Generadores de números aleatorios y su aplicación para cifrar y autenticar datos digitales

por

José David Rodríguez Muñoz

Tesis sometida como requisito parcial
para obtener el grado de:

MAESTRO EN CIENCIAS EN LA ESPECIALIDAD DE ELECTRÓNICA

por el

Instituto Nacional de Astrofísica, Óptica y Electrónica
Noviembre 2024
Tonantzintla, Puebla

Supervisada por:

Dr. Esteban Tlelo Cuautle
Investigador Titular del INAOE

y

Dr. Luis Gerardo de la Fraga
Departamento de Computación, Cinvestav

A mi familia y amigos, quienes siempre me han apoyado.

Resumen

Actualmente, el mundo ha experimentado un incremento en el número de dispositivos electrónicos conectados a las redes inalámbricas. Por esta razón, los investigadores buscan desarrollar algoritmos criptográficos más robustos que garanticen privacidad y seguridad en el proceso de comunicación. Un componente con aplicaciones generales en el campo de la criptografía es el generador de números aleatorios (RNG), el cual proporciona una secuencia de números sin patrón predecible que se emplean en algoritmos de cifrado y autenticación de datos. Sin embargo, el reto actual consiste en garantizar la mejor aleatoriedad de las secuencias, la cual es medida mediante pruebas estadísticas como las NIST y TestU01. Una opción en el diseño de estos generadores es el uso de sistemas caóticos de tiempo continuo y discreto, debido a las propiedades de estos modelos matemáticos. Estos sistemas tienen como ventaja un bajo consumo de elementos de hardware digital que pueden describirse en Verilog y llevarse a layout para su fabricación en tecnologías de circuitos integrados CMOS. Por otra parte, son fáciles de codificar en un lenguaje de programación de alto nivel, como Python, para diversas aplicaciones en sistemas embebidos. Debido a lo anterior, en esta tesis se diseñan generadores de números pseudoaleatorios (PRNGs) basados en sistemas caóticos, los cuales se sintetizan en un arreglo de compuertas programables en campo (FPGA) usando aritmética de 64-bit y también se implementan en una Raspberry Pi. Además, se verifica la aleatoriedad de las secuencias generadas por cada PRNG y se aplican en un sistema de comunicación con etapas de cifrado y de autenticación de datos mediante un algoritmo de función hash, que permite generar etiquetas para corroborar la integridad de la información que se envía o almacena, como imágenes en tono de grises, imágenes RGB y texto. En el caso del sistema sintetizado en la FPGA, la comunicación se realiza entre la FPGA y una computadora utilizando el protocolo RS232. Por otro lado, en el sistema implementado en Raspberry Pi se emplea el protocolo de comunicación MQTT entre tres de estos dispositivos.

Abstract

Nowadays, there has been a significant increase in the number of electronic devices connected to wireless networks. As a result, researchers are seeking to develop more robust cryptographic algorithms that ensure privacy and security in the communication process. A component with general applications in cryptography is the Random Number Generator (RNG). This generator produces a sequence of numbers without a predictable pattern, and it can be used in encryption and data authentication algorithms. However, the current challenge is how to ensure the best possible randomness of the binary sequences, which can be measured by statistical tests such as NIST and TestU01. Due to the properties of these mathematical models, continuous and discrete-time chaotic systems are an option in the design of these generators. These systems offer the advantage of low consumption of digital hardware elements, which could be described using Verilog and are suitable for manufacturing in CMOS integrated circuit technologies. Moreover, they are easy to code in a high-level programming language, such as Python, for various applications in embedded systems. Due to the above, in this thesis, Pseudo-Random Number Generators (PRNGs) based on chaotic systems are designed and synthesized in a Field-Programmable Gate Array (FPGA) using 64-bit arithmetic and also implemented on a Raspberry Pi. In addition, the randomness of the sequences generated by each PRNG is verified and applied in encryption and authentication stages using a hash function algorithm, which allows the generating of authentication tags for the information that is sent or stored, such as grayscale images, RGB images, and text. In the case of the chaos-based system synthesized in the FPGA, the communication occurs between the FPGA and a computer using the RS232 protocol. On the other hand, in the system implemented in Raspberry Pi the communication uses the MQTT communication protocol among three of these devices.

Agradecimientos

Quiero comenzar expresando mi agradecimiento a Dios por permitirme trabajar en este proyecto y por concederme la sabiduría para llevarlo a cabo. Mi agradecimiento más sincero para mi madre Inés Gabriela Muñoz González, mi padre Hermilo Rodríguez García y mis hermanos Josué Daniel y Uzziel Abisai, quienes me brindaron su apoyo incondicional durante todo el desarrollo de esta tesis.

Agradezco a mis dos asesores por su invaluable guía, por la constante enseñanza y por resolver cada una de mis dudas a lo largo del proyecto. Su apoyo y orientación han sido fundamentales para esta tesis.

Al CONAHCYT México, por otorgarme la beca que me permitió llevar a cabo mis estudios de maestría en el INAOE.

Por último, agradezco a mis amigos y compañeros, quienes compartieron su tiempo, conocimiento y amistad.

Muchas gracias a todos.

Índice general

Capítulo 1

Introducción

1.1. Concepto y características del caos

El comportamiento caótico, un tema de interés en los recientes años, está presente en sistemas naturales y sociales, como en el clima, el fluido de un líquido, el crecimiento de una población, las epidemias, o incluso en una crisis financiera [1]. La definición de la palabra caos ha ido evolucionando a lo largo de la historia y hoy en día sugiere el desorden de algo [2]. En el contexto riguroso de las matemáticas y la física, el caos se define como el comportamiento determinista, acotado y aperiódico de un sistema dinámico, cuya característica principal es la alta sensibilidad a las condiciones iniciales [3]. Los modelos matemáticos que tienen este comportamiento emplean términos o funciones no lineales, y una primera clasificación para esta dinámica se basa en el tipo de ecuaciones utilizadas, las cuales pueden ser descripciones en tiempo continuo o discreto. Los sistemas que se modelan por ecuaciones diferenciales ordinarias en tiempo continuo, son llamados sistemas caóticos. Hoy en día, los sistemas caóticos están representados mediante ecuaciones diferenciales de orden entero o incluso de orden fraccionario. Por otro lado, los sistemas que generan caos y que se modelan por ecuaciones en tiempo discreto, son conocidos como mapas caóticos y usan ecuaciones de diferencias para su representación [4]. Un punto importante acerca de los modelos matemáticos, es que un sistema no lineal no garantiza el comportamiento caótico. Para presentarlo, el sistema debe tener al menos un exponente de Lyapunov positivo [5].

Como los modelos caóticos describen sistemas deterministas, la evolución temporal de sus variables de estado sigue una regla determinada por las condiciones iniciales y los parámetros

del sistema. Sin embargo, la alta sensibilidad a las condiciones iniciales provoca que, ante pequeños cambios en las condiciones, se obtengan trayectorias muy diferentes entre sí a medida que aumenta el tiempo. Además, en un espacio de fases, estas trayectorias se mantienen en una región delimitada conocida como atractor extraño, siempre y cuando las condiciones iniciales estén dentro de un rango válido de valores [6].

Dependiendo del análisis de los modelos matemáticos que describen a los sistemas caóticos, existe una clasificación que se basa en la definición del tipo de atractor, que puede ser autoexitado u oculto. En el primer caso, un atractor autoexcitado tiene una cuenca de atracción ligada a un punto de equilibrio inestable, mientras que un atractor oculto tiene una cuenca de atracción que no se cruza en la vecindad de algún punto de equilibrio [7]. Por tal motivo, los puntos de equilibrio y los valores propios de los modelos matemáticos asociados a los atractores autoexitados, se pueden encontrar mediante cálculos manuales. Por otro lado, los modelos matemáticos de los atractores ocultos, son difíciles de resolver y requieren de un procedimiento computacional avanzado para identificar y estimar los puntos de equilibrio y su dinámica en general [8].

Uno de los primeros científicos en trabajar con el caos fue Edward Norton Lorenz, quien en 1963 descubrió el comportamiento caótico mientras desarrollaba un modelo para predicción climática. Lorenz popularizó el concepto de efecto mariposa, que explica la sensibilidad de los sistemas caóticos a las condiciones iniciales [6]. El trabajo de Lorenz revolucionó el concepto del caos y sentó las bases para el desarrollo de la teoría del caos.

1.2. Análisis de sistemas caóticos

Los sistemas dinámicos en tiempo continuo, como los caóticos, están descritos por ecuaciones diferenciales ordinarias, cuya solución analítica o exacta no es posible de lograr en muchos casos. Por tal motivo, estos modelos se resuelven utilizando métodos numéricos que aproximan la solución a un problema de valor inicial de la forma $\dot{\vec{x}} = f(x_1, x_2, \ldots, x_n, t)$, donde $\vec{x} = [x_1, x_2, \ldots, x_n]$. Estos métodos resultan relativamente fáciles de implementar en una computadora. Básicamente, los métodos numéricos transforman las ecuaciones diferenciales ordinarias en un sistema de ecuaciones discretas iterativas, las cuales permiten calcular los valores futuros de las variables de estado a partir de valores pasados (o actuales) [9]. Sin embargo, cada método numérico requiere de un análisis para estimar el error que puede generar y el valor apropiado del tamaño de paso

de integración, para asegurar convergencia de la solución.

En la literatura existen varios métodos numéricos, categorizados como métodos de un solo paso o multi pasos, y en ambos casos pueden ser explícitos o implícitos [10]. Cada uno de ellos ofrece diferentes niveles de aproximación con la solución exacta, tiempo de cómputo y región de estabilidad. El más usado y considerado simple de programar en software o un entorno digital, es conocido con el nombre de Euler hacia adelante (Forward Euler). Este método, de un solo paso y explícito, aproxima la solución de un sistema de ecuaciones diferenciales ordinarias utilizando un bajo número de cálculos. Por ello, su tiempo de procesamiento en una computadora o sistema embebido es el de menor duración. Además, cuando se desea un diseño electrónico, el método de Euler hacia adelante requiere el menor consumo de recursos lógicos usando dispositivos como el arreglo de compuertas programables en el campo (FPGA) [11]. Sin embargo, su desventaja es que la solución aproximada arroja un mayor error comparado con otros métodos de mayor orden o de múltiples pasos, como la familia de métodos Adams Bashforth y Runge Kutta. Si la solución requiere de una mejor aproximación, el método de Runge Kutta de orden 4, es una de las opciones más utilizadas, aunque toma un mayor tiempo de procesamiento computacional. No obstante, la elección de uno u otro método numérico depende de la aplicación que se esté buscando.

La solución a un sistema de ecuaciones diferenciales ordinarias, aplicando métodos numéricos explícitos, requerirá de valores pasados, los cuales dependen del orden polinomial. Por ejemplo, la tabla 1.1 muestra las ecuaciones iterativas para los métodos explícitos de Euler hacia adelante (FE), Adams Bashforth de orden 4 (AB4) y Runge Kutta de orden 4 (RK4). Se puede apreciar que el método FE requiere de un solo valor pasado descrito por y_i, mientras que el método AB4 requiere de 4 valores previos, descritos por: y_{i-3}, y_{i-2}, y_{i-1}, y y_i. Por su parte, el método RK4 depende de un solo valor pasado descrito por y_i, sin embargo, evalúa 4 valores que son promediados por la expresión: $\frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$.

La selección de un tamaño de paso de integración h adecuado asegura la estabilidad del método numérico. Una forma para estimar dicho valor, consiste en analizar los valores propios del sistema dinámico, aunque este criterio podría no cumplirse en algunos problemas no lineales [12]. De esta manera, una vez estimado el paso de integración, se obtienen series temporales asociadas a cada variable de estado del sistema, y con ellas se efectúan estudios para determinar la posibilidad de generar secuencias binarias aleatorias.

El caos se puede evaluar haciendo uso de técnicas o herramientas matemáticas que ayudan a

Tabla 1.1: Métodos numéricos y sus ecuaciones iterativas [9].

Método numérico	**Ecuación iterativa**
FE	$y_{i+1} = y_i + hf(x_i, y_i)$
AB4	$y_{i+1} = y_i + h\left(\frac{55}{24}f(x_i, y_i) - \frac{59}{24}f(x_{i-1}, y_{i-1}) + \frac{37}{24}f(x_{i-2}, y_{i-2}) - \frac{9}{24}f(x_{i-3}, y_{i-3})\right)$
RK4	$y_{i+1} = y_i + \frac{1}{6}h\left(k_1 + 2k_2 + 2k_3 + k_4\right)$ $k_1 = f(x_i, y_i)$ $k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right)$ $k_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right)$ $k_4 = f\left(x_i + h, y_i + k_3 h\right)$

estudiar la complejidad de la dinámica caótica. Por ejemplo, el caos se puede medir a través de la evaluación de los exponentes de Lyapunov (LEs), la dimensión Kaplan Yorke (D_{KY}) y la entropía de Kolmogorov-Sinai (KSE). Los exponentes de Lyapunov brindan una medida global del grado de la convergencia o divergencia de dos trayectorias cercanas en el espacio de fases [6, 13, 14], donde el número de estos exponentes corresponde con la dimensión del sistema no lineal [5]. Por ejemplo, el sistema caótico de Lorenz es tridimensional (3D), por lo tanto, tiene tres exponentes de Lyapunov. Además, el conjunto de estos LEs, ordenado de mayor a menor ($\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$), se conoce como el espectro de Lyapunov [15]. En específico, si un sistema tiene al menos un exponente positivo de Lyapunov, se dice que tiene comportamiento caótico. Si tiene dos o más exponentes positivos, se dice que tiene un comportamiento hipercaótico [6]. Por otro lado, la estimación de la dimensión fractal del atractor, puede evaluarse por la dimensión Kaplan-Yorke, cuyo cálculo se basa en los exponentes de Lyapunov, tal y como se muestra en la ecuación (1.1) [15, 16].

$$D_{KY} = j + \frac{\sum_{i=1}^{j} \lambda_i}{|\lambda_{j+1}|} \tag{1.1}$$

Por su parte, la entropía de Kolmogorov-Sinai es una medida de la divergencia de pequeños errores en una trayectoria caótica, dentro de un espacio de fases multidimensional. Su valor se

obtiene sumando los exponentes positivos de Lyapunov, como se describe en la referencia [15].

Encontrar los valores de los exponentes de Lyapunov de un sistema caótico es relevante, porque permite identificar los rangos de valores de los parámetros de un sistema, en el que exhibe un comportamiento caótico y forma un atractor extraño en el espacio de fases. Además, los valores de los exponentes de Lyapunov se pueden usar para calcular la dimensión Kaplan-Yorke y la entropía de Kolmogorov-Sinai. En la mayoría de los sistemas dinámicos no lineales, el cálculo de los LEs por medio de una solución analítica no es posible, ya que se requiere una solución analítica de las ecuaciones diferenciales ordinarias. Debido a esto, en muchos de los casos se emplean algoritmos numéricos para encontrarlos, como el método de Wolf [17]. Otra opción en el cálculo de los exponentes de Lyapunov consiste en utilizar el software de TISEAN, el cual proporciona los valores de los exponentes a partir de las series temporales del sistema caótico, solucionado por métodos numéricos [8].

1.3. Generadores de números aleatorios

En la actualidad, los avances tecnológicos han propiciado un aumento en el número de dispositivos electrónicos conectados a las redes inalámbricas. Cada año, una mayor cantidad de dispositivos inteligentes como televisores, sensores, cámaras, lavadoras, relojes, vehículos, entre otros, son conectados a internet. Algunos autores [18], estiman un total de 30 billones de dispositivos del Internet de las Cosas (IoT) conectados para 2027. Ante este panorama, es crucial garantizar la seguridad de la información durante su transmisión por las redes de comunicación. Por ello, los investigadores de diferentes campos trabajan en conjunto en el desarrollo de algoritmos criptográficos más rápidos, robustos y complejos que garanticen la privacidad, confidencialidad y autenticidad de los mensajes enviados y recibidos por los dispositivos electrónicos. Un concepto esencial que aparece en muchos de estos algoritmos criptográficos son los números aleatorios [19], los cuales son proporcionados por los generadores de números aleatorios (RNGs). El RNG es un algoritmo o un circuito capaz de crear una secuencia con números seleccionados al azar, dentro de un rango finito donde todos ellos tienen la misma probabilidad de aparecer. Aunque su aplicación más crítica es la rama de la criptografía, suelen verse en otras áreas, por ejemplo, para introducir aleatoriedad en los eventos de los juegos de video, elegir el ganador de una lotería, en simulaciones de procesos, entre otros [19].

La clasificación de los RNG comprende dos principales tipos: los generadores de números verdaderamente aleatorios (TRNG) y los generadores de números pseudoaleatorios (PRNG) [19–21].

Un TRNG es un sistema no determinista porque no tiene una regla o ecuación matemática que defina los valores futuros de la secuencia. Para lograr la verdadera aleatoriedad, los TRNGs emplean un componente de hardware físico, que realiza la lectura de algún fenómeno físico impredecible, como el ruido atmosférico, para generar los bits aleatorios. A tales dispositivos de hardware se les conoce como fuentes de entropía. Generalmente, los diseños de TRNGs están compuestos por el circuito que mide el fenómeno físico y un extractor de entropía, el cual es un algoritmo de postprocesamiento que genera los números aleatorios a partir de los datos de la fuente de entropía [19]. Debido al no determinismo, no es posible reproducir la misma secuencia. Es por ello que estos sistemas son esenciales en aplicaciones criptográficas. Sin embargo, en aplicaciones que requieren de alta rapidez de procesamiento, como en el cifrado de flujo, algunos TRNGs pueden no ser adecuados, ya que la lectura del fenómeno físico suele ser lenta [21].

Por su parte, el PRNG está asociado a un sistema determinista, ya que cuenta con una regla que define los valores futuros de la secuencia. Para iniciar el proceso de generación de números pseudoaleatorios, este tipo de sistemas requieren de valores iniciales conocidos como la semilla. Los PRNGs pueden ser implementados en software o con elementos de hardware. Al ser sistemas deterministas, la secuencia de números aleatorios es reproducible empleando la misma semilla. Sin embargo, no todo PRNG es útil en aplicaciones criptográficas; para ello, la secuencia de números debe cumplir o superar una serie de pruebas. Entre las pruebas estadísticas se encuentran las NIST y las TestU01, que miden las propiedades de aleatoriedad y uniformidad de la secuencia. Existen otras pruebas que miden la resistencia a la predicción y al retroceso [19], las cuales buscan asegurar que un atacante no pueda predecir valores pasados o futuros de la secuencia, a partir de un grupo de números observados, pero sin conocimiento de la semilla. Algunos autores emplean el concepto de PRNG criptográficamente seguro (CS-PRNG) para aquellos PRNG útiles en aplicaciones de criptografía [21]. Una ventaja de los PRNG es que los algoritmos o sistemas digitales presentan una alta rapidez de procesamiento.

También, es posible encontrar sistemas de RNGs que combinan un TRNG con un PRNG para formar un sistema híbrido, donde el TRNG actúa como el “sembrador” de las semillas para PRNG [20]. La tabla 1.2 muestra las características básicas de los tipos de generadores de números

aleatorios.

Tabla 1.2: Características básicas de los RNG [19].

Tipo de RNG:	**TRNG.**	**PRNG.**
Tipo de sistema:	No determinista	Determinista
Secuencia:	Con verdadera aleatoriedad	Semejante a una de verdadera aleatoriedad
Componentes:	Fuente de entropía Extractor de entropía	Algoritmo diseñado en software o hardware
Rapidez de procesamiento:	Generalmente lenta	Rápida
Para su funcionamiento:	Miden un fenómeno físico	Requieren una semilla

Las pruebas estadísticas, como las NIST y TESTU01, pueden aplicarse a cualquier tipo de RNG para verificar si las secuencias generadas superan las pruebas de aleatoriedad.

Como caso particular, los PRNG se pueden diseñar con una amplia variedad de algoritmos matemáticos, desde modelos tan simples como los lineales, hasta más complejos como los no lineales. Dentro de estos últimos, los basados en sistemas caóticos son una opción por tener en cuenta, ya que este tipo de PRNGs explotan la propiedad de la alta sensibilidad a las condiciones iniciales para generar secuencias muy diferentes entre sí, al variar ligeramente la semilla, la cual está formada por los valores de las condiciones iniciales de las variables de estado del sistema caótico [22, 23]. Un ejemplo de sensibilidad a las condiciones iniciales se observa en la figura 1.1, en la cual se grafica la variable de estado x del sistema de Lorenz, cuando las ecuaciones diferenciales ordinarias se solucionan con dos condiciones iniciales muy cercanas, con una variación de una millonésima.

La serie temporal, como la que se muestra en la figura 1.1, puede procesarse para generar una secuencia binaria de números pseudoaleatorios. Tal secuencia binaria se obtiene a partir de la dinámica caótica de las variables de estado, y para ello se pueden utilizar las técnicas de umbral, módulo o del formato de punto fijo [24]. La aleatoriedad de la secuencia generada a partir de un sistema caótico, se evalúa aplicando pruebas estadísticas, como se muestra en el capítulo 3, en la página 47.

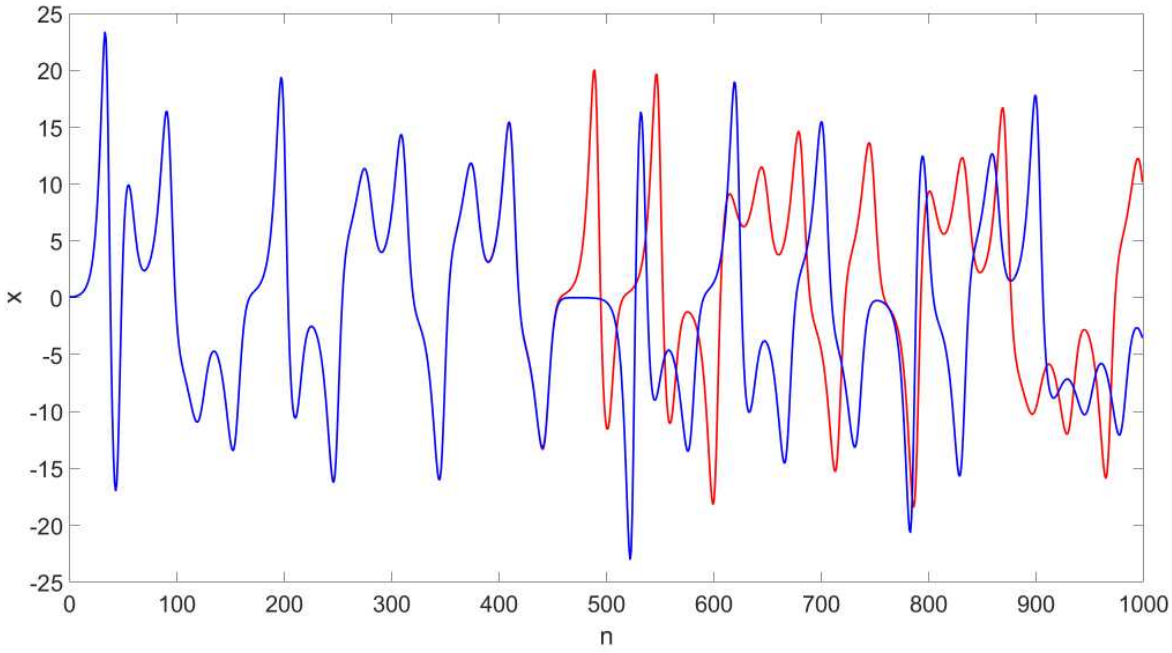


Figura 1.1: Evolución de la serie temporal de la variable de estado x del sistema de Lorenz, utilizando las condiciones iniciales x_0=0.1 (rojo) y x_0=0.100001 (azul).

De esta manera, si se acepta que los PRNGs basados en sistemas caóticos son adecuados para criptografía, entonces pueden aplicarse en la generación de llaves de algunos algoritmos de cifrado y autenticación de mensajes.

1.4. Cifrado y autenticación de mensajes

En la criptografía, los protocolos para comunicaciones seguras se diseñan tomando en cuenta ciertos aspectos claves, dos de estos son la confidencialidad e integridad del mensaje [25,26]. La confidencialidad busca evitar que un intruso (usuario no autorizado) sea capaz de leer un mensaje transmitido o almacenado. Para ello, los protocolos usan algoritmos de cifrado de información. El cifrado consiste en transformar un mensaje legible, llamado texto plano (texto, imágenes, archivos), en un mensaje no legible, denominado texto cifrado, utilizando un algoritmo y una llave antes de su transmisión [26,27]. Existen algoritmos que emplean la misma llave para cifrar y descifrar la información, conocidos como algoritmos de clave simétrica [25]. Estos sistemas pueden llevar a cabo el cifrado utilizando un bloque de n bits (cifrado en bloques) o realizando el proceso bit a bit o byte a byte (cifrado de flujo) [28]. En el caso de los esquemas de cifrado/descifrado de flujo, el

funcionamiento general es el siguiente: un PRNG genera un flujo de números pseudoaleatorios, denominado flujo de llaves, que se combinan byte a byte con el texto plano mediante la operación XOR, y produce el texto cifrado que luego se transmite por un canal de comunicación. Para el proceso de descifrado, se ejecuta nuevamente la operación XOR, pero ahora entre el texto cifrado y el mismo flujo de llaves, para recuperar el texto plano original [25, 26]. En este tipo de sistemas de cifrado, o incluso en otros más complejos, los PRNG basados en caos pueden ser aplicados para proporcionar el flujo de llaves para el algoritmo.

La integridad busca asegurar que el mensaje recibido sea idéntico al enviado y no haya sufrido alteraciones durante su transmisión [25,26]. Para lograrlo, en esta tesis se propone el uso de funciones de hash unidireccionales, conocidas también como resúmenes de mensaje. Las funciones de hash consisten en algoritmos matemáticos que transforman un bloque de texto plano en un conjunto de caracteres alfanuméricos de longitud fija, denominados valor de hash, que representa una etiqueta de autenticación para el mensaje enviado [26]. Algunas características de estas funciones de hash son [26]:

1. Admiten datos de entrada de cualquier longitud.
2. Producen una salida de longitud fija.
3. Son unidireccionales, es decir, dado un valor de hash, es prácticamente imposible recuperar el texto plano original.
4. Ofrecen resistencia a la colisión, lo que significa que dos textos planos diferentes no deberían producir el mismo valor de hash.
5. Un cambio de un solo bit en la entrada genera una salida completamente distinta.

En un sistema de comunicación emisor/receptor que utiliza el resumen de mensaje, el funcionamiento general es el siguiente: primero, el emisor calcula el valor de hash del texto plano y lo envía al receptor junto con el texto cifrado. Luego, el receptor aplica la misma función de hash con el texto plano obtenido por el procedimiento de descifrado. Al final, se comparan ambos valores de hash, y si son diferentes, esto indica que un intruso ha modificado al menos un bit del mensaje. Por lo tanto, la función de hash funciona como un tipo de firma que permite determinar si un mensaje es auténtico [29].

En la literatura, existen varios tipos de funciones de hash muy utilizadas, por ejemplo, SHA-1, SHA-2. Además, se han propuesto funciones de hash unidireccionales que utilizan números pseudoaleatorios para generar las etiquetas de autenticación, por ejemplo, el método del producto pseudo punto [30], donde los PRNG basados en caos pueden encontrar aplicación.

1.5. Acerca de la FPGA y la Raspberry Pi 3

La implementación de sistemas caóticos puede llevarse a cabo empleando recursos de hardware analógico o digital. En el caso particular de un entorno digital, dispositivos como la FPGA y el sistema embebido Raspberry Pi 3 son opciones adecuadas, debido a que ofrecen diversas características y funcionalidades útiles para esta tarea. La FPGA es un dispositivo reconfigurable basado en matrices de bloques lógicos configurables, cuya interconexión se define mediante un lenguaje de descripción de hardware (HDL), como Verilog. Entre sus principales ventajas están el rápido prototipado, la reconfigurabilidad de los diseños, el bajo consumo de recursos y la capacidad de ejecutar múltiples operaciones en paralelo, con el fin de lograr mayores velocidades de procesamiento. El desarrollo de una arquitectura digital en FPGA se basa en la conexión de circuitos digitales combinacionales y secuenciales, desde simples compuertas lógicas hasta circuitos más complejos como contadores, registros de desplazamiento, máquinas de estados, entre muchos otros. La lógica de cada circuito y las conexiones entre estos se describen mediante el lenguaje HDL y se sintetizan en configuraciones específicas dentro de los bloques lógicos configurables, empleando celdas lógicas (o elementos lógicos) que cuentan con: LUTs (Look-Up Tables), flip-flops, registros y multiplexores. Además, los dispositivos FPGA suelen proporcionar otros componentes de hardware digital, como multiplicadores embebidos, bloques de memoria RAM, bucles de amarre de fase (PLL) y otros núcleos de propiedad intelectual [31], para el diseño de sistemas de mayor complejidad.

Por otro lado, el sistema empotrado Raspberry Pi 3 es una computadora de placa única para aplicaciones de propósito general. Este dispositivo es útil en aplicaciones para el Internet de las Cosas, porque tiene la ventaja de contar con una amplia variedad de bibliotecas de programación para manejar protocolos de mensajería como el transporte de telemetría de colas de mensajes (MQTT), y conectividad a Internet. El dispositivo funciona con un sistema operativo basado en Linux y permite programar aplicaciones en lenguajes de alto nivel, como C, Python o Java. Algunas

características del dispositivo son: un procesador de 64 bit ARM Cortex-A53 a 1.2 GHz, 1 GB de memoria RAM y diferentes puertos de entrada y salida de datos [32].

En ambos dispositivos digitales se pueden implementar PRNGs basados en caos para desarrollar sistemas de cifrado y autenticación de mensajes.

1.6. Hipótesis

La implementación de sistemas caóticos con múltiples variables de estado puede mejorar el throughput de un generador de números pseudoaleatorios y aumentar la longitud de las etiquetas de autenticación producidas por una función de hash basada en números pseudoaleatorios.

1.7. Objetivos

1.7.1. Objetivo general

Diseñar un sistema de autenticación de datos utilizando generadores de números pseudoaleatorios basados en sistemas caóticos de distintas dimensiones, los cuales servirán para aumentar la longitud de las etiquetas de autenticación de una función de hash.

