REAL-TIME DSP (ECS732P)

Final Assignment – Digital Audio Effects on Bela

Kranthi Yanamandra (170797647)

Overview:

This project explores four digital audio effects typically observed in guitar effects pedals and amplifiers. These are:

1. Vibrato
2. Tremolo
3. Ring Modulation
4. Distortion

The effects are applied in real-time to an incoming audio signal. Each of these effects operate with certain parameters that can be fine-tuned by using four different potentiometers. Distortion was implemented using two methods, namely soft exponential clipping and half wave rectification. The final output signal is the combination of each of these effects. As such, each of these effects can be heard in the output signal to varying extents, depending on their corresponding potentiometer values. The following sections illustrate the theory behind the effects, some design decisions, implementation techniques and an evaluation on the system that was built.

Background:

Audio signals are electronic representations of sound waves which travel through air, consisting of compressions and rarefactions. To perform signal processing on audio using digital circuits such as microprocessors and computers, a digital representation is used which expresses these sound waves as a sequence of numbers. Digital audio effects comprise of many such signal processing techniques to intentionally modify how a musical instrument or an audio source sounds. By chaining together different effects, musicians can achieve a unique tone.

Work on audio effects dates back to the 1940s when recording engineers used reel tapes to create delays, echoes and other sounds effects. The first known ‘stompbox’ or effects processor was built by Harry DeArmond in 1948, which created a tremolo effect by running electric current of the audio signal through a liquid. With the advent of technology, today’s audio effects come in both physical and digital forms. In the physical form, effects are usually rack-mounted devices that have cables running to and from a mixing console or a guitar effects pedal which receives a signal from the instrument and then modifies it. On the digital side, audio effects are found in most music recording software packages such as Ableton and Logic Pro X. In the context of the present project, we look at some of these effects as used typically by guitarists.

Theory and Implementation:

This section provides a brief overview of the theory behind each effect and how these were implemented in the project.

1. ***Vibrato – Vibrato is defined as a small, quasi-periodic variation in the pitch of a tone. It is a frequency modulation effect.***

Vibrato is based on the concept that a time-varying delay changes the pitch of a sound. Hence, vibrato is implemented digitally through the use of a modulated delay line, whose delay length changes over time under the influence of a low frequency oscillator. Vibrato is characterised by its *frequency* (how often the pitch changes) and *width* (total amount of pitch variation). It is generally accompanied by some degree of amplitude modulation (tremolo), as is done in this project as well. It has to be noted that delay alone cannot introduce a pitch shift. Suppose the length $M[n]$ of the delay line does not change. Then at every sampling period n, exactly one sample $x[n]$ goes in and one sample $y[n\text{-}M]$ comes out. The sound will be delayed but otherwise identical to the original. But suppose that the length $M[n]$ decreases each sample by an amount Δm:

$$M[n]=M_{\max} - (\Delta m)n$$

Examining the output of the delay line, we can see that each new input sample n moves the output by $(1 + \Delta m)$ samples:

$$y[n] = x[n - M_{\max} + (\Delta m)n] = x[(1 + \Delta m)n - M_{\max}]$$

This would mean that the playback rate from the buffer is $(1 + \Delta m)$ times the input rate. Correspondingly, the frequencies in the input signal $x[n]$ will all be scaled upwards by a factor of $(1 + \Delta m)$. As can be seen from the above equation, it is only defined for integer samples of x, i.e. when $(1 + \Delta m)n$ - M_{max} is an integer. In general, this is not always the case for a modulated delay line where delay lengths change gradually from sample to sample, so *interpolation* must be used to approximate non-integer values of x.

