Automotive Product Development A Systems Engineering Implementation 1st edition Bhise Solutions Manual

SKU: 9781498706810-SOLUTIONS

Chapter 2 Problems

1. Suppose we want to delay a signal by 2.5 samples. Briefly explain two different methods of calculating the new signal and their relative advantages and disadvantages.

2. Consider a signal $x[0]=0.8$, $x[1]=0.4$, $x[2]=0.1$, $x[3]=-0.15$, $x[4]=-0.4$. Use zero, first, and second order interpolation to estimate the value $x[1.7]$.

3. a) Draw a block diagram for delay with feedback. Label the input $x[n]$, output $y[n]$ and any other commonly used parameters.

b) Under what conditions is the system stable? Why?

4. A delay without feedback produces notches at 300 Hz and 900Hz. List at least three other frequencies where notches will also be present, and explain why. For a sample rate of 48 kHz, how many samples of delay could produce this comb filter?

5. There are two microphones on a guitar, one close microphone placed only 5 cm away, picking up the direct sound, another placed 1.5 meters away from the guitar, picking up the room sound. How much delay should be added to the close microphone to align the two signals?

6. a) Draw a block diagram of a basic flanger. Now draw a block diagram of a flanger incorporating feedback or regeneration.

b) Derive the frequency response of a flanger without feedback.

c) Based on your block diagram in part a), write the difference equation(s) for a flanger **with** feedback, relating the output $y[n]$ to the input $x[n]$. Define any other variables you use in your equation(s).

d). Describe whether or not the flanger is: *linear*, *time-invariant*, and *stable*. Explain why, and whether the answer depends on parameter settings or whether feedback is used.

7. Define the following parameters for a flanger: *depth*, *delay*, *sweep width*, *sweep rate*. Describe the effect of varying their settings.

8. Why do we not hear an echo when flanging is applied?

9. How does chorus differ from flanger in terms of the LFO settings and use of feedback?

10. Suppose we implement the flanger with a circular buffer. We may hear artefacts as the delay length changes. Explain why this occurs and suggest a solution to overcome it.

11. a) Starting with the template below, draw a block diagram for a ping-pong delay with mono input and stereo output. Label the inputs $x_1[n]$ and $x_2[n]$, the outputs $y_1[n]$ and $y_2[n]$ and any other commonly used parameters.

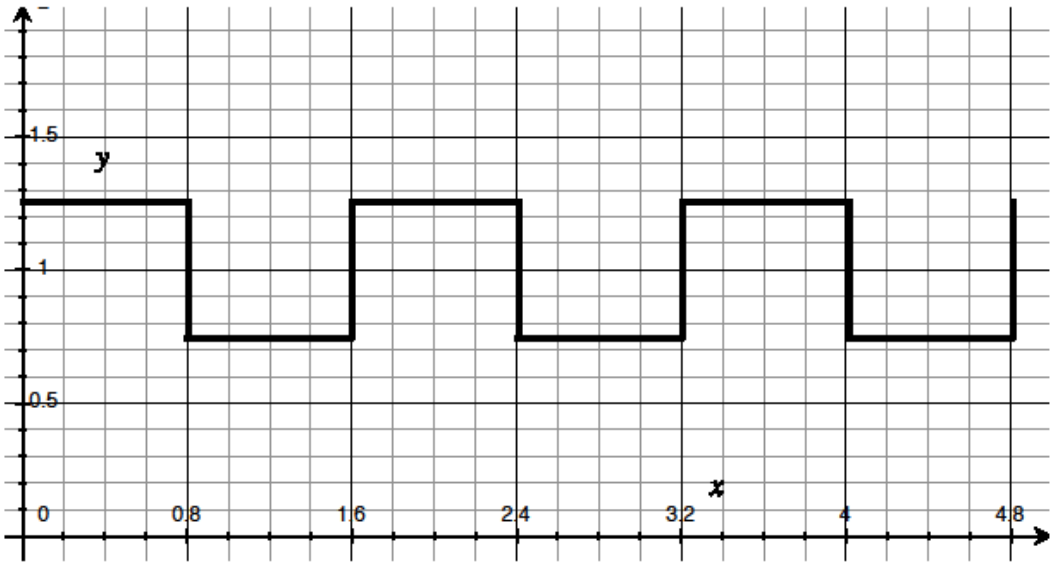
b) Give the transfer function and difference equation for ping-pong delay. That is, find $Y_1(z)$ and $Y_2(z)$ in terms of $X_1(z)$ and $X_2(z)$.

c) Suppose the ping pong delay was extended to four channel-output, with the input sound bounced sequentially around them. Draw a block diagram for this arrangement.

d) With delays of length D, how large does the total buffer need to be (in samples) for the two and four-channel ping-pong delays?

12. a) For a vibrato effect, suppose that the delay time is given by $d(t)=M+W\sin(2\pi ft)$, where M is the average delay, W is the sweep width and f is the LFO frequency. If M=5 ms and f=6 Hz, find the value of W needed to give a maximum pitch shift of 1.03 (roughly half a semitone). You can leave the result as a fraction.

b) The plot below shows the relative pitch at the output of a vibrato unit. What LFO waveform was used to produce this result, and why?



Chapter 2 Solutions

1. Method 1: rounding to nearest sample (no interpolation). Take the value at 2 samples instead. Not recommended as the quality is very poor, but it is the fastest of all methods.

Method 2: linear interpolation. The value at 2.5 samples is the weighted average of samples 2 and 3 (in this case, an even weight between them). Advantage: fast, simple and reasonable quality. Disadvantage: better quality can be had with cubic interpolation.

Method 3: cubic interpolation. The value at 2.5 samples is calculated based on a cubic spline through samples 1, 2, 3 and 4. Advantage: higher quality than linear interpolation results in fewer artefacts. Disadvantage: more computationally expensive.

Method 4: discrete-to-continuous, continuous-to-discrete conversion. Equivalently: sinc interpolation. Literally convert the discrete-time signal to continuous time, delay is by half a sampling period, then resample it. Advantage: mathematically exact implementation, highest possible quality. Disadvantage: sinc interpolation requires perfect knowledge of the complete signal from N = -∞ to ∞, which is not possible in real-time situations.

Any two of these can constitute a complete answer, with Methods 2 and 3 the expected choices.

2. Zero order: Use nearest value, $x[1.7]= x[2] =0.1$

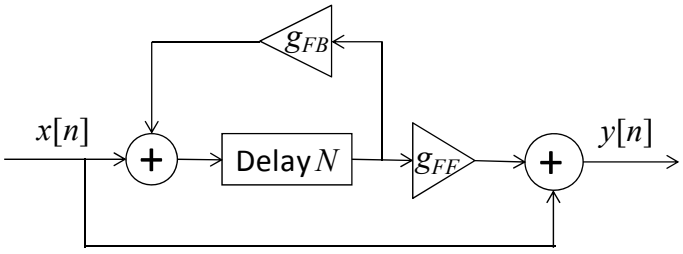
First order: Use linear interpolation, $x[1.7]= 0.3x[1] + 0.7\, x[2]= 0.12+0.07 = 0.19$.

Second order: Nearest step is n=2. Three nearest samples are, $x[1]$=, $x[2]$, and $x[3]$. So,

$$x(t) = \frac{(t-n-1)(t-n)x[n-1] - 2(t-n-1)(t-n+1)x[n] + (t-n)(t-n+1)x[n+1]}{2}$$

$$x[1.7] = \frac{.39x[1] + 1.82x[2] + .21x[3]}{2} = \frac{.39*.4 + 1.82*.1 - .21*.15}{2} = 0.15325 \quad . \qquad (0.1)$$

3.



The system is stable if and only if the magnitude of the feedback gain $|g_{FB}|$ is strictly less than 1. If $|g_{FB}|\geq 1$, the signal around the feedback loop can grow without bound, failing the bounded-input, bounded-output test for stability.

$y[n]=x[n]+x[n-d]$

4. $H(z)=\frac{z^d+1}{z}$

$zeros: z^d=e^{j\omega d}=-1\rightarrow \omega d=\pi,3\pi,5\pi...$

So the notches occur at odd harmonics. We would expect more notches at 1500 Hz, 2100 Hz…

To find the delay associated with these notches,

$\omega d=\pi, \omega=2\pi f/f_s \rightarrow$

$\pi/d=2\pi f/f_s$

$d=f_s/(2f)=48000/600=80$

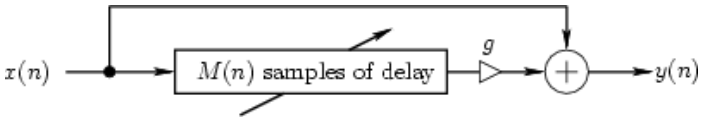
5. Consider sound at time *t*. The close microphone receives it at *t*+0.05/*c*.The room microphone receives it at *t*+15/c. So, to align the two signals, we need to add a delay of (1.5-0.05)/*c* seconds to the close microphone signal.

Based on the speed of sound in air, the delay would be (1.5-0.05)/c = 4.26ms, assuming c = 340.3 m/s.

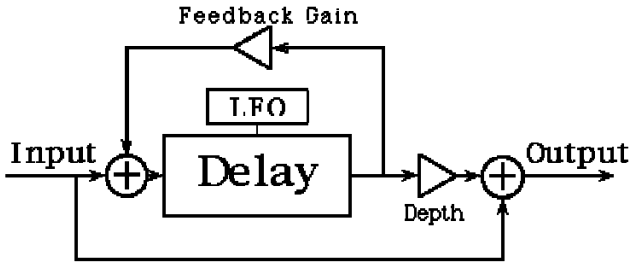
If the sample rate is f_s, then this is 1.45 f_s/c samples. So, for sample rate 44.1kHz, this is approx. 188 samples.

6. a)

Without feedback:



With feedback:



b)

$y(t)=x(t)+x(t-Dt)$

$Y(s)=X(s)+e^{jwDt}X(s)$

$H(s)=1+e^{jwDt}$

$H(s)=e^{jwDt/2}(e^{-jwDt/2}+e^{jwDt/2})$

$|H(s)|=|2\cos(wDt/2)|$

c) Let D be the delay in samples, f the feedback gain and a the depth. We can write an intermediate variable d[n] as the output of the delay unit. Then we have:

d[n] = x[n-D] + fd[n-D]

y[n] = x[n] + ad[n]

This is a sufficient answer, but we can also substitute and reduce:

$d[n] = (1/a)(x[n] - y[n]) = x[n-D] + (f/a)(x[n-D] - y[n-D])$

$(1/a)y[n] = (1/a)x[n] - x[n-D] - (f/a)x[n-D] + (f/a)y[n-D]$

$y[n] = x[n] - (a + f)x[n-D] + fy[n-D]$

d) The flanger is linear, whether or not feedback is used and regardless of parameter settings. The sum of two input signals will produce the sum of flanging each one separately. (The flanger contains only additions, multiplications and delays, all linear operations.)

The flanger is time-variant in the general case: a low-frequency oscillator is typically used to control the delay value, which means that an input arriving at a different time will produce a different output depending on the current phase of the LFO. The only case in which the flanger is time-invariant is if the delay is unchanging (speed set to 0), though in this case the flanger reduces to a static comb filter and is not musically useful.

The flanger is stable in all cases except where feedback is used with a feedback gain greater than or equal to 1. In this case the signal around the feedback loop can grow without bound, failing the bounded-input, bounded-output test for stability.

7. *Depth* or *Mix* is the amount of delayed signal mixed with original signal. It controls how deep the notches will go. The larger the depth, the more pronounced the notches in the flanger. If depth is set to zero, the frequency response is flat, but when the depth is higher, the notches begin to appear and extend downward, and reaching zero when the depth is one.

Delay specifies the minimum delay used on the copying of the original signal. In the other words, the delay parameter determines how high the first notch will go. As the delay is increased, the first notch drops down.

Sweep Width decides how wide the sweep is in terms of the delay time (the width of the low frequency oscillator). The sweep width is the maximum additional delay that is added to the signal in addition to the delay in the delay parameter and it determines how low the first notch in the frequency response will reach. A low width keeps the variance in the delay time small, whereas the high width causes the notches in the frequency response to sweep over a larger area. The maximum delay is the sum of the delay and sweep width parameters.

Sweep Rate is the frequency of the low-frequency oscillator, and controls how many times per second the delay sweeps between its minimum and maximum settings. Higher sweep rate will produce a more quickly-changing flanger effect, and higher sweep rates for the same sweep width will also introduce more pitch modulation. (Answers do not need to mention pitch modulation to be correct.)

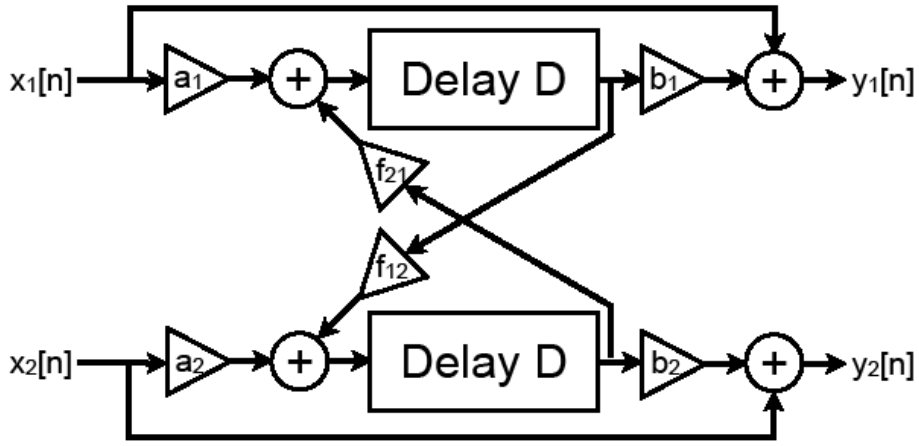
8. Usually, the delay times for a flanger ranges from 1 to 10 miliseconds, however, we perceive an echo if the delay is more than 50-70 milliseconds. Therefore, we don’t hear an echo because the delay is too short. For larger values of delay time, t, we can hear an echo that is distinct from the direct signal. On the other hand, for smaller value of t, our ear can no longer segregate the time events but can notice the spectral effect of the comb filter. Therefore, we need to set delay t as small as possible to create flanger.

9. The chorus is essentially the same algorithm as a flanger, but the flanger is characterized by its smaller delay lengths (from about 1 ms. to 10 ms.) and an optional feedback path that routes the delayline output back into the input.

10. Expected answer: The read and write pointers in the circular buffer are moving at different speeds as the delay length changes. To produce a smooth, continuous change in delay, the read pointer will often point to a non-integer sample number within the circular buffer. Since non-integer samples don't exist, we might round the read pointer to the nearest sample, but this will cause the read pointer to move in erratic hops rather than a smooth line. The result will be noise and artefacts. To overcome this problem, an interpolated delay line is needed which approximates the value for non-integer samples using weighted combinations of the ones around it.

An alternative correct answer: if we are not careful with the location of the read and write pointers, it is possible for the pointers to cross since the read pointer moves at a different rate than the write pointer. In this case we will hear artefacts when samples are read from the buffer before they have been written. A solution is to make sure the total delay length never dips below 0 at any point in the LFO cycle.

11. a) Label names may vary; a sufficient solution may also leave out either the gains a_i or b_i, but not both, and not f_{ij}.



b)

$[a_1X_1(z)+b_1W_2(z)]\, z^{-N} = W_1(z)$

$[a_2X_2(z)+b_2W_1(z)]\, z^{-N} = W_2(z)$

So we can solve for W_1;

$a_1X_1(z)\, z^{-N} + b_1a_2X_2(z)\, z^{-2N} = [1 - b_1b_2\ z^{-2N}]W_1(z)$, and

$Y_1(z) = c_1W_1(z) + X_1(z)$

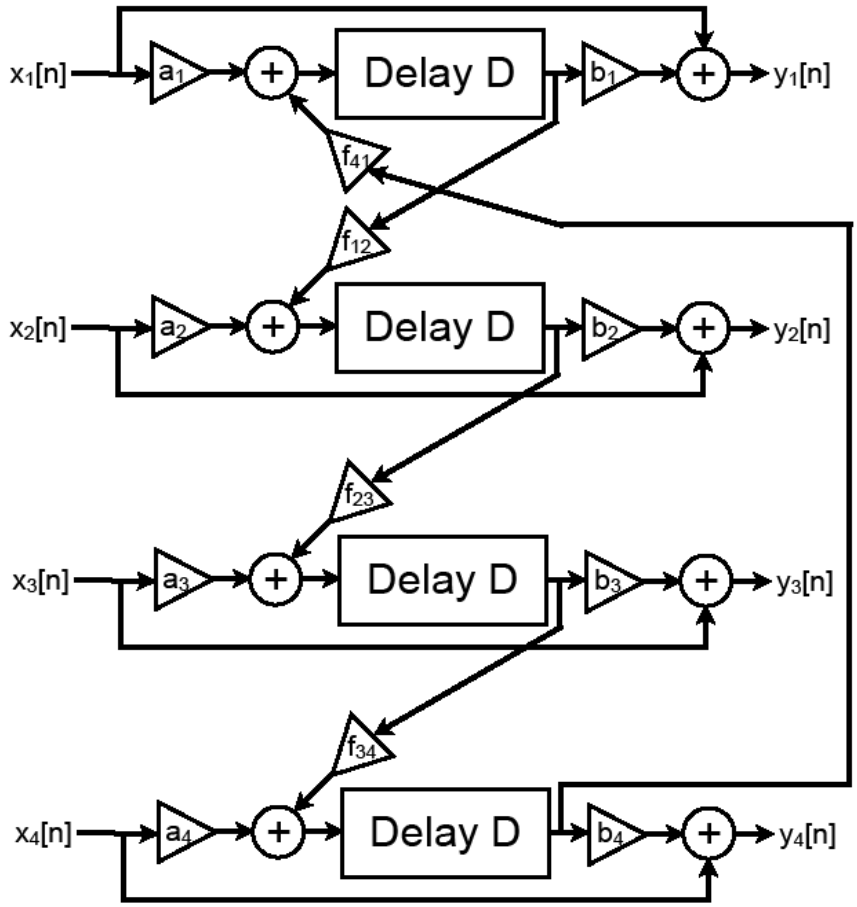
$= c_1\{ a_1X_1(z)\, z^{-N} + b_1a_2X_2(z)\, z^{-2N} \}/\{1 - b_1b_2\ z^{-2N} \} + X_1(z)$

and similarly,

$Y_2(z) = c_2W_2(z) + X_2(z)$

$= c_2\{ a_2X_2(z)\, z^{-N} + b_2a_1X_1(z)\, z^{-2N} \}/\{1 - b_2b_1\ z^{-2N} \} + X_2(z)$

c) The important part of the answer is the signal flow from one channel to the next.



d) The total buffer is the sum of the buffer for each channel: 2*D* for the two-channel ping-pong delay, 4*D* for the four-channel version.

12. a) The pitch shift is determined by the derivative of the delay, $f_{ratio}(t) = 1 - 2\pi fW\cos(2\pi ft)$. The cosine function has values between -1 and 1, so the maximum pitch shift will occur when its value is -1. Solving for f_{ratio}, $f_{ratio} = 1.03 = 1+2\pi 5W$. Thus $W = 0.03/(10\pi)$. This works out to about 0.95ms, but a numerical answer is not required.

b) A triangle wave was used. The pitch shift is dependent on the derivative of the delay, and a square wave is the derivative of a triangle wave since the triangle wave has only two slopes.

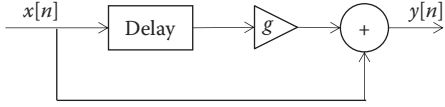


Figure 2.1
Diagram of the basic delay unit, or an echo device. The box marked "delay" is commonly known as a delay line.

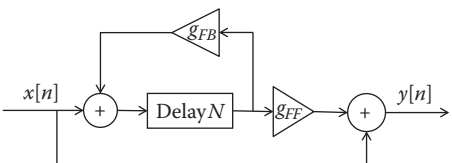


Figure 2.2
Diagram of the basic delay unit with feedback.