1.7.2. Objetivos específicos

- Diseñar generadores de números pseudoaleatorios basados en sistemas caóticos de dos, tres y cuatro dimensiones.
- Garantizar la aleatoriedad de las secuencias generadas por los generadores de números pseudoaleatorios mediante pruebas estadísticas NIST.
- Implementar en FPGA y Raspberry Pi los tres PRNGs basados en sistemas caóticos 2D, 3D y 4D.
- Describir en un lenguaje de hardware y sintetizar en una FPGA un sistema de cifrado y autenticación empleando una función de hash, basada en generadores de números pseudoaleatorios.

- Implementar en Raspberry Pi un sistema de cifrado y autenticación usando la función de hash y empleando el protocolo MQTT para la transmisión de datos.
- Autenticar imágenes a escala de grises, RGB y texto.

1.8. Organización de la tesis

Esta tesis está organizada en 6 capítulos. El capítulo 1, describe de forma general el objetivo de esta investigación. En el capítulo 2, se analizan las características de tres sistemas caóticos: un mapa en dos dimensiones (2D), y dos sistemas caóticos en tiempo continuo en 3D y 4D, cuya solución numérica emplea el método de Euler hacia adelante. Además, se presentan las series temporales y los atractores de cada sistema caótico. En el capítulo 3, se describe el diseño a bloques de un sistema caótico y su descripción en Verilog para su síntesis en un dispositivo FPGA. Se explican las técnicas para la generación de las secuencias pseudoaleatorias a partir de los sistemas caóticos y se muestran los resultados de las pruebas estadísticas NIST para cada PRNG. El capítulo 4, detalla los algoritmos para cifrar y autenticar imágenes y texto. El algoritmo de cifrado emplea la operación XOR, mientras que el de autenticación utiliza una función de hash, basada en el producto pseudo punto. Además, se describe la implementación de un sistema de comunicación segura que incluye las etapas de cifrado y autenticación de mensajes. De manera similar, el capítulo 5, aborda la implementación en Raspberry Pi 3 de los PRNGs y el esquema de comunicación segura, que incluye el protocolo MQTT para la transmisión y recepción de datos entre dos dispositivos Raspberry Pi. Por último, el capítulo 6 muestra las conclusiones y trabajos a futuro.

Capítulo 2

Sistemas caóticos

En este capítulo se describen tres sistemas caóticos con atractores autoexitados y dimensiones que abarcan 2D, 3D, y 4D. El primer caso de estudio es un mapa caótico 2D capaz de proporcionar cuatro diferentes comportamientos caóticos distintos, dependiendo de los valores seleccionados en los parámetros de las ecuaciones discretas. El segundo objeto de estudio aborda un sistema caótico 3D basado en un modelo de memristor, que bajo ciertos parámetros exhibe un comportamiento caótico y muestra multiestabilidad. Por último, el tercer caso de estudio presenta un sistema 4D de tiempo continuo, capaz de producir un comportamiento caótico y multiestabilidad, entre otras propiedades. Además, se proporcionan simulaciones de series temporales, los atractores obtenidos en el espacio de fases y otras características clave, como los puntos de equilibrio, valores propios y exponentes de Lyapunov. Cabe destacar que la solución de los sistemas de tiempo continuo emplea el método numérico de Euler hacia adelante, elegido por su simplicidad y bajo número de operaciones computacionales.

2.1. Descripción de los sistemas caóticos

2.1.1. Mapas caóticos

Los mapas caóticos son modelos matemáticos compuestos por una o varias ecuaciones discretas que exhiben un comportamiento caótico. Típicamente, se definen por una regla recursiva de la forma $x_{n+1} = f(x_n)$, donde $f(x_n)$ es una función que determina el valor siguiente (x_{n+1}) a partir del actual (x_n) [33]. Por ejemplo, un mapa caótico, ampliamente estudiado, es el mapa

logístico de una dimensión. Este mapa 1D, demuestra que el comportamiento caótico se origina con una simple función discreta no lineal, la cual se muestra en la ecuación (2.1).

$$x_{n+1} = r x_n (1 - x_n) \tag{2.1}$$

Los mapas generan nuevos valores en cada iteración $(n + 1)$, cuyas operaciones iterativas pueden ser sintetizadas en hardware digital. Debido a la alta sensibilidad a las condiciones iniciales, variar una millonésima dos de estas, provoca que las series temporales diverjan significativamente, produciendo diferentes trayectorias al evolucionar en tiempo, tal como sucede con los sistemas de tiempo continuo, cuyo ejemplo se observa en la figura 1.1. Una técnica común que ayuda a encontrar los rangos de los parámetros donde se genera comportamiento caótico en un mapa o sistema dinámico, son los diagramas de bifurcación, los cuales permiten visualizar cómo la variación de un parámetro puede propiciar comportamientos caóticos o converger a valores específicos.

En la literatura, existe una gran variedad de mapas caóticos con distintas funciones no lineales y de diferentes dimensiones. Muchos de estos se han aplicado en la generación de números pseudoaleatorios y en esquemas de comunicaciones seguras para el cifrado de datos. Por ejemplo, entre los mapas unidimensionales más comunes se encuentran el de corrimiento de Bernoulli, el mapa tienda de campaña y el Zigzag. En el caso de mapas bidimensionales, destacan los mapas de Henon, Arnold 2D y el Baker, entre otros [34]. Hoy en día, muchos investigadores continúan desarrollando nuevos modelos novedosos de mapas caóticos en 3D, 4D, 5D, etc., con mejores propiedades de aleatoriedad en las secuencias binarias y con mayores exponentes positivos de Lyapunov.

En la siguiente subsección se detallan algunas propiedades de un mapa caótico bidimensional, que es un caso de estudio en esta tesis para el desarrollo de un PRNG.

2.1.2. Mapa caótico 2D de Sprott

El mapa caótico 2D, propuesto por Sprott en [35], está definido por dos variables de estado, designadas como x_n y y_n. Las expresiones matemáticas que describen la dinámica de este mapa se presentan en la ecuación (2.2). En ellas, se observa que los valores de las variables de estado en la siguiente iteración $(n + 1)$ se calculan mediante operaciones de adición y multiplicación,

utilizando los valores actuales de las variables de estado y un conjunto de parámetros constantes, denotados como a_i.

$$\begin{aligned} x_{n+1} &= a_1 + a_2 x_n + a_3 x_n^2 + a_4 x_n y_n + a_5 y_n + a_6 y_n^2 \\ y_{n+1} &= a_7 + a_8 x_n + a_9 x_n^2 + a_{10} x_n y_n + a_{11} y_n + a_{12} y_n^2 \end{aligned} \tag{2.2}$$

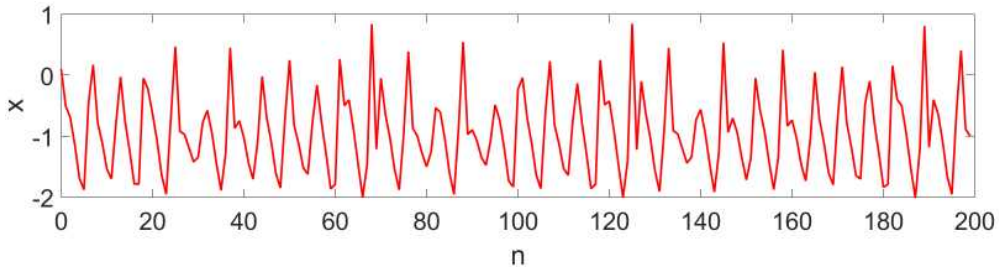
Este mapa caótico puede exhibir diversas dinámicas en función de los valores seleccionados para los coeficientes a_i. Por ello, en el trabajo [30], se proporcionan un conjunto de coeficientes que dan lugar a cuatro atractores distintos en el espacio de fases. Estos valores se resumen en la tabla 2.1.

Tabla 2.1: Coeficientes a_i para diversas dinámicas caóticas con el mapa caótico 2D.

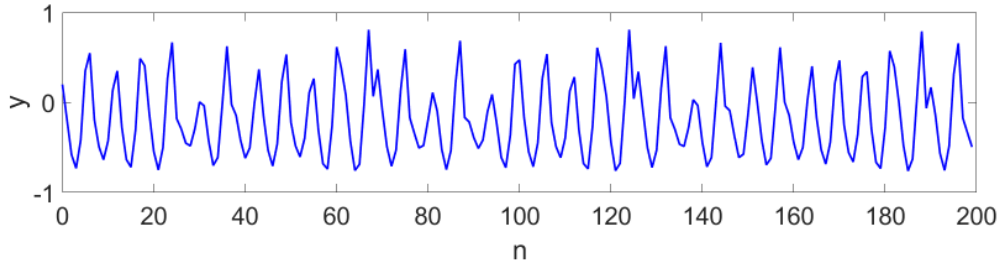
Mapa	**Valores de los coeficientes**											
	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
1	-0.6	-0.1	1.1	0.2	-0.8	0.6	-0.7	0.7	0.7	0.3	0.6	0.9
2	-1.0	0.9	0.4	-0.2	-0.6	-0.5	0.4	0.7	0.3	-0.5	0.7	-0.8
3	0.8	1.0	-1.2	-1.0	1.1	-0.9	0.4	-0.4	-0.6	-0.2	-0.5	-0.7
4	-0.6	-0.4	-0.4	-0.8	0.7	0.3	-0.4	0.4	0.5	0.5	0.8	-0.1

2.1.2.1. Simulación de las series temporales y atractores

Dado que los mapas caóticos están basados en ecuaciones de diferencias discretas, son fáciles y rápidos de implementar en entornos digitales. Estas ecuaciones se pueden programar y simular utilizando lenguajes de alto nivel, como Python, C++, Java, entre otros. En el caso del mapa 2D de Sprott, en esta tesis la simulación se llevó a cabo en el software de MATLAB® empleando aritmética de punto flotante. Para simular las series temporales, se utilizaron las condiciones iniciales $x_0 = 0.1$ y $y_0 = 0.2$. Como ejemplo, la figura 2.1 ilustra las series temporales simuladas con el conjunto de coeficientes a_i que forman el mapa 4 (ver tabla 2.1). En dicha simulación numérica se realizaron 200 iteraciones. De manera similar, la figura 2.2 ilustra los cuatro atractores distintos en el espacio de fases, formados utilizando el grupo de coeficientes a_i de la tabla 2.1. Todos los atractores caóticos se graficaron en el espacio de fases, realizando 50000 iteraciones en cada simulación.



(a)



(b)

Figura 2.1: Series temporales del mapa 2D para las variables de estado a) x_n, b) y_n, con los coeficientes a_i del mapa 4 mostrados en la tabla 2.1.

2.1.2.2. Exponentes de Lyapunov y dimensión fractal

En el caso de los exponentes de Lyapunov y la dimensión fractal, la tabla 2.2 (tomada de [35]) agrupa los resultados. En ella se observa que, aunque el mapa 4 muestra la mayor sensibilidad a las condiciones iniciales debido a su mayor exponente positivo de Lyapunov, presenta el atractor menos complejo en términos de la dimensión fractal. Por otro lado, el mapa 1, tiene la menor sensibilidad a las condiciones iniciales, pero muestra el atractor más complejo en la dimensión fractal.

Tabla 2.2: Exponentes de Lyapunov y la dimensión fractal para los diferentes conjuntos de coeficientes a_i del mapa.

Mapa	Exponente positivo de Lyapunov	Dimensión Fractal
1	0.12	1.77
2	0.14	1.79
3	0.15	1.69
4	0.16	1.50

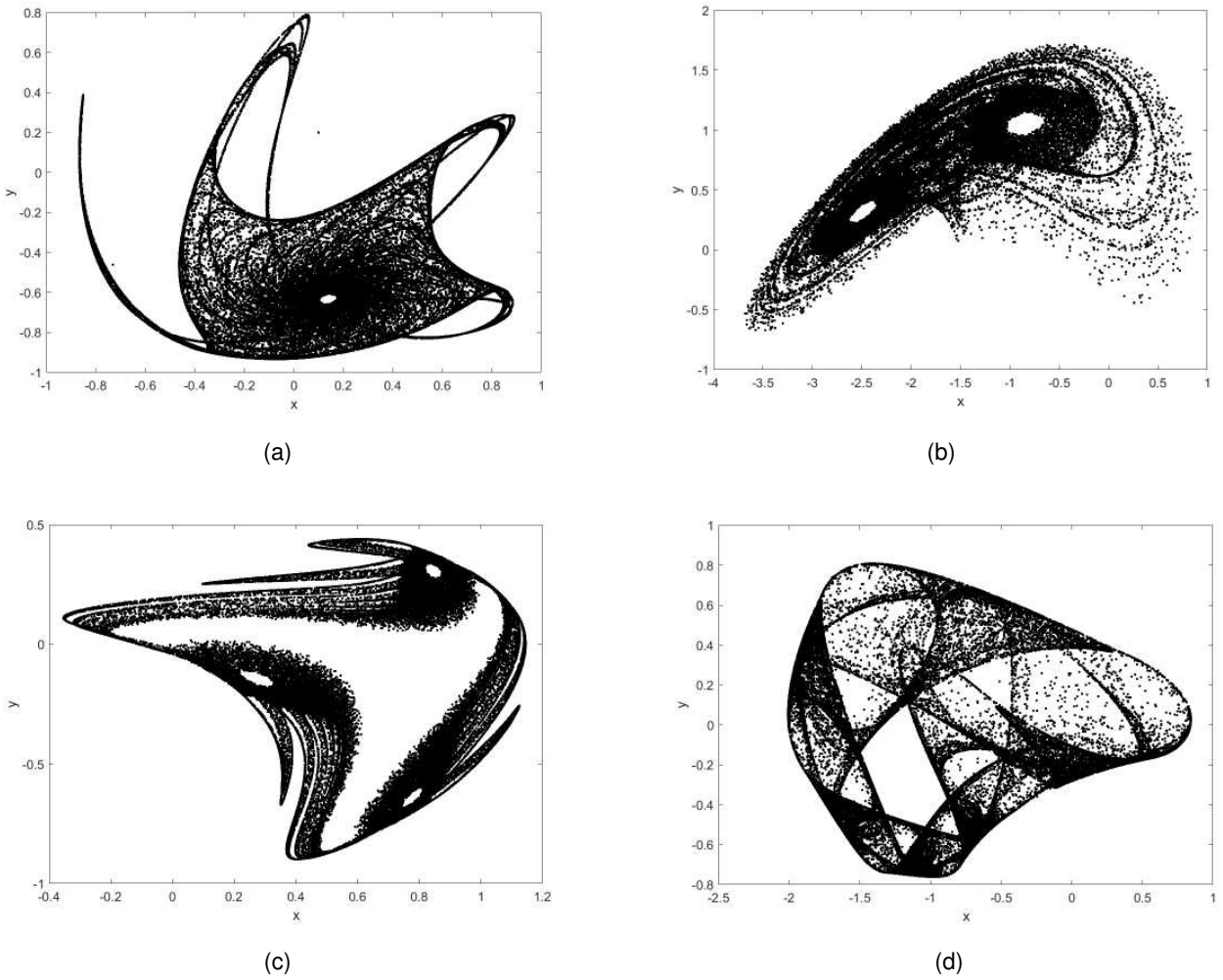


Figura 2.2: Retratos de fase $x_n - y_n$ con los diferentes conjuntos de coeficientes a_i de la tabla 2.1: (a) mapa 1, (b) mapa 2, (c) mapa 3, (d) mapa 4.

2.1.3. Sistemas caóticos en tiempo continuo

Los sistemas caóticos de tiempo continuo se modelan por ecuaciones diferenciales ordinarias de orden entero o fraccionario. A diferencia de los mapas caóticos, estos modelos matemáticos requieren métodos numéricos para ser resueltos e implementados en entornos digitales. En la literatura, un sistema caótico de orden entero muy popular y estudiado es el sistema de Lorenz, publicado en 1963 en [36], cuyas tres ecuaciones diferenciales ordinarias se muestran en la ecuación (2.3).

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - xz - y \\ \dot{z} &= xy - \beta z \end{aligned} \tag{2.3}$$

Como puede observarse, el sistema de Lorenz de orden entero está formado por operaciones de resta y multiplicación, donde la no linealidad se introduce a través de los productos de las variables de estado xz y xy. Además, en sus ecuaciones se pueden observar los parámetros con valores constantes σ, ρ y β. Estos parámetros son esenciales, ya que una elección apropiada de sus valores puede dar lugar a series temporales con mejor comportamiento caótico. Una región caótica muy estudiada se encuentra con los parámetros establecidos en los valores 10, 28 y $8/3$, respectivamente.

Hoy en día, existe una extensa investigación para desarrollar nuevos sistemas caóticos en tiempo continuo, con diferentes dimensiones, como 3D, 4D, 5D, etc. Incluso, algunos de ellos están basados en modelos neuronales y de memristores. Las siguientes subsecciones detallan dos sistemas caóticos en tiempo continuo 3D y 4D, que son útiles en el desarrollo de PRNGs.

2.1.4. Sistema caótico 3D basado en un memristor

El memristor es considerado el cuarto componente eléctrico fundamental, cuyas bases teóricas fueron establecidas por León O. Chua en 1971 [37]. Debido a las dificultades en el desarrollo físico del componente, muchos investigadores han optado por crear emuladores del memristor, los cuales pueden ser implementados en sistemas de hardware analógico y digital. Asignando los valores apropiados a los parámetros, los sistemas dinámicos basados en memristores tienen la capacidad de exhibir el comportamiento caótico. Un ejemplo de ello es el sistema caótico memristivo de tres variables de estado introducido en [38], que también presenta la propiedad de multiestabilidad, la cual se refiere a la coexistencia de múltiples atractores en un sistema con los mismos parámetros fijos, pero con diferentes condiciones iniciales [39]. El modelo matemático de este sistema es presentado en la ecuación (2.4).

$$\begin{aligned} \dot{x} &= ayz, \\ \dot{y} &= x - y \\ \dot{z} &= k - b(\gamma xy - \delta xy^3) \end{aligned} \tag{2.4}$$

La dinámica caótica de este sistema y la propiedad de multiestabilidad aparecen con los parámetros establecidos como: $a = 3.5$, $k = 1.2$, $b = 1.8$, $\gamma = 2$, $\delta = 0.1$.

2.1.4.1. Puntos de equilibrio y valores propios

En el análisis de estabilidad, los valores propios o eigenvalores del sistema se obtienen a partir de la matriz jacobiana [38], como se muestra en la ecuación (2.5). El sistema cuenta con cuatro puntos de equilibrio y sus correspondientes valores propios se presentan en la tabla 2.3.

$$J(x^\star, y^\star, z^\star) = \begin{pmatrix} 0 & az^\star & ay^\star \\ 1 & -1 & 0 \\ -b(\gamma y^\star - \delta y^{\star 3}) & -b(\gamma x^\star - 3\delta x^\star y^{\star 2}) & 0 \end{pmatrix} \tag{2.5}$$

Tabla 2.3: Puntos de equilibrio y sus correspondientes valores propios.

Puntos de equilibrio	**Valores propios**
$(0.58, 0.58, 0)$	$(1.01, 0, -2.01)$
$(-0.58, -0.58, 0)$	$(0, -0.5 + 1.34i, -0.5 - 1.34i)$
$(4.43, 4.43, 0)$	$(3.47, 0, -4.47)$
$(-4.43, -4.43, 0)$	$(0, -0.5 + 3.90i, -0.5 - 3.90i)$

2.1.4.2. Simulación de las series temporales y atractores

Para solucionar y obtener las series temporales y los atractores de este sistema de tiempo continuo, se requiere aplicar un método numérico. Después de seleccionar el método de Euler hacia adelante y emplearlo en las ecuaciones diferenciales ordinarias dadas en la ecuación (2.4), se obtiene el sistema de ecuaciones discretas mostrado en la ecuación (2.6).

$$
\begin{aligned}
x_{n+1} &= x_n + h(a y_n z_n), \\
y_{n+1} &= y_n + h(x_n - y_n), \\
z_{n+1} &= z_n + h(k - b\gamma x_n y_n - b\delta x_n y_n^3),
\end{aligned}
\tag{2.6}
$$

El sistema de ecuaciones anterior puede ser implementado haciendo uso de elementos de hardware digital. Para obtener la respuesta caótica, se estableció un ancho de paso h de 0.01025, mientras que las condiciones iniciales se fijaron en $x_0 = 0.2$, $y_0 = 0.1$ y $z_0 = 0$.

La figura 2.3 muestra las series temporales de las tres variables de estado, efectuando una simulación de 5000 iteraciones. De manera similar, la figura 2.4 ilustra los atractores resultantes al simular el sistema con 100000 iteraciones.

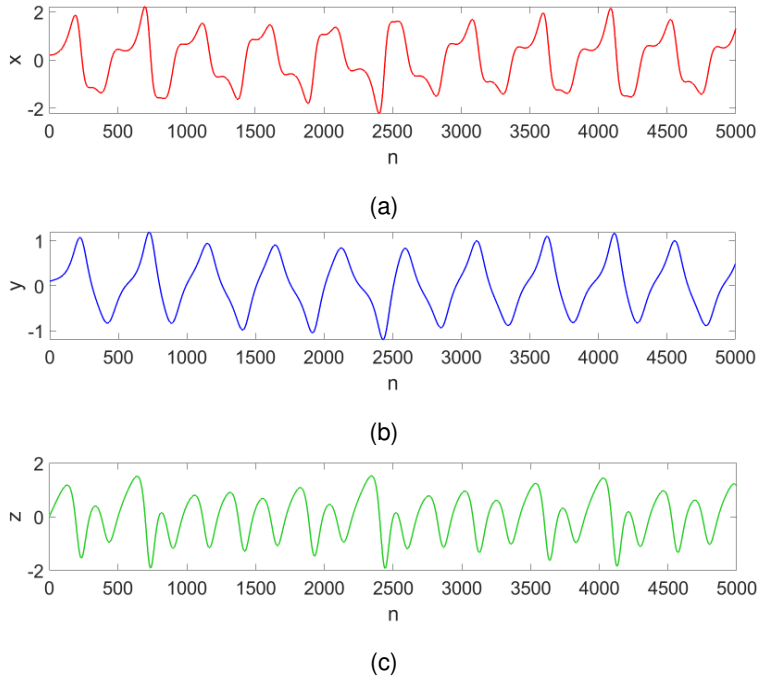


Figura 2.3: Series temporales del sistema 3D para las variables de estado: a) x_n, b) y_n, c)z_n, con parámetros establecidos como: $h = 0.01025$, $a = 3.5$, $k = 1.2$, $b = 1.8$, $\gamma = 2$ y $\delta = 0.1$.

2.1.4.3. Exponentes de Lyapunov y dimensión Kaplan-Yorke

En la tabla 2.4, se muestran los exponentes de Lyapunov y la dimensión Kaplan-Yorke obtenidos para el sistema 3D basado en un memristor.

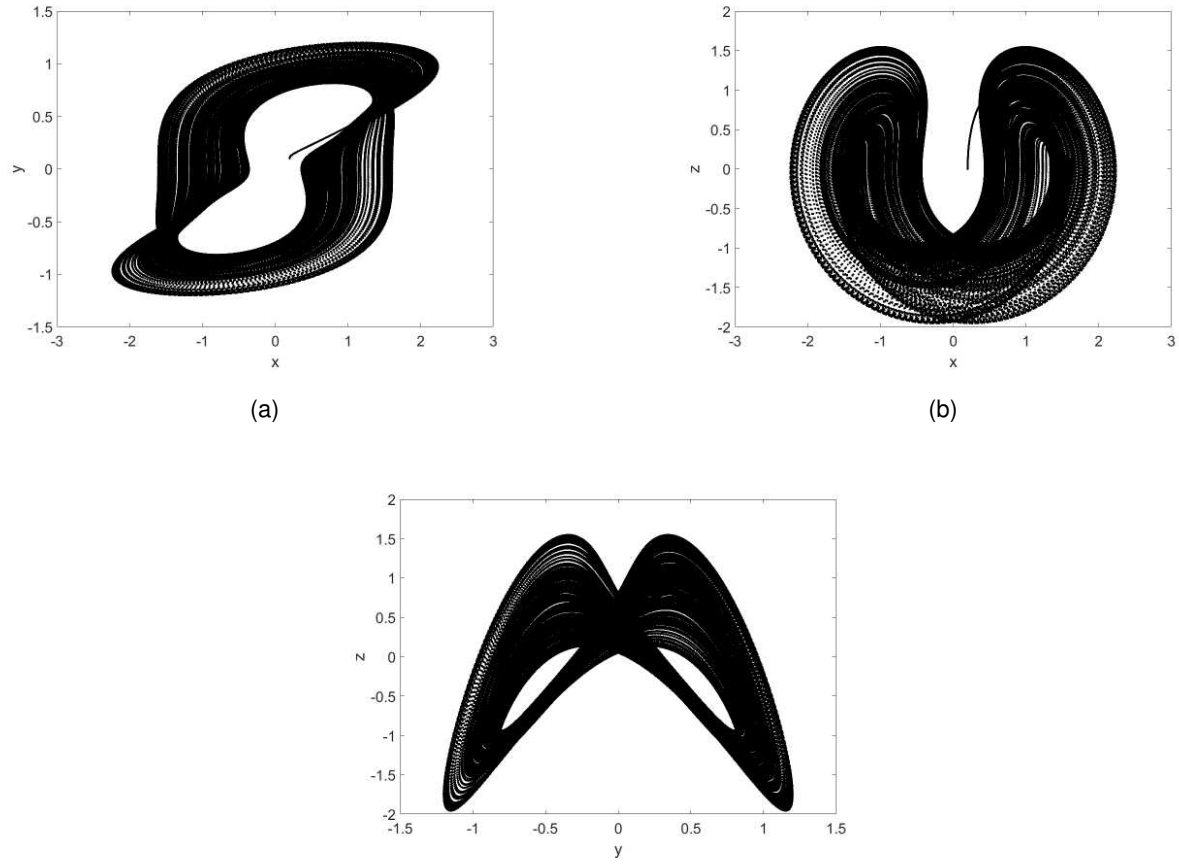


Figura 2.4: Retratos de fase del sistema 3D: (a) $x - y$, (b) $x - z$, (c) $y - z$.

Tabla 2.4: Exponentes de Lyapunov y la dimensión Kaplan-Yorke para el sistema 3D.

Exponentes de Lyapunov	Dimensión Kaplan-Yorke
$LE_1 = 0.22$ $LE_2 = 0$ $LE_3 = -1.18$	2.1864

2.1.5. Sistema caótico 4D

Este sistema caótico, propuesto por Sambas et al. en [40], está conformado por cuatro ecuaciones diferenciales ordinarias, por lo cual se clasifica como un sistema "hyperjerk". El modelo

matemático contiene ocho términos, en los cuales aparece una función de valor absoluto y dos parámetros constantes positivos, a y b. Además, este sistema cuenta con multiestabilidad y exhibe un comportamiento antimonotónico. Las ecuaciones que describen a este sistema se muestran en la ecuación (2.7).

$$\begin{cases} \dot{x} = y, \\ \dot{y} = z, \\ \dot{z} = w, \\ \dot{w} = -x - |x| - ay - bw - xz, \end{cases} \tag{2.7}$$

El comportamiento caótico para este sistema surge cuando los parámetros se fijan en: $a = 4$ y $b = 2$. La no linealidad de este sistema se introduce por el producto xz y la función valor absoluto $|x|$.

2.1.5.1. Puntos de equilibrio

Los puntos de equilibrio se obtienen al resolver el sistema estableciendo todas las variables de estado en cero en la ecuación (2.7). Al llevar a cabo el análisis, el conjunto de puntos de equilibrio S se simplifica como: $S = \{(x,\, y,\, z,\, w) |\, x \leq 0,\, y = 0,\, z = 0,\, w = 0\}$, que corresponde a una semirrecta sobre el eje x negativo en $\mathbf{R}^4$ [40].

2.1.5.2. Simulación de las series temporales y atractores

Al aplicar el método de Euler hacia adelante al modelo matemático mostrado en la ecuación (2.7), se obtiene el conjunto de expresiones matemáticas discretas presentadas en la ecuación (2.8), las cuales están listas para ser programadas en un lenguaje de alto nivel.

$$\begin{aligned} x_{n+1} &= x_n + h(y_n) \\ y_{n+1} &= y_n + h(z_n) \\ z_{n+1} &= z_n + h(w_n) \\ w_{n+1} &= w_n + h(-x_n - |x_n| - ay_n - bw_n - x_n z_n) \end{aligned} \tag{2.8}$$

Las series temporales se obtuvieron considerando un ancho de paso $h = 0.02$, y estableciendo las cuatro condiciones iniciales en 0.2.

La figura 2.5 muestra las series temporales de las cuatro variables de estado, al realizar una simulación con 10000 iteraciones. Por otro lado, la figura 2.6 muestra cuatro retratos de fase al simular el sistema con 100000 iteraciones.

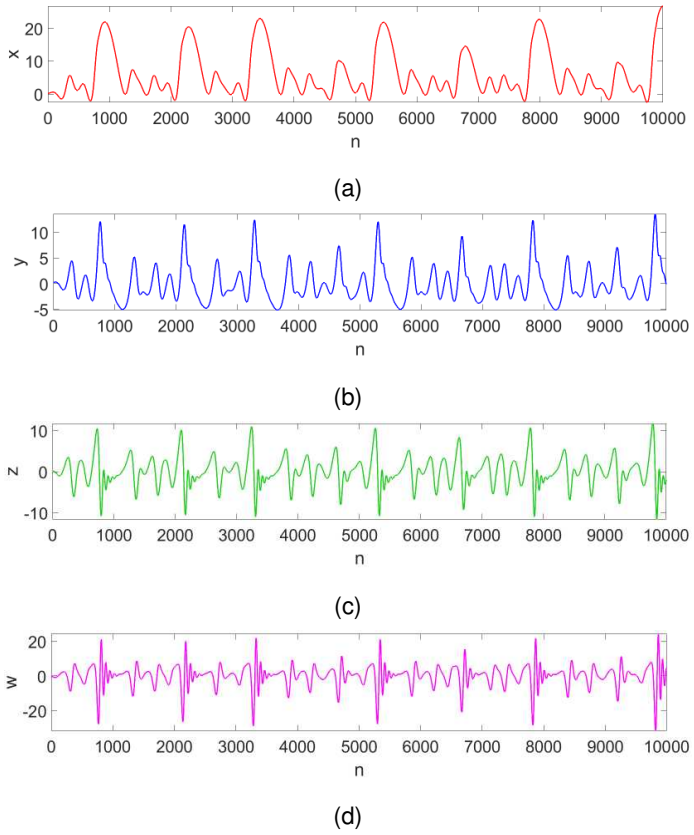


Figura 2.5: Series temporales del sistema 4D para las variables de estado: a) x_n, b) y_n, c)z_n, d)w_n, con los parámetros establecidos como: $h = 0.02$, $a = 4$ y $b = 2$.

