CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACATENCO

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

SECCIÓN DE ELECTRÓNICA DEL ESTADO SÓLIDO

Prototipo en FPGA de red neuronal con memristores

TESIS QUE PRESENTA:

ING. ERICK MORENO CUELLAR

PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS

EN LA ESPECIALIDAD DE:

INGENIERÍA ELÉCTRICA

DIRECTORES DE TESIS:

DR. FELIPE GÓMEZ CASTAÑEDA

DR. JOSÉ ANTONIO MORENO CADENAS

Ciudad de México Diciembre, 2019

Agradecimientos

A mi familia, por el constante apoyo durante estos dos años dedicados al estudio de mi maestría; gracias por siempre estar ahí, y alentarme en los momentos más difíciles.

Al Consejo Nacional de Ciencia y Tecnología (**CONACYT**), por la beca otorgada para el desarrollo de este trabajo.

A mis asesores de tesis, el Dr. Felipe Gómez Castañeda y el Dr. José Antonio Moreno Cadenas, por dedicar el tiempo y la paciencia para transmitir sus conocimientos a lo largo de esta etapa.

Al M. en C. Luis Martín Flores Nava, por todo su apoyo y consejos en el desarrollo de este proyecto; demostrando tener una admirable dedicación a su profesión.

A mis revisores y sinodales, el Dr. Gabriel Romero Paredes Rubio y el Dr. Ramón Peña Sierra, por su valioso tiempo dedicado a la revisión de este trabajo.

Al personal académico y administrativo de la SEES, por proporcionar las herramientas, consejos y orientación durante mi estancia en el CINVESTAV.

CONTENIDO

Definiciones:

- Matlab: Software de computación numérica.
- Simulink: aplicación de matlab, en la cual es posible programar de manera gráfica
- System Generator: aplicación dentro de Simulink que la programación de manera gráfica del FPGA, así como la simulación y co-simulacion.
- Jitter: ligera desviación en la exactitud de señales.
- Unit 8: tipo de archivo de datos en matlab, donde los valores se encuentran entre 0-255.
- STDP: abreviatura de Spike Timing Dependent Plasticity, es una regla de aprendizaje de aprendizaje encargada de modificar los pesos sinápticos por los cuales se comunican las neuronas.

Resumen

En este trabajo de tesis, se aborda una metodología para la simulación e implementación de un modelo del memristor bipolar que es controlado por voltaje y de tipo genérico, según la clasificación de memristores propuesta por el Dr. L.O. Chua ref. [1], tomados como una familia completa de componentes y sistemas Memristivos. Este modelo es utilizado en aplicaciones que consideran las siguientes tareas: ejecución de la función de memoria asociativa en redes neuronales pulsadas, el reconocimiento de caracteres, la optimización al problema de rutas mediante la emulación del algoritmo de hormigas basados en la metaheurística y el procesamiento de imágenes.

Para llevar a cabo lo anterior, es necesario revisar el origen del memristor propuesto por el Dr. L.O. Chua ref. [2], dando una breve introducción de los modelos o paradigmas del memristor desde su concepción, partiendo desde el modelo reportado por los laboratorios Hewlett Packard, el cual está basado en un memristor físico.

También, será necesario conocer las bases de las redes neuronales pulsadas, y como es que el memristor se asocia con ellas, a partir de su semejanza con la sinapsis biológica y la regla de aprendizaje: Plasticidad Dependiente del Tiempo en Pulsos, conocida como: STDP (Spike Time Dependent Plasticity)

Se utilizará un modelo del memristor que sea útil en todas las aplicaciones, tanto en la red neuronal pulsada, como al ser usado en redes de memristores. Para poder usar este modelo en los desarrollos será necesario incluir una descripción fundamental del algoritmo de colonia de hormigas, así como su análogo dinámico con redes de memristores.

Abstract

This work addresses a methodology for the simulation and implementation of a bipolar memristor model that is controlled by voltage and generic type is approached, within the classification of memristors proposed by Dr. L.O. Chua [1], taken as a complete family of memristive components and systems. This model is used in applications that consider the following tasks: execution of the associative memory in spiking neural networks, character recognition, optimization to the paths problem by emulating the ant algorithm belonging to the collective intelligence algorithms and image processing.

For the implementation of the above described, it is necessary to review the origin of memristor proposed by Dr. L.O. Chua [2], introducing a brief explanation of the models or paradigms from its conception, starting from the model reported by the Hewlett Packard laboratories, which is based on a physical memristor.

Also, it will be necessary to know the bases of the spiking neural networks, and how the memristor is associated with them, from its similarity with the synapse and the learning rule: Spike Time Dependent Plasticity (STDP).

The same memristor model will be used in all applications, both in the spiking neural network, and when used in memristors networks. In order to use this model, it will be necessary to include a fundamental description of the ant colony algorithm, as well as its dynamic analogue with memristors networks.

Introducción

El profesor de Berkeley, Leon Chua publicó en 1971 ref. [2] “Memristor-The Missing Circuit Element”, proponiendo un nuevo elemento eléctrico pasivo de dos terminales conocido como memristor (resistencia con memoria), más tarde en 1973 junto con su alumno Kang presentaron los sistemas Memristivos ref. [3], sin embargo, el concepto de memristor continuaba sin tener una relevancia significativa.

Fue hasta el año 2008 cuando el equipo de investigadores de Hewlett-Packard aseguró haber encontrado el memristor propuesto por Chua, que la investigación sobre este dispositivo se vio favorecida. El trabajo de los investigadores se basaba en una estructura del tipo Metal-Insulator-Metal desarrollada para aplicaciones de memoria no volátiles.

Hoy en día, se tiene un conocimiento mucho más extenso, e inclusive es conocido que los memristores pueden ser de diferentes materiales, además de que este tipo de comportamiento puede ser encontrado en entes vivos, como en amibas, calamares y plantas, e inclusive en el tradicional arco eléctrico.

A pesar de la controversia que existe alrededor del término memristor, los esfuerzos han sido encaminados en hallar un modelo que sea capaz de representar el comportamiento eléctrico.

También se han propuesto aplicaciones futuras de este dispositivo, dado que el Memristor combina la conmutación entre un estado alto de resistencia y uno bajo, junto con la memoria no volátil (procesamiento y almacenamiento de datos), los usos van desde memorias de mayor capacidad en un menor espacio, osciladores, el cómputo memristivo o “memcomputing”, hasta representar a un equivalente de la sinapsis electroquímica presente en el cerebro humano.

Por estos motivos, se propone el uso de un modelo de memristor implementado en tecnología digital, el cual sea capaz de interconectarse con otros dispositivos de manera física o simulada. Para así poder realizar experimentos e implementaciones y obtener los resultados que se esperarían de un memristor físico.

Objetivos

Objetivo General

- Simular e implementar un modelo del Memristor en tecnología digital usando un sistema FPGA y mostrar diferentes aplicaciones.

Objetivos Específicos

- Implementar el modelo del Memristor bipolar con umbral en la tarjeta Nexys 4DDR (FPGA) de Xilinx y realizar la comprobación de los fingerprints que avalen su funcionamiento.
- Simular e implementar una neurona pulsada, accesible computacionalmente para su uso en un FPGA.
- Realizar la simulación de una red neuronal pulsada, la cual reproduzca el experimento Pavlov´s Dog usando el Memristor como sinapsis entre las neuronas y posteriormente implementarla en la tarjeta Nexys 4DDR(FPGA).
- Simular e implementar una red neuronal pulsada para el reconocimiento de caracteres en una matriz simple de 3x3 dinámica y realizar las pruebas de funcionamiento en el FPGA.
- Realizar una red de Memristores usándolos como análogos del algoritmo metaheurístico de hormigas, para encontrar el camino más corto entre dos puntos de la red, además de comprobar el correcto desempeño con la implementación en el FPGA.
- Realizar la simulación de una red de Memristores usada en la detección de bordes en imágenes y compararla con el algoritmo de hormigas.

Capítulo 1 Memristores

1.1 Introducción

Por la relevancia del memristor, así como en el mercado existen solo algunos laboratorios encargados de su producción, además de contar con pocos memristores físicos accesibles para su experimentación, surge la necesidad de una implementación, ya sea analógica o digital, con la cual sea posible interactuar junto con otros elementos fundamentales y dispositivos electrónicos. Siendo así, que los reportes de investigación del memristor han incrementado de manera notable en los últimos años como se visualiza en la *Figura* 1.1 de acuerdo a la ref. [4].

En este capítulo se presenta un marco de referencia que permitirá comprender el trabajo realizado, así como los resultados obtenidos, en cuanto a las aplicaciones dadas a un modelo de Memristor accesible computacionalmente y conveniente para la implementación digital.

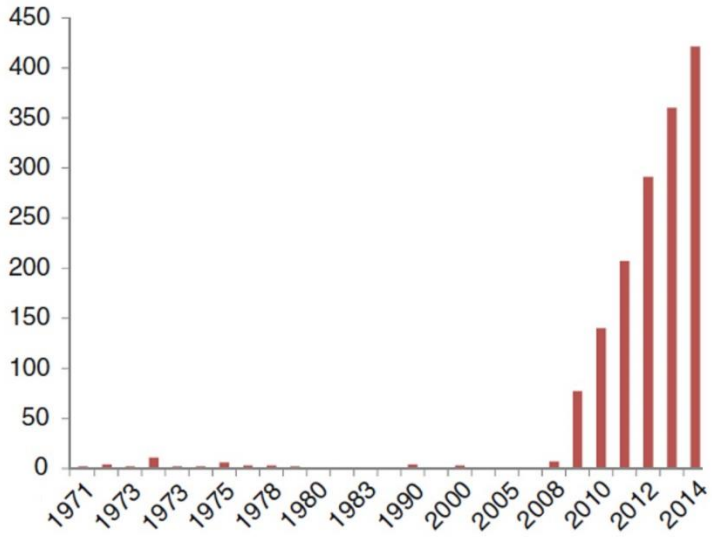

Figura 1.1 Número de publicaciones acerca de Memristores en Engineering Village, ref. [4].

1.2 El Memristor

Con base en la simetría observada entre los elementos eléctricos fundamentales resistencia (R), capacitor (C) e inductor (L), así como las variables que ellos asocian: voltaje, corriente, carga y flujo magnético (v, i, q, φ), respectivamente, el profesor Leon Chua propuso que debería existir un cuarto elemento para la relación faltante entre flujo magnético $\varphi(t) \equiv \int_{-\infty}^{t} v(\tau)d\tau$ y la carga $q(t) \equiv \int_{-\infty}^{t} i(\tau)d\tau$, cuando se incluyen variaciones con el tiempo. Al profundizar más en la matemática y propiedades de este "nuevo" elemento, encontró que debería ser resistencia con memoria, así que lo nombró Memristor, ref. [1].

Como se puede apreciar en la *Tabla* 1.1, con las cuatro variables que se tienen, es posible realizar 6 combinaciones, cinco de ellas ya tenían definición, pero en cuanto al flujo magnético y carga aún no se contaba con ella.

Variables	***Relación***
$\boldsymbol{v, i}$	$R = \frac{dv}{di}$
$\boldsymbol{v, q}$	$C = \frac{dq}{dv}$
$\boldsymbol{v, \varphi}$	$v = \frac{d\varphi}{dt}$
$\boldsymbol{i, q}$	$i = \frac{dq}{dt}$
$\boldsymbol{i, \varphi}$	$L = \frac{d\varphi}{di}$
$\boldsymbol{\varphi, q}$	*Relación faltante*

Tabla 1.1 Las 5 relaciones conocidas previas al Memristor.

La propuesta de Chua justificaba la relación faltante con un "nuevo" elemento de dos terminales en el cual el flujo magnético y la carga se encontraban vinculados como se visualiza en la ecuación (*1.1*), así como en la *Figura 1.2*.

$$M = \frac{d\varphi}{dq} \qquad (1.1)$$

En el caso de elementos lineales, M es una constante idéntica a la resistencia (Memristancia), pero en el caso donde M es una función en si misma de q, produce una relación y elemento no lineales, en donde este "nuevo" elemento fundamental no puede ser emulado con ninguna combinación de los otros tres conocidos.

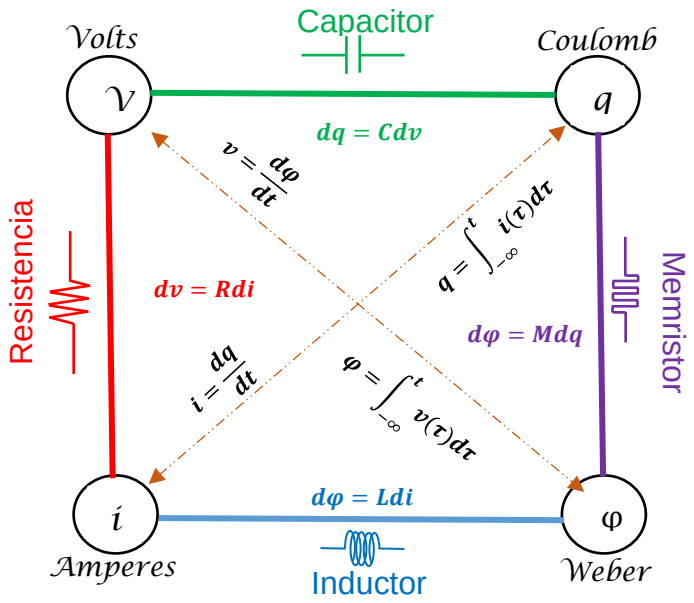

Figura 1.2 Elementos fundamentales donde se aprecia la simetría y relaciones entre las variables de cada uno, ref. [4].

En la *Figura* 1.2 se muestran las relaciones entre las cuatro variables fundamentales de la teoría de circuitos; con la propuesta de Chua, se agrega el Memristor, donde se aprecia la relación entre flujo magnético y carga.

Los modelos de Memristores pueden clasificarse de distintas maneras, por el tipo de control (i, v, q, φ), características, etc. Una de las maneras más sencillas para clasificarlos es dependiendo de la complejidad de las ecuaciones que los definen; debido a esto pueden ser ideales, genéricos ideales, genéricos y extendidos. También gracias a la experimentación con dispositivos físicos, es conocido que la

mayoría suele tener un umbral a partir del cual comienzan a mostrar sus propiedades.

Como se nuestra en la *Figura* 1.3, la manera en la que Chua explica el memristor es partiendo de la idea de un memristor ideal, expandiendo la definición hasta llegar al memristor extendido.

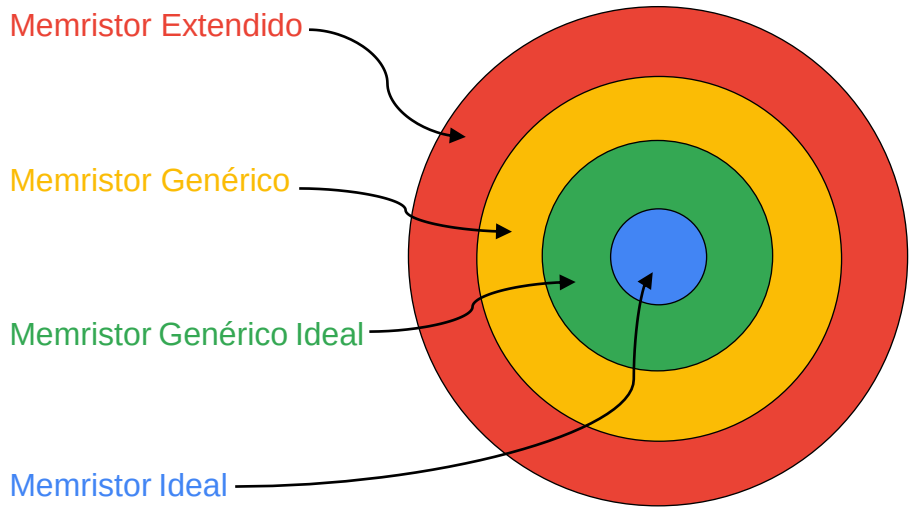

Figura 1.3 El universo del memristor, según la ref. [1].

Para entender el comportamiento del memristor en función de una corriente aplicada, es necesario dar un breve repaso por un elemento lineal ya conocido (resistencia), hasta llegar al memristor extendido.

Elemento lineal, la resistencia

Uno de los elementos fundamentales donde se puede apreciar la ley de Ohm, así como la función lineal para la gráfica v-i, según la ecuación (1.2) e ilustrado en la *Figura* 1.4.

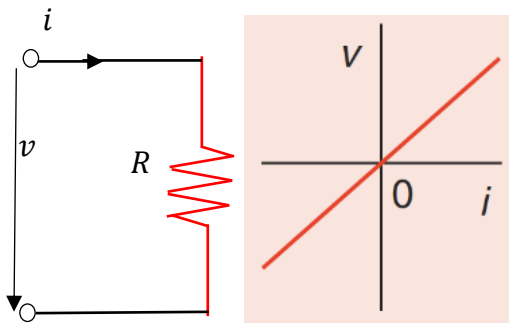

Figura 1.4 Gráfica I-V de una resistencia típica.

$$v = Ri \qquad (1.2)$$

Donde *R* es la resistencia de valor constante.

Elemento no lineal, una resistencia no lineal

En este caso el valor de la resistencia se encuentra en función de la corriente que se inyecta al circuito como se observa en la ecuación (1.3), esto provoca un cambio no lineal del voltaje, como se visualiza en la *Figura* 1.5.

$$v = R(i)i \qquad (1.3)$$

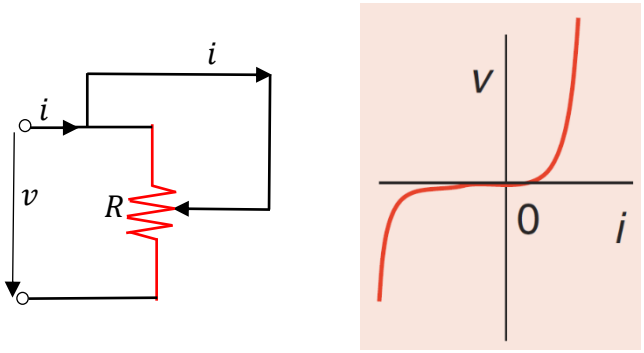

Figura 1.5 Cura I-V de una resistencia no lineal.

Ya que la resistencia es $R(i)$, al inicializarla en un valor alto (*R>100*) el cambio exponencial estará presente casi de inmediato.

1.2.1 Memristor Ideal

El memristor ideal como menciona Chua en la ref. [2], se define en función del mecanismo de control (carga o flujo), a través de las relaciones constitutivas:

$$\varphi = \hat{\varphi}(q) \qquad (1.4)$$

$$q = \hat{q}(\varphi) \qquad (1.5)$$

Para un memristor controlado por carga derivando la ecuación (1.4) se obtiene la ecuación (1.6), que se escribe como:

$$\frac{d\varphi}{dt} = \frac{d\hat{\varphi}(q)}{dq}\frac{dq}{dt} \qquad (1.6)$$

$$v = R(q)\ i$$

Dado que la corriente es el cambio de la carga en función del tiempo ($i = \frac{dq}{dt}$) como se muestra en la *Figura* 1.2, es posible identificar el equivalente de la resistencia en la expresión (1.6), la ya difiere de la ecuación (1.3).

$$R(q) = \frac{d\hat{\varphi}(q)}{dq} \qquad (1.7)$$

Con esto, es posible describir al memristor controlado por carga, adicionalmente se puede obtener la ecuación constitutiva por medio de $\varphi(0)$ para validar la igualdad:

$$\varphi \triangleq \varphi(0) + \int_0^q R(q)\, dq \triangleq \hat{\varphi}(q) \qquad (1.8)$$

Ahora, revisando los términos de la ecuación (1.8) y realizando la derivada en función del tiempo para este tipo de control, se tiene que el voltaje es:

$$v = R(q)\frac{dq}{dt} \qquad (1.9)$$

Obteniendo así el memristor controlado por corriente, en donde la variable responsable del cambio de resistencia es la carga; este sistema está descrito por:

$$v = R(q)i \tag{1.10}$$

$$\text{Con: } \frac{dq}{dt} = i \tag{1.11}$$

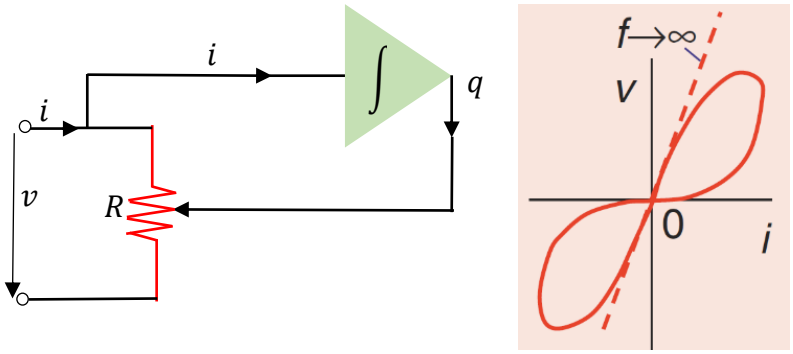

Figura 1.6 Representación gráfica del memristor ideal de acuerdo a la ref. [5].

En la *Figura* 1.6 se representa el caso ideal, donde la única variable responsable del cambio de resistencia (Memristancia) es la carga, así como el circuito equivalente del memristor ideal. De manera similar es posible tomar la relación constitutiva (1.5) del control por flujo para obtener el memristor controlado por voltaje como en la ecuación (1.12):

$$q \triangleq q(0) + \int_0^{\varphi} G(\varphi)\, d\varphi \triangleq \hat{q}(\varphi) \tag{1.12}$$

Nuevamente, realizando la derivada en el tiempo y observando los términos de esta ecuación tenemos:

$$\frac{dq}{dt} = G(\varphi)\frac{d\varphi}{dt} \tag{1.13}$$

Ahora, G es la Memductancia, y obtenemos un memristor controlado por voltaje como se muestra:

$$i = G(\varphi)v \tag{1.14}$$

$$\frac{d\varphi}{dt} = v \tag{1.15}$$

Otra forma de validar que la ecuación constitutiva mantiene la igualdad, como se menciona en la ref. [6], es a través de la ecuación de la carga mostrada en la *Figura* 1.2, así como en la ecuación (1.16):

$$q(t) = \int_{-\infty}^{t} i(\tau)d\tau \tag{1.16}$$

Tomando la ecuación de la corriente (1.14) y sustituyéndola en (1.16) se tiene:

$$q(t) = \int_{-\infty}^{t} G\big(\varphi(\tau)\big)\frac{d\varphi(\tau)}{d\tau}d\tau \tag{1.17}$$

Realizando la integral para obtener la carga como función del tiempo, obtenemos:

$$q(t) = \int_{q(-\infty)}^{q(t)} G(\varphi)d\varphi = \hat{q}\big(\varphi(t)\big) \tag{1.18}$$

$$q = \hat{q}(\varphi)$$

Donde se precisa nuevamente la relación constitutiva para el memristor controlado por flujo, asumiendo entonces la definición de las ecuaciones (1.14) y (1.15). Además, para fines prácticos M sustituirá a R para la Memristancia.

Adicionalmente, Chua menciona en la ref. [1] dos aspectos relevantes para la comprensión del memristor ideal, y para poder enlazar estas ideas enuncia:

Los lazos de la histéresis con cruce en el origen dependen de los estados iniciales.

> *"Dos lazos de histéresis con cruce en el origen medidos a partir de un memristor ideal excitado por la misma fuente de voltaje periódico, pero con diferentes estados iniciales pueden verse muy diferentes entre sí."*

Teorema de simetría del lazo de la histéresis con cruce en el origen

> *"El lazo de histéresis con cruce en el origen de cualquier memristor ideal activado por una entrada impar-simétrica de media onda, fuente de voltaje o corriente, es impar-simétrico y se cruzan entre sí las curvas de corriente y voltaje con distintas pendientes en el origen, siempre que la función corriente-voltaje sea bivaluada"*

1.2.2 Memristor Genérico Ideal

Dado un memristor ideal es posible generar un número infinito de Memristores hermanos genéricos ideales, esto a partir de escoger alguna función 1:1 diferenciable por tramos, como en la ref. [1]; los pasos para crear un memristor hermano controlado por voltaje son:

Dada la relación constitutiva $q = \hat{q}(\varphi)$

1) Escoger una función 1:1 diferenciable por partes: $x = \hat{x}(\varphi)$ y despejar la función inversa: $\varphi = \hat{x}^{-1}(x)$

2) Usar la ecuación conocida

$$G(x) \triangleq \frac{d\hat{q}(\varphi)}{d\varphi}\Big|_{\varphi=\hat{x}^{-1}(x)} \qquad (1.19)$$

3) Definir la nueva variable de estado

$$\hat{g}(x) = \frac{d\hat{x}(\varphi)}{d\varphi}\Big|_{\varphi=\hat{x}^{-1}(x)} \quad (1.20)$$

4) Reescribir el memristor hermano como:

$$i = G(x)v \quad (1.21)$$

$$\frac{dx}{dt} = \hat{g}(x)v \quad (1.22)$$

De la misma manera, se puede realizar el proceso para el memristor genérico ideal controlado por corriente, tomando la relación constitutiva del flujo $\varphi = \hat{\varphi}(q)$

1) Usar una ecuación 1:1 diferenciable por partes $x = \hat{x}(q)$, despejando la función inversa $q = \hat{x}^{-1}(x)$

2) Tomar la ecuación conocida para evaluar y definir la nueva variable de estado:

$$M(x) \triangleq \frac{d\hat{\varphi}(q)}{dq}\Big|_{q=\hat{x}^{-1}(x)} \quad (1.23)$$

$$\hat{f}(x) = \frac{d\hat{x}(q)}{dq}\Big|_{q=\hat{x}^{-1}(x)}$$

3) Reescribir el Memristor Genérico ideal controlado por corriente, siguiendo los pasos como en el controlado por voltaje.

$$v = M(x)i \quad (1.24)$$

$$\frac{dx}{dt} = \hat{f}(x)i \quad (1.25)$$

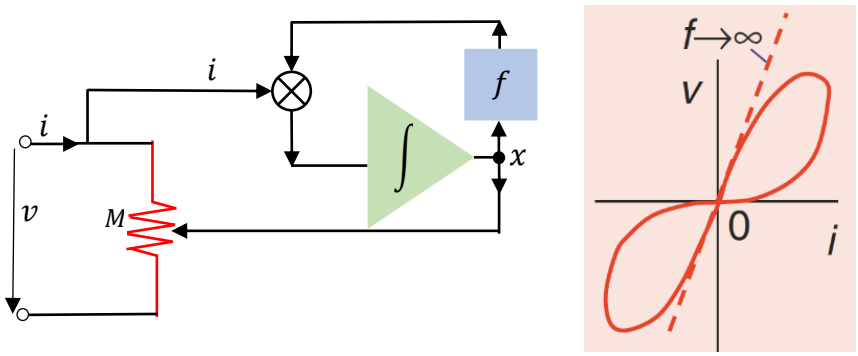

Figura 1.7 Curva Memristor Genérico Ideal de acuerdo a la ref. [5].

