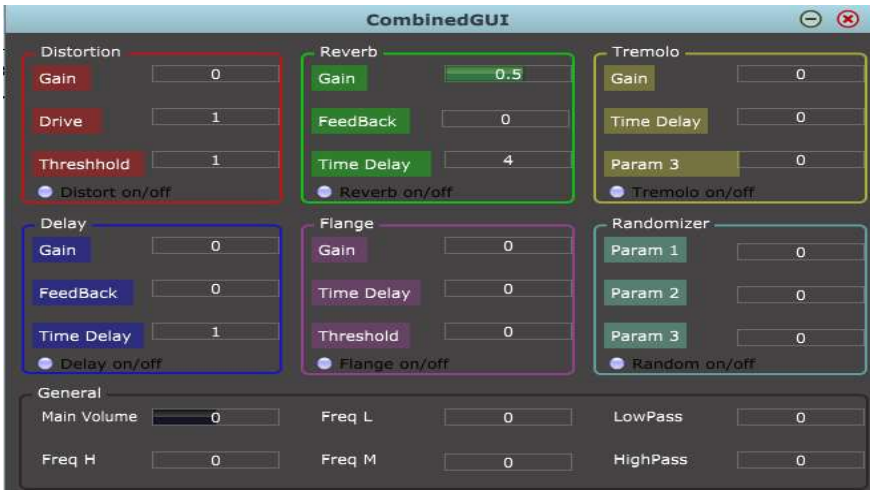
RUTGERS
School of Engineering

Real-Time VST Guitar Effects Processor



By: Henry Chen, Derrick Louie

Advisor: Dr. Sophocles J. Orfanidis

Table of Contents

I. Abstract- pg 3

II. Introduction/Overview- pg 5

III. Approach/Methods/Results- pg 7

A. Design Phase I- Matlab/Simulink Research and Implementation- pg 7
B. Design Phase II- VST (C++) Final Implementation- pg 20
C. Usage (GUI)- pg 26
D. Results/Examples (Signal Plots)- pg 27

IV. Cost/Sustainability Analysis- pg 29

V. Summary- pg 31

VI. Bibliography/Works Cited- pg 32

VII. Special Acknowledgement- pg 33

VIII. Appendix (C++ Codes)- pg 34

Abstract

In the music industry, digital guitar processors had long been a stable of both professional and studio musicians. The advancements of Digital Effects continue to revolutionize music by enabling portability without sacrificing sound. Therefore, the goal of our processor is to efficiently address the needs of traveling professional and studio musicians who constantly seek to perform their jobs on the fly

In Digital Signal Processing, we sample continuous-time signals, turning them into discrete-time signals. After the discrete-time signal had been changed with respective parameters, it is then converted back to continuous –time as the output. This process is carried out by the computer soundcard. Since each computer is different in specifications, our program is developed to successfully interact with every respective soundcards. By utilizing the VST-Host, we can interact with the soundcard by establishing a signal sampling rate and an audio buffer size. For the accuracy purpose of our project, we utilized the smallest sampling rate possible at 44100Hz. After undergoing testing, we had concluded that the buffer size of 448 samples worked best with most computers. The bulk of the Discrete –Time Signal Processing is carried out utilizing VST-plugins through interfaces designed with the JUCE wrapper, in which we programmed our guitar effects and Audio GUI. The guitar effects loop is then put together as a signal block (.dll file) with the VST Plug-in host, which connects the computer input (Guitar) to the effects and from the effects to the output (Speakers).

The guitar effects that we designed are Non-Linear Effects, Delay-based Effects, and Modulation effects. Controlling the overall volume is a buffer gain control that precedes the sound processing blocks. Each effect also has their corresponding buffer gain controls to further enhance their capabilities. Non-Linear Effects create a saturated signal commonly known as distortion. Between the fuzz and overdriving algorithms, we selected the fuzz algorithm due to the fact that it was easier to control and did not require a unique preamp or sound filter. The Delay-based algorithms include Echo and Reverb. Echo and Reverb differs in that the Echo delay buffer is fed directly into the processing block output while the Reverb delay buffer back into the input of the processing block and then copied again through the signal block. To further enhance the effect, we utilized 4 separate pointers in the delay buffer to be fed back into the processing block input and thus increase the gain.

With these traits together, the guitar output sounds like it is bouncing off a wall. The Modulation effects include Flanger and tremolo. The tremolo effect utilizes varying amplitude modulation, which creates a vibrato sound. Flanger is also related to Delay-based effects in that it uses a delay buffer fed back into the output of the processing box. The driving factor of Flanger is phase modulation in that the frequency of the delay buffer is continuously changing and thus the output combines different signals at different frequencies.

To further improve the flexibility of the effects, a parametric equalizer is needed to filter out and cancel out specific sound frequencies. This can allow the inclusion of other distortion effects such as Overdrive, which requires a pre-amp to boost specific frequencies. Furthermore to improve memory usage, the delay-based effects will be combined as one effect, exclusively using pointers to carry out the different effects. Above all, the goal of the project is to address the needs of both professional and studio musicians who are constantly on the fly. By ensuring compatibility on every Windows computer, anyone can use this processor without the need to purchase external hardware or 32-bit compatible components.

Introduction/Overview

The sole purpose of the project is to design a real-time processor that will emulate a series of guitar effects compatible on any Windows or Mac computers without external hardware. From the analog perspective, traveling musicians carry multiple guitar effect pedals during music tours and shows as seen from the above figure. As time accumulates, the expense needed to replace components gradually builds up. With the advent of multiple effects available in a software, this much more efficient in sustain the same, if not better sound quality as the product of analog signal processing devices. To better design the project; the design phase was split into two sections. Matlab and Simulink were used for fundamental research/development, and VST was used for final software development.

Matlab and Simulink helped us create and better understand the fundamentals of multiple guitar effects through the availability of specialized functions. However, the main drawback of utilizing Matlab is the lack of Real-Time Processing Capability. In each of the guitar effects created on Matlab, a sample track is used in order to sample any input signal. Through the use of built-in functions, we can better control the parameters of sound processing and accumulate understanding of remaking similar, if not exact effects for the final product. Utilizing Simulink along the way provides actual real-time processing through the use of actual Matlab functions programmed in custom blocks. However, Simulink poses another problem in the extent of soundcard interaction and buffer parameters. After undergoing various testing, Matlab's default settings impacts Simulink real-time simulations by creating a 4-second delay. This is most likely explained in that Matlab's default settings utilize a buffer size of above 448 samples at a sampling rate of over 44100Hz. To further eliminate the delay problems, we decided that it's best if we developed our final project utilizing actual Real-Time Processing Software Platforms.

In the initial stages of the project, we considered developing an Android or Apple IOS guitar processing app. However due to Android's limitations on sound processing and lack of experience in IOS app development, we decided upon Steinberger's VST with the C++ Language under the advice of a friend who previously developed a synthesizer under the same platform. As VST provides 3rd Party Developers with free development software/wrappers, the sound processing effects can easily be developed at no cost. On this

platform, we can customize the buffer size parameters to depend on each individual effect and ensure compatibility on every computer. After undergoing extensive testing, we had settled that the buffer size is best at 448 samples for guitar inputs. Increasing this buffer size will cause delays as with Simulink's default settings. While VST only allows a minimum sampling rate ($f_{s)}$ of 44100Hz, it was originally desired that we set a sampling rate (f_s) of 32000Hz. This rate corresponds with the algorithm of utilizing a Nyquist Sampling Rate in that a computer input signal bandwidth (f_{max}) is usually 16000Hz and lower. [1] With these parameters set, the rest of the software development algorithm revolved around interactions with the sound card, where the continuous guitar signal is converted to discrete-time for processing, and vice-versa for output. The next section of the report will highlight the research and development of algorithms to efficiently carry out the many guitar effects.

Approach/Methods/Results

Design Phase I- Matlab/Simulink Research/Development:

Non-Linear Effect(s):

The first and foremost guitar effect that changed the sound of music, especially in Rock Music is the distortion guitar effect. Distortion comes through the saturation of the input guitar signal either by an absolute value or by saturating a frequency range to shape the distortion sound quality. For the purpose of time, we have researched the two most used distortion effect algorithms utilized in both analog and digital signal processing.

In the days when analog electronics shaped the music industry, the fundamental effect producing the distortion sound is the Fuzz effect. As a matter of a fact, it is not necessarily needed to have a separate analog component to cause the Fuzz effect. Musicians often unintentionally produce the effect through the use of weak speakers on the amplifiers. Once a speaker reaches its maximum output frequency, the saturated output sound is technically a fuzz effect. Unfortunately since one of our computers has an extremely weak speaker, many of our distortion effect algorithm tests failed as the output signal was too great for the speaker to handle and thus incorrectly saturated the processed signal. In the extent of separate components with a tube-amplifer, the Fuzz effect is often produced using a “Hard-Clipping” algorithm produced by a series of transistors and diodes. [2] In other words, the entire signal is boosted by an absolute level before outputting through the amplifier. “Hard-Clipping” differs from “Soft-Clipping” in the extent that the sine-wave saturation transition point is more abrupt. In all, this is the most fundamental

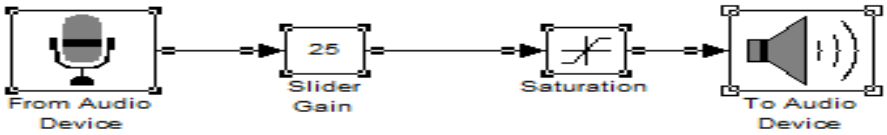
method in creating distortion. The effect of “Soft-Clipping” is on the other hand implemented in the Overdrive Effect described below. The difference is that Overdrive is not naturally produced due to the accidental saturation of poorly designed speakers or input devices.

Most modern amplifiers today, especially Solid-State and Tube-Amps with Digital Effects now utilize the Overdrive Effect to bring about distortion. Overdrive works in the same manner of Fuzz in the extent of boosting signal by absolute values. However, the trait that distinct Overdrive from Fuzz is the use of a Pre-Amp unit used to stop or boost specific frequencies. In our research and testing, we had settled that 7 kHz is the ideal center boost frequency for an electric guitar signal while 3 kHz is the ideal center boost frequency for Acoustic-Electric Guitars with Piezo pickups. [3] In doing so, we had implement Dr. Orfanidis’ Parametric Equalizer Matlab Algorithm and applied it onto a custom block for Simulink. Unfortunately due to time constraints, this effect was not implemented on VST with the final product. This effect will be further expanded as a separate effect in this report. As a result of boosting a specific frequency, a more-defined distortion output is produced as soft-clipping occurs when the signal amplitude exceeds the fixed maximum threshold value. Our group has tested a couple of overdrive algorithms and experimented with different ways to boost specific frequencies. One of the codes to adjust the tone utilizing a Butterworth Filter to boost the frequencies is show below is shown below:

```
>>bandNormalized= erf(inputSample*Gain); % Linearize the Sine Wave Amplitude to vary (max) >>%between -1 and 1
>>
>>[Z,P,K]=butter(1,7000/(fs/2),'high'); %Boosts 7 kHz and above for Electric >>%Guitar
>>[B,A]=zp2tf(Z,P,K);
>>
>>outputSample= filter(B,A,bandNormalized)
```

For ease of use and control, the Fuzz Algorithm was selected for the project. Moreover due to time constraints, we found a challenge to translate the Matlab filtering algorithms into C++. Our group had first implemented the Fuzz effect on Simulink, where

we simply took the entire input and directly saturated the signal by an absolution amplitude value before outputting onto the speakers.



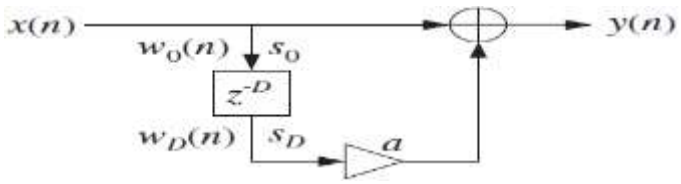
Through the use of the hard-clipping algorithm, we have developed parameter that better controls the effect. Based on the Simulink setup above, the gain is used to set the threshold amplitude range. The higher the gain, the higher the processed signal sustaining effect. This increased gain is then fed onto the saturation section, where hard-clipping occurs when the input amplitude exceeds the threshold and is normalized to the maximum amplitude value and thus "Fuzzing" up the output signal. [4] The Matlab code below outlines this algorithm:

```
>>% -1<amplitudeRange<1 => Linearize the Sine Wave Amplitude to vary (max)
>>%between -1 and 1
>>
>>amplitudeBoost = 2* amplitudeRange /(1- amplitudeRange);
>>out = (1+ amplitudeBoost)*(inputSample)./(1+ >>amplitudeBoost*abs(inputSample));
```

Delay Effects:

The most fundamental of the delay effects is the Echo effect. Depending on the gain and the rate of delay, most guitar Echo effects are described as the original signal followed

by a repeated time-delayed signal. What actually happens is that the original input sound buffer is added with a copied delay sound buffer and thus putting output multiple signals (two signals if gain is not high) with a delay signal. This is often described as a comb-filter effect from a plot perspective in discrete time, this relationship is exhibited through the relationship y(n)= x(n)+ax(n-D). The value "D" is the "round-trip travel time" or the time it takes for the delay to come around the delay loop as depicted in the block diagram below. The value "a" serves very much like a gain in that it's the coefficient that measures the reflection and propagation losses. In this scenario, the coefficient "a" is always less than or equal to 1 and greater than zero. The transfer equation in Z-domain is then given as $H(z)= 1+az^{-D}$ and the signal block algorithm is depicted as:



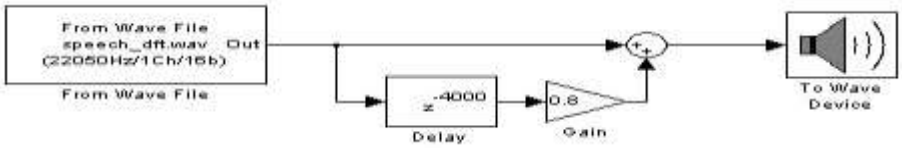
[1]

The Echo effect, much like the other delay effects proved to be a challenge to implement on both Matlab/Simulink and VST. In Matlab/Simulink, the default buffer always got in the way of real-time simulations upon implementing custom blocks on Simulink. Since every real-time implementation required a for-loop to process the delay buffer each sample, it crashed Simulink when this for-loop got in the way of Matlab's default buffer. Since loops could not be executed, we have tried utilizing a Comb-Filtering method through the use of the Echo transfer equation. This is our Comb-Filter Simulink Code segment below:

```
>>combNum=[0,zeros(1,40),1];
>>combDen=[1,zeros(1,40),-0.8];
>>output=filter(combNum,combDen,input);
>>output= output+input*0.9; %Output high gain normalized to 1
```

Unfortunately, the tests only gave audible results when the delay factor was set to less than 10 milliseconds and under. In the end, only sample-by-sample testing could be done

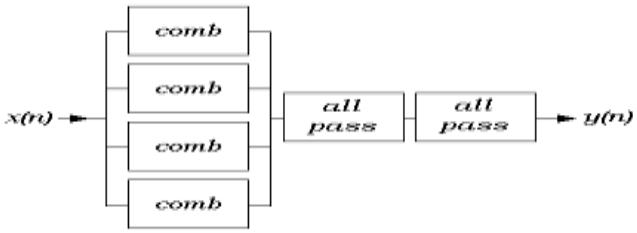
with Matlab for testing the algorithms. Since a custom block could not be implemented in Simulink, our group tried alternative to emulate the existing Simulink flow diagram below:



Obviously with this high of a "D" value, the Echo is barely audible as it corresponds to an actual delay time of less than a second. When our group tested through replacing the Wave File with a Guitar Input and the Wave Device with a Speaker, the effect that resulted was more of a "chopping block" with no audible sound of the guitar signal. This is most likely due to the fact that the default Simulink buffer size exceeds 448 samples and thus utilizes too much memory in the processing. After testing, we have concluded with a threshold range of 2 seconds and under for the Echo effect with a gain of no more than an absolute value of 1. For our VST C++, we decided to utilize a different algorithm whereby taking divisions of the sampling rate to determine the delay time. This method is further expanded later on in the report.

The Reverb algorithm is very much like the Echo algorithm. The difference is that the Reverb algorithm utilizes multiple parallel Echo effects fed back into the input instead of the output. Instead of hearing the original signal as a result of the delay, the algorithm results with a sound "bouncing off the wall" effect. In a real-world setting, sound is

obviously shaped by its environment. Having a Reverberating setting may not be ideal for a recording studio as this would increase external unwanted Echo in recording music. On the other hand in a church or concert hall, the design is opposite that of a recording studio in its purpose to Reverberation to strengthen the warmth and amplitude of natural sound. For this reason, our group has decided upon an algorithm that will amplify all frequencies with flat magnitude responses. [1] Therefore, the Schroeder Reverberator was selected as the algorithm to carry out the effect, as shown below.



