Diseño Embebido de Sistemas Caóticos de Orden Fraccional

por

MC Daniel Clemente López

Tesis sometida como requisito parcial para obtener el grado de

DOCTORADO EN CIENCIAS EN LA ESPECIALIDAD DE ELECTRÓNICA

en el

Instituto Nacional de Astrofísica, Óptica y Electrónica
Enero 2025
Tonantzintla, Puebla

Supervisada por:

Dr. José de Jesús Rangel Magdaleno
Investigador Titular INAOE
Dr. Jesús Manuel Muñoz Pacheco
Investigador Titular BUAP

Diseño Embebido de Sistemas Caóticos de Orden fraccional

Tesis de Doctorado

Por:

MC Daniel Clemente Lopez

Asesores:

Dr. José de Jesús Rangel Magdaleno (INAOE)

Dr. Jesús Manuel Muñoz Pacheco (BUAP)

Instituto Nacional de Astrofísica Óptica y Electrónica

Coordinación de Electrónica

Tonantzintla, Puebla. Enero 2025

A Dios, por ser mi guía y fortaleza en cada paso. A mi familia, por ser mi mayor motor y refugio. A mi padre Gustavo y a mi madre María, por su amor incondicional y sabiduría infinita. A mi hermano Gustavo, cuyo apoyo y confianza han sido fundamentales en este camino.

A mi hijo Daniel, quien es mi mayor inspiración y motivo para seguir adelante. Su presencia llena mi vida de alegría y propósito.

Agradecimientos

En primera instancia, agradezco al Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE) por brindarme el apoyo necesario durante mi estancia en el Doctorado en Ciencias en Electrónica.

Agradezco profundamente a mis asesores de tesis, el Dr. José de Jesús Rangel Magdaleno y el Dr. Jesús Manuel Muñoz Pacheco, quienes con su invaluable orientación, conocimientos y compromiso me ayudaron a culminar este trabajo.

Extiendo mi agradecimiento al jurado de tesis, compuesto por la Dra. Luz del Carmen Gómez Pavón, el Dr. José Hugo Barrón Zambrano, el Dr. Juan Manuel Ramírez Cortés, el Dr. Israel Cruz Vega y el Dr. Óscar Martínez Fuentes. Sus observaciones y comentarios enriquecieron significativamente el desarrollo de este trabajo.

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo brindado a través de la beca doctoral con CVU No. 859036, la cual fue fundamental para la realización de mis estudios.

Resumen

Este trabajo presenta el desarrollo y la implementación de sistemas caóticos de orden fraccional utilizando un método de descomposición por tramos lineales (PWL-DM) y una plataforma embebida basada en la arquitectura RISC-V. La investigación parte de una revisión detallada de los operadores fraccionales de clase F_1, con énfasis en los operadores de Caputo y Grünwald-Letnikov, que permiten modelar sistemas dinámicos de orden fraccional preservando propiedades críticas como la memoria y la herencia.

A lo largo del proyecto, se analizaron métodos de solución como FDA, GL y ADM, identificando sus limitaciones en aplicaciones prácticas. Esto condujo al desarrollo del método PWL-DM, diseñado para resolver sistemas PWL de orden fraccional que presentan atractores caóticos de múltiples enrollamientos. El método PWL-DM ofrece una estructura eficiente basada en operadores lineales y vectores constantes, lo que facilita su aplicación en sistemas caóticos complejos al permitir una representación precisa de los atractores caóticos mediante una partición estructurada del espacio fase.

El trabajo experimental incluyó la implementación de estos sistemas caóticos en una FPGA mediante la síntesis estándar, y posteriormente en un sistema embebido basado en RISC-V. La implementación en FPGA aprovechó las capacidades de cómputo paralelo y reconfiguración de hardware para alcanzar un rendimiento computacional elevado, aunque cada sistema caótico requirió de un diseño de arquitectura específica. En contraste, el sistema embebido basado en RISC-V se caracteriza por su flexibilidad, permitiendo manejar diferentes sistemas caóticos sin necesidad de modificar el hardware, siempre y cuando estos puedan ser representados en una palabra de 32 bits. Además, el sistema embebido demostró una eficiencia energética superior, optimizando el uso de recursos mediante un protocolo de comunicación interno.

Los resultados obtenidos confirman que el diseño embebido basado en RISC-V,

en combinación con el método PWL-DM, proporciona una solución escalable y adaptable para la implementación de sistemas caóticos de orden fraccional. Este enfoque presenta un balance entre eficiencia energética, rendimiento computacional y flexibilidad, lo cual lo convierte en una opción viable para aplicaciones en modelado de sistemas dinámicos y seguridad. Este trabajo sienta las bases para futuras investigaciones en el campo de sistemas caóticos y sus aplicaciones en plataformas embebidas, contribuyendo al avance de la ingeniería en el análisis y control de sistemas dinámicos complejos.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Capítulo 1
Introducción

La teoría del **caos** es una rama de las matemáticas que se dedica al estudio de comportamientos complejos que emergen en sistemas dinámicos no lineales, conocidos como sistemas caóticos [1]. Esta teoría se enfoca en fenómenos como la extrema sensibilidad a las condiciones iniciales y a los parámetros del sistema, transitividad, y la existencia de un conjunto denso de órbitas periódicas [2]. Los sistemas caóticos se pueden encontrar en casi todos los campos de la ciencia e ingeniería, como la astrofísica [3], mecánica [4], economía [5], comunicaciones seguras [6], criptografía [7], robótica [8] y control [9].

En la literatura se reporta dos formas para clasificar a los sistemas caóticos en cuanto a su cuenca de atracción [10]. Por un lado, los sistemas caóticos con atractores autoexcitados son aquellos cuya cuenca de atracción intersecta con un equilibrio inestable. Por otro lado, los sistemas caóticos con atractores ocultos tienen una cuenca de atracción que no intersecta con ningún vecindario abierto de los equilibrios. Desde que se encontró el primer atractor oculto en el circuito clásico de Chua [11], se ha prestado mucha más atención a este nuevo tipo de atractor. En primer lugar, la dinámica de los atractores ocultos es altamente compleja de analizar, ya que pueden llevar a comportamientos inesperados. Por ejemplo, fenómenos complejos como la "Multiestabilidadz la "Multiestabilidad Extrema"son propensos a aparecer en sistemas dinámicos con atractores ocultos [12]. Los fenómenos de multiestabilidad se refieren a cuando dos o más atractores diferentes coexisten bajo los mismos parámetros, mientras que la multiestabilidad extrema ocurre cuando coexisten infinitos atractores. Debido a que estos fenómenos dependen de las condiciones iniciales para cambiar de un atractor a otro totalmente diferente, se pueden usar como una fuente adicional de aleatoriedad para aplicaciones potenciales. Por ejemplo, podrían mejorar el rendimiento en criptografía [13] y comunicaciones seguras [14].

En relación con el estudio de sistemas caóticos, el cálculo fraccional es un tema que ha adquirido un gran interés en tiempos recientes. Este se refiere a la generalización de integrales y derivadas de orden entero a órdenes arbitrarios. El cálculo fraccional es un área de las matemáticas propuesta hace más de 300 años, pero solo recientemente ha emergido como un tema de alto interés en investigación debido a sus características superiores para el modelado de sistemas complejos. Por ejemplo, las definiciones del cálculo fraccional poseen propiedades de memoria que permiten una descripción más precisa de varios fenómenos físicos en comparación con sus contrapartes de orden entero. (por ejemplo, véase: procesos biológicos [15], economía [16], mecánica [17], electrónica [18], fenómenos sociales [19]). De esta manera, los sistemas caóticos de orden fraccional (SCOF) pueden considerarse como la capa de interacción que unifica las ramas matemáticas del caos y el cálculo de orden fraccional. A conocimiento de los autores, el sistema de Chua propuesto por Hartley et al. [20], puede considerarse el primer manuscrito que reportó un sistema caótico con derivadas fraccionales. Desde entonces, un número creciente de sistemas caóticos de orden fraccional se han reportado en la literatura, incluidos los sistemas clásicos de Lorenz [21], Chen [22], y Duffing [23].

Es importante remarcar que la dimensión efectiva en sistemas dinámicos tridimensionales de orden fraccional dada por $\sum_{i=1}^{n=3} q_i$, con n y $q_i \in (0, 1]$ siendo el número de variables pseudo-estado y el orden fraccional, respectivamente, puede ser menor que tres. Sin embargo, sorprendentemente, pueden generar comportamientos caóticos. Así, el teorema de Poincaré-Bendixson no se aplica a esos sistemas. Debido a que la evolución dinámica en los sistemas de orden fraccional contiene efectos de memoria de todos los eventos pasados, se ha utilizado para modelar fenómenos caóticos en física e ingeniería. Aquí, las derivadas de orden fraccional sirven como un grado de libertad adicional para mejorar las aplicaciones de ingeniería basadas en el caos. Por ejemplo, en criptografía y comunicaciones seguras, los sistemas caóticos de orden fraccional pueden generar un espacio de claves más grande que los de orden entero [24, 25]. Además, se han propuesto para algoritmos de cifrado de imágenes [26, 27], algoritmos de cifrado de sonido [28, 29], algoritmos de autenticación para Internet de las Cosas (IoT) [30, 31]. De la misma manera, las aplicaciones de robótica basadas en el caos, como la planificación de rutas [32, 33], pueden beneficiarse del grado de libertad adicional para optimizar la cobertura de una determinada área.

Las técnicas convencionales de sistemas clásicos se han generalizado a sistemas de orden fraccional para aplicaciones en control del caos y sincronización del caos [34]. Se ha encontrado que los sistemas caóticos de orden fraccional pueden generar varias familias de atractores ocultos y auto-excitados en sus modelos conmensurados e inconmensurados, y también se observan fenómenos de multi-estabilidad [35, 36]. Una derivada de orden fraccional es capaz de proporcionar un mejor control en la propagación de atractores caóticos que su contraparte de orden entero [37]. Aprovechando esta ventaja, los sistemas caóticos basados en funciones PWL (Lineales por Segmento por sus siglas en ingles Piece Wise Linear), son de gran interés debido a su capacidad para generar atractores caóticos de múltiples enrollamientos, los cuales pueden desplegarse en diferentes direcciones 1D, 2D y 3D y por lo tanto, pueden tener comportamientos dinámicos más complejos.

Sin embargo, la mayoría de estos trabajos permanecen en el campo teórico o solo se implementan en simuladores de circuitos, en lugar de realizaciones de hardware físico que son necesarias para validar las aplicaciones prácticas de los modelos matemáticos. Por lo tanto, uno de los principales desafíos en los sistemas caóticos de orden fraccional es su diseño de circuitos, que se puede realizar mediante dispositivos electrónicos analógicos o digitales, de modo que se puedan utilizar para varias aplicaciones, como cifrado, memristores, robótica, generadores de números aleatorios verdaderos, bioingeniería, sistemas de control, filtros, y muchos más.

1.0.1. Antecedentes

Para aplicaciones prácticas, la derivada de orden fraccional puede implementarse utilizando circuitos analógicos o digitales [38]. Por un lado, las implementaciones analógicas se realizan empleando circuitos analógicos elaborados a partir de amplificadores operacionales, resistores, capacitores, bobinas y diodos, entre otros componentes [39]. Esta técnica resulta en una gran complejidad debido a los efectos no lineales intrínsecos en los componentes pasivos y activos [40]. Esta complejidad se hace evidente en varias publicaciones, como [41] y [42]. En cuanto a los sistemas caóticos, los circuitos analógicos están sujetos a perturbaciones externas como ruido y calor que pueden destruir el comportamiento caótico [43]. Otras limitaciones se presentan

en [44], donde la implementación analógica de un sistema caótico se realiza con fines de comunicaciones seguras. A pesar de que esta implementación conserva las propiedades caóticas teóricas para aplicaciones de seguridad, la respuesta en frecuencia de esta implementación es difícil de mejorar y, por lo tanto, el rendimiento en la transmisión de datos se ve afectado.

Por otro lado, la complejidad de las implementaciones digitales radica en la formulación de algoritmos elegantes para resolver los modelos matemáticos que representan la derivada FO [45], que luego deben implementarse en plataformas digitales. A diferencia de las implementaciones analógicas, las digitales poseen características atractivas adicionales, como robustez, portabilidad, mayor flexibilidad para variar parámetros y condiciones iniciales del sistema [30, 46, 47]. Además, dado su enfoque intrínsecamente digital, ofrecen una mayor adaptabilidad para integrarse con aplicaciones emergentes, como el Internet de las Cosas (IoT). Por lo tanto, el interés de este trabajo se centra en la implementación digital.

Las preocupaciones actuales de las implementaciones digitales de SCOF incluyen el tipo de familia del sistema, la velocidad de operación, interfaces de comunicación y consumo de hardware y energía. Según la literatura, existen dos enfoques principales para la implementación digital de sistemas caóticos de orden fraccional [48]:

- **Implementaciones embebidas**: Las implementaciones integradas están orientadas a la computación secuencial en procesadores de señal digital (DSP) o plataformas digitales basadas en Advanced RISC Machine (ARM), que son ejecuciones desarrolladas con lenguajes de programación de alto nivel como C o Python. Por ejemplo, las plataformas basadas en ARM presentan características atractivas como bajo consumo de energía, comunicación de hardware, costo relativamente bajo y amplio dominio sobre los dispositivos electrónicos de consumo, por lo que se utilizan en varias publicaciones [49–51]. Sin embargo, en aplicaciones en tiempo real, el análisis de rendimiento inevitablemente se degrada debido a los efectos de latencia durante la operación del hardware. Lo mismo se puede extender para las implementaciones basadas en DSP. Una técnica de mejora para esta tecnología puede obtenerse mediante el uso de procesamiento paralelo, aunque esta técnica introduce una complejidad considerable. Esto se debe a la necesidad de diseñar algoritmos que distribuyan las tareas equitativamente entre los procesadores y manejen la sincronización y comunicación entre ellos.

- **Implementaciones no embebidas**: Estas se enfocan en la computación paralela sobre hardware reconfigurable como tarjetas FPGAs. Estas implementaciones se diseñan utilizando un lenguaje de descripción de hardware como Verilog o VHDL para elaborar los circuitos digitales, los cuales pueden ser reutilizados en diferentes implementaciones. La tecnología FPGA es adecuada para implementar sistemas complejos con alto rendimiento computacional. Sin embargo, el diseño de sistemas caóticos de orden fraccional en FPGAs introduce una complejidad adicional, ya que cada tipo de sistema puede requerir una arquitectura de hardware específica. A pesar de esto, en en los ultimos años se han reportado varios trabajos sobre implementaciones de FPGAs de estos sistemas [52–56].

Para comprender mejor ambos enfoques, en la Tabla 1.1 se describen las principales características cada uno. En la Tabla 1.2, se presenta el número de publicaciones científicas relacionadas con las implementaciones embebidas y no embebidas de sistemas caóticos de orden fraccional para los principales chips digitales. Se puede observar que el número de trabajos ha mantenido un interés constante en los últimos años.

Tabla 1.2: Número de publicaciones por año relacionadas con las implementaciones embebidas y no embebidas de SCOF.

Año	ARM	DSP	FPGA	Total
2015	0	2	0	2
2016	0	1	0	1
2017	1	1	10	12
2018	0	3	10	13
2019	1	3	11	15
2020	3	9	8	20
2021	1	12	1	14

1.0.2. Motivación y Justificación

Uno de los principales desafíos para implementar SCOF en plataformas digitales consiste en seleccionar métodos de resolución adecuados, como los algoritmos numéricos de Adams-Bashforth-Moulton (ABM) y Grünwald-Letnikov (GL), el Método de Descomposición de Adomian (ADM), los métodos de Aproximación en el Dominio de la Frecuencia (FDA). Además, es crucial reducir la complejidad de estos métodos para que los recursos de hardware disponibles puedan manejar todos los estados anteriores

Tabla 1.1: Características de diseño embebido y no embebido.

	Enfoque embebido	**Enfoque no embebido**
Arquitectura de hardware	Configuración predeterminada basada en arquitecturas ARM o Harvard. [57, 58].	Una extensa colección de celdas lógicas y bloques de memoria para una configuración de una arquitectura flexible [59].
Configuración	Opciones de lenguajes de programación como *C* o *Python* [60, 61].	Configuración basada en Lenguajes de Descripción de Hardware como Verilog o VHDL [62, 63].
Procesamiento en paralelo	Paralelismo limitado debido a la baja cantidad de núcleos de procesador disponibles. [64].	Alto paralelismo mediante la configuración flexible de las celdas lógicas, que pueden ser programadas para ejecutar múltiples operaciones simultáneamente [59].
Interfaces de comunicación	Alta conectividad con otros dispositivos electrónicos gracias a las interfaces de comunicación integradas, como UART, SPI, y video [65].	Requiere la integración de protocolos de comunicación específicos en el diseño, ya que no vienen predefinidos en el FPGA [66].
Resumen	Diseño optimizado para programación versátil y sencilla, pero con opciones limitadas para el procesamiento paralelo.	Flexibilidad para diseñar soluciones digitales con procesamiento en paralelo para lograr altas velocidades de ejecución, aunque su configuración es compleja.

y calcular la solución actual, es decir, manejar los efectos de la memoria del sistema de orden fraccional.

Es importante destacar que las implementaciones digitales de sistemas de orden fraccional son relativamente un tema nuevo y en desarrollo. Podemos rastrear uno de los primeros trabajos en [67], donde se aborda la implementación digital de un controlador de orden fraccional. Varios años después, las publicaciones sobre sistemas caóticos con derivadas de orden fraccional se volvieron extensas y populares. A pesar de los numerosos trabajos publicados en los últimos años, las implementaciones digitales de sistemas caóticos de orden fraccional todavía enfrentan muchos problemas prácticos, como el costo, la complejidad y el rendimiento, que limitan las aplicaciones potenciales al interés académico.

Motivado por el resumen anterior, este trabajo está orientado a la implementación digital de sistemas caóticos de orden fraccional. La complejidad tanto en la teoría matemática como en la configuración del hardware ha hecho que esta tesis tenga dos objetivos. La primera parte investiga los enfoques de las derivadas de orden fraccional y su potencial implementación en plataformas digitales. Con los resultados obtenidos, queremos generar una nueva metodología para el proceso de discretización de sistemas caóticos de orden fraccional. La segunda parte se ocupa de la configuración del hardware. Con las tecnologías emergentes de SoC, creemos que la comunicación de procesadores basados en RISC y plataformas FPGA ofrece más flexibilidad en aplicaciones en tiempo real. Por lo tanto, la segunda parte de este trabajo tiene como objetivo desarrollar una arquitectura de hardware optimizada para sistemas de orden fraccional en una plataforma FPGA.

1.0.3. Planteamiento del Problema

En términos generales, la implementación digital de SCOF requiere formulaciones teóricas que proporcionen aproximaciones efectivas de derivadas fraccionales y la elección de una arquitectura de hardware eficiente que ofrezca tanto precisión computacional como costos razonables. Por lo tanto, los principales desafíos a resolver en el trabajo propuesto son los siguientes:

- **Amplio rango de comportamiento caótico**. Para validar los esfuerzos de un nuevo diseño de arquitectura embebida, los sistemas propuestos a implementar deben presentar un comportamiento dinámico dentro de un amplio rango de sus parámetros y de la derivada de orden fraccional.

- **Baja complejidad en la configuración del sistema**. Además de la flexibilidad para variar las propiedades dinámicas de un sistema dado, la arquitectura propuesta requiere una herramienta que permita configurar e implementar los sistemas caóticos de manera eficiente y utilizando un alto nivel de abstraccón.

- **Rendimiento Computacional**. Los resultados de la implementación digital deben presentar un alto rendimiento considerando un bajo uso de recursos de hardware para que estos puedan ser usados en aplicaciones practicas de ingeniería.

1.0.4. Hipótesis

La formulación de algoritmos eficientes basados en la teoría matemática para resolver sistemas de orden fraccional (FO) permitiría realizar un diseño embebido específico basado en FPGA para la implementación de sistemas caóticos de orden fraccional. Dicho diseño podría proporcionar un buen equilibrio entre los recursos de hardware y el rendimiento, además de ser capaz de representar las ricas dinámicas de los sistemas caóticos de orden fraccional.

1.0.5. Objetivo General

El objetivo de este trabajo es investigar estrategias para la implementación embebida de sistemas caóticos de orden fraccional (SCOF) y, con los resultados obtenidos, formular algoritmos eficientes para un diseño embebido basado en FPGA.

1.0.6. Objetivos Particulares

Los objetivos particulares de este trabajo se enumeran a continuación:

1 . **Estudio de Derivadas de Orden Fraccional**. Realizar un análisis de las definiciones y propiedades de las derivadas de orden fraccional, así como los métodos existentes para resolver sistemas de orden fraccional.

2 . **Caracterización de sistemas caóticos**. Caracterizar la dinámica de sistemas caóticos de orden fraccional. Esto se refiere al estudio del comportamiento caótico a través de análisis de puntos de equilibrio, calculo de exponentes de Lyapunov y elaboración diagramas de bifurcación.

3 . **Comparación de métodos para resolver sistemas de orden fraccional**. Realizar una comparación de los métodos reportados en aplicaciones practicas de sistemas caóticos de orden fraccional. El objetivo es identificar ventajas, desventajas y áreas de oportunidad para proponer una metodología de diseño de sistemas caóticos de orden fraccional en sistemas embebidos.

4 . **Diseño de un sistema embebido**. Basado en la metodología propuesta en el punto anterior, proponer un diseño embebido para ser implementado en plataformas FPGA. Este diseño debe permitir la implementación de algoritmos

de sistemas caóticos de orden fraccional, asegurando un procesamiento lo suficientemente rápido para la implementación en aplicaciones prácticas de estos sistemas.

5 . **Implementación de sistemas caóticos de orden fraccional**. Demostrar la metodología propuesta mediante la implementación de sistemas caóticos de orden fraccional.

1.0.7. Metodología

Esta sección introduce las directrices que orientaron el trabajo de investigación, alineándose con los objetivos planteados, considerando que el objetivo principal es la implementación de sistemas caóticos de orden fraccional en un diseño embebido basado en FPGA, la metodología de investigación se dividio de la siguiente manera:

1.- **Análisis de las propiedades de derivadas de orden fraccional**

- **Definiciones Fundamentales**: Esta parte se dedica al estudio de las definiciones formales de las derivadas de orden fraccional, como la derivada de Riemann-Liouville, la derivada de Caputo y la derivada de Grünwald-Letnikov. Cada una de estas definiciones proporciona diferentes perspectivas y herramientas matemáticas para modelar fenómenos con memoria y dependencias a largo plazo.
- **Propiedades Matemáticas**: Investigación de las propiedades clave de las derivadas fraccionales, incluyendo linealidad, conmutatividad con respecto a la integración, y comportamiento en el dominio de la frecuencia. Estas propiedades son esenciales para entender cómo las derivadas fraccionales pueden ser aplicadas a sistemas caóticos.
- **Comparación de Métodos**: Comparación de los métodos basados en estas derivadas para resolver sistemas fraccionales, evaluando su precisión, estabilidad y eficiencia computacional. Este análisis es fundamental para seleccionar el método más adecuado para diferentes tipos de aplicaciones.

2.- **Análisis de Sistemas Caóticos de Orden Fraccional** :
Esta etapa se centra en la caracterización y análisis de sistemas caóticos de orden fraccional. Este análisis incluye:

- **Caracterización de la Dinámica Caótica**: Realizar un análisis de la estabilidad de los puntos de equilibrio y determinar el orden fraccional mínimo necesario para la existencia de caos.
- **Implementación en Software**: Uso de herramientas de simulación numérica para validar el comportamiento caótico mediante retratos de fase, cálculo de exponentes de Lyapunov.

3.- **Análisis de métodos para aplicaciones prácticas de sistemas caóticos de orden fraccional**: Este paso consiste en el análisis y comparación de diferentes métodos numéricos para resolver sistemas de orden fraccional en aplicaciones prácticas. El objetivo es determinar que método se adecua en distintas aplicaciones para IoT. Este análisis es esencial para seleccionar el método más eficaz según el contexto y las necesidades específicas de implementación.
Basado en los análisis realizados en los puntos uno y dos de esta metodología, esta sección se enfoca en los siguientes métodos:

- **Método de Grünwald-Letnikov (GL)**: Seleccionado por su precisión y capacidad para manejar discretizaciones directas de las derivadas fraccionales, lo que facilita su implementación en sistemas embebidos.
- **Método de Descomposición de Adomian (ADM)**: Seleccionado por su eficiencia computacional y capacidad para descomponer problemas no lineales complejos en subproblemas más simples, mejorando así el rendimiento en implementaciones embebidas.

4.- **Implementación y validación de un diseño embebido Basado en FPGA** La etapa final se centra en la implementación de un procesador basado en RISC-V, optimizado para manejar sistemas caóticos de orden fraccional. Se propone utilizar un procesador basado en RISC-V debido a su capacidad para ofrecer un alto rendimiento computacional con un bajo consumo de energía, aprovechando la reconfigurabilidad del FPGA y la flexibilidad de la arquitectura RISC-V, para personalizar la solución en función de las necesidades específicas del sistema.

1.1. Alcance de la Metodología y Resultados Preliminares

La metodología propuesta aborda los sistemas caóticos de orden fraccional, considerando tanto su análisis teórico como su implementación práctica en sistemas embebidos. Esta metodología ha sido validada mediante estudios comparativos y resultados experimentales los cuales se han reportado en diversas publicaciones, de las cuales destacan las siguientes:

1.- En la referencia [68], se realizó un análisis detallado de las ventajas y desventajas de las implementaciones embebidas y no embebidas de sistemas caóticos en FPGA. Este estudio demostró que las soluciones embebidas presentan un menor costo y menor consumo de energía, a pesar de un rendimiento ligeramente inferior en términos de velocidad. Estas implementaciones pueden beneficiarse del uso de hardware reconfigurable. Lo que oriento la decisión de utilizar un enfoque embebido optimizado para aplicaciones en tiempo real.

2.- En la referencia [69] se implementó un sistema PWL basado en el método de Grünwald-Letnikov, el cual fue aplicado en la encriptación de imágenes en tiempo real. Este estudio validó la viabilidad de los sistemas caóticos de orden fraccional en aplicaciones de comunicación en tiempo real y destacó la complejidad computacional del método de Grünwald-Letnikov.

En este trabajo se han seleccionado los sistemas caóticos PWL (Piecewise Linear o lineales por segmento) como caso de estudio, estos fueron elegidos debido a sus características únicas, como una estructura matemática simple y eficiente para generar comportamientos dinámicos complejos con múltiples atractores. Estos sistemas demostraron ser ideales para aplicaciones en criptografía tiempo real, según los resultados obtenidos en las publicaciones realizadas.

1.- En la referencia [70], se propuso un sistema basado en funciones signo, caracterizado por su simplicidad matemática y eficiencia computacional. Este sistema fue utilizado para aplicaciones de seguridad en datos médicos, particularmente en el cifrado ligero de señales biométricas en sistemas IoT de salud. Los resultados experimentales mostraron que este enfoque garantiza un nivel alto de

seguridad mientras mantiene un bajo consumo de recursos, haciendo viable su implementación en dispositivos con capacidades de hardware limitadas.

2.- De igual forma, en la referencia [69], se estudió la complejidad dinámica de los sistemas caóticos basados en funciones PWL, validada por la cantidad máxima de enrollamientos generados y el amplio régimen caótico con respecto a los parámetros y orden de la derivada fraccional, que estos sistemas pueden producir. Esta característica resulta en un espacio de llaves extremadamente grande, haciéndolo prácticamente invulnerable a ataques de fuerza bruta.

En ambos escenarios, se evaluó la resiliencia de los esquemas de seguridad basados en sistemas PWL de orden fraccional frente a diversos tipos de ataques, tales como, ataques de fuerza bruta, análisis estadísticos y la recuperación de datos en condiciones de ruido y pérdida parcial de información durante la transmisión de datos.

Las publicaciones mencionadas reflejan el amplio análisis realizado durante el desarrollo de la investigación, validando la relevancia y el potencial de los sistemas caóticos de orden fraccional, particularmente aquellos basados en funciones PWL, para aplicaciones prácticas en criptografía y seguridad de datos. Este documento se enfoca en los resultados clave relacionados con el alcance de los métodos para resolver sistemas caóticos de orden fraccional y el desarrollo de un sistema embebido basado en RISC-V

1.2. Organización del documento

Organización del Documento

El presente trabajo está estructurado en seis capítulos y un apéndice, organizados de la siguiente manera:

- **Capítulo 1: Introducción**
  Este capítulo presenta el contexto general del trabajo, incluyendo la motivación, los objetivos generales y específicos, así como la relevancia del estudio. También se introducen los sistemas caóticos de orden fraccional y su importancia en aplicaciones prácticas.

- **Capítulo 2: Análisis de Derivadas de Orden Fraccional**
  Se exponen los fundamentos teóricos de las derivadas de orden fraccional, abordando sus principales definiciones y propiedades. Además, se analiza la aplicabilidad de diferentes métodos numéricos para resolver sistemas fraccionales, considerando sus ventajas y limitaciones.

- **Capítulo 3: Formulación del Método de Descomposición PWL para Sistemas Caóticos de Orden Fraccional**
  En este capítulo se desarrolla y formula el método de descomposición PWL para la solución de sistemas caóticos de orden fraccional basados en funciones PWL, obteniendo soluciones que permiten analizar su naturaleza caótica.

- **Capítulo 4: Desarrollo Experimental**
  Este capítulo describe el diseño e implementación de un sistema embebido basado en RISC-V para la resolución y validación de sistemas caóticos de orden fraccional.

- **Capítulo 5: Análisis de Resultados**
  Se presentan y discuten los resultados obtenidos durante el desarrollo experimental, incluyendo métricas de rendimiento como throughput, uso de recursos de hardware y consumo de energía.

- **Capítulo 6: Conclusiones**
  En este capítulo se resumen los hallazgos principales de la investigación y se presentan las conclusiones generales del trabajo.

- **Apéndice**
  En el apéndice se incluyen las publicaciones realizadas durante el desarrollo de la investigación, las cuales respaldan y amplían los resultados presentados en este documento.

Capítulo 2

Análisis de Derivadas de Orden Fraccional

2.1. Derivadas de Orden Fraccional

Según la clasificación propuesta por Teodoro, Machado y Oliveira [71], los operadores de orden fraccional se pueden clasificar en cuatro clases, de las cuales la clase F_1 incluye los operadores fraccionales clásicos definidos por Riemann-Liouville, Caputo y Grünwald-Letnikov. Estos operadores fraccionales de la clase F_1 preservan las propiedades de memoria y son no locales.

Consideramos únicamente las definiciones de operadores fraccionales de la clase F_1, ya que cumplen con el criterio de Ross [72] y con el Criterio de Sentido Amplio propuesto por Ortigueira y Machado [73], para evidenciar operadores fraccionales genuinos. En las siguientes subsecciones, introducimos los conceptos matemáticos básicos para la derivada de Caputo y su transformada de Laplace, la derivada de Grünwald-Letnikov, la integral de orden fraccional de Riemann-Liouville y sus propiedades, y la definición de sistemas de ecuaciones diferenciales fraccionales.

Definición 1 *La definición de Caputo para una derivada de orden fraccional se presenta de la siguiente manera [74, 75]:*

$$ {}^{C}D^{q}_{t_0} f(t) := \begin{cases} \dfrac{1}{\Gamma(m-q)} \displaystyle\int_{t_0}^{t} \dfrac{f^{(m)}(\tau)}{(t-\tau)^{q+1-m}} d\tau & m-1 < q \leq m, \\ \dfrac{d^m}{dt^m} f(t) & q = m, \end{cases} \tag{2.1.1} $$

donde $q \in \mathbb{R}^+$ es el orden fraccional, $m \in \mathbb{N}$ es el primer entero mayor que q, y $\Gamma(\cdot)$

es la función gamma de Euler. Sea $m = 1$ *y* $\alpha = -q$*, entonces*

$$\Gamma(m-q) = \Gamma(\alpha+1) = \alpha\Gamma(\alpha), \quad \Gamma(\alpha) = \int_0^\infty t^{\alpha-1}e^{-t}dt. \tag{2.1.2}$$

Proposición 1 *Sea* $q > 0$ *y* $m = \lceil q \rceil$*. La transformada de Laplace de la derivada de Caputo de una función* $f(t)$ *es [76]:*

$$\mathcal{L}\{{}^C D_{t_0}^q f(t)\} = s^q F(s) - \sum_{k=0}^{m-1} s^{q-1-k} f^{(k)}(t_0), \tag{2.1.3}$$

con $m-1 < q \leq m$*. Para condiciones iniciales nulas, la transformada de Laplace de la derivada de Caputo se reduce a* $\mathcal{L}\,{}^C D_{t_0}^q f(t) = s^q F(s)$*.*

La definición de Grünwald-Letnikov para una derivada de orden fraccional se presenta de la siguiente manera [77, 78]:

Definición 2 *Sea* $q > 0$ *y* $t \in \mathbb{R}$*. La derivada fraccional de Grünwald-Letnikov de orden* q *es:*

$${}^{GL} D_t^q f(t) = \lim_{h\to 0} \frac{1}{h^q} \sum_{j=0}^{\infty} (-1)^j \binom{q}{j} f(t-jh), \tag{2.1.4}$$

donde los coeficientes binomiales $\binom{q}{j}$ *pueden calcularse de la siguiente manera:*

$$\binom{q}{0} = 1, \quad \binom{q}{j} = \frac{\Gamma(q+1)}{\Gamma(j+1)\Gamma(q-j+1)}, \quad j = 1, 2, ... \tag{2.1.5}$$

En la práctica, se utiliza la derivada fraccional truncada de Grünwald-Letnikov para simplificar el cálculo en funciones definidas solo a partir de t_0 [79].

Definición 3 Derivada fraccional truncada de Grünwald-Letnikov. *Sea* $q > 0$ *y* $t_0 \in \mathbb{R}$*. La derivada fraccional truncada de Grünwald-Letnikov de orden* q *es:*

$${}^{GL} D_{t_0}^q f(t) = \lim_{h\to 0} \frac{1}{h^q} \sum_{j=0}^{N} (-1)^j \binom{q}{j} f(t-jh), \tag{2.1.6}$$

with $t > t_0$ *and* $N = \lceil \frac{t-t_0}{h} \rceil$*.*

La definición de Riemann-Liouville para un operador de integral fraccional $J_{t_0}^q$ se establece como [80]:

Definición 4 Operador de integral fraccional de Riemann-Liouville. *Sea $q \in \mathbb{R}^+$. Entonces, el operador de integral fraccional $J_{t_0}^q$ para una función $f(t)$ definida en el espacio de Lebesgue $L^1([t_0, T])$ se define como:*

$$J_{t_0}^q f(t) := \frac{1}{\Gamma(q)} \int_{t_0}^{t} (t-\tau)^{q-1} f(\tau) d\tau, \; t \in [t_0, T], \tag{2.1.7}$$

donde $\Gamma(\cdot)$ es la función gamma de Euler, y con $J_{t_0}^0 f(t) = f(t)$.

