Revista EIA, ISSN 1794-1237 / Year XI / Volume 11 / Issue N.22 / July-December 2014 / pp. 123-136
Technical-scientific biannual publication / Escuela de Ingeniería de Antioquia —EIA—, Envigado (Colombia)

DESIGN OF A PROGRAMMABLE MICROSYSTEM FOR DIGITAL AUDIO EFFECTS USING FPGAS

John Michael Espinosa-Duran[1]
Pedro P. Liévano-Torres[2]
Claudia P. Rentería-Mejía[3]
Jaime Velasco-Medina[4]

ABSTRACT

This paper describes the design of a programmable microsystem for processing digital audio effects implemented in an FPGA. The microsystem is designed using an application-specific reconfigurable processor, a bank of RAMs, and a graphical user interface based on an LCD touch panel. The processor is designed using 15 audio effects based on delays and dynamic domain and frequency domain processing. The effects are designed using Megafunctions and the Quartus II FIR compiler , simulated in Simulink using DSP Builder, and configured using a user graphic interface. The programmable microsystem is implemented on the DE2-70 development board, and its operation is verified using an MP3 player and a speaker. Additionally, the microsystem allows the generation of effects with high fidelity using a maximum sample rate of 195.62 MSPS and can be embedded into a SoC.

KEYWORDS: digital audio effects, dynamic domain, frequency domain, embedded systems, FPGAs

DISEÑO DE UN MICROSISTEMA PROGRAMABLE PARA EFECTOS DE AUDIO DIGITAL USANDO FPGAS

RESUMEN

Este artículo describe el diseño de un microsistema programable para el procesamiento de efectos de audio digital implementado en un FPGA. El microsistema es diseñado usando un procesador de propósito específico y reconfigurable, un banco de RAMs y una interfaz gráfica de usuario basada en una pantalla táctil LCD. El procesador es diseñado usando 15 efectos de audio basados en retardos y procesamiento en el dominio dinámico y de la frecuencia. Los efectos son diseñados usando Megafunciones y el compilador FIR de Quartus II, son simulados en Simulink[5] usando DSP Builder[6], y

[1] Electronic Engineer, Universidad del Valle. Master's in Engineering, Universidad del Valle. Doctoral student in Chemistry and Associate Professor at the University of Indiana, Bloomington, USA.
[2] Electronic Engineer, Universidad del Valle. Coordinator of projects and development in IP Innovatech, Cali, Colombia.
[3] Electronic Engineer, Universidad del Valle. Master's in Engineering, Universidad del Valle. Doctoral student in Engineering at the Universidad del Valle, Cali, Colombia.
[4] Electrician Engineer, Universidad del Valle. Ph.D. in Microelectronics, Institute National Polytechnique de Grenoble, Grenoble, France. Professor and director of Bionanoelectronics research group, Engineering and Electronics School, Universidad del Valle, Cali, Colombia.

[5] Entorno de programación visual de Matlab
[6] Herramienta de desarrollo usada como interface entre el software Quartus II y Matlab/Simulink

Correspondence author: Velasco-Medina, J. (*Jaime*). Universidad del Valle: Calle 13 # 100 - 00, Cali (Valle, Colombia).
Tel: (572) 3303436.
Email: jaime.velasco@correounivalle.edu.co

Paper history:
Paper received on: 8-X-2013 / Approved: 17-III-2014
Available online: December 30 2014
Open discussion until December 2015

DOI: http:/dx.doi.org/10.14508/reia.2014.11.22.133-146

son configurados utilizando una interfaz gráfica de usuario. El microsistema programable es implementado en el sistema de desarrollo DE2-70, y su funcionamiento es verificado usando un reproductor MP3 y un parlante. Adicionalmente, el microsistema permite la generación de efectos con alta fidelidad usando una tasa de muestreo máxima de 195.62 MSPS, y puede ser embebido en un SoC.

PALABRAS CLAVES: efectos de audio digital; dominio dinámico; dominio de la frecuencia; sistemas embebidos; FPGAs.

PROJETO DE UM MICROSISTEMA PROGRAMÁVEL PARA EFEITOS DE ÁUDIO DIGITAL USANDO FPGAS

RESUMO

Este artigo descreve o desenho de um microsistema programável para o processamento de efeitos de auidio digital implementado em um FPGA. O microssistema é projetado usando um processador específico e reconfigurável um banco de RAMs e uma interface gráfica de usuario baseada em uma tela sensível ao toque de LCD. O processador foi projetado com 15 efeitos de áudio com base em atrasos e domínio de processamento e frequência dinâmica. Os efeitos são projetados usando Megafunciones e o compilador FIR de Quartus II são simulados utilizando Simulink 1 usando DSP Builder2 e são configurados através de uma interface gráfica de usuario. O microssistema programável é implementado no sistema de desenvolvimento DE2-70, e seu desempenho é verificado através de um leitor de MP3 e um alto-falante. Além disso, o micro-sistema permite a geração de efeitos, com elevada fidelidade, utilizando uma taxa de amostragem máxima de 195,62 MSPS, e pode ser incorporado em um SoC.

PALAVRAS-CHAVE: efeitos de áudio digital, de domínio dinâmico, domínio da freqüência, sistemas embarcados, FPGAs.

1. INTRODUCTION

Audio effects are used by musicians to create special sounds when playing an instrument. These effects can be created using analog or digital processing systems. Nowadays, digital audio effects are more widely used and are implemented with electronic systems that perform delay-based, dynamic domain, or frequency domain processing (Zölzer, 2002).

Digital audio effects are generally implemented on DSPs, PCs, or GPUs (Berdahl & Smith, 2006), (Fernandez & Casajus, 2000), (Guillermard, Ruwwue & Zölzer, 2005), (Karjalainen, Penttinen & Valimaki, 2000), (Ling, Khuen & Radhakrishnan, 2000), (Oboril et al., 2000), (Schimmel, Smekal & Krkavec, 2002), (Tsai, Wang & Su, 2010), (Verfaille, Zölzer & Arfid, 2006). However, the designs presented in the papers mentioned do not have high sampling rates, and the majority of the effects are based on delays. Pfaff et al. (2007) implemented the chorus, delay, echo cancellation, flanger, and wah-wah effects using hardware-software co-design. The flanger effect was implemented using a lookup table to generate the sinusoidal signal, and the wah-wah effect was implemented using a second-order allpass filter and a lookup table to vary the cut-off frequency, but the study does not present verification tests of the designs on hardware. Byun et al. (2009) implement reverb, chorus, flanger, phaser, tremolo, auto wah, pitch shift, distortion, and multi-band equalizer effects using C language and an embedded DSP in an FPGA. However, this article does not describe the algorithms used to implement these effects in detail, nor does it present verification tests of the designs. In a previous study (Liévano, Espinosa y Velasco, 2013), we implemented 14 audio effects using delay-based processing (5), dynamic domain processing (8), and frequency domain processing (1). These effects are described in VHDL, simulated in Simulink using DSP Builder, and synthesized in an FPGA using Quartus II.

Considering the information above, in this article we present the design of a microsystem for processing digital audio effects based on a dedicated and configurable processor. The microsystem was implemented in

the development system DE2-70, and the processor was synthesized in this card's FPGA.

