DISEÑO E IMPLEMENTACIÓN DE UN GENERADOR DE NÚMEROS ALEATORIOS EN UN SISTEMA EMBEBIDO

JUAN DIEGO BOLAÑOS LANZZIANO
VÍCTOR MANUEL MONSALVE HERNÁNDEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2015

DISEÑO E IMPLEMENTACIÓN DE UN GENERADOR DE NÚMEROS ALEATORIOS EN UN SISTEMA EMBEBIDO

JUAN DIEGO BOLAÑOS LANZZIANO
VÍCTOR MANUEL MONSALVE HERNÁNDEZ

Trabajo de Grado para optar al título de
Ingeniero Electrónico

Director.
OSCAR MAURICIO REYES TORRES
Dr. Ingeniería Electrónica

Codirector.
HÉCTOR IVÁN GÓMEZ ORTIZ
MSc. Ingeniería Electrónica

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2015

…a mi padre y mi madre

A mi hermana

Por su apoyo y amor…

JUAN DIEGO

…a *mis padres, hermanos y a ella,*
Por su amor, comprensión
Y apoyo incondicional…

VÍCTOR

AGRADECIMIENTOS

Durante esta etapa de mi vida, han existido una gran cantidad de personas que de alguna manera contribuyeron en la realización de mis objetivos profesionales y personales. Llegando al final de este periodo, quiero expresarle mi agradecimiento:

A mi padre, madre y hermana, por durante toda mi vida apoyar de forma incondicional los objetivos que me he trazado, y en especial, por motivarme a seguir adelante en los momentos más difíciles.

A Karent por brindarme su compañía y constante apoyo. De igual forma, a la señora Gloria e hijos, por acogerme en su hogar y brindarme un ambiente agradable durante los últimos años.

A todos los amigos y amigas que durante este periodo colaboraron en mi desarrollo personal. En especial a Daniel, Felipe, Héctor, Juan Camilo, Luisa, Margarita, Paula, Roberth, Tavo y Yojanes. Igualmente, agradezco a Víctor, quien además de ser mi amigo, fue quien me acompaño durante el desarrollo de este proyecto.

A la universidad Industrial de Santander y especialmente, a los profesor y funcionarios de la escuela de ingeniería electrónica, que fomentaron mi aprendizaje. De igual forma, al profesor Oscar Reyes por su apoyo y asesoría durante el desarrollo de este proyecto de investigación.

JUAN DIEGO

AGRADECIMIENTOS

Finalizando esta importante etapa de mi vida, deseo expresar mi más sincero agradecimiento a las personas, que de una u otra forma, contribuyeron para alcanzar esta meta.

Agradezco a mis padres y hermanos, por su apoyo incondicional para lograr este objetivo y por ser el motivo que me impulsa a ser cada día mejor persona.

Agradezco a todos mis amigos por acompañarme durante el camino, con ellos compartí gratos momentos que sin duda recordaré a lo largo de mi vida.

Agradezco a Juan Diego, su compromiso y dedicación en este trabajo y su amistad, representan el valor de un gran profesional y amigo.

Agradezco al profesor Daniel Sierra por brindarme su amistad. Sus consejos me ayudan a ser un mejor profesional.

Por último, pero no menos importante, agradezco a los profesores de la escuela de ingeniería electrónica por sus labor, en especial al profesor Oscar Reyes por su compromiso con este proyecto de investigación.

Agradezco especialmente a esa persona que siempre ha estado a mi lado desde el momento que coincidimos. Jamás olvidaré tu apoyo, amor y compañía, gracias Anita.

VÍCTOR

TABLA DE CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

Pág.

LISTA DE CUADROS

Pág.

LISTA DE ANEXOS

Pág.

RESUMEN

TÍTULO: DISEÑO E IMPLEMENTACIÓN DE UN GENERADOR DE NÚMEROS ALEATORIOS EN UN SISTEMA EMBEBIDO*.

AUTORES: JUAN DIEGO BOLAÑOS LANZZIANO, VÍCTOR MANUEL MONSALVE HERNÁNDEZ**.

PALABRAS CLAVES: Generador de números aleatorios, TRNG, fluctuación de fase aleatoria, muestreo coherente, prueba estadística de aleatoriedad, sistema embebido.

DESCRIPCIÓN:

En este trabajo se presenta el análisis, diseño, desarrollo, implementación y validación de un generador de números realmente aleatorios (TRNG). Se tienen como requisitos un reducido consumo de recursos, propiedades estadísticas adecuadas para aplicaciones criptográficas y una implementación completamente digital a través de los recursos disponibles en cualquier tipo de FPGA.

El análisis toma en cuenta los principales tipos de TRNG y dando un valor cuantitativo a las características más relevantes se presenta una tabla comparativa para facilitar la escogencia según los requerimientos establecidos.

El diseño escogido utiliza el fenómeno de fluctuación de fase aleatorio producido por dos anillos oscilantes (RO) como fuente de entropía y el método de muestreo coherente implementado en *hardware* para extraer los valores no determinísticos, pero con un alto nivel de sesgo, el cual es mitigado y corregido por medio de una buena práctica de post-procesado basado en una implementación en *hardware* de la función H con compuertas XOR.

El sistema es desarrollado utilizando lenguaje de descripción de *hardware* para circuitos integrados de muy alta velocidad (VHDL) y la herramienta de diseño ISE/PlanAhead. El diseño es implementado en un sistema embebido reconfigurable, específicamente una FPGA Spartan-3AN Starter Kit y se utiliza una transmisión serial para capturar los datos usando el *software* RealTerm.

El generador es validado a través de las pruebas estadísticas para generadores aleatorios y pseudo-aleatorios en aplicaciones criptográficas establecidas por el *National Institute of Standards and Technology* (NIST).

*Trabajo de Grado.

**Facultad de Ingenierías Fisicomecánicas, Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones, Director Dr.-Ing. Oscar Mauricio Reyes Torres, Codirector MSc. Héctor Iván Gómez Ortiz.

ABSTRACT

TITLE: DESIGN AND IMPLEMENTATION OF A RANDOM NUMBER GENERATOR IN EMBEDDED SYSTEM*.

AUTHORS: JUAN DIEGO BOLAÑOS LANZZIANO, VÍCTOR MANUEL MONSALVE HERNÁNDEZ**.

KEY WORDS: Random number generator, TRNG, random jitter, coherent sampling, statistical test for randomness, embedded system.

DESCRIPTION:

In this paper is presented the analysis, design, development, implementation and validation of a true random number generator (TRNG). It has requirements as a reduced consumption of resources, suitable statistical properties for cryptographic applications and a fully digital implementation with the available resources in any kind of FPGA.

The analysis reviews the main types of TRNG and giving a quantitative value to the most relevant features, a comparative table is presented to facilitate the choice according to the established requirements.

The chosen design utilizes the phenomenon of random phase jitter by two ring oscillator (RO) as entropy source and coherent sampling method implemented in hardware for extracting the non-deterministic values but with a high level of biased, which is mitigated and corrected by a good way of post-processing based on a hardware implementation of the H function with XOR gates.

The system is developed using very high speed integrated circuit hardware description language (VHDL) and the design tool ISE/PlanAhead. The design is implemented in a reprogrammable embedded system, specifically a Spartan-3AN FPGA Starter Kit and serial transmission is used to capture data using the RealTerm software.

The generator is validated with the statistical tests for random and pseudo-random generators in cryptographic applications established by the National Institute of Standards and Technology (NIST).

*Degree project.

**Physical-Mechanical Engineering Faculty, Electrical, Electronic and Telecommunications Engineering School, Director Dr.-Eng. Oscar Mauricio Reyes Torres, Codirector MSc. Héctor Iván Gómez Ortiz.

INTRODUCCIÓN

El presente libro tiene por objetivo presentar el trabajo de investigación realizado para implementar un generador de números realmente aleatorios (TRNG) en un sistema embebido y validar sus características estadísticas para aplicaciones criptográficas, utilizando pruebas diseñadas por una institución especializada.

El diseño realizado para este proyecto está dirigido principalmente, pero no exclusivamente, a cumplir los requerimientos establecidos por la línea de investigación en criptografía del grupo de investigación CPS, de la Universidad Industrial de Santander.

El libro está organizado en siete capítulos. Los cinco primeros son el contenido del documento y los dos últimos capítulos se centran en las conclusiones y recomendaciones.

En el primer capítulo, se describe el problema investigado, contextualizando la problemática identificada. Como punto de partida para conducir y verificar el desarrollo del proyecto de investigación, se presenta el alcance y los objetivos definidos en el plan de trabajo. Además, se indica el plan de trabajo y sus respectivos ajustes durante la ejecución.

En el segundo capítulo, se explica de manera general la teoría de los generadores de números aleatorios (RNG). Se profundiza en la estructura general de los generadores de números realmente aleatorios (TRNG), definiendo sus componentes principales y presentando los diseños investigados.

En el tercer capítulo, se presentan las generalidades de la prueba estadística para generadores de números aleatorios propuesto por el *National Institute of Standard*

and Technology (NIST), describiendo cada una de las pruebas y resumiéndolas finalmente, buscando facilitar el entendimiento de los resultados entregados.

En el cuarto capítulo, se muestra la metodología utilizada en el desarrollo del generador. El análisis presenta los requerimientos y parámetros tomados en cuenta para escoger el diseño de TRNG más apropiado. En el diseño se describe el funcionamiento de cada módulo y en el desarrollo se describen los elementos digitales internos que logran el objetivo planteado. Posteriormente, se presenta la implementación, mostrando la ubicación final de los recursos en cada módulo de alto nivel. Por último, se presentan las pruebas del NIST implementadas en software, las cuales son utilizadas para validar el generador.

En el quinto capítulo, se entregan los resultados de la implementación, analizando el consumo de recursos por el TRNG y el módulo completo necesario para las mediciones del proyecto. También se muestran los resultados de la prueba estadística en diferentes etapas del proceso, y se analizan haciendo una comparativa gráfica con los resultados de cada prueba.

En el sexto capítulo, se presentan las conclusiones, verificando el cumplimento de los objetivos, por medio de los resultados obtenidos en las diferentes versiones realizadas. En el séptimo capítulo se hacen algunas observaciones y recomendaciones para adaptar el diseño en un circuito integrado, que es la intención por parte del grupo de investigación. Además, se proponen alternativas de diseño y mejoras dependiendo de las características de la tecnología manejada.

En el libro no se anexan las fuentes utilizadas en la versión final del generador, por la extensión de hojas impresas que supone, sin embargo, en la sección 4.4 se comentan las consideraciones más relevantes al describir el diseño. Si el lector está interesado en verificar el funcionamiento de algún módulo, se puede comunicar con los autores para obtener más información sobre la descripción en hardware.

1. DESCRIPCIÓN DEL PROYECTO

1.1. IDENTIFICACIÓN DEL PROBLEMA

En criptografía, los generadores de números aleatorios son utilizados en tecnologías de seguridad para sistemas de comunicación, protección de la información y protocolos de autenticación, entre otros. Generar números aleatorios se puede considerar como una tarea sencilla, después de todo existe gran cantidad de eventos aleatorios que pueden ser utilizados para esta labor. Sin embargo, seleccionar la fuente de aleatoriedad es una actividad crítica que depende del entorno donde se especifica el sistema. Su elección puede afectar la seguridad interna al incorporar zonas de ruptura o introducir sesgos a los números generados. Además, el diseño de un RNG requiere de los conocimientos estadísticos para definir correctamente la fuente aleatoria del sistema, comprobar a través de pruebas estadísticas confiables, si el sistema en realidad produce un flujo de bits aleatorios y establecer cuáles son sus propiedades estadísticas.

La naturaleza determinística del *software* exige al diseñador investigar soluciones en *hardware* para producir verdaderos números aleatorios.

1.2. JUSTIFICACIÓN

El grupo de investigación CPS ha propuesto el desarrollo de una línea de investigación en el área de la criptografía. Por lo tanto, se hace necesario establecer una base de conocimiento relacionada con la generación de números aleatorios, y construir un TRNG basado en hardware completamente digital, teniendo en cuenta las características necesarias en los sistemas criptográficos.

La investigación sobre el estudio de TRNGs, utilizando las herramientas de búsqueda de información en las bases de datos suscritas y catálogo bibliográfico

disponible por la universidad, expone la falta de desarrollo y documentación sobre el tema a nivel local, lo que representa una situación y oportunidad muy importante para avanzar en el proceso de investigación de la criptografía.

1.3. OBJETIVOS

1.3.1. Objetivo general: implementar un generador de números aleatorios en un sistema embebido.

1.3.2. Objetivos específicos:

- Diseñar un generador embebido de números aleatorios verdaderos en un lenguaje de descripción como VHDL o Verilog, corrigiendo sus características hasta considerarlo realmente aleatorio.
- Implementar en un sistema de desarrollo FPGA disponible las progresivas versiones del generador, extrayendo y almacenando muestras de números para su comprobación estadística.
- Realizar en *software* las pruebas estadísticas del NIST o *diehard* para comprobar la aleatoriedad del dispositivo con base en métricas establecidas.
- Realizar la documentación de la implementación y utilización del generador en su versión final.

1.4. ALCANCE

Implementar un generador de números verdaderamente aleatorios, el cual debe entregar valores binarios sin un patrón predecible, basado en un proceso físico dentro de un sistema digital, por tanto, el sistema no debe requerir ninguna semilla o señal aleatoria externa.

Para la implementación se plantea utilizar un lenguaje descripción de hardware y realizar su configuración en un sistema embebido, específicamente en dispositivos reprogramables como una FPGA. Para la comprobación se proponen pruebas estadísticas basadas en software y reconocidas por la comunidad científica internacional (NIST o *diehard*).

1.5. PLAN DE TRABAJO

La propuesta de investigación aprobada, distribuye las actividades en 16 semanas de trabajo y cinco grupos de actividades, iniciando el 18 de Agosto y finalizando el 20 de diciembre de 2014 [1]. Sin embargo, por los compromisos académicos de los autores, se acuerda con el director y codirector del proyecto, mover las semanas del calendario, manteniendo el número de semanas y las actividades propuestas. El cronograma final queda pactado entre el 8 de enero y 22 de abril de 2015.

Además, para mejorar el seguimiento del diseño y las pruebas estadísticas, se reagrupan las actividades relacionadas según la metodología del capítulo 4 del presente documento.

2. GENERADORES DE NÚMEROS ALEATORIOS

2.1. GENERALIDADES

La seguridad en aplicaciones criptográficas depende en gran medida de la calidad de los números aleatorios, determinada por los valores de las propiedades estadísticas y el nivel de entropía del generador utilizado. Un nivel alto de entropía, secuencia de números aleatorios independientes e impredecibles y protección contra posibles ataques externos, son las principales características que se necesitan de un generador de números aleatorios (RNG).

Básicamente hay dos tipos de (RNGs): generador de números realmente aleatorios (TRNG) y generador de números pseudo-aleatorios (PRNG). Los TRNGs producen números aleatorios a partir de fenómenos físicos aleatorios y se caracterizan por su comportamiento estocástico, es decir, no se puede predecir el valor siguiente a partir de los valores anteriores. A diferencia de los TRNGs, los PRNGs son algoritmos desarrollados en sistemas determinísticos, que producen secuencias recurrentes definidas por la semilla utilizada en el algoritmo. Los RNGs además se diferencian por el tipo de implementación utilizada, que puede ser *software* o *hardware.* Los diseños implementados en *software* se caracterizan por su portabilidad y bajo costo. Las implementaciones en *hardware* normalmente son sistemas más seguros por su nivel de integración y la complejidad del diseño.

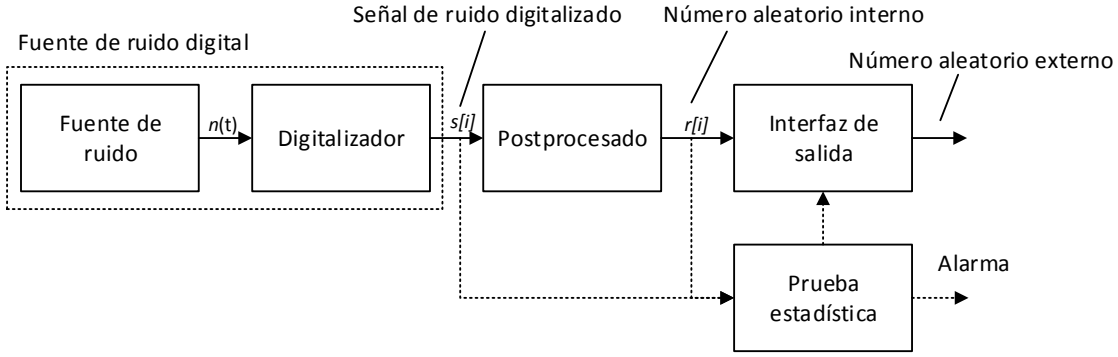
La necesidad de desarrollar generadores más robustos en sistemas criptográficos fomenta el estudio de diferentes fuentes de fenómenos aleatorios para utilizar TRNGs, fenómenos como el ruido resistivo, de unión PN, el ruido generado por conversores A/D, ruido de fase o *jitter*, meta-estabilidad en componentes biestables, fenómenos mecánicos como la vibración de una máquina en funcionamiento u otros como el ruido térmico o el decaimiento radioactivo.

