A MICRO-CONTROLLED DIGITAL EFFECT UNIT FOR GUITARS

Geovani Cardozo Alves *

Departamento Acadêmico de Eletrotécnica,
Universidade Tecnologica Federal do Paraná
Curitiba, Brazil
geovani_ca@hotmail.com

Marcelo de Oliveira Rosa

Departamento Acadêmico de Eletrotécnica,
Universidade Tecnologica Federal do Paraná
Curitiba, Brazil
mrosa@utfpr.edu.br

ABSTRACT

Here we present a micro-controlled digital effect unit for guitars. Different from other undergraduate projects, we used high-quality 16-bit Analog-to-Digital (A/D) and Digital-to-Analog (D/A) converters operating at 48kHz that respectively transfer data to and from a micro-controller through serial peripheral interfaces (SPIs). We discuss the design decisions for interconnecting all these components, the project of anti-aliasing (low-pass) filters, and additional features useful for players. Finally, we show some results obtained from this device, and discuss future improvements.

1. INTRODUCTION

Analog guitar effects became very popular from 70's to 90's as an artifact that musicians could imprint their own personality touch in their sounds [1]. Once that the micro-controllers become popular among engineers, new and combined effects could be added into the so called digital effect units, which allow a single device to have multiple effects. Nowadays, powerful micro-controllers (with embedded DSP units) can execute sophisticated signal processing algorithms, creating configurable digital audio effect units.

Most of available micro-controlled boards for education purpose come with A/D converters (ADCs) operating at reasonable sampling rates but coding amplitudes at 12 bits. Usually they do not have any D/A converter (DAC) on-board. Since their use is focused on control applications, they usually come with PWM circuits, which are not suitable for audio applications.

Reasonable audio devices require sampling rates of 48kHz and sample resolution of 16bits/sample if we consider studio-quality recordings. Therefore, academic audio projects require the development of custom boards using a micro-controller and an external D/A converter at least, assuming that the micro-controller has an audio-oriented A/D converter.

As an undergraduate project, we envision a programmable digital effect unit that can be useful for students interested on signal and systems and digital signal processing: they will be able to develop their algorithms on tools like Matlab or Octave and convert them in compiled codes to be uploaded into these devices, obtaining real-time processing (at maximum of 48kHz). To achieve this goal, we conceived a device using three evaluation boards from Texas Instruments (TI), respectively dealing with the algorithms (a micro-controller) and with the A/D and D/A conversions. Our primary application is applying distortion effects over electric guitar sounds in real-time, although it can promptly adapted to other instruments.

Similar work was made by Young and Chih [2] using 16-bit converters with 48kHz but with a different micro-controller. In addition, Hasnain and Saleem [3] used another approach to their work on using re-programmable Matlab Simulink blocks of codes in order to generate the audio effects.

Here we will present the design aspects of an academic-oriented device, including the project of some analog filters and amplifier for signal conditioning (particularly avoid signal aliasing). Thus this paper is organized as follow: first we will describe the electronic boards and other components that we used to mount the device, particularly focusing on their connectivity. Next, we will present the design of low-pass filters for anti-aliasing purpose and D/A conversion, and linear amplifiers for signal condition, followed by details of how to implement signal processing algorithms (focusing on time CPU interruptions). Finally some results are presented from real use of the device (capture from digital oscilloscopes) along with conclusions and suggestions of improvements.

2. MATERIALS AND METHODS

Figure 1 presents a block diagram of the proposed device: The electric guitar signal is linearly amplified and filtered by a low-pass filter (LPF) to eliminate aliasing artifacts of the signal before it is sampled. After this signal conditioning, it is sampled by the external ADC in order to be properly read by the micro-controller.

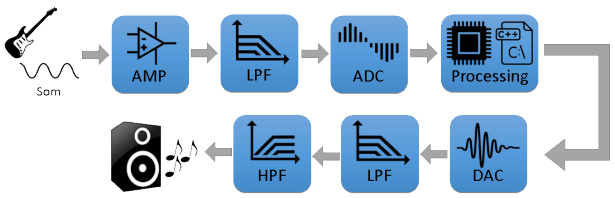

Figure 1: *Project's Block Diagram*

After digitally processed by the micro-controller, the signal passes through the external DAC and filtered by a low-pass filter (LPF) in order to be played. An additional HPF is used to remove the signal DC component since DAC generates analog signals with a fixed offset level equals to 2V.

