Audio Engineering Society

Convention Paper 9889

Presented at the 143rd Convention
2017 October 18–21, New York, NY, USA

An "Infinite" Sustain Effect Designed for Live Guitar Performance

Mark Rau[1] and Orchisama Das[1]

[1]*Center for Computer Research in Music and Acoustics (CCRMA), Stanford University*

Correspondence should be addressed to Mark Rau (mrau@ccrma.stanford.edu)

ABSTRACT

An audio effect to extend the sustain of a musical note in real-time is implemented on a fixed point, standalone processor. Onset detection is used to look for new musical notes, and once they decay to steady state, the audio is looped indefinitely until a new note onset occurs. To properly loop the audio, pitch detection is performed to extract one period and the new output buffer is written in a phase aligned manner.

1 Introduction

While guitar effects are a well developed area of signal processing with commercially available analog effect pedals and digital plug-ins, a cheap digital implementation of a guitar sustain pedal is a novel idea. Effects such as compressors can achieve a sustain like effect by non-linearly altering the level of guitar note over time, making it sound like the note sustains for a longer time. However, compressors can only create this effect as long as the physical guitar string is producing sound, and cannot sustain the note indefinitely. There have been attempts to produce an infinite sustain effects using hardware implementations which provide feedback to the guitar string through a transducer [1]. However, these methods are expensive and specific to instruments, limiting flexibility.

We propose a cheap, efficient "infinite" sustain pedal implemented in fixed point arithmetic for use on a dedicated digital signal processor which could be used in a guitar foot pedal. The goal of the effect is to have real-time monophonic sustain. This is achieved by detecting note onset, followed by performing pitch detection, and continuously re-writing the output buffer with part of the input signal corresponding to the period of the fundamental frequency detected. A note sustains as long as the user does not play a new note.

To implement our "sustain" algorithm, we check each incoming audio input buffer for a note onset. If an onset is detected, we wait for a number of buffers until we are certain that the present buffer represents the steady state portion of the guitar note. Then, we extract one period of the fundamental waveform by detecting its pitch. We loop this constantly in the output buffer with phase alignment, until a new onset is detected. This process is summarized the following block diagram shown in Figure 1.

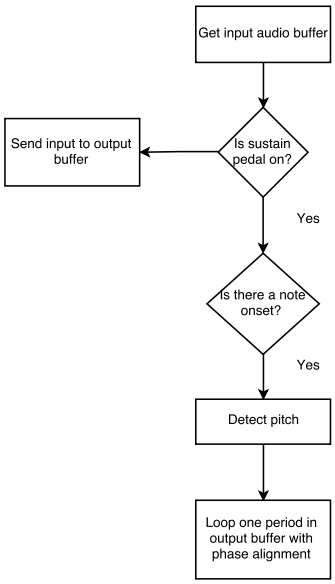

Fig. 1: Flowchart of proposed algorithm.

2 Methods

The sustain effect is implemented in 16-bit fixed point at a sampling rate of 48000, on a microprocessor known as the *DSP Shield* [2].[1] The *DSP Shield* has an embedded Texas Instruments DSP processor - TMS320C5535 [3]. To achieve real-time performance, a buffer length of 1024 samples is used. This results in a delay of 21 ms, within the range of real-time performance. A hardware controller has been developed to enable user control over parameters related to the onset detection and audio modulation.

2.1 Onset Detection

Onset detection is performed using a leaky integrator level detector [4]. The update equation for the level detector is:

$$\begin{aligned} &\text{if } x(n) > \hat{\lambda}: \\ &\quad \hat{\lambda} = \hat{\lambda} + (1 - e^{\frac{-1}{\tau_a f_s}})(|x(n)| - \hat{\lambda}) \\ &\quad \text{FoundOnset} = \text{true} \\ &\text{else:} \\ &\quad \hat{\lambda} = \hat{\lambda} + (1 - e^{\frac{-1}{\tau_r f_s}})(|x(n)| - \hat{\lambda}) \\ &\quad \text{FoundOnset} = \text{false} \end{aligned} \tag{1}$$

where $x(n)$ is the current sample, $\hat{\lambda}$ is the level estimate, τ_a is the attack time constant, τ_b is the decay time constant, f_s is the sample rate, and **FoundOnset** is a flag to determine if an onset is found. When the input level is greater than the level estimate, the level detection is realized as an exponential approaching the input level. Thus, τ_a is set to a short time of roughly 10ms to quickly detect an onset. When the input level is lower than the level estimate, a slowly decaying exponential is the result. This occurs because τ_r is set to a high value of roughly 100ms. An onset is detected when the input level is greater than the previous level estimate. The response of the level estimate to a guitar signal is shown in Figure 2. This method of onset detection is reliable and requires very few computations.

