Flexible Pseudorandom Number Generator for Tinnitus Treatment Implemented on a Dspic

A. J. Uriz, *Student Member, IEEE,* P. D. Agüero, J. Moreira, R. M. Hidalgo, E. L. González and J. C. Tulli

***Abstract*— Treatment of tinnitus with masking sounds has reach a significant developed in recent years. It is mainly because it has been possible to implement noise sinthesizers based on random number generators in digital signal processors (DSP), which form a part of almost any digital hearing aid device. One limitation of these methods is that limitations of the DSP architecture prevent pseudo white noise of being generated conform to a real white noise statistics. In this paper, a technique to generate adittive white gaussian noise (AWGN) or noise with uniform distribution using coefficients stored in memory of the DSP program is proposed. An implementation of the technique is carried out on a dsPIC from Microchip and subjective experiments and experimental measurements are performed to validate the performance of the developed technique.**

***Keywords*— Tinnitus treatment, Masking sounds, Hearing aid device, Digital signal processor, Pseudorandom number generator.**

I. Introducción

EL RUIDO generalmente es considerado una parte indeseable de una señal y que debe ser removido para analizar correctamente la misma. Pero, también el ruido es una herramienta muy útil para el análisis y la caracterización de sistemas [1,2]. Por ejemplo, es posible utilizar ruido para analizar sistemas de comunicaciones [3,4], para caracterizar filtros [1], para análisis de compatibilidad electromagnética de sistemas electrónicos (EMC) [5] e incluso para filtros de partículas [6].

Además, en los últimos años el ruido ha sido utilizado con fines médicos, como por ejemplo tratamiento de acúfenos [7-9]. En dicha aplicación se sintetiza ruido (blanco o coloreado) con el fin de enmascarar el acúfeno y mejorar la calidad de vida de la persona que lo padece [8,9].

Para ello, se debe implementar un generador de ruido (basado en un generador de números pseuadoaleatorios) en un dispositivo de asistencia auditiva, el cual para realizar esta función debe estar basado en un procesador digital de señales (DSP).

A. J. Uriz, CONICET - Facultad de Ingeniería (UNMdP), Argentina, ajuriz@conicet.gov.ar
P. D. Agüero, Facultad de Ingeniería (UNMdP), Argentina, pdaguero@fi.mdp.edu.ar
J. Moreira, CONICET - Facultad de Ingeniería (UNMdP), Argentina, casti@fi.mdp.edu.ar
R. H. Hidalgo, Facultad de Ingeniería (UNMdP), Argentina, rhidalgo@fi.mdp.edu.ar
E. L. González, Facultad de Ingeniería (UNMdP), Argentina, elgonzal@fi.mdp.edu.ar
J. C. Tulli, Facultad de Ingeniería (UNMdP), Argentina, jctulli@fi.mdp.edu.ar

En este campo es posible encontrar soluciones [10,11] que implementan generadores de ruido basados en algoritmos fractales u otros tipos de generadores de ruido.

Uno de los problemas que aparecen al tratar de llevar a cabo esta tarea es la complejidad de los algoritmos necesarios para obtener ruido de distribución uniforme con las propiedades estadísticas adecuadas. Además, en algunos casos las funcionalidades principales del dispositivo de asistencia auditiva requieren un elevado porcentaje de memoria de datos y consumen un alto porcentaje del tiempo disponible para procesamiento.

En este trabajo se presenta una técnica para generar secuencias de números pseudoaleatorios en un DSP mediante tablas almacenadas en su memoria de programa. Esta técnica permite mejorar el rendimiento del sistema fundamentalmente porque se reducen las operaciones de conversión entre tipos de números. Además, al estar los coeficientes almacenados en memoria de programa, se reduce el tiempo de ejecución respecto a otras técnicas.

Se realizan experimentos y mediciones para analizar el rendimiento de la técnica. En particular se presentan los resultados del test de Marsaglia [12] para el método propuesto y otros existentes. Adicionalmente, se describe la implementación realizada en un dsPIC de Microchip® [13], con el cual es viable implementar un dispositivo de asistencia auditiva.

El trabajo se organiza de la siguiente manera: en la Sección II se presenta la técnica propuesta. A lo largo de la Sección III se presentan los experimentos y las mediciones realizadas. Por último, en la Sección IV se presentan las conclusiones del trabajo y las tareas a desarrollar en el futuro.

