Generador de Números Pseudoaleatorios Mediante el Sistema Numérico de Residuos, Estudio Estadístico

Carlos Arturo Gayoso, Claudio Marcelo González, Leonardo Arnone y Miguel Rabini
Laboratorio de Componentes Electrónicos, Departamento de Electrónica
Universidad Nacional de Mar del Plata
Mar del Plata, República Argentina
cgayoso@fi.mdp.edu.ar

Resumen — **Este trabajo estudia la implementación en hardware de nuevos generadores de números pseudoaleatorios (*Pseudo Random Number Generators,* PRNGs o Generadores de Números Pseudoaleatorios, GNPA), en lógica programable (*Field Programmable Gate Arrays* o FPGA). Se investiga el empleo del sistema numérico de residuos (*Residue Number System* o RNS) para incrementar la velocidad a la que los generadores producen los números aleatorios y para que posea una dinámica distinta a los generadores conocidos. El circuito propuesto ya se evaluó mediante tests básicos y el conjunto de tests desarrollados por George Marsaglia para su generador Diehard [1]. El trabajo está organizado de la siguiente manera: comienza con la definición de sistemas determinísticos y aleatorios junto con una introducción a la denominada complejidad estadística y dos de las métricas propuestas, luego se describe el generador de números pseudoaleatorios propuesto junto la explicación de cada uno de los bloques que lo constituyen y finalmente se presentan los aportes y conclusiones del trabajo realizado.**

Sistema Numérico de Residuos; Aritmética de residuos; Números pseudoaleatorios; Lógica Programable; Complejidad estadística.

I. INTRODUCCIÓN

A. Determinismo y aleatoriedad

Desde el punto de vista de su comportamiento los sistemas se pueden clasificar como deterministas, aleatorios o caóticos. En los primeros se puede precisar cualquiera de sus futuros estados conociendo su estado presente, no hay participación del azar en ninguna de sus variables ni en las relaciones entre ellas. Por el contrario, en los sistemas aleatorios el azar es el componente esencial [2], [3]. Tal es así, que no se puede determinar la evolución del mismo, ni siquiera su próximo estado, conociendo con una precisión ilimitada su salida actual y las anteriores, no se puede encontrar ningún tipo de patrón o regularidad en su comportamiento. Los sistemas caóticos son un tipo intermedio entre los deterministas y los aleatorios, pues si bien están descriptos por ecuaciones bien conocidas, no se puede determinar con precisión sus estados futuros, puesto que su evolución se desarrolla en órbitas acotadas que no se superponen y con trayectorias muy sensibles a las condiciones iniciales y perturbaciones.

Existen diversos circuitos electrónicos o algoritmos que tienen la particularidad de generar secuencias de números aleatorios o caóticos. Pero, en este caso, a diferencia de procesos naturales, las series generadas tienen un período, que si bien puede ser muy grande, es finito. Por esta razón a estos sistemas se los denomina pseudoaleatorios o pseudocaóticos.

B. Tests de aleatoriedad

Los tests de aleatoriedad se emplean para analizar una secuencia de números, provenientes de una fuente natural o artificial, para determinar si se trata de un proceso estocástico [2], [3]. Si bien no se puede afirmar de manera concluyente que una serie de números es aleatoria existe una serie de tests, que de dar resultado satisfactorio, dan sustento a la hipótesis de aleatoriedad. De manera que para determinar el comportamiento del circuito electrónico o algoritmo se somete a las series numéricas entregadas por éstos a una serie de ensayos.

Un estudio preliminar de la secuencia de datos, por ejemplo uniformidad, o autocorrelación, puede poner en evidencia un patrón de comportamiento fácilmente predecible que descarte un comportamiento aleatorio. Sin embargo, en otros casos, con las herramientas estadísticas tradicionales no se puede detectar una estructura determinista. Es por ello que se han ideado distintos tipos de tests a los que es sometida la secuencia numérica bajo estudio en busca de concluir sobre su carácter determinístico o aleatorio.

En [1] se sometió a los GNPAs propuestos a uno de los ensayos más utilizados para medir la calidad de una secuencia de números supuestos aleatorios el denominado test Diehard [4], [5]. En realidad es un conjunto de 17 tests desarrollados por George Marsaglia de manera progresiva desde 1985 y publicados por primera vez juntos en 1995. Los generadores propuestos superaron esta batería de tests.

Otra forma de medir la calidad aleatoria de la salida numérica de un circuito electrónico es mediante la denominada complejidad estadística. Existen numerosos intentos o aproximaciones para medir esta característica, fácil de definir pero difícil de cuantificar. En este trabajo se toman dos de esos cuantificadores y se aplican a los GNPAs propuestos. También se les calcula la entropía o cantidad de información.