Each component will be described in next sections.

2.1. Digital Processing Unit

Here we used the TI LaunchPad development kit (TIVA) that comes with the micro-controller TM4C1294NCPDT [4] shown in Figure 2. Such a kit has easy access to the micro-controller ports and comes with DIP switches and LEDs that can be programmed (we used them to allow its users to respectively choose a digital effect and to have a feedback of their choices). A motivation to use

* This work was part the author's undergraduate project.

this kit is that it has connectors to attach expansion boards named as BoosterPacks (both the A/D and D/A used here are expansion boards sold by TI).

It also has a Ethernet connector that can be used for data transfer that later will be used for direct-to-computer music recording and remote board configuration.

Figure 2: *TI's development kit for TM4C1294NCPDT (EK-TM4C1294XL)*

All codes were implemented, debugged, and uploaded to the kit in TI integrated development environment (Code Composer Studio - CCS).

2.2. External A/D and D/A Converters

Although the kit used here has a A/D converter, it only supports a bit depth of 12bits/sample, not meeting our requirements. Therefore we used an external A/D converter (ADC161S626) provided by TI [5] as an electronic board kit (a BoosterPack) that can easily be attached to the micro-controller board. Such an expansion board has an operational amplifier to offset the signal to a mid voltage reference. Its analog-to-digital conversion uses successive approximation register (SAR) architecture. The expansion board is shown in Figure 3.

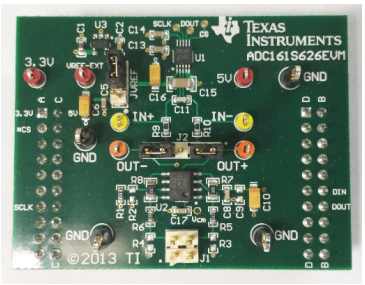

Figure 3: *ADC in TI's BoosterPack kit (ADC161S626EVM)*

After the signal is processed by the micro-controller, it is converted back into audio, from a digital form to a voltage signal. Another expansion board is used for this purpose once the micro-controller's kit does not have any digital-to-analog converter (DAC) in its circuitry. Since we required a DAC with 16-bit conversion resolution as a project specification, we chose DAC161S055, also provided by TI [6] in a BoosterPack (Figure 4). It has a resistor matrix topology and internal registers to setup its operational mode.

In this project, the DAC was configured to operate in write-through mode, which means it updates the voltage output as soon as data transfer is completed (other offered modes would require a delay between these two conversion steps). Therefore, we increased the available processing time between two successive sampling steps (time to execute the effects).

Figure 4: *DAC in TI's BoosterPack kit (DAC161S055EVM)*

While ADC samples are coded in two-complement binary representation (with the most significant bit for the math sign) in order to represent both positive and negative amplitudes, DAC samples represents only positive values since it produces only positive analog signals (a reference voltage is used to correctly understand the output signal). Therefore a simple integer math was required in order to obtain right conversions.

2.3. Analog Filters and Amplifier

Both filters and amplifier were based on operational amplifiers (op-amp) due to their simplicity. In both cases we powered the op-amps with 5V because this voltage was supplied by the USB port of the micro-controller board. In case of the amplifier, we added a biased voltage of 2.5V (a DC level) to the input signal since ADC and DAC require positive voltages. Consequently the outputs of all amplifiers bounce around $\pm 2.5\text{V}$ and the saturation is achieved for voltages exceeding 0 to 5V limits. To do that, the following circuit was implemented (Figure 5), where carefully chosen resistors and capacitors amplify only the AC part of the signal, ignoring its DC level.

