elektron ámbar

Original Article

Revista elektron, Vol. 6, No. 1, pp. 46-51 (2022)
https://doi.org/10.37537/rev.elektron.6.1.156.2022

Diseño e implementación de PRNG caótico con distribución variable en el tiempo

Design and Implementation of a Chaotic-Time-Varying Distribution PRNG

Raúl Eduardo Lopresti[†1], Maximiliano Antonelli[†2], Julio Dondo[*3] y Luciana De Micco[†4]

[†]*Instituto de Investigaciones Científicas y Tecnológicas en Electrónica (ICYTE)*
Facultad de Ingeniería, Universidad Nacional de Mar del Plata (FI-UNMdP)
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
[1]lopresti@fi.mdp.edu.ar
[2]maxanto@fi.mdp.edu.ar
[4]ldemicco@fi.mdp.edu.ar
[*]*Dto. de Electrónica*
Facultad de Ciencias Físico-Matemáticas y Naturales, Universidad Nacional de San Luis (UNSL)
[3]jdondo@unsl.edu.ar

Recibido: 15/03/22; Aceptado: 08/05/22

***Resumen*—Gran parte de las aplicaciones electrónicas requieren de números pseudoaleatorios, en general, para mejorar su funcionamiento. Tal es el caso de los sistemas de encriptación, codificación y modulación digital. Si, además, los números pseudoaleatorios varían dinámicamente su función densidad de probabilidad (PDF) es posible potenciar el efecto causado. En este trabajo se presenta el diseño e implementación de un circuito capaz de entregar números pseudoaleatorios que varían su PDF en el tiempo. Para ello, se utilizan como base mapas caóticos, los que son diseñados según las PDF deseadas. Luego, la implementación se realiza mediante Reconfiguración Parcial Dinámica (RPD) la cual permite modificar, en tiempo de ejecución, parte del circuito para variar la PDF de la salida generada.**

Palabras clave: Generador de números pseudoaleatorios; Función Densidad de Probabilidad; Reconfiguración Parcial Dinámica; Caos.

***Abstract*—Many electronic applications require pseudo-random numbers, in general, to improve their performance. Such is the case of encryption, coding and digital modulation systems. In addition, it is possible to enhance the effect if the pseudorandom numbers dynamically vary their probability density function (PDF). In this article, a circuit that generates pseudo-random numbers that are capable of varying their PDF in time is presented. To this aim, chaotic maps are used as a base, which are designed according to the desired PDF. Then, the implementation is done through Dynamic Partial Reconfiguration (RPD) which allows modifying, at run time, part of the circuit to vary the PDF of the generated output.**

Keywords: Pseudorandom number generator, Probability Density Function, Dynamic Partial Reconfiguration, Chaos.

I. Introducción

La pandemia COVID-19 ha producido una aceleración en el crecimiento del tráfico y en la, ya creciente, migración de datos digitales a los centros de datos y la nube pública. El incremento en el tráfico fue impulsado en gran medida por el uso de conferencias a través de internet, reuniones o clases virtuales, así como juegos en línea y streaming de video. Todo incremento en la recolección de datos aumenta los posibles problemas de privacidad, tanto en la transmisión como en el almacenamiento de los mismos. El cifrado es la base principal de la seguridad de datos para evitar ataques y robo de información. Recientemente han surgido técnicas de cifrado que varían en el tiempo distintas partes de sus bloques constituyentes utilizando secuencias pseudoaleatorias. Ejemplo de esto es la variación dinámica aleatoria de los codificadores constituyentes de un codificador turbo, esto es propuesto y desarrollado en [1]. En [2]–[5] los autores proponen sistemas de encriptación en los que se varía en tiempo de ejecución la clave de encriptación empleada. Lo que proponemos en este artículo es avanzar a un nivel mayor de seguridad variando dinámicamente la PDF de los números aleatorios generados. Recientemente se ha hecho posible este tipo de implementación gracias al desarrollo de la tecnología RPD. Esta tecnología permite la implementación de bloques con alto grado de complejidad, ocupando un área mínima, y permitiendo su integración con el resto del sistema. Mediante RPD es posible la reconfiguración en tiempo de ejecución de una partición específica en una FPGA (Field Programmable Gate Array).