Ahora x es una incógnita escalar en lugar de ser solo la carga, donde será la responsable del cambio de memristancia (ecuación (1.24)). En la *Figura* 1.7 se aprecia la relación de las ecuaciones con este modelo; para este caso, Chua también menciona en la ref. [1] un aspecto relevante de esta clase de memristor:

Teorema del lazo de histéresis con cruce en el origen idéntico

"Cada memristor ideal y sus infinitos Memristores hermanos ideales genéricos exhiben idénticos lazos de histéresis con cruce en el origen en el plano de voltaje-corriente, cuando son accionados por la misma señal de entrada y las mismas condiciones iniciales"

1.2.3 Memristor Genérico

Un memristor es llamado genérico si, a diferencia del memristor genérico ideal, en su ecuación de estado la corriente aparece dentro de la función no lineal como se muestra en la ecuación (1.27):

$$v = M(x)i \qquad (1.26)$$

$$\frac{dx}{dt} = f(x,i) \qquad (1.27)$$

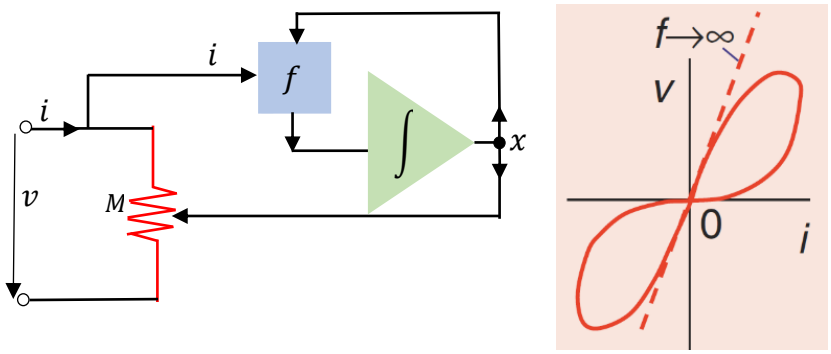

Figura 1.8 Curva Memristor Genérico acorde a la ref. [5].

Como se aprecia en la *Figura* 1.8, la corriente pasa a ser parte de la función que modifica el cambio de la variable x responsable del valor de la Memristancia, a diferencia del memristor genérico ideal en donde solo es un valor que multiplica a la variable de estado.

Para explicar este modelo del memristor, Chua menciona en la ref. [1] varios ejemplos de sistemas los cuales dejan de ser Memristores genéricos ideales, dando paso al modelo genérico.

Ejemplo 1

Termistor NTC (Coeficiente de temperatura negativo) controlado por voltaje según la ref. [1], en donde la resistencia decrece conforme la temperatura aumenta, el cual puede ser modelado a partir de la ley de ohm dependiente del estado (ecuación (1.28)) y la ecuación de estado (1.29):

$$i = W(x)v \tag{1.28}$$

$$\frac{dx}{dt} = \frac{\delta_N}{H_{CN}}(T_{0N} - x) + \frac{W(x)}{H_{CN}}v^2 \tag{1.29}$$

La Memductancia en este ejemplo es definida por:

$$W(x) = \left[R_{ON}e^{\beta_N\left(\frac{1}{x}-\frac{1}{T_{0N}}\right)}\right]^{-1} \tag{1.30}$$

En donde $\delta_N, \beta_N, H_{CN}, R_{ON}, T_{0N}$ son constantes y la variable de estado x denota la temperatura del termistor NTC.

Ejemplo 2

Termistor PTC (Coeficiente de temperatura positivo) controlado por voltaje, en donde la resistencia aumenta conforme al incremento de temperatura, dicho comportamiento es modelado mediante:

$$i = W(x)v \tag{1.31}$$

$$\frac{dx}{dt} = \frac{\delta_P}{H_{CP}}(T_{0P} - x) + \frac{W(x)}{H_{CP}}v^2 \tag{1.32}$$

Teniendo como en el caso anterior la ley de ohm dependiente del estado (ecuación (1.31)) y la variable de estado (ecuación (1.32)), así como una Memductancia descrita por:

$$W(x) = \left[R_{OP}e^{\beta_P(x - T_{0P})}\right]^{-1} \tag{1.33}$$

En donde $\delta_p, \beta_p, H_{Cp}, R_{Op}, T_{0p}$ son constantes, y la variable de estado x indica la temperatura interna del termistor PTC.

Ejemplo 3

Modelo de la neurona de Hodgkin-Huxley para canal de iones de sodio, el cual describe como se generan y transmiten los potenciales de acción en una neurona. En donde la ley de ohm dependiente del estado (ecuación (1.34)), así como sus ecuaciones de estado (1.35) y (1.36) son:

$$i = G(x_1, x_2)v \tag{1.34}$$

$$\frac{dx_1}{dt} = f_m(x_1, v) \tag{1.35}$$

$$\frac{dx_2}{dt} = f_h(x_2, v) \tag{1.36}$$

En este caso, la Memductancia está descrita en la ecuación (1.37) como:

$$G(x_1, x_2) = \bar{g}_{Na} x_1^3 x_2 \tag{1.37}$$

Donde las funciones $f_m \; y \; f_h$ son las ecuaciones (1.38) y (1.39) respectivamente, además $\bar{g}_{Na} \;\; y \; E_{Na}$ son constantes.

$$f_m(x_1, v) \triangleq \left\{ \frac{0.1[(v - E_{Na}) + 25]}{exp\left[\frac{(v - E_{Na}) + 25}{10}\right] - 1} \right\} (1 - x_1) - 4\left\{ exp\left[\frac{(v - E_{Na})}{18}\right] \right\} x_1 \tag{1.38}$$

$$f_h(x_2, v) \triangleq \left\{ 0.007 \; exp\left[\frac{(v - E_{Na})}{10}\right] \right\} (1 - x_2) - \left\{ \frac{1}{exp\left[\frac{(v - E_{Na}) + 30}{10}\right] + 1} \right\} x_2 \tag{1.39}$$

Ejemplo 4

Memristor de segundo orden según la ref. [1], donde se tienen dos variables de estado para representar con una mayor precisión los mecanismos internos responsables de la conmutación resistiva.

Ley de ohm dependiente del estado:

$$i = G(x_1, x_2)v \tag{1.40}$$

Ecuaciones de estado:

$$\frac{dx_1}{dt} = f_1(x_1, x_2, v) \tag{1.41}$$

$$\frac{dx_2}{dt} = f_2(x_1, x_2, v) \tag{1.42}$$

La memductancia está dada por:

$$G(x_1, x_2) = \frac{1}{\left[(K_1 e^{\beta_1 (x_1 - \gamma_1)}) + \left(K_2 e^{\beta_2 \left(\frac{1}{x_2} - \frac{1}{\gamma_2}\right)}\right)\right]} \tag{1.43}$$

$$f_1(x_1, x_2, v) \triangleq \frac{1}{\alpha_1}\left[\delta_1(\gamma_1 - x_1) + \frac{K_1 e^{\beta_1 (x_1 - \gamma_1)}}{\left[(K_1 e^{\beta_1 (x_1 - \gamma_1)}) + \left(K_2 e^{\beta_2 \left(\frac{1}{x_2} - \frac{1}{\gamma_2}\right)}\right)\right]^2} v^2\right] \tag{1.44}$$

$$f_2(x_1, x_2, v) \triangleq \frac{1}{\alpha_2}\left[\delta_2(\gamma_2 - x_2) + \frac{K_2 e^{\beta_2 \left(\frac{1}{x_2} - \frac{1}{\gamma_2}\right)}}{\left[(K_1 e^{\beta_1 (x_1 - \gamma_1)}) + \left(K_2 e^{\beta_2 \left(\frac{1}{x_2} - \frac{1}{\gamma_2}\right)}\right)\right]^2} v^2\right] \tag{1.45}$$

Donde $\alpha_1, \alpha_2, \beta_1, \beta_2, \delta_1, \delta_2, \gamma_1, \gamma_2, K_1$ y K_2 son constantes.

Como se observa en los ejemplos anteriores, la variable x deja de ser un escalar para dar paso a la posibilidad de ser un vector, en donde varios parámetros modificarán la Memristancia o Memductancia; en tales casos es apreciable como la función que representa el cambio de la variable x ahora se encuentra influenciada por la variable de entrada (v o i) a diferencia del memristor ideal genérico.

1.2.4 Memristor Extendido

Dado que el memristor propuesto por Chua aún no estaba presente físicamente, pero si sistemas que presentaban similitudes con él y que, aunque no eran modelados de manera real, ya podían observarse, Chua y su alumno Kang se dieron a la tarea de investigar más a fondo.

Como Chua mencionó en 1976 en la ref. [3] "*El memristor es solo un caso especial de una clase mucho más amplia llamada Sistemas Memristivos"*.

La propuesta de Chua y Kang es un sistema de dos ecuaciones para describir el comportamiento de los sistemas Memristivos. La primera ecuación (1.46) representa la salida (v o i), en cuanto a la segunda ecuación (1.47) presenta como evoluciona el sistema a partir de una variable de entrada u (i o v);

$$y = G(x,u,t)u \tag{1.46}$$

$$\frac{dx}{dt} = f(x,u,t) \tag{1.47}$$

En donde x es el estado del sistema, la función *f* es una función vectorial de n dimensiones y *G* es una función escalar continua que representa la Memristancia o Memductancia, ya sea el caso.

Posteriormente, en el año 2015 Chua reclasifica a los Memristores según la ref. [1] para una mejor comprensión, por lo que los sistemas Memristivos pasan a ser considerados en la categoría del Memristor Extendido.

En este trabajo, Chua enuncia: "*Un Memristor es llamado Memristor Extendido si su Memristancia* $(M(x,i))$ *o Memductancia (*$G(x,v)$*) no solo es función de las variables de estado* $x = (x_1, x_2, \dots, x_n)$*, sino también de la fuente de corriente o voltaje, dependiendo del caso"*

Lo cual concuerda con la definición dada en 1976 ref. [3]. Observando las ecuaciones (1.46) y (1.48), se aprecia como el voltaje también modifica la Memductancia directamente para el Memristor controlado por voltaje, a diferencia del memristor genérico donde la variable de entrada (voltaje o corriente) afecta solo a la variable de estado x (ecuaciones (1.26) y (1.27)).

$$i = G(x,v)v$$
$$G(x,0) \neq \infty \tag{1.48}$$

$$\frac{dx}{dt} = g(x,i) \tag{1.49}$$

En adición, se menciona: "*la memductancia o memristancia debe ser un número finito distinto de infinito cuando se tiene un estímulo nulo*", como en las ecuaciones (1.48) y (1.50), lo cual es una característica de los Memristores.

Para el caso del control por corriente se tiene:

$$v = M(x,i)i$$
$$R(x,0) \neq \infty \qquad (1.50)$$

$$\frac{dx}{dt} = f(x,i) \qquad (1.51)$$

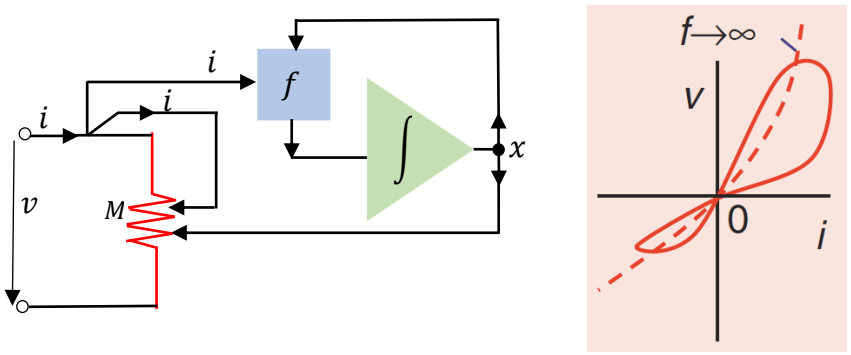

Figura 1.9 Diagrama y grafica I-V del Memristor extendido según ref. [5].

En la *Figura* 1.9, se visualiza un diagrama, el cual representa el Memristor extendido, donde adicionalmente a la variable de estado x, la corriente interviene en el cambio de la Memristancia directamente, similar a una combinación de la resistencia no lineal y el Memristor Genérico. Se muestra también la gráfica i-v en

la cual se aprecia la intervención de la corriente, modificando la histéresis y el tamaño de los lóbulos.

Resumiendo, en la Tabla 1.2 se presenta la clasificación de los Memristores, donde se encuentran las cuatro clases mostradas anteriormente, así como las ecuaciones que las describen, para cada tipo de control (corriente o voltaje).

Dispositivo	*Controlado por Corriente*	*Controlado por Voltaje*
Memristor Ideal	$v = M(q)i$ $\frac{dq}{dt} = i$	$i = G(\varphi)v$ $\frac{d\varphi}{dt} = v$
Memristor Genérico Ideal	$v = M(x)i$ $\frac{dx}{dt} = \hat{f}(x)i$	$i = G(x)v$ $\frac{dx}{dt} = \hat{g}(x)v$
Memristor Genérico	$v = M(x)i$ $\frac{dx}{dt} = f(x,i)$	$i = G(x)v$ $\frac{dx}{dt} = g(x,v)$
Memristor Extendido	$v = M(x,i)i$ $R(x,0) \neq \infty$ $\frac{dx}{dt} = f(x,i)$	$i = G(x,v)v$ $G(x,0) \neq \infty$ $\frac{dx}{dt} = g(x,i)$

Tabla 1.2 Clasificación de Memristores según su complejidad y tipo de control.

1.3 Fingerprints del Memristor

Los memristores cuentan con ciertas características con las cuales es posible identificarlos, estas características son conocidas como los fingerprints del memristor, siendo posible determinar si un dispositivo es un memristor aun desconociendo el comportamiento físico interno. Para catalogar un dispositivo como Memristor debe tener 3 características importantes mencionadas en la ref. [7].

- Una histéresis con cruce en el origen (Figura de Lissajous) para su grafico i-v o v-i dependiendo del caso.
- La histéresis disminuye proporcionalmente con el aumento de la frecuencia.

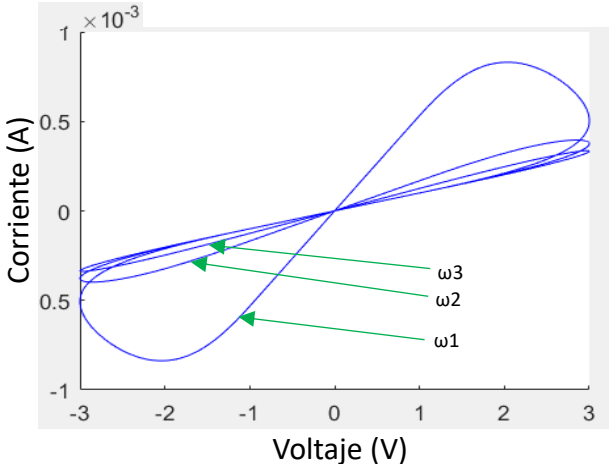

Figura 1.10 Histéresis característica del Memristor, donde ω1<ω2<ω3.

Es así que si la señal de excitación es demasiado rápida solo se obtendrá una recta similar a la de una resistencia como se aprecia en la *Figura* 1.10, y el fenómeno memristivo quedara oculto, razón por la que pudo haber pasado desapercibido, como se menciona en las referencias [6] y [7].

- La no volatilidad de su memoria

Resultado de la dependencia de la variación de su estado, por lo que recuerda el último valor de resistencia aun cuando la señal de excitación haya cesado (v o i) y la cual cambiará hasta que nuevamente una señal le sea aplicada, dicha capacidad le otorga la posibilidad de ser utilizado como memoria no volátil.

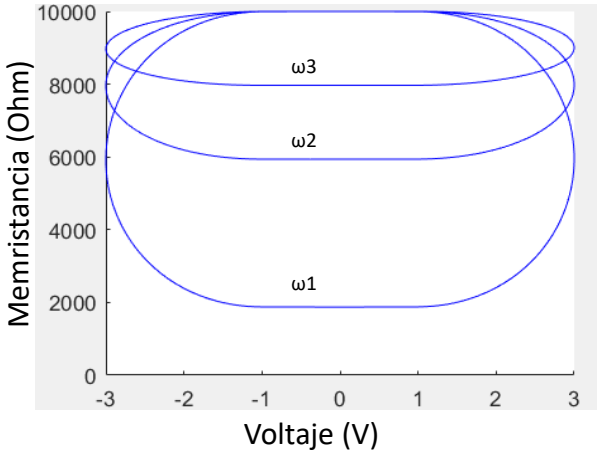

Figura 1.11 Cambio de resistencia dependiente de la frecuencia, donde ω1<ω2<ω3.

En la *Figura* 1.11, se observa como la resistencia alterna entre dos estados, alto y bajo, en donde dependiendo de la polarización ésta se verá inclinada hacia uno de los extremos, además la frecuencia delimita el valor mínimo que pueda tomar el Memristor.

Como se aprecia en la *Figura* 1.11, a una mayor frecuencia los valores entre los que alterna la resistencia son cada vez más cercanos, por lo que si se continúa aumentado la frecuencia la Memristancia no se ve afectada, provocando esto un comportamiento lineal de la misma manera que en la resistencia. Observando las figuras *Figura* 1.10 y *Figura* 1.11, se nota como en la frecuencia más baja la histéresis es claramente visible, así como su conmutación; en cambio en alta

frecuencia, la histéresis casi desaparece y su cambio de resistencia, que aún está presente es mínimo.

Estas propiedades dan una idea amplia en la cual el Memristor es capaz de alternar entre dos estados de resistencia mínima y máxima, además de poder recordar la dirección, y el tiempo en el que la señal de excitación le fue aplicada; estos fenómenos solo son apreciables en baja frecuencia ya que al aumentarla comienza a asemejar a una resistencia como en la *Figura* 1.4.

1.4 Conmutación Resistiva

Como se mostró en la clasificación de los Memristores, una de las características principales es la capacidad de alternar entre dos estados de resistencia máxima y mínima cuando se encuentra operando en sus límites; este fenómeno llamado, conmutación resistiva, es provocado por una combinación de efectos físicos y químicos. Como se presenta en la ref. [8], dicha conmutación puede ser clasificada en dos grupos:

- Unipolar (URS): donde el cambio de resistencia depende de la amplitud de voltaje aplicado.
- Bipolar (BRS): en donde la conmutación toma lugar cuando la polarización aplicada se invierte.

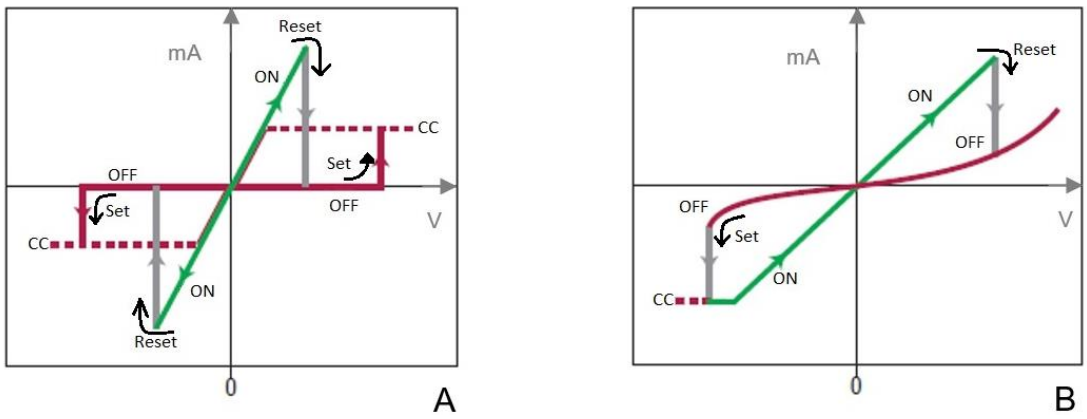

Figura 1.12 Conmutación A) Unipolar, B) Bipolar.

En el caso unipolar A) ref. [8], el voltaje de set (rojo) siempre es mayor que el voltaje de reset y la corriente está limitada por CC (compliance current) durante la operación; para invertir los estados la corriente de reset (verde) es mayor que la CC manteniendo un voltaje menor que el de set. Para la conmutación bipolar B) ref. [8], el estado set toma lugar con una polaridad de voltaje o corriente, mientras que el reset se observa cuando dicha polaridad es invertida, esta imagen podría ser atribuida a un memristor incremental.

Ciertamente, existen numerosos mecanismos involucrados en ambos tipos de conmutación, por lo que una manera fácil de comprenderlos es usando la clasificación propuesta en la ref. [9] (*Figura* 1.13):

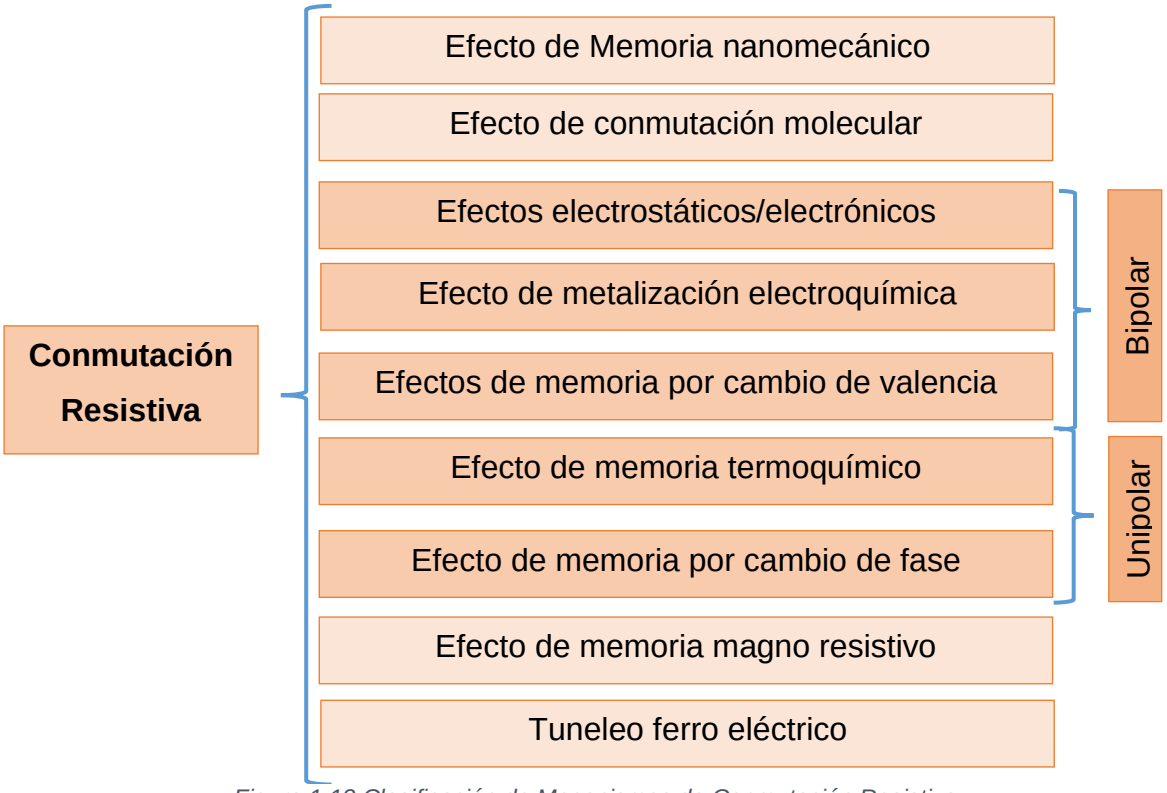

Figura 1.13 Clasificación de Mecanismos de Conmutación Resistiva.

Es importante conocer ambos tipos de conmutación, ya que como se menciona en la ref. [10], puede ser posible la coexistencia de ambos tipos de conmutación en estructuras Pt/TiO_2/Pt dadas ciertas condiciones.

Otra forma de clasificar Memristores según la ref. [11], es por a los mecanismos de conmutación, cambio de fenómenos o materiales de conmutación, adicional a la conmutación iónica.

Como se menciona en la ref. [11], los óxidos han sido estudiados alrededor de 50 años. En específico, los materiales con capacidad de conmutación basados en aniones son principalmente:

MgO, TiO_x, ZrO_x, HfO_x, VO_x, NbO_x, TaO_x, CrO_x, MoO_x, WO_x, MnO_x, FeO_x, CoO_x, NiO_x, CuO_x, ZnO_x, AlO_x, GaO_x, SiO_x, SiO_xN_y, GeO_x, SnO_2, BiO_x, SbO_x,

También, se ha demostrado la capacidad de conmutación en materiales nitruros como el AlN y teluros ZnTe.

1.5 El Memristor de Hewlett-Packard

Desde la década de los 90 las previsiones de HP según la ley de Moore indicaban una inminente confrontación con las limitantes físicas presentes en la continua capacidad de integración de dispositivos electrónicos; dada la mejora y duplicación de capacidad en un chip cada dos años surgió en Hewlett-Packard la necesidad de buscar soluciones y alternativas que retrasaran el inminente encuentro con los límites físicos.

Es así como en 1995 surge el grupo de trabajo liderado por R. Stanley Williams cuyo objetivo propuesto fue la “electrónica molecular”, dejando el camino abierto a un tema de su elección, el cual fue la ley de Moore.

Enfocándose en una arquitectura de computadora del tipo Teramac tomando una abstracción de barras cruzadas ref. [12], generando una matriz en donde cada intersección fuera un interruptor capaz de abrir o cerrar con un voltaje aplicado, su objetivo era poder utilizar esta estructura como memoria, en donde la densidad de integración sobrepasaría a las arquitecturas actuales.

La estructura concebida originalmente según la ref. [12] era un sándwich de dos electrodos de platino en los extremos, donde se oxidaba la superficie inferior

creando una capa de dióxido de platino altamente conductiva, sobre esta era depositada una monocapa especial de moléculas para conmutación y entre ambas partes una capa de titanio era colocada para unirlas.

Sin embargo, el pasar de experimentos frustrantes hasta un dispositivo que realmente cambiara la resistencia pasaron varios años, más aún no era posible modelar los mecanismos físicos que gobernaban dicho comportamiento, hasta que al revisar el artículo publicado por Chua ref. [2], notaron que su dispositivo tenía similitud con la descripción presentada.