The main contribution of this article is the design of a digital audio effects processor in real time with very high fidelity due to the fact that it has a sampling rate of 195.62 MSPS using 16 bits of data. The sampling rate corresponds to the processor's maximum operating frequency (195.62 MHz). In addition, the processor we designed allows for the generation of uncommon audio effects since it is possible to mix effects of different and/or the same processing type. We implemented 15 audio effects on this processor, 14 of which were presented in Liévano, Espinosa & Velasco (2013). However, the chorus, reverb, and wah-wah effects were redesigned, and a multi-band equalizer was added. The microsystem verification tests were completed using an MP3 player and a speaker connected to the DE2-70 card's audio input and output terminals, respectively.

This article is organized as follows: section 2 describes the design methodology for the chorus, reverb, wah-wah, and multi-band equalizer effects; this section also presents the microsystem design and the graphical user interface. Section 3 presents the synthesis results and hardware verification tests. Finally, section 4 presents our conclusions.

2. METHODOLOGY

The design of the hardware system for processing digital audio effects was completed in four stages: a) selection and functional simulation of the effects in Simulink using DSP Builder; b) implementation of the effects on the hardware using Quartus II and verification of effects using Matlab to graph the output signals; c) design of the processor and tactile graphic interface for configuring the effects parameters; d) design and verification of the microsystem for processing audio effects.

2.1. Selection and functional simulation of audio effects

The audio effects selected are delay-based or based on dynamic domain or frequency domain processing. These effects are implemented in the processor, and 14 of them are described in (Liévano, Espinosa & Velasco, 2013). In addition, we added a multi-band equalizer and redesigned the chorus, reverb, and wah-wah effects. The functional simulation of these audio effects was completed in Simulink using DSP Builder.

2.1.1 Delay-based processing

Delay-based processing consists of adding the audio signal to itself, attenuated and/or out of phase. Five effects of this type were implemented on the processor, in which delay, flanger, and phaser are described in Liévano, Espinosa & Velasco (2013), and chorus and reverb are described below.

Chorus is described by **Equation 1** and is an effect that emulates two or more musicians simultaneously playing the same instrument and the same piece of music (Zölzer, 2002). This effect is obtained by adding the current input signal with a previous input signal attenuated by random factor *g*.

$$y\,(n) = x(n) + g * x\,(n + del) \quad (1)$$

In which *is the delay between* 10 and 25 ms.

The reverb effect is described by **Equation 2** and is generated when the audio signal's acoustic reflexes are added to the audio signal. This effect is emulated by adding the input signal with its respective responses, which have different delays and attenuations (Zölzer, 2002).

$$y(n) = x(n) + g_1 x(n+del) + g_2 x(n+2del) + g_3 x(n+3del) + g_4 x(n+4del) \quad (2)$$

2.1.2 Dynamic domain processing

This type of processing is generally non-linear and considers the signal's dynamic. Eight effects were implemented on the processor: compressor, expander, noise gate, soft and hard clipping, sigmoidal distortion, sigmoidal piecewise distortion, polynomial distortion, and ring modulator, which are described in Liévano, Espinosa & Velasco, (2013).

2.1.3 Frequency domain processing

Frequency domain processing is based on modifying the sound spectrum using digital filters (Khosravi, 2007). Two effects were implemented on the processor: wah-wah and multi-band equalizer. The wah-wah effect is obtained by filtering the input signal using a narrow band-pass filter with a variable central frequency, which generates a sound similar to the word 'wah-wah' (Zölzer, 2002). The multi-band equalizer

effect modifies the audio spectrum of a signal through the amplification of certain frequency bands. This effect is implemented using first- and second-order shelving and peak filters which are connected in series and independently controlled. The shelving filters amplify or attenuate the high and low frequency bands using the cutoff frequency parameters f_c and gain G. Peak filters amplify or attenuate medium frequency bands using the cutoff frequency parameters f_c, band width f_b, and gain G (Zölzer, 2002).

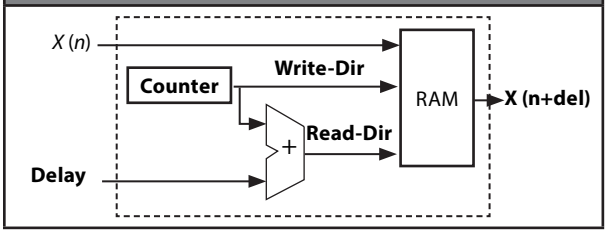
2.2. Implementation on hardware and verification of audio effects

2.2.1 Chorus and reverb implementation

These effects use a circular buffer which is implemented on a dual-port RAM (Altera, 2011). The write and read directions on the 16Kx16-bit RAM are generated by a 14-bit counter. In this case, read used an indexed direction in which the index is the *delay* described by **Equation 3** (see **Figure 1**).

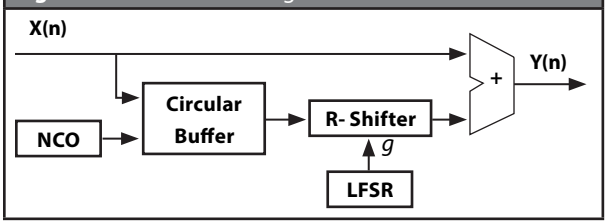
$$retardo = \frac{del}{1000} * fs \qquad (3)$$

Figure 1. *Circular buffer block diagram*



The chorus effect is designed using a circular buffer, an adder, a right-shifter, a low-frequency NCO, and an LFSR (Pérez, 2006) as shown in **Figure 2**. In a previous study (Liévano, Espinosa & Velasco, 2013), the chorus effect delay is implemented using an LFSR, but in this study the delay is implemented with an NCO given that Pérez (2006) recommends using an LFO.

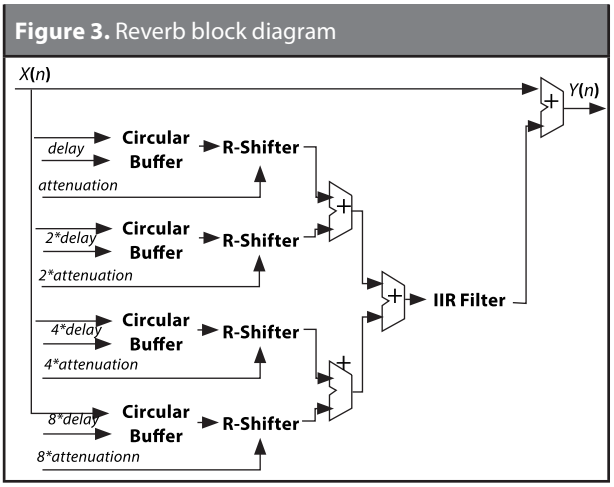
Figure 2. Chorus block diagram



The reverb effect is designed using four circular buffers, four right-shifters, a low-pass IIR filter, and four adders (Pérez, 2006) as shown in **Figure 3**. In this case, four configurable values were used for the attenuation and delay in order to emulate the input signal's acoustic reflexes. The first values for attenuation and delay are configuration parameters, and the other values are generated by multiplying the first values by 2, 4, and 8. The IIR filter was implemented using **Equation 4**.

$$Y(n) = 0.4y(n) - 0.2499y(n\text{-}2) + 0.0441y(n\text{-}3) + 0.5814x(n\text{-}1) + 0.2142x(n\text{-}2) \qquad (4)$$

Figure 3. Reverb block diagram



In a previous study (Liévano, Espinosa & Velasco, 2013), the reverb effect was implemented without using an IIR filter, but in this study we used a low-pass IIR filter in order to obtain a realistic emulation of the reverberation (Pérez, 2006).