En todos los casos se comparan los resultados con secuencias de números entregadas por generadores bien conocidos

C. *El Sistema Numérico de Residuos*

Los circuitos aritméticos, por ejemplo sumadores, basados en la notación de complemento a 2, deben propagar la información de acarreo desde el bit menos significativo al más significativo, de manera que a medida que el número de éstos aumenta el rendimiento se degrada. El RNS [6], [7], [8] es una técnica eficiente para superar este problema, dado que se trabaja sobre canales independientes sin necesidad de intercambio de información entre ellos. De esta manera los sistemas basados en el RNS están compuestos de una serie de canales pero, cada uno de ellos, con un número reducido de bits. Los circuitos aritméticos de b bits se pueden transformar, mediante el empleo del RNS, en $O(b/\log_2(b))$ canales de $\lceil \log_2(b) \rceil$ bits cada uno [9], Fig. 1. Esta característica lo hace apropiado para la realización de un número importante de aplicaciones en procesamiento digital de señales [10], [11], [12].

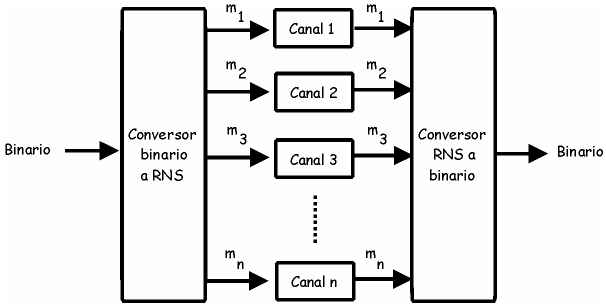

Figura 1. Esquema general de un circuito RNS.

Un sistema basado en RNS se define mediante un conjunto de enteros relativamente primos entre si $\{m_1, m_2..., m_L\}$, llamados módulos. Su rango dinámico, el número de cantidades distintas que se puede representar, es $M = \prod m_i$ (i=1, 2, …, L), y de manera que cualquier entero $0 \le X < M$ se representa mediante un conjunto de L residuos $[x_1, x_2, ..., x_L]$, con $x_i = X \bmod m_i$ (i=1, 2, …, L). La principal ventaja del RNS radica en su capacidad para realizar sumas, restas y multiplicaciones a alta velocidad, debido a que la aritmética de residuos se define sobre un anillo de enteros módulo M tal que:

$$Z = (X \Diamond Y) \bmod M \quad \leftrightarrow \quad z_i = (x_i \Diamond y_i) \bmod m_i \quad (i = 1,...,L) \tag{1}$$

donde ◊ significa suma, resta o multiplicación en módulo. En la Ec. (1) se puede apreciar el potencial del RNS, puesto que las operaciones en módulo M se computan en paralelo sobre L canales independientes (Fig. 1). Debido a que no se presentan dependencias de acarreo o de datos entre los canales, el rendimiento total del sistema estará dado por la velocidad de procesamiento o *throughput* de cada canal en módulo m_i. Aunque operaciones tales como división o comparación son muy difíciles de realizar, esto no limita la aplicación del RNS, de hecho el procesamiento digital de señales se ha transformado en su campo de aplicación preferido. De esta manera los algoritmos de multiplicación y suma, muy comunes en DSP (procesamiento digital de señales), pueden incrementar su velocidad de funcionamiento mediante su implementación en RNS, como se ha demostrado en aplicaciones tales como transformadas discretas, filtrado digital o procesamiento de imágenes [13], [14].

Por otro lado los fabricantes de dispositivos lógicos programables (*Field Programmable Logic*) proveen circuitos integrados programables en campo para casi todas las aplicaciones de la electrónica digital. Para la implementación de circuitos aritméticos en el RNS las FPGAs [15], [16] se han convertido en una seria alternativa al empleo de circuitos implementados con aritmética convencional [17]. Por esta razón el circuito propuesto se ha implementado sobre una FPGA FLEX10K20RC240-4 [18] de Altera Corporation.

II. GENERADOR PSEUDOALEATORIO PROPUESTO (RNS-LCG)

El generador propuesto, denominado RNS-LCG (*Residue Number System-Linear Congruential Generator*) se basa en el *Linear Congruential Generator* (LCG). Sin embargo su dinámica es distinta. En efecto, no se trata de realizar un LCG mediante RNS sino en usar el RNS para obtener un generador de números pseudoaleatorios con una dináminca completamente distinta. Un LCG común responde a la siguiente ecuación:

$$x_i = (a\, x_{i-1} + b) \text{mód}\, M \tag{2}$$

, con a y b constantes y M el módulo que determina el rango dinámico del sistema.

La implementación circuital para los generadores RNS-LCG Tipo I y II para n canales, cada uno de los cuales trabaja con h_j bits, se muestra en Fig. 2. Cada canal es un pequeño generador lineal congruente que realiza el cálculo de la Ec. (3). En ésta g_j y m_j son la raíz primitiva y el módulo de trabajo de cada uno de los canales y $\text{residuo}_{j,i}$ es el residuo del canal j para la iteración i.

$$residuo_{j,i} = (g_j\, residuo_{j,i-2} + b_{j,i}) \text{mód}\, m_j \tag{3}$$

Para introducir un mayor desorden los canales se perturban entre sí, de manera que el valor b de la Ec. (2) ya no es una constante. En los generadores Tipo I (RNS-LCG-I), con n igual al número de módulos, para el canal j en la iteración i se tiene:

$$b_{j,i} = \sum_{\substack{k=0 \\ k \neq j}}^{k=n-1} residuo_{k,i-1} \tag{4}$$

, en tanto que para los Tipo II (RNS-LCG-II), siendo Θ la operación or exclusiva bit a bit, será:

$$b_{j,i} = \bigoplus_{\substack{k=0 \\ k \neq j}}^{k=n-1} residuos_{k,i-1} \tag{5}$$