Un TRNG digital implementado en *hardware* aplica un método seguro para extraer entropía de un fenómeno físico aleatorio presente en un elemento digital, modifica los bits obtenidos para mejorar las propiedades estadísticas y aumentar el nivel de entropía. Además de generar números aleatorios estadísticamente independientes e impredecibles, ofrece mayor seguridad al mantener todas las operaciones críticas en la generación de los números aleatorios al interior del dispositivo.

2.2. ESTRUCTURA GENERAL DE UN TRNG

Un TRNG básico consiste en una fuente de aleatoriedad (entropía) y un método de extracción. Generalmente las propiedades estadísticas medidas a la salida del extractor son deficientes para utilizar directamente el TRNG en aplicaciones criptográficas, porque se presenta baja entropía en la fuente de aleatoriedad, correlación o sesgo entre los bits. Por esta razón, es necesario emplear un algoritmo de post procesado como un bloque adicional a la salida del TRNG básico, para mejorar las propiedades estadísticas de los bits generados. Los TRNGs no están sujetos a un estándar; sin embargo, existen algunos esquemas propuestos para esta clase de sistemas, como el que se muestra en la figura 1 [2].

Figura 1. Concepto general de un TRNG



Fuente: adaptado de [2].

2.2.1. Fuente de entropía: la fuente de entropía o aleatoriedad de un TRNG, consiste en un fenómeno físico aleatorio del cual se puede obtener una señal con la tasa de entropía mínima necesaria para construir el generador con los parámetros de salida establecidos.

La entropía de una señal aleatoria es una medida de la información contenida en la señal. Se puede definir el nivel de entropía de una secuencia aleatoria de bits, como la tasa de convergencia mínima del número de bits necesarios, para reproducir la misma secuencia con la tasa de error más cercana a cero [3]. Para el límite, donde la longitud de una secuencia de valores aleatorios independientes y equitativamente distribuidos (IID) tiende a infinito, es virtualmente imposible obtener la secuencia exacta utilizando un número de bits menor que la entropía de la fuente.

En la teoría de la información de Shannon [4], se tiene una variable discreta aleatoria n que toma valores y_k con probabilidad q_k (k=1,2,…), descrita por:

$$\mathrm{P}(n = y_k) = q_k, \text{ tal que } q_k \geq 0 \ (k = 1,2, \dots) \ y \ \sum_{k=1}^{\infty} q_k = 1, \tag{1}$$

donde $P(A)$ denota la probabilidad del evento A.
Según Shannon, la entropía de n se puede expresar como $\mathrm{H}_0(n)$, cuyo valor se determina según:

$$\mathrm{H}_0(n) = -\sum_{k=1}^{\infty} q_k \log_2 q_k. \tag{2}$$

El valor máximo de entropía de un proceso X con dos posibles resultados equiprobables, según la ecuación anterior, es $\mathrm{H}_0(X) = 1$. Este resultado permite afirmar que en un generador de bits aleatorios el valor máximo de entropía que se puede alcanzar es 1, es decir, cada bit generado tiene la misma posibilidad de estar en cualquiera de los dos estados lógicos independientemente del estado lógico del bit anterior o sucesor.

Para el caso particular de los dispositivos digitales, que están diseñados para implementar procesos determinísticos, un proceso de comportamiento estocástico y con alto nivel de entropía no está contemplado. Sin embargo, la tecnología base de estos dispositivos está constituida por circuitos analógicos, cuyo funcionamiento se rige por procesos físicos, de modo que, pese al empeño por minimizarlos, es inevitable encontrar sucesos impredecibles debidos a la naturaleza física de la tecnología base.

La fluctuación del retardo en compuertas lógicas, el comportamiento meta-estable de ciertos elementos y el ruido térmico dentro del dispositivo son los fenómenos comúnmente usados como fuente de aleatoriedad por TRNGs digitales [5].

La variabilidad de la duración del retardo ocasionado por las compuertas lógicas, provoca una ligera desviación del tiempo exacto del cruce de umbral de una señal digital periódica. Esta desviación temporal, también conocida como *jitter*, da lugar a la variación del transcurso real de la señal con respecto a su trayecto ideal. Existen métodos para provocar la presencia de *jitter* en una señal digital, como la implementación de anillos oscilantes (ROs) [6] o de sistemas de sintetización de señales como PLLs (Phase-Locked Loops) [7] o DLLs (Delay-Locked Loops) [8].

Un RO consiste en la configuración de elementos de retardo encadenados formando un lazo cerrado [6]. Los ROs usan compuertas lógicas como inversores y buffers para constituir los lazos con la condición de incluir al menos un inversor por lazo. Los elementos de retardo utilizados introducen un retardo debido al elemento y un retardo debido a la conexión entre ellos.

Los factores involucrados en el retardo de las compuertas y la conexión entre ellos están descritos por las ecuaciones:

$$d_i = D_i + \Delta d_i = D_i + \Delta d_{Li} + \Delta d_{Gi}, \tag{3}$$

$$r_i = R_i + \Delta r_i = R_i + \Delta r_{Li} + \Delta r_{Gi}, \tag{4}$$

donde,

- d_i, r_i: retardos del elemento i-ésimo y de la conexión del elemento i con el elemento $i + 1$.
- $\Delta d_i, \Delta r_i$: variación del retardo (*jitter*).
- D_i: retardo nominal y constante del elemento i correspondiente al valor nominal de tensión de alimentación y temperatura de operación.
- R_i: retardo constante de la conexión.
- $\Delta d_{Li}, \Delta r_{Li}$: *jitter* causados por eventos físicos locales.
- $\Delta d_{Li}, \Delta r_{Li}$: *jitter* introducido por perturbaciones aleatorias como fuentes de ruido y fluctuaciones de las condiciones de funcionamiento del dispositivo.

El movimiento aleatorio de los electrones en un metal debido a la temperatura, definido como ruido térmico, es un suceso inherente al funcionamiento de un dispositivo electrónico, caracterizado por ser inevitable y uniformemente distribuido en el espectro de frecuencias. La principal limitación de utilizar el ruido térmico como fuente de aleatoriedad de un TRNG digital, es la complejidad del mecanismo para extraer directamente una señal de un elemento digital, que herede el comportamiento aleatorio del ruido térmico, sin utilizar elementos o señales externas.

La meta-estabilidad es un estado de operación impredecible en los *flip-flops*, ocurre cuando la señal de entrada cambia justo en el momento en el cual un flanco de reloj habilita su operación. El fabricante define el tiempo mínimo que debe mantenerse fija la entrada, antes y después del instante donde ocurre un flanco de reloj, y el tiempo que toma el *flip-flop* para la transición de estado de su salida. La violación de estos tiempos impone un funcionamiento erróneo del circuito causando el estado particular de meta-estabilidad.

2.2.2. Método de extracción: la fuente de entropía define de qué forma debe construirse la etapa de extracción para obtener una señal aleatoria. Cada fuente de entropía tiene su propio método de extracción, aunque es común encontrar un mecanismo de muestreo cuando se trata de fuentes digitales de aleatoriedad. Este mecanismo debe ser habilitado por la fuente de entropía para muestrear el proceso aleatorio y obtener una señal cuyo valor sea impredecible. Para mejorar la calidad de la señal, la fuente de aleatoriedad debe indicar en qué momento el nivel de entropía es suficientemente alto para realizar un mejor muestreo. El método de extracción y la fuente de entropía deben garantizar que para la señal de salida x la distribución de probabilidad sea uniforme (o similar), no se presente sesgo en el muestreo y que la autocorrelación $R_x(\tau)$ tenga un pico elevado en $\tau = 0$ y valores cercanos a cero sin patrón alguno para cualquier otro valor de τ.

2.2.3. Post-procesamiento: generalmente las características estadísticas a la salida del extractor no son suficientemente altas para utilizar el TRNG en aplicaciones criptográficas, porque el fenómeno físico presente en la fuente de aleatoriedad tiene ciertas falencias que degradan la calidad estadística de los números aleatorios generados. Las falencias más comunes de los fenómenos físicos utilizados son: un nivel insuficiente de entropía y la existencia de autocorrelación de la señal extraída. También se puede obtener valores muy bajos de entropía por aplicar el mecanismo de extracción de manera incorrecta.

Se recomienda, utilizar algoritmos para mejorar las propiedades estadísticas de los números aleatorios generados, para contrarrestar los efectos negativos de las falencias encontradas en las etapas de fuente y extracción de aleatoriedad. Aún en el caso de extraer correctamente la aleatoriedad de una fuente de entropía adecuada, sigue siendo recomendable aplicar un algoritmo de post procesado antes de usar la salida del TRNG.

Los algoritmos de post procesado tienen como propósito reducir el sesgo del muestreo, disminuir la correlación entre los bits generados y mejorar la distribución de probabilidad de la señal aleatoria extraída. Este tipo de algoritmos suelen disminuir el volumen de trabajo o *throughput* del generador, para aumentar la entropía de cada bit. Por el contrario, algunos algoritmos de post procesado se emplean para incrementar el *throughput*. Estos últimos generalmente son utilizados por PRNGs. Los algoritmos criptográficos usados para encriptar información también pueden ser usados en la etapa de post procesado, sin embargo utilizan funciones que son relativamente complejas y costosas para implementar.

Las técnicas más comunes de post procesado, toman un número n de bits de entrada, que se asumen son bits aleatorios estadísticamente independientes, con un número finito de sesgos, para producir un bit de salida con un número aceptable de sesgos, de valores muy pequeños.

Probablemente, el método más sencillo es la operación de la compuerta or-exclusiva (XOR) de n bits. Se puede observar la tabla de verdad de la compuerta XOR con $n = 2$ en la tabla 1. La principal ventaja de un corrector basado en XOR es que su latencia está acotada.

Tabla 1. Tabla de verdad: XOR

Entrada A	Entrada B	Salida
0	0	0
0	1	1
1	0	1
1	1	0

El corrector de Von Neumann es un método útil para remover ciertos sesgos presentes en un RNG, pero no más eficiente que otros métodos. Debido a la lógica de su algoritmo, el corrector de Von Neumann puede descartar hasta el 75% de

todos los bits de entrada, cuando la entrada sea perfectamente aleatoria. El algoritmo opera sobre un par de bits, como se indica en la tabla 2.

Tabla 2. Tabla de verdad: corrector de Von Neumann

Entrada A	Entrada B	Salida
0	0	Descartada/Nula
0	1	0
1	0	1
1	1	Descartada/Nula

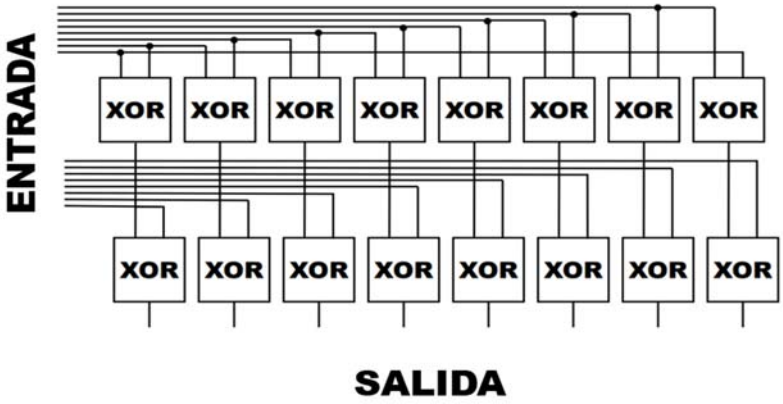
La desventaja principal es que la latencia de este corrector no está acotada o definida para todo momento, es decir, la tasa de bits producidos cambia inevitablemente con la secuencia a la entrada del corrector.

Aunque ambos métodos cumplen la función de eliminar algunos sesgos y disminuir otros, tienen restricciones si los bits aleatorios generados por el RNG no son todos estadísticamente independientes. Un ejemplo de ello es una cadena de bits de entrada de la forma “1010101010…” que para la XOR de orden 2 y el corrector de Von Neumann produce a la salida la secuencia “11111…”.

Otros dos métodos de post procesado más elaborados son la función H y S [9], en las cuales se utiliza un arreglo de compuertas XOR, y las entradas de varios bytes generados son utilizadas para generar un único byte. Una implementación en *hardware* de la función H se muestra en la figura 2, el arreglo utiliza una entrada de 16 bits, divididos en 2 bytes, para realizar la función XOR del primer byte con los datos sucesivos, desde el primer bit con el segundo, hasta el último bit con el primero. Posteriormente, la salida de estas compuertas se vuelve a utilizar como una entrada de otra compuerta XOR, esta vez aplicada uno a uno con el segundo byte. Por otro lado la función S, es una mejora de la función H que utiliza 32 bits en su entrada, mejorando la corrección de entropía pero desaconsejada por el alto consumo de recursos.

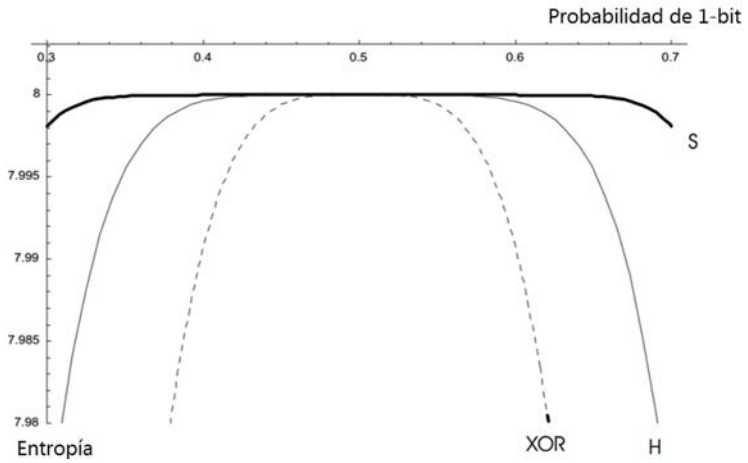
En la figura 3 se observa una gráfica comparativa del resultado esperado de entropía para el byte de salida, según la probabilidad que tengan los bits generados. Dependiendo del sesgo de la fuente de entropía se puede escoger una implementación adecuada para el generador.

Figura 2. Función de post-procesado H



Fuente: adaptada de [9].

Figura 3. Corrección de entropía para función XOR, H y S según probabilidad



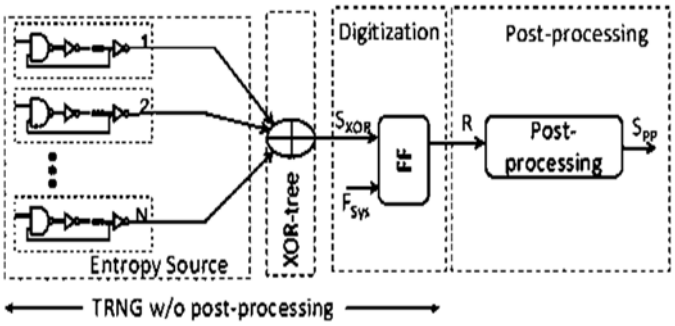
Fuente: adaptada de [9].

2.3. DISEÑOS DE TRNGS

Algunos diseños prácticos de TRNGs digitales implementados en *hardware,* encontrados en la documentación, son presentados y brevemente explicados, destacando sus ventajas y desventajas.

2.3.1. Diseño basado en anillos oscilantes: el diseño de un TRNG de anillos oscilantes como el propuesto en [6], utiliza el principio de ruido de fase o *jitter* de una señal digital creada por un lazo de elementos combinacionales (compuertas lógicas), que incluye al menos un inversor en el lazo. La oscilación de la señal creada por el lazo, tiene un tiempo de propagación establecido por la componente determinística de los retrasos de propagación, en cada una de las compuertas lógicas empleadas. Cada compuerta lógica también agrega un componente aleatorio, conocido como *jitter*, el cual es responsable de la fluctuación aleatoria del tiempo de propagación de la señal. El número de elementos de retraso y negadores se determina de acuerdo a la frecuencia y el *jitter* deseados. El esquema propuesto de este diseño se muestra en la figura 4.

Figura 4. Anillos oscilantes



Fuente: adaptado de [6].

Un conjunto de anillos oscilantes se conecta a una compuerta lógica de múltiples entradas, que se encarga de fijar una señal aleatoria de alta frecuencia. Esta señal

es muestreada por un reloj de baja frecuencia, a través de un *flip-flop* tipo D. El principal reto de este diseño es conseguir que cada anillo oscilante sea independiente con respecto a los otros.

Las ventajas destacables de este diseño son:

- Construido por elementos básicos (compuertas lógicas y *flip-flops*) que permite diseñar e implementar sobre cualquier FPGA de forma sencilla.
- Rendimiento relativamente alto y constante.