This interpolation could be linear, second-order or cubic. Linear interpolation is fast but at the cost of lower quality. Cubic interpolation produces better and cleaner results at the expense of more computation. Both of these methods were tested in the present project and it was decided that cubic interpolation would be used, since there was no evident lag or latency and it produced better results. The following code snippet illustrates this:

```
int sample_1 = (int)floorf(gReadPointer);

int sample_2 = (sample_1 + 1) % DELAY_BUFFER_SIZE;

int sample_3 = (sample_2 + 1) % DELAY_BUFFER_SIZE;
```

```
int sample_0 = (sample_1 - 1 + DELAY_BUFFER_SIZE) % DELAY_BUFFER_SIZE;

float fraction = gReadPointer - floorf(gReadPointer);

float fracsquare = fraction * fraction;

float a0 = -0.5f * gDelayBuffer[sample_0] + 1.5f * gDelayBuffer[sample_1] -
1.5f * gDelayBuffer[sample_2] + 0.5f * gDelayBuffer[sample_3];

float a1 = gDelayBuffer[sample_0] - 2.5f * gDelayBuffer[sample_1] + 2.0f *
gDelayBuffer[sample_2] - 0.5f * gDelayBuffer[sample_3];

float a2 = -0.5f * gDelayBuffer[sample_0] + 0.5f * gDelayBuffer[sample_2];

float a3 = gDelayBuffer[sample_1];

gInterpolatedSample = a0 * fraction * fracsquare + a1 * fracsquare + a2 *
fraction + a3;
```

As mentioned earlier, a low frequency oscillator was used to modulate the input signal. The frequency of this LFO is user-selectable with the aid of a potentiometer connected to one of the analog inputs on Bela. The higher this frequency, the more rapid is the vibrato. The current delay for a particular iteration of the render() function is calculated using the formula:

```
gCurrentDelay = gSweepWidth * lfo_vibrato;
```

This changes for each iteration, thus changing the position of the read pointer and consequently the write pointer. The write pointer moves at a constant rate but the read pointer moves at different rates depending on the LFO settings, sweep width and the current delay. The read pointer position is calculated with respect to the write pointer as follows.

```
gReadPointer = fmodf((float)gWritePointer - (float)(gCurrentDelay*context->
audioSampleRate)+(float)(DELAY_BUFFER_SIZE) - 3.0, (float)DELAY_BUFFER_SIZE);
```

Once the write pointer reaches the end of the delay line size, it is wrapped back to the beginning of the buffer:

```
if(++gWritePointer >= DELAY_BUFFER_SIZE)
gWritePointer = 0;
```

Ultimately, a vibrato effect is induced. This is stored in a variable called output_1.

2. ***Tremolo – Tremolo is an amplitude modulation effect, where the amplitude/ volume of the signal changes over time, creating a shuddering or pulsating effect.***

A tremolo effect is achieved by multiplying two signals together, where the amplitude of one signal is continuously modulated by the amplitude of the other. Digitally, this can be achieved by multiplying the desired signal by a LFO. While LFOs are in the range of 0 – 20 Hz, it was found that a cap of 15 Hz works well for

tremolo. In the current implementation, this LFO frequency is user-selectable with the use of a potentiometer.

The following code snippet illustrates the implementation of a simple tremolo effect:

```
float lfo_tremolo = sinf(glfoPhase_tremolo) * 0.5 + 0.5f;
// Keep track and wrap the phase of the sinewave
gLfoFrequency_2 = map(analogRead(context, n, 2), 0.0, 1.0, 0.0, 15.0);
glfoPhase_tremolo +=  2.0 * M_PI * gLfoFrequency_2 * gInverseSampleRate;
if(glfoPhase_tremolo > 2.0 * M_PI)
glfoPhase_tremolo -= 2.0 * M_PI;
output_2 =  input * lfo_tremolo;
```

Depending on the frequency read in from the potentiometer, the phase of the LFO changes accordingly. If the phase exceeds 2π, it is wrapped back. One parameter that was tested but not considered during the implementation was the depth of the effect. Depth defines the amount of modulation the LFO applies on the original signal. Introducing this parameter resulted in audible artefacts and unpleasant clipping. To counter this, instead of multiplying the signals, other mathematical formulae were considered to apply the effect. Two of these were:

`output_2 = (1 - gDepth) + gDepth * input * lfo_tremolo;` and

`output_2 = input * (1.0 – gDepth * lfo_tremolo);`

These were slight modifications to traditionally used formulae and were considered because the output would never exceed 1. However, as mentioned above, these resulted in undesirable artefacts and clicks. It was therefore decided that the depth parameter would not be considered while applying the effect.