El operador $J_{t_0}^q$ tiene las siguientes propiedades [74, 76, 80]:

- (a) *Conmutativa y Aditiva*

$$J_{t_0}^q J_{t_0}^p f(t) \Longleftrightarrow J_{t_0}^p J_{t_0}^q f(t) = J_{t_0}^{p+q} f(t), \; q \geq 0, \; p \geq 0; \tag{2.1.8}$$

- (b) *Dada una función de potencia $f(t) = (t-t_0)^\gamma, \; \gamma > -1$*

$$J_{t_0}^q (t-t_0)^\gamma = \frac{\Gamma(\gamma+1)}{\Gamma(\gamma+1+q)} (t-t_0)^{\gamma+q}; \tag{2.1.9}$$

- (c) *Dada una constante $C \in \mathbb{R}$*

$$J_{t_0}^q C = \frac{C}{\Gamma(q+1)} (t-t_0)^q. \tag{2.1.10}$$

- *La operación $J_{t_0}^q \; {}^C D_{t_0}^q f(t)$, regresa:*

$$\begin{aligned} J_{t_0}^q \, {}^C D_{t_0}^q f(t) &= J_{t_0}^q (J_{t_0}^{m-q} \, {}^C D_{t_0}^m) f(t), \\ &= J_{t_0}^m \, {}^C D_{t_0}^m f(t) \\ &= f(t) - \sum_{k=0}^{m-1} f^k(t_0^+) \frac{(t-t_0)^k}{k!}. \end{aligned} \tag{2.1.11}$$

- *La operación ${}^C D_{t_0}^q \; J_{t_0}^q f(t)$, regresa: $f(t)$.*

Definición 5 Sistema de Ecuaciones Diferenciales Fraccionales. *Un Problema de Valor Inicial (IVP) para una ecuación diferencial fraccional en el sentido de Caputo se puede definir como:*

$$\begin{cases} {}^C D_{t_0}^q x(t) = f(t, x(t)) \\ x(t_0) = x_0, \; x'(t_0) = x_0^{(1)}, \ldots, \; x^{(m-1)}(t_0) = x_0^{(m-1)}, \end{cases} \tag{2.1.12}$$

con $x_0, ; x_0^{(1)}, \ldots, ; x_0^{(m-1)}$ *como las condiciones iniciales en* t_0*. De esta manera, los sistemas fraccionales basados en Caputo tienen un significado físico más claro porque el problema de valor inicial depende únicamente de derivadas de orden entero. Además, el sistema* (2.1.12) *se puede expresar en forma matricial:*

$$ {}^{C}D_{t_0}^{q}x(t) = Ax(t) + Bu(t), \tag{2.1.13} $$

donde $x(t) \in \mathbb{R}^n$*,* $u(t) \in \mathbb{R}^m$*,* $A \in \mathbb{R}^{n\times n}$*,* $B \in \mathbb{R}^{n\times m}$*,* ${}^{C}t_0D^qtx(t) = [{}^{C}t_0D^qtx_1(t), \ldots, ; {}^{C}\, t_0D^qtx_n(t)]^T$*,* $(n, m) \in \mathbb{N}$*,* $q \in (0, 1]$*. Cuando el sistema (13) es autónomo, se puede reescribir como* ${}_{t_0}^{C}D_t^q x(t) = Ax(t)$*, con* $x(0) = x_0$*,* $0 < q < 1$*, y* $x \in \mathbb{R}^n$*.*

Definición 6 Estabilidad en sistemas de orden fraccional. *Considere el sistema de orden fraccional*

$$ D_{t_0}^{q}x(t) = f\big(x(t)\big), \tag{2.1.14} $$

donde $(0 < q < 1)$ *es la derivada de orden fraccional,* $x(t) = [x_1(t), x_2(t), ..., x_n(t)]^T \in \mathbb{R}^n$*, y* $f\big(x(t)\big) = [f_1\big(x(t)\big), f_2\big(x(t)\big), ..., f_n\big(x(t)\big)]^T$*. Sea eq. (2.1.14) un sistema de orden fraccional conmensurado* $q = q_1 = q_2, = \cdots q_n$*, con un punto de equilibrio hiperbólico en* $\bar{x} = [\bar{x}_1, \bar{x}_2, ..., \bar{x}_n]$*, se puede estimar que el comportamiento caótico puede emerger si el sistema satisface la siguiente condición [81]:*

$$ q > \frac{2}{\pi} \arg\big(\|\lambda_i\big(J(\bar{x})\big)\|\big). \tag{2.1.15} $$

Donde $J(\bar{x}) \in \mathbb{R}^{n\times n}$ *es la matriz jacobiana del sistema (2.1.14) evaluada en los puntos de equilibrio* $\bar{x}$ *y los valores propios correspondientes satisfacen* $\ni \lambda_i$ $(i = 1, 2, \cdots, n)$ *con* $Re(\lambda_i) > 0$*,* $Im(\lambda_i) \neq 0$*.*

2.2. Métodos de solución para sistemas de orden fraccional

El paso crucial para implementar sistemas caóticos fraccionales utilizando hardware digital, tanto embebido como no embebido, es aplicar métodos de solución confiables, precisos y eficientes. Entre estos métodos se incluyen los métodos de aproximación (FDA), los métodos numéricos (ABM) y los métodos semianalíticos (ADM).

No obstante, al resolver ecuaciones diferenciales de orden fraccional mediante alguno de estos métodos, pueden presentarse varios problemas [51, 82–86]. Las principales puntos a tomar en cuenta son:

- Cómo manejar adecuadamente la memoria persistente, ya que las propiedades de memoria prolongada de las derivadas fraccionales pueden hacer que el cálculo sea extremadamente lento y computacionalmente costoso.
- En otros casos, los métodos de solución presentan una baja precisión para emular el orden fraccional requerido.
- Además, para ciertos métodos de solución, sus coeficientes pueden necesitar ajustes antes de la implementación para coincidir con los formatos aritméticos disponibles en el hardware digital.

Por consiguiente, en las siguientes secciones, se analizan un conjunto de métodos de solución comúnmente utilizados en la literatura.

2.2.1. Aproximación en el dominio de la frecuencia

La Aproximación en el Dominio de la Frecuencia (FDA, por sus siglas en inglés) es una técnica conocida y ampliamente utilizada para diseñar controladores de orden fraccional [87, 88]. Además, ofrece una buena opción para resolver sistemas de orden fraccional [89, 90]. El método funciona de la siguiente manera. Primero, la derivada fraccionaria se representa en el dominio de la frecuencia mediante la transformada de Laplace. Luego, se reemplaza, en un ancho de banda dado, por su aproximación de orden entero. La función de transferencia en el dominio s obtenida puede usarse para la implementación analógica de sistemas caóticos de orden fraccional. Para el alcance de nuestro trabajo, la implementación basada en circuitos digitales se obtiene mediante ciertas manipulaciones de procesamiento de señales, donde se puede derivar una función de transferencia en el dominio z [91], o incluso un sistema dinámico extendido de orden entero utilizando la fórmula de Faà di Bruno [49]. A continuación, se presentan las formulaciones teóricas del método FDA.

Consideremos la transformada de Laplace dada en (2.1.3) cuando se asumen condiciones iniciales nulas, entonces la solución para el sistema (2.1.12) se da por

$$x(s) = \frac{1}{s^q}\mathcal{L}f\big(x(t)\big), \tag{2.2.1}$$

donde el operador $1/s^q$ es un integrador de orden fraccional de Laplace y, desde el punto de vista de la realización de circuitos, representa la función de transferencia de un capacitor fraccional teórico. Desafortunadamente, ese elemento de circuito no puede ser implementado de manera directa ya que no existen dispositivos físicos comerciales. Por lo tanto, para aplicaciones prácticas, es necesario aproximar $1/s^q$ mediante una función de transferencia de orden entero de la siguiente manera:

$$\frac{1}{s^q} \approx H(s) = \frac{b_\alpha s^\alpha + ... + b_1 s + b_0}{s^p + ... + a_1 s + a_0}, \tag{2.2.2}$$

con $(\alpha, p) \in \mathbb{N}$ y $(a_i, b_i) \in \mathbb{R}$. Numerosas guías de diseño para la ubicación de polos y ceros en la ecuación (2.2.2) han sido reportadas en la literatura [87, 92–94]. Por ejemplo, el enfoque de Charef [92] funciona de la siguiente manera. Dada una discrepancia de Δ dB para el ancho de banda de frecuencia $[\omega_c, \omega_{max}]$ (donde ω_c es la frecuencia de corte y ω_{max} es la frecuencia máxima $\omega_{max} \gg \omega_c$), la aproximación de $1/s^q$ se realiza mediante un Polo de Potencia Fraccionaria (FPP, por sus siglas en inglés) de la siguiente manera:

$$\frac{1}{s^q} \approx \frac{1}{\left(1 + \frac{s}{\omega_c}\right)^q} \approx H(s) = \frac{\Pi_{k=0}^{N-1}\left(1 + \frac{s}{z_k}\right)}{\Pi_{k=0}^{N}\left(1 + \frac{s}{p_k}\right)}, \tag{2.2.3}$$

donde $a' = 10^{[\Delta/10(1-q)]}$, $b' = 10^{[\Delta/10q]}$ y $N = 1 + Int\left(\frac{log\left(\frac{\omega_{max}}{p_0}\right)}{log(a'b')}\right)$.

Luego, los ceros z_k y los polos p_k se pueden calcular de manera recursiva mediante

$$\begin{aligned} p_0 &= w_c 10^{\Delta/20q}, \\ z_{k-1} &= p_{k-1} 10^{[\Delta/10(1-q)]}, \\ p_k &= z_{k-1} 10^{[\Delta/10q]}. \end{aligned} \tag{2.2.4}$$

La interpretación gráfica del método de Charef se ilustra en la Figura 2.1. Como se puede observar, la respuesta en frecuencia del FPP se caracteriza por una pendiente de $-20q$ dB/década. Como resultado, la expresión (2.2.3) se basa en la aproximación de la pendiente de $-20q$ dB/década con una serie de líneas en zigzag conectadas entre sí con pendientes alternadas de 0 y -20 dB/década.

En otro escenario, sustituyamos la aproximación de $1/s^q$ dada por la ecuación (2.2.3) en la expresión (2.2.2), obteniendo así:

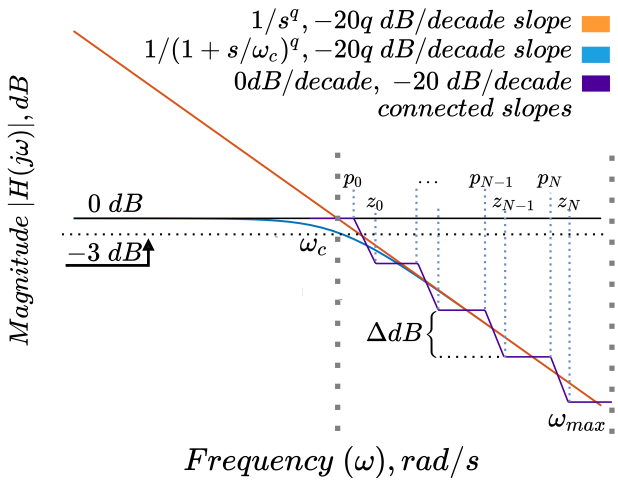


Figura 2.1: Comparación de diagramas de Bode entre $1/s^q$ y $H(s)$ utilizando el método de Charef.

$$\begin{aligned} s^p x(s) + ... + a_1 s x(s) + a_0 x(s) = b_\alpha s^\alpha \mathcal{L}\{f(x(t))\} + ... \\ + b_1 s \mathcal{L}\{f(x(t))\} + b_0 \mathcal{L}\{f(x(t))\}, \end{aligned} \tag{2.2.5}$$

y al tomar la transformada inversa de Laplace $\mathcal{L}^{-1}\cdot$ de la ecuación (2.2.5), se obtiene el siguiente sistema de orden entero con estados aumentados que aproxima al sistema original de orden fraccional:

$$\begin{aligned} \frac{d^p}{dt^p} x(t) + ... + a_1 \frac{d}{dt} x(t) + a_0 x(t) = b_\alpha \frac{d^\alpha}{dt^\alpha} f(x(t)) + ... \\ + b_1 \frac{d}{dt} f(x(t)) + b_0 f(x(t)). \end{aligned} \tag{2.2.6}$$

Finalmente, la expresión (2.2.6) ahora puede ser resuelta utilizando métodos numéricos tradicionales, como el método de Euler o los métodos de Runge Kutta de orden superior [95].

2.2.1.1. Alcance y limitaciones de FDA para la implementación de sistemas caóticos de orden fraccional.

FDA es un método útil para el diseño de controladores de orden fraccional. Además, podría ser un buen candidato para la implementación analógica/digital de

sistemas caóticos de orden fraccional. Sin embargo, es necesario discutir dos importantes desventajas.

- **Pérdida de propiedades de memoria.** Como se ha discutido anteriormente, las derivadas de orden fraccional de la clase F_1 son operadores globales que implican una fuerte dependencia de la memoria, mientras que las derivadas de orden entero son operadores locales. Por lo tanto, ninguno de los métodos de FDA para operadores de orden fraccional preserva las propiedades de memoria [85]. Esta desventaja lleva a la pérdida de parte de la dinámica del sistema original de orden fraccional, por ejemplo, en la Ref. [96] se afirma claramente que un sistema caótico de orden fraccional con cuatro enrollamientos decae a un atractor de un solo enrollamiento cuando se aplica el enfoque FDA.

- **Inconsistencias en los puntos de equilibrio**. La ganancia en estado estacionario de la expresión (2.2.2) requiere ser lo suficientemente grande, para que los puntos de equilibrio en el sistema aumentado (2.2.6) sean consistentes con los del sistema original [85]. De lo contrario, el número, la ubicación y la estabilidad de los puntos de equilibrio aproximados pueden diferir de los originales. Por ejemplo, Tavazoei y Haeri [97] observaron que un sistema de orden fraccional elegido conduce a un comportamiento caótico mientras que es estable bajo la aproximación FDA. Y viceversa, cuando el sistema de orden fraccional tiende a tener un comportamiento estable, es caótico para el método FDA.

Como resultado, el enfoque FDA no debería utilizarse en sistemas caóticos representados con derivadas fraccionales de clase F_1, ya que los comportamientos caóticos observados son cuestionables. Otras desventajas están relacionadas con la complejidad del enfoque. La discrepancia y el rango de frecuencia en la ecuación (2.2.3) deben elegirse cuidadosamente para evitar grandes errores entre los resultados aproximados y los analíticos [98]. Además, representar una derivada fraccionaria con un orden q de varios dígitos después del punto decimal (por ejemplo, $q = 0.9236$) requiere una función de transferencia de orden superior en comparación con aquellas con solo un dígito, es decir, $q = 0.9$. Por lo tanto, el método FDA se vuelve muy complicado para ser implementado en hardware. Además, se requiere un rediseño completo para modificar el valor de orden fraccional.

2.2.2. Algoritmo numérico de Adams-Bashforth-Moulton

El algoritmo de Adams-Bashforth-Moulton (ABM) es un esquema numérico para resolver ecuaciones diferenciales de orden fraccional [99,100]. Es un método predictor-corrector porque la solución aproximada, obtenida con la fórmula de cuadratura trapezoidal, implica calcular una aproximación preliminar utilizando la regla del rectángulo [99]. Cabe destacar que el algoritmo ABM preserva los efectos de memoria de las derivadas de orden fraccional. Por lo tanto, puede considerarse un método formal para estudiar sistemas de orden fraccional. En la literatura, el enfoque ABM ha sido ampliamente utilizado para calcular las soluciones numéricas de sistemas caóticos de orden fraccional [35,101,102]. El método ABM se describe a continuación.

Sea $[0, T_M]$ un intervalo de solución para el sistema (2.1.12), entonces una trayectoria de interés en la cuadrícula uniforme $t_N = Nh \mid N = 0, 1, ..., M$ con algún entero M, $h = (T_M)/M$, y $q \in (0, 1]$, se puede calcular como [86,99]:

$$\begin{aligned} x(t_{N+1}) = x(t_0) + \frac{h^q}{\Gamma(q+2)} f(x^P(t_{N+1})) \\ + \sum_{j=0}^{N} \frac{h^q \gamma_{j,N+1}}{\Gamma(q+2)} f(x(t_j)), \end{aligned} \tag{2.2.7}$$

donde $x^P(t_{N+1})$ es el esquema predictor calculado por:

$$x^P(t_{N+1}) = x(t_0) + \sum_{j=0}^{N} \frac{h^q \beta_{j,N+1}}{q\Gamma(q)} f(x(t_j)), \tag{2.2.8}$$

y los coeficientes $\gamma_{j,N+1}$, $\beta_{j,N+1}$ son:

$$\gamma_{j,N+1} = \begin{cases} N^{q+1} - (N-q)(N+1)^q, & j = 0, \\ (N-j-2)^{q+1} + (N-j)^{q+1} - 2(N-j+1)^{q+1}, & 1 \le j \le N, \\ 1, & j = N+1, \end{cases} \tag{2.2.9}$$

$$\beta_{j,N+1} = (N-j+1)^q - (N-j)^q. \tag{2.2.10}$$

En el campo del cálculo fraccional, el método de ABM es ampliamente reconocido como una herramienta valiosa para calcular la solución de ecuaciones diferenciales fraccionales, incluidas aquellas que llevan a un comportamiento caótico, debido a su excelente aproximación de la solución verdadera [82,99].

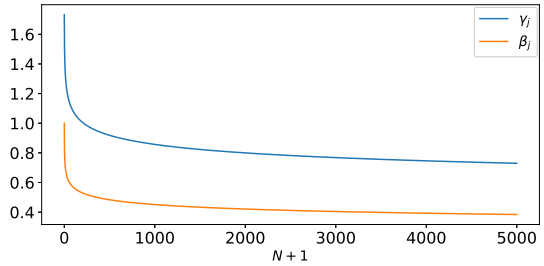


Figura 2.2: Evolution of the weights $\gamma_{j,N+1}$ and $\beta_{j,N+1}$ in the ABM algorithm with $q = 0.9$ and $N = 5000$.

2.2.2.1. Alcance y limitaciones del método ABM para la implementación de sistemas caóticos de orden fraccional.

Por otro lado, el algoritmo ABM presenta los siguientes problemas relacionados con la complejidad computacional:

- **Cálculo de memoria infinita.** Debido a que el método ABM utiliza toda la memoria para calcular una solución, es confiable para resolver sistemas de orden fraccional. Por otro lado, esta dependencia de la memoria prolongada provoca una computación extremadamente lenta desde el punto de vista computacional. Las implicaciones de la dependencia de la memoria para un caso hipotético con orden fraccional $q = 0.9$ se ilustran en la Figura 2.2. Observamos que, aunque los pesos $\gamma_{j,N+1}$ y $\beta_{j,N+1}$ disminuyen a medida que N aumenta, nunca desaparecen por completo. Por lo tanto, la complejidad resultante del método ABM es enorme, ya que requiere conocer toda la historia de la evolución del sistema, por ejemplo, la solución $x(t_{N+1})$ requiere todos los valores anteriores $x(t_0), x(t_1), ..., x(t_N)$.

- **Alto costo computacional para dispositivos digitales con recursos limitados.** Como se mencionó en el punto anterior, el cálculo de memoria infinita induce un alto costo computacional. Como se puede ver en la ecuación (2.2.8), el índice inferior de la suma comienza en $j = 0$, y por lo tanto, a medida que N tiende a infinito, los datos acumulados aumentan considerablemente. Dado que el diseño empotrado y no empotrado se basa en hardware digital con recursos limitados, estos no pueden procesar una gran cantidad de datos. Esto es un grave inconveniente porque las FPGAs, procesadores ARMs, DSPs, etc., tienen un número determinado de bloques de memoria de acceso aleatorio y de memoria

de solo lectura que están físicamente disponibles para almacenar los datos de la solución resultante. Como resultado, implementar sistemas caóticos de orden fraccional utilizando el algoritmo ABM no es factible.

En la literatura se han propuesto algunos esfuerzos para mitigar los problemas mencionados. En [103] se utilizó una técnica de computación paralela para mejorar la eficiencia del algoritmo ABM; sin embargo, esa estrategia estaba planeada para ser ejecutada en entornos de computación de alto rendimiento. Otro enfoque se presenta en [104], donde se trunca la longitud de la memoria. Aunque el costo computacional es menor que la versión no truncada (2.2.7), una pérdida de precisión es inevitable. De hecho, esta versión truncada del algoritmo ABM todavía requiere al menos la mitad del conocimiento del sistema de memoria para lograr la convergencia, es decir, para calcular la solución $x(t_{N+1})$, se requieren los valores de $x(t_0), x(t_1), ..., x(t_{[N/2]})$. Por lo tanto, la truncación de la memoria propuesta no elimina el costo computacional para valores grandes de N. En resumen, el método numérico ABM puede aplicarse con éxito en el estudio de sistemas caóticos de orden fraccional. Desafortunadamente, el ABM no es elegible para aplicaciones de ingeniería reales debido a su alto costo computacional.

2.2.3. Algoritmo de Grünwald-Letnikov

La aproximación de Grünwald-Letnikov (GL) es otro esquema numérico bien conocido para resolver ecuaciones diferenciales fraccionales. La solución del conjunto de ecuaciones diferenciales fraccionales en el sentido de Caputo (2.1.12) puede aproximarse mediante el enfoque GL utilizando diferencias finitas de orden fraccional, como se mostró en [78, 105]. También se considera una generalización de los métodos de Euler para ecuaciones diferenciales ordinarias [105]. La idea básica es que el operador derivativo de Grünwald-Letnikov (2.1.4), al seleccionar un valor inicial homogéneo, es equivalente a las definiciones de Caputo y Riemann-Liouville [74, 76, 80].

Consideremos la definición de un sistema de ecuaciones diferenciales fraccionales dado en (2.1.12), entonces una solución numérica general en los puntos $kh, ; k = 1, 2, \ldots$ puede expresarse como:

$$x(t_k) = f\big(x(t_{k-1})\big)h^q - \sum_{j=1}^{k} C_j^q x(t_{k-j}), \tag{2.2.11}$$

donde h es el paso de integración y C_j^q son los coeficientes binomiales expresados en (2.1.5), que se calculan según la relación recursiva:

$$C_0^q = 1, \;\; C_j^q = \big(1 - \frac{(1+q)}{j}\big) C_{j-1}^q, \; j = 1, 2, ..., k. \tag{2.2.12}$$

Las propiedades de memoria prolongada de la derivada fraccionaria están presentes en la definición (2.2.11), donde el índice superior del componente de suma denota la dependencia de todos los valores anteriores de $x(t_k)$ [106, 107]. En la figura 2.3, se ilustra el comportamiento de los factores de amortiguación (coeficientes binomiales) para las contribuciones de memoria en un caso hipotético con orden fraccional $q = 0.9$. Se puede observar que cuando j tiende a k, los valores numéricos de C_j^q disminuyen gradualmente pero no alcanzan el valor cero. De la misma manera que el método ABM

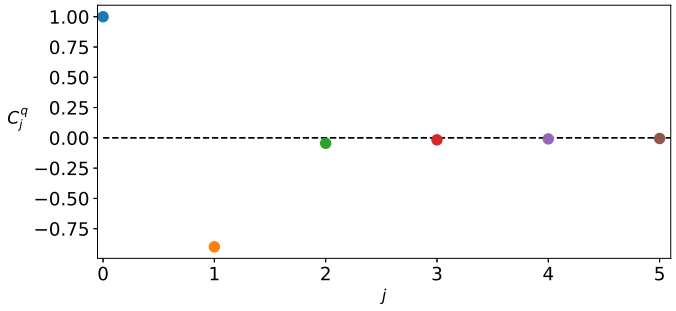


Figura 2.3: Valores numéricos de los coeficientes binomiales (2.2.12) en $q = 0.9$ y $j = 0, 1, \ldots 5$.

(2.2.7), la aproximación GL implica tiempos de simulación relativamente largos. Para reducir su alto costo computacional, el índice inferior en la suma de la ecuación (2.2.11) se modifica por $j = v$, donde:

$$v = \begin{cases} 1, & k \leq [\frac{L_m}{h}] \\ [k - \frac{L_m}{h}], & k > [\frac{L_m}{h}] \end{cases} \tag{2.2.13}$$

con $[\cdot]$ denotando la parte entera más cercana y L_m es la longitud de memoria del sistema subyacente. Este procedimiento se conoce como el principio de memoria corta (SMP) de la aproximación GL.

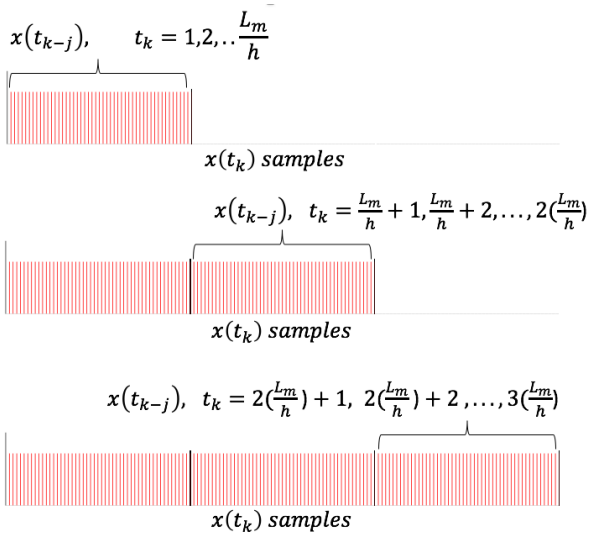


Figura 2.4: Una gestión típica de datos para resolver sistemas caóticos de orden fraccional con la aproximación SMP-GL. Los corchetes denotan la longitud de la memoria utilizada para calcular la solución $x(t_k)$ con las muestras anteriores $x(t_{k-j})$ almacenadas durante todo el tiempo de simulación.

2.2.3.1. Alcance y limitaciones del enfoque numérico GL para la implementación de sistemas caóticos de orden fraccional.

La aproximación GL (2.2.11) se ha aplicado satisfactoriamente en el análisis e implementación de sistemas caóticos de orden fraccional [107–111] debido a que incorpora el SMP, lo cual reduce los datos generados para obtener una solución $x(t_k)$. Es fundamental una cuidadosa selección de la longitud de la memoria para evitar problemas numéricos en la representación de la derivada fraccional. Los principales problemas del esquema GL están relacionados con la precisión y la complejidad en la implementación física con dispositivos digitales:

- **Longitud de la memoria.** Dado que se utiliza el SMP para minimizar el costo computacional en la implementación de sistemas fraccionales, la aproximación SMP-GL sufre una pérdida de precisión. Es importante destacar que una cuidadosa selección de la longitud de la memoria es esencial para evitar problemas numéricos en la representación válida de una derivada fraccional. Incluso se necesita una longitud de memoria relativamente grande para mantener el comportamiento dinámico verdadero [53].

- **Complejidad en la gestión de datos.** Desde un punto de vista computacional, el método GL con SMP no elimina las soluciones precedentes $x(t_{k-1})$, que

ya han sido utilizadas. Por lo tanto, el vector de los datos de solución $x(t_k)$ puede aumentar indefinidamente a medida que el número de iteraciones tiende a infinito, como se ilustra en la Figura 2.4 [51].

Varios trabajos se han reportado para reducir los problemas mencionados de la aproximación GL. En [112], la precisión del SMP se mejoró utilizando un paso de integración más pequeño. En [113], se analizó la diferencia entre el enfoque GL con longitud de memoria completa y el SMP, y luego se mejoró el error de aproximación utilizando el SMP al restar la mediana de la función del sistema. Otro enfoque fue reportado en [84], donde se mejoró la precisión en un conjunto de datos pequeño mediante técnicas de preprocesamiento de señales basadas en un lenguaje de programación de alto nivel. Finalmente, los autores en [51] propusieron un esquema numérico eficiente del método SMP-GL en el que solo se conservan los datos esenciales relacionados con la longitud de memoria elegida. Al mismo tiempo, se eliminan las muestras de datos antiguas.

En conclusión, la aplicación de la aproximación SMP-GL logra un buen equilibrio entre precisión y costo computacional. Además, requiere menos esfuerzo computacional en comparación con el algoritmo ABM, por lo que es una buena alternativa como solucionador numérico para implementar sistemas caóticos de orden fraccional en dispositivos digitales con limitaciones. Cabe señalar que la estimación de la longitud de la memoria no es una tarea trivial y se necesitan estudios futuros al respecto.

2.2.4. Método de Descomposición de Adomian

El Método de Descomposición de Adomian (ADM) se considera un método semi-analítico para resolver ecuaciones diferenciales de orden fraccional (FO) [114–117]. La idea básica de este método es reemplazar las no linealidades del sistema dinámico fraccional con una expansión en series, y luego esas series se reformulan en los llamados "polinomios de Adomian" [118]. El ADM se describe a continuación.

Consideremos el sistema no lineal fraccional (2.1.12). Posteriormente, los elementos lineales y no lineales del sistema subyacente pueden dividirse de la siguiente manera:

$$ {}^{C}D^{q}_{t_0}x(t) = Lx(t) + Nx(t) + g(t), \tag{2.2.14}$$

donde L y N denotan las partes lineales y no lineales respectivamente, y $g(t) \in \mathbb{R}^n$ es una función analítica conocida. Al aplicar el operador $J^{q}_{t_0}$ a ambos lados de la

ecuación (2.2.14), obtenemos:

$$x(t) = J^q_{t_0} Lx(t) + J^q_{t_0} Nx(t) + J^q_{t_0} g(t) + \eta, \tag{2.2.15}$$

donde $\eta = \sum_{k=0}^{m-1} f^{(k)}(t_0^+)\frac{(t_1-t_0)^k}{k!}$. Cuando $m = 1$, el vector η contiene las condiciones iniciales $f^{(0)}(t_0^+) = x^(t_0^+)$ (aquí, $+$ significa causalidad, esto implica que $f(t)$ es una función causal de t, es decir, $f(t) = 0$ para $t < 0$). ADM describe la solución del sistema (2.1.12) mediante una descomposición de $x(t)$ en una serie de componentes [114, 116]:

$$x(t) = \underbrace{J^q_{t_0} \sum_{k=0}^{i-1} x^k(t)}_{Lx(t)} + \underbrace{J^q_{t_0} \sum_{k=0}^{i-1} A^k\big(x^0(t), ..., x^k(t)\big)}_{Nx(t)} + J^q_{t_0} g(t) + \eta, \tag{2.2.16}$$

donde A^k son los llamados polinomios de Adomian, calculados de la siguiente manera [116, 118]:

$$A^k_p = \frac{1}{k!}\left[\frac{d^k}{d\lambda^k} N(v^k_p(\lambda))\right]_{\lambda=0}, \quad v^k_p(\lambda) = \sum_{u=0}^{k} (\lambda)^u x^u_p, \tag{2.2.17}$$
$$k = 0, 1, ..., (i-1), \ p = 1, 2, ..., n.$$

La expresión (2.2.16) se puede reformular de la siguiente manera:

$$\begin{aligned} x^0(t) &= \eta + J^q_{t_0} g(t), \\ x^1(t) &= J^q_{t_0} x^0(t) + J^q_{t_0} A^0\big(x^0(t)\big), \\ &\vdots \\ x^i(t) &= J^q_{t_0} x^{i-1}(t) + J^q_{t_0} A^{i-1}\big(x^0(t), ..., x^{i-1}(t)\big), \\ x(t) &= \sum_{k=0}^{i} x^k(t). \end{aligned} \tag{2.2.18}$$

Vale la pena mencionar que la solución (2.2.18) tiene un pequeño radio de convergencia en relación con la trayectoria real del sistema (2.2.18). Es fácil verificar que la expresión (2.2.18) se desvía de la solución real durante períodos prolongados [119].

Para resolver este problema, el intervalo $[t_0, t]$ para una trayectoria de interés se

mapea a M subintervalos $[t_0, t_1]$, $[t_1, t_2]$, ..., $[t_{M-1}, t_M]$, divididos por un tamaño de paso $h = (t_M - t_{M-1})$.

A continuación, considerando las propiedades (2.1.8), (2.1.9) y (2.1.10), la solución ADM del sistema de orden fraccional (2.1.12) se da por:

$$\begin{aligned} c^0 =& \frac{h^q}{\Gamma(q+1)} g(t) + \eta, \\ c^1 =& Lc^0(t_j) + A^0\big(c^0\big), \\ &\vdots \\ c^i(t_j) =& Lc^{i-1} + A^{i-1}\big(c^0, ..., c^{i-1}\big), \\ x(t_{j+1}) =& \sum_{k=0}^{i} c^k \frac{h^{kq}}{\Gamma(kq+1)}, \ j = 1, 2, ..., M. \end{aligned} \tag{2.2.19}$$

Como método de solución, la implementación del ADM necesita menos recursos computacionales, no trunca la memoria del sistema y sus soluciones producen resultados precisos.

2.2.4.1. Alcance y limitaciones del enfoque ADM para la implementación de sistemas caóticos de orden fraccional

El ADM es un esquema numérico semi-analítico para resolver ecuaciones diferenciales fraccionales no lineales sin requerir largos tiempos de cálculo y con resultados de precisión adecuada [114–117, 120]. Por lo tanto, se convierte en un método efectivo para implementar sistemas caóticos de orden fraccional en FPGAs, ARMs, DPS, y similares [121–126]. Sin embargo, existen desafíos restantes que necesitan ser abordados. Estos se pueden clasificar de la siguiente manera:

- **Polinomios de Adomian.** ADM no considera funciones lineales por segmento (PWL) como Heaviside, signum, valor absoluto, diente de sierra y saturadas, que por su naturaleza no pueden ser expresadas en la forma de polinomios de Adomian, pero son esenciales en los sistemas dinámicos no lineales para generar atractores caóticos de múltiples enrollamiento [56, 110, 127, 128].
- **Complejidad de la formulación.** El ADM falla en la derivación de los polinomios de Adomian. Como se reporta en [129], cuanto mayor es el número de no linealidades, más complicada es la formulación de los polinomios de Adomian.

Además, durante la formulación pueden generarse funciones Gamma de Euler adicionales ($\Gamma(\cdot)$) según la no linealidad subyacente.

- **Rango numérico.** Cuando el ADM se aplica en el sistema de orden fraccional, la representación numérica de sus componentes (coeficientes $C^i(t_j)$ y funciones Gamma $\Gamma(\cdot)$) tiende a tener valores grandes y pequeños, respectivamente. Esto requiere formatos aritméticos de longitud extendida capaces de representar esos valores numéricos adecuadamente; sin embargo, los requisitos de hardware aumentan considerablemente.

Para comprender completamente el último problema (*Rango numérico*), es conveniente tener en cuenta el siguiente ejemplo. Consideremos el siguiente sistema de orden fraccional tridimensional

$$ {}^{C}D^{q}_{t_0}x(t) = Ax(t), \tag{2.2.20} $$

donde ${}^{C}D^{q}_{t_0}$ es la derivada de Caputo, y con cierto abuso de notación $x(t) = [x(t), y(t), z(t)]^T$. $A \in \mathbb{R}^{3\times 3}$ es una matriz real definida como

$$ A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a & -b & -c \end{bmatrix}. \tag{2.2.21} $$

De esta manera, el sistema (2.2.20) admite solo el punto de equilibrio trivial $(\bar{x}, \bar{y}, \bar{z}) = (0, 0, 0)$. De acuerdo con la condición de estabilidad $\mid \arg(\text{eig}(A)) \mid > \frac{q\pi}{2}$, el punto de equilibrio es un nodo estable cuando $q = 0.9$, $a = 5$, $b = 1.8$ y $c = 5$, y, por lo tanto, todas las trayectorias en cualquier punto dado convergen a él [130].

Al tomar los primeros $i = 4$ términos del ADM (2.2.19), la solución para el sistema (2.2.20) tiene la forma de:

$$ \begin{cases} c_1^0 = x(t_0), & c_1^1 = c_2^0, \\ c_2^0 = y(t_0), & c_2^1 = c_3^0, \\ c_3^0 = z(t_0), & c_3^1 = -ac_1^0 - bc_2^0 - c \times c_3^0, \end{cases} $$

$$\begin{cases} c_1^2 &= c_2^1, & c_1^3 = c_2^2, \\ c_2^2 &= c_3^1, & c_2^3 = c_3^2, \\ c_3^2 &= -ac_1^1 - bc_2^1 - c \times c_3^1, & c_3^3 = -ac_1^2 - bc_2^2 - c \times c_3^2, \end{cases}$$

$$\begin{cases} c_1^4 &= c_2^3, \\ c_2^4 &= c_3^3, \\ c_3^4 &= -ac_1^3 - bc_2^3 - c \times c_3^3, \end{cases}$$

$$\begin{aligned} x(t_{j+1}) &= \sum_{k=0}^{4} c_1^k \frac{h^{kq}}{\Gamma(kq+1)}, \\ y(t_{j+1}) &= \sum_{k=0}^{4} c_2^k \frac{h^{kq}}{\Gamma(kq+1)}, \\ z(t_{j+1}) &= \sum_{k=0}^{4} c_3^k \frac{h^{kq}}{\Gamma(kq+1)}, \ j = 0, 1, ..., M. \end{aligned} \tag{2.2.22}$$

La Figura 2.5 ilustra los resultados de la simulación numérica que son consistentes con el análisis de estabilidad del punto de equilibrio. Por otro lado, debemos prestar atención al comportamiento de los coeficientes (Figura 2.6 y Tabla 2.1) y las funciones Gamma (Tabla 2.2) con $k = 1, \ldots, 4$ y $p = 1, 2, 3$. Observamos que, mientras los coeficientes c_p^k pueden crecer considerablemente, el componente $\frac{h^{kq}}{\Gamma(kq+1)}$ tiende a valores muy pequeños. En ambos casos, el rango numérico depende del parámetro k; es decir, a medida que k tiende a infinito, los valores máximos y mínimos de los componentes mencionados divergen hacia valores extremos que no podrían ser abordados satisfactoriamente por dispositivos digitales, haciendo que el ADM sea inadecuado para aplicaciones de ingeniería.

Tabla 2.1: Rango numérico de los coeficientes ADM c_p^k $(k = 1, \ldots, 4;\ p = 1, 2, 3)$ considerando el sistema de orden fraccional (2.2.20).