2.1.5.3. Exponentes de Lyapunov y dimensión Kaplan-Yorke

Para obtener los exponentes de Lyapunov y la dimensión Kaplan-Yorke D_{KY} del sistema 4D, se utilizó la aproximación de Wolf [40]. La tabla 2.5 muestra los resultados.

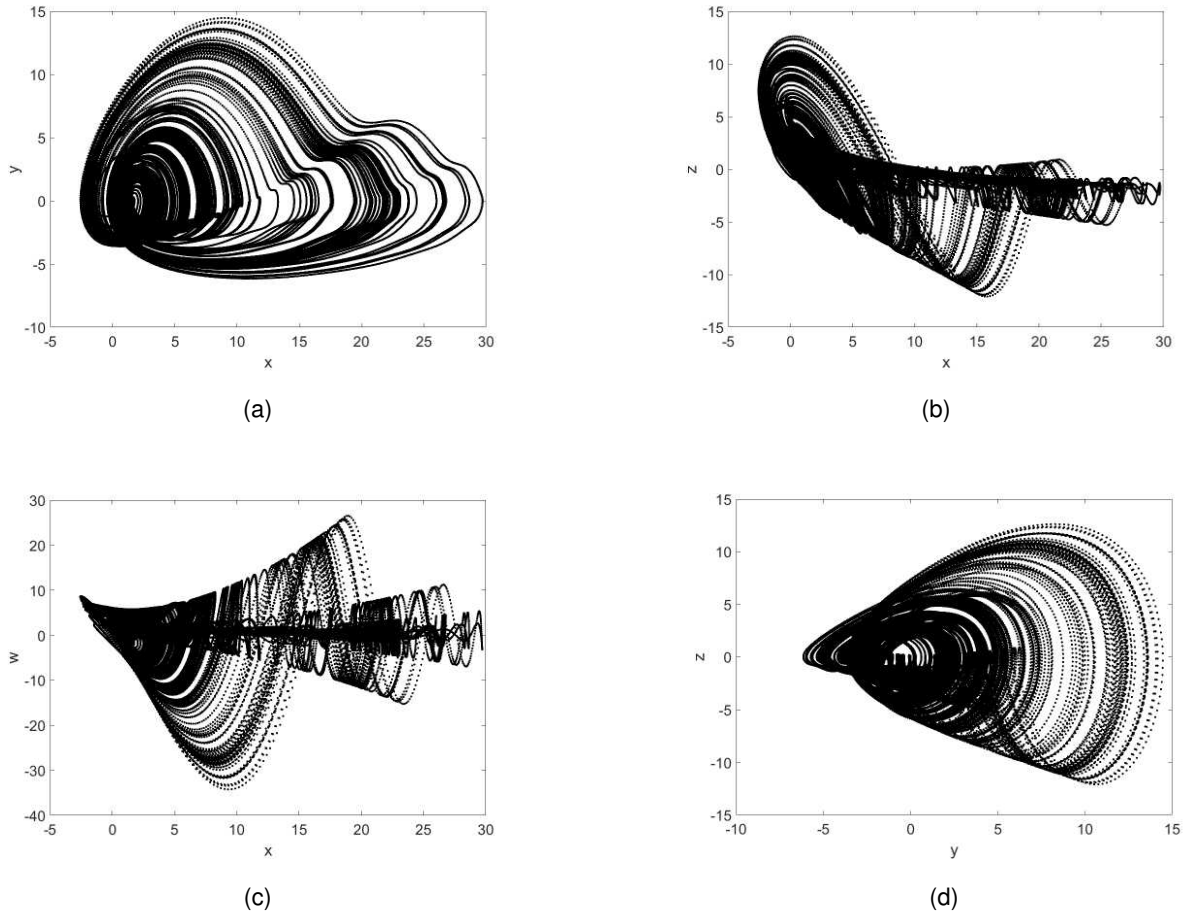


Figura 2.6: Retratos de fase del sistema 4D: (a) $x - y$, (b) $x - z$, (c) $x - w$, (d) $y - z$.

Tabla 2.5: Exponentes de Lyapunov y la dimensión Kaplan-Yorke para el sistema 4D.

Exponentes de Lyapunov	Dimensión Kaplan-Yorke
$LE_1 = 0.1214$ $LE_2 = 0$ $LE_3 = -0.7292$ $LE_4 = -1.3924$	2.1665

Capítulo 3

Descripción y síntesis de sistemas caóticos y PRNGs en FPGAs

Las ecuaciones matemáticas discretas que gobiernan el comportamiento de un sistema dinámico caótico pueden encontrar aplicaciones prácticas en la generación de números pseudoaleatorios y en esquemas criptográficos al implementarse en entornos digitales, que abarcan computadoras, sistemas embebidos, microcontroladores y FPGAs. Por ello, este capítulo se centra en la descripción mediante bloques digitales de los tres sistemas caóticos descritos en el capítulo anterior, empleando el lenguaje de descripción de hardware de Verilog. Además, se presentan simulaciones de las series temporales producidas por cada sistema, su síntesis en una FPGA de Intel/Altera, los atractores experimentales obtenidos, el uso de los sistemas como PRNGs y los resultados de las pruebas estadísticas NIST de los bits pseudoaleatorios de cada sistema. La descripción de las ecuaciones mediante un lenguaje de hardware, se lleva a cabo considerando la conexión de múltiples bloques digitales combinacionales y secuenciales, que incluyen elementos lógicos y aritméticos. Estos bloques, al conectarse, forman una arquitectura digital más compleja que finalmente se sintetiza en una FPGA por medio de los conocidos bloques lógicos configurables que conforman a este dispositivo. La FPGA destaca por su capacidad de procesamiento en paralelo, que permite la ejecución de múltiples procedimientos o tareas en un mismo ciclo de reloj, favoreciendo a un incremento en la velocidad de procesamiento del sistema. Este dispositivo también permite verificar si un diseño basado en bloques digitales funciona correctamente, cuando todos los elementos de hardware trabajan en conjunto.

Cabe mencionar que cada descripción y simulación se realizó en el software de Active-HDL 15, mientras que la síntesis para FPGA se efectuó en el software de Quartus-II.

3.1. Cuestiones sobre la aritmética de computadora

Los sistemas caóticos descritos con ecuaciones discretas, ya sean mapas o sistemas en tiempo continuo discretizados mediante un método numérico, generalmente suelen estar expresados por operaciones aritméticas simples que trabajan con cantidades con punto decimal. Estas operaciones se pueden implementar y sintetizar en FPGA utilizando bloques digitales tales como, sumadores, restadores y multiplicadores. No obstante, es necesario seleccionar una aritmética de computadora adecuada para la ejecución de estas operaciones en los bloques digitales. Las aritméticas disponibles son la de punto flotante y la de punto fijo. La aritmética de punto flotante ofrece una mayor precisión y un rango más amplio para expresar números grandes o pequeños. Sin embargo, los algoritmos para su implementación suelen ser complejos de realizar. Por otro lado, la aritmética de punto fijo es más sencilla y rápida de implementar [41], ya que solo requiere repartir el número total de bits de la aritmética en tres partes: un bit de signo para indicar si el número es positivo o negativo, un conjunto de bits para la parte entera y otro para la parte fraccionaria. La precisión decimal depende de la cantidad de bits asignados a la parte fraccionaria, la cual mejora al aumentar la cantidad de ellos. No obstante, su desventaja es que el rango de operación está limitado al número de bits asignados a la parte entera del formato.

Por lo anterior, cuando se realizan descripciones de sistemas caóticos en Verilog o cualquier otro lenguaje de descripción de hardware, es fundamental llevar a cabo un análisis previo para estimar el rango de operación, es decir, determinar los valores máximos y mínimos que alcanzan las variables de estado y las operaciones intermedias de cada sistema dinámico [31], con el fin de evitar desbordamientos en los bloques aritméticos. Este análisis puede llevarse a cabo en un lenguaje de alto nivel.

En la literatura, los sistemas caóticos implementados en FPGA con aritmética de punto fijo suelen utilizar 16, 24, 32, 40 e incluso 64 bits. Aunque un mayor número de bits mejora significativamente la precisión decimal, también incrementa el consumo de recursos, ya que los bloques aritméticos deben trabajar con más bits. No obstante, dado que estos sistemas caóticos se utilizan para el desarrollo de PRNGs, es preferible emplear una aritmética con muchos bits y así

contar con un mayor número de condiciones iniciales o semillas para el PRNG. Por ello, los tres sistemas caóticos descritos en el capítulo anterior, se diseñan utilizando aritmética de punto fijo de 64 bits, donde cada uno de los sistemas tiene su propia distribución de bits para el formato.

3.2. Descripciones de sistemas caóticos

El lenguaje de Verilog permite la descripción de circuitos digitales combinacionales y secuenciales, como multiplexores, decodificadores, sumadores, multiplicadores, registros, contadores, entre otros, a un alto nivel de abstracción. Esto significa que Verilog describe el comportamiento de un bloque digital junto con sus puertos de entrada y salida de datos, y los pines de control, sin proporcionar detalles a un nivel más bajo, como el diseño basado en transistores. Cada bloque puede visualizarse como una caja negra, donde se conocen sus puertos y su comportamiento, pero no sus elementos internos. Además, Verilog permite crear arquitecturas digitales de forma jerárquica, en las cuales un bloque digital de mayor jerarquía se compone mediante la conexión interna de múltiples bloques simples [42]. Estas estructuras jerárquicas son posibles gracias a la instanciación de bloques, que consiste en la creación de copias de un bloque definido previamente. De esta manera, un bloque puede repetirse las veces que sea necesario dentro de otro bloque de mayor jerarquía.

En las siguientes subsecciones se presenta la descripción de los tres sistemas caóticos mediante diagramas de bloques, los cuales aprovechan el diseño jerárquico de Verilog. Primero se describe el sistema 3D, dado que es sencillo de implementar en 64 bits.

3.2.1. Descripción del sistema 3D basado en un memristor

3.2.1.1. Análisis de amplitudes para el formato de punto fijo

El análisis de amplitudes máximas y mínimas de las variables de estado es esencial para determinar el rango de valores en los cuales el sistema está acotado, y con ello definir de forma adecuada una distribución del formato de punto fijo para la parte entera y fraccionaria. Aunque el análisis se puede realizar considerando condiciones iniciales fijas, esto no garantiza que otros valores de ellas produzcan series temporales con amplitudes dentro del rango establecido. Por dicha razón, lo mejor es ejecutar múltiples pruebas con diferentes conjuntos de condiciones iniciales.

El sistema 3D se simuló en un lenguaje de alto nivel, utilizando los parámetros especificados en el capítulo 2 y considerando condiciones iniciales dentro del rango observado en los retratos de fase de la figura 2.4. En cada simulación de 100000 iteraciones, se despreciaron las primeras 20000, para eliminar el tiempo transitorio y obtener series temporales diferentes. Los rangos aproximados obtenidos para cada variable fueron: $x \in [-2.2, 2.2]$, $y \in [-1.2, 1.2]$ y $z \in [-1.9, 1.5]$. Dichos resultados coinciden con las amplitudes observadas en las gráficas de los atractores. A partir de estos rangos, se pueden definir posibles condiciones iniciales para el sistema y determinar las amplitudes de los productos intermedios en las ecuaciones, los cuales, sin considerar el transitorio, están dentro del rango $\pm$ 15. Por lo cual, 4 bits de parte entera en el formato de punto fijo serían suficientes. Sin embargo, también es necesario tomar en cuenta el comportamiento de las series caóticas durante el tiempo transitorio, ya que con ciertas condiciones iniciales podrían presentarse amplitudes mayores. Por ejemplo, en este sistema, al establecer condiciones iniciales muy pequeñas como: $x_0 = 1 \times 10^{-8}$ $y_0 = -1 \times 10^{-8}$; $z_0 = 0$, se forma el atractor al evolucionar el tiempo, pero las amplitudes durante el transitorio crecen significativamente, provocando que los productos alcancen valores aproximados a $\pm$ 128. Esto ocasionaría la pérdida de la secuencia caótica si solo se utilizaran 4 bits de parte entera. Por ello, una mejor opción es emplear 7 bits.

La tabla 3.1 muestra la distribución de punto fijo seleccionada para este sistema en arquitectura de 64 bits. Con esta distribución se tiene una resolución de 1.38×10^{-17}, lo cual es suficiente para mantener el atractor extraño.

Tabla 3.1: Distribución de la aritmética de punto fijo con 64 bits para el sistema 3D.

Signo	Parte entera	Parte fraccionaria
1 bit	7 bits	56 bits

3.2.1.2. Descripción del sistema mediante bloques digitales

Una vez definido el formato de punto fijo, el siguiente paso es llevar a cabo un análisis de las ecuaciones del sistema. En este caso, las expresiones matemáticas presentadas en la ecuación (2.6) consisten solamente en operaciones de suma, resta y multiplicación. Los bloques aritméticos digitales que realizan estas operaciones son circuitos combinacionales, es decir, no dependen de una señal de reloj para su funcionamiento. En lugar de ello, cada bloque requiere un breve tiempo

para producir un resultado, conocido como tiempo de propagación, que depende de la tecnología de fabricación de la FPGA. El tiempo de propagación suele ser del orden de los nanosegundos o incluso menor y se puede entender como el tiempo necesario para que la salida de un bloque combinacional produzca un resultado en respuesta a un cambio en las entradas. Sin embargo, cuando dos o más bloques combinacionales se conectan uno tras otro, el tiempo total para obtener un resultado está determinado por la suma de los tiempos de propagación de cada bloque individual. A medida que este tiempo se incrementa, la frecuencia máxima de reloj que puede operar en el sistema para controlar otros circuitos secuenciales, se reduce significativamente. Una manera de evitar esta reducción en la frecuencia de reloj, consiste en segmentar cada ruta de datos añadiendo registros de entrada y salida paralela (PIPO) entre la conexión de dos bloques combinacionales [43]. De esta forma, cada salida de un bloque combinacional se conecta con un registro PIPO, convirtiéndolo en un bloque secuencial. Como ejemplo, la figura 3.1 muestra el circuito interno del bloque sumador, que está compuesto por el sumador combinacional y el registro PIPO para convertirlo en un circuito secuencial, el cual cuenta con un pin de habilitación (ENB). En dicho esquema, las líneas punteadas indican el bloque de mayor jerarquía denominado "Sumador". Por su parte, las líneas gruesas en color gris representan los buses de datos de 64 bits para las operaciones, mientras que las líneas negras son los pines de control para la señal de reloj (CLK) y de reinicio o reset (RST).

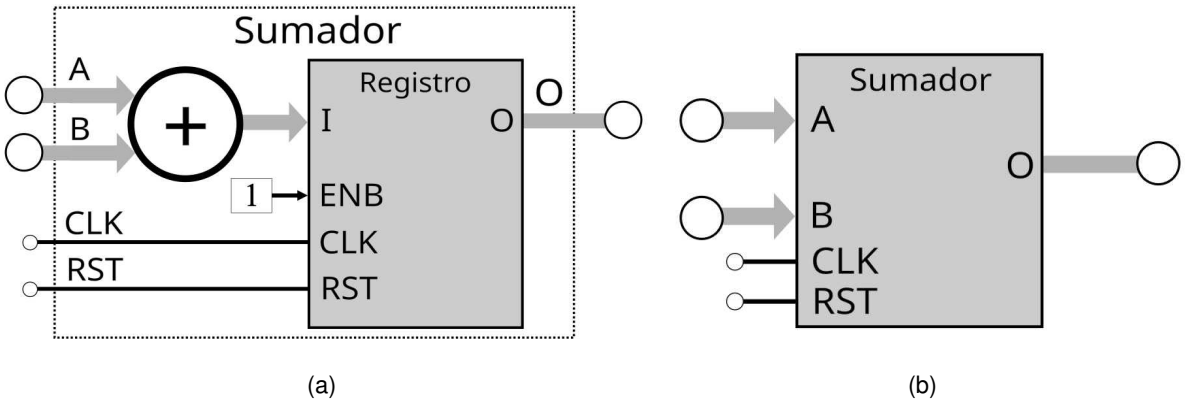


Figura 3.1: Sumador: (a) diagrama de hardware digital interno, y (b) representación como bloque secuencial.

El diseño anterior también se aplica para los bloques restadores y multiplicadores. Cabe aclarar que en el bloque restador, la operación de diferencia se ejecuta como: $A - B$.

Con la segmentación, el tiempo para obtener un resultado depende de la ejecución de varios ciclos de reloj. Estos mismos ciclos se pueden aprovechar para ejecutar otras tareas, mejorando así el procesamiento en paralelo. Además, en algunos sistemas digitales es posible implementar la técnica de la tubería (pipeline); sin embargo, debido a la dependencia de las entradas con las salidas de estos sistemas, esta técnica no puede ser aprovechada por estos diseños.

La figura 3.2 ilustra el diagrama de bloques que corresponde con la ecuación (2.6). En el esquema se aprecia la interconexión de bloques secuenciales como sumadores, restadores, multiplicadores y multiplicadores de una sola constante (SCM), junto con bloques combinacionales, indicados con la palabra “com”, para desarrollar cada una de las ecuaciones de las variables de estado x_{n+1}, y_{n+1} y z_{n+1}. La ruta crítica de este sistema, que involucra más bloques y, por lo tanto, más ciclos de reloj, sucede con la variable de estado z_{n+1}. Además, el diagrama incluye el bloque Integrador_FE, formado internamente por un SCM y un bloque sumador.

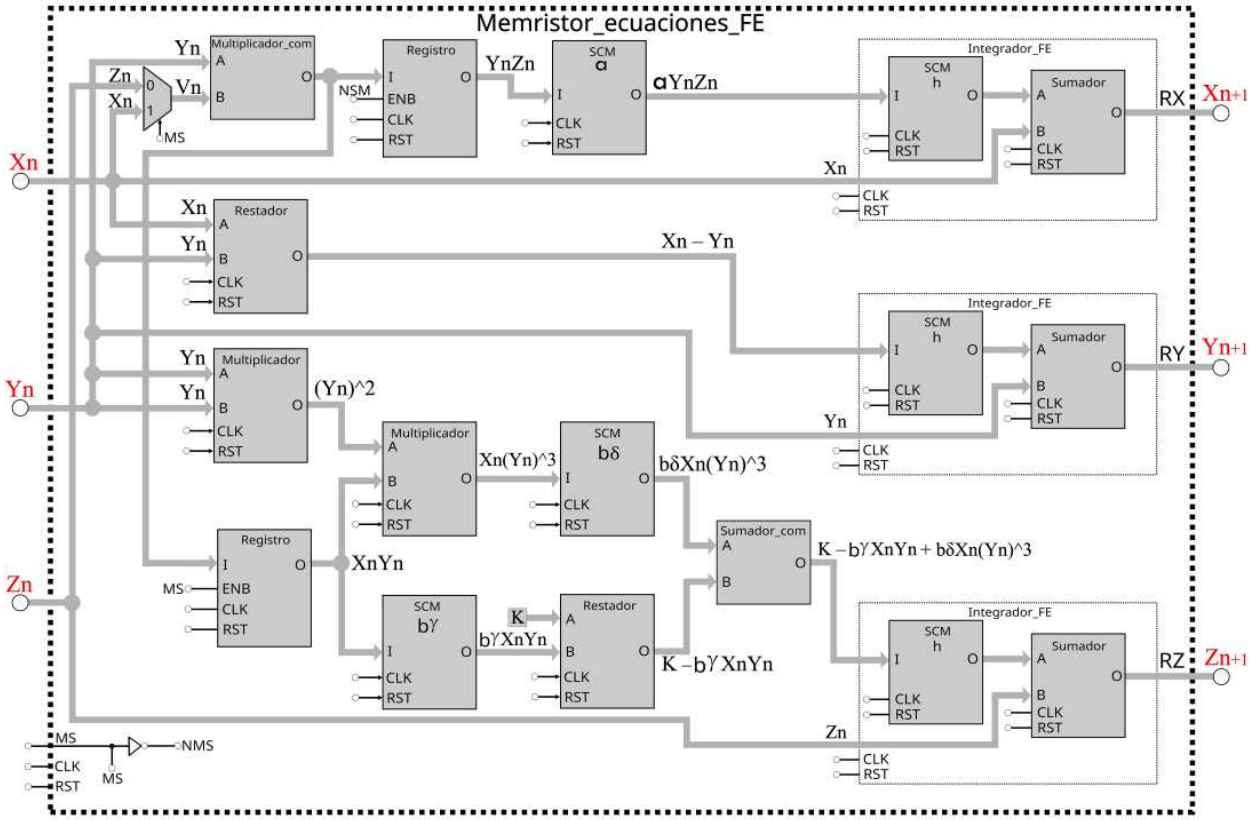


Figura 3.2: Diagrama de bloques de las ecuaciones del sistema 3D basado en un memristor [38].

Dependiendo del modelo de FPGA, los bloques de multiplicación entre variables se pueden

sintetizar utilizando recursos de hardware conocidos como multiplicadores embebidos, que operan con 9 o 18 bits. Sin embargo, cada modelo de FPGA cuenta con un limitado número de estos recursos, y en arquitecturas de 64 bits, estos se consumen rápidamente al emplear unos pocos bloques de multiplicación entre variables. Una manera de reducir el uso de los multiplicadores embebidos es reutilizar un solo bloque multiplicador para realizar varios productos entre variables. En el esquema de la figura 3.2 se observa la conexión de un multiplexor a la entrada B de un multiplicador combinacional, cuya salida O está conectada a dos registros PIPO. Esta conexión permite calcular los productos $x_n y_n$ y $y_n z_n$ reutilizado el mismo bloque de multiplicación en dos ciclos de reloj. Dado que la variable z_{n+1} representa la ruta crítica, en el primer ciclo de reloj se calcula el producto $x_n y_n$, y en el segundo, el producto $y_n z_n$. Para lograr lo anterior, el pin de entrada MS controla el selector del multiplexor y los habilitadores de los registros. Antes del primer ciclo, el pin debe tener un estado alto, y posteriormente debe colocarse en bajo. Mediante una compuerta lógica NOT, se garantiza que solo un registro a la vez almacene el resultado de una multiplicación.

Otra manera de ahorrar multiplicadores embebidos ocurre cuando en un sistema aparece la multiplicación entre una variable y una constante. Esta operación se puede realizar mediante bloques multiplicadores de una sola constante, como se muestra en [31], los cuales emplean recursos lógicos en su síntesis. El bloque SCM realiza la multiplicación, mediante operaciones de suma, resta y desplazamiento. Su funcionamiento se basa en la suma y resta de potencias de dos de la variable a multiplicar. Cada SCM se diseña específicamente para una aritmética de punto fijo particular, utilizando el sistema numérico de dígitos con signo (Signed-digit), el cual representa a los números binarios con los dígitos 1, -1 y 0. La figura 3.3 muestra la estructura interna de los cuatro SCM que se utilizan en este sistema.

El diagrama de la figura 3.2, que describe las ecuaciones discretas del sistema caótico 3D basado en un memristor, calcula los valores de la siguiente iteración $[n+1]$ para cada variable de estado a partir de los valores actuales n de las mismas, después de cinco ciclos de reloj. Sin embargo, este bloque por sí solo no es suficiente para generar las series temporales de forma indefinida. Para lograrlo, se requiere retroalimentar las salidas RX, RY y RZ con las entradas XN, YN y ZN, además de considerar que en la primera iteración las condiciones iniciales deben conectarse a las entradas del bloque. Debido a estos detalles, es necesario añadir otros recursos de hardware digital. La figura 3.4 muestra el diagrama del oscilador completo, en el cual se

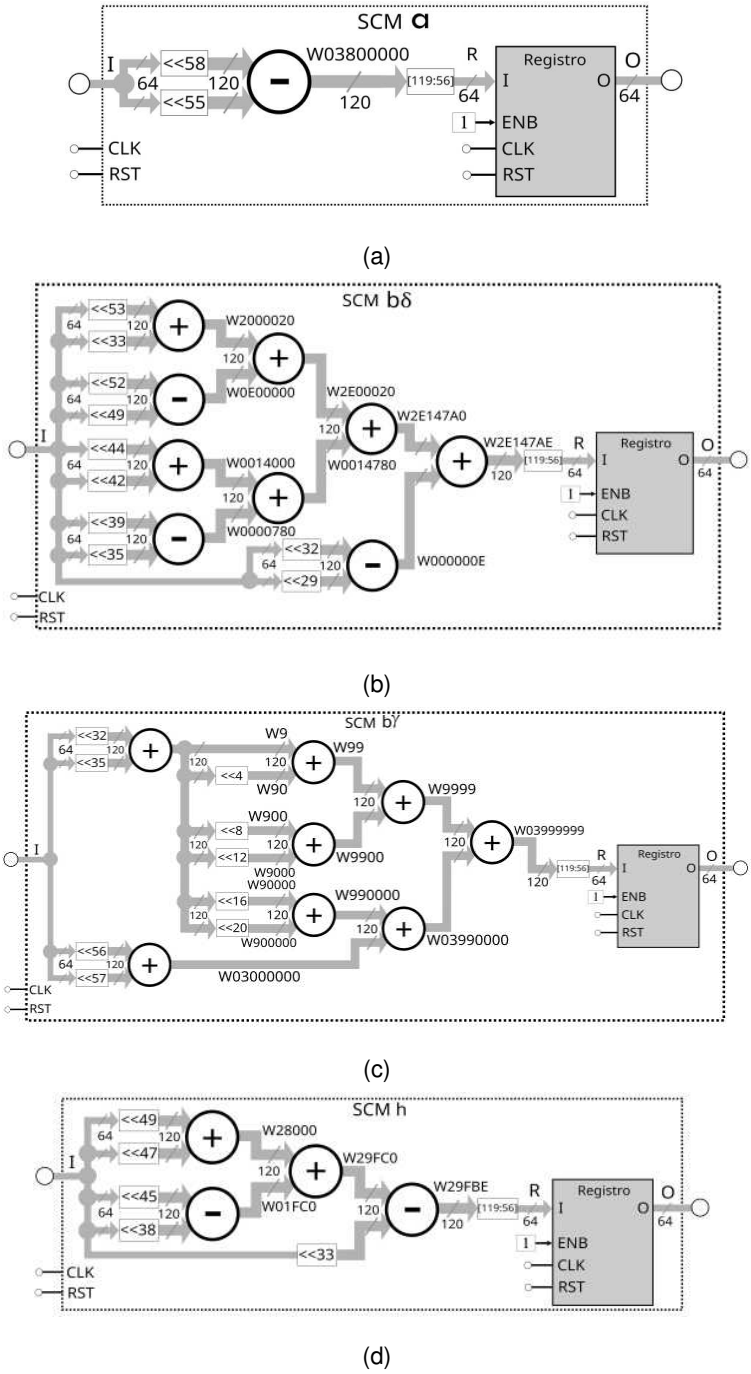


Figura 3.3: Diagramas de bloques de los cuatro SCM mostrados en la figura 3.2 diseñados con una aritmética de punto fijo de 64 bits $(1:7:56)$: a) a, b) $b\delta$, c) $b\gamma$ y d) h.

observa que el bloque Memristor_ecuaciones_FE tiene sus tres salidas conectadas a registros de entrada y salida paralela, los cuales se encargan de almacenar los resultados de cada itera-

ción y devolverlos a las entradas para el cálculo de la siguiente. Las salidas de estos registros PIPO están conectadas a multiplexores, que pueden proporcionar las condiciones iniciales para el sistema caótico o conectar las salidas con las entradas.

La arquitectura de la figura 3.4 requiere seis ciclos de reloj para actualizar cada variable de estado. Durante los primeros cinco ciclos, los registros PIPO mantienen el valor actual de cada iteración, y en el sexto ciclo se actualizan con nuevos valores. Esto es posible gracias a un contador cuya salida de control OPR, activa los habilitadores de los registros.

Al iniciar o reiniciar el sistema, un flip-flop tipo D comienza en un estado lógico bajo, para que las condiciones iniciales se apliquen a las entradas del bloque que desarrolla las ecuaciones. Después de los seis ciclos de reloj, este flip-flop cambia a un estado lógico alto, lo cual conecta las salidas de los registros PIPO con las entradas del bloque Memristor_ecuaciones_FE, a través de los multiplexores. Para iniciar una nueva secuencia con otras condiciones iniciales, es necesario utilizar la señal de reinicio.

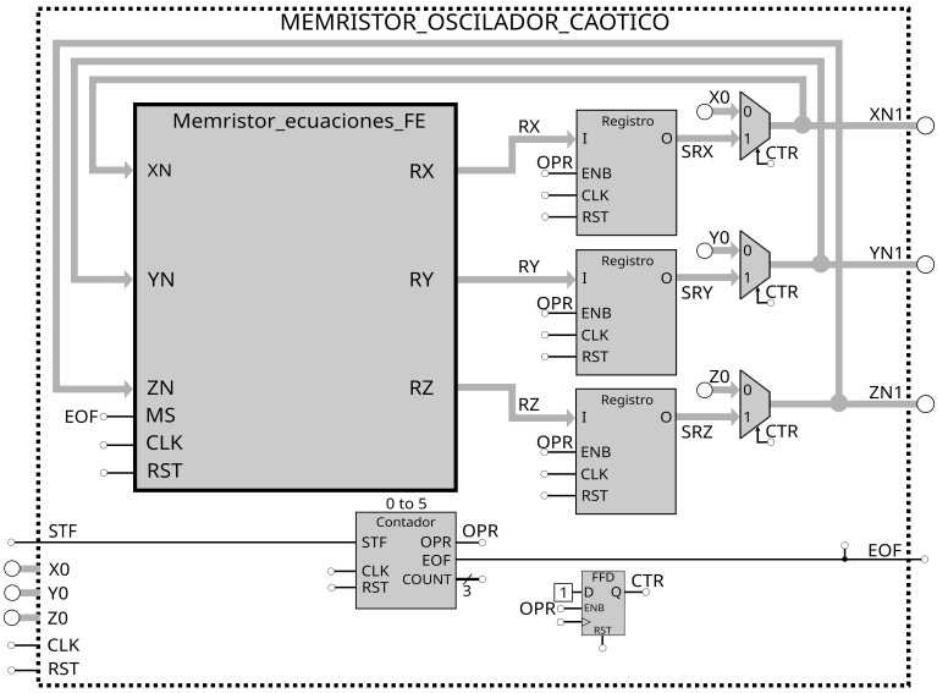


Figura 3.4: Diagrama para el oscilador 3D basado en un memristor.