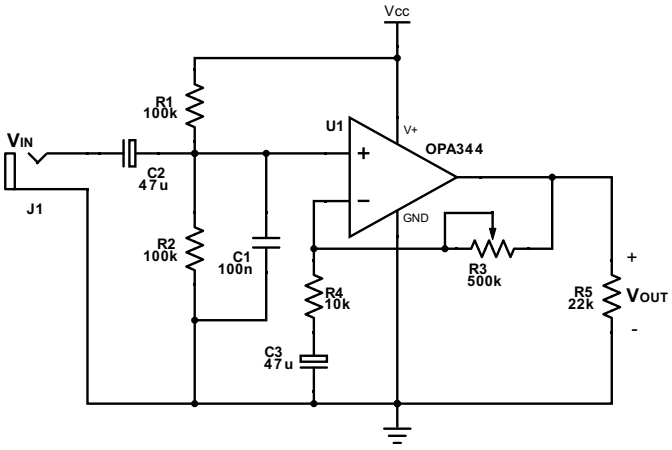

Figure 5: *Amplifier circuit*

The two LPFs in Figure 1 correspond to fourth order active low-pass filters - using two Sallen-Key topologies (quality factor $Q = 0.5$ and cut frequency $f_c = 24,405.14\text{Hz}$) in cascade - were built to work as anti-aliasing filters (the value of f_c is due to the use of electrical components that were commercially available).

Its resulting cutoff frequency was set to approximately 24kHz following Nyquist theorem [7].

TI OPA344 was the op-amp chosen for both amplifiers and LPFs, which is a low power single-supply rail-to-rail op-amp eligible for audio applications (also it comes in dual in-line package - DIP - which is suitable for breadboard testing and building academic circuits without any specific tool like the ones for SMD packages).

A passive HPF with a low cutoff frequency was designed to remove the DC signal component. A common RC passive topology was chosen with a resistor ($22\text{k}\Omega$) and a capacitor ($10\mu\text{F}$), resulting in a cutoff frequency of 0.72Hz.

Figures 6 and 7 shows, respectively, the frequency response of both LP and HP filters.

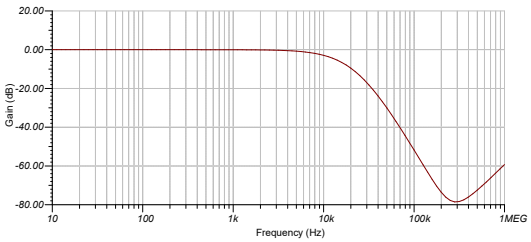

Figure 6: *Active Low-pass Filter Frequency Response*

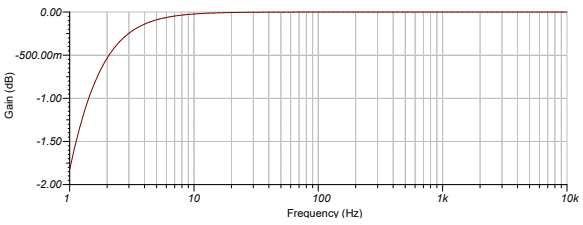

Figure 7: *Passive High-pass Filter Frequency Response*

2.4. Electrical Connections

Both converters use a synchronous serial interface (SSI) to communicate with the micro-controller. In such a data bus, a master device manages the communication while the slave ones answer back. Here we set the micro-controller as the master device and both converters as slave ones. To properly work four pins of master and slave devices should be used: the chip select pin (`CS`) to select the correct peripheral to send/receive data (one of the converters), the serial clock pin (`SCL`) to synchronize the data transfer, the synchronous serial transmitter (`SSTx`), and the synchronous serial receiver (`SSRx`) pins to send/receive bit streams between the devices. An interconnection diagram for SSI used here is illustrated in Figure 8. Programmatic, both ADC and DAC have internal FIFO queues to send and receive bit-oriented streams of data or commands.

The SSI clocks (or bit rate) for both ADC and DAC devices were set at their maximum values, 5MHz and 20MHz respectively. It was set higher than the required frequency for converting bits to voltage and vice-versa (24 bits/sample $\times 48.000$ samples/sec) in order to leave enough time for audio signal processing.

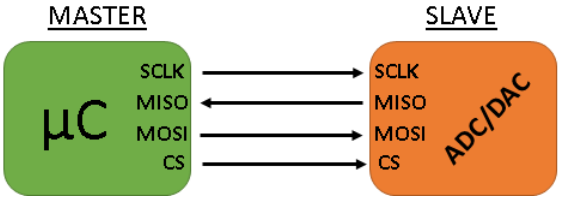

Figure 8: *Diagram of electrical interconnection between the micro-controller and both A/D and D/A converteres (Synchronous Serial Interface (SSI) Diagram)*

Electrical wires connect the analog filters (LPFs and HPFs) and amplifiers to the input of ADC board and output of the DAC board. We used a 6.3mm female J1 connector to plug an electric guitar in, and the same kind of connector is used to plug in an external speaker or a sound mixer.

