Implementación de algoritmos para efectos de audio digital con alta fidelidad usando *hardware* programable[1]

Algorithm Implementation in High-Fidelity Digital Audio Effects Using Programmable Hardware[2]

Implementação de algoritmos para efeitos de áudio digital com alta fidelidade usando-se *hardware* programável[3]

Pedro P. Liévano-Torres[4]
John M. Espinosa-Durán[5]
Jaime Velasco-Medina[6]

SICI: SICI: 0123-2126(201301)17:1<93:IAPEAD>2.0.TX;2-H

[1] Fecha de recepción: 14 de marzo de 2012. Fecha de aceptación: 11 de septiembre de 2012. Este artículo se deriva de un proyecto de investigación denominado *Diseño de un procesador multi-efecto para guitarra eléctrica basado en FPGA* (código 2627), desarrollado por el grupo de investigación Bionanoelectrónica de la Universidad del Valle, Cali, Colombia.

[2] Reception date: March 14th 2012. Admission date: September 11th 2012. This paper originated from a research project titled *Diseño de un procesador multi-efecto para guitarra eléctrica basado en FPGA* (code 2627), carried out by the Bio Bionanoelectronics Research Group of the Universidad del Valle, Cali, Colombia.

[3] Data de recepção: 14 de março de 2012. Data de aprovação: 11 de setembro de 2012. Este artigo origina-se do projeto de pesquisa chamado *Diseño de un procesador multi-efecto para guitarra eléctrica basado en FPGA* [*Projeto de um processador multi-efeito para guitarra baseado em FPGA*] (código 2627), desenvolvido pelo grupo de pesquisa Bionanoelectrónica da Universidad del Valle, Cali, Colômbia.

[4] Ingeniero electrónico, Universidad del Valle, Cali, Colombia. Correo electrónico: pablolievanot@gmail.com.

[5] Ingeniero electrónico, Universidad del Valle, Estudiante de Doctorado en Química, Indiana University, Bloomington, Estados Unidos. Correo electrónico: michaele@univalle.edu.co.

[6] Ingeniero electricista, Universidad del Valle, Cali, Colombia. DEA en Microelectrónica, Universidad Joseph Fourier, Grenoble, Francia. PhD en Microelectrónica, Instituto Nacional Politécnico de Grenoble. Profesor titular, Universidad del Valle, Cali, Colombia. Correo electrónico: jaime.velasco@correounivalle.edu.co.

Resumen

Generalmente, los efectos de audio digital son implementados en *software* usando DSP o PC; sin embargo, hoy en día se están considerando algunas implementaciones que usan *hardware* dedicado. Este artículo presenta la implementación en *hardware* de quince algoritmos para el procesamiento en tiempo real de efectos de audio digital con tasas de muestreo de hasta 100,88 MSPS y alta fidelidad. Los diseños se simularon usando DSP-Builder y se sintetizaron en el FPGA EP2C70F896C6. Las pruebas de verificación en *hardware* de los efectos implementados muestran que los bloques diseñados pueden usarse como IP *cores* para el diseño de un procesador multiefecto embebido en un SoC.

Palabras clave

Efectos de audio digital, sistemas embebidos, DSP, procesamiento en dominio dinámico.

Abstract

Digital audio effects are generally implemented in software using DSP or PC; nevertheless, some implementations using dedicated hardware are currently being considered. This paper presents the implementation in hardware of fifteen algorithms for the real-time processing of digital audio effects with sampling rates up to 100.88 MSPS and high-fidelity. Designs were simulated using DSP-Builder and were synthesized in the FPGA EP2C70F896C6. The hardware verification tests of the implemented effects show that the designed blocks can be used as IP cores for the design of a multi effects processor embedded in a SoC.

Keywords

Digital audio effects, embedded systems, DSP, dynamic domains processing.

Resumo

De modo geral, os efeitos de áudio digital são implementados em software usando-se DSP ou PC; no entanto, hoje em dia estão sendo consideradas algumas implementações que usam *hardware* dedicado. Este artigo apresenta a implementação em *hardware* de quinze algoritmos para o processamento em tempo real de efeitos de áudio digital com taxas de amostragem de até 100,88 MSPS e alta fidelidade. Os projetos foram simulados usando-se DSP-Builder e foram sintetizados no FPGA EP2C70F896C6. As provas de verificação em *hardware* dos efeitos implementados demonstram que os blocos projetados podem ser usados como IP *cores* para o projeto de um processador multiefeito embarcado em um SoC.

Palavras chave

Efeitos de áudio digital, sistemas embarcados, DSP, processamento domínio dinâmico.

Introducción

Debido a que los músicos pretenden generar un sonido diferente y único en el momento de interpretar un instrumento, los efectos de audio son una alternativa para lograr este propósito. Actualmente, estos efectos se generan usando sistemas de procesamiento analógico o digital y se pueden clasificar, según el dominio de procesamiento, en dinámico, frecuencia y usando retardos (Zölzer, 2002).

En la literatura se encuentran trabajos que presentan implementaciones de efectos de audio usando procesadores digitales de señales (DSP), computadores (PC) y unidades de procesamiento gráfico (GPU) (Verfaille, Zölzer y Arfid, 2006; Berdahl y Smith, 2006; Guillemard *et al.*, 2005; Schimmel, Smekal y Krkavec, 2002; Fernández y Casajus, 2000; Ling *et al.*, 2000; Oboril *et al.*, 2000; Karjalainen *et al.*, 2000; Tsai, Wang y Su, 2010). Sin embargo, en estos trabajos no se obtienen altas tasas de muestreo y la mayoría de los efectos de audio implementados en PC o DSP se diseñaron usando retardos. También en la literatura se encuentran algunas implementaciones de efectos de audio usando un *Field Programmable Gate Array* (FPGA).