La generación clásica de números pseudoaleatorios con distintas distribuciones de probabilidad es un problema estudiado principalmente en ingeniería de software y se basa en el desarrollo de algoritmos a ejecutarse sobre plataformas secuenciales [6]. Estos algoritmos parten de generadores de números pseudoaleatorios con distribución uniforme de base y son en general muy complejos. Por su parte, los mapas caóticos son capaces de generar secuencias de números con apariencia aleatoria y dada la estructura de estos mapas permiten una ejecución concurrente. Esto los hace ideales para su implementación en dispositivos tales como FPGAs. En cuanto a las características de los datos generados la distribución (Invariant Density Probability Function, IPDF) que tendrán los números aleatorios generados y la velocidad con la que se "mezclan"(constante de *mixing*) depende únicamente de la estructura del mapa. De esta forma es posible "sintetizar" un mapa caótico a partir de la IPDF y la constante de *mixing* deseadas, esto es conocido como Problema Inverso de Perron-Frobenius (IFPP) [7].

En este artículo se presenta la implementación de un circuito, basado en mapas caóticos, capaz de variar en el tiempo la PDF de su datos de salida. La RPD permite ahorrar recursos de la FPGA implementando distintos circuitos (multiplexados en el tiempo) en el espacio que ocupa uno sólo. Todo el circuito está confinado en una pequeña área de la FPGA. El bloque desarrollado, además de ser empleado para incrementar la seguridad en algoritmos de encriptamiento, abre un nuevo abanico de posibilidades de implementar en FPGAs algoritmos concurrentes.

II. Diseño de mapas caóticos con características preestablecidas

Existen muchos estudios que abordan la búsqueda de una solución al IFPP [7]–[9]. Este trabajo empleó la desarrollada por Rogers et. al. [10] quienes presentan una solución basada en la teoría de matrices positivas. Este método es usado exitosamente en [11] para implementar en una FPGA un generador de ruido con distribución Gaussiana. El método se basa en una matriz estocástica (matriz A) que describirá la dinámica del mapa caótico a diseñar. Cada componente de esta matriz expresa la probabilidad de transición de un intervalo a otro. El autovector asociado al autovalor unitario de la matriz A se corresponde con la IPDF del mapa. Por lo tanto, diseñando adecuadamente esta matriz se consigue el comportamiento deseado. La limitación del método es que sólo puede generar mapas que posean densidades invariantes lineales por tramos. Por lo tanto se deberán crear tantos tramos (filas y columnas de la matriz red$A$) como sean necesarios para conseguir la aproximación requerida.

Para definir la matriz se particiona el intervalo $(0, 1)$ en n subintervalos, I_1, ..., I_n . La probabilidad de transición $p_{i,j}$ del intervalo I_i al intervalo I_j se corresponde con el recíproco del elemento $a_{i,j}$ de la matriz A:

$$A = \begin{pmatrix} \beta_1 + \alpha_1(1-\beta_1) & \alpha_1(1-\beta_2) & \dots & \alpha_1(1-\beta_n) \\ \alpha_2(1-\beta_1) & \beta_2 + \alpha_2(1-\beta_2) & & \\ \dots & & & \dots \\ \alpha_n(1-\beta_1) & \dots & & \beta_n + \alpha_n(1-\beta_n) \end{pmatrix}. \quad (1)$$

La matriz A es una matriz estocástica por lo que es siempre positiva cuando $\alpha_i \geq 0$ y $0 < \beta_i < 1 \ \forall \ i \ \epsilon \ 1, ..., n$. De la teoría de matrices positivas se sabe que uno de los autovalores de la matriz A es unitario, y su autovector asociado, x_p, corresponde a la densidad IPDF del proceso gobernado por A, y tiene la siguiente forma:

$$x_p = \left(\frac{\alpha_1}{1-\beta_1}, ..., \frac{\alpha_n}{1-\beta_n} \right). \quad (2)$$

Eligiendo debidamente los valores de α_i y β_i se consigue que el autovector correspondiente al autovalor unitario presente valores determinados, y así se consigue que los números generados por el mapa tengan la densidad invariante deseada. Los pasos son los siguientes:

1. Se establecen los n puntos que se empelarán para aproximar la PDF deseada. Considerando que una mayor cantidad de puntos resultan en una mejor aproximación, pero implican un mapa con más tramos (esto se verá más adelante).
2. Estos puntos se igualan al vector de la IPDF x_p (ec. 2).
3. Se establece un valor para todos los parámetros β_i (o α_i) y se despejan los restantes.
4. Una vez establecidos los valores de los β_i y los α_i se arma la matriz A según la ecuación 1.
5. El mapa caótico resultante presentará n tramos de igual tamaño (correspondientes con cada columna de la matriz A). Cada elemento fila de una columna dada representa el porcentaje de ocupación de esa fila en el ancho total del tramo de la columna correspondiente (Fig. 2). Cabe destacar que la forma del mapa en cada tramo no esta establecida por la metodología, donde una posibilidad es utilizar funciones lineales como las empleadas en la Fig. 2.
6. Finalmente iterando en el mapa caótico obtenido, se generan secuencias cuya PDF coincide con la aproximada en el punto 1.

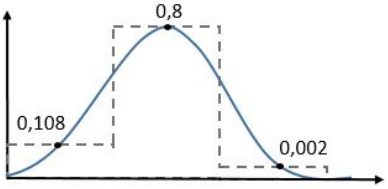


Figura 1. Ejemplo de aproximación de PDF, en este caso se tomaron tres puntos ($n = 3$). Aquí $x_p = (0{,}108; 0{,}8; 0{,}002)$. En azul la PDF original, en línea punteada la aproximación tomada a dicha PDF.

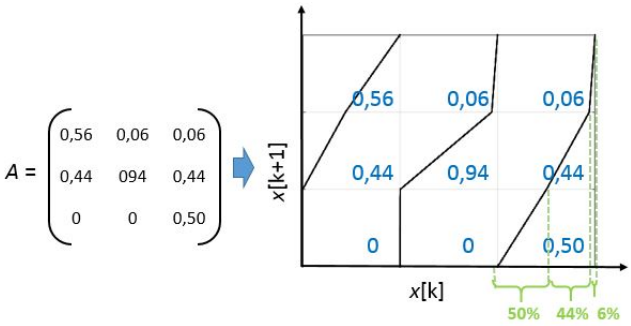


Figura 2. Mapa caótico lineal por tramos generado a partir de la matriz A correspondiente.

III. Diseño e implementación del PRNG

El generador a implementar debe ser capaz de variar en tiempo de ejecución las distribuciones de los datos que genere. Por ello se definieron las PDFs a emplear, se determino con cuántos puntos aproximarlas y en base a ello se diseñaron los mapas caóticos que generarán los datos con tales PDFs, mediante el método descripto en la sección anterior. De esta forma, todos los mapas caóticos a implementar tienen una estructura lineal por tramos como la de la Fig. 2. Se ideó un diseño que cumpliera con las siguientes especificaciones:

- El diseño tiene que ser simple para que pueda trasladarse de manera directa a hardware y para que requiera pocos recursos en una FPGA.
- Debe ser simulado en una computadora de escritorio (PC) trabajando con números de punto flotante para verificar la correcta implementación del método descrito en el apartado anterior.

- Tiene que ser traducido a variables de punto fijo y ser simulado en PC para obtener una primera aproximación del comportamiento funcional que tendría el circuito ya implementado en hardware, permitiendo a la vez determinar la cantidad de bits necesarios para conseguir los resultados deseados.
- Debe implementarse en un procesador embebido a fin de comparar su desempeño con el de una implementación puramente en lógica programable.
- Los resultados de las simulaciones en PC deben ser almacenados para una posterior comparación con los datos que genere el diseño ya implementado en FPGA a fin de validar su correcto funcionamiento, tanto para la versión con procesador como para la versión con solo lógica programable

Por lo tanto, el desarrollo contó con las etapas que se describen en los siguientes apartados.

III-A. Elaboración y prueba del algoritmo en PC

Se escogió implementar los tramos de los mapas caóticos mediante ecuaciones lineales, de la forma de pendiente y ordenada al origen, a fin de que la implementación requiera menos cálculos y utilice menos recursos de hardware. Entonces, el algoritmo consultará una tabla previamente almacenada en la que se define el mapa caótico. La tabla tiene tantas filas como tramos del mapa caótico (n), cada una de estas filas almacena un valor de pendiente (m) y de ordenada al origen (b). La secuencia se genera con un algoritmo iterativo de cuatro pasos, el mismo se presenta en la Fig. 3 y se describen a continuación:

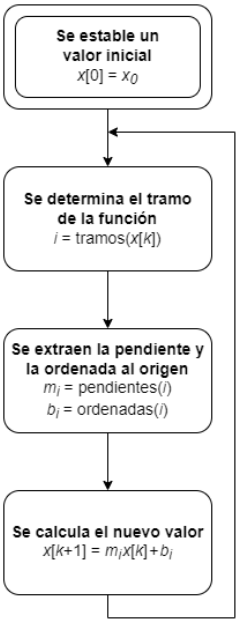


Figura 3. Algoritmo que implementa el mapa caótico para la generación de secuencias con una determinada PDF.