2.5. Implementing Digital Distortions

Basically the distortion routines are implemented as a sequential procedure of acquiring digital samples from the ADC, processing them according to a predefined audio effect, and converting the result into an analog signal. Considering the sampling rate used here (48kHz), we coded these routines as part of a micro-controller timer interruption in order to minimize jitter effects on the output signal. Alternatively we considered to implement direct memory access (DMA) data transfer to speed up the process but we felt that our current implementation with timer interruptions was efficient enough for running some the digital distortions we describe here.

Therefore, each time this interruption is triggered, it executes the following steps:

1. Read a sample from ADC (waiting until the ADC release a 16-bit sample);
2. Process the sample;
3. Send the processed sample to the DAC (waiting until it finishes the conversion);

Naturally, the period between the execution of two successive timer interruptions was $(1/48000)\text{sec}$. There are different but almost fixed Δt's for running different audio effects, however the time interval between executions is constant (it means there should be low variable delays between input and output signals in current implementation).

The audio distortions are selected by user when he/she presses an specific button. It generates an interruption that alter a global counter/variable, which is used by the timer interruption code to sequentially select an audio distortion.

Prior to this endless procedure (meaning that our ADC and DAC never stop acquiring and generating audio signals), both micro-controller and external devices are properly configured.

We implemented four different audio effects: distortion (or saturation), delay, loop, and tremolo. The first one (the simplest effect) is the distortion. It adds harmonics to the output sound that make the sound look like an electric guitar played in a rock concert with its amplifiers saturating the sound levels. This effect basically clips the sound wave (regardless positive or negative amplitudes) and the amount of clipping defines how much "fuzzy" will the output sound be. The Algorithm 1 shows the basic operation of this effect: an if-then-else statement compares the input

value from ADC to the distortion limit set previously, and if the absolute magnitude of the sample is greater than the distortion limit value, then the output value will be the distortion limit, otherwise will be the value received at first.

Input: i^{th} sample from ADC: sample_to_adc
Output: i^{th} sample to DAC: sample_to_dac
if sample_to_adc >= distortion_limit **then**
  sample_to_dac = distortion_limit
else if sample_to_adc <= -distortion_limit **then**
  sample_to_dac = -distortion_limit
else
  sample_to_dac = sample_to_adc
end

Algorithm 1: Distortion implementation

The delay effect simply delays the sound signal by a fixed amount of time. It requires storing a quantity of samples in a vector/array. Here we used 1sec delay, which requires storing $48,000$ samples. To avoid moving data in order to store new samples, we logically implemented a circular buffer with a single variable to control the access to it. Algorithm 2 shows this implementation.

Input: i^{th} sample from ADC: sample_from_adc
Output: i^{th} sample to DAC: sample_to_dac
sample_to_dac = sound_array [i]
sound_array [i] = sample_from_adc
i ++
i = i % 48000

Algorithm 2: Delay implementation

The loop effect replays a pre-recorded signal in loop fashion, basically giving a base sound for musical arrangements. To do that, first an array of fixed size (here we used a size equivalent to 1sec) receives all samples from ADC (up to 48000 samples). Once the buffer is full, our routine starts to send all these stored samples to DAC indefinitely.

The user can set another pre-recorded signal by pressing a button which triggers an interruption where a variable called isRecording is set in order to enable the recording mode of Algorithm 2.

Input: i^{th} sample from ADC: sample_from_adc
Output: i^{th} sample to DAC: sample_to_dac
if isRecording **then**
  sound_array [i] = sample_from_adc
  i ++
  if i = i_{max} **then**
    i = 0
    isRecording = false
  end
else
  sample_to_dac = sound_array [i]
  i ++
  i = i % 48000
end

Algorithm 3: Loop implementation (Both i and isRecording are set by an interruption triggered by a button pressed)

The last effect is tremolo: it modulates the ADC signal according to a preset signal. Analog tremolos are implemented by a low-frequency oscillator (LFO) - whose frequency ranges from 0.5 to 10Hz - to vary the sound amplitude. Here we digitally implemented it using a 10000-samples array containing a 4.8Hz sinusoidal signal with amplitude equals to 0.25 and an offset of 0.75 (these values affect the way the input signal is altered). This array was generated in MATLAB and hardcoded in the tremolo routine.

The implementation consists of multiplying samples of this array by the samples from ADC. Therefore the amplitude of the input signals are attenuated to a maximum of 50% according to the preset sinusoidal signal used.