Pfaff *et al.* (2007) presentan la implementación en un FPGA de los efectos *chorus*, *delay*, *echo cancellation*, *flanger* y *wah-wah* usando técnicas de codiseño *hardware-software*. El *flanger* se implementó usando una *look-up table* para generar la señal sinusoidal, y el *wah-wah*, usando un filtro paso-todo de segundo orden y una *look-up table* para variar la frecuencia de corte. No obstante, este trabajo no presenta las pruebas de verificación de los diseños. Byun *et al.* (2009) describe la implementación de los efectos *reverb*, *chorus*, *flanger*, *phaser*, *tremolo*, *auto wah*, *pitch shift*, *distortion* y *mulit-band equalizator* en un DSP embebido en un FPGA usando lenguaje C. Este artículo no describe con detalle los algoritmos usados para implementar los efectos y no presenta las pruebas de verificación de los diseños.

Teniendo en cuenta lo anterior y con el propósito de implementar efectos de audio en tiempo real, en este trabajo presentamos la implementación en un FPGA de quince algoritmos para efectos de audio en los tres dominios de procesamiento. En este caso, se implementaron ocho efectos en el dominio dinámico,

cinco usando retardos y uno en el dominio de la frecuencia. Los diseños son simulados con DSP-*Builder*, y las pruebas de verificación en *hardware* se llevan a cabo usando un reproductor MP3 y un PC.

Este artículo está organizado de la siguiente manera: en las secciones 1 y 2 se describen los algoritmos y el diseño de los efectos de audio, respectivamente; en la sección 3 se presentan los resultados de simulación y las pruebas de verificación, y en última sección se presentan las conclusiones y el trabajo futuro.

1. Descripción de los efectos de audio

1.1. Procesamiento en el dominio dinámico

Generalmente, este procesamiento es no lineal y considera la dinámica de la señal. En este trabajo se implementaron ocho efectos y en la figura 1 se muestran las funciones de transferencia de siete efectos. Los efectos *compressor* y *expander* atenúan la señal de entrada usando un factor *a* configurable cuando su magnitud está por encima y por debajo de un umbral, respectivamente. El efecto *noise gate* atenúa la señal de entrada completamente cuando su magnitud está por debajo del umbral. Estos efectos son descritos por las ecuaciones (1), (2) y (3), respectivamente (Pérez, 2006).

$$y(n)=\begin{Bmatrix} umbral+a*\left(x(n)-umbral\right), x(n)>umbral \\ -umbral+a*\left(x(n)+umbral\right), x(n)<-umbral \\ x(n), |\,x(n)\,|<umbral \end{Bmatrix} \quad (1)$$

$$y(n)=\begin{Bmatrix} x(n), |\,x(n)\,|>umbral \\ a*x(n), |\,x(n)\,|<umbral \end{Bmatrix} \quad (2)$$

$$y(n)=\begin{Bmatrix} x(n), |\,x(n)\,|>umbral \\ 0, |\,x(n)\,|<umbral \end{Bmatrix} \quad (3)$$

El efecto *distortion* se genera cuando un sistema de amplificación alcanza la saturación y produce un sonido estridente. Algunos efectos típicos de distorsión son:

Distortion clásica o *clipping*, descrita por la ecuación (4), en que la señal de salida es el valor de saturación cuando la señal de entrada sobrepasa el valor umbral; en caso contrario, es la entrada multiplicada por un factor de amplificación *A* (Zölzer, 2002):

$$y(n)=\begin{Bmatrix} A*x(n), |\,x(n)\,|<umbral \\ saturación, |\,x(n)\,|>umbral \end{Bmatrix} \quad (4)$$

Sigmoidal piecewise distortion, descrita por la ecuación (5), donde la señal de salida es 1 si la entrada es mayor que 1/3; -1 si la entrada es menor que -1/3; una curva de compresión suave si el valor absoluto de la entrada es mayor que 1/6 y menor que 1/3, y una línea recta si la entrada es mayor que -1/6 y menor que 1/6 (Zölzer, 2002). Esta ecuación es modificada para la implementación *hardware*:

$$y(n)=\begin{Bmatrix} 1, x(n)>\frac{1}{3} \\ \frac{3-(2-6*x(n))^2}{3}, \frac{1}{6}<x(n)<\frac{1}{3} \\ 4*x(n), -\frac{1}{6}<x(n)<\frac{1}{6} \\ -\frac{3-(2-6*|x(n)|)^2}{3}, \frac{1}{3}<x(n)<-\frac{1}{6} \\ -1, x(n)<-\frac{1}{3} \end{Bmatrix} \quad (5)$$