[6]

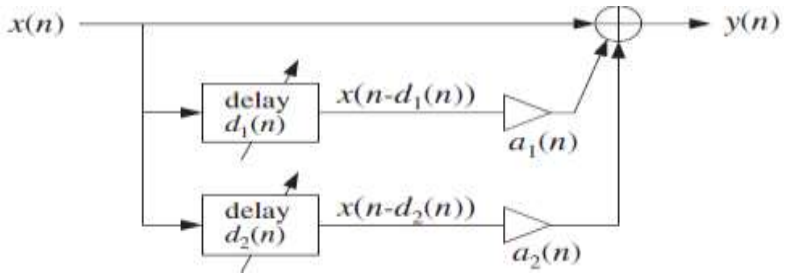
Note that the "comb" filters in the diagram actual represent parallel "Echo" delay effects. The transfer equation for the allpass Reverberator is $H(z)= (-a+z^{-D})/(1-az^{-D})$. To ensure that the effect does not come back as a convoluted Echo effect, the delay factor for each of the parallel Echoes fed back into the input has threshold values of no more than 500-700ms. To enhance the effect, a higher gain has to be set to ensure that the sound exponentially decays and up to four parallel buffers are implemented with the same delay frequency. For these reasons, the Reverb effect algorithm challenging to test out as it relied on one's interpretation of recognizing any outstanding sound Reverberation. In Matlab, we implemented the following filtering algorithm:

```
>>samples= ceil(100*exp(-3));
>>revNum= [0,zeros(1,samples),1];
>>revDen= [1,zeros(1,samples),-0.8];
>>outputSignal= filter(revNum, revDum, inputSignal);
```

Unfortunately due to Matlab's limitations in soundcard interaction, the algorithms that we utilized for Matlab did not produce audible Reverberation. For obvious reasons due to

Simulink's fixed buffer, Reverb did not work on Simulink much like the other delay effects that utilized loops to sample input. For our final development on VST, we took advantage of C++'s pointer capabilities in maximizing efficiency for our Reverb algorithm.

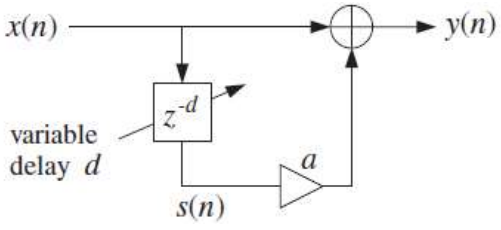
The Chorus Effect is a reverse Reverb effect. In other words, the effect produced resembles that of multiple sound sources or in this case multiple guitars playing without signal canceled out. In doing so, the delay has to be at most 500ms and to make the sound more pronounced, multiple parallel buffers are used much like Reverb. However unlike Matlab, the output of the four delay buffers are fed into output rather than fed back into the input. This ultimately eliminates the "bouncing off the wall" effect as that of a Reverb effect. The Chorus Effect is defined by the equation d(n)= D(0.5+v(n)), v symbolizing delay variation. Delay variation in the sense similar to that of the Flanger effect (described below) helps output simultaneous sounds without letting the synchronizing of similar frequencies cancel the output signals out. Unfortunately due to the limitations of our speakers, differentiating the Chorus and Reverb effects in our trials proved nearly impossible as both effects used similar algorithms. In the end, our group decided to exclude the Chorus Effect even if we had the correct algorithms for both Matlab and C++. The similarities and differences between the Chorus Effect as well as the Echo and Reverb effects can be depicted in the block diagram below:



[1]

Modulation Effects:

The Flanger effect combines the Echo effect with phase/frequency modulation. In short, the algorithm combines a delay buffer and a LFO (Low Frequency Oscillator). [5] This effect originated from musicians who tried to emulate the effect by periodically slowing down the play speed of a recording tape. To simplify it, the delay buffer is copied from the original buffer much like the Echo effect. As the input is continuous, each sample in the delay buffer is delayed with a varying frequency and thus outputting a 'sweeping' or 'jet stream noise.' Varying the time delay causes the delay signal to sweep up and down the frequency spectrum. Therefore, the output is usually described as a swept comb filter from a plotting perspective. The challenge with create this effect is finding the best method to carry out the varying of the delay signal frequency. Depicted below is the block diagram for Flanger:



[1]

With this in mind, delay varying frequency sometimes creates a non-integer delay value. In Matlab, we ultimately solved the problem of non-integer delay values by utilizing Linear Interpolation. With this algorithm, we take a delay value and move it in fractional segments in respect to its original delay buffer. The Flanger effect can be represented with the equation y(n)= x(n)-ax(n-d(n)), which explains that the delay buffer frequency is

continuously changing as time goes on. Rather than utilizing a set of threshold frequency numbers for adjusting the frequency, the parameters are adjusted based on a range of frequency; a maximum number and a minimum number, depending on the user's preference.

In Matlab, we used this sample by sample algorithm for each input wav file:

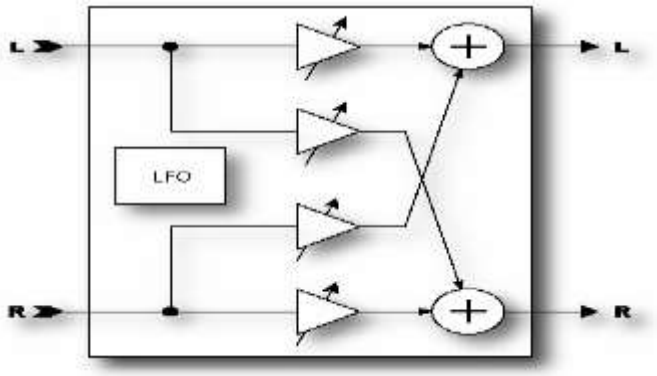
```
>>for x = (maxDelay+1):1024, %Extracts from default buffer size of 1024 samples
   >>curentSinVal=abs(sinRef(x));    %abs of current sin val 0-1
   >>currentDelay=ceil(curSinVal*maxDelay);  % generate delay from 1-max_samp_delay
   >> y(x) = (amplitude*inputSignal(x)) + amplitude*(inputSignal(x-currentDelay));   % add
  >>%delayed sample
>>end
```

Notice that Matlab's default buffer size (at least based on our tests) is around 1024 samples. As said before, this buffer size takes up excessive memory and for Matlab sample by sample processing, running the algorithm takes more time than running with a small audio buffer size.

In addition to Phase/Frequency modulation effects, Amplitude modulation effects also have a massive impact in guitar effects. The most famous of amplitude modulation is the Wah-Wah effect. In recent years, due to the varying nature of the Wah-Wah effect with various analog effects equipment, people have opted to use a varying bandpass filter algorithm over an amplitude modulation algorithm. With the bandpass algorithm, guitar players can channel focus the center frequency on either the bass end (3 kHz and under) or the treble end (7 kHz and above). Centering on the bass will channel out the upper harmonics, while center towards the treble will omit the low-end growl and making a

guitar note sound like a “shrieking cry” or “Wah-Wah” sound. Since our project does not utilize analog equipment to manually control without the use of more than two hands, our amplitude modulation effects are fixed on set ranges of frequencies that the amplitude varies. Further developments may incorporate the use of sliders as part of the GUI, but that will defeat the purpose as guitar players need both of their hands to play their instruments while utilizing pedals to manually modulate their signal amplitudes.

For the final development of the project, the Wah-Wah effect was ultimately dropped along with the fact that we did not develop and control slider that can be adjusted on the fly. In place for an amplitude modulation effect, our group decided to only develop the tremolo effect. The background of this effect stems from musicians who utilize the vibrato technique (bending strings) to continuously vary the sound pitch or through the use of an electric guitar tremolo tailpiece in which either stretches or loosen the strings. The ultimate goal is to design an automatic algorithm which adjusts the range of amplitude magnitude (absolute value) modulation; the longer the range, the slower the amplitude modulation. The equation describing the Tremolo effect is $y(n)= (1+\alpha m(n)):x(n))$. α controls the percentage of modulation as m is the LFO. As our group designed the effect, we realized that the Wah-Wah effect could definitely be incorporated should we have carried over the parametric equalizer and incorporated a separate analog component. But since one of the goals of our project is to minimize external hardware, we decided that an automatic tremolo was satisfying as there are a split of musicians using automatic tremolo and manual adjusting tremolo. Below is the block diagram for the Tremolo effect:



[7]

The Tremolo effect depicted above comes from two inputs and channels into two speakers, which is different from the effect in our project as the input is a mono input. For the final development on VST, we again took advantage of using pointers to allocate memory and vary the real-time amplitude of the guitar input signals.

Parametric Equalizer Effect:

Utilizing Dr. Orfanidis' Parametric Equalizer Algorithm, our group successfully built a working Equalizer for both Matlab and Simulink. The motivation for creating this effect is to address the guitarists who emphasize more on the clean sound. As friends of Jazz/Blues musicians, the primary feedback that we had received for our project was that it was too focused on creating "dirty" or "distorted" outputs more slated for Rock and other heavy music. With only a master gain to control the original buffer, one can only adjust the overall volume but not specific frequencies. As stated before, electric guitar music of both "clean" and "dirty" sound best when boosted at 3 kHz and 7 kHz. [3] The frequencies at 1 kHz and lower should be blocked out to prevent negative feedback. From this perspective, a Parametric Equalizer comes in handy. Parametric Equalizer is more advantageous than

Graphical Equalizers when it comes to recording and live adjustments as the frequencies are not predefined by design. In essence, the effect is the linear combination of a notch filter (bandstop) and a peaking filter (bandpass). Since the frequency adjustment algorithm is in series (versus parallel on Graphical Equalizers), the sound is a more pronounced as the adjustment of a previous frequency also affects the latter frequencies. For the purpose of affecting three frequency ranges, our group designed a 3-band Parametric Equalizer.

The transfer equation as described in Dr. Orfanidis' algorithm for this effect is:

$$H(z) = G_0 H_{\text{notch}}(z) + G H_{\text{peak}}(z)$$ [1]

$$H(z) = \frac{\left(\frac{G_0 + G\beta}{1+\beta}\right) - 2\left(\frac{G_0 \cos\omega_0}{1+\beta}\right) z^{-1} + \left(\frac{G_0 - G\beta}{1+\beta}\right) z^{-2}}{1 - 2\left(\frac{\cos\omega_0}{1+\beta}\right) z^{-1} + \left(\frac{1-\beta}{1+\beta}\right) z^{-2}}$$ [1]

One of the challenges that prevented us from making this effect work at first was misunderstanding that the bandwidth size Δω directly impacts the reference gain for each of the three bands. When our group tested for the first month, we were only able to stop frequencies and pass frequencies as soft as 3 db. In the end, we found that are bandwidth size Δω was the problem in that it was not wide enough decrease the gain lower than 3 db. The most misunderstood part in our testing was the fact that the unit bandwidth and center frequency is rad/sample. Through the use of absolute values, the output was the result of aliasing in that it was passed as completely different center frequency and bandwidth values on the unit circle. Another mistake was putting absolute values for reference and boost/notch gain instead of dB values, which resulted in distorted outputs. As a group, we regret not being able to translate Dr. Orfanidis' 'Parmeq.m' function from Matlab into C++ and thus preventing us from including the Parametric Equalizer as part of our final VST development. This outcome impacted our project as it does not provide sufficient support to enhance "clean" guitar sounds for softer music like Jazz and Blues. As a matter of fact, including the Parametric Equalizer is the priority is the future development of our Real-Time Guitar Effects Processor. The Simulink code below combines Dr. Orfanidis'

Parametric Equalizer Algorithm with the three band filters in series; this code can be modified for sample by sample use of Matlab:

```
%Sampling/Filtering Parameters
fs= 44100       $Sampling Frequency
bw= 600*(2*pi)/fs               %Bandwidth (rad/sample)
cF= (2*pi)/fs             %Center Frequency  Conversion (rad/sample)

%Reference Gains in dB
G1= 10^(10/20)
G2= 10^(9/20)
G3= 10^(10/20)

%Dr. Orfanidis' Parametric Equalizer Algorithm
[b, a] = parmeq(1, 0, 1/sqrt(2), 600*cF, bw);   %coefficients for Bass Cut
[d, c] = parmeq(1, G2, 1/sqrt(2), 3000*cF, bw);   %coefficients for Mid Boost
[f, e] = parmeq(1, G3, 1/sqrt(2), 7000*cF, bw); %coefficients for Treble %Boost

%Filters in Series
newm1 = filter(b, a, x); %Bass output filter
newm2 = filter(d, c, newm1); %Mid output filter
newm3 = filter(f, e, newm2);           %Treble Output filter
```

[1]- Part of the code is Dr. Orfanidis' Parametric Equalizer Algorithm

Design Phase II- VST (C++) Final Implementation:

VST

In order to convert the Matlab code discussed in the previous sections into a workable application, we could not use the conventional programs we had. The default programs from Windows and Matlab would not suffice digital signal processing (DSP). Instead, we started to research various solutions to control the data from the soundcard. However, most of the searches resulted in software ranging in the hundreds of dollars. We ideally wanted to find a free open source solution and we found it in the form of Virtual Studio Technology (VST).

VST is a software interface created by the musical software company Steinberg that interacts with the soundcard. It was one of the first options we found that was feasible. VST is usually used as plugins for audio processing or synthesized instruments as a digital replacement in lieu of normal, conventional hardware. Immediately when we discovered this technology, we knew this would be a primary candidate to use. This interface supports digital signal processing and many digital activities that would normally be done in hardware. It is able to take the digital input from the sound card and translate into a buffer that is set by the user. Furthermore, by using the methods and functions provided the interface, we can distort the original sound, making it sound different. This creates the digital sound effects that we wanted to replicate from hardware and the Matlab simulations. This code compiles into a .dll file, so we still needed to utilize accessible software that can run these files.

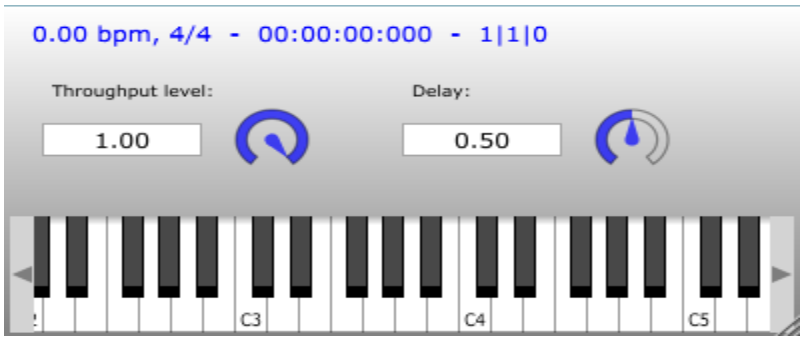


Figure : A demonstration of how a plugin functions. As shown it is a GUI that controls the backend of the effect. This particular plugin has midi inputs controlled by the keyboard. Various parameters are controlled by the knobs or text fields.

JUCE

While looking VST up, we found much other commercial software that utilizes its functionalities. However, most of the software were not cheap and not adequate for a design implementation. However, we did find one program that would both help the writing of the VST code and running the .dll files. Jules' Utility Class Extensions (JUCE) is a free open source C++ class library that was able to create plugins from VST functionalities. It also provided helpful tools in the form of a fully documented API. It also comes with a GUI to run the plugins we would be making and organize them in an orderly format. It allowed us to change what inputs we would be receiving and where the outputs would go. We found that JUCE does not currently support VST 3. In order to work in conjunction with JUCE, we used VST 2.4 as it was difficult finding a good plugin software that would be as accessible as JUCE. Even though it was an earlier version, we did not have any trouble with using the older version versus the new SDK.

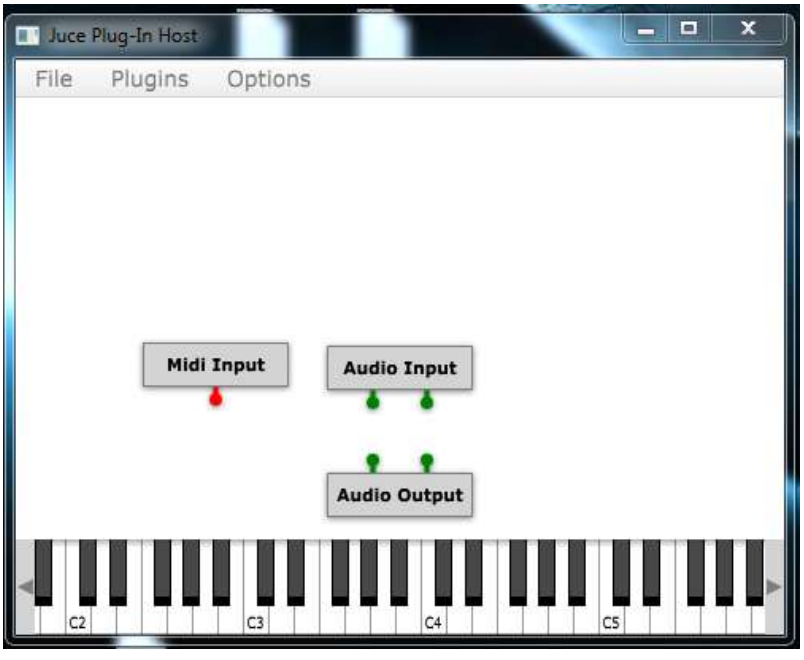


Figure: How the Juce Plugin Host works

Note: As shown there are midi inputs with the pitch corresponding with the keyboard key held. As we are not using synthesizers, we are ignoring this part of the host. Rather, we are interested in the audio input and outputs. Since out guitar input comes in a mono format, we decided to create plugins that would return the original input on the left output channel and return the effect on the right channel. In the input shown, the mono input is the left audio input. As our output is can be in stereo, we can input the output into both channels. All of the Juce plugins we developed are stereo input and output.

