Generador de números pseudoaleatorios mediante RNS y caos, implementación en lógica programables.

C. A. Gayoso, L. J. Arnone, M. R. Rabini y C. M. González
Laboratorio de Componentes Electrónicos
Universidad Nacional de Mar del Plata, Argentina
Email: cgayoso@fi.mdp.edu.ar

***Resumen*—**
En este trabajo se presenta un generador de números pseudoaleatorios sintetizado en FPGA. Este generador está basado en teoría del caos y en el sistema numérico de residuos. Para validar el generador propuesto se empleó la batería de tests *Diehard*.

***Index Terms*—Sistema Numérico de Residuos, Aritmética de Residuos, Lógica Programable, Caos, Números Aleatorios.**

I. Introducción

Este trabajo estudia la implementación en hardware de generadores de números pseudoaleatorios (Pseudo Random Number Generators o PRNGs), 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. Se propone un nuevo esquemas de PRNGs, basados en el RNS y teoría de caos. El circuito propuesto se evaluó desde el punto de vista estadístico mediante tests básicos y el banco de tests desarrollado por George Marsaglia para su generador *Diehard*. El trabajo está organizado de la siguiente manera. Se comienza con la definición de sistemas determinísticos, caóticos y aleatorios junto con la presentación del test *Diehard* y su empleo. Luego se describe el generador de números pseudoaleatorios propuesto junto con la explicación de cada uno de los bloques que lo constituyen. Se finaliza presentando los aportes y conclusiones del trabajo realizado.

I-A. Determinismo, caos y aleatoriedad

En general los sistemas se pueden clasificar como deterministas, aleatorios o caóticos. En los sistemas deterministas 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 [1] [2]. 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 o las anteriores, no se puede encontrar ningún tipo de patrón o regularidad. En la década de 1960 se comienza a trabajar con un tipo de sistemas en los cuales, a pesar de estar descriptos por ecuaciones deterministas, su sensibilidad a las condiciones iniciales o a la precisión de las herramientas de cálculo que se emplean hacen que su evolución futura, en órbitas que no se superponen, sea impredecible. Este último tipo es un caso intermedio entre los dos primeros, los sistemas que presentan este comportamiento se denominan caóticos [1] [2].

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

I-B. Tests de aleatoriedad

Los test 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 [3] [4]. 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 test a los que es sometida la secuencia numérica bajo estudio en busca de concluir sobre su carácter deterministico, caótico o aleatorio.

Entre los test más utilizados para medir la calidad de una secuencia de números supuestos aleatorios se encuentra el denominado *Diehard* [5] [6]. En realidad es una batería de 17 tests desarrollados por George Marsaglia de manera progresiva desde 1985 y publicados por primera vez juntos en 1995.

El generador propuesto en este trabajo se sometió a dicho test pues es uno de los más díficiles de superar. El paquete *Diehard* tiene como entrada un archivo de al menos 11 Mbytes con la secuencia de números cuya aleatoriadad se desea determinar y entrega una serie de 229 valores denominados p, cada uno de los cuales debe cumplir $0,025 < p < 0,975$ para que la serie se considere aleatoria.

I-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 aumentan, el rendimiento se degrada. El Sistema Numérico de Residuos (RNS) [7] [8] [9] 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 n bits se pueden transformar, mediante el empleo del RNS, en $O\left(\frac{n}{log_2(n)}\right)$ canales de $\lceil log_2(n) \rceil$ bits cada uno [10], ver ecuación 1. Esta característica lo hace apropiado para la realización de un número importante de aplicaciones en procesamiento digital de señales (DSP) [11] [12] [13].

Un sistema basado en RNS se define mediante un conjunto de enteros relativamente primos entre si $\{m_1, m_2..., m_i\}$, llamados módulos. Su rango dinámico, el número de cantidades distintas que se puede representar, es $M = \prod_{m_i} \quad (i = 1, 2, ..., L)$, y de manera que cualquier entero $0 \leq X < M$ se representa mediante un conjunto de L residuos $[x_1, x_2, ..., x_L]$, con $x_i = X \mod m_i \quad (i = 1, 2, ..., L)$. 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:

$$Z = (X \diamond Y) mod\, M \leftrightarrow z_i = (x_i \diamond y_i) mod\, m_i \, (i = 1, 2, ..., L) \quad (1)$$

, donde $\diamond$ significa suma, resta o multiplicación en módulo. En la ecuación 1 se puede apreciar el potencial del RNS, puesto que las operaciones en módulo M se computan en paralelo sobre L canales independientes (ver figura 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 *thoughput* 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 , 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 [14] [15]. Por otro lado los fabricantes de dispositivos lógicos programables (FPL-Field Programmable Logic) proveen circuitos integerados programables en campo para casi todas las aplicaciones de la electrónica digital. Para la implementación de circuitos aritméticos en el RNS los FPLs [16] [17] se han convertido en una seria alternativa al empleo de circuitos implementados con aritmética convencional [18]. Por esta razón el circuito propuesto se ha implementado sobre una FPGA (Field Programmable Gate Array) Cyclone II EP2C35F672C6 [19] de Altera Corporation.