3. ***Ring Modulation – Ring modulation is an effect that produces unusual, and sometimes discordant and dissonant sounds. This is achieved by multiplying an input signal with a periodic carrier signal.***

The difference between amplitude modulation and ring modulation is that in the latter, the modulating signal's frequency is in the audible range as opposed to a LFO. This results in the formation of sidebands and the output is usually not harmonic, even for simple inputs. The ring modulator has a characteristic, gong-like or robotic sound to it which results from the particular properties of the input and carrier signals. Consider a simple case, where each of the signals is a single sinusoid:

$$x[n] = \cos(\omega n),\ m[n] = \cos(\omega_c\, n)$$

The cosine of the sum or difference of two angles may be given by

$$\cos(A+B) = \cos(A)\cos(B) - \sin(A)\sin(B),$$

$$\cos(A\text{-}B) = \cos(A)\cos(B) + \sin(A)\sin(B).$$

So,

$$m[n]x[n] = \cos(\omega_C n)\cos(\omega n) = [\cos((\omega_C - \omega)n) + \cos((\omega_C + \omega)n))] / 2$$

This means that multiplying two sinusoids results in a *sum* and *difference* of frequencies. For example, if our input was a 400Hz sine wave and the carrier was a 100Hz sine wave, the result would not be the original frequencies at all, but instead would be sinusoids at 300Hz and 500Hz. Ring modulation can also be considered as a frequency domain convolution with the carrier spectrum.

In the current implementation, the carrier signal frequency has been made user-selectable by using a potentiometer. The following code snippet illustrates the digital implementation of a ring modulator. There is an internal LFO that in turn modulates the carrier signal:

```
output_3 = input * sinf(2.0 * M_PI * gCarrierPhase);

gCarrierFrequency = map(analogRead(context, n, 3), 0.0, 1.0, 100, 500);

float lfo_ringmod = sinf(glfoPhase_ringmod) * 0.5 + 0.5f;

glfoPhase_ringmod += gLfoFrequency_1 * gInverseSampleRate;

if(glfoPhase_ringmod >= 1)

      glfoPhase_ringmod -= 1.0;

gCarrierPhase += (gCarrierFrequency + gSweepWidth * lfo_ringmod) *
gInverseSampleRate;

if(gCarrierPhase >= 1.0)

     gCarrierPhase -= 1.0;
```

4. ***Distortion – Distortion is a form of signal processing used to alter the sound of amplified electronic music instruments, usually by increasing their gain, producing a bright and harsh tone.***

Distortion is achieved by altering the instruments sound by clipping the signal (pushing it past its maximum, which shears off the edges of these waves) and adding harmonic and inharmonic overtones. In the current implementation, two kinds of digital distortions were explored, namely soft exponential clipping and half wave rectification.

Soft exponential distortion is achieved by using the formula:

$$f(x) = \text{sgn}(x)(1-e^{-|Gx|})$$

In this equation, the output asymptotically approaches the clipping point as the input gets larger but never reaches it. The amount of distortion added to the sound increases smoothly as the input level increases. In the current implementation the input is multiplied with a gain factor that is user-selectable with the help of a potentiometer. The overall output from this is halved to ensure that the audio does not become too loud. The following code snippet illustrates this:

```
float gain = powf(10.0f, gGain/20.0f);
float in = input_ * gain;
gGain = map(analogRead(context, channel, 0), 0.0, 1.0, -24.0, 24.0);
if(in > 0.0)
{
      output_4 = 1.0 - expf(-in);
}
else
{
      output_4 = - 1.0 + expf(in);
}

output_4 /= 2.0;
```

