Emulación en FPGA de un Circuito Memristor Hipercaótico

Diego Armando-Ríos Mejía[a], Opeyemi Micheal-Afolabi[b], Luis Jothan Gamez Palacios[a], Dra. Yuma-Sandoval Ibarra[a], Dr. José Cruz-Núñez Pérez[b].

[a] Tecnológico Nacional de México, Instituto Tecnológico de Tijuana, {diego.rios18,luis.gamez18,yuma.sandoval}@tectijuana.edu.mx, Tijuana, Baja California, México.
[b] Instituto Politécnico Nacional, IPN-CITEDI, nunez@citedi.mx, Tijuana, Baja California, México.

Resumen

En este artículo se presenta la metodología de diseño de un oscilador caótico basado en un memristor y emulado en una tarjeta de FPGA. Además, se detallan los métodos numéricos de Runge-Kutta de cuarto orden (RK4) y Runge-Kutta de orden fraccional explicito (EFORK) para darle solución a estos sistemas de ecuaciones dinámicos. Finalmente, se simuló un oscilador caótico de orden entero y fraccional utilizando IOMR y FOMR en Simulink, con el blockset de Vitis Model Composer, para emular su funcionamiento en una tarjeta de desarrollo Artix-7 AC701 de Xilinx. Posteriormente, se genera el código VHDL del sistema diseñado y se obtiene el reporte de los recursos empleados por el oscilador caótico usando el programa Vivado.

Palabras clave— Caos, FPGA, Memristor, Matlab, Multisim, Orden fraccional.

Abstract

This article presents the design methodology of a chaotic oscillator based on a memristor and implemented on an FPGA board. Therefore, first of all it is necessary to address the topics of fractional calculus and the systems of fractional differential equations, the memristor and the chaotic oscillator circuit mathematical models. In addition, the numerical methods of fourth-order Runge-Kutta (RK4) and explicit fractional order Runge-Kutta (EFORK) are detailed to provide a solution to these systems of dynamical equations. Finally, a chaotic integer and fractional order oscillator were simulated using IOMR and FOMR in Simulink using the Vitis Model Composer blockset for emulating its operation on a Xilinx Artix-7 AC701 development board. Subsequently, the VHDL code of the designed system is generated and the report of the resources used by the chaotic oscillator are obtained using the Vivado program.

Keywords— *Chaos, FPGA, Fractional order, Memristor, Matlab, Multisim.*

1. INTRODUCCIÓN

En 1971, Leon O. Chua propuso la existencia de un cuarto elemento básico en circuitos, y fue llamado memristor, una combinación de *memoria* y *resistor*, que era tan básico como los otros tres elementos, resistor, inductor y capacitor. Resalta que su propiedad más importante es su pasividad que brinda las bases para su realización física, también muestra cómo se realizó el memristor con circuitos activos y presentó una interpretación del campo electromagnético que caracteriza el memristor. Además menciona aplicaciones del memristor como modelado de dispositivos y procesamiento de señales [1]. Fue hasta 2008 cuando se mostró, usando un ejemplo analítico, que la memresistencia surge naturalmente en sistemas a nanoescala en los cuales el transporte iónico y de electrones en estado sólido están acoplados bajo una fuente de voltaje externa. Estos resultados sirven como los fundamentos para entender el amplio rango del comportamiento histerético entre corriente-voltaje observado en varios dispositivos electrónicos a nanoescala [2]. Los sistemas caóticos son osciladores aperiódicos no lineales que presentan sensibilidad a las condiciones iniciales. En 1963, Edward N. Lorenz desarrolló el primer sistema de ecuaciones diferenciales de tercer orden que presentaba una dinámica caótica. Él se dio cuenta que cualquier variación en las condiciones iniciales afectaba las condiciones finales. Posteriormente, en 1979, Otto Rössler propone el primer sistema hipercaótico. Este sistema consiste en una ecuación diferencial de cuarto orden. A diferencia de los sistemas caóticos, este tipo de sistemas presentan un comportamiento más complejo, es decir, tiene un atractor más complejo [3].

La elaboración de este trabajo de investigación facilitará el análisis del memristor en sistemas caóticos dentro de los centros de investigación, donde el software Matlab es de los más usados. Por lo tanto, esto llevará a la aplicación del memristor en conjunto con la teoría del caos a distintas ramas de la ingeniería con mayor facilidad. Por ejemplo, en el procesamiento de señales ya sea como transmisión de datos o encriptación de estas mismas.