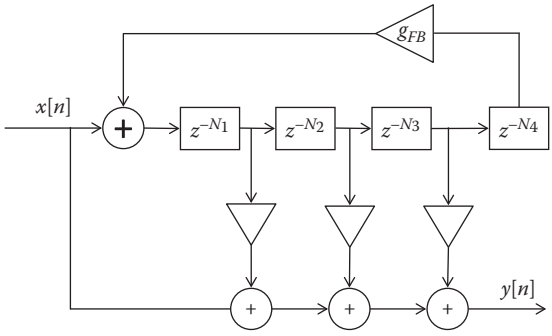


Figure 2.3
Flow diagram of a three-tap delay. If the last delay value is zero, and only the third tap is used, the system is equivalent to the basic delay.

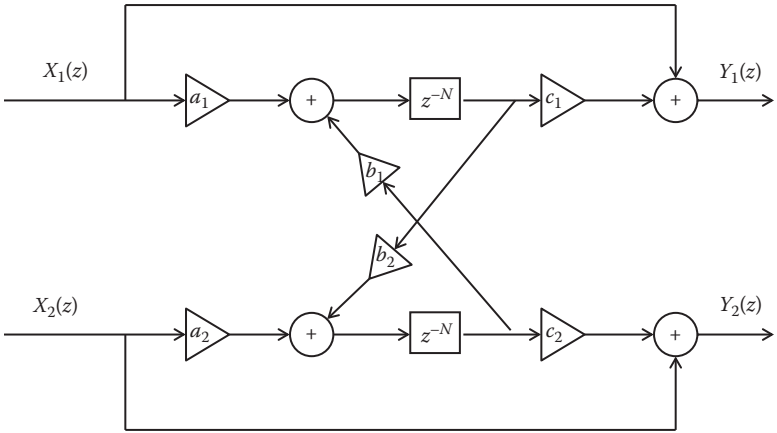


Figure 2.4
Flow diagram of a ping-pong delay unit.

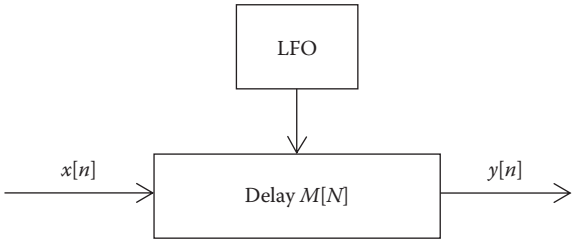


Figure 2.5
Modulated delay and pitch shift.

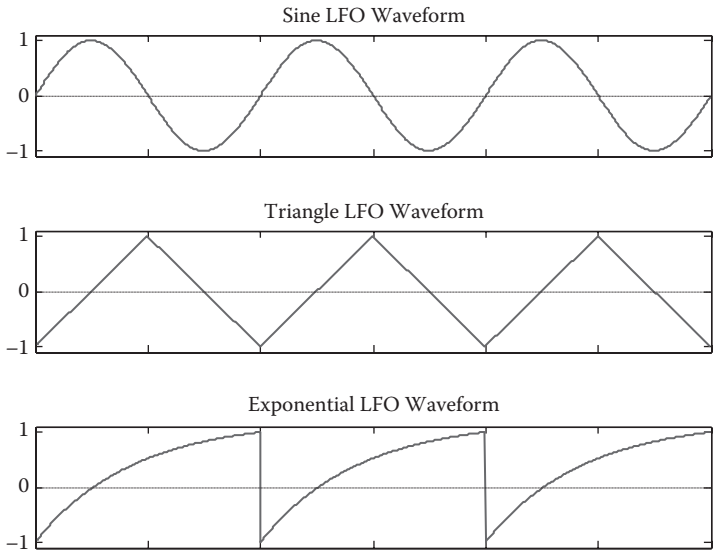


Figure 2.6
Three commonly used low-frequency oscillator (LFO) waveforms.

002x006.eps

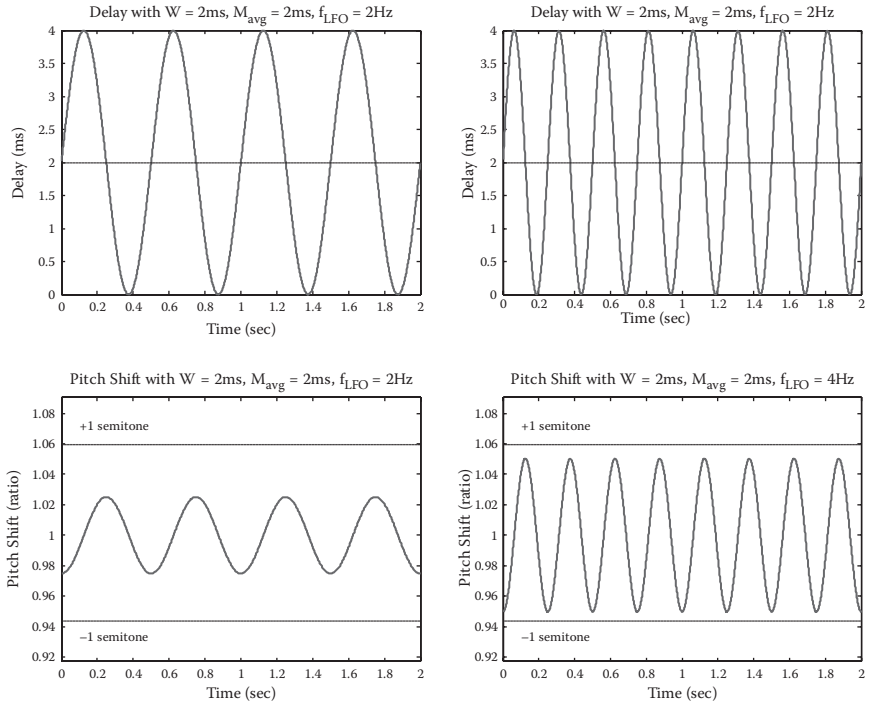


Figure 2.7
Vibrato in operation. The LFO waveform on top and the pitch shift on bottom, for LFO frequency 2 Hz (left) and 4 Hz (right).

002x007.eps

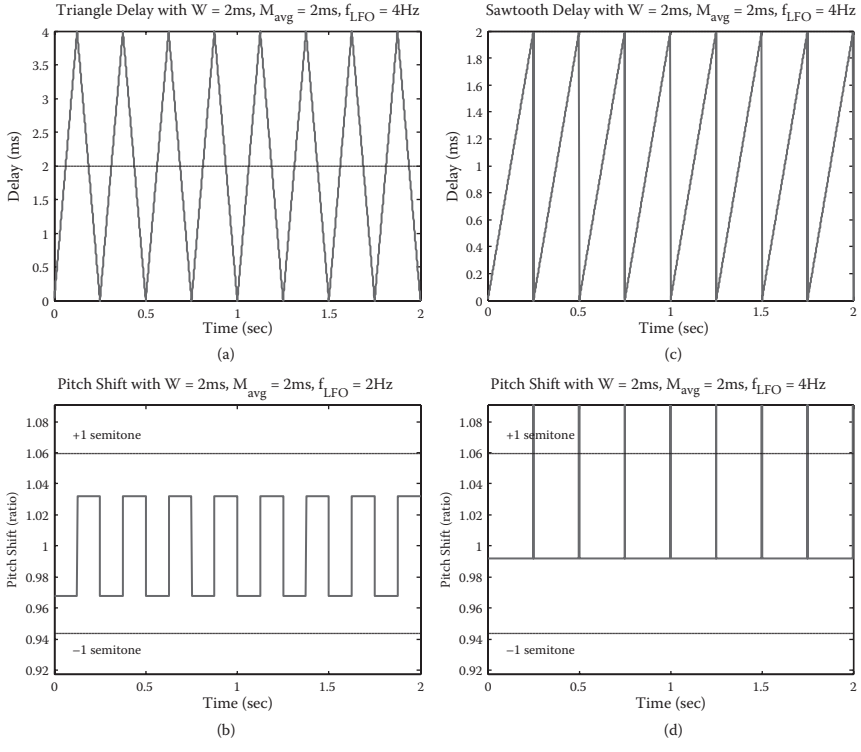


Figure 2.8
Triangular (a) and sawtooth LFOs (b), with corresponding pitch shift (c and d).

002x008.eps

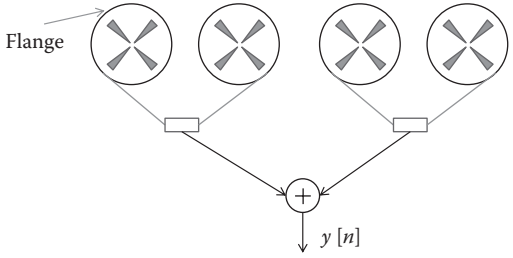


Figure 2.9
Two tape machines configured to produce a flanging effect.

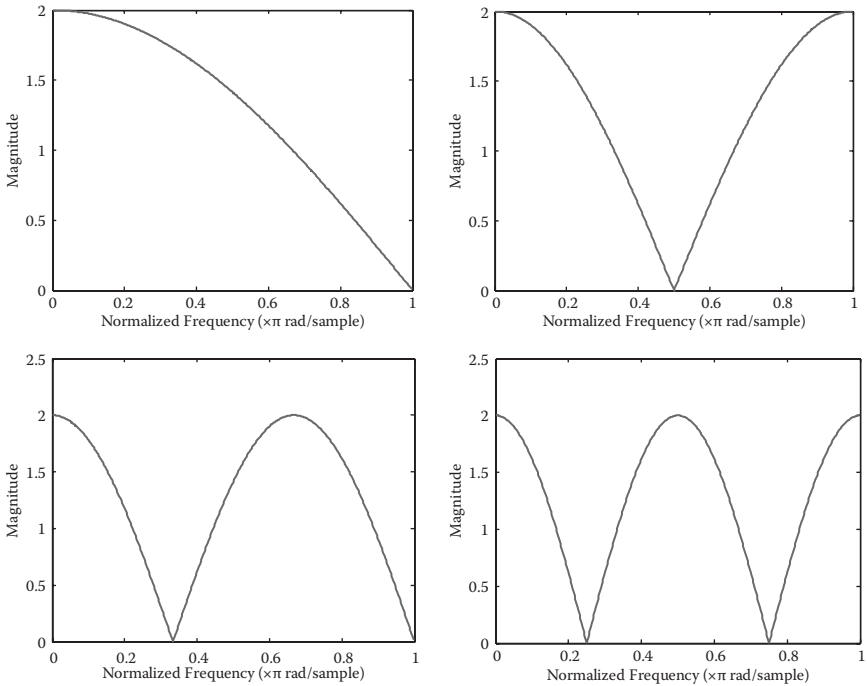


Figure 2.10
The magnitude response of a flanger with depth set to 1 and delay times set to one, two, three, and four samples.

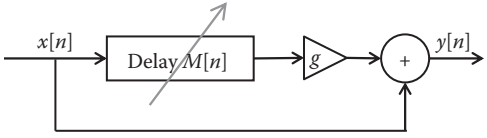


Figure 2.11
Block diagram of a basic flanger without feedback. The delay length $M[n]$ changes over time.

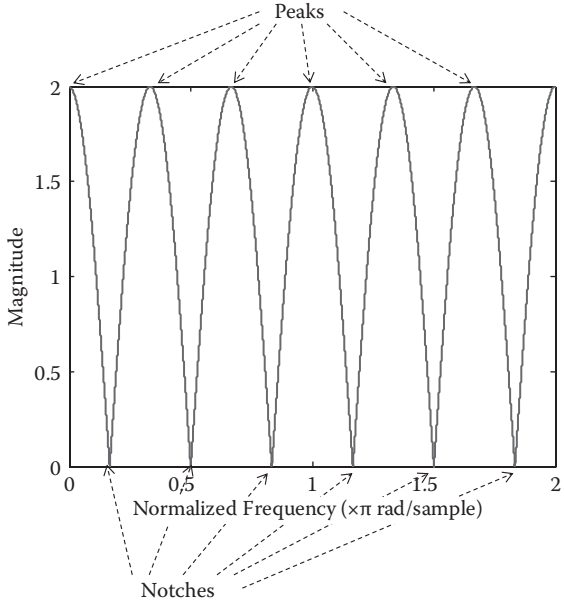


Figure 2.12
The frequency response of a simple flanger with a six-sample delay and depth set to 1. The locations of the six peaks and six notches over the whole frequency range from 0 to 2π are shown.

002x012.eps

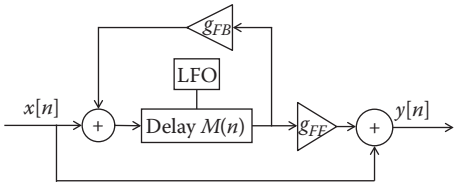


Figure 2.13
Flanger with feedback. Delay in all flangers is controlled by a low-frequency oscillator (LFO).

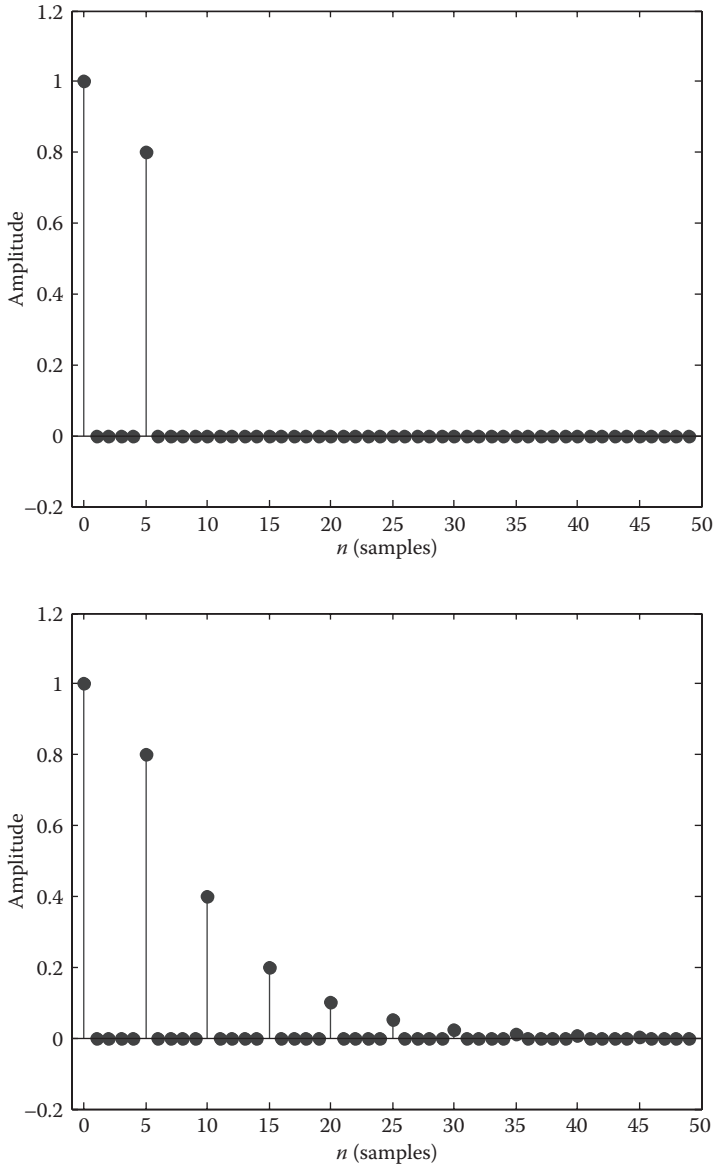


Figure 2.14
Impulse responses for a comb filter (i.e., delay effect) with a five-sample delay. On top, $g = 0.8$ and no feedback. On bottom, $g_{FF} = 0.8$ and $g_{FB} = 0.5$.

002x014.eps

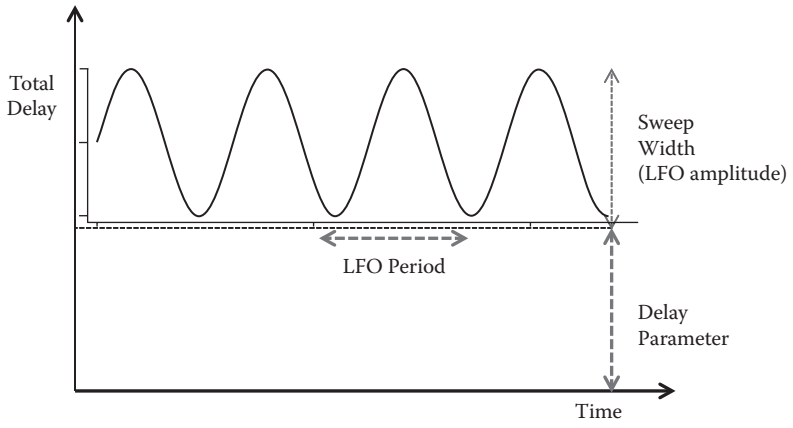


Figure 2.15
The maximum delay is the sum of the *sweep width* and *delay* parameters. The delay changes over time according to the *sweep rate*.

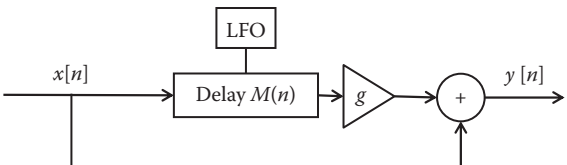


Figure 2.16
The flow diagram for the chorus effect including its LFO dependence. The delay changes with time.

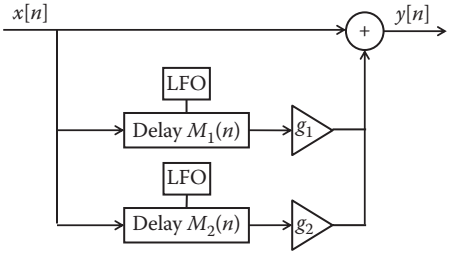


Figure 2.17
A multivoice chorus diagram.

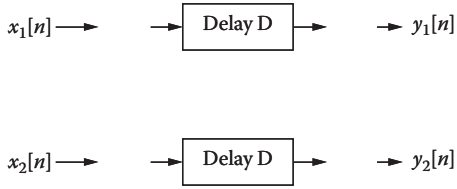


Figure 2.18
Template for a ping-pong delay.

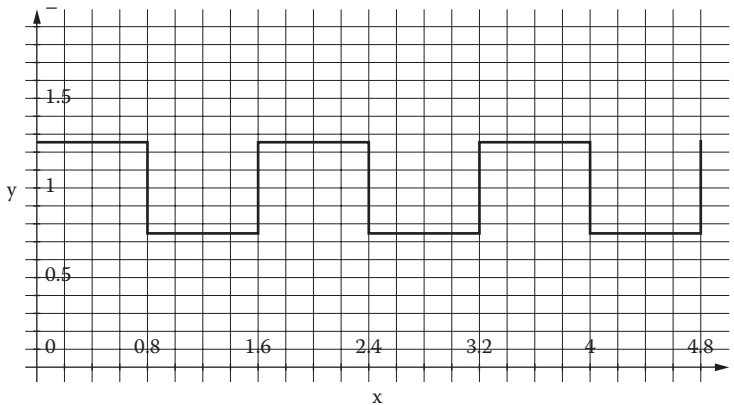


Figure 2.19
Relative pitch at the output of a vibrato unit.