1. Se establece un valor inicial x_0 para la secuencia caótica $x[k]$. Este valor es equivalente a lo que se conoce como semilla en muchos algoritmos aleatorios.
2. Se determina a cuál tramo i de la ecuación (tramo del mapa caótico) corresponde el valor actual de la secuencia.
3. Se obtienen de la tabla almacenada los valores de ordenada al origen b_i y pendiente m_i correspondientes al tramo i.
4. Con estos tres datos se calcula el nuevo valor de la secuencia ($x[k+1] = m_i x[k] + b_i$), el cual se utiliza para calcular el subsiguiente valor repitiendo los pasos del 2 al 4 de este algoritmo.

Inicialmente se implementó el algoritmo en una PC mediante un software de simulación utilizando aritmética de punto flotante con el objetivo de validar el mismo. Luego se iteró el algoritmo utilizando distintas aritméticas. Se utilizó primero aritmética de punto flotante (estándar IEEE754), tanto en precisión simple (32 bits) como en precisión doble (64 bits). Con el objetivo de reducir los recursos empleados y simplificar su implementación en hardware se utilizó también aritmética de punto fijo con distintas cantidades de bits (n_{bits}).

La representación numérica juega un papel importante debido a la sensibilidad a las condiciones iniciales, inherente a los mapas caóticos [12]. Aquí entra una relación de compromiso entre precisión numérica y área de circuito. A modo gráfico en la Fig 4 se muestra las PDFs obtenidas en el caso de aproximar la PDF triangular con 13 puntos (en negro) y las PDFs generadas iterando en los mapas obtenidos para distintas aritméticas (en celeste). Se generaron secuencias de 40.000 datos cada una y se eliminó el transitorio descartando los primeros 10.000 datos. Se puede ver que en punto fijo con 24 bits la aproximación no es buena (Fig. 4.a), con 38 bits la PDF obtenida se ajusta a la deseada (Fig. 4.b). En el caso de trabajar con punto flotante, utilizando precisión simple (Fig. 4.c) no se ajusta a la PDF deseada, en cambio, precisión doble se ajusta a la PDF deseada (Fig. 4.d).

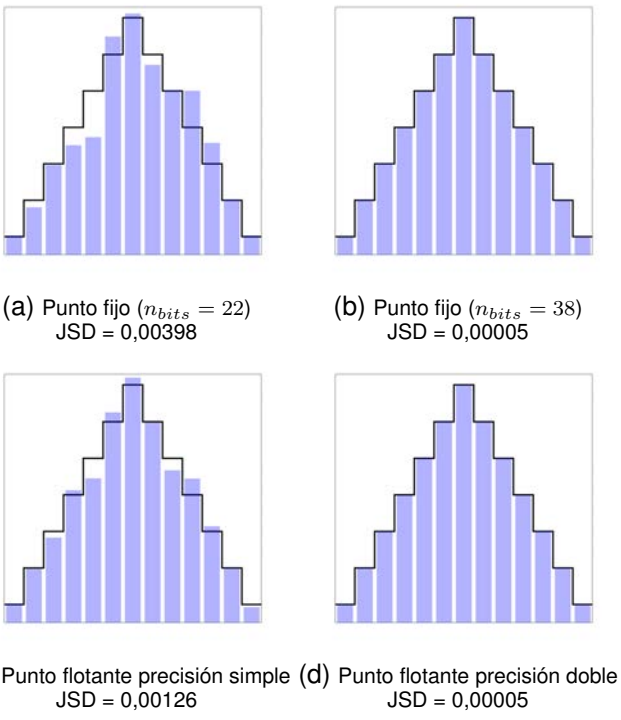
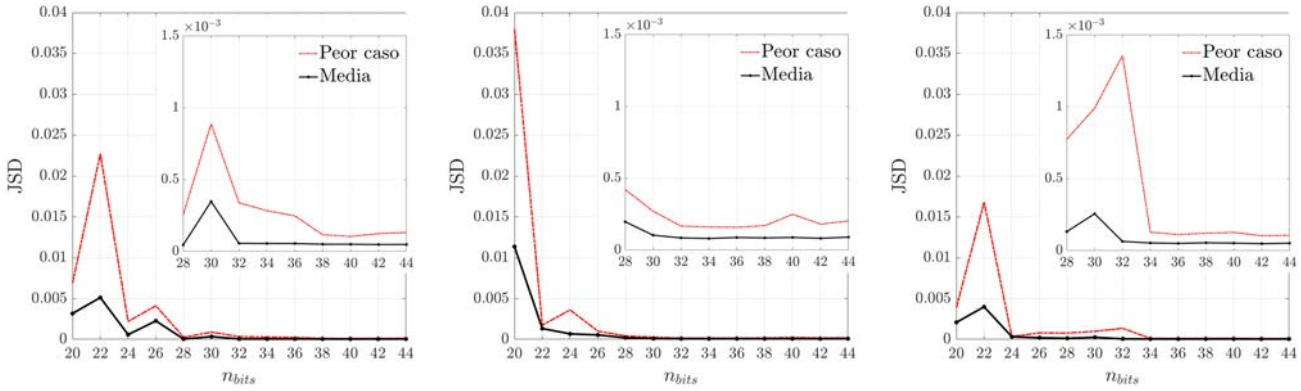