Este artículo está organizado de la siguiente manera, la sección 2 contiene la teoría relevante, como métodos numéricos, cálculo fraccional, memristores y osciladores caóticos. En la sección 3 se muestran los resultados obtenidos con los osciladores caóticos IOMR y FOMR y el consumo de recursos del FPGA Artix-7 AC701 de Xilinx. Por último, en la sección 4 se presentan las conclusiones.

2. MARCO TEÓRICO

Esta sección explica las bases teóricas de cálculo fraccional, los métodos numéricos de Runge-Kutta de cuarto orden y orden fraccional explícito, memristor y los sistemas caóticos.

2.1. Cálculo fraccional

Es una rama del análisis matemático que estudia la posibilidad de tomar potencias reales del operador integral y derivativo, dejando de limitar dichos operadores a un orden de tipo entero. El primer paso es obtener una fórmula general para resolver una integral de n grados, donde n es un número entero positivo, como lo muestra la ecuación (1),

$$\mathcal{D}^{-n}f(t) = \int_0^t \frac{f(y)(t-y)^{n-1}}{(n-1)!}dy \quad (1)$$

Como se observa en la ecuación (1), se requiere realizar una operación factorial, por eso Cauchy decidió sustituir dicha función factorial por la función Gamma La función Gamma (Γ) permite utilizar la notación factorial con números reales y complejos, siempre y cuando la parte real del número sea positiva, como se observa en la ecuación (2),

$$\Gamma(z) = \lim_{n\to\infty}\left(\frac{n!\,(z-1)!}{(z+n)!}\right)n^2 \quad (2)$$

De esta forma Cauchy logró expresar de manera general una integral de orden real o de orden fraccional (FO). En la ecuación (3) se observa la fórmula de Cauchy donde *n* fue sustituida por *α*, representando así un número real entero [4],

$$\mathcal{D}^{-\alpha}f(t) = \frac{1}{\Gamma(n)}\int_0^t f(y)(t-y)^{\alpha-1}dy \quad (3)$$

Con los aportes de Cauchy fue posible calcular una integral de OF, ahora el problema era obtener una derivada de OF. Como se sabe, la operación opuesta de una integral es una derivada, en base a esto se desarrolló el método de derivación fraccional de Riemann-Liouville, el cual consiste en obtener la derivada fraccional de cualquier función, donde primero se debe integrar fraccionariamente y después derivar enteramente. Por ejemplo, si se desea obtener la derivada de grado ½, primero se debe obtener la integración fraccional cuando α = ½ y posteriormente, a dicho resultado se le aplica la operación de derivación de grado 1. La derivación fraccional de Riemann-Liouville se expresa en la ecuación (4),

$$\mathcal{D}^{-\alpha}f(t) = \frac{1}{\Gamma(n-\alpha)}\frac{d^n}{dx^n}\int_{-\infty}^{x}\frac{f(t)}{(x-t)^{\alpha-n+1}}dt, \quad -\infty < x < \infty \quad (4)$$

El problema con la derivada de Riemann-Liouville es que, al realizar modelizaciones matemáticas de fenómenos físicos reales por medio de ecuaciones diferenciales de FO, es necesario trabajar con condiciones iniciales de FO, las cuales no tienen una interpretación física clara. Por ello el operador diferencial de Caputo invierte el orden de la derivación en la definición de Riemann-Liouville, es decir, primero se debe realizar la derivada de grado entero y posteriormente se realiza la integración fraccional. De esta manera se utilizan las condiciones iniciales derivadas de orden entero. En la ecuación (5) se observa el operador diferencial de Caputo,

$$\mathcal{D}^{-\alpha}f(t) = \frac{1}{\Gamma(n-\alpha)}\int_\alpha^t (t-s)^{n-\alpha-1}f^{(n)}(s)ds, \;\; t > \alpha \quad (5)$$

Ecuaciones diferenciales fraccionales (EDFs)

Las ecuaciones fraccionales diferenciales son modelos matemáticos que se representan en ecuaciones conteniendo derivadas de orden arbitrario. Estas ecuaciones diferenciales serían lineales o no lineales dependiendo de los términos contenidos en la ecuación y son las extensiones de ecuaciones diferenciales ordinarias al dominio de ordenes reales [5]. Las ecuaciones en la forma (6) con representación de condición inicial en (7) son EDFs en el sentido Caputo.