Input: i^{th} sample from ADC: sample_from_adc
Output: i^{th} sample to DAC: sample_to_dac
sample_to_dac = tremolo_array [i] $\times$
  sample_from_adc
i ++
i = i % 10000

Algorithm 4: Tremolo implementation

3. RESULTS

The resulting device is depicted in Figure 9. To demonstrate its usefulness, we first present the A/D and D/A conversions carried out by our prototype with no digital distortions and no analog filtering been applied to the signal except by the analog amplification. Two sinusoidal (narrow-band) signals were separately applied to the prototype input jack and the DAC output pin (therefore using no output LP and HP filters) was connected to an oscilloscope. Figures 10 and 11 shows this output signals for a 1kHz and 5kHz sinusoidal signals.

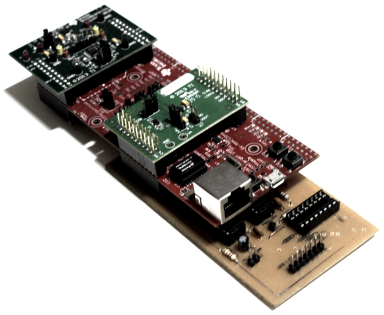

Figure 9: *Picture of the device*

The difference on voltage scale of each channel occurred because each converter had different numeric ranges: ADC works with 15-bit values plus one bit for signal (the most significant one) in two's complement notation, while DAC uses all 16-bits to represent positive output values. That led to an output value equals to the half of an input value in this no-distortion scenario.

The stair effect on channel 2 was given by the zero-order holder effect of DAC. Figure 11 shows that clearly. It also shows its influence on the signal frequency captured by the oscilloscope. Considering that we were focusing on building a guitar effect unit, such a problem may not be a big deal since an in-tune guitar has frequencies ranging from 80Hz up to 1200Hz and we were using high

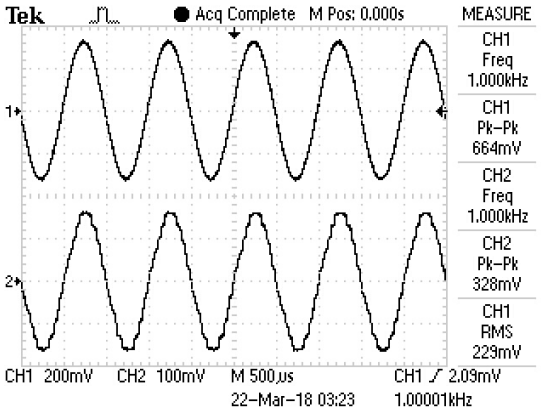

Figure 10: 1kHz *test sinusoid wave: channel 1 and 2 registers the input and output signals, respectively (this setting will be used in all following figures)*

clock rate to excite our DAC module. However, to confidently cope with any input signal, we used low-pass filters (Section 2.3 to finish the DA conversion, as shown in Figure 12. Note that these kind of filter impose a delay (in case of a 5kHz sinusoidal signal, it is about $52\mu s$).

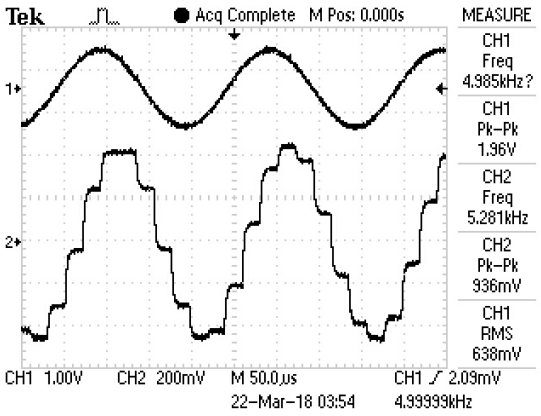

Figure 11: 5kHz *test sinusoid wave*

Figure 13 shows a guitar signal captured after directly connecting the guitar cable jack in an oscilloscope: its peak-to-peak amplitude does not exceed 200mV. Although it varies according to instrument technology, brand, and age, for example, the voltage amplitude never reaches 1V. Such voltage values demands a pre-amplifier for using sound systems as usually digital effect units requires. Our proposed amplifier at the input of the system allows some adjustments before applying analog filters and digital effects coded into the micro-controller.