Another form of distortion is achieved by using half wave or full wave rectification. In the former, the positive half of a waveform is omitted, or in digital terms, set to zero. In the latter, the negative cycle of the waveform is inverted to the positive side. The half-wave rectifier is mathematically equivalent to the average of the input and its full-wave rectified version. Thus, it contains both the original fundamental frequency and its octave harmonic. This is achieved in code as follows:

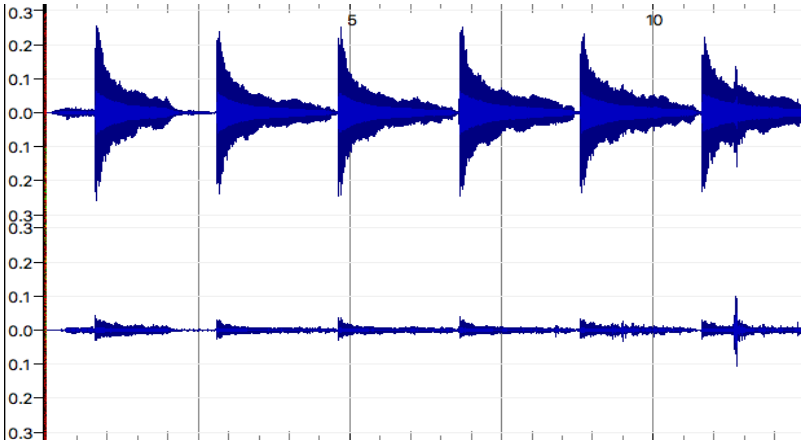
```
output_5 = 0.5 * (fabs(input) + input);
```

Evaluation:

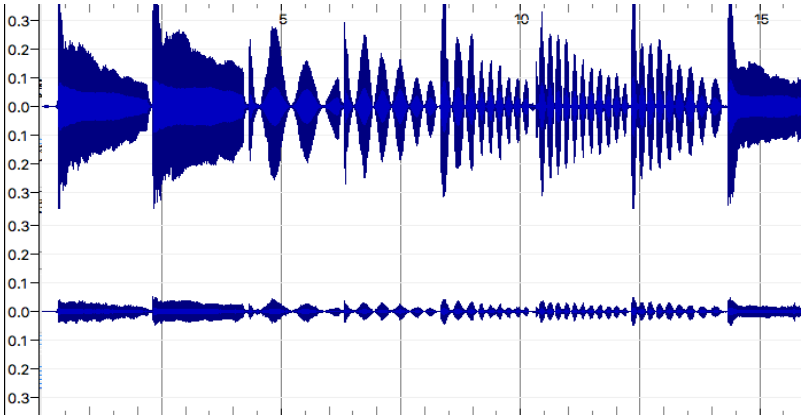
The system was evaluated for each effect individually as well as by combining them as an effects chain. They work as intended, with the vibrato changing the pitch of the input signal, the tremolo modulating the amplitude of the signal, the ring modulator creating side bands and introducing metallic sounds, and the distortions creating an overdrive guitar tone. All of the potentiometers worked as expected by changing their respective effects' parameters.

Multiple input signals were used to evaluate the system, which included guitar sounds, piano sounds and human voices. For the sake of simplicity, the current evaluation shown is that of a clean guitar playing a standard D note, plucked at regular intervals. The following figures illustrate each of these effects in action:

Original signal (time on x-axis and amplitude on y-axis):

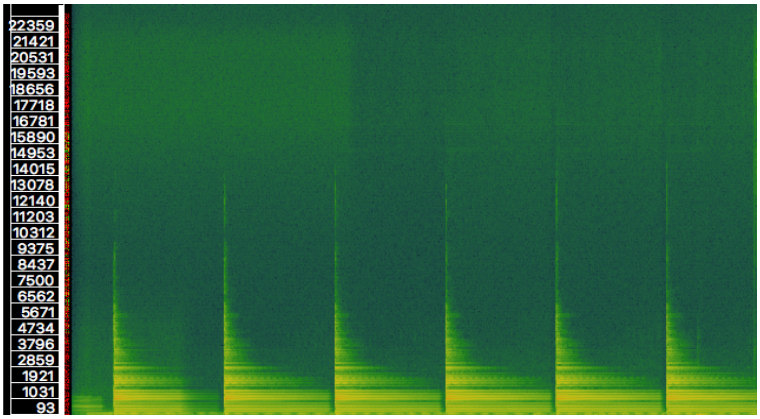


Output with Tremolo applied (time on x-axis, amplitude on y-axis):