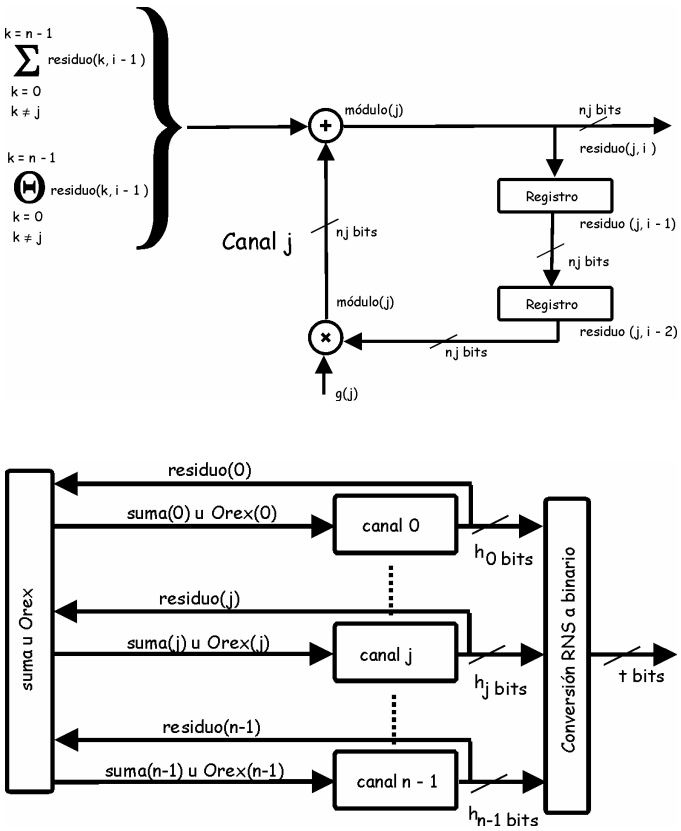

Figura 2 Generadores RNS-LCG-I y RNS-LCG-II. Diagrama en bloques para un canal genérico j, arriba. Esquemático total, abajo.

Ahora bien, si se desea generar números de t bits se presenta la siguiente dificultad. Para trabajar en el RNS se debe elegir un conjunto $\boldsymbol{m}$ de n módulos relativamente primos con lo que se obtiene un rango dinámico M igual al producto de los módulos, con $2^t < M < 2^{t+1}$. Es decir que M no es una potencia exacta de 2. Por lo tanto para trabajar con t bits se selecciona un conjunto $\boldsymbol{m}$ que siempre excederá a 2^t, lo que trae aparejado el siguiente inconveniente. Aún cuando los números leídos en decimal sean equiprobables, los 0s y 1s en cada posición no lo serán. Este problema se ejemplifica en la TABLA I, para tener $M = 11$ se necesita $t = 4$. Como se puede observar se tiene:

$$Para\ la\ posición \begin{cases} b_0\ 6\ ceros\ y\ 5\ unos \\ b_1\ 6\ ceros\ y\ 5\ unos \\ b_2\ 7\ ceros\ y\ 4\ unos \\ b_3\ 8\ ceros\ y\ 3\ unos \end{cases}$$

Los 0s y 1s no son equiprobables en cada posición, algo indeseado en un buen GNPAs. Esto ocurre a pesar de que los números del 0 al 10 tengan una distribución uniforme perfecta.

Para salvar esta situación se implementaron tres estrategias, denominadas A, B y C.

A. Estrategia A

Se toma un conjunto $\boldsymbol{m}$ tal que cumpla con $2^t < M < 2^{t+1}$. Para el caso, $t = 32$, se eligió $\boldsymbol{m} = \{3, 11, 17, 19, 23, 29, 31, 37\}$ con lo que se obtiene $M = 8.154.657.291 > 2^{32} = 4.294.967.296$, es decir se puede representar el rango deseado. Si el GNPA bajo estudio es bueno, cosa que se demostrará mediante los tests posteriores, se pueden tomar sólo aquellos valores que sean menores que 2^{32} y descartar el resto. Por ejemplo, si el GNPA tiene distribución uniforme entre 0 y M – 1, la serie obtenida tendrá distribución uniforme entre 0 y 2^{32} – 1. Trabajar de esta manera trae dos consecuencias, una intuitiva y otra práctica. La intuitiva dice que si existiera alguna estructura en la secuencia generada, al quitar algunos de sus elementos, en este caso un número importante, casi uno de cada dos, en la nueva serie esa estructura se desvanecerá o atenuará. En el caso práctico se tiene el problema de que no se entregará, debido al descarte, un número en cada iteración. Esto puede subsanarse mediante el agregado de hardware, por ejemplo una memoria FIFO, para la cual habrá de realizarse un estudio a fin de determinar su capacidad y con el GNPA funcionando a una frecuencia mayor que el circuito que los procesa, por ejemplo el que encripta.