Vibrato

Vibrato simulation

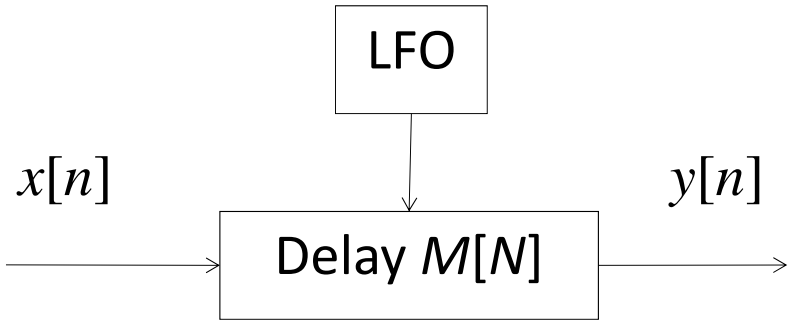
- Small, quasi-periodic variations in pitch of tone
  - Typical vibrato depth is on order of 1 percent
  - One semitone = $\sqrt[12]{2} \approx 5.9\%$ variation
- Often accompanied by tremolo
  - Amplitude modulation at same frequency as vibrato
- To apply vibrato
  - Quasi-periodic frequency shift
  - Use a modulated delay line
- Time-varying delay changes pitch of sound
  - Simulated Doppler shift

Vibrato in the acoustic world

- Violin
  - Vibrato produced by wiggling finger
  - Changes where the string is stopped on the fingerboard
  - Vibrato frequency can be very slow
    - circa 6 Hz for example
  - Frequency modulations of string vibrations produce amplitude modulations
    - Due to complex frequency response of violin body
- Singing voice
  - Vibrato produced by modulating tension of vocal folds

Digital implementation

- Variable delay alone, no mix
  - Delay modulation yields vibrato when the modulation is sinusoidal
  - Reduced case of the flanger
  - Variation controlled by low-frequency oscillator
  - Interpolated (fractional) delay line is required
- Typical values
  - M (average delay): 5-10ms
  - f = $\omega/2\pi$ (LFO frequency): 5-14Hz



Modulated delay and pitch shift

Determining vibrato width

Width (max frequency deviation) of vibrato depends on LFO frequency and sweep width:

- Consider average delay *M*, sweep width *W*, frequency *f*
- Total delay $d(t) = M + W\sin(2\pi ft)$
- Derivative of delay is difference in speed between read and write pointers:

$$\frac{\partial d}{\partial t} = 2\pi fW\cos(2\pi ft)$$

- Write pointer always moves at constant speed
  - 1 sample written per input sample
- Speed of the read pointer determines pitch shift:

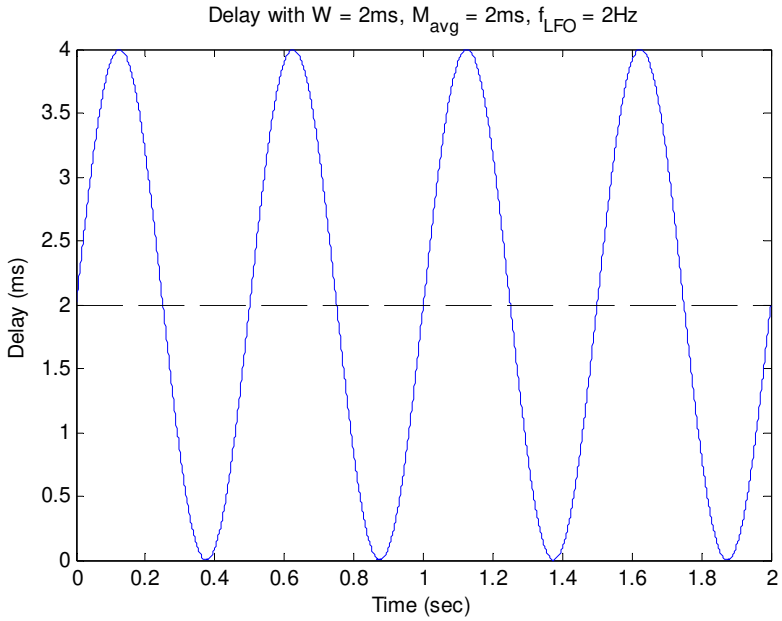
$$\frac{f_{out}}{f_{in}} = 1 - 2\pi fW\cos(2\pi ft)$$

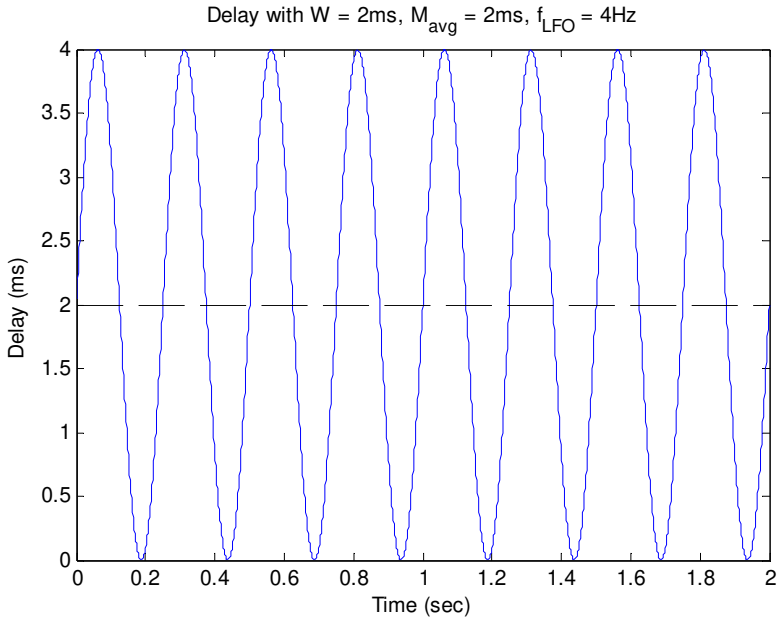
- Example for one semitone vibrato (1.059), f = 5Hz:

$$\left(\frac{f_{out}}{f_{in}}\right)_{max} = 1 + 2\pi fW = 1.059 \implies W = \frac{0.059}{2\pi f} = .00187s$$

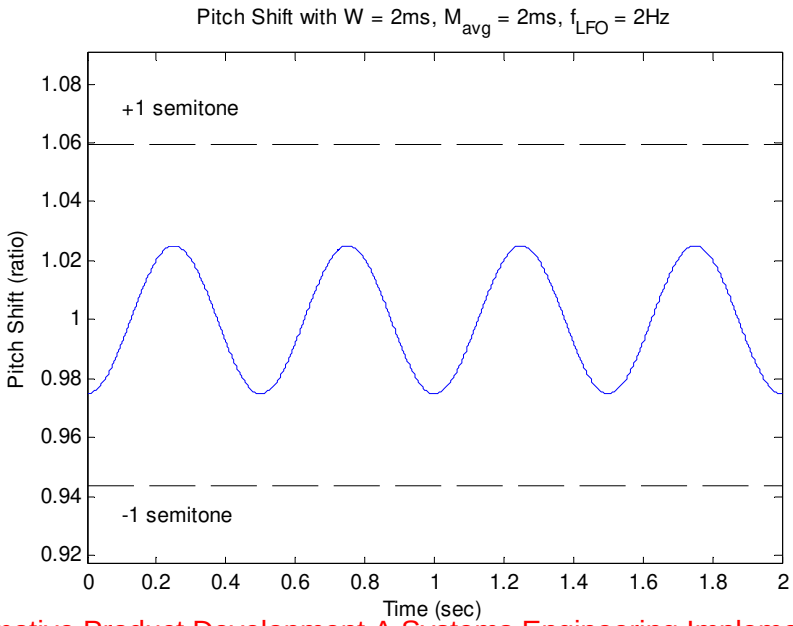
Vibrato in operation. LFO waveform on top and pitch shift on bottom, LFO frequency 2Hz (left) and 4Hz (right)

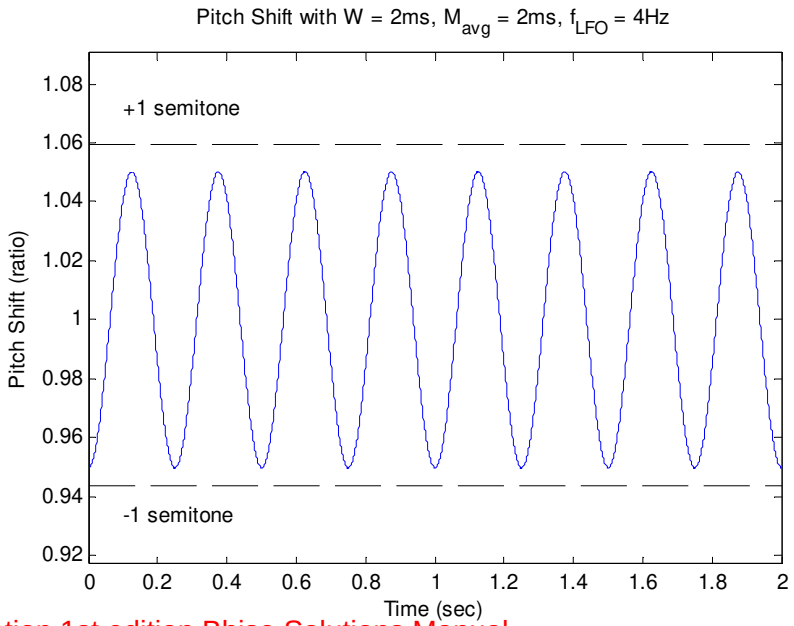
LFO waveform



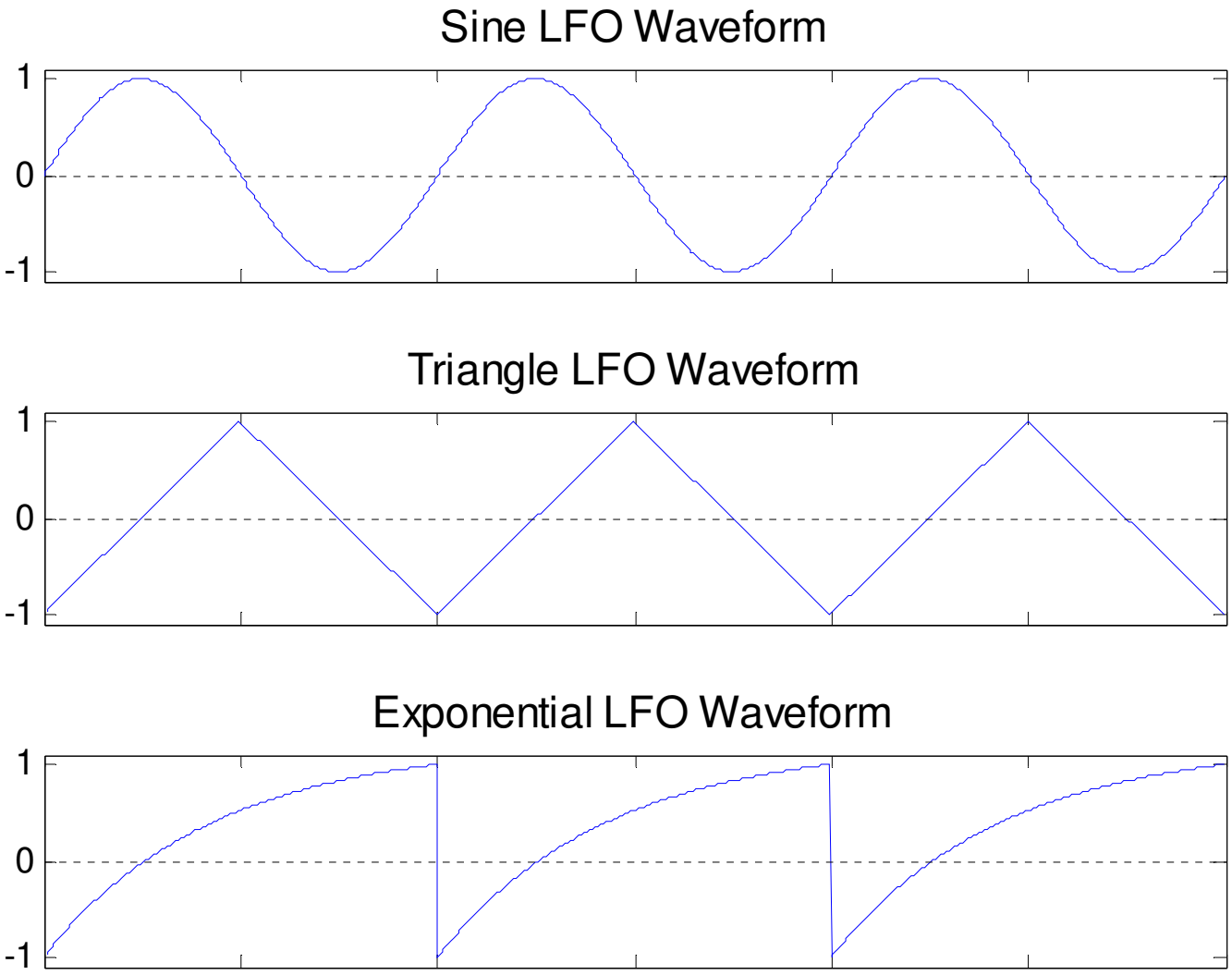


Pitch shift





Three commonly used low frequency oscillator (LFO) waveforms

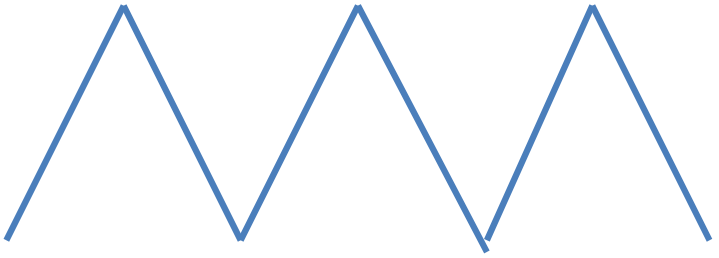


Other LFO waveforms

- Sinusoidal LFO modulation is the most common
  - Resembles vibrato in the physical/acoustic world
- What if we used a triangle wave LFO?

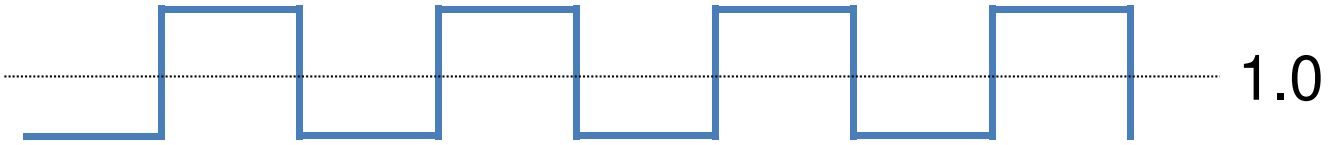
$$d(t) = M + W\,\text{triangle}(ft)$$

$$\text{triangle}(ft) = \begin{cases} 4t - 1 & : 0 \le t\%1 < 0.5 \\ -4t + 3 & : 0.5 \le t\%1 < 1 \end{cases}$$

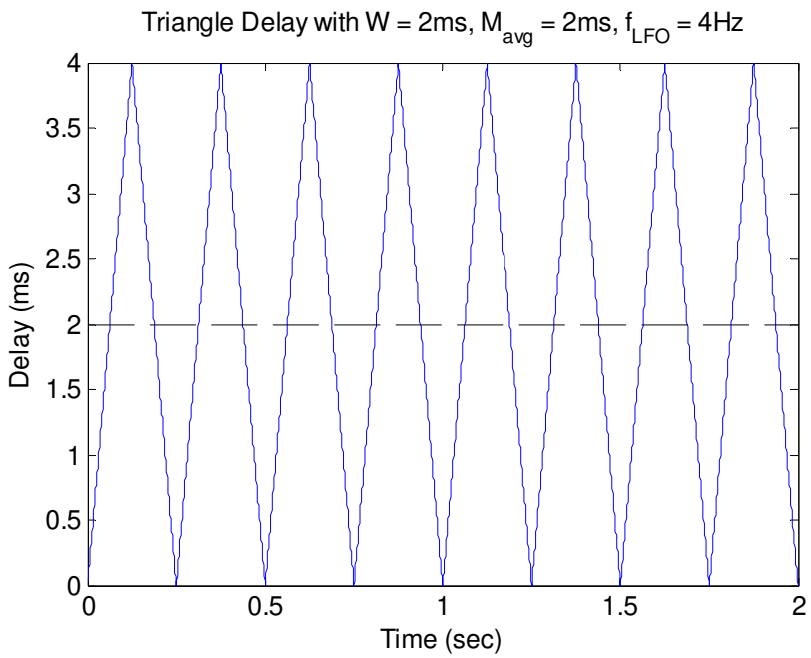


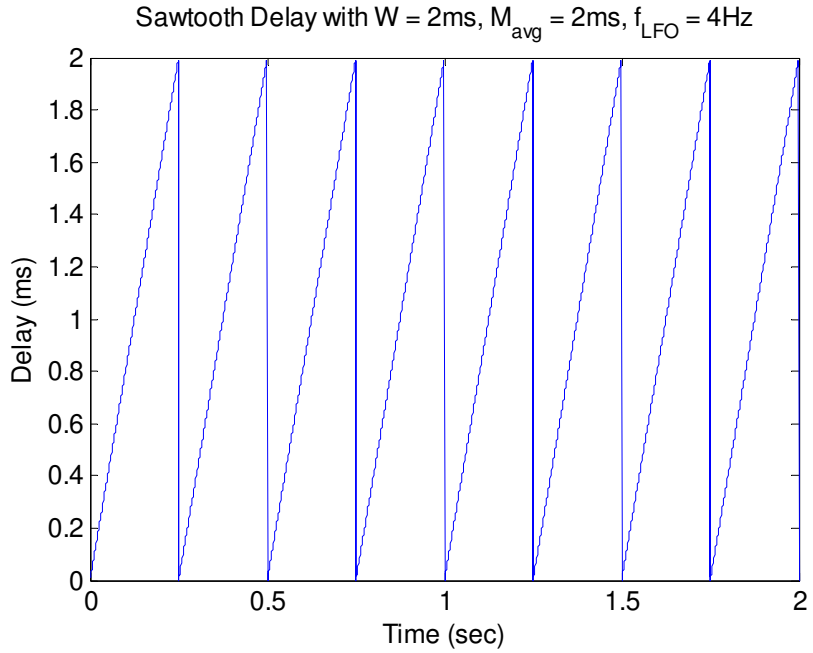
$$\frac{\partial d}{\partial t} = fW\frac{\partial \text{triangle}(ft)}{\partial t} = \begin{cases} 4fW & : 0 \le t\%1 < 0.5 \\ -4fW & : 0.5 \le t\%1 < 1 \end{cases}$$

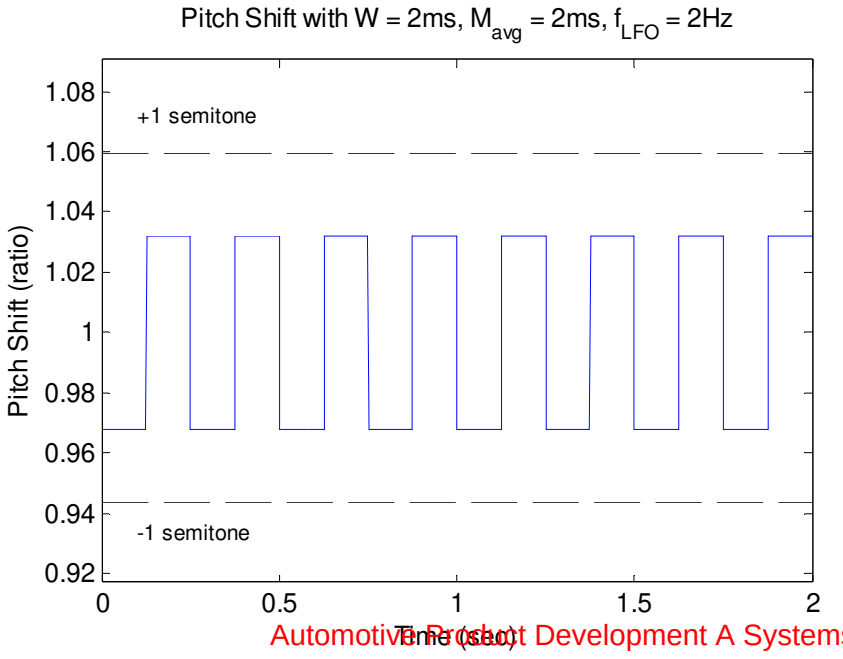
$$\frac{f_{out}}{f_{in}}(t) = \begin{cases} 1 - 4fW & : 0 \le t\%1 < 0.5 \\ 1 + 4fW & : 0.5 \le t\%1 < 1 \end{cases}$$