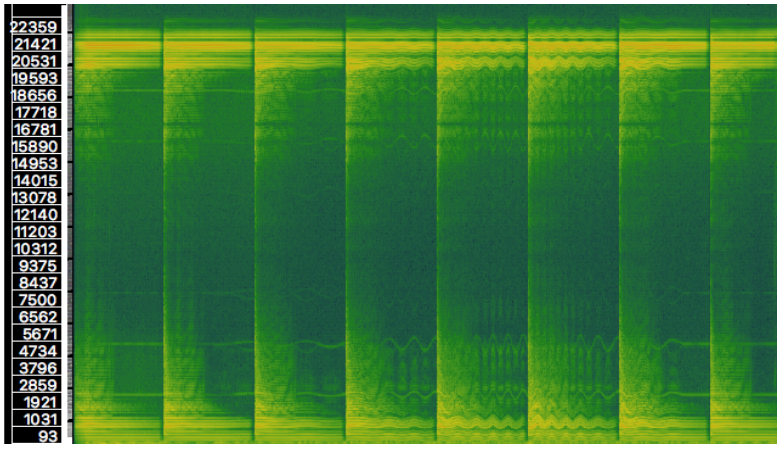


As can be seen in this figure, the amplitude of the signal drops and rises with time. The rate at which this happens also changes over time, as the potentiometer values associated with it were changed.

Original signal (spectrogram, with time on x-axis and frequency on y-axis):

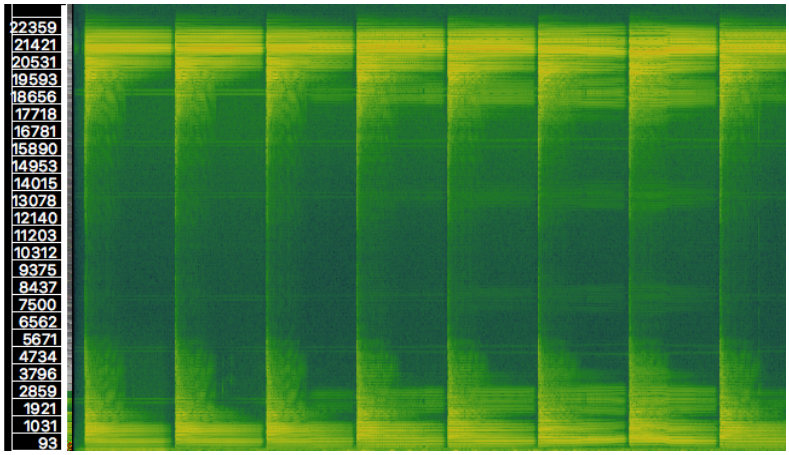


Output with Vibrato applied (spectrogram, with time on x-axis and frequency on y-axis):



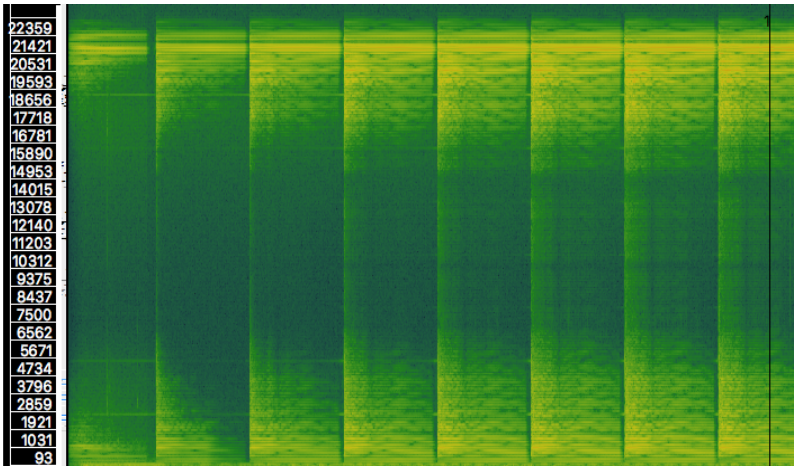
As can be seen in this spectrogram, the vibrato effect introduces frequency modulation in the frequency bin range of 500 Hz to 5000 Hz.