II. Técnica propuesta

Tal como se mencionó previamente, el objetivo de esta técnica es implementar un sintetizador de ruido en un dispositivo de asistencia auditiva. Para ello, se debe tener en cuenta que el sistema debe realizar otras funciones en simultáneo y debe operar en tiempo real. Además, se debe tener en cuenta que actualmente muchos DSP disponen de una gran cantidad de memoria de programa (EPROM) y de menor cantidad de memoria de datos (RAM). Por ello, existe una tendencia a almacenar los datos constantes (tal como ventanas para enventanado, coeficientes para el cómputo de una transformada rápida de Fourier (FFT), etcétera) en ROM. En este trabajo, se propone utilizar esta misma filosofía para construir un generador de ruido almacenando coeficientes en la memoria de programa del DSP.

El primer paso de la técnica consiste en generar los valores pseudoaleatorios en una PC utilizando las funciones *rand()* o *randn()* de MatLab®, dependiendo la naturaleza del ruido a sintetizar. Luego, estos valores generados, se almacenan en la memoria de programa del DSP.

De esta forma, mediante un puntero a una posición de memoria de programa al cual se le sume un valor entero aleatorio *j* (el cual se genera utilizando la arquitectura del DSP), se podrían generar segmentos de datos que contengan los valores generados previamente por MatLab®, pero con un orden de aparición variable dependiendo del generador de saltos.

Una de las ventajas que tiene este generador es que puede ser implementado en un DSP sin una arquitectura dedicada para procesamiento de operaciones en punto flotante. Esto se debe a que las funciones que generan valores pseudoaleatorios en formato punto flotante (tal como se propone en este trabajo), requieren trabajar directamente en ese formato, o convertir a través de un *casting* los valores obtenidos por el generador a un formato de datos que soporte la arquitectura. Esto último es una operación que requiere muchos ciclos de máquina en procesadores sin arquitectura dedicada a operaciones en punto flotante.

Además, debe tenerse en cuenta que los datos generados en MatLab® pueden almacenarse en memoria en el formato mas apropiado para ser sintetizados por el conversor digital a analógico del DSP. Esto se debe a que con la técnica propuesta se puede eliminar el procesamiento de las muestras de ruido.

Con el fin de analizar el generador, se supone que se desean generar vectores de longitud $N=256$. Entonces, los coeficientes generados en MatLab® son almacenados en una tabla que contenga *256* columnas. La cantidad de filas M de dicha tabla será variable y es un factor que se ajustará dependiendo de la cantidad de memoria de programa disponible en el DSP para la implementación del generador.

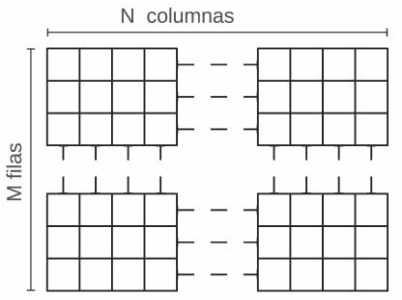

Figura 1. Esquema propuesto para el almacenamiento de los coeficientes.

En la Fig. 1 se presenta un esquema de la implementación propuesta. En ella puede verse que la matriz de los coeficientes generados (en este caso usando la función *rand()*) contiene $N=256$ columnas y M filas. La técnica propuesta funciona de acuerdo al diagrama de flujos de la Fig. 2.

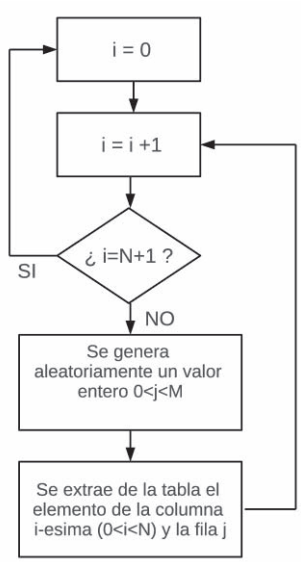

Figura 2. Diagrama de flujos de la técnica propuesta.

Para el primer elemento del vector a generar ($i=1$) se extrae uno de los elementos de la primera columna de la matriz de coeficientes. El índice $1<j<M$ del elemento dentro de la columna se elige fijando el puntero en la posición inicial en memoria de dicha columna e incrementándole un salto aleatorio entero j (menor a M). Este proceso se repite para cada uno de los elementos del vector que se desea generar. Al llegar a la posición $N=256$, el proceso se repite para el vector siguiente.