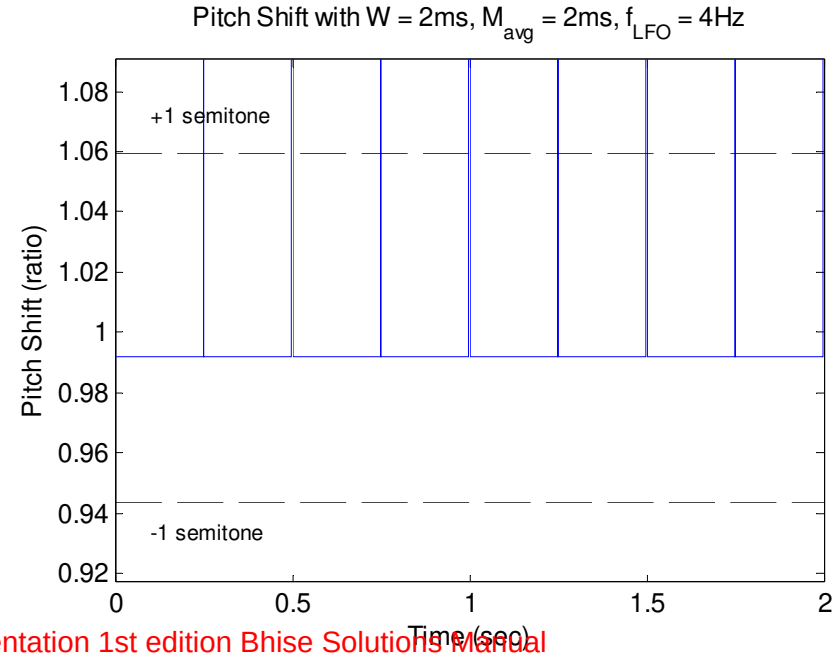


Triangular and sawtooth LFOs, with corresponding pitch shift









Vibrato C++ code(1/2)

```
// Variables used in this example whose values are set externally:
int numSamples; // Indicates how many audio samples to process
float *channelData; // Array of audio samples, length numSamples
float *delayData; // Our circular delay buffer of audio samples
int delayBufLength; // Length of our delay buffer in samples
int dpw; // Write pointer into the delay buffer
float ph; // Current phase of the LFO, always between 0-1
float inverseSampleRate; // 1/f_s, where f_s is sampling frequency

// User-adjustable effect parameters:
float frequency_; // Frequency of the low-frequency oscillator
float sweepWidth_; // Width of the LFO in samples
```

Vibrato C++ code(2/2)

```
for (int i = 0; i < numSamples; ++i)
{
   const float in = channelData[i];
   // Recalculate read pointer position with respect to write pointer.
   float currentDelay = sweepWidth_*(0.5 + 0.5*sinf(2 * M_PI * ph));
   //Subtract 3 samples to delay pointer, ensure we have enough written samples to interpolate
   float dpr = fmodf(dpw-currentDelay*getSampleRate())+delayBufLength - 3, delayBufLength);
   // Use linear interpolation to read fractional index into buffer. Find fraction by
   // which read pointer sits between two samples and use this to adjust sample weights
   float fraction = dpr - floorf(dpr);
   int previousSample = (int)floorf(dpr);
   int nextSample = (previousSample + 1) % delayBufLength;
   Float interpolatedSample= fraction*delayData[nextSample]+(1-fraction)*delayData[previousSample];
   // Store current information in delay buffer.
   delayData[dpw] = in;
   //Increment write pointer at constant rate. Read pointer moves at different rates depending
   //on settings of LFO, the delay and sweep width.
   if (++dpw >= delayBufLength)
   dpw = 0;
   // Store output sample in buffer, replacing input. Delayed sample is only
   //component of output (no mixing with dry signal)
   channelData[i] = interpolatedSample;
   // Update the LFO phase, keeping it in the range 0-1
   ph += frequency_*inverseSampleRate;
   if(ph >= 1.0) ph -= 1.0;
}
```

Chorus

The chorus effect

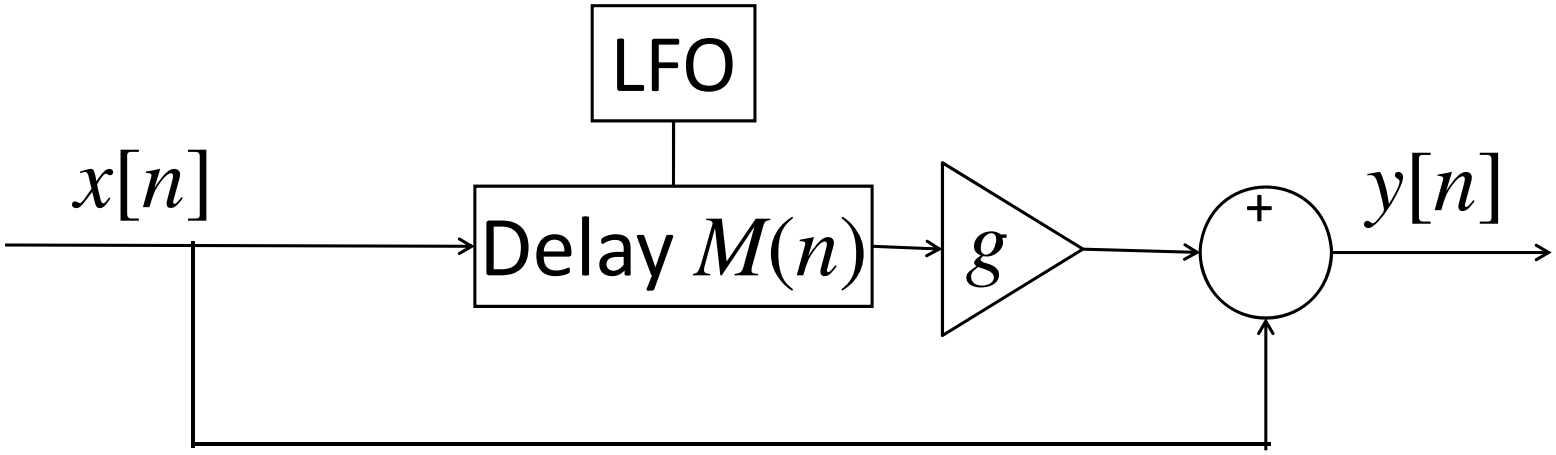
- Chorus = a group of singers
- Chorus effect
  - Make one instrument sound like many played together
  - Adds thickness to the sound
  - Often described as ‘lush’ or ‘rich’

How it works

- Two people playing instruments in unison:
  - Won't be precisely synchronised
    - Some delay between sounds they produce
  - Instrument pitches deviate, despite careful tuning
- Chorus effect reproduces these irregularities
  - Slight delay (implemented with delay line)
  - Detuning
    - Variable delay, like flanger and vibrato
- Picture delay as recording device.
- Store exact copy of input signal as it arrives, outputs later at same rate

Chorus and flanger

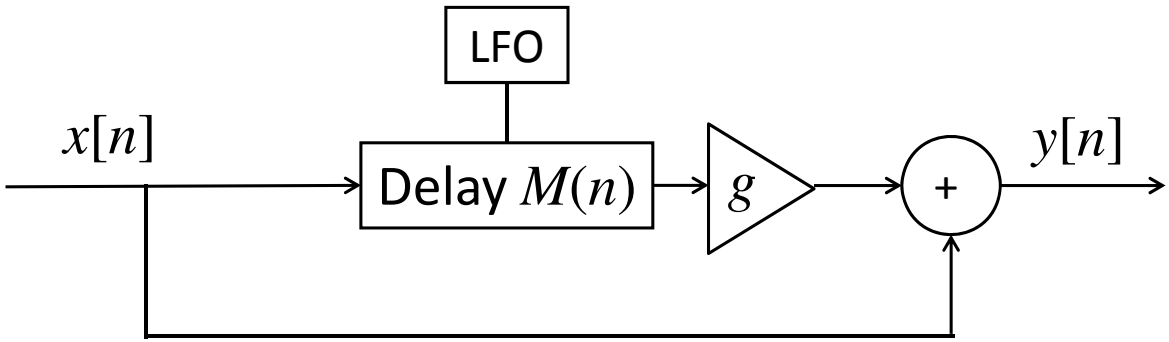
- Mix original signal with delayed and pitch modulated copy



- Similar to flanger, *except*
  - Delay times in chorus are larger than in flanger
    - Chorus delay between 20-30ms
    - Flanger delay usually 1-10ms
    - Long delay doesn't produce sweeping sound of flanger
  - For chorus, generally no feedback used

Chorus and LFO

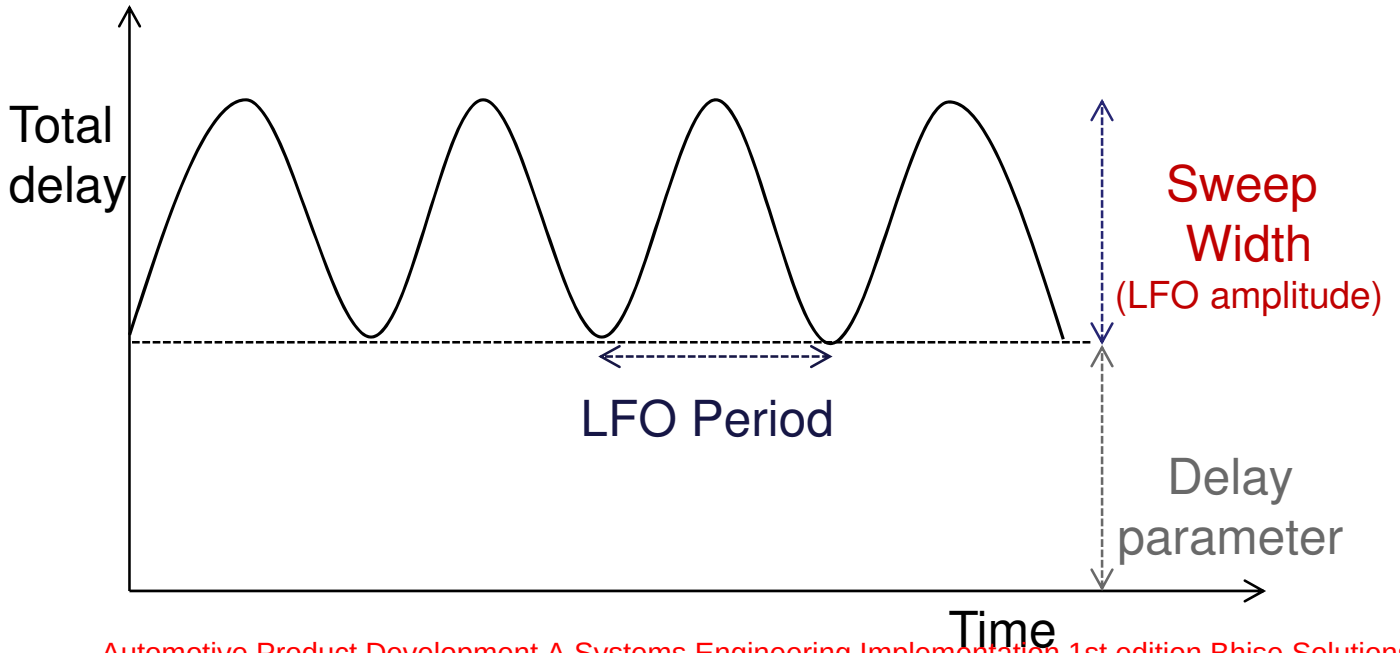
- Delay time changes with low frequency oscillator
  - Periodic waveform (e.g. sine, triangle)
  - Waveform changes slowly (3 Hz and below)
  - LFO controls:
    - Frequency
    - Amplitude
    - Shape (waveform)



- Can use randomly changing delay instead of LFO
  - Arguably, better models musicians playing in unison
- Multiple players in unison show loudness differences
  - Can vary amplitude of delayed signal
  - Amplitude parameter could be controlled by another LFO

Common parameters

- Delay
  - Controls minimum delay time that is used
  - For very small delay, chorus acts as flanger
  - Typical delay times between 20 and 30ms.
- Sweep width (or depth)
  - Controls how much total delay time changes over time
  - Expressed in milliseconds
  - Max delay: sum of sweep width and ‘delay’ setting
  - Think of sweep depth as amplitude of LFO



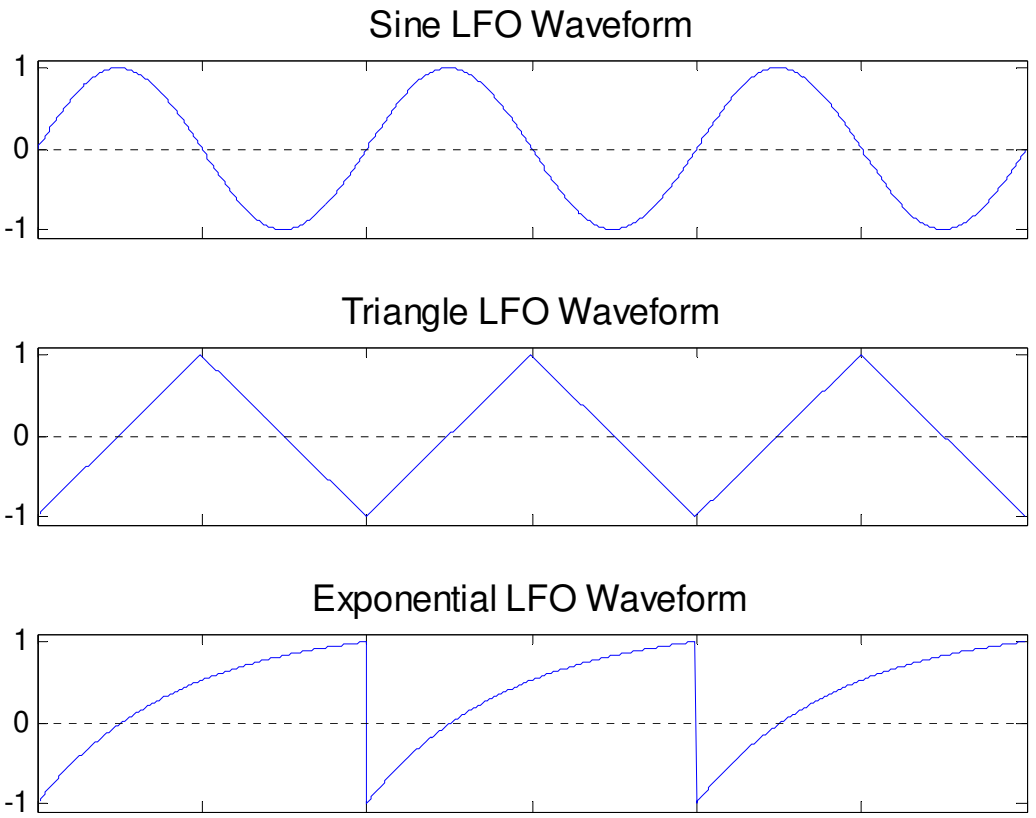
Common parameters

- Sweep width (continued)
  - Have to read faster/slower to cover total change in time
  - Increased sweep widths increase pitch modulation
  - Large sweep depths create a ‘warble’

- LFO (Low Frequency Oscillator) settings
  - Waveform describes how delay changes over time
  - Max of waveform = max delay time
  - Increasing LFO value = lower pitch (reading slower)
  - Pitch modulation related to speed of LFO (as before)
    - Steepest portions on waveform produce large pitch modulation
    - Increase speed→ compress waveform→ steeper→ pitch altered more

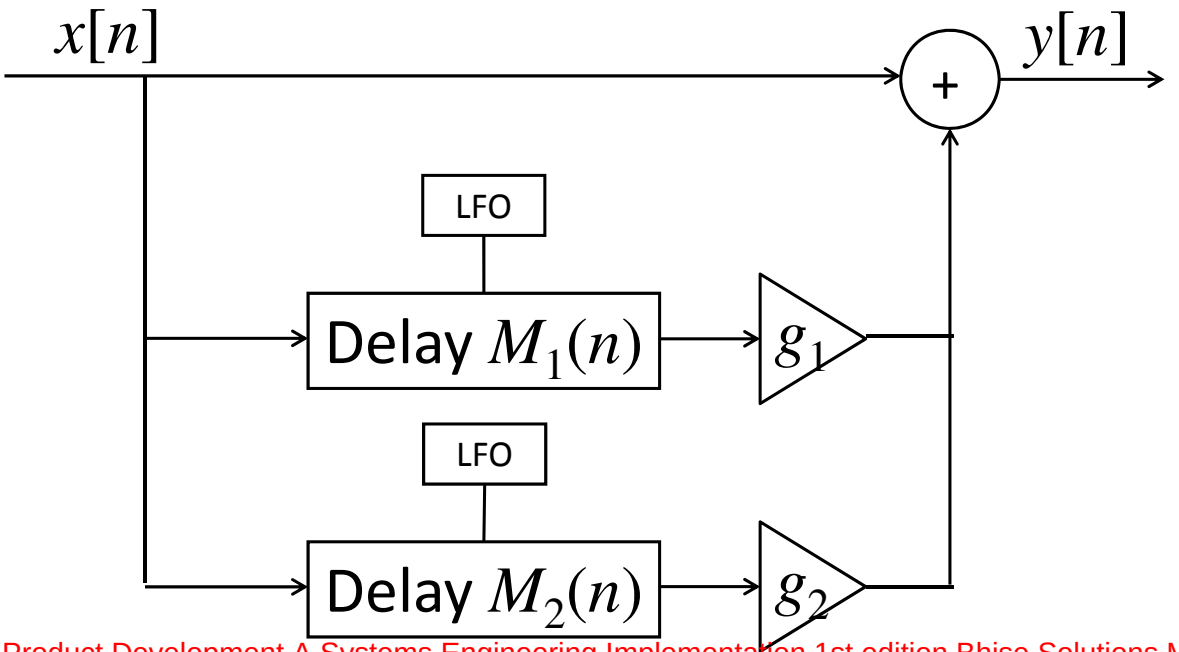
LFO waveform

- Sine
  - Smooth
  - Continuously changing, both pitch and delay
- Triangle
  - Only 2 pitches because only 2 slopes (positive and negative)
  - Sudden change between pitches
- Log (exponential)
  - Smooth over cycle, but jump at end of each cycle
  - Slope at beginning and end of waveform differ
    - Causes abrupt change in pitch
  - Large sweep width stresses LFO effect



More parameters

- Number of voices
  - Multiple copies of sound
    - Model more than 2 instruments being played
  - Multi-voice chorus uses single LFO waveform for all voices
  - Each voice has different phase
    - At any point in time, each voice is at different point in waveform
    - Different delay times
    - If all in phase, same effect as single voice chorus with increased level



Stereo chorus

- Run 2 monophonic choruses in quadrature phase
  - Same delay, width, etc.
  - Each LFO differs in phase by 90° (1/4 wavelength)
- Sound arriving at each ear is different
  - Creates wider sound
  - Same principle as stereo flanger
- For multi-voice chorus
  - Create stereo chorus simply by panning voices away from centre of mix

Aside: building real-time effects

- Delay, Vibrato, Flanger, Chorus all use buffers to hold delayed audio samples
  - How long does buffer need to be?
    - As long as the maximum delay (for any point in the LFO cycle)
  - But how does audio get *into* the buffer?

- Real-time audio systems break the stream into discrete buffers
  - Can usually set the buffer size for the audio device
    - How much data at a time comes from the device?
  - But this is different than the size of the delay buffer!