2.2.2 Wah-wah implementation

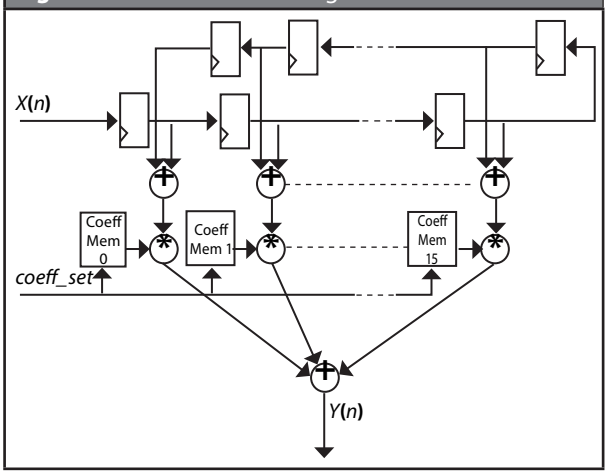
The wah-wah effect is implemented using a 30-order band-pass FIR filter, which is designed using a Hamming window, a symmetrical architecture, a configurable central frequency, and a bandwidth of 1000 Hz. In this case, the FIR filter is configured from the touch screen (see **Figure 11**), specifically by varying the wah-wah control icon and selecting one of the FIR filters presented in **Table 1**. For example, FIR filter 6 has lower and upper cutoff frequencies of 750 Hz and 1750 Hz, respectively. Therefore, the wah-wah sound will be generated if the audio signal is within the respective window of the selected filter.

The 30-order FIR filter's implementation on the hardware used 16 coefficients and was designed using Altera's FIR Compiler 10.1 (Altera, 2013). In this design, 16 16x16-bit RAMs were used to store the 16 coefficients for each of the 16 filters presented in **Table 1**, in which the *coeff-set* signal is the direction for selecting the filter coefficients. **Figure 4** shows a block diagram of the configurable FIR filter.

Table 1. Wah-wah low-pass filter configuration parameters

Filter	Coeff-set	Lower cutoff frequency (Hz)	Upper cutoff frequency (Hz)
1	0000	1.000	---------
2	0001	150	1.150
3	0010	300	1.300
4	0011	450	1.450
5	0100	600	1.600
6	0101	750	1.750
7	0110	900	1.900
8	0111	1.050	2.050
9	1000	1.200	2.200
10	1001	1.350	2.350
11	1010	1.500	2.500
12	1011	1.650	2.650
13	1100	1.800	2.800
14	1101	1.950	2.950
15	1110	2.100	3.100
16	1111	2.250	3.250

Figure 4. Wah-wah block diagram.



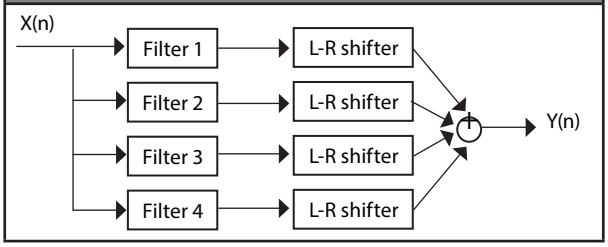
2.2.3 Implementation of multi-band equalizer

Holters & Zölzer (2006) propose designing a three-band digital parametric equalizer using a sampling rate of 48 kHz. The equalizer is implemented as a cascade of shelving filters. In this case, we designed the multi-band equalizer to attenuate or amplify components of low, mid-low, mid-high, and high frequencies. It is implemented using four 100-order FIR filters and four configurable displacement registries. One low-pass and three band-pass filters are used. They are designed with a Blackman window using an FIR Compiler, and their cutoff frequencies are presented in **Table 2**. The L-R shifter displacement registries amplify or attenuate the output signal of the FIR filters, multiplying or dividing the signal by 2^n, where n is equal to one or two. **Figure 5** shows a block diagram of the multi-band equalizer.

Table 2. Multi-band equalizer FIR filters

Filter	Lower cutoff frequency (Hz)	Upper cutoff frequency (Hz)
1	125	----------
2	125	500
3	500	2.000
4	2.000	20.000

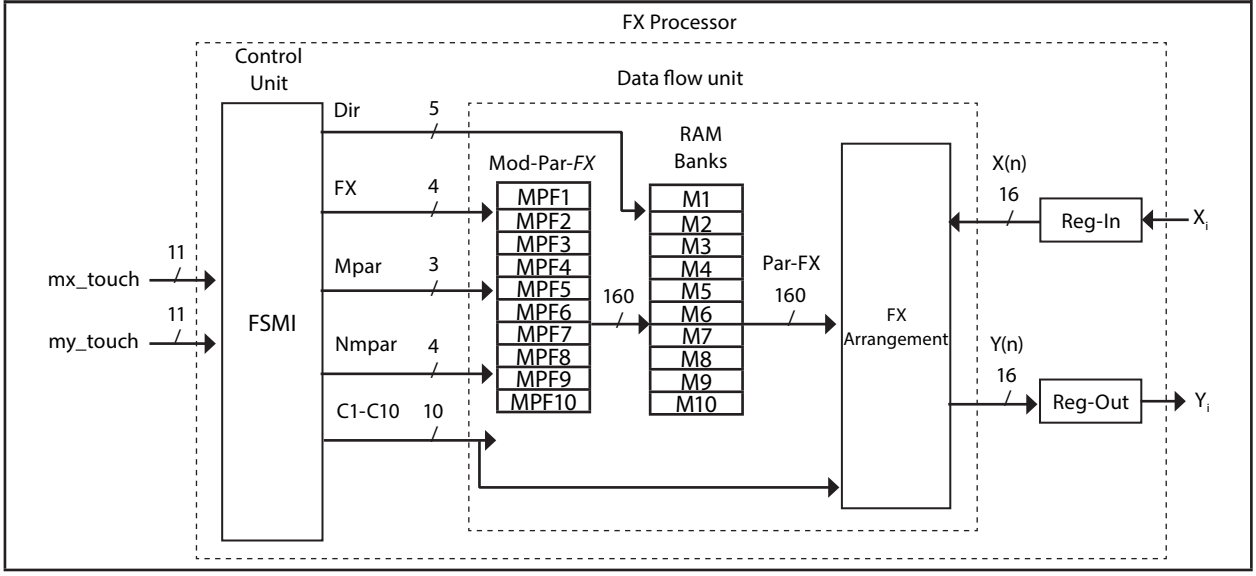
Figure 5. Multi-band equalizer block diagram



The design of each audio effect was verified by calculating the correlation between the results of the functional simulation in DSP-Builder, which uses a 64-bit floating-point arithmetic, and the results of the verification with hardware, which used 16-bit fixed-point arithmetic. In this case, the error was calculated using **Equation 5**, in which *Ym*Yr* is the correlation between the simulation results and the hardware verification tests.

$$error = 100 \times (1 - (Ym * Yr)) \quad (5)$$

Figure 6. Digital audio effects processor architecture



The processor is designed using a data flow unit and a control unit, as is shown in **Figure 6**. The data flow unit is designed using an arrangement of digital audio effect blocks (*FX Arrangement*) and a bank of RAMs for storing each block's effect configurations. The control unit is designed using a state machine (*FSM*) described in behavioral VHDL. In addition, the processor has a serial input registry and 16-bit parallel output (*Reg-In*) for storing the input signal X_i ,and a 16-bit parallel input registry and serial output (*Reg-Out*) for storing the output signal Y_i.