TABLA I. Ejemplo para $M = 11$ y $t = 4$.

	b_3	b_2	b_1	b_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

B. Estrategia B

En este caso se trata de seleccionar un conjunto $\boldsymbol{m}$ que cumpla con $2^t < M < 2^{t+1}$, pero de manera tal que la diferencia $M - 2^{32}$ sea mínima, y puesto que el número de bits es 33 simplemente se descarta el más significativo. En el caso ejemplificado en la TABLA I se obtendrán números comprendidos entre 0 y 10 en la que al descartar el bit de mayor peso se reducirá a una cadena de dígitos entre 0 y 7. Las consecuencias son las siguientes. Se mejora el *throughput*, puesto que en cada iteración se obtiene un dato. Se empeora su función distribución puesto que la combinación "000" es más probable que la "111", dado que la primera puede ocurrir si el número presentado por el generador fue "0000" o "1000" en tanto que la segunda de dará sólo cuando su salida sea "0111". Cuanto menor sea la diferencia entre M y $2t$ menor será la "perturbación" en la función distribución, supuesta uniforme. Como ventaja tiene la característica de su sencillez, puesto que

ni siquiera es necesario un comparador, en efecto, de las tres clases de circuitos desarrollados es el más sencillo en su hardware. Para identificar los conjuntos de módulos que mejor se adaptan a esta estrategia se realizó un estudio exhaustivo con módulos primos de hasta 7 bits es decir con el conjunto {3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127}. Se buscaron aquellos conjuntos de módulos que superaran a 2^{32} en no más de un 0,01%. Es decir combinaciones de los 30 módulos tomados de a 4, 5, 6, 7 y 8. Con 4 módulos no se puede llegar a 2^{32}, en los restantes casos los mejores resultados fueron los que se ilustran en la TABLA II.

Como se puede apreciar si se toma el primer conjunto, $\boldsymbol{m}$ = {3, 43, 47, 67, 97, 109}, que es el grupo con el cual se trabajó, la función distribución de probabilidad uniforme se verá "alterada", habrá números más probable que otros, pero serán sólo el 0,000171% del total. De manera que se puede afirmar que para la mayoría de los fines prácticos los 0s y 1s serán prácticamente equiprobables en cada posición y que la función densidad de probabilidad es uniforme.

TABLA II. Conjuntos de módulos que más se aproximan a 2^{32} en exceso.

Exceso de 2^{32} en porcentaje	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0,000171%	3	43	47	67	97	109		
0,000329%	5	11	13	17	53	59	113	
0,001602%	3	7	13	37	47	83	109	
0,001765%	5	11	17	23	29	71	97	
0,001930%	11	23	53	59	61	89		
0,002493%	5	11	13	19	61	71	73	
0,002979%	3	11	13	23	67	73	89	
0,003663%	7	31	41	43	103	109		
0,003954%	3	7	13	37	53	71	113	
0,004044%	13	17	37	61	79	109		
0,004114%	13	19	53	59	67	83		
0,005759%	3	7	13	19	71	107	109	
0,005835%	3	7	11	13	29	31	37	43
0,006079%	3	5	7	53	73	97	109	
0,006208%	3	17	23	31	41	43	67	
0,006232%	11	17	31	79	83	113		
0,006623%	5	37	47	67	73	101		
0,006680%	7	17	37	89	97	113		
0,007078%	3	7	19	31	67	71	73	
0,008061%	13	23	47	53	73	79		
0,008383%	17	19	41	47	67	103		
0,009021%	3	19	59	89	113	127		
0,009294%	11	23	43	67	71	83		
0,009370%	3	5	7	13	23	41	47	71
0,009494%	3	7	19	29	43	89	97	
0,009509%	5	13	19	23	37	61	67	

C. Estrategia C

Este enfoque elimina las limitaciones que tienen las estrategias A y B. Puesto que entrega un número en cada iteración y la distribución es uniforme en toda su extensión.

Como en el caso B se toma un conjunto $\boldsymbol{m}$ tal que M sea mayor que 2^{32} pero cercano a él. Es decir que el generador entregará números que requieren para su representación en binario 33 bits. Los números entregados por el GNPA ingresan al bloque que se ilustra en Fig. 3, denominado circuito de corrección. Como se puede apreciar, si el bit más significativo entregado por el conversor RNS a binario es cero ($a_{32} = 0$), el número entregado por el GNPA sigue sin cambios su camino hasta la salida, Registrod. El Registroe periódicamente actualiza su valor de la siguiente manera. Si un determinado número de los bits menos significativos, por ejemplo 5, del Registroc, son iguales a un valor predeterminado, elegido de manera arbitraria, se almacena un nuevo valor. Esto significa que estadísticamente su valor se modifica, en caso de tomar los 5 bits menos significativos de Registroc, con un frecuencia de 1 cada 32 iteraciones y el valor que se almacena es el del Registrod. Se realimenta y almacena el contenido de Registrod y no en el Registroc, de lo contrario los últimos bits de Registroe serian siempre los mismos. La idea es que cuando el conversor RNS a binario entregue un número tal que $a_{32} = 1$ los bits $a_{31}\ldots\ a_0$ sean reemplazados por alguna operación entre éstos y el valor de Registroe, que se almacenó en algún momento en el pasado. Estadísticamente tanto más alejados del valor actual cuanto mayor sea el número de bits tomados de Registroc para cargar al Registroe. La operación entre Registroe y $a_{31}\ldots\ a_0$ debe ser una relación sencilla en la que, además, para cada bit del resultado los 0s y 1s sean equiprobables. El elemento que cumple con esta condición es la or exclusiva realizada bit a bit.