$$\mathcal{D}^{\alpha}y(t) = f\big(t, y(t)\big) \quad (6)$$
$$\mathcal{D}^{k}y(t) = b_k, k = 1,2, \dots n-1 \quad (7)$$

La solución general del problema con valor inicial es dada por la ecuación (8),

$$y(t) = \sum_{i=1}^{n}\frac{b_i}{\Gamma(a_i)}t^{a_i-1} + \frac{1}{\Gamma(a_n)}\int_0^t (t-\tau)^{a_n-1}f\big(\tau, y(\tau)\big)\,d\tau \quad (8)$$

2.2. Método Runge-Kutta de cuarto orden

El método numérico Runge-Kutta fue desarrollado por los matemáticos alemanes Carl Runge y Martin Kutta en el siglo XX. La idea comenzó con Carl en 1895 cuando en su artículo científico desarrollaron un algoritmo con menos errores basado en el método de Euler para analizar ecuaciones diferenciales ordinarias numéricamente [6]. Desde entonces este método ha ganado popularidad especialmente el método de cuarto orden debido a su alta tasa de convergencia para resolver problemas con valores iniciales con la forma de la ecuación (9) con condiciones iniciales como (10) definido en el dominio uniformemente distante [x_0, T] con tamaño de paso $h = x_n - x_{n-1}$, donde $n = 1, 2, \dots N$.

$$\frac{dy}{dx} = f(x,y) \quad (9)$$
$$y(x_0) = y_0 \quad (10)$$

La forma generalizada de *s*-etapas del método Runge-Kutta es definida en [7] por las ecuaciones (11) y (12),

$$\begin{aligned} k_1 &= f(x_0, y_0) \\ k_2 &= f\big(x_0 + c_2h, y_0 + h(a_{21}k_1)\big) \\ k_3 &= f\big(x_0 + c_3h, y_0 + h(a_{31}k_1 + a_{32}k_2)\big) \quad (11) \\ &\dots \\ k_s &= f\left(x_0 + c_sh, y_0 + h\big(a_{s1}k_1 + \cdots + a_{s,s-1}k_{s-1}\big)\right) \\ y_1 &= y_0 + h(b_1k_1 + \cdots b_sk_s) \end{aligned}$$

donde *s* es un entero, a_{21}, a_{31}, a_{32}, …, a_s, a_{s-1}, b_1, b_s, c_1, …, c_s son coeficientes reales y la ecuación (12) se satisface,

$$c_i = \sum_{j=0}^{i-1} a_{ij} \tag{12}$$

El método Runge-Kutta de cuarto orden está dado por el algoritmo de la ecuación (13),

$$\begin{aligned} K_1 &= hf(x_n, y_n) \\ K_2 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{K_1}{2}\right) \\ K_3 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{K_2}{2}\right) \\ K_4 &= hf(x_n + h, y_n + K_3) \\ y_{n+1} &= y_n + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) \end{aligned} \tag{13}$$

Cabe notar que el algoritmo (13) se aplica para sistemas dinámicos lineales y no lineales usando los mismos pasos.

2.3. Método Runge-Kutta de orden fraccional explícito

Considerando el problema de valor inicial definido por la derivada fraccional de Caputo en la ecuación (14),

$$\begin{cases} \mathcal{D}^{\alpha} y(t) = f\big(t, y(t)\big), & y \in [t_0, T] \\ y(t_0) = y_0 \end{cases} \tag{14}$$

con $0 < \alpha \leq 1$, y donde $y(t) \in [t_0, T]$, $f(t,y(t)) \in [t_0, T] \times \mathbb{R}$ y t_0 es el punto base de la derivada fraccional. Sea $t_n = t_0 + nh$ para $n = 0, 1, \ldots, N^m$, donde $h = (T - t_0)/N^m$ es el tamaño de paso y N es un entero positivo. Un EFORK de s-etapas es definido por las ecuaciones (15) y (16),

$$\begin{aligned} K_1 &= h^{\alpha} f(t, y) \\ K_2 &= h^{\alpha} f(t + c_2 h, y + a_{21} K_1) \\ K_3 &= h^{\alpha} f(t + c_3 h, y + a_{31} K_1 + a_{32} K_2) \\ &\ldots \\ K_s &= h^{\alpha} f\big(t + c_s h, a_{s1} K_1 + a_{s2} K_2 + \cdots + a_{s,s-1} K_{s-1}\big) \end{aligned} \tag{15}$$