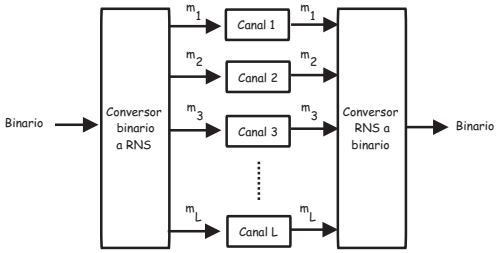

Figura 1. Esquema general de un sistema RNS.

II. Generdor pseudoaleatorio propuesto

II-A. Generador RNS cuasicaótico

En [20] y [21] se presenta un esquema formado por una cascada de filtros de primer orden, cuyos polos coinciden con la raíz primitiva de primer orden del módulo correspondiente a esa etapa. Este esquema se muestra en la figura 2. El circuito está compuesto por n etapas en cascada, en cada uno de ellas los cómputos se realizan sobre un módulo primo. Así, por ejemplo, la salida y_k, con $k \geq 1$ se puede calcular como:

$$\begin{aligned} x_k^1 &= \mid g^1 x_{k-1}^1 + u_k \mid_{m_1} \\ x_k^2 &= \mid g^2 x_{k-1}^2 + x_k^1 \mid_{m_2} \\ ... &= ... \\ y_k = x_k^n &= \mid g^n x_{k-1}^n + x_k^{n-1} \mid_{m_n} \end{aligned} \quad (2)$$

, donde u_k es la entrada al sistema y x_k^i con $i = 1, ..., n$ es la salida de la i_{esima} sección en el instante k.

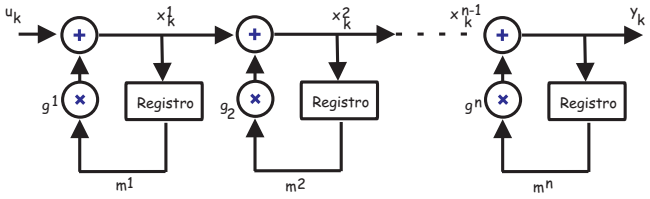

Figura 2. Diagrama esquemático del generador RNS cuasicaótico.

Aún con entrada nula el generador se comporta como un generador cuasicaótico. En efecto, dadas las condiciones iniciales x_0^i, la salida es periódica con un periodo igual al mínimo común múltiplo de los $(m_i - 1)$. Por ejemplo: con $n = 8$ y $m = \{257, 263, 347, 359, 383, 467, 479, 503\}$, el periodo es igual a $256 \times 131 \times 173 \times 179 \times 191 \times 233 \times 239 \times 251 \approx 2,7 \times 10^{18}$.

La forma de la señal generada no depende de los valores iniciales, si estos cambian se obtiene la misma secuencia pero desplazada. Por el contrario si se cambian las raíces primitivas se genera una secuencia distinta. Cada m_i tiene $\phi(\phi(m_i))$ raíces primitivas, siendo m_i primo y $\phi(x)$ la función ϕ de Euler. De manera que el número total de secuencias distintas es para el ejemplo dado:

$$\begin{aligned} F &= \prod_{i=1}^{n} \phi(\phi(m_i)) \\ F &= 128 \times 130 \times 172 \times 178 \times 232 \times 258 \times 250 \\ F &\approx 1,3 \times 10^{18} \end{aligned} \quad (3)$$

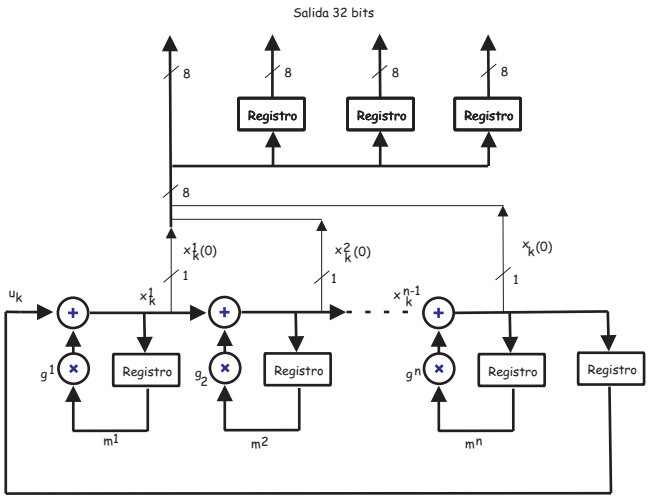

Figura 3. Generador de números pseudoaleatorios de 32 bits propuesto.