Como puede verse, el modelo propuesto se diferencia de un generador de señales por síntesis digital directa o DDS (*Digital Direct Synthesis*), en el hecho no recorre la señal almacenada en memoria de forma lineal o en saltos de longitud finita y constante, sino que se varía el desplazamiento a lo largo de la memoria para cada valor obtenido. De esta manera, se obtiene una reducción en la repetición de muestras consecutivas.

Cabe destacar que al aumentar el valor de M se obtendrá un generador de ruido con mejores prestaciones desde el punto de vista estadístico, ya que al disponer de más valores por columna, es posible aumentar el **período de repetición de cada valor.** Es por ello que en los experimentos, también se estudiará como el rendimiento del sistema varía de acuerdo al valor de M.

Otro aspecto interesante de la técnica propuesta es que el tipo de ruido que produce el sintetizador puede ser modificado con solo modificar los valores almacenados en la tabla. De esta forma, si por ejemplo en lugar de utilizar la función de MatLab® *rand()*, se utilizara *randn()*, a la salida del generador se obtendrían números pseudoaleatorios con distribución gaussiana.

En la próxima Sección se presentan los experimentos y mediciones realizados para analizar el funcionamiento del sistema.

III. Experimentos

Con el fin de analizar el rendimiento de la técnica propuesta, en primer lugar se realizaron simulaciones en MatLab®.

Tal como se mencionó, el objetivo de este trabajo es implementar el generador de ruido en un dispositivo de asistencia auditiva basado en un dsPIC33EP256MU806 de Microchip® [13], el cual dispone de *256 kB* de memoria de programa. Trabajos previos [14,15] demuestran que se requiere solo el 10% de memoria de programa para implementar en este dsPIC las funciones mas importantes de un dispositivo de asistencia auditiva. Por lo que, para el generador de números pseudoaleatorios se podría utilizar hasta el 80% de memoria de programa, reservandose el 10% remanente para desarrollos futuros. En estas condiciones, se disponen de *209715* bytes para la implementación del generador, por lo que si se toma *N=256*, y datos de 16 bits, se podría utilizar hasta *M=409*.

Teniendo en cuenta estas condiciones se generaron variaciones de la técnica utilizando *M=100*, *M=200* y *M=400*. Además, con el fin de analizar el rendimiento del sistema con valores de *M* mayores, se incluyeron los casos donde *M=1000*, *M=10000* y *M=100000*.

A. Análisis de la distribución de los datos

En primer lugar se realizó un histograma de los valores obtenidos por el sistema al generar un vector de *NxM= 10240000* elementos.

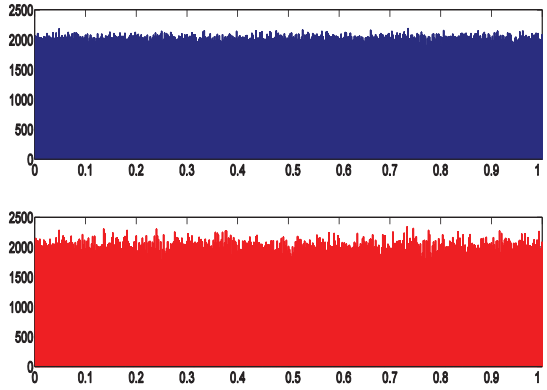

Figura 3. En color azul se presenta el histograma obtenido utilizando la función *rand()* de MatLab®, mientras que en color rojo se presenta el histograma de los datos generados utilizando la técnica propuesta. En este caso los elementos almacenados en memoria tienen distribución uniforme. Puede verse que los histogramas se ajustan a lo esperado.

En la Fig. 3, se presenta en la parte superior, el histograma de un vector de ruido generado utilizando la función *rand()* de MatLab®, mientras que en la parte inferior se presenta el histograma de la señal obtenida por el sistema propuesto para el caso de ruido de distribución uniforme con *M=1000*. En la Fig. 4, se vuelven a presentan dos histogramas, pero para el caso de la función *randn()* de MatLab® y de la señal obtenida por la técnica propuesta para el caso de ruido de distribución normal.

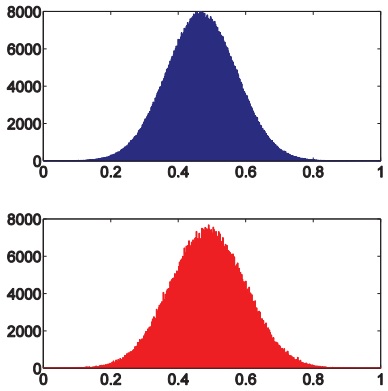

Figura 4. En color azul se aprecia el histograma generado utilizando la función *randn()* de MatLab®, mientras que en color rojo se presenta el histograma de los datos generados utilizando la técnica propuesta. En este caso los elementos almacenados en memoria tienen distribución normal. Nuevamente, los resultados obtenidos se ajustan a lo esperado.