To demonstrate the distortion effect, a sinusoid signal were applied: Figure 14 shows the resulting effect whose $distortion\_limit$ was set to 3000 which is equivalent to 0.92V after the analog-to-digital conversions.

For delay, loop, and tremolo effects, we played a few tones in an electric guitar connected to our device, which is altered by such effects. In all cases, the oscilloscope was set to capture 5sec of input (channel 1) and output (channel 2) signals after triggered.

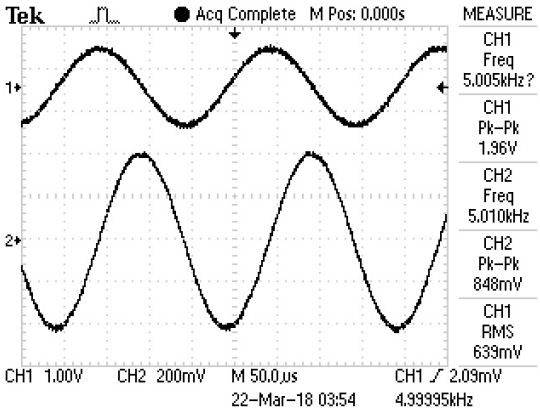

Figure 12: 5kHz *test sinusoid wave after the output LP filter*

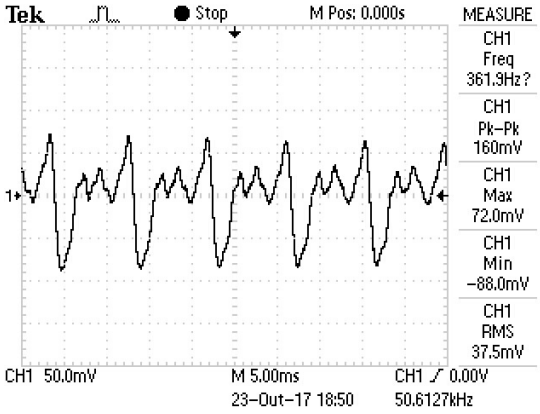

Figure 13: *Example of guitar signal*

Figure 15 shows the delay effect: channel 2 shows the delayed version of the input signal (delay of 1sec). In case of loop effect, one second of an input signal was previously recorded by the device (not shown here). After that, Figure 16 shows that an input signal captured by the device was ignored, and the recorded signal was repetitively reproduced by the device as its output signal.

Finally, the tremolo effect altered the input signal (channel 1) by modulating it with a sinusoidal signal. Channel 2 of Figure 17 shows the expected result. Due to the nature of the input signal, this modulation is more evident at the middle of the oscilloscope screen in this example.

4. CONCLUSIONS AND FUTURE IMPROVEMENTS

Here we present a digital audio effect for guitars that was implemented with a micro-controller and external digital converters - ADC and DAC - to operate at 48kHz. Additional circuitry for amplifiers and low pass filters were designed to cope with Nyquist limits, and a few digital effects were implemented to demonstrate the use of the unit. The use of timer interruptions to filter the input signal, sample-by-sample, before sending it to DAC minimized the jitter level. All the system requires a voltage source of $+5$V (0.47W), which can be powered through the USB connector of the micro-controller kit.

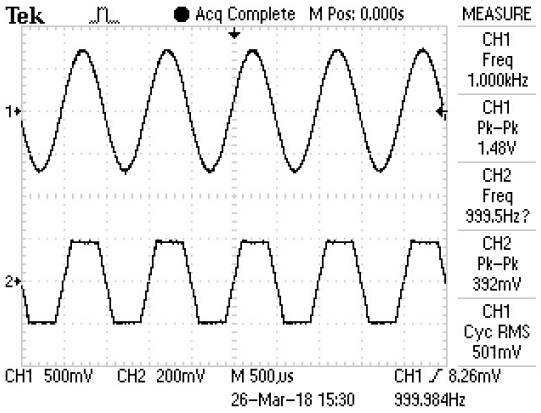

Figure 14: *Distortion result from a* 1kHz *sinusoid wave*

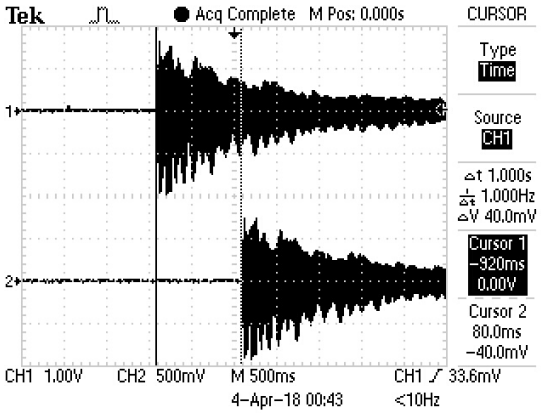

Figure 15: *Result (Channel 2) of applying the delay effect (*1sec*) over a signal produced by a electric guitar (Channel 1)*