Sigmoidal distortion, descrita por la ecuación (6). Aquí la señal de salida es una curva en forma de *S* (McGovern, 2004):

$$y(n)=\frac{(1+k)*x(n)}{1+k*|x(n)|}, donde\ k=\frac{2*a}{1-a} \quad (6)$$

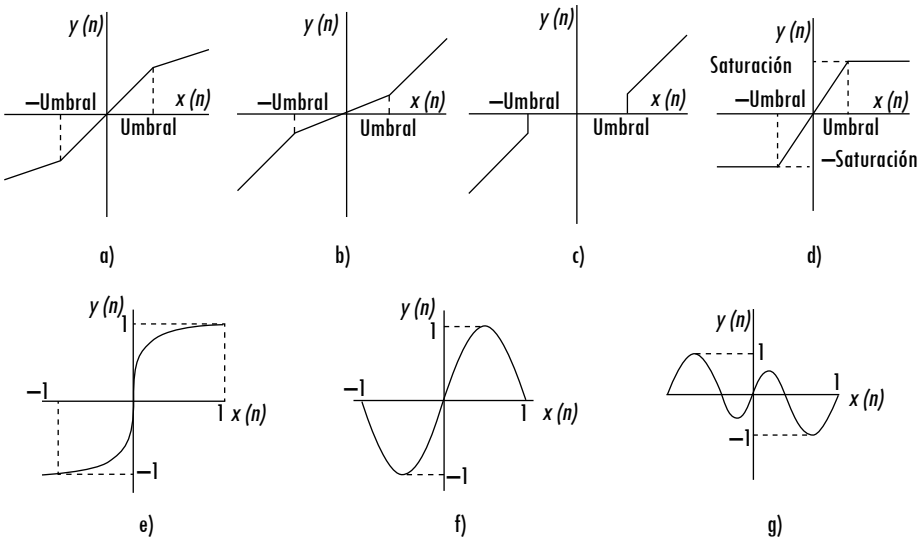
Polynomial distortion, descrita por la ecuación (7). La señal de salida es una función polinomial, es decir, la magnitud de la salida crece hasta alcanzar su valor máximo (1 o -1) y decrece después de este valor:

$$y(n)=\begin{Bmatrix} 4x(n)+4x(n)^2, x(n)<0 \\ 4x(n)-4x(n)^2, x(n)\geq 0 \end{Bmatrix} \quad (7)$$

El efecto *ring modulator* es descrito por la ecuación (8), y se genera al multiplicar la señal de entrada por una señal sinusoidal (Zölzer, 2002):

$$y(n) = (x(n) * seno\ (w * n) \quad (8)$$

Figura 1. Función de transferencia de los efectos: a) *Compressor*, b) *Expander*, c) *Noise gate*, d) *Clipping*, e) *Sigmoidal piecewise distortion*, f) *Polynomial distortion* y g) *Ring modulator*



Fuente: presentación propia de los autores.

1.2. Procesamiento usando retardos

Este procesamiento se basa en usar la señal de entrada retardada. En este trabajo se implementaron cinco efectos y en la figura 2 se presentan los diagramas de bloques de cada efecto. El efecto *delay* es descrito por la ecuación (9), donde *del* es el retardo (Zölzer, 2002). Este es el efecto básico para el procesamiento usando retardos y es generado al sumar la señal de audio con ella misma usando un retardo entre 1 y 50 ms.

$$y(n) = x(n) + x\,(n + del) \tag{9}$$

El efecto *chorus* se describe en la ecuación (10) y emula dos o más músicos cuando interpretan simultáneamente el mismo tipo de instrumento y la misma pieza musical (Zölzer, 2002). En este caso, las señales de audio generadas no están sincronizadas, están desfasadas y tienen diferentes amplitudes. Por lo tanto, este efecto se obtiene usando un factor de atenuación aleatorio g y un retardo *del* entre 10 y 25 ms.

$$y(n) = x(n) + g * x\,(n + del) \tag{10}$$

El efecto *reverb* lo describe la ecuación (11) y lo genera la señal de audio y sus reflexiones acústicas (Zölzer, 2002). Este efecto es generado emulando reflexiones acústicas, cada una con un retardo *n*del* y un factor de atenuación aleatorio g_n.

$$y(n) = x(n) + g_1 x(n + del) + g_2 x(n + 2del) + g_3 x(n + 3del) + g_4 x(n + 4del) \quad (11)$$

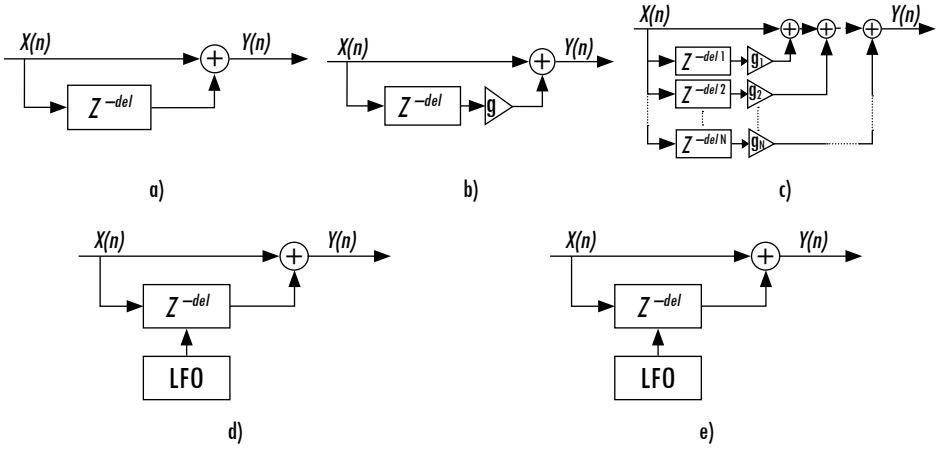
El efecto *flanger* se describe en la ecuación (12) y emula el sonido producido por una turbina de avión (Zölzer, 2002). Este efecto es generado usando un retardo del_{LFO}, controlado por un *Low Frecuency Oscillator* (*LFO*) con frecuencia entre 66 Hz y 1 KHz.

$$y(n) = x(n) + x(n + del_{LFO}) \quad (12)$$

El efecto *phaser*, descrito por la ecuación (13), es generado cuando se suma la señal de audio con ella misma usando un desfase entre 0° y 180°. El retardo del_{LFO} para el *phaser* es controlado con un *LFO* que trabaja con una frecuencia menor a 1 Hz.

$$y(n) = x(n) - x(n + del_{LFO}) \quad (13)$$

Figura 2. Diagrama de bloques: a) *Delay*, b) *Chorus*, c) *Reverb*, d) *Flanger* y e) *Phaser*



Fuente: presentación propia de los autores.