Aside: building real-time effects

- The audio hardware, not your program, controls when sound is processed
  - You create a callback function to process audio
    - The system "calls you back" whenever there's work to be done
  - The system will tell you the buffer size
    - i.e. how many audio samples are there to process *right now*?

callback function

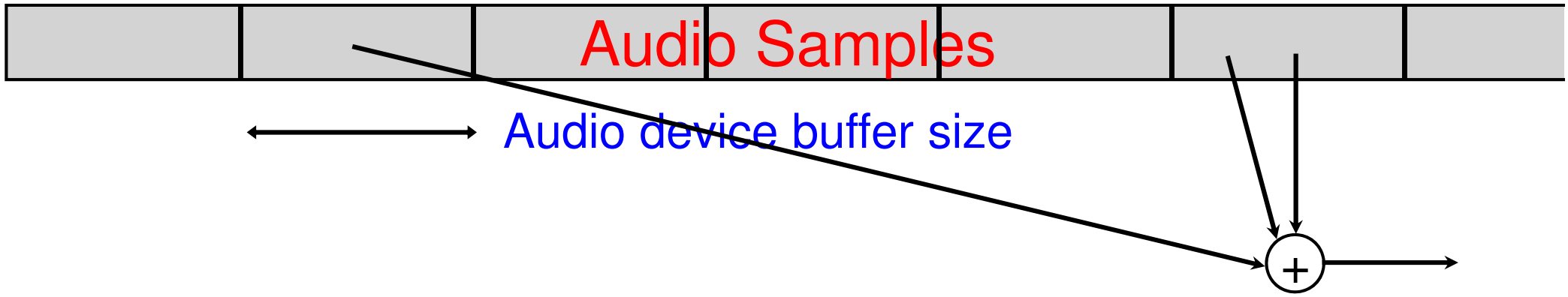
```
void DelayAudioProcessor::processBlock (AudioSampleBuffer& buffer, MidiBuffer& midiMessages)
{
    // Helpful information about this block of samples:
    const int numInputChannels = getNumInputChannels();      // How many input channels for our effect?
    const int numOutputChannels = getNumOutputChannels();    // How many output channels for our effect?
    const int numSamples = buffer.getNumSamples();           // How many samples in the buffer for this block?

    ...
```

audio device buffer size

  - Your job now is to process only those samples that just came in. Run your effect and return when finished.
    - But you might need to remember what happened previously!

Aside: building real-time effects



- What happens if we need a delay longer than the audio buffer size?
  - This is why we allocate a separate delay buffer
  - Delay buffer holds the past M samples, regardless of how the audio device chooses to deliver them
- What if the delay is shorter than the audio buffer size?
  - Same problem! Might need a sample from last time the callback function was run
  - So, always keep a separate delay buffer

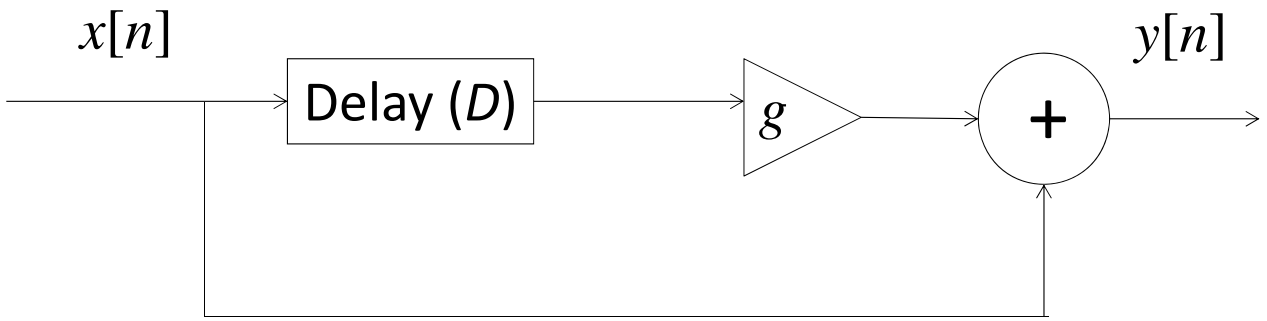
Delay-Based Effects

Basic delay

- From a digital system perspective, delay is simple:
  - Difference equation adding the current input sample to one *D* samples prior

$$y[n]=x[n]+ax[n-D] \qquad |a|<1 \quad \text{(usually)}$$

- For audio:
  - Delay ranges from milliseconds to several seconds
  - What values of *D* does this imply for sampling rate= 44.1kHz?



1 second = 44100 samples 22.6μS = 1 sample

1 ms = ~ 44 samples

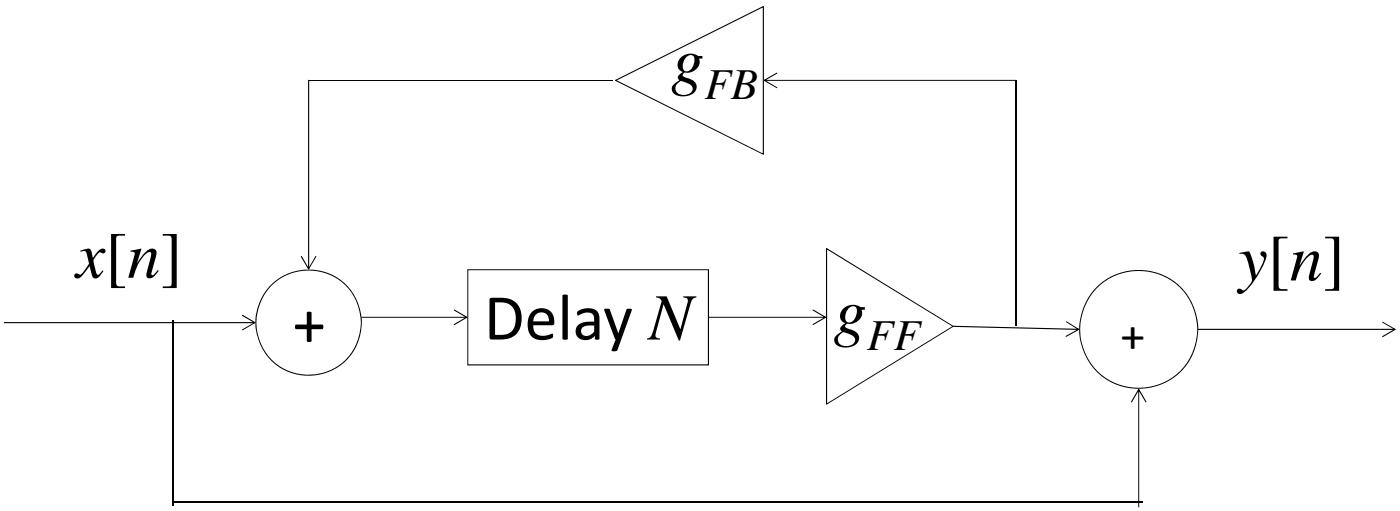
Basic delay: aesthetics

- Takes audio and plays it back after delay time
  - One of the simplest audio effects
- Use delay to:
  - Bring to life dull mixes
  - Widen an instrument's sound
  - Solo over yourself
- Delay is building block for other effects
  - Echo
  - Reverb
  - Chorus
  - Flanging
  - ...

Delay with feedback

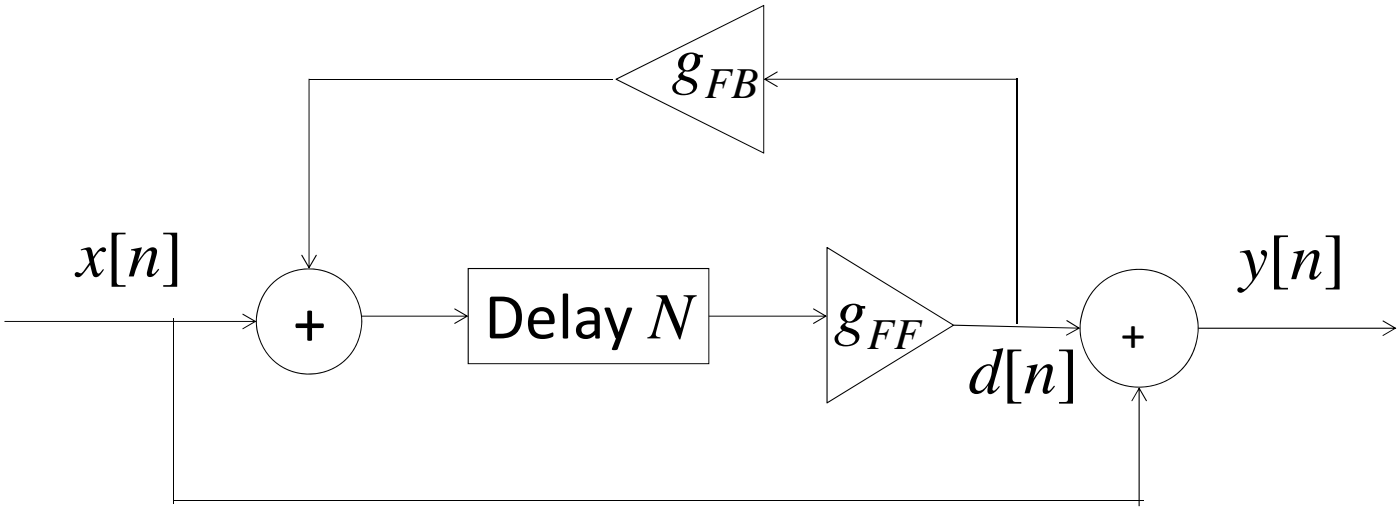
- Most delays have feedback control (regeneration)
  - takes delay output, sends it back to input
  - repeat sound over and over
- Quieter each time it plays back
  - assuming feedback gain is less than one
    - Most delay devices restrict gain to less than one for stability
  - after some point, drops below noise floor → inaudible

Basic delay unit with feedback



Delay with feedback

- To derive $y[n]$ in terms of $x[n]$:



$$d[n]=g_{FF}\{x[n-N]+g_{FB}d[n-N]\} \quad , \quad y[n]=x[n]+d[n]$$

→

$$d[n-N]=y[n-N]-x[n-N]$$

$$d[n]=g_{FF}\{x[n-N]+g_{FB}d[n-N]\}$$

$$y[n]=x[n]+g_{FF}\{x[n-N]+g_{FB}d[n-N]\}$$

$$=x[n]+g_{FF}\{x[n-N]+g_{FB}(y[n-N]-x[n-N])\}$$

$$=x[n]+g_{FF}(1-g_{FB})x[n-N]+g_{FF}g_{FB}y[n-N]$$

Delay with feedback C++ code

```
int numSamples;     // how many audio samples to process
float *channelData;// Array of audio samples, length numSamples
float *delayData;  // Our circular delay buffer of audio samples
int delayBufLength;// Length of delay buffer in samples
int dpr, dpw; // Read and write pointers into the delay buffer
// User-adjustable effect parameters:
float dryMix_; // Level of dry (undelayed) signal
float wetMix_; // Level of wet (delayed) signal
float feedback_; // Feedback level for the delay (0 if no feedback used)
for (int i = 0; i < numSamples; ++i)
{
   const float in = channelData[i];
   // Output is input plus weighted contents of delay buffer.
   float out = (dryMix_ * in + wetMix_ * delayData[dpr]);
   //Store current information in delay buffer. delayData[dpr] is delay sample we
   //read (what came out of buffer). Write delayData[dpw] to buffer (what goes in)
   delayData[dpw] = in + (delayData[dpr] * feedback_);
   if (++dpr >= delayBufLength) dpr = 0;
   if (++dpw >= delayBufLength) dpw = 0;
   // Store output sample in buffer, replacing the input
   channelData[i] = out;
}
```

Long delays

- Delay times above 100ms no longer a subtle effect
  - Can match delay time to tempo of song
    - Delayed copies of sound fall on a beat
- Extending to very long delays: 1 second or more
  - Can play over yourself
  - develop harmonies even though you may only play one note at a time
- See *A Study of The Edge's (U2) Guitar Delay* by Tim Darling
  - http://www.amnesta.net/edge_delay/

Looping and sampling

- Record only a segment of playing and loop it
  - e.g. a chord progression
  - Play recorded audio over and over
  - Solo over yourself, without rhythm player
- Some delay pedals include sampling capability
  - Length of sample often limited to 2 seconds or less
- For serious looping, need longer recording time
  - Memory depends on duration, sample rate, bit depth
- Popular units offer additional capabilities
  - Recording additional sounds onto sample
  - Playing loop backwards
  - Recording several different loops

Delay and mixing

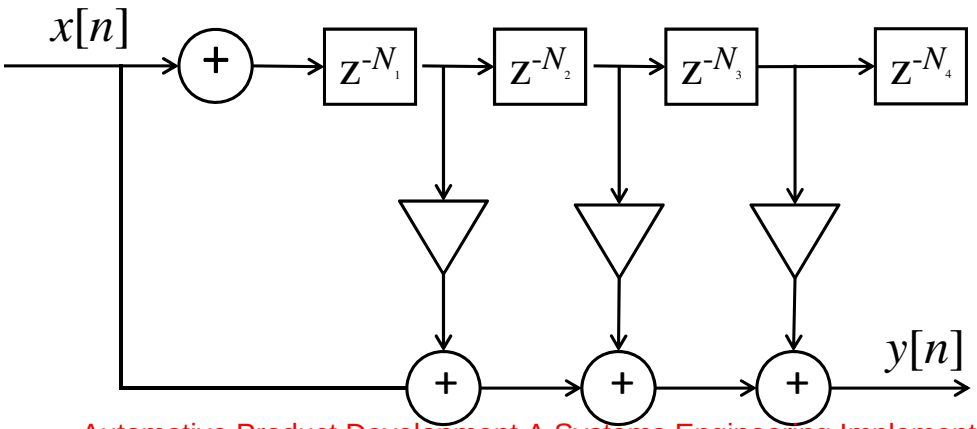
- Important when mixing for stereo
  - Enhance stereo placement of instruments
  - Make mix sound “bigger”
- Small delay can be more effective than panning for spreading tracks out in stereo field
  - Simple delay ca. 20ms can make big difference
- We discuss this when covering spatial effects

Slapback and echo

- Slapback delay is not new algorithm
  - same as basic delay without feedback
- Delay is called slapback delay if delay time fairly short
  - between 40 and 120 milliseconds.
- Longer delay called echo

Multi-tap delay

- Simple delay: outputs taken after total delay time
- Multi-tap delay is more flexible
  - Take outputs after only portion of delay time
  - Known as tapping the delay line
    - Like tap in water pipe allows you to get water at various points
- Units labeled with number of available taps
  - 3-tap delay has 3 taps to use, a 4-tap has 4, etc.
  - Unwanted taps removed by setting output level to 0
  - Amount of delay between taps can be different

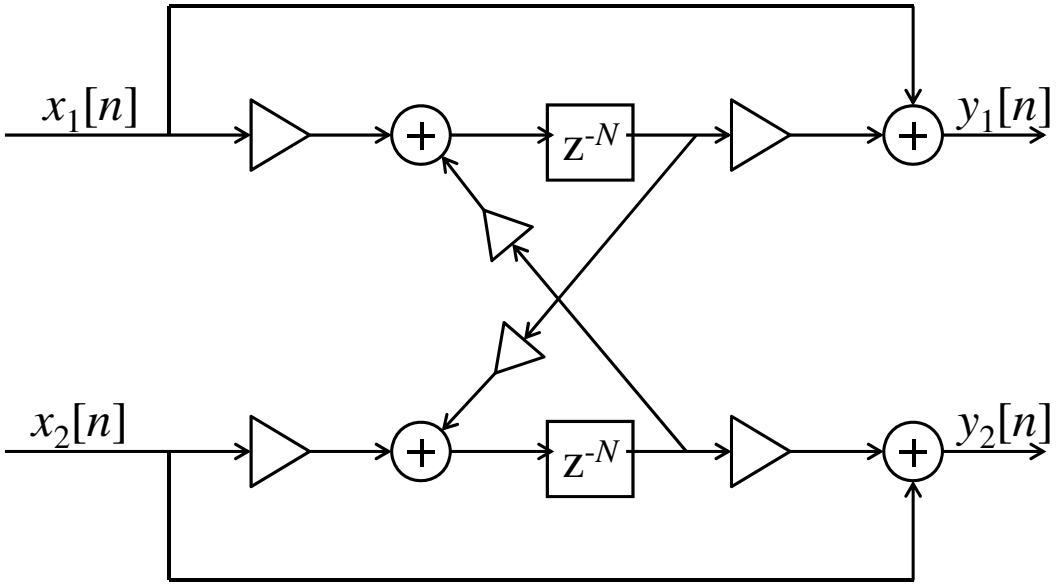


Multi-tap delay

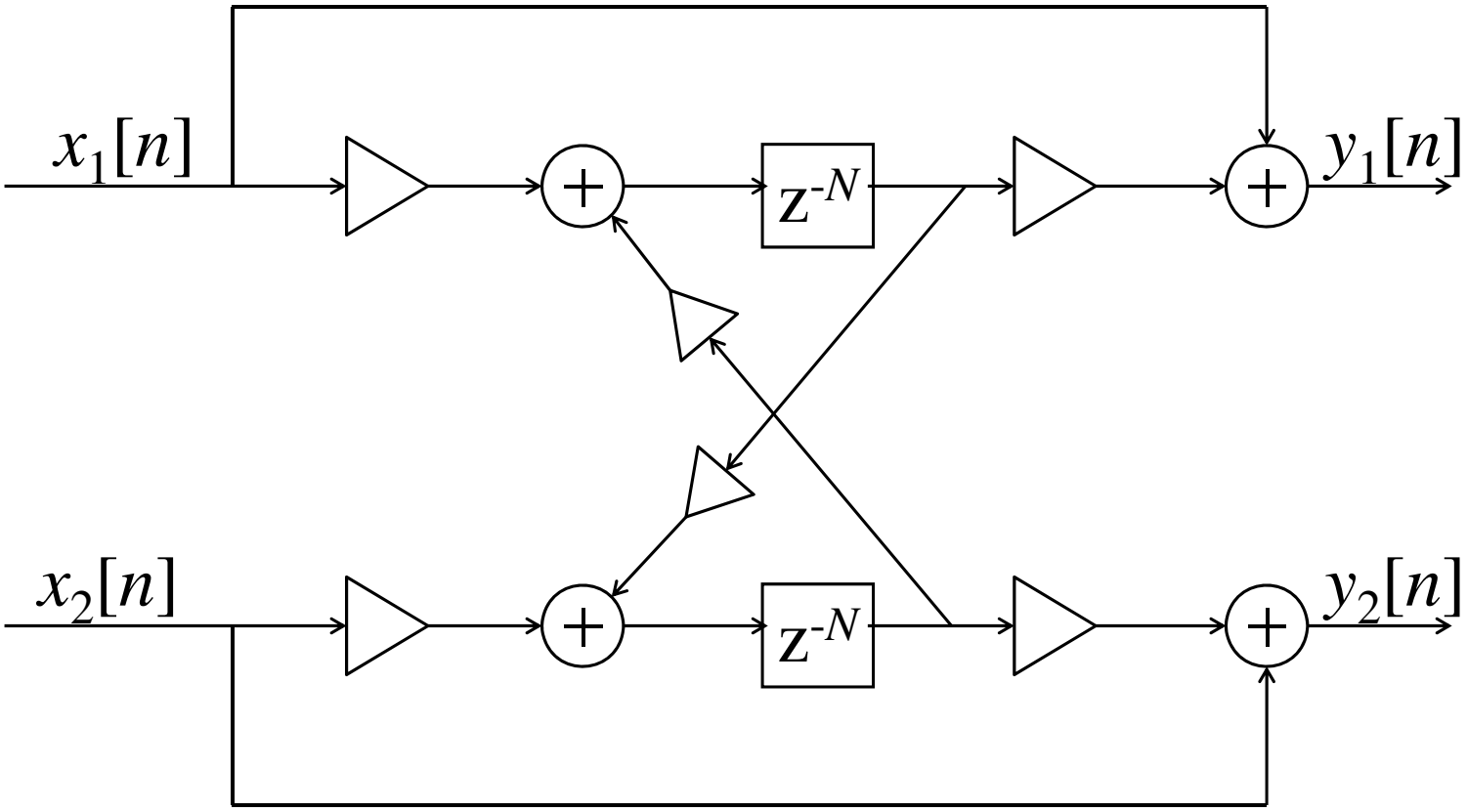
- General case of basic delay design
  - Set all but one tap gain to 0, place remaining tap at delay line end → results in basic delay line
- Multi-tap delay can be generalised further
  - Allow feedback from *each* delay output to beginning
  - Easy to create unstable system!
- Looking only at single tap
  - Sound repeats according to total delay time
  - Input sound will appear at tap output before total delay time (except last, rightmost tap)