Como puede verse en las Figs. 3 y 4, los histogramas de los vectores obtenidos mediante la técnica propuesta utilizando *M=1000* se ajustan a lo esperado. Además, para el caso del ruido de distribución uniforme se calculó la entropía del histograma (H_{HIST}=*0.9663386534*) y la entropía de Bandt y Pompe [16] (H_{BP}=*0.999994037*). Debe tenerse en cuenta que dichos valores para el caso de la función *rand()* de MatLab son H_{HIST}=*0.996979* y H_{BP}=*0.999941*, respectivamente.

Dado que los resultados obtenidos son favorables, a continuación se presenta una serie de experimentos estadísticos más complejos.

B. Análisis utilizando los tests de Marsaglia

En esta Subsección se presentan los resultados obtenidos por el generador de números al ser examinado utilizando una batería de tests llamada “Test de Marsaglia” [12]. Esta serie de experimentos son ampliamente utilizados para caracterizar generadores de ruido de distribución uniforme. Para ello, se dispone de una herramienta que se puede descargar desde la página de Internet de *The Florida Estate University* [17] de forma gratuita e incluso dispone de otros generadores de números pseudoaleatorios con el fin de comparar los resultados con los obtenidos por el método bajo estudio.

Se obtuvieron vectores generados en base a los distintos valores de *M* y se les ejecutó el test de Marsaglia. Este experimento se caracteriza por entregar como resultado a los denominados valores-p [12] para cada una de 15 pruebas que se le realizan al vector bajo estudio. Dichos valores-p deben tener distribución uniforme en el intervalo entre 0 y 1 [12].

En la Fig. 5 se aprecia un gráfico de cajas (*boxplots*) donde se presentan los resultados obtenidos para este experimento por 5 generadores de números pseudoaleatorios. El método 1 se corresponde a un vector generado utilizando la función *rand()* de MatLab®, la cual debido a que es usada para generar los coeficientes, es tomada como el mejor valor que podrían alcanzar la técnica propuesta. Luego, los métodos 2 a 5, presentan cuatro implementaciones de la técnica propuesta, para *M = 100*, *200*, *400* y *1000*.

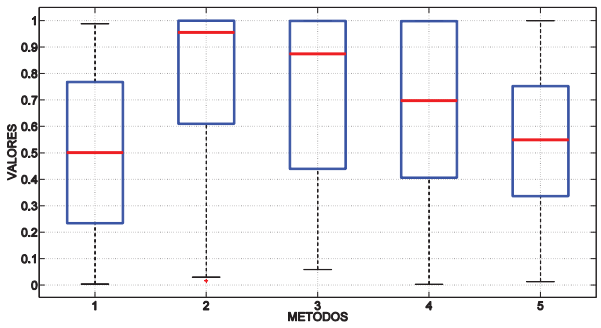

Figura 5. Boxplot de los valores p obtenidos en el test de Marsaglia por la función *rand()* de MatLab® y cuatro implementaciones de la técnica propuesta.

Debido a que los valores de p tienen distribución uniforme, en el gráfico de cajas la mediana debería estar en *0.5* y los primer y tercer cuartil en *0.25* y *0.75*, respectivamente. En la Fig. 5 puede verse que, además de la función *rand()*, los casos con *M=400* y *M=1000* se aproximan a estas condiciones.

Por otro lado, con el fin de estudiar cómo afecta el valor de *M* al desempeño del generador, se graficó mediante boxplots la distribución de los valores de *p*, para cinco valores de *M= 100*, *200, 400*, *1000*, *10000*. Puede verse en la Fig. 6, donde *M* crece de izquierda a derecha, que para grandes valores de dicho parámetro, los resultados se ajustan a lo esperado.

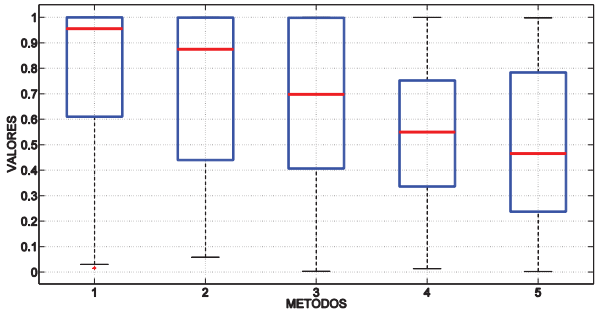

Figura 6. Boxplot de los valores p obtenidos en el test de Marsaglia para la técnica propuesta, con diversos valores de M.