Figura 4. En azul histogramas generados iterando en los mapas caóticos con diferentes aritméticas (punto fijo y punto flotante), en negro IPDF del mapa caótico.



(a) PDF Gaussiana (b) PDF Logística (c) PDF Triangular

Figura 5. Divergencia de Jensen–Shannon calculada para tres PDFs utilizando distinta cantidad de bits en aritmética de punto fijo.

Para cuantificar la aproximación de la distribución de las secuencias generadas a la distribución ideal (aproximada mediante el método) utilizamos la divergencia de Jensen-Shannon (JSD) [13]. Este cuantificador está definido como una medida de distancia en el espacio de probabilidad y se calcula en base a dos PDFs, el valor resultante es una medida de distancia estadística entre ellas. Utilizamos tres distribuciones de prueba (Gaussiana, Logística y Triangular) que se aproximaron utilizando trece puntos ($n = 13$) mediante la metodología detallada en la sección II. Los mapas caóticos resultantes se evaluaron para distintas aritméticas para definir una cantidad mínima de bits para los cuales esta aproximación funciona. En estas pruebas se calcula la PDF sobre los datos generados por el mapa caótico y se calcula su distancia estadística con la PDF esperada. Dado que la secuencia de salida de cada mapa caótico será distinta dependiendo de su condición inicial, se generaron 100 secuencias cada una para distintas condiciones iniciales elegidas aleatoriamente (surrogados) de 40.000 datos cada una. En todos los casos se eliminó el transitorio descartando los primeros 10.000 datos. En la Fig. 5 se muestran estos resultados para aritmética de punto fijo. En el eje de abcisas se representa la cantidad de bits de precisión (n_{bits}) mientras que en las ordenadas se representa la JSD entre las dos PDFs. Cuanto más alto es el valor representado en estas curvas mayor es la distancia entre las dos PDFs. En el gráfico se reportan el promedio de las 100 JSDs en negro y el peor caso entre las 100 JSDs en rojo. Vemos en el subgráfico una ampliación de la zona estable. Se puede ver que este error es despreciable para n_{bits}>32 en todos los casos. Este cuantificador se calculó para todas las aritméticas empleadas, en la Fig. 4 puede verse el valor obtenido de JSD para cada uno de los cuatro casos mostrados allí.

III-B. Traslado del algoritmo a un microprocesador embebido

A fin de poder comparar el desempeño se realizó la implementación en un microprocesador los mapas caóticos propuestos. Dado que en la simulación en PC con variables de punto flotante de precisión simple no se obtuvieron buenos resultados, en esta implementación se trabajó solamente con variables de punto flotante de precisión doble. En este caso, las secuencias generadas con resultaron iguales a las generadas en PC y se obtuvieron en un tiempo significativamente menor.