The FX arrangement is implemented using 10 digital audio effect blocks, *FX1-FX10*, connected in a cascade, as is shown in **Figure 7**. The order or sequence of the cascade of effect blocks corresponds to that used by commercial pedal manufacturers (Sound Laboratory Zoom, 2013). The two effect blocks adapt the signal to be processed by the other blocks; that is, the noise gate effect eliminates the signal noise, and the compressor/expander block compresses or expands the signal according to its voltage level. The order of the ring modulator, phaser/flanger, chorus, and delay effects is irrelevant. The multi-band equalizer and reverb effects improve the audio signal processed by the previous effects. The wah-wah effect comes at the end of the cascade because its implementation requires many area resources, which implies that the signal is degraded when this block is located among the other effects. The majority of the blocks have one effect, except blocks *FX2*, *FX3*, and *FX6*, which have 2 or 3 effects. The blocks are configured using 10 registries, *RC1-RC10*, which store the 16-bit configuration word for each effect, and the output of each block is connected to a multiplexer, which allows us to select the input signal for the next block. In other words, the multiplexers allow us to select the set of effect blocks that process the input signal X_i.

The memory bank is made up of 10 RAMs, and each RAM has 32 16-bit words, where each word stores an audio effect configuration. For example, if a block has only one effect, this effect can have 32 configuration options. The first 16 positions of each RAM store the predetermined configurations for each effect block using a .mif file, and the last 16 positions of each RAM are used to store the new configurations developed by the user.

The effect configuration in each of the M_i RAMs is completed using the write signal *WE*, the direction signal *Dir*, and the 16-bit configuration word obtained from the Mod-Par-FX block, which is based on comparators and multiplexers, as shown in **Figure 8**.

We can see in **Figure 8** that modifying one of the effect configuration word parameters is done if the values of the signal *FX&Mpar* (*FX* effect and *Mpar* parameter) and the constant $Const_i$ are equal. If this is not the case, the configuration word is not modified. We can also see that the *FX* effect code has 4 bits, the parameter *Mpar* code has 3 bits, the parameter value

Nmpar has 2, 3, or 4 bits, and each constant $Const_i$ has 7 bits, where *i* is a whole number between 1 and 4.

The processor's control unit is implemented using the FSM1 state machine, which controls the user interface's LCD touch screen sonsor and generates the control signals for the FX arrangement, the Mod-Par-FX block, and the bank of RAMs. The FSM1 constantly supervises the graphical interface's graphic objects or icons in order to generate the control signals that allow for modification of an effect's configuration word. This procedure is completed in four steps: 1) loading one of the 16 predetermined configurations in each RC configuration registry for each effect block, 2) selecting the set of audio effect blocks in the FX arrangement that will process the digitalized audio signal *X(n)* using signals *C1-C10*, 3) loading the new configuration determined by the user in the RAM that corresponds to the effect block (his configuration is made by modifying a predetermined configuration found in one of the RAM's first 16 positions or by generating a new configuration to be stored in one of the RAM's last 16 positions), and 4) loading the new configuration from the RAM into the RC registries; that is, a predetermined configuration, a modified predetermined configuration, or a new configuration. **Figure 8** shows the ASM (Algorithmic State Machine) diagram of the FSM1.

Figure 9 shows that the control signals *Ini-Conf* and *Fin-Conf* are used to initiate and finalize a new configuration, respectively. These two signals are generated by the used from the touch screen using the LTM-SoPC controller (Altera, 2011a).

2.4. Design of hardware microsystem for processing audio effects

In order to use the processor in a real application, we designed a processing microsystem for the electric guitar, which is implemented using a hardware unit and a graphical user interface based on an LCD touch screen to configure the processor designed. The hardware unit is implemented in the Terasic DE2-70 development system using the FPGA, the codec, and the DRAM memory. In the FPGA, we synthesized the audio effects processor, three ROM memories, and the LTM-SoPC controller for the LCD touch screen (LTP: LCD Touch Panel) as shown in **Figure 10**.