C. Experimentos subjetivos

Con el fin de analizar si un individuo es capaz de discernir entre señales sintetizadas usando diversos valores *M*, se sintetizaron señales de ruido (ya sea de distribución uniforme como normal). Luego, un conjunto de 15 voluntarios (quienes aceptaron de forma expresa un consentimiento informado), evaluaron una serie de audios sintetizados utilizando las técnicas descriptas a continuación.

1- Ruido de distribución uniforme: Generado utilizando la función *rand()* de MatLab® y también la técnica propuesta con *M=100*, *M=200*, *M=400*, *M=1000, M=10000* y *M=40000*.
2- Ruido de distribución normal: Generado utilizando la función *randn()* de MatLab® y también la técnica propuesta con *M=100, M=200, M=400, M=1000, M=10000* y *M=40000*.

Para cada técnica se les pidió que indicaran si era posible discernir entre los sonidos sintetizados generados por cada técnica y las funciones de referencia (*rand()* o *randn()* dependiendo del caso). Para ello, los voluntarios los calificaron en una en una escala de (1 a 5). Donde 1 equivale a "sonidos totalmente diferentes" y 5 "sonidos indistinguibles". En la Tabla I se presentan los resultados del experimento para el generador sintetizando ruido de distribución uniforme, mientras que en la Tabla II se presentan los resultados para el generador sintetizando ruido de distribución normal.

TABLA I. RESULTADOS DEL EXPERIMENTO SUBJETIVO, PARA EL GENERADOR CONFIGURADO PARA SINTETIZAR RUIDO DE DISTRIBUCIÓN UNIFORME CON DIVERSOS VALORES DE M.

M=100	M=200	M=400	M=1000	M=10000	M=40000
4.52	4.61	4.92	5	5	4.98

TABLA II. RESULTADOS DEL EXPERIMENTO SUBJETIVO, PARA EL GENERADOR CONFIGURADO PARA SINTETIZAR RUIDO DE DISTRIBUCIÓN NORMAL CON DIVERSOS VALORES DE M.

M=100	M=200	M=400	M=1000	M=10000	M=40000
4.51	4.81	4.93	5	4.98	5

En los resultados que se presentan en la Tablas I y II se puede ver que los evaluadores humanos fueron incapaces de distinguir entre ruido generado utilizando la función *rand()* y ruido generado utilizando la técnica propuesta con un valor de *M* superior a *200*. De esta forma se podría concluir que el ruido generado, con un valor relativamente bajo de *M* satisface las necesidades de esta aplicación.

Además, tal como se calculó, en el dsPIC en el cual se desea implementar la técnica es posible almacenar hasta *400* segmentos de datos de *256* muestras cada uno. Si se escoge *M=200*, se podrían almacenar dos matrices: una que contenga muestras de ruido de distribución uniforme de dimensiones y otra que contenga muestras de ruido de distribución normal.

Por último, es posible además incorporar un filtro digital de tipo IIR, el cual permitiría generar ruido coloreado o ruido filtrado según las necesidades de cada individuo.

D. Implementación en un dsPIC33EP256MU806

Con el fin de validar el rendimiento del sistema, se implementó la técnica en un dsPIC33EP256MU806 de Microchip. El lenguaje utilizado es el C30 [18], el cual dispone de una gran cantidad de librerías las cuales permiten una rápida implementación de nuevas funcionalidades en el dispositivo de asistencia.

Se implementó la técnica y se midió el tiempo requerido para generar un vector de *256* muestras. Para el experimento, el sistema fue configurado con una frecuencia de muestreo (*Fs*) de *16384 Hz*. En estas condiciones, el tiempo de adquisición de un segmento es *T=15.63 ms*, por lo que el tiempo de procesamiento de cada uno de los mismos debe ser inferior a esta cota. El tiempo de procesamiento medido para la implementación fue de *3.32 ms*, verificándose que el mismo es inferior al tiempo de adquisición del segmento. Además, se midió la utilización de memoria de la implementación llevada a cabo. Un solo generador de ruido requiere un 79% de memoria de programa y un 40% de memoria de datos (RAM). Además, con el fin de independizar la adquisición de los datos

y el procesamiento de los mismos se está utilizando la totalidad de la memoria de datos disponible para acceso directo a memoria (DMA). En la Fig. 7 se aprecia un diagrama en bloques del prototipo implementado.