III-C. Traslado del algoritmo a lógica programable

La Fig. 6 presenta un esquema del circuito implementado en hardware. Este consiste básicamente en la implementación del algoritmo de la Fig. 3.

El bloque “Registros” latchea la salida de la secuencia a la frecuencia del reloj (clk). El resto de los bloques son combinacionales y determinan la máxima frecuencia de generación de los datos (clk máximo). El bloque “Parámetros” determina el tramo i correspondiente del valor actual de la secuencia y con esta información se extraen los parámetros de m_i y b_i (pasos 2 y 3 del algoritmo 3). El bloque “Cálculos” recibe los parámetros m_i y b_i y el valor actual de la secuencia, para calcula el valor siguiente (paso 4 del algoritmo).

El bloque “Parámetros” es el objeto de la RPD, ya que su configuración determina la distribución que tendrán los datos generados. El resto de los bloques son estáticos pues no cambian su configuración, realizan siempre las mismas operaciones.

Se realizaron síntesis, implementaciones y simulaciones en Vivado. Con las diferentes simulaciones se verificó que las secuencias generadas coincidían con las obtenidas en las simulaciones realizadas con Matlab. Con las síntesis se determinaron los recursos de hardware que requiere cada una de las implementaciones del PRNG en la FPGA. Con las implementaciones se determinaron las frecuencias máximas para la generación de las secuencias pseudoaleatorias. Estos resultados se presentan en la sección V.

IV. Diseño e implementación del PRNG con RPD

La implementación y pruebas del PRNG diseñado se llevaron a cabo en un kit de desarrollo y evaluación de la marca Xilinx. El diseño y la generación de los archivos (bitstreams) de configuración y de reconfiguraciones parciales de la FPGA se realizaron con el software Vivado provisto por dicha marca. Además, Xilinx provee algunos logiCORE IP y una amplia documentación que simplifica la introducción al diseño con RPD. Con el objeto de probar el PRNG descripto en una implementación con RPD, se diseñó el circuito que se esquematiza en la Fig. 7. El mismo consta de:

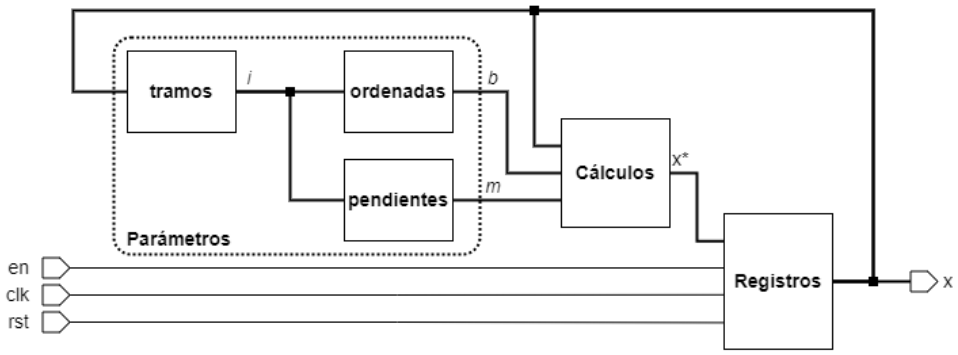


Figura 6. Esquema del diseño del PRNG para implementación en FPGA.

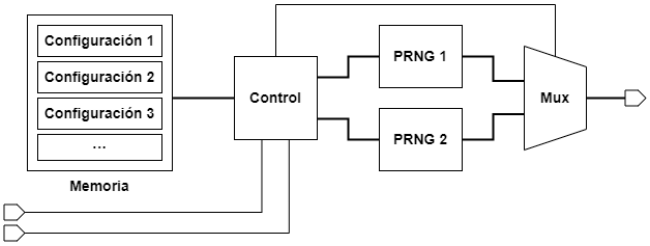


Figura 7. Esquema del PRNG implementado con RPD.