Sus principales desventajas son:

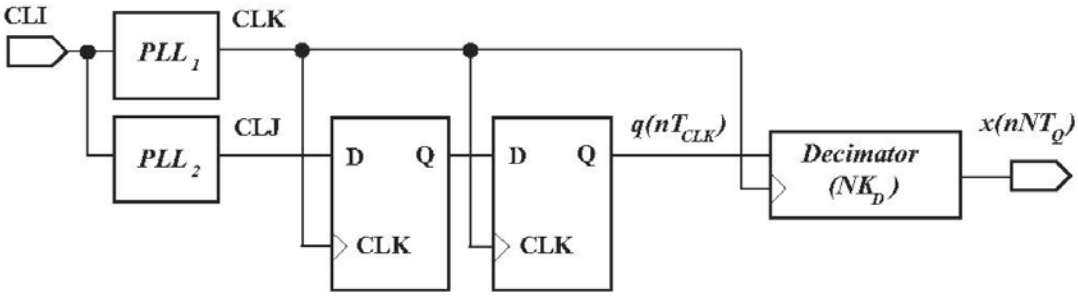
- Alto consumo de recursos físicos y energía, debido al gran número de anillos y elementos por anillo utilizados.
- Dificultad para disminuir la correlación entre los anillos oscilantes, que limita la calidad de la aleatoriedad en la salida del generador.
- Vulnerabilidad ante posibles amenazas externas, causada por la falta de robustez en el mecanismo de extracción.

2.3.2. Diseño basado en lazos de seguimiento de fase: el diseño de un TRNG basado en PLL como el propuesto en [7], básicamente extrae entropía del *jitter* ligado a la señal de reloj de salida de un lazo de seguimiento de fase (PLL). El objetivo de este método es establecer las frecuencias adecuadas para las señales de reloj de entrada y salida del PLL, para que el generador pueda realmente extraer un elemento aleatorio a partir de *jitter.* El principio básico de extracción es muestrear una señal de reloj afectada por *jitter* utilizando un *flip-flop* síncrono.

El esquema propuesto de este diseño se muestra en la figura 5. Se observan dos PLLs, encargados de generar las dos señales de reloj utilizadas en la etapa de muestreo, compuesta por dos *flip-flops* tipo D organizados en cascada, con el

propósito de disminuir la probabilidad de producir una salida meta-estable, causada por la minimización de la distancia entre los flancos de las señales de reloj. El siguiente módulo se encarga de tomar la salida del segundo *flip-flop*, que hasta el momento es de un solo bit de ancho, y convertirlo en un vector de n bits que puede ser procesado por una etapa de corrección de datos o directamente utilizado como salida del generador.

Figura 5. Lazos de seguimiento de fase



Fuente: tomado de [7].

Las ventajas destacables de este diseño son:

- No requiere post procesado.
- Diseño e implementación sencillos.
- Bajo consumo de energía.

Sus principales desventajas son:

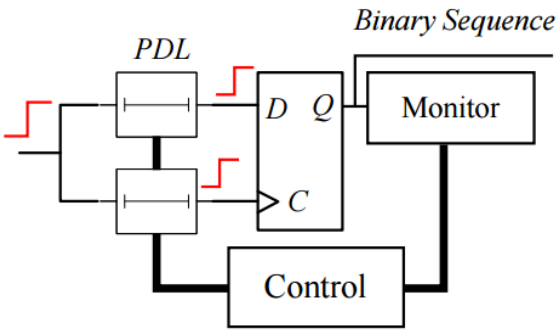
- Baja velocidad de generación de bits aleatorios.
- El uso de PLLs limita la cantidad de dispositivos donde puede estar implementado.

2.3.3. Diseño basado en meta-estabilidad de flip-flops: el diseño de un TRNG basado en la meta-estabilidad de *flip-flops* como el propuesto en [10], induce a un *flip-flop* a operar en condición meta-estable. La operación meta-estable de un *flip-flop*, desafía el esfuerzo de los diseñadores de circuitos digitales para resolver problemas de meta-estabilidad en sus dispositivos. El método para generar esta condición utiliza un circuito electrónico de retardo de línea, para alterar el tiempo de propagación de manera controlada [11].

Se usa este circuito para ajustar el tiempo preciso de llegada de la señal al *flip-flop*, con la capacidad de ajustar el retardo de la señal con una resolución de picosegundos. El diseño contempla un mecanismo de realimentación para afinar el circuito de retardo a un valor adecuado, cuando se detecte sesgo en la salida del generador. Este ajuste induce nuevamente al *flip-flop* a entrar en operación meta-estable. El mecanismo ofrece mayor seguridad al generador cuando las condiciones externas cambian.

El esquema propuesto para este diseño se muestra en la figura 6. El circuito de retardo actúa sobre un *flip-flop* tipo D, cuya salida es analizada para determinar si se presenta sesgo en la salida y aplicar el ajuste correspondiente.

Figura 6. Meta-estabilidad de *flip-flops*



Fuente: tomado de [10].

Las ventajas destacables de este diseño son:

- Alta velocidad de generación de bits aleatorios.
- Bajo consumo de energía.
- Robusto ante cambios de las condiciones externas.

Sus principales desventajas son:

- Se requiere un circuito de retardo de línea, no disponible en cualquier FPGA.
- No recomendable para algunos modelos de FPGA.
- Dificultad para asegurar la operación apropiada del generador.

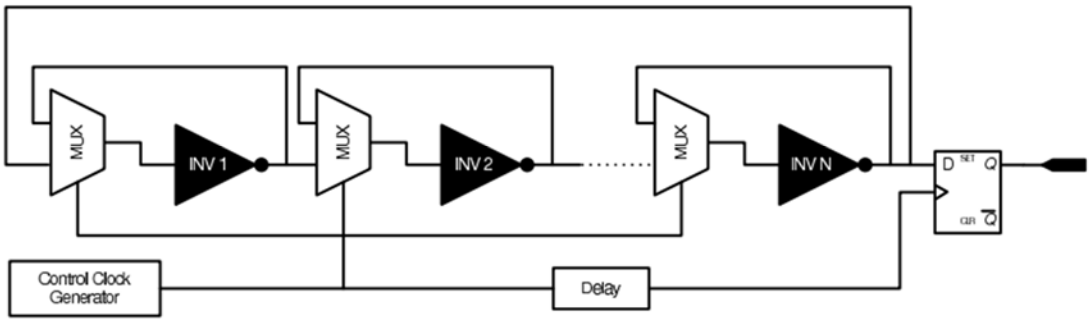
2.3.4. Diseño basado en meta-estabilidad con anillos oscilantes: el diseño de un TRNG basado en meta-estabilidad utilizando anillos oscilantes como el propuesto en [12], extrae entropía de la condición meta-estable de un dispositivo digital afectado por *jitter*. Este método busca reducir el tiempo necesario para extraer entropía e incrementar la velocidad de generación. El componente principal de este diseño es un anillo oscilante con la capacidad de establecer su operación en la región meta-estable.

El esquema del TRNG propuesto se muestra en la figura 7. El anillo oscilante está compuesto por un lazo de inversores y cada inversor tiene su propio interruptor, que permite conectar la entrada del inversor con su respectiva salida, cuando el interruptor opera en estado alto, o conectar la salida del inversor anterior con la entrada del inversor correspondiente, cuando opera en el estado bajo.

El generador puede controlar el cambio de estado de todos los interruptores asociados a los inversores del anillo oscilante. Entonces, el generador tiene dos configuraciones de operación según la posición de los interruptores. Cuando todos los interruptores individuales están en alto y los inversores forman, cada uno, un

lazo cerrado, el generador entra en operación meta-estable. El voltaje de la señal en cada inversor fluctúa sobre el nivel meta-estable debido al ruido térmico, mientras el interruptor se mantenga en ese estado. Cuando los interruptores cambian, los inversores forman un solo lazo cerrado, cuyo valor inicial depende por completo de la fluctuación que ocurrió en cada inversor.

Figura 7. Meta-estabilidad con anillos oscilantes



Fuente: tomado de [12].

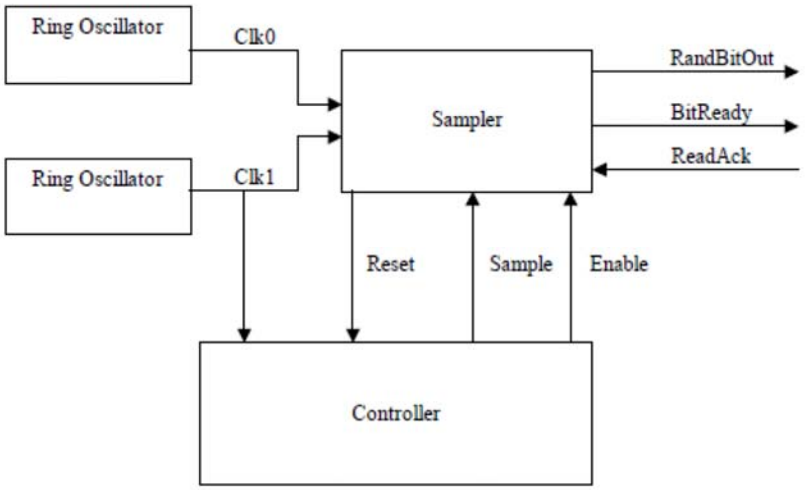
2.3.5. Diseño basado en muestreo coherente: el diseño de un TRNG basado en muestreo coherente como el propuesto en [13], establece como fuente de entropía el ruido de fase aleatorio contenido en la señal producida por un anillo oscilante, que se extrae con el mecanismo de muestreo coherente. Este diseño puede producir bits aleatorios a un velocidad de hasta 0.5 Mbits/s con las características estadísticas adecuadas. Un aspecto importante del generador es que solo utiliza elementos digitales básicos, que pueden encontrarse en cualquier clase de FPGA.

El esquema general del generador con muestreo coherente mostrado en la figura 8, está compuesto por dos anillos oscilantes, muestreador y controlador. Los anillos oscilantes entregan dos señales de reloj al muestreador, que se encarga de extraer bits aleatorios con el apoyo del controlador.

El muestreo coherente es un método que toma dos señales periódicas y muestrea una de ellas utilizando el periodo de la otra. Los periodos de ambas señales deben ser ligeramente diferentes.

El proceso más crítico del diseño es la generación de las dos señales de reloj, con frecuencias muy cercanas pero no idénticas, para obtener la cantidad suficiente de entropía. Es recomendable que la diferencia entre los periodos de los dos relojes esté en el orden de las decenas de pico segundos. Por lo tanto, se requiere hacer dentro de la FPGA, la ubicación manual de los componentes que emplea cada anillo oscilante, que satisfaga la apreciación de las frecuencias de reloj.

Figura 8. Muestreo coherente



Fuente: tomado de [14].

Las ventajas destacables de este diseño son:

- Construido por recursos disponibles en cualquier FPGA.
- Modelo matemático realizable.
- Bajo consumo de recursos físico y energía.

Sus principales desventajas son:

- Grado de dificultad mayor, debido a la necesidad de ubicar manualmente los elementos.
- Requiere post procesado, para reducir el sesgo en los datos causado por el muestreo.

3. PRUEBAS ESTADÍSTICAS PARA GENERADORES ALEATORIOS

3.1. GENERALIDADES

Para utilizar un RNG en aplicaciones criptográficas se requiere la validación de sus características aleatorias, haciendo necesario tener una serie de pruebas que corroboren las propiedades estadísticas aleatorias de la secuencia de bits generados. Existen diferentes agrupaciones de pruebas (denominadas paquetes o baterías) propuestas por grupos especializados en el análisis de generadores PRNG y TRNG, los cuales normalmente entregan algoritmos diseñados para comprobar o refutar una hipótesis con base en las características deseables o indeseables determinadas.

Dos de las pruebas más extendidas en la documentación científica son el paquete de pruebas estadísticas para generadores aleatorios y pseudo-aleatorios en aplicaciones criptográficas establecidas por el *National Institute of Standards and Technology* (NIST) [15], diseñado específicamente para aplicaciones criptográficas y secuencias de longitud moderada (menor a un millón de bits), y la batería intransigente (*diehard*) de pruebas de aleatoriedad desarrolladas por George Marsaglia [16], considerada más exigente, general y recomendándose para secuencias de longitud mayores a un millón de bits.

En este trabajo de investigación, debido al enfoque hacía aplicaciones criptográficas y la extensa documentación proporcionada para el entendimiento de la prueba, se decide realizar la validación del generador con el paquete de pruebas estadístico del NIST.

3.2. PAQUETE DE PRUEBAS ESTADÍSTICO DEL NIST

Contenido en la publicación especial 800-22, revisión 1a [15], propone 15 pruebas estadísticas empíricas para los generadores de números aleatorios utilizados en criptografía.

En cada prueba se plantea una hipótesis nula, al realizar la prueba se realiza un contraste de hipótesis calculando el valor-p y suponiendo que la hipótesis nula es cierta. El número se considera aleatorio si la hipótesis nula es aceptada, es decir si el valor-p asociado al resultado es mayor al 1% (0.01). El resumen de resultados final muestra la proporción de aprobación para cada prueba, que debe superar un umbral determinado por la prueba, según el número de bits ingresados.

Para facilitar el análisis y entendimiento de los resultados, las siguientes subsecciones describen el principio de funcionamiento, propósito y criterio de aceptación de cada una de las pruebas estadísticas aplicadas. Sin embargo, no se profundiza sobre las funciones matemáticas o algoritmos utilizados por cada prueba, los cuales se encuentran descritos minuciosamente en la documentación entregada al público por el NIST [15].

3.2.1. Prueba de frecuencia: mide la proporción de unos y ceros en toda la secuencia, determinando si se aproxima al valor esperado en una secuencia aleatoria, es decir, verifica si posee el mismo número de unos y ceros. Es la prueba más básica y debe ser la primera en aprobarse. Además se recomienda que la secuencia sea de al menos 100 bits.

3.2.2. Prueba de frecuencia en bloque: en bloques de M-bits generados a partir de la secuencia de datos de n-bits, mide la proporción de bits con valor uno, a cada bloque se le calcula su proporción la cual debería tender a M/2. M debe ser mayor a 20 y a n/10, el valor máximo aconsejado para M es de 100.

3.2.3. Prueba de corridas: mide el número total de corridas en una secuencia, donde la corrida es una serie ininterrumpida de bits idénticos, el propósito de la prueba es determinar si el número de corridas de unos y ceros de varias longitudes es el esperado en una secuencia aleatoria. Específicamente la prueba determina si la oscilación ente ceros y unos es demasiado rápida o demasiado lenta. Se aconseja un valor n de 100 bits.

3.2.4. Prueba de largas corridas de unos en un bloque: mide las largas corridas de unos en bloques de M-bits generados a partir de la secuencia de datos de n-bits, se determina si la longitud de la corrida más larga de unos es consistente con lo esperado en una secuencia aleatoria. La prueba sólo toma en cuenta los 1, porque una irregularidad en la longitud de las cadenas de unos implica una irregularidad en la longitud de las cadenas de ceros.

3.2.5. Prueba de rango para matrices binarias: determina el rango de sub-matrices disyuntivas generadas a partir de la secuencia completa. El propósito de esta prueba es comprobar la dependencia lineal entre cadenas de longitud fija de la secuencia original. No cumplir la regla de decisión indicaría dependencia lineal en las cadenas generadas.

3.2.6. Prueba espectral o de la transformada discreta de Fourier: el objetivo de esta prueba es medir los picos de la transformada discreta de Fourier en la secuencia, buscando detectar características periódicas (patrones repetitivos que están cerca uno del otro), si el número de picos que superan el umbral de 95% es significativamente diferente al 5% del total de picos. La n mínima es de 1000 bits.

3.2.7. Prueba de plantillas coincidentes no solapadas: detecta el número de apariciones de cadenas pre-especificadas (plantillas). El propósito de la prueba es detectar generadores que producen patrones muy frecuentemente, pero dentro de secuencia no periódicas. El algoritmo genera una ventana de M bits con un patrón

inicial y recorre la secuencia de datos, si el patrón se encuentra, la ventana se restablece el bit siguiente al patrón que encontrado en la secuencia y la búsqueda se reanuda contabilizando el número de apariciones, en caso de no encontrarse o terminar la búsqueda, se aumenta el bit y el proceso se repite. En caso de encontrar demasiadas coincidencias para una misma secuencia la hipótesis nula sería falsa.

3.2.8. Prueba de plantillas coincidentes solapadas: similar a la anterior detecta el número de apariciones de plantillas pre-especificadas. La diferencia entre las dos pruebas es que el algoritmo reinicia la búsqueda del patrón un bit después del primer bit en el patrón coincidente, pudiendo determinar coincidencias que comparten bits con la anterior coincidencia (solapamiento).

3.2.9. Prueba "estadística universal" de Maurer: el objetivo de esta prueba es medir el número de bits entre los patrones coincidentes (una medida que está relacionada con la longitud de una secuencia comprimida). El propósito de la prueba es detectar si la secuencia puede ser significativamente comprimida sin pérdida de información. Las secuencias altamente comprimibles no se consideran aleatorias.

3.2.10. Prueba de complejidad lineal: se utiliza el algoritmo de Berlekamp-Massey para determinar la complejidad lineal de los N bloques en que se divide la secuencia de datos, este valor indica la longitud necesaria en un registro de desplazamiento con retroalimentación lineal (LFSR) para generar la secuencia. LFSR largos indicarán secuencias aleatorias.

3.2.11. Prueba de series: se mide la frecuencia de aparición de todos los patrones posibles de M-bits solapados en la secuencia de datos, la probabilidad de cada patrón en una secuencia aleatoria debería ser idéntica, en el caso de M ser igual a 1, esta prueba tendría el mismo funcionamiento de la prueba de frecuencia para bits.