Al realizar más pruebas en su dispositivo y tratar de comprender como es que este fenómeno se presentaba, llegaron a la conclusión de que las responsables eran las vacancias de oxígeno presentes en el titanio, al reproducir estos resultados concluyen que se trataba del Memristor.

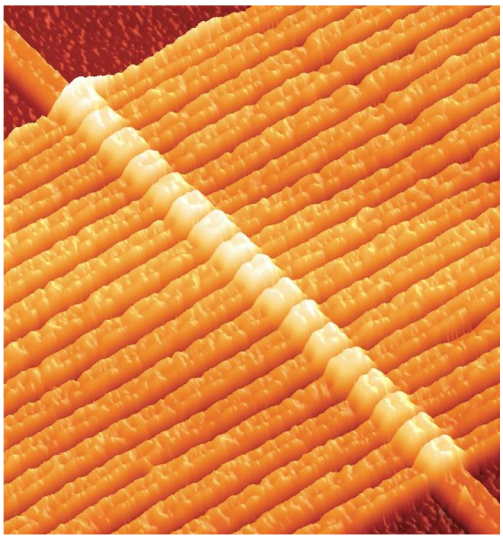

Figura 1.14 Arreglo de barras cruzadas para unos Memristores visto a través de un microscopio de efecto túnel por R. STANLY WILLIAMS/HP Labs. Ref. [12].

Es así que en el año 2008 el interés por los Memristores se ve nuevamente avivado gracias a la publicación de la ref. [13], en donde los investigadores Dimitri B. Strukov, Gregory S. Snider, Duncan R. Stewart y R. Stanley Williams presentaron un modelo de un dispositivo físico de dos terminales, construido a partir de una película de óxido de titanio de espesor nanométrico, la cual se encontraba con

vacancias de oxígeno (TiO_{2-x}), unida a otra película, pero sin dichas vacancias (TiO_2) como se muestra en la Figura 1.15.

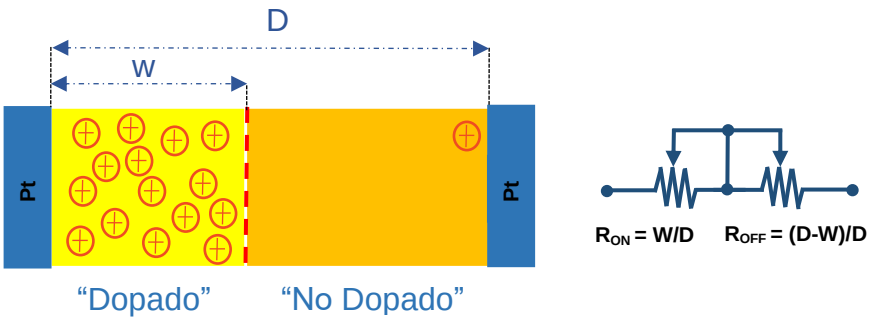

Figura 1.15 Memristor de HP.

Nota: En este caso al referirse a una región dopada, se hace alusión a un tipo de comportamiento de una estructura de óxido de titanio con vacancias de oxígeno (TiO_{2-x}), mientras que el termino no dopado, se refiere a un oxido de titanio sin vacancias de oxígeno (TiO_2), según la ref. [12], donde no se realizó ningún proceso para introducir agentes exógenos.

Este modelo considera dos resistencias en serie, una asociada a la región dopada (R_{ON}), y otra a la no dopada (R_{OFF}), la frontera entre ellas es capaz de moverse, extendiendo la región dopada o reduciéndola si se aplica una tensión positiva o negativa, respectivamente.

Si un voltaje positivo es aplicado en la región dopada, este repele la carga presente, lo que se refleja en el aumento de esta región, por lo que la conductancia a través del dispositivo aumenta (*Figura* 1.16A); en cambio, si se aplica un voltaje negativo éste atraerá las cargas, reduciendo así el área dopada y con ello disminuyendo la conductividad (*Figura* 1.16B)

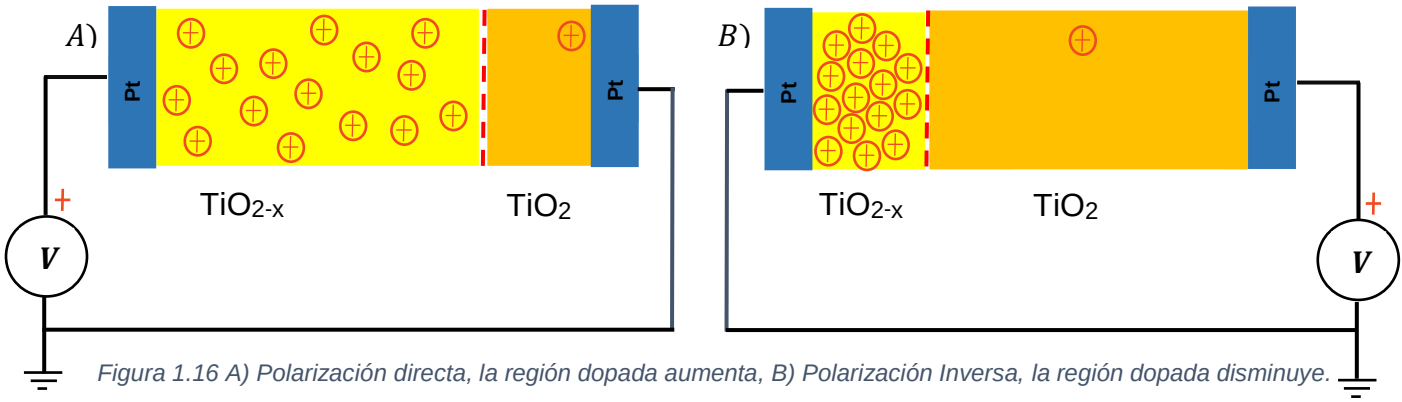

Figura 1.16 A) Polarización directa, la región dopada aumenta, B) Polarización Inversa, la región dopada disminuye.

Así que cuanto más tiempo se aplique un voltaje positivo más disminuirá su resistencia hasta llegar a su valor mínimo; por el contrario, al aplicar un voltaje negativo la resistencia incrementará, finalmente la característica principal por la que fue llamado Memristor, como se indica en las referencias [12], [13] y [14], esto debido a que, al retirar la señal de excitación las cargas detienen su movimiento y así la resistencia interna total deja de cambiar y se mantiene.

En la *Figura* 1.17 se muestran los resultados experimentales obtenidos por el equipo de Hewlett-Packard, en donde se aprecia el cruce por cero, así como una buena característica de la histéresis.

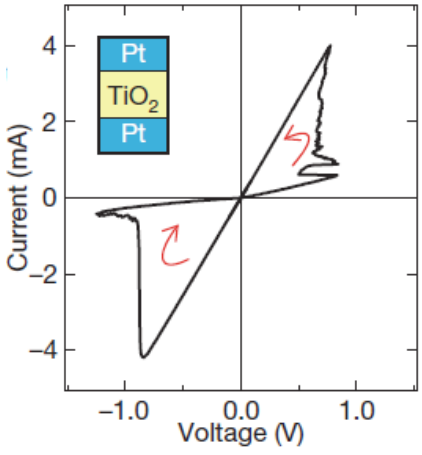

Figura 1.17 Resultados experimentales del dispositivo construido por "HP".

En la *Figura* 1.18 se aprecia la estructura física del memristor desarrollado por los laboratorios Hewlett-Packard, en donde se observa la región dopada (TiO_{2-x} parte superior), y la no dopada (TiO_2 parte inferior), así como los contactos de platino.

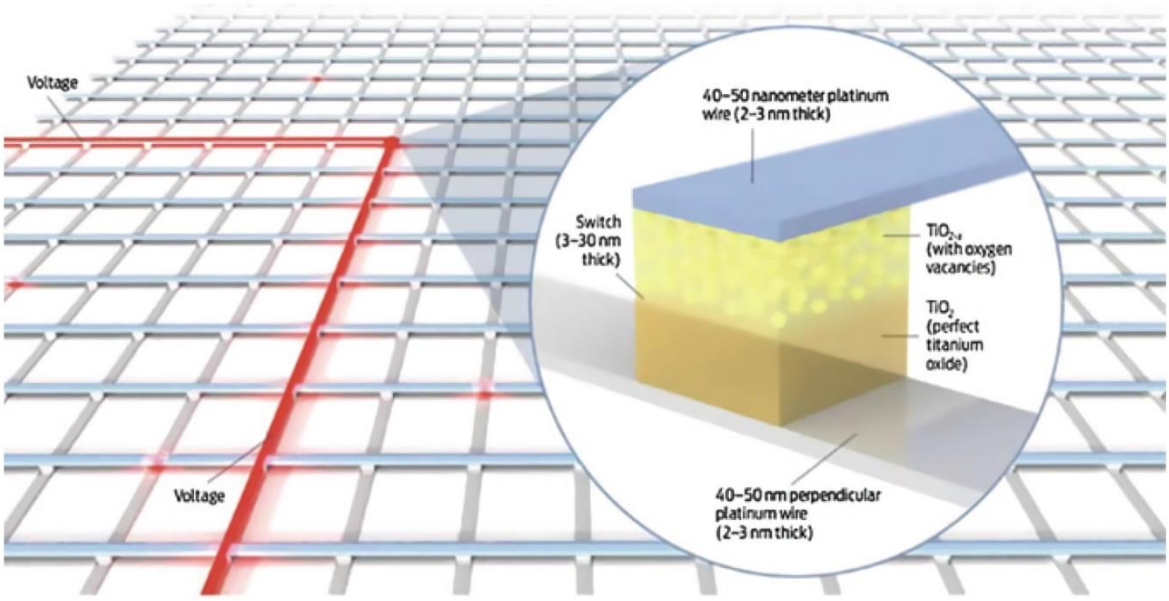

Figura 1.18 Estructura física del Memristor construido por HP presentada en la ref. [8].

Para explicar el comportamiento del dispositivo que se había construido, los investigadores de Hewlett-Packard describieron un modelo simplificado, en donde se toma un caso simple de conducción óhmica y arrastre lineal de las impurezas a través de un campo eléctrico.

El comportamiento de este dispositivo fue descrito por los investigadores de Hewlett-Packard en el artículo [6] con el siguiente sistema de ecuaciones:

$$v(t) = \left(R_{ON} \frac{w(t)}{D} + R_{OFF} \left(1 - \frac{w(t)}{D} \right) \right) i(t) \qquad (1.52)$$

$$\frac{dw(t)}{dt} = u_v \frac{R_{ON}}{D} i(t) \qquad (1.53)$$

En donde **D** es la longitud total del dispositivo, u_v es la movilidad promedio, y **w** es el tamaño de la región dopada. Resolviendo la ecuación (1.53), es posible obtener w (t) como se muestra en la ecuación (1.54).

$$w(t) = u_v \frac{R_{ON}}{D} q(t) \qquad (1.54)$$

Entonces, al sustituir la ecuación (1.54) en (1.52) y despejando para obtener la resistencia (memristancia en función de la carga) tenemos:

$$M(q) = R_{OFF}\left(1 - \frac{u_v R_{ON}}{D^2}\, q(t)\right) \qquad (1.55)$$

Este modelo asemejaba con éxito los experimentos realizados; sin embargo, no contaba con un umbral y no consideraba otros fenómenos exógenos a la carga como responsable del cambio en w, por lo que otros modelos surgieron al tratar de subsanar estos aspectos y así acercarse más al comportamiento físico del Memristor.

A continuación, en la *Tabla* 1.3 se muestra algunos de los modelos propuestos que asemejan con éxito el memristor, pero en los cuales su complejidad aumenta al acercarse más al comportamiento físico, por lo que algunos no son viables para ser simulados o implementados en una escala mayor como es requerido para los objetivos planteados previamente.

Tabla 1.3 Principales modelos del memristor, tomados de la ref. [1].

Modelo	Relación Corriente- Voltaje	Derivada de la variable de estado
Arrastre lineal	$v(t) = \left(R_{ON}\frac{x(t)}{D} + R_{OFF}\left(1 - \frac{x(t)}{D}\right)\right) i(t)$	$\frac{dw(t)}{dt} = \frac{u_v R_{on}}{D} i(t)$
Arrastre no lineal	$i(t) = w^n(t)\beta \sinh(\alpha v(t)) + \chi\left[\exp(\gamma v(t)) - 1\right]$	$\frac{dw(t)}{dt} = a v^m(t) f(w)$
Barrera de túnel Simmons	$i(t) = \tilde{A}(x, v_g)\phi_1(v_g, x)\, \mathrm{x} \exp\left(-B(x, v_g)\cdot \phi_1^{0.5}(x, v_g)\right)$ $- \tilde{A}(x, v_g)(\phi_1(v_g, x)$ $+ e\lvert v_g\rvert)\, \mathrm{x} \exp\left(B(v_g, x)(\phi_1(v_g, x) + e v_g)^{0.5}\right)$ $v_g = v - i(t)R_s$	$\frac{dx(t)}{dt}$ $= \begin{cases} C_{off} \sinh\left(\frac{i}{i_{off}}\right) exp\left[-exp\left(\frac{x - a_{off}}{w_c} - \frac{\lvert i\rvert}{b}\right) - \frac{x}{w_c}\right] & i > 0 \\ C_{on} \sinh\left(\frac{i}{i_{on}}\right) exp\left[-exp\left(\frac{x - a_{on}}{w_c} - \frac{\lvert i\rvert}{b}\right) - \frac{x}{w_c}\right] & i < 0 \end{cases}$
TEAM	$v(t) = \left[R_{on} + \frac{R_{OFF} + R_{OFF}}{x_{Off} - x_{on}}(x - x_{on})\right] \cdot i(t)$ $v(t) = R_{ON} \cdot \ exp\left(\frac{\lambda}{x_{Off} - x_{on}}(x - x_{on})\right) \cdot i(t)$	$\frac{dx(t)}{dt} = \begin{cases} K_{off}\left(\frac{i(t)}{i_{off}} - 1\right)^{\alpha_{off}} \cdot f_{off}(x) & 0 < i_{off} < i \\ 0 & i_{on} < i < i_{off} \\ K_{on}\left(\frac{i(t)}{i_{on}} - 1\right)^{\alpha_{on}} \cdot f_{on}(x) & i < i_{on} < 0 \end{cases}$

Es entonces que surge la necesidad de obtener un modelo simplificado, para realizar simulación de redes complejas de Memristores. Existiendo prometedores modelos que no comprometen la semejanza del comportamiento físico con junto con la posibilidad de simulación aceptable.

1.6 Aplicaciones

Desde la publicación del artículo de [13], las aplicaciones propuestas han continuado en aumento; como se menciona en las referencias [15] y [16], este nuevo dispositivo físico expande las posibilidades de diseño en el campo de la electrónica sin llegar a ser el sustituto absoluto del transistor, de tal manera se espera su pronta incursión en el mercado.

Se considera que el memristor tiene potencial en el área de memorias de acceso aleatorias no volátiles (NVRAM), ya que no requiere un consumo de energía continua, además de su alta escala de integración en dimensiones reducidas. Sin embargo, no es la única área en donde destaca; como se menciona en la ref. [12] los Memristores junto con los transistores permitirían emular las funciones de un cerebro, en lugar de simular solo redes de neuronas y sinapsis. A continuación, se mencionan algunas de las aplicaciones más prometedoras propuestas para los memristores.

1.6.1 Memoria Asociativa

Este tipo de memorias apoya aplicaciones como el reconocimiento de patrones, por lo que se trata de una parte fundamental de la computación cognitiva, usado como medio de almacenamiento.

En este caso, los Memristores son usados como memorias, las cuales mapean un patrón de entrada validando la similitud con uno almacenado previamente. Como se visualiza en la *Figura* 1.19 son colocados dos Memristores (X, Y) en serie los cuales pueden ser programados para almacenar un uno o cero lógicos.

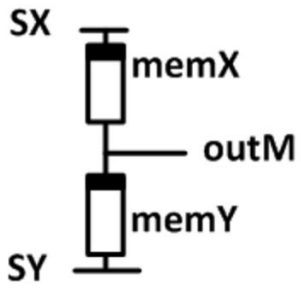

Figura 1.19 Celda básica de una memoria asociativa (tomada de la ref. [11]).

Esta celda opera en dos estados (0,1), los cuales son programados modificando la resistencia interna de cada memristor como en la *Figura* 1.20. Teniendo en cuenta que Roff= alta resistencia y Ron=baja resistencia, es posible almacenar un cero lógico sí MemX=Roff, MemY=Ron, y viceversa para el uno lógico.

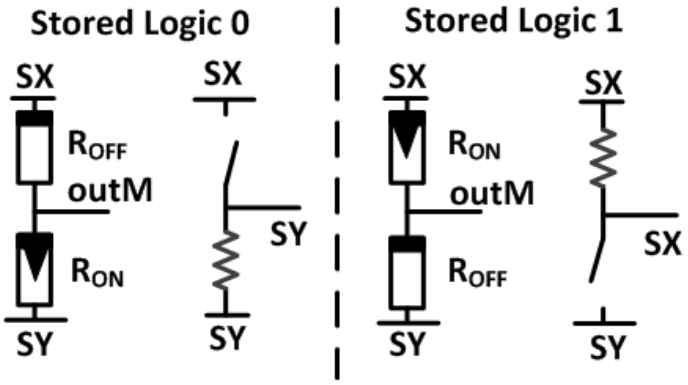

Figura 1.20 Estados de operación de la celda básica de memoria asociativa (tomada de la ref. [11]).

Estos estados pueden ser comparados con 1 lógico si se coloca SY a un voltaje positivo y SX a gnd, así también con un 0 lógico colocando SY a gnd y SX a voltaje positivo como en la *Tabla* 1.4 proveniente de la ref. [17], en dónde al buscar un valor lógico (0,1) si coincide con el dato almacenado a la salida se verá reflejado un 0, en caso contrario (no coincidir) será 1.

	MemX	MemY	SX	SY	outM
0 lógico almacenado	0	1	Ingresando1 lógico		1
			0(gnd)	1(VDD)	**No coincide**
	0	1	Ingresando 0 lógico		0
			1(VDD)	0(gnd)	**Coincide**
1 lógico almacenado	1	0	Ingresando1 lógico		0
			0(gnd)	1(VDD)	**Coincide**
	1	0	Ingresando 0 lógico		1
			1(VDD)	0(gnd)	**No coincide**

Tabla 1.4 Memoria asociativa de la celda básica (Figura 9).

Este uso toma al memristor de los extremos de su resistencia interna para el propósito, pero también puede ser usado de manera analógica, usando valores intermedios de resistencia como una carga eléctrica dinámicamente ajustable.

1.6.2 Carga Eléctrica

Debido a que, en muchos circuitos de alta frecuencia como amplificadores o filtros, las resistencias deben programarse para adaptarse a situaciones particulares o compensar variaciones, los Memristores pueden ser utilizados en dichos circuitos, gracias a su capacidad de variar la resistencia interna.

En la *Figura* 1.21, se muestra un amplificador diferencial en donde es posible configurar la resistencia del memristor en baja frecuencia, separando así la señal de alta frecuencia de los pulsos de programación, lo que permite el ajuste de la carga mientras se encuentra en ejecución como se menciona en la ref. [18], esto gracias a que el memristor solo presenta la característica de histéresis en baja frecuencia.

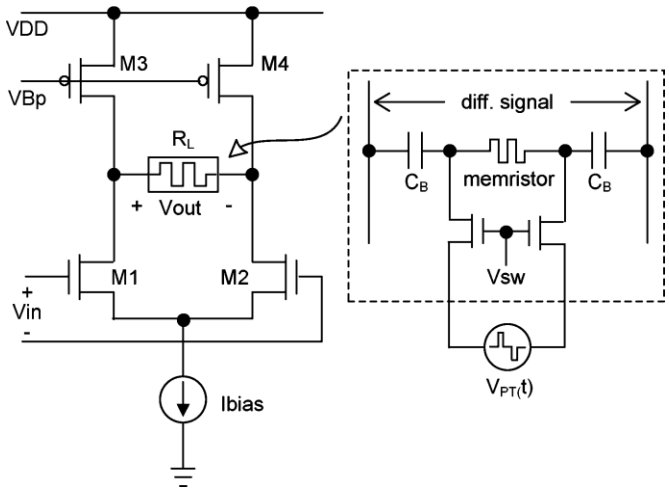

Figura 1.21 Amplificador diferencial programable, ref. [12].

1.6.3 Procesamiento de Imágenes

Utilizado una red de Memristores es posible reconocer los cambios significativos presentes en la imagen lo que resulta en la detección de bordes, como en la *Figura* 1.22.

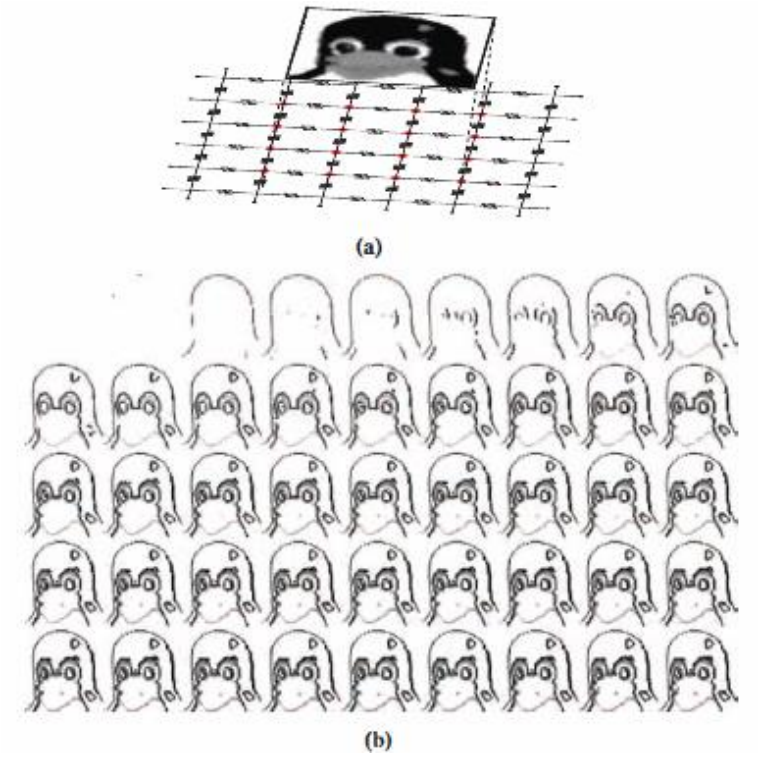

Figura 1.22 a) Red de memristores, b) Imagen obtenida (tomada de la ref. [19]).

En este caso, cada nodo de la red está conectado a un voltaje proporcional a un valor de la imagen digital, lo que resulta en la progresiva identificación de bordes debido a la distinta distribución de corriente a través de los Memristores.

1.6.4 Osciladores Caóticos

En un oscilador caótico la trayectoria de las oscilaciones depende de las condiciones iniciales del sistema, Chua desarrolló en la década de los 80 el llamado circuito de Chua, que presentaba comportamiento caótico, el cual se conforma por tres partes, la reactancia, un elemento no lineal, y uno activo.

La parte reactiva se conforma por lo menos de 3 elementos que almacenan energía, la cual aisladamente proporciona una oscilación amortiguada. El elemento activo es usado para evitar la amortiguación, a través de una resistencia negativa, y el elemento no lineal es el encargado de conmutar la resistencia negativa, obteniendo así los puntos de equilibrio del sistema. Debido a que el memristor es un elemento no lineal, se propone como remplazo del diodo Chua en el circuito original.

En la *Figura* 1.23 se muestra un ejemplo de un oscilador caótico no autónomo, donde el comportamiento no lineal es sustituido por el memristor; como en la ref. [20], este tipo de osciladores pueden ser utilizados en detección de señales débiles periódicas, lo cual es utilizado en hallar fallas en maquinarias de manera temprana por medio del ruido que generan.

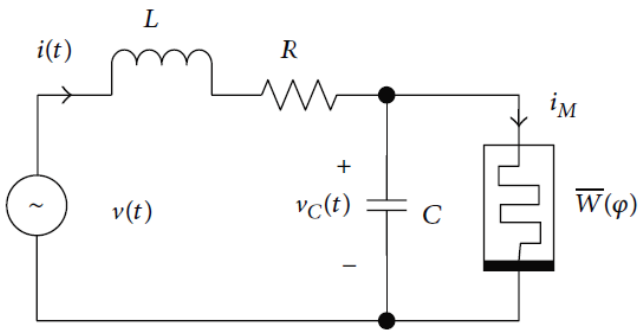

Figura 1.23 Oscilador caótico no autónomo, según la ref. [21].

Como se mostró, el memristor puede utilizarse de dos maneras principales, en arreglos de redes, o de manera discreta, así como en un uso digital o análogo, por lo que las aplicaciones más relevantes fueron resumidas en la ref. [16], como se muestra en la *Figura* 1.24.

