Ruido $1/f^d$ implementado en FPGA

O. G. Zabaleta*, L. De Micco*, C. M. González†, C. M. Arizmendi* y H. A. Larrondo*
*Departamento de Física, Facultad de Ingeniería, UNMDP
†Departamento de Ingeniería Electrónica, Facultad de Ingeniería, UNMDP
Juan B. Justo 4302, Mar del Plata, Argentina
Email: ldemicco@fi.mdp.edu.ar

***Resumen*—Las señales caóticas y estocásticas aparecen en la electrónica ya sea como estrategia de diseño (por ejemplo en las comunicaciones de espectro esparcido, los sistemas de mejora de compatibilidad electromagnética, el muestreo aleatorio, etc.), o bien como señales indeseables (ruido). Es importante conocer cómo se ve afectado el sistema al cambiar las características estadísticas y frecuenciales de esas señales estocásticas. Para ello se requiere contar con fuentes de ruido que las generen. La generación puede realizarse por software o por hardware. La generación en hardware ha estado restringida a sistemas caóticos pseudo estocásticos de baja dimensión. Pero el desarrollo reciente de las Field Programmable Gate Arrays (FPGAs) permite encarar implementaciones en hardware con características estadísticas y frecuenciales especificadas. En este trabajo se propone la implementación en FPGA de un generador de ruido pseudo estocástico coloreado, con espectro de tipo f^{-d}. La metodología propuesta se ejemplifica para los casos $d = 1,\ 2$. Se presentan los resultados obtenidos utilizando $QUARTUS^{©}$ II Web Edition y $DspBuilder^{©}$.**

I. Introducción

Se dice que una señal es aleatoria cuando no es predecible. Es decir con aleatoriedad nos referimos a la imposibilidad de predecir el futuro de la señal, a partir de la información sobre su pasado. El caos determinista ha demostrado que la apariencia aleatoria de una señal no es una evidencia de que el sistema que la genera tenga un espacio de estados de dimensión elevada.

Los sistemas caóticos son modelados por un número pequeño de ecuaciones no lineales y por lo tanto son sencillos de implementar [1], [2], [3]. Se los ha empleado en diversas aplicaciones tales como generadores de números pseudo aleatorios [4], [5], encriptado caótico [6], [7], [8], compatibilidad electromagnética [9], [10], filtrado de ruidos de alta frecuencia [11], entre otras.

Desde el punto de vista físico los ruidos caóticos y estocásticos son series temporales que comparten varias propiedades que los hacen casi indistinguibles: un espectro de banda ancha, una función de autocorrelación tipo delta de Dirac, un comportamiento temporal irregular, etc. Recientemente, sin embargo, se ha encontrado la forma de distinguirlos, a partir de las estructuras geométricas que sí aparecen en los ruidos caóticos y no en los estocásticos [12], [13], [14]. Es previsible entonces que muchos sistemas físicos puedan diferenciar entre ambos tipos de ruido. Para permitir un estudio experimental es necesario contar con generadores de ruido tanto caóticos como estocásticos implementados tanto en software como en hardware.

Entre los ruidos estocásticos se encuentran los conocidos como *ruidos* f^{-d} que son los estudiados en este trabajo. El nombre se debe a que su espectro de potencias es de la forma f^{-d}. Ejemplos son el ruido blanco con $d = 0$, ruido rosa $d = 1$, ruido rojo $d = 2$, entre otros.

La principal contribución de este trabajo es la implementación en hardware de un generador que produce una secuencia de salida, de ruido coloreado, a partir de una señal de entrada de ruido blanco, generada por ejemplo por un mapa caótico [1]. El dispositivo utilizado es una FPGA Cyclone II EP2C35F672C6, que forma parte de un kit de desarrollo y educación (DE2) de $ALTERA^{©}$. Esta FPGA es de bajo costo, su consumo es bajo y puede trabajar a grandes velocidades.

El trabajo está organizado del siguiente modo: en la sección II se describe la implementación en software del generador mencionado, utilizando $MATLAB^{©}$. En la sección III se describe la metodología empleada para la implementación en FPGA, se reporta la arquitectura final obtenida y se presentan los resultados en la sección IV. Finalmente las conclusiones y el trabajo futuro son expuestos en la sección V.