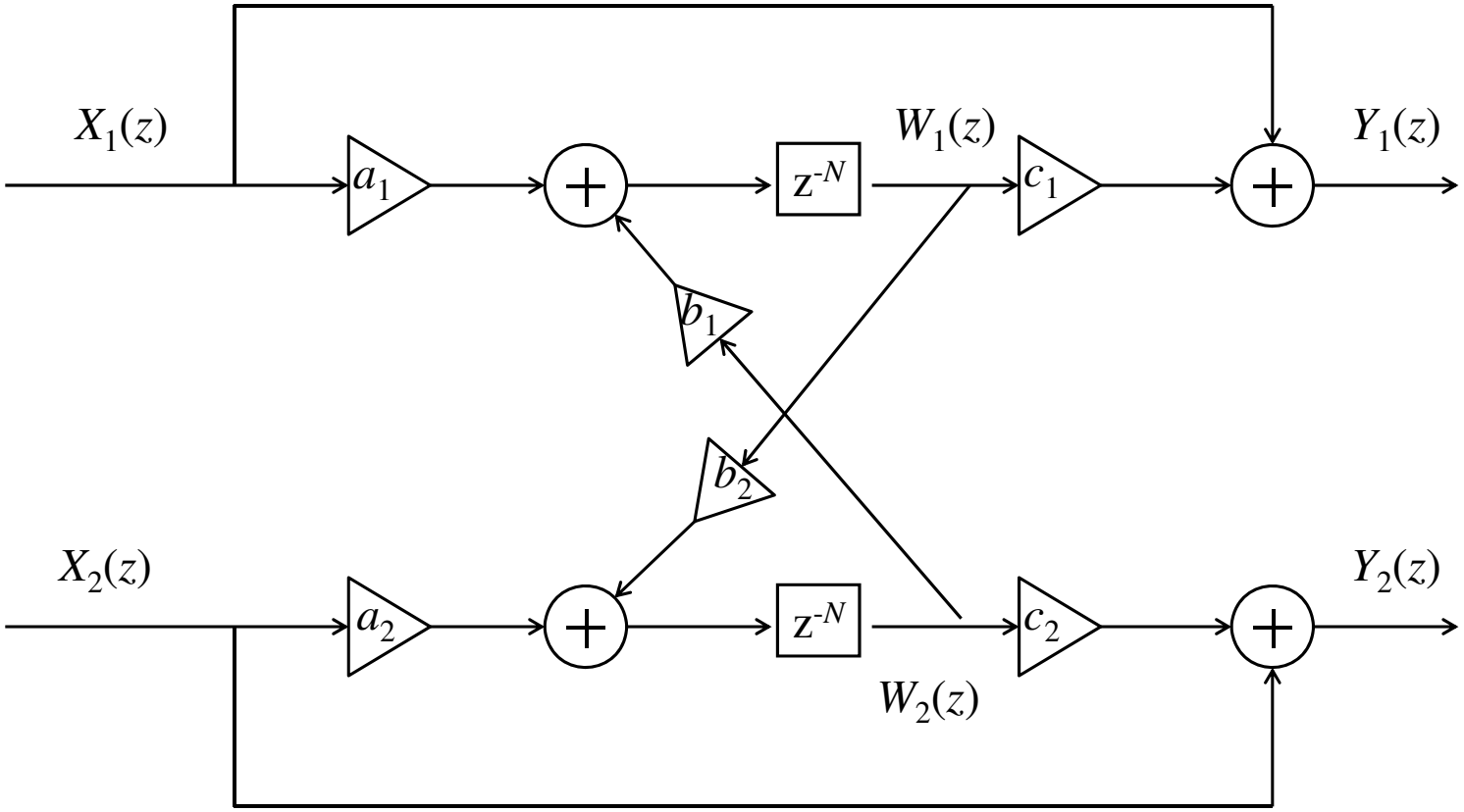
Ping-pong delay

- For stereo audio systems
- Creates bouncing effect
  - Sound typically jumps between left and right channels
- 2 delay lines with 2 inputs, 2 outputs
  - Can use same signal for each input
  - Typically pan outputs hard left and hard right
- Feedback from each delay line goes to opposite line



Lets work out the transfer functions





$W_1(z)=[b_1 W_2(z)+a_1 X_1(z)]\ z^{-N}$

$W_2(z)=[b_2 W_1(z)+a_2 X_2(z)]\ z^{-N}$

Solve these equations for W_1

$\rightarrow b_1\ b_2 W_1 z^{-2N} + b_1\ a_2 X_2\ z^{-2N} + a_1 X_1 z^{-N} = W_1$

$\rightarrow W_1=[a_2\ b_1\ X_2\ z^{-2N} + a_1 X_1\ z^{-N}]/[1-b_1\ b_2\ z^{-2N}]$

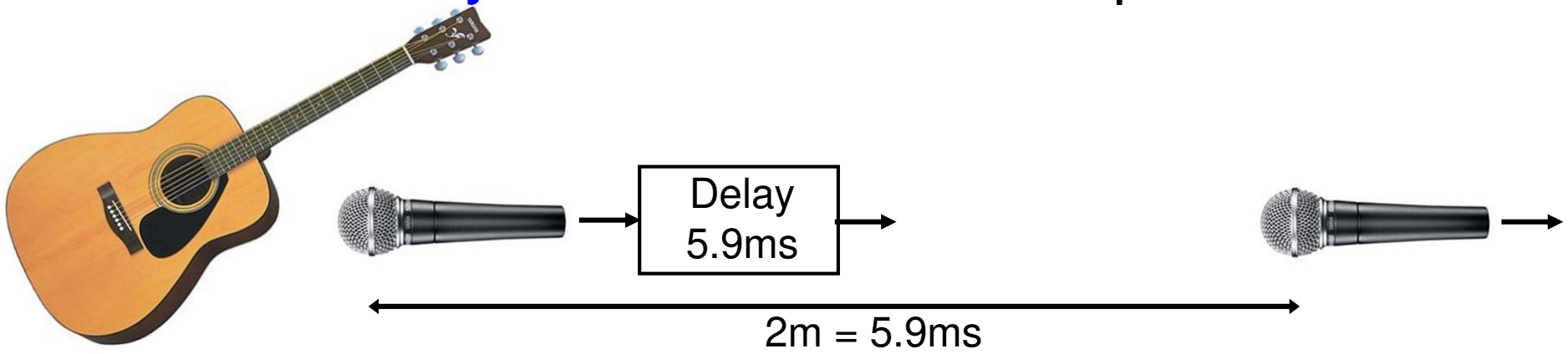
Now plug into

$Y_1(z) = c_1 W_1(z)+X_1(z)$

...and similar for the other channel

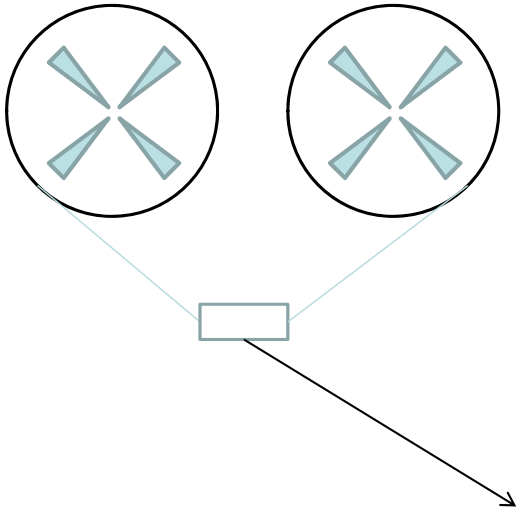
Delay in the recording process

- Suppose we record an instrument with a close microphone and a distant microphone
  - Sound travels at 340.3m/s
  - If mics are 2m apart, sound arrives 5.9ms later to the distant mic
- Adding closely timed but not identical signals creates phase problems
  - Selective cancellation of certain frequencies
  - This can be used deliberately as a musical effect (more on this soon)
- Solution: delay the close mic to compensate



Analogue implementation

- Magnetic tape
  - Open reel tape decks have record head and a play head, located separately
  - Write signal to tape on record head, read it back from play head



- Delay is time for tape to travel between heads
  - Adjust by changing tape speed or head distance
- To add feedback
  - Feed some signal from play head back to record head
  - Multi-tap delay created with multiple play heads

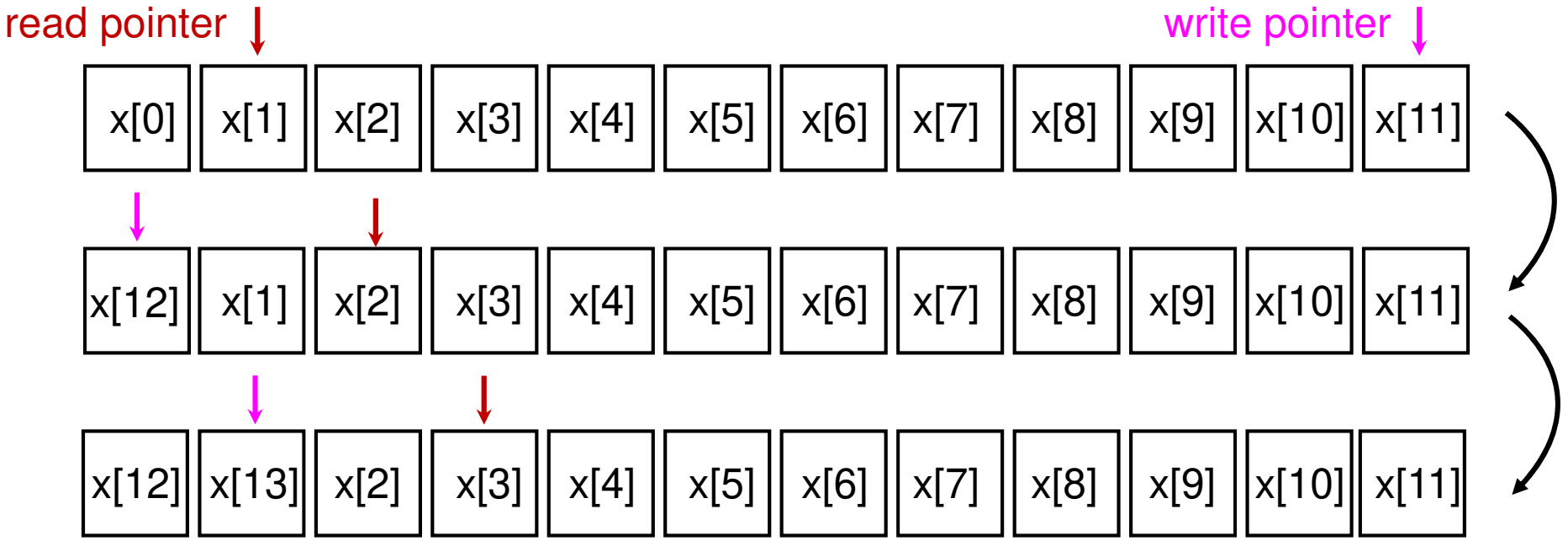
Analogue implementation 2

- Interesting tape delay features difficult to implement digitally
  - Feedback gain > 1 → signal on tape will grow
  - But growth limited by capacity of tape as it saturates
  - Not desirable for all applications, but could be used selective (e.g. vary feedback gain over time)
- If tape machine has moveable heads
  - Move heads while sound plays
  - Varies pitch of sound
- Very short delays implemented in analogue using sample-and-hold “bucket brigade devices”
  - Desired time reached by cascading devices together
  - More precise control this way than with tape

Digital delay implementation

- Operation
  - Read an input recorded previously
  - Store current input in another memory location
    - Or may be same location that was just read
    - This is why value read before writing
  - Next sampling period
    - Read & write next location in memory
    - When end of memory, loop around to first memory location
    - This is a circular buffer, and it is quite efficient
- Programming delays
  - Read and write pointers keep track of where to read/write in memory
  - Pointers increment each sample
  - Multi-tap delays created using additional read pointers

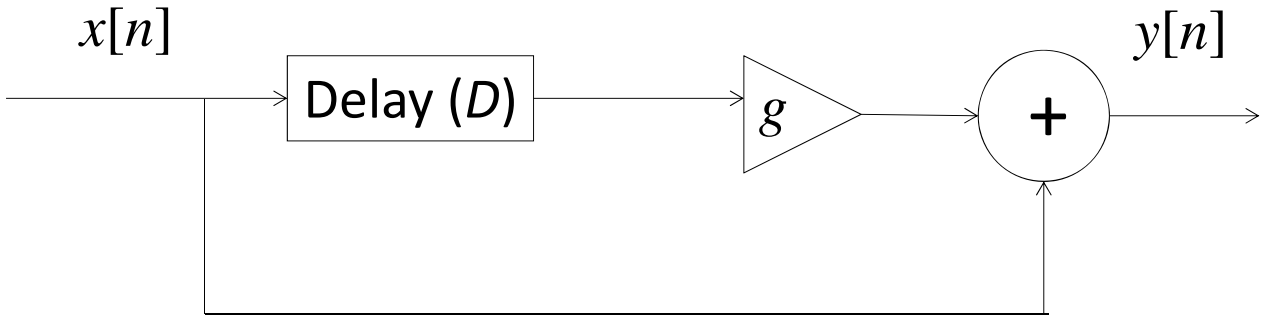
Delay on a circular buffer



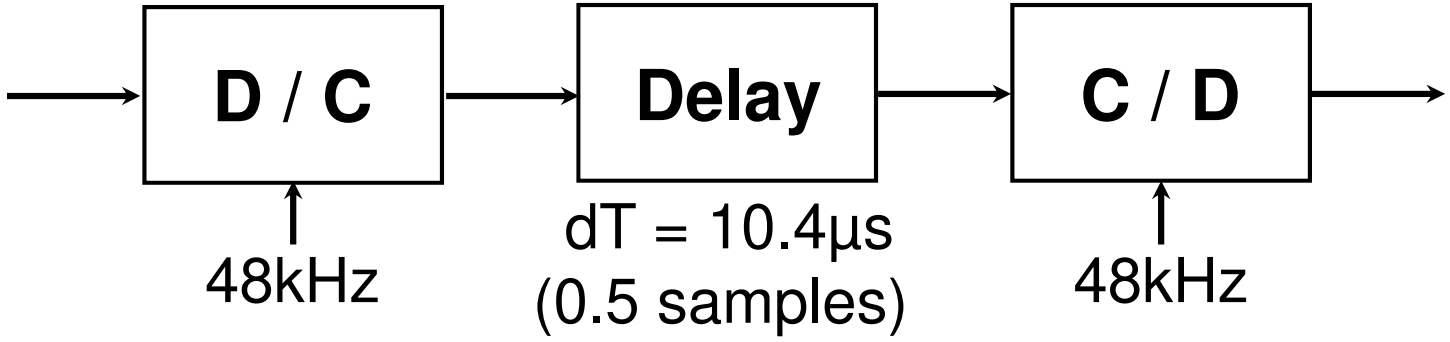
- Buffer is a contiguous block of memory
  - Read and write pointers point to specific locations within this block
  - Difference in read/write pointer location determines delay
  - This is equivalent to difference equation representations

Fractional delay

$$y[n] = x[n] + ax[n - D] \qquad |a| \leq 1 \ \text{(usually)}$$



- What happens when we want non-integer *D*?
  - x[n] undefined if n is not an integer
- Consider this discrete/continuous system:



Fractional delay

- Ideal discrete-to-continuous converter
  - "sinc reconstruction"

- Delay of D samples corresponds to a convolution:

$$y[n] = x[n] * h[n] \quad \text{where}$$

$$h[n] = \frac{\sin \pi(n - D)}{\pi(n - D)} \quad -\infty < n < \infty$$

  - Not causal
  - Infinitely long convolution

Full derivation: Oppenheim, Schafer and Buck,
Discrete-Time Signal Processing, 2nd ed., Sec 4.5

Fractional delay in practice

- Use interpolation between samples
  - Linear (fast), second order or cubic (better quality)
- Pseudocode (next slide...)
  - Input:
    - An input signal x[n] at sampling rate `R`
    - Required delay line size `D`
    - Starting delay for read pointer `initialdelay`
    - Read pointer update rate `updateread` (write pointer update rate=1)
  - Output:
    - N samples of signal y[n] at sampling rate R (output of read pointer)

Fractional delay in practice

```
Allocate buffer delayline of size D for storing samples
writepos=0; // write pointer position, start at beginning of buffer
readpos=(writepos+D-initialdelay)%D; // read pointer position, modulo keeps indexing correct
for (n=0; n<=N-1; n++)
{
  prev=floor(readpos); // get previous integer index
  next=(prev+1)%D; // get next integer index; because of wraparound, need modulo for safety
  t=readpos - prev; // get fractional position of read pointer
  y[n]=(1-t)*delayline[prev]+t*delayline[next]; // Linear interpolation
  readpos=(readpos+updateread)%D; // update read pointer for next time, modulo wraps
  delayline[writepos]=x[n]; // write current sample to delay line
  writepos=(writepos+1)%D; // update write pointer for next time, modulo wraps
}
```

Adapted from Nick Collins, Introduction to Computer Music, 2010

- Code doesn't check whether pointers cross
  - pointers are moving at different speeds here
  - won't happen when moving at same speed
- Read always happens before write to delays of full buffer length work properly

Flanging

Overview

- Story
- Background
- How it works
- Frequency Response
  - Interference
- Sound and Pitch Modulation
- Common parameters
  - Speed and Excursion
  - Depth Control
- Inverted Mode
- Flanger Feedback Control
- Implementation

Story of the flanger

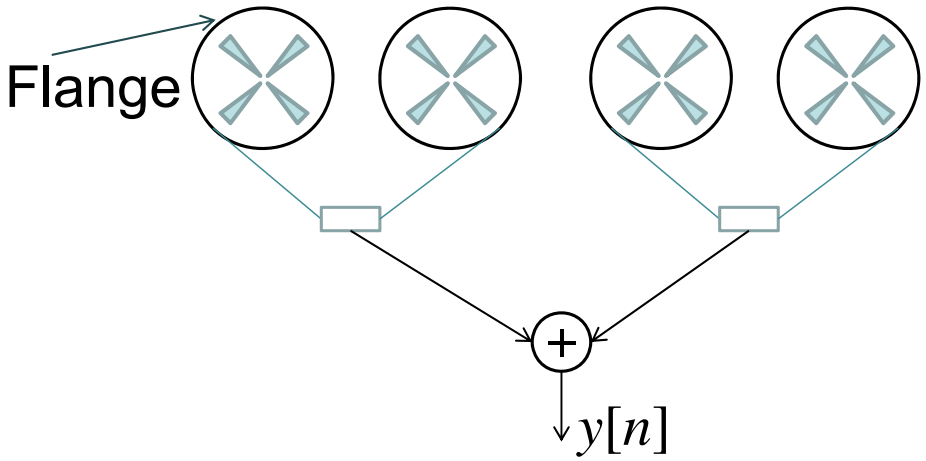
"They often liked to double-track their vocals, but it's quite a laborious process and they soon got fed up with it. So, after one particularly trying night-time session doing just that, I was driving home and suddenly had an idea."
Ken Townsend, Beatles Studio Technician

- Artificial Double Tracking (ADT)
  - Recorded vocal track re-recorded onto second machine
  - Second machine has variable (oscillating) tape speed
    - Alternatively, change speed with finger on rim (flange) of wheel
  - Combine (mix) original and modulated signals
  - Irregularities mimic the natural variations in the human voice when overdubbed

"I knew he'd never understand it, so I said, 'Now listen, it's very simple. We take the original image and split it through a double vibrocated sploshing flange with double negative feedback...' He said, 'You're pulling my leg, aren't you?' I replied, 'Well, let's flange it again and see.' From that moment on, whenever he wanted ADT he would ask for his voice to be 'flanged,' or call out for 'Ken's flanger.' George Martin, Beatles Producer