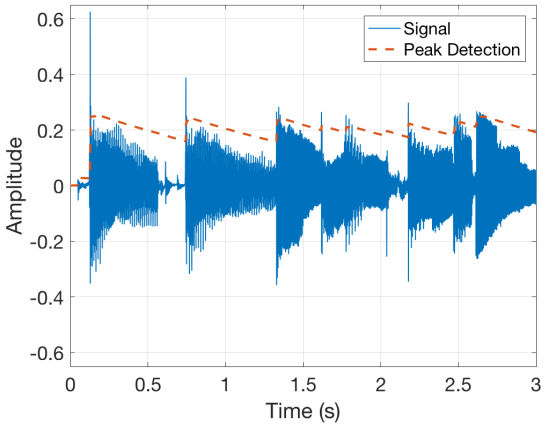

Fig. 2: Example of leaky integrator peak detection for onset detection.

2.2 Pitch Detection

A simple algorithm has been used for pitch detection based on the method proposed in [5] which makes use of the Average-Magnitude Difference Function

[1] https://web.stanford.edu/group/kovacslab/cgi-bin/index.php?page=dsp-shield

(AMDF). Although more robust pitch detection algorithms exist, we chose this method since it requires minimal computations and can easily be implemented in real-time with fixed point processing without worrying about overflow and loss in precision.

Our buffer length is long enough to accommodate at least one period of the lowest note of a guitar in standard tuning (82.41 Hz) at a sampling frequency of 48 kHz. We are only concerned with predicting the pitch of the fundamental frequency so we wish to filter out any unnecessary upper harmonics. The highest fundamental of a guitar is around 1400 Hz so we use a low pass filter to remove harmonics above this frequency. The signal is filtered with a low pass filter designed by the Parks-McClellan method [6] in MATLAB with a passband of 1kHz and a stop-band of 1.5 kHz. This resulted in a filter order of 187. Although this has a large number of real multiplications, an FIR filter ensures that there is no phase distortion. The frequency response of the filter is given in Figure 3.

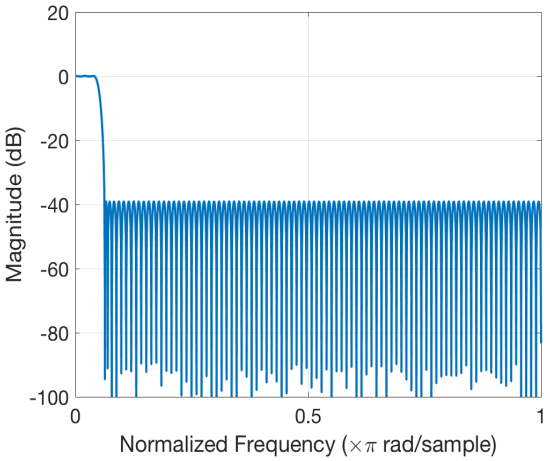

Fig. 3: Magnitude response of the low pass filter.

Let a sample in the signal be denoted as *x(n)*, where $n = 0,1,\ldots,$ N-1 and N = 1024. The Average Magnitude Difference Function is defined as:

$$D(\tau) = \frac{1}{N}\sum_{n=1}^{N} |x(n) - x(n-\tau+\tau_{min})| \qquad (2)$$
$$\tau = 0,1,\ldots,\tau_{max}-\tau_{min}$$

Thus a difference signal D, is formed by delaying the signal by various lags, subtracting the delayed waveform from the original, and summing the magnitude of the differences between sample values. If the signal is quasi-periodic with a period p, then D will have minima whenever $\tau - \tau_{min}$ equals any multiple of p. However, we must ensure that $\tau_{max} > p$. The lowest E string has the largest periodicity, $p_{max} = \frac{48000}{82.41} \approx 583$. To be safe we set $\tau_{max} = 600$. Similarly, the highest possible note on the guitar is about 1381.50 Hz, so the lowest period is $p_{min} = \frac{48000}{1381.50} \approx 35$, hence we set $\tau_{min} = 32$. An example of the AMDF showing regular dips is shown in Figure 4.