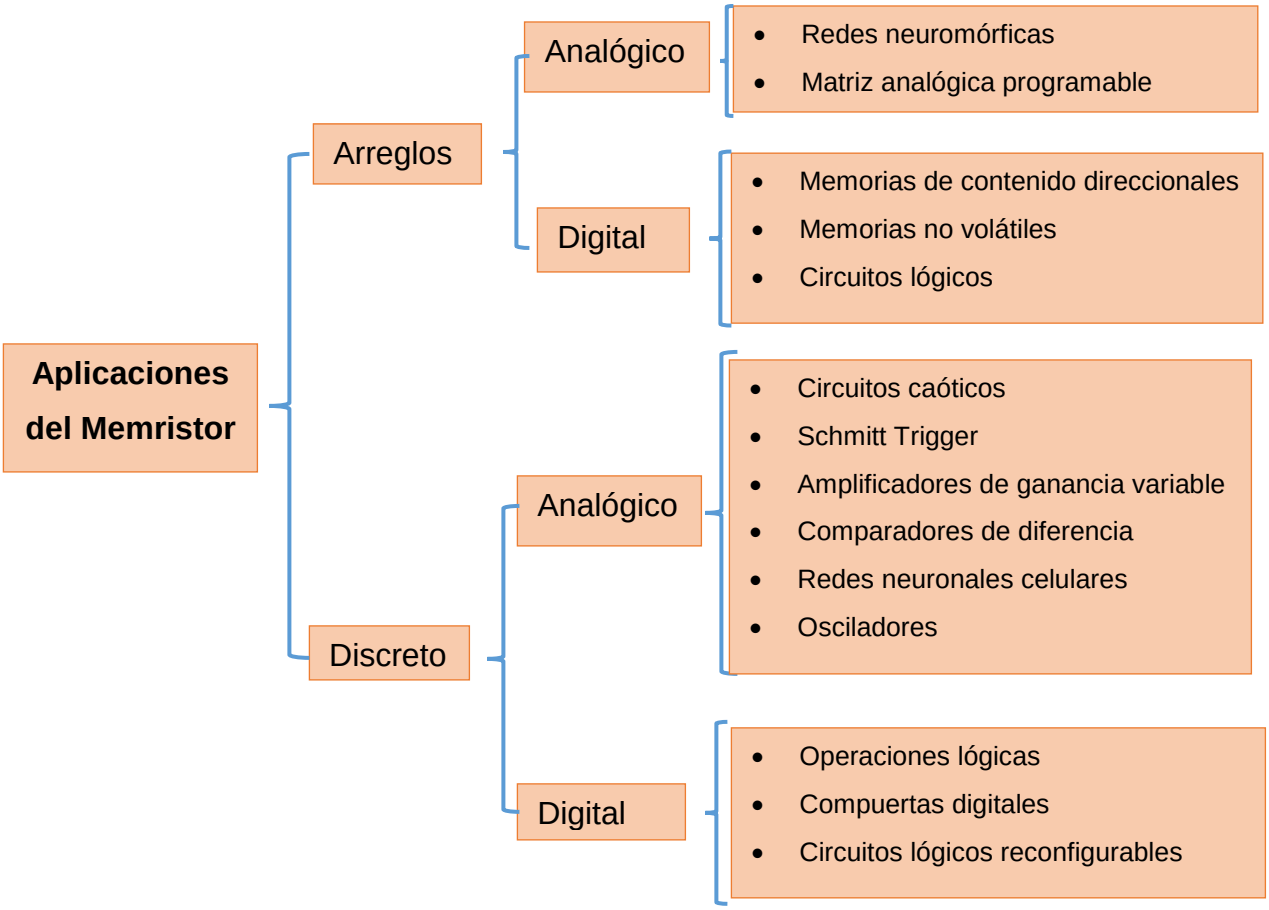

Figura 1.24 Principales aplicaciones propuestas para el memristor.

1.7 Resumen

En este capítulo, se presentó una recopilación de la historia general del memristor, abordando desde su propuesta y primeros modelos, hasta el modelo físico publicado por “HP”; se presentó, el modelo del memristor ideal, memristor genérico ideal, memristor genérico, y memristor extendido. Además, se mencionaron algunos ejemplos de sistemas que son considerados memristores.

Dadas sus características y algunas de las propuestas de uso más comunes, como las redes memristivas para almacenar datos o procesar imágenes, se comprueba la versatilidad del memristor para distintas aplicaciones.

Esto en conjunto establece las bases para comprender el funcionamiento del memristor en las implementaciones que serán realizadas posteriormente.

1.8 Referencias

[1] L. O. Chua, "*Everything You Wish to Know About Memristors but Are Afraid to Ask*", Radioengineering, Vol. 24, pp. 319-368 ,2015.

[2] L. O. Chua, *"Memristor—The Missing Circuit Element"*, IEEE Trans., circuit Theory, vol. CT-18, No. 5, pp. 507-519, 1971.

[3] L. O. Chua, and S. M. Kang, *"Memristive Devices and Systems",* Proc. IEEE, vol. 64, pp. 209-223, 1976.

[4] A. G. Radwan and M. E. Fouda, "*On the Mathematical Modeling of Memristor, Memcapacitor, and Meminductor*", Springer, pp. 9, 2015.

[5] D. Biolek and Z. Biolek, "*About Fingerprints of Chua's Memristors*" IEEE Circuits and Systems Magazine, vol. 18, no. 2, pp. 35-47, 2018.

[6] I. Vourkas and G. C. Sirakoulis, "*Memristor- Based Nanoelectronic Computing Circuits and Architectures",* Springer, pp. 11, 2016.

[7] S. P. Adhikari, M. P. Sah, H. Kim and L. O. Chua, *"Three Fingerprints of Memristor"* in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 60, no. 11, pp. 3008-3021, 2013.

[8] M. A. Rainer Waser, "*Nanoionics Based Resistive Switching*", Nature Materials 6, pp. 833-840, 2007.

[9] M. A. Rainer Waster, R. Dittman, G. Staikov and K. Szot, *"Redox-Based Resistive Switching Memories – Nanoionic Mechanisms, Prospects, and Challenges",* InterScience, pp. 2632-2663, 2009.

[10] D. S. Jeong, H. Schroeder, and R. Waser, "*Coexistence of Bipolar and Unipolar Resistive Switching Behaviors in a Pt / TiO2 / Pt Stack",* Electrochemistry and Solid-State Letters, 10: pp. G51-G53, 2007.

[11] J. J. Yang, D. B. Strukov, and D. R. Stewart, "*Memristive Devices for Computing",* Nature Nanotechnology, pp. 13-24, 2007.

[12] R. S. Williams, "*How We Found the Missing Memristor*" in IEEE Spectrum*,* vol. 45, no. 12, pp. 28-35, 2008.

[13] D. B. Strukov, G. S. Snider, D. R. Stewart and R. S. Williams, *"The Missing Memristor Found"*, Nature 453, pp. 80-83, 2008.

[14] R. Tetzlaff, "*Memristors and Memristive Systems"*, Springer, pp. 14, 2014.

[15] P. Mazumber, S.M. Kang and R. Waster, *"Memristors: Devices, Models and Applications",* Proc. IEEE, Vol 100 no. 6, pp. 1911-1919, 2012.

[16] R. Marani, G. Gelao and A. G. Perri "*A Review on Memristor Applications"*, Int. J. Adv. Eng. Techonol. 8, pp. 294, 2015.

[17] Y. Yang, J. Mathew, and D. K. Pradhan, *"Matching in Memristor Based Auto-Associative Memory with Application to Pattern Recognition"*, Proceedings of 12th International Conference on Signal Processing (ICSP), pp. 1463-1468, 2014.

[18] S. Shin, K. Kim, and S. M. Kang, *"Memristor Applications for Programable Analog ICs"*, Proceedings of the IEEE, Vol 10, No. 2, pp. 266-274, 2011.

[19] T. Prodromakis and C. Toumazou, "*A Review on Memristive Devices and Applications*,", 17th IEEE International Conference on Electronics, Circuits and Systems, Athens, 2010, pp. 934-937.

[20] F. Wang, H. Xing, S. Duan, and H. Yu, "*Study on Chaos-Based Weak Signal Detection Method with Duffing Oscillator*". Math. Probl. Eng., pp. 21-26, 2015.

[21] G. Wang, M. Cui, B. Cai, X. Wang, and T. Hu, "*A Chaotic Oscillator Based on HP Memristor Model*". Math. Probl. *Eng., Vol. 2015, Article ID 561901, pp. 12 ,2015.*

Capítulo 2 Redes Neuronales Artificiales

2.1 Introducción

Con los avances tecnológicos en el área de la electrónica, hoy es posible tener capacidad de cómputo en sistemas tanto digitales como analógicos, los cuales no fueron experimentados hace una década. En particular, se tienen sistemas de Redes Neuronales Artificiales (Artificial Neural Networks, ANNs), cuyos modelos pueden ser ejecutados en software o en hardware de tipo neuromórfico. Asimismo, los modelos biológicamente plausibles representados por Redes Neuronales Pulsadas (Spiking Neural Networks, SNNs) son un medio para emular cercanamente las funciones encontradas en los sistemas nerviosos de seres vivos, en especial las de las redes de la corteza cerebral de los seres humanos.

En el curso del desarrollo de modelos neuronales artificiales, se destaca el algoritmo de entrenamiento de arquitecturas multicapa, conocido como Retro-Propagación de Errores (Errors Back-Propagation, EBP). En tanto que, para las SNNs, el entrenamiento se lleva a cabo con la regla: Plasticidad Dependiente de Pulsaciones en el Tiempo (Spike-Timing Dependent Plasticiy, STDP).

Para los sistemas SNNs, el procesamiento y transmisión de información es a través de señales analógicas pulsadas llamadas Potenciales de Acción, los cuales son modulados por las sinapsis entre neuronas. Esta modulación está basada en el valor de conductancia eléctrica de la sinapsis, permitiendo en mayor o menor grado la contribución de un pulso de una neurona a otra vecina.

En este contexto, aparece el Memristor o Resistencia con Memoria, el cual es un elemento eléctrico pasivo que complementa las relaciones fundamentales en la teoría de sistemas eléctricos, definidas por el Voltaje, la Corriente, el Flujo Magnético y la Carga Eléctrica.

Es entonces, que ahora en lugar de simular redes neuronales artificiales, es posible emular redes neuronales pulsadas con el uso del memristor, ya que como se mencionara más adelante, este se propone como una analogía a la sinapsis electroquímica presente en las redes neuronales biológicas, ya que es capaz de cambiar su valor de resistencia dependiendo de la polarización que se le aplique, teniendo un efecto directo en la comunicación entre una neurona y otra.

Ahora, si bien es posible generar neuronas artificiales pulsadas a partir de modelos que asemejan con éxito el comportamiento físico, y con el conocido cuarto elemento fundamental, el memristor propuesto como análogo a la sinapsis, aún queda camino por recorrer, ya que la densidad de integración lograda hasta el momento al emular redes neuronales pulsadas es inferior a la existente en un sistema biológico.

En este capítulo se abordarán temas generales acerca de las redes neuronales artificiales (ANN's) así como las redes neuronales pulsadas (SNN´s), entendiendo como el memristor puede ser un equivalente de la sinapsis, hasta como es que el mecanismo STDP emerge de manera natural al colocar un memristor entre dos neuronas, todo ello para construir una red neuronal pulsada con Memristores usada en el reconocimiento de patrones.

2.2 ANNs Clásicas

Las redes Neuronales Artificiales (Artificial Neural Network), surgen como una propuesta de reproducir el comportamiento de su análogo biológico (funciones en la corteza cerebral), pero de una manera mucho más simplificada, ya que el funcionamiento del cerebro humano es altamente paralelo y con la capacidad de resolver problemas complejos. Entonces, se planteó la posibilidad de generar neuronas artificiales, así como un mecanismo que las interconecte, y a través de un entrenamiento, logre aprender, como lo hace un sistema biológico ref. [1].

En la *Figura* 2.1, se puede observar la estructura general de una neurona, así como las partes que la componen, además de su interconexión con otra neurona. Cabe resaltar que según la ref. [2], la cantidad estimada de neuronas presentes en un

cerebro humano es 10^{11}, teniendo una cantidad de conexiones aproximada en 10^{4} por cada neurona.

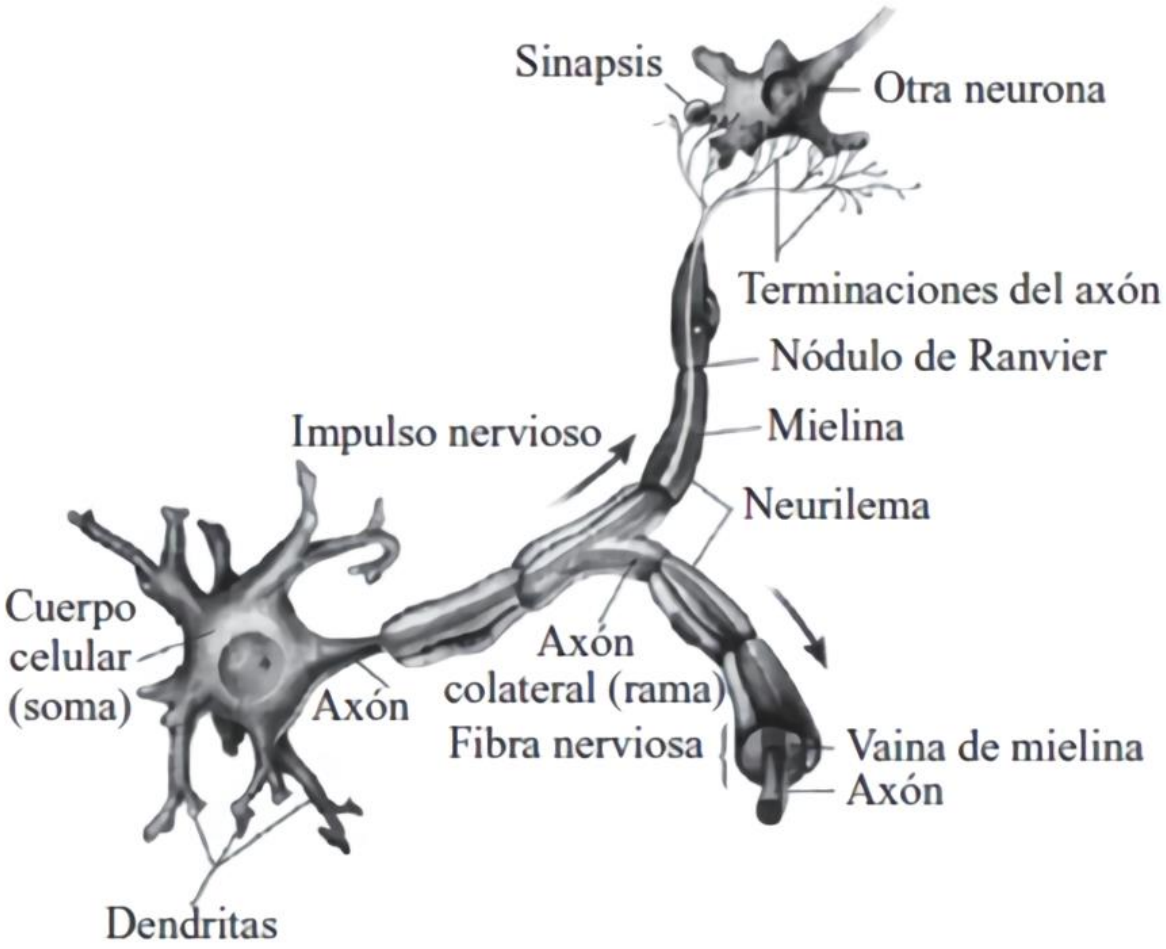

Figura 2.1 Estructura de una neurona presentada en la ref. [1].

La primera propuesta conocida como lógica umbral, tenía sus bases en funciones de activación, donde al sobrepasar un cierto umbral, se disparaba una señal, fue así que era posible simular neuronas básicas; posteriormente fue presentado un modelo conocido como perceptrón, este era capaz de emular ciertas compuertas lógicas, excepto la or exclusiva, es así que, junto con su intento de refutación, la investigación se vio minimizada. Posteriormente, el interés de las redes neuronales continuó creciendo hasta llegar a sistemas mucho más complejos, como las redes neuronales multi-capa, Figura 2.2.

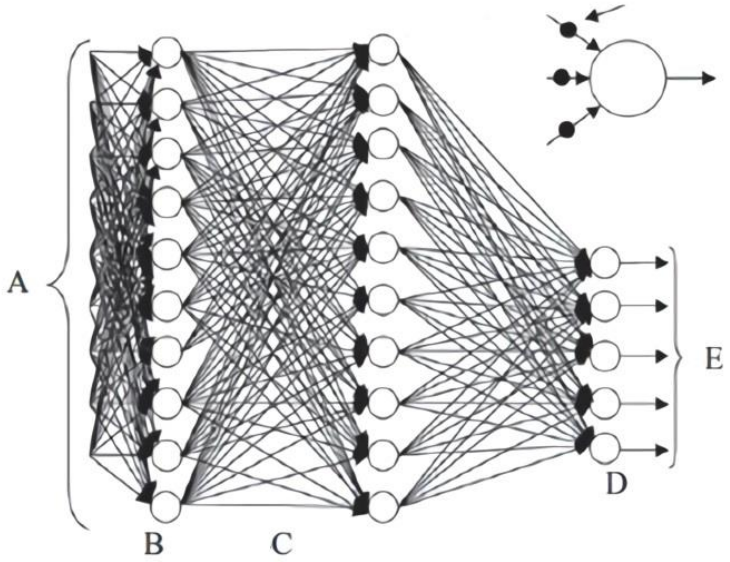

Figura 2.2 Estructura de una Red Neuronal Artificia multi-capa, presentada en la ref. [1].

En la *Figura* 2.2 se observa la estructura de una red neuronal multi-capa (10-10-5) con 10 neuronas entradas y 5 neuronas de salidas, además de 10 neuronas en la capa oculta 10, donde los pesos sinápticos encargados de modificar la información que viaja entre neuronas, están representados por líneas continuas.

Sin embargo, aunque este tipo de redes neuronales son mucho más complejas y con algoritmos de entrenamiento verificados para resolver problemas difíciles de programar, aún no asemejan por completo a su homologo biológico, el cual no trabaja con valores discretos, sino que lo hace a través de potenciales de acción, lo que dio paso al concepto de Redes Neuronales Pulsadas (SNN).

2.3 La Neurona

Se trata de un elemento biológico encargado de recibir, procesar y transmitir información a otras neuronas por medio de señales electroquímicas, dichas señales consisten en pulsos eléctricos cortos llamados potenciales de acción, según la ref. [2], tienen una amplitud aproximada de 100 mV, con una duración de 1 a 2 ms.

En la *Figura* 2.3 se observa la estructura simplificada de una neurona, en donde se coloca un electrodo para observar el pulso generado y como se propaga a través del axón hasta otras neuronas por medio de la sinapsis.

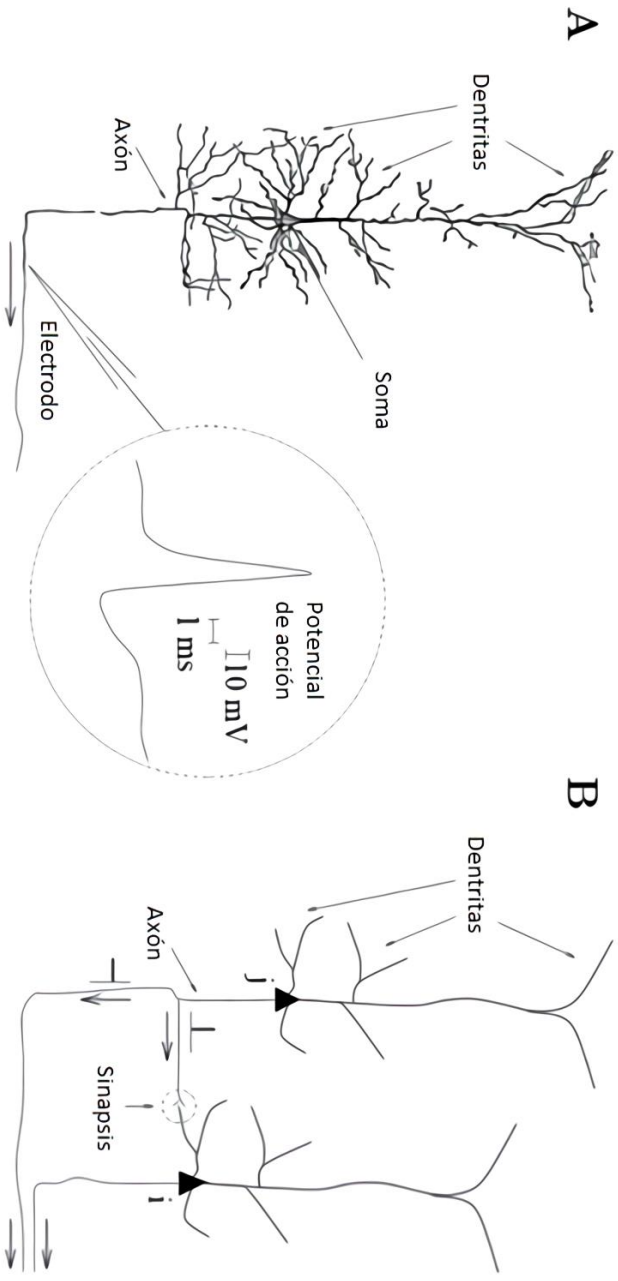

Figura 2.3 Neurona A) Vista general simplificada con un potencial de acción, B) transmisión de una señal desde una neurona pre sináptica j a una neurona post sináptica i, tomada de la ref. [3].

Cuando una serie de pulsos son emitidos por una neurona se le conoce como tren de pulsos (*Figura* 2.4), pero dado que la forma de los pulsos es similar, ésta no lleva información como tal, más bien lo que es relevante es la cantidad y separación de los pulsos generados.

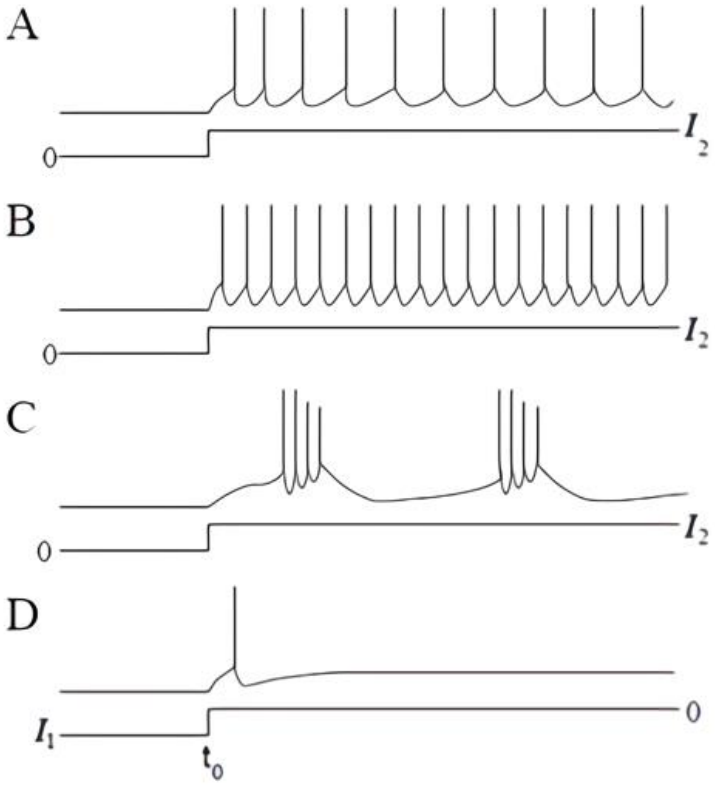

Figura 2.4 Distintos trenes de pulsos tomada de [3].

La separación entre pulsos generados de una neurona a pesar de no estar exactamente definida, cuenta una distancia mínima entre dos picos que es conocida como el periodo refractario, aproximadamente 0.5ms según la ref. [3], donde es imposible excitar la neurona para provocar un pulso inmediatamente después de otro, posteriormente sigue un periodo refractario relativo hasta la recuperación de la neurona, en donde es posible volver a generar un pulso.

Para que una neurona sea capaz de generar un pulso (respuesta), ésta debe recibir una señal externa que sobrepase un valor establecido como umbral, en la *Figura* 2.5 se puede observar un potencial de membrana (pulso) en acción.

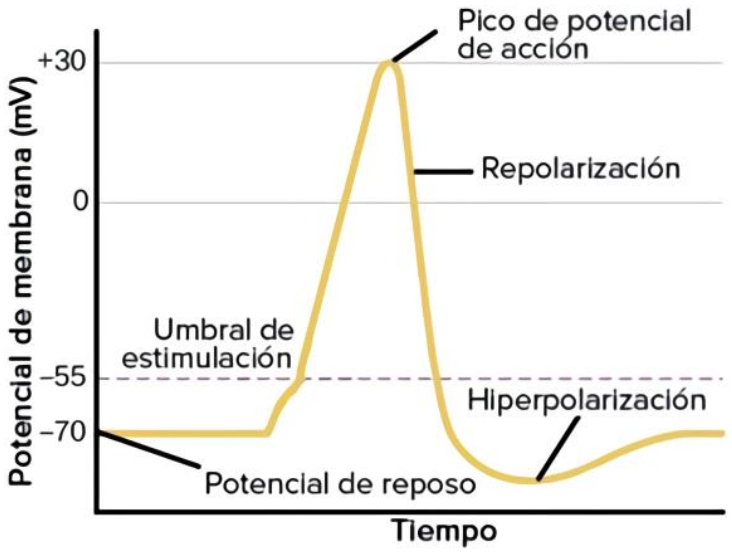

Figura 2.5 Ejemplo de un potencial de membrana generado por una neurona biológica, según la ref. [4].

Como ya se mencionó, al superar un determinado umbral, la neurona dispara, llega hasta un pico máximo reportado de 30 mV, y un tiempo después, decae a un valor inferior del reposo, conocido como el tiempo refractario, para después volver a su estado inicial -60 a -70 mV, siguiendo los pasos: despolarización, repolarización hiperpolarización, según la ref. [4]. Para explicar este comportamiento surgieron varios modelos; algunos de los más relevantes son:

2.3.1 Modelo Hodgkin-Huxley

Alan Lloyd Hodgking y Andrew Huxley presentaron en 1952 un modelo matemático capaz de describir la generación y propagación de un potencial de acción, a partir de las mediciones realizadas con un electrodo en el axón de un calamar gigante, encontraron que había tres diferentes tipos de corrientes iónicas, sodio (Na^+), potasio (k^+), y una corriente de pérdida principalmente compuesta por iones de cloro (Cl^-).

Para entender el modelo propuesto se muestra la Figura 2.6 tomada de la ref. [4] en donde se observa como existe una membrana semipermeable que divide el interior de la célula del fluido extracelular actuando como un capacitor, si una corriente es inyectada puede agregar más carga en el capacitor, o filtrarse a través de los canales de la membrana, por ejemplo si hay canales abiertos puede que los iones de sodio K^+ salgan, o que los iones de cloro Cl^- entren causando una hiperpolarización, por el contrario si entran iones de sodio Na^+ esto causa una despolarización; así, debido al transporte activo de iones a través de la membrana celular, la concentración interna de iones difiere de la extracelular.

También se aprecia en la *Figura* 2.6, como en el interior de la célula existen iones de potasio, y en el exterior iones de sodio, con unos canales que pueden estar abiertos o cerrados, visto desde este modo se puede entender por qué el potencial en reposo (Potencial de Nernst) es negativo.