Many of our early problems originated with our lack of understanding of project building and with both JUCE and VST. However, once we learned from tutorials like RedWoodAudio, we were able to start implementing the Matlab functions. [10] Once we were able to use JUCE's editor to create a project, we started to develop a standard for the buffer, sample rate, and input, we settled on a standard 44100Hz which is the standard for audio recording. It should be noted that the plugin host would not work if the sample rates for the input and output are not synchronized. The plugin host would pick up an error. It is crucial that the recording device and output device are sampling the same rate. Furthermore, there is the buffer size that helps determine how much of the input signal is stored. Since we did not want a large delay from the input signal to the output sound, we kept it to a minimal 10 ms or 448 samples. It creates a delay that is audible, but not enough to make a difference to the sound quality. It also gives us a large enough buffer so that we can perform our processing on it.

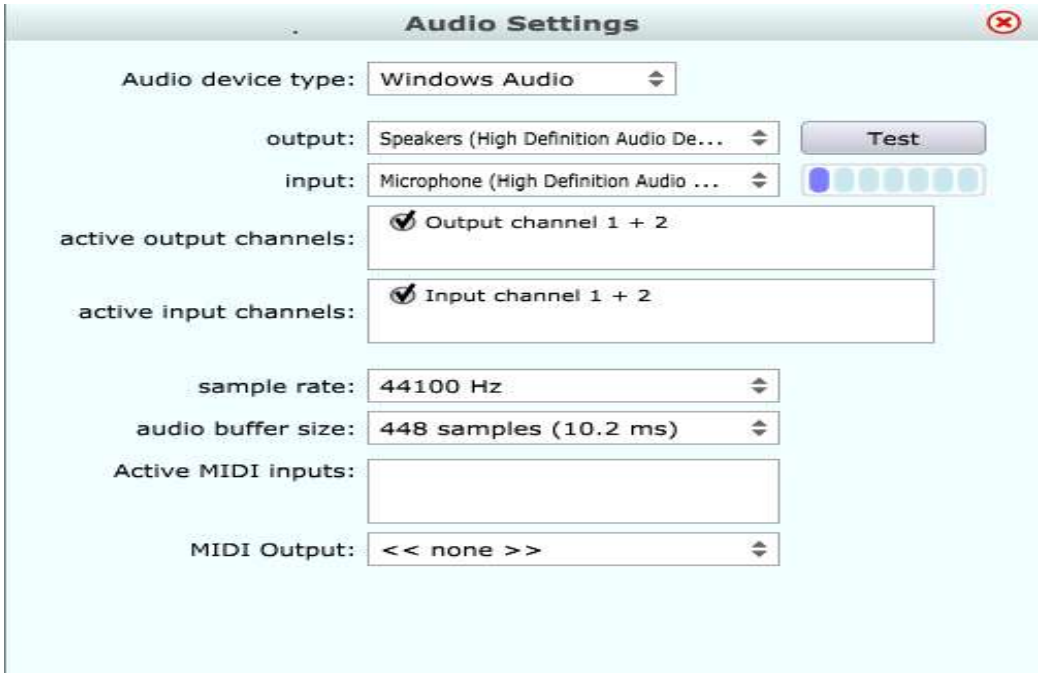


Figure: The input and output can be changed based on the desired devices and is not randomly chosen. The sample rate is based off the input of the soundcard. This is not easily changed and also varies per computer. The buffer size has a range from 3 ms to 46ms.

The JUCE project relies on the VST software to control the soundcard. It then it uses the user created code and its own library modules to create a VST plugin that can be accessible by any plugin host, commercial or open source. The user created code is given in two forms, the editor and the processor, named as PluginEditor and PluginProcessor. The editor handles the user interface, and acts as the IO controller between a GUI and the processor creating the effect. Changing a value in the GUI would change the value of a variable in the algorithm, thereby allowing us to manually change the parameters of the code.

Distortion:

The first effect we attempted to replicate was a simple distortion. This required only real time processing and no memory involved. The Matlab equivalent was simple and easy to understand. However, we were novices to JUCE and VST, we ran into some troubleshooting issues. There would be noise when we ran the first iteration of the code. Later, we found out that this was due to the fact that code handles the buffer in samples, and not the entire buffer altogether. This led to one sample in the buffer being distorted. Once we solved this, we were able to get the results we wanted. At this time, we wanted to send code to the entire group for transparency. We quickly found out that the project properties do not easily transfer as the JUCE and VST libraries are held in different locations at different PCs. One solution is to coordinate a folder that would be in the same location in all PCs to mitigate this. Furthermore, the plugin's quality varied from computer to computer. We surmise this is due to the effects of the soundcard. Even without effects, one of the PC's we were using had unintentional distorted output.

Once we created our simple effect, we created the GUI to be able to change the parameters of the effects. There were three values accessible to the user: gain, drive, and bandwidth. As this was for internal testing, the values have not been labeled, but are noted from top to bottom as Gain, drive, and bandwidth.

We created a standard that allowed us to be organized with our development and gave us small goals to strive for. For each effect, we followed the same procedure as distortion:

- Create a working effect.
- Create a user-controlled version.
- Create a final product

Delay

This effect gave us introduction to the AudioSampleBuffer class inside JUCE. We created an instantiation of this class to store the current values of the output buffer. To delay the effect, we added the contents of the stored buffer to the output buffer at a time delay, delaying the output by a marginal amount.

Our user defined variables are feedback and time delay. The time delay the buffer by an absolute value in reference to the sample rate. The feedback changes the gain of the storage buffer added back into the output. As a user may want a repeating loop, we settled for values between 0 and 1 to prevent a potential overflow. It is noted that a gain of 1 may also cause overflow and is not recommended.

Reverb

This effect was remarkably similar to the delay effect previously. Several of the variables were changed, but the core still remained the same. However, this required

several pointers to different parts of the storage buffer. Eventually, we created an effect that had variable time delay, which had different parameters due to the nature of controlling multiple pointers and handling out of bounds exceptions.

Flange

When we created this effect it had the resulted in a fluctuating digitized signal, whose output depended on the amplitude of the input. Later we discovered that this was caused by feedback going back into the storage buffer. This distorted the storage buffer immensely. We started calling this a possible pipe organ digitizer as it resembled the large instrument. Instead, we changed how the buffer would be changed and created a semi-working effect. With more time and resources, this would be fixed to a better quality.

Tremolo

The tremolo was added as a replacement for the Wah-Wah effect which would have taken a long time to complete. Rather, we varied the gain of the input to a variable amount that can be changeable by the user. When turned on, this would increase and decrease the sound, oscillating the amplitude. This was one of the later effects that did not require the use of the buffer or pointers.

Randomizer

This was an experimental effect that can be optimized at a later time. This stemmed from the experimentation of the buffer itself and how it correlated to analog sound. Instead of outputting a similar sound, we ended up inverting the buffer so as to change the signal altogether. This did not just modulate the sound, but change the pitches as well. This effect would only be used for fun and not in any practical situation.

As we created each effect, we became more and more familiar with the code. We were able to create a new buffer as a storage for delays, Reverbs and flanges. With this new buffer, we started to experiment with the values in order to get a better sense of the utilization. The Reverb showed the usage of various pointers that can be used for a single buffer. The tremolo effect showed the direct influence on a variable scale.

Once we created the 6 effects, we put them together in series to create a final GUI. All the effects we created are functional and can be controlled by the user. We had plans to implement a parametric equalizer that did not pan out due to time constraints and issues transferring the code from Matlab.

GUI Design:

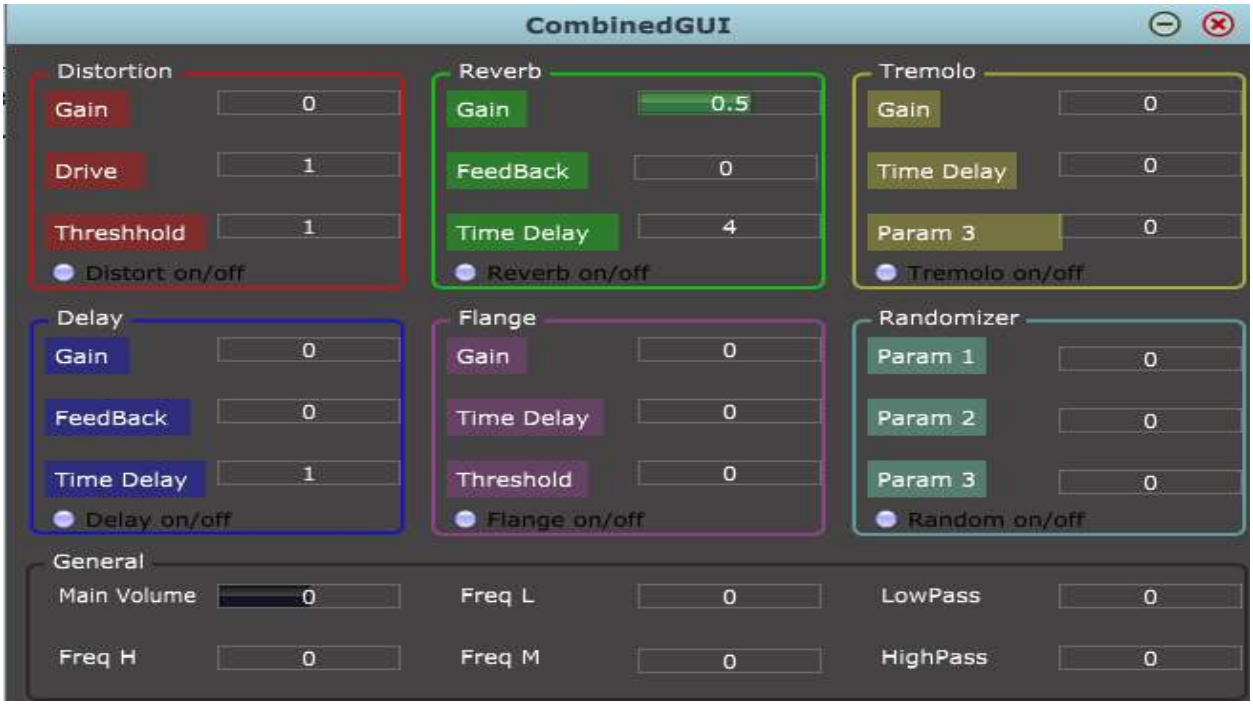
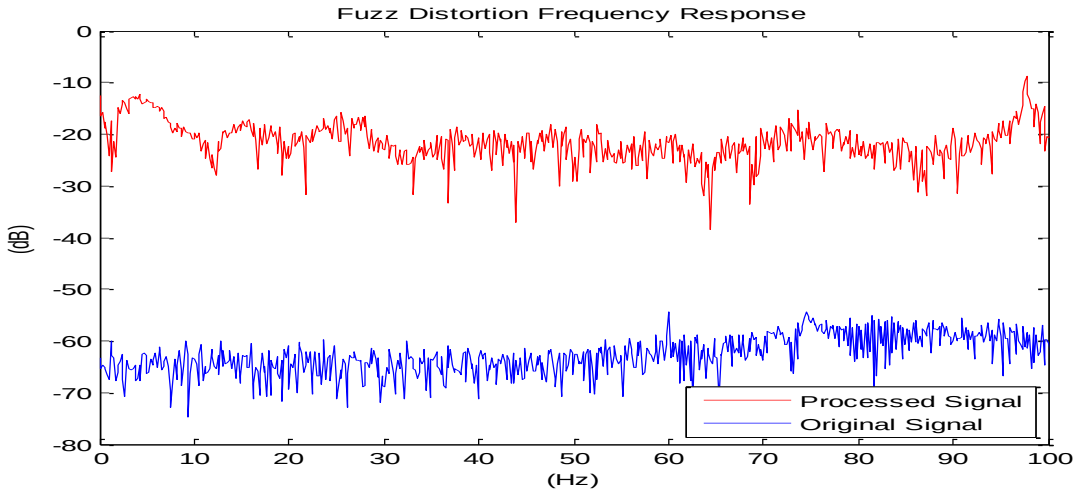


Figure : Final GUI representation

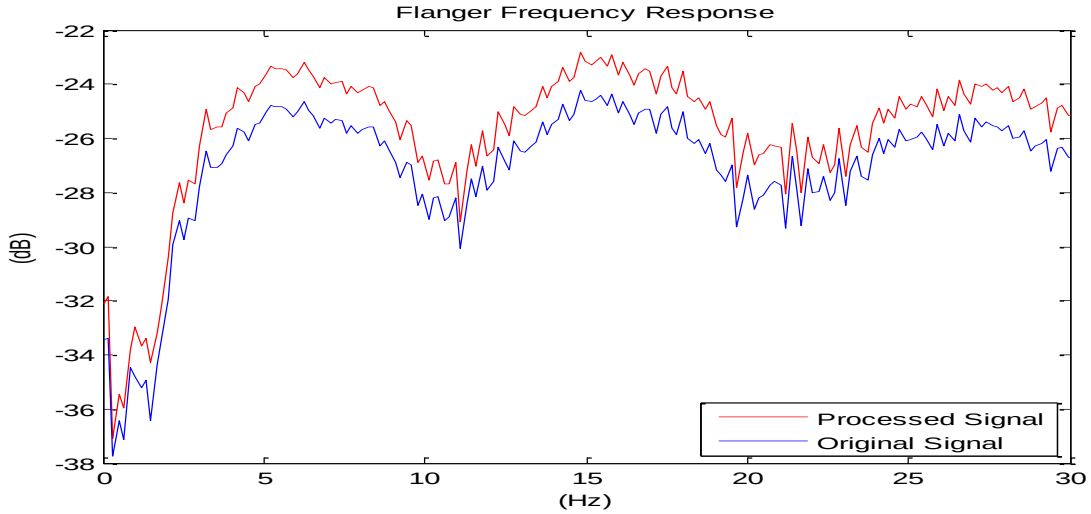
We have the ability the turn any of the effects on or off depending on the situation. Even though they are in series, this would not affect any output and would run as intended.

Examples (Signal Plots):

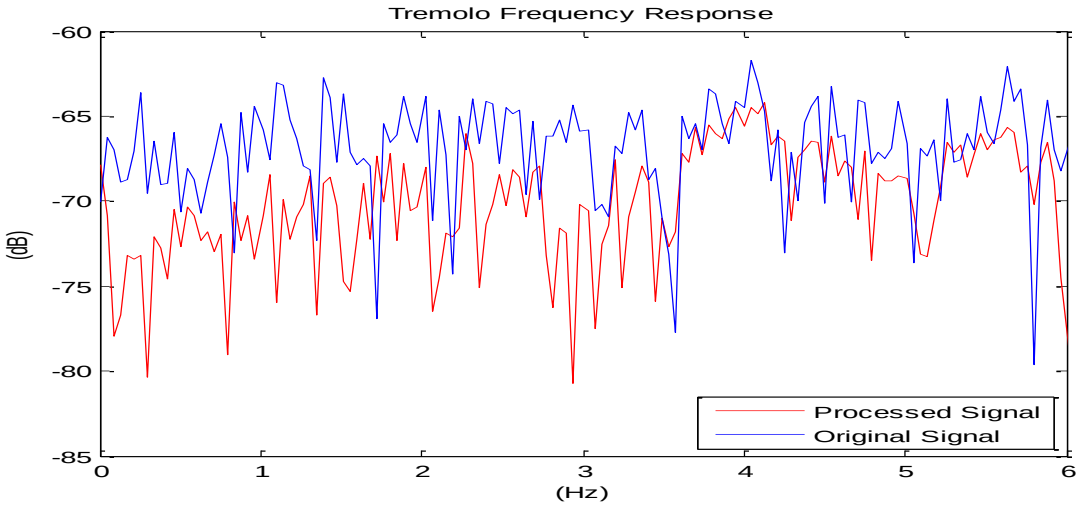
To prevent the overlapping of both the input and output (processed) signals in the plots, the frequency responses are taken with respect to dB magnitudes on a log scale. Please note that negative dB values do not mean that the signals have no sound output in absolute value.