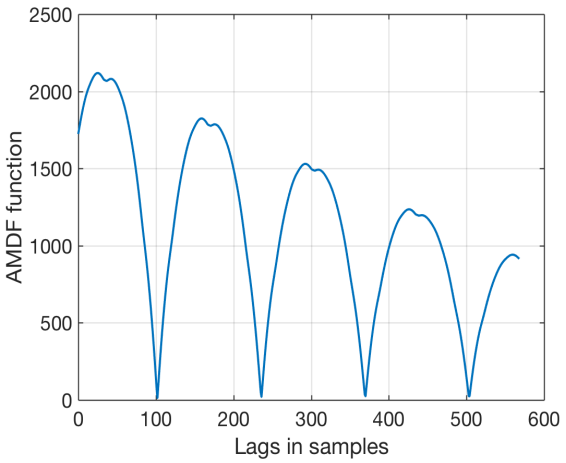

Fig. 4: Average Magnitude Difference Function.

To calculate period, we look for the first local minimum in AMDF, then perform parabolic interpolation for more accurate peak position detection, and add τ_{min} to get the final pitch period.

2.2.1 Robust Pitch Estimate

It is quite likely that spurious dips in the AMDF might lead to incorrect pitch estimation. For more robust pitch estimate, we detect pitch for four consecutive buffers in the steady state. We calculate the mean absolute deviation in detected pitch as:

$$\delta = \frac{1}{4}\sum_{i=1}^{4} |p_i - \mu_p| \qquad (3)$$

where p_i is the pitch estimate for the ith buffer and μ_p is the mean pitch estimate of all four buffers. If δ is less than a small threshold, then we can confidently conclude that we have the correct pitch estimate. If not, we keep checking each following buffer until four

consecutive buffers give a small enough value of δ. If this condition is satisfied, then the pitch period is taken to be that of the last buffer in the set of consecutive buffers.

2.3 Buffer Looping

One might wonder why we did not just loop the steady state output buffer as long as we wanted the note to sustain. If we did so, the phases of two consecutive output buffers would not be aligned, and hence we would hear clicks in the audio.

To phase align consecutive buffers, we find the pitch period (in samples) of the steady state buffer we want to loop. We extract exactly one fundamental period from the steady state buffer and loop it in the output buffer. Let us denote the extracted fundamental as x_p, where p stands for the pitch period in samples. We keep track of the index of the sample in x_p that was sent out as the last sample of the previous buffer. Let this index be denoted as n. We ensure that the first sample of the current buffer follows the last sample of the previous buffer. Since the last sample of the previous buffer was $x_p[((n))_p]$, the first sample in the current buffer would simply be $x_p[((n+1))_p]$.[2]

This explains why we need to detect pitch and extract the fundamental. In the case when $n = p-1$, $x_p[((n+1))_p]$ will wrap back around to $x_p[0]$. If x_p was not periodic with period p, then phase alignment would not have been possible.

2.4 Physical Controller

The *DSP Shield* includes analog inputs whose values can be read from the uploaded program, allowing them to be used as an interface for physical control of the system's parameters. A potentiometer is wired as a voltage divider with one side connected to an analog input. This value of this analog input is mapped to the onset detection release time constant, τ_r, allowing the user to easily control the sensitivity of the pedal's note onset detection. A switch is also wired to an analog input to act as an on/off switch for the effect. An LED is wired with the switch to show the user if the pedal is on. The potentiometer and switch were mounted on a typical guitar pedal enclosure as shown in Figure 5.

[2]The sign $((\))_p$ denotes a modulo p operation.

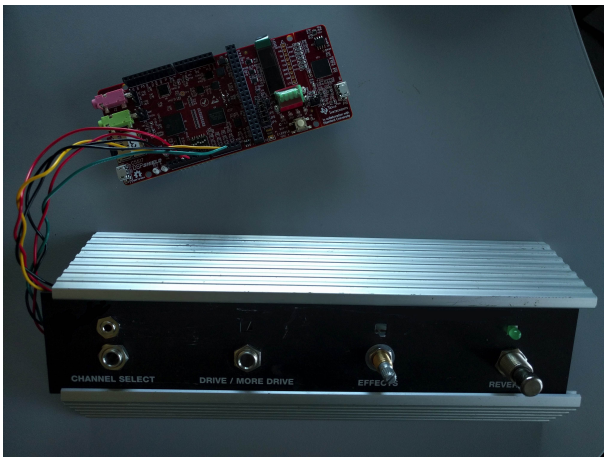

Fig. 5: Physical controller connected to the DSP Shield.

3 Results

An example of a sustained guitar note is shown in Figure 6. The upper graph shows the natural decay of the note, while the lower graph shows the result of the note being passed through the sustain pedal. It is to be noted that the steady state region is looped for an indefinite amount of time, creating the "infinite" sustain. The looping continues until the pedal is switched off or a new note is detected.

