)*pi;
BW = (2*lfo/Fs/Q)*pi;
Ab = 3;
Gb = 10^(-Ab/20);
beta = (sqrt(1-Gb.^2)/Gb)*tan(BW/2);
gain = 1/(1+beta);
b = gain*[1 -2*cos(Wo) 1];
a = [1 -2*gain*cos(Wo) (2*gain-1)];
dr = a(2:end)';
nr = b';
end
```

5. Double-click on the "Sine Wave" block to change its parameters. Change "Samples per period" to "44100/300", and "Amplitude" to "0.5".
6. Add a "Digital Clock" block from the "Simulink -> Sources" section. During runtime, this block will output the current simulation time. Double-click it and set the "Sample Time" parameter to "1024/44100".
7. Add a **<Switch>** block, from the library **<Simulink><SignalRouting>,** between the **<Biquad>** block and the **<Audio Device Writer>** block, and connect the **<Digital Clock>** output to its second input. This will act as the "control" input

and whenever a specified condition is satisfied, the **<Switch>** block will output the first input; otherwise, it will output the third input.

8. Double-click the **<Switch>** block and set the **<Criteria for passing first input>** parameter to **"u2 > Threshold".** Also set the "Threshold" parameter to "3", i.e. 3 seconds into the simulation this block will start outputting the first input

7. Now connect all the blocks until the **<Biquad>** block as shown in the figure below.



8. In the **"Spectrum Analyzer"** window(double-lick the block if the window doesn't appear automatically during run-time), select **<Tools> Measurements -> Cursor Measurements"** or click on the shortcut in the toolbar.

9. Adjust the measurement bars to measure the frequency of the two peaks of the yellow wave, which corresponds to the input to the "In" port of the **<Biquad>** block. The frequencies must be approximately 300 Hz in magnitude, exactly corresponding to the sinusoidal tone added to the guitar audio signal. This window should look something like the one shown below the model diagram.