The Fuzz Distortion algorithm directly saturates the input signal. Using the relationship between absolute value and dB magnitudes, it is shown that the input signal was saturated by an absolute value of 100.

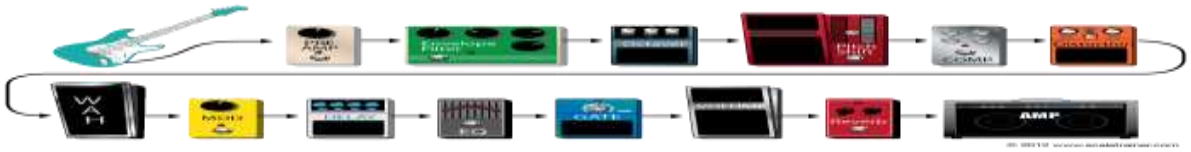


The Flanger Effect combines the Echo Effect with a Phase Modulator. By creating a phase shift, the important part of the Phase Modulation is mixing the time-shifted signal back into the original signal. By doing this, certain places where the phase modulated signal cancels with the input signal, and thus creating destructive interference between the two. This results with the 'sweep' as depicted in the plot/



The Tremolo Effect is the result of Amplitude Modulation. As the plot depicts, the output signal is a reflection of the input signal. The timing of the highest and lowest volume signals is equal with the switching between high and low occurring abruptly and thus producing the Tremolo Effect.

Cost/Sustainability Analysis



Using a solid effects unit is more cost advantageous over buying separate guitar effects in the extent of using less external equipment and thus replacing less equipment when the time comes. As a matter of fact, this project was developed for use on any Windows computer whether it is 32-bit or 64-bit. In 2012, statistics on "Laptop Mag" have shown that the average person owns a personal laptop averaging around $456. [8] At this rate, the average has gone down by $200 since 2010 and thus showing decline in usage as well as a prediction that the prices will only decrease in the coming years. It is worthwhile to mention that an average laptop, if treated properly can last beyond five years. Above all within a ten year time span, a traveling musician only needs to invest in $400 worth of equipment which covers nearly every Guitar Effect.

On the article "Best Guitar Multi-Effects Processor," it is shown that the best combined effects processor costs around $500 to $530. This means that if one were to buy a single unit alone for a single effect, it will probably average around $100 each. In light of inflation and constant upgrades, the chance that an effect needs replacement will eventually supersede the costs of replacing a computer that can last ten years. The cost analysis of using VST over purchasing separate Effect Processors is shown below.

There are many musicians that desire professional equipment without the resources or the capability to get them. This hardware can range from the hundreds to the thousands of dollars in pedal effects. Furthermore, these hardware are usually adhered to a definite hard board to prevent any modification. This makes it arduous to travel with these boards. They are bulky and can be damaged in transit. Rather, out solution is to aid these aspiring musicians and hope to inspire a new generation of musicians. Musicians are driven in their pursuit for new tastes and sounds, and our plugin interface serves only to give them the tools to create their own content.

Our project cost us no extra funds in hardware. The VST and JUCE libraries are free as open source software. In the booming computer markets, it is rarer to find those without computers than those with a PC. Our design is only limited by the quality of the soundcard, and it would still work even on older PCs.

Part Name	**Cost**
Virtual Studio Technology (Steinberg SDK)	$0
Jules Utilities Class Extensions (Raw Material Software)	$0
PC [already owned]	$0
Microsoft Visual Studio 2012 Express	$0
	Total Cost
	$0

Since, our project is aimed at aspiring musicians that may lack resources, we needed to make our design accessible to the entire public. Our solution was to create quick and efficient software with little maintenance. It is entirely possible to create more in-depth algorithms and programs with higher level software, which may be possible later in the business timeline. A possible business plan for the current program is to sell it as an application.

Our sustainability is bounded only by the needs of the population. Mass production is dependent only on a website or a developer's advertisement as we can upload and distribute our program and market it there. Compatibility with future technologies only needs a plugin host with our without JUCE in order to run. Future technologies would only make the system easier as the sound cards that are being developed overshadow the sounds cards that already exist.

Future development may include adding more algorithms and effects as well as creating a mobile application for the large mobile base.

Summary

Above all, most of our project is a success in achieving cost, algorithm, and equipment efficiency. The use of Matlab and Simulink enhanced our understanding of guitar effects and develop the effect algorithms made it easier for our group to translate logic from one programming language to another. While there were limitations in Simulink, our group further understood that in order for us to fully control out effects, we would have to utilize software that will also give us control of audio buffer parameters. This brought us to use Steinberg's VST, which avails its software to third party vendors at no charge for personalized software development. The first three months in setting up VST were the most frustrating part of the project as we had to make sure that the current version of Microsoft Visual Studio was compatible to the current version of VST. After three months, we were finally able to develop and implement the most fundamental algorithms. In the end, we also able to develop three algorithms based on the Modulation and Delay Effects in addition to the fundamental effects on the processor.

For future development, the Parametric Equalizer is the first effects under consideration. If our group had more time, the Parametric Equalizer would have been our first effect as it is the most fundamental effect in manipulating real-time sound processing. To further improve the flexibility of the effects, a parametric equalizer is needed to filter out and cancel out specific sound frequencies. This can allow the inclusion of other distortion effects such as Overdrive, which requires a pre-amp to boost specific frequencies. The equalizer can also address the needs of Jazz/Blues musicians who prefer 'clean' and 'bright' guitar sounds over distorted sounds. Furthermore to improve memory usage, the delay-based effects will be combined as one effect, exclusively using pointers to carry out the different sounds. To conclude, the experience that our group accumulated through this project further convinces us of the successful future that Digital Signal Processing brings forth in simplifying the lives of both amateur and professional musicians.

Bibliography

[1] Sophocles J. Orfanidis, (2010) "IIR Digital Filter Design," in Introduction to Signal Processing [Online]. Available: http://eceweb1.rutgers.edu/~orfanidi/intro2sp/orfanidis-i2sp.pdf
[2] Hunter, D. (2008, June 18). Effects Explained: Overdrive, Distortion, Fuzz. . Retrieved March 2, 2014, from http://www2.gibson.com/News-Lifestyle/Features/en-us/effects-explained-overdrive-di.aspx
[3] Dennis, R. (1998, November 25). Recommended Equalization Frequencies. . Retrieved March 2, 2014, from http://www.recordingeq.com/Subscribe/tip/tascam.htm
[4] Smyth, T. (2007, October 26). Delay Effects: Flanging, Phasing, Chorus, Artificial Reverb. . Retrieved March 2, 2014, from http://www.cs.sfu.ca/~tamaras/effects/effects.pdf
[5] Poongbunkor, P. (2001, May 1). DSP Audio Effects. . Retrieved March 4, 2014, from http://courses.physics.illinois.edu/phys406/Student_Projects/Spring01/PPoongbunkor/Piya_Poongbunkor_DSP.pdf
[6] Marshall, D. (2010, January 24). Digital Audio Effects. . Retrieved January 10, 2014, from http://www.cs.cf.ac.uk/Dave/CM0268/PDF/10_CM0268_Audio_FX.pdf
[7] Pan/tremolo. (2007, June 10). . Retrieved March 30, 2014, from http://wiki.fractalaudio.com/index.php?title=Pan/tremolo
[8] Piltch, A. (2012, March 13). The Average PC Laptop Cost $513 in February. . Retrieved May 1, 2014, from http://blog.laptopmag.com/the-average-pc-laptop-cost-513-in-february
[9] Best Guitar Multi-Effects Processor. (2012, November 1). . Retrieved May 1, 2014, from
[10] http://www.bestcovery.com/best-guitar-multi-effects-processor
Redwood Audio (2014) JUCE for VST Plugin Development. Retrieved January 27 , 2014 from http://www.redwoodaudio.net/Tutorials/juce_for_vst_development__intro.html
[11] Steinberg Media Technologies (2014) 3rd Party Developer for VST From http://www.steinberg.net/en/company/developer.html
[12] Raw Material Software (2014) Juce Downloads. From http://www.juce.com/downloads

Special Acknowledgement

We would like to thank you Dr. Orfanidis, for giving us your time, patience, and knowledge. As a proud team we cherished every minute in making this project successful. Your guidance inspires us not only to reach beyond our goals, but to also inspire others in partaking in such a revolutionary methods. Thank you so much for everything.

Appendix (C++ Codes)