3.2.1.3. Simulación del sistema en Verilog

El software de Active-HDL permite realizar simulaciones de los sistemas caóticos a través de bancos de pruebas, para verificar el funcionamiento del sistema antes de su síntesis en una FPGA, así como la visualización de las series temporales. Además, ofrece herramientas que permiten generar archivos de texto con los resultados de cada iteración. La figura 3.5 muestra las tres series temporales obtenidas mediante la herramienta de forma de onda de Active-HDL.

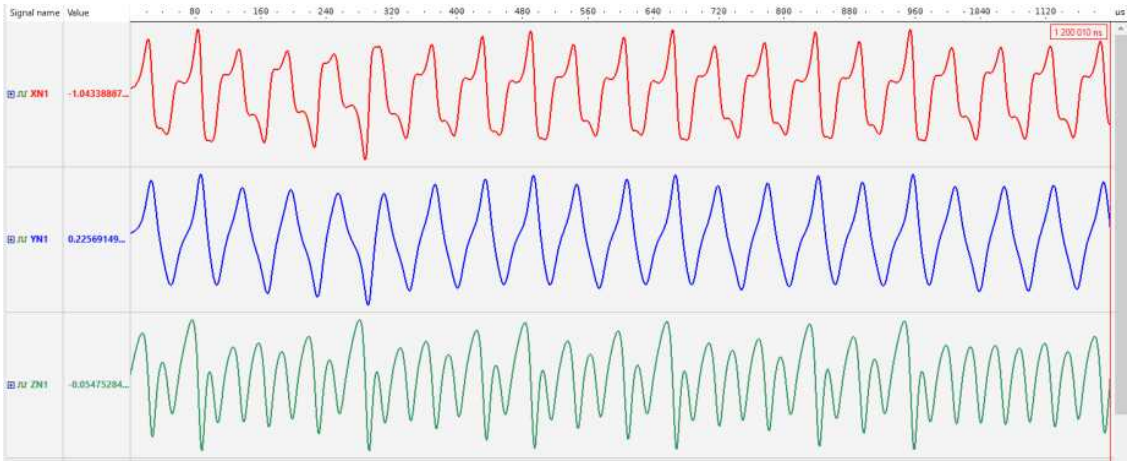


Figura 3.5: Series temporales del sistema 3D simuladas en Active-HDL.

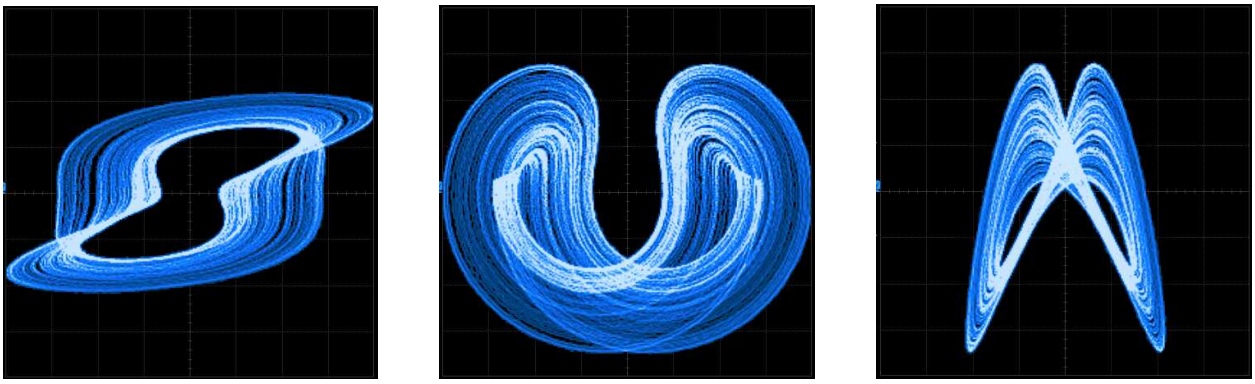
3.2.1.4. Resultados experimentales

La síntesis de este sistema caótico 3D basado en un memristor, en una FPGA de Altera/Intel, se llevó a cabo en el software de Quartus II. Para visualizar las series temporales y los atractores experimentales, se utilizó un convertidor digital a analógico (DAC) de 16 bits. Por ello, fue necesario añadir bloques adicionales para controlar el DAC por medio del protocolo de interfaz periférica serie (SPI) y realizar el procedimiento de truncamiento de 64 a 16 bits. La figura 3.6 muestra los atractores experimentales obtenidos.

3.2.2. Descripción hardware del mapa caótico 2D

3.2.2.1. Análisis de amplitudes para el formato de punto fijo

Al igual que con el sistema caótico 3D, previo a la descripción en Verilog de los bloques digitales, es necesario simular este mapa 2D en un software de alto nivel, para determinar el rango



(a) (b) (c)

Figura 3.6: Retratos de fase experimentales del sistema 3D: (a) $x - y$, (b) $x - z$, (c) $y - z$ [38].

de amplitudes de las dos variables de estado y los productos que aparecen en las ecuaciones. Después de realizar simulaciones con 1000000 de iteraciones y diferentes condiciones iniciales, los rangos máximos y mínimos, considerando los cuatro conjuntos de parámetros a_1 del mapa, estaban dentro del rango aproximado ± 16. Por lo cual, para este mapa el formato de punto fijo solo requiere 4 bits de parte entera. La tabla 3.2 muestra la distribución del formato de punto fijo en 64 bits para este sistema de dos dimensiones.

Tabla 3.2: Distribución de la aritmética de punto fijo con 64 bits para el mapa caótico en 2D.

Signo	Parte entera	Parte fraccionaria
1 bit	4 bits	59 bits

3.2.2.2. Descripción del sistema mediante bloques digitales

Al analizar la ecuación (2.2) del mapa caótico en 2D, se observa que las expresiones que describen a las variables de estado x_{n+1} y y_{n+1} presentan una estructura con un orden de operaciones similar, pero con diferentes coeficientes a_i. La descripción con bloques digitales para FPGA podría implementarse mediante la conexión en serie de sumadores, multiplicadores entre variables y multiplicadores de una sola constante, como se efectuó para el sistema anterior. Sin embargo, para este mapa 2D, una descripción directa de las ecuaciones requeriría del diseño de alrededor de ocho bloques SCM a 64 bits para un único conjunto de coeficientes a_i o utilizar

un alto número de multiplicadores embebidos. Debido a esto, el diseño de este sistema por este método podría ser laborioso. Por lo tanto, un enfoque más práctico para desarrollarlo consiste en emplear bloques multiplicador acumulador (MAC), los cuales permiten reducir el consumo de elementos lógicos y facilitan la reconfiguración de los coeficientes a_i, al almacenar los distintos conjuntos en una memoria de tipo ROM.

La figura 3.7 muestra la estructura interna de un bloque multiplicador acumulador, formada a partir de un bloque multiplicador con habilitador (sintetizado con multiplicadores embebidos), un sumador combinacional y dos registros de entrada y salida paralela [41]. El bloque cuenta con dos entradas de datos A y B, y una salida SUM, además de cinco entradas de control, incluyendo las señales de reloj y de reinicio. Su funcionamiento general es el siguiente: primero, se activa el pin de control LAC para que el multiplicador y el registro acumulador ejecuten las sumas de productos. En cada ciclo de reloj, las entradas del bloque MAC deben actualizarse. Una vez realizadas todas las sumas de productos, se activa el pin de control LFR para que el segundo registro almacene el valor final. Por último, el pin de control CAC elimina el valor en el registro acumulador, para reiniciar el proceso. Los bloques MAC suelen estar acompañados de máquinas de estados finitos, contadores o una combinación de ambos, para establecer el número de productos mediante las entradas de control.

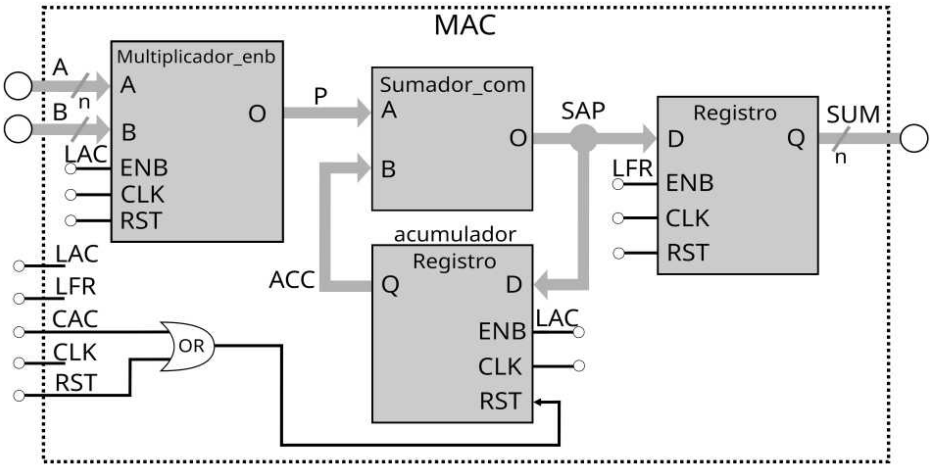


Figura 3.7: Diagrama para el multiplicador acumulador, basado en [41].

La figura 3.8 ilustra la implementación del mapa caótico de Sprott utilizando dos bloques MAC para calcular las variables de estado x_{n+1} y y_{n+1}. En la arquitectura digital se observa el bloque

de control para los MAC, denominado Control_MAC, compuesto por un contador binario y una máquina de estados finitos tipo Moore. También se incluyen dos memorias ROM, un multiplicador multiplexado que calcula los productos x_n^2, $x_n y_n$ y y_n^2, y un multiplexor de ocho entradas y una salida, encargado de seleccionar las variables que se multiplican con los coeficientes a_i dentro del MAC.

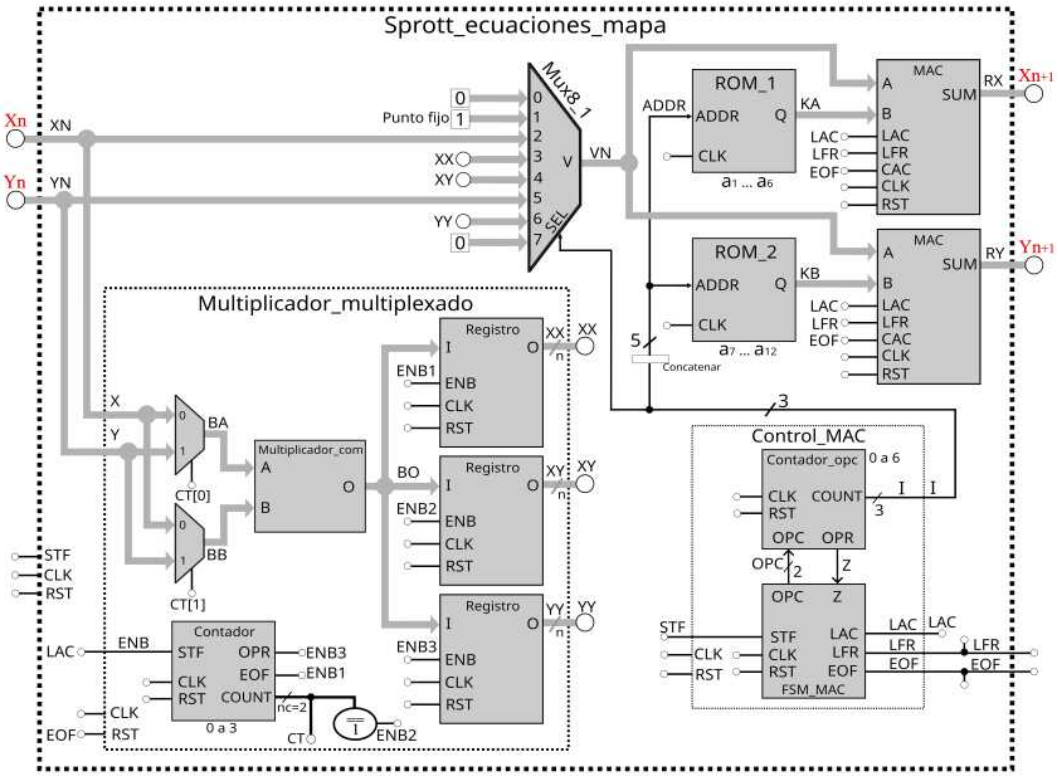


Figura 3.8: Diagrama para el mapa caótico en 2D.

Como primer punto, la máquina de estados dentro del bloque Control_MAC cuenta con cuatro entradas y cuatro salidas, las cuales se describen brevemente a continuación. La entrada STF inicia el procedimiento para calcular una nueva iteración, mientras que la entrada Z recibe un estado lógico alto, proveniente del contador binario, cuando se han efectuado todas las sumas de productos. Las dos entradas restantes corresponden a la señal de reloj y de reinicio. Por otro lado, la salida OPC, de dos bits, se encarga de establecer el contador a cero (OPC=00) o de permitir el conteo ascendente (OPC=10). La salida LAC habilita la suma de productos, y LFR habilita el registro que almacena el valor final. Finalmente, la salida EOF tiene como función reiniciar el registro acumulador del MAC. La transición de estados de la FSM se detalla en la tabla 3.3.

Tabla 3.3: Tabla de verdad de la máquina de estados.

Estado	**Entradas**		Estado	**Salidas**			
actual	STF	Z	siguiente	OPC	LAC	LFR	EOF
0	0	x	0	00	0	0	1
0	1	x	1				
1	x	0	1	10	1	0	0
1	x	1	2				
2	x	x	0	00	1	1	0
3	x	x	0	00	0	0	1

En el bloque Control_MAC el bus de salida I, de 3 bits, funciona como un apuntador de direcciones para las dos memorias ROM y como un selector para el multiplexor. Este bus incrementa su valor de 0 a 7, de tal forma que en cada ciclo de reloj las ROM suministran un coeficiente en la entrada B de cada MAC. La ROM_1 almacena los coeficientes a_1 a a_6, mientras que la ROM_2 almacena de a_7 a a_{12}. Ambas memorias deben almacenar los valores de los coeficientes de acuerdo con el formato de punto fijo de la tabla 3.2.

El funcionamiento de esta arquitectura requiere de nueve ciclos de reloj para proporcionar nuevos resultados. De estos, siete ciclos ocurren durante el estado 1 (hasta que Z toma el valor 1), permitiendo que se realicen las sumas de productos a medida que el contador incrementa. Dado que las dos primeras operaciones en los MAC no requieren los valores x_n^2 y y_n^2 ni el producto $x_n y_n$, es posible utilizar un multiplicador multiplexado para obtener estos valores. Este diseño está compuesto por un multiplicador combinacional, tres registros PIPO, dos multiplexores y un contador ascendente, el cual permite obtener los productos antes de necesitarlos en el MAC, usando un solo multiplicador y, por tanto, ahorrando recursos de hardware. Cada registro del multiplicador multiplexado almacena un producto en un ciclo de reloj gracias a los habilitadores controlados por el contador. El bloque comienza a realizar los productos cuando la señal LAC se activa y los registros se reinician a cero cuando EOF se activa.

Para generar las series temporales utilizando el sistema representado en la figura 3.8, es necesario proporcionar las condiciones iniciales y retroalimentar las salidas con las entradas. Por ello, en la figura 3.9 se presenta el diagrama completo del sistema, denominado “SPROTT_MAPA_CAOTICO”, el cual está compuesto por el bloque de ecuaciones, dos multiplexores y un flip-flop tipo D. A di-

ferencia del sistema caótico en 3D, descrito en la sección anterior, este bloque no requiere de un contador ni de registros PIPO de salida, ya que internamente el bloque tiene una máquina de estados que controla y sincroniza todas las operaciones.

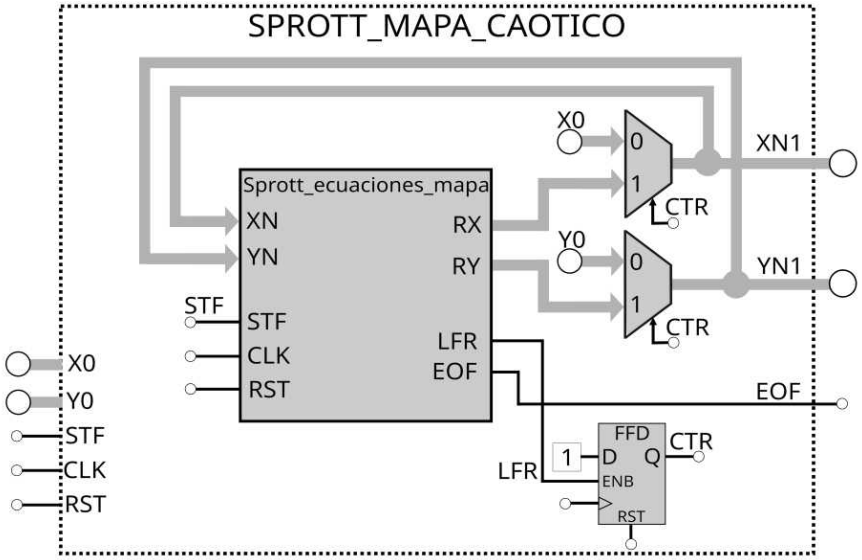


Figura 3.9: Diagrama para el mapa caótico en 2D.

3.2.2.3. Simulación del sistema en Verilog

Las simulaciones de este mapa 2D también se llevaron a cabo en la herramienta de forma de onda de Active-HDL. La figura 3.10 muestra las dos series temporales obtenidas para los coeficientes a_i del mapa 4 de la tabla 2.1, con una simulación de 600 iteraciones.

3.2.2.4. Resultados experimentales

Este mapa caótico también se sintetizó en la FPGA por medio del software de Quartus II. Para poder observar los resultados experimentales, se utilizó el mismo diseño para controlar el convertidor digital a analógico de 16 bits. La figura 3.11 muestra los cuatro atractores experimentales obtenidos para los diferentes conjuntos de coeficientes a_i.

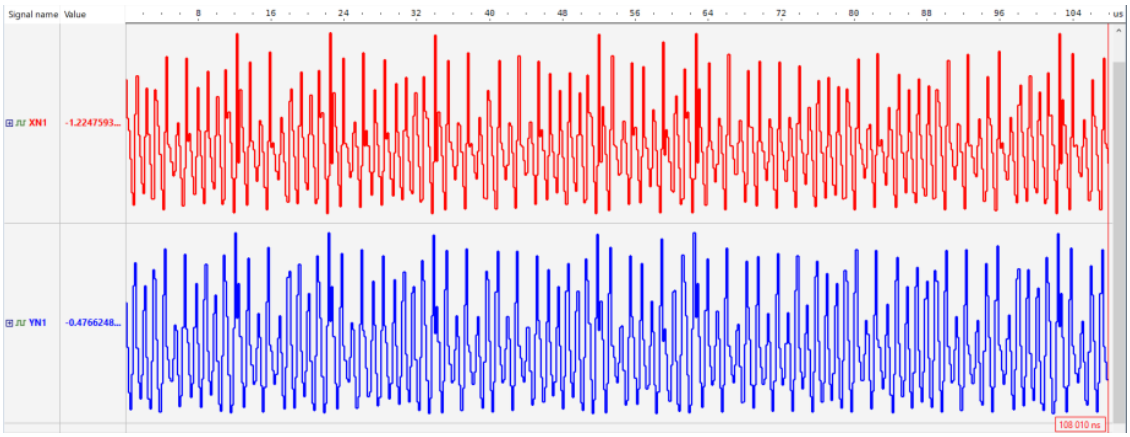


Figura 3.10: Series temporales del mapa 2D para las variables de estado a) x_n, b) y_n, con los coeficientes a_i del mapa 4 en Active-HDL.

3.2.3. Descripción del sistema caótico en 4D

3.2.3.1. Análisis de amplitudes para el formato de punto fijo

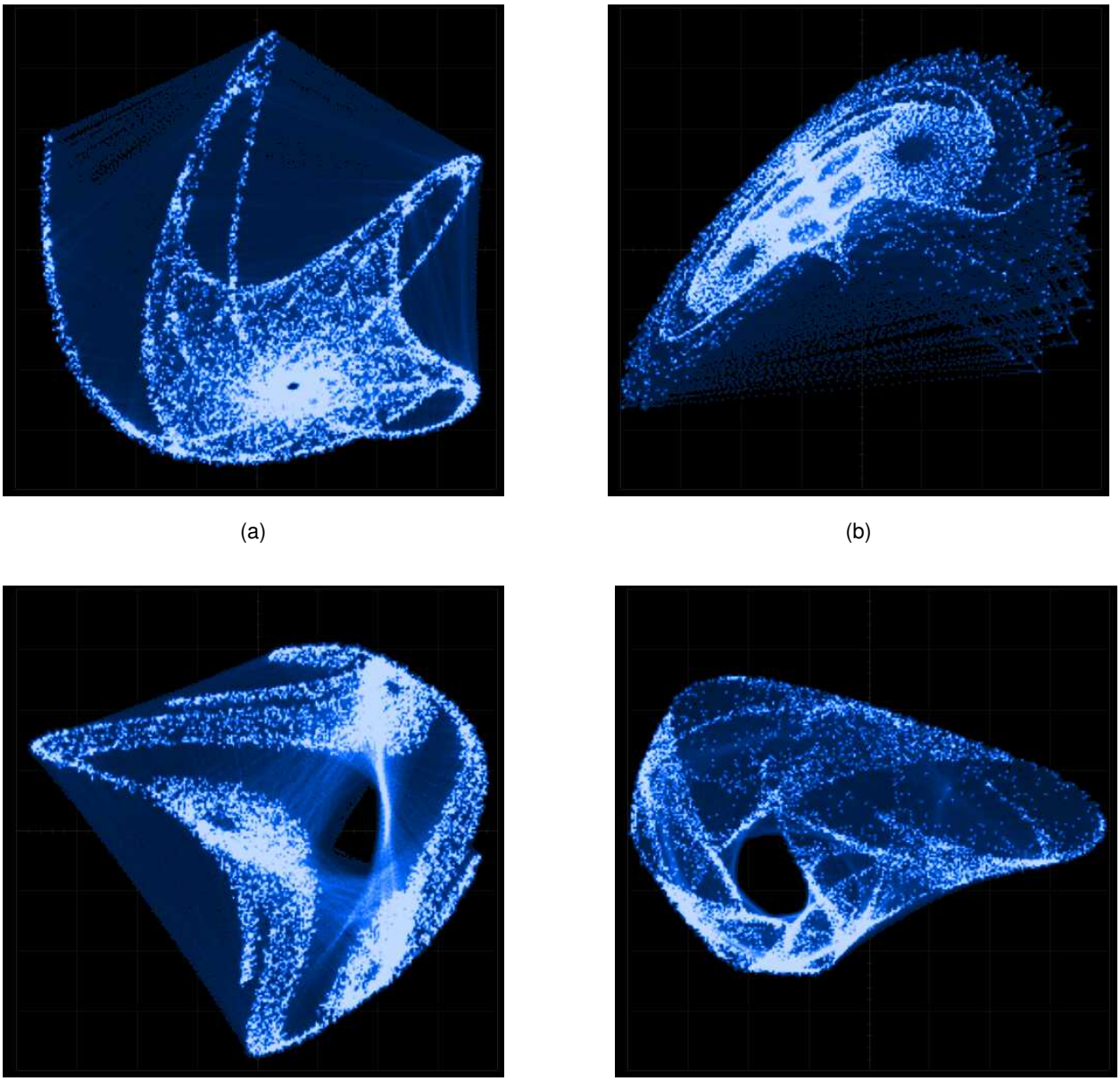
El análisis de amplitudes máximas y mínimas del sistema de 4 variables de estado también se llevó a cabo en un software de alto nivel. En cada simulación se utilizaron los parámetros propuestos en el capítulo 2, diferentes condiciones iniciales y 1000000 de iteraciones. Los resultados en cada prueba mostraban amplitudes dentro del rango ± 256, por ello, 8 bits de parte entera en este sistema permiten generar y mantener las series caóticas. La tabla 3.4 muestra una adecuada distribución del formato de punto fijo para este sistema.

Tabla 3.4: Distribución de la aritmética de punto fijo con 64 bits para el sistema caótico en 4D.

Signo	Parte entera	Parte fraccionaria
1 bit	8 bits	55 bits

3.2.3.2. Descripción del sistema mediante bloques digitales

La implementación del sistema caótico en 4D en una FPGA, utilizando el método de Euler hacia adelante, puede desarrollarse de manera similar al sistema 3D mediante una descripción directa de sus expresiones discretas mostradas en la ecuación (2.8). Esto es posible debido a que el sistema de ecuaciones solo presenta múltiples operaciones aritméticas en la variable de estado



(a) (b)

(c) (d)

Figura 3.11: Retratos de fase experimentales $x_n - y_n$ con los cuatro conjuntos de coeficientes de la tabla 2.1: (a) mapa 1, (b) mapa 2, (c) mapa 3, (d) mapa 4.

ω_{n+1}. El diagrama de bloques para implementar este sistema se ilustra en la figura 3.12, donde se introducen nuevos bloques, como el Restador_2N, que realiza la operación de resta entre sus puertos de la forma $O = -A - B$, y el bloque X_ABS, que calcula la diferencia $-x_n - |x_n|$. En este bloque, cuando x_n tiene un valor positivo, el resultado es $-2x_n$, mientras que si x_n es negativo,

devuelve el valor 0. Los SCM que forman parte de este sistema se desarrollaron de forma similar al sistema 3D, con las constantes $a = 4$, $b = 2$ y $h = 0.02$ y considerando el formato en punto fijo de la tabla 3.4.

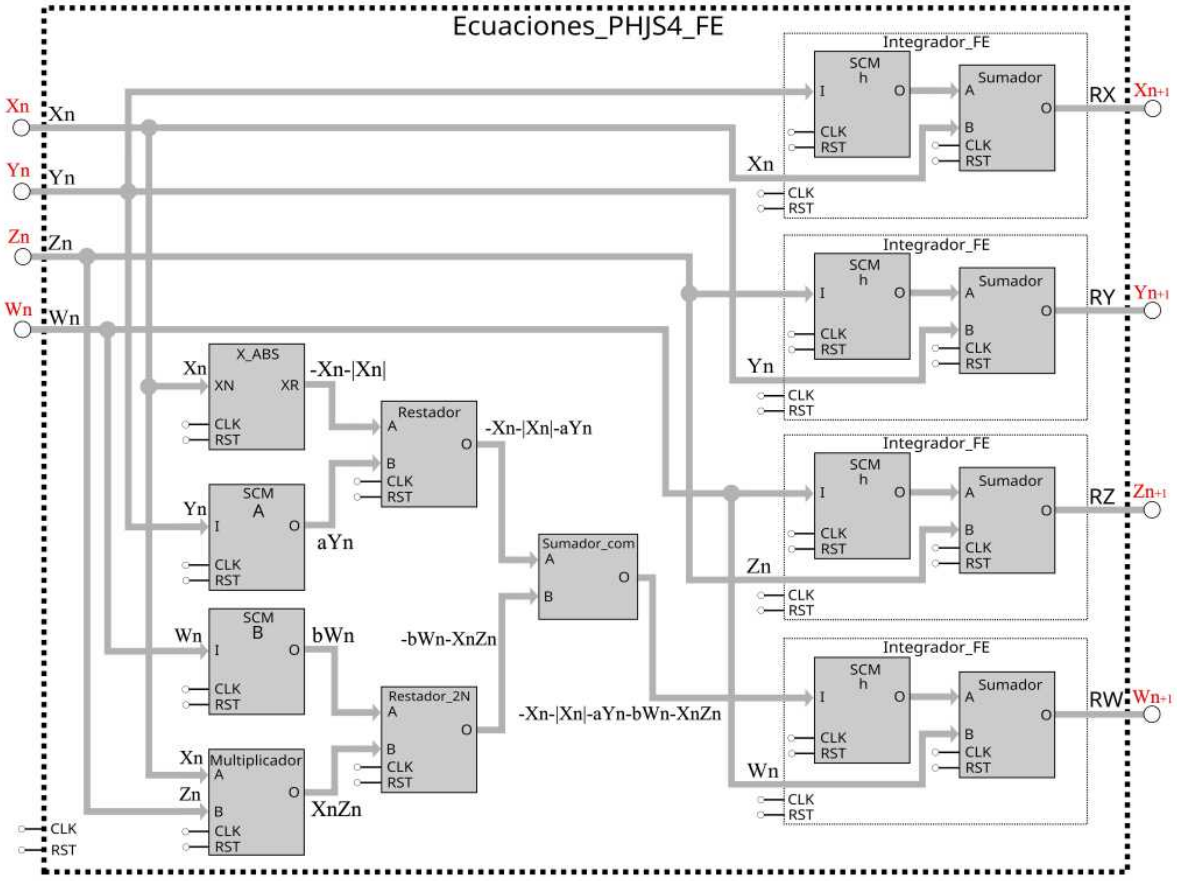


Figura 3.12: Diagrama de bloques de las ecuaciones del sistema 4D, basado en [40].

La arquitectura digital para el oscilador completo 4D es similar al de la figura 3.4. En este sistema, se requieren cinco ciclos de reloj para generar nuevos valores de una iteración. A diferencia del oscilador 3D, en el 4D se agrega un nuevo registro PIPO adicional para almacenar los resultados de la variable ω_n y se incorpora otro puerto de entrada para la condición inicial ω_0. Además, dado que el bloque de ecuaciones solo utiliza un multiplicador entre variables, no es necesario reutilizar el multiplicador.