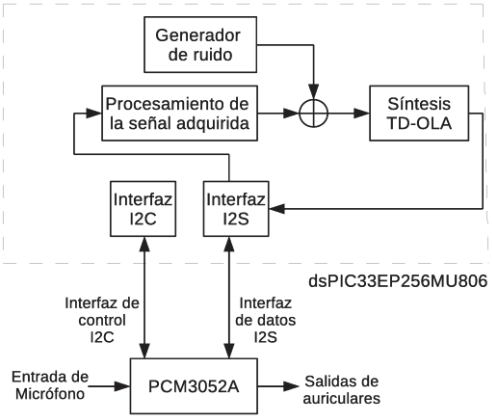

Figura 7. Diagrama de bloques del sistema implementado.

Luego, con el fin de analizar el rendimiento de la técnica desarrollada en el prototipo, se implementó un banco de experimentación, en el cual se analizó la señal sintetizada por el DSP utilizando un medidor de la función densidad de probabilidad o PDF por su traducción del inglés (*Probability Density Function*) [19]. El esquema del banco se presenta en la Fig. 8. En ella puede apreciarse que el medidor de PDF posee dos salidas: una de rampa de barrido horizontal y otra que contiene la amplitud. Es por ello que el osciloscopio para esta medición debe ser configurado en modo XY.

Para la medición, se sintetizó ruido con una amplitud de *2Vpp*. Esto fue realizado por un lado mediante el dispositivo implementado (para verificar el rendimiento de la técnica propuesta). Por otro lado, con el fin de contrastar los resultados obtenidos, se sintetizaron archivos en formato WAVE, con una frecuencia de muestreo de *16 kHz* y *16* bits de resolución (ajustes similares a los del dispositivo implementado) conteniendo señales generadas mediante las funciones *rand()* y *randn()* de MatLab®. Estas últimas señales, también fueron analizadas utilizando el banco de mediciones y se compararon los resultados obtenidos por las mismas con los de la técnica propuesta.

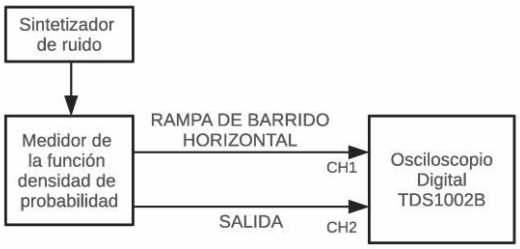

Figura 8. Banco de pruebas utilizado.

En la Fig. 9 se aprecia una captura de la señal medida con el osciloscopio para el caso del sistema operando como generador de ruido de distribución uniforme. Puede verse que la gráfica se ajusta a lo esperado de acuerdo a la Fig. 3.

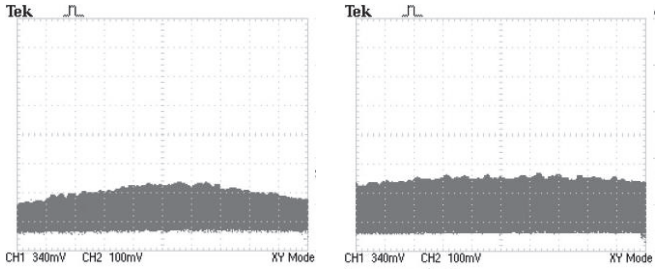

Figura 9. Capturas de pantalla del osciloscopio donde se grafica la función densidad de probabilidad (PDF) medida a la señal de salida del sistema. En la parte izquierda se presenta el caso de la señal generada utilizando la técnica propuesta para generar ruido de distribución uniforme, mientras que a la derecha se presenta la pdf medida al sintetizar ruido utilizando la función *rand()* de MatLab®.

Por último, en la Fig. 10 se presenta una captura de la pantalla del osciloscopio para el caso del sistema operando como generador de ruido de distribución normal. Nuevamente, los resultados obtenidos se ajustan a lo esperado de acuerdo a la Fig. 4.