Figure 7. Audio effects processor FX arrangement

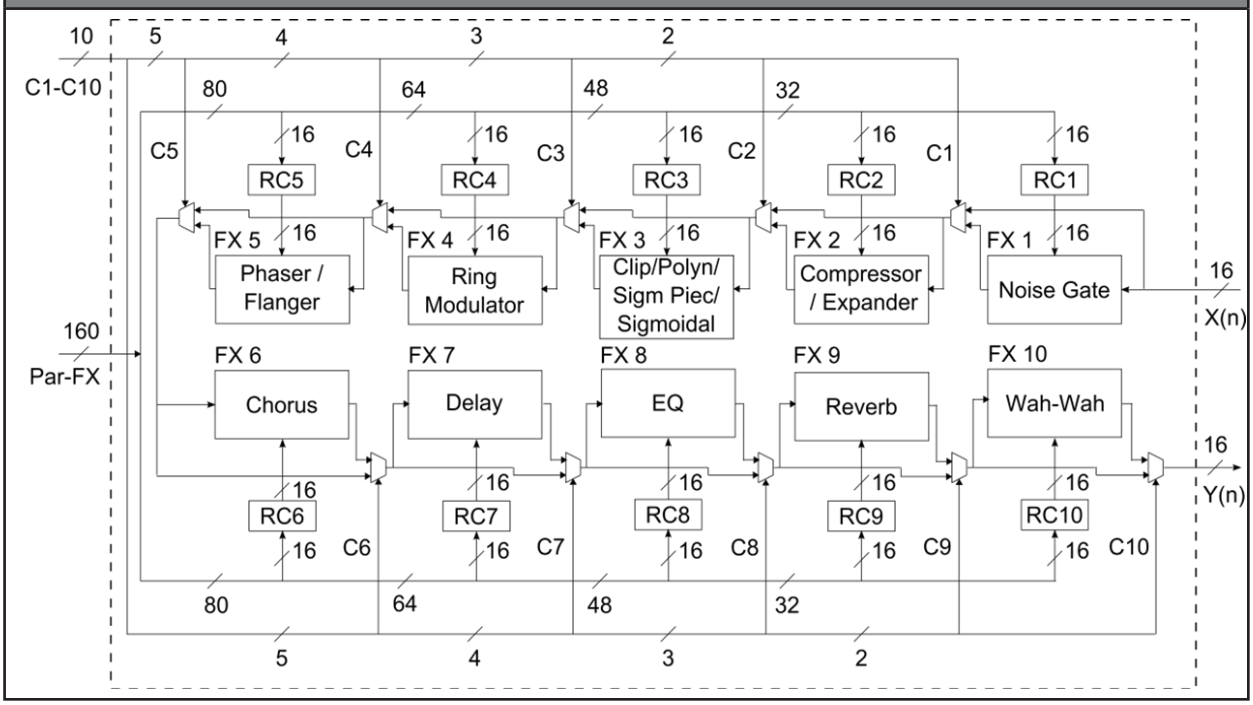


Figure 8. Block diagram of effect parameters configuration

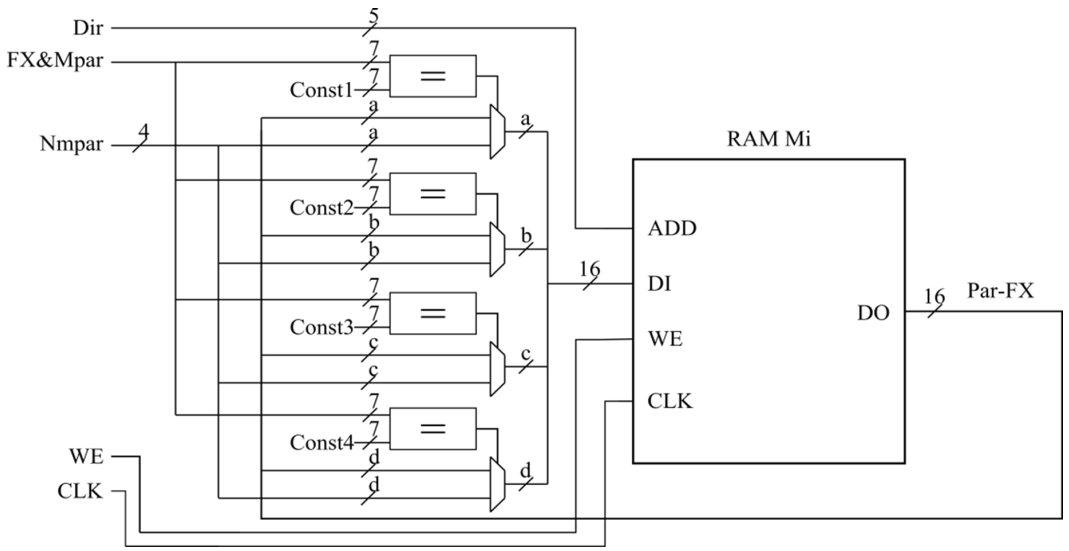
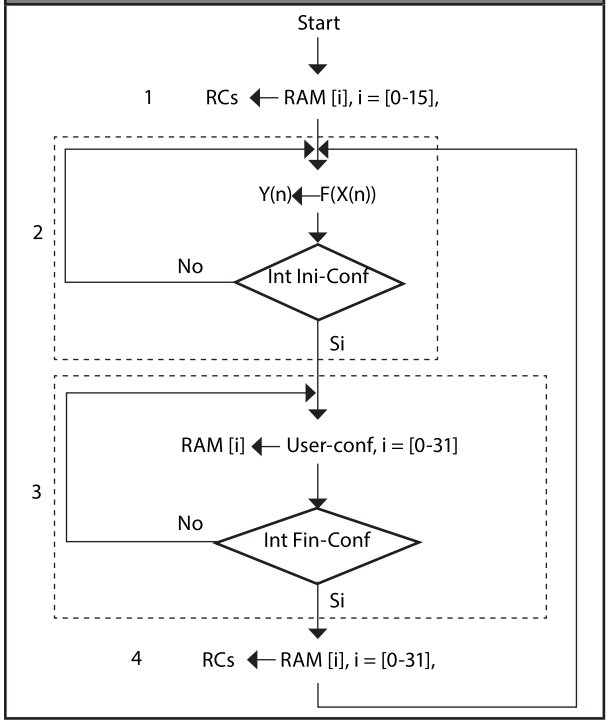


Figure 9. ASM diagram of FSM1



The graphical user interface is designed using the FSM2 state machine which generates pixels on the LCD touch screen corresponding to the images for the background and the fixed and variable graphic objects. The latter are generated by the hardware unit and could be, for example, the image for sound volume.

The FSM2 controls generation of the images using information stored in the three ROM memories and completes the transfer of the images to the RAM memory and the LCD touch screen using the LTM-SoPC controller.

Figure 11 shows the graphical user interface, which includes the following graphic objects and their respective visualizers: effects, parameter, parameter value, sector, patch, volume, wah-wah control, and save. An effect's parameter is modified by selecting a sector and a patch, which can be used to direct the position of the RAM memory where the new effect configuration word will be stored.

The graphical interface has sectors A, B, C, and D to select the 32 configuration words. Each sector has eight memory positions, and each position in the sector is called a patch. Sectors *A-B* and *C-D* allow the user to direct the first 16 and last 16 positions on the M_i RAM, respectively.

The parameter of an effect's configuration word is modified as follows: 1) selecting one of the ten effect blocks; 2) selecting the parameter, and 3) modifying the parameter value. **Figure 12** shows the graphical user interface displayed on the LCD touch screen.

Figure 10. Audio effects microsystem block diagram

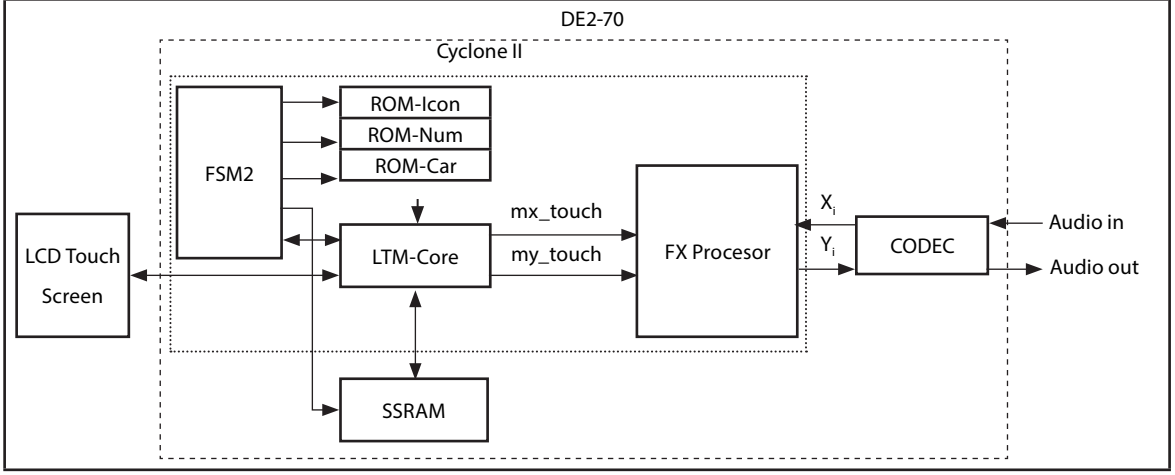
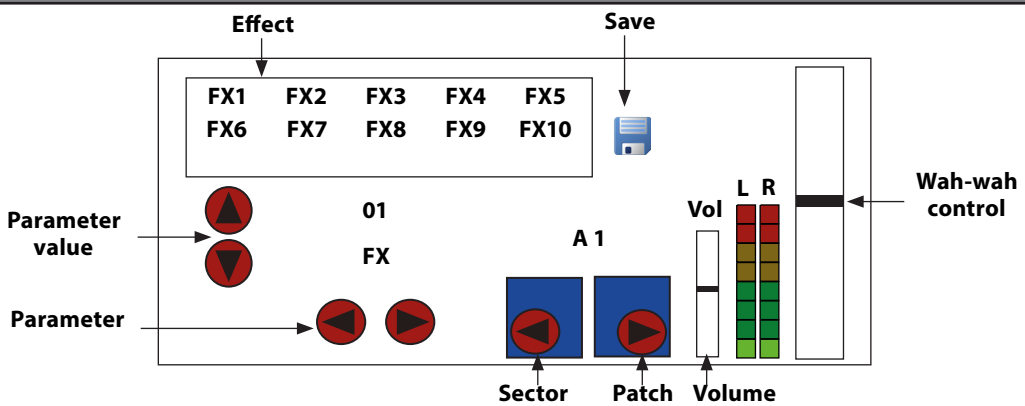