con

$$y_{n+1} = y_n \sum_{t=1}^{s} w_t K_t \tag{16}$$

donde los coeficientes desconocidos a_{ij} para $i = 2, \ldots, s$, $j = 1, \ldots, i-1$ y los pesos desconocidos c_i para $i = 2, \ldots, s$ y w_i para $i = 1, \ldots, s$ deben determinarse [8]. En este artículo, el EFORK de orden 2α de la ecuación (17) con los coeficientes y pesos óptimos de la ecuación (18) es usado como aproximación numérica de sistemas dinámicos de orden fraccional,

$$\begin{aligned} K_1 &= h^{\alpha} f(t, y) \\ K_2 &= h^{\alpha} f(t + c_2 h, y + a_{21} K_1) \\ y_{n+1} &= y_n + w_1 K_1 + w_2 K_2 \end{aligned} \tag{17}$$

con

$$c_2 = \left(\frac{4\Gamma(\alpha+1)}{\Gamma(3\alpha+1)}\right)^{\frac{1}{\alpha}}, a_{21} = \frac{4}{\Gamma(3\alpha+1)} \tag{18}$$

$$w_1 = \frac{4}{\Gamma(\alpha+1)} - \frac{4\Gamma(3\alpha+1)}{4\Gamma(2\alpha+1)}, w_2 = \frac{4\Gamma(3\alpha+1)}{4\Gamma(2\alpha+1)}$$

donde el método EFORK es apto para sistemas dinámicos de orden fraccional lineales y no lineales de orden arbitrario α.

2.4. Memristor

La memristencia se refiere al comportamiento similar a resistencia que cambia en respuesta al historial de corriente o voltaje. En el caso de un sistema memristivo controlado por carga, la relación ideal que gobierna la operación del dispositivo cuando un voltaje es aplicado a través de sus terminales está dada por la ecuación (19),

$$\begin{aligned} v_m(t) &= M(x, q, t) i(t) \\ \dot{x} &= g(x, q, t) \end{aligned} \tag{19}$$

donde M es la memristencia que depende de la cantidad total de carga eléctrica y la derivada del flujo φ con respecto a la carga acumulada en el memristor, en la ecuación (20),

$$M(q) = \frac{d\varphi(q)}{dq} \tag{20}$$

El modelo matemático de un sistema memristivo se generaliza a órdenes de valores arbitrarios usando calculo fraccional [9]. Así, el sistema memristivo controlado por carga definido en (19) se reescribe como la ecuación (21),

$$\begin{aligned} \mathcal{D}^{\alpha} v_m(t) &= M(x, q, t) i(t) \\ \dot{x} &= g(x, q, t) \end{aligned} \tag{21}$$

Para un sistema memristivo controlado por voltaje, la relación matemática del sistema es la ecuación (22),

$$\begin{aligned} i_m(t) &= G(x, v, t) v(t) \\ \dot{x} &= g(x, v, t) \end{aligned} \tag{22}$$

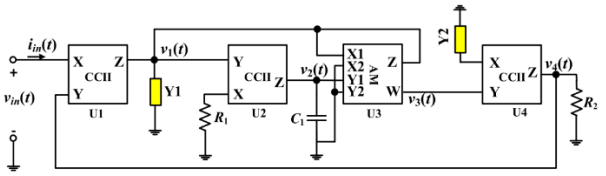
donde G es la memductancia del sistema memristivo en el tiempo t. El sistema memristivo controlado por voltaje, de la misma forma que el controlado por carga, se generaliza a un orden arbitrario de valores como describe la ecuación (23),

$$\begin{aligned} \mathcal{D}^{\alpha} i_m(t) &= G(x, v, t) v(t) \\ \dot{x} &= g(x, v, t) \end{aligned} \tag{23}$$

2.4. Modelos universales de memelementos

Un modelo universal de memelementos fue propuesto en [10] como muestra la Fig. 1, y se puede realizar tanto un memristor, un memcapacitor y un meminductor. Este emulador ofrece las ventajas de baja complejidad del circuito y uso mínimo de componentes pasivos. Utiliza tres transportadores de corriente de segunda generación de tipo positivo (CCII+), un multiplicador análogo (AM) y cinco componentes pasivos para conexión a tierra.