II. Implementación por software

El ruido de tipo f^{-d} se puede generar por software de manera relativamente simple utilizando las funciones de librería de $MATLAB^{©}$, siguiendo el procedimiento propuesto a continuación:

1. Mediante la función *rand* de $MATLAB^{©}$, o algún otro algoritmo de generación de números pseudo aleatorios, se obtiene una secuencia $y_{(n)}$ en el intervalo $(-0{,}5, 0{,}5)$, con espectro de potencia plano, función densidad de probabilidad uniforme y valor medio cero.
2. Se calcula la FFT de la secuencia $y_{(n)}$ obteniéndose el vector complejo $Y_{[k]}$.
3. Se multiplica el vector complejo $Y_{[k]}$ por la secuencia $k^{-d/2}$ obteniéndose $W_{[k]}$.
4. Luego, se simetriza $W_{[k]}$, de forma de que cumpla $W_{[-k]} = conj(W_{[k]})$ para que represente una función real del tiempo, es decir, $Re(W_{[-k]}) = Re(W_{[k]})$ y $Im(W_{[-k]}) = Im(-W_{[k]})$.
5. Finalmente se antitransforma la secuencia $W_{[k]}$ y se descartan las pequeñas componentes imaginarias que surgen del error producido por la utilización de precisión finita. Se obtiene así la serie temporal, $w_{(n)}$ cuyo espectro de potencia tiene la forma k^{-d} deseada.

Este método constructivo produce un ruido no Gaussianos. La metodología utilizada para la implementación en hardware sigue los mismos lineamientos expuestos como se describe en la sección siguiente.

III. Diseño en hardware

En esta sección se describen las herramientas utilizadas en el diseño general, se detallan las etapas más importantes del mismo, y finalmente se explican con más detalle los bloques más significativos.

El diseño se desarrolló en parte utilizando programación VHDL, cuando se necesitaron bloques sencillos, o bien para aplicaciones específicas para las cuales no existen modelos pre-diseñados. Siempre que fue posible, como es aconsejable, se utilizaron bloques de $ALTERA^{©}$ ya que éstos están optimizados para trabajar con las placas de este fabricante.

Para la programación en VHDL se utilizó $QUARTUS^{©}$ II Web Edition. Este software permite no solo compilar el diseño de forma funcional, sino que también permite realizar el análisis temporal, la distribución en la placa, etc. Si bien el diseño completo puede ser analizado y simulado con este software, cuando se trata de procesamiento de señales, no se puede extraer suficiente información de las simulaciones. Por este motivo se utilizó $Simulink^{©}$ de $MATLAB^{©}$ con el toolbox ALTERA $DSPBuilder^{©}$ Blockset versión 7,2. Este toolbox contiene todos los bloques admitidos por los distintos modelos de FPGA de $ALTERA^{©}$, e incluso se puede hacer uso de un bloque llamado HDL_import, que básicamente es una caja negra que permite importar un diseño VHDL, [15]. Además de las funciones básicas, el toolbox de ALTERA consta de funciones más complejas agrupadas en la librería ALTERA $IP - MegaCores^{©}$ [16].

La principal ventaja de programar en el entorno $MATLAB^{©}$ es poder simular el sistema utilizando todas las herramientas disponibles, tales como generadores de señales e instrumentos virtuales de medición, que permiten verificar el correcto funcionamiento de cada etapa del proyecto.

La arquitectura adoptada consta de cuatro etapas. En la Fig. 1 se presenta un diagrama en bloques básico del sistema.

La primera etapa realiza la FFT de un ruido blanco con PDF uniforme y valor medio cero. En la segunda etapa, el bloque Gf junto con los dos divisores, se encargan de "colorear" el espectro recibido, es decir, convertir el espectro del ruido blanco en un espectro de ruido de tipo f^{-d}, con $d = 1, 2, 3$. En la tercera etapa se realiza la IFFT de la señal coloreada y finalmente en la cuarta etapa, representada por el sumador, se prepara los datos para la salida. Cada bloque detalla a continuación.