Figure 11. Graphical user interface for programming digital audio effects



2.5. Verification of the microsystem for processing audio effects

The microsystem is verified using an MP3 player and a PC. Initially, each of the processor's effect blocks were verified, and then functioning verifications were completed for two sets of effects connected in cascade. To verify each effect in the processor, we used an audio signal stored in an MP3 player whose audio output was connected to the analog input of the DE2-70 development system's codec. **Figure 13** shows the test audio signal in the codec's input, which was configured to sample the signal at a frequency of 44.1 kHz. The 16-bit X_i digitalized audio signal is then obtained from the codec's output. This signal has a two's complement representation format and is used as the input signal for the processor.

Figure 12. User interface visualized on LCD touch screen

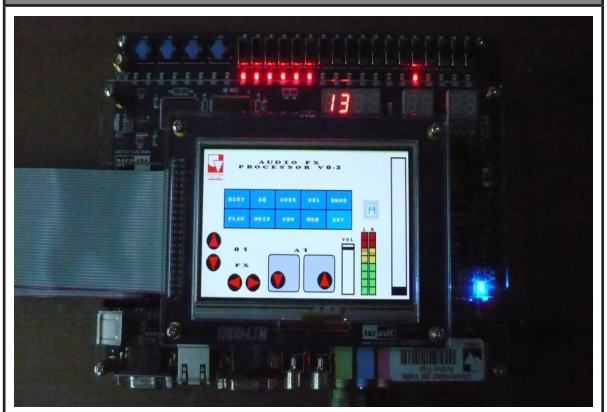


Figure 13. Audio signal at codec analog input

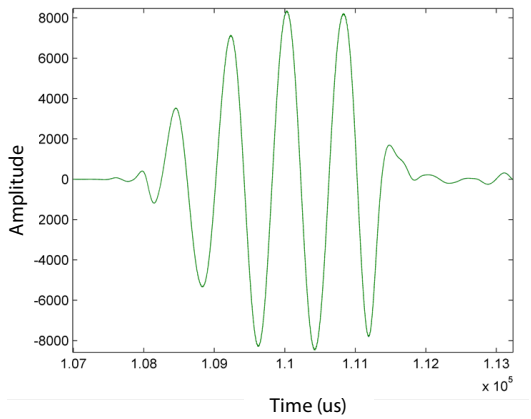
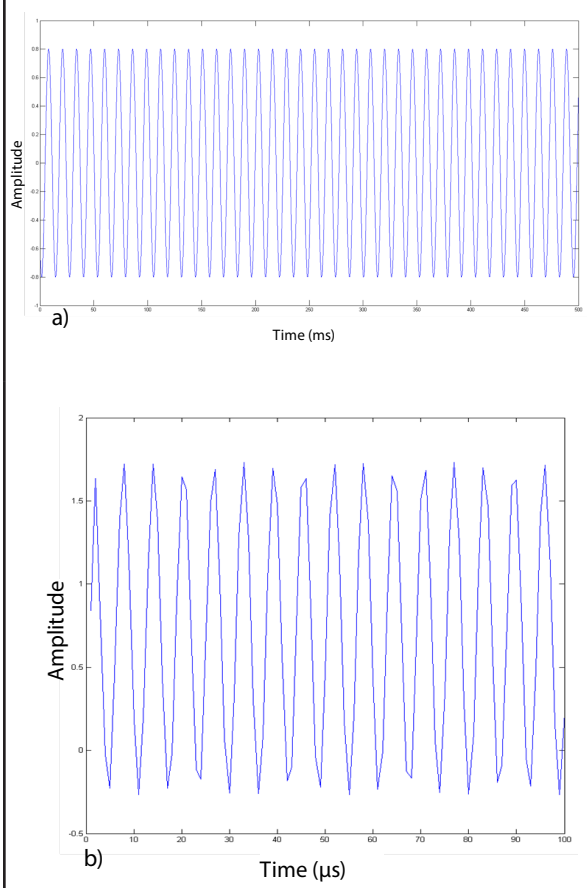


Figure 14. Reverb effect output signal**: a) Matlab simulation. b) processor.**



The output signal Y_i from the audio effect processor is connected to the codec's digital input. The codec's analog output is connected to the audio input of a PC. The analog signal generated by the codec is digitalized by the PC's codec, and this signal is graphed using Matlab. **Figure 14** shows the reverb effect's output signal simulated in Matlab using a sinusoidal signal as input and the processor's output signal Y_i for the same effect using the audio signal as input.

We can observe in **Figure 14** that the reverb effect's output signal simulated in Matlab is similar to the processor's output signal for the same effect.

3. RESULTS

The audio effect processor, the three ROM memories, the FSM2 state machine, and the LTM-SoPC controller are synthesized on the EP2C70F896C6 FPGA using 30,040 ALUTs (Adaptative Look-Up Tables) and 10,239 registries which correspond to 44% and 15% of the FPGA's area resources, respectively. The processor's maximum operating frequency is 195.62 MHz. In addition, **Table 3** presents the area resources used in the FPGA for each audio effect block with regards to the microsystem's total resources. We can conclude from the results in **Table 3** that the chorus effect uses the least area resources, while the wah-wah effect uses the greatest amount of area resources, corresponding to 80% of the microsystem's total resources.

Table 4 shows the area resources of the microsystem's blocks in the FPGA and the percentage of area resources for each microsystem block with regards to the microsystem's total area resources. We can conclude from these results that the FX arrangement block uses the greatest amount of area resources, which is 92% for ALUTs and 62% for the registries.

In order to experimentally verify (an auditory procedure) the microsystem's functioning for processing audio effects, we implemented two processor configurations and used two signals: a sinusoidal signal and a song played on an MP3 player, shown in **Figures 15** and **16**, respectively.