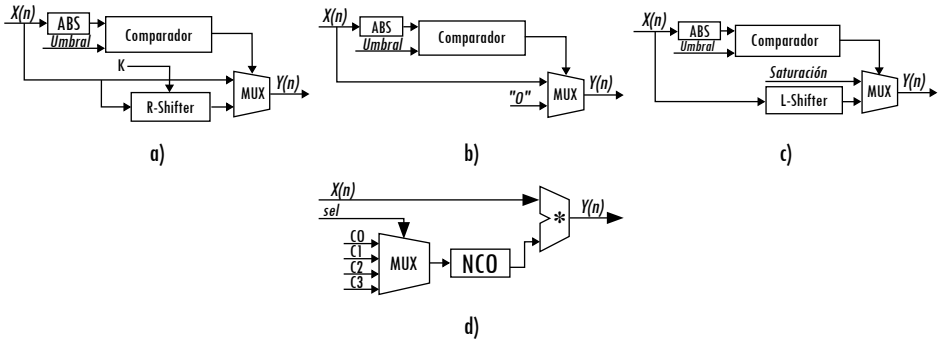
2. Implementación *hardware* de los efectos de audio

2.1. Implementación en el dominio dinámico y la frecuencia

Los efectos de audio *compressor* y *expander* se implementan usando un *right-shift register* con desplazamiento aritmético para multiplicar la señal por el factor 2^{-k}, donde k es el número de desplazamientos; un bloque *ABS* para calcular el valor absoluto; un comparador para evaluar si la señal de entrada está por encima o por debajo del umbral seleccionado, y un *multiplexor* para seleccionar la señal de acuerdo con el umbral (figura 3a). El efecto *noise gate* se implementa usando

un bloque *ABS* y un comparador que controla un multiplexor. En este caso, la señal de salida es totalmente atenuada si la señal de entrada está por debajo del umbral (figura 3b). Los efectos *soft* y *hard clipping* se implementan usando un *left-shift register*, un bloque *ABS* y un comparador para controlar un multiplexor, donde la entrada es multiplicada por 2 o 4 para *soft* o *hard*, respectivamente (figura 3c). El efecto *ring modulator* se implementa usando la megafunción *Numeric Controlled Oscillator* (*NCO*), la cual genera una señal sinusoidal discreta. En este caso, el *NCO* se configura para generar señales sinusoidales con frecuencias de 0,5; 1; 1,5 y 2 KHz usando un muestreo de 44,1 KHz (figura 3d).

Figura 3. Diagrama de bloques de los efectos: a) *Compressor* y *expander*, b) *Noise gate*, c) *Soft* y *hard clipping* y d) *Ring modulator*

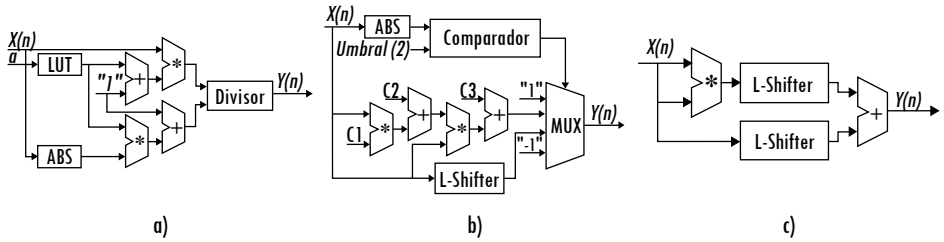


Fuente: presentación propia de los autores.

El efecto *sigmoidal distortion* se implementa usando dos sumadores, un bloque *ABS*, una *look-up table*, dos multiplicadores y un divisor (figura 4a). El efecto *sigmoidal piecewise distortion* se implementa usando un multiplexor, un bloque *ABS*, un comparador, dos sumadores, un *left-shift register* y dos multiplicadores (figura 4b). El efecto *polynomial distortion* se implementa usando dos *left-shift registers*, un sumador y un multiplicador (figura 4c). Adicionalmente, para obtener una mayor fidelidad en estos efectos de distorsión se implementa un filtro paso-bajo en la salida.

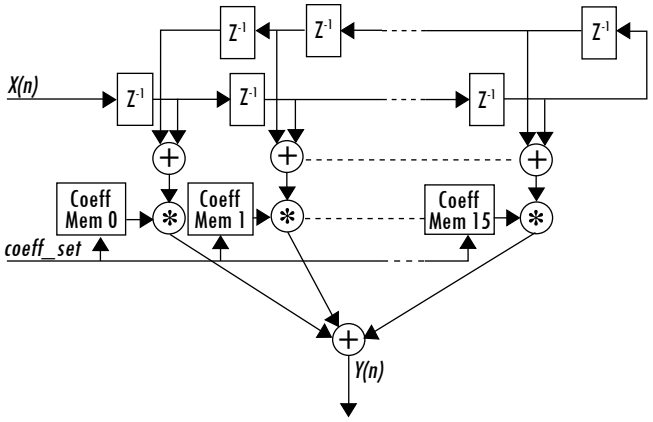
El efecto *wah-wah* emula a una persona pronunciando la palabra *wah-wah* y se implementa usando un filtro pasa-banda de orden 30 con frecuencia de corte variable (figura 5). Este filtro es efectúa con *FIR compiler MegaCore Function* de Altera.