Flanging introduction

- First: 2 tape machines playing same tape
  - Outputs mixed equally
  - Flange of supply reel touched to make it play slower
  - Delay develops between machines
- Then: flange released, flange of other machine touched
  - Delay gradually disappears, grows in opposite direction
  - Total delay kept below echo perception threshold
    - A few milliseconds in each direction
- Repeat as desired, alternating flanges

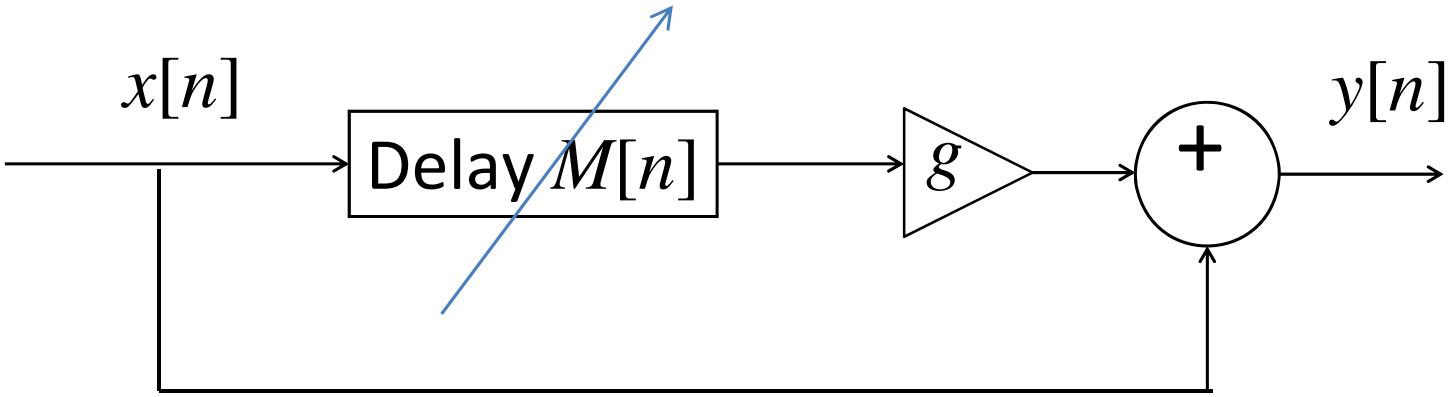


Flanging introduction

- Characteristic sound referred to as *whooshing*
  - Like jet plane flying overhead
- Rapid flanging introduces Doppler effect
  - Like Leslie speaker on electric organs
- Creates equally-spaced notches in audio spectrum
- Later: compare to phaser
  - Also based on notches
  - Spacing can be arbitrary
  - Phaser uses all-pass filters instead of delay

Leslie speaker
Creative commons image from Wikipedia

Flanging: how it works



- Mix signal with slightly delayed copy of itself
  - Length of delay constantly changing
- Depth control
  - How much delayed signal added to original
- Not an echo effect
  - Typical delays are 1-10ms
  - Human ear perceives echo starts at 50-70ms

Flanging: how it works

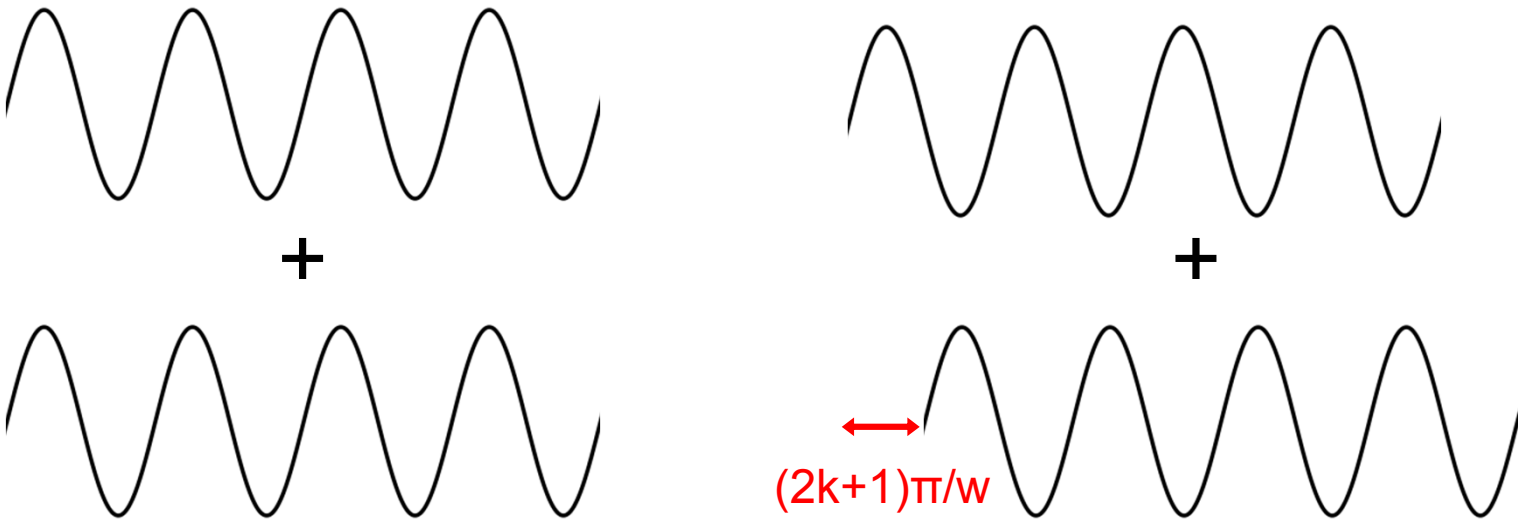
- Feedforward comb filter
- Input-output relation: $y[n]=x[n]+gx[n-M(n)]$
  - $x[n]$ - input signal
  - $y[n]$ - output signal
  - g - depth of flanging effect
  - $M(n)$ - length of delay line at sample n
- $M(n)$ must vary smoothly over time
  - Interpolated (fractional) delay line allows noninteger M

Delay and interference

- Consider two related signals:
  - $x[n] = \cos(\omega n)$
  - $x[n - D] = \cos(\omega(n - D))$
- Constructive interference:
  - $D = 2\pi k/\omega$ for integer *k*
  - $x[n] + x[n - D] = cos(\omega n) + cos(\omega n - 2\pi k) = 2cos(\omega n)$
- Destructive interference:
  - $D = ((2k + 1)\pi)/\omega$ for integer *k*
  - $x[n] + x[n - D] = cos(\omega n) + cos(\omega n - (2k + 1)\pi)$
    $= cos(\omega n) - cos(\omega n) = 0$
- Interference depends on delay with respect to frequency of the signal

Delay and interference

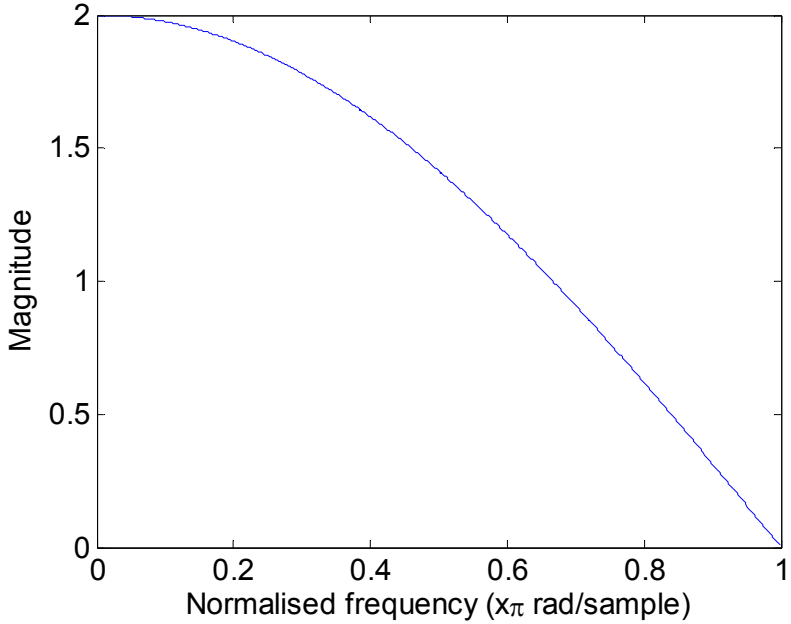
- (Usually undesired) destructive interference, known as comb filtering, is common in mixing
  - Multiple microphones to record single source
  - Mixing electric instrument direct signal with acoustic amplified signal
  - Parallel signal paths with different latency times

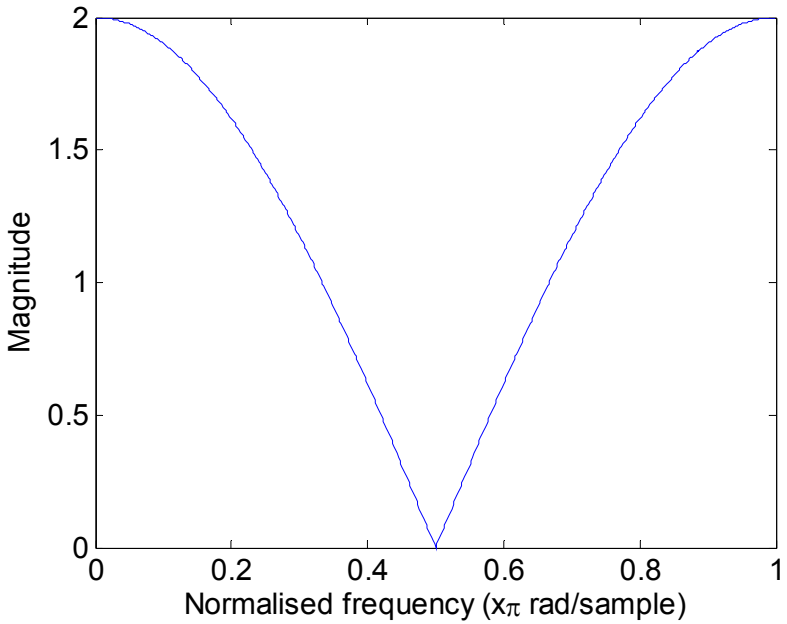


Frequency response

- Delay & add has filtering effect on signal
  - Creates series of notches in frequency response
  - Eliminates periodically spaced set of frequencies
  - Other frequencies passed with amplitude change
- Comb filter - notches resemble teeth on comb

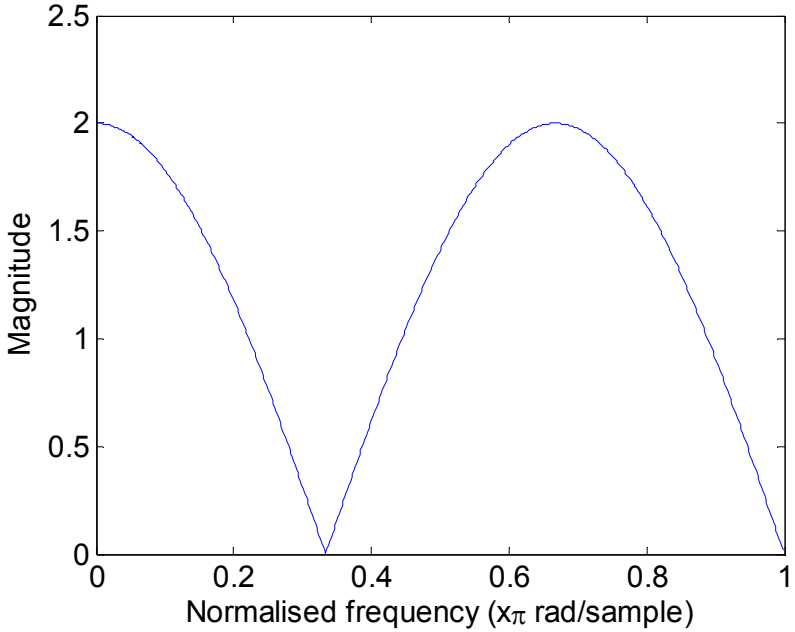
Frequency response of simple flanger

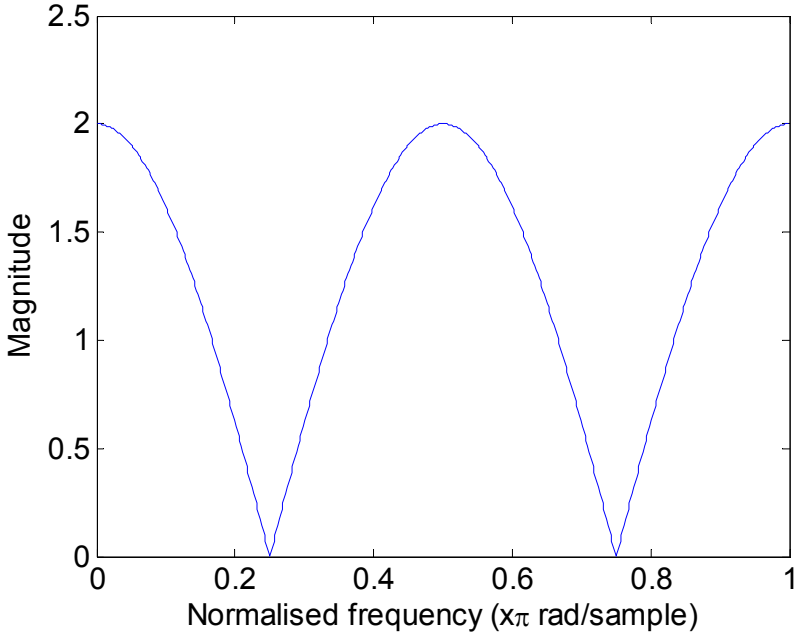




depth set to 1

delay values are 1, 2, 3 and 4 samples



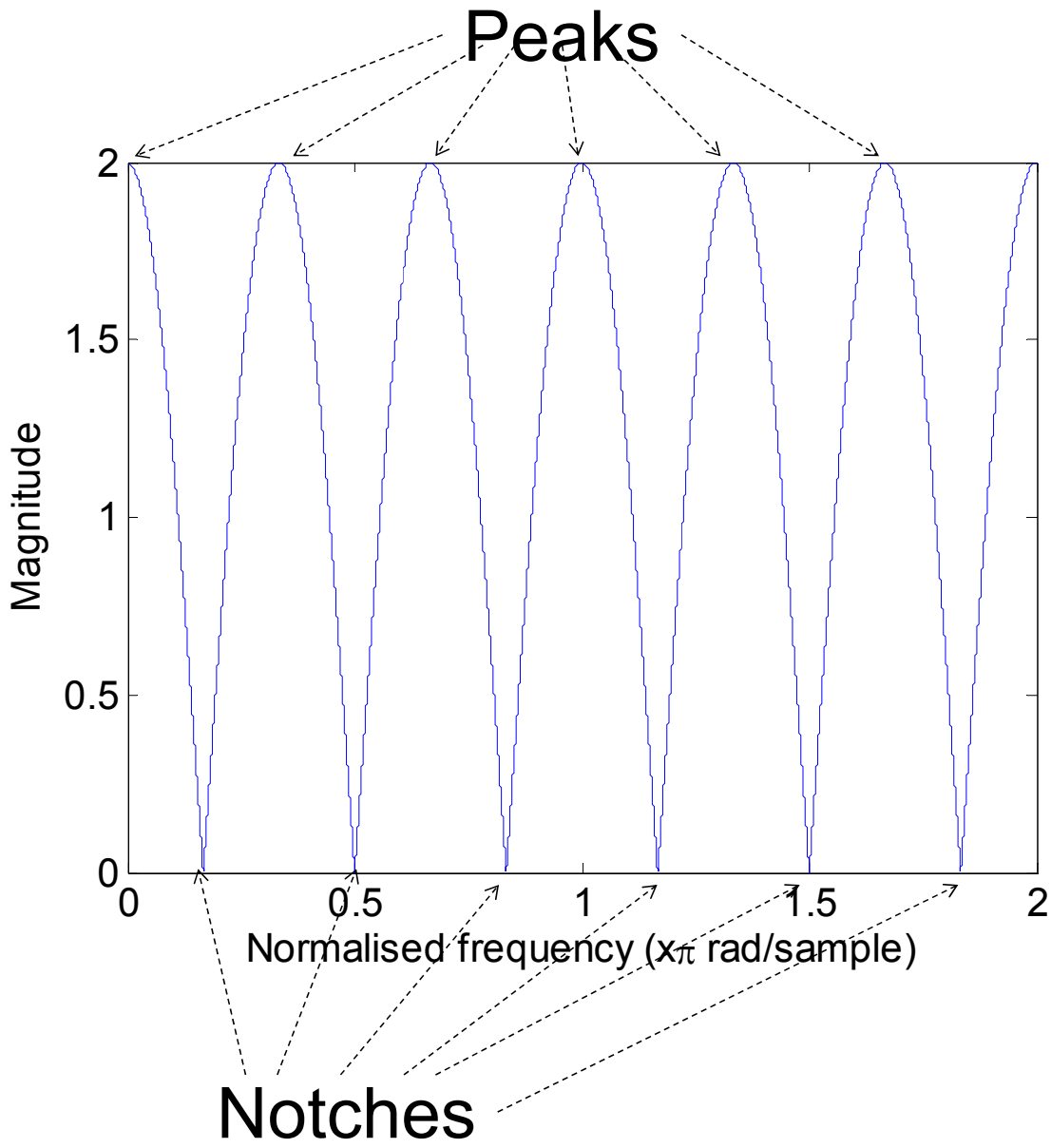


Frequency response of a simple flanger

six sample delay and depth set to 1. The locations of the six peaks and six notches over the whole frequency range from 0 to 2π is shown

Six notches at π/6, 3π/6, 5π/6, 7π/6, 9π/6, 11π/6.