- Una memoria, que puede ser interna o externa al chip de la FPGA, donde se almacenan los archivos para las reconfiguraciones parciales. Estos son los responsables de modificar el generador para que cuente con distintas PDFs. En este trabajó se utilizó la memoria DDR3 incluida en el chip de la FPGA.
- Un bloque de control que se encarga de gestionar las reconfiguraciones. En la diseño realizado se utilizó el logiCORE IP "Dynamic Function eXchange Controller" provisto por Xilinx para tal fin, el cual agilizó la implementación para realizar las pruebas.
- Dos bloques llamados PRNG, esquematizados en la Fig. 6, que son los que permiten generar datos con PDFs distintas. La parte de estos bloques llamada "Parámetros" es la que se reconfigura con los distintos bitstreams. Además, en este diseño mientras un PRNG se reprograma el otro provee la secuencia caótica, evitando así tiempos muertos durante las reprogramaciones.
- Un multiplexor para seleccionar la salida del generador activo. De esta forma el tiempo mínimo que se mantendrá un generador con cada PDF queda determinado por el tiempo que demora la reconfiguración del otro generador. Una manera de disminuir este tiempo es empleando más bloques PRNG en el diseño.

V. Resultados

El diseño realizado se implementó y probó en tres soportes diferentes y con tres PDF distintas. Las PDF de las secuencias generadas fueron: Beta, Logística y Dos Escalones. Por otro lado, los soportes empleados fueron: una PC con sistema operativo Windows 10, 8 GB de memoria RAM y un procesador Intel i3 haciendo uso del software Matlab; un microprocesador embebido Cortex-A9 de ARM (Advanced RISC Machine) corriendo el algoritmo programado en bare-metal, y en lógica programable sobre una FPGA.

Los últimos dos soportes usados pertenecen a un mismo chip de la placa ZedBoard de Xilinx. Se escogió esta placa para realizar las pruebas por ser compatible con la plataforma de radio frecuencia ADFMCOMMS4-EBZ de la empresa Analog Devices [14]. La cual se prevee usar para implementar un diseño de SDR (Software Defined Radio) completo que pueda aprovechar el PRNG desarrollado en este trabajo.

En la implementación se generaron secuencias caóticas con aritmética de punto flotantes y de punto fijo. En el primer caso se emplearon ambas precisiones del estándar IEEE754, pero sólo se obtuvieron buenos resultados al usar la precisión doble. En el segundo caso las distribuciones obtenidas fueron buenas al usar una precisión de 34 bits o superior.

Los parámetros para los mapas caóticos obtenidos en PC, tanto en punto flotante como en punto fijo, son los que se emplearon en la implementación sobre los otros dos escenarios. En la implementación en ARM se realizaron los cálculos en punto flotante de precisión doble y a la máxima frecuencia admitida por el procesador, 666,666 MHz. Las secuencias obtenidas fueron las mismas que las generadas en PC, y a su vez requirió un menor tiempo de ejecución. Finalmente se generaron secuencias en la FPGA utilizando aritmética en punto fijo de 34 bits. Las secuencias se obtuvieron en un tiempo mucho menor que el requerido en los otros escenarios. Las secuencias generadas fueron las mismas que las generadas en PC con punto fijo y presentaron las distribuciones esperadas.

Esto se resume en la Tabla I donde se presentan los tiempos requeridos para generar una secuencia de 40.000 muestras en los diferentes soportes y con diferentes PDFs. A su vez, en la Tabla pueden verse los recursos de hardware requeridos por el generador configurado con distintas PDFs, y en la Tabla III se observan los recursos requeridos al implementar en la ZedBoard el diseño de la Fig. 7.

VI. Conclusiones

Se diseñó, implementó y probó en FPGA un PRNG que puede cambiar su PDF en el tiempo mediante RPD. El circuito implementado posee potencial para ser empleado en diversos bloques de distintas aplicaciones electrónicas, principalmente en encriptación. Los histogramas de todas las secuencias generadas se ajustaron a las formas de las

Cuadro I
TIEMPO (EXPRESADOS EN MILISEGUNDOS) REQUERIDO PARA GENERAR 40.000 MUESTRAS MEDIANTE LOS MAPAS CAÓTICOS LINEALES POR TRAMOS CON TRES IPDFS DISTINTAS.