Figura 4. Efectos a) *Sigmoidal distortion*, b) *Sigmoidal piecewise distortion* y c) *Polynomial distortion*



Fuente: presentación propia de los autores.

Figura 5. Efecto *wah-wah*



Fuente: presentación propia de los autores.

2.2. Implementación usando retardos

Los efectos usando retardos se diseñan con un *buffer* circular implementado en la megafunción RAM de doble puerto. Las direcciones para escribir en la RAM las genera un contador y para leer se usa un direccionamiento indexado, donde el índice *retardo* es descrito por la ecuación (14) (figura 6a).

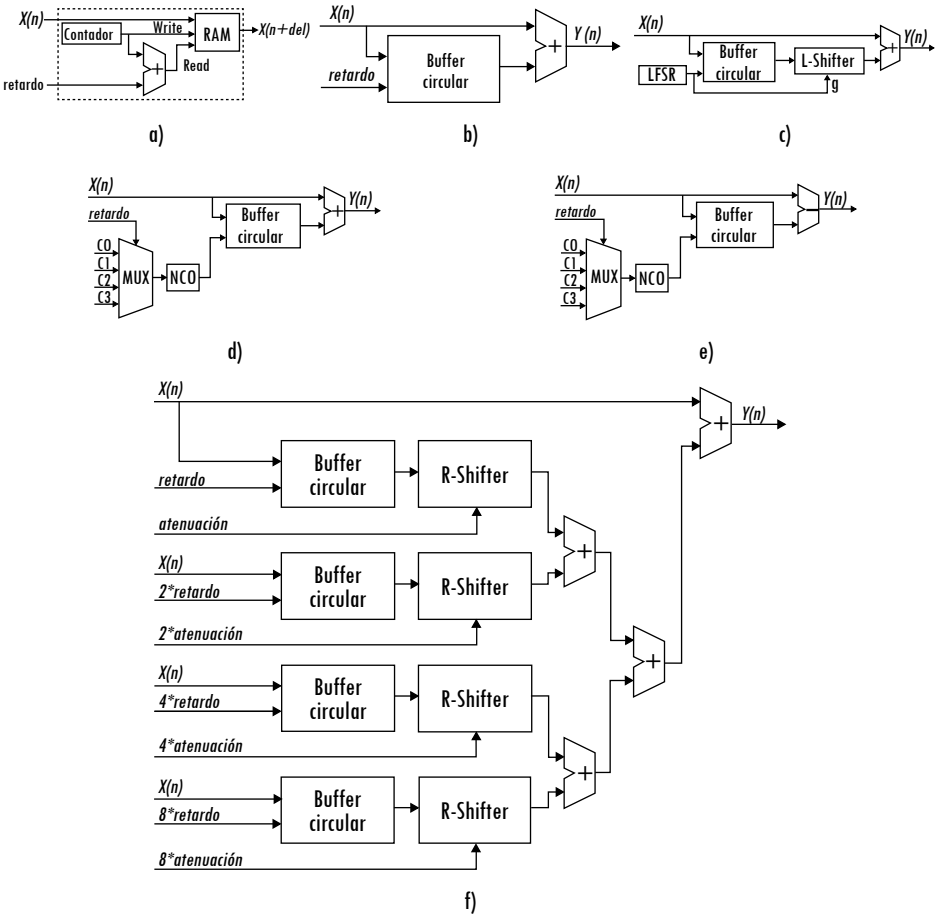
$$retardo = \frac{del}{1000} * fs \tag{14}$$

El efecto *delay* se implementa usando un *buffer* circular y un sumador (figura 6b), y el efecto *chorus*, usando un *buffer* circular, un sumador, un *left-shift register* y un generador de números seudoaleatorios basado en un *Linear Feedback Shift Register* (*LFSR*) de quinto orden (Peralta, Duchén y Vázquez, 2009) para

generar la dirección de lectura del *buffer* (figura 6c). En este efecto, tambien el retardo puede ser generado usando un LFO (Pérez, 2006). El efecto *flanger* se implementa usando un *buffer* circular, un sumador y un *NCO* de frecuencia variable, cuya salida es descrita por la ecuación (15) (figura 6d).

$$NCO_{output} = 1 - \sin\left(f * 2\pi n\right), f = \frac{10}{retardo * fs} \tag{15}$$

Figura 6. Diagrama de bloques: a) *buffer* circular y de los efectos; b) *Delay*, c) *Chorus*, d) *Flanger*, e) *Phaser* y f) *Reverb*



Fuente: presentación propia de los autores.

El efecto *phaser* se implementa usando un *buffer* circular, un restador y un *NCO* de frecuencia variable, cuya salida es descrita por la ecuación (16) (figura 6e).