PluginEditor.cpp

```
/*

  ==============================================================================

    This is an automatically generated GUI class created by the Introjucer!

    Be careful when adding custom code to these files, as only the code within
    the "//[xyz]" and "//[/xyz]" sections will be retained when the file is loaded
    and re-saved.

    Created with Introjucer version: 3.1.0

    ------------------------------------------------------------------------------

    The Introjucer is part of the JUCE library - "Jules' Utility Class Extensions"
    Copyright 2004-13 by Raw Material Software Ltd.

  ==============================================================================
*/

//[Headers] You can add your own extra header files here...
//[/Headers]
```

```
#include "PluginEditor.h"



//[MiscUserDefs] You can add your own user definitions and misc code here...

//[/MiscUserDefs]


//==============================================================================

CombinedGuiAudioProcessorEditor::CombinedGuiAudioProcessorEditor
(CombinedGuiAudioProcessor* ownerFilter)

   : AudioProcessorEditor(ownerFilter)

{

   addAndMakeVisible (groupComponent = new GroupComponent ("new group",

                                          "Distortion"));

   groupComponent->setColour (GroupComponent::outlineColourId, Colour (0xaaff0000));

   groupComponent->setColour (GroupComponent::textColourId, Colour (0xfff8f4f4));


   addAndMakeVisible (groupComponent2 = new GroupComponent ("new group",

                                          "Delay"));

   groupComponent2->setColour (GroupComponent::outlineColourId, Colour (0xaa0000ff));

   groupComponent2->setColour (GroupComponent::textColourId, Colours::white);


   addAndMakeVisible (groupComponent3 = new GroupComponent ("new group",

                                          "Reverb"));

   groupComponent3->setColour (GroupComponent::outlineColourId, Colour (0xaa00ff00));

   groupComponent3->setColour (GroupComponent::textColourId, Colours::white);


   addAndMakeVisible (groupComponent4 = new GroupComponent ("new group",

                                          "Flange"));
```

```
groupComponent4->setColour (GroupComponent::outlineColourId, Colour (0xaabf40bf));
groupComponent4->setColour (GroupComponent::textColourId, Colours::white);

addAndMakeVisible (groupComponent5 = new GroupComponent ("new group",
                                   "Tremolo"));
groupComponent5->setColour (GroupComponent::outlineColourId, Colour (0xaae1e13a));
groupComponent5->setColour (GroupComponent::textColourId, Colours::white);

addAndMakeVisible (groupComponent6 = new GroupComponent ("new group",
                                   "Randomizer"));
groupComponent6->setColour (GroupComponent::outlineColourId, Colours::cadetblue);
groupComponent6->setColour (GroupComponent::textColourId, Colours::white);

addAndMakeVisible (SlideDriveDistort = new Slider ("new slider"));
SlideDriveDistort->setRange (1, 100, 1);
SlideDriveDistort->setSliderStyle (Slider::LinearBar);
SlideDriveDistort->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideDriveDistort->setColour (Slider::thumbColourId, Colour (0x50ff0000));
SlideDriveDistort->setColour (Slider::rotarySliderFillColourId, Colour (0x50ff0000));
SlideDriveDistort->setColour (Slider::textBoxTextColourId, Colours::white);
SlideDriveDistort->addListener (this);

addAndMakeVisible (SlideThreshDistort = new Slider ("new slider"));
SlideThreshDistort->setRange (1, 50, 1);
SlideThreshDistort->setSliderStyle (Slider::LinearBar);
SlideThreshDistort->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideThreshDistort->setColour (Slider::thumbColourId, Colour (0x50ff0000));
SlideThreshDistort->setColour (Slider::rotarySliderFillColourId, Colour (0x50ff0000));
```

```
SlideThreshDistort->setColour (Slider::textBoxTextColourId, Colours::white);
SlideThreshDistort->setColour (Slider::textBoxHighlightColourId, Colour (0x50ff0000));
SlideThreshDistort->addListener (this);

addAndMakeVisible (SlideGainDelay = new Slider ("new slider"));
SlideGainDelay->setRange (0, 1, 0);
SlideGainDelay->setSliderStyle (Slider::LinearBar);
SlideGainDelay->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideGainDelay->setColour (Slider::thumbColourId, Colour (0x500000ff));
SlideGainDelay->setColour (Slider::textBoxTextColourId, Colours::white);
SlideGainDelay->addListener (this);

addAndMakeVisible (SlideFeedDelay = new Slider ("new slider"));
SlideFeedDelay->setRange (0, 10, 0);
SlideFeedDelay->setSliderStyle (Slider::LinearBar);
SlideFeedDelay->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideFeedDelay->setColour (Slider::thumbColourId, Colour (0x500000ff));
SlideFeedDelay->setColour (Slider::textBoxTextColourId, Colours::white);
SlideFeedDelay->addListener (this);

addAndMakeVisible (SlideTimeDelay = new Slider ("new slider"));
SlideTimeDelay->setRange (1, 10, 1);
SlideTimeDelay->setSliderStyle (Slider::LinearBar);
SlideTimeDelay->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideTimeDelay->setColour (Slider::thumbColourId, Colour (0x500000ff));
SlideTimeDelay->setColour (Slider::textBoxTextColourId, Colours::white);
SlideTimeDelay->addListener (this);
```

```
addAndMakeVisible (SlideGainReverb = new Slider ("new slider"));
SlideGainReverb->setRange (0, .8, .1);
SlideGainReverb->setSliderStyle (Slider::LinearBar);
SlideGainReverb->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideGainReverb->setColour (Slider::thumbColourId, Colour (0x5000ff00));
SlideGainReverb->setColour (Slider::textBoxTextColourId, Colours::white);
SlideGainReverb->addListener (this);
        SlideGainReverb->setValue(.5);

addAndMakeVisible (SlideFeedReverb = new Slider ("new slider"));
SlideFeedReverb->setRange (0, 10, 0);
SlideFeedReverb->setSliderStyle (Slider::LinearBar);
SlideFeedReverb->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideFeedReverb->setColour (Slider::thumbColourId, Colour (0x5000ff00));
SlideFeedReverb->setColour (Slider::textBoxTextColourId, Colours::white);
SlideFeedReverb->addListener (this);

addAndMakeVisible (SlideTimeReverb = new Slider ("new slider"));
SlideTimeReverb->setRange (4, 20, 1);
SlideTimeReverb->setSliderStyle (Slider::LinearBar);
SlideTimeReverb->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideTimeReverb->setColour (Slider::thumbColourId, Colour (0x5000ff00));
SlideTimeReverb->setColour (Slider::textBoxTextColourId, Colours::white);
SlideTimeReverb->addListener (this);
        SlideTimeReverb->setValue(4);

addAndMakeVisible (Slide1waa = new Slider ("new slider"));
Slide1waa->setRange (0, 10, 0);
```

```
Slide1waa->setSliderStyle (Slider::LinearBar);
Slide1waa->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
Slide1waa->setColour (Slider::thumbColourId, Colour (0x50e1e13a));
Slide1waa->setColour (Slider::textBoxTextColourId, Colours::white);
Slide1waa->addListener (this);

addAndMakeVisible (Slide2Waa = new Slider ("new slider"));
Slide2Waa->setRange (0, 10, 0);
Slide2Waa->setSliderStyle (Slider::LinearBar);
Slide2Waa->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
Slide2Waa->setColour (Slider::thumbColourId, Colour (0x50e1e13a));
Slide2Waa->setColour (Slider::textBoxTextColourId, Colours::white);
Slide2Waa->addListener (this);

addAndMakeVisible (Slide3Waa = new Slider ("new slider"));
Slide3Waa->setRange (0, 10, 0);
Slide3Waa->setSliderStyle (Slider::LinearBar);
Slide3Waa->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
Slide3Waa->setColour (Slider::thumbColourId, Colour (0x50e1e13a));
Slide3Waa->setColour (Slider::textBoxTextColourId, Colours::white);
Slide3Waa->addListener (this);

addAndMakeVisible (groupComponent7 = new GroupComponent ("new group",
                                    "General"));
groupComponent7->setColour (GroupComponent::textColourId, Colours::white);

addAndMakeVisible (SlideMainGain = new Slider ("new slider"));
SlideMainGain->setRange (-20, 20, 0);
```

```
SlideMainGain->setSliderStyle (Slider::LinearBar);
SlideMainGain->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideMainGain->setColour (Slider::thumbColourId, Colour (0xaa000000));
SlideMainGain->setColour (Slider::textBoxTextColourId, Colour (0xfff8f8f8));
SlideMainGain->addListener (this);

addAndMakeVisible (SlideFreqH = new Slider ("new slider"));
SlideFreqH->setRange (0, 10, 0);
SlideFreqH->setSliderStyle (Slider::LinearBar);
SlideFreqH->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideFreqH->setColour (Slider::thumbColourId, Colour (0xaa000000));
SlideFreqH->setColour (Slider::textBoxTextColourId, Colours::white);
SlideFreqH->addListener (this);

addAndMakeVisible (SlideFreqL = new Slider ("new slider"));
SlideFreqL->setRange (0, 10, 0);
SlideFreqL->setSliderStyle (Slider::LinearBar);
SlideFreqL->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideFreqL->setColour (Slider::thumbColourId, Colour (0xaa000000));
SlideFreqL->setColour (Slider::textBoxTextColourId, Colours::white);
SlideFreqL->addListener (this);

addAndMakeVisible (SlideFreqM = new Slider ("new slider"));
SlideFreqM->setRange (0, 10, 0);
SlideFreqM->setSliderStyle (Slider::LinearBar);
SlideFreqM->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideFreqM->setColour (Slider::thumbColourId, Colour (0xaa000000));
SlideFreqM->setColour (Slider::textBoxTextColourId, Colours::white);
```

```
SlideFreqM->addListener (this);

addAndMakeVisible (SlideLow = new Slider ("new slider"));
SlideLow->setRange (0, 10, 0);
SlideLow->setSliderStyle (Slider::LinearBar);
SlideLow->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideLow->setColour (Slider::thumbColourId, Colour (0xaa000000));
SlideLow->setColour (Slider::textBoxTextColourId, Colours::white);
SlideLow->addListener (this);

addAndMakeVisible (slider19 = new Slider ("new slider"));
slider19->setRange (0, 10, 0);
slider19->setSliderStyle (Slider::LinearBar);
slider19->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
slider19->setColour (Slider::thumbColourId, Colour (0xaa000000));
slider19->setColour (Slider::textBoxTextColourId, Colours::white);
slider19->addListener (this);

addAndMakeVisible (SlideGainFlange = new Slider ("new slider"));
SlideGainFlange->setRange (0, 10, 0);
SlideGainFlange->setSliderStyle (Slider::LinearBar);
SlideGainFlange->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideGainFlange->setColour (Slider::thumbColourId, Colour (0x508a2be2));
SlideGainFlange->setColour (Slider::textBoxTextColourId, Colours::white);
SlideGainFlange->addListener (this);

addAndMakeVisible (SlideTimeFlange = new Slider ("new slider"));
SlideTimeFlange->setRange (0, 10, 0);
```

```
SlideTimeFlange->setSliderStyle (Slider::LinearBar);
SlideTimeFlange->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideTimeFlange->setColour (Slider::thumbColourId, Colour (0x508a2be2));
SlideTimeFlange->setColour (Slider::textBoxTextColourId, Colours::white);
SlideTimeFlange->addListener (this);

addAndMakeVisible (Slide3Flange = new Slider ("new slider"));
Slide3Flange->setRange (0, 10, 0);
Slide3Flange->setSliderStyle (Slider::LinearBar);
Slide3Flange->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
Slide3Flange->setColour (Slider::thumbColourId, Colour (0x508a2be2));
Slide3Flange->setColour (Slider::textBoxTextColourId, Colours::white);
Slide3Flange->addListener (this);

addAndMakeVisible (Slide1ETC = new Slider ("new slider"));
Slide1ETC->setRange (0, 10, 0);
Slide1ETC->setSliderStyle (Slider::LinearBar);
Slide1ETC->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
Slide1ETC->setColour (Slider::thumbColourId, Colour (0x505f9ea0));
Slide1ETC->setColour (Slider::textBoxTextColourId, Colours::white);
Slide1ETC->addListener (this);

addAndMakeVisible (Slide2ETC = new Slider ("new slider"));
Slide2ETC->setRange (0, 10, 0);
Slide2ETC->setSliderStyle (Slider::LinearBar);
Slide2ETC->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
Slide2ETC->setColour (Slider::thumbColourId, Colour (0x505f9ea0));
Slide2ETC->setColour (Slider::textBoxTextColourId, Colours::white);
```

```
Slide2ETC->addListener (this);

addAndMakeVisible (Slide3ETC = new Slider ("new slider"));
Slide3ETC->setRange (0, 10, 0);
Slide3ETC->setSliderStyle (Slider::LinearBar);
Slide3ETC->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
Slide3ETC->setColour (Slider::thumbColourId, Colour (0x505f9ea0));
Slide3ETC->setColour (Slider::textBoxTextColourId, Colours::white);
Slide3ETC->addListener (this);

addAndMakeVisible (label = new Label ("new label",
                                      "Gain\n"));
label->setFont (Font (15.00f, Font::plain));
label->setJustificationType (Justification::centredLeft);
label->setEditable (false, false, false);
label->setColour (Label::backgroundColourId, Colour (0x50ff0000));
label->setColour (Label::textColourId, Colours::white);
label->setColour (Label::outlineColourId, Colour (0x0000f7f7));
label->setColour (TextEditor::textColourId, Colours::black);
label->setColour (TextEditor::backgroundColourId, Colour (0xb6f7f7f7));

addAndMakeVisible (label2 = new Label ("new label",
                                       "Drive"));
label2->setFont (Font (15.00f, Font::plain));
label2->setJustificationType (Justification::centredLeft);
label2->setEditable (false, false, false);
label2->setColour (Label::backgroundColourId, Colour (0x50ff0000));
label2->setColour (Label::textColourId, Colours::white);
```

```
label2->setColour (TextEditor::textColourId, Colours::black);
label2->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label3 = new Label ("new label",
                                       "Threshhold"));
label3->setFont (Font (15.00f, Font::plain));
label3->setJustificationType (Justification::centredLeft);
label3->setEditable (false, false, false);
label3->setColour (Label::backgroundColourId, Colour (0x50ff0000));
label3->setColour (Label::textColourId, Colours::white);
label3->setColour (TextEditor::textColourId, Colours::black);
label3->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label4 = new Label ("new label",
                                       "Gain\n"));
label4->setFont (Font (15.00f, Font::plain));
label4->setJustificationType (Justification::centredLeft);
label4->setEditable (false, false, false);
label4->setColour (Label::backgroundColourId, Colour (0x5000ff00));
label4->setColour (Label::textColourId, Colours::white);
label4->setColour (TextEditor::textColourId, Colours::black);
label4->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label5 = new Label ("new label",
                                       "FeedBack"));
label5->setFont (Font (15.00f, Font::plain));
label5->setJustificationType (Justification::centredLeft);
label5->setEditable (false, false, false);
```

```
label5->setColour (Label::backgroundColourId, Colour (0x5000ff00));
label5->setColour (Label::textColourId, Colours::white);
label5->setColour (TextEditor::textColourId, Colours::black);
label5->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label6 = new Label ("new label",
                                       "Time Delay\n"));
label6->setFont (Font (15.00f, Font::plain));
label6->setJustificationType (Justification::centredLeft);
label6->setEditable (false, false, false);
label6->setColour (Label::backgroundColourId, Colour (0x5000ff00));
label6->setColour (Label::textColourId, Colours::white);
label6->setColour (TextEditor::textColourId, Colours::black);
label6->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label7 = new Label ("new label",
                                       "Gain"));
label7->setFont (Font (15.00f, Font::plain));
label7->setJustificationType (Justification::centredLeft);
label7->setEditable (false, false, false);
label7->setColour (Label::backgroundColourId, Colour (0x50e1e13a));
label7->setColour (Label::textColourId, Colours::white);
label7->setColour (TextEditor::textColourId, Colours::black);
label7->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label8 = new Label ("new label",
                                       "Time Delay"));
label8->setFont (Font (15.00f, Font::plain));
```

```
label8->setJustificationType (Justification::centredLeft);
label8->setEditable (false, false, false);
label8->setColour (Label::backgroundColourId, Colour (0x50e1e13a));
label8->setColour (Label::textColourId, Colours::white);
label8->setColour (TextEditor::textColourId, Colours::black);
label8->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label9 = new Label ("new label",
                                       "Param 3"));
label9->setFont (Font (15.00f, Font::plain));
label9->setJustificationType (Justification::centredLeft);
label9->setEditable (false, false, false);
label9->setColour (Label::backgroundColourId, Colour (0x50e1e13a));
label9->setColour (Label::textColourId, Colours::white);
label9->setColour (TextEditor::textColourId, Colours::black);
label9->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label10 = new Label ("new label",
                                        "Gain\n"));
label10->setFont (Font (15.00f, Font::plain));
label10->setJustificationType (Justification::centredLeft);
label10->setEditable (false, false, false);
label10->setColour (Label::backgroundColourId, Colour (0x500000ff));
label10->setColour (Label::textColourId, Colours::white);
label10->setColour (TextEditor::textColourId, Colours::black);
label10->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label11 = new Label ("new label",
```

```
                    "FeedBack\n"));
label11->setFont (Font (15.00f, Font::plain));
label11->setJustificationType (Justification::centredLeft);
label11->setEditable (false, false, false);
label11->setColour (Label::backgroundColourId, Colour (0x500000ff));
label11->setColour (Label::textColourId, Colours::white);
label11->setColour (TextEditor::textColourId, Colours::black);
label11->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label12 = new Label ("new label",
                    "Time Delay\n"));
label12->setFont (Font (15.00f, Font::plain));
label12->setJustificationType (Justification::centredLeft);
label12->setEditable (false, false, false);
label12->setColour (Label::backgroundColourId, Colour (0x500000ff));
label12->setColour (Label::textColourId, Colours::white);
label12->setColour (TextEditor::textColourId, Colours::black);
label12->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label13 = new Label ("new label",
                    "Gain\n"));
label13->setFont (Font (15.00f, Font::plain));
label13->setJustificationType (Justification::centredLeft);
label13->setEditable (false, false, false);
label13->setColour (Label::backgroundColourId, Colour (0x50b140b1));
label13->setColour (Label::textColourId, Colours::white);
label13->setColour (TextEditor::textColourId, Colours::black);
label13->setColour (TextEditor::backgroundColourId, Colour (0x00000000));
```

```
addAndMakeVisible (label14 = new Label ("new label",
                                        "Time Delay"));
label14->setFont (Font (15.00f, Font::plain));
label14->setJustificationType (Justification::centredLeft);
label14->setEditable (false, false, false);
label14->setColour (Label::backgroundColourId, Colour (0x50b140b1));
label14->setColour (Label::textColourId, Colours::white);
label14->setColour (TextEditor::textColourId, Colours::black);
label14->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (Param3 = new Label ("new label",
                                       "Threshold"));
Param3->setFont (Font (15.00f, Font::plain));
Param3->setJustificationType (Justification::centredLeft);
Param3->setEditable (false, false, false);
Param3->setColour (Label::backgroundColourId, Colour (0x50b140b1));
Param3->setColour (Label::textColourId, Colours::white);
Param3->setColour (TextEditor::textColourId, Colours::black);
Param3->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label16 = new Label ("new label",
                                        "Param 1"));
label16->setFont (Font (15.00f, Font::plain));
label16->setJustificationType (Justification::centredLeft);
label16->setEditable (false, false, false);
label16->setColour (Label::backgroundColourId, Colour (0x507fffd4));
label16->setColour (Label::textColourId, Colours::white);
```

```
label16->setColour (TextEditor::textColourId, Colours::black);
label16->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label17 = new Label ("new label",
                                        "Param 2"));
label17->setFont (Font (15.00f, Font::plain));
label17->setJustificationType (Justification::centredLeft);
label17->setEditable (false, false, false);
label17->setColour (Label::backgroundColourId, Colour (0x507fffd4));
label17->setColour (Label::textColourId, Colours::white);
label17->setColour (TextEditor::textColourId, Colours::black);
label17->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label18 = new Label ("new label",
                                        "Param 3"));
label18->setFont (Font (15.00f, Font::plain));
label18->setJustificationType (Justification::centredLeft);
label18->setEditable (false, false, false);
label18->setColour (Label::backgroundColourId, Colour (0x507fffd4));
label18->setColour (Label::textColourId, Colours::white);
label18->setColour (TextEditor::textColourId, Colours::black);
label18->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label19 = new Label ("new label",
                                        "Main Volume"));
label19->setFont (Font (15.00f, Font::plain));
label19->setJustificationType (Justification::centredLeft);
label19->setEditable (false, false, false);
```

```
label19->setColour (Label::textColourId, Colours::white);
label19->setColour (TextEditor::textColourId, Colours::black);
label19->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label20 = new Label ("new label",
                                        "Freq H"));
label20->setFont (Font (15.00f, Font::plain));
label20->setJustificationType (Justification::centredLeft);
label20->setEditable (false, false, false);
label20->setColour (Label::textColourId, Colours::white);
label20->setColour (TextEditor::textColourId, Colours::black);
label20->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label22 = new Label ("new label",
                                        "Freq L"));
label22->setFont (Font (15.00f, Font::plain));
label22->setJustificationType (Justification::centredLeft);
label22->setEditable (false, false, false);
label22->setColour (Label::textColourId, Colours::white);
label22->setColour (TextEditor::textColourId, Colours::black);
label22->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label23 = new Label ("new label",
                                        "Freq M"));
label23->setFont (Font (15.00f, Font::plain));
label23->setJustificationType (Justification::centredLeft);
label23->setEditable (false, false, false);
label23->setColour (Label::textColourId, Colours::white);
```

```
label23->setColour (TextEditor::textColourId, Colours::white);
label23->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label25 = new Label ("new label",
                                        "LowPass"));
label25->setFont (Font (15.00f, Font::plain));
label25->setJustificationType (Justification::centredLeft);
label25->setEditable (false, false, false);
label25->setColour (Label::textColourId, Colour (0xfffdf9f9));
label25->setColour (TextEditor::textColourId, Colours::white);
label25->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (label26 = new Label ("new label",
                                        "HighPass"));
label26->setFont (Font (15.00f, Font::plain));
label26->setJustificationType (Justification::centredLeft);
label26->setEditable (false, false, false);
label26->setColour (Label::textColourId, Colours::white);
label26->setColour (TextEditor::textColourId, Colours::black);
label26->setColour (TextEditor::backgroundColourId, Colour (0x00000000));

addAndMakeVisible (SlideGainDistort = new Slider ("new slider"));
SlideGainDistort->setRange (0, 10, 0);
SlideGainDistort->setSliderStyle (Slider::LinearBar);
SlideGainDistort->setTextBoxStyle (Slider::TextBoxLeft, false, 80, 20);
SlideGainDistort->setColour (Slider::backgroundColourId, Colour (0x00000000));
SlideGainDistort->setColour (Slider::thumbColourId, Colour (0x50ff0000));
SlideGainDistort->setColour (Slider::trackColourId, Colour (0x50ff0000));
```

```
SlideGainDistort->setColour (Slider::rotarySliderFillColourId, Colour (0x50ff0000));
SlideGainDistort->setColour (Slider::textBoxTextColourId, Colours::white);
SlideGainDistort->addListener (this);

addAndMakeVisible (toggleDistort = new ToggleButton ("Distort on/off"));
toggleDistort->addListener (this);

addAndMakeVisible (toggleReverb = new ToggleButton ("Reverb on/off"));
toggleReverb->addListener (this);

addAndMakeVisible (toggleTremolo = new ToggleButton ("Tremolo on/off"));
toggleTremolo->addListener (this);

addAndMakeVisible (toggleRandom = new ToggleButton ("Random on/off"));
toggleRandom->addListener (this);

addAndMakeVisible (toggleFlange = new ToggleButton ("Flange on/off"));
toggleFlange->addListener (this);

addAndMakeVisible (toggleDelay = new ToggleButton ("Delay on/off"));
toggleDelay->addListener (this);


//[UserPreSize]
//[/UserPreSize]

setSize (650, 500);
```

```
    //[Constructor] You can add your own custom stuff here..
    startTimer(200);//starts timer with interval of 200mS
    //[/Constructor]
}

CombinedGuiAudioProcessorEditor::~CombinedGuiAudioProcessorEditor()
{
    //[Destructor_pre]. You can add your own custom destruction code here..
    //[/Destructor_pre]

    groupComponent = nullptr;
    groupComponent2 = nullptr;
    groupComponent3 = nullptr;
    groupComponent4 = nullptr;
    groupComponent5 = nullptr;
    groupComponent6 = nullptr;
    SlideDriveDistort = nullptr;
    SlideThreshDistort = nullptr;
    SlideGainDelay = nullptr;
    SlideFeedDelay = nullptr;
    SlideTimeDelay = nullptr;
    SlideGainReverb = nullptr;
    SlideFeedReverb = nullptr;
    SlideTimeReverb = nullptr;
    Slide1waa = nullptr;
    Slide2Waa = nullptr;
    Slide3Waa = nullptr;
```

```
groupComponent7 = nullptr;
SlideMainGain = nullptr;
SlideFreqH = nullptr;
SlideFreqL = nullptr;
SlideFreqM = nullptr;
SlideLow = nullptr;
slider19 = nullptr;
SlideGainFlange = nullptr;
SlideTimeFlange = nullptr;
Slide3Flange = nullptr;
Slide1ETC = nullptr;
Slide2ETC = nullptr;
Slide3ETC = nullptr;
label = nullptr;
label2 = nullptr;
label3 = nullptr;
label4 = nullptr;
label5 = nullptr;
label6 = nullptr;
label7 = nullptr;
label8 = nullptr;
label9 = nullptr;
label10 = nullptr;
label11 = nullptr;
label12 = nullptr;
label13 = nullptr;
label14 = nullptr;
Param3 = nullptr;
```

```
    label16 = nullptr;
    label17 = nullptr;
    label18 = nullptr;
    label19 = nullptr;
    label20 = nullptr;
    label22 = nullptr;
    label23 = nullptr;
    label25 = nullptr;
    label26 = nullptr;
    SlideGainDistort = nullptr;
    toggleDistort = nullptr;
    toggleReverb = nullptr;
    toggleTremolo = nullptr;
    toggleRandom = nullptr;
    toggleFlange = nullptr;
    toggleDelay = nullptr;



    //[Destructor]. You can add your own custom destruction code here..
    //[/Destructor]
}

//==============================================================================
void CombinedGuiAudioProcessorEditor::paint (Graphics& g)
{
    //[UserPrePaint] Add your own custom painting code here..
    //[/UserPrePaint]
```

```
    g.fillAll (Colour (0xff454444));

    //[UserPaint] Add your own custom painting code here..
    //[/UserPaint]
}

void CombinedGuiAudioProcessorEditor::resized()
{
    groupComponent->setBounds (10, 10, 200, 151);
    groupComponent2->setBounds (10, 170, 200, 150);
    groupComponent3->setBounds (220, 10, 210, 151);
    groupComponent4->setBounds (220, 170, 210, 151);
    groupComponent5->setBounds (440, 10, 210, 151);
    groupComponent6->setBounds (440, 170, 210, 151);
    SlideDriveDistort->setBounds (110, 70, 96, 16);
    SlideThreshDistort->setBounds (110, 110, 96, 16);
    SlideGainDelay->setBounds (110, 190, 96, 16);
    SlideFeedDelay->setBounds (110, 230, 96, 16);
    SlideTimeDelay->setBounds (110, 270, 96, 16);
    SlideGainReverb->setBounds (330, 30, 96, 16);
    SlideFeedReverb->setBounds (328, 72, 96, 16);
    SlideTimeReverb->setBounds (330, 110, 96, 16);
    Slide1waa->setBounds (550, 30, 96, 16);
    Slide2Waa->setBounds (550, 70, 96, 16);
    Slide3Waa->setBounds (550, 110, 96, 16);
    groupComponent7->setBounds (8, 328, 643, 100);
    SlideMainGain->setBounds (110, 350, 96, 16);
    SlideFreqH->setBounds (110, 390, 96, 16);
```

```
SlideFreqL->setBounds (330, 350, 96, 16);
SlideFreqM->setBounds (330, 392, 96, 16);
SlideLow->setBounds (550, 350, 96, 16);
slider19->setBounds (550, 390, 96, 16);
SlideGainFlange->setBounds (330, 190, 96, 16);
SlideTimeFlange->setBounds (330, 230, 96, 16);
Slide3Flange->setBounds (330, 270, 96, 16);
Slide1ETC->setBounds (550, 196, 96, 16);
Slide2ETC->setBounds (550, 236, 96, 16);
Slide3ETC->setBounds (550, 276, 96, 16);
label->setBounds (20, 30, 44, 24);
label2->setBounds (20, 70, 52, 24);
label3->setBounds (20, 110, 84, 24);
label4->setBounds (230, 30, 42, 24);
label5->setBounds (230, 70, 74, 24);
label6->setBounds (230, 110, 90, 24);
label7->setBounds (450, 30, 38, 24);
label8->setBounds (450, 70, 78, 24);
label9->setBounds (450, 110, 102, 24);
label10->setBounds (20, 190, 44, 24);
label11->setBounds (20, 230, 76, 24);
label12->setBounds (20, 270, 84, 24);
label13->setBounds (230, 190, 42, 24);
label14->setBounds (230, 230, 82, 24);
Param3->setBounds (230, 270, 74, 24);
label16->setBounds (450, 190, 62, 24);
label17->setBounds (450, 230, 62, 24);
label18->setBounds (450, 270, 62, 24);
```

```
    label19->setBounds (22, 345, 82, 24);
    label20->setBounds (22, 385, 58, 24);
    label22->setBounds (232, 345, 56, 24);
    label23->setBounds (232, 385, 64, 24);
    label25->setBounds (452, 345, 76, 24);
    label26->setBounds (452, 385, 76, 24);
    SlideGainDistort->setBounds (110, 30, 96, 16);
    toggleDistort->setBounds (20, 136, 150, 24);
    toggleReverb->setBounds (230, 136, 150, 24);
    toggleTremolo->setBounds (450, 136, 150, 24);
    toggleRandom->setBounds (450, 296, 150, 24);
    toggleFlange->setBounds (230, 296, 150, 24);
    toggleDelay->setBounds (20, 296, 150, 24);
    //[UserResized] Add your own custom resize handling here..
    //[/UserResized]
}

void CombinedGuiAudioProcessorEditor::sliderValueChanged (Slider* sliderThatWasMoved)
{
    //[UsersliderValueChanged_Pre]
    //[/UsersliderValueChanged_Pre]

    if (sliderThatWasMoved == SlideDriveDistort)
    {
        //[UserSliderCode_SlideDriveDistort] -- add your slider handling code here..
        //[/UserSliderCode_SlideDriveDistort]
    }
    else if (sliderThatWasMoved == SlideThreshDistort)
```

```
{
   //[UserSliderCode_SlideThreshDistort] -- add your slider handling code here..
   //[/UserSliderCode_SlideThreshDistort]
}
else if (sliderThatWasMoved == SlideGainDelay)
{
   //[UserSliderCode_SlideGainDelay] -- add your slider handling code here..
   //[/UserSliderCode_SlideGainDelay]
}
else if (sliderThatWasMoved == SlideFeedDelay)
{
   //[UserSliderCode_SlideFeedDelay] -- add your slider handling code here..
   //[/UserSliderCode_SlideFeedDelay]
}
else if (sliderThatWasMoved == SlideTimeDelay)
{
   //[UserSliderCode_SlideTimeDelay] -- add your slider handling code here..
   //[/UserSliderCode_SlideTimeDelay]
}
else if (sliderThatWasMoved == SlideGainReverb)
{
   //[UserSliderCode_SlideGainReverb] -- add your slider handling code here..
   //[/UserSliderCode_SlideGainReverb]
}
else if (sliderThatWasMoved == SlideFeedReverb)
{
   //[UserSliderCode_SlideFeedReverb] -- add your slider handling code here..
   //[/UserSliderCode_SlideFeedReverb]
```

```
}
else if (sliderThatWasMoved == SlideTimeReverb)
{
   //[UserSliderCode_SlideTimeReverb] -- add your slider handling code here..
   //[/UserSliderCode_SlideTimeReverb]
}
else if (sliderThatWasMoved == Slide1waa)
{
   //[UserSliderCode_Slide1waa] -- add your slider handling code here..
   //[/UserSliderCode_Slide1waa]
}
else if (sliderThatWasMoved == Slide2Waa)
{
   //[UserSliderCode_Slide2Waa] -- add your slider handling code here..
   //[/UserSliderCode_Slide2Waa]
}
else if (sliderThatWasMoved == Slide3Waa)
{
   //[UserSliderCode_Slide3Waa] -- add your slider handling code here..
   //[/UserSliderCode_Slide3Waa]
}
else if (sliderThatWasMoved == SlideMainGain)
{
   //[UserSliderCode_SlideMainGain] -- add your slider handling code here..
   //[/UserSliderCode_SlideMainGain]
}
else if (sliderThatWasMoved == SlideFreqH)
{
```

```
    //[UserSliderCode_SlideFreqH] -- add your slider handling code here..
    //[/UserSliderCode_SlideFreqH]
}
else if (sliderThatWasMoved == SlideFreqL)
{
    //[UserSliderCode_SlideFreqL] -- add your slider handling code here..
    //[/UserSliderCode_SlideFreqL]
}
else if (sliderThatWasMoved == SlideFreqM)
{
    //[UserSliderCode_SlideFreqM] -- add your slider handling code here..
    //[/UserSliderCode_SlideFreqM]
}
else if (sliderThatWasMoved == SlideLow)
{
    //[UserSliderCode_SlideLow] -- add your slider handling code here..
    //[/UserSliderCode_SlideLow]
}
else if (sliderThatWasMoved == slider19)
{
    //[UserSliderCode_slider19] -- add your slider handling code here..
    //[/UserSliderCode_slider19]
}
else if (sliderThatWasMoved == SlideGainFlange)
{
    //[UserSliderCode_SlideGainFlange] -- add your slider handling code here..
    //[/UserSliderCode_SlideGainFlange]
}
```

```
else if (sliderThatWasMoved == SlideTimeFlange)
{
   //[UserSliderCode_SlideTimeFlange] -- add your slider handling code here..
   //[/UserSliderCode_SlideTimeFlange]
}
else if (sliderThatWasMoved == Slide3Flange)
{
   //[UserSliderCode_Slide3Flange] -- add your slider handling code here..
   //[/UserSliderCode_Slide3Flange]
}
else if (sliderThatWasMoved == Slide1ETC)
{
   //[UserSliderCode_Slide1ETC] -- add your slider handling code here..
   //[/UserSliderCode_Slide1ETC]
}
else if (sliderThatWasMoved == Slide2ETC)
{
   //[UserSliderCode_Slide2ETC] -- add your slider handling code here..
   //[/UserSliderCode_Slide2ETC]
}
else if (sliderThatWasMoved == Slide3ETC)
{
   //[UserSliderCode_Slide3ETC] -- add your slider handling code here..
   //[/UserSliderCode_Slide3ETC]
}
else if (sliderThatWasMoved == SlideGainDistort)
{
   //[UserSliderCode_SlideGainDistort] -- add your slider handling code here..
```

```
        //[/UserSliderCode_SlideGainDistort]
    }
    //[UsersliderValueChanged_Post]
    //[/UsersliderValueChanged_Post]
}
void CombinedGuiAudioProcessorEditor::buttonClicked (Button* buttonThatWasClicked)
{
    //[UserbuttonClicked_Pre]
    //[/UserbuttonClicked_Pre]

    if (buttonThatWasClicked == toggleDistort)
    {
        //[UserButtonCode_toggleDistort] -- add your button handler code here..
        //[/UserButtonCode_toggleDistort]
    }
    else if (buttonThatWasClicked == toggleReverb)
    {
        //[UserButtonCode_toggleReverb] -- add your button handler code here..
        //[/UserButtonCode_toggleReverb]
    }
    else if (buttonThatWasClicked == toggleTremolo)
    {
        //[UserButtonCode_toggleTremolo] -- add your button handler code here..
        //[/UserButtonCode_toggleTremolo]
    }
    else if (buttonThatWasClicked == toggleRandom)
    {
        //[UserButtonCode_toggleRandom] -- add your button handler code here..
```

```
    //[/UserButtonCode_toggleRandom]
  }
  else if (buttonThatWasClicked == toggleFlange)
  {
    //[UserButtonCode_toggleFlange] -- add your button handler code here..
    //[/UserButtonCode_toggleFlange]
  }
  else if (buttonThatWasClicked == toggleDelay)
  {
    //[UserButtonCode_toggleDelay] -- add your button handler code here..
    //[/UserButtonCode_toggleDelay]
  }

  //[UserbuttonClicked_Post]
  //[/UserbuttonClicked_Post]
}
```

```
//[MiscUserCode] You can add your own definitions of your custom methods or any other code here...
void CombinedGuiAudioProcessorEditor::timerCallback()
{
    CombinedGuiAudioProcessor* ourProcessor = getProcessor();
            //distort
            ourProcessor->gain=(float)SlideGainDistort->getValue();
            ourProcessor->bubble=1/(float)SlideDriveDistort->getValue();
            ourProcessor->mult=(float)SlideThreshDistort->getValue();
            //flange
```

```
        //reverb
        ourProcessor->reverbTDelay =(float)SlideTimeReverb->getValue();
        ourProcessor->reverbGain = (float)SlideGainReverb->getValue();
                //ourProcessor->reverbfeed =(float)SlideFeedReverb->getValue();
        //delay
        ourProcessor->delayGain = (float)SlideGainDelay->getValue();
        ourProcessor->delayTDelay = (float)SlideTimeDelay->getValue();
        //random
        //tremolo
                //tremmultiplier
        ourProcessor->maingain=SlideMainGain->getValue();
        //on.off
        ourProcessor->boolDelay =toggleDelay->getToggleState();
       ourProcessor->boolFlange=toggleFlange->getToggleState();
       ourProcessor->boolReverb=toggleReverb->getToggleState();
       ourProcessor->boolRandom=toggleRandom->getToggleState();
       ourProcessor->boolDistort=toggleDistort->getToggleState();
       ourProcessor->boolTremolo=toggleTremolo->getToggleState();

    //exchange any data you want between UI elements and the Plugin "ourProcessor"
}
//[/MiscUserCode]



//==============================================================================
===
#if 0
/*  -- Introjucer information section --
```

```
  This is where the Introjucer stores the metadata that describe this GUI layout, so
  make changes in here at your peril!


BEGIN_JUCER_METADATA


<JUCER_COMPONENT documentType="Component"
classname="CombinedGuiAudioProcessorEditor"
        componentName="" parentClasses="public AudioProcessorEditor, public Timer"
        constructorParams="CombinedGuiAudioProcessor* ownerFilter"
variableInitialisers="AudioProcessorEditor(ownerFilter)"
        snapPixels="8" snapActive="1" snapShown="1" overlayOpacity="0.330"
        fixedSize="0" initialWidth="650" initialHeight="500">
  <BACKGROUND backgroundColour="ff454444"/>
  <GROUPCOMPONENT name="new group" id="c79c5580dee58ae"
memberName="groupComponent"
         virtualName="" explicitFocusOrder="0" pos="10 10 200 151" outlinecol="aaff0000"
         textcol="fff8f4f4" title="Distortion"/>
  <GROUPCOMPONENT name="new group" id="9dfc525602cd9730"
memberName="groupComponent2"
         virtualName="" explicitFocusOrder="0" pos="10 170 200 150" outlinecol="aa0000ff"
         textcol="ffffffff" title="Delay"/>
  <GROUPCOMPONENT name="new group" id="881f12f2a31d9c18"
memberName="groupComponent3"
         virtualName="" explicitFocusOrder="0" pos="220 10 210 151" outlinecol="aa00ff00"
         textcol="ffffffff" title="Reverb"/>
  <GROUPCOMPONENT name="new group" id="dddf81cbf89999e5"
memberName="groupComponent4"
         virtualName="" explicitFocusOrder="0" pos="220 170 210 151" outlinecol="aabf40bf"
         textcol="ffffffff" title="Flange"/>
  <GROUPCOMPONENT name="new group" id="65a2699eacf4fdd8"
memberName="groupComponent5"
```

```
        virtualName="" explicitFocusOrder="0" pos="440 10 210 151" outlinecol="aae1e13a"
        textcol="ffffffff" title="Tremolo"/>
 <GROUPCOMPONENT name="new group" id="b084d38ae4ab7e2e"
memberName="groupComponent6"
        virtualName="" explicitFocusOrder="0" pos="440 170 210 151" outlinecol="ff5f9ea0"
        textcol="ffffffff" title="Randomizer"/>
 <SLIDER name="new slider" id="54e4868633d385f2" memberName="SlideDriveDistort"
    virtualName="" explicitFocusOrder="0" pos="110 70 96 16" thumbcol="50ff0000"
    rotarysliderfill="50ff0000" textboxtext="ffffffff" min="0" max="10"
    int="0" style="LinearBar" textBoxPos="TextBoxLeft" textBoxEditable="1"
    textBoxWidth="80" textBoxHeight="20" skewFactor="1"/>
 <SLIDER name="new slider" id="305a291d9bc743d7" memberName="SlideThreshDistort"
    virtualName="" explicitFocusOrder="0" pos="110 110 96 16" thumbcol="50ff0000"
    rotarysliderfill="50ff0000" textboxtext="ffffffff" textboxhighlight="50ff0000"
    min="0" max="10" int="0" style="LinearBar" textBoxPos="TextBoxLeft"
    textBoxEditable="1" textBoxWidth="80" textBoxHeight="20" skewFactor="1"/>
 <SLIDER name="new slider" id="1cbe5375ffd4dc6c" memberName="SlideGainDelay"
    virtualName="" explicitFocusOrder="0" pos="110 190 96 16" thumbcol="500000ff"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
 <SLIDER name="new slider" id="260cc5fdd990dc7c" memberName="SlideFeedDelay"
    virtualName="" explicitFocusOrder="0" pos="110 230 96 16" thumbcol="500000ff"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
 <SLIDER name="new slider" id="1f79de5256140b8e" memberName="SlideTimeDelay"
    virtualName="" explicitFocusOrder="0" pos="110 270 96 16" thumbcol="500000ff"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
```

```
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="8ca405cbc458a30e" memberName="SlideGainReverb"
    virtualName="" explicitFocusOrder="0" pos="330 30 96 16" thumbcol="5000ff00"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="528ab5dc8958e94a" memberName="SlideFeedReverb"
    virtualName="" explicitFocusOrder="0" pos="328 72 96 16" thumbcol="5000ff00"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="2dd38340897bcf90" memberName="SlideTimeReverb"
    virtualName="" explicitFocusOrder="0" pos="330 110 96 16" thumbcol="5000ff00"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="5092d28a7efa47b5" memberName="Slide1waa"
    virtualName="" explicitFocusOrder="0" pos="550 30 96 16" thumbcol="50e1e13a"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="1bcb7b2fd4d2a0f7" memberName="Slide2Waa"
    virtualName="" explicitFocusOrder="0" pos="550 70 96 16" thumbcol="50e1e13a"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="f2b716743e23c97c" memberName="Slide3Waa"
```

```
    virtualName="" explicitFocusOrder="0" pos="550 110 96 16" thumbcol="50e1e13a"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
  <GROUPCOMPONENT name="new group" id="22dc506a40fa89e5"
memberName="groupComponent7"
        virtualName="" explicitFocusOrder="0" pos="8 328 643 100" textcol="ffffffff"
        title="General"/>
  <SLIDER name="new slider" id="c040df7067a77ad7" memberName="SlideMainGain"
    virtualName="" explicitFocusOrder="0" pos="110 350 96 16" thumbcol="aa000000"
    textboxtext="fff8f8f8" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
  <SLIDER name="new slider" id="93f0d7d628632f9e" memberName="SlideFreqH"
    virtualName="" explicitFocusOrder="0" pos="110 390 96 16" thumbcol="aa000000"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
  <SLIDER name="new slider" id="d3cb742293274033" memberName="SlideFreqL"
    virtualName="" explicitFocusOrder="0" pos="330 350 96 16" thumbcol="aa000000"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
  <SLIDER name="new slider" id="f5761927ca7c4588" memberName="SlideFreqM"
    virtualName="" explicitFocusOrder="0" pos="330 392 96 16" thumbcol="aa000000"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
  <SLIDER name="new slider" id="9fd381e960e87018" memberName="SlideLow"
```

```
    virtualName="" explicitFocusOrder="0" pos="550 350 96 16" thumbcol="aa000000"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="c6c488dee6939cc" memberName="slider19"
    virtualName="" explicitFocusOrder="0" pos="550 390 96 16" thumbcol="aa000000"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="2347507be8656258" memberName="SlideGainFlange"
    virtualName="" explicitFocusOrder="0" pos="330 190 96 16" thumbcol="508a2be2"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="9b9071f920878fd5" memberName="SlideTimeFlange"
    virtualName="" explicitFocusOrder="0" pos="330 230 96 16" thumbcol="508a2be2"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="436e310157b6bb58" memberName="Slide3Flange"
    virtualName="" explicitFocusOrder="0" pos="330 270 96 16" thumbcol="508a2be2"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="bd2d51232829e1df" memberName="Slide1ETC"
    virtualName="" explicitFocusOrder="0" pos="550 196 96 16" thumbcol="505f9ea0"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
```

```
    textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="81d1a10cb27e3589" memberName="Slide2ETC"
    virtualName="" explicitFocusOrder="0" pos="550 236 96 16" thumbcol="505f9ea0"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="30eecf884674489" memberName="Slide3ETC"
    virtualName="" explicitFocusOrder="0" pos="550 276 96 16" thumbcol="505f9ea0"
    textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
    textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
    textBoxHeight="20" skewFactor="1"/>
<LABEL name="new label" id="6fd9973b663fafd0" memberName="label" virtualName=""
    explicitFocusOrder="0" pos="20 30 44 24" bkgCol="50ff0000" textCol="ffffffff"
    outlineCol="f7f7" edTextCol="ff000000" edBkgCol="b6f7f7f7" labelText="Gain&#10;"
    editableSingleClick="0" editableDoubleClick="0" focusDiscardsChanges="0"
    fontname="Default font" fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="4878d75203385d7e" memberName="label2" virtualName=""
    explicitFocusOrder="0" pos="20 70 52 24" bkgCol="50ff0000" textCol="ffffffff"
    edTextCol="ff000000" edBkgCol="0" labelText="Drive" editableSingleClick="0"
    editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
    fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="5a71efd9e4c37459" memberName="label3" virtualName=""
    explicitFocusOrder="0" pos="20 110 84 24" bkgCol="50ff0000" textCol="ffffffff"
    edTextCol="ff000000" edBkgCol="0" labelText="Threshhold" editableSingleClick="0"
    editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
    fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="30c000e53723a426" memberName="label4" virtualName=""
    explicitFocusOrder="0" pos="230 30 42 24" bkgCol="5000ff00" textCol="ffffffff"
```

```
    edTextCol="ff000000" edBkgCol="0" labelText="Gain&#10;" editableSingleClick="0"
    editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
    fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="f9333d84927751bb" memberName="label5" virtualName=""
    explicitFocusOrder="0" pos="230 70 74 24" bkgCol="5000ff00" textCol="ffffffff"
    edTextCol="ff000000" edBkgCol="0" labelText="FeedBack" editableSingleClick="0"
    editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
    fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="278f9da4093f0a4b" memberName="label6" virtualName=""
    explicitFocusOrder="0" pos="230 110 90 24" bkgCol="5000ff00"
    textCol="ffffffff" edTextCol="ff000000" edBkgCol="0" labelText="Time Delay&#10;"
    editableSingleClick="0" editableDoubleClick="0" focusDiscardsChanges="0"
    fontname="Default font" fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="6a91a1746337f3ca" memberName="label7" virtualName=""
    explicitFocusOrder="0" pos="450 30 38 24" bkgCol="50e1e13a" textCol="ffffffff"
    edTextCol="ff000000" edBkgCol="0" labelText="Gain" editableSingleClick="0"
    editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
    fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="b8dfe831ec993564" memberName="label8" virtualName=""
    explicitFocusOrder="0" pos="450 70 78 24" bkgCol="50e1e13a" textCol="ffffffff"
    edTextCol="ff000000" edBkgCol="0" labelText="Time Delay" editableSingleClick="0"
    editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
    fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="22e83b94aece90e7" memberName="label9" virtualName=""
    explicitFocusOrder="0" pos="450 110 102 24" bkgCol="50e1e13a"
    textCol="ffffffff" edTextCol="ff000000" edBkgCol="0" labelText="Param 3"
    editableSingleClick="0" editableDoubleClick="0" focusDiscardsChanges="0"
    fontname="Default font" fontsize="15" bold="0" italic="0" justification="33"/>
```