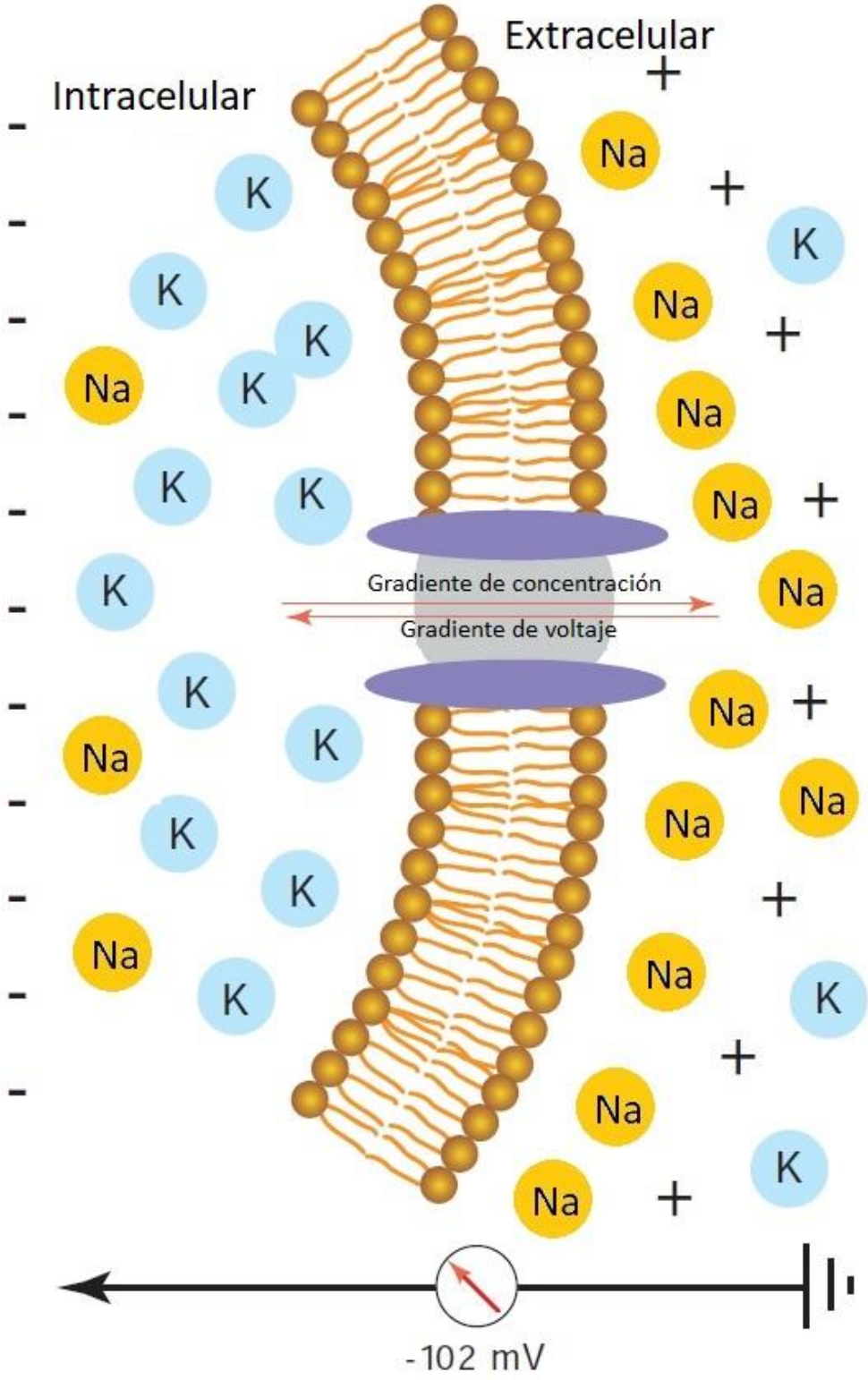

Figura 2.6 Potencial de membrana en reposo tomado de la ref. [4], donde la concentración de K^+ hace que los iones fluyan hacia fuera, sin embargo el potencial de membrana negativo los atrae de nuevo al interior de la célula; estos dos mecanismo se equilibran entre sí para formar el potencial de equilibrio.

La diferencia en la concentración de iones genera el llamado Potencial de equilibrio de Nernst, representado en la *Figura* 2.7 por unas baterías.

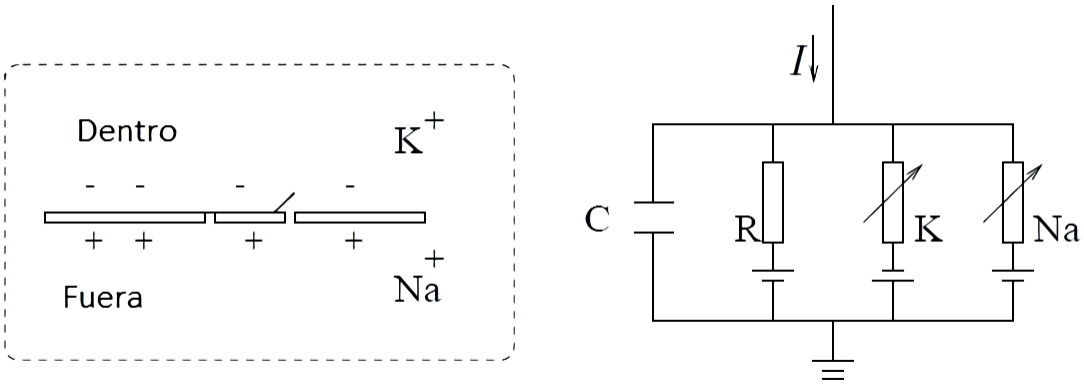

Figura 2.7 Diagrama del modelo Hodgking-Huxley tomado de [3].

Por lo que la corriente total inyectada en una neurona se puede dividir entre una corriente que carga el capacitor y la suma de las corrientes iónicas Na^+, K^+ y R, en donde R es la resistencia del canal de fuga sin especificar.

$$I(t) = I_C(t) + \sum_k I_K(t) \qquad (2.1)$$

Visto desde otra manera, la corriente en el capacitor se puede expresar como:

$$C\frac{du}{dt} = -\sum_k I_K(t) + I(t) \qquad (2.2)$$

Donde u representa el voltaje en la membrana, así como $\sum_k I_K$ es la suma de las corrientes iónicas que fluyen a través de la membrana, estas corrientes que pasan por los canales pueden caracterizarse por medio de su resistencia o conductancia. Es así como el canal de fuga está descrito por una tensión independiente de la conductancia $g_L = \frac{1}{R}$, mientras que en los otros canales la conductancia es dependiente del voltaje y del tiempo.

Como se menciona en la ref. [3] si todos los canales se encuentran abiertos, la transmisión de la corriente es máxima al tener una mayor conductancia g_{Na} o g_K, pero normalmente algunos canales están bloqueados, así que la probabilidad de

que un canal esté abierto es descrita con las variables m, n, h, en donde la combinación de m, h controla los canales de sodio (Na) y los canales de potasio (K) son controlados por n.

Es así que el modelo propuesto por Hodgking y Huxley está determinado por un conjunto de ecuaciones diferenciales, en donde la ecuación (2.3) modela la variación de potencial de membrana; el resto (ecuaciones (2.4), (2.5), (2.6)) son para la conductancia en los canales de sodio y potasio.

$$\sum_k I_K = g_{Na} m^3 h(u - E_{Na}) + g_K n^4 (u - E_K) + g_L (u - E_L) \qquad (2.3)$$

$$\frac{dm}{dt} = \alpha_m(u)(1 - m) - \beta_m(u)m \qquad (2.4)$$

$$\frac{dn}{dt} = \alpha_n(u)(1 - n) - \beta_n(u)n \qquad (2.5)$$

$$\frac{dh}{dt} = \alpha_h(u)(1 - h) - \beta_h(u)h \qquad (2.6)$$

Donde los parámetros E_{Na}, E_K y E_L son los potenciales inversos, y las variables m, n, h evolucionan conforme a la definición de las ecuaciones diferenciales. Los valores empíricos originales reportados por Hodgking y Huxley son los de la *Tabla* 2.1 :

X	Ex	gx	X	$\alpha_x\left(\frac{u}{mV}\right)$	$\beta_x\left(\frac{u}{mV}\right)$
Na	$11\ \text{mV}$	$120\frac{mS}{cm^2}$	n	$\frac{(0.1 - 0.01u)}{exp(1 - 0.1u) - 1}$	$0.125\exp\left(\frac{-u}{80}\right)$
K	$-12\ \text{mV}$	$36\frac{mS}{cm^2}$	m	$\frac{(2.5 - 0.1u)}{exp(2.5 - 0.1u) - 1}$	$4\exp\left(\frac{-u}{18}\right)$
L	$10.6\ \text{mV}$	$0.3\frac{mS}{cm^2}$	h	$0.07\exp\left(\frac{-u}{20}\right)$	$\frac{1}{\exp(3 - 0.1u) + 1}$

Tabla 2.1 Parámetros originales de Hodgking-Huxley, la capacitancia de la membrana es C=1uF/cm2.

Estos valores fueron modificados posteriormente para tener los parámetros más exactos que se manejan hoy en día, es así que el valor estándar del potencial inverso del sodio como se menciona en la ref. [3] es: $\mathrm{E_{Na}} = 50\mathrm{mV}$, mientras el del potasio es: $\mathrm{E_K} = -77\mathrm{mV}$.

A pesar del buen comportamiento que tiene este modelo, no cuenta con un umbral como en el caso de modelos más recientes; también, al ser un modelo físicamente bio-realista y muy exacto, el simular un conjunto de neuronas haciendo uso de las ecuaciones diferenciales consume demasiados recursos computacionales, como para ser implementada en gran escala, por lo que modelos mucho más simples surgieron para intentar solucionar este problema.

2.3.2 Integración y Disparo

Modelo básico pero muy útil, ya que a diferencia del anterior este no simula las conductancias, sino que se basa en simular el potencial de membrana dependiente de los estímulos que han sido recibidos, cuando se supera un cierto umbral, un disparo o pulso es generado por la neurona, posteriormente esta pasa a un estado previo de subumbral esperando que nuevamente se sobrepase el límite establecido.

Este modelo consiste en un capacitor C en paralelo con una resistencia R, controlados por una corriente de entrada $\mathrm{I(t)}$, como en la *Figura* 2.8.

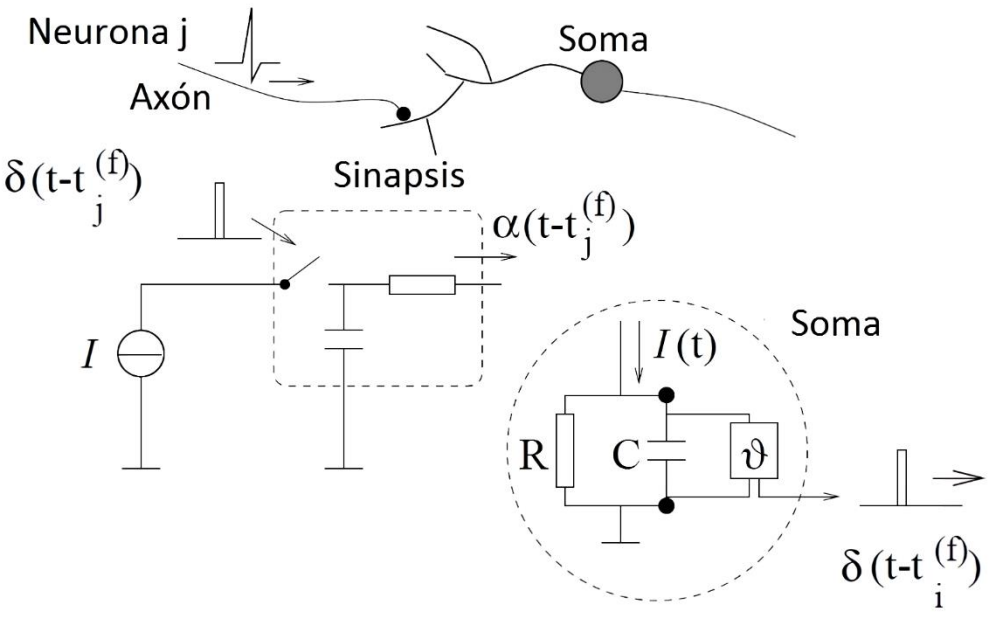

Figura 2.8 Modelo Integrate and Fire tomado según la ref. [3], donde se aprecia como el voltaje en el capacitor es comparado con un umbral ϑ, si el potencial de membrana u(t)= ϑ en un tiempo $t_i^{(f)}$, un pulso de salida $\delta(t-t_i^{(f)})$ es generado. También se observa como el pulso pre sináptico $\delta(t-t_j^{(f)})$ viaja a través de un filtro pasa bajas en la sinapsis, generando el pulso de corriente $\alpha(t-t_j^{(f)})$.

La corriente inyectada se divide entre una corriente en el capacitor y una en la resistencia, por lo que la corriente total se puede expresar como:

$$I(t) = I_R + I_C \qquad (2.7)$$

$$I(t) = \frac{u(t)}{R} + C\frac{du}{dt} \qquad (2.8)$$

Donde u es el voltaje a través de la resistencia, así como en el capacitor. Entonces si la ecuación (2.8) es multiplicada por R, e introduciendo una constante tiempo $\tau_m = RC$ se obtiene como en la ref. [3] la forma estándar:

$$\tau_m \frac{du}{dt} = -u(t) + RI(t) \qquad (2.9)$$

Ahora, si bien u representa el potencial de membrana y τ_m es la constante de tiempo de membrana en la neurona, este modelo no describe explícitamente los potenciales de acción, ya que los pulsos están caracterizados con el llamado tiempo de disparo; según la ref. [3] este tiempo de disparo $t^{(f)}$ está definido por un umbral:

$$t^{(f)}: \quad u\left(t^{(f)}\right) = \vartheta \tag{2.10}$$

Inmediatamente después de $t^{(f)}$, el potencial en la membrana es restablecido a un nuevo valor inferior al umbral $u_r < \vartheta$

$$\lim_{t \to t^{(f)};\, t > t^{(f)}} u(t) = u_r \tag{2.11}$$

Como se menciona en la ref. [3], para $t > t^{(f)}$ la dinámica sigue a la ecuación (2.9) hasta que el umbral sea superado nuevamente.

El modelo de integración y disparo también incorpora un período refractario absoluto, en donde si u alcanza el umbral en un tiempo $t = t^{(f)}$ se ve detenida la ecuación (2.9) durante un tiempo llamado periodo refractario, y se reinicia la integración en $t^{(f)} + \Delta^{asb}$ con una nueva condición inicial u_r.

Ahora, considerando una corriente total $I(t) = I_0$, así como un potencial de restablecimiento $u_r = 0$, y la condición inicial $u\left(t^{(1)}\right) = u_r = 0$, para encontrar el potencial de membrana, se integra la ecuación (2.9) y como en la ref. [3] se tiene:

$$u(t) = RI_0\left[1 - exp\left(-\frac{t - t^{(1)}}{\tau_m}\right)\right] \tag{2.12}$$

Donde en un valor de $RI < \vartheta$ no pueden producirse los pulsos, a menos que se supere el umbral.

2.3.3 Neurona de Izhikevich

Este modelo dado a conocer en 2003 por el Dr. Eugene M. Izhikevich, presenta varias ventajas respecto a otros modelos existentes, ya que logra asemejar con precisión el comportamiento físico de una neurona biológica, sin consumir demasiados recursos, fue propuesto a partir de una reducción del modelo biológico de Hodgkin–Huxley combinado con la eficiencia de las neuronas de integración y disparo.

Este modelo, representa una neurona en donde esta dispara si la tensión aplicada en ella supera un determinado nivel de umbral; dicho modelo consta de dos ecuaciones y se plantea como un modelo lo suficientemente exacto como para asemejar una neurona biológica sin llegar a ser prohibitiva su implementación a gran escala.

$$\frac{dv(t)}{dt} = 0.04v^2 + 5v + 140 - u + I \qquad (2.13)$$

$$\frac{du}{dt} = a(bv - u) \qquad (2.14)$$

En la ref. [5] es propuesto un nivel de umbral en 30 mV, y las variables u y v toman los valores siguientes:

$$if\ v \geq 30mV, entonces \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \qquad (2.15)$$

Donde v y u son variables, a, b, c, d son parámetros descritos en la ref. [5] como:

- a Describe una escala de tiempo para la variable de recuperación u, en valores pequeños es una recuperación lenta. Un valor típico es a = 0.02.
- b Es la sensibilidad de la variable de recuperación u a las fluctuaciones sub-umbrales del potencial de membrana v, a grandes valores (b>1) u y v se combinan con mayor fuerza, lo que resulta en posibles oscilaciones por

debajo del umbral y dinámicas de pulsaciones de umbral bajo, un valor típico es b = 0.2.

- c Describe el valor de retorno del potencial de membrana v después del potencial de acción causado por las conductancias de potasio rápidas y de umbral alto, típicamente asume el valor de c = 65mV.
- d Es el valor de retorno de la variable de recuperación u después del potencial de acción causado por las conductancias lentas de sodio y potasio de umbral alto, típicamente tiene un valor de d = 2.

Las diferentes combinaciones producidas al elegir los valores de cada uno de los parámetros, dan lugar a la generación de distintos patrones de disparo o trenes de pulsos generados que pueden ser asociados a las distintas neuronas que existen en el cerebro humano.

Ejemplos diferentes de patrones de disparo se muestran en la *Figura* 2.9 para las ecuaciones (2.13) y (2.14).

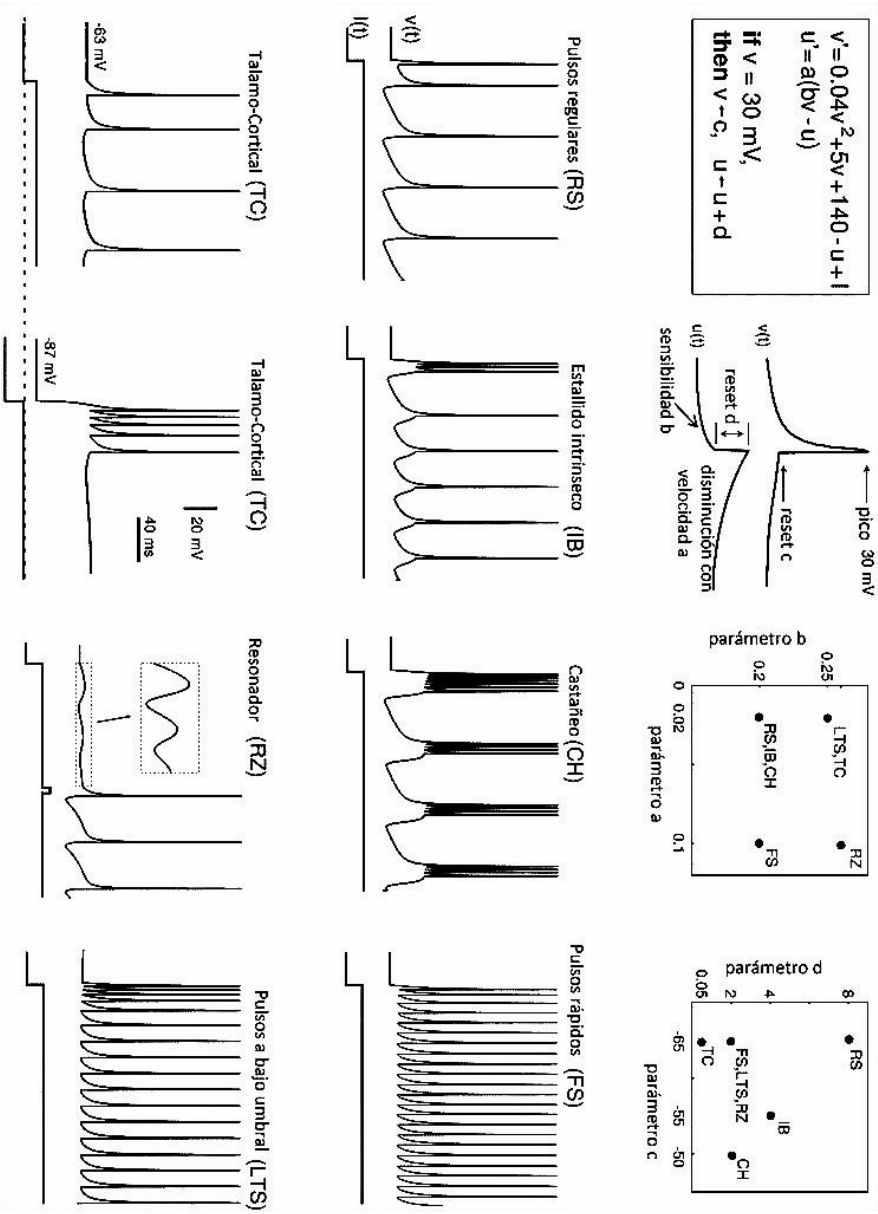

Figura 2.9 Diferentes patrones de pulsos generados dependiendo de los valores elegidos de a, b, c, d, tomada de la ref. [5], donde se aprecia cómo es posible asemejar varios tipos de neuronas.

2.3.4 Resumen de modelos

Una vez presentados los tres modelos de neuronas más relevantes (Hodgkin-Huxley, integración y disparo, neurona de Izhikevich), es necesario compararlos con otros modelos existentes para conocer las ventajas y desventajas que cada uno presenta, ya que no son los únicos existentes, además como se menciona en la ref.

[6] hay una variedad de modelos de neuronas dependiendo de la neurona a simular, así como de su exactitud respecto a su homologo biológico, algunos ejemplos son:

- Modelo de conductancia de Thomas Nowotny.
- Modelo Fitzhugh-Nagumo.
- Modelo de Hindmarsh-Rose.
- Mapa de Rulkov
- Resonate and Fire

Dependiendo del modelo, existe un compromiso entre lo biológicamente plausible que resulta su exactitud vs los recursos que son consumidos al llevar a la implementación computacional.

En la *Figura* 2.10 se aprecia como al elegir un modelo, dependiendo de la aplicación, en la mayoría existe un compromiso entre las características y la semejanza biológica. Si es fácil de implementar, los recursos consumidos son pocos, ayudando a mejorar la densidad de neuronas, pero con un desempeño pobre; por el contrario, un modelo muy exacto puede consumir tantos recursos que solo sería posible implementar algunas neuronas.

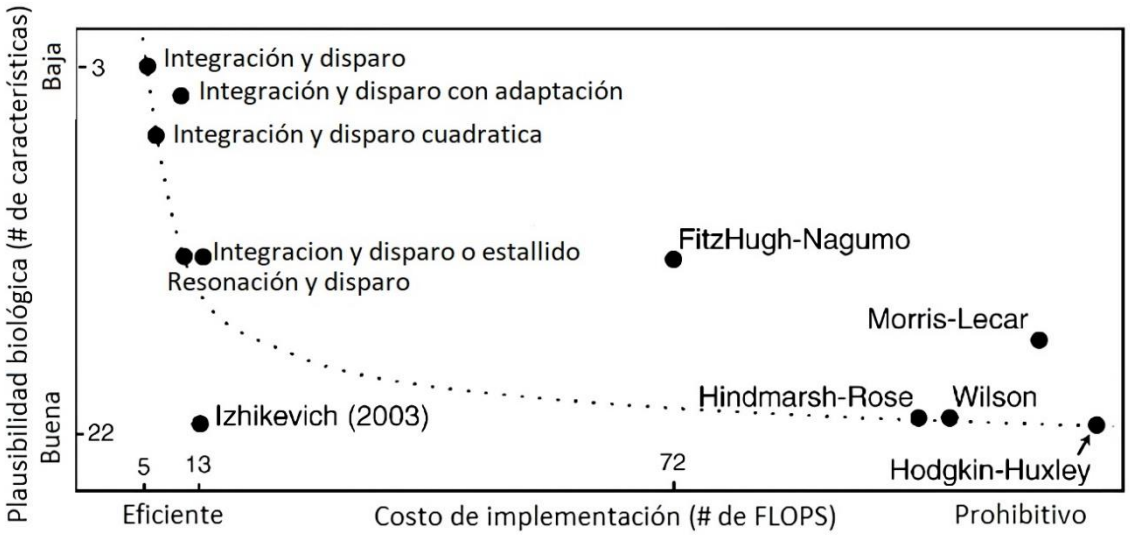

Figura 2.10 Algunos modelos de neuronas comparando semejanza biológica vs costo de implementación, según la ref. [7].

2.4 Sinapsis

Es la unión biológica mediante la cual las neuronas se comunican entre ellas, en este lugar el potencial de acción o pulso generado por una neurona pre sináptica llega al final del axón, desencadenando una serie de procesos electroquímicos que influyen de manera directa en la generación de un pulso post sináptico.

Existen varios tipos de sinapsis, se pueden resaltar dos principales, la sinapsis eléctrica y la sinapsis química.

- Sinapsis eléctrica

Se caracteriza porque en ella la transmisión de un potencial de acción fluye directamente entre las membranas, esto sucede debido a que los iones viajan a través de uniones gap, por lo que no es necesaria la liberación de neurotransmisores.

Este intercambio de iones se puede realizar de manera bidireccional, por lo que dependiendo de la resistencia que presente cada canal y la diferencia de potencial entre las neuronas, se dicta en qué dirección se da el flujo, también al no ser necesario el uso de neurotransmisores es mucho más rápida.

- Sinapsis química

En este tipo de sinapsis las neuronas no están unidas completamente, sino que existe un pequeño espacio entre ambas en donde se produce la trasmisión de información.

Como se observa en la *Figura* 2.11 las neuronas no están conectadas totalmente, entre cada axón al final existe una pequeña separación, cuando un potencial de acción llega desde la neurona pre sináptica, son liberadas biomoléculas llamadas neurotransmisores que llegan hasta la membrana post sináptica, los cuales son detectados por receptores químicos, lo que provoca que ciertos canales existentes asociados al sodio o al potasio se abran o cierren.