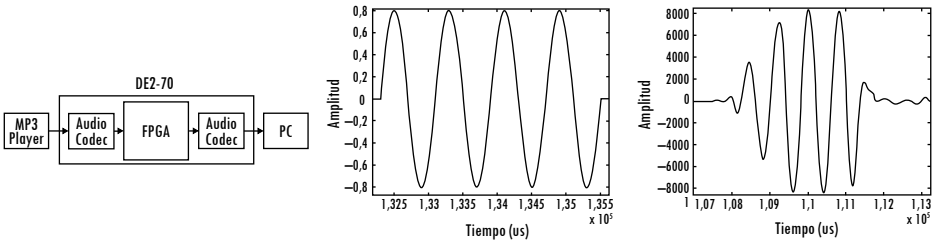
$$NCO_{output} = \sin(f * 2\,nn) \quad (16)$$

El efecto *reverb* se implementa usando cuatro *buffers* circulares, cuatro *right-shift registers* y un sumador paralelo. En este caso, se usan cuatro ganancias y retardos para emular las reflexiones acústicas de la señal de entrada (figura 6f). Adicionalmente, un filtro paso-bajo se debe conectar en la salida del sumador de las señales atenuadas y retardadas.

3. Resultados de pruebas de verificación y simulación

Con el propósito de verificar el correcto funcionamiento de los efectos de audio implementados en *hardware*, se implementó el sistema de prueba mostrado en la figura 7a. En este caso, la señal audio de prueba es leída desde un reproductor MP3 (figura 7b), cuya salida es conectada a la entrada analógica del *codec* de audio de la tarjeta DE2-70 (figura 7c). El *codec* trabaja con una frecuencia de muestreo de 44,1 KHz y genera una señal digital de 16 bits (complemento a dos para valores negativos), la cual es la señal de entrada para los efectos de audio implementados digitalmente.

Figura 7. a) Diagrama de bloques para verificar el funcionamiento de los efectos. b) Señal de audio almacenada en el MP3. c) Señal de audio en la entrada del *codec* de la DE2



a) b) c)

Fuente: presentación propia de los autores.

La señal digital de salida de los efectos implementados es conectada al *codec* de audio de la tarjeta DE2-70, cuya salida analógica es conectada a la entrada de audio de un PC. Esta señal análoga es digitalizada por el *codec* del PC, procesada por Matlab y almacenada en formato .wav. Los efectos son descritos usando VHDL, simulados con DSP-Builder y sintetizados sobre el FPGA EP2C70F896C6. La tabla 1 presenta los recursos de *hardware* y la máxima frecuencia de operación para cada efecto.

Tabla 1. Recursos usados y frecuencia máxima para los efectos de audio

Efecto	Logic Elements	Combinat. Functions	Registers	Memory Bits	Embedded Multipliers	Fmax (MHz)
Compressor	84	64	50	0	0	180,86
Expander	84	64	50	0	0	181,06
Noise gate	65	49	48	0	0	176,24
Soft y hard clipping	80	70	49	0	0	142,11
Sigmoidal distortion	1225	806	891	832	14	100,88
Sigmoidal piecewise distortion	231	170	131	0	12	194,33
Polynomial distortion	64	29	63	0	6	216,73
Ring modulator	539	349	405	12.288	6	200,00
Delay	49	33	49	32.768	0	169,98
Chorus	44	28	44	61.440	0	157,18
Flanger	534	380	390	15.872	4	169,75
Reverb	214	205	49	118.784	0	108,31
Phaser	516	356	386	8704	4	177,46
Wah-wah	24.009	23.138	3743	204	0	166,14
Total	27.738	25.741	6348	250.892	46	

Fuente: presentación propia de los autores.

3.1. Resultados de las pruebas de verificación

Los parámetros de operación y las señales generadas por los efectos de audio en el dominio dinámico son presentados en la tabla 2 y la figura 8, respectivamente. El *sigmoidal distortion* usa un nivel de efecto de 9,9, y el *ring modulator* usa una frecuencia de 500 Hz.

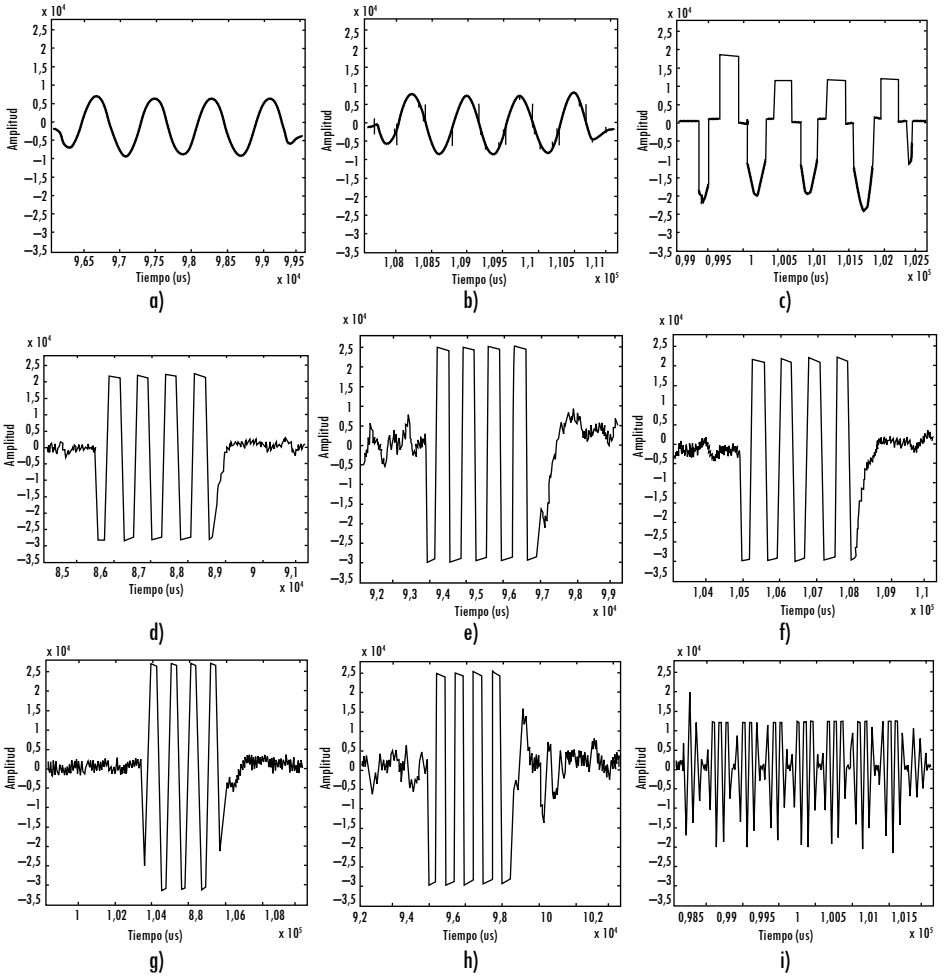
El parámetro *del* para los efectos usando retardos es: 1 ms en el *Delay*, 0 ms en el *Flanger*, 0,875 ms para el *Phaser* y 0,460 ms para la *Reverb.* Este último efecto tiene un factor de atenuación de 0,5. La figura 9 muestra las señales generadas por los efectos usando retardos y el *wah-wah*.