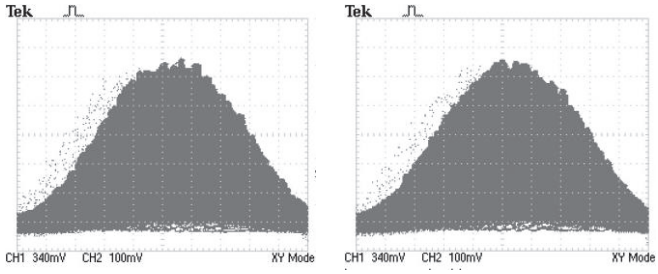

Figura 10. Capturas de pantalla del osciloscopio donde se grafica la función densidad de probabilidad (PDF) medida a la señal de salida del sistema. En la parte izquierda se presenta el caso de la señal generada utilizando la técnica propuesta para generar ruido de distribución normal, mientras que a la derecha se presenta la pdf medida al sintetizar ruido utilizando la función *randn()* de MatLab®.

En las Figs. 9 y 10 puede verse que mas allá de pequeñas discrepancias, las funciones densidad de probabilidad obtenidas por la técnica propuesta (izquierda), y las de la señal sintetizada utilizando las funciones de MatLab® (derecha) son similares y se ajustan a lo esperado. En el caso de la Fig. 9, la diferencia en los extremos de la PDF se debe al efecto de la ventana aplicada por el medidor de PDF utilizado [19].

IV. Conclusiones

En este trabajo se presentó una técnica para generación de ruido de distribución uniforme o normal a partir de información almacenada en memoria de programa de un DSP.

Se demostró que al ser aplicada en un dispositivo de asistencia auditiva, la técnica funciona de forma satisfactoria.

Además, en base a los resultados obtenidos en los test de Marsaglia, se demostró que si se almacena gran cantidad de información en la memoria de programa del DSP, el rendimiento del método es similar a la función *rand()* de MatLab®. Cabe destacar que, debido a la forma en que este generador se desarrolló, puede ser implementado utilizando recursos que suelen estar ociosos en un DSP.

Los resultados obtenidos permiten construir de forma

sencilla un generador de ruido para tratamiento de acúfenos, el cual además, si el usuario requiera otra naturaleza de ruido, el sistema podría ser ajustado mediante el agregado de un filtro IIR, consiguiendo de esta forma cumplir con las necesidades del individuo. Esta ventaja es de suma utilidad en el campo de aplicación de este desarrollo, ya que permitiría desarrollar sintetizadores de ruido enfocados al tratamiento de acúfenos en microcontroladores que incluso no dispongan de una arquitectura destinada al procesamiento de datos en punto flotante. Esto se debe a que los datos utilizados para la síntesis de la señal estarían almacenados en la memoria de programa del mismo.

En el futuro se desean continuar el desarrollo de generadores de diversos tipos de ruido, teniendo en cuenta los requisitos para una implementación en DSP.

V. Agradecimientos

Los autores agradecen a CONICET y a la Universidad Nacional de Mar del Plata por el financiamiento recibido a través de los proyectos “Secuencias caóticas digitales en procesamiento y encriptado de señales” y “Análisis de señales destinado a mejorar la calidad de vida”, respectivamente.