Figure 15. Sinusoidal signal

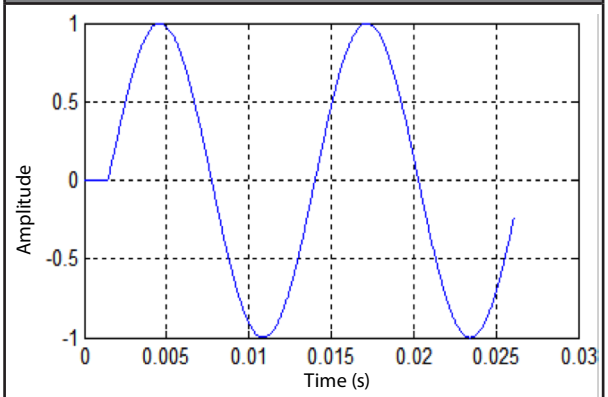
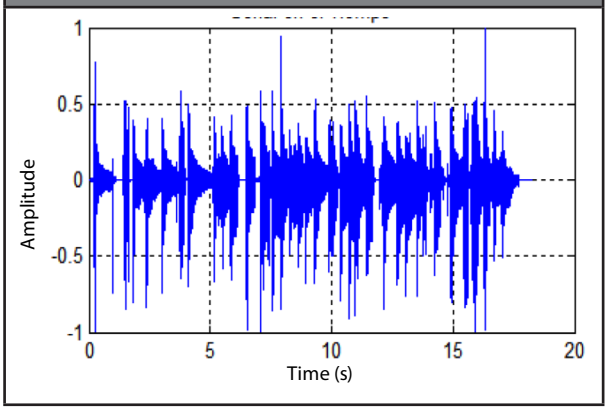


Figure 16. Song played on MP3 player



The first processor configuration consists of connecting in cascade the noise gate, hard clipping, and delay effects, whose operation parameters are: threshold 0.1, threshold 0.4, and delay 10 ms, respectively. The processor's output signals for this configuration, using the sinusoidal signal and the song as inputs, are graphed in Matlab. These are shown in **Figures 17** and **18**, respectively.

Figure 17. Processor output signal for first configuration, sinusoidal input

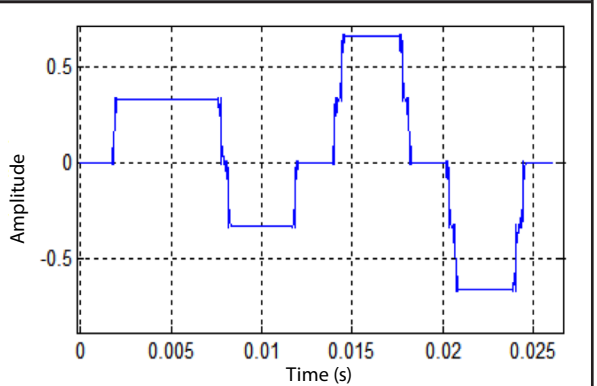


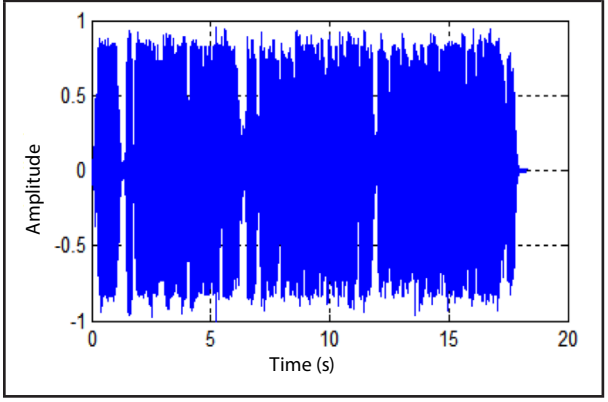
Table 3. Audio effects area resources

Compressor	ALUTs	% of ALUTs in microsystem	Registries	% of registries in microsystem
Expander	84	0.28	50	0.49
Noise gate	84	0.28	50	0.49
Soft and hard clipping	65	0.22	48	0.47
Sigmoidal distortion	80	0.27	49	0.48
Sigmoidal piecewise distortion	1225	4.08	891	8.70
Polynomial distortion	231	7.69	131	1.28
Ring modulator	64	0.21	63	0.61
Delay	539	1.79	405	3.95
Chorus	49	0.16	49	0.48
Flanger	44	0.15	44	0.43
Reverb	534	1.78	390	3.81
Phaser	214	0.71	49	0.48
Wah-wah	516	1.72	386	3.77
Wah-wah	24009	79.92	3743	36.55

Table 4. Block area resources in programmable microsystem for digital audio effects

Block	ALUTs	% of ALUTs in microsystem	Registries	% of registries in microsystem
FX arrangement	27738	92.34	6348	62.00
Mod_par	278	0.92	0	0
Control Unit	2025	6.74	3891	38.00

Figure 18. Processor output signal for first configuration, song input



The second processor configuration connects in cascade the compressor, soft clipping, ring modulator, and reverb effects, whose operation parameters are: threshold 0.4 and attenuation 0.6; threshold 0.6; sinusoidal with frequency of 150 Hz; and attenuation 4 and delay 15 ms, respectively. The processor's output signals for the second configuration, using the previous signals, are graphed in Matlab. These are shown in **Figures 19** and **20**, respectively.

Figure 19. Processor output signal for second configuration, sinusoidal input

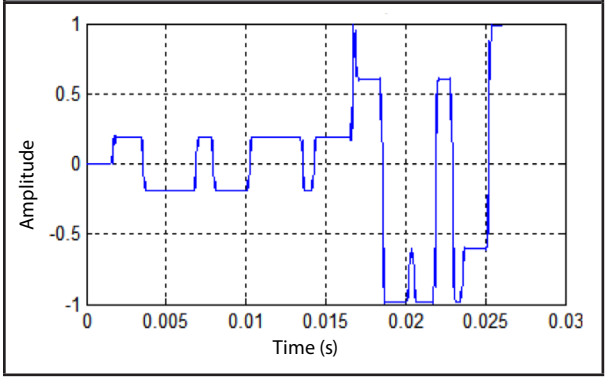
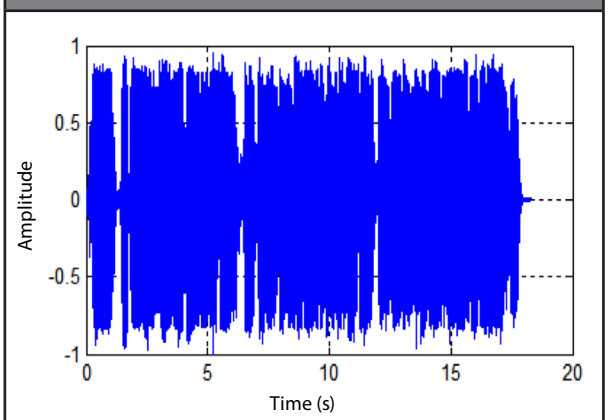


Figure 20. Processor output signal for second configuration, song input



From **Figures 17** through **20**, we can observe that the test signals are modified by the audio effects programmed in the processor; that is, digitalized input signal Xi is processed by the respective sequence of effects. However, in order to verify the correct processing of the effects configured in the processor, we must complete a digital processing of the output signal using the transformed FFTs or Wavelets, or we must consult someone with an excellent musical ear.