Fig. 1. Modelo universal de memelementos



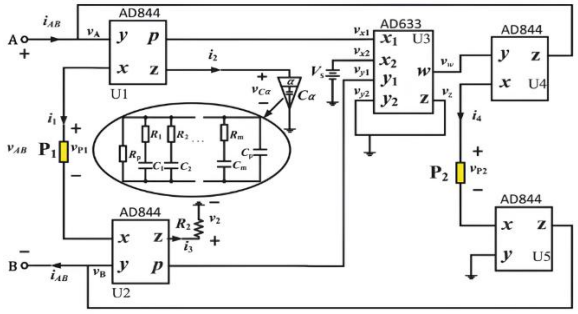
Fuente: Q. Zhao, C. Wang, y X. Zhang [10].

Los amplificadores operacionales AD844 y AD633 se usaron para implementar las funciones de CCII+ y el multiplicador análogo, respectivamente. Para lograr un memristor con el emulador universal de la Fig. 1, los componentes Y_1 e Y_2 requieren ser resistores R_0 y R_3, respectivamente. La relación *v-i* que describe el memristor está dada por la ecuación (24),

$$v_{in}(t) = -\frac{R_0 R_2}{R_3} i_{in}(t) + \frac{{R_0}^2 R_2}{10 R_1 R_3 C_1} q(t) i_{in}(t) \qquad (24)$$

En [11] se propone un modelo de tipo flotante para memelementos de orden fraccional (FOME) como muestra la Fig. 2. En este modelo se utilizó un capacitor de orden fraccional (FOC) para proveer la operación de orden integral en el emulador. El modelo consta de cuatro amplificadores operacionales AD844, los cuales funcionan como transportadores de corriente y seguidores de voltaje, un multiplicador analógico AD633, y tres componentes pasivos adicionales, los cuales incluyen varias combinaciones de capacitores, resistores, e inductores dependiendo del memelemento específico bajo consideración.

Fig. 2. Interfaz universal para memelementos de orden fraccional



Fuente: Y. Li, et al. [11].

Para realizar un memristor de orden fraccional (FOMR) del FOME en la Fig. 2 el elemento electrónico P_1 y P_2 requieren sustituirse por resistores R_1 y R_3, respectivamente. La relación *v-i* que describe el FOMR está dado por la ecuación (25),

$$G_m(t) = \frac{i_{AB}(t)}{v_{AB}(t)} = -\frac{R_2}{10 C_\alpha R_3 {R_1}^2} J^\alpha v_{AB}(t) - \frac{R_2 V_s}{10 R_1 R_3} \qquad (25)$$

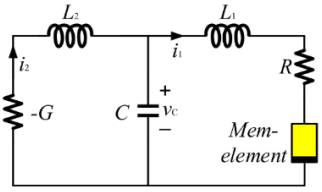
donde $G_m(t)$ es la memconductancia del FOMR, $i_{AB}(t)$, $v_{AB}(t)$ representan la corriente de entrada y voltaje de entrada, respectivamente, J^α es el operador de integral fraccional descrito en la ecuación (26), C_α es el valor del FOC utilizado en el diseño, V_s es el puerto de voltaje de la terminal x_2 del multiplicador AD633 utilizado, R_1, R_2, R_3 son resistores y α es el valor de orden fraccional.

$$J^\alpha f(t) = \frac{1}{\Gamma(\alpha)} \int_0^t (t-\tau)^{\alpha-1} f(\tau)\, d\tau \qquad (26)$$

2.5. Oscilador caótico universal de memelementos

El circuito oscilador mostrado en la Fig. 3 es un circuito oscilador dual de Chua modificado.

Fig. 3. Diagrama de circuito caótico universal de memelementos



Fuente: Q. Zhao, C. Wang, y X. Zhang [10].

El circuito es de una conductancia negativa *-G*, dos inductores L_1, y L_2, un resistor *R*, un capacitor *C* y un modelo de memelemento caracterizado por el voltaje en las terminales $v_m(q)$. Al aplicar la ley de Kirchoff, el sistema dinámico del circuito oscilador se deriva como la ecuación (27), donde i_1, i_2, q y v_c son variables de estado.

$$\begin{cases} L_1 \dfrac{di_1}{dt} = v_C - R i_1 - v_m(q) \\ C \dfrac{dv_C}{dt} = i_2 - i_1 \\ L_2 \dfrac{di_2}{dt} = \dfrac{i_2}{G} - v_C \\ \dfrac{dq}{dt} = i_1 \end{cases} \qquad (27)$$