Tabla 2. Parámetros de operación para efectos en el dominio dinámico

Efecto / Parámetro	*Compressor*	*Expander*	*Noise Gate*	*Hard clipping*	*Soft Clipping*
Umbral	1,000	0,000	0,875	0,460	0,670
Atenuación	0,500	0,125	-	-	-

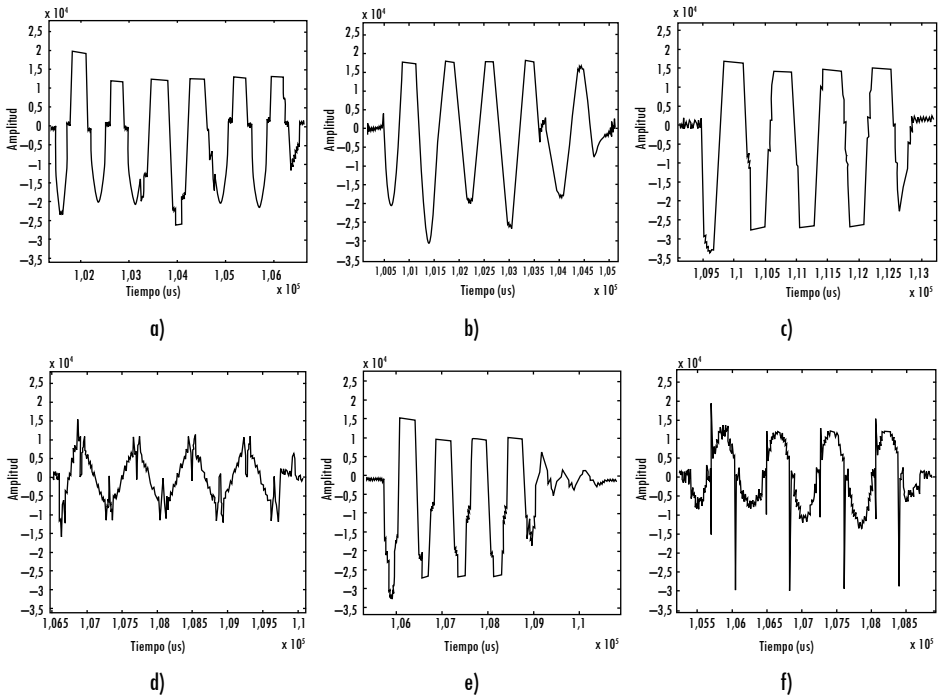
Fuente: presentación propia de los autores.

Figura 8. Señales generadas por los efectos: a) *Compressor*, b) *Expander*, c) *Noise gate*, d) *Soft-clipping*, e) *Hard-clipping*, f) *Sigmoidal distortion*, g) *Sigmoidal piecewise distortion*, h) *Polynomial distortion*, i) *Ring modulator*



Fuente: presentación propia de los autores.

Figura 9. Señales generadas por los efectos: a) *Delay*, b) *Chorus*, c) *Flanger*, d) *Phaser*, e) *Reverb*, f) *Wah-wah*



Fuente: presentación propia de los autores.

3.2. Comparación entre los resultados de simulación y las pruebas de verificación

Con el propósito de verificar el correcto funcionamiento de los efectos implementados en *hardware* se calculó la correlación entre la simulación funcional realizada en DSP-Builder y las pruebas de verificación real. En este caso, el error se calculó usando la ecuación (17), donde *Ym* y *Yr* son los resultados de simulación y las pruebas de verificación real, respectivamente.

$$\text{error} = 100 \times (1 - (Ym * Yr)) \quad (17)$$

Los valores del error para cada efecto se presentan en la tabla 3, donde las diferencias entre la simulación funcional y las pruebas de verificación en *hardware* se deben a que las implementaciones en *hardware* usan aritmética de punto fijo de 16 bits, mientras que la simulación funcional usa aritmética de punto flotante de 64 bits. Sin embargo, estos errores son muy pequeños y el mayor error se presenta en el *flanger* (0,674 %).