3.2.3.3. Simulación del sistema en Verilog

De manera similar a los dos casos de estudio anteriores, para este sistema se llevaron a cabo simulaciones en Active-HDL. La figura 3.13 muestra las cuatro series temporales obtenidas con una simulación de 500 iteraciones.

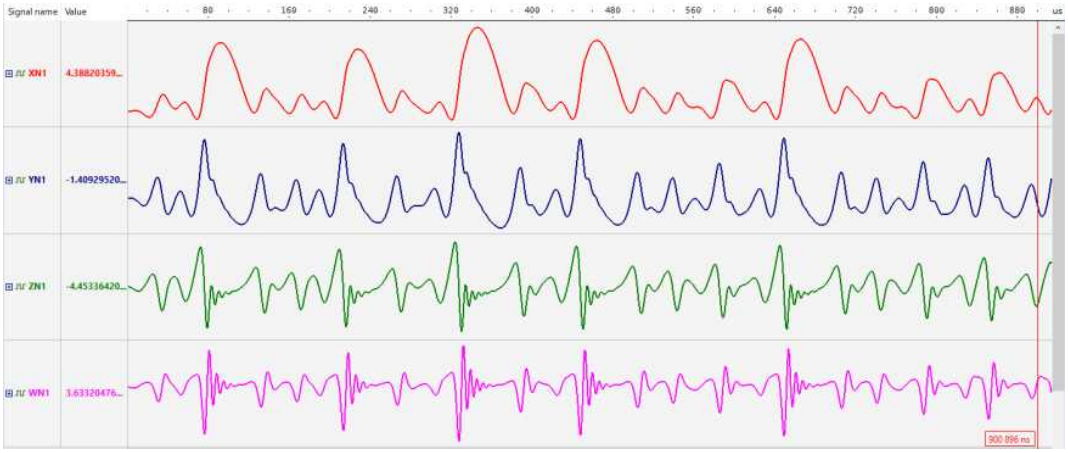


Figura 3.13: Series temporales del sistema 4D simuladas en Active-HDL.

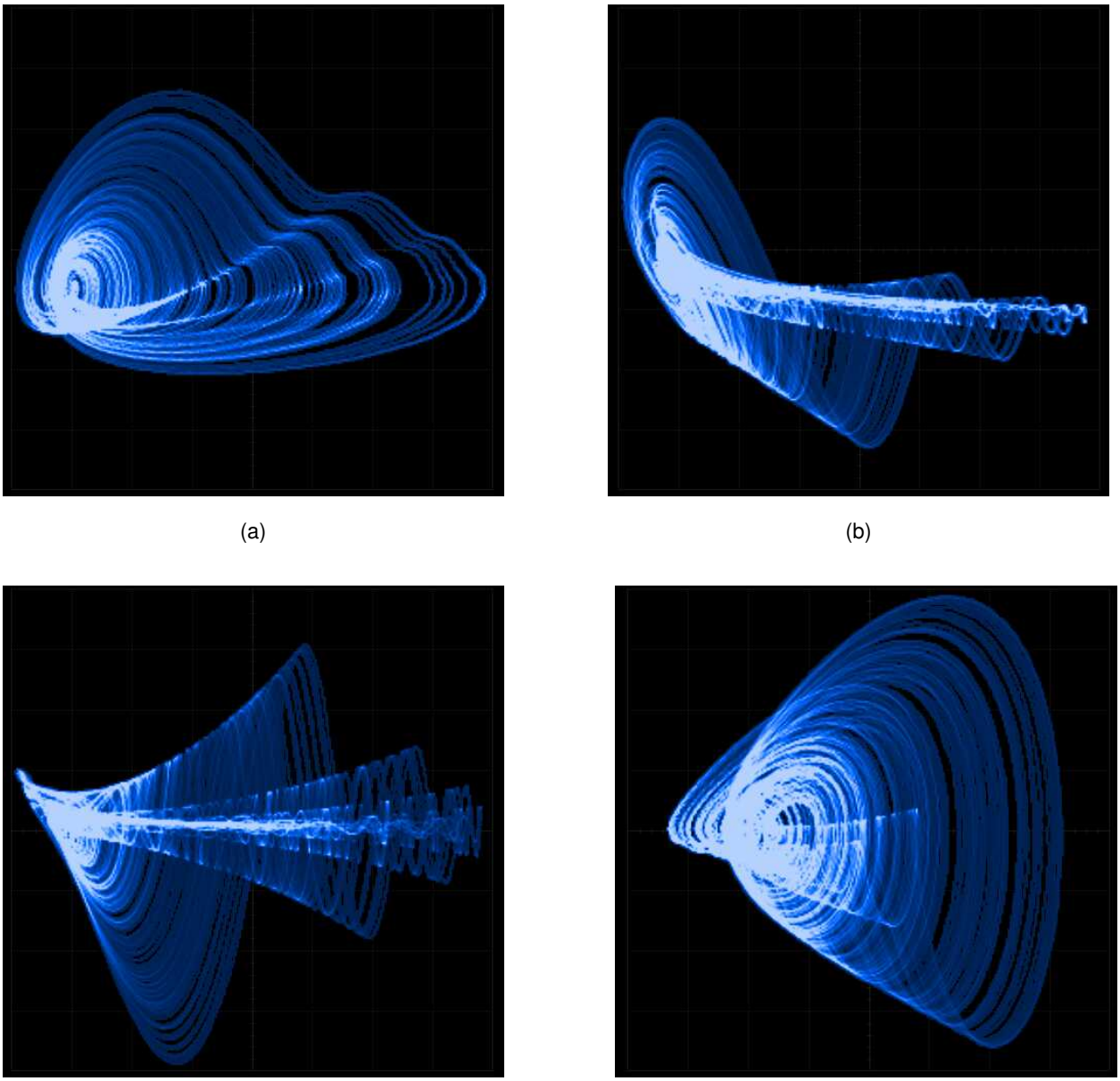
3.2.3.4. Resultados experimentales

Para visualizar los atractores y las series temporales, se emplearon los bloques de control para el convertidor digital a analógico de 16 bits. Además, se añadieron bloques para mostrar las combinaciones de las variables en los espacios de fases. La figura 3.14 muestra cuatro atractores experimentales.

3.3. Síntesis en FPGA de los sistemas caóticos

La tabla 3.5 muestra el consumo de recursos digitales resultante de la síntesis de los sistemas completos 2D, 3D y 4D en una FPGA Cyclone III EP3C16F484C6. En el cálculo del throughput solo se consideran los ocho bits menos significativos de cada variable de estado, ya que son los bits empleados por estos diseños para generar los números aleatorios.

A partir de la tabla 3.5 se observa que el mapa 2D tiene el menor consumo de elementos lógicos y registros, pero presenta la mayor latencia, la frecuencia máxima de operación más baja y el menor throughput. Por otro lado, el sistema 4D destaca por ser el más eficiente, ya que



(a) (b)

(c) (d)

Figura 3.14: Retratos de fase experimentales del sistema 4D: (a)$x - y$, (b)$x - z$, (c)$x - \omega$, (d)$y - z$.

ofrece el mayor throughput y el menor consumo de multiplicadores embebidos. Sin embargo, estos resultados solo muestran el desempeño en cuanto a velocidad y uso de recursos en la FPGA, sin garantizar la complejidad del sistema ni la aleatoriedad de las secuencias generadas.

Tabla 3.5: Consumo de recursos en una FPGA Cyclone III EP3C16F484C6 para los sistemas 2D, 3D y 4D en arquitectura de 64 bits.

Recursos	Mapa 2D	Sistema 3D	Sistema 4D	Disponibles
Elementos lógicos (LE)	$2,095$	$4,034$	$2,149$	$15,408$
Funciones combinacionales	$2,012$	$4,008$	$2,086$	$15,408$
Registros lógicos	585	$1,080$	$1,138$	$15,408$
Bits de memoria	1024	0	0	$516,096$
Multiplicadores embebidos de 9-bit	96	96	32	112
Frecuencia máxima (MHz)	55.94	57.85	75.3	–
Ciclos de reloj	9	6	5	–
Latencia a 50 MHz (ns)	180	120	100	–
Throughput a 50 MHz (Mbps)	88.8	200	320	–
Eficiencia (Mbps/LE)	0.04	0.05	0.15	–

3.4. Diseño de PRNGs a partir de sistemas caóticos

Como se mencionó en el capítulo 1, es posible generar cadenas binarias pseudoaleatorias a partir de las series temporales de un sistema caótico utilizando los métodos como el umbral, la operación módulo, el formato de punto fijo [44], entre otros. En el primer método, las series binarias se obtienen comparando el valor de la serie temporal con una cantidad fija o umbral, como en [45]. Si el valor de la serie es mayor que el umbral, se asigna un 1 en la serie binaria; de lo contrario, se añade un 0. El segundo método emplea la operación módulo entre el valor de la serie temporal y algún número fijo, como 255, para producir el conjunto de bits de la serie. Por último, el tercer método toma los bits con mayor cambio en un formato de punto fijo, por ejemplo, los menos significativos de la parte fraccionaria. Incluso, este método se puede aplicar a sistemas implementados con aritmética de punto flotante, simplemente convirtiendo el número flotante a un entero sin signo al multiplicar el número flotante por una gran potencia de 2.

Hoy en día, en la literatura existe una extensa investigación acerca de PRNGs basados en sistemas caóticos. Por ejemplo, en el trabajo de [46] se muestra el desarrollo e implementación en FPGA de un PRNG basado en un modelo de memristor con comportamiento caótico, que utiliza operaciones aritméticas simples en su modelado. El sistema fue resuelto aplicando los métodos numéricos de Euler hacia adelante y Runge Kutta de orden 4, empleando aritmética

tanto de punto fijo como de punto flotante. En este caso, las series binarias pseudoaleatorias se generaron mediante una operación de postprocesamiento, que incluye varias operaciones módulo 2 para producir 8 bits de cada variable de estado. Para verificar la aleatoriedad de las series, se aplicaron pruebas estadísticas NIST. De manera similar, en el artículo de [47] se propone un PRNG basado en un mapa caótico en 1D, llamado mapa Siponi, el cual es una modificación del mapa logístico. En dicho trabajo, las series de bits pseudoaleatorias se obtuvieron mediante el método del umbral y, para verificar su aleatoriedad, se aplicaron pruebas estadísticas DieHard y NIST. Además, el mapa se implementó en una FPGA en aritmética de punto flotante a 64 bits, donde el sistema digital utilizó un multiplicador multiplexado para reducir el uso de multiplicadores embebidos. Existen también otros trabajos que exploran PRNGs basados en sistemas caóticos con un mayor número de variables de estado. Por ejemplo, [48] presenta un sistema caótico en 5D, basado en otro modelo de memristor, controlado por una no linealidad cuadrática. Este sistema presenta atractores coexistentes y un comportamiento hipercaótico. Al ser un sistema de tiempo continuo, su solución se realizó utilizando el método de Runge Kutta 4, en un software de alto nivel. Las series binarias pseudoaleatorias de 21 bits se generaron aplicando operaciones XOR entre cuatro de las cinco variables de estado. Cabe destacar que, aunque los tres sistemas anteriores están compuestos únicamente por operaciones aritméticas simples, los sistemas caóticos no se limitan a este tipo de funciones. Existen PRNGs diseñados a partir de sistemas caóticos que incluyen ecuaciones con funciones trigonométricas. Por ejemplo, en el trabajo de [49] se propone un modelo caótico basado en una función coseno no lineal y una función valor absoluto, que puede aplicarse a sistemas caóticos ya existentes para generar comportamientos caóticos más complejos. Para generar las secuencias binarias pseudoaleatorias, el modelo emplea la operación módulo y, en su implementación en FPGA se utilizan bloques más avanzados para realizar las funciones como lo es el método CORDIC. Otro ejemplo es el trabajo de [50] que aborda un PRNG desarrollado partir de un mapa hipercaótico 2D, donde la variable x_{n+1} está basada en el mapa 1D de Henon y la variable y_{n+1} en el mapa 1D de seno. Además, el mapa incluye una función de división. Su implementación se llevó a cabo en un Arduino en aritmética de punto flotante de 32 bits. Las variables de estado se escalaron de 0 a 1, y para obtener 8 bits de series aleatorias, se empleó un producto con 255 y una operación de redondeo. Por último, el trabajo de [51], propone un PRNG basado en una red neuronal de Hopfield. Este sistema de ecuaciones presenta funciones hiperbólicas, y su implementación se llevó a cabo en un FPGA Xilinx utilizando

el método numérico de Runge Kutta 4 en aritmética de punto flotante de 32 bits. El sistema genera 64 bits aleatorios en cada iteración, y las series binarias se obtienen mediante una operación de postprocesamiento basada en operaciones XOR. En todos los sistemas anteriores se ejecutó alguna prueba estadística para verificar la aleatoriedad de los bits.

Como caso particular, en esta tesis se utilizó el método del formato de punto fijo, ya que los sistemas fueron implementados en FPGA aplicando esta aritmética. Se seleccionaron los ocho bits menos significativos de cada variable de estado para generar las secuencias binarias pseudoaleatorias. La figura 3.15 ilustra el PRNG basado en el sistema caótico 3D, el cual produce 24 bits pseudoaleatorios en cada iteración. Este mismo esquema se aplica para los sistemas 2D y 4D, los cuales producen 16 y 32 bits pseudoaleatorios, respectivamente. Sin embargo, es necesario verificar la aleatoriedad de estas secuencias mediante pruebas estadísticas antes de utilizarlas en esquemas de comunicaciones seguras.

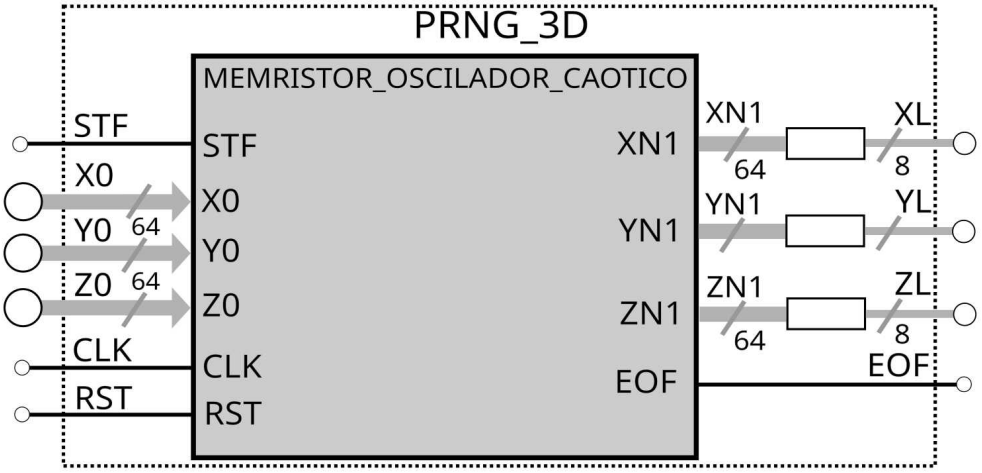


Figura 3.15: PRGN basado en el sistema caótico 3D.

3.5. Pruebas estadísticas NIST de aleatoriedad

Las pruebas estadísticas NIST permiten medir y validar la propiedad de aleatoriedad de una serie binaria generada por un RNG, ya sea aleatoria o pseudoaleatoria, con el objetivo de garantizar que la secuencia de ceros y unos sea candidata para aplicaciones de criptografía [52]. El conjunto de 15 pruebas NIST se pueden realizar sobre un grupo de 1000 series, donde cada una

está compuesta por un millón de bits. En el caso de los PRNGs, el archivo que contiene las series binarias pseudoaleatorias se forma utilizando 1000 semillas diferentes. Estas pruebas verifican que los ceros y unos en la serie binaria tengan la misma probabilidad de aparecer y que no exista correlación entre las series.

Para determinar si las series binarias pseudoaleatorias superan las pruebas estadísticas, se utiliza el valor p de la significancia estadística. En una prueba con 1000 series, este valor de p debe ser mayor que $\alpha = 0.001$, lo cual indica que las series se consideran aleatorias con una confianza de 99,9 %. Un valor p igual a cero indica que la serie no es aleatoria. Cada prueba evalúa una propiedad específica de la serie. Por ejemplo, la primera es la prueba de frecuencia, que determina si la proporción de ceros y unos en la secuencia es aproximadamente igual.

Las tablas 3.6, 3.7 y 3.8 muestran los resultados al aplicar las 15 pruebas estadísticas NIST a las series binarias pseudoaleatorias generadas por los sistemas 2D, 3D y 4D, respectivamente. Se puede observar en las tablas que todas las series binarias de cada variable de estado pasaron las pruebas estadísticas, por lo cual, son secuencias pseudoaleatorias que podrían utilizarse en aplicaciones criptográficas.

Tabla 3.6: Resultados de las pruebas NIST utilizando el método de punto fijo para las variables de estado x y y del mapa caótico 2D (mapa 4), con condiciones iniciales que variaron en el rango $[-0.4, 0.4]$.

Prueba estadística	X		Y	
	P-Value	Proportion	P-Value	Proportion
Frequency	0.4904	987/1000	0.5002	991/1000
Block Frequency	0.1795	992/1000	0.0292	991/1000
Cumulative Sums	0.1855	992/1000	0.9047	989/1000
Runs	0.7538	988/1000	0.5790	989/1000
Longest Run	0.2743	991/1000	0.1815	987/1000
Rank	0.2531	986/1000	0.1563	990/1000
FFT	0.5002	986/1000	0.3553	990/1000
Non Overlapping Template	0.2505	989/1000	0.0227	991/1000
Overlapping Template	0.3267	990/1000	0.7830	990/1000
Universal	0.5996	987/1000	0.8074	993/1000
Approximate Entropy	0.3838	992/1000	0.8513	989/1000
Random Excursions	0.5421	609/618	0.6440	603/608
Random Excursions Variant	0.6217	611/618	0.9069	598/608
Serial	0.7714	991/1000	0.5914	989/1000
Linear Complexity	0.3115	990/1000	0.1436	991/1000

Tabla 3.7: Resultados de las pruebas NIST utilizando el método de punto fijo para las variables de estado x, y y z del sistema caótico 3D, con condiciones iniciales que variaron en el rango $[-0.5, 0.5]$.

Prueba estadística	X		Y		Z	
	P-Value	Proportion	P-Value	Proportion	P-Value	Proportion
Frequency	0.5101	992/1000	0.4446	992/1000	0.5341	995/1000
Block Frequency	0.6972	988/1000	0.4983	992/1000	0.4827	987/1000
Cumulative Sums	0.2442	994/1000	0.8110	992/1000	0.9127	994/1000
Runs	0.8165	989/1000	0.8628	991/1000	0.8343	995/1000
Longest Run	0.5141	992/1000	0.7906	989/1000	0.9857	985/1000
Rank	0.4904	984/1000	0.1075	991/1000	0.4299	992/1000
FFT	0.5955	990/1000	0.1461	987/1000	0.3488	980/1000
Non Overlapping Template	0.9252	991/1000	0.8201	996/1000	0.8463	983/1000
Overlapping Template	0.2968	990/1000	0.2925	988/1000	0.6600	985/1000
Universal	0.2467	987/1000	0.3026	988/1000	0.3586	994/1000
Approximate Entropy	0.7734	989/1000	0.9703	992/1000	0.6433	990/1000
Random Excursions	0.5421	633/639	0.9758	580/585	0.5591	607/613
Random Excursions Variant	0.7909	631/639	0.9562	575/585	0.1671	609/613
Serial	0.7887	992/1000	0.9473	992/1000	0.0284	986/1000
Linear Complexity	0.9743	993/1000	0.6059	988/1000	0.5769	995/1000

Tabla 3.8: Resultados de las pruebas NIST utilizando el método de punto fijo para las variables de estado x, y, z y w del sistema caótico 4D, con condiciones iniciales que variaron en el rango $[-0.4, 0.4]$.

Prueba estadística	X		Y		Z		W	
	P-Value	Proportion	P-Value	Proportion	P-Value	Proportion	P-Value	Proportion
Frequency	0.3889	985/1000	0.9100	998/1000	0.6745	990/1000	0.2812	991/1000
Block Frequency	0.6246	990/1000	0.4865	993/1000	0.9357	988/1000	0.1681	993/1000
Cumulative Sums	0.3924	984/1000	0.4521	989/1000	0.0390	991/1000	0.6848	993/1000
Runs	0.0165	984/1000	0.5564	990/1000	0.9033	993/1000	0.1404	992/1000
Longest Run	0.4885	994/1000	0.5221	989/1000	0.1160	993/1000	0.7656	991/1000
Rank	0.0977	987/1000	0.4846	995/1000	0.1311	991/1000	0.0942	985/1000
FFT	0.0620	988/1000	0.5281	984/1000	0.1396	987/1000	0.8308	992/1000
Non Overlapping Template	0.2531	987/1000	0.5544	987/1000	0.0211	996/1000	0.6703	992/1000
Overlapping Template	0.4578	993/1000	0.5402	988/1000	0.4409	985/1000	0.0227	990/1000
Universal	0.9521	985/1000	0.0660	987/1000	0.0588	986/1000	0.2716	988/1000
Approximate Entropy	0.8343	987/1000	0.9634	985/1000	0.8129	991/1000	0.5181	991/1000
Random Excursions	0.3057	638/641	0.3008	595/598	0.6539	615/628	0.9867	621/632
Random Excursions Variant	0.8644	636/641	0.1362	595/598	0.8083	619/628	0.4869	627/632
Serial	0.5081	991/1000	0.8129	989/1000	0.1303	992/1000	0.0135	985/1000
Linear Complexity	0.6183	991/1000	0.8846	988/1000	0.7597	993/1000	0.3145	991/1000

Capítulo 4

PRNGs basados en caos y sus aplicaciones en criptografía

Este capítulo se centra en el uso de los PRNGs basados en sistemas caóticos, descritos en el capítulo anterior, para realizar aplicaciones criptográficas. En específico, se detalla el empleo de las series temporales caóticas, las cuales se han transformado a cadenas binarias cuya aleatoriedad se evalúa a través de pruebas NIST. Las cadenas binarias se usan en un sistema de comunicación segura, donde se aplican para el cifrado y la autenticación de mensajes. Las aplicaciones incluyen datos de mensajes como: imágenes en escala de grises, imágenes a color o RGB y texto.

El proceso de cifrado se lleva a cabo mediante la operación XOR entre el mensaje y la cadena binaria generada por el PRNG, mientras que la autenticación se realiza utilizando una función de hash. Además, se presentan los diagramas de bloques digitales que detallan el esquema de cifrado y autenticación para su implementación y síntesis en FPGA. Al final, se ilustran los resultados del procedimiento de comunicación entre una FPGA y una computadora a través de la interfaz RS232.

4.1. Sistemas de cifrado y autenticación utilizando PRNGs para comunicaciones seguras

En la actualidad, el mundo experimenta un intercambio masivo de datos digitales que contienen información crítica y sensible tanto de personas como de instituciones. Para proteger esta información, se emplean diversos sistemas criptográficos que garantizan la confidencialidad e integridad de los datos, evitando que usuarios no autorizados o hackers puedan leer o alterar parcial o totalmente los mensajes [53]. Dentro de estos sistemas, los PRNGs han sido utilizados para la generación de llaves criptográficas y el desarrollo de esquemas de cifrado y autenticación robustos, que buscan brindar seguridad durante el proceso de comunicación.

4.1.1. Esquemas de cifrado

Los esquemas de cifrado basados en sistemas caóticos han sido objeto de estudio y mejora desde su aparición en la década de 1990 [54]. Un esquema simple de cifrado simétrico basado en un PRNG, diseñado con sistemas caóticos, consiste en generar el texto cifrado enmascarando el texto plano (imagen o texto) con las cadenas binarias pseudoaleatorias que genera el PRNG, antes de su transmisión por un canal de comunicación. Típicamente, en el proceso de enmascaramiento se emplea la operación XOR. Este esquema es útil porque permite un cifrado rápido de la información y, en combinación con otras técnicas más complejas, permite desarrollar cifradores de flujo. La figura 4.1 ilustra el esquema de cifrado utilizando PRNGs, donde se observa que tanto el bloque transmisor como el receptor tienen el mismo PRNG.

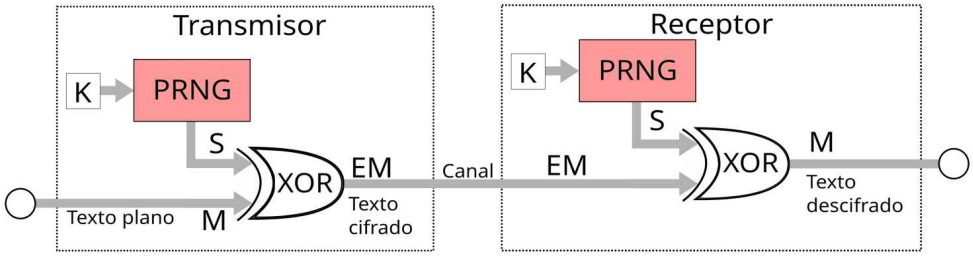


Figura 4.1: Esquema básico de cifrado utilizando un PRNG [54].

El funcionamiento del diagrama de la figura 4.1 es el siguiente: en el dispositivo transmisor, el PRNG es alimentado con la semilla para que genere las series de números pseudoaleatorios.

Estas series binarias se utilizan para enmascarar los píxeles de una imagen o los caracteres de un texto, byte a byte, antes de su transmisión. Durante la transmisión por el canal de comunicación, los datos viajan cifrados, de modo que un usuario no autorizado no podrá descifrarlos, incluso si conoce el sistema caótico, pero sin conocimiento de la semilla. Al llegar al dispositivo receptor, el mensaje es descifrado aplicando la operación XOR entre las mismas cadenas binarias, generadas por el PRNG del receptor, y los datos cifrados.

Un punto fundamental de este tipo de sistemas es el intercambio seguro de las semillas entre el transmisor y el receptor. Una solución para este problema, es el uso de algoritmos de sincronización de sistemas caóticos, como las formas Hamiltonianas y las técnicas OPCL, discutidas en [8]. Estos procedimientos de sincronización se ejecutan después de definir la semilla del PRGN en el transmisor, y una vez sincronizados ambos sistemas, se inicia la transmisión de los datos cifrados.

Hoy en día, continúa una extensa investigación y desarrollo de este tipo de algoritmos y esquemas que logran resistir ataques criptográficos. Por ejemplo, Guanrong Chen en [55], presenta un esquema de cifrado simétrico de alta seguridad en tiempo real para imágenes, utilizando un mapa caótico tridimensional como generador de secuencias aleatorias. En este esquema, se alteran tanto las posiciones de los píxeles de la imagen como sus valores de tonos de gris. Además, se emplea otro mapa caótico para confundir la relación entre la imagen cifrada y la original, lo cual incrementa la resistencia a los ataques diferenciales y estadísticos. De forma similar, en [56], se presenta otro esquema de cifrado de imágenes que utiliza un PRNG basado en un sistema caótico 3D. En este esquema, cada secuencia pseudoaleatoria generada a partir de una variable de estado se emplea para realizar el desorden a nivel píxel, el desorden a nivel bit, y la difusión de los valores de la imagen, reduciendo así patrones y la correlación entre los píxeles con el fin de aumentar la seguridad. El sistema ha demostrado ser resistente a ataques diferenciales y de fuerza bruta. Además, la semilla del PRNG proviene de una función de hash SHA-256 aplicada a la imagen. Por su parte, los autores en [57], describen un esquema para el cifrado de señales utilizando un PRNG basado en un sistema caótico 3D, implementado en una Raspberry Pi 3 mediante el método numérico RK4. El cifrado en este esquema se realiza bit a bit mediante la operación XOR, con un tamaño de bloque de 512 bits para la información cifrada. Las series pseudoaleatorias basadas en caos, también se aplican en esquemas de cifrado de video en tiempo real, como en [58], en los que se emplean múltiples sistemas caóticos para generar una llave

criptográfica que cifra el video en tiempo real mediante la operación XOR. Por otro lado, los esquemas de cifrado, también pueden implementarse utilizando TRNGs, como en [59], que utiliza las secuencias de un TRNG para la confusión de una imagen.

El cifrado de información puede llevarse a cabo por una gran variedad de técnicas, que pueden emplear o no PRNGs, por ejemplo, las cajas de sustitución, los cifradores de bloque y los esquemas de cifrado por clave asimétrica.

4.1.2. Etapas para autenticar mensajes

Cuando un mensaje viaja entre un dispositivo emisor y un receptor a través de un canal de comunicación, está expuesto a hackers, espías y criptoanalistas, quienes podrían alterarlo para su beneficio [53]. Por esta razón, uno de los criterios más importantes en el desarrollo de protocolos y esquemas criptográficos entre dos o más dispositivos, es garantizar la integridad de los datos durante su transmisión. Para asegurar esta propiedad, los esquemas de protección suelen utilizar funciones de hash, las cuales generan etiquetas de autenticación o resúmenes de mensaje, que funcionan como una firma digital y permiten verificar la autenticidad de la información. Las funciones de hash son algoritmos matemáticos utilizados en otros servicios, como la autenticación e identificación de usuarios mediante contraseñas.

En la actualidad, existen funciones de hash ampliamente conocidas y utilizadas en diversos protocolos. Sin embargo, el desarrollo de algoritmos más robustos y seguros sigue siendo objeto de investigación. Por ello, se han propuesto nuevas funciones de hash, incluidas aquellas que emplean sistemas caóticos en sus algoritmos. Por ejemplo, en el trabajo [60], se presenta un algoritmo de función de hash con excelentes propiedades estadísticas y una fuerte resistencia a la colisión, cuyo proceso para generar el valor de hash utiliza una red formada por 16 mapas caóticos unidimensionales. De manera similar, el artículo [61], propone un algoritmo de hash confiable basado en un mapa caótico con parámetros cambiables, acompañado de un análisis teórico sobre su eficiencia. Además, en [62], se describe una función de hash formada a partir de un mapa caótico, que destaca por la rapidez del algoritmo, su buena resistencia a la colisión y su facilidad para implementar en hardware y software. Otro trabajo relevante es [63], que introduce una función de hash ligera basada en el cifrador de bloque Saturnin. Finalmente, en [30] se presenta un algoritmo de función de hash, junto con los códigos de alto nivel para su implementación en un

microcontrolador, que emplea el método denominado el producto pseudo punto, el cual genera la etiqueta de autenticación de 112 bits utilizando las series pseudoaleatorias de un PRNG.