Cuando el memelemento del circuito de la Fig. 3 es un memristor y el voltaje en las terminales $v_m(q)$ es sustituido por el modelo descrito en la ecuación (24) el sistema dinámico en la ecuación (27) se convierte en la ecuación (28),

$$\begin{cases} L_1 \dfrac{di_1}{dt} = v_C - R i_1 - \left(-\dfrac{R_0 R_2}{R_3} + \dfrac{{R_0}^2 R_2}{10 R_1 R_3 C_1} q \right) \\ C \dfrac{dv_C}{dt} = i_2 - i_1 \\ L_2 \dfrac{di_2}{dt} = \dfrac{i_2}{G} - v_C \\ \dfrac{dq}{dt} = i_1 \end{cases} \qquad (28)$$

Al hacer una escala del dominio del tiempo en el sistema (28), la representación matemática se simplifica al sistema de la ecuación (29) usando $t = L_2 d\tau$, $x = i_1$, $y = v_c$, $z = i_2$, $w = q/L_2$, y ajustando los parámetros del sistema para $a = L_2/L_1$, $b = L_2/C$, $c = R$, $k = 1/G$, $m = -R_0 R_2/R_3$, y $n = R_0\, 2R_2/R_1 R_3 C_1$,

$$\begin{cases} \dot{x} = a(y - cx - (m + nw)x) \\ \dot{y} = b(z - x) \\ \dot{z} = kz - y \\ \dot{w} = x \end{cases} \quad (29)$$

donde *x, y, z,* y *w* son los estados variables y *a, b, c, k, m* y *n* son valores constantes reales.

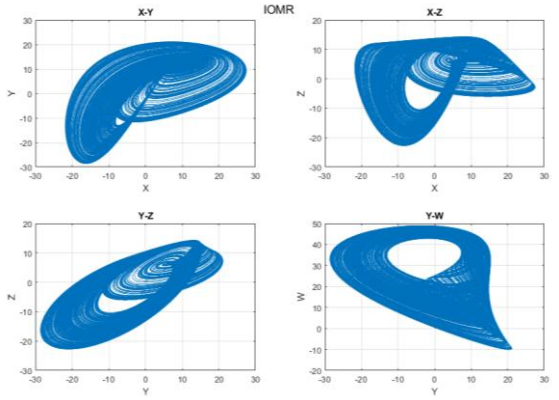
3. RESULTADOS

Una vez desarrollado el código de Matlab para los métodos RK4 y EFORK, se diseñan los osciladores caóticos para IOMR y FOMR en Simulink usando bloques de la biblioteca *AMD Tools* para una tarjeta Artix-7 AC701 de Xilinx y se genera su código VHDL.

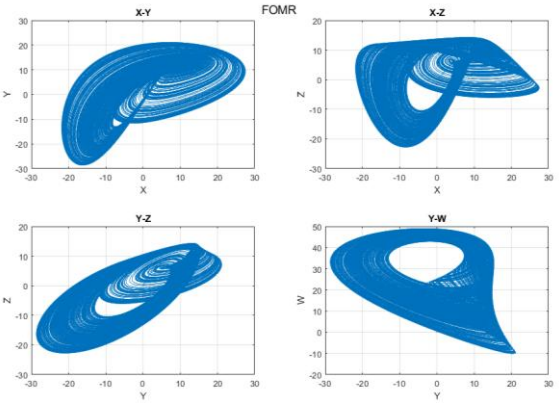
3.1. Oscilador caótico en Matlab

Se usó el modelo matemático de la ecuación (29) con los parámetros *a=2, b=1, c=0.2, k=0.92, m=-0.002, n=0.04*, con los valores iniciales $x_0=y_0=z_0=w_0=0.01$ y un paso de muestreo *h=0.01*. Resolviendo con RK4 de la ecuación (13) se obtienen las oscilaciones de la Fig. 4a para los planos *x-y, x-z, y-z, y-w*.

Fig. 4. Oscilaciones caóticas en un circuito a) IOMR, b) FOMR.



a)



b)

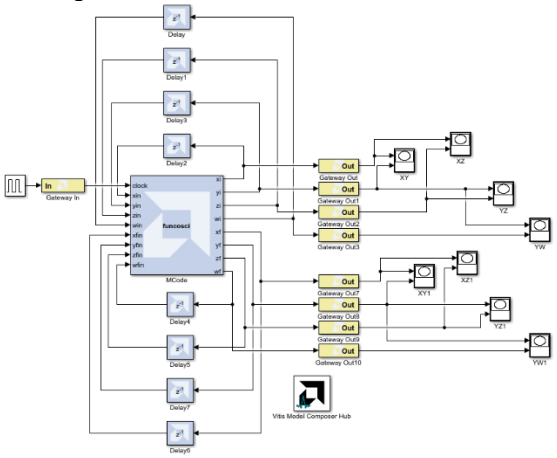
Fuente: elaboración propia a partir del sistema (29)

En el modelo FOMR se utilizó el mismo sistema de ecuaciones, parámetros y valores iniciales del IOMR, aplicando el método EFORK de la ecuación (17) con $\alpha = 0.999999$ resultando en las oscilaciones de la Fig. 4b.