Soporte	Arquitectura	PDF Beta	PDF Logística	PDF Dos escalones
PC	Flotante precisión simple	322,607	308,981	297,855
PC	Flotante precisión doble	277,504	293,542	289,628
ARM	Flotante precisión doble	126,663	126,503	144,851
FPGA	Punto fijo 34 bits	1,000	1,000	1,000

Cuadro II
RECURSOS EMPLEADOS EN LA IMPLEMENTACIÓN A 40 MHZ DE TRES PRNGS EN UNA ZEDBOARD DE XILINX.

bloque (PDF)	LUTs	Registros	Slice	DSP
Cálculos + Registros	154	40	62	6
Parámetros (Beta)	2.229	0	528	0
Parámetros (Logística)	2.363	0	643	0
Parámetros (Dos escalones)	1.101	0	331	0

Cuadro III
RECURSOS EMPLEADOS EN LA IMPLEMENTACIÓN A 40 MHZ DE UNA CONFIGURACIÓN DEL DISEÑO CON RPD EN UNA ZEDBOARD DE XILINX.

bloque (PDF)	LUTs	Registros	Slice	DSP
PRNG 1	2.517	40	695	6
PRNG 2	2.517	40	695	6
Mux	2	1	3	0
Control	1.104	1.217	393	0
Requeridos	6.140	1.298	1.786	12
Disponibles	53.200	106.400	13.300	220
Ocupación %	11,54	1,22	13,45	5,45

PDF deseadas. Se implementó el algoritmo en diferentes escenarios y se compararon todos los tiempos de ejecución, resultando menor el correspondiente a la implementación en FPGA. Además, se presentaron los elementos de hardware requerido en dicha implementación. Como trabajo futuro se prevé integrar diseño en la parte de encriptación de un sistema SDR para incrementar su nivel de seguridad.

AGRADECIMIENTOS

Este trabajo fue parcialmente financiado por CONICET (PIP2017 11220170100553CO), la UNMDP, la Agencia I+D+i (PICT19 3024) y Proyecto PROICO 03-2320 de la UNSL.

REFERENCIAS

[1] L. De Micco, D. Petruzzi, H. A. Larrondo, and J. C. C. Moreira, "Randomness of finite-state sequence machine over gf (4) and quality of hopping turbo codes," *IET Communications*, vol. 7, no. 9, pp. 783–790, 2013.

[2] K. Abd El-Latif, H. Hamed, and E. Hasaneen, "Fpga implementation of the pipelined data encryption standard (des) based on variable time data permutation," *the online journal on electrics and electrical engineering (OJEEE)*, vol. 2, no. 3, pp. 298–302, 2010.

[3] P. Cao, X. Hu, J. Wu, L. Zhang, X. Jiang, and Y. Su, "Physical layer encryption in ofdm-pon employing time-variable keys from onus," *IEEE Photonics Journal*, vol. 6, no. 2, pp. 1–6, 2014.

[4] S. Oukili and S. Bri, "High throughput fpga implementation of data encryption standard with time variable sub-keys," *International Journal of Electrical and Computer Engineering*, vol. 6, no. 1, p. 298, 2016.

[5] S. L. Davis, "Enhancing system security using dynamic hardware," Ph.D. dissertation, University of South Alabama, 2022.

[6] R. C. Cheng, "Random variate generation," *Handbook of Simulation*, pp. 139–172, 1998.

[7] D. Pingel, P. Schmelcher, and F. Diakonos, "Theory and examples of the inverse frobenius–perron problem for complete chaotic maps," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 9, no. 2, pp. 357–366, 1999.

[8] N. Wei, "Solutions of the inverse frobenius-perron problem," Ph.D. dissertation, Concordia University, 2015.

[9] C. Fox, L.-J. Hsiao, and J.-E. K. Lee, "Solutions of the multivariate inverse frobenius–perron problem," *Entropy*, vol. 23, no. 7, p. 838, 2021.

[10] A. Rogers, R. Shorten, and D. M. Heffernan, "Synthesizing chaotic maps with prescribed invariant densities," *Physics Letters A*, vol. 330, no. 6, pp. 435–441, 2004.

[11] L. De Micco and H. Á. Larrondo, "Síntesis e implementación en fpga de un mapa caótico con pdf gaussiana," in *II Congreso de Microelectrónica Aplicada (μEA 2011)(La Plata, 7 al 9 de septiembre de 2011)*, 2011.

[12] L. De Micco, M. Antonelli, and H. A. Larrondo, "Stochastic degradation of the fixed-point version of 2d-chaotic maps," *Chaos, Solitons & Fractals*, vol. 104, pp. 477–484, 2017.

[13] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Transactions on Information theory*, vol. 37, no. 1, pp. 145–151, 1991.

[14] Analog Devices, *ADFMCOMMS4-EBZ*. [Online]. Available: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/eval-ad-fmcomms4-ebz.html