$c_1^1 \in [-1, 1]$	$c_1^2 \in [-1, 1]$	$c_1^3 \in [-10, 1]$	$c_1^4 \in [-1, 49]$
$c_2^1 \in [-1, 1]$	$c_2^2 \in [-10, 1]$	$c_2^3 \in [-1, 49]$	$c_2^4 \in [-228, 1]$
$c_3^1 \in [-10, 1]$	$c_3^2 \in [-1, 49]$	$c_3^3 \in [-228, 1]$	$c_3^4 \in [-1, 1104]$

Para superar parcialmente los problemas del ADM, la Ref. [127] reportó una simulación adecuada de un sistema caótico de orden fraccional basada en una función signum, pero no estudiaron el costo computacional [131]. Además, los términos adicionales de las funciones Gamma se evitaron aplicando el algoritmo Conformable

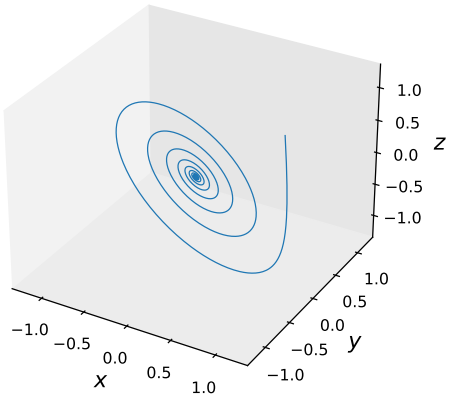


Figura 2.5: Diagrama de fases del sistema fraccional tridimensional (2.2.20) utilizando el ADM con $h = 0.01$ y condiciones iniciales $[x(t_0), y(t_0), z(t_0)] = [1, 0, 1]$.

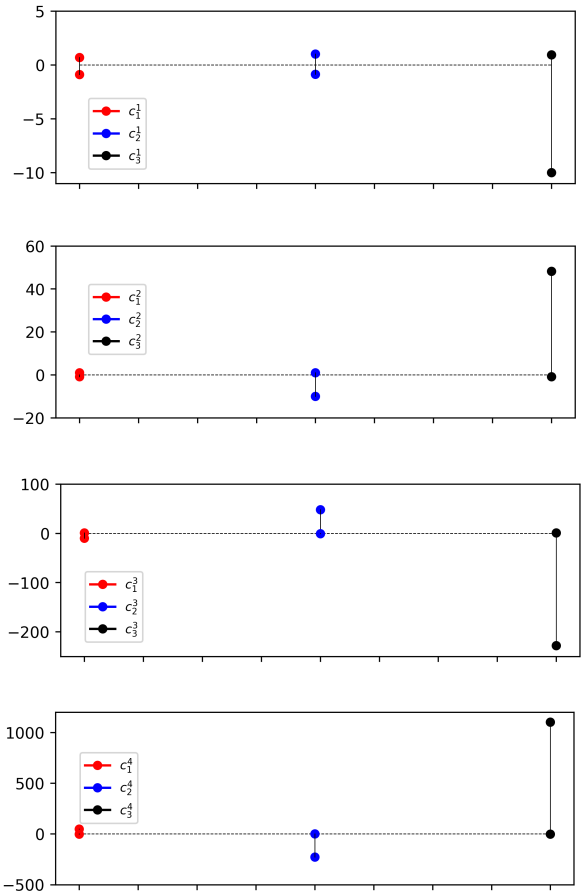


Figura 2.6: Evolución del comportamiento de los coeficientes ADM c_p^k $(k = 1, \ldots, 4;\ p = 1, 2, 3)$ para el sistema de orden fraccional (2.2.20).

Tabla 2.2: Rango numérico de los términos ADM $\frac{h^{kq}}{\Gamma(kq+1)}$ con $q = 0.9$ para el sistema de orden fraccional (2.2.20).

Terms $\frac{h^{iq}}{\Gamma(iq+1)}$	Values
$\frac{h^{q}}{\Gamma(q+1)}$	1.647×10^{-2}
$\frac{h^{2q}}{\Gamma(2q+1)}$	1.498×10^{-4}
$\frac{h^{3q}}{\Gamma(3q+1)}$	9.545×10^{-7}
$\frac{h^{4q}}{\Gamma(4q+1)}$	4.715×10^{-9}

ADM (CADM). Aunque el enfoque propuesto reduce la complejidad de la formulación tradicional del ADM, los resultados son sospechosos ya que se basan en una derivada fraccional conforme que no pertenece a la clase de derivadas fraccionales F_1 [132].

2.3. Desempeño de los métodos de implementación de sistemas caóticos de orden fraccional

En la Tabla 2.3, se observa que ADM tiene mejores características que los algoritmos de ABM y GL en términos de complejidad computacional [121]. En cuanto aplicaciones practicas, los algoritmos de ADM y GL proporcionan resultados adecuados y son los métodos de solución más dominantes tanto para implementaciones basadas en FPGAs como en ARMs y DSPs, tal como se muestra en la Figura 2.7. En cuanto a la utilización de recursos de hardware y la flexibilidad en el diseño, la aproximación GL requiere un gran número de muestras pasadas para representar los efectos de memoria larga de las derivadas fraccionales, lo que provoca costos computacionales y de hardware considerables. Como consecuencia de estas limitaciones, la aproximación GL apenas se emplea para implementaciones basadas en hardware embebido (DSP, ARM, etc.), porque estos chips digitales tienen características computacionales limitadas. Sin embargo, el método numérico GL se puede implementar como un Bloque de Construcción Básico y, por lo tanto, reutilizarse para diversas implementaciones. Así, ofrece mayor flexibilidad para obtener realizaciones físicas de varios tipos de sistemas caóticos de orden fraccional [53, 133, 134]. Esta característica particular convierte al algoritmo de GL en la opción preferida para implementaciones basadas en FPGA.

Por otro lado, se ha analizado extensamente en la sección 2.2.4 que ADM no implica tiempos de simulación largos para representar la memoria de las derivadas fraccionales. Por lo tanto, se reconoce como la opción más conveniente para implementar sistemas caóticos de orden fraccional con menores recursos de hardware [135, 136]. Sin embargo, desde el punto de vista algorítmico, uno de los principales inconvenientes del enfoque ADM es la complejidad asociada para determinar los polinomios de Adomian para los distintos tipos de no linealidades en los sistemas caóticos de orden fraccional. Esto significa que se necesita un rediseño casi por completo para cada sistema caótico particular, lo que provoca bajas características de reutilización del diseño. Este inconveniente convierte a ADM en una opción preferida para implementaciones basadas en sistemas embebidos.

Otra característica importante en implementaciones de sistemas caóticos de orden fraccional es el **número de enrollamientos que componen al atractor caótico**. Este parámetro es especialmente importante en aplicaciones prácticas como la criptografía, la robótica y control, ya que un mayor número de enrollamientos generalmente implica un comportamiento más complejo, lo que a su vez se traduce en un mejor rendimiento en estas aplicaciones. En la literatura, se han realizado diversas implementaciones digitales de sistemas caóticos de orden fraccional, como se observa en las Tablas 2.4, 2.5 y 2.4. Las implementaciones reportadas se centran mayormente en sistemas con funciones no lineales de potencia, como $f(x) = x^n$, y particularmente en funciones de producto cruzado, que son comunes en sistemas tipo Lorenz. La Figura 2.7 ilustra una clasificación de estas implementaciones según el método utilizado y el tipo de no linealidad. En cuanto a funciones PWL, la mayoría de los trabajos aborda el método de GL como opción principal, debido a su capacidad para manejar la complejidad de estas funciones y su flexibilidad en la discretización. Por otra parte, son escasos los trabajos que abordan este tipo de funciones utilizando el método ADM. Es importante mencionar que estos estudios generalmente se enfocan en sistemas con una baja complejidad en las funciones PWL, como las funciones signo. Si bien, una función signo no es estrictamente una función lineal por segmentos, puede interpretarse como una serie de funciones con diferentes niveles de saturación en lugar de tramos lineales continuos y, por lo tanto, describirse de la misma manera que una función PWL. Esta simplificación resulta en

atractores con un menor número de enrollamientos.

En cuanto a la **representación aritmética digital**, las implementaciones de sistemas caóticos de orden fraccional en ARM y DSPs suelen utilizar una representación aritmética de punto flotante, debido a que estos dispositivos incluyen una unidad de punto flotante (FPU, por sus siglas en inglés). Estas implementaciones pueden beneficiarse de un mayor rango de valores y una mayor precisión, por otra parte, las implementaciones basadas en FPGA siguen un enfoque de punto fijo donde la precisión en el ancho de bits debe determinarse adecuadamente para evitar efectos de degradación dinámica; sin embargo, este sigue siendo un problema no resuelto. Aunque no existe un estudio formal que responda a esta cuestión, a partir de los trabajos reportados, se observa que la longitud de bits elegida para la mayoría de los trabajos publicados es de 32 bits de precisión para implementaciones basadas en aritmética de punto fijo. El paso empírico consiste en seleccionar el número mínimo necesario de bits para la parte entera y dejar los bits restantes para la parte no entera, respectivamente. En caso de que la precisión seleccionada sea menor a 32 bits, se puede producir degradación dinámica, tal como se reporta en [137], donde se detectó una degradación de la dinámica caótica en un sistema de orden fraccional a medida que disminuía el número de bits para la parte no entera.

Finalmente, es fundamental considerar que la selección del método de implementación y la representación aritmética debe estar alineada con las especificaciones y limitaciones del sistema objetivo. La correcta integración de estos aspectos no solo asegura la eficiencia y precisión del sistema caótico de orden fraccional, sino que también optimiza su desempeño en aplicaciones prácticas, permitiendo aprovechar al máximo las capacidades de las plataformas de hardware disponibles.

Tabla 2.3: Comparación de rendimiento entre los métodos de solución ABM, GL y ADM.

	ABM	GL	ADM
Complejidad temporal	$O(n^2)$	$O(n^2)$	$O(n)$
Complejidad espacial	$O(n)$	$O(n)$	$O(1)$

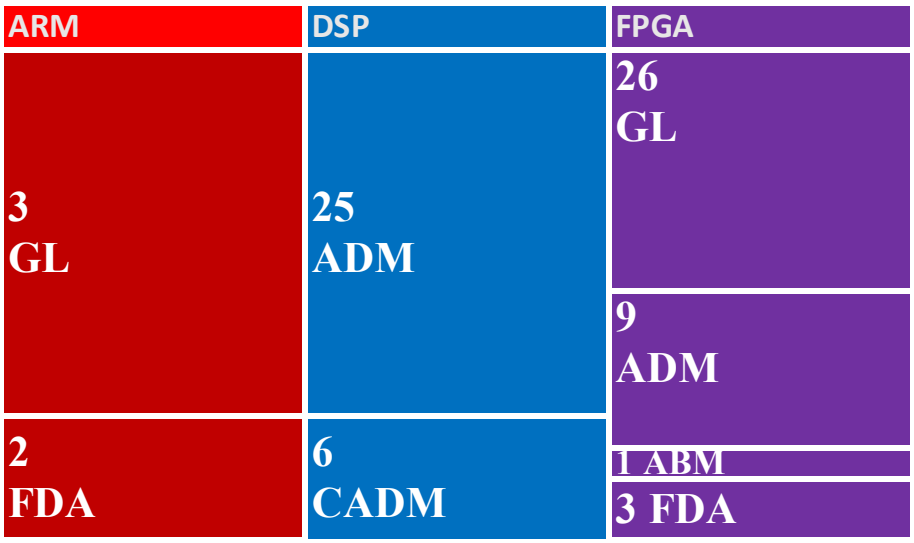


Figura 2.7: Gráfico de mapa de colores ponderado de las publicaciones existentes sobre implementaciones embebidas y no embebidas de sistemas caóticos de orden fraccional. Los colores indican la *Plataforma* utilizada: ARM (rojo), DSP (azul), FPGA conforme (morado); y estos, a su vez, se subdividen en cuadros que denotan el número de publicaciones asociadas con los diferentes *métodos de solución*.

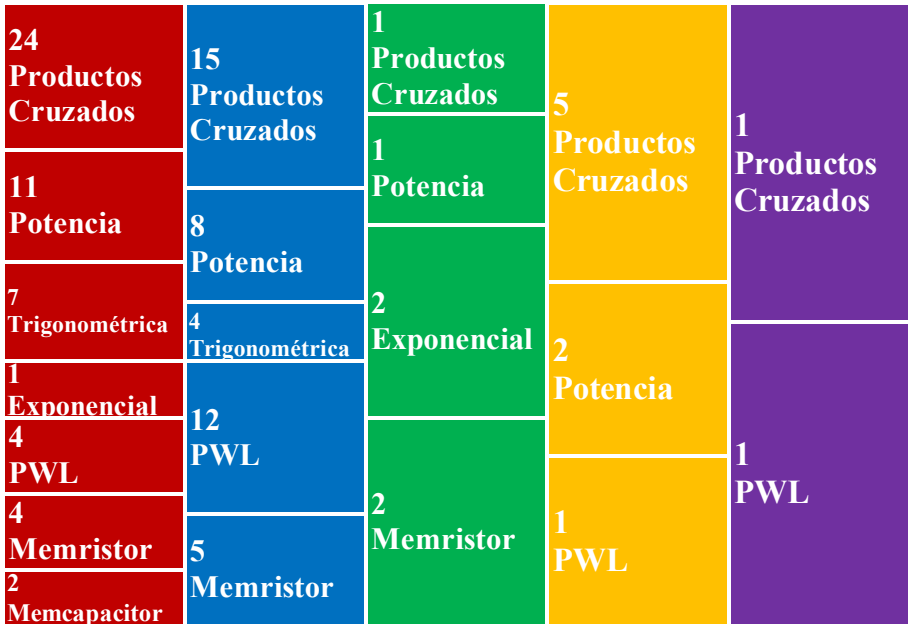


Figura 2.8: Gráfico de mapa de colores ponderado de las publicaciones existentes sobre implementaciones embebidas y no embebidas de sistemas caóticos de orden fraccional. Los colores indican el *Método de Solución* utilizado: ADM (rojo), aproximación GL (azul), ADM conforme (verde), FDA (amarillo) y ABM (morado); y estos, a su vez, se subdividen en cuadros que denotan el número de publicaciones asociadas con los diferentes *Tipos de Funciones No Lineales*.

Tabla 2.4: Trabajos que reportan implementaciones de sistemas caóticos de orden fraccional en plataformas FPGA.

Función no lineal	Tipo de atractor	Dinámicas Reportadas	Número de enrollamientos	Método de resolución	Representación aritmética	Lenguaje de síntesis	Referencia
-Producto cruzado -Potencia -PWL	-Autoexitado -Oculto	-Hipercaos	× 2	ADM	–	Vivado	[138], 2017
Memristor	-Autoexitado -Oculto	-Hipercaos	× 2	GL	Fijo	-Simulink -Vivado	[139],2017
-Producto cruzado	-Autoexitado -Oculto	–	× 1	GL	–	Xilinx tools	[140], 2017
-Producto cruzado	-Oculto	-Hipercaos	× 2	ADM	–	–	[141], 2018
-Producto cruzado -Potencia	-Autoexitado	–	× 1	GL	Fijo	-Simulink -Vivado	[142], 2018
-Memcapacitor	-Oculto	-Hipercaos -Multistability	× 1	ADM	–	Vivado	[143], 2018
-Producto cruzado -Potencia -PWL	-Autoexitado -Oculto	–	× 6 6×6 6×6×6	GL	Fijo	–	[56], 2019
Memristor	-Autoexitado	Multistability	× 4	ADM	–	–	[136], 2019
PWL	-Autoexitado	Multiscroll	Multiscroll × 16	GL	Fijo	Verilog	[144], 2019
-Producto cruzado -Potencia -Trigonometric	-Autoexitado	–	× 4	GL		-Simulink -Vivado	[145], 2019
PWL	Autoexitado	–	× 2	GL	Fijo	Verilog	[146], 2020
-Producto cruzado -Potencia	Autoexitado	–	× 2	GL	Fijo	Verilog	[137], 2020
-Producto cruzado -Potencia	Autoexitado	Time-delay	× 1	GL	Fijo	Verilog	[147], 2020

Tabla 2.5: Trabajos reportan implementaciones de sistemas caóticos de orden fraccional en plataformas DSP.

Función no lineal	Atractor tipo	Dinámicas reportadas	Número de enrollamientos	Método de resolución	Representación aritmética	Lenguaje de síntesis	Referencia
-Producto cruzado	-Autoexitado	–	× 2	ADM	Fijo	–	[148], 2015
-Producto cruzado	-Autoexitado	–	× 2	ADM	Fijo	–	[149], 2015
-Producto cruzado	-Autoexitado	–	× 2	ADM	Flotante	–	[150], 2016
-Producto cruzado	-Autoexitado	-Hipercaos -Multistabilidad	× 4	ADM	Fijo	–	[151], 2017
-Producto cruzado	-Autoexitado	–	× 1	ADM	Fijo	–	[152], 2018
-Memristor	-Autoexitado	–	× 1	C-ADM	Fijo	–	[153], 2018
-Producto cruzado	-Autoexitado	Hipercaos	× 2	ADM	Flotante	–	[154], 2018
-Trigonométrico	-Autoexitado	Hipercaos	× 4	ADM	Flotante	–	[127], 2019
-Memristor	-Oculto	Multistabilidad	× 3	ADM	Fijo	–	[155], 2019
-Producto cruzado	-Autoexitado	–	× 2	ADM	Flotante	–	[156], 2019
-Memristor	-Autoexitado	Extremo Multistabilidad	× 1	C-ADM	Flotante	–	[125], 2020
-Producto cruzado	-Autoexitado	Multistabilidad	× 2	ADM	Flotante	C	[126], 2020
-Producto cruzado	-Autoexitado	Hipercaos	× 2	ADM	Flotante	–	[157], 2020
-Trigonométrico	-Autoexitado	Extremo Multistabilidad	× 2	ADM	–	C	[158], 2020
-Producto cruzado -Potencia -PWL	-Oculto	Extremo Multistabilidad	× 1	ADM	Flotante	–	[159], 2021
-PWL	-Autoexitado	Multistabilidad	× 1	ADM	Flotante	C	[160], 2021
-Producto cruzado -Potencia -Exponencial	-Autoexitado -Oculto	Multistabilidad	× 2	ADM	Flotante	C	[161], 2021

Tabla 2.6: Trabajos reportan implementaciones embebidas de sistemas caóticos de orden fraccional en plataformas ARM.

Función no lineal	Tipo de atractor	Dinámica reportada	Número de Enrollamientos	Método de solución	Representación aritmética	Lenguaje de síntesis	Referencia
PWL	Autoexcitado	–	× 2	FDA	–	Python	[49], 2017
Memristor	Oculto	Hipercaos	× 2	GL	–	C	[51], 2019
Producto cruzado	Autoexcitado	–	× 4	FDA	Flotante	C++	[90], 2020
Producto cruzado -Potencia	Autoexcitado	–	× 1	GL	–	–	[162], 2020
Producto cruzado	Autoexcitado	–	× 2	GL	Flotante	Python	[30], 2020

Capítulo 3

Formulación del Método de Descomposición PWL para Sistemas Caóticos de Orden Fraccional

En este capítulo, se llevará a cabo un análisis detallado de sistemas caóticos de orden fraccional, centrándose en aquellos basados en funciones PWL. Los sistemas caóticos de orden fraccional han demostrado ser una poderosa herramienta en el modelado de fenómenos complejos en diversas áreas, desde la física hasta la ingeniería. Otro de los motivos para enfocar el análisis en sistemas basados en funciones PWL, radica en su versatilidad. Estas funciones pueden integrarse de forma relativamente simple en modelos lineales para inducir dinámicas caóticas, lo que las convierte en una opción atractiva para estudiar sistemas de orden fraccional. Además, las funciones PWL forman una extensa familia dentro de los sistemas de orden fraccional, lo que permite su aplicación en un amplio espectro de problemas prácticos.

Una de las principales áreas de mejora identificadas en el capítulo anterior, es la necesidad de un esquema para ADM que permita abordar de manera efectiva los sistemas basados en funciones PWL. Este esquema, no solo busca ampliar el alcance del análisis, sino también sentar las bases para futuras investigaciones en el ámbito de los modelos de orden fraccional.

Por lo tanto, este capitulo explora las características y dinámicas de los sistemas caóticos de orden fraccional basados en funciones PWL y busca aportar un esquema basado en ADM para el estudio de sistemas caóticos de orden fraccional.

3.1. Método de descomposición propuesto (enfoque PWL-DM)

Considere una clase de sistemas PWL de orden fraccional definidos por

$$ {}^{C}D^{q}_{t_0}X = AX + BU(X) + G, \tag{3.1.1} $$

donde ${}^{C}D^{q}_{t_0}X = \left[{}^{C}D^{q}_{t_0}x_1, {}^{C}D^{q}_{t_0}x_2, \ldots, {}^{C}D^{q}_{t_0}x_n\right]^T$, $X = \left[x_1, x_2, \ldots, x_n\right]^T \in \mathbb{R}^n$, ${}^{C}D^{q}_{t_0}$ denota el operador derivado de Caputo de orden q con $0 < q \leq 1$, $A = \{\alpha_{\zeta_1,\zeta_2}\}^n_{\zeta_1,\zeta_2=1} \in \mathbb{R}^{n\times n}$ y $B = \{\beta_{\zeta_1,\zeta_2}\}^n_{\zeta_1,\zeta_2=1} \in \mathbb{R}^{n\times n}$ son operadores lineales con α_{ζ_1,ζ_2}, $\beta_{\zeta_1,\zeta_2} \in \mathbb{R}^n$, $G = \left[g_1, g_2, \ldots, g_n\right]^T \in \mathbb{R}^n$, es la constante para el sistema autónomo. $n \in \mathbb{N}^+$ representa el número de variables del sistema, y $U : \mathbb{R}^n \to \mathbb{R}^n$ representa un vector PWL que se define de la siguiente manera:

$$ U(X) = \begin{cases} U_1(X), & \text{if } X \in \Phi_1, \\ U_2(X), & \text{if } X \in \Phi_2, \\ \vdots & \\ U_{p_1}(X), & \text{if } X \in \Phi_{p1}, \end{cases} \tag{3.1.2} $$

donde $\Phi = \Phi_1, \Phi_2, \ldots, \Phi_{p_1}$ es una partición finita del espacio de fases, que satisface $\mathbb{R}^n = \bigcup_{\varsigma=1}^{p_1} \Phi_\varsigma$. El vector U cambia dependiendo del dominio $\Phi_\varsigma \subset \mathbb{R}^n$ en el que se encuentre la trayectoria. Para este propósito, las funcionws PWL U_ς con $\varsigma = 1, 2, \ldots, p_1$, se pueden definir como $U_\varsigma = \left[u_{\varsigma 1}, \ldots, u_{\varsigma n}\right]^T \in \mathbb{R}^n$, cuyos componentes están dados por:

$$ \begin{bmatrix} u_{\varsigma 1} \\ u_{\varsigma 2} \\ \vdots \\ u_{\varsigma n} \end{bmatrix} = \begin{bmatrix} f_1(x_1) \\ f_2(x_2) \\ \vdots \\ f_n(x_n) \end{bmatrix}. \tag{3.1.3} $$

y las funciones $f_1, f_2, \ldots, f_n$ se definen de la siguiente manera:

$$f_i(x_i) = \begin{cases} m_1^i x_i + c_1^i, \text{ if } X \in H_1^i = \{x_i < s_1^i\}, \\ m_2^i x_i + c_2^i, \text{ if } X \in H_2^i = \{s_1^i \leq x_i < s_2^i\}, \\ \quad \vdots \\ m_{k_i}^i x_i + c_{k_i}^i, \text{ if } x_i \in H_{k_i}^i = \{s_{k_i-1}^i \leq x_i\}, \end{cases} \tag{3.1.4}$$

donde $m_j^i, c_j^i \in \mathbb{R}$ para $i = 1, 2, \ldots, \varphi$, $j = 1, 2, \ldots, k_i$, representan la pendiente y la intercepción de cada recta en las funciones PWL. $\varphi \in \mathbb{N}^+$ indica el número de funciones PWL consideradas en la ecuación (3.1.3), $k_i \in \mathbb{N}^+$ denota el número total de segmentos afines que forman cada función $f_i(x_i)$, y el subíndice j simboliza el elemento (pendiente o intercepción) que pertenece a una función PWL dada por el superíndice i. Así, cada función $f_i(x_i)$ genera subdominios $H^i = \left\{H_1^i, H_2^i, \ldots, H_{k_i}^i\right\} = H_j^i$ con $H = \left\{H^1, H^2, \ldots, H^\varphi\right\}$, donde las superficies S^i en términos de s_j^i con $1 \leq j \leq k_i - 1$ especifican los límites entre dos subdominios consecutivos H_j^i. Cabe señalar que las intersecciones entre esos subdominios componen la partición finita del espacio de fases Φ_ς para $\varsigma = 1, 2, \ldots, p_1$ con $p_1 = \prod_{i=1}^{\varphi} k_i$. En particular, el rol de la matriz B en la ecuación (3.1.1) es definir cuáles y cuántas funciones PWL están activas en una variable dada. En el caso general, se asume que el número de funciones PWL equivale al número de variables del sistema, es decir, $\varphi = n$.

Considerando el escenario con tres funciones PWL ($\varphi = 3$) compuesta por k_i segmentos afines y $X \in \mathbb{R}^3$, los vectores U_ς se representan por $\left[u_{\varsigma 1}, u_{\varsigma 2}, u_{\varsigma 3}\right]^T = \left[f_1(x_1), f_2(x_2), f_3(x_1)\right]^T$. Como resultado, el sistema lineal afín que gobierna la dinámica en el dominio Φ_ς, con $\varsigma = j$ dado que $p_1 = k_1$, se da por

$${}^C D_{t_0}^q X = AX + B\left[\left(mj^1 x_1 + c_j^1\right), \left(mj^2 x_2 + c_j^2\right), \left(mj^3 x_3 + c_j^3\right)\right]^T + G, \tag{3.1.5}$$

o equivalentemente a

$${}^C D_{t_0}^q X = AX + B\left[mj^1 x_1, mj^2 x_2, mj^3 x_3\right]^T + B\left[c_j^1, c_j^2, c_j^3\right]^T + G. \tag{3.1.6}$$

De esta manera, el componente $U(X)$ en la ecuación (3.1.1) se puede dividir en un vector por partes $U_L(X)$ y un vector constante por partes U_C de acuerdo con

las pendientes m_j^i y las intercepciones c_j^i de las funciones PWL, respectivamente. Entonces, el sistema PWL de orden fraccional (3.1.1) se convierte en:

$$ {}^{C}D_{t_0}^{q} X = AX + BU_L(X) + BU_C + G. \tag{3.1.7} $$

Aplicando el operador integral fraccional $J_{t_0}^{q}$ y la propiedad (2.1.11) a ambos lados de la ecuación (3.1.7) con el estado inicial $X(t_0^+)$ para $m = 1$, se obtiene:

$$ X = J_{t_0}^{q} AX + J_{t_0}^{q} BU_L(X) + J_{t_0}^{q} BU_C + J_{t_0}^{q} G + X(t_0^+), \tag{3.1.8} $$

basado en el método de descomposición de Adomian, la solución numérica de la ecuación (3.1.8) es:

$$ X = \sum_{\ell=0}^{\infty} X^{\ell} = J_{t_0}^{q} \sum_{\ell=0}^{\infty} AX^{\ell} + J_{t_0}^{q} \sum_{\ell=0}^{\infty} BU_L(X^{\ell}) + J_{t_0}^{q} BU_C + J_{t_0}^{q} G + X(t_0^+), \tag{3.1.9} $$

donde $\ell = 0, 1, \ldots, \infty$.

Ahora, el siguiente paso consiste en derivar la serie de descomposición lineal por partes a partir de (3.1.9). El primer paso es determinar la partición finita Φ_ς del espacio de fases. Dado un numero de funciones PWL (φ) no necesariamente equivalen a las variables del sistema, ya que $\varphi \leq n$, y sabiendo que cada función lineal por partes puede tener un número arbitrario de segmentos afines k_i, es decir, una longitud diferente de elementos (pendientes e intercepciones), se establece el Teorema 1.

Teorema 1 *Para una clase de sistemas PWL de orden fraccional dada por la Ecuación (3.1.1) que contiene varias funciones PWL, es decir, $\varphi \geq 1$, con cada función teniendo una longitud arbitraria k_i, la partición finita Φ_ς para $\varsigma = 1, 2, \ldots, p_1$ del espacio de fases con $H = \left\{H^1, H^2, \ldots, H^{\varphi}\right\} = H^i$ y $H^i = \left\{H_1^i, H_2^i, \ldots, H_{k_i}^i\right\} = H_j^i$, $i = 1, 2, \ldots, \varphi$, $j = 1, 2, \ldots, k_i$, se determina por:*

$$ \Phi_\varsigma = \left\{ \cap \left(H^1 \times H^2 \times \ldots \times H^{\varphi}\right)\right\}, \tag{3.1.10} $$

donde $\left(H^1 \times H^2 \times \ldots \times H^{\varphi}\right)$, con algún abuso de notación, es el producto cartesiano definido como el conjunto de todos los φ-tuplas ordenadas posibles. $\cap(\cdot)$ representa la intersección entre cada una de las φ-tuplas resultantes. Así, $\mathbb{R}^n$ se particiona en p_1

subdominios con $p_1 = \prod_{i=1}^{\varphi} k_i$.

Demostración 1 *Sin pérdida de generalidad, se asumen tres funciones PWL* ($\varphi = 3$)*, con longitudes* $k_1 = 2$, $k_2 = 3$ *y* $k_3 = 4$, *respectivamente. Así, el número de subdominios es* $p_1 = \prod_{i=1}^{3} k_i = 24$, *ya que la cardinalidad de un producto cartesiano de tres conjuntos es igual al producto de las cardinalidades de los conjuntos:* $|H^1 \times H^2 \times H^3| = |H^1| \times |H^2| \times |H^3|$, *con* $H^1 = \left\{H_1^1, H_2^1\right\}$, $H^2 = \left\{H_1^2, H_2^2, H_3^2\right\}$, *y* $H^3 = \left\{H_1^3, H_2^3, H_3^3, H_4^3\right\}$, *respectivamente. Como resultado de las* p_1 *combinaciones, la partición finita* Φ *del espacio de fases contiene los subdominios:*

$$\begin{aligned}\Phi_\varsigma =&\left\{H_1^1 \cap H_1^2 \cap H_1^3, H_1^1 \cap H_1^2 \cap H_2^3, H_1^1 \cap H_1^2 \cap H_3^3, H_1^1 \cap H_1^2 \cap H_4^3, \ldots, H_2^1 \cap H_3^2 \cap H_1^3, H_2^1,\right. \\ &\left. \cap H_3^2 \cap H_2^3, H_2^1 \cap H_3^2 \cap H_3^3, H_2^1 \cap H_3^2 \cap H_4^3\right\},\end{aligned} \tag{3.1.11}$$

donde las combinaciones posibles se pueden reformular como el producto cartesiano de $\left(H^1 \times H^2 \times H^3\right)$ *a partir de todas las posibles 3-tuplas ordenadas.*

Este resultado es válido para un número arbitrario de elementos k_i *que componen las funciones* φ*–PWL. Sean* $H^1, \ldots, H^\varphi$ φ*–conjuntos no vacíos. El producto cartesiano* $\left(H^1 \times \ldots \times H^\varphi\right)$ *se obtiene por el conjunto de todas las posibles* φ*–tuplas ordenadas* $(H_1^1, ..., H_j^i)$*, donde* $H_j^i \in H^i$, $i = 1, \ldots, \varphi$. *Además, la cardinalidad del producto cartesiano de los* φ*–conjuntos se obtiene mediante* $|H^1 \times \ldots \times H^\varphi| = |H^1| \times \ldots \times |H^\varphi|$, *demostrando que el número total de subdominios es* $p_1 = \prod_{i=1}^{k+1} k_i$, *completando la prueba.*

Una vez que se han determinado los subdominios en la partición finita del espacio de fases, la siguiente tarea consiste en incorporar las pendientes m_j^i y las intersecciones c_j^i de todas las funciones PWL en el enfoque propuesto.

Para una clase de sistemas PWL de orden fraccional en la ecuación (3.1.1) que cumplen con el Teorema 1, los componentes $U_L(X)$ y U_C en el método de descomposición e (3.1.9) pueden reorganizarse en pares de matrices $\mathcal{M}$ y $\mathcal{C}$, compuestas por las pendientes e intercepciones de las funciones PWL, respectivamente. El proceso para construir estas matrices es el siguiente:

Considerando φ conjuntos M^i que contienen los elementos m_j^i, donde $i = 1, 2, \ldots, \varphi$ y $j = 1, 2, \ldots, k_i$. Cada M^i corresponde a un conjunto de pendientes de funciones PWL, con k_i siendo el número de segmentos en la i-ésima función PWL, (por simplicidad, se omiten las variables x_i). La matriz $\mathcal{M}$ se construye combinando

los elementos de estos conjuntos de manera que cada elemento m_j^i se repite un número de veces determinado por las combinaciones posibles con los elementos de los otros conjuntos:

1. **Vector fila** M^1: El vector $M^1 = [m_1^1, m_2^1, \ldots, m_{k_1}^1]$ se coloca en la primera fila de la matriz $\mathcal{M}$. Cada elemento m_j^1 se repite $\prod_{i=2}^{\varphi} k_i$ veces.

2. **Vector fila** M^2:** El segundo vector $M^2 = [m_1^2, m_2^2, \ldots, m_{k_2}^2]$ se coloca en la segunda fila de la matriz $\mathcal{M}$. Cada elemento m_j^2 se repite $\prod_{i=3}^{\varphi} k_i$ veces, pero todo el vector M^2 se repite k_1 veces.

3. **Vector fila** M^3: El tercer conjunto $M^3 = [m_1^3, m_2^3, \ldots, m_{k_3}^3]$ se coloca en la tercera fila de la matriz $\mathcal{M}$. Cada elemento m_j^3 se repite $\prod_{i=4}^{\varphi} k_i$ veces, pero todo el vector M^3 se repite $k_1 \times k_2$ veces.

4. **Vector** M^i: El procedimiento continúa de esta manera, repitiendo cada elemento de M^i en función del producto de los valores de k_i correspondientes a los conjuntos posteriores, mientras que todo el vector M^i se repite un número de veces determinado por el producto de los valores de k_i de los conjuntos anteriores.

5. **Elemento Final** M^{φ}: El último vector $M^{\varphi} = [m_1^{\varphi}, m_2^{\varphi}, \ldots, m_{k_{\varphi}}^{\varphi}]$ se coloca en la fila φ de la matriz $\mathcal{M}$. Cada elemento m_j^{φ} no se repite y todo el vector M^{φ} se repite $\prod_{i=1}^{\varphi-1} k_i$ veces.

Ejemplo: Considerando dos vectores $M^1 = [m_1^1 x_1, m_2^1 x_1, m_3^1 x_1]$ y $M^2 = [m_1^2 x_2, m_2^2 x_2]$ con $k_1 = 3$ y $k_2 = 2$, la matriz $\mathcal{M}$ se construye como:

$$\mathcal{M} = \begin{bmatrix} m_1^1 x_1, m_1^1 x_1, m_2^1 x_1, m_2^1 x_1, m_3^1 x_1, m_3^1 x_1 \\ m_1^2 x_2, m_2^2 x_2, m_1^2 x_2, m_2^2 x_2, m_1^2 x_2, m_2^2 x_2 \end{bmatrix}.$$

En este caso, cada elemento de M^1 se repite k_2 veces, y cada elemento de M^2 se repite k_1 veces para generar todas las combinaciones posibles.

El procedimiento mencionado es el mismo para calcular la matriz $\mathcal{C}$, donde $\mathcal{C}$ se construye a partir de los vectores C^i, que corresponden a las intercepciones de las funciones PWL.