Tabla 3. Comparación entre los resultados de simulación y la verificación en hardware

Efecto	Error (%)	Efecto	Error (%)	Efecto	Error (%)
Compressor	0,001	*Sigmoidal distortion*	0,374	*Chorus*	0,001
Expander	0,001	*Sigmoidal piecewise distortion*	0,058	*Flanger*	0,674
Noise gate	0,001	*Polynomial distortion*	0,010	*Reverb*	0,007
Soft-clipping	0,002	*Ring modulator*	0,001	*Phaser*	0,001
Hard-clipping	0,002	*Delay*	0,002	*Wah-wah*	0,098

Fuente: presentación propia de los autores.

Conclusiones y trabajo futuro

Este trabajo presenta la implementación en *hardware* de quince efectos de audio. La principal ventaja de estos diseños respecto a las implementaciones basadas en DSP, PC o GPU es que permiten obtener una mayor fidelidad y una respuesta en tiempo real. Adicionalmente, es posible procesar diferentes efectos en paralelo, lo cual puede generar efectos de audio muy diferentes a los comúnmente usados por los músicos.

Las implementaciones en *hardware* presentan un error por debajo del 1 % respecto a las simulaciones funcionales, donde el mayor error es presentado en el *flanger*. El efecto *sigmoidal distortion* tiene la menor frecuencia máxima ($Fmax$ = 100,88 MHz), debido a que usa un divisor, y el efecto *wah-wah* usa la mayor área, porque se implementa usando 16 filtros pasa-banda de orden 30.

Los efectos son descritos en VHDL, simulados usando DSP-Builder, sintetizados en el FPGA EP2C70F896C6 y verificados usando el kit de desarrollo DE2-70 y un PC. Los resultados de simulación y las pruebas de verificación en *hardware* permiten concluir que los efectos implementados en *hardware* son adecuados para procesar señales de audio y pueden sintetizarse en un FPGA de bajo costo. Por lo tanto, estos diseños se pueden usar en la implementación de aplicaciones de audio embebidas en un SoC.

El trabajo futuro será orientado a diseñar bloques *hardware* utilizando aritmética de punto flotante; un procesador programable para generar efectos de audio, y una interfaz de usuario basada en un *LCD touch panel* para configurar los efectos.

Referencias

BERDAHL, E. y SMITH, J. Some physical audio effects. *9th International Conference on Digital Audio Effects.* Montreal, Canada, 2006, pp. 165-168.

BYUN, K.; KOWN, Y.; KOO, B.; EUM, N.; JEONG, K. y KOO, J. Implementation of digital audio effect SoC. *IEEE International Conference on Multimedia and Expo*. New York, USA, 2009, pp. 1194-1197.

FERNÁNDEZ, P. y CASAJUS, J. Multiband approach to digital audio FX. *IEEE International Conference on Multimedia and Expo*. Nueva York, USA, 2000, pp. 1747-1750.

GUILLEMARD, M.; RUWWE, C. y ZÖLZER, U. J-DAFX - digital audio effects in JAVA. *8th International Conference on Digital Audio Effects*. Madrid, España, 2005, pp. 1-6.

KARJALAINEN, M.; PENTTINEN, H. y VALIMAKI, V. Acoustic sound from the electric guitar using DSP techniques. *IEEE International Conference on Proceedings of the Acoustics, Speech, and Signal Processing*. Istanbul, Turkey, 2000.

LING, F.; KHUEN, F. y RADHAKRISHNAN, D. An audio processor card for special sound effects. *IEEE Midwest Symposium on Circuits and Systems*. Michigan, USA, 2000, pp. 730-733.

MCGOVERN, S. Guitar distortion effect [documento en línea]. 2004. <http://www.mathworks.com/matlabcentral/fileexchange/6639-guitar-distortion-effect> [consulta: 13-03-2012].

OBORIL, D.; BARIK, M.; SCHIMMEL, J.; SMEKAL, Z. y KRKAVEC, P. Modelling digital musical effects for signal processors, based on real effect manifestation analysis. *Conference on Digital Audio Effects*. Verona, Italia, 2000, pp. 1-6.

PERALTA, F.; DUCHÉN, G. y VÁZQUEZ, R. *Información tecnológica* [documento en línea]. 2006, vol. 17. <http://www.scielo.cl/scielo.php?pid=S0718-07642006000300023&script=sci_arttext> [consulta: 13-03-2012].

PÉREZ, A. *Estudio de efectos de audio para guitarra, e implantación mediante DSP*. Tesis de pregrado, Universidad Pontificia Comillas, 2006.

PFAFF, M.; MALZNER, D.; SEIFERT, J.; TRAXLER, J.; WEBER, H. y WIENDL, G. Implementing digital audio effects using hardware/software co-design approach. *10th International Conference on Digital Audio Effects*. Bordeux, Francia, 2007, pp. 1-8.

SCHIMMEL, J.; SMEKAL, Z. y KRKAVEC, P. Optimizing digital musical effect implementation for multiple processor DSP systems. *5th International Conference on Digital Audio Effects*. Hamburgo, Alemania, 2002, pp. 81-84.

TSAI, P.; WANG, T. y SU, A. GPU-based spectral model synthesis for real-time sound rendering. *13th International Conference on Digital Audio Effects*. Graz, Austria, 2010, pp. 1-5.

VERFAILLE, V.; ZÖLZER, U. y ARFID, D. Adaptive digital audio effects (A-DAFx): a new class of sound transformations. *IEEE Transactions on Audio, Speech, and Language Processing*. 2006, vol. 14, núm. 5, pp. 1817-1831.

ZÖLZER, U. *DAFX - Digital Audio Effects*. London: John Wiley & Sons, 2002.