3.2.12. Prueba de la entropía aproximada: al igual que la prueba de series, el objetivo de esta prueba es medir la frecuencia de aparición de todos los patrones posibles de M-bits solapados en la secuencia de datos, sin embargo en este caso se compara las frecuencias en dos bloque consecutivos (M y M+1), verificando que su probabilidad sea la esperada en una secuencia aleatoria.

3.2.13. Sumas acumulativas: se calcula la excursión máxima (desde cero) al recorrer la secuencia sumándolos los valores (-1,+1) según los dígitos en la secuencia. Se busca determinar si la suma de las secuencias parciales es demasiado alta o pequeña según el comportamiento esperado en una secuencia aleatoria. Para una secuencia aleatoria la excursión del recorrido debería ser cero.

3.2.14. Prueba de excursión aleatoria: se contabiliza el número de ciclos con exactamente K visitas en un valor de suma acumulada durante el recorrido aleatorio. Un ciclo de recorrido aleatorio consiste en una secuencia de pasos de longitud tomadas al azar que comienzan y finalizan en el origen. El propósito de esta prueba es determinar si el número de visitas a un estado particular dentro de un ciclo se desvía de lo que cabría esperar de una secuencia aleatoria.

3.2.15. Prueba de variación en la excursión aleatoria: se mide el número total de veces que se visitó un estado particular en una suma acumulada de recorrido aleatorio. El propósito de esta prueba es detectar desviaciones del número de visitas a un estado durante varios recorridos aleatorios. Esta prueba es en realidad una serie de dieciocho pruebas, siguiendo la secuencia -9,8,…,-1 y 1,2,…,9.

3.2.16. Resumen de pruebas estadísticas del NIST: en la tabla 3 se listan las pruebas estadísticas del NIST en español e inglés y se describe la falla que detecta cada una.

Tabla 3. Resumen de pruebas estadísticas del NIST

Pruebas estadísticas	**Detecta**
Frecuencia (*frequency*)	Proporción no uniforme de 1's o 0's.
Frecuencia en bloque (*block Frequency*)	Proporción no uniforme de 1's o 0's en bloques de M-bits.
Corridas (*runs*)	Gran (pequeño) número de oscilaciones en la cadena de bits
Largas corridas de unos en bloque (*longest runs of ones in a block*)	Gran (pequeño) número de oscilaciones en bloques de bits
Rango de matrices binarias (*binary matrix rank*)	Dependencia lineal debida a la periodicidad.
Transformada discreta de Fourier (*discrete Fourier transform*)	Características periódicas en la cadena de bits.
Plantillas coincidente no solapadas (*non-overlapping template matching*)	Muchas ocurrencias de un patrón no periódico.
Plantillas coincidente solapadas (*overlapping template matching*)	Muchas ocurrencias de un patrón no periódico entre bloques.
"Estadística universal" de Maurer (*Maurer's "universal statistical"*)	Compresibilidad
Complejidad lineal (*linear complexity*)	Poca complejidad para reproducir en un LFSR
Series (*serial*)	Proporción no uniforme en bloques de M-bits.
Entropía aproximada (*approximate entropy*)	Periodicidad por baja entropía.
Sumas acumulativas (*cumulative sums*)	Demasiados 1's o 0's continuos.
Excursión aleatoria (*random excursions*)	Desviación en el número de visitas a un estado en un recorrido aleatorio.
Variación en la excursión aleatoria (*random excursions variant test*)	Desviación en el número de visitas a un estado en varios recorridos aleatorio

4. METODOLOGÍA

4.1. MODELO

Se realiza una adaptación del modelo de desarrollo en cascada iterativo [17], para definir las etapas que conforman el proyecto. La metodología se escoge por su amplia utilización en el desarrollo de *software* y la similitud de este último con el *hardware* reprogramable [18]. Las fases en que se descompone el proyecto se observan en la figura 9, y son definidas así:

- **Análisis**: se estudian los requerimientos y características deseadas, para seleccionar el sistema que mejor se adapte a las necesidades, según la documentación recopilada.

- **Diseño**: se determina la estructura general de los diferentes módulos que componen el sistema, buscando que el conjunto logre la función deseada.

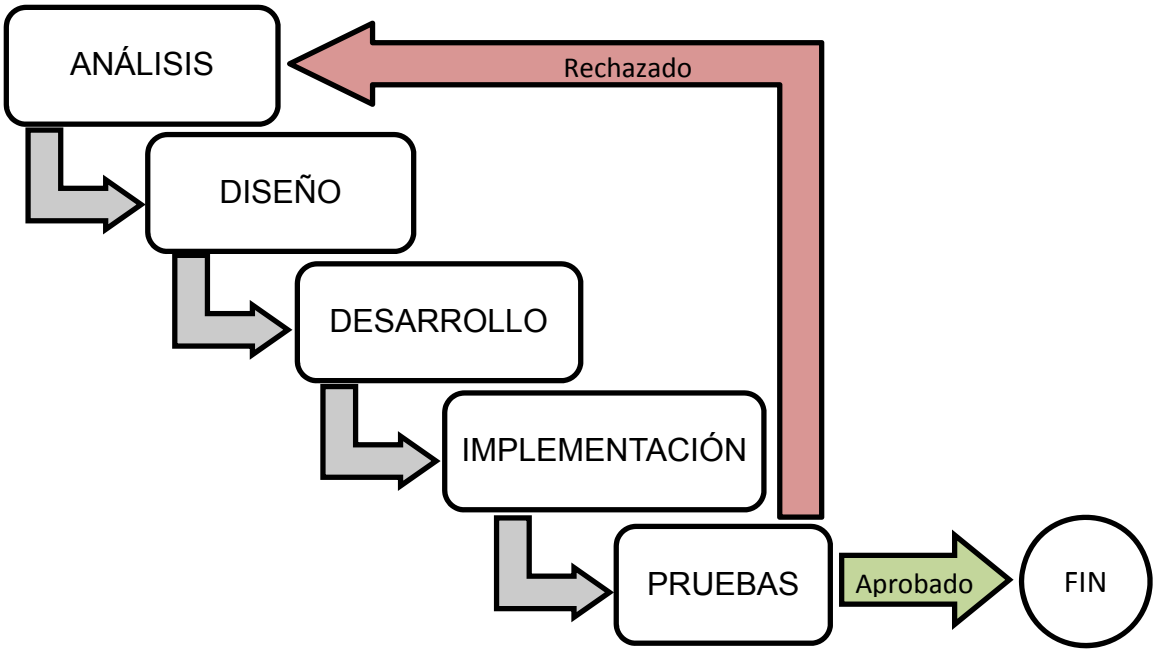
- **Desarrollo**: cada módulo es definido en detalle y se realiza la descripción de *hardware* respectiva, registrando las consideraciones a tomar en cuenta en la implementación.

- **Implementación**: se examina el desempeño de la implementación del circuito dentro de la FPGA y se verifica el cumplimiento de las condiciones establecidas por las etapas anteriores.

- **Pruebas**: se utiliza un banco de pruebas para comprobar la validez de las características estadísticas aleatorias de los datos generados.

En este capítulo se presenta la documentación de la iteración final y funcional del TRNG.

Figura 9. Modelo de desarrollo en cascada iterativo adaptado



4.2. ANÁLISIS

Según las consideraciones iniciales, el RNG debe utilizar como fuente de aleatoriedad un fenómeno físico interno, no debe requerir de circuitos o señales externas y debe estar basado en *hardware* digital. Por tanto se delimita el área de estudio a los TRNGs, los cuales cumplen con las características anteriormente mencionadas.

También se tienen varios requerimientos específicos para el diseño según las necesidades del grupo de investigación, descritas a continuación:

- Posibilidad de ser configurado en cualquier FPGA, buscando facilitar en un futuro su adaptación a un sistema no reprogramable.

- Consumo de recursos físicos reducido, que permita integrar otros circuitos dentro del mismo chip.
- Altos valores de calidad aleatoria y robustez del generador, comprobado por las pruebas correspondientes.

Con base en estos requisitos se definen unos parámetros que faciliten el análisis de los diferentes diseños configurables en una FPGA, permitiendo dar una valoración a cada uno y determinar cuál corresponde al diseño idóneo para el caso específico tratado. Los parámetros escogidos son cuatro y se definen así:

- Configurable en cualquier FPGA.
- Cantidad de recursos requeridos.
- Calidad aleatoria.
- Robustez

También se definen cuatro parámetros que determinan otras características deseables pero no indispensables en el diseño.

- Simplicidad del diseño: facilita el diseño, desarrollo e implementación.
- Síntesis automatizada: no es necesario considerar el posicionamiento de los elementos durante la implementación.
- Post-procesamiento: aunque consume más recursos, puede ser necesario para mejorar las características aleatorias de los bits.
- Volumen de trabajo (*throughput*): Su valor determina las aplicaciones criptográficas donde se puede utilizar el generador.

Los generadores considerados son cinco: anillos oscilantes, lazo de seguimiento de fase (PLL), *flip-flop* meta-estable, oscilador de anillos meta-estable y muestreo coherente, los cuales son descritos y referenciados en la sección 2.3.

El análisis realizado se resume en la tabla 4, de donde se determina que el generador por muestreo coherente es el idóneo, pues cumple con los requisitos establecidos. Sin embargo, tiene como desventaja la necesidad de posicionamiento manual (síntesis no automatizada) y la necesidad de corrección de características aleatorias por medio de post-procesamiento.

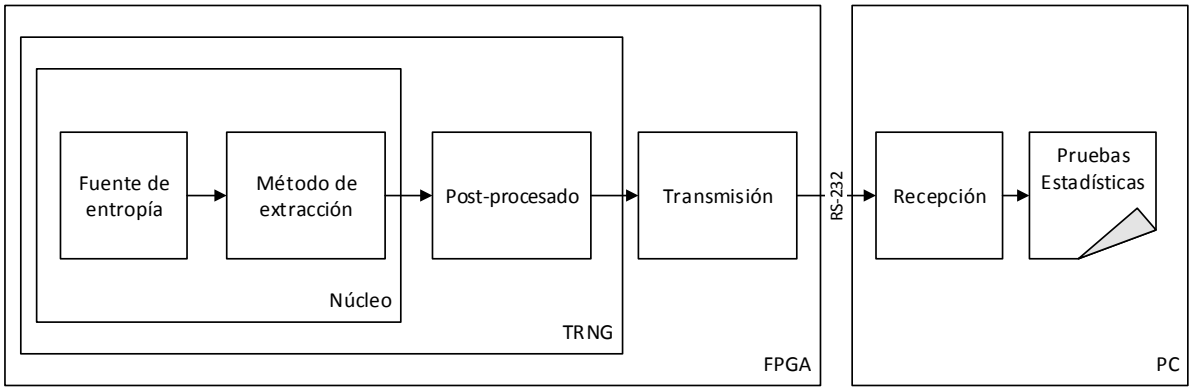
Tabla 4. Comparativa de características en los TRNGs analizados

	Anillos Oscilan.	**PLL**	**FF Meta-estable**	**Osc. Meta-estable**	**Muestr. Coher.**
Requisitos					
Configurable en cualquier FPGA	Sí	No	No	Sí	Sí
Cantidad de recursos requeridos	Alta	Baja	Baja	Baja	Baja
Calidad aleatoria	Alta	Alta	Alta	Alta	Alta
Robustez	Baja	Alta	Alta	Baja	Alta
Características deseables					
Simplicidad del diseño	Alta	Alta	Baja	Baja	Alta
Síntesis automatizada	Sí	Sí	No	No	No
Requiere post-procesado	Sí	No	No	No	Sí
Rendimiento (*throughput*)	Alto	Bajo	Alto	Alto	Alto

4.3. DISEÑO

En esta sección se explica con mayor detalle el funcionamiento del TRNG basado en muestreo coherente descrito en la subsección 2.3.5. La figura 10 expone la estructura principal de este TRNG. La fuente de entropía y el método de extracción conforman el núcleo del TRNG. A su vez, el núcleo junto con el módulo de post-procesado son los componentes principales del TRNG. Los bits generados se envían al computador a través de una interfaz RS-232, utilizando el controlador UART, presente en el módulo de transmisión.

Figura 10. Diagrama de bloques para el muestreo coherente

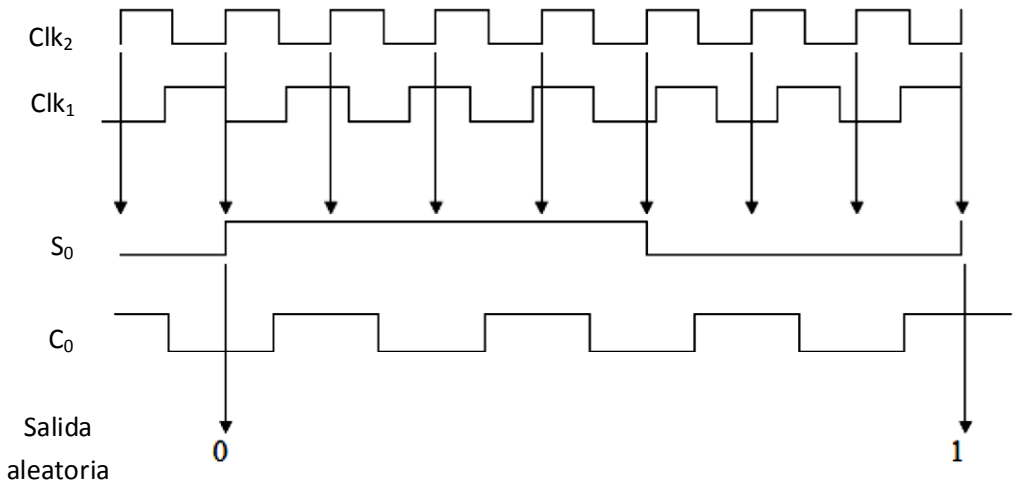


La fuente de entropía emplea dos anillos oscilantes para crear dos señales de reloj clk_1 y clk_2, con periodos cercanos pero no idénticos. La diferencia de los periodos de los dos relojes debe ser del orden de las decenas de picosegundos, para aumentar el desempeño en el proceso de extracción de entropía. Es muy importante asegurar el cumplimiento de esta condición, pues el funcionamiento del generador depende de la relación entre ambos periodos [19]. Para garantizar que las frecuencias de las dos señales generadas por los anillos oscilantes sean lo más cercanas posibles, deben ubicarse en posiciones adyacentes dentro del chip, estar físicamente aislados del sistema y su organización interna debe ser simétrica una respecto a la otra.

Para extraer la entropía, el generador utiliza el mecanismo de muestreo coherente, el cual muestrea la señal clk_1 con la señal clk_2 para obtener una señal envolvente S_o. La señal S_o posee largas secuencias de 1s o 0s debido a la cercanía entre los periodos de las señales de reloj. El periodo de esta señal es inversamente proporcional a la diferencia de los periodos de las señales clk_1 y clk_2. Además, el periodo de la señal S_o es un múltiplo entero del periodo de clk_2. Sin embargo este periodo contiene un número aleatorio de ciclos de reloj clk_2 debido al estado impredecible que toma la señal clk_1 cuando el circuito muestrea. Entonces, el bit aleatorio se puede obtener contando el número de periodos de clk_2 que pasan durante un periodo de la señal S_o.

Por simplicidad, el método de extracción utiliza un *flip-flop* tipo T como contador del número de ciclos de clk_2. El contador es de módulo dos y su salida se nombra como C_0. Entonces, el extractor obtiene un bit aleatorio a partir del muestreo de la señal C_0 por cada flanco de subida de la señal S_o. El diagrama de tiempos del proceso de extracción para dos bits aleatorios se muestra en la figura 11.

Figura 11. Diagrama de tiempos: extracción para dos bits aleatorios



Fuente: adaptado de [19].

Para evitar problemas de meta-estabilidad, las señales C_0 y S_o se desfasan medio ciclo de reloj. Para ello, el contador de ciclos opera con el flanco descendente del reloj clk_2 mientras el muestreo de clk_1se realiza por cada flanco ascendente.

Para prevenir correlación entre bits sucesivos, el mecanismo de muestreo puede reiniciar el contador antes de cada nueva generación de bits, por medio de una señal R_0. El circuito incluye un *flip-flop* cuya salida se pone en alto por cada nuevo bit generado y permite ser limpiado por una señal R_0. Tanto este *flip-flop* de aviso como el *flip-flop* usado para el muestreo de C_0 usan el flanco ascendente de la señal S_0. Ambos *flip-flops* se conectan a una señal de habilitación E_0 con el propósito de mantener el bit aleatorio sin cambios mientras es leído por el generador. Incluso, es necesario deshabilitar por un tiempo predeterminado el muestreo de la señal C_0, después de un primer flanco ascendente de S_0, porque pueden ocurrir varios cambios de estado en C_0 que alteran el bit que se desea leer.

Al igual que los anillos oscilantes, los elementos utilizados en la etapa de extracción deben ser ubicados manualmente para garantizar que las señales involucradas se encuentren lo más cercanas posibles y estén aisladas de los demás circuitos.