Otra relación adicional entre una función de hash y los sistemas caóticos, consiste en que los valores generados por estas funciones pueden utilizarse como condiciones iniciales o claves secretas para los sistemas caóticos. Por ejemplo, en el trabajo [64], se utiliza una función de hash SHA-512 para generar las condiciones iniciales de un sistema caótico que participa en un esquema de cifrado. De forma similar, en [65], se emplea una función de hash SHA-256 para generar la semilla.

Muchos algoritmos de funciones de hash son de dominio público, lo cual permite que cualquier persona pueda analizarlos y buscar vulnerabilidades para encontrar sus debilidades. Por ejemplo, en [66] se detalla como romper el algoritmo de función de hash MD5, mientras que [67], se centra en investigar acerca de la resistencia a la colisión de la función de hash SHA-1.

Como se demuestra en los trabajos mencionados, las funciones de hash basadas en modelos caóticos ofrecen un área de oportunidad para el desarrollo de algoritmos seguros y eficientes útiles en esquemas criptográficos.

4.1.2.1. Función de hash basada en el producto pseudo punto

La función de hash basada en el producto pseudo punto (PDP), propuesta en [68], genera el valor de hash para un mensaje de cualquier tamaño aplicando operaciones XOR, multiplicaciones y operaciones módulo con números primos grandes, entre el mensaje M y una clave S. Esta clave está compuesta por una serie de números pseudoaleatorios provenientes de un PRNG. Básicamente, el algoritmo concatena un grupo de variables H, que van desde H_1 hasta H_b, donde cada una tiene una longitud igual al número de bits generados por una iteración del PRNG. Por ejemplo, el PRNG_3D descrito en el capítulo anterior produce 24 bits en cada iteración. Si se elige $b = 7$, el valor de hash tendrá una longitud de 168 bits al concatenar 7 variables H. La ecuación 4.1 define el valor de hash basado en las variables H_b.

$$PDP(M, S) = H_1//H_2//\ldots//H_b \tag{4.1}$$

El algoritmo requiere dividir el mensaje M en porciones de igual tamaño, formando un grupo de datos concatenados como $M = M_1//M_2//M_3\ldots//M_m$. Por su parte, la clave S se debe

generar a partir de cadenas de bits concatenadas de la forma $S = S_1//S_2//S_3 \ldots //S_{m+2b-2}$. La ecuación 4.2 muestra las operaciones matemáticas necesarias para calcular cada variable H. Se observa en dichas expresiones que cada variable H sigue la misma estructura matemática, pero utiliza una serie S diferente. Además, se aprecia que en los primeros productos de H_1 y H_b, participan las series S_1 a la S_{2b}, lo cual hace necesario almacenar de forma temporal $2b$ iteraciones del PRNG y dos mensajes M, para poder calcular las variables H de forma paralela, considerando una implementación en FPGA.

Por otro lado, dado que la operación de multiplicación puede generar resultados cuyo tamaño en bits es el doble del original, es necesario aplicar una operación módulo para mantener el tamaño de los resultados en su longitud original [30]. Además, el algoritmo utiliza números primos grandes n_p para lograr una mejor distribución de los valores de hash y reducir las colisiones. Cabe destacar que el algoritmo también podría emplear operaciones módulo con potencias de 2, es decir, números de la forma 2^n. No obstante, esto reduciría la resistencia a la colisión.

$$
\begin{aligned}
H_1 =&[(M_1 \oplus S_1)(M_2 \oplus S_2) \bmod n_p] \oplus [(M_3 \oplus S_3)(M_4 \oplus S_4) \bmod n_p] \oplus \\
&\cdots \oplus [(M_{m-1} \oplus S_{m-1})(M_m \oplus S_m) \bmod n_p], \\
H_2 =&[(M_1 \oplus S_3)(M_2 \oplus S_4) \bmod n_p] \oplus [(M_3 \oplus S_5)(M_4 \oplus S_6) \bmod n_p] \oplus \\
&\cdots \oplus [(M_{m-1} \oplus S_{m+1})(M_m \oplus S_{m+2}) \bmod n_p], \\
H_3 =&[(M_1 \oplus S_5)(M_2 \oplus S_6) \bmod n_p] \oplus [(M_3 \oplus S_7)(M_4 \oplus S_8) \bmod n_p] \oplus \\
&\cdots \oplus [(M_{m-1} \oplus S_{m+3})(M_m \oplus S_{m+4}) \bmod n_p], \\
&\cdots\cdots, \\
H_b =&[(M_1 \oplus S_{2b-1})(M_2 \oplus S_{2b}) \bmod n_p] \oplus [(M_3 \oplus S_{2b+1})(M_4 \oplus S_{2b+2}) \bmod n_p] \oplus \\
&\cdots \oplus [(M_{m-1} \oplus S_{m+2b-3})(M_m \oplus S_{m+2b-2}) \bmod n_p]
\end{aligned}
\tag{4.2}
$$

En esta tesis se utilizan 7 variables H ($b = 7$) y los sistemas 3D y 4D permiten aumentar la longitud de los valores de hash a 168 y 224 bits.

4.1.3. Esquema de cifrado y autenticación general

La figura 4.2 ilustra el diagrama de bloques general del esquema de cifrado por XOR y de autenticación por resúmenes de mensaje empleando la función de hash PDP. De forma general, el sistema funciona de la siguiente manera:

1. El transmisor ejecuta la función de hash PDP al mismo tiempo que cifra y envía los mensajes M por el canal de comunicación. En ambas operaciones, se utilizan las cadenas binarias S proporcionadas por el PRNG del transmisor.

2. El transmisor envía el valor de hash calculado.

3. El receptor descifra el texto cifrado y, al mismo tiempo, ejecuta la misma función de hash con los datos descifrados y las mismas series S producidas por el PRNG del receptor.

4. El receptor obtiene el valor de hash proveniente del transmisor.

5. Se comparan los valores de hash del transmisor y del receptor para autenticar el mensaje.

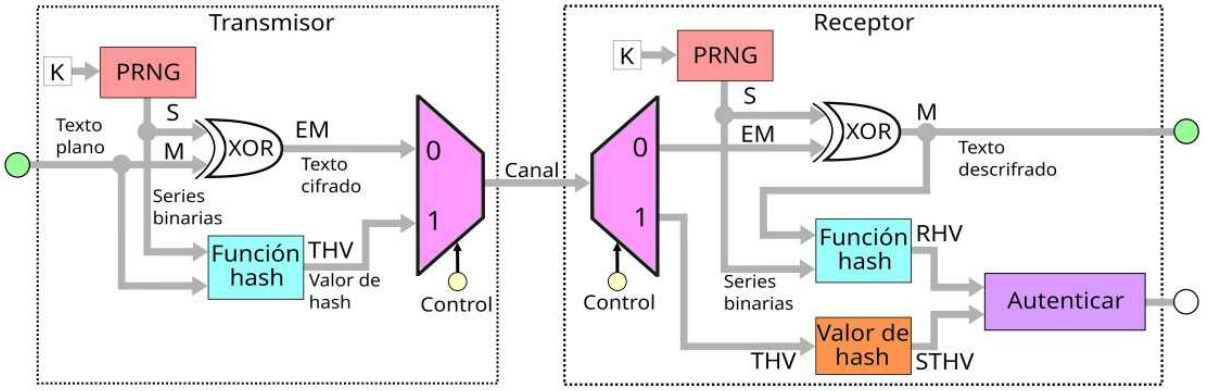


Figura 4.2: Esquema general de cifrado y autenticación.

Este esquema se puede aplicar a cualquier imagen y texto. Además, funciona con PRNGs que proporcionan diferentes cantidades de bits pseudoaleatorios.

4.1.4. Esquema de cifrado y autenticación en FPGA

El esquema de cifrado y autenticación de imágenes y texto para un dispositivo transmisor se muestra en el diagrama de bloques de la figura 4.3. Para facilitar la identificación y comprensión de las diferentes partes, el esquema está compuesto por bloques de distintos colores, cada uno encargado de una tarea específica. Los dos bloques en color rojo corresponden al PRNG, basado en el mapa caótico 2D, y a un contador con bit de parada, el cual descarta las series pseudoaleatorias generadas durante el tiempo transitorio del oscilador. En verde, se encuentran una memoria

ROM, que almacena el texto plano (imagen o texto), y un registro PIPO, el cual guarda un mensaje anterior M, necesario para el bloque de función de hash. Los bloques azules incluyen una memoria basada en registros para almacenar un conjunto de cadenas binarias S, el bloque encargado de calcular la función de hash, un multiplexor, y un contador con parada que controla el selector del multiplexor. Los bloques amarillos corresponden con la lógica combinacional y una máquina de estados finitos tipo Moore, que sincronizan y controlan la operación del sistema. Finalmente, los bloques en color rosa, se encargan de transmitir el texto cifrado y la etiqueta de autenticación, mientras que la compuerta XOR, en color blanco, realiza el procedimiento de cifrado.

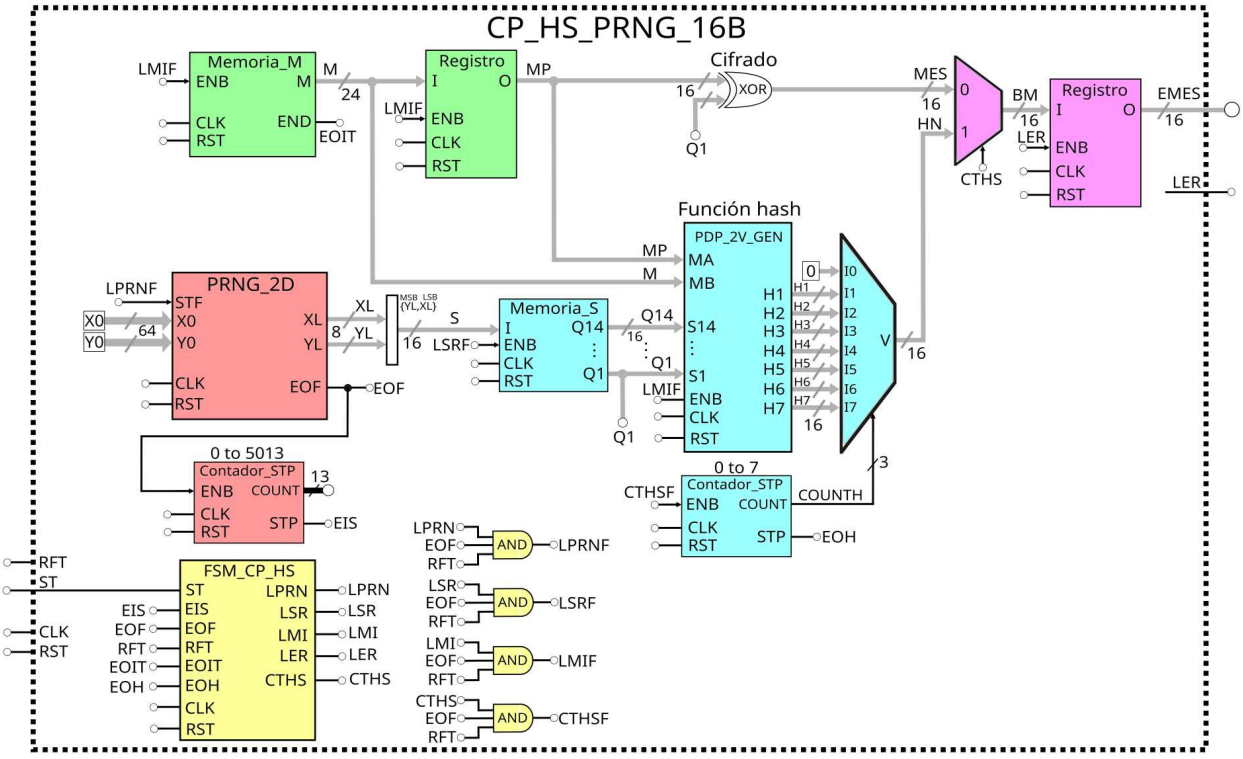


Figura 4.3: Esquema del bloque transmisor para la etapa de cifrado y autenticación en FPGA.

El bloque de mayor jerarquía, denominado CP_HS_PRNG_16B, fue descrito en Verilog y sintetizado en una FPGA para enviar mensajes cifrados y valores de hash a una computadora. Cabe recalcar que el esquema aprovecha la propiedad de paralelismo de la FPGA, ya que la función de hash ocurre de forma simultánea al proceso de cifrado, donde ambos emplean las mismas series de números aleatorios S. Los esquemas de cifrado y autenticación que utilizan los PRNGs 3D y 4D presentan una estructura similar; la diferencia está en las longitudes de las cadenas de bits

pseudoaleatorias S y los mensajes M.

En las siguientes subsecciones se explican algunas características de los bloques del esquema.

4.1.4.1. Bloque para almacenar el mensaje

El bloque denominado Memoria_M está formado internamente por un contador y una o más memorias ROM (dependiendo del tipo de texto plano). Estas memorias almacenan el mensaje dividido, representado como $M = M_1//M_2//\ldots//M_m$. Por ejemplo, en el caso de una imagen en escala de grises usando el PRNG_2D que genera 16 bits por iteración, una imagen de tamaño $w \times h$ debe transformarse en una matriz unidimensional donde cada celda almacene el valor de un píxel, es decir, un byte de información. Así, una imagen de 128×128 píxeles produciría una matriz de 16384 elementos.

La figura 4.4 muestra el diagrama interno del bloque Memoria_M, que almacena una imagen en escala de grises de 128×128. Cada mensaje M producido por este bloque se forma concatenando 2 píxeles contiguos. Por ello, el contador conectado a la memoria aumenta 2 unidades en cada iteración. El mismo bloque de memoria puede emplearse para textos, donde cada mensaje M consiste en dos caracteres contiguos.

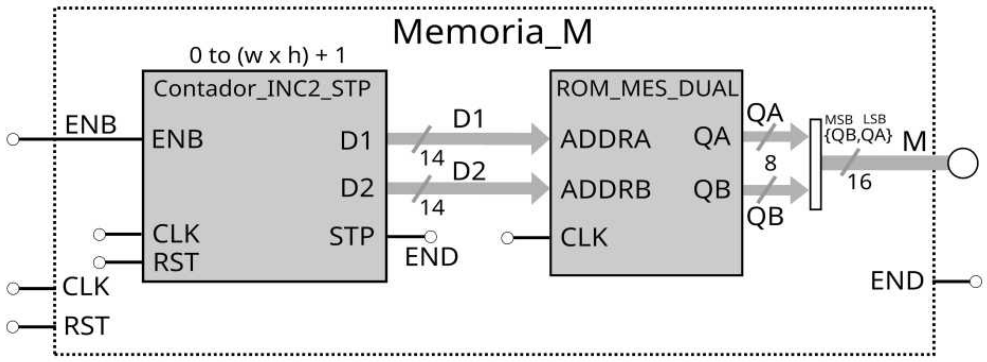


Figura 4.4: Bloque de memoria para almacenar los mensajes.

Las formas de almacenar imágenes varían de acuerdo al tipo de esta y al tamaño del mensaje M. Por ejemplo, en el caso de una imagen RGB en un sistema que utiliza cadenas binarias de 24 bits, una manera de abordar el problema es almacenar cada componente de color de un píxel de forma separada en tres memorias ROM, y luego formar el mensaje M concatenando los tres

colores para obtener 24 bits.

4.1.4.2. Bloque de función de hash

Al analizar el algoritmo matemático descrito en la ecuación 4.1 se observa que las operaciones tienen una estructura repetitiva, en la cual solo cambian los valores de los mensajes M y las cadenas binarias S. Por lo tanto, cada variable H puede expresarse como una secuencia de operaciones iterativas, donde los nuevos valores de H dependen de los anteriores. Como ejemplo, la tabla 4.1 muestra las operaciones iterativas para obtener los valores de H_1. Aunque las variables H no son arreglos matriciales, por conveniencia, las iteraciones se denotan como $H[1]$, $H[2]$ y $H[3]$ para destacar la diferencia entre los valores previos y los nuevos.

Tabla 4.1: Ejemplo de la ecuación iterativa para la variable H_1.

Iteración	Ecuación iterativa
1	$H_1[1] = [(M_1 \oplus S_1)(M_2 \oplus S_2) \bmod n_p] \oplus H_1[0]$
2	$H_1[2] = [(M_3 \oplus S_3)(M_4 \oplus S_4) \bmod n_p] \oplus H_1[1]$
3	$H_1[3] = [(M_5 \oplus S_5)(M_6 \oplus S_6) \bmod n_p] \oplus H_1[2]$

Gracias al algoritmo iterativo, se puede utilizar una arquitectura digital con un bloque XOR acumulador (similar al MAC). Dicha estructura se encuentra dentro del bloque PDP_2V_GEN de la figura 4.3.

La figura 4.5 presenta un diagrama de bloques simplificado acerca de cómo implementar de forma iterativa la ecuación 4.1 en hardware digital, considerando 7 variables H para obtener el valor de hash. Dicha figura ilustra un conjunto de 14 registros conectados a 7 bloques PDP. En una iteración del algoritmo, cada bloque PDP realiza las operaciones XOR, la multiplicación y el módulo para calcular el valor de una H. El esquema interno de estos bloques se detalla en la figura 4.6, donde las entradas requieren dos valores de M, dos de S y el valor anterior de H.

La figura 4.5 se divide en dos partes: la mitad izquierda muestra la configuración de los registros y las operaciones realizadas durante la primera iteración, mientras que la parte derecha ilustra la configuración para la segunda iteración del algoritmo. Para actualizar las series S, los registros realizan operaciones de desplazamiento. Este proceso se repite de manera iterativa hasta que se procesa el último valor del mensaje M.

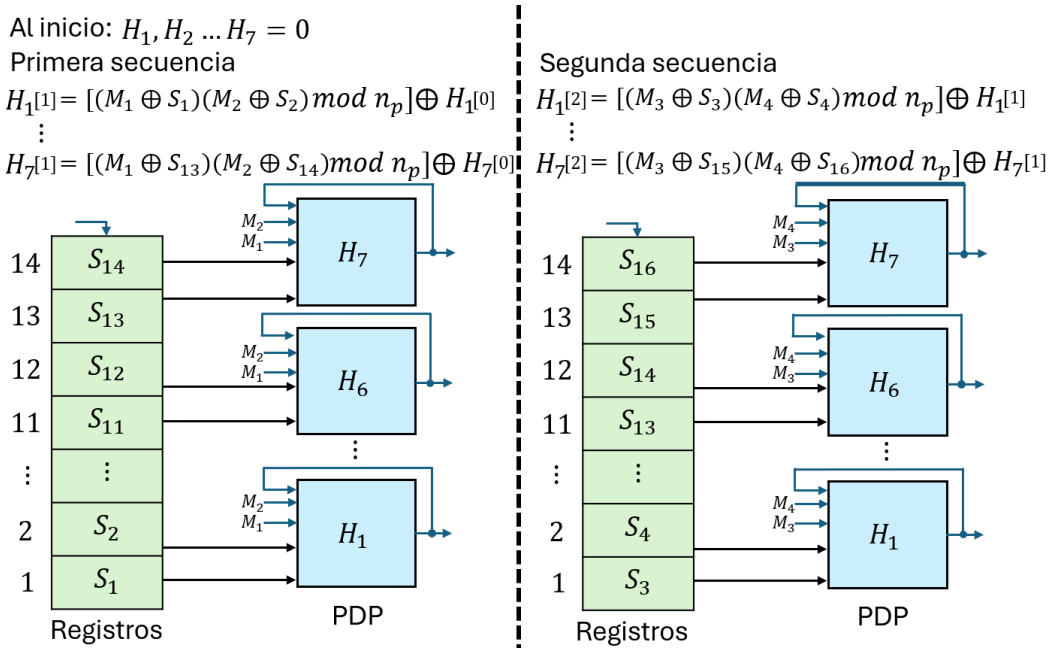


Figura 4.5: Diagrama de bloques general para la función de hash PDP con $b = 7$.

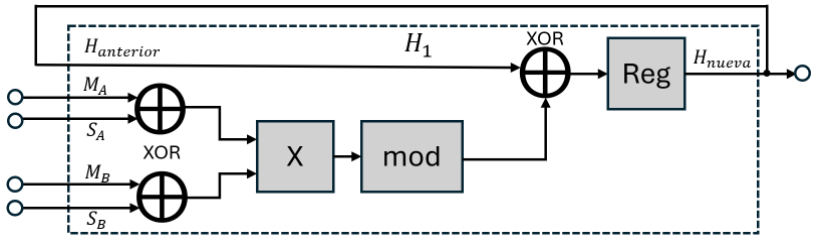


Figura 4.6: Operaciones para calcular una variable H.

Debido al número limitado de recursos en la FPGA, una forma de reducir el uso de hardware es multiplexar la estructura de la figura 4.6. Por esta razón, el bloque PDP_2V_GEN, detallado en la figura 4.7, incluye internamente dos bloques PDP_MOD_MUX_H que permiten calcular cuatro valores de H reutilizando la misma estructura digital. Además, dicho bloque incluye siete bloques XOR acumuladores, un contador que controla los selectores para que, en cuatro ciclos de reloj, se calculen las siete variables H, y un grupo de flip-flops necesarios para controlar la acumulación de los resultados.

En el caso de la operación módulo con los números primos, el software de Quartus II permite sintetizar dicha operación utilizando el símbolo %. Los números primos seleccionados para los diferentes PRNGs se muestran en la tabla 4.2.

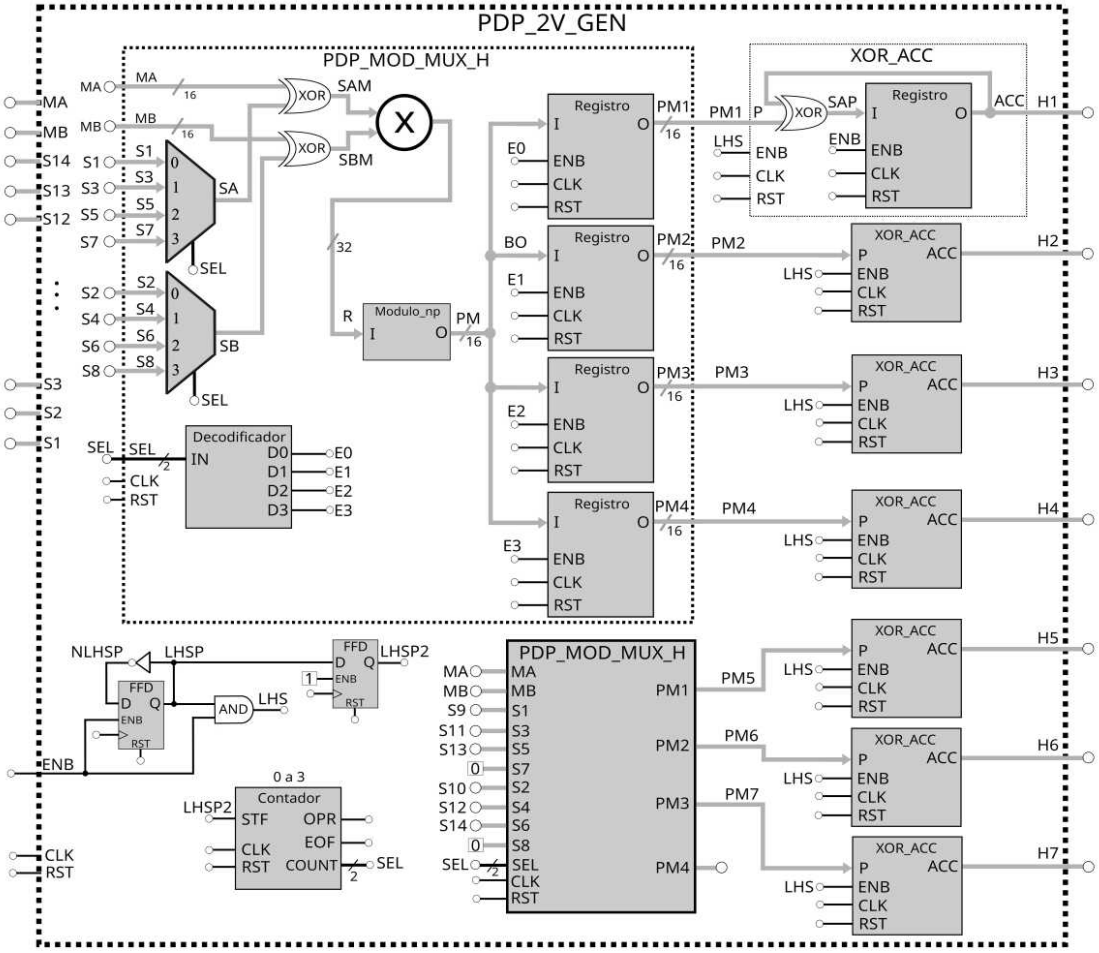


Figura 4.7: Diagrama de bloques para la función de hash PDP empleando un valor de hash de 112 bits.

Tabla 4.2: Longitud de multiplicaciones y números primos utilizados en las operaciones módulo.

PRNG	Longitud de bits para multiplicaciones	Números primos
2D	32	$2^{16}+1$
3D	48	$2^{24}+43$
4D	64	$2^{31}-1$

El sistema 3D no debe considerar resultados mayores al valor 2^{24}.

Por último, en el diagrama de mayor jerarquía de la figura 4.3 aparecen otros tres bloques en color azul. El bloque Memoria_S está compuesto por 14 registros PIPO conectados en serie, que se encargan de almacenar las últimas 14 cadenas binarias generadas por el PRNG. Por su parte, el multiplexor se encarga de enviar los valores de cada variable H a un registro, para

transmitirlos a través del canal de comunicación. El selector de este multiplexor está controlado por un contador, el cual comienza a contar una vez que se ha enviado el mensaje cifrado.

4.1.4.3. Bloque de control

El bloque de control del diagrama mostrado en la figura 4.3 consiste en una máquina de estados tipo Moore, la cual sincroniza las memorias, la función de hash y los bloques para el cifrado y la transmisión de los datos. La tabla de verdad que describe los estados de la FSM se presenta en la tabla 4.3.

Tabla 4.3: Tabla de verdad de la máquina de estados para el sistema de cifrado y autenticación.

Estado	**Entradas**						Estado	**Salidas**				
actual	ST	EIS	EOF	RFT	EOIT	EOH	siguiente	LPRN	LSR	LMI	LER	CTHS
0	x	0	x	x	x	x	0	1	1	0	0	0
0	x	1	x	x	x	x	1					
1	0	x	x	x	x	x	1	0	0	0	0	0
1	1	x	x	x	x	x	2					
2	x	x	0	0	x	x	2	1	1	1	0	0
2	x	x	0	1	x	x	2					
2	x	x	1	0	x	x	2					
2	x	x	1	1	x	x	3					
3	x	x	x	x	x	x	4	1	0	0	0	0
4	x	x	x	x	0	x	4	1	0	0	1	0
4	x	x	x	x	1	x	5					
5	x	x	0	0	x	x	5	1	0	0	0	1
5	x	x	0	1	x	x	5					
5	x	x	1	0	x	x	5					
5	x	x	1	1	x	x	6					
6	x	x	x	x	x	x	7	1	0	0	0	1
7	x	x	x	x	x	0	7	1	0	0	1	1
7	x	x	x	x	x	1	8					
8	x	x	x	x	x	x	8	0	0	0	0	0
Res	x	x	x	x	x	x	0	0	0	0	0	0

Una descripción general de las entradas y las salidas de la FSM es la siguiente: (los nombres

de los puertos de la FSM están basados en siglas en inglés).

- ST (Start): señal de inicio para todo el sistema.
- EIS (End Initial Series): se activa cuando el bloque Counter_ST termina de contar cierto número específico de iteraciones del PRNG, descartando las primeras iteraciones del PRNG correspondientes al transitorio del oscilador.
- EOF (End of function): señal que se activa al terminar cada iteración del PRNG.
- RFT (Ready For Transmission): señal que se activa cuando el bloque de comunicación serial ha terminado de enviar una trama.
- EOIT (End of Image Transmission): señal que indica que todos los mensajes M han sido cifrados.
- EOH (End of Hash): indica que se ha transmitido todo el valor hash.
- LPRN (Load PRNG): inicia al PRNG basado en sistema caótico.
- LSR (Load S Registers): habilita al bloque Memory_S para almacenar las últimas 14 cadenas binarias S generadas por el PRNG.
- LMI (Load Memory Image): actualiza un nuevo mensaje M y ejecuta una iteración en la función de hash.
- LER (Load Encryption Register): habilita el registro que envía los datos cifrados al puerto serial.
- CTHS (Counter hash): incrementa una unidad del contador que controla la entrada del multiplexor del bloque de hash, conectando una variable H a la salida.

4.1.4.4. Transmisor y receptor con etapa RS232

El diagrama de bloques del transmisor que utiliza el bloque CP_HS_PRNG_16B y el bloque para la interfaz RS232 se muestra en la figura 4.8. La finalidad del esquema completo es mostrar la aplicación de los PRNGs en las etapas de cifrado y autenticación de mensajes, así como una

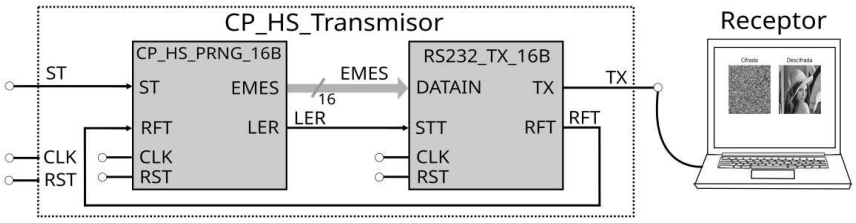


Figura 4.8: Diagrama para el sistema transmisor receptor.

manera de implementar la función de hash PDP con una mayor longitud de valores de hash. Sin embargo, el sistema aún no representa un protocolo criptográfico completo.