Ejemplo: Considerando dos vectores $C^1 = [c_1^1, c_2^1, c_3^1]$ y $C^2 = [c_1^2, c_2^2]$ con $k_1 = 3$ y $k_2 = 2$, la matriz $\mathcal{C}$ se construye siguiendo el mismo método de repetición y combinación utilizado para $\mathcal{M}$:

$$\mathcal{C} = \begin{bmatrix} c_1^1, c_1^1, c_2^1, c_2^1, c_3^1, c_3^1 \\ c_1^2, c_2^2, c_1^2, c_2^2, c_1^2, c_2^2 \end{bmatrix}.$$

Basado en el Teorema 1 y en el procedimiento anterior, se puede reformular la solución numérica de la ecuación (3.1.9) en una serie de descomposición por segmentos lineales, denominada como PWL-DM, de la siguiente manera:

$$X = \sum_{\ell=0}^{\infty} X^{\ell} = J_{t_0}^{q} \sum_{\ell=0}^{\infty} AX^{\ell} + J_{t_0}^{q} \sum_{\ell=0}^{\infty} B\mathcal{M}_{(*,\varsigma)}(X^{\ell}) + J_{t_0}^{q} B\mathcal{C}_{(*,\varsigma)} + J_{t_0}^{q} G + X(t_0^{+}), \tag{3.1.12}$$

con

$$\mathcal{M}_{(*,\varsigma)}, \mathcal{C}_{(*,\varsigma)} = \begin{cases} \mathcal{M}_{(*,1)}, \mathcal{C}_{(*,1)}, & \text{if } X \in \Phi_1, \\ \mathcal{M}_{(*,2)}, \mathcal{C}_{(*,2)}, & \text{if } X \in \Phi_2, \\ \vdots & \\ \mathcal{M}_{(*,p1)}, \mathcal{C}_{(*,p1)}, & \text{if } X \in \Phi_{p1}, \end{cases} \tag{3.1.13}$$

donde $\mathcal{M}_{(*,\varsigma)}$ y $\mathcal{C}_{(*,\varsigma)}$, $\varsigma = 1, 2, \ldots, p_1$ representan la ς-ésima columna de las matrices $\mathcal{M}$ y $\mathcal{C}$. En particular, la notación $(*, \varsigma)$ se refiere a la expresión de indexación para obtener todos los elementos de fila en la ς-ésima columna. Además, Φ_{ς} denota la partición finita del espacio de fases dada por las p_1 intersecciones en la ecuación (3.1.10), es decir, $\mathbb{R}^n = \bigcup_{\varsigma=1}^{p_1} \Phi_{\varsigma}$, $\bigcap_{\varsigma=1}^{p_1} \Phi_{\varsigma} = \emptyset$, mientras que el superíndice ℓ simboliza el elemento de la serie de descomposición. Por lo tanto, la serie de descomposición $X^0, X^1, \ldots, X^{\ell}$ en cada sub-dominio se obtiene de la siguiente forma:

$$\begin{aligned} X^0 &= J_{t_0}^{q} B\mathcal{C}_{(*,\varsigma)} + J_{t_0}^{q} G + X(t_0^{+}), \\ X^1 &= J_{t_0}^{q} AX^0 + J_{t_0}^{q} B\mathcal{M}_{(*,\varsigma)}(X^0), \\ X^2 &= J_{t_0}^{q} AX^1 + J_{t_0}^{q} B\mathcal{M}_{(*,\varsigma)}(X^1), \\ &\vdots \\ X^{\ell} &= J_{t_0}^{q} AX^{\ell-1} + J_{t_0}^{q} B\mathcal{M}_{(*,\varsigma)}(X^{\ell-1}), \end{aligned} \tag{3.1.14}$$

La ecuación (3.1.14), se define como la solución exacta de la ecuación (3.1.1), pero es importante notar que en implementaciones prácticas, es imposible calcular infinitos términos de X^{ℓ}. No obstante, debido a que el enfoque de descomposición converge muy rápido [163, 164], la solución del sistema puede aproximarse a partir de los primeros términos (3.1.1). Para discretizar esta ecuación, se divide el intervalo $[t_0, t]$ en subintervalos $[t_n, t_{n+1}]$. De esta manera, la solución aproximada del término ρ se puede expresar como $X(t_{n+1}) = \sum_{\ell=0}^{\rho-1} X^{\ell}(t_n) = F\big(X(t_n)\big)$. Entonces se puede obtener la forma iterativa discreta $X(t_{n+1}) = F\big(X(t_n)\big)$, que se denota como $X(n+1) = F(X(n))$ con $\rho = 5$ para casos generales.

3.2. Utilizando PWL-DM para resolver sistemas caóticos PWL de orden fraccional

En esta sección, el método de descomposición propuesto (PWL-DM) se aplica para obtener la solución de sistemas caóticos PWL orden fraccional con características dinámicas conocidas, los cuales ha sido reportados en [165, 166]. Cabe destacar que el algoritmo PWL-DM puede aplicarse a la clase general de sistemas dinámicos PWL de orden fraccional (3.1.1).

3.2.1. Ejemplo 1: Demostración paso a paso para un sistema caótico PWL de orden fraccional de 2 enrollamientos

Considera un sistema dinámico de orden fraccional dado por la ecuación (3.1.1) en $\mathbb{R}^3$ con operadores lineales:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\alpha_{31} & -\alpha_{32} & -\alpha_{33} \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \beta_{31} & 0 & 0 \end{bmatrix}, \qquad G = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \tag{3.2.1}$$

onde α_{31}, α_{32}, α_{33} y β_{31} son constantes reales positivas. Además, el sistema posee una única función PWL f_1, expresada en la forma de la ecuación (3.1.4) como:

$$f_1(x_1) = \begin{cases} m_1^1 x_1 + c_1^1, & \text{if } x_1 \in H_1^1 = \{x_1 < s_1^1\}, \\ m_2^1 x_1 + c_2^1, & \text{if } x_1 \in H_2^1 = \{s_1^1 \leq x_1 < s_2^1\}, \\ m_3^1 x_1 + c_3^1, & \text{if } x_1 \in H_3^1 = \{s_2^1 \leq x_1\}. \end{cases} \tag{3.2.2}$$

Por lo tanto, el vector U_ς en la ecuación (3.1.3) con $\varsigma = 1, 2, 3$ tienen componentes definidas por:

$$\begin{bmatrix} u_{\varsigma 1} \\ u_{\varsigma 2} \\ u_{\varsigma 3} \end{bmatrix} = \begin{bmatrix} f_1(x_1) \\ 0 \\ 0 \end{bmatrix}. \tag{3.2.3}$$

En este escenario, el sistema de orden fraccional tiene una única función PWL, es decir, $\varphi = 1$. Se Observa que $f_1(x_1)$ contiene tres segmentos, es decir, $k_1 = 3$, por lo que el espacio $\mathbb{R}^3$ se particiona en tres sub-dominios: $\Phi_1 = H_1^1$, $\Phi_2 = H_2^1$ y $\Phi_3 = H_3^1$ con límites s_1^1 y s_2^1. Además, las pendientes e intercepciones de cada recta se expresan como $M^1 = \left[m_1^1 x_1, m_2^1 x_1, m_3^1 x_1\right]$ y $C^1 = \left[c_1^1, c_2^1, c_3^1\right]$, y conforman las matrices

$$\mathcal{M} = \begin{bmatrix} m_1^1 x_1 & m_2^1 x_1 & m_3^1 x_1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathcal{C} = \begin{bmatrix} c_1^1 & c_2^1 & c_3^1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{3.2.4}$$

Una consecuencia clara del resultado anterior es que los vectores columna se obtienen de la siguiente manera:

$$\mathcal{M}_{(*,1)} = \begin{bmatrix} m_1^1 x_1 \\ 0 \\ 0 \end{bmatrix}, \mathcal{M}_{(*,2)} = \begin{bmatrix} m_2^1 x_1 \\ 0 \\ 0 \end{bmatrix}, \mathcal{M}_{(*,3)} = \begin{bmatrix} m_3^1 x_1 \\ 0 \\ 0 \end{bmatrix}, \tag{3.2.5}$$

$$\mathcal{C}_{(*,1)} = \begin{bmatrix} c_1^1 \\ 0 \\ 0 \end{bmatrix}, \mathcal{C}_{(*,2)} = \begin{bmatrix} c_2^1 \\ 0 \\ 0 \end{bmatrix}, \mathcal{C}_{(*,3)} = \begin{bmatrix} c_3^1 \\ 0 \\ 0 \end{bmatrix}. \tag{3.2.6}$$

En este ejemplo, se considera solo los elementos $\mathcal{M}_{(i,\varsigma)}, \mathcal{C}_{(i,\varsigma)} \neq 0$, $\varsigma = 1, 2, 3$, $\forall i = 1, 2, 3$, y considerando que este sistema presenta un atractor caótico de 2 enrollamientos para $\alpha_{31} = 1.4$, $\alpha_{32} = 1$, $\alpha_{33} = 0.7$, $\beta_{31} = 2.1$, $m_1^1 = 0$, $m_2^1 = 6.5$, $m_3^1 = 0$, $c_1^1 = -6.5$, $c_2^1 = 0$, $c_3^1 = 6.5$, $s_1^1 = -1$, y $s_2^1 = 1$. La expresión (3.2.7) puede reescribirse

como:

$$\mathcal{M}_{(1,\varsigma)}, \mathcal{C}_{(1,\varsigma)} = \begin{cases} \mathcal{M}_{(1,1)} = 0, \quad \mathcal{C}_{(1,1)} = -6.5, & \text{if } x_1 \in H_1^1, \\ \mathcal{M}_{(1,2)} = 6.5x_1, \quad \mathcal{C}_{(1,2)} = 0, & \text{if } x_2 \in H_2^1, \\ \mathcal{M}_{(1,3)} = 0, \quad \mathcal{C}_{(1,3)} = 6.5, & \text{if } x_3 \in H_3^1. \end{cases} \tag{3.2.7}$$

Ahora, se procede a calcular la solución para los primeros cinco elementos de la serie de descomposición X^0, X^1, X^2, X^3 y X^4 en cada sub-dominio para obtener la solución del sistema:

$$X = \sum_{\ell=0}^{4} X^{\ell}. \tag{3.2.8}$$

Para evitar confusiones con las variables del sistema, se definen los coeficientes $X^0 = [\Delta_1^0, \Delta_2^0, \Delta_3^0]^T$, $X^1 = [\Delta_1^1, \Delta_2^1, \Delta_3^1]^T$, y sucesivamente hasta $X^4 = [\Delta_1^3, \Delta_2^3, \Delta_3^3]$. Usando la relación recursiva (3.1.14) en la solución PWL-DM (3.2.7) con condiciones iniciales $X(t_0^+) = \left[x_1(t_0), x_2(t_0), x_3(t_0)\right]^T$ y utilizando las propiedades fundamentales del operador $J_{t_0}^q$ (2.1.9) y (2.1.10), se proceder a calcular X^0 de la siguiente manera:

$$X^0 = \begin{cases} \Delta_1^0 = x_1(t_0), \\ \Delta_2^0 = x_2(t_0), \\ \Delta_3^0 = x_3(t_0) + \beta_{31}\mathcal{C}(1,\varsigma)\dfrac{(t-t_0)^q}{\Gamma(q+1)}, \end{cases} \tag{3.2.9}$$

Ahora, X^1 se obtiene aplicando la ecuacion $X^1 = J_{t_0}^q AX^0 + J_{t_0}^q B\mathcal{M}_{(*,2)}(X^0)$ de (3.1.14):, con la matrices A y B proveniente de (3.2.1):

$$X^1 = \begin{bmatrix} \Delta_1^1 = J_{t_0}^q(\Delta_2^0) = \Delta_2^0\dfrac{(t-t_0)^q}{\Gamma(q+1)}, \\ \Delta_2^1 = J_{t_0}^q(\Delta_3^0) = \Delta_3^0\dfrac{(t-t_0)^q}{\Gamma(q+1)}, \\ \Delta_3^1 = J_{t_0}^q\big(-\alpha_{31}\Delta_1^0 - \alpha_{32}\Delta_2^0 - \alpha_{33}\Delta_3^0 + \beta_{31}M_{(1,\varsigma)}(\Delta_1^0)\big) \\ \quad = \big(-\alpha_{31}\Delta_1^0 - \alpha_{32}\Delta_2^0 - \alpha_{33}\Delta_3^0 + \beta_{31}M_{(1,\varsigma)}(\Delta_1^0)\big)\dfrac{(t-t_0)^q}{\Gamma(q+1)}, \end{bmatrix} \tag{3.2.10}$$

de forma sucesiva, X^2, X^3 y X^4 se calculan de la siguiente forma:

$$X^2 = \begin{bmatrix} \Delta_1^2 = J_{t_0}^q(\Delta_2^1) = \Delta_2^1 \dfrac{(t-t_0)^q}{\Gamma(q+1)}, \\ \Delta_2^2 = J_{t_0}^q(\Delta_3^1) = \Delta_3^1 \dfrac{(t-t_0)^q}{\Gamma(q+1)}, \\ \Delta_3^2 = J_{t_0}^q\big(-\alpha_{31}\Delta_1^1 - \alpha_{32}\Delta_2^1 - \alpha_{33}\Delta_3^1 + \beta_{31}M_{(1,\varsigma)}(\Delta_1^1)\big) \\ \quad = \big(-\alpha_{31}\Delta_1^1 - \alpha_{32}\Delta_2^1 - \alpha_{33}\Delta_3^1 + \beta_{31}M_{(1,\varsigma)}(\Delta_1^1)\big)\dfrac{(t-t_0)^q}{\Gamma(q+1)}, \end{bmatrix} \tag{3.2.11}$$

$$X^3 = \begin{bmatrix} \Delta_1^3 = J_{t_0}^q(\Delta_2^2) = \Delta_2^2 \dfrac{(t-t_0)^q}{\Gamma(q+1)}, \\ \Delta_2^3 = J_{t_0}^q(\Delta_3^2) = \Delta_3^2 \dfrac{(t-t_0)^q}{\Gamma(q+1)}, \\ \Delta_3^3 = J_{t_0}^q\big(-\alpha_{31}\Delta_1^2 - \alpha_{32}\Delta_2^2 - \alpha_{33}\Delta_3^2 + \beta_{31}M_{(1,\varsigma)}(\Delta_1^2)\big) \\ \quad = \big(-\alpha_{31}\Delta_1^2 - \alpha_{32}\Delta_2^2 - \alpha_{33}\Delta_3^2 + \beta_{32}M_{(1,\varsigma)}(\Delta_1^2)\big)\dfrac{(t-t_0)^q}{\Gamma(q+1)}, \end{bmatrix} \tag{3.2.12}$$

$$X^4 = \begin{bmatrix} \Delta_1^4 = J_{t_0}^q(\Delta_2^3) = \Delta_2^3 \dfrac{(t-t_0)^q}{\Gamma(q+1)}, \\ \Delta_2^4 = J_{t_0}^q(\Delta_3^3) = \Delta_3^3 \dfrac{(t-t_0)^q}{\Gamma(q+1)}, \\ \Delta_3^4 = J_{t_0}^q\big(-\alpha_{31}\Delta_1^3 - \alpha_{32}\Delta_2^3 - \alpha_{33}\Delta_3^3 + \beta_{31}M_{(1,\varsigma)}(\Delta_1^3)\big) \\ \quad = \big(-\alpha_{31}\Delta_1^3 - \alpha_{32}\Delta_2^3 - \alpha_{33}\Delta_3^3 + \beta_{32}M_{(1,\varsigma)}(\Delta_1^3)\big)\dfrac{(t-t_0)^q}{\Gamma(q+1)}, \end{bmatrix} \tag{3.2.13}$$

Finalmente, en la Figura 3.1 se muestran los resultados de la simulación numérica obtenidos a partir del método propuesto (3.2.7), con un tamaño de paso $h = (t-t_0) = 0.01$, condiciones iniciales $\big[0.1, 0.1, 0.1\big]^T$, y un orden fraccional $q = 0.92$.
Los resultados obtenidos son consistentes con los resultados reportados en las referencias [165, 166], demostrando la utilidad del método de descomposición propuesto. Para explicar este punto. Tavazoei y Haeri [167] indicaron que una condición necesaria para la existencia del atractor de 2-scroll en sistemas de orden fraccional es que $|\arg(\mathrm{eig}(A))| > q\pi/2$. Para el sistema de orden fraccional (3.2.1)-(3.2.3), esta condición se cumple para $q = 0.92$. Además, en sistemas caóticos, se demostró que si el sistema tiene un atractor de dos enrollamientos, uno de sus puntos de equilibrio es el punto silla de índice 1, y los otros son puntos silla de índice 2 [167]. Los enrollamientos se generan solo alrededor de los puntos silla de índice 2, mientras que los puntos

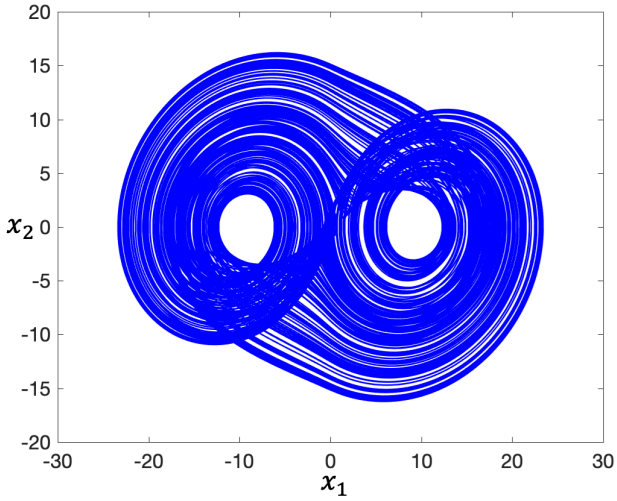


Figura 3.1: Simulación numérica del atractor caótico de 2 enrollamientos con un orden fraccional $q = 0.92$ utilizando PWL-DM.

silla de índice 1 solo se encargan de conectar los scrolls. Para el sistema subyacente (3.2.1)-(3.2.3), los puntos de equilibrio son $E_1 = (0, 0, 0)$, $E_2 = (\beta_{31}c_1^1)/\alpha_{31}, 0, 0)$, $E_3 = (\beta_{31}c_3^1)/\alpha_{31}, 0, 0)$, lo cual satisface las condiciones previas [166].

3.2.2. Ejemplo 2: Sistema caótico de múltiples enrollamientos de orden fraccional

En esta sección, se utilizará el método propuesto para abordar un sistema caótico de orden fraccional con múltiples enrollamientos en 2D. El enfoque DM-PWL se aplicará para resolver este sistema caótico, que presenta una estructura matemática más compleja, validando así la capacidad del método para manejar sistemas con dinámicas caóticas más complejas.

Considere un sistema dinámico de orden fraccional dado por (3.1.1) en $\mathbb{R}^3$ con operadores:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\alpha_{31} & -\alpha_{32} & -\alpha_{33} \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & -\beta_{12} & 0 \\ 0 & 0 & 0 \\ \beta_{31} & \beta_{32} & 0 \end{bmatrix}, \qquad G = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \tag{3.2.14}$$

donde α_{31}, α_{32}, α_{33}, β_{12}, β_{31} y β_{32} son constantes reales positivas. En este caso, el

sistema posee un par de funciones PWL f_1 y f_2, expresadas en la forma de la ecuación (3.1.4) como:

$$f_1(x_1) = \begin{cases} m_1^1 x_1 + c_1^1, & \text{if} \quad x_1 \in H_1^1 = \{x_1 < s_1^1\}, \\ m_2^1 x_1 + c_2^1, & \text{if} \quad x_1 \in H_2^1 = \{s_1^1 \leq x_1 < s_2^1\}, \\ m_3^1 x_1 + c_3^1 & \text{if} \quad x_1 \in H_3^1 = \{s_2^1 \leq x_1 < s_3^1\}, \\ m_4^1 x_1 + c_4^1, & \text{if} \quad x_1 \in H_4^1 = \{s_3^1 \leq x_1 < s_4^1\}, \\ m_5^1 x_1 + c_5^1, & \text{if} \quad x_1 \in H_5^1 = \{s_4^1 \leq x_1 < s_5^1\}, \\ m_6^1 x_1 + c_6^1, & \text{if} \quad x_1 \in H_6^1 = \{s_5^1 \leq x_1 < s_6^1\}, \\ m_7^1 x_1 + c_7^1, & \text{if} \quad x_1 \in H_7^1 = \{s_6^1 \leq x_1\}, \end{cases}$$

$$f_2(x_2) = \begin{cases} m_1^2 x_2 + c_1^2, & \text{if} \quad x_2 \in H_1^2 = \{x_2 < s_1^2\}, \\ m_2^2 x_2 + c_2^2, & \text{if} \quad x_2 \in H_2^2 = \{s_1^2 \leq x_2 < s_2^2\}, \\ m_3^2 x_2 + c_3^2 & \text{if} \quad x_2 \in H_3^2 = \{s_2^2 \leq x_2 < s_3^2\}, \\ m_4^2 x_2 + c_4^2, & \text{if} \quad x_2 \in H_4^2 = \{s_3^2 \leq x_2 < s_4^2\}, \\ m_5^2 x_2 + c_5^2, & \text{if} \quad x_2 \in H_5^2 = \{s_4^2 \leq x_2 < s_5^2\}, \\ m_6^2 x_2 + c_6^2, & \text{if} \quad x_2 \in H_6^2 = \{s_5^2 \leq x_2 < s_6^2\}, \\ m_7^2 x_2 + c_7^2, & \text{if} \quad x_2 \in H_7^2 = \{s_6^2 \leq x_2\}. \end{cases} \tag{3.2.15}$$

En consecuencia, el vector U en la ecuación (3.1.2) cambia entre p_1 subdominios, donde p_1 se obtiene utilizando el Teorema 1 . Por lo tanto, la forma del vector U está dada por:

$$U(X) = \begin{cases} U_1(X), & \text{if } X \in \Phi_1, \\ U_2(X), & \text{if } X \in \Phi_2, \\ \vdots & \\ U_{p_1}(X), & \text{if } X \in \Phi_{p_1}. \end{cases} \tag{3.2.16}$$

Asociado con este resultado, los vectores U_ς en la ecuación (3.1.3) con $\varsigma = 1, 2, \ldots, p_1$, están definidos por:

$$\begin{bmatrix} u_{\varsigma 1} \\ u_{\varsigma 2} \\ u_{\varsigma 3} \end{bmatrix} = \begin{bmatrix} f_1(x_1) \\ f_2(x_2) \\ 0 \end{bmatrix}. \tag{3.2.17}$$

Al elegir el siguiente conjunto de parámetros del sistema $\alpha_{31} = 0.6, ; \alpha_{32} = 0.58, ; \alpha_{33} = 0.58, ; \beta_{12} = 1, ; \beta_{31} = 0.6, ; \beta_{32} = 0.6$, $m_1^1 = m_1^2 = 0$, $m_2^1 = m_2^2 = 10$, $m_3^1 = m_3^2 = 0$, $m_4^1 = m_4^2 = 10$, $m_5^1 = m_5^2 = 0$, $m_6^1 = m_6^2 = 10$, $m_7^1 = m_7^2 = 0$, $c_1^1 = c_1^2 = -30$, $c_2^1 = c_2^2 = 180$, $c_3^1 = c_3^2 = -10$, $c_4^1 = c_4^2 = 0$, $c_5^1 = c_5^2 = 10$, $c_6^1 = c_6^2 = -180$, $c_7^1 = c_7^2 = 30$, $s_1^1 = s_1^2 = -21$, $s_2^1 = s_2^2 = -19$, $s_3^1 = s_3^2 = -1$, $s_4^1 = s_4^2 = 1$, $s_5^1 = s_5^2 = 19$, $s_6^1 = s_6^2 = 21$, el sistema genera un atractor caótico en una orientación

2D de 4×4 enrollamientos [165, 166].

Debido a que el sistema PWL de orden fraccional tiene un par de funciones PWL, se aplica el procedimiento generalizado del Teorema 1 para derivar la partición finita del espacio de fases $\mathbb{R}^3$, y posteriormente obtenter las matrices $\mathcal{M}$ y $\mathcal{C}$, respectivamente.

Primero, se observa que el número total de subdominios utilizando $p_1 = \prod_{i=1}^{\varphi} k_i$ con $\varphi = 2$, $k_1 = 7$, y $k_2 = 7$, es $p_1 = 49$. Por lo tanto, el espacio de fases se particiona considerando los subdominios generados por cada función PWL de la siguiente manera: $H = H^1, H^2$, donde $H^1 = H_1^1, H_2^1, H_3^1, H_4^1, H_5^1, H_6^1, H_7^1$, y $H^2 = H_1^2, H_2^2, H_3^2, H_4^2, H_5^2, H_6^2, H_7^2$.

De esta manera, los p_1 subdominios en la partición finita Φ_ς se determinan mediante el producto cartesiano $(H^1 \times H^2)$ de todos los pares ordenados posibles, es decir, $\Phi_\varsigma = [H_1^1 \cap H_1^2, H_1^1 \cap H_2^2, H_1^1 \cap H_3^2, \ldots, H_7^1 \cap H_5^2, H_7^1 \cap H_6^2, H_7^1 \cap H_7^2]$ para $\varsigma = 1, 2, \ldots, 49$.

De acuerdo con el procedimiento descrito en la sección 3.1, las matrices $\mathcal{M}$ y $\mathcal{C}$ tienen la siguiente forma:

$$\mathcal{M} = \begin{bmatrix} m_1^1 x_1 \ldots m_1^1 x_1 & m_2^1 x_1 \ldots m_2^1 x_1 & m_3^1 x_1 \ldots m_3^1 x_1 & m_4^1 x_1 \ldots m_4^1 x_1 & m_5^1 x_1 \ldots m_5^1 x_1 & m_6^1 x_1 \ldots m_6^1 x_1 & m_7^1 x_1 \ldots m_7^1 x_1 \\ m_1^2 x_2 \ldots m_7^2 x_2 & m_1^2 x_2 \ldots m_7^2 x_2 & m_1^2 x_2 \ldots m_7^2 x_2 & m_1^2 x_2 \ldots m_7^2 x_2 & m_1^2 x_2 \ldots m_7^2 x_2 & m_1^2 x_2 \ldots m_7^2 x_2 & m_1^2 x_2 \ldots m_7^2 x_2 \\ 0 & & & \ldots & & & 0 \end{bmatrix} \tag{3.2.18}$$

$$\mathcal{C} = \begin{bmatrix} c_1^1 \ldots c_1^1 & c_2^1 \ldots c_2^1 & c_3^1 \ldots c_3^1 & c_4^1 \ldots c_4^1 & c_5^1 \ldots c_5^1 & c_6^1 \ldots c_6^1 & c_7^1 \ldots c_7^1 \\ c_1^2 \ldots c_7^2 & c_1^2 \ldots c_7^2 & c_1^2 \ldots c_7^2 & c_1^2 \ldots c_7^2 & c_1^2 \ldots c_7^2 & c_1^2 \ldots c_7^2 & c_1^2 \ldots c_7^2 \\ 0 & & & \ldots & & & 0 \end{bmatrix} \tag{3.2.19}$$

Una vez que se han determinado las matrices $\mathcal{C}$ y $\mathcal{M}$, la solucion de este sistema se obtiene mediante la ecuación (3.1.12), donde los vectores columna $\mathcal{M}(*, \varsigma)$ y $\mathcal{C}(*, \varsigma)$ se definen de la siguiente forma

$$\mathcal{M}_{(*,\varsigma)},\mathcal{C}_{(*,\varsigma)} = \begin{cases} \mathcal{M}_{(*,1},\mathcal{C}_{(*,1)}, & \text{if} \quad x_1, x_2 \in H_1^1 \cap H_1^2, \\ \mathcal{M}_{(*,2},\mathcal{C}_{(*,2)}, & \text{if} \quad x_1, x_2 \in H_1^1 \cap H_2^2, \\ \qquad\qquad \vdots & \qquad\qquad \vdots \\ \mathcal{M}_{(*,24},\mathcal{C}_{(*,24)}, & \text{if} \quad x_1, x_2 \in H_4^1 \cap H_3^2, \\ \mathcal{M}_{(*,25},\mathcal{C}_{(*,25)}, & \text{if} \quad x_1, x_2 \in H_4^1 \cap H_4^2, \\ \mathcal{M}_{(*,26},\mathcal{C}_{(*,26)}, & \text{if} \quad x_1, x_2 \in H_4^1 \cap H_5^2, \\ \qquad\qquad \vdots & \qquad\qquad \vdots \\ \mathcal{M}_{(*,48},\mathcal{C}_{(*,48)}, & \text{if} \quad x_1, x_2 \in H_7^1 \cap H_6^2, \\ \mathcal{M}_{(*,49},\mathcal{C}_{(*,49)}, & \text{if} \quad x_1, x_2 \in H_7^1 \cap H_7^2. \end{cases} \tag{3.2.20}$$

Los elementos $X^0 ... X^4$ en (3.1.14) se calculan de forma similar al ejemplo anterior. Se considera solo los elementos $\mathcal{M}_{(i,\varsigma)},\mathcal{C}_{(i,\varsigma)} \neq 0$, $\varsigma = 1, 2, 3,\ \forall i = 1, 2, 3$
Para evitar confusiones con las variables del sistema, se definen los coeficientes $X^0 = [\Delta_1^0, \Delta_2^0, \Delta_3^0]^T$, $X^1 = [\Delta_1^1, \Delta_2^1, \Delta_3^1]^T$, y sucesivamente hasta $X^4 = [\Delta_1^3, \Delta_2^3, \Delta_3^3]$. Usando la relación recursiva (3.1.14) en la solución PWL-DM (3.2.7) con condiciones iniciales $X(t_0^+) = \left[x_1(t_0), x_2(t_0), x_3(t_0)\right]^T$ y utilizando las propiedades fundamentales del operador $J_{t_0}^q$ (2.1.9) y (2.1.10), se proceder a calcular X^0 de la siguiente manera:

$$X^0 = \begin{cases} \Delta_1^0 = x_1(t_0) - \beta_{12}\mathcal{C}(2,\varsigma)\dfrac{(t-t_0)^q}{\Gamma(q+1)}, \\ \Delta_2^0 = x_2(t_0), \\ \Delta_3^0 = x_3(t_0) + \beta_{31}\mathcal{C}(1,\varsigma)\dfrac{(t-t_0)^q}{\Gamma(q+1)} + \beta_{32}\mathcal{C}(2,\varsigma)\dfrac{(t-t_0)^q}{\Gamma(q+1)}, , \end{cases} \tag{3.2.21}$$

Ahora, X^1 se obtiene aplicando la ecuacion $X^1 = J_{t_0}^q AX^0 + J_{t_0}^q B\mathcal{M}_{(*,2)}(X^0)$ de (3.1.14):, con la matrices A y B proveniente de (3.2.1):

$$X^1 = \begin{bmatrix} \Delta_1^1 = J_{t_0}^q\big(\Delta_2^0 - \beta_{12}\mathcal{M}(2,\varsigma)(\Delta_2^0)\big) = \big(\Delta_2^0 - \beta_{12}\mathcal{M}(2,\varsigma)(\Delta_2^0)\big)\dfrac{(t-t_0)^q}{\Gamma(q+1)} \\ \Delta_2^1 = J_{t_0}^q(\Delta_3^0) = \Delta_3^0\dfrac{(t-t_0)^q}{\Gamma(q+1)} \\ \Delta_3^1 = J_{t_0}^q\big(-\alpha_{31}\Delta_1^0 - \alpha_{32}\Delta_2^0 - \alpha_{33}\Delta_3^0 + \beta_{31}M_{(1,\varsigma)}(\Delta_1^0) + \beta_{32}M_{(2,\varsigma)}(\Delta_2^0)\big) \\ \quad = \big(-\alpha_{31}\Delta_1^0 - \alpha_{32}\Delta_2^0 - \alpha_{33}\Delta_3^0 + \beta_{31}M_{(1,\varsigma)}(\Delta_1^0) + \beta_{32}M_{(2,\varsigma)}(\Delta_2^0)\big)\dfrac{(t-t_0)^q}{\Gamma(q+1)}, \end{bmatrix} \tag{3.2.22}$$

de forma sucesiva, X^2, X^3 y X^4 se calculan de la siguiente forma:

$$X^2 = \begin{bmatrix} \Delta_1^2 = J_{t_0}^q\big(\Delta_2^1 - \beta_{12}\mathcal{M}(2,\varsigma)(\Delta_2^1)\big) = \big(\Delta_2^1 - \beta_{12}\mathcal{M}(2,\varsigma)(\Delta_2^1)\big)\dfrac{(t-t_0)^q}{\Gamma(q+1)} \\ \Delta_2^2 = J_{t_0}^q(\Delta_3^1) = \Delta_3^1\dfrac{(t-t_0)^q}{\Gamma(q+1)} \\ \Delta_3^2 = J_{t_0}^q\big(-\alpha_{31}\Delta_1^1 - \alpha_{32}\Delta_2^1 - \alpha_{33}\Delta_3^1 + \beta_{31}M_{(1,\varsigma)}(\Delta_1^1) + \beta_{32}M_{(2,\varsigma)}(\Delta_2^1)\big) \\ \quad = \big(-\alpha_{31}\Delta_1^1 - \alpha_{32}\Delta_2^1 - \alpha_{33}\Delta_3^1 + \beta_{31}M_{(1,\varsigma)}(\Delta_1^1) + \beta_{32}M_{(2,\varsigma)}(\Delta_2^1)\big)\dfrac{(t-t_0)^q}{\Gamma(q+1)}, \end{bmatrix} \tag{3.2.23}$$

$$X^3 = \begin{bmatrix} \Delta_1^3 = J_{t_0}^q\big(\Delta_2^2 - \beta_{12}\mathcal{M}(2,\varsigma)(\Delta_2^2)\big) = \big(\Delta_2^2 - \beta_{12}\mathcal{M}(2,\varsigma)(\Delta_2^2)\big)\dfrac{(t-t_0)^q}{\Gamma(q+1)} \\ \Delta_2^3 = J_{t_0}^q(\Delta_3^2) = \Delta_3^2\dfrac{(t-t_0)^q}{\Gamma(q+1)} \\ \Delta_3^3 = J_{t_0}^q\big(-\alpha_{31}\Delta_1^2 - \alpha_{32}\Delta_2^2 - \alpha_{33}\Delta_3^2 + \beta_{31}M_{(1,\varsigma)}(\Delta_1^2) + \beta_{32}M_{(2,\varsigma)}(\Delta_2^2)\big) \\ \quad = \big(-\alpha_{31}\Delta_1^2 - \alpha_{32}\Delta_2^2 - \alpha_{33}\Delta_3^2 + \beta_{31}M_{(1,\varsigma)}(\Delta_1^2) + \beta_{32}M_{(2,\varsigma)}(\Delta_2^2)\big)\dfrac{(t-t_0)^q}{\Gamma(q+1)}, \end{bmatrix} \tag{3.2.24}$$

$$X^4 = \begin{bmatrix} \Delta_1^4 = J_{t_0}^q\big(\Delta_2^3 - \beta_{12}\mathcal{M}(2,\varsigma)(\Delta_2^3)\big) = \big(\Delta_2^3 - \beta_{12}\mathcal{M}(2,\varsigma)(\Delta_2^3)\big)\dfrac{(t-t_0)^q}{\Gamma(q+1)} \\ \Delta_2^4 = J_{t_0}^q(\Delta_3^3) = \Delta_3^3\dfrac{(t-t_0)^q}{\Gamma(q+1)} \\ \Delta_3^4 = J_{t_0}^q\big(-\alpha_{31}\Delta_1^3 - \alpha_{32}\Delta_2^3 - \alpha_{33}\Delta_3^3 + \beta_{31}M_{(1,\varsigma)}(\Delta_1^3) + \beta_{32}M_{(2,\varsigma)}(\Delta_2^3)\big) \\ \quad = \big(-\alpha_{31}\Delta_1^3 - \alpha_{32}\Delta_2^3 - \alpha_{33}\Delta_3^3 + \beta_{31}M_{(1,\varsigma)}(\Delta_1^3) + \beta_{32}M_{(2,\varsigma)}(\Delta_2^3)\big)\dfrac{(t-t_0)^q}{\Gamma(q+1)}, \end{bmatrix} \tag{3.2.25}$$

La Figura 3.2 muestra el atractor caótico con un paso de integración $h = 0.01$, $\big[x_1(t_0), x_2(t_0), x_3(t_0)\big]^T = \big[1, 0, 1\big]^T$ y un orden fraccional $q = 0.93$. Como era de

esperarse, el atractor caótico resultante es consistente con la teoría de las referencias [165, 166].

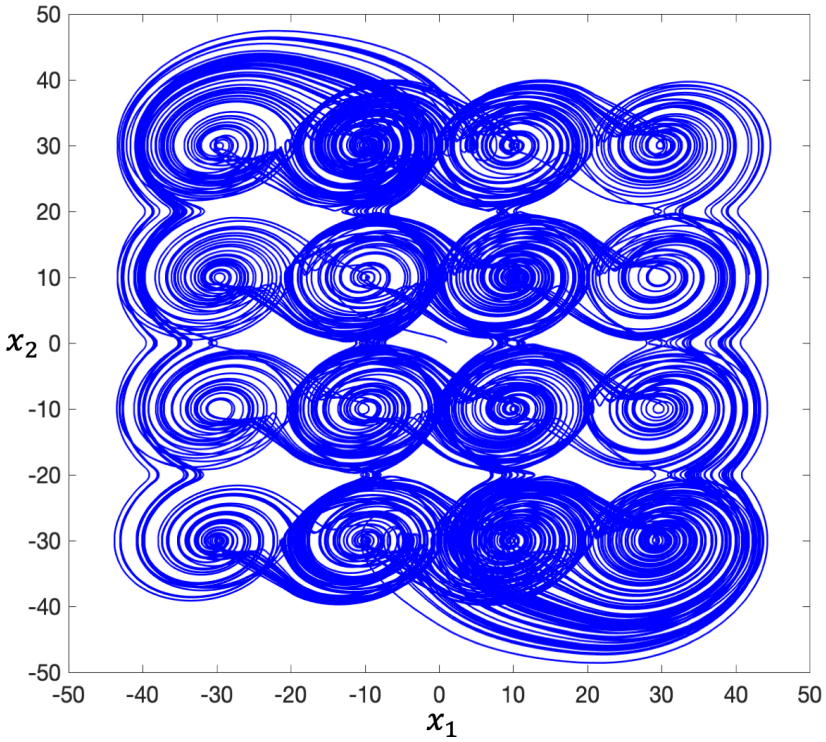


Figura 3.2: Simulación numérica del atractor caótico de orientación 2D de 4×4 enrollamientos con un orden fraccional $q = 0.93$.