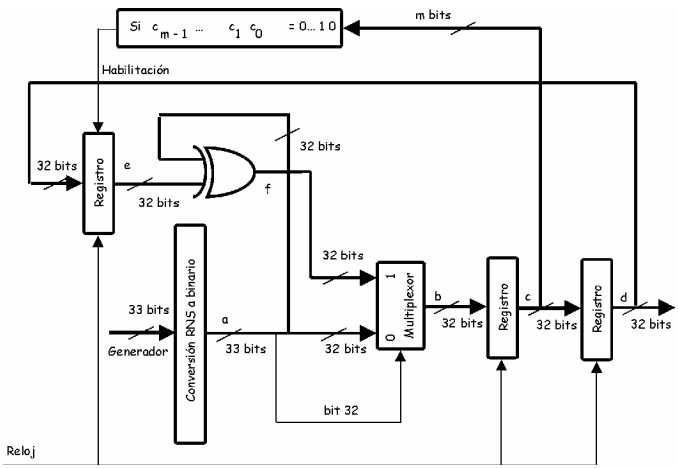

Figura 3 Esquema de corrección para los circuitos Clase C.

Con este circuito de corrección se logran dos cosas. Primero, en cada iteración se obtiene un número de la serie pseudoaleatoria. Segundo, no se altera la distribución uniforme, puesto que no existen valores privilegiados, como en la estrategia denominada B, dado que para cada bit f_{31} a f_0 los 0s y 1s son equiprobables.

III. PRUEBA DE LOS GENERADORES PROPUESTOS

En [1] se mostró el desempeño de los generadores propuestos bajo el conjunto de tests Diehard, demostrando que los circuitos presentados superaban este test de aleatoriedad. Ahora bien, el que las series de números entregadas por un circuito electrónico superen un test de aleatoriedad no significa que quede demostrado que la serie es estocástica. Lo que hace este procedimiento es agregar un elemento de plausibilidad o credibilidad a la hipótesis de que el circuito tiene un comportamiento aleatorio. Entonces, si bien no se puede demostrar que las series entregadas son aleatorias, cuanto más tests superen éstas, más sustento tendrá la hipótesis de que los circuitos propuestos entregan secuencias verdaderamente estocásticas. Por ello en el presente apartado se presentan los resultados obtenidos de someter a los GNPAs a otros ensayos estadísticos.

Para el testeo de los generadores propuestos se tomó $\boldsymbol{m}$ = {3, 11, 17, 19, 23, 29, 31, 37} para los algoritmos A, en tanto que para los B y C $\boldsymbol{m}$ = {3, 43, 47, 67, 97, 109}. Los coeficientes que operan sobre los valores anteriores en cada tipo de generador son las raíces primitivas $\boldsymbol{g}$ = {2, 2, 3, 2, 5, 2, 3, 2} en el primer caso y $\boldsymbol{g}$ = {2, 3, 5, 2, 5, 6} en el segundo y tercero.

A. Medida de complejidad mediante programas compactadores (zippers)

En [19] se propone el uso de algoritmos empleados para la compactación de archivos como una herramienta para medir la complejidad de un sistema. El teorema de Shannon-McMillan determina que para una dada cadena[1] existe un límite máximo al que se la puede comprimir sin pérdida de información. Este límite es proporcional a la entropía, en el contexto de la teoría de la información, que ella contiene. De manera que una buena medida de cuantificar la cantidad de información es ver cuanto se puede comprimir una cadena sin pérdida de información.

Según este razonamiento parece natural tomar los resultados que entregan los compresores de archivos más comunes como una herramienta de medir su entropía. Se dice que un algoritmo de compresión es óptimo si es capaz de alcanzar el límite enunciado por Shannon-McMillan.

Uno de los algoritmos más empleados por los compactadores comerciales es el de Lempel-Ziv, que es capaz de acercarse asintóticamente al límite impuesto por el teorema de Shannon-McMillan. Lempel y Ziv definieron en 1976 como medida de complejidad de una serie de bits el número de patrones diferentes que se pueden encontrar en la secuencia. El programa WinZip 9.0[2] utiliza una versión mejorada del algoritmo anterior denominado Lempel-Ziv-Welch. La complejidad zipping se define entonces como:

$$C_{zipping} = \frac{N^o\,de\ bits\ del\ archivo\ compactado}{N^o\,de\ bits\ del\ archivo\ sin\ compactar} \qquad (6)$$

Para medir la complejidad de zipping se uso la versión 9.0 de WinZip en el modo de compresión máxima y se analizaron 60 series entregadas por los GNPAs propuestos. En los generadores Clase A, como se explicó, no se obtienen archivos de la misma longitud debido al descarte, en tanto que, en los Clase B y C todos tienen el mismo número de bytes.