El módulo de post-procesado tiene como función leer las señales de salida del método de extracción y controlar que muestree correctamente. Este módulo gestiona las señales de habilitación del muestreo y de reinicio del contador, cada vez que se genera un bit aleatorio. Para generar un bit aleatorio, el módulo habilita el muestreo y espera por la señal que le indique cuando hay un nuevo bit aleatorio. Cuando ocurre, la señal de aviso se pone en alto y el módulo deshabilita el muestreo y procede a leer el bit aleatorio. Después de leerlo, espera un tiempo predeterminado para habilitar nuevamente el muestreo, reiniciar el contador y regresar al estado de espera inicial.

Mientras tanto, un circuito serie-paralelo almacena los bits leídos para obtener un vector de 16 bits que ingresa al corrector función H para incrementar la calidad de las propiedades estadísticas disminuyendo los sesgos en los bits. Finalmente la salida del generador produce un flujo de bytes aleatorios procesados y una señal de pulsos por cada byte generado. El flujo de datos se almacena y espera por su transmisión hacia el computador.

4.3.1. Fuente de entropía: este módulo se encarga de generar dos señales de reloj afectadas por *jitter* con periodos casi idénticos, pero no iguales, utilizando anillos oscilantes. El *jitter* de la señal generada por un anillo oscilante está relacionada con el número de elementos de retardo y negadores que forman el lazo cerrado. Como un anillo oscilante debe tener al menos un negador para funcionar [20], se establece el uso de un único negador para disminuir a una sola variable, la cantidad de elementos utilizados por cada lazo cerrado.

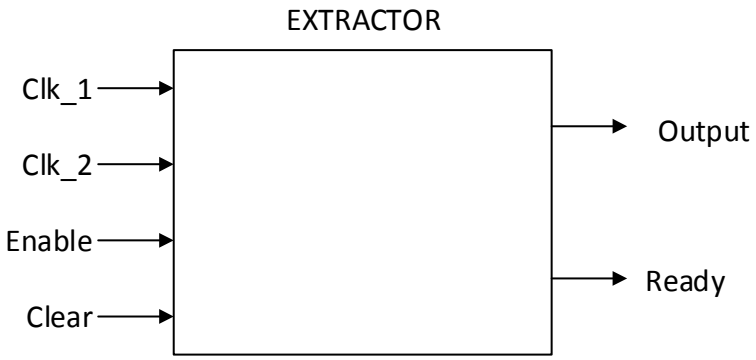
El esquema del módulo se muestra en la figura 12. La fuente de entropía debe asegurar un método para detener la generación de las señales oscilantes, cuando el generador cambie al estado de *reset*. También se debe garantizar que las dos señales de salida tengan la forma de una señal de reloj para no obtener resultados inesperados en la etapa de muestreo.

Figura 12. Entidad: fuente de entropía

4.3.2. Método de extracción: este módulo utiliza las dos señales de reloj creadas por el módulo fuente de entropía para generar la señal que contiene el bit aleatorio y la señal que indica cuando está listo el bit para ser leído. Además, dos señales ingresan a este bloque para habilitar o deshabilitar el muestreo y reiniciar el contador y la bandera de aviso.

El esquema de esta etapa se muestra en la figura 13. Al igual que el módulo anterior, los elementos utilizados por el método de extracción deben ubicarse manualmente dentro de la FPGA y estar aislados de los elementos que no correspondan al núcleo del generador.

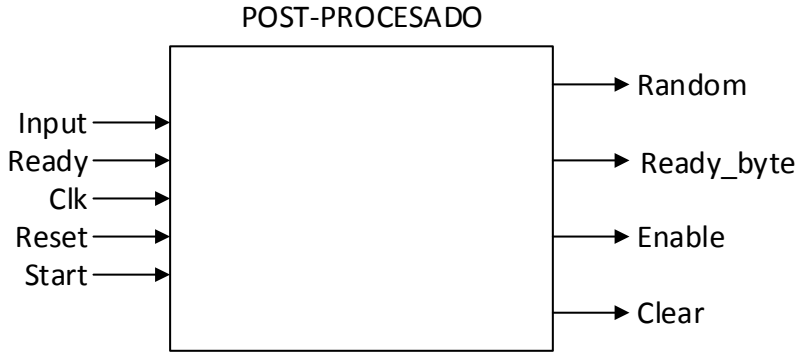
Figura 13. Entidad: extractor



4.3.3. Post-procesado: este módulo cumple múltiples funciones dentro del TRNG, una de ellas es leer los valores generados por el módulo del método de extracción y gestionar su funcionamiento a través de las señales de habilitación y de reinicio conectadas entre estos dos módulos. Además, utiliza un circuito para convertir de serie a paralelo, los datos aleatorios obtenidos. Por último, utiliza la función H sobre los vectores de datos obtenidos, para entregar a la salida del generador un flujo de bytes aleatorios corregidos.

El esquema del módulo se muestra en la figura 14. En total, son cinco entradas que corresponden a las dos señales de salida hacia el extractor, la señal de reloj y la señal de reinicio del sistema junto con la señal que activa la generación del TRNG. El post-procesado produce una señal de salida denominada *Ready_byte*, para indicar cuando hay un nuevo dato en la señal llamada *Random,* que corresponde al flujo de bytes aleatorios corregidos.

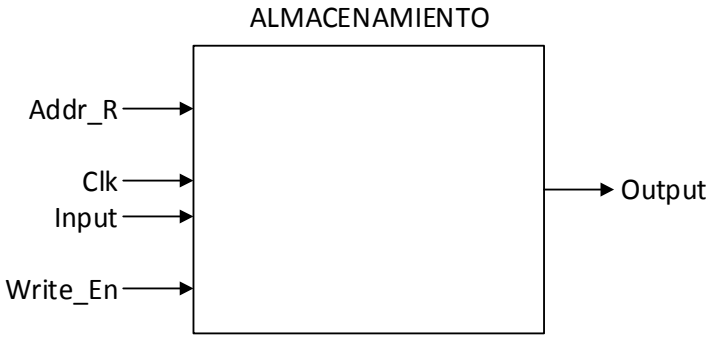
Figura 14. Entidad: post-procesado



4.3.4. Almacenamiento: este módulo se encarga de almacenar los bytes generados por el TRNG en memoria RAM, de forma que cada nuevo byte ocupa un espacio de la memoria hasta que se llene. Cuando la memoria RAM está llenada por completo, una señal debe habilitar el módulo de transmisión, permitiendo al usuario iniciar el proceso de envío de los datos al computador.

El esquema de la memoria utilizada se muestra en la figura 15. Dentro del módulo, un contador se encarga de recorrer todas las posiciones de memoria, aumentando una posición cada vez que la señal *Write_En* de entrada se pone en alto, hasta alcanzar el valor máximo, donde la escritura se deshabilita. Se propone utilizar los recursos de memoria RAM disponibles en la FPGA, que para la tarjeta de desarrollo utilizada (Spartan-3AN Starter Kit), corresponde a 20 módulos de almacenamiento RAM.

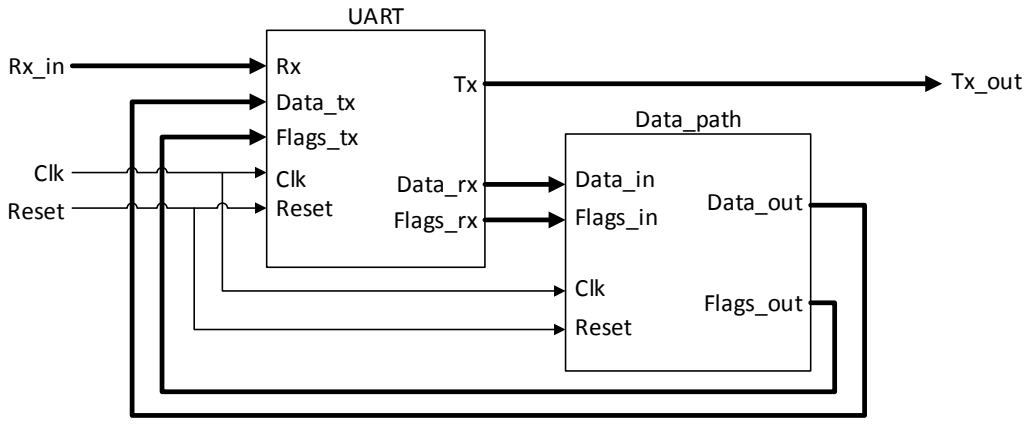
Figura 15. Entidad: almacenamiento



4.3.5. Transmisión: este módulo utiliza un controlador UART para realizar la transmisión serial de datos entre la FPGA y el computador. La codificación serial utilizada corresponde a un bit de inicio, un bit de parada y no incluye bit de paridad. Se utiliza el puerto RS-232 disponible en la FPGA para comunicar los datos.

El esquema utilizado por el controlador UART se muestra en la figura 16. Para el TRNG, solo es de interés utilizar el controlador UART como un bloque funcional para transmitir los datos.

Figura 16. Entidades: transmisión

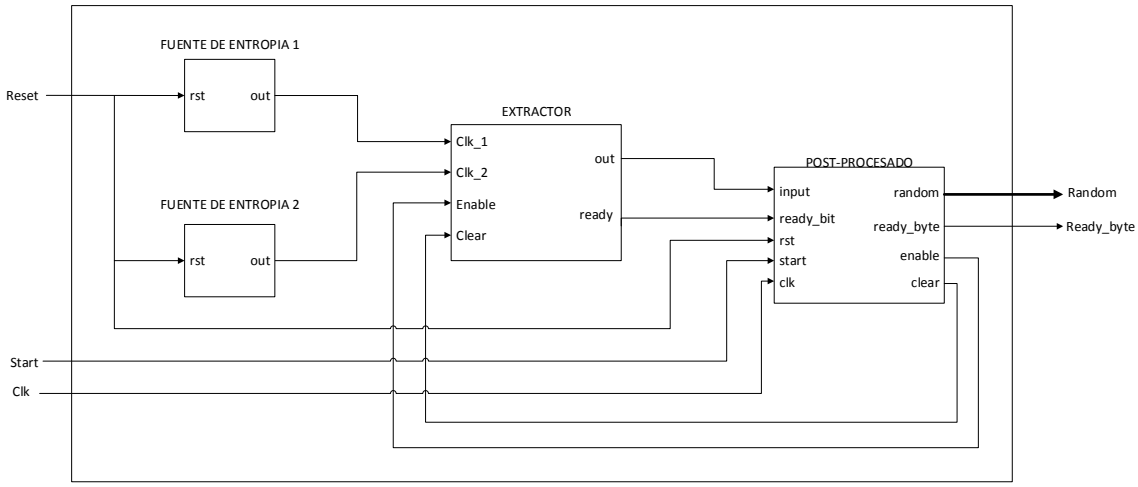


El TRNG y el módulo de almacenamiento se instancian dentro del módulo Data_Path, que se encarga del proceso de lectura y escritura de datos en el módulo UART. El controlador UART crea la interfaz de comunicación entre el Data_Path y el computador, recibiendo las instrucciones del computador con destino al generador y transmitiendo los datos almacenados. El funcionamiento interno del controlador UART no concierne al desarrollo de este documento.

4.4. DESARROLLO

Cada módulo que conforma el TRNG se implementa en VHDL cumpliendo lo descrito por la etapa de diseño. En la figura 17 se observa el esquemático de alto nivel del proyecto implementado para el TRNG, señalando los módulos que corresponden al núcleo del generador y las conexiones con el módulo de post-procesado. El núcleo del generador extrae los bits aleatorios mientras el post-procesado se encarga de gestionar el funcionamiento del núcleo y leer los bits aleatorios para corregirlos y proporcionar el flujo de bytes de salida del generador.

Figura 17. Esquemático: TRNG basado en muestreo coherente



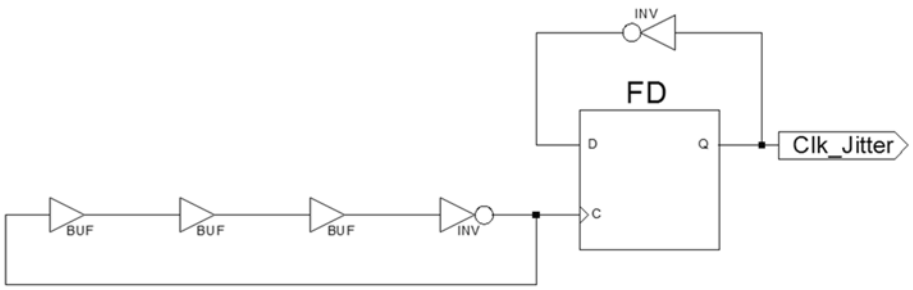
La implementación de cada módulo que conforma el TRNG, el módulo de almacenamiento y el sistema de transmisión se explican detalladamente en las siguientes subsecciones.

4.4.1. Fuente de entropía: el esquemático de la implementación del anillo oscilante de la figura 18, es el mismo para los dos anillos oscilantes necesarios en este diseño. El anillo oscilante, compuesto por un lazo cerrado de un único inversor y elementos de retardo, se conecta a la entrada de reloj de un *flip-flop* tipo T, construido a partir de un *flip-flop* tipo D y un inversor. El número óptimo de elementos en el lazo cerrado es 4, establecido a partir del tiempo de retardo de un LUT (cuadro 1) y la restricción de no generar bits a una tasa mayor que la frecuencia del sistema, en la salida del extractor. Para el valor definido, la frecuencia de Clk_Jitter es de 97,2 MHz y la tasa máxima de salida en el extractor es igual a 48.6 Mbps.

Cuadro 1. Reporte de síntesis: restricciones de tiempos.

Cell:in->out	fanout	G Delay	N Delay	Logical Name
LUT2:I1->O	1	0.643	0.000	delay_lut

Figura 18. Esquemático: fuente de aleatoriedad



Los anillos oscilantes son circuitos asíncronos no comúnmente disponibles en diseños básicos sobre una FPGA. La implementación en VHDL de cada anillo se debe realizar de forma especial porque la herramienta de síntesis o de mapeo omite automáticamente los elementos de retardo utilizados. Para evitar esto, se

construyen los anillos oscilantes en formato esquemático, usando elementos primitivos y asignando el valor verdadero al atributo “KEEP”, para restringir la eliminación de las conexiones involucradas en el circuito.

Cada retardo e inversor que conforma el anillo oscilante tiene una entrada adicional para la señal de *reset*. Estos elementos son construidos a partir de LUTs. Básicamente un LUT es una tabla que almacena un valor de salida por cada combinación posible de las cuatro variables de entrada. Para definir el funcionamiento lógico del LUT se utiliza el atributo “INIT”. El valor asignando al atributo “INIT” corresponde a los valores de salida, agrupados de a cuatro, de la tabla de verdad del LUT. Por ejemplo, el único inversor presente en el lazo cerrado debe poner la salida en cero, sólo cuando la señal de *reset* esté en cero y la señal de reloj del anillo en uno; en los demás casos la salida de este inversor debe permanecer en uno. Por lo tanto, el valor del atributo “INIT” del LUT correspondiente al inversor mencionado, debe estar fijado en el valor hexadecimal “B”, como se muestra en el cuadro 2.

Cuadro 2. Atributo INIT

```
attribute INIT : string;
attribute INIT of inv : label is "B";
```

En el cuadro 3 se muestra un ejemplo del código empleado para asignar un valor al atributo “KEEP” de una señal, para evitar que esta señal sea suprimida por las herramientas automáticas del entorno de trabajo.

Cuadro 3. Atributo KEEP

```
attribute KEEP : string;
attribute KEEP of buf1_signal : signal is "true";
```

La ubicación manual de los elementos del anillo oscilante, se realiza utilizando el atributo “LOC” para especificar el *slice* que se decide emplear dentro de la FPGA.

El código para la ubicación de los elementos que conforman el anillo oscilante uno, se muestra en el cuadro 4.

Cuadro 4. Atributo LOC

```
attribute LOC : string ;
attribute LOC of buf1     : label is "SLICE_X14Y11";
attribute LOC of buf2     : label is "SLICE_X14Y11";
attribute LOC of buf3     : label is "SLICE_X15Y11";
attribute LOC of inv_ring : label is "SLICE_X15Y11";
attribute LOC of inv_ff   : label is "SLICE_X16Y11";
attribute LOC of ff       : label is "SLICE_X16Y11";
```

El atributo “BEL” es utilizado para relacionar cada elemento del circuito con el elemento correspondiente dentro del *slice*. Por ejemplo, en el código del cuadro 5 se establece que el elemento “buf1” debe corresponder al LUT llamado “F”, dentro del *slice* donde se ubica el elemento.