Por otro lado, el bloque de recepción para descifrar y autenticar los mensajes también se puede desarrollar en una FPGA. De hecho, es posible utilizar la interfaz RS232 para la transmisión y recepción de los mensajes empleando dos FPGAs. Solo se requiere que el mensaje descifrado se almacene en una memoria RAM para posteriormente enviarlo a una computadora, así como alguna manera para verificar el valor de ambas etiquetas de autenticación. La figura 4.9, muestra el diagrama interno del bloque receptor para su síntesis en FPGA. Su configuración es similar al bloque transmisor, solo añade otros bloques que controlan el proceso de recepción por RS232 y que almacenan el mensaje descifrado en una memoria RAM. Además, el sistema incluye unos bloques decodificadores para visualizar los valores hexadecimales en indicadores de 7 segmentos y así verificar los valores de la hash.

4.2. Resultados del esquema de cifrado y autenticación en FPGA

En la siguiente subsección se muestran los resultados del proceso de cifrado y autenticación de imágenes en escala de grises, imágenes RGB y un texto, utilizando el esquema de comunicación transmisor receptor entre una FPGA y una computadora. Para las imágenes en escala de grises, se utilizaron las cadenas binarias generadas por el PRNG 2D, mientras que para las imágenes RGB se emplearon las del PRNG 3D, y para el texto, las del PRNG 4D. Los diagramas de los esquemas del bloque transmisor que utilizan los PRNGs 3D y 4D presentan la misma estructura mostrada en la figura 4.3, donde se debe cambiar el número primo y ajustar la longitud de los buses de datos a 24 y 32 bits, respectivamente. Además, se requiere modificar la memoria ROM para almacenar los distintos tipos de mensaje M. Cabe destacar que en todos los experimentos

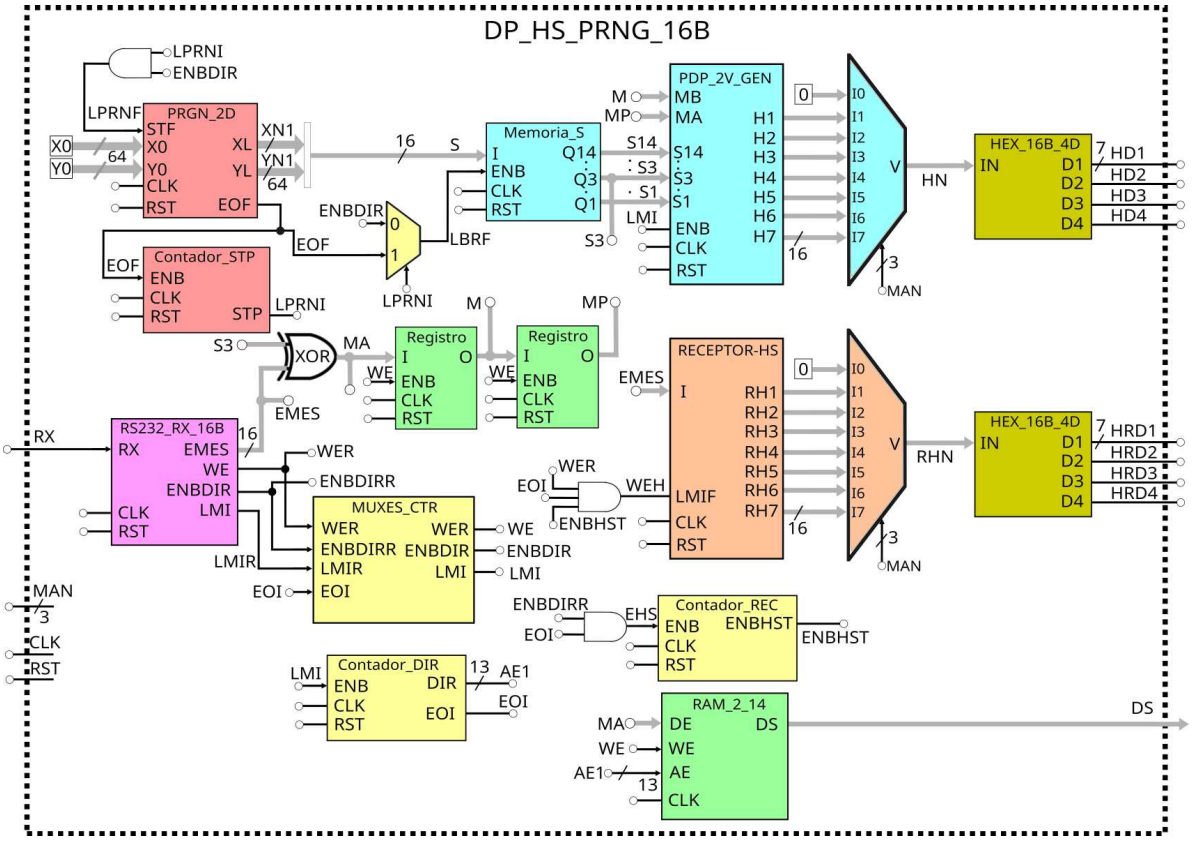


Figura 4.9: Esquema del bloque receptor para la etapa de descifrado y autenticación en FPGA.

se consideró un total de 20000 iteraciones del PRNG para eliminar el tiempo transitorio.

Para las pruebas de ambas etapas, el bloque transmisor fue sintetizado en una FPGA Cyclone III EP3C16F484C6, mientras que el bloque receptor se realizó mediante código en Matlab. La comunicación se realizó por la interfaz RS232. No obstante, debido a las limitaciones de memoria de dicho modelo de FPGA, en las pruebas de imágenes RGB se utilizaron tamaños de 128×128 píxeles.

4.2.1. Cifrado y autenticación de imágenes a escala de grises

La primera prueba con imágenes en escala de grises se llevó a cabo utilizando una imagen de Lena de 160×160 píxeles. En la figura 4.10 se presenta la imagen original, cifrada y descifrada. Debido a que los datos cifrados no sufrieron alteraciones durante su transmisión a través del canal de comunicación, ambas etiquetas de autenticación resultaron idénticas. La tabla 4.4 comparte la etiqueta de autenticación generada por ambas imágenes.

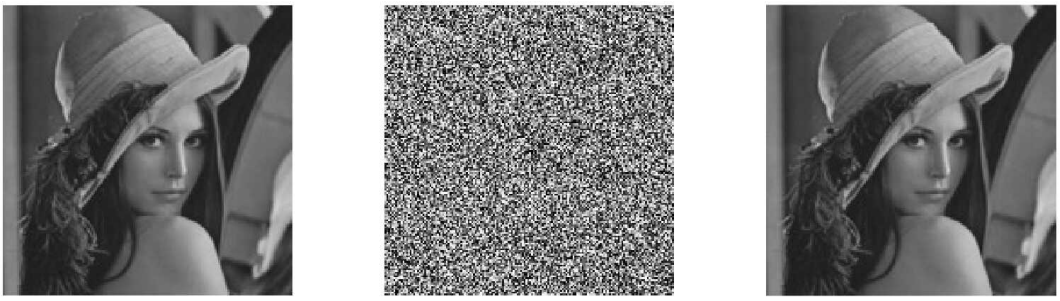


Figura 4.10: Cifrado y descifrado con la imagen de Lena a escala de grises 160×160. Original (izquierda), cifrada (centro) y descifrada (derecha).

Tabla 4.4: Etiqueta de autenticación obtenida con la imagen de Lena a escala de grises.

Dispositivo	Valor de hash de 112 bits
Transmisor/receptor	0x32877FCA6648B91CBCFC4119EB27

De manera similar, se realizó una prueba con la imagen del camarógrafo. La figura 4.11 ilustra tres imágenes a escala de grises del camarógrafo, con una resolución de 180×180 píxeles. En este caso, se modificó un píxel a color blanco en los mensajes descifrados de cada imagen, con el objetivo de verificar que un cambio mínimo, como la alteración de un solo píxel, genera diferencias significativas en las etiquetas de autenticación. La tabla 4.5 proporciona los resultados de los valores de la hash al cambiar un píxel en una posición específica.

Figura 4.11: Imágenes del camarógrafo a escala de grises 180×180, con un píxel blanco en la posición: [60,60] (izquierda), [90,90] (centro) y [120,120] (derecha).

Tabla 4.5: Etiquetas de autenticación de las imágenes del camarógrafo.

Imagen	Valor de hash de 112 bits
Original	0xE1C9237411B7A0F75C6E49A63359
[60,60]	0x877359AEF710BC1B151A542893C1
[90,90]	0xC665147F99C7DE46A2C407BADF24
[120,120]	0xA9EF9F852B24C520654ECE9F549B

4.2.2. Cifrado y autenticación de imágenes RGB

Al igual que con las ilustraciones anteriores, se efectuaron pruebas en imágenes RGB modificando ciertas posiciones de píxeles, utilizando los colores blanco y negro. Primero, la figura 4.12 muestra la imagen RGB de Lena, con una resolución de 128×128 píxeles. De izquierda a derecha se aprecia la imagen original, la cifrada y la descifrada. Por su parte, la figura 4.13 presenta la misma imagen tres veces, donde en cada una de ellas se ha modificado un píxel con color blanco, con el fin de comprobar el comportamiento de la función hash. La tabla 4.6 presenta los valores de hash resultantes. De manera similar, la figura 4.14 presenta tres imágenes de frutas, con la misma resolución, en las que se ha modificado un píxel con un color negro. Las etiquetas de autenticación correspondientes se anexan en la tabla 4.7.

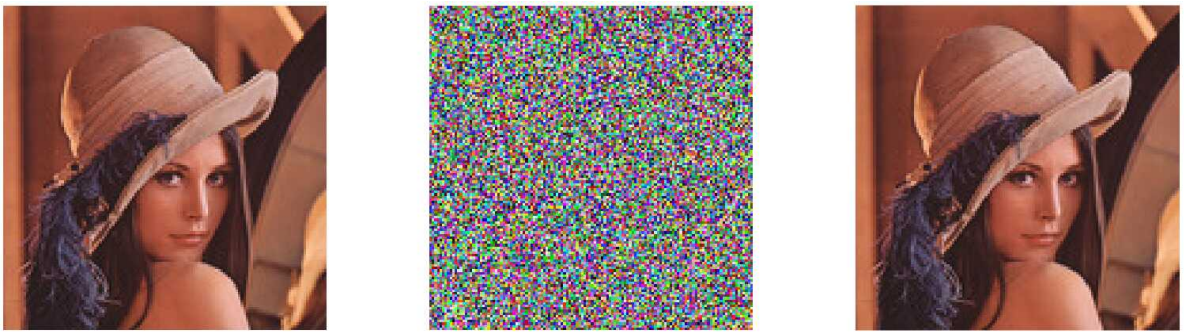


Figura 4.12: Cifrado/descifrado con la imagen RGB de Lena de 128×128. Original (izquierda), cifrada (centro) y descifrada (derecha).

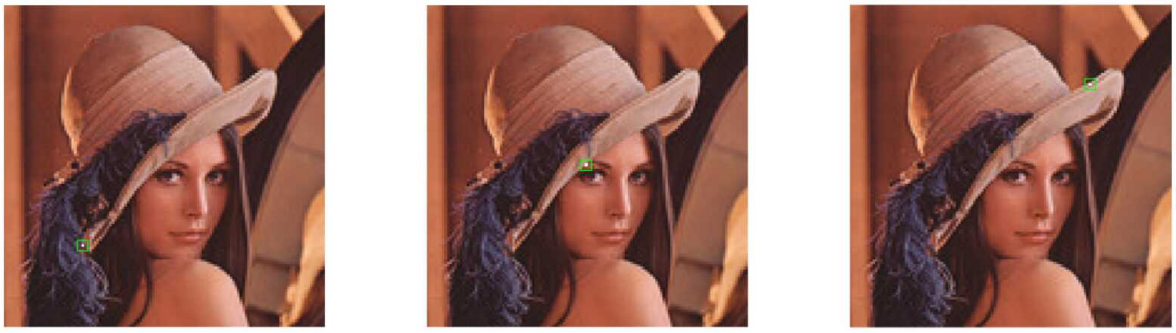


Figura 4.13: Imágenes RGB de Lena de 128×128, con un píxel blanco en la posición: [96,32] (izquierda), [64,64] (centro) y [32,96] (derecha).

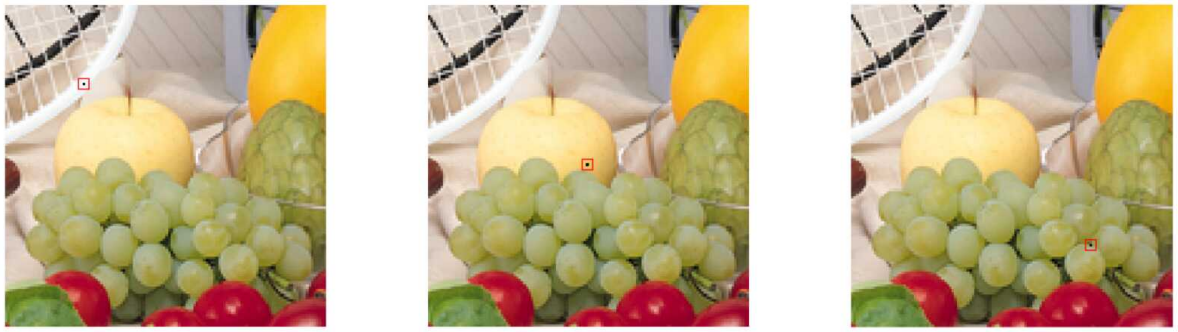


Figura 4.14: Imágenes RGB de frutas de 128×128, con un píxel negro en la posición: [32,32] (izquierda), [64,64] (centro) y [96,96] (derecha).

Tabla 4.6: Etiquetas de autenticación de las imágenes RGB de Lena de la figura 4.13.

Imagen	Valor de hash de 168 bits
Original	0x5B1A6DB8F6034F97462389100D3E20A9AA8F073B72
[32,96]	0x0C8D635E4968B301DE98B60C9977AE1281773E6F40
[64,64]	0x0DBAB976A95960BD277D6442F33EED41192B6230E1
[96,32]	0x4134A439A3D770AD54E419B2BFAD391316A77DC8C9

4.2.3. Cifrado y autenticación de texto

Para el caso del cifrado y autenticación de un texto, los valores ASCII de cada carácter se almacenaron en la memoria ROM de la FPGA. Dado que el PRNG 4D genera series de 32 bits en cada iteración, cada mensaje M se formó utilizando 4 caracteres ASCII contiguos. En la

Tabla 4.7: Etiquetas de autenticación de las imágenes RGB de frutas de la figura 4.14.

Imagen	Valor de hash de 168 bits
Original	0x4E260CFAFB172D69D637AF29416EA761A57ADD66A3
[32,32]	0xCDF6332129D7B58401446A61F881C589BA84CF0D73
[64,64]	0x42DB2876FB9F984B6BC069FFE21A2CF395E03986D9
[96,96]	0x2797DCA20D2C3AD4BA3DE131EF098C58E85896410D

figura 4.15 se presenta un conjunto de textos de 512 caracteres: el primero es un resumen corto de esta tesis, el segundo corresponde al texto cifrado, y el tercero al texto descifrado. En este último, se puede observar que la letra a fue modificada por una b. Como resultado, las etiquetas de autenticación del texto original y el descifrado son diferentes, ya que un cambio en un solo carácter genera un valor de hash completamente distinto. Los valores de hash se detallan en la tabla 4.8. Es importante mencionar que el texto para este esquema de cifrado y autenticación podría codificarse en el sistema posicional enumerado BASE64, para emplear un formato de texto útil en protocolos basados en texto.

Tabla 4.8: Etiquetas de autenticación de los textos del resumen corto de esta tesis.

Dispositivo	Valor de hash de 224 bits
Transmisor	0x2A58F6BE7E0C047E486F429257A6DDCF2E2922457E848EA31BF2663E
Receptor	0x5747F9CF1196C2953A3052BE1E1FF00775A930AC3782F85E4BD6C4CC

4.2.4. Consumo de recursos en la FPGA

La tabla 4.9 presenta el consumo de recursos al sintetizar en una FPGA Cyclone III EP3C16F484C6 los bloques transmisores, que incluyen las etapas de cifrado y autenticación utilizando un PRNG.

Resumen: Los generadores de números pseudoaleatorios tienen aplicaciones generales en criptografía. Sin embargo, el reto actual es garantizar la mejor aleatoriedad que es medida a través de pruebas NIST y TestU01. Los sistemas caóticos de tiempo continuo y discreto son una opción para diseñar los generadores, cuya ventaja es el uso de poco hardware que puede describirse en Verilog, y llevarse a layout para su fabricación en tecnologías de circuitos integrados CMOS. En esta presentación se muestran tres PRNG

(a)

v.÷£®_Ø\<b@□Rwc|/îms&Ä´ýD§²NE□¿Vã'ò□à¦^ÏÔ□Q^ö[P□÷e,□¡¢
¨sRe~y¯□¢Ùô»g□□K²²□qî@å□â¸îx□\s§□ñÄ□Àµ□'! °©k□□
Ëð?Í□¼jÛþ;éfÚ,□□dmXÐ×0S□Þ□ÔÝI□G1□s|»êÊ
G³ñ(Æ$Ó¢iH¹□[Kv□%Đc□□Æ|QÎá¬2&áÏ'G□Ï+□ªg7¯6ñ□ºÔ□Tq□~òÿ¬□ýãà□□¦¶HL²□Ú□□
□ZG □□xÎîÆ

(b)

Resumen: Los generadores de números pseudoaleatorios tienen aplicaciones generales en criptografía. Sin embargo, el reto actual es garantizar la mejor aleatoriedad que es medida a través de pruebas NIST y TestU01. Los sistemas caóticos de tiempo continuo y discreto son una opción para diseñar los generadores, cuya ventaja es el uso de poco hardware que puede describirse en Verilog, y llevarse b layout para su fabricación en tecnologías de circuitos integrados CMOS. En esta presentación se muestran tres PRNG

(c)

Figura 4.15: Texto corto del resumen de la tesis: a) texto original, b) texto cifrado y c) texto descifrado con una modificación en un carácter.

Tabla 4.9: Consumo de recursos en una FPGA Cyclone III EP3C16F484C6 de los bloques transmisores.

Transmisor con PRNG:	2D	3D	4D	Disponibles
Elementos lógicos (LE)	4,382	8,812	9,878	15,408
Funciones combinacionales	4,201	8,473	9,511	15,408
Registros lógicos	1,141	1,877	2,150	15,408
Bits de memoria	263,168	393,216	393,216	516,096
Multiplicadores embebidos de 9-bit	100	110	48	112
Frecuencia máxima (MHz)	14.88	8.67	5.81	–

Capítulo 5

Esquema de cifrado y autenticación bajo el protocolo MQTT en Raspberry Pi

5.1. Introducción

En este capítulo se detalla la implementación de un esquema de comunicación segura en Raspberry Pi, para el cifrado y la autenticación de mensajes utilizando los PRNGs caóticos 2D, 3D y 4D, programados en Python. A diferencia de la implementación anterior en FPGA, este esquema utiliza el protocolo MQTT para la transmisión y recepción de datos cifrados, empleando tres dispositivos Raspberry Pi 3, configurados como servidor (broker), publicante y suscriptor. Además, se presenta el diagrama del esquema de comunicación, algunos pseudocódigos que detallan las funciones del sistema y los resultados del cifrado y autenticación de imágenes.

5.2. Diseño de los PRNGs en Python

Como se mencionó en el capítulo 3, existen diversas formas para obtener cadenas binarias pseudoaleatorias a partir de las series temporales de un sistema caótico. Cada técnica requiere diferentes operaciones y, por lo tanto, de un mayor o menor tiempo de procesamiento. Así, cuando un sistema caótico se describe con aritmética de punto flotante en un lenguaje como Python,

una manera sencilla y con bajo costo computacional para generar las cadenas de bits, consiste en convertir los números en punto flotante a un número entero, y a partir de este, tomar los 8 bits menos significativos de una variable entera mediante la operación módulo 256. El proceso de conversión solo requiere de un producto 2^f [69], tal y como se muestra en la ecuación 5.1. No obstante, en esta conversión es necesario conocer los rangos de amplitud de las series temporales para seleccionar el mejor valor de f, que corresponde con el número de bits fraccionarios de la distribución de punto fijo en 32 bits.

$$Valor_{punto-fijo} = Valor_{punto-flotante} \times 2^f \tag{5.1}$$

Tomando como ejemplo el sistema 4D, la dinámica caótica se mantiene con una distribución de punto fijo 1:8:23, por lo tanto, el factor de conversión de valores en punto flotante a número entero es 2^{23}.

El proceso para obtener las cadenas de bits pseudoaleatorias a partir del sistema caótico 4D en Python es mostrado en el pseudocódigo 5.1. En esta función los parámetros de entrada son: la semilla para el PRNG, el número de cadenas binarias necesarias en el proceso de cifrado y el número de iteraciones que han de eliminar el tiempo transitorio del oscilador. Como salida, la función devuelve un arreglo de cadenas binarias (S) de 32 bits. La misma secuencia de comandos se aplica para otros sistemas caóticos 2D, 3D, 4D, 5D, etc., con una estructura similar. Es importante resaltar que el número de cadenas binarias dependerá del tamaño del mensaje original por cifrar. Por ello, es necesario contar con alguna función previa que determine el tipo y tamaño del mensaje, ya sea una imagen o texto.

```
# Entradas: semilla (K), número de series (NS), transitorio (TT)
# Salidas: arreglo de secuencias binarias (S) de 32 bits
funcion PRNG_4D(K, NS, TT):
    # Iniciar las condiciones iniciales: x[0], y[0], z[0], w[0] ← K
    # Definir parámetros del sistema caótico: (a,b,h)
    # Inicializar arreglos
    # Ejecutar el sistema caótico
    for n in range(NS - 1):
        x[n + 1] = x[n] + h(...)          # Ecuaciones discretas
```

```
        ...
        w[n + 1] = w[n] + h(...)
        XFIX = int(x[n + 1] * 2**23)     # Convertir a número entero
        ...
        WFIX = int(w[n + 1] * 2**23)
        BX = XFIX % 256                    # Obtener cadenas binarias
        ...
        WZ = WFIX % 256
        SI[n] = concatenar(BW, BZ, BY, BX)  #concatenar 4 bytes
    S = SI[TT:NS]  # Eliminar transitorio
    return S       # Regresar series binarias
```

Pseudocódigo 5.1: Función para generar las cadenas binarias utilizando el PRNG 4D.

5.3. Pruebas estadísticas NIST

Al igual que con cualquier otro PRNG, las cadenas de bits pseudoaleatorios producidos deben ser validados mediante pruebas estadísticas. La tabla 5.1, comparte los resultados de las pruebas NIST para el sistema 4D, tomando en cuenta 1000 secuencias de 10^6 bits cada una.

5.4. Sistema de cifrado y autenticación de mensajes por protocolo MQTT

El MQTT es un protocolo ligero y eficiente para transmitir cualquier tipo de mensajes, ya sean imágenes, archivos o textos, entre uno o varios dispositivos [70]. Debido a su bajo consumo de potencia y ancho de banda, típicamente, se utiliza en aplicaciones del Internet de las Cosas para la lectura y control de diversos sensores y actuadores. También, tiene aplicación en la domótica para control de dispositivos inteligentes.

El modelo del MQTT está constituido por tres partes bien definidas. La primera de ellas, es el servidor, que actúa como un intermediario para enrutador los mensajes entre los publicantes y los suscriptores. En este protocolo, la comunicación entre el publicante y el suscriptor no es directa,

Tabla 5.1: Resultados de las pruebas NIST utilizando la conversión de punto flotante a número entero para las variables de estado x, y, z y w del sistema caótico 4D. Las condiciones iniciales se variaron en el rango $[-0.4, 0.4]$.

Prueba estadística	X		Y		Z		W	
	P-Value	Proportion	P-Value	Proportion	P-Value	Proportion	P-Value	Proportion
Frequency	0.6329	992/1000	0.6100	987/1000	0.0887	994/1000	0.4635	992/1000
Block Frequency	0.9442	989/1000	0.3145	987/1000	0.1031	987/1000	0.5022	989/1000
Cumulative Sums	0.6558	992/1000	0.2178	986/1000	0.5625	993/1000	0.4208	993/1000
Runs	0.6848	996/1000	0.8786	987/1000	0.0766	993/1000	0.4769	988/1000
Longest Run	0.7116	993/1000	0.7597	991/1000	0.3685	992/1000	0.7439	992/1000
Rank	0.4064	988/1000	0.0747	989/1000	0.2675	987/1000	0.1113	993/1000
FFT	0.1001	984/1000	0.5121	987/1000	0.7538	990/1000	0.6745	981/1000
Non Overlapping Template	0.3941	985/1000	0.0106	993/1000	0.1202	989/1000	0.0184	988/1000
Overlapping Template	0.8724	987/1000	0.2635	985/1000	0.9987	992/1000	0.5022	989/1000
Universal	0.7849	991/1000	0.7217	992/1000	0.2033	987/1000	0.1075	986/1000
Approximate Entropy	0.6558	990/1000	0.6287	989/1000	0.7791	991/1000	0.9521	990/1000
Random Excursions	0.4854	608/616	0.2277	633/640	0.1346	571/577	0.0188	604/611
Random Excursions Variant	0.6285	609/616	0.9360	634/640	0.6663	574/577	0.4433	606/611
Serial	0.5831	992/1000	0.4807	992/1000	0.0681	995/1000	0.4208	992/1000
Linear Complexity	0.1037	994/1000	0.6910	991/1000	0.7636	989/1000	0.3669	986/1000

más bien el servidor recibe los mensajes de los publicantes y los reenvía a los suscriptores que estén suscritos a los tópicos correspondientes. El tópico es una cadena de texto que el servidor utiliza para enrutar los datos.

La segunda parte es el publicante, el cual envía mensajes sobre un tópico específico al servidor. Por último, la tercera parte es el suscriptor, el cual recibe los mensajes del servidor, siempre y cuando esté suscrito a un tópico. Para la configuración del protocolo, el publicante y el suscriptor necesitan conocer la dirección IP del servidor y estar configurados al mismo tópico.

El MQTT puede brindar confidencialidad e integridad por medio de protocolos estándar, como la seguridad de la capa de transporte (TLS). Sin embargo, también es posible implementar otros métodos de cifrado y de autenticación, como la función de hash PDP, para brindar seguridad en las comunicaciones.

El esquema de cifrado y autenticación se puede implementar con tres dispositivos Raspberry Pi 3: uno programado como transmisor en el publicante, otro como servidor, y el tercero como receptor en el suscriptor. En la Raspberry Pi, para utilizar el protocolo MQTT en Python, se pue-

de utilizar la biblioteca paho-mqtt. La figura 5.1 muestra el esquema entre los tres dispositivos Raspberry Pi. En este esquema la comunicación es unidireccional, es decir, un Raspberry Pi solo envía mensajes y otro solo los recibe, utilizando el tópico denominado “prueba”. No obstante, la comunicación puede hacerse bidireccional, al agregar un nuevo tópico y emplear los códigos de publicante y suscriptor en los dos dispositivos Raspberry Pi que no los tienen.

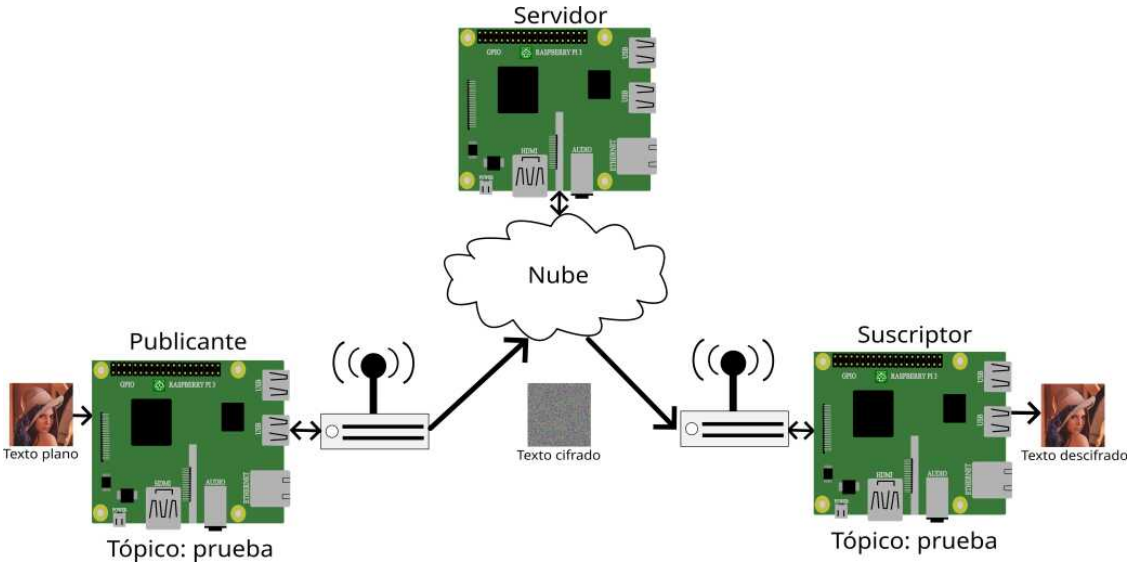


Figura 5.1: Esquema de comunicación segura empleando tres Raspberry Pi mediante el protocolo MQTT.

El funcionamiento de la etapa de comunicación de este sistema es simple y se ejecuta de forma secuencial. Primero, el publicante cifra el mensaje aplicando la operación XOR entre las cadenas binarias generadas por el PRNG y el mensaje. Luego, se ejecuta la función de hash para obtener la etiqueta de autenticación. Posteriormente, el publicante envía ambos datos al servidor, el cual los retransmite al suscriptor. Al final, el suscriptor descifra los datos empleando las mismas cadenas binarias y luego ejecuta la misma función de hash, para comparar ambas etiquetas y autenticar el mensaje. Si un atacante cambia un píxel, los valores de hash serán diferentes.

5.4.1. Etapa de cifrado y descifrado

Antes de efectuar la etapa de cifrado en el publicante, el mensaje total debe ser dividido en mensajes M formados al concatenar un grupo de bytes. En el caso de las imágenes, se debe ejecutar una función que transforme la imagen bidimensional en un arreglo unidimensional, y a

partir de este concatenar un grupo de píxeles para formar los mensajes M. El número de bytes concatenados dependerá de la longitud de una cadena binaria S.