Para implementar las funciones de FFT e IFFT se utilizó un bloque de la librería de ALTERA IP-$MegaCores^{©}$, fft_$v7_2$. Este bloque permite la selección en forma externa de la transformación a realizarse (FFT ó IFFT). El bloque calcula la transformada discreta de Fourier (DFT) de una señal de entrada muestreada de longitud N. Para reducir la complejidad del cálculo utiliza el algoritmo Cooley-Tukey radix-r decimación-en-frecuencia (DIF) FFT. Este algoritmo particiona recursivamente la secuencia de entrada en N/r secuencias de longitud r y requiere $log_r(N)$ etapas de cálculo. Se puede adoptar para r uno de los valores 2, 4 ó 16. Se utilizó aritmética de punto fijo, con resolución de 18 bits. Sin embargo, la función utiliza internamente aritmética de punto flotante en bloque (BFP) para realizar los cálculos. La aritmética BFP es un compromiso entre las aritméticas de punto fijo y de punto flotante. Los datos de entrada ingresan como valores en punto fijo. Internamente, los pares de valores enteros complejos se representan con un único factor de escala: luego de cada etapa de la FFT, se detecta el máximo valor de salida y los resultados intermedios son escalados para mejorar la precisión. El exponente almacena la cantidad de desplazamientos utilizados para realizar el escaleo. Las salidas $source\_real$ y $source\_imag$ combinadas con la salida $source\_exp$ entregan las partes real e imaginaria respectivamente.

El bloque Gf se encarga de generar los valores de k^{-d}, a la frecuencia de clock del sistema. Básicamente, el bloque cuenta con una memoria ROM en la cual se han almacenado los valores de $k^{d/2}$, con $k = 1, 2, 3, ..., N$ y $d = 1, 2, 3$. El bloque de memoria a leer, según el valor de d que se requiera, es seleccionado mediante la entrada d_Sel. En realidad, no es necesario guardar los N valores pues la salida de FFT contiene la misma información en forma de espejo. Por lo tanto, se almacenan los valores de $k = 1$ a $N/2$ para cada valor de d. Un contador recorre los $N/2$ valores almacenados en la memoria primero de forma ascendente y luego de forma descendente. Este bloque comienza a dar salida cuando el bloque de FFT entrega salida válida, esto es valor lógico $'1'$ en la salida $source\_valid$ del bloque FFT.

Los datos de la memoria están representados por 24 bits de los cuales los 6 primeros representan el exponente del número y los 18 restantes la mantisa. El exponente es un número constante. Para ejemplificar el por qué de esta manera de representar los datos veamos el caso $d = 1$. Si $N = 1024$, el máximo dato almacenado en memoria es $(f_{N/2})^{1/2} = 512^{1/2} \cong 22,627417$ en formato decimal. Como trabajamos en formato BPF, representamos este número decimal por medio de una mantisa y un exponente. Ambos son enteros en complemento a dos. Se debió encontrar el exponente adecuado de modo de poder distinguir entre cada uno de los valores consecutivos de la secuencia $k^{d/2}$. Al mismo tiempo, como la división es entera se requiere que el dividendo sea varios órdenes de magnitud mayor que el divisor, es decir, que los valores de mantisa entregados por el bloque FFT sean mayores que los entregados por el bloque Gf, es por eso que tampoco podemos afectar al número por un exponente demasiado grande. Para el caso del ejemplo, $d = 1$, el exponente que mejor se ajustó a estos requisitos fue el valor 5.

Entonces, el procedimiento que se siguió para calcular los datos que luego se almacenaron en memoria consta de los siguientes pasos:

1. Multiplicar el número decimal por 2^5,
2. convertir el número a formato binario,

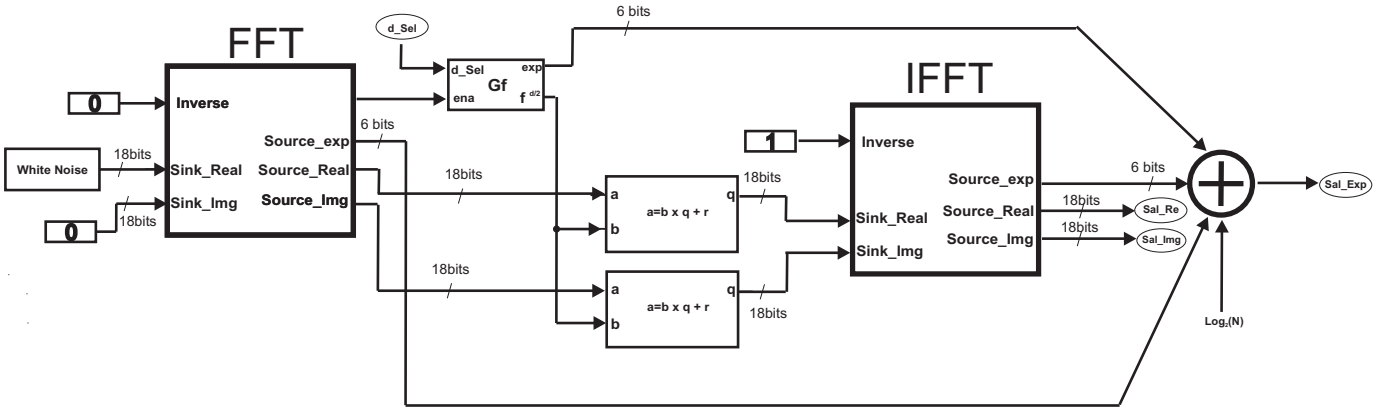

Figura 1. Diagrama en bloques

3. mantener solo la parte entera.

El número que se almacena en memoria para el ejemplo es 00010100000000101101010O.

Como se observa en el diagrama, existe un bloque divisor para la parte real y otro para la parte imaginaria del espectro entregado por el bloque FFT. El bloque divisor desarrollado es básico y ocupa escasos recursos. Los datos de entrada son un divisor y un dividendo de tipo entero de 18 bits con signo, al igual que los datos de salida. Los resultados de ambos bloques ingresan al bloque IFFT a las entradas $sink\_real$ y $sink\_img$ respectivamente. Este bloque entrega la mantisa de las parte real e imaginaria de la señal de salida, por medio de $source\_real$ y $source\_img$ respectivamente. Entrega además una tercera señal, $source\_exp$, que representa el exponente de la señal transformada.

Se deben tener en cuenta los tres exponentes antes mencionados y un factor de escala $1/N$ correspondiente a la transformada de Fourier, ya que la FFT no lo aplica internamente. Por lo tanto el, exponente total al final del proceso está representado por la Eq. (1) en tanto que la secuencia de salida estará dada por (2).

$$exp = expGf + expFFT + expIFFT + log_2(N) \quad (1)$$

$$Sal = (source\_real + i \cdot source\_imag \cdot) 2^{-exp} \quad (2)$$

La parte imaginaria debería ser cero, sin embargo, aparece un pequeño valor como consecuencia de trabajar con aritmética finita, por lo tanto se la descarta.

IV. Resultados

En la Fig. 2 se muestran los espectros de los ruidos $1/f$ y $1/f^2$ generados mediante la arquitectura descripta.

El valor de m indicado en estas figuras corresponde a la pendiente estimada de los puntos ajustados a una recta mediante la función *polyfit* de $MATLAB$©. También se indica el período de muestreo ,T_{ck}, y $k = d$ representa el tipo de ruido elegido $(1/f^d)$. El período de muestreo es directamente proporcional al período de reloj utilizado. Puede verse que el espectro obtenido presenta la forma de ruido coloreado esperado en cada caso.

V. Conclusión

Se realizó un generador de ruido coloreado de tipo $1/f^d$ optimizado para ser implementado en una FPGA de bajo costo, y que permite una alta velocidad de reloj. Se utilizaron las herramientas de diseño de hardware, así como también las herramientas de simulación y la interacción entre ambas.

La principal conclusión en esta etapa de diseño es la ventaja de la utilización del entorno $Simulink$ de $MATLAB$© para el diseño y simulación, dado que permite en forma relativamente simple verificar el correcto funcionamiento del sistema. Por otra parte, mediante la misma metodología es posible la implementación de otros tipos de ruido estocástico.

VI. Agradecimientos

Este trabajo ha sido parcialmente financiado por CONICET (PIP2004), ANPCyT (PICT 04) y UNMDP.

Referencias

[1] C.M.González, H.A.Larrondo, C.A.Gayoso, and L.J.Arnone. Secuencias binarias generadas por un mapa caótico 2d. *Proceedings del X Iberchip*, 2004.

[2] C.M. González, H.A. Larrondo, C.A. Gayoso, and L.J. Arnone. Implementación de sistemas caóticos en dispositivos lógicos programables. *XI Workshop IBERCHIP*, 2005.

[3] C. M. González, H. A. Larrondo, C. A Gayoso, and L. J. Arnone. Generación de secuencias binarias pseudo aleatorias por medio de un mapa caótico 3d. In *Proceedings del IX Workshop de IBERCHIP*, 2003.