Although the delays imposed by the analog LP filters, this academic prototype of a digital effect unit worked fine. Other effects can be readily implemented in order to have near real-time digital effects. Our next steps are:

- Add digital dithering to improve sound quality, specially for live guitar sounds;
- Reduce the jitter caused by lengthy complex digital filters by using two different timer interrupts (respectively for ADC and DAC procedures) and respective data buffers that operates in parallel;
- Create an internet (tcp/ip) server inside the micro-controller that transfers all sampled data to a remote client applications, allowing to create a virtual mixer with multiple channels (limited by the computer client capacity);
- Create an internet (tcp/ip) server inside the micro-controller that receives additional digital effects algorithms (and its configurations) from remote client applications, which allows the digital effect unit be remotely programmed.

5. ACKNOWLEDGMENTS

This work was funded by Fundação Araucária, Paraná, Brazil.

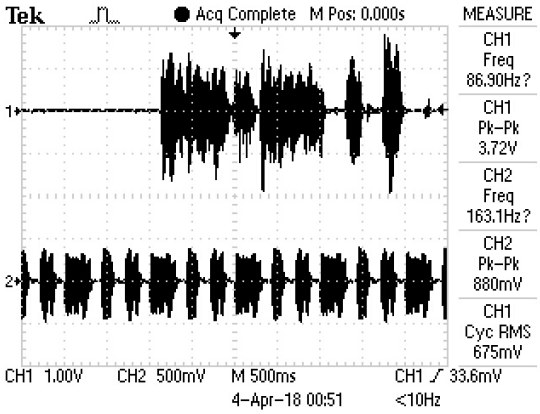

Figure 16: *Result (Channel 2) produced by a previous recorded sound from an electric guitar. The guitar keeps playing (Channel 1) but the loop, once recorded, wont change its output values*

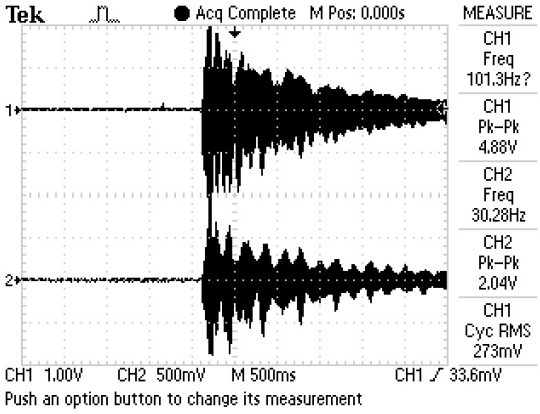

Figure 17: *Result (Channel 2) of applying the tremolo effect over a signal produced by a electric guitar (Channel 1)*

6. REFERENCES

[1] B. Tarquin, *Stomp on this! The Guitar Pedal Effects Guidebook*, Cengage Learning, 2015.

[2] C.-F. Yang and H.-Y. Chih, "An open source audio effect unit," in *IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC)*. IEEE, 2016, pp. 638–643.

[3] S. Hasnain, A. Daruwalla, and A. Saleem, "A unified approach in audio signal processing using the tms320c6713 and simulink blocksets," in *2nd Int. Conf. on Computer, Control and Comm. (IC4)*. IEEE, 2009, pp. 1–5.

[4] Texas Instruments, *TIVA™ TM4C1294NCPDT Microcontroller*, 2014.

[5] Texas Instruments, *ADC161S626 16-Bit, 50 to 250 kSPS, Differential Input, MicroPower ADC*, 2008.

[6] Texas Instruments, *DAC161S055 Precision 16-Bit, Buffered Voltage-Output DAC*, 2010.

[7] A. Oppenheim, *Discrete-time signal processing*, Pearson Education India, 1999.