En la TABLA III se muestra el resultado del cálculo de la $Complejidad_{zipping}$ para los generadores Clase A. Se puede apreciar que en todos los casos es ligeramente mayor a uno. Esto se debe a que la relación de compresión obtenida es del 0%, es decir incompresible por este algoritmo, pero además al archivo comprimido se le agregan algunos bytes de información adicionales, como por ejemplo el nombre del archivo contenido.

TABLA III. $Complejidad_{zipping}$ de 20 series para los generadores RNS-LCG-I y RNS-LCG-II Clase A.

Simulación	RNS-LCG-I	RNS-LCG-II
	$\mathbf{Complejidad_{zipping}}$	$\mathbf{Complejidad_{zipping}}$
0	1,0001622	1,00016214
1	1,00016211	1,00016219
2	1,00016229	1,00016217
3	1,00016224	1,00016227
4	1,00016214	1,00016203
5	1,0001622	1,00016212
6	1,0001622	1,00016211
7	1,00016221	1,0001622
8	1,00016215	1,00016233
9	1,00016215	1,00016216

Para los generadores Clase B la longitud de todos los archivos sin comprimir es de 12.000.000 de bytes. En todos los casos la relación de compresión es del 0% y el archivo compactado ocupa 12.003.785 bytes. Con lo que se obtiene como resultado: $Complejidad_{zipping}$ = 1,000315417. Los archivos son incompresibles mediante este procedimiento.

Finalmente los generadores Clase C tienen todos una longitud de 11.999.992 bytes sin comprimir. Para cada serie la relación de compresión es del 0% y el archivo compactado ocupa 12.001.947 bytes. Nuevamente los algoritmos que emplea el programa WinZip no logran comprimir a los archivos presentados. El resultado es: $Complejidad_{zipping}$ = 1,000162917.

B. Exponente de Hurst

H. E. Hurst [20] [21] trabajó en el proyecto de la represa del Río Nilo a principios del siglo XX. Su problema era determinar la capacidad del reservorio a construir y que

[1] En este contexto una cadena puede ser, una secuencia de DNA, texto, bits en comunicación digital de datos, bit en almacenamiento magnético, etc.

[2] Copyright 1991-2004 WinZip Computing Inc.

volumen de agua se puede dejar salir por año, sin que este se vacíe ni sea superado en su nivel máximo. Hasta entonces, y dada la cantidad de variables involucradas en el sistema del Río Nilo se suponía que el nivel del mismo se podía modelizar mediante la estrategia del camino aleatorio.

Einstein había mostrado, que en el fenómeno conocido como movimiento Browniano, la distancia media que recorre una partícula se incrementa con la raíz cuadrada del tiempo transcurrido entre una observación y otra. La expresión matemática es:

$$R = T^{0,5} \tag{7}$$

, donde R es distancia recorrida media y T el intervalo de tiempo entre observaciones.

Hurst descubrió que la Ec. (7), que se aplica sólo a series con valor medio cero y varianza igual a uno, se puede generalizar a:

$$\left(\frac{R}{S}\right)_n = c.n^H \tag{8}$$

, donde R es el rango, S desviación estándar, c es un constante, *n* el número de puntos de la serie y H el hoy denominado exponente de Hurst. Este análisis también se conoce como análisis de rango reescalado, debido a que los datos se procesan para tener valor medio cero y se normaliza con respecto a su desviación estándar. Se divide la serie ***x*** de *n* puntos en *b* subseries con *n* / *b* valores cada una. A cada subserie se le realiza el proceso de reescalado, calculando el rango (R) dividido la desviación estándar. Se varía *n* / *b* desde 4 a *n* / 2 y se levanta una curva como la de la Fig. 4. A la curva obtenida se la aproxima por mínimos cuadrados, la pendiente de esta recta es el exponente de Hurst. El resultado del estudio puede dar:

$$\begin{cases} a) & H = 0{,}50 \\ b) & 0{,}50 < H \le 1{,}00 \\ c) & 0 \le H < 0{,}50 \end{cases}$$

Si H = 0,50 se trata de un proceso independiente. Ninguno de sus valores está relacionado con los anteriores. Es un proceso aleatorio, independientemente del tipo de función distribución de que se trate. Si el caso es el b, la serie es persistente y presenta efectos de memoria a largo plazo, lo que ocurre hoy afecta al futuro, para siempre, aunque con un efecto menor a medida que nos alejamos de ese valor en particular. Si en un determinado momento la serie presenta valores consecutivos ascendentes es muy probable que el próximo también lo sea. Si por el contrario los valores son descendentes es muy posible que el consecutivo siga esta tendencia. Por último, si H cae en el caso c la serie es antipersistente, el sistema cubre menos recorrido que en el camino aleatorio, su pendiente se invierte más seguido que en el movimiento Browniano. La fuerza con que un fenómeno es persistente o antipersistente depende del valor de H.