<LABEL name="new label" id="2a2c821e03b7bf0e" memberName="label10" virtualName=""
explicitFocusOrder="0" pos="20 190 44 24" bkgCol="500000ff" textCol="ffffffff"
edTextCol="ff000000" edBkgCol="0" labelText="Gain
" editableSingleClick="0"
editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="9841d6e2e1cc68d6" memberName="label11" virtualName=""
explicitFocusOrder="0" pos="20 230 76 24" bkgCol="500000ff" textCol="ffffffff"
edTextCol="ff000000" edBkgCol="0" labelText="FeedBack
" editableSingleClick="0"
editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="ad45c0341d7c0cf2" memberName="label12" virtualName=""
explicitFocusOrder="0" pos="20 270 84 24" bkgCol="500000ff" textCol="ffffffff"
edTextCol="ff000000" edBkgCol="0" labelText="Time Delay
"
editableSingleClick="0" editableDoubleClick="0" focusDiscardsChanges="0"
fontname="Default font" fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="4462cfb1296ae79c" memberName="label13" virtualName=""
explicitFocusOrder="0" pos="230 190 42 24" bkgCol="50b140b1"
textCol="ffffffff" edTextCol="ff000000" edBkgCol="0" labelText="Gain
"
editableSingleClick="0" editableDoubleClick="0" focusDiscardsChanges="0"
fontname="Default font" fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="7faaf379e4005cf9" memberName="label14" virtualName=""
explicitFocusOrder="0" pos="230 230 82 24" bkgCol="50b140b1"
textCol="ffffffff" edTextCol="ff000000" edBkgCol="0" labelText="Time Delay"
editableSingleClick="0" editableDoubleClick="0" focusDiscardsChanges="0"
fontname="Default font" fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="f185e4cbda4befa7" memberName="Param3" virtualName=""
explicitFocusOrder="0" pos="230 270 74 24" bkgCol="50b140b1"
textCol="ffffffff" edTextCol="ff000000" edBkgCol="0" labelText="Threshold"

```
    editableSingleClick="0" editableDoubleClick="0" focusDiscardsChanges="0"
    fontname="Default font" fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="6c3a2cc3960bc9d6" memberName="label16" virtualName=""
    explicitFocusOrder="0" pos="450 190 62 24" bkgCol="507fffd4"
    textCol="ffffffff" edTextCol="ff000000" edBkgCol="0" labelText="Param 1"
    editableSingleClick="0" editableDoubleClick="0" focusDiscardsChanges="0"
    fontname="Default font" fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="941bd6e0b6920de0" memberName="label17" virtualName=""
    explicitFocusOrder="0" pos="450 230 62 24" bkgCol="507fffd4"
    textCol="ffffffff" edTextCol="ff000000" edBkgCol="0" labelText="Param 2"
    editableSingleClick="0" editableDoubleClick="0" focusDiscardsChanges="0"
    fontname="Default font" fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="cb58c80ed434b177" memberName="label18" virtualName=""
    explicitFocusOrder="0" pos="450 270 62 24" bkgCol="507fffd4"
    textCol="ffffffff" edTextCol="ff000000" edBkgCol="0" labelText="Param 3"
    editableSingleClick="0" editableDoubleClick="0" focusDiscardsChanges="0"
    fontname="Default font" fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="79caafb32cd90a22" memberName="label19" virtualName=""
    explicitFocusOrder="0" pos="22 345 82 24" textCol="ffffffff"
    edTextCol="ff000000" edBkgCol="0" labelText="Main Volume" editableSingleClick="0"
    editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
    fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="8788b707236b8301" memberName="label20" virtualName=""
    explicitFocusOrder="0" pos="22 385 58 24" textCol="ffffffff"
    edTextCol="ff000000" edBkgCol="0" labelText="Freq H" editableSingleClick="0"
    editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
    fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="6c4703043c8da9cb" memberName="label22" virtualName=""
```

```
    explicitFocusOrder="0" pos="232 345 56 24" textCol="ffffffff"
    edTextCol="ff000000" edBkgCol="0" labelText="Freq L" editableSingleClick="0"
    editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
    fontsize="15" bold="0" italic="0" justification="33"/>
  <LABEL name="new label" id="506c972a14684807" memberName="label23" virtualName=""
    explicitFocusOrder="0" pos="232 385 64 24" textCol="ffffffff"
    edTextCol="ffffffff" edBkgCol="0" labelText="Freq M" editableSingleClick="0"
    editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
    fontsize="15" bold="0" italic="0" justification="33"/>
  <LABEL name="new label" id="9208fdc96131b155" memberName="label25" virtualName=""
    explicitFocusOrder="0" pos="452 345 76 24" textCol="fffdf9f9"
    edTextCol="ffffffff" edBkgCol="0" labelText="LowPass" editableSingleClick="0"
    editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
    fontsize="15" bold="0" italic="0" justification="33"/>
  <LABEL name="new label" id="14a27d226aa21dde" memberName="label26" virtualName=""
    explicitFocusOrder="0" pos="452 385 76 24" textCol="ffffffff"
    edTextCol="ff000000" edBkgCol="0" labelText="HighPass" editableSingleClick="0"
    editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
    fontsize="15" bold="0" italic="0" justification="33"/>
  <SLIDER name="new slider" id="68a8bbf71948149c" memberName="SlideGainDistort"
     virtualName="" explicitFocusOrder="0" pos="110 30 96 16" bkgcol="0"
     thumbcol="50ff0000" trackcol="50ff0000" rotarysliderfill="50ff0000"
     textboxtext="ffffffff" min="0" max="10" int="0" style="LinearBar"
     textBoxPos="TextBoxLeft" textBoxEditable="1" textBoxWidth="80"
     textBoxHeight="20" skewFactor="1"/>
  <TOGGLEBUTTON name="new toggle button" id="cfe8d6e6ee2ac669"
memberName="toggleDistort"
        virtualName="" explicitFocusOrder="0" pos="20 136 150 24" buttonText="new toggle
button"
```