Peaks at 0, 2π/6, 4π/6, 6π/6, 8π/6, 10π/6.



z-transform and delay theorem

- Recall the z-transform $X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}$
- When $z = e^{j\omega}$ we can find the frequency response
  - Normalised units, frequency ranges from 0 to 2π
- Suppose we know the z-transform of a sequence

$$\mathcal{Z}\{x[n]\} = X(z)$$

  - Then we have $\mathcal{Z}\{x[n-1]\} = z^{-1}X(z)$
  - and in general $\mathcal{Z}\{x[n-N]\} = z^{-N}X(z)$

Flanger's Frequency Response

Delay Equation: $$y(t) = x(n) + x(n - M)$$

Z-Transform: $$Y(z) = X(z) + z^{-M} X(z), \quad z = e^{jw}$$

Transfer Function: $$H(z) = Y(z) / X(z) = z^{-0} + z^{-M}$$

$$H(jw) = e^{-j0} + e^{-jwM}$$

Frequency Response: $$H(jw) = e^{-jwM/2}(e^{-jwM/2} + e^{jwM/2})$$

$$2\cos(wM/2)$$

Amplitude Response: $$|H(jw)| = |2\cos(wM/2)|$$

Frequency response

With depth control

Delay equation: $y[n] = x[n] + gx[n - M]$

z-Transform: $Y(z) = X(z) + gz^{-M}X(z)$

Transfer function: $H(z) = \frac{Y(z)}{X(z)} = z^{-0} + gz^{-M}$ where $z = e^{j\omega}$

Frequency response: $H(j\omega) = e^{-j0} + ge^{-j\omega M}$

Amplitude response:

$$|H(j\omega)| = \sqrt{(1 + ge^{-j\omega M})(1 + ge^{j\omega M})}$$

$$= \sqrt{1 + ge^{-j\omega M} + ge^{j\omega M} + g^2}$$

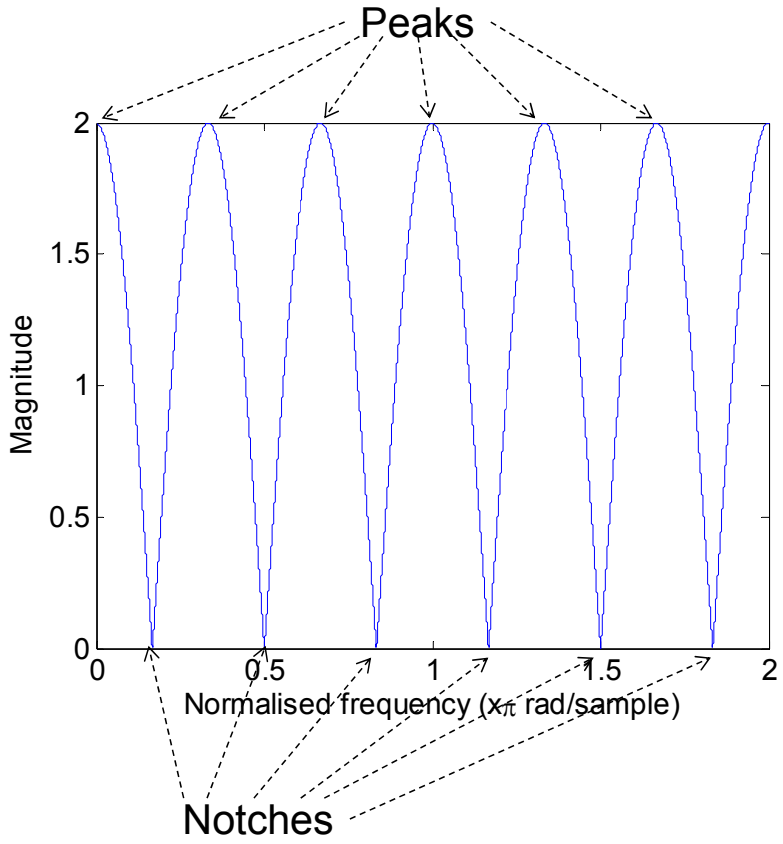
$$= \sqrt{1 + 2g\cos(\omega M) + g^2}$$

Notice when g = 1, reduces to previous case

Amplitude response

- Input-output relation for basic flanger $y[n]=x[n]+gx[n- M(n)]$
- Transfer function $H(z)=1+gz^{-M}$
- Amplitude response (gain versus frequency) $|H(\omega)|=\sqrt{1+2g\cos(\omega M)}$

- When $g > 0$
  - M peaks in response
  - Here M = 6, 6 peaks
- Peaks move slowly over time by changing M



Amplitude response

- Peaks located at:

$$\omega_k^{(p)} = 2\pi k / M, \quad k = 0,1,2,...M-1$$

- When depth g = 1:
  - Peaks maximally pronounced
  - M notches
    - Elimination of sound at single frequency or narrow freq. interval
- Notches (or minima) located at: $\omega_k^{(n)} = \omega_k^{(p)} + \pi / M$
  - g < 1 will produce less-than-complete attenuation
- Peak / notch spacing at intervals of f_s/M Hz
  - f_s is sample rate
- Time variation of delay-line length M[n]
  - Flanging created by moving notches in spectrum
  - Notch motion *essential* for flanging effect
    - Static row of notches sounds like being in an acoustic tube
    - Isolated notch may be inaudible

Sound of a flanger

- Characteristic sound results when these notches sweep up and down frequency axis over time
- Sweeping action of notches achieved by continuously changing amount of delay used
- As delay increases, notches go ...
  - Lower in the spectrum
- Delay change set by low frequency oscillator (LFO)
  - For example, sine wave at < 5 Hz

Pitch modulation

- Changing $M(n)$ over time
  - What does this mean in practice?
  - Read faster or slower from delayed signal
  - dM/dt (or $M[n]$ - $M[n\text{-}1]$) indicates difference in playback speed of delayed signal
- Classic analogue tape example
  - Pitch changes as tape speeds up or slows down
- Only delayed copy has pitch change
  - Mixed with original signal
  - Perceived pitch warbles

Depth control *g*

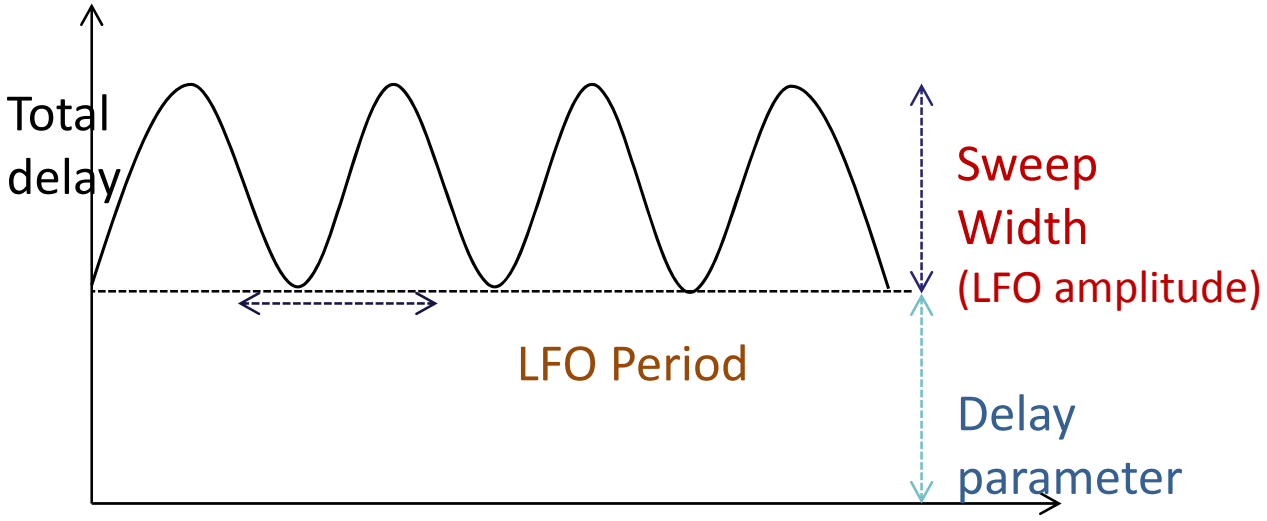
- Determines how deep notches go
  - How pronounced the effect is
- When depth is 0
  - frequency response is flatt
  - No flanging
- Increase depth→ notches appear and extend downward
- When depth=1
  - notches extend down to 0
  - Peaks maximally pronounced
- Even if notches<<1, they have audible effect
- In multieffects units, *g* may only be controllable in mixer section, not flanging processor itself

Delay parameter

- *Minimum* delay used on copy of signal
  - Determines how high the first notch will go
  - Increasing delay drops the first notch downward
- If minimum delay = 0 ...
  - Notches sweep to uppermost frequency range, momentarily disappear

Sweep width

- Range of delay times (min to max delay)
  - Width (amplitude) of LFO
- Sweep width is maximum additional delay
  - In addition to delay parameter (determines minimum)
  - Maximum delay = delay parameter + sweep width
- Determines how low the first notch will go
- Small width keeps variance in delay time small
- Large width causes notches to sweep larger area
- Sometimes called “sweep depth”
  - Not to be confused with depth *g*!



The maximum delay is the sum of the delay and sweep width parameters. The delay changes over time according to the sweep rate

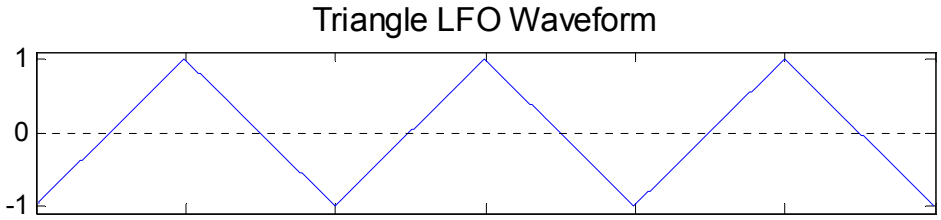
Sweep width

- Increased width → increased pitch modulation
  - Why?
  - dM/dt will grow
    - Flanger needs to read faster and slower to change delay in same amount of time
- Using sweep parameters:
  - Set delay to determine high frequency of sweep
    - Delay changes both upper and lower points of first notch
  - Adjust width to determine low frequency of sweep
    - Width only changes lower limit

plucked electric guitar string, first flanged
with depth of 2 ms, then with 6 ms
Latter varies more in pitch

LFO settings

- Delay M(n) modulated by low-frequency oscillator (LFO)
  - Waveform usually triangular, sinusoidal, or exponential
    - Exponential = triangle on log-frequency scale
- For sinusoidal case: $M[n]=M_0(1+A\sin(2\pi fnT))$
  - f - speed (or rate) of flanger in cycles per second
  - A - excursion or sweep (maximum delay swing)
  - M_0 - average delay length (controls notch density)
    - Either A or M_0 often not user-controllable
  - T - sampling period
- Triangular LFO common in most flangers



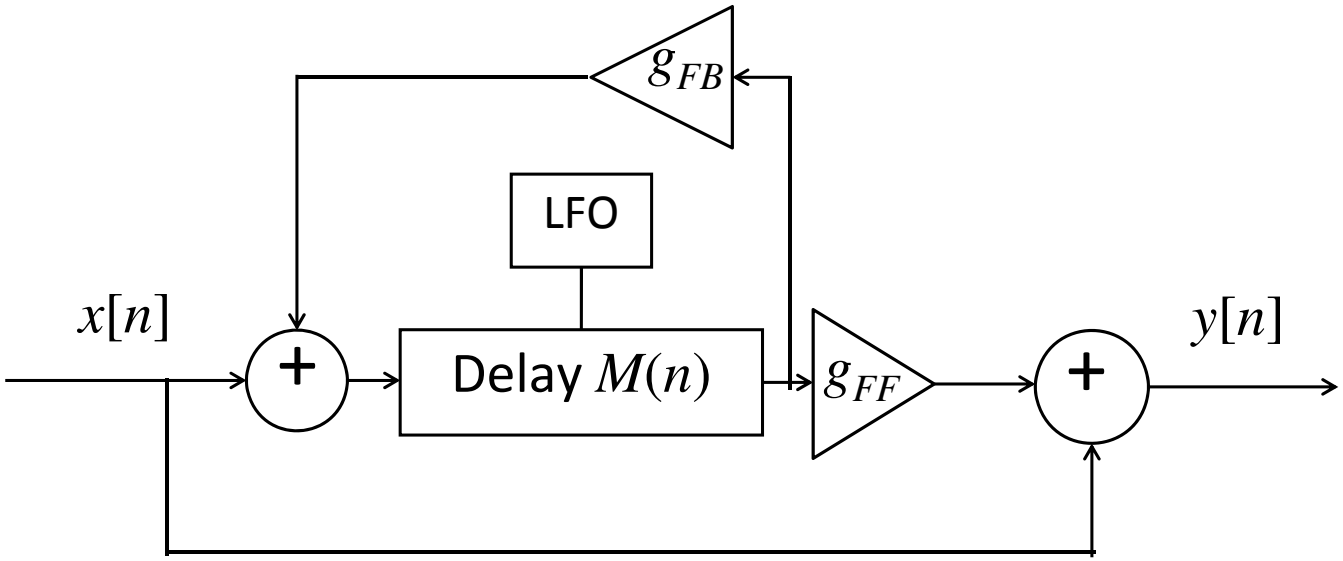
LFO settings

$$M[n] = M_0(1 + A\sin(2\pi f n T))$$

- Speed control
  - How many times per second notches oscillate
  - Also affects amount of pitch modulation (why?)
    - Derivative of sin term above:
    - $2\pi A f T \cos(2\pi f n T)$
    - Delay has to travel the same distance in less time

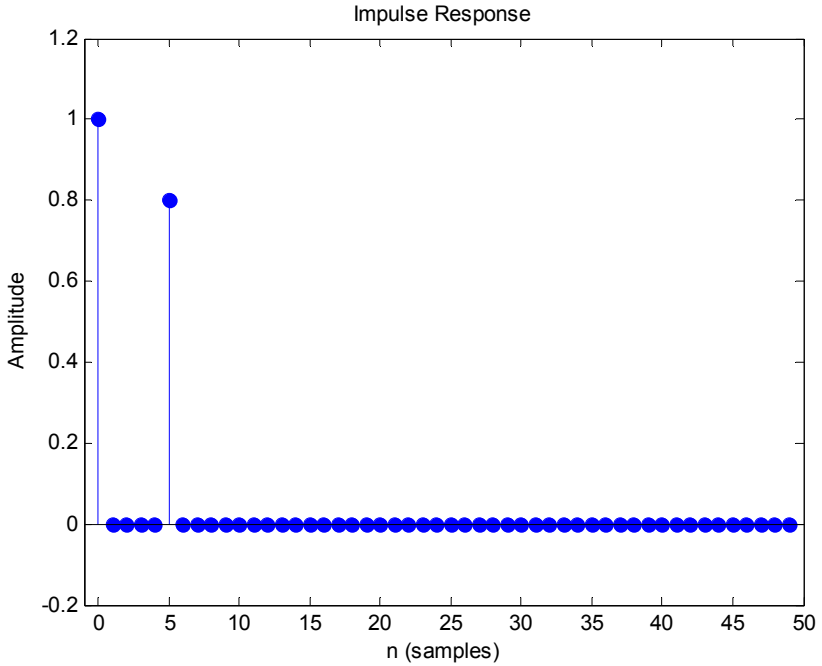
Feedback / regeneration

- Take portion of flanger output, feed back to input
  - Regeneration control sets level of feedback
  - Feedback comb filter in addition to previous feedforward comb filter
  - Large feedback creates intense or “metallic” sound

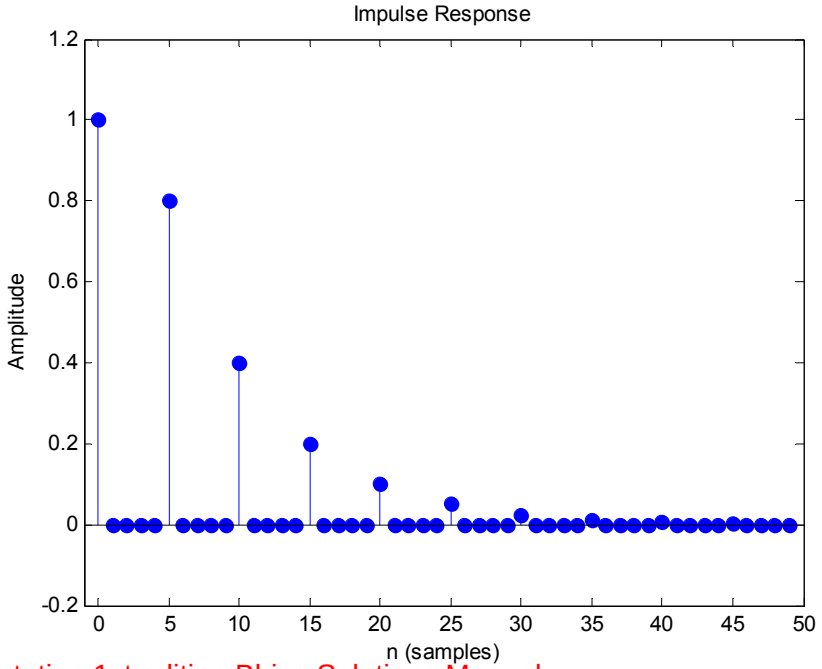


Impulse responses for comb filter with 5 samples delay

gain g= 0.8 and no feedback



g_{FF}= 0.8 and g_{FB}= 0.5



Feedback / regeneration

- As feedback goes to 1, can become unstable
  - Just as in other feedback delay cases
  - Result: overflow or clipping
- Control options
  - Add or subtract feedback signal
  - Feedback from *any subset* of allpass sections
    - Usually in small amounts

Flanger inverted mode

- Different type of maximum depth obtained for $g = -1$
  - Peaks and notches trade places relative to $g = 1$
  - First notch at DC (f = 0)
  - Results in weak bass response
    - Unless minimum delay M >> 0
- Depth control g usually constrained to [0, 1] interval
  - Sign inversion controlled with separate switch
  - Sounds high-pass filtered relative to g > 0
- As M shrinks (and notch spacing grows):
  - Amplitude response approaches first-order difference

$$y[n]=x[n]-x[n-1]$$

  - Approximates (continuous) differentiator $y(t) = \frac{d}{dt} x(t)$
  - Behaviour of ideal differentiator:
    - Eliminates DC
    - Progressive high-frequency boost (6dB/octave)
    - Amplitude response is $|H(\omega)|=|\omega|$

Digital implementation

- Use delay lines (generally circular buffers)
  - Contiguous block of memory to store samples
  - Continuously read and write this space
- ✗ Changing delay creates challenges
  - Samples are one sampling period apart
  - Delay line has to be an integer value
- ✓ Solution: interpolation (fractional delay)
  - Estimate value between 2 samples
  - Zeroeth order: choose closest point to read from
    - Zipper noise: clicking in output as delay length changes
  - Linear interpolation: read value from line between points
    - Okay but still noisy
  - Cubic interpolation
    - Better, but computationally expensive

Notch spacing issues

- Only uniformly spaced notches
- non-ideal because:
  - Ear uses logarithmic frequency scale
    - Flanger notches spaced linearly
    - Exponentially spaced notches sound more perceptually uniform
  - Uniform peaks/notches impose discernible resonant pitch
    - Impression of being inside resonant tube

Disappearing instruments

- If instrument harmonics land on notches
  - instrument may be eliminated from output
- Exact alignment unlikely, but...
  - Loudness can be modulated as notches move through spectrum
  - Flanging best used with percussive, noisy or inharmonic sounds
    - Works well with drums
  - For harmonic sounds, need non-uniform notches
- Flanging often used on entire mix, rather than specific instrument within mix

Stereo flangers

- 2 monophonic flangers in quadrature phase
  - Same depth, width, delay, etc.
  - LFO in each monophonic flanger differs in phase by 90 degrees (1/4 wavelength)
- Creates ‘wider’ sound
  - Sound arriving at each ear is different
- Sound examples
  - Dry, then monophonic flanger
  - Then stereo flanger

Flanger with feedback code (1)

```
// Variables used in this example whose values are set externally:
int numSamples;      // Indicates how many audio samples to process
float *channelData; // Array of audio samples, length numSamples
float *delayData;   // Our own circular delay buffer of audio samples
int delayBufLength; // Length of our delay buffer in samples
int dpw;            // Write pointer into the delay buffer
float ph;           // Current phase of the LFO, always between 0-1
float inverseSampleRate; // 1/f_s, where f_s is sampling frequency

// User-adjustable effect parameters:
float frequency_;   // Frequency of the low-frequency oscillator
float sweepWidth_;  // Width of the LFO in samples
float depth_;       // Amount of delayed signal mixed with original
  (0-1)
float feedback_;    // Amount of feedback on the delay (>= 0, < 1)
```

Flanger with feedback code (2)

```
for (int i = 0; i < numSamples; ++i)
{
  const float in = channelData[i];
  float interpolatedSample = 0.0;
  // Recalculate read pointer position with respect to write pointer. More efficient implementation might
  // increment read pointer based on derivative of LFO without running equation again, but
  // this format makes the operation clearer.
  float currentDelay = sweepWidth_*(0.5f + 0.5f*sinf(2.0 * M_PI * ph));
  // Subtract 3 samples to delay pointer to make sure we have enough written samples to interpolate with
  dpr = fmodf((dpw - (currentDelay * getSampleRate()) + delayBufLength - 3, delayBufLength);
  // Use linear interpolation to read fractional index into buffer.
  // Find fraction by which read pointer sits between 2 samples and use this to adjust weights of samples
  float fraction = dpr - floorf(dpr);
  int previousSample = (int)floorf(dpr);
  int nextSample = (previousSample + 1) % delayBufLength;
  interpolatedSample = fraction*delayData[nextSample] + (1.0f-fraction)*delayData[previousSample];
  // Store current information in delay buffer. With feedback, what we read is included in what gets
  // stored in buffer, otherwise it's just a simple delay line of the input signal.
  delayData[dpw] = in + (interpolatedSample * feedback_);
  //Write pointer moves at constant rate. Read pointer rate depends on LFO settings, delay & sweep width.
  if (++dpw >= delayBufLength) dpw = 0;
  // Store the output sample in the buffer, replacing the input
  channelData[i] = in + depth_ * interpolatedSample;
  // Update the LFO phase, keeping it in the range 0-1
  ph += frequency_*inverseSampleRate;
  if(ph >= 1.0) ph -= 1.0;
}
```