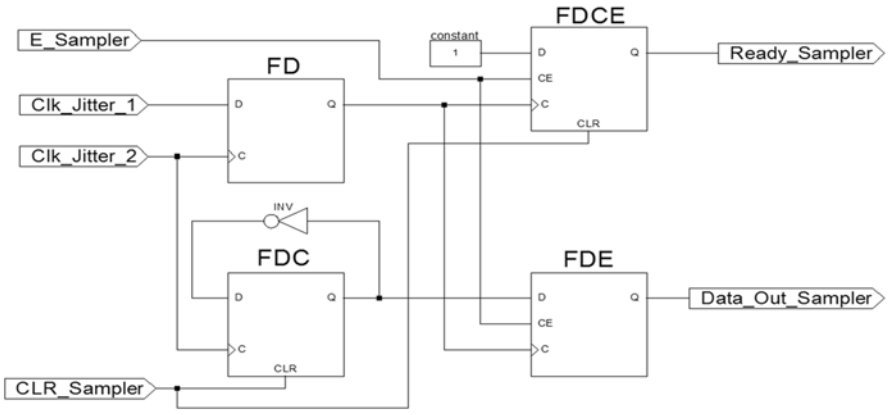
Cuadro 5. Atributo BEL

```
attribute BEL : string;
attribute BEL of buf1 : label is "F";
```

4.4.2. Método de extracción: el método de extracción se implementa en VHDL utilizando cuatro *flip-flops* tipo D, como se observa en el esquemático de la figura 19. Los *flip-flops* conectados a las señales de salida del módulo pueden ser habilitados por la señal *E_Sampler*.

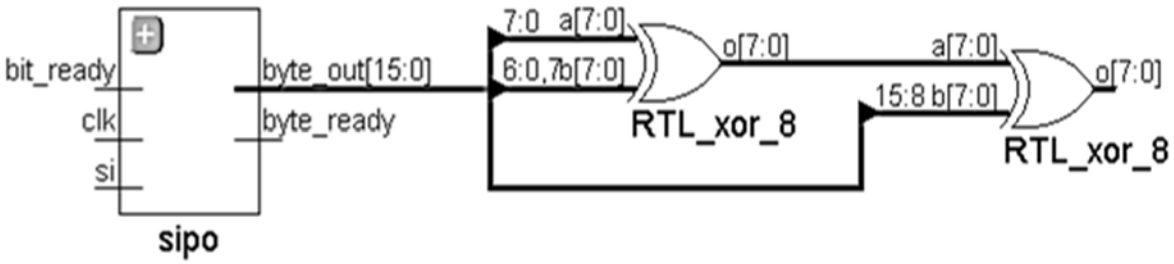
El *flip-flop* adaptado para contar los ciclos de reloj de *Clk_Jitter_2* y el *flip-flop* de la señal de aviso *Ready_Sampler* permiten ser reiniciados por la señal *CLR_Sampler.* Al igual que la fuente de entropía (ver subsección 4.4.1), el método de extracción utiliza los atributos “BEL” y “LOC” para posicionar y asignar los elementos del circuito dentro de la FPGA.

Figura 19. Esquemático: extractor



4.4.3. Post-procesado: el esquema de alto nivel del módulo de post-procesado, está compuesto por tres partes fundamentales: una máquina de estados finitos (FSM), un circuito serie-paralelo y un circuito para la función H, como se observa en la figura 20.

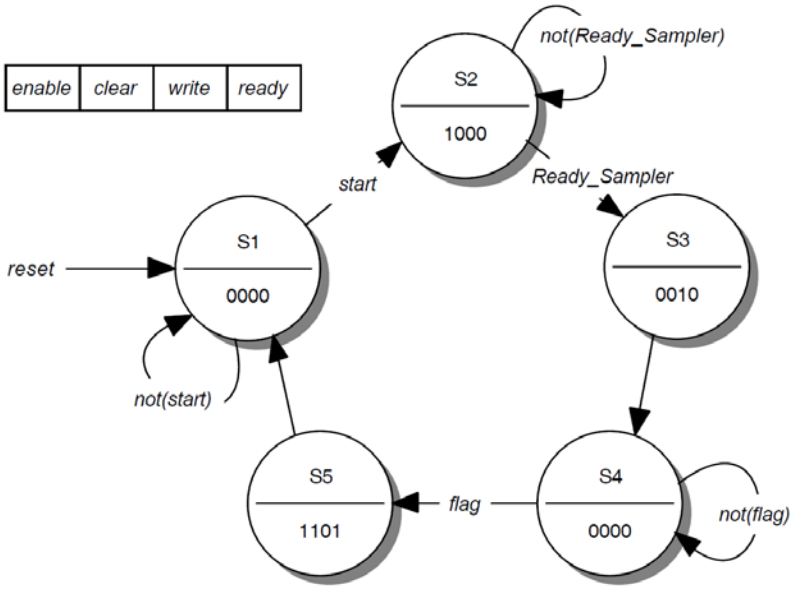
Figura 20. Esquemático: post-procesado



La máquina de estados registra las señales de salida del extractor y lo gestiona para que opere correctamente. Cada vez que la señal *active* se pone en alto, la máquina de estados pasa del estado de espera S_1 a un estado de operación S_2, donde permanece hasta que el extractor, a través de la señal *Ready_Sampler,* indica que hay un nuevo bit aleatorio listo para ser leído. En el siguiente estado (S_3), el circuito le ordena al *Datapath* que lea la señal *Data_Out_Sampler*, que contiene el bit aleatorio, y deshabilita el muestreo del extractor.

Del estado S_3 salta al estado S_4, en el que la máquina de estados espera por la señal *flag*, que se activa cuando un contador de ciclos alcanza un tiempo predeterminado. Después de esperar el tiempo predeterminado, la máquina de estados reinicia los *flip-flops* del extractor y habilita el muestreo. El diagrama de estados se presenta en la figura 21.

Figura 21. Máquina de estado: post-procesado



El circuito *Datapath* se encarga de leer el bit aleatorio y registrarlo en la entrada del circuito serie-paralelo, cada vez que se lo indique la máquina de estados. Además, genera la señal que advierte al circuito serie-paralelo cuando hay un nuevo bit aleatorio.

El circuito serie-paralelo lee el bit aleatorio cuando la máquina de estados le señala que hay un nuevo bit, almacena hasta obtener 16 bits y construye 2 bytes a partir de ellos. Los 2 bytes ingresan al circuito de función H, que opera sobre ellos para obtener 1 byte aleatorio corregido. El código utilizado para implementar la función H en VHDL, se muestra en el cuadro 6.

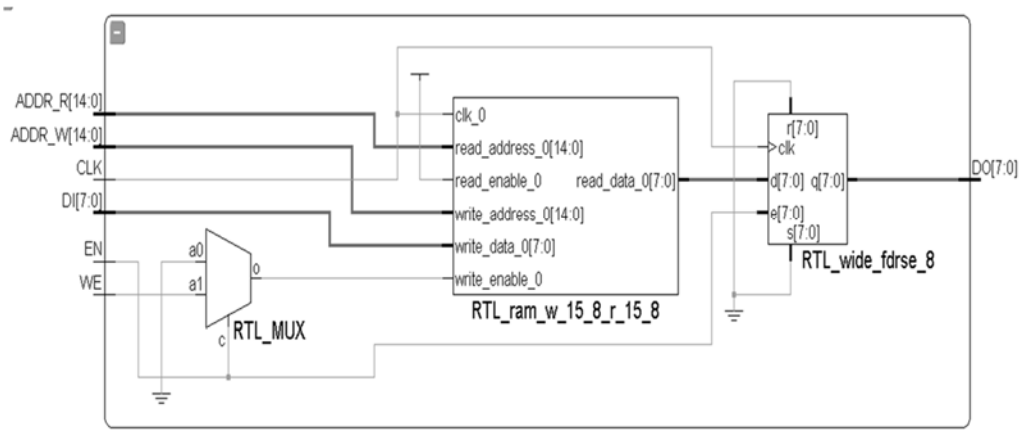
Cuadro 6. Función H

```
out <= (in(7 downto 0)xor(in(6 downto 0) & in(7)))xor in(15 downto 8);
```

4.4.4. Almacenamiento: el módulo de almacenamiento se construye a partir de la instanciación de bloques de memoria RAM *dual-port*. Para realizar las pruebas, se requiere almacenar un número mayor de 387.840 bits por secuencia [15]. Por lo tanto, se utilizan 16 módulos de memoria RAM de la FPGA Spartan-3AN Starter Kit, para almacenar 32678 bytes (261.424 bits) aleatorios y construir una secuencia admisible, a partir de dos generaciones seguidas.

Este módulo almacena los bytes generados por el TRNG en los bloques de memoria RAM habilitados hasta ocupar todas sus direcciones, utilizando el puerto de escritura. A su vez, permite al módulo de transmisión leer los bytes almacenados a través del puerto de lectura de la memoria RAM.

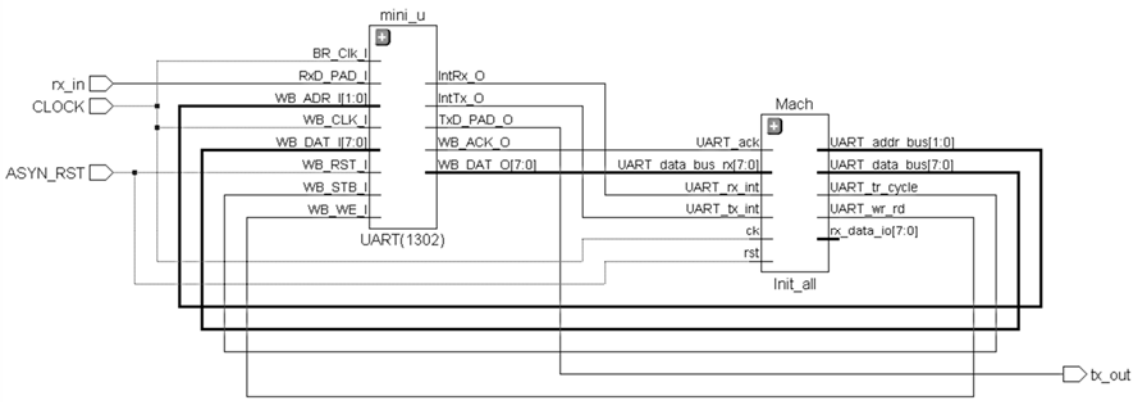
Figura 22. Esquemático: almacenamiento



4.4.5. Transmisión: para la comunicación serial utilizada en la transmisión de los datos al computador, se utiliza el módulo UART de bajo consumo de recursos,

propuesto en el sitio web opencore.org[1], descrito en VHDL y adaptado para transmitir los datos almacenados en memoria RAM, al recibir el carácter “1” en ANSI, desde cualquier emulador de terminal en el computador. Los datos se transmiten a una tasa de 9600 baudios, con caracteres de 8 bits, un bit de parada y sin bit de paridad, ni control de flujo por *hardware*. En la figura 23 se observa el esquemático del módulo de transmisión utilizado.

Figura 23. Esquemático: transmisión

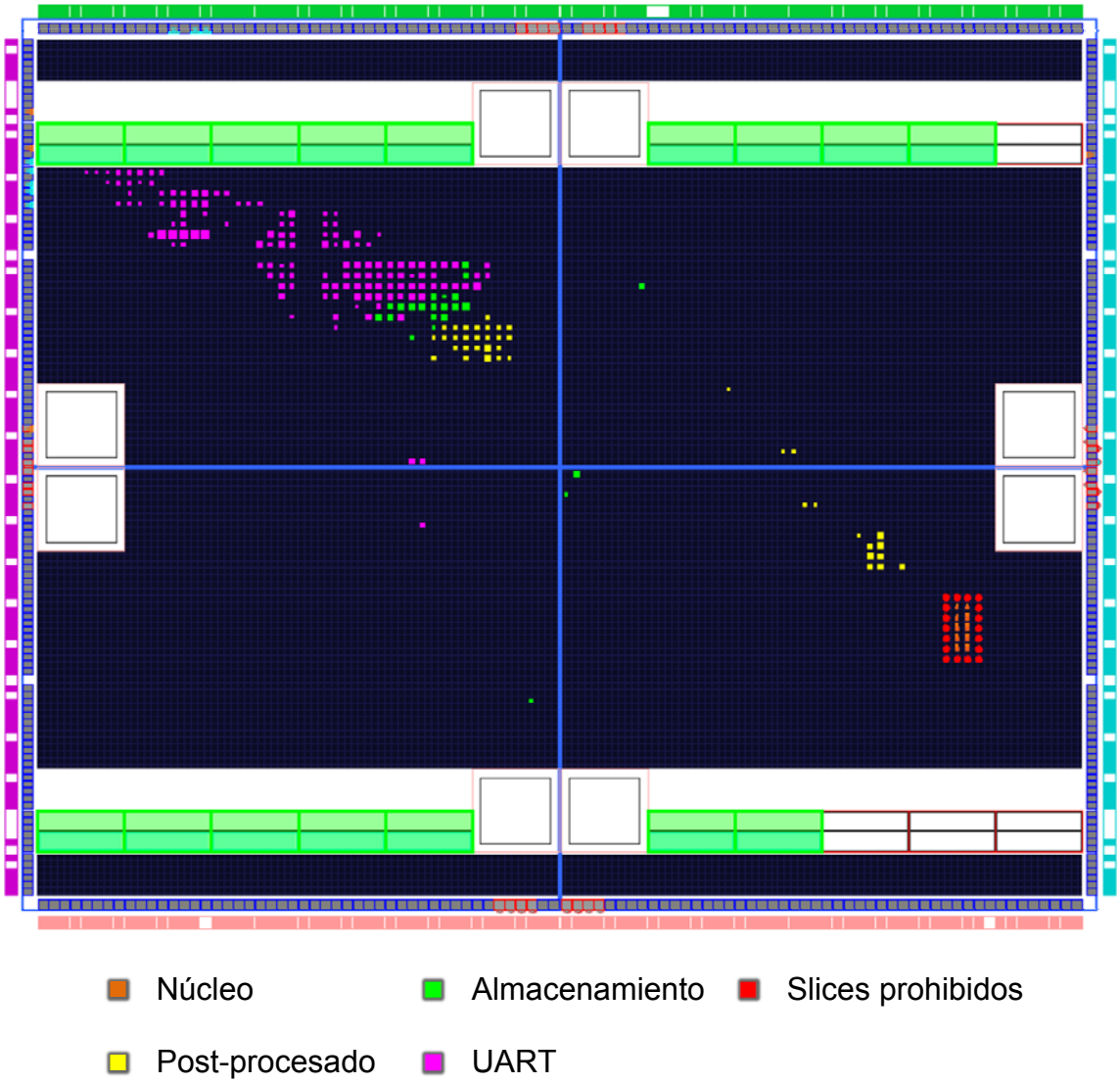


4.5. IMPLEMENTACIÓN

El proyecto fue implementado en el sistema de desarrollo Spartan-3AN Starter Kit, el cual está disponible como recurso de laboratorio en la Universidad Industrial de Santander. Para generar el archivo de configuración del dispositivo (.bit), se utiliza el *software* Plan Ahead, incluido dentro del paquete ISE Web Pack, debido a la facilidad para ubicar manualmente elementos dentro del chip. La configuración del archivo .bit se realiza con el *software* iMPACT. En la figura 24 se muestra el mapeo de recursos del proyecto completo, generado por la herramienta de implementación incluida en Plan Ahead.

[1] http://opencores.org/project,muart

Figura 24. Mapeo: implementación completa



Para aislar el núcleo del generador de los demás circuitos dentro del chip, es necesario incluir un anillo de guarda alrededor de los *slices* donde está ubicado. El anillo de guarda consiste en un perímetro construido por *slices*, sin posibilidad de utilizarse para otros propósitos. Los *slices* del anillo de guarda se señalan con el atributo “CONFIG PROHIBIT” dentro del archivo de restricciones (.ucf), para impedir que el *software* haga uso de ellos.

En la figura 25, se observa el circuito del núcleo del generador, ubicado sobre 10 slices y rodeado por 18 slices dummy (deshabilitados) que conforman el anillo de guarda. Dentro de los slices, el núcleo utiliza 11 LUTs y 6 *flip-flops* organizados en dos zonas correspondientes a la fuente de entropía (recuadro azul) y el extractor (recuadro morado).

Figura 25. Mapeo: núcleo del generador

4.6. PRUEBAS

En las pruebas estadísticas de verificación y aprobación del generador se utiliza el paquete de prueba estadístico para la validación de generadores de números aleatorios y pseudo-aleatorios en aplicaciones criptográficas propuesto por el Instituto Nacional de Estándares y Tecnología (NIST) [15], el cual por medio de 15 pruebas a la secuencia de datos generados codificados en bits o bytes entrega en cada una dos parámetros cuantitativos (valor-p y la proporción de aprobación) los

cuales deben pasar un mínimo para considerar el generador con las propiedades estadísticas adecuadas para ser utilizado en criptografía.

La implementación de la prueba se realiza utilizando los códigos fuente en lenguaje C disponibles en el sitio web del NIST[2]. Sin embargo, dado que los códigos están diseñados para el sistema operativo Solaris, se utiliza un proyecto de Microsoft Visual Studio 2013 publicado en el sitio web de desarrollo GitHub por el usuario "denisov-v"[3] para crear un ejecutable de Microsoft Windows que permite acceder por DOS a las pruebas propuestas.

En la figura 26 se observa la consola de comandos ejecutando el *software* y las opciones de configuración. Su primer menú permite escoger si se utiliza un algoritmo de generación por *software* (varios disponibles) o se ingresa un archivo de datos ya generado; el segundo menú permite escoger las pruebas a realizar y el menú final configurar las pruebas parametrizables si se han escogido.