connectedEdges="0" needsCallback="1" radioGroupId="0" state="0"/>

<TOGGLEBUTTON name="new toggle button" id="90b3cae3e5a09459" memberName="toggleReverb"

virtualName="" explicitFocusOrder="0" pos="230 136 150 24" buttonText="new toggle button"

connectedEdges="0" needsCallback="1" radioGroupId="0" state="0"/>

<TOGGLEBUTTON name="new toggle button" id="775cebabb7b2667a" memberName="toggleTremolo"

virtualName="" explicitFocusOrder="0" pos="450 136 150 24" buttonText="new toggle button"

connectedEdges="0" needsCallback="1" radioGroupId="0" state="0"/>

<TOGGLEBUTTON name="new toggle button" id="6029c1040af8dae1" memberName="toggleRandom"

virtualName="" explicitFocusOrder="0" pos="450 296 150 24" buttonText="new toggle button"

connectedEdges="0" needsCallback="1" radioGroupId="0" state="0"/>

<TOGGLEBUTTON name="new toggle button" id="45aa0f40bdc9f011" memberName="toggleFlange"

virtualName="" explicitFocusOrder="0" pos="230 296 150 24" buttonText="new toggle button"

connectedEdges="0" needsCallback="1" radioGroupId="0" state="0"/>

<TOGGLEBUTTON name="new toggle button" id="4476dd045668e0ab" memberName="toggleDelay"

virtualName="" explicitFocusOrder="0" pos="20 296 150 24" buttonText="new toggle button"

connectedEdges="0" needsCallback="1" radioGroupId="0" state="0"/>

</JUCER_COMPONENT>

END_JUCER_METADATA

*/

#endif

//[EndFile] You can add extra defines here...