La etapa de descifrado en el suscriptor debe generar las mismas cadenas binarias S. En el caso de imágenes, después de descifrar el mensaje y separar los píxeles, se debe transformar el arreglo unidimensional a bidimensional para recuperar la imagen.

5.4.2. Etapa de autenticación mediante la función de hash PDP

El algoritmo del producto pseudo punto, descrito en las ecuaciones 4.1 y 4.2, se puede realizar mediante el pseudocódigo 5.2 en un lenguaje de alto nivel. La función PDP_4D recibe como parámetros de entrada los mensajes M y las cadenas binarias S, las cuales deben tener la misma cantidad de elementos. El resultado de la función es un valor de hash con longitud de 224 bits.

El procedimiento comienza obteniendo la longitud de la matriz de mensajes M. Después, se almacenan las primeras 14 cadenas binarias S en un arreglo temporal TS para su uso en el algoritmo. Luego, las variables H se inicializan con cero y se carga el número primo.

A partir de la línea 11 comienza un ciclo for que, en cada iteración, calcula la variable H_1, mediante dos operaciones XOR, una multiplicación, una operación módulo y otra operación XOR con el valor anterior de H_1. Este mismo proceso se repite para las otras seis variables H, donde la diferencia entre cada una es el valor de la cadena binaria que se utiliza.

En la parte final de la estructura for, en cada repetición se actualiza el arreglo TS con dos valores de S y desplazando los elementos del arreglo TS. Al terminar el ciclo for, el valor de hash se obtiene concatenando las siete variables H.

```
1 #Entradas: mensajes (M), cadenas binarias (S)
2 #Salida: valor de hash (HV)
3 funcion PDP_4D(M,S):
4     # Obtener la longitud de la matriz M y agregar 14 más.
5     m = len(M) + 14
6     #Primeras 14 iteraciones de S en un arreglo temporal.
7     TS = S[0:13]
8     {H1, ... H7} = 0  # Iniciar en cero.
9     np = 2^31 − 1      # Número primo.
```

```
    for j=0 : (m-1) : j = j+2   #Ejecutar función hash
            MS1 = M[j] ⊕ TS[0]
            MS2 = M[j+1] ⊕ TS[1]
            PM1 = ((MS1 × MS2) mod np)
            H1 = PM1 ⊕ H1
            MS3 = M[j] ⊕ TS[2]
            MS4 = M[j+1] ⊕ TS[3]
            PM2 = ((MS3 × MS4) mod np)
            H2 = PM2 ⊕ H2
            ...
            MS13 = M[j] ⊕ TS[12]
            MS14 = M[j+1] ⊕ TS[13]
            PM7 = ((MS13 × MS14) mod np)
            H7 = PM7 ⊕ H7     # Variable H número 7.
            #Actualizar arreglo TS.
            TS[0] = TS[2]
            TS[1] = TS[3]
            TS[2] = TS[4]
            ...
            TS[11] = TS[13]
            TS[12] = S[j+14]
            TS[13] = S[j+15]
    #Concatenar las varaibles H.
    HV = H1 // H2 // H3 // H4 // H5 // H6 // H7
    return HV  #Devolver el valor de hash.
```

Pseudocódigo 5.2: Función de hash usando el producto pseudo punto.

5.5. Resultados del esquema de cifrado y autenticación en Raspberry

En esta sección se comparten los resultados del cifrado y autenticación de imágenes utilizando el protocolo MQTT para la comunicación entre dos dispositivos Raspberry pi. Las pruebas se llevaron a cabo transmitiendo algunas imágenes RGB y una a escala de grises con una resolución de 256x256 píxeles.

5.5.1. Cifrado y autenticación de imágenes RGB

En la primera prueba se envió la imagen RGB de unos pimientos en la cual no se modificó ningún píxel, dando como resultado el mismo valor de hash. En dicho procedimiento se emplearon las cadenas binarias del PRNG 4D. La figura 5.2 ilustra la imagen original, cifrada y descifrada, mientras que la tabla 5.2 muestra la etiqueta de autenticación resultante.

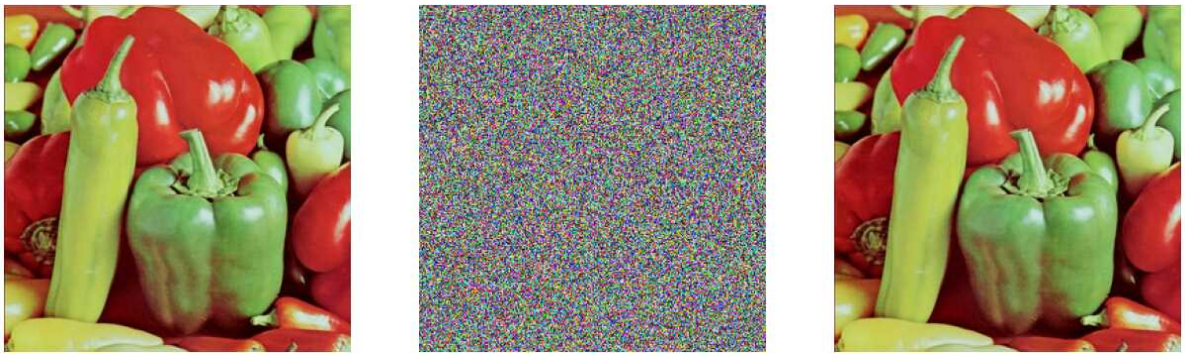


Figura 5.2: Cifrado/descifrado con la imagen RGB de los pimientos de 256×256. Original (izquierda), cifrada (centro) y descifrada (derecha).

Tabla 5.2: Etiqueta de autenticación obtenida con la imagen de los pimientos.

Dispositivo	Valor de hash de 224 bits
Transmisor/receptor	0x61F3E46D1FDBA0E622AC6BB94762942519134F0A7988EEE4103E827B

Al igual que con el sistema sintetizado en FPGA, para comprobar el funcionamiento del algoritmo de hash se efectuaron experimentos modificando píxeles en varias imágenes. Los siguientes resultados muestran los valores de hash obtenidos al modificar el color de un píxel en distintas

posiciones de una misma imagen, y considerando las mismas cadenas binarias. La figura 5.3 muestra tres imágenes de Lena modificadas con un píxel blanco y utilizando el PRNG 4D. Por su parte, la figura 5.4 presenta tres imágenes de un babuino, sometidas a un cambio de píxel en color negro y empleando las cadenas binarias del PRNG 3D. Las tablas 5.3 y 5.4 comparten las etiquetas de autenticación para las imágenes RGB de Lena y los babuinos, respectivamente.

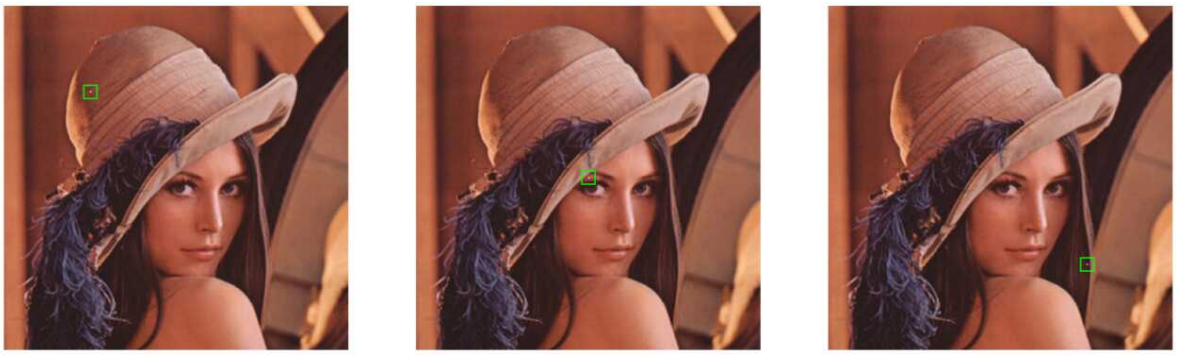


Figura 5.3: Imágenes RGB de Lena de 256×256, con un píxel blanco en la posición: [64,64] (izquierda), [128,128] (centro) y [192,192] (derecha).

Figura 5.4: Imágenes RGB del babuino de 256×256, con un píxel negro en la posición: [90,90] (izquierda), [130,130] (centro) y [160,160] (derecha).

Tabla 5.3: Etiquetas de autenticación de las imágenes RGB de Lena de la figura 5.3.

Imagen	Valor de hash de 224 bits
Original	0x1512E8590B96DE662E4D0FF24CD7584339A81E8B2CE8B6611C6C0AE4
[64,64]	0x0EA6CA16087EA7AB24319D7149E543222CD0F8305353A1A24A67F4F0
[128,128]	0x43125DC34C4B90B142B3E4DD7145D9E632ECD91B5013F92F3FEF0A5A
[192,192]	0x4BE9222568B7BDE1136F4C534C195C9E4213D9090CC79146368B1CD4

Tabla 5.4: Etiquetas de autenticación de las imágenes RGB del babuino de la figura 5.4.

Imagen	Valor de hash de 168 bits
Original	0xFEBFC72F0E887EF322156136826E8CCF4CB8584BFA
[90,90]	0x53D1E2F720164CB175F47FDF994FC2FA094FE6B954
[130,130]	0x515A9CCCE0ED1187AA9D7B588D9E02F26E76E1F64F
[160,160]	0x48B16B1E9346278601211568417 42F4BC3A925D0B4

5.5.2. Cifrado y autenticación de imágenes a escala de grises

En el caso de la prueba con una imagen a escala de grises, se utilizó la imagen de Lena. La figura 5.5 muestra tres imágenes modificadas con dos píxeles en color blanco. Por otro lado, la tabla 5.5 anexa los resultados de la función de hash que emplea las cadenas binarias del PRNG 2D.

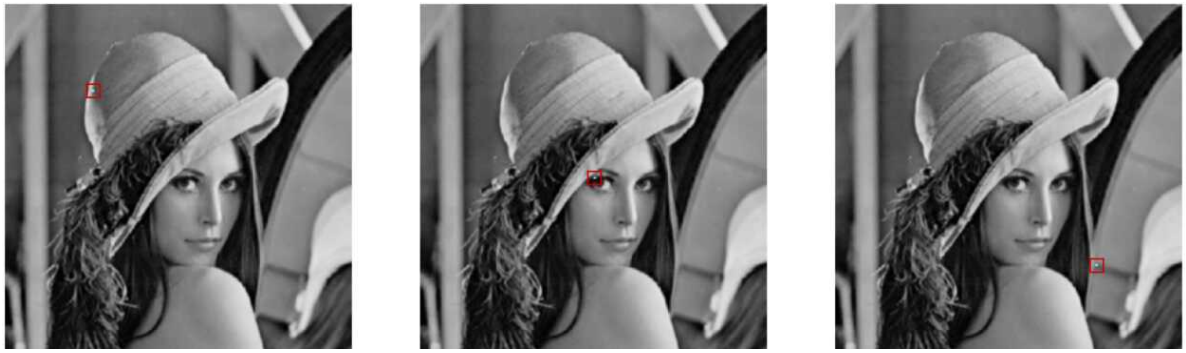


Figura 5.5: Imágenes a escala de grises de Lena de 256×256, con dos píxeles blancos cercanos a la posición: [64,64] (izquierda), [128,128] (centro) y [192,192] (derecha).

Tabla 5.5: Etiquetas de autenticación de las imágenes a escala de grises de Lena de la figura 5.5.

Imagen	Valor de hash de 112 bits
Original	0x376483B1DA2333A24E56F34A5187
[64,64]	0x0CAF5AFAB0BD30A3208B6C19752E
[128,128]	0x95F714BE9E251708FD72D6F283DF
[192,192]	0xB8FC5067FC709031A01C6285B86A

Capítulo 6

Conclusiones

Utilizar sistemas caóticos con un mayor número de variables de estado permite incrementar la longitud de las etiquetas de autenticación del algoritmo de función de hash del producto pseudo punto. No obstante, al aumentar el número de bits de las cadenas binarias del algoritmo, se deben considerar números primos más grandes.

La función de hash basada en el producto pseudo punto es un algoritmo con rápido procesamiento en FPGA. Las pruebas demuestran que el algoritmo produce valores de hash significativamente diferentes al modificar un solo dato del mensaje.

Los números pseudoaleatorios generados por los PRNGs, basados en sistemas caóticos, superaron las pruebas estadísticas NIST de aleatoriedad. Esto sugiere que los números generados no presentan patrones entre ellos y cuentan con la misma probabilidad de aparecer. Por lo tanto, se concluye que este tipo de sistemas caóticos descritos en aritmética de 64 bits son una buena opción para generar bits pseudoaleatorios.

En el caso de los sistemas caóticos de tiempo continuo, aunque el método numérico de Euler hacia adelante presenta el mayor error con respecto a otros métodos, es adecuado para la generación de números pseudoaleatorios en aquellos entornos que requieren un bajo consumo de recursos digitales.

Es esencial estudiar los parámetros y las características de un sistema caótico, como sus exponentes de Lyapunov y la dimensión Kaplan-Yorke, para evaluar la sensibilidad a las condiciones iniciales y la complejidad geométrica del atractor. Este análisis permite identificar

de forma preliminar si el sistema es candidato para el diseño de un PRNG.

El análisis de amplitudes máximas y mínimas de un oscilador caótico es indispensable para determinar la mejor distribución de bits de un formato de punto fijo, y así evitar desbordamientos en los bloques aritméticos o las variables. Se deben considerar condiciones iniciales con una parte decimal pequeña para garantizar la formación de las series temporales.

En la implementación de sistemas caóticos en FPGA, sobre todo en arquitecturas de 64 bits, siempre se debe optar por diseñar esquemas que utilicen la menor cantidad de bloques lógicos y multiplicadores embebidos. Para ello, se pueden crear diseños basados en bloques multiplicador acumulador, como en el caso del mapa 2D, así como multiplexar recursos y utilizar multiplicadores de una sola constante.

Como se observa en la tabla 3.5, el oscilador caótico 4D fue el que obtuvo el menor consumo de multiplicadores embebidos, así como la mayor frecuencia de operación y throughput en comparación con los sistemas 2D y 3D. Esto se debe a que el sistema de ecuaciones 4D solo requiere de un producto entre variables y una operación de valor absoluto.

Los PRNG sintetizados en una FPGA permiten corroborar que los diagramas de bloques digitales funcionan correctamente.

Los procedimientos de cifrado y autenticación en FPGA suceden de forma paralela, es decir, ambos algoritmos se ejecutan simultáneamente, lo cual permite lograr mayores velocidades de procesamiento.

La operación módulo es sintetizable en una FPGA de Altera Intel, sin embargo, reduce significativamente la frecuencia máxima de operación, tal y como se muestra en la tabla 4.9.

El esquema de comunicación segura implementado en Raspberry Pi mediante el protocolo MQTT, es fácil de programar y útil para sistemas basados en resúmenes de mensaje. En el proceso de comunicación, el servidor no puede descifrar la información debido a que desconoce la semilla utilizada.

6.1. Trabajo futuro

Agregar al sistema de cifrado y autenticación un método de sincronización de sistemas caóticos, o alguna técnica para compartir las claves. Se requiere una sincronización con error cero, ya que la función de hash producirá etiquetas de autenticación completamente diferentes ante un pequeño cambio en un dato.

Implementar en FPGA sistemas caóticos de orden fraccionario, los cuales presentan mayor complejidad y comportamientos más divergentes.

Utilizar más pruebas estadísticas, como TestU01 y DieHard, para garantizar la aleatoriedad de las cadenas binarias.

Llevar a cabo pruebas de ataques de predicción a los esquemas de comunicación segura.

Llevar a silicio los sistemas caóticos descritos en FPGA empleando software como Cadence. Si los sistemas requieren un alto consumo de área, se requiere reducir el número de bits de la aritmética.

Índice de figuras

Índice de tablas

Bibliografía

[1] E. Zeraoulia, *Models and applications of chaos theory in modern sciences*. CRC Press, 2011.

[2] H. G. Schuster and W. Just, *Deterministic chaos: an introduction*. John Wiley & Sons, 2006.

[3] D. Kaplan and L. Glass, *Understanding nonlinear dynamics*. Springer Science & Business Media, 2012.

[4] S. H. Strogatz, *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

[5] M. Kopp and A. Kopp, "A new 6d chaotic generator: Computer modelling and circuit design," *Int. J. Eng. Technol. Innov*, vol. 12, no. 4, pp. 288–307, 2022.

[6] J. H. Argyris, G. Faust, M. Haase, and R. Friedrich, *An exploration of dynamical systems and chaos: completely revised and enlarged second edition*. Springer, 2015.

[7] S. Jafari, J. Sprott, and F. Nazarimehr, "Recent new examples of hidden attractors," *The European Physical Journal Special Topics*, vol. 224, no. 8, pp. 1469–1476, 2015.

[8] E. Tlelo-Cuautle, A. D. Pano-Azucena, O. Guillén-Fernández, and A. Silva-Juárez, *Analog/-digital implementation of fractional order chaotic circuits and applications*. Springer, 2020.

[9] R. P. Canale and S. C. Chapra, *Numerical methods for engineers*. Mcgraw-hill Education-Europe, 2014.

[10] F. E. Cellier and E. Kofman, *Continuous system simulation*. Springer Science & Business Media, 2006.

[11] O. Guillén-Fernández, M. F. Moreno-López, and E. Tlelo-Cuautle, “Issues on applying one- and multi-step numerical methods to chaotic oscillators for fpga implementation,” *Mathematics*, vol. 9, no. 2, p. 151, 2021.

[12] O. Guillén-Fernández, E. Tlelo-Cuautle, G. Rodríguez-Gómez, L. G. de la Fraga, and R. Li, “Chapter 3 - on the fpga implementation of chaotic oscillators based on memristive circuits,” in *Mem-elements for Neuromorphic Circuits with Artificial Intelligence Applications* (C. Volos and V.-T. Pham, eds.), Advances in Nonlinear Dynamics and Chaos (ANDC), pp. 41–66, Academic Press, 2021.

[13] S. Li, Y. Wu, and X. Zhang, “Analysis and synchronization of a new hyperchaotic system with exponential term,” *Mathematics*, vol. 9, no. 24, p. 3281, 2021.

[14] X. Wang, L. Feng, R. Li, and F. Zhang, “A fast image encryption algorithm based on non-adjacent dynamically coupled map lattice model,” *Nonlinear Dynamics*, vol. 95, pp. 2797–2824, 2019.

[15] Y.-S. Hou, L.-L. Yi, G.-Q. Xia, and Z.-M. Wu, “Exploring high quality chaotic signal generation in a mutually delay coupled semiconductor lasers system,” *IEEE Photonics Journal*, vol. 9, no. 5, pp. 1–10, 2017.

[16] V. Buyadzhi, A. Glushkov, O. Y. Khetselius, A. Kuznetsova, A. Buyadzhi, G. Prepelitsa, and V. Ternovsky, “Nonlinear dynamics of laser systems with elements of a chaos: Advanced computational code,” in *Journal of Physics: Conference Series*, vol. 905, p. 012007, IOP Publishing, 2017.

[17] J. Awrejcewicz, A. V. Krysko, N. P. Erofeev, V. Dobriyan, M. A. Barulina, and V. A. Krysko, “Quantifying chaos by various computational methods. part 1: simple systems,” *Entropy*, vol. 20, no. 3, p. 175, 2018.

[18] D. Canavese, L. Mannella, L. Regano, and C. Basile, “Security at the edge for resource-limited iot devices,” *Sensors*, vol. 24, no. 2, p. 590, 2024.

[19] D. Johnston, *Random number generators—principles and practices: a guide for engineers and programmers*. Walter de Gruyter GmbH & Co KG, 2018.

[20] K. H. Tsoi, K. H. Leung, and P. H. W. Leong, “Compact fpga-based true and pseudo random number generators,” in *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.*, pp. 51–61, IEEE, 2003.

[21] E. Almaraz Luengo and J. Román Villaizán, “Cryptographically secured pseudo-random number generators: Analysis and testing with nist statistical test suite,” *Mathematics*, vol. 11, no. 23, p. 4812, 2023.

[22] L. Wang and H. Cheng, “Pseudo-random number generator based on logistic chaotic system,” *Entropy*, vol. 21, no. 10, p. 960, 2019.

[23] M. Garcia-Bosque, A. Pérez-Resa, C. Sánchez-Azqueta, C. Aldea, and S. Celma, “Chaos-based bitwise dynamical pseudorandom number generator on fpga,” *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 1, pp. 291–293, 2018.

[24] J. D. Díaz-Muñoz, I. Cruz-Vega, E. Tlelo-Cuautle, J. M. Ramirez Cortes, and J. de Jesús Rangel-Magdaleno, “Kalman observers in estimating the states of chaotic neurons for image encryption under mqtt for iot protocol,” *The European Physical Journal Special Topics*, pp. 1–18, 2021.

[25] A. Bensky, *Short-range wireless communication*. Newnes, 2019.

[26] A. S. Tanenbaum, *Redes de computadoras*. Pearson educación, 2003.

[27] S. Farahani, *ZigBee wireless networks and transceivers*. newnes, 2011.

[28] C. Fontaine and F. Galand, “A survey of homomorphic encryption for nonspecialists,” *EURASIP Journal on Information Security*, vol. 2007, pp. 1–10, 2007.

[29] B. Sklar, *Digital communications: fundamentals and applications*. Pearson, 2021.

[30] L. G. De la Fraga, C. Mancillas-López, and E. Tlelo-Cuautle, “Designing an authenticated hash function with a 2d chaotic map,” *Nonlinear Dynamics*, vol. 104, no. 4, pp. 4569–4580, 2021.

[31] E. Tlelo-Cuautle, L. De La Fraga, and J. Rangel-Magdaleno, “Engineering applications of fpgas,” *Engineering Applications of FP-GAs*, 2016.

[32] D. Molloy, *Exploring Raspberry Pi: interfacing to the real world with embedded Linux*. John Wiley & Sons, 2016.

[33] H. Nagashima, *Introduction to chaos: physics and mathematics of chaotic phenomena*. CRC Press, 2019.

[34] R. B. Naik and U. Singh, "A review on applications of chaotic maps in pseudo-random number generators and encryption," *Annals of Data Science*, vol. 11, no. 1, pp. 25–50, 2024.

[35] J. C. Sprott, "Automatic generation of strange attractors," *Computers & Graphics*, vol. 17, no. 3, pp. 325–332, 1993.

[36] E. N. Lorenz, "Deterministic nonperiodic flow," *Journal of atmospheric sciences*, vol. 20, no. 2, pp. 130–141, 1963.

[37] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[38] Z. Kotadai, C. Fischer, J. D. Rodríguez-Muñoz, E. Tlelo-Cuautle, and E. J. Yves, "Multistability and initial-offset boosting dynamics in a new 3d memristive chaotic system with fpga implementation," *The European Physical Journal Plus*, vol. 139, no. 1, p. 70, 2024.

[39] A. N. Pisarchik and U. Feudel, "Control of multistability," *Physics Reports*, vol. 540, no. 4, pp. 167–218, 2014.

[40] A. Sambas, M. Miroslav, S. Vaidyanathan, B. Ovilla-Martínez, E. Tlelo-Cuautle, A. A. Abd El-Latif, B. Abd-El-Atty, K. Benkouider, and T. Bonny, "A new hyperjerk system with a half line equilibrium: Multistability, period doubling reversals, antimonotonocity, electronic circuit, fpga design and an application to image encryption," *IEEE Access*, 2024.

[41] U. Meyer-Baese and U. Meyer-Baese, *Digital signal processing with field programmable gate arrays*, vol. 65. Springer, 2007.

[42] R. W. Mehler, *Digital integrated circuit design using verilog and systemverilog*. Elsevier, 2014.

[43] S. Harris and D. Harris, *Digital design and computer architecture*. Morgan Kaufmann, 2015.

[44] A. M. González-Zapata, E. Tlelo-Cuautle, I. Cruz-Vega, and W. D. León-Salas, “Synchronization of chaotic artificial neurons and its application to secure image transmission under mqtt for iot protocol,” *Nonlinear Dynamics*, vol. 104, no. 4, pp. 4581–4600, 2021.

[45] A. Kanso and N. Smaoui, “Logistic chaotic maps for binary numbers generations,” *Chaos, Solitons & Fractals*, vol. 40, no. 5, pp. 2557–2568, 2009.

[46] S. Haliuk, O. Krulikovskyi, D. Vovchuk, and F. Corinto, “Memristive structure-based chaotic system for prng,” *Symmetry*, vol. 14, no. 1, p. 68, 2022.

[47] M. Magfirawaty, A. A. Lestari, A. R. A. Nurwa, S. MT, and K. Ramli, “A novel discrete-time chaos-function-based random-number generator: design and variability analysis,” *Symmetry*, vol. 14, no. 10, p. 2122, 2022.

[48] F. Yu, L. Liu, S. Qian, L. Li, Y. Huang, C. Shi, S. Cai, X. Wu, S. Du, and Q. Wan, “Chaos-based application of a novel multistable 5d memristive hyperchaotic system with coexisting multiple attractors,” *Complexity*, vol. 2020, no. 1, p. 8034196, 2020.

[49] F. Zhang, J. Tang, Z. Zhang, Z. Huang, and T. Huang, “An improved absolute-cosine chaotification model and its simple application in prng,” *IEEE Access*, vol. 11, pp. 59346–59356, 2023.

[50] D. Murillo-Escobar, M. Á. Murillo-Escobar, C. Cruz-Hernández, A. Arellano-Delgado, and R. M. López-Gutiérrez, “Pseudorandom number generator based on novel 2d hénon-sine hyperchaotic map with microcontroller implementation,” *Nonlinear Dynamics*, vol. 111, no. 7, pp. 6773–6789, 2023.

[51] F. Yu, Z. Zhang, H. Shen, Y. Huang, S. Cai, J. Jin, and S. Du, “Design and fpga implementation of a pseudo-random number generator based on a hopfield neural network under electromagnetic radiation,” *Frontiers in Physics*, vol. 9, p. 690651, 2021.

[52] L. E. Bassham III, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, E. B. Barker, S. D. Leigh, M. Levenson, M. Vangel, D. L. Banks, *et al.*, “Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications,” 2010.

[53] J. M. Kizza, W. Kizza, and Wheeler, *Guide to computer network security*, vol. 8. Springer, 2013.

[54] T. Yang, "A survey of chaotic secure communication systems," *International journal of computational cognition*, vol. 2, no. 2, pp. 81–130, 2004.

[55] G. Chen, Y. Mao, and C. K. Chui, "A symmetric image encryption scheme based on 3d chaotic cat maps," *Chaos, Solitons & Fractals*, vol. 21, no. 3, pp. 749–761, 2004.

[56] A. Sambas, S. Vaidyanathan, X. Zhang, I. Koyuncu, T. Bonny, M. Tuna, M. Alçin, S. Zhang, I. M. Sulaiman, A. M. Awwal, *et al.*, "A novel 3d chaotic system with line equilibrium: multistability, integral sliding mode control, electronic circuit, fpga implementation and its image encryption," *IEEE Access*, vol. 10, pp. 68057–68074, 2022.

[57] G. Xu, Y. Shekofteh, A. Akgül, C. Li, and S. Panahi, "A new chaotic system with a self-excited attractor: entropy measurement, signal encryption, and parameter estimation," *Entropy*, vol. 20, no. 2, p. 86, 2018.

[58] E. İnce, B. Karakaya, and M. Türk, "Designing hardware for a robust high-speed cryptographic key generator based on multiple chaotic systems and its fpga implementation for real-time video encryption," *Multimedia Tools and Applications*, pp. 1–34, 2024.

[59] R. Sivaraman, S. Rajagopalan, and R. Amirtharajan, "Fpga based generic ro trng architecture for image confusion," *Multimedia Tools and Applications*, vol. 79, no. 19, pp. 13841–13868, 2020.

[60] H. Yang, K.-W. Wong, X. Liao, Y. Wang, and D. Yang, "One-way hash function construction based on chaotic map network," *Chaos, Solitons & Fractals*, vol. 41, no. 5, pp. 2566–2574, 2009.

[61] D. Xiao, X. Liao, and S. Deng, "One-way hash function construction based on the chaotic map with changeable-parameter," *Chaos, Solitons & Fractals*, vol. 24, no. 1, pp. 65–71, 2005.

[62] A. Kanso, H. Yahyaoui, and M. Almulla, "Keyed hash function based on a chaotic map," *Information Sciences*, vol. 186, no. 1, pp. 249–264, 2012.

[63] S. Windarta, S. Suryadi, K. Ramli, A. A. Lestari, W. Wildan, B. Pranggono, and R. W. Wardhani, “Two new lightweight cryptographic hash functions based on saturnin and beetle for the internet of things,” *IEEE Access*, 2023.

[64] X. Chai, X. Zheng, Z. Gan, D. Han, and Y. Chen, “An image encryption algorithm based on chaotic system and compressive sensing,” *Signal Processing*, vol. 148, pp. 124–144, 2018.

[65] A. Belazi, M. Talha, S. Kharbech, and W. Xiang, “Novel medical image encryption scheme based on chaos and dna encoding,” *IEEE access*, vol. 7, pp. 36667–36681, 2019.

[66] X. Wang and H. Yu, “How to break md5 and other hash functions,” in *Annual international conference on the theory and applications of cryptographic techniques*, pp. 19–35, Springer, 2005.

[67] X. Wang, Y. L. Yin, and H. Yu, “Finding collisions in the full sha-1,” in *Advances in Cryptology–CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005. Proceedings 25*, pp. 17–36, Springer, 2005.

[68] D. Chakraborty, C. Mancillas-López, and P. Sarkar, “Stes: A stream cipher based low cost scheme for securing stored data,” *IEEE Transactions on Computers*, vol. 64, no. 9, pp. 2691–2707, 2014.

[69] S. A. Khan, *Digital design of signal processing systems: a practical approach*. John Wiley & Sons, 2011.

[70] D. Parikh, *Raspberry Pi and MQTT Essentials: A complete guide to helping you build innovative full-scale prototype projects using Raspberry Pi and MQTT protocol*. Packt Publishing Ltd, 2022.