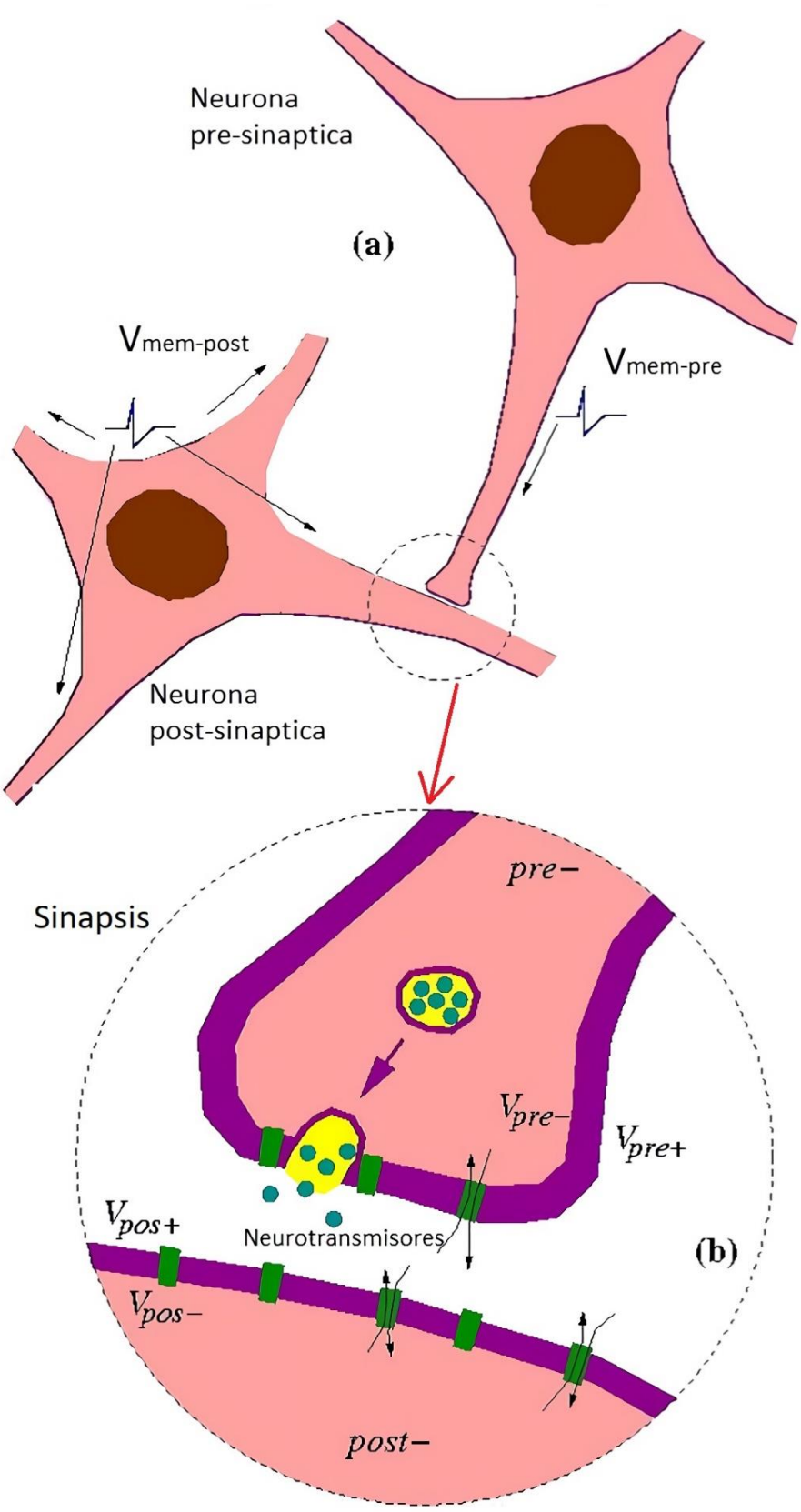

Figura 2.11 Representación de la sinapsis entre dos neuronas tomada de la ref. [8].

Una vez que los canales de la membrana post sinápticas están abiertos se produce un intercambio de iones (Na^+, K^+) lo que da como resultado una corriente, modificando así la conductancia entre las neuronas y cambiando el potencial de membrana post sináptico, incrementando o disminuyendo la probabilidad de que la neurona post sináptica genere un potencial de acción.

Como se observa en la *Figura* 2.12 tomada de la ref. [4], antes de la generación del potencial de acción los canales asociados al sodio Na^+ no están activados ni desactivados, al activarse un canal los iones de Na^+ ingresan en la célula despolarizando la membrana, activando así los canales de K^+; al cerrarse los canales la membrana se hiperpolariza.

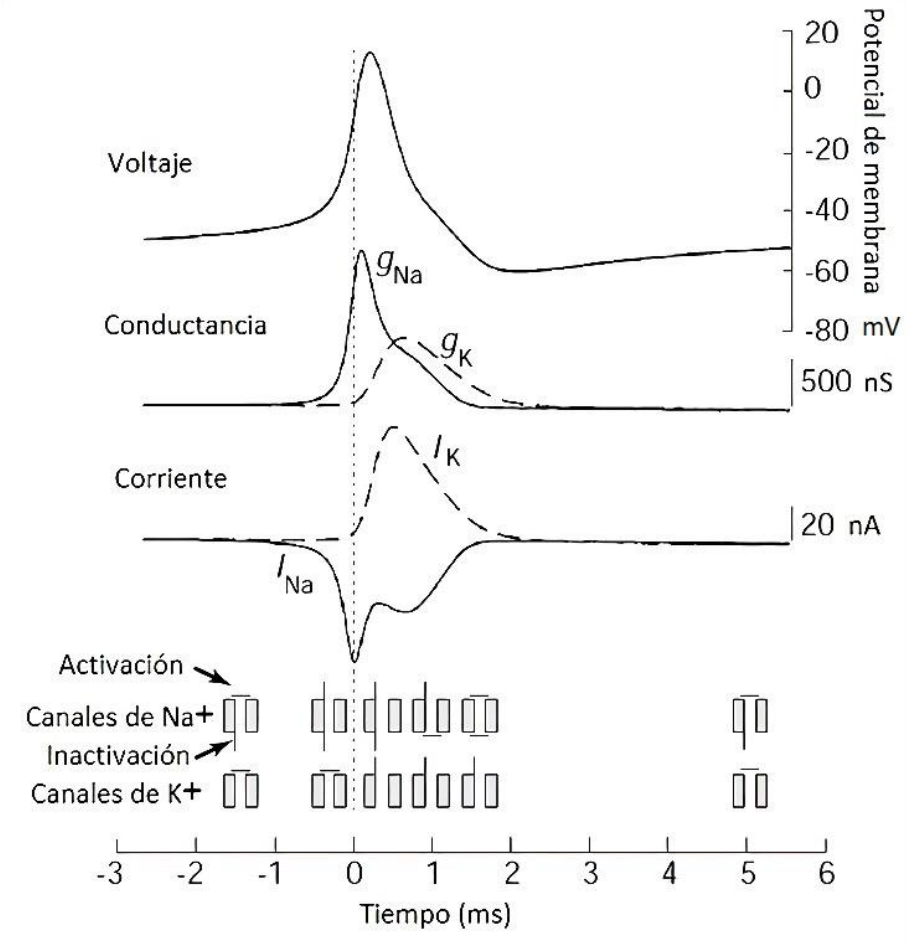

Figura 2.12 Generación de un potencial de acción asociado con el incremento de Na en la membrana, así como la conductancia y la corriente.

2.5 Plasticidad Dependiente de Pulsaciones en el Tiempo (Spike Time Dependent Plasticity, STDP)

Como se mencionó, las neuronas intercambian información entre ellas a través de los potenciales de acción que viajan a través de la sinapsis, este esquema visto desde una manera más general, es un valor analógico no volátil w, el cual puede ser interpretado como la cantidad de neurotransmisores liberados y cuyo valor determina la eficacia de un pulso pre sináptico en la posible generación de un pulso post sináptico.

Sin embargo, el valor de la sinapsis puede cambiar en función de la actividad neuronal, dicho fenómeno fue observado e informado por Hebb en 1949, y como se menciona en la ref. [8], "*cuando un axón de una celda A esta lo suficientemente cerca como para excitar una celda B y repetidamente o persistentemente toma parte en el disparo de ésta, algún proceso de crecimiento o cambio metabólico toma lugar en una o ambas celdas de tal manera que la eficiencia de A como una celda que dispara B se incrementa"*.

El cambio en el peso w puede ser positivo aumentando la fuerza de conexión entre las neuronas, pero también negativo disminuyendo la aportación que tiene un potencial de acción sobre otra neurona. Para explicar este fenómeno, fue propuesto un algoritmo conocido como STDP (Spike Time Dependent Plasticity) el cual es un mecanismo de aprendizaje Hebbiano, encargado de ajustar la fuerza de las conexiones sinápticas entre las neuronas.

Este proceso se basa en la forma como se presentan los pulso pre y post sinápticos, la repetida llegada de un pulso pre antes que un post en una cierta ventana de tiempo conlleva a la potenciación del valor w, mientras que en el caso contrario una repetida llegada de un pulso post antes que el pulso pre, conduce a la depreciación a largo plazo del valor w, por lo que el cambio del peso Δw_j de la sinapsis de una neurona pre sináptica j depende del tiempo relativo entre la llegada del pulso pre j y post i. El modelo básico del STDP (introducido por Gerstner and al. 1996, Kempter et al. 1999) establece que el cambio total Δw_j está definido por la ecuación (2.16).

$$\Delta w_j = 3\sum_{f=1}^{N}\sum_{n=1}^{N} W\left(t_i^n - t_i^f\right) \qquad (2.16)$$

En donde $W(x)$ describe una de las funciones STDP (*Figura* 2.13), también llamadas ventanas de aprendizaje; esta función es determina por medio de las ecuaciones:

$$W(x) = A_+ exp\left(\frac{-x}{\tau_+}\right) \qquad (2.17)$$

$$W(x) = -A_- exp\left(\frac{x}{\tau_-}\right) \qquad (2.18)$$

Los parámetros A_+ y A_- dependen de los valores actuales de los pesos sinápticos, así como τ_- y τ_+son las constantes de tiempo, alrededor de 10ms.

En la *Figura* 2.13 tomada de la ref. [9], se observa que es lo que ocurre cuando un pulso pre sináptico aparece antes que uno post sináptico en una ventana de tiempo, mostrando como el pulso pre contribuye directamente a la generación del post, por lo que un valor analógico asociado a la conductancia w es aumentado, haciendo mayor la contribución del pulso pre, cuando se presente de nuevo.

Por el contrario, cuando un pulso post sináptico se presenta antes que un pulso pre sináptico, este no contribuye a la generación del pulso post, por lo que la conductancia se ve disminuida, haciendo que el siguiente pulso que se presente en las mismas condiciones disminuya aún más la fuerza sináptica w.

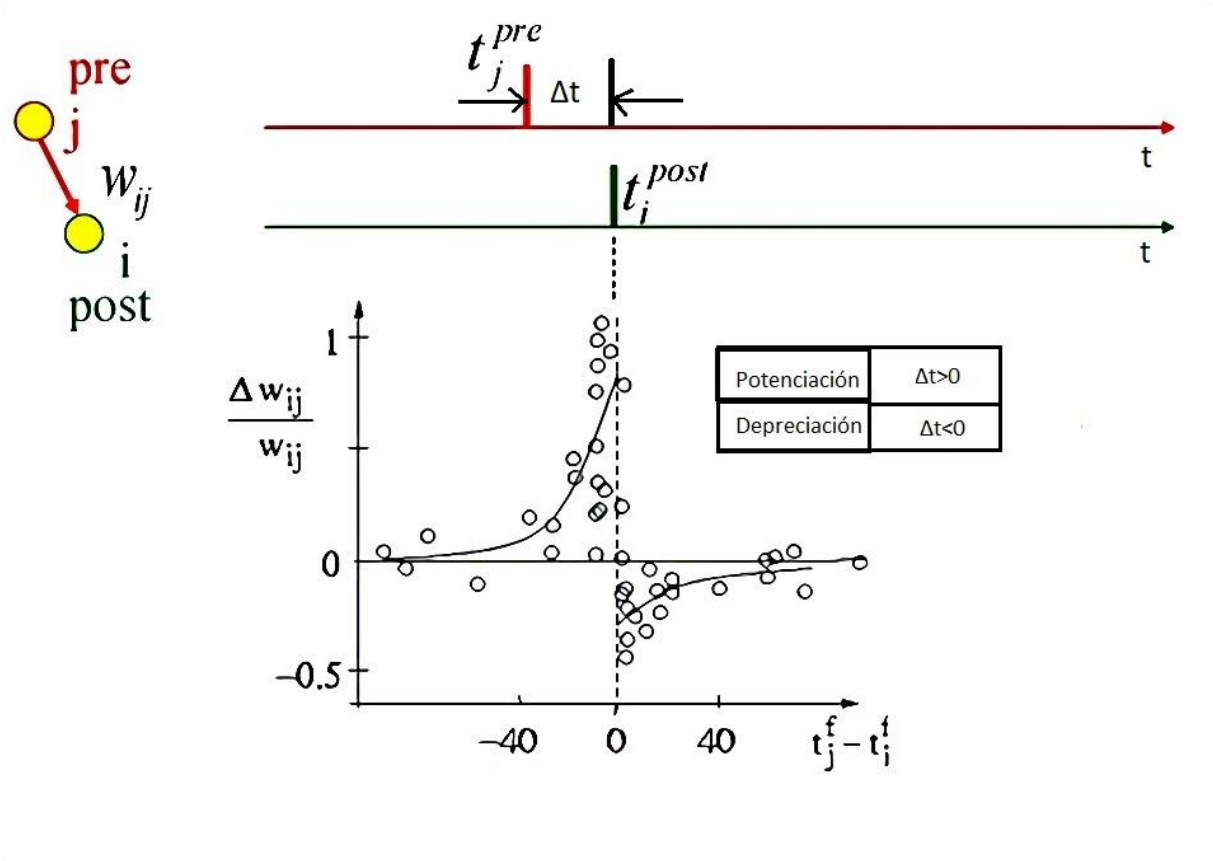

Figura 2.13 Función STDP según la ref. [9], mostrando la evolución en relación a los pulsos pre y post sinápticos, donde los círculos muestran datos experimentales.

Ahora para comprender como es que la sinapsis y el memristor se encuentran vinculados, es necesario observar la *Figura* 2.14 ,donde se muestran los casos para un pulso pre antes que post, y para un pulso post antes que uno pre.

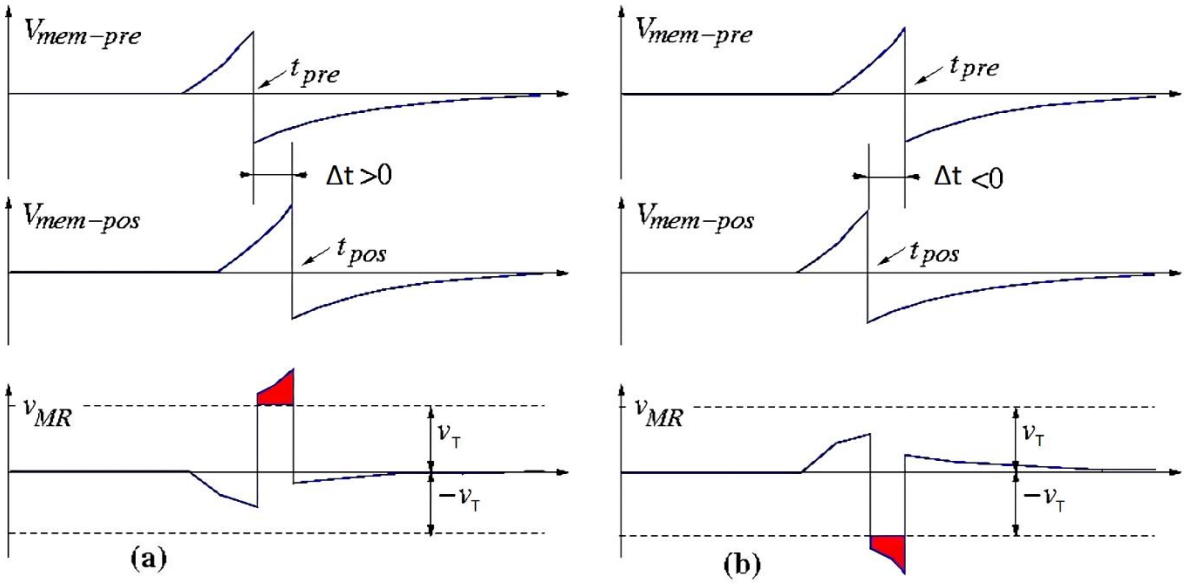

Figura 2.14 Voltaje de membrana pre y post para los casos a) pre antes que post, b) post antes que pre, imagen tomada de la ref. [8]. V_T representa el voltaje de umbral del memristor detallado en el capítulo siguiente.

Si ocurre la situación pre antes que post y suponiendo como en la ref. [8] que las ecuaciones que dominan el movimiento de los electrones en un memristor son similares al fenómeno asociado al intercambio de iones en la sinapsis, es posible colocar un memristor entre dos neuronas, por lo que al darse el primer caso donde se tiene un $\Delta T > 0$ el memristor es polarizado de manera directa (V_{MR}), y recordando el capítulo 1 donde se menciona que el memristor suele tener un umbral, señalado en color rojo cunado es superado, el memristor disminuye su resistencia interna, aumentado la conductancia análoga al valor de la fuerza sináptica w.

En cambio, si se tiene un post antes que un pre ($\Delta T < 0$) el memristor es polarizado inversamente y, de igual manera como se observa en color rojo, el voltaje de umbral se ve superado pero de manera inversa.

Recordando que un memristor cambia el valor de su resistencia interna dependiendo de la polarización que se le aplique y el tiempo, al estar inversamente polarizado, la resistencia crece, disminuyendo así el peso sináptico w, lo que implica que el mecanismo STDP emerge de manera automática sin necesidad de establecer un algoritmo de entrenamiento específico.

Ahora como determinar que pulso se encuentra primero (pre y post) cuando se tiene un tren de pulsos en una conexión sináptica depende de una ventana de tiempo, como en la *Figura* 2.15.

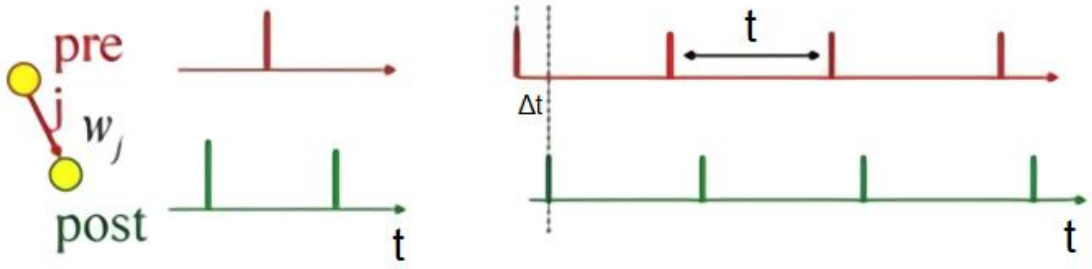

Figura 2.15 Pulsos pre y post en ventanas de tiempo tomada de la ref. [9].

Una ventana de tiempo establecida, como se menciona en la ref. [8], se observa en la *Figura* 2.13, típicamente es de 40ms, por lo que si dos pulsos comparados no se encuentran en ese lapso de tiempo, ya no se produce un voltaje de membrana que modifique el peso sináptico (el memristor no cambia sus valores de resistencia interna).

2.6 Resumen

En este capítulo, se abordó de manera general el primer modelo de neurona pulsada (Hodgkin-Huxley), así como algunos de los mecanismos que provocan su comportamiento. Se corroboró a través de los modelos: Hodgkin-Huxley, integración y disparo y Neurona de Izhikevich, que dependiendo de cuantas más características incluya un modelo, es más complejo de simular e implementar. Por lo que es necesario la correcta elección de un modelo de neurona.

Además, se presentó la idea de sustituir la sinapsis y su mecanismo de aprendizaje (STDP), por un memristor, esto gracias a que sus características lo hacen idóneo, para emular dicho comportamiento.

2.7 Referencias

[1] P. Ponce Cruz, "*Inteligencia Artificial con Aplicaciones a la Ingeniería*", Alfaomega, pp. 8-120 ,2010.

[2] M. T. Hagan "*Neural Network Design*", Oklahoma State University Sillwater, Oklahoma, pp. 30, 2014.

[3] W. Gerstner and W. M. Kistler, "*Spiking Neuron Models: Single Neurons, Populations, Plasticity*", Cambridge University Press, UK, pp. 50-220, 2002.

[4] L. R. Squire, F. E. Bloom, N. C. Spitzer, Sasha du Lac, A. Ghosh, D. Berg, "*Fundamental Neuroscience Third Edition",* Elsevier, pp. 75-225 ,2008.

[5] E. M. Izhikevich, "*Simple Model of Spiking Neurons*," in IEEE Transactions on Neural Networks, vol. 14, no. 6, pp. 1569-1572, 2003.

[6] J. Cubillos Martínez, "*Modelos de Neuronas Artificiales en Software para su uso en Preparaciones de Electrofisiología*", Universidad Autónoma de Madrid, Escuela Politécnica Superior, pp.14, 2016.

[7] E. M. Izhikevich, "*Which Model to Use for Cortical Spiking Neurons*?", in IEEE Transactions on Neural Networks, vol. 15, no. 5, pp. 5, 2004.

[8] B. Linares-Barroco and T. Serrano-Gotarredona, "*Memristance can Explain Spike-Time- Dependent-Plasticity in Neural Synapses*", Nature Precedings, pp. 3, 2009.

[9] J. Sjöström and W. Gerstner (2010) Spike Timing Dependent Plasticity. Scholarpedia, 5(2):1362., revision #184913.

Capítulo 3 Algoritmos Metaheurísticos

3.1 Introducción

Los problemas de optimización requieren disminuir recursos y maximizar los resultados, problemas tales como minimizar o maximizar una función (dependiendo el caso) al encontrar sus valores óptimos, problemas NP hard según la ref. [1], o simplemente funciones de desempeño en donde se tienen demasiados mínimos locales son difíciles de resolver, ya que no cuentan con un algoritmo especifico que otorgue una solución satisfactoria.

Fue así que en la década de 1980 fueron propuestos los primeros algoritmos metaheurísticos, donde gracias a su riqueza, estos pueden ser utilizados en todo tipo de extensión, como: optimización multiobjetivo, donde se trata de optimizar varios objetivos simultáneamente; optimización multimodal, utilizado para localizar un conjunto completo global u optimo local; optimización dinámica, que trata con variaciones temporales de la función objetivo.

Los algoritmos metaheurísticos pueden estar basados en la naturaleza, en una forma de inteligencia de enjambre, en algoritmos genéticos, e incluso pueden ser basados en trayectorias como la búsqueda local, estocástica o búsqueda tabú.

En este capítulo se describe de manera general que son los algoritmos metaheurísticos desde su propuesta y su respectiva mejora hasta nuestros días. Se mencionan algunos problemas que pueden ser resueltos de manera más eficiente identificando que algoritmo usar en cada caso, haciendo especial hincapié en el algoritmo de optimización de hormigas, y como es que gracias al memristor, este puede ser implementado de manera física usando una red de Memristores.

3.2 Algoritmos Metaheurísticos y Optimización

Los algoritmos metaheurísticos son una técnica de optimización aproximada, ya que, si bien no garantizan la optimalidad de las soluciones en todos los casos, estos ofrecen soluciones aceptables en un tiempo razonable. Pueden ser clasificados de distintas maneras, en función del problema a resolver, basados en poblaciones, trayectorias, implícitos, explícitos, etc.; sin embargo, lo que todos ellos buscan es optimizar, ya sea maximizando o minimizando una función objetivo dados ciertos parámetros de entrada. En la *Figura* 3.1 se observa un ejemplo de una función, la cual se desea optimizar, donde dependiendo de cada configuración (C_0, C_1, C_n) de sus valores de entrada llamados variables de decisión, es posible obtener valores distintos de salida.

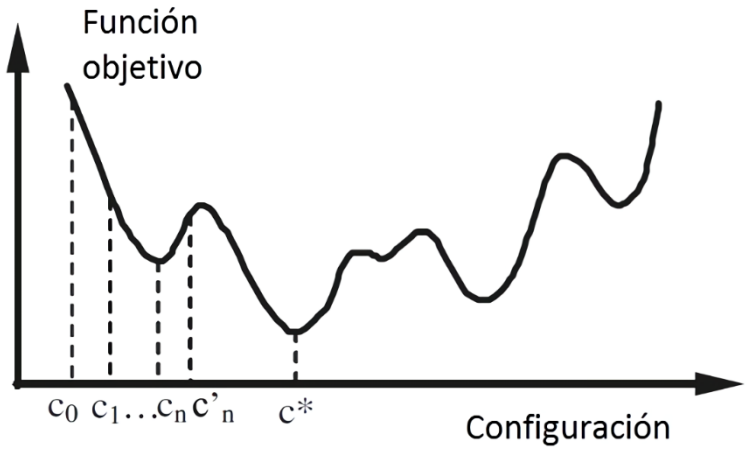

Figura 3.1 Mínimos locales en una función objetivo tomada de la ref. [1].

Por ejemplo, se deseará minimizar el valor de la función en la que en ella existen algunos mínimos locales cercanos al mínimo global, lo cual dificulta el hallar los parámetros óptimos, ya que las configuraciones C_n pueden ser tan complejas como el problema a resolver; es aquí que resulta en un esfuerzo mayor el tratar de optimizar los valores pudiendo caer en un mínimo local. Los algoritmos metaheurísticos se basan en procedimientos abstractos y genéricos con los cuales resulta sencillo resolver este tipo de problemas, ya que al caer en un mínimo local pueden salir de este para seguir buscando configuraciones.

Si bien este es un ejemplo fácil de comprender, las tareas de optimización resultan ser más complejas, y debido a esto, un problema puede ser resuelto con diferentes algoritmos metaheurísticos, en cada caso, resultando más factible uno que otro, todo esto debe tenerse en cuenta en el momento de abordar un problema.

Ya que el desempeño de un algoritmo metaheurísticos depende del problema a tratar, y existiendo varias alternativas para solucionarlo, se describe de manera breve algunos de los más populares.

3.2.1 Búsqueda Tabú

Propuesta en 1986 por Fred Glover, hace referencia al uso de la memoria de corto plazo, donde la idea principal es memorizar una lista de datos que la búsqueda local no puede utilizar, llamada lista tabú, escanea el conjunto de soluciones vecinas en cada iteración y selecciona lo mejor que no esté prohibido, incluso si la solución es peor que la actual, forzando de esta manera soluciones equivocadas y evitando así que se bloquee la búsqueda, al final de las iteraciones es determinada la mejor solución dependiendo de los criterios a evaluar del problema.

3.2.2 Algoritmo de Colonia de Abejas

Propuesto en 2005 por Karaboga está inspirado en el comportamiento de las abejas en la búsqueda de néctar, se basa en 3 tipos de abejas: obreras, exploradoras y observadoras. Las fuentes de alimento son evaluadas por las abejas tomando en cuenta criterios como la distancia, disponibilidad de extracción, etc.