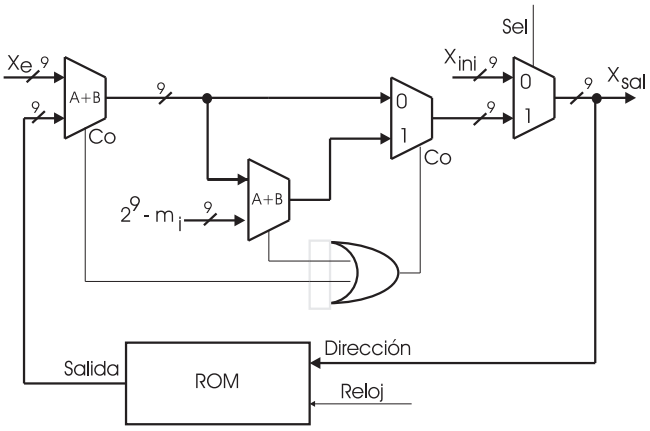

Figura 4. Etapa individual del generador RNS.

Con este esquema se pueden construir generadores cuasi-caóticos con periodos muy largos y cada uno de ellos con un gran número de secuencias distintas.

Si se analiza las secuencias generadas por el circuito descripto mediante la batería de tests *Diehard* se ve que éstas no pasan satisfactoriamente las exigencias de una serie pseudoaleatoria, principalmente por el hecho de trabajar con módulos distintos de 2^n, que hace que los bits generados no son equiprobales.

Se propone entonces modificar el circuito anterior de modo que pueda pasar dichos tests y así podes utilizarlo como generador de números pseudaleatorios.

II-B. Circuito propuesto, test e implementación FPGA

El generador completo propuesto se muestra en el diagrama de la figura 3. En ella se ve un esquema similar al de la figura 2 con las siguientes modificaciones. En dicha figura se observa que la salida de la última etapa se conecta, mediante un registro, a la entrada u_k genérica del circuito inicial. Los registros además permiten ingresar las condiciones iniciales a partir de las cuales comenzará a trabajar el sistema.

La salida de cada canal se denomina x_k^i, es decir el canal i en el instante k, cada uno de ellos de 9 bits. De cada canal se toma el bits menos significativo $x_k^i(0)$, formando una palabra de 8 bits. En cuatro transiciones del reloj del sistema se obtienen palabras de 32 bits, que es la salida final del generador.

Se simuló numéricamente el sistema propuesto a través de MATLAB generando secuencias de 2500000 palabras de 32 bits, dicha secuencias pasaron satisfactoriamente la batería de tests estadísticos *Diehard* de Marsaglia. Por lo que se considera a este generador apto para ser utilizado como generador de números pseudoaleatorios de 32 bits. Es de notar que las condiciones iniciales son determinadas por una palabra llave de 72 bits

Para implementar el generador completo se utilizó el software Quartus II 7.2 de Altera y el integrado Cyclone II EP2C35F672C6 de la placa de desarrollo DE2.

La implementación del sistema se basó en los esquemas de las figuras 2 y 3, cada etapa se implementó con el circuito mostrado en la figura 4 utilizando captura esquemática, mediante bloques de biblioteca propios de Altera.

En cada etapa se realiza la suma en módulo m_i de la señal actual más la señal anterior afectada por el multiplicador g^i, esta suma se realiza directamente si el resultado es menor que m_i o se corrige sumando $2^9 - m_i$ en el caso contrario, con los acarreos de los sumadores se controla el multiplexor que elige una u otra opción.

La multiplicación de la salida por g^i se realiza por medio de una tabla de búsqueda implementada en un bloque de memoria ROM temporizada por el reloj del sistema. Adicionalmente se agregó un multiplexor a la salida para cargar la condición inicial, esta operación se controla con la señal "sel".

Las ocho etapas se conectan en cascada ocupando en total 361 Elementos Lógicos (LEs) y 36864 bits de memoria. Utilizando la herramienta "Timing Analyzer" del Quartus II se verificó por simulación una máxima frecuencia de trabajo de 20,25 MHz. Por lo tanto la frecuencia final de generación de números será de 5,0625 MHz.

III. Conclusiones

En este trabajo se presentó un nuevo generador de números pseudoaleatorios basado en teoría de caos y en el sistema numérico de residuos. El mismo se implemento en lógica programable. La característica distintiva de este nuevo generador es su modularidad. En efecto, al trabajar con el sistema numérico de residuos su rango dinámico y periodo se pueden incrementar adicionándole más canales RNS, sin que esto se vea traducido en una desmejora importante en su frecuencia de funcionamiento. Como trabajo futuro se estudiará la posibilidad de segmentación de los canales y sumadores RNS a fin de incrementar su *throughput*.

Referencias

[1] 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.

[2] 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.

[3] G. Marsaglia, "A current view of random number generation",*Proc. Comput. Sci. Statistics: Sixteenth Symp*. Interface Atlanta, GA, Mar. 1984.

[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] 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.

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

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

[8] 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.

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

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

[20] M. Panella, G. Martinelli, "RNS quasi-chaotic generators", *Electonics Leters, IEEE*. Volumen: 36, 20 de julio de 2000, pp. 1325-1326.

[21] M. Panella, G. Martinelli, "RNS quasi-chaotic generators for self-correcting secure communication", *Electonics Leters, IEEE*, Volumen: 37, 1 de marzo de 2001, pp.325-327.