Una vez finalizadas las pruebas, la consola de comando vuelve a la raíz de la carpeta y en la carpeta de *Experiments* > *AlgorithmTesting,* dentro de la carpeta donde está contenido el *software*, se puede leer el resumen de resultados.

En la figura 27 se observa un reporte de ejemplo, donde cada prueba estadística configurada muestra sus resultados, mostrando en las 10 primeras columnas la distribución de los resultados según su valor p y la proporción de aceptación. Según la cantidad de datos el *software* establece un margen mínimo por cumplir.

[2] http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html

[3] https://github.com/denisov-v/NIST_STS

Figura 26. *Software* de la prueba estadística del NIST en ejecución

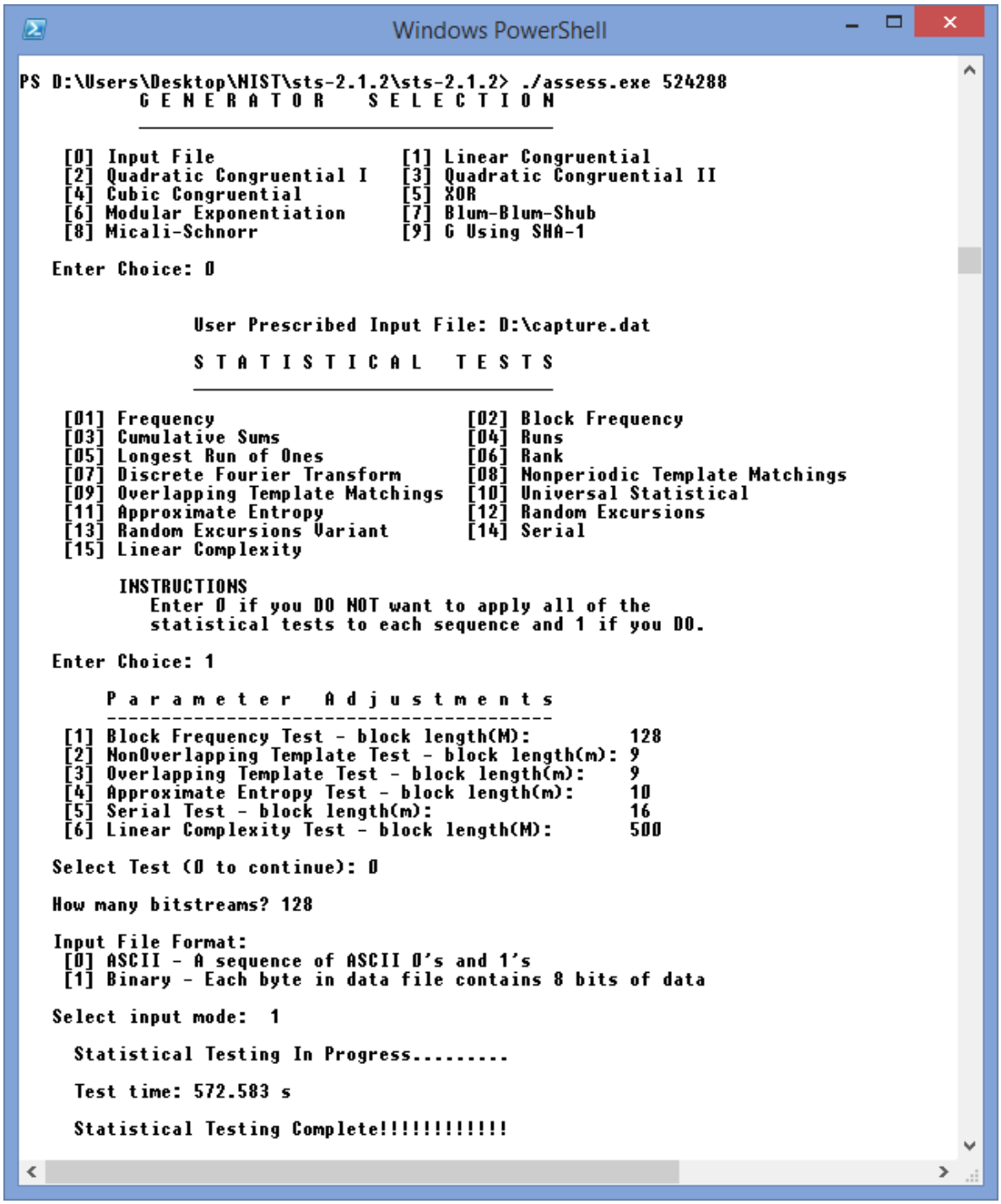


Figura 27. Ejemplo de reporte: resultados de uniformidad del valor-p y la proporción de secuencias aprobadas

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------

------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
 17   8  15  24  19  15  16  13  19  14  0.340461   0.9938    frequency
 12  16  18  15  15  14  10  17  14  29  0.098036   0.9938    cumulative-sums
 14  12  16  14  14  15  10  25  21  19  0.258961   0.9938    cumulative-sums
 20  12  23  11  21  18  12  20  15   8  0.105618   0.9875    runs