//[/EndFile]

PluginEditor.h

```
/*

  ==============================================================================
  This is an automatically generated GUI class created by the Introjucer!

  Be careful when adding custom code to these files, as only the code within
  the "//[xyz]" and "//[/xyz]" sections will be retained when the file is loaded
  and re-saved.
  Created with Introjucer version: 3.1.0

  ------------------------------------------------------------------------------
  The Introjucer is part of the JUCE library - "Jules' Utility Class Extensions"
  Copyright 2004-13 by Raw Material Software Ltd.

  ==============================================================================
*/

#ifndef __JUCE_HEADER_EB75F46DAAF9F729__
#define __JUCE_HEADER_EB75F46DAAF9F729__

//[Headers]     -- You can add your own extra header files here --
#include "JuceHeader.h"
#include "PluginProcessor.h"
//[/Headers]



//==============================================================================
```

```
/**
                                    //[Comments]
   An auto-generated component, created by the Introjucer.

   Describe your class and how it works here!
                                    //[/Comments]
*/
class CombinedGuiAudioProcessorEditor  : public AudioProcessorEditor,
                        public Timer,
                        public SliderListener,
                        public ButtonListener
{
public:

//==============================================================================
   CombinedGuiAudioProcessorEditor (CombinedGuiAudioProcessor* ownerFilter);
   ~CombinedGuiAudioProcessorEditor();


//==============================================================================
   //[UserMethods]     -- You can add your own custom methods in this section.
   void timerCallback();
   CombinedGuiAudioProcessor* getProcessor() const
      {return static_cast <CombinedGuiAudioProcessor*>(getAudioProcessor());}
   //[/UserMethods]

   void paint (Graphics& g);
   void resized();
```

```
    void sliderValueChanged (Slider* sliderThatWasMoved);
    void buttonClicked (Button* buttonThatWasClicked);
private:
    //[UserVariables]   -- You can add your own custom variables in this section.
    //[/UserVariables]
//==============================================================================
    ScopedPointer<GroupComponent> groupComponent;
    ScopedPointer<GroupComponent> groupComponent2;
    ScopedPointer<GroupComponent> groupComponent3;
    ScopedPointer<GroupComponent> groupComponent4;
    ScopedPointer<GroupComponent> groupComponent5;
    ScopedPointer<GroupComponent> groupComponent6;
    ScopedPointer<Slider> SlideDriveDistort;
    ScopedPointer<Slider> SlideThreshDistort;
    ScopedPointer<Slider> SlideGainDelay;
    ScopedPointer<Slider> SlideFeedDelay;
    ScopedPointer<Slider> SlideTimeDelay;
    ScopedPointer<Slider> SlideGainReverb;
    ScopedPointer<Slider> SlideFeedReverb;
    ScopedPointer<Slider> SlideTimeReverb;
    ScopedPointer<Slider> Slide1waa;
    ScopedPointer<Slider> Slide2Waa;
    ScopedPointer<Slider> Slide3Waa;
    ScopedPointer<GroupComponent> groupComponent7;
    ScopedPointer<Slider> SlideMainGain;
    ScopedPointer<Slider> SlideFreqH;
    ScopedPointer<Slider> SlideFreqL;
    ScopedPointer<Slider> SlideFreqM;
```

```
ScopedPointer<Slider> SlideLow;
ScopedPointer<Slider> slider19;
ScopedPointer<Slider> SlideGainFlange;
ScopedPointer<Slider> SlideTimeFlange;
ScopedPointer<Slider> Slide3Flange;
ScopedPointer<Slider> Slide1ETC;
ScopedPointer<Slider> Slide2ETC;
ScopedPointer<Slider> Slide3ETC;
ScopedPointer<Label> label;
ScopedPointer<Label> label2;
ScopedPointer<Label> label3;
ScopedPointer<Label> label4;
ScopedPointer<Label> label5;
ScopedPointer<Label> label6;
ScopedPointer<Label> label7;
ScopedPointer<Label> label8;
ScopedPointer<Label> label9;
ScopedPointer<Label> label10;
ScopedPointer<Label> label11;
ScopedPointer<Label> label12;
ScopedPointer<Label> label13;
ScopedPointer<Label> label14;
ScopedPointer<Label> Param3;
ScopedPointer<Label> label16;
ScopedPointer<Label> label17;
ScopedPointer<Label> label18;
ScopedPointer<Label> label19;
ScopedPointer<Label> label20;
```

```
    ScopedPointer<Label> label22;
    ScopedPointer<Label> label23;
    ScopedPointer<Label> label25;
    ScopedPointer<Label> label26;
    ScopedPointer<Slider> SlideGainDistort;
    ScopedPointer<ToggleButton> toggleDistort;
    ScopedPointer<ToggleButton> toggleReverb;
    ScopedPointer<ToggleButton> toggleTremolo;
    ScopedPointer<ToggleButton> toggleRandom;
    ScopedPointer<ToggleButton> toggleFlange;
    ScopedPointer<ToggleButton> toggleDelay;
//==============================================================================
    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (CombinedGuiAudioProcessorEditor)
};

//[EndFile] You can add extra defines here...
//[/EndFile]

#endif   // __JUCE_HEADER_EB75F46DAAF9F729__
```

PluginProcessor.cpp

```
/*

  ==============================================================================

    This file was auto-generated!

    It contains the basic startup code for a Juce application.

  ==============================================================================
*/

#include "PluginProcessor.h"
#include "PluginEditor.h"

//==============================================================================
CombinedGuiAudioProcessor::CombinedGuiAudioProcessor():storageTremolo(1,1),storageDelay(1
,1),storageReverb(1,1),storageFlange(1,1),tremCounter2(0),tremCounter(1),reverbTDelay(4),flang
ecounter(0),flangeadder(1),flangebool(1),delayTDelay(1)
        ,boolDelay(false),boolDistort(false),boolReverb(false),boolTremolo(false),boolRandom(1),b
oolFlange(false)
{
}

CombinedGuiAudioProcessor::~CombinedGuiAudioProcessor()
{
}
```

```
//==============================================================================
const String CombinedGuiAudioProcessor::getName() const
{
    return JucePlugin_Name;
}

int CombinedGuiAudioProcessor::getNumParameters()
{
    return 0;
}

float CombinedGuiAudioProcessor::getParameter (int index)
{
    return 0.0f;
}

void CombinedGuiAudioProcessor::setParameter (int index, float newValue)
{
}

const String CombinedGuiAudioProcessor::getParameterName (int index)
{
    return String::empty;
}

const String CombinedGuiAudioProcessor::getParameterText (int index)
{
```

```
    return String::empty;
}

const String CombinedGuiAudioProcessor::getInputChannelName (int channelIndex) const
{
    return String (channelIndex + 1);
}

const String CombinedGuiAudioProcessor::getOutputChannelName (int channelIndex) const
{
    return String (channelIndex + 1);
}

bool CombinedGuiAudioProcessor::isInputChannelStereoPair (int index) const
{
    return true;
}

bool CombinedGuiAudioProcessor::isOutputChannelStereoPair (int index) const
{
    return true;
}

bool CombinedGuiAudioProcessor::acceptsMidi() const
{
   #if JucePlugin_WantsMidiInput
    return true;
   #else
```

```
    return false;
   #endif
}

bool CombinedGuiAudioProcessor::producesMidi() const
{
   #if JucePlugin_ProducesMidiOutput
    return true;
   #else
    return false;
   #endif
}

bool CombinedGuiAudioProcessor::silenceInProducesSilenceOut() const
{
    return false;
}

double CombinedGuiAudioProcessor::getTailLengthSeconds() const
{
    return 0.0;
}

int CombinedGuiAudioProcessor::getNumPrograms()
{
    return 0;
}
```

```
int CombinedGuiAudioProcessor::getCurrentProgram()
{
    return 0;
}

void CombinedGuiAudioProcessor::setCurrentProgram (int index)
{
}

const String CombinedGuiAudioProcessor::getProgramName (int index)
{
    return String::empty;
}

void CombinedGuiAudioProcessor::changeProgramName (int index, const String& newName)
{
}

//==============================================================================
void CombinedGuiAudioProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)
{
    // Use this method as the place to do any pre-playback
    // initialisation that you need..
}

void CombinedGuiAudioProcessor::releaseResources()
{
    // When playback stops, you can use this as an opportunity to free up any
```

```
   // spare memory, etc.
}


void CombinedGuiAudioProcessor::processBlock (AudioSampleBuffer& buffer, MidiBuffer&
midiMessages)
{
                int numsamples = buffer.getNumSamples();
                int samprate = getSampleRate();
                float* channelData = buffer.getSampleData (0);



                //flange


                        //tremolo
                        if (storageTremolo.getNumSamples() != numsamples)
                {
                        storageTremolo.setSize(1, numsamples);
                        storageTremolo.clear();
                        tremW =  storageTremolo.getSampleData(0);
                        tremR = storageTremolo.getSampleData(0);
                }


                        //delay
                        if (storageDelay.getNumSamples() !=
numsamples+getSampleRate()/delayTDelay)
                {
                        storageDelay.setSize(1, numsamples+getSampleRate()/delayTDelay);
                        storageDelay.clear();
```

```
                    delayW =  storageDelay.getSampleData(0) + samprate/(int)delayTDelay;
                    delayR = storageDelay.getSampleData(0);


                    }
                    //flange
                    if (storageFlange.getNumSamples() != numsamples+getSampleRate()/20)
                {
                    storageFlange.setSize(1, numsamples+getSampleRate()/20);
                    storageFlange.clear();


                    flangeW1 =  storageFlange.getSampleData(0)+samprate/10;
                    flangeR1 = storageFlange.getSampleData(0);
                    }
                //Reverb
                if (storageReverb.getNumSamples() !=
numsamples+getSampleRate()/reverbTDelay)
                {
                    storageReverb.setSize(1, numsamples+getSampleRate()/reverbTDelay);
                    storageReverb.clear();


                    //reverb
                    reverbW =  storageReverb.getSampleData(0) + samprate/reverbTDelay;
                    reverbR1 = storageReverb.getSampleData(0);
                    reverbR2 = storageReverb.getSampleData(0)+
storageReverb.getNumSamples()/reverbTDelay;
                    reverbR3 = storageReverb.getSampleData(0)+
storageReverb.getNumSamples()*2/reverbTDelay;
                    reverbR4 = storageReverb.getSampleData(0)+
storageReverb.getNumSamples()*3/reverbTDelay;
```

```
}

for (int sample = 0;sample<numsamples;sample++)
{
	//DISTORT
	if (boolDistort)
	{
	if (*channelData>bubble)
		*channelData = mult*bubble;
	else if (*channelData<-bubble)
		*channelData = - bubble *mult;
	else{
		*channelData = mult * *channelData;
	}
	}
	//DELAY
	if(boolDelay)
	{
		*channelData += delayGain * *delayR;
		// Save current output data into delay buffer
		*delayW = *channelData;
		// Increment pointers
		delayW++;
		delayR++;
		if (delayR > storageDelay.getSampleData(0) +
storageDelay.getNumSamples())
				delayR = storageDelay.getSampleData(0);
		if (delayW > storageDelay.getSampleData(0) +
storageDelay.getNumSamples())
```

```
                                        delayW = storageDelay.getSampleData(0);
                }
                //FLANGE
                if(boolFlange)
                {
                *flangeW1 = *channelData;
                if(sample>(numsamples*2/10) && sample<(numsamples*7/10))
                                *channelData += ( .5 **channelData + (.2* *flangeR1)); //
Controlled by vstSynthEditor::delayFBSlider .2/.8 for chorus

                                flangeW1++;// Increment pointers
                                flangeR1+= flangeadder;
                while (flangeR1 > storageFlange.getSampleData(0) +
storageFlange.getNumSamples()*7/10)
                        flangeR1 -=storageFlange.getNumSamples()*7/10;
                if(flangeR1 <
storageFlange.getSampleData(0)+storageFlange.getNumSamples()*2/10)//
+delay.getNumSamples()*0)
                        flangeR1 = flangeW1;
                if (flangeW1 > storageFlange.getSampleData(0) +
storageFlange.getNumSamples())
                        flangeW1 = storageFlange.getSampleData(0);
                }
                //RANDOMIZER
                if(boolRandom)
                {
                }

                //REVERB
                if(boolReverb)
```

```
                        {
                        *channelData +=
reverbGain*(*reverbR1+*reverbR2+*reverbR3+*reverbR4)/4;
        // Save current output data into delay buffer
                        *reverbW = *channelData;
        // Increment pointers
        reverbW++;
                        reverbR1++;
                        reverbR2++;
                        reverbR3++;
                        reverbR4++;


                        if (reverbR1 > storageReverb.getSampleData(0) +
storageReverb.getNumSamples())
                                        reverbR1 = storageReverb.getSampleData(0);
                        if (reverbW > storageReverb.getSampleData(0) +
storageReverb.getNumSamples())
                                        reverbW = storageReverb.getSampleData(0);
                        if (reverbR2 > storageReverb.getSampleData(0) +
storageReverb.getNumSamples())
                                        reverbR2 = storageReverb.getSampleData(0);
                        if (reverbR3 > storageReverb.getSampleData(0) +
storageReverb.getNumSamples())
                                        reverbR3 = storageReverb.getSampleData(0);
                        if (reverbR4 > storageReverb.getSampleData(0) +
storageReverb.getNumSamples())
                                        reverbR4 = storageReverb.getSampleData(0);
                        }
                                //TREMOLO
                        if(boolTremolo)
                        {
```

```
                *tremW = *channelData;
                //increase or decrease the output by the variable amount
                *channelData = *tremR*tremCounter/5;
                // Increment pointers
                tremW++;
                tremR++;
                if (tremR > storageTremolo.getSampleData(0) +
storageTremolo.getNumSamples())
                        tremR = storageTremolo.getSampleData(0);
                if (tremW > storageTremolo.getSampleData(0) +
storageTremolo.getNumSamples())
                        tremW = storageTremolo.getSampleData(0);
                }
                channelData++;

        }
        //misc code
                flangecounter++;
                if (flangecounter%10==0)
                {
                        flangeadder+=flangebool;
                                if(flangeadder >10)
                                {
                                        flangeadder--;
                                        flangebool = -1;
                                }

                                if(flangeadder <3)
                                {
```

```
                                        flangeadder++;
                                        flangebool = 1;
                                }
                        }
                        tremCounter+=tremCounter2;
                        if (tremCounter>20)
                        {
                                tremCounter--;
                                tremCounter2=-1;
                        }
                        else if (tremCounter<4)
                        {
                                tremCounter++;
                                tremCounter2=1;
                        }
    // ..do something to the data...
                buffer.addFrom(1,0, buffer ,0,0,numsamples);
  // In case we have more outputs than inputs, we'll clear any output
  // channels that didn't contain input data, (because these aren't
  // guaranteed to be empty - they may contain garbage).
  for (int i = getNumInputChannels(); i < getNumOutputChannels(); ++i)
  {
    buffer.clear (i, 0, buffer.getNumSamples());
  }

        float gainabs = powf(10,maingain/20);
        buffer.applyGain(gainabs);
}
```

```
//==============================================================================
bool CombinedGuiAudioProcessor::hasEditor() const
{
   return true; // (change this to false if you choose to not supply an editor)
}

AudioProcessorEditor* CombinedGuiAudioProcessor::createEditor()
{
   return new CombinedGuiAudioProcessorEditor (this);
}

//==============================================================================
void CombinedGuiAudioProcessor::getStateInformation (MemoryBlock& destData)
{
   // You should use this method to store your parameters in the memory block.
   // You could do that either as raw data, or use the XML or ValueTree classes
   // as intermediaries to make it easy to save and load complex data.
}

void CombinedGuiAudioProcessor::setStateInformation (const void* data, int sizeInBytes)
{
   // You should use this method to restore your parameters from this memory block,
   // whose contents will have been created by the getStateInformation() call.
}

//==============================================================================
```

```
// This creates new instances of the plugin..
AudioProcessor* JUCE_CALLTYPE createPluginFilter()
{
   return new CombinedGuiAudioProcessor();
}
```

PluginProcessor.h

```
/*

  ==============================================================================

    This file was auto-generated!

    It contains the basic startup code for a Juce application.

  ==============================================================================
*/

#ifndef PLUGINPROCESSOR_H_INCLUDED
#define PLUGINPROCESSOR_H_INCLUDED

#include "../JuceLibraryCode/JuceHeader.h"



//==============================================================================
/**
*/
class CombinedGuiAudioProcessor  : public AudioProcessor
{
public:

//==============================================================================
```

```
    CombinedGuiAudioProcessor();
    ~CombinedGuiAudioProcessor();

//==============================================================================
    void prepareToPlay (double sampleRate, int samplesPerBlock);
    void releaseResources();

    void processBlock (AudioSampleBuffer& buffer, MidiBuffer& midiMessages);

//==============================================================================
    AudioProcessorEditor* createEditor();
    bool hasEditor() const;

//==============================================================================
    const String getName() const;

    int getNumParameters();

    float getParameter (int index);
    void setParameter (int index, float newValue);

    const String getParameterName (int index);
    const String getParameterText (int index);

    const String getInputChannelName (int channelIndex) const;
```

```
    const String getOutputChannelName (int channelIndex) const;
    bool isInputChannelStereoPair (int index) const;
    bool isOutputChannelStereoPair (int index) const;

    bool acceptsMidi() const;
    bool producesMidi() const;
    bool silenceInProducesSilenceOut() const;
    double getTailLengthSeconds() const;


//==============================================================================
    int getNumPrograms();
    int getCurrentProgram();
    void setCurrentProgram (int index);
    const String getProgramName (int index);
    void changeProgramName (int index, const String& newName);


//==============================================================================
    void getStateInformation (MemoryBlock& destData);
    void setStateInformation (const void* data, int sizeInBytes);

        //==========================================================uservariables
        //distort variables
        float gain;
        float bubble;
        float mult;
        //delay variables
```

```
        float* delayW;
float* delayR;
        float delayGain;
        float delayTDelay;
        //reverb variables
        float* reverbW;
        float reverbGain;
float* reverbR1;
float* reverbR2;
float* reverbR3;
float* reverbR4;
        int reverbTDelay;
        //random variables
        //tremolo variables
        float* tremW;
        float* tremR;
        int tremMultiplier;
        int tremCounter;
        int tremCounter2;
        //flange variables
        float* flangeW1;
float* flangeR1;
        int flangeadder;
        int flangebool;
        int flangecounter;

        //misc variables
        AudioSampleBuffer storageTremolo;
```

```
        AudioSampleBuffer storageDelay;
        AudioSampleBuffer storageReverb;
        AudioSampleBuffer storageFlange;
        float maingain;
        bool boolDelay;
        bool boolFlange;
        bool boolReverb;
        bool boolRandom;
        bool boolDistort;
        bool boolTremolo;

private:

//==============================================================================
    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (CombinedGuiAudioProcessor)
};


#endif  // PLUGINPROCESSOR_H_INCLUDED
```