[4] H. A. Larrondo, C. M. González, M. T. Martin, A. Plastino, and O. A. Rosso. Intensive statistical complexity measure of pseudorandom number generators. *Physica A*, 356:133–138, 2005.

[5] H. A. Larrondo, M. T. Martin, C.M. González, A. Plastino, and O. A. Rosso. Random number generators and causality. *Phys. Lett. A*, 352((4-5)):421–425, Abril 2006.

[6] M. Pecora, L. Carroll, and L. Thomas. Synchronization in chaotic systems. *Phys. Rev. Lett.*, 64(8):821–824, Febrero 1990.

[7] R. M. Hidalgo, J. G. Fernández, R. R. Rivera, and H. A. Larrondo. Versatile dsp-based chaotic communication system. *Electronic Letters*, 37:1204–1205, 2001.

[8] J.G. Fernández, H. A. Larrondo, H. A. Slavin, D. G. Levin, R. M. Hidalgo, and R. R. Rivera. Masking properties of apd communication systems. *Physica A*, 354:281–300, 2005.

[9] S. Callegari, R. Rovatti, and G. Setti. Spectral properties of chaos-based fm signals: theory and simulation results. *IEEE Trans. Circuits Sys. I*, 50(1):3–15, 2003.

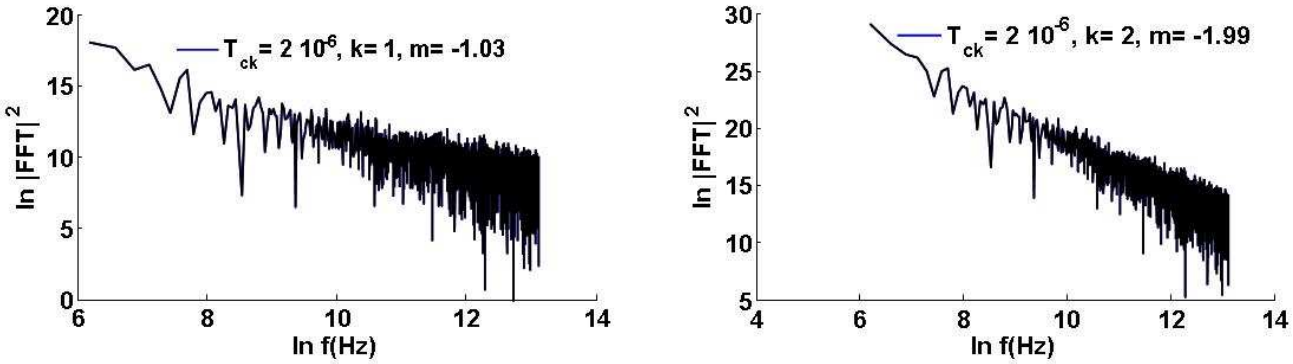

Figura 2. Espectro de potencias del ruido generado por el circuito propuesto: *(a)* $d = 1$, *(b)* $d = 2$

[10] L. De Micco, R. A. Petrocelli, and H. A. Larrondo. Constant envelope wideband signals using arbitrary chaotic maps. *Proceedings de la XII Reunión de Trabajo en Procesamiento de la Información y Control*, 2007.

[11] R. A. Petrocelli, L. De Micco, D. O. Carrica, and H. A. Larrondo. Acquisition of low frequency signals immersed in noise by chaotic sampling and fir filters. *Proceedings of WISP2007 (IEEE proceedings ISBN 1-4244-0830-X)*, pages 351–356, 2007.

[12] O. A. Rosso, H. A Larrondo, M. T. Martin, A. Plastino, and M. A. Fuentes. Distinguishing noise from chaos. *Physs. Rev. Lett., pp154102-154106*, 99, 2007.

[13] L. De Micco, H. A. Larrondo, A. Plastino, and O. A. Rosso. Quantifiers for randomness of chaotic pseudo random number generators. *arxiv.org/0812.2250v1*, preprint, 2008.

[14] O. A. Rosso, L. De Micco, H. A. Larrondo, M. T. Martin, and A. Plastino. Generalized statistical complexity measure: a new tool for dynamical systems. *submitted to International Journal of Bifurcation and Chaos*, preprint, 2008.

[15] Altera. *Black-Boxing in DSP Builder*.

[16] ALTERA. ip-basesuite.html. *http://www.altera.com/products/ip/design/basesuite/ip-basesuite.html.*, 2008.