Considering the literature reviewed, a similar audio effect processor implementation is presented by Pfaff et al. (2007). In this study, a processor synthesized in the EP2C35F FPGA was designed using hardware/software co-design, and five effects which could be configured by the user were implemented. However, our processor has more effects, and they are organized in an order that allows us to complete correct processing of the audio signal. In addition, the authors mentioned do not present synthesis and verification results. It is therefore not possible to make a real comparison with our processor.

4. CONCLUSIONS

This article presents the implementation of a microsystem for processing digital audio effects based on an application-specific reconfigurable processor. The main advantage of this microsystem in comparison to implementations based on DSPs, PCs, or GPUs is that it allows us to obtain a very high sampling rate to process high-quality audio in real time. In other words, the hardware system based on the digital audio effect processor could be used professionally.

In addition, the user can generate uncommon audio effects due to the fact that he or she can configure a chain of effects of the same type. This is due to the ease and flexibility of modifying the effect parameters and the processor's low processing latency.

The processor is configured using a graphical user interface based on an LCD touch screen. In this case, the user can select one of the 16 predetermined audio effect configurations or store 16 new configurations. This interface projects the system as a prototype that could generate an FPGA-based commercial product.

The microsystem is described in VHDL, synthesized in the EP2C70F896C6 FPGA, and implemented in the DE2-70 development system with an LCD touch screen. The synthesis and verification tests on hardware allow us to conclude that the audio effects processor has a maximum sampling rate of 195.62 MSPS and that it can be used as an embedded core in an SoC for audio applications, for a pedal, for example.

ACKNOWLEDGEMENTS

The authors would like to thank Professor Jaime Andrés Arteaga for his help and assessment on the design of the graphic interface using an LCD touch panel and also the student Joaquín Andrés Alarcón for his help in verifying the chorus and reverb effects.

REFERENCES

Altera. Documentation: SOPC Builder [online] 2011: [4 October 2013] Retrieved from: < http://www.altera.com/literature/lit-sop.jsp>

Altera. FIR Compiler user guide [online] 2011: [4 October 2013] Retrieved from: <http://www.altera.com/literature/ug/fircompiler_ug.pdf>

Altera. Quartus II Development Software Handbook v9.0 [online] 2009: [4 October 2013]. Retrieved from:
<http://www.altera.com/literature/hb/qts/archives/quartusii_handbook_9.0.pdf>

Berdahl, Edgar and Smith, Julius O. (2006). Some Physical Audio Effects. International Conference on Digital Audio Effects. Montreal, Canada. (September 18-20), pp. 165-168.

Byun, Kyungjin; Kown, Young-Su; Koo, Bon-Tae; Eum, Nak-Woong; Jeong, Koang-Hui and Koo, Jae-Eul (2009). Implementation of Digital Audio effect SoC. IEEE Interntational Conference Multimedia and Expo. New York, U.S.A. (June 28 – July 2), pp. 1194-1197.

Fernández, Pablo and Casajús Javier (2000). Multiband Approach to Digital Audio FX. IEEE International Conference Multimedia and Expo. Nueva York, U.S.A. (July 30 – August 2), pp. 1747-1750.

Guillemard, Mijail; Ruwwe, Christian and Zölzer, Udo (2005). J-DAFX - Digital Audio Effects in JAVA. International Conference on Digital Audio Effects. Madrid, España. (September 20-22), pp. 1-6.

Holters, Martin and Zölzer, Udo (2006) Parametric higher-order Shelving filters. European Signal Processing Conference. Florence, Italy. (September 4-8).

Karjalainen, Matti; Penttinen, Henry and Välimäki, Vesa (2000) Acoustic Sound from the Electric Guitar Using DSP Techniques. IEEE International Conference on Acoustics, Speech and Signal Processing. Istanbul, Turkey. (June 5-9), pp. 773-776.

Khosravi, P. On the Design of Spectral Tools in Blue. Csound Journal [online] 2007, (7): [4 October 2013] Retrieved from: <http://www.csounds.com/journal/issue7/onTheDesignOfSpectralToolsInBlue.html>

Liévano, P. P.; Espinosa-Duran, J. M. y Velasco-Medina, J (2013). "Implementación de Algoritmos para efectos de audio digital con alta fidelidad usando hardware programable." Revista Ingeniería y Universidad, Vol. 17, No. 1 (January), pp. 93-108.

Ling, Fan; Khuen, Fung and Radhakrishnan, Damu (2000) An Audio Processor Card for Special Sound Effects. IEEE Midwest Symposium on Circuits and Systems. Michigan, USA. (August 8-11), pp. 730-733.

Oboril, David; Barik, Miroslav; Schimmel, Jiri; Smekal, Zdenek and Krkavec, Petr (2000) Modelling Digital Musical Effects for Signal Processors, Based on Real Effect Manifestation Analysis. Cost G-6 Conference on Digital Audio Effects. Verona, Italia. (December 7-9), pp. 1-6.

Pérez, A. *Estudio de efectos de audio para guitarra, e implantación mediante DSP* Undergraduate thesis, Escuela

Técnica Superior de Ingeniería, U. P. Comillas, Madrid, Spain, 2006.

Pfaff, Markus; Malzner, David; Seifert, Johannes; Traxler, Johannes; Weber, Horst and Gerhard Wiendl (2007) Implementing Digital Audio Effects Using Hardware/ Software Co-Design Approach. International Conference on Digital Audio Effects. Bordeux, Francia. (September 10-15), pp. 1-8.

Schimmel, Jiri; Smekal, Zdenek and Krkavec, Petr (2002) Optimizing Digital Musical Effect Implementation for Multiple Processor DSP Systems. International Conference on Digital Audio Effects. Hamburg, Germany. (September 26-28), pp. 81-84.

Sound laboratory Zoom. Zoom70II guitar operation manual. [online] 2013: [4 October 2013] Retrieved from: <http://www.zoom.co.jp/downloads/707ii/software/>

Tsai, Pei-Yin; Wang, Tien-Ming and Su, Alvin (2010) GPU-Based Spectral Model Synthesis for Real-Time Sound Rendering. International Conference on Digital Audio Effects. Graz, Austria. (September 6-10), pp. 1-5.

Verfaille, Vincent; Zölzer, Udo and Arfib, Daniel (2006) "Adaptive Digital Audio Effects (A-DAFx): A New Class of Sound Transformations." *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 5, (September), pp. 1817-1831.

Zölzer,U. DAFX - *Digital Audio Effects*. Chichester, England: J. Wiley & Sons, 2002.

TO REFERENCE THIS ARTICLE / PARA CITAR ESTE ARTÍCULO / PARA CITAR ESTE ARTIGO /

Espinosa-Duran,J.M.; Liévano-Torres, P.P.; Rentería-Mejía, C.P.; Velasco-Medina, J. (2014). Design of a Programmable Microsystem for Digital Audio Effects Using FPGAs. Revista EIA, 11(22) July-December, pp. 123-136. [Online]. Available on: http:/dx.doi.org/10.14508/reia.2014.11.22.133-146