Capítulo 4

Desarrollo experimental

En este capítulo se detalla la síntesis e implementación de los sistemas abordados en el presente trabajo de tesis. Como se mencionó en el Capítulo 1, el enfoque digital se centra en el desarrollo de una arquitectura basada en FPGA, cuyo propósito es encontrar un equilibrio adecuado entre el nivel de abstracción y el uso eficiente de los recursos disponibles. Este balance es crucial para lograr un rendimiento óptimo en los sistemas caóticos propuestos, sin comprometer la escalabilidad o la capacidad de implementación en plataformas con recursos limitados.

A lo largo de este capítulo, se analizarán las decisiones clave en el proceso de diseño, tales como la elección de la arquitectura digital, la representación numérica de los sistemas, y los pasos de síntesis en FPGA. Estas decisiones están orientadas a maximizar la eficiencia en el uso del hardware, minimizando el consumo de recursos sin sacrificar la precisión y el rendimiento de los sistemas implementados.

4.1. Procedimiento Estadar para la Síntesis e Implementación en FPGA de Sistemas Caóticos de Orden Fraccional

En esta sección se presenta una metodología de diseño sobre cómo implementar sistemas caóticos de orden fraccional utilizando plataformas FPGA [168].
En el diagrama de flujo de la figura 4.1 se describe esta metodología. Primero, se simula el sistema caótico de orden fraccional, esto permite obtener el rango de los valores numéricos. Con esta información, se calcula el número de bits necesario para representar dichos valores, empleando la representación en formato de punto fijo $Q_{I,nI}$.

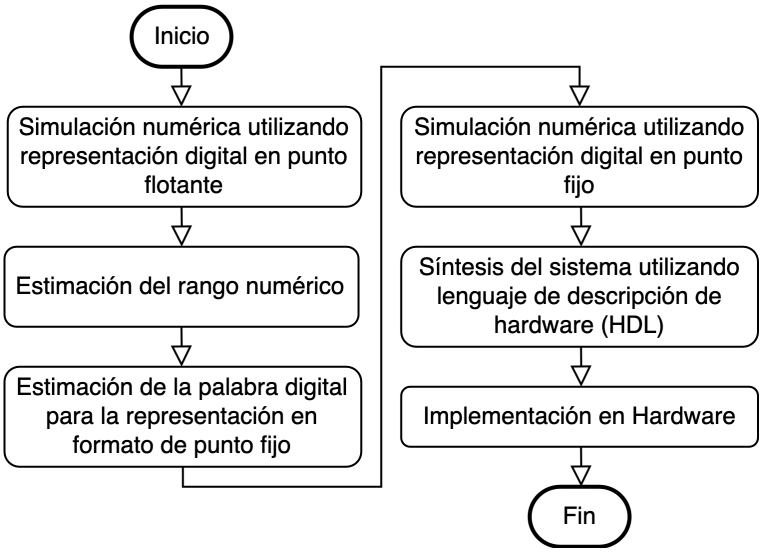


Figura 4.1: Procedimiento estándar para implementaciones en FPGA de sistemas caóticos de orden entero y orden fraccional.

En este formato, I corresponde al número de bits destinados a la representación de la parte entera más el bit de signo, y nI se refiere al número de bits para la representación de la parte fraccionaria. Posteriormente, la estimación del número de bits obtenida en el paso anterior se valida mediante simulación numérica. Una vez validados los resultados, se procede a traducir la solución del sistema caótico de orden fraccionario a hardware. El bloque de "Síntesis del sistema...", puede contener cualquiera de los métodos de solución analizados en el capítulo 2, este proceso consiste en utilizar algún lenguaje de descripción de hardware como Verilog o VHDL para generar el diseño diseño digital a partir de celdas lógicas y registros. Finalmente, en el ultimo paso Ïmplementación en Hardware"se asigna el diseño generado al chip FPGA.

4.1.1. Síntesis del sistema caótico PWL de orden fraccional con 2 enrollamientos

En esta sección se llevará a cabo la Síntesis de sistemas caóticos PWL de orden fraccional bajo el procedimiento estandar. Esta sección presta atención de forma particular a la implementación del **sistema caótico PWL de orden fraccional con 2 enrollamientos**. Se ha seleccionado este sistema para explicar paso a paso el procedimiento estandar.

4.1.1.1. Estimación de la palabra digital

Esta sección aborda los puntos 2 y 3 del diagrama de flujo 4.1.
El rango de cada variable de estado puede estimarse a partir de los datos obtenidos en la simulación del sistema, la cual se detalla en la sección 3.2.1. Este proceso de estimación es fundamental, ya que, con base en los límites de las variables de estado, se pueden estimar los valores máximos y mínimos que se alcanzarán dentro del algoritmo utilizado para la computación numérica del sistema. Determinar estos límites es crucial para garantizar que las variables caóticas se representen adecuadamente en el sistema de cálculo y evitar posibles errores de truncamiento o saturación durante la implementación.

Con base en la simulación, se ha determinado que los rangos estimados para cada variable de estado son los siguientes:

- $x_1 \in [-23, 24]$: Este valor indica que la primera variable de estado oscila entre -23 y 24.
- $x_2 \in [-19, 19]$: Este valor indica que la segunda variable de estado oscila entre -19 y 19.
- $x_3 \in [-19, 19]$: La tercera variable de estado tiene un comportamiento similar al de x_2, oscilando en el mismo rango.

Por conveniencia en la claridad de la evaluación, la estructura matemática del sistema PWL de orden fraccional se expresa de la siguiente forma:

$$ {}^{C}D_{t_0}^{q} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ -\alpha_{31}x_1 - \alpha_{32}x_2 - \alpha_{33}x_3 + \beta_{31}f_1(x_1) \end{bmatrix}, \tag{4.1.1} $$

A continuación, el conjunto de ecuaciones que describen al sistema caótico PWL de orden fraccional, se evalúa en los puntos $(x_1, x_2, x_3) = (1, -19, 19)$, los cuales representan una selección conveniente dentro de los rangos previamente estimados.

$$ {}^{C}D_{t_0}^{q} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -19 \\ -19 \\ -\alpha_{31}(1) - \alpha_{32}(-19) - x_3(-19) + \beta_{31}f_1(1) = 44.55 \end{bmatrix}, \tag{4.1.2} $$

A partir del punto anterior, se observa que el valor entero máximo a representar es 44. Entonces, el número de bits necesario se puede estimar de la siguiente manera:

$$\lceil \log 2(44) \rceil = 6, \tag{4.1.3}$$

donde $\lceil \cdot \rceil$ denota la función techo (ceiling). Por lo tanto, el número de bits necesario es de $6 + 1$ (más el bit de signo). Para una precisión de 32 bits, los 25 bits restantes se pueden utilizar para la representación de la parte no entera, por lo que el formato en punto fijo se expresa como $Q_{7,25}$.

4.1.1.2. Simulación numérica en punto fijo

La simulación numérica del sistema en punto fijo se realiza en el lenguaje de programación C.
La Figura 3.1 muestra los resultados de la simulación numérica. Los retratos de fase observados coinciden con los obtenidos en la Figura 4.2, ya que convergen al mismo atractor caótico. Por lo tanto, el formato en punto fijo elegido es adecuado para el diseño en FPGA. Cabe remarcar que este tipo de verificación es fundamental para

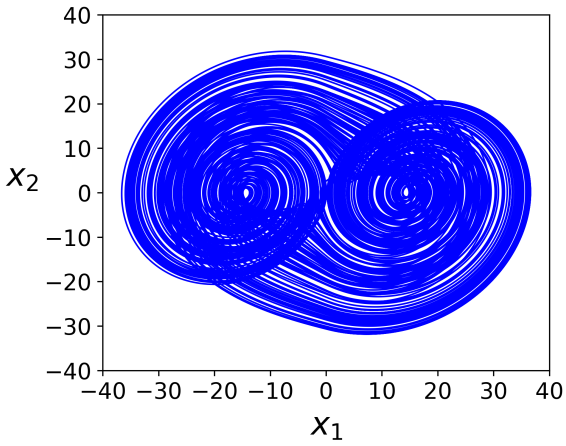


Figura 4.2: Simulación con punto fijo del atractor caótico de 2 enrollamientos con un orden fraccional $q = 0.92$ utilizando PWL-DM.

evitar reconfiguraciones innecesarias en etapas posteriores del diseño. Las reconfiguraciones del hardware suelen ser costosas tanto en tiempo como en recursos, y pueden retrasar el proceso de implementación. Por lo tanto, al realizar esta verificación previa, se asegura que el diseño del sistema en FPGA sea óptimo desde el inicio, minimizando posibles ajustes y garantizando una mayor eficiencia en la implementación final.

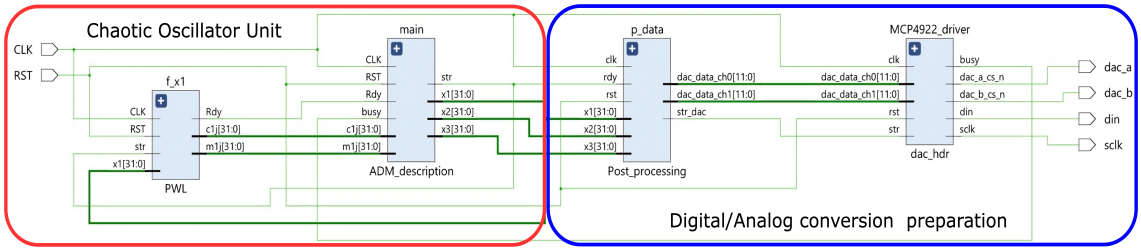


Figura 4.3: Esquemas RTL del sistema caótico PWL de orden fraccional con 2 enrollamientos en FPGA.

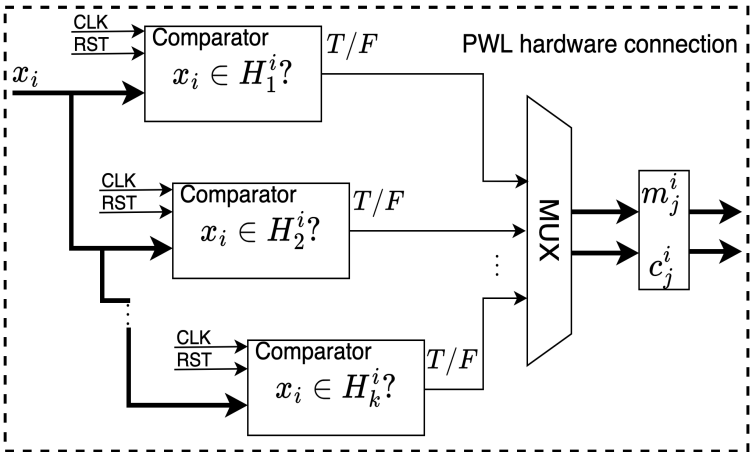


Figura 4.4: Interconexión de hardware de alto nivel para las funciones PWL $f_i(\cdot)$ (3.1.4).

4.1.1.3. Síntesis del sistema utilizando lenguaje de descripción de hardware

La Síntesis del sistema se realiza en VHDL. La figura 4.3 ilustra el diagrama general RTL del diseño digital para la implementación del atractor caótico de 2 enrollamientos, el cual esta compuesto de la siguiente forma:

1.- **Chaotic Oscillator Unit**: Los elementos de la implementación en FPGA para el atractor de $\times 2$ enrollamientos está marcado en color rojo. El nivel más alto del diseño en FPGA para este atractor comprende dos bloques, un bloque etiquetado como `PWL` y otro etiquetado como `ADM_description`, cada uno de los cuales realiza cálculos de manera separada.
En la figura 4.5 se presenta un esquema detallado sobre el mecanismo para la generación de los atractores caóticos de dos enrollamientos de orden fraccional en FPGA.

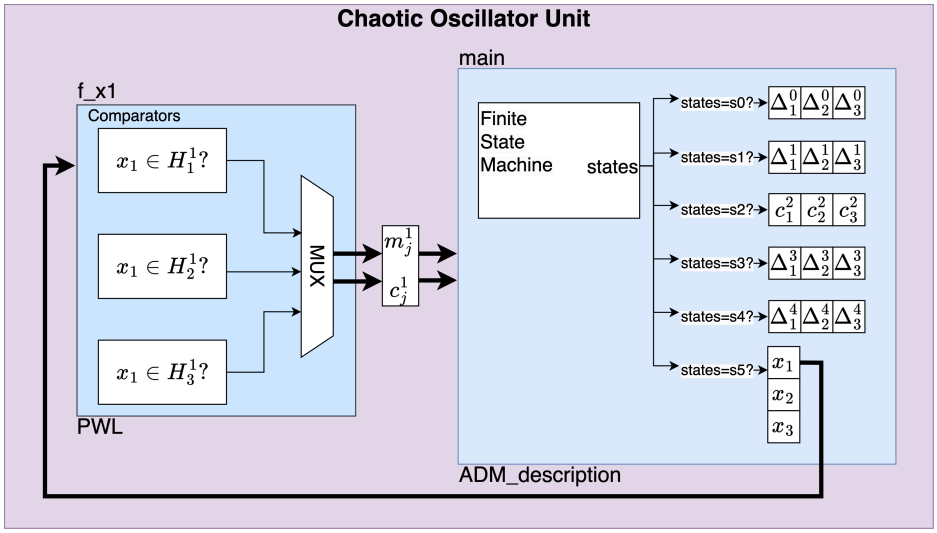


Figura 4.5: Diagrama de bloques del mecanismo de generación del atractor caótico de 2 enrollamientos.

1.1 **Comparación de Señal**. El proceso comienza con el generador de funciones PWL `f_x1`, que alimenta la señal en el bloque de `Comparadores`. Este bloque contiene varias unidades de comparación que determinan si la señal x_1 pertenece a regiones específicas H_1^1, H_2^1 y H_3^1. Con el fin de ejemplificar cómo se realiza la descripción e implementación de las funciones PWL, como la utilizada en este sistema (3.2.2), se presenta la Figura 4.4. Esta Figura ofrece una visión general del proceso mediante el cual se lleva a cabo la partición de la señal de entrada en distintas regiones y la selección de los parámetros correspondientes. Estos bloques, encargados de las comparaciones y de la selección de coeficientes, permiten implementar de manera eficiente las funciones PWL en hardware, facilitando su aplicación en sistemas caóticos.

1.1.1 **Comparator** (comparador). Los comparadores en la parte izquierda del diagrama tienen la función de determinar si la señal de entrada x_i pertenece a una región específica H_1^i, H_2^i, ..., H_k^i. Estos bloques reciben la señal x_i, el reloj (CLK) y la señal de reinicio (RST). Para cada región, el comparador devuelve un valor verdadero (T) o falso (F) dependiendo de si la señal de entrada x_i está dentro de la región predeterminada.

1.1.2 **Mux** (multiplexor). El multiplexor selecciona los parámetros adecuados m_j^i y c_j^i basándose en los resultados de los comparadores. Estos

parámetros son cruciales para la descripción de la función PWL. Los valores seleccionados determinan las pendientes y constantes que caracterizan la función por tramos (PWL).

1.1.3 **Output** (salida). Las salidas m_j^i y c_j^i corresponden a los parámetros seleccionados que describen la función PWL. Los coeficientes m_j^i y c_j^i se utilizan para calcular los valores que definen los segmentos lineales de la función por tramos, dependiendo de la región en la que se encuentre x_i.

1.2 **Descripción PWL-DM**. Posterior a la etapa de comparación, el bloque `ADM_description` contiene la descripción de hardware del método PWL-DM (3.2.8) para resolver el sistema de orden fraccional. Los parámetros m_j^1 y c_j^1 se introducen en el bloque `ADM_description`, que utiliza una Máquina de Estados Finitos (FSM) que transiciona del estado $s0$ al estado $s5$. Cada estado procesa cálculos en paralelo correspondientes a un conjunto de coeficientes Δ_i^ℓ. En el estado $s5$, los resultados de los cálculos de los coeficientes Δ_i^ℓ se utilizan para calcular la solución del sistema y, por lo tanto, generar un atractor caótico de 2 enrollamientos.

2.- **Digital/Analog Conversion**: Para la visualización de los atractores caóticos en un osciloscopio, los bloques `Post_processing` y `dac_hdr` (cuadro azul en la Figura 4.2), controlan el convertidor D/A de doble canal MCP4922 de 12 bits a 10 MHz. Cabe recordar que ambos bloques pueden ser eliminados en aplicaciones prácticas, es decir, no deben considerarse como parte de la implementación final, ya que solo se utilizan con fines de observación experimental.

4.1.2. Síntesis del sistema caótico PWL de orden fraccional con 4×4 enrollamientos

En esta sección se presenta la síntesis del sistema caótico PWL de orden fraccional con 4×4 enrollamientos. De manera similar al caso anterior, esta síntesis se realiza siguiendo el procedimiento estándar. Por simplicidad, se han omitido los pasos 1 al 4 de la metodología presentada en la Figura 4.1. Estos pasos iniciales, que incluyen la simulación numérica, la estimación del rango de las variables y la validación del número de bits requeridos, han sido claramente expuestos en el ejemplo anterior y son

aplicables de manera similar a este caso, esta sección aborda únicamente la síntesis del sistema.

La Figura 4.6 ilustra el esquemático RTL del diseño digital. Similar al caso anterior

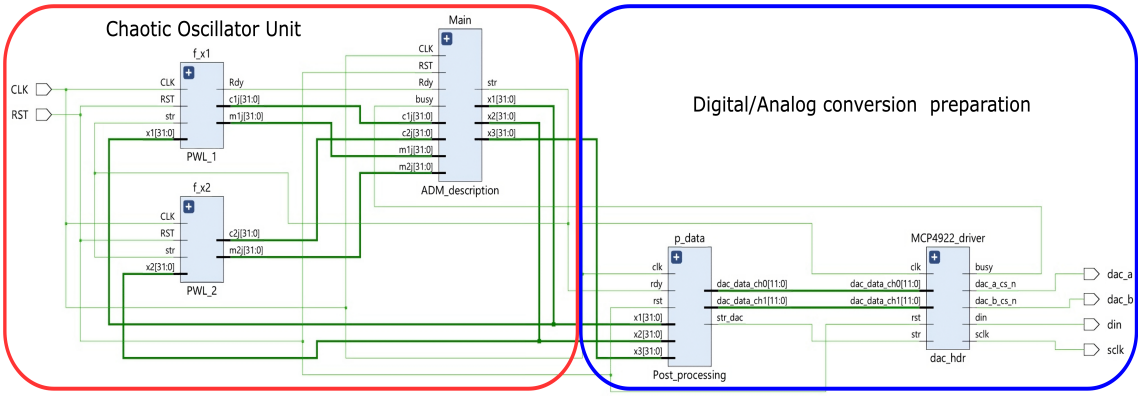


Figura 4.6: Esquemas RTL del sistema caótico PWL de orden fraccional con 4×4 enrollamientos en FPGA.

del sistema de 2 enrollamientos, este diseño se organiza en dos secciones principales: la sección marcada en rojo, que representan la parte de generación del atractor caótico y la sección marcada en azul, muestra los elementos dedicados a la conversión digital-analógica para la visualización del atractor caótico en un osciloscopio.

En la Figura 4.7, se muestra el mecanismo para la generación del atractor caótico con 4×4 enrollamientos. La generación de este atractor caótico presenta una complejidad mayor debido a la inclusión de dos señales de entrada, x_1 y x_2, y a la mayor cantidad de regiones a comparar (3.2.20). A continuación se detalla el proceso de síntesis para la generación del atractor caótico:

1.- **Comparators** (comparadores). El proceso comienza con las señales x_1 y x_2, las cuales se alimentan a sus respectivos bloques de `comparadores`. Cada señal se compara con un conjunto de regiones predefinidas, H_j^1 para x_1 y H_j^2 para x_2, donde $j = 1, 2, \cdots, 7$. Las unidades de comparación devuelven un valor verdadero (T) o falso (F) dependiendo de la región a la que pertenezca cada señal de entrada en ese momento.

2.- **Mux** (multiplexores) Cada una de las señales x_1 y x_2 cuenta con su propio multiplezor `MUX`, estos se encargan de seleccionar los parámetros adecuados para las funciones PWL. Para x_1, los parámetros seleccionados son m_j^1 y c_j^1, y para x_2,

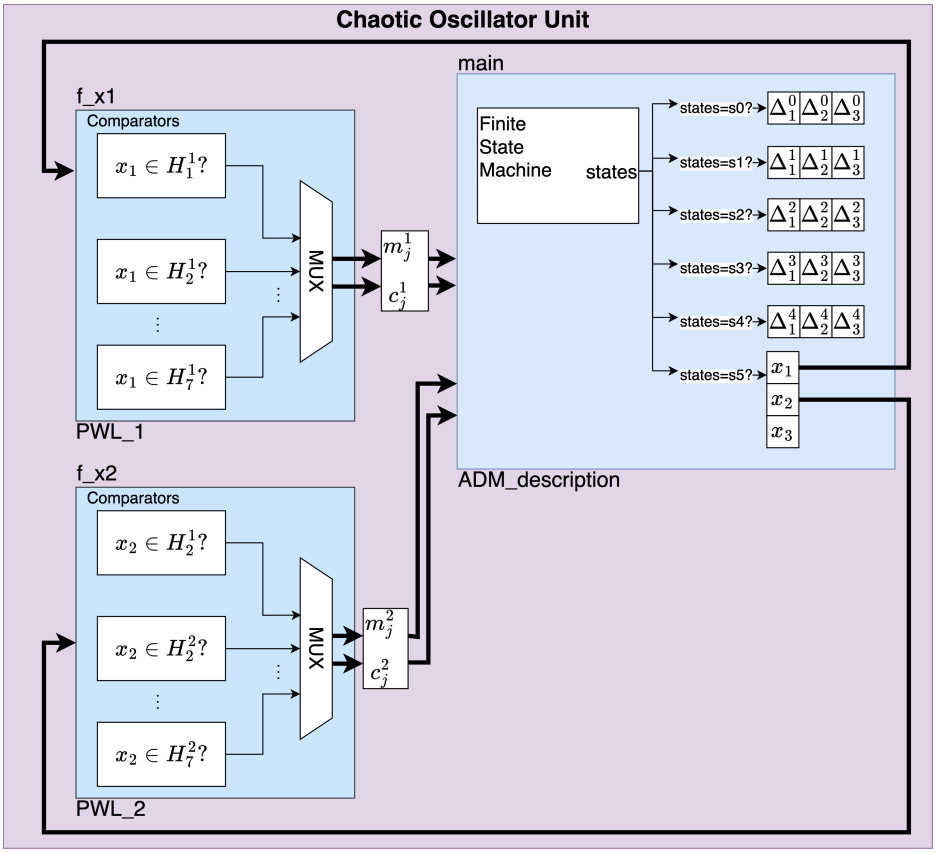


Figura 4.7: Diagrama de bloques del mecanismo de generación del atractor caótico de 4x4 enrollamientos.

se seleccionan m_j^2 y c_j^2. Esta selección es clave para la correcta implementación de las trayectorias del sistema caótico, ya que define las pendientes y las constantes para cada región.

3.- **Descripción PWL-DM:** Las salidas de los dos bloques MUX se envían a un único bloque `ADM_description`, que contiene la descripción en hardware del método PWL-DM. Este bloque utiliza una Máquina de Estados Finitos (FSM) que transiciona a través de varios estados ($s0$ a $s5$) para procesar los coeficientes de las funciones PWL correspondientes a ambas señales. La FSM calcula los coeficientes Δ_i^ℓ para cada estado y asegura que, al finalizar el proceso en el estado $s5$, se generen las trayectorias caóticas deseadas para ambos ejes, dando lugar al atractor caótico de 4×4 enrollamientos.

En resumen, la síntesis del sistema caótico PWL de orden fraccional con 4×4 enrollamientos sigue un proceso similar al caso anterior, pero introduce una mayor

complejidad debido al uso de dos señales caóticas y la necesidad de manejar múltiples parámetros y regiones.

4.2. Diseño embebido basado en RISC-V para la implementación de sistemas caóticos de orden fraccional

En esta sección se aborda el diseño e implementación de sistemas caóticos de orden fraccional utilizando una arquitectura embebida basada en RISC-V (Computadora con un Conjunto de Instrucciones Reducidas Versión 5). El objetivo principal es proporcionar una plataforma eficiente para la implementación y control de sistemas caóticos, aprovechando las características de los procesadores RISC, como su simplicidad de instrucciones, bajo consumo de energía y capacidad de computo para realizar operaciones en tiempo real.

La arquitectura RISC-V ha ganado reconocimiento en el campo de los sistemas embebidos y la investigación académica debido a su flexibilidad y eficiencia [**?**]. A diferencia de arquitecturas cerradas como x86, RISC-V permite personalizar y adaptar las instrucciones a los requisitos específicos de una aplicación, donde el conjunto de instrucciones se simplifica para mejorar el rendimiento y reducir la complejidad del hardware, siendo así una opción atractiva para aplicaciones en tiempo real y sistemas embebidos [169, 170].

En el contexto de los sistemas caóticos, la arquitectura RISC-V ofrece varias ventajas, como una mayor eficiencia en la ejecución de algoritmos matemáticos intensivos y la capacidad de manejar instrucciones personalizadas para optimizar operaciones específicas del sistema caótico. Además, la modularidad de RISC-V permite diseñar extensiones específicas que mejoran el manejo de las operaciones de punto fijo, cruciales para el cálculo de sistemas de orden fraccional [68].

4.2.1. Arquitectura del Conjunto de Instrucciones (ISA) RISC-V

La arquitectura del conjunto de instrucciones (ISA) RISC-V es la base sobre la cual se construye la flexibilidad y la eficiencia de los diseños de procesadores RISC-

V. Este conjunto de instrucciones define la interfaz funcional entre el hardware y el software, especificando las instrucciones que el procesador puede ejecutar, los tipos de datos soportados, y el modelo de registros [171].

La base de la ISA incluye un núcleo de instrucciones (RV32I para 32 bits y RV64I para 64 bits), que proporciona instrucciones esenciales de aritmética, carga y almacenamiento, control de flujo y operaciones lógicas.
Para este trabajo, se ha seleccionado la versión de 32 bits de la arquitectura RISC-V (conocida como RV32I). Esta versión de 32 bits permite un equilibrio entre eficiencia de cálculo y consumo de recursos, lo cual es fundamental en el diseño de un procesador que se implementará en una FPGA.

La ISA RV32I proporciona una variedad de instrucciones que cubren las operaciones esenciales para el procesamiento de datos y el control de flujo. A continuación se describen las principales categorías de instrucciones en esta ISA:

- Instrucciones Aritméticas: Estas incluyen operaciones de suma, resta y, opcionalmente, multiplicación para manejar las operaciones matemáticas básicas de los modelos caóticos.

- Instrucciones Lógicas: Las operaciones lógicas (AND, OR, XOR, y desplazamientos de bits) permiten el manejo de datos en la implementación de algoritmos de control y operaciones de manipulación de bits, esenciales en muchos algoritmos caóticos.

- Instrucciones de Carga y Almacenamiento: Estas instrucciones (LOAD y STORE) permiten la interacción con la memoria, facilitando la carga y almacenamiento de datos.

- Instrucciones de Control de Flujo: Las instrucciones de salto condicional e incondicional (BEQ, BNE, JAL, etc.) permiten manejar el flujo del programa, gestionando ciclos y condiciones en la ejecución de los programas.

La naturaleza abierta de la ISA RISC-V permite la personalización para ajustarse a las necesidades específicas del sistema caótico de orden fraccional. Algunas de las personalizaciones realizadas en el diseño incluyen:

- Instrucciones específicas para optimización de cálculos caóticos: Se han añadido instrucciones personalizadas que permiten realizar ciertos cálculos caóticos

recurrentes de forma más eficiente, mejorando el rendimiento sin incrementar el tamaño de la instrucción.

- Adaptaciones en el flujo de datos: El diseño del conjunto de registros y las rutas de datos se ha optimizado para permitir un acceso rápido y eficiente a las variables de estado, asegurando una ejecución fluida de las operaciones caóticas en el modelo de 32 bits.

4.2.1.1. Codificación de Instrucciones en RISC-V

En la arquitectura RISC-V, cada instrucción está compuesta de 32 bits y se organiza en varios campos que especifican el tipo de operación, los registros involucrados y otros detalles necesarios para ejecutar la instrucción. Cada tipo de instrucción tiene su propio formato, lo que permite identificar rápidamente la operación que debe realizar el procesador. A continuación, se explica cómo se codifican los diferentes tipos de instrucciones: R-type, I-type, S-type, B-type, U-type y J-type.

R-Type

Las instrucciones de tipo R son operaciones aritméticas y lógicas que utilizan dos registros fuente (`rs1` y `rs2`) y un registro de destino (`rd`). La codificación de las instrucciones de tipo R utiliza los siguientes campos:

- **opcode (7 bits)**: Define el tipo general de operación (por ejemplo, aritmética).
- **rd (5 bits)**: Especifica el registro de destino.
- **funct3 (3 bits)**: Define la suboperación específica dentro de la categoría indicada por el `opcode`.
- **rs1 (5 bits)**: Registro fuente 1.
- **rs2 (5 bits)**: Registro fuente 2.
- **funct7 (7 bits)**: Define variantes de la operación, como en el caso de `ADD` (funct7 = 0x00) y `SUB` (funct7 = 0x20).

Ejemplo: La instrucción `ADD rd, rs1, rs2` tiene los siguientes campos:

- `opcode = 0110011`
- `funct3 = 000`
- `funct7 = 0000000` (para `ADD`), o `funct7 = 0100000` (para `SUB`).

I-Type

Las instrucciones de tipo I incluyen operaciones con un registro fuente y un valor inmediato o direcciones de carga y salto. Incluyen los siguientes campos:

- **opcode (7 bits)**: Define el tipo de operación.
- **rd (5 bits)**: Registro de destino.
- **funct3 (3 bits)**: Especifica la suboperación dentro de la categoría indicada por el `opcode`.
- **rs1 (5 bits)**: Registro fuente.
- **imm (12 bits)**: Valor inmediato de 12 bits, que puede representar un desplazamiento o una constante.

El formato binario de una instrucción de tipo I es:
Ejemplo: La instrucción `ADDI rd, rs1, imm` tiene los siguientes campos:

- `opcode = 0010011`
- `funct3 = 000` (para `ADDI`)
- `imm`: Valor inmediato.

S-Type

Las instrucciones de tipo S se utilizan para operaciones de almacenamiento en memoria. Estas instrucciones incluyen:

- **opcode (7 bits)**: Define el tipo de operación (por ejemplo, almacenamiento).
- **imm[11:5] (7 bits)**: Parte alta del valor inmediato.

- **rs2 (5 bits)**: Registro fuente cuyo valor se almacena en memoria.
- **rs1 (5 bits)**: Registro base.
- **funct3 (3 bits)**: Define el tipo de almacenamiento.
- **imm[4:0] (5 bits)**: Parte baja del valor inmediato.

El formato binario de una instrucción de tipo S es:
Ejemplo: La instrucción `SW rs2, imm(rs1)` tiene:

- `opcode = 0100011`
- `funct3 = 010` (para `SW`)
- `imm`: Desplazamiento dividido en `imm[11:5]` y `imm[4:0]`.

B-Type

Las instrucciones de tipo B son ramas condicionales:

- **opcode (7 bits)**: Operación de rama.
- **imm[12—10:5] (7 bits)**: Parte del valor inmediato.
- **rs2 (5 bits)**: Registro fuente 2.
- **rs1 (5 bits)**: Registro fuente 1.
- **funct3 (3 bits)**: Condición de la rama.
- **imm[4:1—11] (5 bits)**: Parte del valor inmediato.

Formato binario:
Ejemplo: `BEQ rs1, rs2, imm`

- `opcode = 1100011`
- `funct3 = 000`
- `imm`: Desplazamiento.

U-Type

Las instrucciones de tipo U cargan un valor inmediato de 20 bits en la parte superior de un registro.

- **opcode (7 bits)**: Tipo de operación.
- **rd (5 bits)**: Registro de destino.
- **imm[31:12] (20 bits)**: Valor inmediato de 20 bits.

Formato binario:
Ejemplo: `LUI rd, imm`

- `opcode = 0110111`
- `imm`: Parte superior del valor inmediato.

J-Type

Las instrucciones de tipo J se utilizan para saltos incondicionales.

- **opcode (7 bits)**: Operación de salto.
- **rd (5 bits)**: Registro de destino.
- **imm[20—10:1—11—19:12] (20 bits)**: Valor inmediato.

Formato binario:
Ejemplo: `JAL rd, imm`

- `opcode = 1101111`
- `imm`: Desplazamiento de salto.

Instrucciones personalizadas

En este diseño, se han añadido varias instrucciones personalizadas que amplían las capacidades de la arquitectura RISC-V para manejar configuraciones de punto fijo, interrupciones, y temporizadores. A continuación, se detallan cada una de estas

instrucciones, su funcionalidad, y la codificación de los campos.

Configuración de Punto Fijo: `fixed_point_t`

La instrucción `fixed_point_t` permite ajustar la configuración de punto fijo, definiendo la cantidad de bits para la parte entera y la parte decimal según el formato `QI,nI`. La instrucción recibe un valor inmediato que especifica la cantidad de bits para la parte decimal.

- **Instrucción:** `fixed_point_t val`
- **Campos:**
  - `imm11`: Guarda el valor de los bits necesarios para la parte decimal.
  - `rs1`: Apunta a `x0` (siempre).
  - `funct3`: 000.
  - `rd`: Apunta a `x0` (siempre).
  - `opcode`: 0000001.

Habilitación de Interrupciones

Se han diseñado tres instrucciones personalizadas para habilitar interrupciones basadas en un temporizador, una entrada, o una salida. Estas instrucciones reciben un valor de estado (0 o 1) que indica si la interrupción está habilitada o deshabilitada y un enlace a la función de interrupción correspondiente.

- **Instrucciones:**
  - `set_timer_interrupt state timer_interrupt_function`
  - `set_input_interrupt state input_interrupt_function`
  - `set_output_interrupt state output_interrupt_function`
- **Campos:**
  - `imm11`: Almacena la dirección de memoria de la función de interrupción en los bits 11 a 1, y el estado en el bit 0.

- `rs1`: Apunta a `x0` (siempre).
- `funct3`:
  - 001 para `set_timer_interrupt`.
  - 010 para `set_input_interrupt`.
  - 011 para `set_output_interrupt`.
- `rd`: Apunta a `x0` (siempre).
- `opcode`: 0000001.

Configuración del Temporizador

Las instrucciones de configuración del temporizador permiten inicializarlo en modo de conteo ascendente o descendente, obtener el valor actual del temporizador, y restablecerlo.

- **Instrucciones:**
  - `timer_init_up`: Configura el temporizador con un conteo ascendente.
  - `timer_init_down`: Configura el temporizador con un conteo descendente.
  - `timer_get`: Obtiene el valor actual del temporizador y lo almacena en `x15`.
  - `timer_reset`: Restablece la configuración del temporizador.
- **Campos para `timer_init_up` y `timer_init_down`:**
  - `rs2`: Registro que almacena el valor donde llegará el temporizador.
  - `rs1`: Apunta a `x0` (siempre).
  - `rd`: Apunta a `x0` (siempre).
  - `funct3`:
    - 000 para `timer_init_up`.
    - 001 para `timer_init_down`.
  - `opcode`: 0000010.
- **Campos para `timer_get`:**
  - `rs2`: Apunta a `x0` (siempre).

- `rs1`: Apunta a `x0` (siempre).
- `rd`: Apunta a `x15` (siempre, aquí se almacena el resultado del temporizador).
- `funct3`: 010.
- `opcode`: 0000010.

- **Campos para `timer_reset`:**
  - `rs2`: Apunta a `x0` (siempre).
  - `rs1`: Apunta a `x0` (siempre).
  - `rd`: Apunta a `x0` (siempre).
  - `funct3`: 011.
  - `opcode`: 0000010

4.2.1.2. Registros

En la arquitectura RISC-V, los registros juegan un papel fundamental en la ejecución eficiente de instrucciones. Los registros permiten un acceso rápido a los datos, optimizando el procesamiento en comparación con el acceso a la memoria principal. En el conjunto de instrucciones RISC-V, existen 32 registros de propósito general, cada uno de 32 bits en la variante RV32I, numerados de `x0` a `x31`. No obstante, en este diseño personalizado de la arquitectura RISC-V, se han configurado 19 registros, en lugar de los 32 registros de propósito general estándar. Cada registro tiene una función específica y se han realizado modificaciones importantes, como el uso de un temporizador en el registro `x15`. A continuación, se detallan los registros considerados en este diseño:

Registros de Propósito General Los registros de propósito general permiten el almacenamiento temporal de datos y se utilizan ampliamente en las operaciones aritméticas, lógicas, de carga y de almacenamiento. En este diseño, los registros desde `x1` hasta `x18` actúan como registros de propósito general, lo que significa que pueden contener valores intermedios, almacenar datos temporales y manejar argumentos de función y valores de retorno, según las necesidades del programa.