Output with Ring Modulation applied:



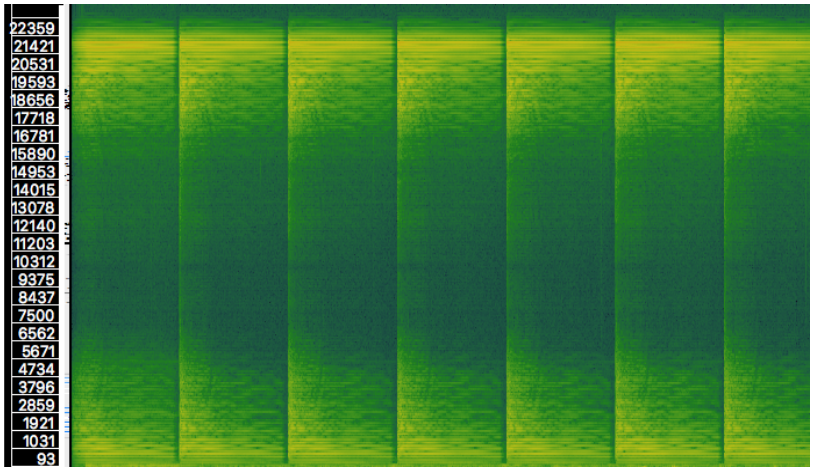
Ring modulation introduces sidebands at the sum and difference of the modulating and carrier frequencies.

Output with Soft Exponential Clipping applied:



Comparing this to the original signal's spectrogram, it can be seen that with this kind of distortion, the exponential function grows gradually more non-linear as the input gain increases, asymptotically approaching the clipping point. Therefore, with time, we can see that energy in the frequency bins is increasing as well.

Output with Half Wave Rectification applied:



Half wave rectification sets the negative cycles of a wave to zero, causing a discontinuity / abrupt transition in the signal, hence inducing a distortion effect. Since there was no parameter to control this effect, it stays the same throughout the signal.

Conclusion and Further work:

The world of digital audio and the possibilities of audio effects are endless. This project focused mainly on the well-established effects found more commonly in the musical community. Through a reasonable amount of code and computational power, these effects were applied in real-time to various audio signals. To make it more interesting, the effects were chained together and external controls in the form of potentiometers were introduced to vary the extent of each of these effects. Possible further improvements include but are not limited to:

1. Use of different kinds of LFOs to modulate the signal, such as square or triangle oscillators.
2. Introducing further parameters such as depth and intensity of the effect that are user-controllable.
3. Use of other interpolation methods in combination with the currently implemented cubic interpolation.
4. Use of lowpass filtering to avoid aliasing at frequencies higher than the Nyquist frequency.

References:

[1] Pirkle, W. (2012). Designing Audio Effect Plug-Ins in C++.

[2] J. Reiss, “ECS730 Digital Audio Effects”, University Lecture Material, 2018.

[3] R. Stewart, “ECS732P Real-Time DSP”, University Lecture Material, 2018.

[4] Logging Data in Bela.io - Bela online forum. Available at https://forum.bela.io/d/304-logging-data-in-bela-io

[5] McPherson, A. & Reiss, J. (2014). Audio Effects: Theory, Implementation and Application.

[6] Audio Signal Processing, CCRMA. Available at https://ccrma.stanford.edu/overview/dsp.html

[7] Signal Processing, Stack Exchange. Available at https://dsp.stackexchange.com/