3.2. Oscilador caótico de memristor en Simulink

Se diseñó el circuito de la Fig. 5 dentro de Simulink, haciendo uso de un bloque *MCode* para calcular valores del oscilador entero con el método RK4 y para el oscilador fraccionario utiliza el método EFORK. El sistema es retroalimentado y se visualizan las gráficas *x-y, x-z, y-z, y-w* en las salidas.

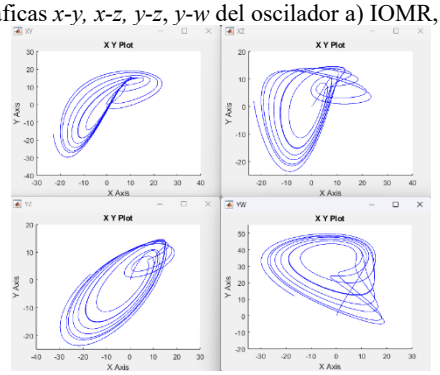
Fig. 5. Circuito caótico usando AMD Toolbox.



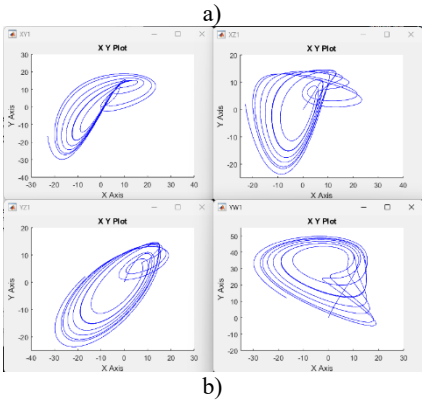
Fuente: elaboración propia a partir del sistema (29)

Después de simular el sistema se obtienen las gráficas de las Fig. 6a y Fig. 6b para los osciladores caótico de orden entero (IOMR) y de orden fraccional (FOMR), respectivamente.

Fig. 6. Gráficas *x-y, x-z, y-z, y-w* del oscilador a) IOMR, b) FOMR.



a)



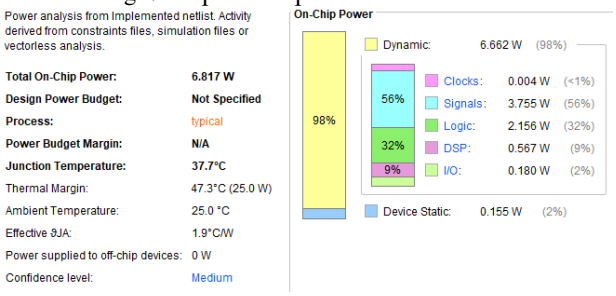
b)

Fuente: elaboración propia a partir del sistema (29)

3.3. Recursos en Artix-7

Para poder generar el código en VHDL, se deben poner los bloques de *Model Composer* en un subsistema, con todo dentro de un subsistema se genera el código de VHDL para usarlo en Vivado. Al finalizar la implementación del proyecto dentro de Vivado se procede a abrir el reporte de potencia y se obtiene la Fig. 7 donde se observa el consumo de potencia por sus diferentes partes con un total de 6.817 W.

Fig. 7. Reporte de potencia en Vivado.



Fuente: elaboración propia.

También se abre el reporte de utilización que muestra la cantidad y especificaciones de los recursos utilizados, esto incluye celdas lógicas, distribución de celdas lógicas, memorias, DSP, especificaciones de I/O y GT, y el reloj. El uso de los recursos más relevantes se muestra en la Tabla 1.

Tabla 1. Utilización de recursos en la tarjeta Artix-7 AC701

Recursos en Artix-7 AC701	Usados	Disponibles	% Utilizado
Slice LUT	114703	133800	85.73
Slice Flip Flop	168	269200	0.06
Slices utilizados	33420	33450	99.91
Block RAM	0	365	0
BUFG	1	32	3.13
IO	338	400	84.50
DSP	718	740	97.03

Fuente: elaboración propia.