En el caso que nos interesa, testear si una serie es aleatoria, debe ser H = 0,50, o de manera más rigurosa, debido a los errores propios de todo procesamiento numérico y dado que se trata de un estudio estadístico, se debe cumplir que H ≈ 0,50 para ser considerada aleatoria.

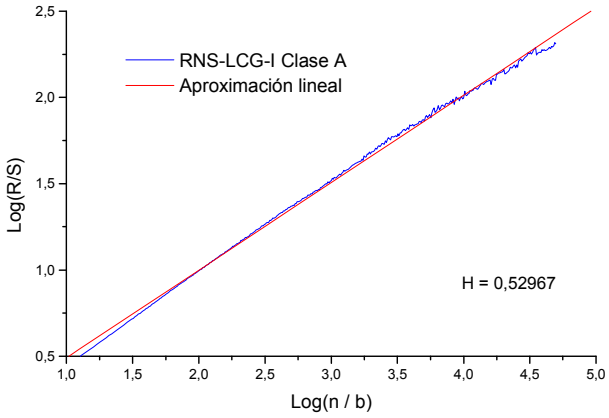

Figura 4 Gráfica para el cálculo del exponente de Hurst para el generador RNS-LCG-I Clase A.

A fin de poder comparar los generadores propuestos con otros ya testeados y conocidos se generaron las siguientes series testigos de números enteros de 32 bits:

♦ Uniforme, computada por el MatLab 7.0[3].
♦ MWCG, Multiply With Carry Generator, con el programa Makewhat del paquete Diehard.
♦ MTHR4, "the mother of all random number generators" del paquete Diehard.
♦ SWBMWC, es una combinación de los generadores Subtract With Borrow y Multiply With Carry, del paquete Diehard.

Para el cálculo del exponente de Hurst se trabajó con series de 100.000 puntos de 32 bits cada uno, generadas por los GNPAs propuestos. El cálculo se realizó de manera tal que se pudieran graficar 500 puntos sobre los ejes log(R/S)–log(*n/b*) a fin de lograr resultados con poco error.

En la TABLA IV se muestran los resultados del cálculo del exponente de Hurst para las series entregadas por los circuitos electrónicos propuestos y los de referencia. Se puede apreciar que no hay diferencia entre los GNPAs propuestos y los tomados como referencia.

IV. Implementación en Lógica Programable

Debido a que los circuitos sumadores y multiplicadores son el núcleo de la mayor parte del hardware RNS, se realizó un amplio estudio sobre este tipo de operaciones en el sistema numérico de residuos [22] y [23]. Esta investigación incluyó distintos tipos de sumadores y multiplicadores presentados en distintos trabajos [24], [25], [26], [27] y [28]. De los resultados obtenidos en [22] y [23] se puede determinar que la frecuencia

[3] Copyright 1984-2004, The MathWorks Inc.

de operación es de 93,4 MHz cuando se emplea una FLEX10K20RC240-4 y trabajando con 32 bits.

TABLA IV. Exponente de Hurst para cada uno de los generadores propuestos y los tomados como referencia.

Generador		H
A	RNS-LCG-I	0,52967
	RNS-LCG-II	0,52944
B	RNS-LCG-I	0,52119
	RNS-LCG-II	0,52314
C	RNS-LCG-I	0,51033
	RNS-LCG-II	0,52219
Uniforme		0,52689
MWCG		0,52616
MTHR4		0,52328
SWBMWC		0,52253

V. CONCLUSIONES

En el presente trabajo se presentaron una serie de generadores de números pseudoaleatorios basados en el sistema numérico de residuos que pasan exitosamente, además de las estadísticas básicas y el paquete Diehard [1], ensayos de complejidad de zipping y el método para determinar la correlación entre valores de una serie propuesto por H. E. Hurst.

REFERENCIAS

[1] C. A. Gayoso, C. González, L. Arnone, M. Rabini "Generador de números pseudoaleatorios mediante el sistema numérico de residuos, implementación en FPGA", Congreso Argentino de Sistemas Embebidos, 2-4 de marzo de 2011.

[2] M. Naito, N. Tanaka, H. Okamoto, "Distinguishing chaos from random fractal sequences by the comparison of forward predictions: utilization of the difference in time reversal symetry of time series," IEEE Proc. Of First International Conference on Knokledge-Based Intelligent Electronic System, 21-23 de mayo de1997, pp. 101-107.

[3] K. Ozdemir, S. Kilinc. S Ozoguz, "Random Number Generator Design Using Continuous-time Chaos," IEEE Signal Processing, Communication and Applications Conference, 20-22 de abril de 2008. pp 1-4.

[4] M. Alioto, S. Bernardi, A. Fort , S. Rocchi and V. Vignoli, "On the suitability of digital maps for integrated pseudo-RNGs," Proc. ECCTD Cracow, Poland, Sep. 2003, p. III/349.

[5] J. Savir "A new empirical test for the quality of random integer generators", IEEE Trans. Comput., vol. C-32, pp. 960, Oct. 1983.