To evaluate the effectiveness of the sustain pedal, an informal listening test was conducted. Overall, it was agreed that the pedal is well suited for use in live performance and could be a valuable tool for a guitarist, especially when used in conjunction with existing guitar effects. A demo video with audio examples and link to the code repository can be found at [7].

4 Discussion

While the pedal performs reasonably well most of the time, there are a few pitfalls. Even with the the extra effort to ensure that correct pitch is detected, the algorithm will occasionally predict an incorrect pitch, which results in a wrong note being sustained. This is more likely to occur at the extreme ranges of the guitar and when the signal is noisy. The pitch detection could likely be improved by using more sophisticated methods such as the YIN algorithm [8], but such algorithms are difficult to implement in fixed point and may compromise the efficiency of the effect. Additionally, since

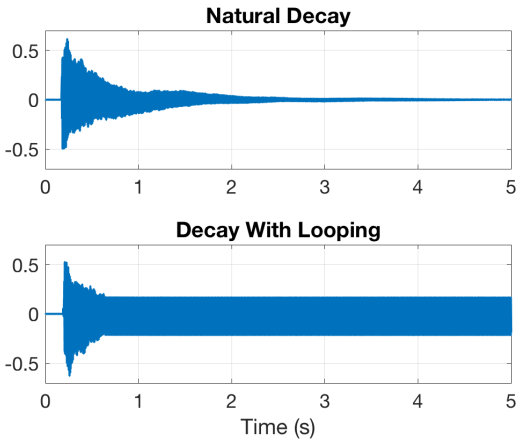

Fig. 6: Recording of a plucked guitar note measured directly (above), and through the sustain pedal (below).

the pitch detection and looping is sample locked, there is a small frequency error which can become noticeable at high frequencies. Future work includes implementing a dropped sample method to correct for the pitch errors. This method entails dropping one sample from the occasional buffer, causing the pitch to be perceived differently. Moreover, we wish to add a small amount of amplitude modulation to make the sustain sound more musically interesting.

To make the effect more widely available, a software implementation is being written in C++. This implementation will not be limited by fixed-point arithmetic or the physical constrains of the DSP shield. The source code will be released at [7] once the implementation is complete.

5 Summary

In this paper, we have described how to make a "do-it-yourself" guitar sustain pedal that runs in real-time on a fixed point DSP processor. The code is efficient and simple, and the controller is easy to hack together. In the process, we have put together C++ classes to do onset detection and pitch detection in real-time. Personally, we are not aware of any other open-source, cheap implementation of a digital sustain pedal for the guitar, which motivated us to take up this particular project. We hope that this work is useful for guitar enthusiasts who like playing around with effects. We believe it is a useful addition to a guitarist's effects pedal repertoire and hope to improve it and use it in our own performances.

References

[1] Hoover, A. A. and Osborne, G. T., "Electro-mechanical transducer which couples positive acoustic feedback into an electric amplified guitar body for the purpose of sustaining played notes," 1989, US Patent 4,852,444.

[2] Mujica, F. A., Esposito, W. J., Gonzalez, A., Qi, C. R., Vassos, C., Wieman, M., Wilcox, R., Kovacs, G. T., and Schafer, R. W., "Teaching digital signal processing with Stanford's Lab-in-a-Box," in *Signal Processing and Signal Processing Education Workshop (SP/SPE), 2015 IEEE*, pp. 307–312, IEEE, 2015.

[3] *TMS320C55x DSP Library Programmer's Reference*, Texas Instruments, 2013.

[4] Zolzer, U., *DAFX: Digital Audio Effects, 2nd Edition*, John Wiley & Sons Ltd, Hamburg, Germany, 2011.

[5] Ross, M., Shaffer, H., Cohen, A., Freudberg, R., and Manley, H., "Average magnitude difference function pitch extractor," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 22(5), pp. 353–362, 1974.

[6] McClellan, J., Parks, T., and Rabiner, L., "A computer program for designing optimum FIR linear phase digital filters," *IEEE Transactions on Audio and Electroacoustics*, 21(6), pp. 506–526, 1973.

[7] Rau, M. and Das, O., "Infinite Sustain Effect Supplemental Materials for AES 143," 2017, available: `https://ccrma.stanford.edu/~orchi/220C/index.html`.

[8] De Cheveigné, A. and Kawahara, H., "YIN, a fundamental frequency estimator for speech and music," *The Journal of the Acoustical Society of America*, 111(4), pp. 1917–1930, 2002.