Una abeja obrera explota una fuente de alimento llevándolo de regreso a la colmena, transmitiendo información de la ubicación a las abejas observadoras; estas a su vez, seleccionan las mejores fuentes de alimento y se dirigen a ellas. Una vez agotada la fuente de alimento, la abeja obrera puede convertirse en abeja exploradora, o regresar a la colmena y convertirse en abeja observadora.

Este comportamiento permite minimizar los recursos en la búsqueda de alimento, asegurándose de explotar solo los mejores lugares de alimentación.

3.2.3 Algoritmos Genéticos y Evolutivos

Este término comenzó a tomar presencia alrededor de 1933, aunque el primer reporte es atribuido a J.F. Bagley, y posteriormente a J.H. Hollannd, son unos de los algoritmos preferidos para implementación e investigación, ya que se basan en los principios de la selección natural.

En este tipo de algoritmos, a partir de expresiones equivalentes en genética como las que se muestra en la Tabla 3.1

Evolución natural	Algoritmo genético
Genotipo	Código de cadena
Fenotipo	Punto sin codificar
Cromosoma	Cadena
Gen	Posición de cadena
Alelo	Valor en una posición determinada
Función de aptitud	Valor de la función objetivo

Tabla 3.1 Equivalencia entre algoritmos genéticos y evolución natural tomada de [cap. 2, Ref. [1]].

Se tienen cadenas compuestas por características que toman diversos valores, dichas características están localizadas en distintas posiciones, por lo que las cadenas pueden combinarse para obtener nuevas soluciones. Entonces, para poder utilizar un algoritmo genético se debe tener al menos:

- Una función a optimizar.
- Un grupo de candidatos para la solución.
- Una función de evaluación para medir la optimización.
- Función de reproducción.

En este caso, por medio de los criterios de la selección natural como: herencia, cruzamiento, y mutación, las distintas cadenas representadas por un individuo con

soluciones, son modificadas obteniendo una nueva generación de soluciones, las cuales pueden ir optimizando la función, al minimizar o maximizar ya sea el caso.

3.3 Algoritmo de Optimización de Hormigas

Es un algoritmo basado en la inteligencia de enjambre (abejas, termitas, hormigas), en este tipo de estructura cada individuo es muy simple, sin embargo, tienen una función específica, lo cual permite una completa organización a la colonia, así en caso de faltar un individuo este es rápidamente remplazado con otros, lo cual implica una alta plasticidad y capacidad de autoorganización. El cómo surge una inteligencia colectiva superior, a partir de la interacción de individuos más simples que por sí solos no son relevantes, aún es investigado y debatido.

Para este caso, ¿Cómo las hormigas optimizan las rutas más cortas entre su nido y fuente de alimento?, fue reportado en la ref. [2] usando un simple experimento, donde una colonia de hormigas se encuentra separada de la fuente de alimento, y las hormigas realizan una optimización natural al elegir la ruta más corta entre ambos puntos.

En la *Figura* 3.2 y *Figura* 3.3 tomada de [2] se observa cómo se coloca una colonia de hormigas a una cierta distancia del área de alimentación, posteriormente se trazan rutas más complejas y las hormigas son liberadas. Al inicio se distribuyen uniformemente por todas las rutas (4 min.), recogiendo alimento y llevándolo de regreso al nido, pero tiempo después (8 min.) comienzan a tomar la ruta más corta ignorando las demás, realizando una optimización natural.

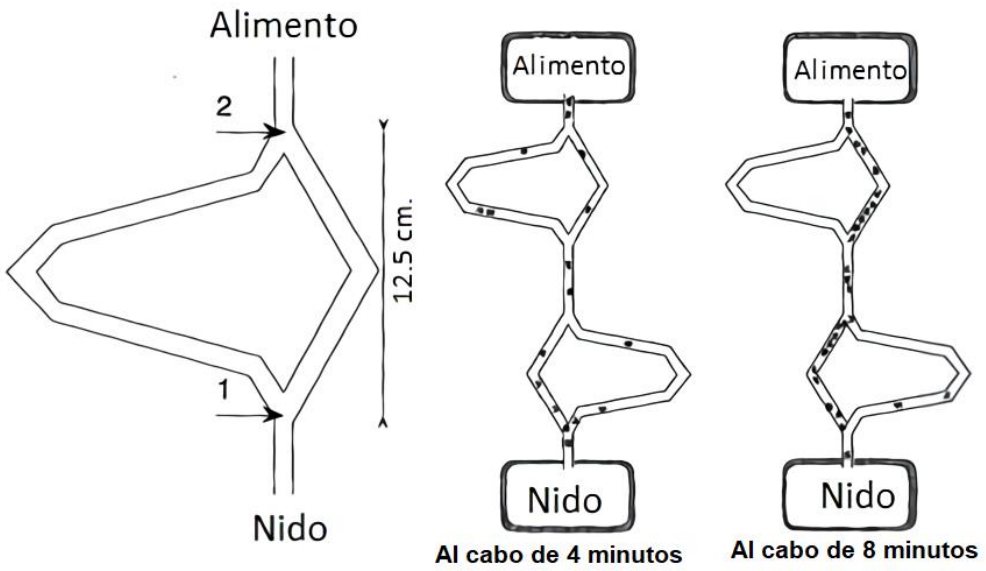

Figura 3.2 Experimento usando una colonia de hormigas Linepithema humile, según la ref. [2].

En la *Figura* 3.3 se observa una colonia de hormigas de la especie Linepithema humile, tomadas del experimento real reportado en la ref. [2], viajando desde el nido hasta el área de alimentación por el camino más corto. Ahora si bien esto es realizando la liberación las hormigas por rutas confinadas, el mismo comportamiento también ocurre al colocar un obstáculo entre ambos puntos.

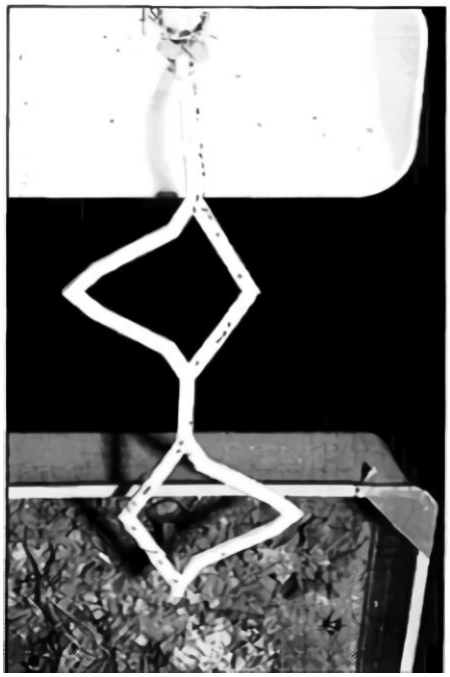

Figura 3.3 Colonia de hormigas eligiendo el camino más corto entre dos puntos, reportada en la ref. [2].

En la *Figura* 3.4 tomada de la ref. [1] se observa como al inicio las hormigas se encontraban en una ruta simple, al colocar un obstáculo las hormigas rodean dicho objeto de manera homogénea, pero al pasar el tiempo eligen la ruta más corta.

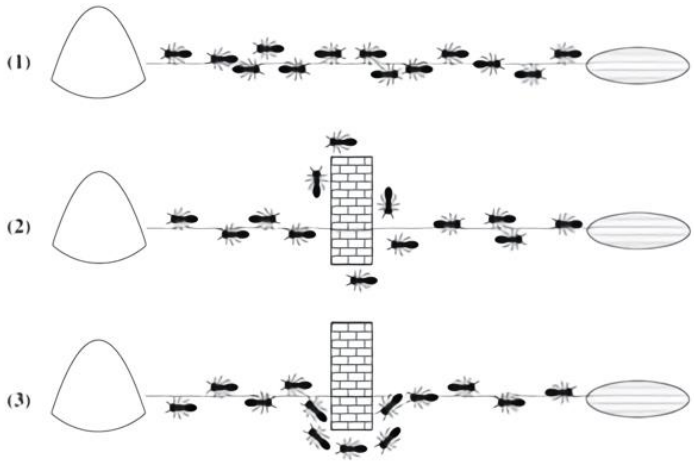

Figura 3.4 Hormigas rodeando un obstáculo para llegar a la fuente de alimento, tomado de la ref. [1].

Este comportamiento fue explicado por Dorigo en la ref. [3], y tal como se menciona, cuando las hormigas viajan a través de una ruta en busca del alimento, van depositando una sustancia llamada feromona, otras hormigas perciben la presencia de esta feromona y tienden a seguir caminos donde la concentración es mayor, sin embargo la feromona se evapora con el tiempo por lo que si una hormiga elige el camino más largo, le tomara un tiempo mayor regresar al nido haciendo que la feromona comience a evaporarse, en cambio al elegir el camino más corto, el tiempo transcurrido es menor y la evaporación de la feromona no es tan relevante como en el camino largo, por lo que al regresar a la fuente de alimento elegirá el camino más corto al ser más reciente el depósito de la feromona, provocando que sea aún más atractiva la ruta corta, volviendo a depositar la feromona en este camino y haciendo menos atractiva la ruta larga al no volver a transitar en ella.

De esta forma les es posible transportar alimento de manera efectiva hacia su nido, ya que no se dispersan por caminos innecesariamente largos. Para explicar este mecanismo, fue presentado en 1991,1992 y 1996 por Dorigo el sistema de hormigas reportado en la ref. [4], donde la característica principal es que las feromonas son actualizadas por todos los individuos que han completado el recorrido:

$$\tau_{ij} = (1-\rho) * \tau_{ij} + \sum_{k=1}^{m} \Delta {\tau_{ij}}^{k} \qquad (3.1)$$

Donde τ_{ij} es el valor de la feromona, ρ∈ (0,1) el cual es el parámetro de evaporación de las feromonas, m es el número de hormigas, i, j asocian el lugar donde fue colocada la feromona.

$$\Delta {\tau_{ij}}^{k} = \begin{cases} \frac{1}{L_k} & si\ la\ hormiga\ k\ utilizó\ dicho\ camino \\ 0 & en\ otro\ caso \end{cases} \qquad (3.2)$$

En este caso L_k es la longitud del camino recorrido por la hormiga, $\Delta {\tau_{ij}}^{k}$ es la cantidad de feromona dejada en la ruta i, j, por lo que solo se ve reforzada la

feromona del camino recorrido y a las otras rutas comienza a afectarles el parámetro de evaporación.

Por último, la probabilidad $p(c_{ij}|s_k^p)$ de tomar un camino para la hormiga que se mueve del punto A al punto B es:

$$p(c_{ij}|s_k^p) = \begin{cases} \dfrac{\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}{\sum_{c_{il} \in \mathrm{N}(s_k^p)} \tau_{il}^{\alpha} * \eta_{il}^{\beta}} & sí\ j \in \mathrm{N}(s_k^p) \\ 0\ en\ otro\ caso \end{cases} \qquad (3.3)$$

Donde $\mathrm{N}(s_k^p)$ es un conjunto de componentes que aún no pertenecen a la solución parcial s_k^p de la hormiga k, α y β son parámetros de control relativo de la importancia de la feromona vs la información heurística $\eta_{ij} = \frac{1}{d_{ij}}$, donde d_{ij} es la longitud del componente c_{ij}.

Entonces, este algoritmo es fácil de comprender, ya que al viajar las hormigas del punto A al punto B, eligen caminos homogéneos al principio; después, las feromonas son actualizadas, y por último la probabilidad de elegir un camino es definida, repitiendo este proceso llega un momento en que la mejor ruta es determinada.

Las aplicaciones para el algoritmo de hormigas han sido enfocadas en la optimización de rutas, como en el ya conocido “Travelling Salesman Problem” donde se desea elegir la ruta más corta para visitar cierto número de ciudades y regresar al punto de partida, también en ruteo de componentes electrónicos donde hay que optimizar el espacio y distancia entre los distintos elementos colocados en una placa, así como en la detección de bordes en imágenes.

3.4 Red de Memristores y el Algoritmo de Hormigas

Dado los buenos resultados obtenidos con el algoritmo de hormigas, así como la investigación acerca de los memristores, una rápida asociación fue dada entre el algoritmo de optimización de hormigas y las redes de memristores usadas en memcomputing, ya que como se plantea en la ref. [5] las redes memristivas pueden ser el análogo eléctrico del algoritmo de hormigas al surgir dicho comportamiento de manera automática.

Para comprender como es que esta asociación existe, en la ref. [5] se presenta una tabla con las variables físicas de la inteligencia de enjambre de las hormigas, su algoritmo y su análogo con los memristores.

Como se observa en la *Tabla* 3.2, los memristores pueden ser el equivalente casi uno a uno del algoritmo de hormigas, sin embargo, para explicar cómo se encuentran relacionados se hace uso del algoritmo presentado en la ref. [6].

Naturaleza	Algoritmo de Hormigas	Memcomputing
Hábitat natural	Grafo	Red de memristores
Hormigas	Hormigas artificiales	Electrones
Longitud del camino	Reciproco de la distancia	Conductancia
Feromonas	Feromonas artificiales	Memoria del dispositivo
Evaporación de feromonas	Evaporación artificial de feromonas	Estado de relajación del dispositivo
Comportamiento de búsqueda	Reglas de selección	Leyes de Kirchhoff

Tabla 3.2 Equivalencia entre algoritmo de hormigas y red de memristores

Este algoritmo, está definido por:

$$p_{path_m} = \frac{\prod_{(i_t,j_t)\in path_m} \tau(i_t,j_t)^{\alpha} \left(\frac{1}{Le_{path_m}}\right)^{\beta}}{\sum_{f=1}^{M} \prod_{(i_t,j_t)\in path_f} \tau(i_t,j_t)^{\alpha} \left(\frac{1}{Le_{path_f}}\right)^{\beta}} \qquad (3.4)$$

$$Le_{path_m} = \sum_{(i_t,j_t)\in path_m} \eta^{-1}_{(i_t,j_t)} \qquad (3.5)$$

$$\tau_{(i,j)}(k+1) = (1-\rho)\tau_{(i,j)} + \frac{vQ}{Le_{path_m}} \qquad (3.6)$$

En donde la ecuación (3.4) p_{path_m} es la probabilidad de tomar un camino, la ecuación (3.5) Le_{path_m}es la longitud de ese camino, $\tau_{(i,j)}$ (ecuación (3.6)) es la feromona asociada a un camino, $\eta^{-1}_{(i_t,j_t)}$ es la heurística asociada ese camino, ρ es el parámetro de evaporación de feromonas (0-1), α y β son parámetros que definen la importancia de la heurística vs la feromona, y por ultimo $v\ y\ Q$ son parámetros de ajuste típicamente valuados en 1.

Ahora para explicar cómo esta equivalencia es válida se hace uso del ejemplo reportado en la ref. [6], donde se tienen dos caminos, como en la *Figura* 3.5 representados por un arreglo de memristores equivalentes, una ruta tiene el doble de distancia que la otra, con valores de $Le_1 = 1.3$, $Le_2 = 2.266$, donde es posible resolver el problema con el uso del algoritmo de hormigas.

Este ejemplo es el más sencillo de mostrar, ya que en él solo existen dos opciones a elegir, donde una ruta es más larga que otra.

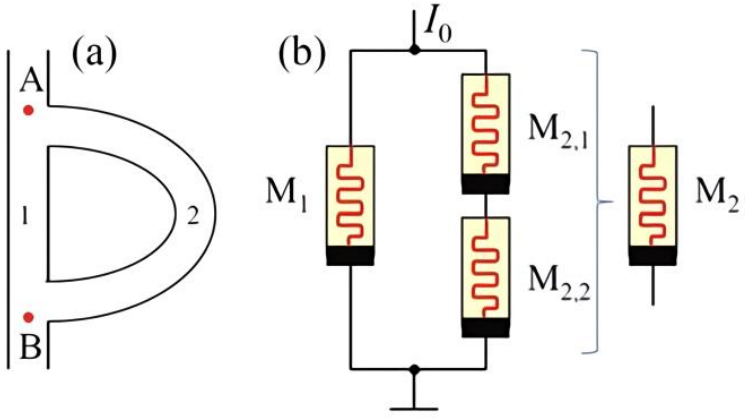

Figura 3.5 Caminos emulados con memristores tomado de la ref. [5].

Este ejemplo puede ser resuelto con el algoritmo de hormigas, así como con su equivalente en redes memristores, en la ref. [6] se reporta como es calculada la probabilidad de elegir una ruta, así como la actualización de las feromonas, dichos valores están definidas por:

$$p_{1(2)} = \frac{\tau_{1(2)}^{\alpha} \left(\frac{1}{Le_{1(2)}}\right)^{\beta}}{\tau_1^{\alpha} \left(\frac{1}{Le_1}\right)^{\beta} + \tau_2^{\alpha} \left(\frac{1}{Le_2}\right)^{\beta}} \qquad (3.7)$$

$$\tau_{1(2)}(k+1) = (1-\rho)\tau_{1(2)} + \frac{vQ}{Le_{1(2)}} \qquad (3.8)$$

Donde 1(2) indican el valor a tomar de Le, y si se asume que un número de hormigas entran en las rutas a una razón constante γ, entonces la cantidad de hormigas agregadas en un tiempo dt es γdt, por lo que haciendo uso de la ecuación (3.8) se tiene que el cambio en las feromonas es:

$$\frac{d\tau_{1(2)}}{dt} = -\gamma\rho\tau_{1(2)} + p_{1(2)}\frac{\gamma vQ}{Le_{1(2)}} \qquad (3.9)$$

Ahora, sustituyendo la ecuación (3.7) en (3.9) se tiene que el cambio de las feromonas en función del tiempo con una entrada constante de individuos (hormigas) es:

$$\frac{d\tau_{1(2)}}{dt} = -\gamma\rho\tau_{1(2)} + \frac{\gamma vQ}{Le_{1(2)}} \frac{\tau_{1(2)}^{\alpha} \left(\frac{1}{Le_{1(2)}}\right)^{\beta}}{\tau_1^{\alpha} \left(\frac{1}{Le_1}\right)^{\beta} + \tau_2^{\alpha} \left(\frac{1}{Le_2}\right)^{\beta}} \qquad (3.10)$$

En cuanto a la posibilidad de resolver este problema con memristores, se hace uso del modelo presentado en la ref. [6], donde el cambio de la variable de estado está definido por:

$$\frac{dx}{dt} = KI(t) + \xi x \qquad (3.11)$$

Donde ξ es el parámetro de relajación, K es el termino de arrastre que formula una dependencia entre x y la corriente que atraviesa el memristor. En este ejemplo se toma como el camino más corto el que presente menor resistencia, por lo que se asume en función de la memductancia (G), la cual está definida por:

$$M^{-1}(x) = G(x) = G_{on}x + G_{off}(1 - x) \qquad (3.12)$$

En este caso G_{on} y G_{off} representan los límites entre los cuales puede oscilar la memductancia, la cual en una analogía según ref. la [6], puede representar la inversa de la distancia, por lo que $G_{off1} = 1/1.3$ y $G_{off2} = 1/2.266$. Entonces, solo resta ver cómo es que la corriente se reparte entre los caminos de memristores, pudiéndose escribir como:

$$I_{1(2)} = I_0 \frac{G_{1(2)}}{G_1 + G_2} \qquad (3.13)$$

Ahora considerando las ecuaciones (3.11) y (3.13) para reescribir la ecuación (3.12), se tiene que el cambio de conductancia en función del tiempo asociada a la corriente en el memristor es:

$$\frac{dGn_{1(2)}}{dt} = -\xi\left(Gn_{1(2)} - 1\right) + KI_0\left(\frac{G_{on1(2)}}{G_{off1(2)}} - 1\right)\frac{Gn_{1(2)}(t)G_{off1(2)}}{Gn_1(t)G_{off1} + Gn_2(t)G_{off2}} \qquad (3.14)$$

Ahora, tanto la ecuación (3.10) como (3.14), pueden ser integradas numéricamente para obtener los resultados mostrados en la *Figura* 3.6, donde se aprecia su comparación al resolver el mismo problema con el algoritmo de hormigas y con la red de memristores.

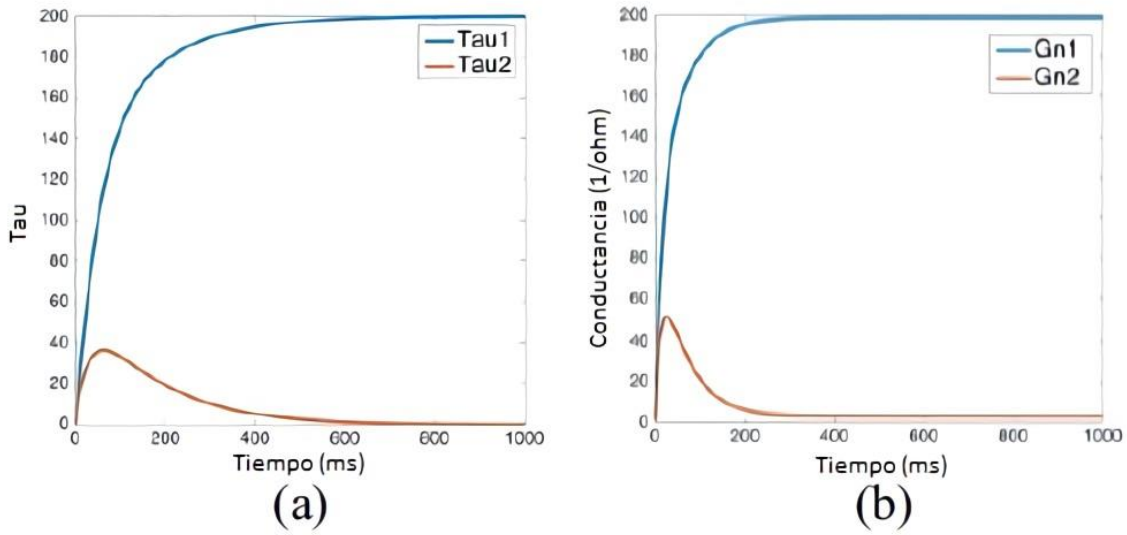

Figura 3.6 Comparación entre: a) algoritmo de hormigas y b) red de memristores.

En la *Figura* 3.6 presentada en la ref. [6], se observa que con ambos métodos es posible hallar la ruta más corta al elegir ambas la opción 1 (Tau 1, Gn1), los parámetros utilizados en (a) son: $\gamma = 20$, $\rho = 1$, $\tau_{1(2)}(0) = 0.01$, así como para (b) son: $I_o = 1$, $\xi = 50$, $G_{off1} = 2.66$, $G_{off2} = 1.3$, $G_{on1} = 2200$, $G_{on2} = 1300$, $Gn_{1(2)} = 1$. En el siguiente capítulo se demostrará el funcionamiento de los memristores para un problema de la misma índole, pero más complejo e implementado en un FPGA.

3.5 Resumen

En este capítulo, se abordó de manera general la explicación de un algoritmo metaheurístico, mencionando algunos ejemplos como: los algoritmos genéticos y evolutivos, la búsqueda tabú y el algoritmo de abejas. Se detalló el funcionamiento del algoritmo de optimización de hormigas, al encontrar la ruta más corta entre dos puntos, y posteriormente fue comparado con una red de memristores, donde se verificó que efectivamente pueden ser equivalentes.

Se detallo cómo se realiza la equivalencia entre el algoritmo de hormigas y la red de memristores, al agregar un término de relajación al modelo del memristor, y se planteó la posibilidad de resolver un problema presentado para el algoritmo de hormigas, con el uso de redes memristivas.

3.6 Referencias

[1] P. Siarry (Editor), *"Metaheuristics"*, Springer International Publishing, Switzerland, pp. 4-20, 2016.

[2] E. Bonabeau, M. Dorigo, G. Theraulaz, "*Swarm Intelligence Form Natural to Artificial Systems*", Oxford University Press, New York, pp. 25-53, 1999.

[3] M. Dorigo and T. Stützle, "*Ant Colony Optimization"*, Cambridge: MIT Press, pp. 7, 2004.

[4] M. Dorigo. (2007) Ant Colony Optimization, *Scholarpedia*, 2(3):1461. [Online]. Available: http://www.scholarpedia.org/article/Ant_colony_optimization .

[5] Pershin, Yuriy V. and Massimiliano Di Ventra. "*Memcomputing and Swarm Intelligence*." ArXiv abs/1408.6741, pp. 6, 2014.

[6] Z. Pajouhi and K. Roy, "*Image Edge Detection Based on Swarm Intelligence Using Memristive Networks*", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 9, pp. 1774-1787, 2018.

Capítulo 4 Implementación del modelo del memristor y neurona

4.1 Introducción

En este capítulo, se presenta la metodología para la implementación digital de un modelo del memristor con umbral controlado por voltaje, el cual presenta ciertas características que lo hicieron idóneo para su uso en FPGA. Verificando los tres fingerprints ya conocidos: 1) histéresis con cruce en el origen, 2) colapso de la histéresis conforme aumenta la frecuencia, 3) capacidad de ajuste multi-estado.

También se detalla la implementación de un modelo de neurona bio-inspirado capaz de asemejar características simplificadas de una neurona real, el cual está basado en un pulsador controlado por voltaje, donde un jitter es incluido para evitar la similitud en la separación de los pulsos. Esta neurona junto con el memristor es utilizada para el diseño de una red neuronal pulsada.

Se muestran aplicaciones realizadas con este modelo, tanto de manera física como simuladas, pasando desde la memoria asociativa, al recrear el experimento de Pávlov, observado desde el punto de vista neurológico; el reconociendo de caracteres en una matriz dinámica, al expandir las bases del primer ejemplo, calculando el error cuadrático medio conforme la red aprende un patrón; la comparación contra el algoritmo de optimización de hormigas, al encontrar la ruta más corta entre dos puntos por medio de un arreglo de memristores; hasta su uso en el procesamiento de imágenes, al utilizar una red de memristores para la detección de bordes en imágenes, comparándolo con otros métodos como el algoritmo de hormigas.

4.2 Modelo del Memristor Utilizado en el Circuito Eléctrico para la Emulación del Comportamiento de la Ameba

El comportamiento memristivo se encuentra presente en materiales, animales, sistemas biológicos, e incluso en el arco eléctrico. Si bien era observado en laboratorios ya en años previos a la propuesta de Chua, este simplemente era asociado a otras posibles causas y pasado por alto; pero gracias al auge que se tuvo en la comunidad científica en el año 2008, pronto se comenzó a asociar el memristor con distintos sistemas.