[6] Fred J. Taylor. "Residue arithmetic: A tutorial with examples," Computer Magazine, IEEE Mayo. 1984. pp. 59-62.

[7] M. A. Bayoumi and G. A. Jullien, "A VLSI Implementation of Residue Adders," IEEE Transactions on Circuits and Systems, vol. 34, no. 3, pp. 284-288, Mar. 1987.

[8] Fred J. Taylor. "Large moduli multipliers for signal procesing," IEEE Transactions on circuits and systems, Volumen CAS-28, Número 7, Julio 1981.

[9] Chin-Liang Wang. "IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing," Noviembre 1994, pp. 768-772.

[10] G. A. Jullien, "Implementation of Multiplication, Modulo a Prime Number, with Applications to Number Theoretic Transforms", IEEE Transactions on Computers, vol. C-29, no. 10, pp. 899-905, Oct. 1980.

[11] J. C. Smith and F. J. Taylor, "A fault-tolerant GEQRNS processing element for linear systolic array DSP application", IEEE Transactions on Computers, vol. 44, no. 9, pp. 1121-1130, Sep. 1995.

[12] J. Ramírez, P. G. Fernández, U. Meyer-Bäse, F. J. Taylor, A. García and A. Lloris, "Design of Index-based RNS-DWT Architectures for Custom IC Designs", Proc. of 2001 IEEE Workshop on Signal Processing Systems SiPS'2001, pp. 70-79, Sep. 2001.

[13] W. A. Chren, "RNS-Based Enhancements for Direct Digital Frequency Synthesis," IEEE Transactions on Circuits and Systems II, vol. 42. no. 8, pp. 516-524, Aug. 1995.

[14] J. Ramírez, A. García, P. G. Fernández, L., Parrilla and A. Lloris, "RNS-FPL Merged Architectures for Orthogonal DWT," Electronics Letters, vol. 36, no. 14, pp. 1198-1199, Jul. 2000.

[15] N. S. Szabo and R. I. Tanaka, "Residue Arithmetic and Its Applications to Computer Technology," McGraw-Hill, NY, 1967.

[16] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien and F. J. Taylor, "Residue Number System Arithmetic: Modern Applications in Digital Signal Processing," IEEE Press, 1986.

[17] U. Meyer-Bäse, A. Garcia and F. J. Taylor, "Implementation of a Communications Channelizer using FPGAs and RNS" Arithmetic Journal of VLSI Signal Processing, vol. 28, no. 1/2, pp. 115-128, May 2001.

[18] Altera Corporation, "Cyclone II Handbook," http://www.altera.com /literature/ds/cycloneIIhandbook.pdf, Nov. 2007.

[19] A. Baronchelli, E. Caglioti, V. Loreto, "Measuring complexity with zippers", European Journal of Physics, 26 (2005) S69–S77.

[20] E. Peters, "Chaos and order in the capital markets. A new view of cycles, prices and market volatility" Editorial John Wiley and Sons Inc. 1991.

[21] E. Peters, "Fractal market analysis. Aplying chaos theory to investment and economics", Editorial John Wiley and Sons Inc. 1995.

[22] C. A. Gayoso, A. García, C. M. González, L. Arnone, J. C. García, E. Boemo, "Estudio sobre el diseño de sumadores en aritmética de residuos en lógica programable ", II Jornadas sobre Computación Reconfigurable y Aplicaciones. 10 al 20 de septiembre de 2002, Almuñecar, Granada, España. Anales pág 203. ISBN: 84-600-9928-8.

[23] C. A. Gayoso, C. González, M. Rabini, L. Arnone, "Estudio de multiplicadores en aritmética de residuos empleando lógica programable", Décimo Tercera Reunión de Trabajo en Procesamiento de la Información y Control RPIC 2009, 16 al 18 de septiembre de 2009. Rosario, Argentina. Pág. 954-959. ISBN: 950-665-340-2.

[24] M. A. Bayoumi and G. A. Jullien, "A VLSI Implementation of Residue Adders", IEEE Transactions on Circuits and Systems, vol. 34, no. 3, pp. 284-288, Mar. 1987.

[25] M Dugdale, "VLSI Implementation of Residue Adders based on Binary Adders", IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 39, no. 5, pp. 325-329, Mar. 1987.

[26] G. A. Jullien, "Implementation of Multiplication, Modulo a Prime Number, with Applications to Number Theoretic Transforms", IEEE Transactions on Computers, vol. C-29, no. 10, pp. 899-905, Oct. 1980.

[27] J. C. Smith and F. J. Taylor, "A fault-tolerant GEQRNS processing element for linear systolic array DSP application", IEEE Transactions on Computers, vol. 44, no. 9, pp. 1121-1130, Sep. 1995.

[28] J. Ramírez, P. G. Fernández, U. Meyer-Bäse, F. J. Taylor, A. García and A. Lloris, "Design of Index-based RNS-DWT Architectures for Custom IC Designs", Proc. of 2001 IEEE Workshop on Signal Processing Systems SiPS'2001, pp. 70-79, Sep. 2001.