Referencias

[1] B.M. Oliver, J.M. Cage, “Electronic Measurement and Instrumentation” McGraw Hill, 1971.
[2] A. Papoulis, “Probability, Random Variables and Stochastic Processes”. Mc Graw Hill. 2002.
[2] A. Ghazel, E. Boutillan, I.L. Danger, G. Gulak, H. Laamari. “Design and Performane Analysis of a High Speed AWGN Communication Channel Emulator”. IEEE PACRIM Conference. pp. 374—377. 2001.
[3] J.L. Danger. A. Ghazel. E. Boutillon, H. Laamari, “Efficient FPGA Implementation of Gaussian Noise Generator for Communication Channel Emulation”. IEEE ICECS. Vol. 1. pp. 366—369. 2000.
[4] R.M Rodriguez-Osorio, A.D.C. Urbina, L.H. Ariet, M.C. Ramon, M.G Sanchez. “A DSP-based impulsive noise generator for QoS and EMC tests in wireless systems”, 60th IEEE Conference: Vehicular Technology Conference, 2004, Volume: 6, 2004.
[5] B. R. Archambeault, J. Drewniak,PCB Design for Real-World EMI Control, Springer, 2002.
[6] M. Sadasivam, J. Wagholikar, H. Sangjin, “A look-up based low-complexity parallel noise generator for particle filter processing”. In Proc. Of International Symposium on Signals, Circuits and Systems 2003 (SCS 2003). Vol. 2. pp. 621--624. 2003.
[7] J.A. Henry, T.L. Zaugg, P.J. Myers, M.A. Schechter, “Using therapeutic sound with progressive audiologic tinnitus management”, Trends on amplification, Vol. 12, No. 3, pp.188-209, 2008.
[8] A.J. Schleuning, R.M. Johnson, “Use of masking for tinnitus”, The International Tinnitus Journal, Vol.3, No.1., pp. 25-29, 1997.
[9] H. Feldmann, “Masking of tinnitus - Historical remarks”. In Proceedings III International Tinnitus Seminar, pp. 210-213, 1987.
[10] Widex Inc.http://www.widex.com/, Lynge, Denmark, 2013.
[|11] Siemens Hearing Instruments, “Introduction to tinnitus management” , http://hearing.siemens.com, 2008.
[12]G. Marsaglia, W.W. Tsang, “Some difficult-to-pass tests of randomness”, Journal Statistical Software, Vol. 7(3). 2002.
[13] Microchip Inc., dsPIC33EP256MU806 Data Sheet, High Performance 16-bit Digital Signal Controllers. http://www.microchip.com/, 2011.
[14] A.J. Uriz, P.D. Agüero, J. Castiñeira Moreira, J.C. Tulli, E.L. González, G. Moscardi, E.E. Sajama. “A development and implementation of a tinnitus treatment method”. Journal of Physics: Conference Series (JPCS). Londres: IOP SCIENCE. 2013 . ISSN: 1742-6596
[15] A.J. Uriz, P.D. Agüero, J. Castiñeira Moreira, J.C. Tulli, E.L. González y R.M Hidalgo. “Low complexity noise power estimator for speech enhancement implemented on a dsPIC”. Anales del CASE 2014. Buenos Aires, Argentina. Agosto 2014.
[16] C. Bandt, B. Pompe, “Permutation Entropy: A Natural Complexity Measure for Time Series”. Physical Review Letters. Vol.88(17). pp. 174102-1--174102-4. 2002
[17] G. Marsaglia, “The Marsaglia Random Number CDROM, with The Diehard Battery of Tests of Randomness”. Disponible en: www.stat.fsu.edu/pub/diehard. 1995.
[18] Microchip Inc. 16-Bit Lenguage Tools Libraries. http://www.microchip.com/, 2005.
[19] Manual del medidor de PDF utilizado: http://www3.fi.mdp.edu.ar/mediciones/apuntes/Manual_Pdfimetro.pdf

Alejandro José Uriz was born in Mar del Plata, Argentina in 1984. He Obtained the Degree of Electronic Engineer in Universidad Nacional de Mar del Plata (UNMdP) in 2010. Currently he is finishing his PhD degree. Nowadays, he is working in the Communications Lab - UNMdP. His current research interests include rehabilitation engineering and speech processing.

Pablo Daniel Agüero was born in Mar del Plata, Argentina in 1977. He received his degree of Electronic Engineeer at this university in 2002. In 2012 he obtained his PhD on "Speech synthesis applied to Speech-to-speech Translation" at University Politecnica de Catalunya.

Jorge Castiñeira Moreira was born in Mar del Plata, Argentina in 1962. received his degree of Electronics Engineer at UNMdP, Argentina, on March 1990. He continued his studies to receive his MSc in Digital Signal Processing Applications in Communications Systems in 1996 and his PhD in Communication Systems, at Lancaster University, Lancaster UK, on 2000. Currently he is an Full Professor in UNMdP and responsible for the teaching area Communications. He is also director of the research project "Communication and Information Theory applied to Data Networks" in the Mar del Plata University.

Roberto Marcelo Hidalgo was born in Mar del Plata, Argentina in 1959. He received his degree in 1985 from the Faculty of Engineering of UNMdP. In 1987 he joined the Signal Processing and Measurement Laboratory, UNMdP. Since then, he has been working on digital signal processing and new measurement techniques. In 2003 he obtained his PhD in “Electronics Engineering” at UNMdP.

Esteban Lucio González was born in Mar del Plata, Argentina in 1959. He received his degree of Electronic Engineer at UNMdP in 1989. In 1991 he moved to Antarctica as a Scientific leader, responsible for San Martín Antarctic Base Laboratory. In 1998 he got his Master of Science in Digital Signal Processing Applications in Communications Systems at Lancaster University, England.

Juan Carlos Tulli was born in Mar del Plata, Argentina in 1947. He received the Eng. degree in electric engineering from UNMdP in 1976. Nowadays, he is Full Professor and Director of the Communications Lab - UNMdP. His current research interest is rehabilitation engineering.