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the random
excursion (variant) test is approximately = 0.966402 for a sample size = 160
binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

```

5. RESULTADOS Y ANÁLISIS

5.1. RESULTADOS Y ANÁLISIS DE IMPLEMENTACIÓN

El análisis de recursos entregado por el *software* PlanAhead (tabla 5) muestra que se emplearon para todos los módulos requeridos en la generación y transmisión, incluyendo los de propósitos de validación, 299 *slices* de los 5888 disponibles, es decir el 5% de los *slices* configurables. Por lo tanto, la utilización de recursos es adecuada para el equipo de desarrollo (FPGA) utilizado, e incluso para modelos con menor cantidad de recursos lógicos. Respecto al uso de memoria RAM, se consumieron 18 de los 20 módulos para almacenar los 32768 bytes por generación. Sin embargo, este consumo se puede disminuir en tanto las restricciones de implementación lo requieran.

Tabla 5. Recursos implementados con el módulo de transmisión y pruebas

Recursos	Utilización	% Utilización	Disponible
Flip-flop	205	1,74%	11776
LUT (4 Entradas)	216	1,83%	11776
Slice	299	5,08%	5888
IO	12	3,23%	372
BUFGMUX	1	4,17%	24

Respecto al consumo de recursos por parte sólo del TRNG (sin módulo de transmisión o pruebas) los *slices* consumidos son 68, los LUT son 30 y los Flip Flops son 56. En la tabla 6 se muestra el resumen de recursos consumidos por el TRNG. Para este caso es importante especificar cada uno de los elementos consumidos, dado que si el generador se quiere implementar en un circuito integrado, el diseñador debe implementar los elementos fundamentales de la FPGA (LUT y *Flip-*

flops) mediante transistores, según la tecnología empleada. Realizando esta salvedad, se considera que el consumo de recursos es bajo y apto para los requerimientos planteados por el proyecto.

Tabla 6. Recursos implementados sólo para el TRNG

Recursos	Utilización	% Utilización	Disponible
Flip-flop	56	0,48%	11776
LUT (4 Entradas)	30	0,25%	11776
Slice	68	1,15%	5888
IO	16	4,30%	372
BUFGMUX	1	4,17%	24

Respecto al funcionamiento del módulo de transmisión, en las pruebas realizadas envía adecuadamente al computador caracteres generados en la FPGA, por medio del cable RS-232/USB a una tasa de 9600 baudios, siendo una velocidad baja para los estándares actuales pero suficientes y adecuados para transmitir los datos almacenados en memoria para su almacenamiento.

5.2. RESULTADOS EN PRUEBAS ESTADÍSTICAS

Para la prueba estadística se ingresó un archivo de caracteres ANSI capturado con el programa RealTerm a una tasa de 9600 bauds. Los 8192 kilobytes de datos recolectados se pasaron al *software* de prueba estadística como 128 cadenas de 524288 bits, siguiendo las recomendaciones para el ingreso de datos en el manual del NIST [15].

En la tabla 7 se muestra el resumen de resultados para el generador final en cada una de las pruebas, según el orden de realización por parte de la batería de pruebas. En los anexos A y B se muestran los resultados para la prueba con el generador sin post-procesado y con post-procesado basado en una compuerta XOR, que corresponden a versiones previas del desarrollo que son relevantes para contrastar el resultado final.

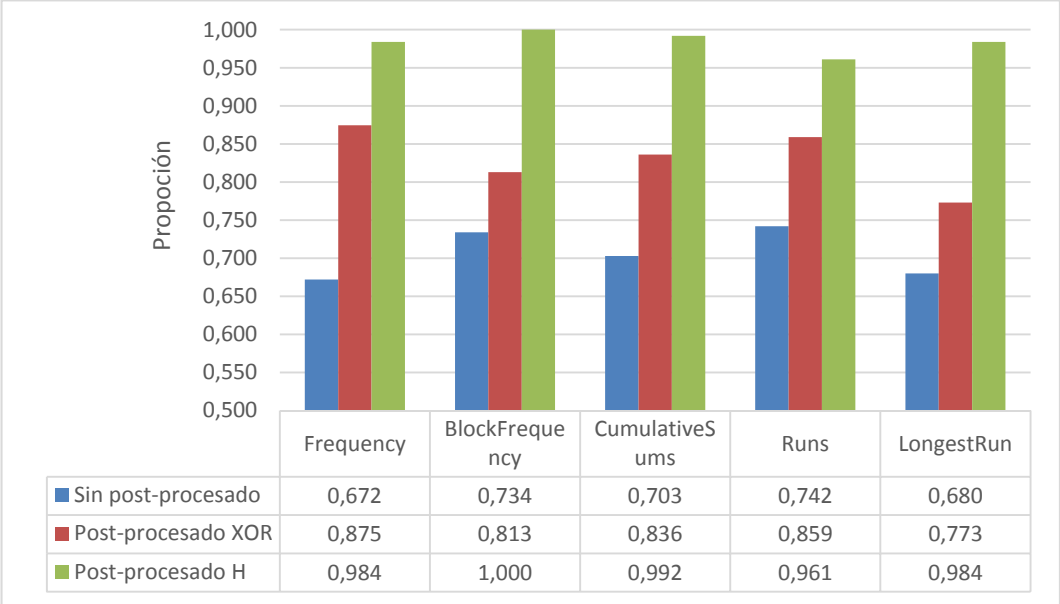
Tabla 7. Resultados en pruebas estadísticas realizadas

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Valor p	Propor.	Prueba Estadís.
14	17	13	11	11	8	16	13	12	13	0.789	0.984	Frequency
7	9	16	13	18	15	12	17	9	12	0.287	1.000	BlockFrequency
17	17	20	7	8	11	9	6	20	13	0.011	1.000	CumulativeSums
13	14	18	10	14	12	11	6	15	15	0.469	0.992	CumulativeSums
19	11	11	13	13	12	12	12	14	11	0.849	0.961	Runs
9	14	15	9	14	15	11	16	9	16	0.619	0.984	LongestRun
11	17	14	9	12	11	15	15	15	9	0.706	0.969	Rank
8	15	10	13	9	16	10	15	15	17	0.485	0.992	FFT
17	16	8	10	12	9	17	15	14	10	0.407	0.992	nonperiodic-temp
13	9	12	15	12	15	12	14	11	15	0.941	0.977	nonperiodic-temp
10	11	10	13	7	18	13	12	15	19	0.287	0.984	nonperiodic-temp
9	11	16	15	17	12	10	13	13	12	0.788	0.992	overlap-temp
14	10	11	18	13	7	17	15	10	13	0.422	0.992	universal
15	16	13	17	15	6	15	17	8	6	0.086	0.992	Apen
11	4	3	5	5	6	5	11	7	5	0.254	0.984	rand-excursions
5	7	6	5	8	7	8	5	5	6	0.672	0.985	rand-exc-variant
15	11	18	10	13	13	12	14	13	9	0.789	0.992	Serial
12	12	10	15	19	10	13	13	10	14	0.706	1.000	linear-complexity

5.3. ANÁLISIS DE RESULTADOS EN PRUEBAS ESTADÍSTICAS

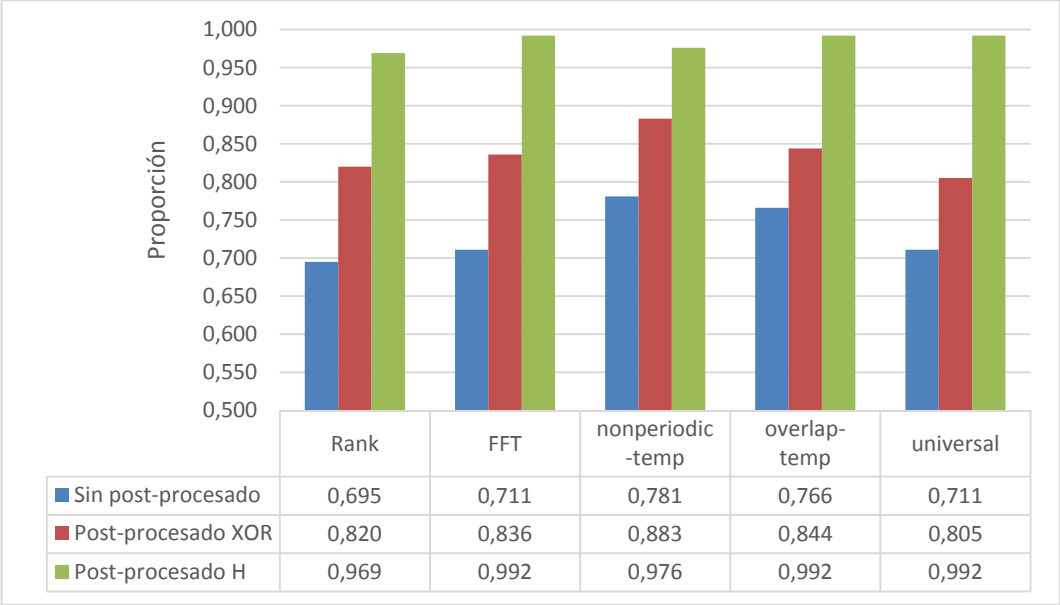
Para cada uno de los 15 tipos de pruebas propuestas por el NIST [15], el generador basado en muestreo coherente y post-procesado con la implementación de función H mostró resultados satisfactorios, pasando el margen indicado para la cantidad de datos ingresado (0,9609375). Al analizar y comparar las pruebas iniciales (generador sin post-procesado y XOR) (Anexos A y B) y la versión final en las figuras 28, 29 y 30, se observan falencias para varias de las pruebas, especialmente en la primera versión del generador. Esto indica que el post-procesado implementado es necesario y eficiente para la fuente de entropía implementada.

Figura 28. Gráfica comparativa de resultados para diferentes métodos de post-procesados (1)



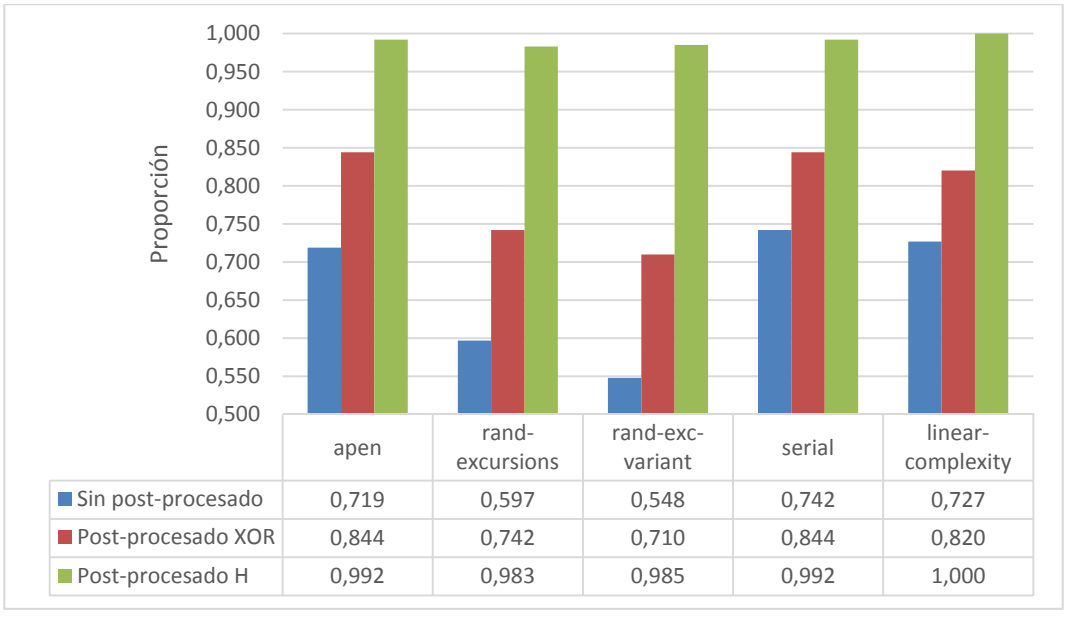
	Frequency	BlockFrequency	CumulativeSums	Runs	LongestRun
Sin post-procesado	0,672	0,734	0,703	0,742	0,680
Post-procesado XOR	0,875	0,813	0,836	0,859	0,773
Post-procesado H	0,984	1,000	0,992	0,961	0,984

Figura 29. Gráfica comparativa de resultados para diferentes métodos de post-procesados (2)



	Rank	FFT	nonperiodic -temp	overlap-temp	universal
Sin post-procesado	0,695	0,711	0,781	0,766	0,711
Post-procesado XOR	0,820	0,836	0,883	0,844	0,805
Post-procesado H	0,969	0,992	0,976	0,992	0,992

Figura 30. Gráfica comparativa de resultados para diferentes métodos de post-procesados (3)



	apen	rand-excursions	rand-exc-variant	serial	linear-complexity
Sin post-procesado	0,719	0,597	0,548	0,742	0,727
Post-procesado XOR	0,844	0,742	0,710	0,844	0,820
Post-procesado H	0,992	0,983	0,985	0,992	1,000

Analizando específicamente las pruebas, se nota que la prueba de corridas (*runs*) es la de menor valor en proporción y valor-p final, esto se debe a la alta correlación inherente a la fuente de entropía y que a pesar de la aplicación de la función H no muestra una gran mejoría. Sin embargo, en los resultados sin post-procesado esta característica era mucho menos favorable, indicando que el control por habilitación realizado por el post-procesado mitiga este comportamiento indeseado.

Respecto a la prueba de rango (*rank*), que en caso de fallar indica una dependencia lineal entre los bloques de la secuencia, el valor también es cercano al límite, mostrando que la correlación de datos también se muestra aunque en menor proporción para los bloques o arreglos de bits generados.

Para las demás pruebas, los resultados son favorables, permitiendo afirmar que el generador corregido con post-procesamiento de función H, es lo suficientemente eficiente para generar cadenas de 524288 bits aleatorios a una velocidad de transferencia por puerto de 9,6 kbps y una capacidad dentro del chip de 500 kbps, calculado con el promedio de los ciclos de reloj empleados para generar cada bit.

6. CONCLUSIONES

En este trabajo de investigación se logra implementar un generador de números aleatorios verdadero (TRNG) en un sistema de desarrollo FPGA, con características aleatorias suficientes para aprobar los 16 tipos de pruebas estadísticas establecidas por el Instituto Nacional de Estándares y Tecnología (NIST) con una proporción de aprobación en cada prueba superior al 96,09%, cumpliendo los objetivos y requerimientos propuestos.

Es importante resaltar que el análisis cualitativo realizado a la documentación de diferentes tipos de TRNG, permite entregar un resumen comparativo que facilita la escogencia de los diseños dependiendo de las características requeridas por la aplicación criptográfica. Según los requerimientos específicos de este trabajo, el diseño adecuado es el basado en muestreo coherente.

Así mismo, se destaca la necesidad de utilizar un módulo de post-procesado adecuado en el diseño escogido, pues las características aleatorias directas del generador sólo alcanzaron una proporción de aprobación entre 54,80% y 78,10% en las pruebas realizadas. La función H es el método de post-procesado que permitió alcanzar la validación del generador, con resultados de aprobación del 96,1% al 100%. Otros métodos implementados como la compuerta or-exclusiva (XOR), no alcanzaron a corregir adecuadamente el sesgo, teniendo un rango de aprobación entre 71% y 88,3%.

A partir de las pruebas de validación realizadas, se puede concluir que el TRNG presentado es adecuado para su utilización en aplicaciones criptográficas, en tanto no se requiera un *throughput* superior a 500 kbps, como es el caso de los trabajos de investigación en que se busca contribuir.

7. OBSERVACIONES Y RECOMENDACIONES

Dado que la implementación de este TRNG está diseñada para los recursos y funcionamiento en una FPGA, se deben tener en cuenta ajustes a la hora de implementarlo en un circuito integrado, que es la intención por parte del grupo de investigación. Se aconseja verificar el funcionamiento de los anillos oscilantes en la tecnología utilizada y definir los valores iniciales de los elementos secuenciales (pre-set) para asegurar el correcto funcionamiento de los módulos.

Además se recomienda verificar el diseño de un PLL como fuente de entropía si en la tecnología manejada se puede implementar dentro del circuito integrado. El proceso de verificación sigue la misma metodología trabajada en este proyecto. Se aconseja su revisión debido a la posibilidad de ajustar con mayor precisión la diferencia de frecuencias, y por tanto el *jitter* [21].

Por último, se propone dar mayor robustez al sistema, al integrar una adaptación en *hardware* de algunas de las pruebas estadísticas utilizadas, para hacer un análisis en tiempo real de las secuencias generadas y detectar secuencias temporales donde el generador no esté funcionando adecuadamente.

REFERENCIAS BIBLIOGRÁFICAS

[1] Bolaños, J.D.; Monsalve, V., "Plan de Trabajo de Grado – Modalidad Investigación: Diseño e implementación de un generador de números aleatorios en un sistema embebido," Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones (E3T), Universidad Industrial de Santander, pp. 17-20, Agosto 2014.

[2] Bruyninckx, H.; Lafitte, F.; Van Heule, D., "Safe cryptographic random number generation using untrusted generators," Communications (ICC), 2014 IEEE International Conference on, pp. 731-736, 10-14 June 2014. doi: 10.1109/ICC.2014.6883406.

[3] Gray, R., “Entropy and information theory”. New York: Springer, pp. xii, 2011. ISBN: 1441979697.

[4] Verdú, S., "Fifty years of Shannon theory," Information Theory, IEEE Transactions on, vol.44, no.6, pp.2057-2078, Oct 1998. doi: 10.1109/18.720531.

[5] Lozach, F.; Ben-Romdhane, M.; Graba, T.; Danger, J.-L., "FPGA Design of an Open-Loop True Random Number Generator," Digital System Design (DSD), 2013 Euromicro Conference on, pp.615-622, 4-6 Sept. 2013 doi: 10.1109/DSD.2013.73.

[6] Rahman, M.T.; Kan Xiao; Forte, D.; Xuhei Zhang; Shi, J.; Tehranipoor, M., "TI-TRNG: Technology independent true random number generator," Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE , pp.1-6, 1-5 June 2014. doi: 10.1145/2593069.2593236.

[7] Simka, M.; Drutarovsky, M.; Fischer, V.; Fayolle, J., "Model of a true random number generator aimed at cryptographic applications," Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on, pp. 4, 21-24, May 2006. doi: 10.1109/ISCAS.2006.1693909.

[8] Kwok, S.H.M.; Lam, E.Y., "FPGA-based High-speed True Random Number Generator for Cryptographic Applications," TENCON 2006. 2006 IEEE Region 10 Conference, pp.1-4, 14-17 Nov. 2006. doi: 10.1109/TENCON.2006.344013.

[9] Dichtl, M., 'Bad and Good Ways of Post-processing Biased Physical Random Numbers', Fast Software Encryption, pp. 137-152, 2007. doi: 10.1007/978-3-540-74619-5_9.

[10] Majzoobi, M.; Koushanfar, F.; Devadas, S., 'FPGA-Based True Random Number Generation Using Circuit Metastability with Adaptive Feedback Control', Cryptographic Hardware and Embedded Systems – CHES 2011, pp. 17-32, 2011. doi: 10.1007/978-3-642-23951-9_2.

[11] Wieczorek, P.Z.; Golofit, K., "Dual-Metastability Time-Competitive True Random Number Generator," Circuits and Systems I: Regular Papers, IEEE Transactions on, vol.61, no.1, pp.134-145, Jan. 2014. doi: 10.1109/TCSI.2013.2265952.

[12] Vasyltsov, I.; Hambardzumyan, E.; Kim, Y.; Karpinskyy, B., 'Fast Digital TRNG Based on Metastable Ring Oscillator', Lecture Notes in Computer Science, pp. 164-180, 2008. doi: 10.1007/978-3-540-85053-3_11.

[13] Valtchanov, B.; Fischer, V.; Aubert, A., "Enhanced TRNG based on the coherent sampling," Signals, Circuits and Systems (SCS), 2009 3rd International Conference on, pp.1-6, 6-8, Nov. 2009. doi: 10.1109/ICSCS.2009.5412601.

[14] Kohlbrenner P.; Gaj K., 'An embedded true random number generator for FPGAs', Proceeding of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays - FPGA '04, 2004. doi: 10.1145/968280.968292.

[15] Rukhin, A.; Soto, J.; Nechvatal, J.; Smid, M.; Barker, E.; Leigh, S.; Levenson, V.; Vangel, M.; Banks, D.; Heckert, A.; Dray, J.; Vo, S.; "A statistical test suite for random and pseudorandom number generators for cryptographic applications, National Institute of Standards and Technology (NIST)," special publication 800-22, pp. 13-62, August 2008. URL: http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf [Accessed: 20- Apr- 2015].

[16] Stat.fsu.edu, 'The Marsaglia Random Number CDROM including the Diehard Batteryof Tests', 2015. [Online]. URL: http://stat.fsu.edu/pub/diehard [Accessed: 20- Apr- 2015].

[17] Read, D.; "Iterative development: Key technique for managing software developments," ICT WA 2005 Conference, pp. 1-5, 2005. doi: 10.1.1.83.6958.

[18] Karsai, G.; Sztipanovits, J.; Ledeczi, A.; Bapty, T., "Model-integrated development of embedded software," Proceedings of the IEEE, vol.91, no.1, pp.145-164, Jan 2003. doi: 10.1109/JPROC.2002.805824.

[19] Özgen, C.; "A true random number generator in FPGA for cryptographic applications." PhD diss., MIDDLE EAST TECHNICAL UNIVERSITY, 2012.

[20] McNeill, J., "Jitter in ring oscillators," Circuits and Systems, 1994. ISCAS '94., 1994 IEEE International Symposium on , vol.6, pp.201,204 vol.6, 30 May-2 Jun 1994. doi: 10.1109/ISCAS.1994.409561

[21] Hancock, J.; Draving, S., "Jitter-Understanding it, Measuring It", High Frequency Electronics, pp. 20-28, May 2004.

BIBLIOGRAFÍA

BOLAÑOS, J.D. y MONSALVE, V. Plan de Trabajo de Grado – Modalidad Investigación: Diseño e implementación de un generador de números aleatorios en un sistema embebido, Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones (E3T), Universidad Industrial de Santander, 2014, p. 17-20.

BRUYNINCKX, H.; LAFITTE, F. y VAN HEULE, D. Safe cryptographic random number generation using untrusted generators, Communications (ICC), 2014 IEEE International Conference on, 2014, p. 731-736, 10-14.

DICHTL, M. Bad and Good Ways of Post-processing Biased Physical Random Numbers, Fast Software Encryption, 2007, p. 137-152.

GRAY, R. Entropy and information theory. New York: Springer, 2011, p. xii.

HANCOCK, J. y DRAVING, S., Jitter-Understanding it, Measuring It, High Frequency Electronics, 2004, p. 20-28.

KARSAI, G. *et al.* Model-integrated development of embedded software, Proceedings of the IEEE, 2003, vol.91, no.1, p.145-164.

KOHLBRENNER, P. y GAJ, K. An embedded true random number generator for FPGAs, Proceeding of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays - FPGA 04, 2004.

KWOK, S.H.M y LAM, E.Y. FPGA-based High-speed True Random Number Generator for Cryptographic Applications, TENCON 2006. 2006 IEEE Region 10 Conference, 2006, p.1-4, 14-17.

LOZACH, F. *et al.* FPGA Design of an Open-Loop True Random Number Generator, Digital System Design (DSD), 2013 Euromicro Conference on, 2013, p. 615-622.

MAJZOOBI, M.; KOUSHANFAR, F. y DEVADAS, S. FPGA-Based True Random Number Generation Using Circuit Metastability with Adaptive Feedback Control, Cryptographic Hardware and Embedded Systems – CHES 2011, 2011, p. 17-32.

MCNEILL, J. Jitter in ring oscillators, Circuits and Systems, 1994. ISCAS 94, 1994 IEEE International Symposium on, 1994, vol.6, p. 201-204.

ÖZGEN, C. A true random number generator in FPGA for cryptographic applications. PhD diss., MIDDLE EAST TECHNICAL UNIVERSITY, 2012.

RAHMAN, M.T. *et al.* TI-TRNG: Technology independent true random number generator, Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE, 2014, p.1-6.

READ, D.; Iterative development: Key technique for managing software developments, ICT WA 2005 Conference, 2005, p. 1-5.

RUKHIN, A. *et al.* A statistical test suite for random and pseudorandom number generators for cryptographic applications, National Institute of Standards and Technology (NIST), special publication 800-22, 2008, p. 13-62.

SIMKA, M. *et al.* Model of a true random number generator aimed at cryptographic applications, Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on, 2006, p. 4, 21-24.

THE FLORIDA STATE UNIVERSITY. The Marsaglia Random Number CDROM including the Diehard Battery of Tests. [en línea]. http://stat.fsu.edu/pub/diehard [citado el 20 de Abril de 2015].

VALTCHANOV, B.; FISCHER, V. y AUBERT, A., Enhanced TRNG based on the coherent sampling, Signals, Circuits and Systems (SCS), 2009 3rd International Conference on, 2009, p. 1-6, 6-8.

VASYLTSOV, I. *et al.* Fast Digital TRNG Based on Metastable Ring Oscillator, Lecture Notes in Computer Science, 2008, p. 164-180.

VERDÚ, S. Fifty years of Shannon theory, Information Theory, IEEE Transactions on, 1998, vol.44, no.6, p. 2057-2078.

WIECZOREK, P.Z. y GOLOFIT, K., Dual-Metastability Time-Competitive True Random Number Generator, Circuits and Systems I: Regular Papers, IEEE Transactions on, 2014, vol.61, no.1, p. 134-145.

ANEXOS

ANEXO A. RESULTADOS PREVIOS SIN POST-PROCESADO

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Valor p	Proporción	Prueba Estadística
35	21	19	16	9	8	3	4	3	10	0.000*	0.672*	Frequency
29	18	13	19	19	4	4	10	6	6	0.001*	0.734*	BlockFrequency
29	22	20	15	10	6	15	2	4	5	0.000*	0.703*	CumulativeSums
28	23	17	20	11	3	17	3	4	2	0.000*	0.742*	CumulativeSums
42	23	10	18	8	5	6	7	3	6	0.000*	0.742*	Runs
27	22	15	14	5	13	12	13	2	5	0.001*	0.680*	LongestRun
26	16	16	15	12	11	15	10	5	2	0.001*	0.695*	Rank
24	22	24	13	12	8	11	2	7	5	0.001*	0.711*	FFT
26	27	20	14	15	6	8	2	3	7	0.000*	0.781*	nonperiodic-temp
25	18	17	15	9	7	5	10	9	13	0.001*	0.719*	nonperiodic-temp
25	19	19	15	18	16	8	5	1	2	0.001*	0.711*	nonperiodic-temp
25	21	18	13	14	7	9	3	9	9	0.001*	0.766*	overlap-temp
29	16	20	15	15	9	10	9	3	2	0.001*	0.711*	universal
27	21	11	21	12	9	10	6	4	7	0.001*	0.719*	apen
10	9	9	8	6	6	2	5	5	2	0.000*	0.597*	rand-excursions
12	10	11	5	6	8	6	1	1	2	0.001*	0.548*	rand-exc-variant
27	13	15	16	12	16	14	8	0	7	0.000*	0.742*	serial
29	14	18	12	11	11	6	10	4	13	0.001*	0.727*	linear-complexity

The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 0.9609375 for a sample size = 128 binary sequences. The minimum pass rate for the random excursion (variant) test is approximately 0.951613 for a sample size = 62 binary sequences.

ANEXO B. RESULTADOS PREVIOS CON POST-PROCESADO XOR

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Valor p	Proporción	Prueba Estadística
18	19	16	16	8	12	7	8	7	17	0.004*	0.875*	Frequency
19	17	13	18	21	8	6	14	9	3	0.011	0.813*	BlockFrequency
20	19	18	14	10	10	18	6	6	7	0.010	0.836*	CumulativeSums
19	20	17	19	11	8	20	5	7	2	0.005*	0.852*	CumulativeSums
32	22	10	16	8	9	11	12	7	1	0.005*	0.859*	Runs
20	18	14	15	6	16	14	17	6	2	0.016	0.773*	LongestRun
17	13	13	15	14	14	17	15	10	0	0.031	0.820*	Rank
14	19	22	13	13	11	13	7	11	5	0.029	0.836*	FFT
18	26	18	14	14	11	11	7	8	1	0.002*	0.883*	nonperiodic-temp
18	15	17	14	11	12	4	18	11	8	0.012	0.805*	nonperiodic-temp
17	15	19	13	19	18	10	9	4	4	0.017	0.828*	nonperiodic-temp
17	20	18	12	15	11	12	6	12	5	0.028	0.844*	overlap-temp
22	13	18	14	17	11	14	11	5	3	0.022	0.805*	universal
20	17	11	20	11	13	14	11	9	2	0.033	0.844*	apen
8	13	7	5	6	8	2	7	5	1	0.009*	0.742*	rand-excursions
9	8	9	6	7	9	7	3	2	2	0.017	0.710*	rand-exc-variant
20	12	13	16	12	20	17	11	3	4	0.005*	0.844*	serial
21	13	15	13	13	16	9	18	5	5	0.028	0.820*	linear-complexity

The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 0.9609375 for a sample size = 128 binary sequences. The minimum pass rate for the random excursion (variant) test is approximately 0.951613 for a sample size = 62 binary sequences.