Registros de Propósito Específico Algunos registros en esta configuración tienen funciones específicas:

- `x0` **(Zero):** Este es un registro de solo lectura que siempre contiene el valor constante cero. Todas las escrituras a `x0` son ignoradas. Su propósito es proporcionar un valor nulo en operaciones sin afectar el contenido de otros registros.
- `x1` **(Link):** Conocido como el registro de enlace o *link register*, `x1` se utiliza para almacenar la dirección de retorno en llamadas de función. Al ejecutar una instrucción de salto y enlace (`JAL` o `JALR`), la dirección de retorno se guarda en `x1`, permitiendo a la función volver al punto desde donde fue llamada.
- `x2` **(Pointer):** Este registro actúa como puntero, Es esencial para la gestión de llamadas a funciones, el manejo de variables locales y el almacenamiento temporal de datos.
- `x15` **(Timer):** En esta configuración, `x15` se utiliza para obtener la lectura de un temporizador.

4.2.2. Diseño de la Microarquitectura RISC-V

La Microarquitectura se refiere al diseño interno de un procesador que define cómo se implementan las instrucciones del conjunto de instrucciones (ISA) en el hardware. Es decir, la microarquitectura especifica los componentes y la organización del procesador, incluyendo la unidad de control, la unidad aritmética y lógica (ALU), el banco de registros, y las unidades de memoria y entrada/salida. El diseño de la microarquitectura no es una tarea trivial y es crucial para maximizar el rendimiento y la eficiencia del procesador, permitiendo que cada instrucción se ejecute de manera óptima.

4.2.2.1. Arquitectura Harvard para la Configuración de Memorias

Para la configuración de memorias en esta microarquitectura, se ha seleccionado la arquitectura Harvard mostrada en la figura 4.8. En este diseño, las memorias de instrucciones y de datos están separadas, permitiendo un acceso simultáneo e independiente a las instrucciones y a los datos. Esto optimiza la velocidad de ejecución, ya que las instrucciones y los datos no compiten por el mismo bus de memoria. Además,

facilita la configuración de las memorias para cumplir con los requisitos específicos de almacenamiento y acceso en los sistemas caóticos.

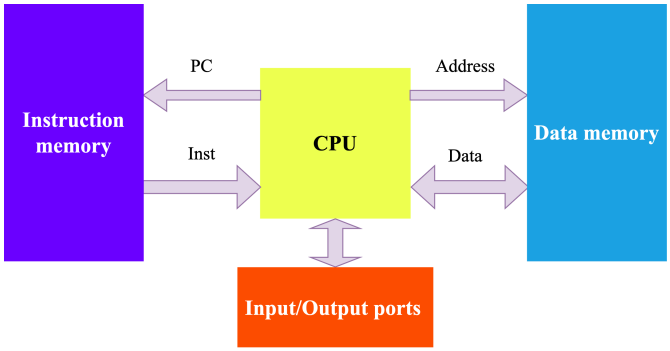


Figura 4.8: Diagrama de bloques de la arquitectura de Harvard.

La microarquitectura implementada se compone de los siguientes elementos principales:

CPU La unidad central de procesamiento (CPU) integra los elementos necesarios para ejecutar instrucciones y realizar operaciones de control y cálculo. En esta CPU, se destacan los siguientes subcomponentes:

- **Unidad de Control:** Responsable de decodificar las instrucciones y generar las señales de control necesarias para coordinar las operaciones en el procesador. La unidad de control ha sido ampliada para manejar las instrucciones personalizadas definidas en este proyecto, como `fixed_point_t` y las instrucciones de temporización e interrupción.

- **Banco de Registros:** Implementa 19 registros, de los cuales `x0` es de solo lectura con valor constante cero, y el resto se utilizan para el almacenamiento temporal de datos y valores intermedios.

- **Unidad Aritmética y Lógica (ALU):** La ALU realiza operaciones aritméticas y lógicas, incluyendo el soporte para punto fijo a través de la instrucción `fixed_point_t`. Esta unidad permite ejecutar operaciones matemáticas intensivas de manera eficiente, siendo un componente clave en el procesamiento de las ecuaciones de los sistemas caóticos.

Memoria de Instrucciones La memoria de instrucciones es una memoria de lectura y escritura donde se almacenan las instrucciones del programa. Esta memoria permite cargar y ejecutar programas personalizados, y puede recibir datos adicionales durante la ejecución del programa. Para cargar el programa en la memoria de instrucciones o enviar datos necesarios para la ejecución, se utiliza el protocolo UART (RS232), que permite la comunicación con el procesador desde una computadora externa. Esta configuración facilita la programación y el control remoto de la microarquitectura, así como la actualización dinámica de datos e instrucciones según los requerimientos de los sistemas caóticos.

Memoria de Datos La memoria de datos complementa el banco de registros y se utiliza para almacenar resultados o datos generados durante la ejecución del programa. Aunque los 19 registros disponibles en esta microarquitectura son suficientes para implementar múltiples sistemas caóticos de orden fraccional, existen aplicaciones en las que se generan datos adicionales que necesitan ser analizados después del cálculo de la solución del sistema. En estos casos, dichos datos se almacenan en la memoria de datos para su posterior procesamiento y análisis, lo que proporciona una mayor flexibilidad en el manejo y almacenamiento de la información generada por el sistema.

Entradas y Salidas El diseño de la microarquitectura incluye configuraciones específicas para la entrada y salida de datos. Como salida, se ha configurado el protocolo SPI (Serial Peripheral Interface) funcionando a 20 MHz, permitiendo una comunicación rápida y eficiente con dispositivos externos. Además, el sistema cuenta con una entrada de 32 bits, donde el bit menos significativo (bit 0) está reservado para verificar el estado del protocolo SPI.

4.2.3. Ciclo de Instrucción en la Microarquitectura Propuesta

El diseño propuesto sigue un ciclo de instrucciones estándar, compuesto por cuatro etapas principales: **Fetch**, **Decode**, **Execute** y **Store**. A continuación, se describe cada etapa en detalle y se explican los elementos de la microarquitectura que intervienen en cada una, según el diagrama de bloques de la figura 4.9. **Etapa de Fetch (Búsqueda)**

En la etapa de **Fetch**, la CPU recupera la instrucción desde la **memoria de instrucciones**. El contador de programa (**PC**) indica la dirección de la siguiente instrucción a cargar. A través de un **multiplexor** (Mux), se selecciona si el valor del **PC** o una dirección alternativa será utilizada para la recuperación de la instrucción.

- **PC (Program Counter):** Contiene la dirección de la instrucción actual. Al final de cada ciclo de instrucción, el **PC** se actualiza para apuntar a la siguiente instrucción en la secuencia.
- **Instruction Memory (Memoria de Instrucciones):** Es una memoria de lectura/escritura donde se almacenan las instrucciones del programa. La instrucción recuperada se almacena temporalmente en un registro de instrucción (**INST[31:0]**), que se utilizará en la etapa de decodificación.
- **PLOAD (Program Loader):** Este módulo se comunica mediante el protocolo UART (RS232) y permite cargar programas y datos en la memoria de instrucciones desde una computadora externa.

Etapa de Decode (Decodificación)

Durante la etapa de **Decode**, la instrucción recuperada se interpreta para identificar el tipo de operación, los registros involucrados y cualquier valor inmediato necesario. Los campos de la instrucción se separan en **opcode**, **funct3**, **funct7**, **rd**, **rs1** y **rs2**.

- **Inst**: La instrucción se decodifica en sus componentes, permitiendo a la unidad de control generar las señales necesarias para la operación.
- **Banco de Registros**: Los registros fuente (`rs1` y `rs2`) se leen desde el banco de registros, proporcionando los valores necesarios para la ejecución de la instrucción. Aunque el banco de registros cuenta con 19 registros en total (incluyendo `x0` y `x15` configurado como temporizador), en esta etapa solo se utilizan los registros especificados por la instrucción.

Etapa de Execute (Ejecución)

La etapa de **Execute** es donde se ejecuta la lógica de la instrucción, los resultados obtenidos se preparan para ser almacenados en registros o ser transferidos a la siguiente

etapa. La operación indicada por la instrucción se lleva a cabo utilizando la **Unidad Aritmética y Lógica (ALU)**, junto con otros elementos de control y temporización si es necesario.

- **ALU (Unidad Aritmética y Lógica):** Realiza las operaciones aritméticas y lógicas, como suma, resta, operaciones de punto fijo, etc. La ALU recibe los valores de los registros fuente (`A[31:0]` y `B[31:0]`) a través de multiplexores, que seleccionan los valores apropiados según la instrucción.
  La inclusión de la entrada `Temp_REG` en la etapa de ejecución permite mayor flexibilidad en la fuente de datos para la ALU, mejorando las capacidades de procesamiento de datos externos.

- **Branch (Unidad de Saltos):** Esta unidad se encarga de gestionar instrucciones de salto y de funciones de interrupción, redirigiendo el flujo del programa si la instrucción así lo requiere. Las instrucciones de configuración de punto fijo, temporización e interrupción también se gestionan en esta etapa mediante señales específicas de control.

Etapa de Store (Almacenamiento)

En la etapa de **Store**, los resultados de la instrucción se guardan en la **memoria de datos** o se envían a un dispositivo de salida, según sea el caso.

- **Data Memory (Memoria de Datos):** Es una memoria de lectura/escritura donde se almacenan los datos generados durante la ejecución de las instrucciones. Aunque el sistema cuenta con 19 registros, que son suficientes para la mayoría de las operaciones, los datos adicionales o resultados intermedios se pueden almacenar en la memoria de datos para su posterior análisis.

- **SPI Module (Módulo SPI):** El protocolo SPI, configurado a 20 MHz, sirve como interfaz de salida para transmitir datos a dispositivos externos. El bit 0 de la entrada de 32 bits de `Input_REG` está reservado para verificar el estado del protocolo SPI.

- **Input_REG (Registro de Entrada):** Este registro de entrada permite recibir datos externos mediante el puerto extendido `Ext_port[30:0]`, que pueden ser almacenados o utilizados en operaciones subsecuentes dentro del CPU.

La etapa de **Store** finaliza el ciclo de instrucción, guardando los resultados y preparando el procesador para la siguiente instrucción.

Es necesario remarcar que el tamaño de las memorias (memoria de instrucciones y memoria de datos) tiene un tamaño inicial de 4096 direcciones de 32 bits cada una. No obstante, este tamaño puede ser ajustado de acuerdo a las necesidades del sistema, permitiendo añadir más direcciones o reducirlas según los requisitos específicos de la implementación.

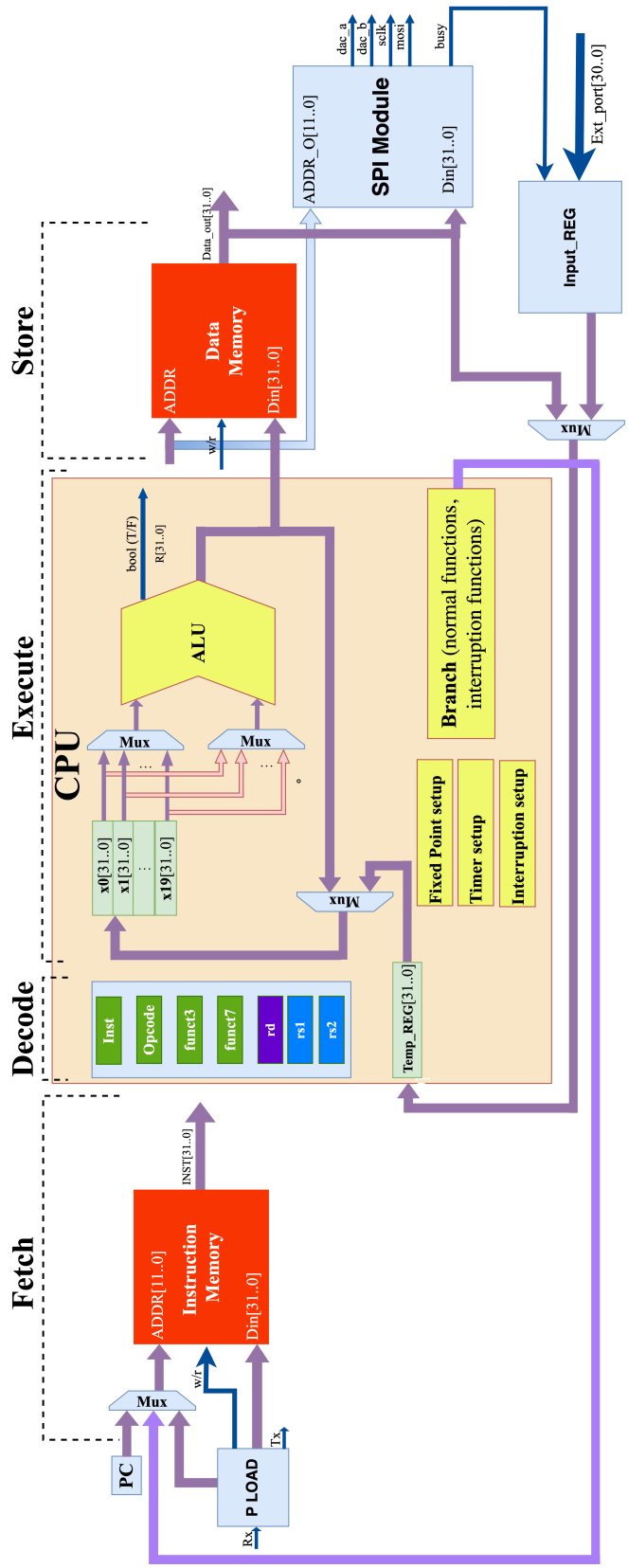


Figura 4.9: Ciclo de instrucción en la microarquitectura propuesta.

4.2.4. Validación del sistema embebido basado en RISC-V

En esta sección se procede a validar el diseño embebido tipo RISC en una FPGA. Se presentarán los resultados de la implementación del sistema embebido, evaluando su rendimiento y funcionalidad dentro del entorno de hardware. Además, se describirá la metodología utilizada para llevar a cabo la implementación de sistemas caóticos de orden fraccional en este diseño.

4.2.4.1. Diseño en FPGA del sistema embebido basado en RISC-V

Al igual que en la sección 4.1, se ha utilizado la tarjeta FPGA Nexys Video y VHDL como lenguaje de configuración. En la Figura 4.10 se puede observar el esquema RTL del sistema embebido. Este diagrama muestra la interconexión de los módulos principales que componen el procesador RISC-V implementado, siguiendo el flujo de instrucciones descrito en la sección anterior, que incluye las etapas de Fetch, Decode, Execute y Store. El esquemático RTL se compone de los siguientes módulos principales, cada uno desempeñando un papel en el flujo de instrucciones del procesador:

- **clock_div_wrapper**: Este módulo divide la señal de reloj de entrada (`clk`) para generar un reloj de sistema adecuado para el funcionamiento del procesador. Proporciona una señal de reloj de salida (`clk_out1`) que sincroniza la operación de todos los módulos dentro del sistema.

- **Rx_mod y Tx_mod (UART Modules)**: Estos módulos implementan la comunicación UART (RS232) para la recepción y transmisión de datos. **Rx_mod** recibe datos de entrada a través del puerto `rx` y los almacena en el registro de recepción (`Rx_reg[7:0]`) mientras que **Tx_mod** envía datos desde el procesador al puerto `tx`. Estos módulos permiten la carga de programas e instrucciones en el procesador desde una computadora externa, interactuando directamente con el módulo **Load**.

- **Load**: Este módulo se encarga de la carga de instrucciones en la **memoria de instrucciones (Mprog)**. Mediante el uso de la señal `Load_mod`, el módulo **Load** permite cargar datos en el procesador y habilita la escritura de instrucciones desde la entrada UART en la memoria de instrucciones (`Mprog`). Este módulo recibe datos desde `Rx_mod` y transmite datos hacia el procesador.

- **Mdata y Mprog (Memorias de Datos e Instrucciones)**:
  - **Mprog (Memoria de Instrucciones)** almacena las instrucciones del programa y se comunica con el procesador a través de las señales `PC[11:0]` y `inst[31:0]`, permitiendo que el procesador recupere las instrucciones durante la etapa de **Fetch**.
  - **Mdata (Memoria de Datos)** se utiliza para almacenar datos generados o necesarios durante la ejecución de instrucciones, interactuando con el procesador en las etapas de **Execute** y **Store**. La memoria de datos se accede mediante la señal de dirección `ADDR[11:0]` y los datos se transfieren a través de `Dout[31:0]`.
- **CPU (Micro)**: Es el núcleo del sistema embebido y realiza las operaciones de procesamiento siguiendo el ciclo de instrucciones estándar. La **CPU** recibe instrucciones desde la **memoria de instrucciones (Mprog)**, decodifica las operaciones, ejecuta los cálculos necesarios en la **ALU** y almacena los resultados en la **memoria de datos (Mdata)** o los envía a los módulos de salida. La CPU también interactúa con el registro de entrada `port_ext[31:0]` a través de `Input_REG` y utiliza multiplexores para seleccionar entre los datos provenientes de la memoria de datos o del registro de entrada, según la configuración.
- **Input_REG**: Este módulo permite al procesador recibir datos externos mediante el puerto `Ext_port[30:0]`, proporcionando una entrada adicional al sistema. La señal `busy` indica el estado del registro de entrada y permite sincronizar la entrada de datos. Durante la etapa de **Execute**, el procesador selecciona entre los datos provenientes de la memoria de datos o `Input_REG` mediante un multiplexor, lo que facilita la manipulación de datos externos.
- **MCP4922_driver (SPI Module)**: Este módulo implementa el protocolo SPI para la salida de datos a dispositivos externos. Configurado a 20 MHz, el módulo SPI envía datos a través de las señales `dac_a`, `dac_b`, `sclk`, y `mosi`. La señal `busy` indica si el módulo SPI está ocupado, permitiendo al procesador gestionar la comunicación de manera eficiente.

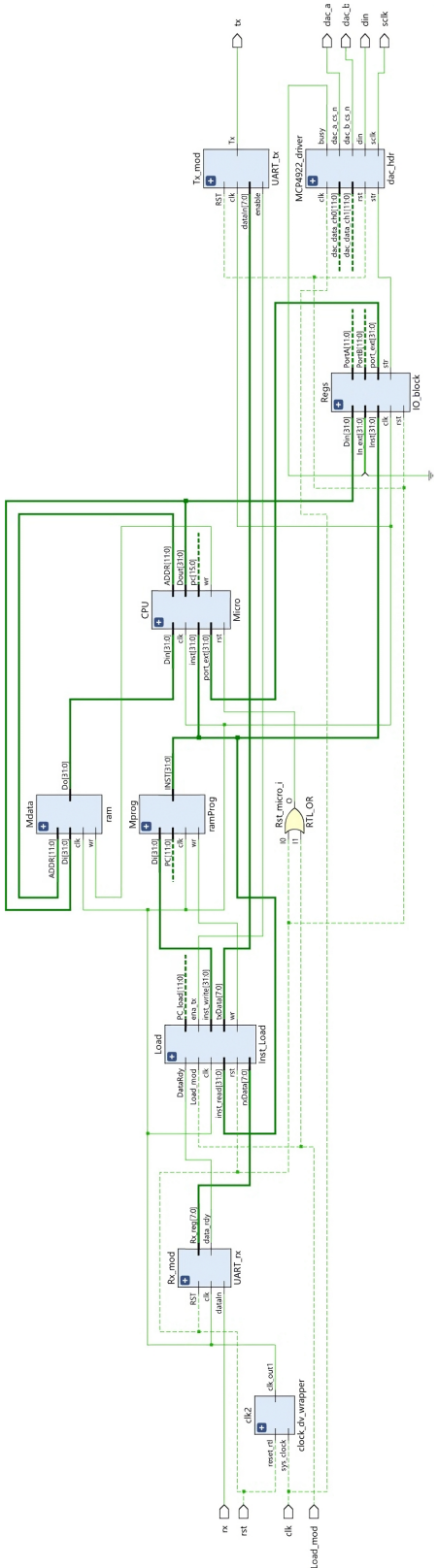


Figura 4.10: Esquemático RTL de la implementación del sistema embebido basdo en RISC-V.

4.2.4.2. Implementación de sistemas caóticos de orden fraccional en el sistema embebido basado en RISC-V

La implementación de sistemas caóticos de orden fraccional en el sistema embebido propuesto sigue una serie de pasos consecutivos, tal como se muestra en el diagrama de flujo de la figura 4.11. Se puede observar, que en esta metodología de implementación, no se requiere una simulación inicial en punto fijo. En este diseño digital, la palabra digital puede validarse con el sistema embebido operando en tiempo real. Esto se debe a que, al utilizar un diseño de alto nivel de abstracción, se reduce significativamente el proceso de depuración, permitiendo que los ajustes requeridos puedan realizarse directamente durante la ejecución del sistema.
A continuación, se describe cada paso de esta metodología.

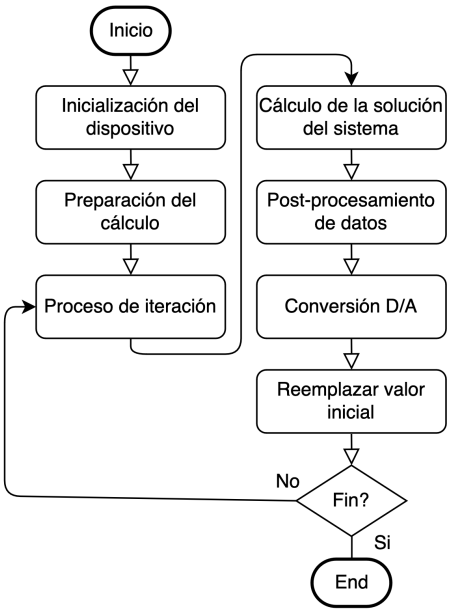


Figura 4.11: Procedimiento para implementaciones de sistemas caóticos de orden fraccional en el diseño embebido propuesto.

- **Inicialización del dispositivo:** Este primer paso garantiza que el hardware del dispositivo esté encendido y en correcto funcionamiento. Durante la inicialización, se establece la comunicación entre la computadora externa y la FPGA donde se encuentra implementado el diseño embebido. Esta comunicación de datos se realiza a través del protocolo RS232, permitiendo que el sistema embebido reciba instrucciones y datos de configuración desde la computadora.

- **Preparación del cálculo:** Con el objetivo de optimizar el rendimiento y reducir la carga de procesamiento, en esta fase se almacenan en la memoria del sistema embebido las constantes necesarias y se realizan cálculos que solo requieren una única ejecución. Estos cálculos incluyen, por ejemplo, valores constantes o parámetros del sistema que no cambian durante el proceso iterativo. De esta manera, el cálculo del sistema puede concentrarse únicamente en las operaciones necesarias en cada iteración.

- **Proceso de iteración:** En esta etapa se define la duración de la ejecución del programa, especificando el número de muestras a obtener del sistema caótico de orden fraccional. La duración puede estar limitada por un número fijo de iteraciones o configurarse para ejecutarse de manera indefinida según las necesidades del usuario.

- **Cálculo de la solución del sistema:** En esta fase se implementa el algoritmo específico para resolver el sistema caótico de orden fraccional. Este cálculo se realiza en el sistema embebido, utilizando solo los recursos de hardware del mismo. La solución obtenida en cada iteración representa una muestra del comportamiento del sistema caótico, y estas muestras son fundamentales para el análisis posterior o visualización del sistema.

- **Post-procesamiento de datos:** Las muestras obtenidas del cálculo de la solución, que se representan en 32 bits en punto fijo, son procesadas para convertirlas a un formato de 12 bits enteros positivos. Esta reducción de bits tiene el propósito de adaptar los datos para ser enviados mediante el protocolo SPI al convertidor digital-analógico (DAC), facilitando así la transmisión de datos y su visualización en etapas posteriores.

- **Conversión D/A:** Los datos procesados, ahora en formato de 12 bits, son enviados al convertidor digital-analógico (DAC) a través del protocolo SPI. La conversión de los datos de formato digital a analógico permite que las señales generadas puedan ser observadas en un osciloscopio, proporcionando una visualización en tiempo real del comportamiento del sistema caótico de orden fraccional.

- **Reemplazar valor inicial:** Después de obtener la solución del sistema en una iteración, el valor inicial utilizado para el cálculo se actualiza con el resultado

recién obtenido. Este paso asegura que el cálculo en la siguiente iteración parta de un estado actualizado, permitiendo que el sistema caótico evolucione de acuerdo a su dinámica inherente.

- **Fin del proceso:** El proceso iterativo continuará ejecutándose durante el número de muestras determinado. Alternativamente, si es requerido, el proceso puede ejecutarse de manera indefinida. El criterio de finalización depende de la configuración establecida en la etapa de **Proceso de iteración**.

Es importante mencionar que para esta primera versión del diseño, se utiliza ensamblador RISC-V para la descripción de las instrucciones del programa, de acuerdo con lo mencionado en la Sección 4.2.1.1. Este ensamblador permite especificar cada operación que el procesador debe realizar, definiendo los registros, valores inmediatos y funciones específicas para cada instrucción.

Capítulo 5

Análisis de Resultados

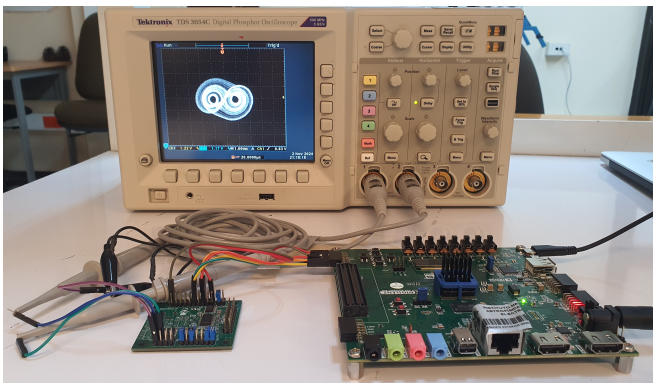
En este capítulo se presentan los resultados obtenidos de las dos metodologías utilizadas en la implementación de sistemas caóticos de orden fraccional: la síntesis estándar de sistemas caóticos en FPGA y el sistema embebido basado en RISC-V. Se incluyen los retratos de fase observados a nivel experimental, así como un análisis del uso de recursos de hardware, consumo de energía y rendimiento computacional de ambas implementaciones. Este análisis permite evaluar las diferencias en términos de rendimiento, escalabilidad y adaptabilidad de cada enfoque para futuros desarrollos.

5.0.1. Configuración experimental

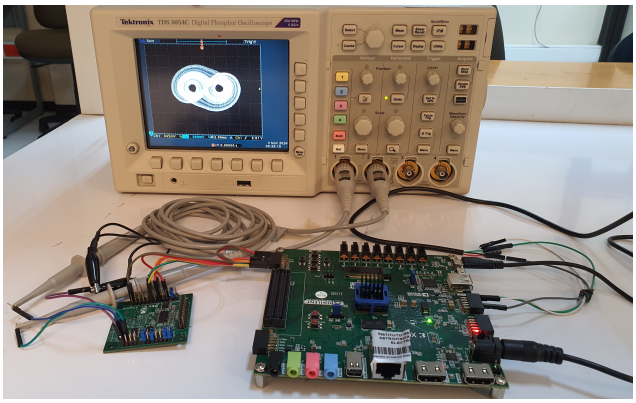
Las figuras 5.1 (a) y (b) muestran la configuración experimental utilizadas por ambas metodologías de implementación. Para fines de evaluación, todas implementaciones se llevaron en una tarjeta de desarrollo Nexys Video, que es una plataforma FPGA basado en un Artix-7 de Xilinx. La unica diferencia en la configuración experimental, se observa en la figura 5.1 (b), en esta configuración destaca la incorporación de una conexión RS-232 que permite el envío de instrucciones y datos desde una computadora externa hacia el sistema embebido implementado en la FPGA.

Las figuras 5.2 y 5.3 presentan los resultados experimentales de las dos metodologías de implementación utilizadas para el sistema caótico PWL de orden fraccional, tanto para el caso de 2 enrollamientos como para el de 4×4 enrollamientos.

- La Figura 5.2 presenta los resultados experimentales de la implementación en FPGA basada en la síntesis estándar de sistemas caóticos. En la Figura 5.2 (a), se observa el retrato de fase x_1, x_2 del sistema caótico PWL de orden fraccional



(a)



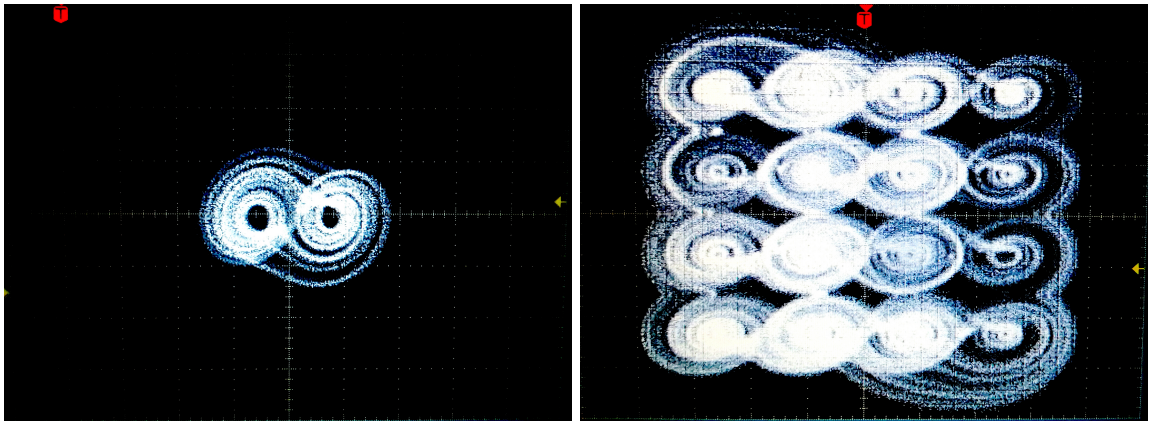
(b)

Figura 5.1: Configuración experimental para las implementaciones de sistemas caóticos de orden fraccional. (a) Configuración experimental utilizando la síntesis estandar de sistemas caóticos y (b) Configuración experimental utilizando el diseño embebido basado en RISC-V.

con 2 enrollamientos, mientras que la subfigura (b) muestra el retrato de fase x_1, x_2 del sistema con 4×4 enrollamientos.

- La Figura 5.3 muestra resultados experimentales obtenidos en el sistema embebido basado en RISC-V, con el mismo esquema de visualización de los atractores caóticos en el retrato de fase x_1, x_2. La 5.3 (a) y 5.3 (b) corresponden al sistema caótico PWL de orden fraccional con 2 enrollamientos y con 4×4 enrollamientos respectivamente.

En ambas metodologías de implementación, los atractores caóticos observados experimentalmente son consistentes con aquellos obtenidos mediante simulaciones numéricas en las figuras 3.1 y 3.2, y también con la teoría subyacente en [165, 166].



(a) (b)

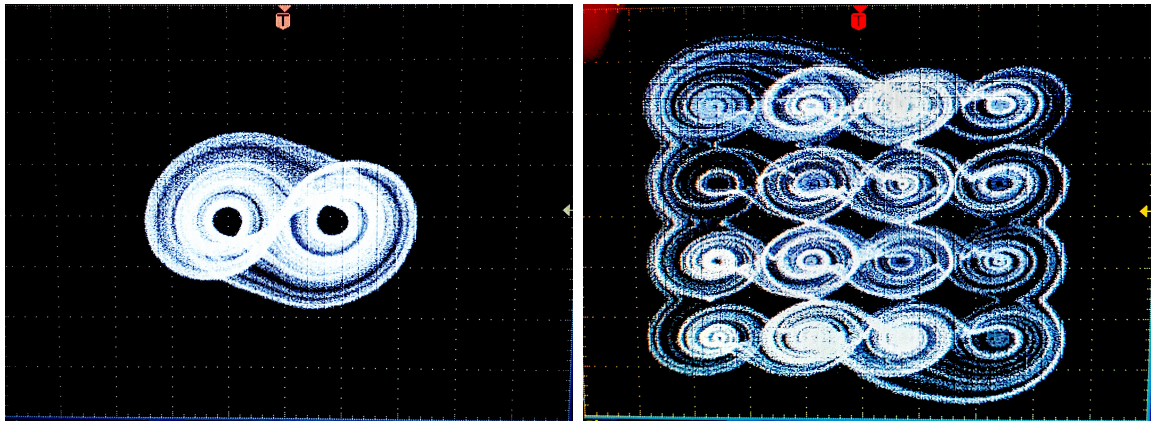
Figura 5.2: Resultados experimentales de la implementación en FPGA basada en la síntesis estándar de sistemas caóticos, (a) retrato de fase x_1, x_2 del sistema del sistema caótico PWL de orden fraccional con 2 enrollamientos, (b) retrato de fase x_1, x_2 del sistema caótico PWL de orden fraccional con 4×4 enrollamientos.

5.0.2. Uso de hardware

- **LUTs (Look-Up Tables)**: El uso de LUTs muestra variaciones significativas entre las implementaciones debido a la complejidad de cada sistema. El sistema de 2 enrollamientos utiliza el 2.75 % de las LUTs disponibles, mientras que el sistema de 4×4 enrollamientos requiere un 4.38 %, reflejando el incremento de la complejidad lógica en este último. La implementación embebida basada en RISC-V utiliza el 4.75 %, un ligero incremento que el sistema de 4×4 enrollamientos bajo sintesis estandar, debido a la arquitectura más abstracta y versátil de RISC-V que requiere operaciones lógicas adicionales para gestionar el sistema.

- **FFs (Flip-Flops)**: La utilización de Flip-Flops se mantiene moderada en todas las implementaciones, oscilando entre el 0.93 % en el sistema embebido y aproximadamente 1.10 % en los sistemas de síntesis estándar. Esta baja demanda de FFs sugiere que ninguno de los sistemas requiere un almacenamiento secuencial intensivo, lo que es favorable para optimizar el uso de estos recursos en otros módulos si se expande el diseño.

- **DSPs (Digital Signal Processing blocks)**: La utilización de bloques DSP varía considerablemente entre las implementaciones. Los sistemas de 2 y 4×4 enrollamientos en la síntesis estándar utilizan el 5.45 % y 7.27 %, respectiva-



(a) (b)

Figura 5.3: Resultados experimentales de la implementación realizada el sistema embebido basado en RISC-V, (a) retrato de fase x_1, x_2 del sistema del sistema caótico PWL de orden fraccional con 2 enrollamientos, (b) retrato de fase x_1, x_2 del sistema caótico PWL de orden fraccional con 4×4 enrollamientos.

mente, debido a la necesidad de manejar operaciones aritméticas intensivas en sistemas caóticos. En contraste, el sistema embebido solo emplea el 1.82 % de los DSPs, ya que optimiza los recursos aritméticos mediante un ruteo eficiente y registros pipeline, reduciendo significativamente la demanda de estos bloques en comparación con las implementaciones estándar.

- **BRAM (Block RAM)**: Los sistemas de síntesis estándar no requieren BRAM, lo cual indica que no necesitan almacenamiento interno adicional para sus operaciones. Sin embargo, el sistema embebido utiliza un 5.71 % de los bloques BRAM, permitiendo un almacenamiento temporal que mejora la eficiencia en la gestión de datos y las instrucciones. Esto es especialmente útil para aplicaciones en tiempo real, donde el sistema embebido puede beneficiarse de una memoria integrada.

- **IO (Input/Output)**: La utilización de pines IO también muestra diferencias marcadas. Las implementaciones de síntesis estándar para los sistemas de 2 y 4×4 enrollamientos utilizan un 49 % de los pines, ya que requieren salidas directas para cada variable del sistema caótico. En contraste, el sistema embebido emplea solo el 4.50 %, utilizando el protocolo SPI para una comunicación más compacta y eficiente. Este enfoque optimiza los recursos IO y añade flexibilidad al diseño, permitiendo controlar el flujo de datos sin modificaciones de hardware

adicionales.

- **BUFG (Global Clock Buffers)**: Los Global Clock Buffers presentan un aumento significativo en el sistema embebido, que utiliza el 9.38 % (3 BUFGs), frente al 3.13 % en los sistemas de síntesis estándar. Este incremento se debe al uso de un módulo de manejo de reloj (MML), que permite ajustar y sincronizar la frecuencia de reloj según las necesidades específicas del sistema embebido. Este ajuste es crucial para manejar múltiples dominios de reloj y asegurar una sincronización precisa, lo cual es fundamental para el funcionamiento correcto del sistema basado en RISC-V.

Tabla 5.1: Utilización de hardware de las implementaciones en FPGA de sistemas caóticos de orden fraccional.