4. CONCLUSIONES

En este artículo se realizó la emulación de un oscilador caótico de orden entero y de orden fraccional, diseñado en Simulink utilizando el modelo matemático de un circuito oscilador caótico universal de memelementos aplicado a un memristor de orden entero (IOMR) y a un memristor de orden fraccional (FOMR). Como se esperaba, las gráficas resultantes para los planos *X-Y, X-Z, Y-Z,* y *Y-W* en Simulink tienen el mismo comportamiento visto previamente en las simulaciones realizadas en Matlab. Mientras que se tiene un alto consumo de energía al estar operando con una potencia total de 6.817 W a una temperatura de 37.7°C en el chip con 25°C de temperatura ambiente, de esta potencia 6.662 W (98%) está dedicada a lo dinámico y, dentro de lo dinámico, se emplean 3.755 W para las señales. En cuanto a los recursos lógicos utilizados, el sistema completo emulado utiliza gran parte de la memoria de la Artix-7, para el funcionamiento de ambos osciladores. Los recursos más utilizados son 114703 LUTs de un total de 133800 abarcando un 85.73%, 33420 registros de 33450 (99.91%), 338 I/O de 400 (84.50%) y 718 DSP de 740 (97.03%); mientras que los demás recursos tienen poco uso. En cuanto a flip flop se usa 0.06%, BUFG usa 3.13% y block RAM no se utiliza nada de los 365 disponibles. Esto muestra el trabajo que necesita hacer el FPGA para realizar las implementaciones internas que se hicieron en los programas de Matlab, y Simulink, y obtener los mismos resultados en ambos osciladores.

5. REFERENCIAS

[1] L. Chua, “Memristor-The missing circuit element”, IEEE Transactions On Circuit Theory, vol. 18, No. 5, pp. 507-519, ene. 1971.

[2] D. B. Strukov, G. S. Snider, D. R. Stewart, y R. S. Williams, “The missing memristor found”, Nature, vol. 453, No. 7191, pp. 80-83, may 2008.

[3] C. Aldrete-Maldonado, et al., “Sincronización de sistemas caóticos de múltiple enrollamiento mediante observador de estado extendido”, PÄDI Boletín Científico de Ciencias Básicas E Ingenierías del ICBI, vol. 11, No. 2, pp. 110-116, sep. 2023.

[4] M. Vinagre, V. Feliu-Batlle y I. Tejado, “Control fraccionario: fundamentos y guía de uso”, Revista Iberoamericana de Automática e Información Industrial, vol. 13, pp. 265-280, 2016.

[5] O. C. Ibe, “Diffusion processes”, en Elsevier eBooks, 2013, pp. 295-327. doi: 10.1016/b978-0-12-407795-9.00010-4.

[6] J. A. T. Machado y V. Kiryakova, “The Chronicles of Fractional Calculus”, Fractional Calculus And Applied Analysis, vol. 20, No. 2, pp. 307-336, abr. 2017, doi: 10.1515/fca-2017-0017.

[7] “Runge-Kutta and Extrapolation Methods”, en Springer eBooks, 2008, pp. 129-353. doi: 10.1007/978-3-540-78862-1_2.

[8] F. Ghoreishi, R. Ghaffari, y N. Saad, “Fractional Order Runge–Kutta Methods”, Fractal And Fractional, vol. 7, No. 3, p. 245, mar. 2023, doi: 10.3390/fractalfract7030245.

[9] Petras, I., & Chen, Y. (2012). Fractional-order circuit elements with memory. Proceedings of the 13th International Carpathian Control Conference (ICCC), Podbanské, Slovakia, 28-31 May 2021, 552–558. https://doi.org/10.1109/CarpathianCC.2012.6228706.

[10] Q. Zhao, C. Wang, y X. Zhang, “A universal emulator for memristor, memcapacitor, and meminductor and its chaotic circuit”, Chaos an Interdisciplinary Journal of Nonlinear Science, vol. 29, No. 1, ene. 2019, doi: 10.1063/1.5081076.

[11] Y. Li, L. Xie, C. Zheng, D. Yu, y J. K. Eshraghian, “Modeling and hardware implementation of universal interface-based floating fractional-order mem-elements”, Chaos An Interdisciplinary Journal Of Nonlinear Science, vol. 33, No. 1, enero 2023, doi: 10.1063/5.0124793.