		Tipo de implementación		
Hardware	Disponible	Síntesis estándar 2 enrollamientos	Síntesis estándar 4 × 4 enrollamientos	Sistema embebido basado en RISC-V
LUT	53200	1469 (2.75 %)	2330 (4.38 %)	2528 (4.75 %)
FF	106400	1155 (1.08 %)	1170 (1.10 %)	991 (0.93 %)
DSP	220	12 (5.45 %)	16 (7.27 %)	4 (1.82 %)
BRAM	140	0 (0 %)	0 (0 %)	8 (5.71 %)
IO	200	98 (49.0 %)	98 (49.0 %)	9 (4.50 %)
BUFG	32	1 (3.13 %)	1 (3.13 %)	3 (9.38 %)

5.0.3. Consumo de Energía

Para realizar una descripción y comparación del consumo de energía en las implementaciones de sistemas caóticos de orden fraccional, primero es necesario definir los términos de energía estática y energía dinámica:

- Energía Estática: Corresponde al consumo de energía constante que ocurre independientemente de las operaciones que realice el circuito. Esta energía está relacionada con el consumo pasivo del dispositivo. En todas las implementaciones, el consumo de energía estática es constante y se mantiene en 0.131 W, representando un porcentaje considerable del consumo total.

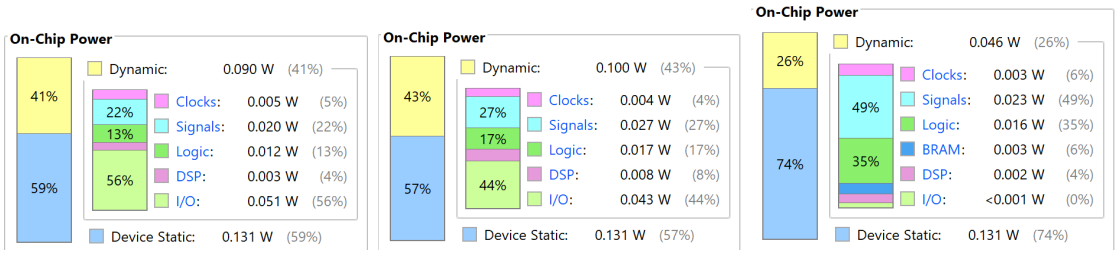
- Energía Dinámica: Este consumo de energía depende de la actividad del circuito, es decir, del número de transiciones lógicas y la frecuencia de reloj. La energía dinámica varía en función de la complejidad de los cálculos y operaciones realizadas por el sistema.

A continuación, se describe y compara el consumo de energía dinámica en las diferentes implementaciones mostradas en la Figura 5.4:

- Síntesis estándar con 2 enrollamientos (Figura 5.4 (a)): La implementación en síntesis estándar del sistema de 2 enrollamientos tiene un consumo dinámico de 0.090 W (41 % del consumo total). La mayor parte de esta energía dinámica se asigna a los componentes de I/O (56 %), debido a la comunicación directa de los datos del sistema caótico. Los componentes de Señales y Lógica también contribuyen significativamente, con un 22 % y un 13 % del consumo dinámico, respectivamente.

- Síntesis estándar con 4×4 enrollamientos (Figura 5.4 (b)): En esta configuración, el consumo dinámico aumenta a 0.100 W (43 % del total), lo que refleja la mayor complejidad del sistema. La distribución de energía dinámica también cambia, con I/O representando el 44 % y Señales el 27 %, lo que sugiere una mayor actividad en el manejo de señales en comparación con el sistema de 2 enrollamientos. El DSP también muestra un incremento en su consumo, pasando del 4 % al 8 %, debido a las operaciones adicionales requeridas para el sistema de 4×4 enrollamientos.

- Sistema embebido basado en RISC-V (Figura 5.4 (c)): En esta implementación, el consumo dinámico se reduce considerablemente a 0.046 W (26 % del total). La mayor parte de esta energía dinámica se destina a Señales (49 %) y Lógica (35 %), reflejando el enfoque optimizado del sistema embebido para manejar operaciones lógicas y de comunicación con menor consumo de I/O (casi insignificante). Este menor consumo de I/O es posible gracias a que el sistema embebido puede gestionar la comunicación de salida internamente, sin necesidad de asignar un gran número de entradas y salidas físicas. En contraste, en las implementaciones de síntesis estándar, es necesario considerar todas las salidas del sistema para adaptarse a diferentes aplicaciones, lo que incrementa el consumo de I/O. Esto resalta la eficiencia del sistema embebido en términos de consumo energético y gestión de recursos.



(a) (b) (c)

Figura 5.4: Consumo de energía de la implementación de sistemas caóticos de orden fraccional. (a) Síntesis estándar 2 enrollamientos, (b) síntesis estándar 4×4 enrollamientos y (c) Implementación en el sistema embebido basado en RISC-V.

5.0.4. Rendimiento Computacional

En esta sección se analiza el rendimiento computacional de las diferentes implementaciones de sistemas caóticos de orden fraccional en términos de throughput o tasa de transferencia de datos. El throughput representa la cantidad de datos procesados por segundo y es un indicador clave del desempeño de un sistema digital en aplicaciones de tiempo real. Para calcular el throughput en cada implementación, se considera el tamaño de la palabra digital en punto fijo, que es de 32 bits, y se multiplica por la frecuencia máxima de reloj de operación del sistema (M) La Tabla 5.2 muestra la comparación del rendimiento computacional para cada implementación:

- Síntesis estándar con 2 enrollamientos: Esta implementación tiene una frecuencia de reloj de 83 MHz, alcanzando un throughput de 2.656 Gbit/s. Esto indica un rendimiento alto en la transmisión de datos, debido a la simplicidad relativa del sistema de 2 enrollamientos, lo que permite una mayor velocidad de procesamiento.

- Síntesis estándar con 4×4 enrollamientos: En esta configuración, la frecuencia de reloj se reduce a 69 MHz, y el throughput disminuye a 2.208 Gbit/s. Este menor rendimiento en comparación con la implementación de 2 enrollamientos es consecuencia de la mayor complejidad del sistema de 4×4 enrollamientos, que demanda más recursos y procesamiento interno, limitando la frecuencia de operación.

- Sistema embebido basado en RISC-V: La implementación embebida basada en RISC-V opera a una frecuencia de 60.6 MHz y tiene un throughput de 1.9392

Gbit/s. Este resultado es menor en comparación con las implementaciones de síntesis estándar debido a que el sistema embebido basado en RISC-V opera a un nivel de abstracción más alto, lo que introduce una mayor complejidad en la administración y control de operaciones.

	Tipo de implementación		
	Síntesis estándar 2 enrollamientos	Síntesis estándar 4×4 enrollamientos	Sistema embebido basado en RISC-V
Clock frequency	83 MHz	69 MHz	60.6 MHz
Throughput	2.656 Gbit/s	2.208 Gbit/s	1.9392 Gbit/s

Tabla 5.2: Comparación del rendimiento computacional entre las diferentes implementaciones de sistemas caóticos de orden fraccional

Aunque esta tasa de transferencia en el diseño embebido basado en RISC-V es releativamente menor a las implementaciones de síntesis estándar, el sistema embebido ofrece la ventaja de que su throughput se mantiene constante independientemente del sistema caótico de orden fraccional a implementar, siempre que este pueda ser representado dentro de una palabra de 32 bits. Esto no se puede decir de las implementaciones basadas en la síntesis estándar, donde cada sistema caótico requiere una arquitectura digital específica cuya complejidad puede aumentar según el sistema, afectando tanto el throughput como el consumo de energía, que también podría incrementarse con sistemas más complejos. Por lo tanto, el diseño embebido basado en RISC-V proporciona ventajas en términos de flexibilidad y escalabilidad, lo cual es crucial en aplicaciones donde la eficiencia energética y la adaptabilidad son más importantes que la máxima velocidad de procesamiento.

Capítulo 6
Conclusiones

En este trabajo se ha abordado la implementación de sistemas caóticos de orden fraccional, utilizando una metodología que combina el desarrollo de un método de descomposición por tramos lineales (PWL-DM) y la implementación en un sistema embebido basado en RISC-V. A continuación, se presentan los principales hallazgos y aportaciones de esta investigación. En primer lugar, el desarrollo matemático realizado se centró en el análisis de operadores de clase F_1 según la clasificación de Teodoro, Machado y Oliveira [71]. Particularmente, se ha mostrado interés por los operadores de Caputo y Grünwald-Letnikov, los cuales permiten preservar las propiedades de herencia y memoria, facilitando la definición del problema de valor inicial de manera coherente, es decir, asegurando que las condiciones iniciales involucren únicamente derivadas de orden entero y no fraccionales.

Al analizar los métodos para resolver sistemas caóticos de orden fraccional, como FDA, ABM y ADM, se evidenciaron varias limitaciones. El método FDA, aunque útil en ciertos contextos, pierde propiedades críticas como la memoria, lo que resulta en una dinámica inconsistente. Esto pone en duda su fiabilidad para representar sistemas fraccionales de clase F_1. Asimismo, el método ABM, a pesar de su precisión, presenta un alto costo computacional debido a su dependencia de memoria infinita, lo que lo hace inviable para su implementación en hardware con recursos limitados.

El método de Grünwald-Letnikov, es un enfoque destacado entre los operadores de clase F_1, es especialmente relevante por su definición discreta de la derivada fraccional, basada en una serie infinita que aproxima la derivada mediante diferencias finitas. Este método captura el efecto de memoria de un sistema al considerar la influencia de todos los estados pasados. Sin embargo, debido al alto costo computacional, se suele aplicar el principio de memoria corta, que trunca la serie infinita a un número finito de términos. Esto hace que el método sea usado en aplicaciones practicas de

sistemas caóticos de orden fraccional, aunque con una posible pérdida de precisión en la representación dinámica del sistema.

El método de ADM se presenta como una opción más adecuada para la aplicación practica de sistemas caóticos de orden fraccional debido a su aproximación y baja complejidad computacional. No obstante, enfrenta desafíos significativos, como la complejidad en la formulación de los polinomios de Adomian y el manejo de valores numéricos, que requieren estrategias de cómputo más complejas. Además, el método ADM presenta limitaciones en la resolución de sistemas basados en funciones PWL, lo que representa una carencia importante. Precisamente en esta área radica su principal área de mejora: la necesidad de un esquema basado en ADM para incluir funciones PWL, que son esenciales en muchos sistemas dinámicos no lineales. A partir de esta necesidad, se propuso el método PWL-DM, que ofrece una estructura robusta para calcular soluciones numéricas en sistemas PWL de orden fraccional con atractores de múltiples enrollamientos. Este método ha demostrado ser aplicable a sistemas PWL de orden fraccional que consisten en funciones PWL con n segmentos, permitiendo que cada función posea una cantidad arbitraria de segmentos afines. Se ha comprobado que la técnica propuesta depende únicamente de operadores lineales y vectores constantes específicos para cada subdominio de la función PWL. Además, las particiones finitas del espacio fase siguen una ley combinatoria, descrita por su producto cartesiano, lo cual conduce a un algoritmo estructurado basado en matrices.

En resumen, el método PWL-DM se posiciona como una herramienta clave para ampliar el alcance de las aplicaciones de ingeniería en sistemas PWL de orden fraccional. Este método no solo ofrece una alternativa para el análisis de sistemas dinámicos, sino que también brinda perspectivas para el desarrollo de aplicaciones prácticas, como el modelado de sistemas y aplicaciones de seguridad.

En cuanto a la realización experimental de sistemas caóticos de orden fraccional en FPGA, el proceso de investigación comenzó con una implementación basada en la síntesis estándar de sistemas caóticos, en la que se destacan varios aspectos clave:

- La necesidad de una estimación y validación de la palabra digital para representar adecuadamente los sistemas caóticos de orden fraccional. Este paso es fundamental para asegurar la precisión en la representación numérica y evitar errores de truncamiento en los cálculos, lo cual es crítico para la dinámica de los sistemas caóticos.

- Un diseño de arquitectura que aproveche las ventajas del cómputo paralelo y el hardware reconfigurable de la FPGA, permitiendo alcanzar un buen rendimiento computacional dentro de recursos de hardware permitidos. Este enfoque posibilita la ejecución de múltiples operaciones de manera simultánea, optimizando así la eficiencia del sistema.

- La complejidad intrinseca de esta metodología, ya que cada sistema caótico implementado requiere de una arquitectura de hardware específica que se adapte a sus necesidades particulares de cómputo. Esto implica que cada nuevo sistema caótico debe ser cuidadosamente analizado y ajustado en términos de arquitectura digital, lo cual aumenta el tiempo de desarrollo y los recursos necesarios.

En cuanto al diseño embebido basado en RISC-V, se adoptó un enfoque diferente que aporta varias ventajas en términos de flexibilidad y escalabilidad:

- El sistema embebido basado en RISC-V no necesita una personalización específica de la arquitectura para cada sistema caótico de orden fraccional. Al operar a un nivel de abstracción más alto, este diseño permite manejar sistemas caóticos diversos sin tener que modificar la arquitectura digital subyacente, siempre y cuando el sistema pueda ser representado en una palabra de 32 bits. Esto implica que el **throughput se mantiene constante** independientemente del sistema caótico, lo que no sucede en las implementaciones de síntesis estándar, donde el rendimiento y el consumo de recursos varían según la complejidad del sistema.

- En términos de consumo de energía, el sistema embebido basado en RISC-V demostró una eficiencia superior al reducir significativamente el uso de entradas y salidas físicas (I/O) gracias a la integración de un protocolo de comunicación interno. Esta estrategia permite manejar la salida de datos desde el propio sistema embebido, evitando el consumo de energía adicional asociado con la conexión directa de múltiples salidas, como ocurre en las implementaciones de síntesis estándar. Esta reducción en el consumo de I/O es beneficiosa en aplicaciones donde la eficiencia energética es prioritaria.

- El diseño embebido basado en RISC-V proporciona una plataforma escalable y adaptable que es capaz de implementar diferentes sistemas caóticos sin requerir grandes modificaciones en el hardware. Esto facilita el desarrollo de sistemas

caóticos más complejos en un entorno flexible y con un bajo costo de hardware, lo cual es ventajoso para aplicaciones que necesitan adaptarse rápidamente a nuevos requisitos o condiciones operativas.

Por lo tanto, mientras que la síntesis estándar en FPGA ofrece un rendimiento computacional alto, presenta una gran complejidad para alcanzar una correcta optimización de recursos , el diseño embebido basado en RISC-V aporta una solución más flexible y escalable, manteniendo un buen equilibrio entre eficiencia energética y velocidad computacional. La combinación del método PWL-DM y la arquitectura embebida RISC-V representa un avance importante en la implementación práctica de sistemas caóticos de orden fraccional, permitiendo aplicaciones más versátiles y eficientes en áreas como el modelado de sistemas complejos y la seguridad. Este enfoque abre nuevas posibilidades para el desarrollo de sistemas caóticos en plataformas embebidas, ofreciendo una base sólida para futuras investigaciones y aplicaciones de ingeniería.

Apéndices

AIMS Mathematics, 7(4): 5871–5894.
DOI: 10.3934/math.2022326
Received: 03 October 2021
Revised: 24 December 2021
Accepted: 31 December 2021
Published: 12 January 2022

http://www.aimspress.com/journal/Math

Research article

Poincaré maps for detecting chaos in fractional-order systems with hidden attractors for its Kaplan-Yorke dimension optimization

Daniel Clemente-López[1], Esteban Tlelo-Cuautle[1,*], Luis-Gerardo de la Fraga[2], José de Jesús Rangel-Magdaleno[1] and Jesus Manuel Munoz-Pacheco[3]

[1] Department of Electronics, Instituto Nacional de Astrofísica, Optica y Electrónica (INAOE), Luis Enrique Erro No. 1, Tonantzintla, Puebla 72840, Mexico
[2] Computer Science Department, Cinvestav, Av. IPN 2508, Mexico City 07360, Mexico
[3] Facultad de Ciencias de la Electrónica, Benemérita Universidad Autónoma de Puebla, Ciudad Universitaria, 18 Sur y Avenida San Claudio San Manuel, Puebla 72592, Mexico

* **Correspondence:** Email: etlelo@inaoep.mx; Tel: +522222663100.

Abstract: The optimization of fractional-order (FO) chaotic systems is challenging when simulating a considerable number of cases for long times, where the primary problem is verifying if the given parameter values will generate chaotic behavior. In this manner, we introduce a methodology for detecting chaotic behavior in FO systems through the analysis of Poincaré maps. The optimization process is performed applying differential evolution (DE) and accelerated particle swarm optimization (APSO) algorithms for maximizing the Kaplan-Yorke dimension (D_{KY}) of two case studies: a 3D and a 4D FO chaotic systems with hidden attractors. These FO chaotic systems are solved applying the Grünwald-Letnikov method, and the Numba just-in-time (jit) compiler is used to improve the optimization process's time execution in Python programming language. The optimization results show that the proposed method efficiently optimizes FO chaotic systems with hidden attractors while saving execution time.

Keywords: chaos; fractional calculus; Poincaré map; differential evolution algorithm; accelerated particle swarm optimization; Kaplan-Yorke dimension
Mathematics Subject Classification: 26A33, 90C29

1. Introduction

Chaos theory is the branch of mathematics that deals with complex behavior emerging in dynamical systems [1]. Chaotic systems possess special features such as extreme sensitivity to initial conditions, random-like unpredictable behavior and ergodicity. Improving these characteristics

Journal of Ambient Intelligence and Humanized Computing (2023) 14:11023–11037
https://doi.org/10.1007/s12652-022-04382-y

ORIGINAL RESEARCH

A comparison of embedded and non-embedded FPGA implementations for fractional chaos-based random number generators

D. Clemente-Lopez[1] · J. J. Rangel-Magdaleno[1] · J. M. Munoz-Pacheco[2] · L. Morales-Velazquez[3]

Received: 4 November 2021 / Accepted: 30 July 2022 / Published online: 4 September 2022

Abstract
Fractional-order chaotic systems are a hot topic of research due to their applicability in all related fields of science and engineering. For instance, they have been pointed out as a potential solution for security in smart cities and Internet of Things networks through data encryption using random number generators. However, for security applications, one of the main challenges is the physical realization of fractional-order chaotic systems using digital platforms. To overcome this, several works have proposed FPGA technology for implementing complex systems with high computational performance. Nevertheless, the FPGA-based fractional-order chaotic systems implementations require a suitable trade-off between speed performance and hardware cost, especially in embedded approaches that regularly incorporate a system-on-a-chip micro-controller. In this paper, a comparative analysis between the embedded and non-embedded FPGA-based designs for implementing a three-dimensional fractional-order chaotic system and a chaos-based true random number generators is reported. First, we compute a semi-analytical solution of the fractional-order chaotic systems using the Adomian decomposition method, the obtained chaotic time series serves as a random source for the true random number generators. Then, the configuration, resource usage, speed performance, and power consumption of the implementations are tested on a Xilinx Zynq-7000 XC7Z020 system-on-a-chip and an xQuP01v0 FPGA-based processor. The results reveal that while the non-embedded approach showed improved efficiency between cost and performance, the embedded method on the xQuP01v0 FPGA-based processor presented an attractive lower cost-power consumption option against commercial processors.

Keywords Embedded systems · True random number generator · Security · FPGA implementation · Fractional-order · Chaotic system

1 Introduction

Chaos theory is the branch of mathematics that deals with complex behavior emerging in dynamical systems. Due to chaotic systems posses special features such as extreme sensitivity to initial conditions, random-like unpredictable behavior and ergodicity, they have found applications in almost every field of science and engineering, such as biology (Dubey et al. 2021), mechanics (Kaviya et al. 2020), economics (Idowu et al. 2018), secure communications (Wang et al. 2016), cryptography (Arroyo et al. 2017), robotics (Zang et al. 2016), control and so forth. On the other hand, fractional calculus refers to the generalization of integer-order integrals and derivatives ($n \in \mathbb{Z}$) to arbitrary orders ($q \in \mathbb{R}$). Although this is a mathematical topic proposed over 300 years ago, only recently the fractional calculus emerged as a research hot topic for having superior

D. Clemente-Lopez, J. J. Rangel-Magdaleno, J.M. Munoz-Pacheco and L. Morales-Velazquez contributed equally to this work.

✉ J. J. Rangel-Magdaleno
jrangel@inaoe.mx

[1] Digital Systems Group, Department of Electronics, Instituto Nacional de Astrofisica, Optica y Electronica (INAOE), Luis Enrique Erro No. 1, Tonantzintla, 72840 Puebla, Mexico

[2] Faculty of Electronics Sciences, Benemérita Universidad Autónoma de Puebla, 18 Sur y Avenida San Claudio, 72570 Puebla, Puebla, Mexico

[3] Faculty of Engineering, Universidad Autónoma de Querétaro, Campus San Juan del Rio, 76803 San Juan del Rio, Querétaro, Mexico

Archives of Computational Methods in Engineering
https://doi.org/10.1007/s11831-022-09824-6

REVIEW ARTICLE

A Review of the Digital Implementation of Continuous-Time Fractional-Order Chaotic Systems Using FPGAs and Embedded Hardware

Daniel Clemente-López[1] · Jesus M. Munoz-Pacheco[2] · Jose de Jesus Rangel-Magdaleno[1]

Received: 5 July 2022 / Accepted: 19 September 2022

Abstract
The hallmark of fractional-order derivatives is a memory kernel to describe real-world phenomena with a better approximation than classical calculus. In fractional-order chaotic systems, the memory kernel improves their complexity, ergodicity, and hidden dynamical behaviors, which become an excellent option to boost applications in data encryption, IoT security, random number generators, and neural networks. From an engineering point of view, the challenge consists of getting feasible electronic implementations of fractional-order chaotic systems on FPGAs and embedded hardware. However, successful implementation requires suitable numerical methods to reduce computational cost and hardware resources while the memory kernel length preserves a relatively large amount of data. This article comprehensively reviews the FPGA-based and embedded physical realization of fractional-order continuous-time chaotic systems under singular kernel fractional derivatives. In particular, the main numerical algorithms for computing a solution of the fractional-order continuous-time chaotic systems are in-depth studied to evidence their computational cost (simulations time, memory data storage, hardware resources, and accuracy) when implemented on digital hardware. We also analyze and demonstrate two methodologies for obtaining suitable digital implementations using FPGAs and embedded hardware by considering the fixed- and floating-point digital arithmetic representation, hardware architecture, programming languages, and power consumption and resources performance. Finally, we discuss the existing open problems in both computational and hardware and forecast the research opportunities in this exciting scientific area.

1 Introduction

Chaos theory is the branch of mathematics that deals with extreme sensitivity to initial conditions and complex behavior emerging in nonlinear dynamical systems [1]. The resulting chaotic systems have been applied in almost every field of science and engineering, such as biology [2], astrophysics [3], mechanics [4], economics [5], secure communications [6], cryptography [7], robotics [8], control [9] and so forth.

On the other hand, *Fractional Calculus* refers to the generalization of integer-order integrals and derivatives ($n \in \mathbb{Z}$) to arbitrary orders ($q \in \mathbb{R}$). Although this is a mathematical topic proposed over 300 years ago, only recently the fractional calculus emerged as a research hot topic for having superior characteristics for modeling complex behaviors [10]. It is well known that fractional-order definitions possesses nonlocality and long memory properties that provide a more accurate description of several physical phenomena than the integer-order counterparts (for instance see: biological processes [11], economics [12], mechanics [13], electronics [14], social phenomena [15]). A seminal work published by Zhou et al. [16] presented a set of cases to reveal the importance of fractional calculus in physical processes such as in nonlinear oscillators and spatiotemporal systems. For instance, nonuniform damping force, field coupling and boundary effect, nonuniform diffusion,

D. Clemente-López, J. M. Munoz-Pacheco and J.J. Rangel-Magdaleno have contributed equally to this work.

✉ Jesus M. Munoz-Pacheco
jesusm.pacheco@correo.buap.mx

[1] Department of Electronics, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Luis Enrique Erro No. 1, 72840 Tonantzintla, Puebla, Mexico

[2] Faculty of Electronics Sciences, Benemérita Universidad Autónoma de Puebla, Av. San Claudio y 18 Sur, 72570 Puebla, Puebla, Mexico

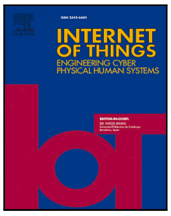
Published online: 14 October 2022

Internet of Things 25 (2024) 101032

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot



Research article

A lightweight chaos-based encryption scheme for IoT healthcare systems

Daniel Clemente-Lopez [a], Jose de Jesus Rangel-Magdaleno [a,*], Jesus Manuel Muñoz-Pacheco [b]

[a] Digital System Group - Instituto Nacional de Astrofísica Optica y Electrónica, Luis Erique Erro #1, Tonantzintla, Puebla, 72840, Mexico
[b] Faculty of Electronics Sciences, Benemerita Universidad Autonoma de Puebla, 4 Sur 104, Colonia Centro, C.P. 72000, Puebla, Mexico

ARTICLE INFO

Keywords:
Chaos
Encryption
IoT

ABSTRACT

The Internet of Things (IoT) has transformed the healthcare industry by enabling new services and capabilities through connected devices and sensors. These devices, such as smartwatches and fitness tracker bands, can monitor various health metrics such as heart rate, blood pressure, and sleep quality. However, the security of these systems is a critical concern, as the unauthorized disclosure or access of healthcare information could have serious consequences for patients. This information can include personal and medical data, diagnostic and treatment information, as well as private locations. Lightweight encryption schemes are commonly used in IoT systems to protect this sensitive data. These schemes are designed to be fast and efficient, allowing them to encrypt and decrypt data in real-time, which is important for systems with limited computing power or storage capacity. In the last decade, there has been a significant increase in research and development in chaos cryptography. Due to chaotic systems' unpredictable and sensitive nature, they can provide robust cryptographic schemes for data transmission between embedded devices. This makes it difficult for attackers to intercept and manipulate the data. Therefore, this work proposes a chaos-based lightweight encryption scheme for IoT healthcare systems with a primary application in the encryption of wearable devices. The proposed encryption scheme is based on a $2D$ 4-scroll chaotic attractor system, uniquely characterized for this work. The scheme is tested on an ARM-based microcontroller for encrypting PPG (Photoplethysmogram) biosignal data. The obtained results show that the chaos-based lightweight encryption scheme effectively improves the security of healthcare IoT systems while maintaining the real-time flow of data.

1. Introduction

The Internet of Things (IoT) is an arrangement of interrelated embedded systems, sensors, actuators, and interconnected processing gadgets for transmitting data over the Internet [1]. As these devices can operate autonomously, IoT technology enables automation, real-time data collection, and decision-making without relying on constant human input. Moreover, with the extensive coverage of wireless networks and continuous progress in chip computers, it has become possible to integrate IoT in almost every field of production and human life [2]. For instance, home automation, surveillance systems, traffic monitoring, transportation, agricultural production, and healthcare are IoT application environments.

* Corresponding author.
E-mail addresses: daniel.clemente@inaoep.mx (D. Clemente-Lopez), jrangel@inaoe.mx (J. Rangel-Magdaleno), jesusm.pacheco@correo.buap.mx (J. Muñoz-Pacheco).

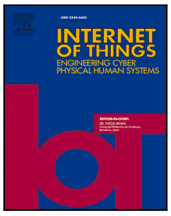
https://doi.org/10.1016/j.iot.2023.101032
Received 7 September 2023; Received in revised form 19 October 2023; Accepted 11 December 2023
Available online 20 December 2023

Internet of Things 25 (2024) 101116

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot



Research article

Experimental validation of IoT image encryption scheme based on a 5-D fractional hyperchaotic system and Numba JIT compiler

Daniel Clemente-López [a,1], Jesus M. Munoz-Pacheco [b,*,1], José de Jesus Rangel-Magdaleno [a,1]

[a] Department of Electronics, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Luis Enrique Erro No. 1, Tonantzintla, Puebla 72840, Mexico

[b] Faculty of Electronics Sciences, Benemérita Universidad Autónoma de Puebla, Av. San Claudio y 18 Sur, Puebla, Puebla 72570, Mexico

ARTICLE INFO

Keywords:
Encryption algorithm
Chaos
Multi-scroll
Fractional-order derivative
ARM processor
Multiprocessing
Just in time compiler
Chaotic systems

ABSTRACT

In the realm of Internet of Things (IoT) systems, the interconnectivity of physical hardware devices is a fundamental aspect, and as a result, data exchange assumes a critical role in the network. Given the sensitivity of such information, it is imperative to adopt appropriate measures to encrypt the data to safeguard it from unauthorized access. It is, therefore, paramount to prove novel encryption algorithms at an experimental level. To overcome that, a potential solution is fractional-order multi-scroll chaotic systems, which provide an extra degree of freedom to enhance ergodicity and random-like behaviors, which can help to improve encryption keyspace to get robust ciphers. However, the numerical algorithms to obtain a fractional chaotic series increase the computational cost because they demand extensive simulation time to model the memory of the standard fractional derivatives, limiting high-speed encryption. In this framework, a fast encryption scheme using a 5D fractional-order (FO) hyper-chaotic multi-scroll (HCMS) system is proposed and verified experimentally. Based on multiprocessing strategies and the Numba just-in-time (JIT) compiler, the Python code that describes the FO-HCMS system is optimized, which enables image encryption in real-time. The physical implementation of the encryption scheme is performed on an Advance RISC Machine (ARM) processor, and it is applied for image encryption on a machine-to-machine (M2M) communication using the message queuing telemetry transport (MQTT) protocol. The obtained results indicate that the proposed encryption scheme can reach encryption throughputs of up to 19.891 Mbps on an ARM with a 1.4 GHz CPU and 77.864 Mbps on a PC with a 3.1 GHz CPU.

1. Introduction

Chaotic systems possess special features such as extreme sensitivity to initial conditions, random-like behavior, ergodicity, and so on. Therefore, they are amply used in almost every field of science and engineering. For instance, the chaotic systems have been regularly utilized for generating pseudo-random number generators (PRNGs), which are the basis for data security applications like encryption and authentication algorithms [1,2].

On the other hand, the long-memory and nonlocality attributes of classical fractional-order derivatives have become excellent options for studying complex phenomena with improved accuracy [3,4]. From an engineering point of view, the fractional order

* Corresponding author.
E-mail address: jesusm.pacheco@correo.buap.mx (J.M. Munoz-Pacheco).

[1] All authors contributed equally to this study.

https://doi.org/10.1016/j.iot.2024.101116
Received 21 November 2023; Received in revised form 17 January 2024; Accepted 6 February 2024
Available online 10 February 2024

fractal and fractional

MDPI

Article

A Piecewise Linear Approach for Implementing Fractional-Order Multi-Scroll Chaotic Systems on ARMs and FPGAs

Daniel Clemente-López [1], Jesus M. Munoz-Pacheco [2,*], Ernesto Zambrano-Serrano [3], Olga G. Félix Beltrán [2] and Jose de Jesus Rangel-Magdaleno [1]

[1] Department of Electronics, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Luis Enrique Erro No. 1, Tonantzintla, Puebla 72840, Mexico; daniel.clemente@susu.inaoep.mx (D.C.-L.); jrangel@inaoe.mx (J.d.J.R.-M.)
[2] Faculty of Electronics Sciences, Benemérita Universidad Autónoma de Puebla, Av. San Claudio y 18 Sur, Puebla 72570, Mexico; olga.felix@correo.buap.mx
[3] Facultad de Ingeniería Mecánica y Eléctrica, Universidad Autónoma de Nuevo León, San Nicolás de los Garza 66455, Mexico; ernesto.zambranos@uanl.edu.mx
* Correspondence: jesusm.pacheco@correo.buap.mx

Abstract: This manuscript introduces a piecewise linear decomposition method devoted to a class of fractional-order dynamical systems composed of piecewise linear (PWL) functions. Inspired by the Adomian decomposition method, the proposed technique computes an approximated solution of fractional-order PWL systems using only linear operators and specific constants vectors for each sub-domain of the PWL functions, with no need for the Adomian polynomials. The proposed decomposition method can be applied to fractional-order PWL systems composed of nth PWL functions, where each PWL function may have any number of affine segments. In particular, we demonstrate various examples of how to solve fractional-order systems with 1D 2-scroll, 4-scroll, and 4×4-grid scroll chaotic attractors by applying the proposed approach. From the theoretical and implementation results, we found the proposed approach eliminates the unneeded terms, has a low computational cost, and permits a straightforward physical implementation of multi-scroll chaotic attractors on ARMs and FPGAs digital platforms.

Keywords: chaotic systems; fractional-order; multi-scroll attractors; ARM implementation; FPGA implementation

Citation: Clemente-López, D.; Munoz-Pacheco, J.M.; Zambrano-Serrano, E.; Félix Beltrán, O.G.; Rangel-Magdaleno, J.d.J. A Piecewise Linear Approach for Implementing Fractional-Order Multi-Scroll Chaotic Systems on ARMs and FPGAs. *Fractal Fract.* **2024**, *8*, 389. https://doi.org/10.3390/fractalfract8070389

Academic Editor: António Lopes

Received: 7 April 2024
Revised: 17 June 2024
Accepted: 25 June 2024
Published: 29 June 2024

1. Introduction

Fractional calculus is the branch of mathematics that studies the non-integer order integrals and derivatives, called fractional derivatives and fractional integrals [1]. Nowadays, we can found several applications of fractional calculus in diverse fields of science [2–5] and engineering [6–9]. Among them, fractional-order chaotic systems are of great interest because they represent the intersection of chaos theory with fractional calculus [10,11], forming a dynamical system with non-local characteristics and high complexity [12–15].

The interest in fractional-order systems is driven by their advantages over traditional integer-order systems. While integer-order systems have been fundamental in understanding the chaotic dynamics, as demonstrated by foundational works such as those by Pecora and Carroll on synchronization of chaotic systems [16] and Ott, Grebogi, and Yorke on controlling chaos [17], they have limitations in describing dynamical systems with memory effects [18]. On the other hand, Fractional-order systems, possess memory properties and hereditary characteristics, which are absent in integer-order systems, and hence, they provide a more comprehensive framework for modeling and analyzing the non-local interactions of dynamical systems. These properties also improve the effectiveness for chaos-based applications as follows,

Bibliografía

[1] H. Dai, X. Yue, D. Xie, y S. N. Atluri, "Chaos and chaotic transients in an aeroelastic system," *Journal of Sound and Vibration*, vol. 333, no. 26, pp. 7267–7285, 2014.

[2] V. D. Tô, "A note on devaney's definition of chaos," *Journal of Dynamical Systems and Geometric Theories*, vol. 2, no. 1, pp. 23–26, 2004.

[3] J. K. Zink, K. Batygin, y F. C. Adams, "The great inequality and the dynamical disintegration of the outer solar system," *The Astronomical Journal*, vol. 160, no. 5, p. 232, 2020.

[4] M. Sajid, "Recent developments on chaos in mechanical systems," *International Journal on Theoretical and Applied Research in Mechanical Engineering*, vol. 2, no. 3, pp. 121–124, 2013.

[5] B. A. Idowu, S. Vaidyanathan, A. Sambas, O. I. Olusola, y O. Onma, "A new chaotic finance system: Its analysis, control, synchronization and circuit design," en *Nonlinear Dynamical Systems with Self-Excited and Hidden Attractors*, pp. 271–295, Springer, 2018.

[6] B. Wang, S. Zhong, y X. Dong, "On the novel chaotic secure communication scheme design," *Communications in Nonlinear Science and Numerical Simulation*, vol. 39, pp. 108–117, 2016.

[7] D. Arroyo, F. Hernandez, y A. B. Orúe, "Cryptanalysis of a classical chaos-based cryptosystem with some quantum cryptography features," *International Journal of Bifurcation and Chaos*, vol. 27, no. 01, p. 1750004, 2017.

[8] X. Zang, S. Iqbal, Y. Zhu, X. Liu, y J. Zhao, "Applications of chaotic dynamics in robotics," *International Journal of Advanced Robotic Systems*, vol. 13, no. 2, p. 60, 2016.

[9] C. Tian, Kun rehman2018noteand Grebogi y H.-P. Ren, "Chaos generation with impulse control: Application to non-chaotic systems and circuit design," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2021.

[10] H. Natiq, M. Said, M. Ariffin, S. He, L. Rondoni, y S. Banerjee, "Self-excited and hidden attractors in a novel chaotic system with complicated multistability," *The European Physical Journal Plus*, vol. 133, no. 12, pp. 1–12, 2018.

[11] G. A. Leonov y N. V. Kuznetsov, "Hidden attractors in dynamical systems. from hidden oscillations in hilbert–kolmogorov, aizerman, and kalman problems to hidden chaotic attractor in chua circuits," *International Journal of Bifurcation and Chaos*, vol. 23, no. 01, p. 1330002, 2013.

[12] P. Sharma, M. Shrimali, A. Prasad, N. Kuznetsov, y G. Leonov, "Control of multistability in hidden attractors," *The European Physical Journal Special Topics*, vol. 224, no. 8, pp. 1485–1491, 2015.

[13] A. K. Farhan, R. S. Ali, H. Natiq, y N. M. Al-Saidi, "A new s-box generation algorithm based on multistability behavior of a plasma perturbation model," *IEEE Access*, vol. 7, pp. 124914–124924, 2019.

[14] F. Yu, Z. Zhang, L. Liu, H. Shen, Y. Huang, C. Shi, S. Cai, Y. Song, S. Du, y Q. Xu, "Secure communication scheme based on a new 5d multistable four-wing memristive hyperchaotic system with disturbance inputs," *Complexity*, vol. 2020, 2020.

[15] A. Khan, J. Gómez-Aguilar, T. Abdeljawad, y H. Khan, "Stability and numerical simulation of a fractional order plant-nectar-pollinator model," *Alexandria Engineering Journal*, vol. 59, no. 1, pp. 49–59, 2020.

[16] V. E. Tarasov, "On history of mathematical economics: Application of fractional calculus," *Mathematics*, vol. 7, no. 6, p. 509, 2019.

[17] T. M. Atanackovic, S. Pilipovic, B. Stankovic, y D. Zorica, *Fractional calculus with applications in mechanics: vibrations and diffusion processes*. John Wiley & Sons, 2014.

[18] M. Khan, A. Rasheed, M. S. Anwar, Z. Hussain, y T. Shahzad, "Modelling charge carrier transport with anomalous diffusion and heat conduction in amorphous semiconductors using fractional calculus," *Physica Scripta*, vol. 96, no. 4, p. 045204, 2021.

[19] W. Liu y K. Chen, "Chaotic behavior in a new fractional-order love triangle system with competition," *J. Appl. Anal. Comput*, vol. 5, no. 1, pp. 103–113, 2015.

[20] T. T. Hartley, C. F. Lorenzo, y H. K. Qammer, "Chaos in a fractional order chua's system," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 42, no. 8, pp. 485–490, 1995.

[21] I. Grigorenko y E. Grigorenko, "Chaotic dynamics of the fractional lorenz system," *Physical review letters*, vol. 91, no. 3, p. 034101, 2003.

[22] C. Li y G. Chen, "Chaos in the fractional order chen system and its control," *Chaos, Solitons & Fractals*, vol. 22, no. 3, pp. 549–554, 2004.

[23] G. Zaslavsky, A. Stanislavsky, y M. Edelman, "Chaotic and pseudochaotic attractors of perturbed fractional oscillator," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 16, no. 1, p. 013102, 2006.

[24] M. Ahmad, U. Shamsi, e I. R. Khan, "An enhanced image encryption algorithm using fractional chaotic systems," *Procedia Computer Science*, vol. 57, pp. 852–859, 2015.

[25] M. Bettayeb, U. M. Al-Saggaf, y S. Djennoune, "Single channel secure communication scheme based on synchronization of fractional-order chaotic chua's systems," *Transactions of the Institute of Measurement and Control*, vol. 40, no. 13, pp. 3651–3664, 2018.

[26] J. Zhao, S. Wang, Y. Chang, y X. Li, "A novel image encryption scheme based on an improper fractional-order chaotic system," *Nonlinear Dynamics*, vol. 80, no. 4, pp. 1721–1729, 2015.

[27] J. Hao, H. Li, H. Yan, y J. Mou, "A new fractional chaotic system and its application in image encryption with dna mutation," *IEEE Access*, vol. 9, pp. 52364–52377, 2021.

[28] H. Jahanshahi, A. Yousefpour, J. M. Munoz-Pacheco, S. Kacar, V.-T. Pham, y F. E. Alsaadi, "A new fractional-order hyperchaotic memristor oscillator: Dynamic analysis, robust adaptive synchronization, and its application to voice encryption," *Applied Mathematics and Computation*, vol. 383, p. 125310, 2020.

[29] N. R. Babu, M. Kalpana, y P. Balasubramaniam, "A novel audio encryption approach via finite-time synchronization of fractional order hyperchaotic system," *Multimedia Tools and Applications*, vol. 80, no. 12, pp. 18043–18067, 2021.

[30] R. Montero-Canela, E. Zambrano-Serrano, E. I. Tamariz-Flores, J. M. Muñoz-Pacheco, y R. Torrealba-Meléndez, "Fractional chaos based-cryptosystem for generating encryption keys in ad hoc networks," *Ad Hoc Networks*, vol. 97, p. 102005, 2020.

[31] N. Fataf, M. A. Rahim, S. He, y S. Banerjee, "A communication scheme based on fractional order chaotic laser for internet of things," *Internet of Things*, p. 100425, 2021.

[32] K. Sridharan y Z. N. Ahmadabadi, "A multi-system chaotic path planner for fast and unpredictable online coverage of terrains," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5268–5275, 2020.

[33] F. B. Duarte y J. T. Machado, "Chaotic phenomena and fractional-order dynamics in the trajectory control of redundant manipulators," *Nonlinear Dynamics*, vol. 29, no. 1, pp. 315–342, 2002.

[34] S. Kumar, A. E. Matouk, H. Chaudhary, y S. Kant, "Control and synchronization of fractional-order chaotic satellite systems using feedback and adaptive control techniques," *International Journal of Adaptive Control and Signal Processing*, vol. 35, no. 4, pp. 484–497, 2021.

[35] J. M. Munoz-Pacheco, E. Zambrano-Serrano, C. Volos, S. Jafari, J. Kengne, y K. Rajagopal, "A new fractional-order chaotic system with different families of hidden and self-excited attractors," *Entropy*, vol. 20, no. 8, p. 564, 2018.

[36] Debbouche, Nadjette and Momani, Shaher and Ouannas, Adel and Grassi, Giuseppe and Dibi, Zohir and Batiha, Iqbal M and others, "Generating multidirectional variable hidden attractors via newly commensurate and incommensurate non-equilibrium fractional-order chaotic systems," *Entropy*, vol. 23, no. 3, p. 261, 2021.

[37] W. S. Sayed y A. G. Radwan, "Self-reproducing hidden attractors in fractional-order chaotic systems using affine transformations," *IEEE Open Journal of Circuits and Systems*, vol. 1, pp. 243–254, 2020.

[38] E. Tlelo-Cuautle, A. D. Pano-Azucena, O. Guillén-Fernández, y A. Silva-Juárez, *Analog/Digital Implementation of Fractional Order Chaotic Circuits and Applications*. Springer, 2020.

[39] C. Muñiz-Montero, L. V. García-Jiménez, L. A. Sánchez-Gaspariano, C. Sánchez-López, V. R. González-Díaz, y E. Tlelo-Cuautle, "New alternatives for analog implementation of fractional-order integrators, differentiators and pid controllers based on integer-order integrators," *Nonlinear Dynamics*, vol. 90, no. 1, pp. 241–256, 2017.

[40] A. G. Radwan, A. Soliman, A. S. Elwakil, y A. Sedeek, "On the stability of linear systems with fractional-order elements," *Chaos, Solitons & Fractals*, vol. 40, no. 5, pp. 2317–2328, 2009.

[41] A. Tepljakov, E. Petlenkov, y J. Belikov, "Efficient analog implementations of fractional-order controllers," en *Proceedings of the 14th International Carpathian Control Conference (ICCC)*, pp. 377–382, IEEE, 2013.

[42] L. Dorvcák, J. Valsa, E. Gonzalez, J. Terpák, I. Petrávs, y L. Pivka, "Analogue realization of fractional-order dynamical systems," *Entropy*, vol. 15, no. 10, pp. 4199–4214, 2013.

[43] A. S. Mansingka, M. A. Zidan, M. L. Barakat, A. G. Radwan, y K. N. Salama, "Fully digital jerk-based chaotic oscillators for high throughput pseudo-random number generators up to 8.77 gbits/s," *Microelectronics Journal*, vol. 44, no. 9, pp. 744–752, 2013.

[44] L. Xiong, Y.-J. Lu, Y.-F. Zhang, X.-G. Zhang, y P. Gupta, "Design and hardware implementation of a new chaotic secure communication technique," *PloS one*, vol. 11, no. 8, 2016.

[45] Y. Xu y Z. He, "The short memory principle for solving abel differential equation of fractional order," *Computers & Mathematics with Applications*, vol. 62, no. 12, pp. 4796–4805, 2011.

[46] R. Rivera-Blas, S. A. Rodríguez Paredes, L. A. Flores-Herrera, e I. Adrián Romero, "Design and implementation of a microcontroller based active controller for the synchronization of the petrzela chaotic system," *Computation*, vol. 7, no. 3, p. 40, 2019.

[47] J.-S. Lin, C.-F. Huang, T.-L. Liao, y J.-J. Yan, "Design and implementation of digital secure communication based on synchronized chaotic systems," *Digital Signal Processing*, vol. 20, no. 1, pp. 229–237, 2010.

[48] M. Qiu, S. Yu, Y. Wen, J. Lü, J. He, y Z. Lin, "Design and fpga implementation of a universal chaotic signal generator based on the verilog hdl fixed-point algorithm and state machine control," *International Journal of Bifurcation and Chaos*, vol. 27, no. 03, p. 1750040, 2017.

[49] E. Zambrano-Serrano, J. Muñoz-Pacheco, y E. Campos-Cantón, "Chaos generation in fractional-order switched systems and its digital implementation," *AEU-International Journal of Electronics and Communications*, vol. 79, pp. 43–52, 2017.

[50] X. Zhang, S. Yu, P. Chen, J. Lü, J. He, y Z. Lin, "Design and arm-embedded implementation of a chaotic secure communication scheme based on h. 264 selective encryption," *Nonlinear Dynamics*, vol. 89, no. 3, pp. 1949–1965, 2017.

[51] D. Clemente-López, J. M. Muñoz-Pacheco, O. G. Félix-Beltrán, y C. Volos, "Efficient computation of the grünwald-letnikov method for arm-based implementations of fractional-order chaotic systems," en *2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pp. 1–4, IEEE, 2019.

[52] K. Rajagopal, A. Karthikeyan, y P. Duraisamy, "Bifurcation analysis and chaos control of a fractional order portal frame with nonideal loading using adaptive sliding mode control," *Shock and Vibration*, vol. 2017, 2017.

[53] M. F. Tolba, A. M. AbdelAty, N. S. Soliman, L. A. Said, A. H. Madian, A. T. Azar, y A. G. Radwan, "Fpga implementation of two fractional order chaotic systems," *AEU-International Journal of Electronics and Communications*, vol. 78, pp. 162–172, 2017.

[54] D. K. Shah, R. B. Chaurasiya, V. A. Vyawahare, K. Pichhode, y M. D. Patil, "Fpga implementation of fractional-order chaotic systems," *AEU-International Journal of Electronics and Communications*, vol. 78, pp. 245–257, 2017.

[55] A. G. Soriano-Sánchez, C. Posadas-Castillo, M. A. Platas-Garza, y A. Arellano-Delgado, "Synchronization and fpga realization of complex networks with fractional–order liu chaotic oscillators," *Applied Mathematics and Computation*, vol. 332, pp. 250–262, 2018.

[56] A. D. Pano-Azucena, E. Tlelo-Cuautle, J. M. Muñoz-Pacheco, y L. G. de la Fraga, "Fpga-based implementation of different families of fractional-order chaotic oscillators applying grünwald–letnikov method," *Communications in Nonlinear Science and Numerical Simulation*, vol. 72, pp. 516–527, 2019.

[57] P. Dutta, G. Upendra, E. Giribabu, B. Sridharan, y V. Tyagi, "A comprehensive review of embedded system design aspects for rural application platform," *International Journal of Computer Applications*, vol. 106, no. 11, 2014.

[58] P. U. Chavan, M. Murugan, y P. P. Chavan, "Hardware and software architecture for embedded distributed control system using adaptive hybrid communication channel," en *Advanced Computing and Communication Technologies*, pp. 513–520, Springer, 2016.

[59] X. Ren y Y. Wang, "Design of a fpga hardware architecture to detect real-time moving objects using the background subtraction algorithm," en *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*, pp. 428–433, IEEE, 2016.

[60] M. Nahas y A. Maaita, "Choosing appropriate programming language to implement software for real-time resource-constrained embedded systems," *Embedded Systems-Theory and Design Methodology*, 2012.

[61] K. Jaskolka, J. Seiler, F. Beyer, y A. Kaup, "A python-based laboratory course for image and video signal processing on embedded systems," *Heliyon*, vol. 5, no. 10, p. e02560, 2019.

[62] L. A. Ajao, M. Adegboye, J. Agajo, A. Ajao, y A. Yunus, "Fpga logic circuit implementation and synthesis with vhdl programming: A learning approach," *International Journal of Computer Science & Communications*, vol. 2, no. 1, pp. 1–11, 2017.

[63] A. Romanov, M. Romanov, y A. Kharchenko, "Fpga-based control system reconfiguration using open source software," en *2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pp. 976–981, IEEE, 2017.

[64] A. F. Lorenzon, A. L. Sartor, M. C. Cera, y A. C. S. Beck, "The influence of parallel programming interfaces on multicore embedded systems," en *2015 IEEE 39th Annual Computer Software and Applications Conference*, vol. 2, pp. 617–625, IEEE, 2015.

[65] O. O. Oyetoke y A. Adedapo, "'embedded systems engineering, the future of our technology world; a look into the design of optimized energy metering devices'," *International Journal of Recent Engineering Science (IJRES). volume18, pp1 December*, 2015.

[66] V. Ivanov y E. Nosov, "Serial communication protocol for fpga-based systems," en *Journal of Physics: Conference Series*, vol. 1326, p. 012044, IOP Publishing, 2019.

[67] B. Vinagre, I. Petras, P. Merchan, y L. Dorcak, "Two digital realizations of fractional controllers: Application to temperature control of a solid," en *2001 European Control Conference (ECC)*, pp. 1764–1767, IEEE, 2001.

[68] D. Clemente-López, J. Rangel-Magdaleno, J. M. Muñoz-Pacheco, y L. Morales-Velazquez, "A comparison of embedded and non-embedded fpga implementa-

tions for fractional chaos-based random number generators," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 8, pp. 11023–11037, 2023.

[69] D. Clemente-López, J. M. Munoz-Pacheco, y J. de Jesus Rangel-Magdaleno, "Experimental validation of iot image encryption scheme based on a 5-d fractional hyperchaotic system and numba jit compiler," *Internet of Things*, p. 101116, 2024.

[70] D. Clemente-Lopez, J. de Jesus Rangel-Magdaleno, y J. M. Muñoz-Pacheco, "A lightweight chaos-based encryption scheme for iot healthcare systems," *Internet of Things*, vol. 25, p. 101032, 2024.

[71] G. S. Teodoro, J. T. Machado, y E. C. De Oliveira, "A review of definitions of fractional derivatives and other operators," *Journal of Computational Physics*, vol. 388, pp. 195–208, 2019.

[72] B. Ross, "A brief history and exposition of the fundamental theory of fractional calculus," en *Fractional calculus and its applications*, pp. 1–36, Springer, 1975.

[73] M. D. Ortigueira y J. T. Machado, "What is a fractional derivative?," *Journal of computational Physics*, vol. 293, pp. 4–13, 2015.

[74] K. Diethelm, *The analysis of fractional differential equations: An application-oriented exposition using differential operators of Caputo type*. Springer Science & Business Media, 2010.

[75] H. U. Rehman, M. Darus, y J. Salah, "A note on caputo's derivative operator interpretation in economy," *Journal of Applied Mathematics*, vol. 2018, 2018.

[76] I. Podlubny, "Fractional differential equations, vol. 198 of mathematics in science and engineering," 1999.

[77] M.-S. Abdelouahab y N.-E. Hamri, "The grünwald–letnikov fractional-order derivative with fixed memory length," *Mediterranean Journal of Mathematics*, vol. 13, no. 2, pp. 557–572, 2016.

[78] I. Petrávs, *Fractional-order nonlinear systems: modeling, analysis and simulation*. Springer Science & Business Media, 2011.

[79] R. Garrappa, E. Kaslik, y M. Popolizio, "Evaluation of fractional integrals and derivatives of elementary functions: Overview and tutorial," *Mathematics*, vol. 7, no. 5, p. 407, 2019.

[80] R. Gorenflo y F. Mainardi, "Fractional calculus," en *Fractals and fractional calculus in continuum mechanics*, pp. 223–276, Springer, 1997.

[81] Z. Wang, X. Huang, y G. Shi, "Analysis of nonlinear dynamics and chaos in a fractional order financial system with time delay," *Computers & Mathematics with Applications*, vol. 62, no. 3, pp. 1531–1539, 2011.

[82] K. Diethelm, R. Garrappa, y M. Stynes, "Good (and not so good) practices in computational methods for fractional calculus," *Mathematics*, vol. 8, no. 3, p. 324, 2020.

[83] R. Garrappa, "Numerical solution of fractional differential equations: A survey and a software tutorial," *Mathematics*, vol. 6, no. 2, p. 16, 2018.

[84] D. W. Brzeziński, "Fractional order derivative and integral computation with a small number of discrete input values using grünwald–letnikov formula," *International Journal of Computational Methods*, vol. 17, no. 05, p. 1940006, 2020.

[85] M. S. Tavazoei, M. Haeri, S. Bolouki, y M. Siami, "Stability preservation analysis for frequency-based methods in numerical simulation of fractional order systems," *SIAM Journal on Numerical Analysis*, vol. 47, no. 1, pp. 321–338, 2009.

[86] H. M. Baskonus y H. Bulut, "On the numerical solutions of some fractional ordinary differential equations by fractional adams-bashforth-moulton method," *Open Mathematics*, vol. 1, no. open-issue, 2015.

[87] K. Bingi, R. Ibrahim, M. N. Karsiti, S. M. Hassam, y V. R. Harindran, "Frequency response based curve fitting approximation of fractional–order pid controllers," *International Journal of Applied Mathematics and Computer Science*, vol. 29, no. 2, pp. 311–326, 2019.

[88] A. A. Dastjerdi, B. M. Vinagre, Y. Chen, y S. H. HosseinNia, "Linear fractional order controllers; a survey in the frequency domain," *Annual Reviews in Control*, vol. 47, pp. 51–70, 2019.

[89] Z. Hammouch y T. Mekkaoui, "Circuit design and simulation for the fractional-order chaotic behavior in a new dynamical system," *Complex & Intelligent Systems*, vol. 4, no. 4, pp. 251–260, 2018.

[90] C. Sánchez-López, "An experimental synthesis methodology of fractional-order chaotic attractors," *NONLINEAR DYNAMICS*, 2020.

[91] R. S. Barbosa, J. T. Machado, e I. M. Ferreira, "Pole-zero approximations of digital fractional-order integrators and differentiators using signal modeling techniques," *IFAC Proceedings Volumes*, vol. 38, no. 1, pp. 309–314, 2005.

[92] A. Charef, H. Sun, Y. Tsao, y B. Onaral, "Fractal system as represented by singularity function," *IEEE Transactions on automatic Control*, vol. 37, no. 9, pp. 1465–1470, 1992.

[93] A. Yüce, F. N. Deniz, y N. Tan, "A new integer order approximation table for fractional order derivative operators," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9736–9741, 2017.

[94] N. Shrivastava y P. Varshney, "Rational approximation of fractional order systems using carlson method," en *2015 International Conference on Soft Computing Techniques and Implementations (ICSCTI)*, pp. 76–80, IEEE, 2015.

[95] U. M. Roslan, Z. Salleh, y A. Kılıccman, "Solving zhou chaotic system using fourth-order runge-kutta method," *World Applied Sciences Journal*, vol. 21, no. 6, pp. 939–944, 2013.

[96] Z. Wang, Y. Sun, G. Qi, y B. J. Van Wyk, "The effects of fractional order on a 3-d quadratic autonomous system with four-wing attractor," *Nonlinear Dynamics*, vol. 62, no. 1, pp. 139–150, 2010.

[97] M. Tavazoei y M. Haeri, "Unreliability of frequency-domain approximation in recognising chaos in fractional-order systems," *IET Signal Processing*, vol. 1, no. 4, pp. 171–181, 2007.

[98] S.-K. Lao, L.-M. Tam, H.-K. Chen, y L.-J. Sheu, "Hybrid stability checking method for synchronization of chaotic fractional-order systems," en *Abstract and Applied Analysis*, vol. 2014, Hindawi, 2014.

[99] K. Diethelm, N. J. Ford, y A. D. Freed, "A predictor-corrector approach for the numerical solution of fractional differential equations," *Nonlinear Dynamics*, vol. 29, no. 1, pp. 3–22, 2002.

[100] C. Li y C. Tao, "On the fractional Adams method," *Computers & Mathematics with Applications*, vol. 58, no. 8, pp. 1573–1588, 2009.

[101] S. He, K. Sun, y Y. Peng, "Detecting chaos in fractional-order nonlinear systems using the smaller alignment index," *Physics Letters A*, vol. 383, no. 19, pp. 2267–2271, 2019.

[102] Y. Cui, H. He, G. Sun, y C. Lu, "Analysis and control of fractional order generalized lorenz chaotic system by using finite time synchronization," *Advances in Mathematical Physics*, vol. 2019, 2019.

[103] C. Bonchics, E. Kaslik, y F. Rocsu, "Hpc optimal parallel communication algorithm for the simulation of fractional-order systems," *The Journal of Supercomputing*, vol. 75, no. 3, pp. 1014–1025, 2019.

[104] E. Tlelo-Cuautle, A. D. Pano-Azucena, O. Guillén-Fernández, y A. Silva-Juárez, "Fpga-based implementations of fractional-order chaotic systems," en *Analog/Digital Implementation of Fractional Order Chaotic Circuits and Applications*, pp. 115–173, Springer, 2020.

[105] R. Scherer, S. L. Kalla, Y. Tang, y J. Huang, "The Grünwald–Letnikov method for fractional differential equations," *Computers & Mathematics with Applications*, vol. 62, no. 3, pp. 902–917, 2011.

[106] S. Pooseh, R. Almeida, y D. F. Torres, "Discrete direct methods in the fractional calculus of variations," *Computers & Mathematics with Applications*, vol. 66, no. 5, pp. 668–676, 2013.

[107] M.-F. Danca, W. K. Tang, Q. Wang, y G. Chen, "Suppressing chaos in fractional-order systems by periodic perturbations on system variables," *The European Physical Journal B*, vol. 86, no. 3, p. 79, 2013.

[108] M.-F. Danca, R. Garrappa, W. K. Tang, y G. Chen, "Sustaining stable dynamics of a fractional-order chaotic financial system by parameter switching," *Computers & Mathematics with Applications*, vol. 66, no. 5, pp. 702–716, 2013.

[109] Y. Guo y G. Qi, "Topological horseshoe in a fractional-order qi four-wing chaotic system," *J. Appl. Anal. Comput*, vol. 5, no. 2, pp. 168–176, 2015.

[110] A. H. ElSafty, M. F. Tolba, L. A. Said, A. H. Madian, y A. G. Radwan, "A study of the nonlinear dynamics of human behavior and its digital hardware implementation," *Journal of Advanced Research*, 2020.

[111] Z. U. A. Zafar, N. Ali, G. Zaman, P. Thounthong, y C. Tuncc, "Analysis and numerical simulations of fractional order vallis system," *Alexandria Engineering Journal*, 2020.

[112] R. Ma, J. Han, y X. Yan, "Improved short memory principle method for solving fractional damped vibration equations," *Applied Sciences*, vol. 10, no. 21, p. 7566, 2020.

[113] Z. Liao, "A new definition of fractional derivatives based on truncated left-handed grünwald-letnikov formula with and median correction," en *Abstract and Applied Analysis*, vol. 2014, Hindawi, 2014.

[114] G. Adomian, *Solving Frontier Problems of Physics: The Decomposition Method*, vol. 1. Springer Science & Business Media, 1994.

[115] V. Daftardar-Gejji y H. Jafari, "Adomian decomposition: a tool for solving a system of fractional differential equations," *Journal of Mathematical Analysis and Applications*, vol. 301, no. 2, pp. 508–518, 2005.

[116] D. Cafagna y G. Grassi, "Fractional-order chua's circuit: time-domain analysis, bifurcation, chaotic behavior and test for chaos," *International Journal of Bifurcation and Chaos*, vol. 18, no. 03, pp. 615–639, 2008.

[117] P. Guo, "The adomian decomposition method for a type of fractional differential equations," *Journal of Applied Mathematics and Physics*, vol. 7, no. 10, pp. 2459–2466, 2019.

[118] A.-M. Wazwaz, "A new algorithm for calculating adomian polynomials for nonlinear operators," *Applied Mathematics and computation*, vol. 111, no. 1, pp. 33–51, 2000.

[119] J.-S. Duan, T. Chaolu, R. Rach, y L. Lu, "The adomian decomposition method with convergence acceleration techniques for nonlinear fractional differential equations," *Computers & Mathematics with Applications*, vol. 66, no. 5, pp. 728–736, 2013.

[120] J.-G. Qu, Y.-H. Cui, y G.-C. Zhou, "Research on adomian decomposition method and its application in the fractional order differential equations," *Bio-Technology: An Indian Journal*, vol. 10, no. 7, pp. 2277–2286, 2014. cited By 0.

[121] G. Li, X. Zhang, y H. Yang, "Numerical analysis, circuit simulation, and control synchronization of fractional-order unified chaotic system," *Mathematics*, vol. 7, no. 11, p. 1077, 2019.

[122] N. I. Razali, M. Chowdhury, y W. Asrar, "The multistage adomian decomposition method for solving chaotic lü system," *Middle-east J. Scientific Research*, vol. 13, pp. 43–49, 2013.

[123] F. Yang y P. Li, "Characteristics analysis of the fractional-order chaotic memristive circuit based on chua's circuit," *Mobile Networks and Applications*, pp. 1–9, 2019.

[124] H. Chen, T. Lei, S. Lu, W. Dai, L. Qiu, y L. Zhong, "Dynamics and complexity analysis of fractional-order chaotic systems with line equilibrium based on adomian decomposition," *Complexity*, vol. 2020, 2020.

[125] T. Liu, J. Yu, H. Yan, y J. Mou, "A fractional-order chaotic system with infinite attractor coexistence and its dsp implementation," *IEEE Access*, vol. 8, pp. 199852–199863, 2020.

[126] C. Ma, J. Mou, J. Liu, F. Yang, H. Yan, y X. Zhao, "Coexistence of multiple attractors for an incommensurate fractional-order chaotic system," *The European Physical Journal Plus*, vol. 135, no. 1, p. 95, 2020.

[127] D. Peng, K. Sun, S. He, L. Zhang, y A. O. Alamodi, "Numerical analysis of a simplest fractional-order hyperchaotic system," *Theoretical and Applied Mechanics Letters*, vol. 9, no. 4, pp. 220–228, 2019.

[128] C. Qin, K. Sun, y S. He, "Characteristic analysis of fractional-order memristor-based hypogenetic jerk system and its dsp implementation," *Electronics*, vol. 10, no. 7, p. 841, 2021.

[129] I. Zaouagui y T. Badredine, "New adomian's polynomials formulas for the non-linear and non-autonomous ordinary differential equations," *J Appl Computat Math*, vol. 6, p. 373, 2017.

[130] M. S. Tavazoei y M. Haeri, "A note on the stability of fractional order systems," *Mathematics and Computers in simulation*, vol. 79, no. 5, pp. 1566–1576, 2009.

[131] S. He, K. Sun, X. Mei, B. Yan, y S. Xu, "Numerical analysis of a fractional-order chaotic system based on conformable fractional-order derivative," *The European Physical Journal Plus*, vol. 132, no. 1, pp. 1–11, 2017.

[132] A. A. Abdelhakim y J. A. T. Machado, "A critical analysis of the conformable derivative," *Nonlinear Dynamics*, vol. 95, no. 4, pp. 3063–3073, 2019.

[133] A. J. Abd El-Maksoud, A. A. Abd El-Kader, B. G. Hassan, M. A. Abdelhamed, N. G. Rihan, M. F. Tolba, L. A. Said, A. G. Radwan, y M. F. Abu-Elyazeed, "Fpga implementation of fractional-order chua's chaotic system," en *2018 7th international conference on modern circuits and systems technologies (MOCAST)*, pp. 1–4, IEEE, 2018.

[134] M. Roshdy, M. F. Tolba, L. A. Said, A. H. Madian, y A. G. Radwan, "Generic hardware of fractional order multi-scrolls chaotic generator based on fpga," en *2019 17th IEEE International New Circuits and Systems Conference (NEWCAS)*, pp. 1–4, IEEE, 2019.

[135] K. Rajagopal, F. Nazarimehr, A. Karthikeyan, A. Srinivasan, y S. Jafari, "Camo: Self-excited and hidden chaotic flows," *International Journal of Bifurcation and Chaos*, vol. 29, no. 11, p. 1950143, 2019.

[136] K. Rajagopal, S. T. Kingni, A. J. M. Khalaf, Y. Shekofteh, y F. Nazarimehr, "Coexistence of attractors in a simple chaotic oscillator with fractional-order-memristor component: analysis, fpga implementation, chaos control and synchronization," *The European Physical Journal Special Topics*, vol. 228, no. 10, pp. 2035–2051, 2019.

[137] A. H. Elsafty, M. F. Tolba, L. A. Said, A. H. Madian, y A. G. Radwan, "Enhanced hardware implementation of a mixed-order nonlinear chaotic system and speech encryption application," *AEU-International Journal of Electronics and Communications*, vol. 125, p. 153347, 2020.

[138] K. Rajagopal, F. Nazarimehr, S. Jafari, y A. Karthikeyan, "Fractional and non-fractional chaotic amphibian attractors with self-excited and hidden properties: numerical dynamics, circuit realization and fpga-based application," *The European Physical Journal Special Topics*, vol. 226, no. 16-18, pp. 3827–3850, 2017.

[139] K. Rajagopal, A. Karthikeyan, y P. Duraisamy, "Hyperchaotic chameleon: Fractional order fpga implementation," *Complexity*, vol. 2017, 2017.

[140] A. Bayani, M. A. Jafari, K. Rajagopal, H. Jiang, y S. Jafari, "A novel fractional-order chaotic system with specific topology: from proposing to fpga implementation," *The European Physical Journal Special Topics*, vol. 226, no. 16, pp. 3729–3745, 2017.

[141] Z. Wei, K. Rajagopal, W. Zhang, S. T. Kingni, y A. Akgül, "Synchronisation, electronic circuit implementation, and fractional-order analysis of 5d ordinary differential equations with hidden hyperchaotic attractors," *Pramana*, vol. 90, no. 4, pp. 1–13, 2018.

[142] A. Karthikeyan y K. Rajagopal, "Fpga implementation of fractional-order discrete memristor chaotic system and its commensurate and incommensurate synchronisations," *Pramana*, vol. 90, no. 1, p. 14, 2018.

[143] K. Rajagopal, S. Jafari, A. Karthikeyan, A. Srinivasan, y B. Ayele, "Hyperchaotic memcapacitor oscillator with infinite equilibria and coexisting attractors," *Circuits, Systems, and Signal Processing*, vol. 37, no. 9, pp. 3702–3724, 2018.

[144] N. S. Soliman, M. F. Tolba, L. A. Said, A. H. Madian, y A. G. Radwan, "Fractional x-shape controllable multi-scroll attractor with parameter effect and fpga automatic design tool software," *Chaos, Solitons & Fractals*, vol. 126, pp. 292–307, 2019.

[145] G. Gugapriya, P. Duraisamy, A. Karthikeyan, y B. Lakshmi, "Fractional-order chaotic system with hyperbolic function," *Advances in Mechanical Engineering*, vol. 11, no. 8, p. 1687814019872581, 2019.

[146] M. F. Tolba, H. Saleh, B. Mohammad, M. Al-Qutayri, A. S. Elwakil, y A. G. Radwan, "Enhanced fpga realization of the fractional-order derivative and application to a variable-order chaotic system," *Nonlinear Dynamics*, pp. 1–12, 2020.

[147] M. Roshdy, W. S. Sayed, L. A. Said, A. H. Madian, A. G. Radwan, y M. Dessouky, "Fpga implementation of delayed fractional-order financial chaotic system," en *2020 16th International Computer Engineering Conference (ICENCO)*, pp. 51–54, IEEE, 2020.

[148] S. He, K. Sun, y H. Wang, "Complexity analysis and dsp implementation of the fractional-order lorenz hyperchaotic system," *Entropy*, vol. 17, no. 12, pp. 8299–8311, 2015.

[149] H. Wang, K. Sun, y S. He, "Characteristic analysis and dsp realization of fractional-order simplified lorenz system based on adomian decomposition method," *International Journal of Bifurcation and Chaos*, vol. 25, no. 06, p. 1550085, 2015.

[150] S. He, K. Sun, y H. Wang, "Dynamics of the fractional-order lorenz system based on adomian decomposition method and its dsp implementation," *IEEE/CAA Journal of Automatica Sinica*, 2016.

[151] L. Zhang, K. Sun, S. He, H. Wang, y Y. Xu, "Solution and dynamics of a fractional-order 5-d hyperchaotic system with four wings," *The European Physical Journal Plus*, vol. 132, no. 1, pp. 1–16, 2017.

[152] H. Wang, S. He, y K. Sun, "Complex dynamics of the fractional-order rössler system and its tracking synchronization control," *Complexity*, vol. 2018, 2018.

[153] J. Ruan, K. Sun, J. Mou, S. He, y L. Zhang, "Fractional-order simplest memristor-based chaotic circuit with new derivative," *The European Physical Journal Plus*, vol. 133, no. 1, pp. 1–12, 2018.

[154] S. He, K. Sun, H. Wang, X. Mei, y Y. Sun, "Generalized synchronization of fractional-order hyperchaotic systems and its dsp implementation," *Nonlinear Dynamics*, vol. 92, no. 1, pp. 85–96, 2018.

[155] J. Wu, G. Wang, H. H.-C. Iu, Y. Shen, y W. Zhou, "A nonvolatile fractional order memristor model and its complex dynamics," *Entropy*, vol. 21, no. 10, p. 955, 2019.

[156] D. Peng, K. H. Sun, y A. O. Alamodi, "Dynamics analysis of fractional-order permanent magnet synchronous motor and its dsp implementation," *International Journal of Modern Physics B*, vol. 33, no. 06, p. 1950031, 2019.

[157] F. Yang, J. Mou, J. Liu, C. Ma, y H. Yan, "Characteristic analysis of the fractional-order hyperchaotic complex system and its image encryption application," *Signal Processing*, vol. 169, p. 107373, 2020.

[158] C. Ma, J. Mou, F. Yang, y H. Yan, "A fractional-order hopfield neural network chaotic system and its circuit realization," *The European Physical Journal Plus*, vol. 135, no. 1, p. 100, 2020.

[159] S. Gu, S. He, H. Wang, y B. Du, "Analysis of three types of initial offset-boosting behavior for a new fractional-order dynamical system," *Chaos, Solitons & Fractals*, vol. 143, p. 110613, 2021.

[160] F. Yang y X. Wang, "Dynamic characteristic of a new fractional-order chaotic system based on the hopfield neural network and its digital circuit implementation," *Physica Scripta*, vol. 96, no. 3, p. 035218, 2021.

[161] T. Liu, H. Yan, S. Banerjee, y J. Mou, "A fractional-order chaotic system with hidden attractor and self-excited attractor and its dsp implementation," *Chaos, Solitons & Fractals*, vol. 145, p. 110791, 2021.

[162] J. M. Munoz-Pacheco, C. Posadas-Castillo, y E. Zambrano-Serrano, "The effect of a non-local fractional operator in an asymmetrical glucose-insulin regulatory system: Analysis, synchronization and electronic implementation," *Symmetry*, vol. 12, no. 9, p. 1395, 2020.

[163] D. Cafagna y G. Grassi, "Adomian descomposition method as a tool for numerical studying multi-scroll hyperchaotic attractors," *International Symposium on*

Nonlinear Theory and its Applications (NOLTA2005) Bruges, Belgium, October 18-21, 2005, p. , 2005.

[164] D. Cafagna y G. Grassi, "Fractional-order chua's circuit: Time-domain analysis, bifurcation, chaotic behavior and test for chaos," *International Journal of Bifurcation and Chaos*, vol. 18, no. 3, pp. 615–639, 2006.

[165] J. Lü y G. Chen, "Generating multiscroll chaotic attractors: theories, methods and applications," *International Journal of Bifurcation and chaos*, vol. 16, no. 04, pp. 775–858, 2006.

[166] W. Deng y J. Lü, "Design of multidirectional multiscroll chaotic attractors based on fractional differential systems via switching control," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 16, no. 4, p. 043120, 2006.

[167] M. S. Tavazoei y M. Haeri, "A necessary condition for double scroll attractor existence in fractional-order systems," *Physics Letters A*, vol. 367, no. 1-2, pp. 102–113, 2007.

[168] E. Tlelo-Cuautle, L. de la Fraga, y J. Rangel-Magdaleno, *Engineering applications of FPGAs*. Springer, 2016.

[169] J. Cureño-Osornio, I. Zamudio-Ramirez, L. Morales-Velazquez, A. Y. Jaen-Cuellar, R. A. Osornio-Rios, y J. A. Antonino-Daviu, "Fpga-flux proprietary system for online detection of outer race faults in bearings," *Electronics*, vol. 12, no. 8, p. 1924, 2023.

[170] E. Galan-Uribe, L. Morales-Velazquez, y R. A. Osornio-Rios, "Fpga-based methodology for detecting positional accuracy degradation in industrial robots," *Applied Sciences*, vol. 13, no. 14, p. 8493, 2023.

[171] K.-M. Ma, D.-H. Le, C.-K. Pham, y T.-T. Hoang, "Design of an soc based on 32-bit risc-v processor with low-latency lightweight cryptographic cores in fpga," *Future Internet*, vol. 15, no. 5, p. 186, 2023.