Uno de estos sistemas es la ameba, la cual es capaz de predecir eventos periódicos según lo reportado en la ref. [1], donde, dadas ciertas condiciones, la ameba recuerda cada cuanto se repite un evento determinado y cambia su comportamiento intentando predecir las condiciones. Rápidamente fue propuesto en la ref. [2] un circuito eléctrico que asemejaba el comportamiento de la ameba, donde se propone el uso del memristor como elemento de memoria.

En la *Figura* 4.1, se puede apreciar en a) el diagrama del circuito encargado de emular el aprendizaje de la ameba, b) función de comportamiento del memristor con umbral, la cual depende del voltaje aplicado.

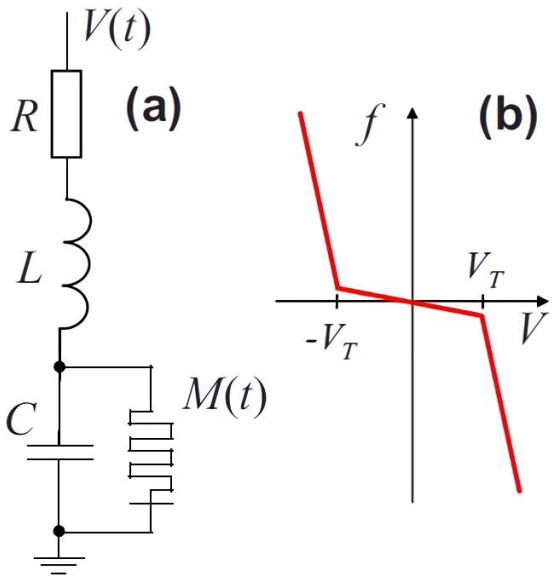

Figura 4.1 a) Circuito de aprendizaje de la ameba realizado con los cuatro elementos fundamentales, b) respuesta obtenida en el memristor, según la ref. [2].

El modelo de este diagrama está dado por la ecuación (4.1).

$$f(v) = -\beta V + 0.5(\beta - \alpha)(|V + V_T| - |V - V_T|) \qquad (4.1)$$

En este caso V_T es el voltaje de umbral, mientras que α y β son parámetros constantes que caracterizan el cambio de memristancia. Es de esta propuesta que se toma el modelo a trabajar, el cual asemeja con éxito las características principales de un memristor, todo esto sin consumir recursos excesivos; esto lo hace idóneo para su implementación en FPGA.

Ahora, según las referencias [3] y [4], un modelo de memristor que remplace un memristor real debe contar con ciertas características:

- Amplio rango de memristancia
- Operación flotante
- Capacidad de conectarse con otros dispositivos
- No volátil
- Configuración de su estado inicial
- Operación para alta frecuencia y señales continuas de entrada.

Por lo que con base en la ecuación (4.1), en la ref. [3] se propone el siguiente sistema de ecuaciones.

$$i(t) = M^{-1}\, v(t) \qquad (4.2)$$

$$\dot{M} = b * v + \frac{1}{2}(\alpha - b)(|V + V_T| - |V - V_T|)\, \theta(R - R_{ON})\, \theta(R_{OFF} - R) \qquad (4.3)$$

Este es el conocido memristor bipolar con umbral controlado por voltaje, donde la ecuación (4.2) indica la corriente según la ley ohm (considerando el memristor como una resistencia instantánea), mientras que la ecuación (4.3) describe el cambio de memristancia en función de los parámetros α, β, V_T y las funciones θ.

- α es la razón de cambio de resistencia antes de superar el umbral. Típicamente es un valor negativo o cero dependiendo del memristor a asemejar.
- β indica la razón de cambio una vez superado el umbral, entre mayor sea este valor más rápido cambia la memristancia. Para un memristor decremental, donde la memristancia disminuye si este es polarizado directamente, e incrementa al polarizarlo inversamente, las constantes deben ser: α, β < 0, y |α| < |β|.
- V_T es el umbral, este puede ser simétrico y usarse el mismo valor en toda la ecuación 4.2, o usarse con distintos valores para asemejar mejor un memristor determinado.
- La función θ limita el rango en el cual puede cambiar la memristancia, en este caso R_{ON} es la mínima resistencia y R_{OFF} con la máxima resistencia.

Realizando una simulación en Matlab de la ecuación 4.2 (*Figura* 4.2), se pueden apreciar los efectos de cada valor, principalmente el umbral y las pendientes generadas por α, β. Donde se aprecia que con valores: α=-2000 $\frac{\Omega}{V \cdot s}$, β=-190 000 $\frac{\Omega}{V \cdot s}$ y V_T=1v para un umbral simétrico, el cambio antes del umbral es pequeño, por otro lado, al superar el umbral V_T, existe un cambio de M mayor.

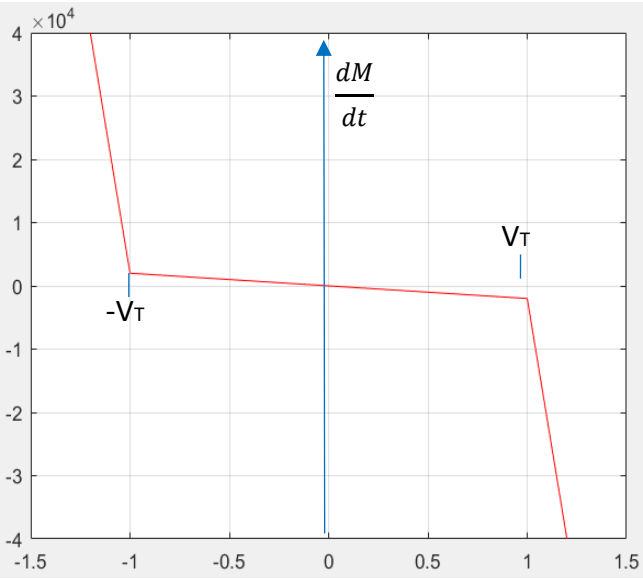

Figura 4.2 Cambio de M dependiendo del voltaje aplicado simulado en Matlab.

4.3 Implementación del Modelo del Memristor en FPGA

Conociendo las características con las que debe contar un memristor, así como las ecuaciones a utilizar, se asume la propuesta presentada en la ref. [3], por lo que el modelo del memristor tendrá que contar con las conexiones mostradas en la *Figura* 4.3 A y 4.3 B.

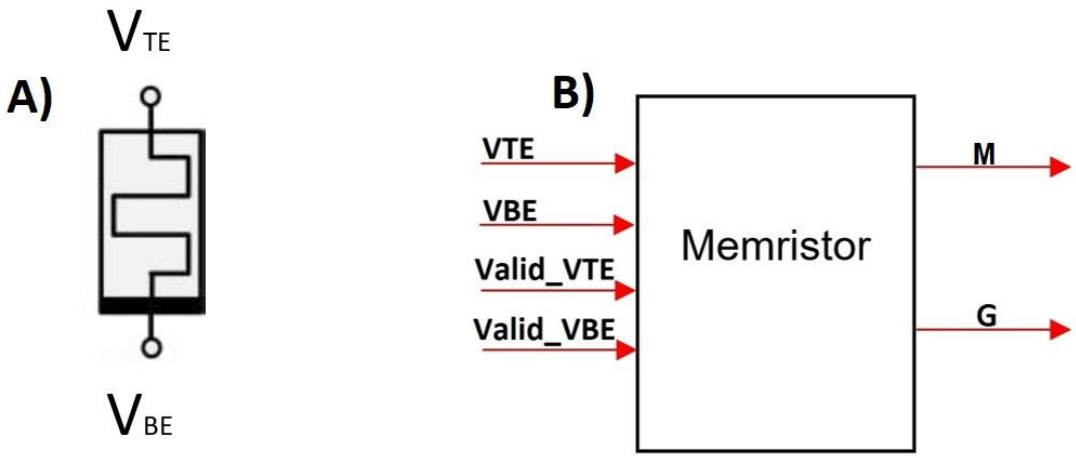

Figura 4.3 A) Símbolo del memristor físico y B) módulo de un memristor digital.

Donde VTE es el voltaje en el electrodo superior, VBE es voltaje en el electrodo inferior y Valid_VTX indica si se cuenta con un valor de voltaje válido para cada electrodo; además, ya que se trata de una implementación digital, el valor que entrega este módulo a su salida será memristancia o memductancia (M, G) dependiendo cual se dese usar. Hay que tomar en cuenta que el valor inicial de resistencia interna se define dentro del módulo, así como los valores de la ecuación (4.3).

Antes de la implementación, se verificará el correcto funcionamiento en el software Matlab (código que se puede encontrar en el apéndice A, sección 2), así como en Simulink. Posteriormente, se realiza en System Generator, se verifica de nuevo y finalmente es descargado en el FPGA.

El FPGA usado para para implementar este módulo, como para los otros ejemplos mostrados más adelante es:

- Nexxys 4DDR Artyxs-7, mostrada en la Figura 4.4

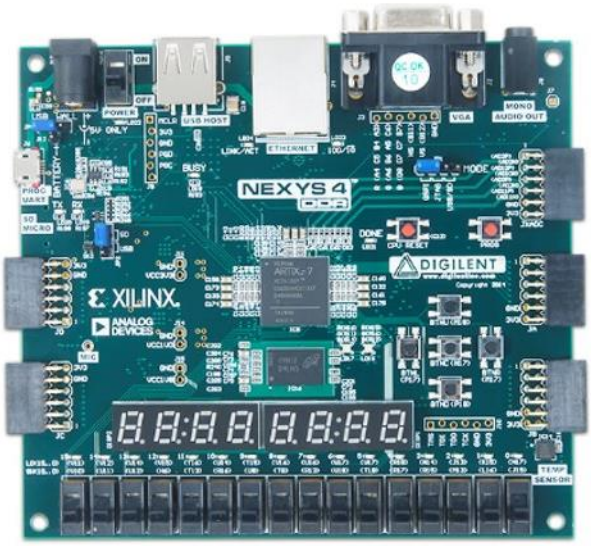

Figura 4.4 FPGA Nexys 4DDR Artix-7.

Esta tarjeta proviene de DIGILENT, una de las marcas de la compañía XILINX. El programa usado para generar el código, programar el FPGA y comprobar por medio de co-simulación el correcto funcionamiento, es conocido como System Generator,

siendo posible enlazarlo con bloques de Simulink a través de una conversión de datos.

Este FPGA cuenta con las siguientes características:

- 15 850 logic slice con cuatro LUT de 6 entradas y 8 flip-flops.
- 240 DSP's.
- 4 860 Kbits de RAM de bloque rápido.
- Velocidades de reloj interno superiores a 450MHz.
- Convertidor analógico-digital (XADC) incorporado.
- Seis bloques para la gestión de reloj, cada uno con un lazo de fase cerrada (PLL).

De aquí en adelante, cuando se mencione Co-Simulación nos referimos a una implementación realizada en el FPGA, pero con estímulos enviados a través de una computadora por medio de USB-Serial. Una vez que el FPGA realiza su función, los datos son enviados de regreso hacia la computadora para poder almacenarlos, o visualizarlos con ayuda de Simulink.

En la *Figura* 4.5, se aprecia un bloque de memristor construido con System Generator, el diagrama interno de este se puede apreciar en la *Figura* 4.6. Para comprobar su funcionamiento se hace uso de la co-simulación, donde todo el módulo del memristor estará en el FPGA y solo los estímulos serán generados por medio de una computadora, los datos recibidos de M y G serán visualizados con ayuda de Simulink.

Este memristor implementado en la placa Nexys 4DDR Artix-7 será usado en 4 ejemplos, los cuales pueden ser seleccionados por medio del Multipor Switch. Los parámetros del memristor serán los mismos para estos ejemplos, y solo cambiará el tipo de estímulo enviado.

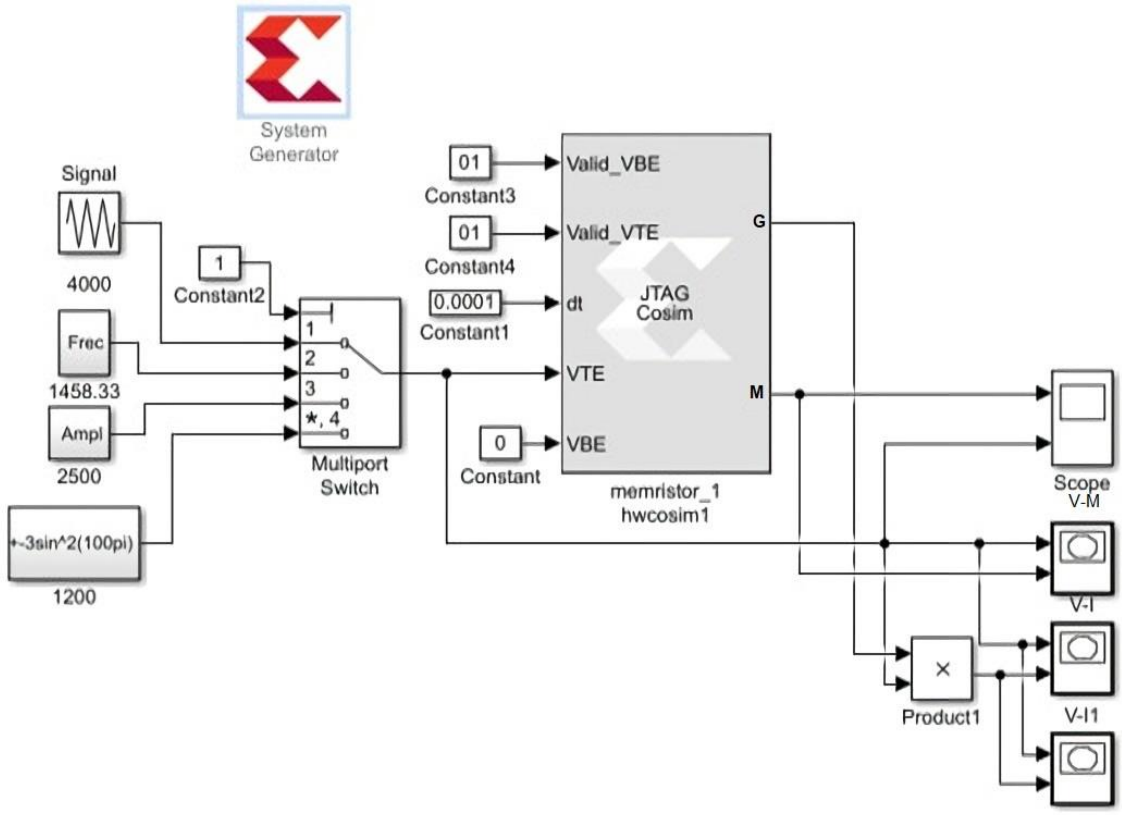

Figura 4.5 Modulo Memristor a corroborar con ayuda de la co-simulación.

Como se aprecia en la *Figura* 4.6, dentro del módulo memristor, existen pequeños bloques amarillos que transforman los datos enviados por Simulink a un tipo de dato compatible con los bloques de System Generator, e inversamente a la salida.

Lo primero que se realiza, es implementar la ecuación 4.2 que indica el cambio de memristancia con respecto al tiempo; los parámetros propuestos en la ref. [3] que se utilizaron son: $a = -2000\frac{\Omega}{V \cdot s}$, b $= -190\,000\frac{\Omega}{V \cdot s}$, $\Delta t = 0.0001s$, $V_T = 1V$, $R_{min} = 100\Omega$, $R_{max} = 10K\Omega$, $R_{init} = R_{max}$.

Una vez construida esta parte del módulo, es necesario integrar para obtener la memristancia M, lo cual se hace con ayuda de un $\Delta \boldsymbol{t}$ y un registro para poder dar las condiciones iniciales; esto se encuentra señalado en la *Figura* 4.6.

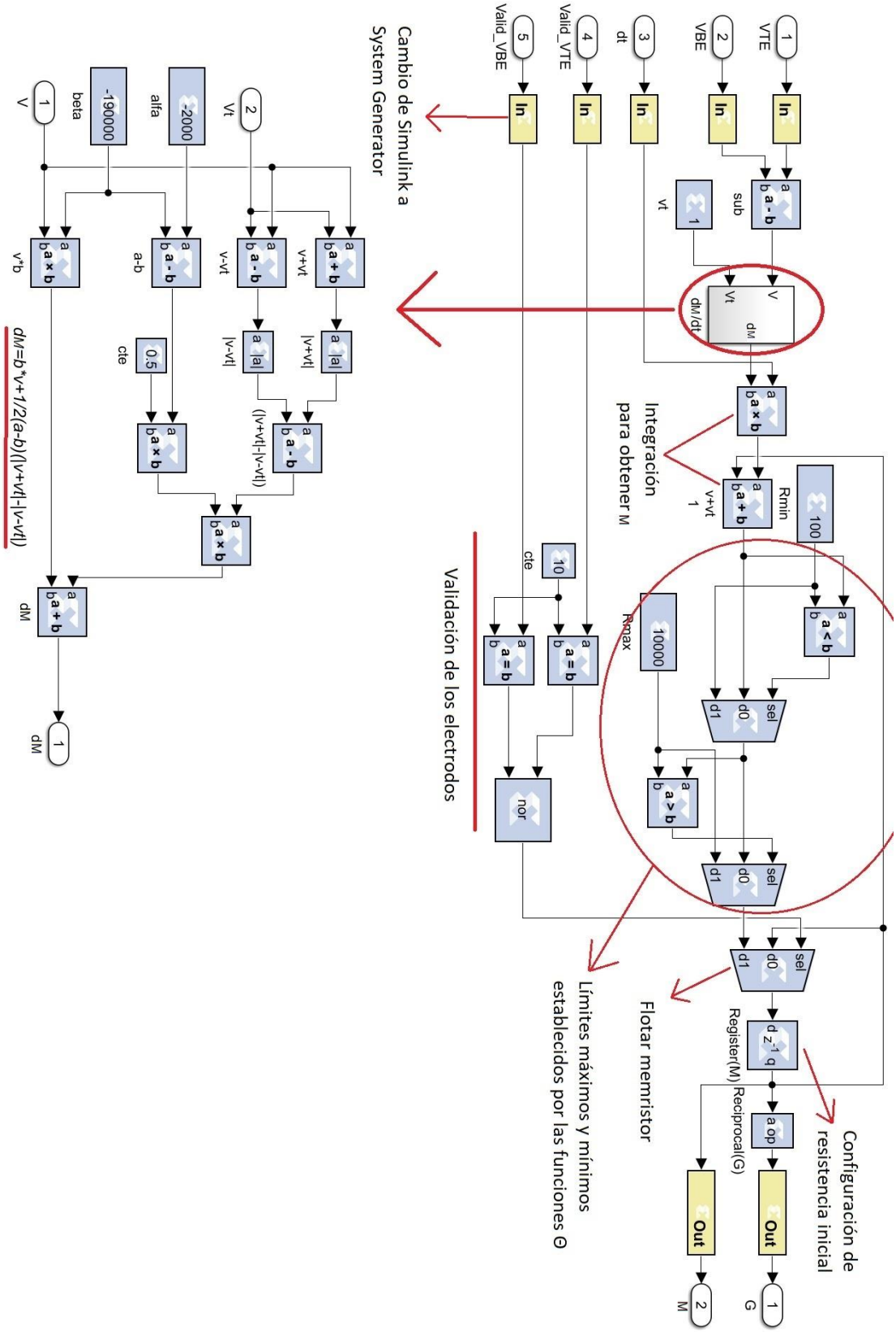

Figura 4.6 Memristor en System Generator.

Ya completado el bloque en System Generator, se procede a la conversión de datos (M, G) a Simulink para poder visualizar los resultados obtenidos del FPGA en la computadora.

Después de programar el FPGA, se hace uso del diagrama de la *Figura* 4.5 donde solo hay que cambiar el estímulo enviado al FPGA para realizar distintas pruebas que se deseen. Es importante mencionar que solo los estímulos y el $\Delta \boldsymbol{t}$ pueden ser modificados, ya que tanto las constantes internas, como la resistencia inicial solo pueden ser modificadas al generar otro código de programación.

Para el primer ejemplo, se aplica una señal triangular con una amplitud que va desde -2v a 2v, distribuida en 4000 pasos de simulación, la *Figura* 4.7 muestra la señal aplicada y los resultados de la implementación.

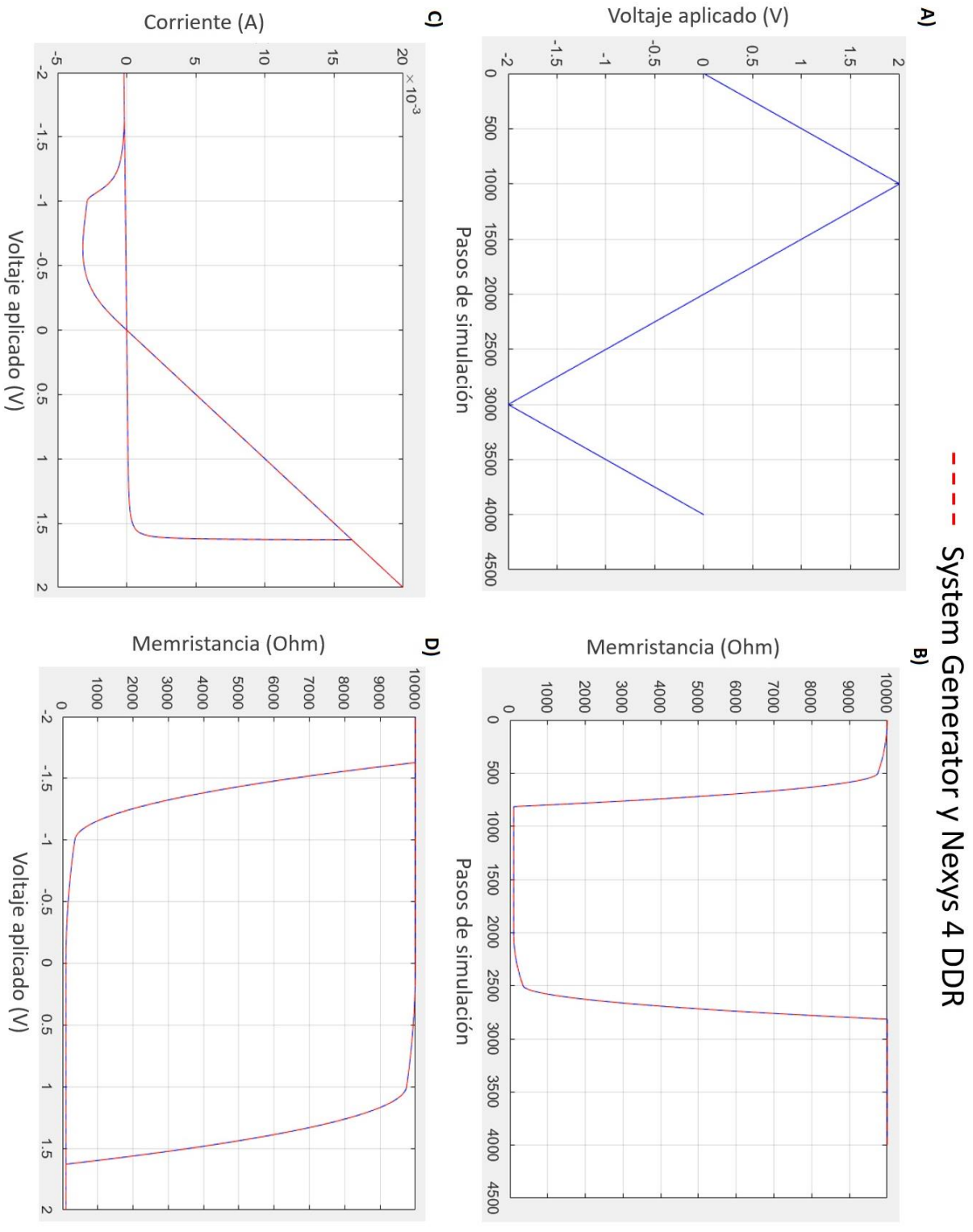

Figura 4.7 A) Voltaje aplicado, B) cambio de memristancia obtenido en Simulink, y en co-simulación con el FPGA, C) cambio de la corriente dependiente del voltaje aplicado, D) cambio de memristancia dependiente del voltaje.

En la *Figura* 4.7A, se aprecia el voltaje aplicado (señal triangular) tanto para la simulación en Simulink, como para la implementación y co-simulación en el FPGA. Se puede observar en la *Figura* 4.7B, como al polarizar de manera directa, la memristancia comienza a disminuir hasta llegar al mínimo establecido, este valor se mantiene hasta que el voltaje aplicado al memristor es invertido. También se puede apreciar un cambio pequeño antes del umbral V_T y un cambio mayor después de este, que está dado por las constantes a y b.

En la *Figura* 4.7C se observa el clásico cruce por cero centrado en el origen para la corriente y voltaje, en azul los resultados con Simulink y en Rojo con líneas discontinuas los resultados en el FPGA, aparentemente no hay diferencia por lo que el resultado obtenido es bastante bueno.

Por último, en la *Figura* 4.7D se aprecia una gráfica de la variación de la memristancia con respecto al voltaje aplicado (positivo incremental), donde es fácil identificar el pequeño cambio antes del umbral V_T, mientras que después de este umbral, el cambio es más significativo llevando la memristancia hasta el mínimo posible llamado Ron. Si el voltaje comienza a disminuir, la memristancia no varía hasta que el voltaje sea invertido, provocando un aumento de memristancia hasta el valor máximo conocido como Roff.

Este ejemplo, comprueba el funcionamiento del modelo del memristor con umbral controlado por voltaje, por lo que se procede a cambiar los estímulos aplicados. Ya que se conoce que un memristor debe contar con 3 fingerprints: 1) Histéresis con cruce por cero centrada en el origen, 2) cierre de los lóbulos conforme aumenta la frecuencia, 3) La capacidad de ajuste muti-estado proporcionada debido a la memoria. Entonces se cambia la señal de excitación triangular a una del tipo seno expresada como: $\boldsymbol{v(t) = 3\sin(2\pi ft)}$, manteniendo los mismos parámetros del memristor del ejemplo anterior. Los resultados obtenidos en co-simulación, así como en Simulink, pueden ser observados en la *Figura* 4.8, *Figura* 4.9 y *Figura* 4.10.

En la *Figura* 4.8 se tiene A) simulación, B) implementación, observando la clásica histéresis con cruce por cero en el origen, así como el cierre de esta al aumentar la frecuencia (12, 24 y 48 Hz). La señal de voltaje aplicada es $\boldsymbol{v(t) = 3\sin(2\pi ft)}$. En este ejemplo se verifican dos fingerprints, histéresis con cruce en el origen y cierre de esta al aumentar la frecuencia, además de que se comprueba que no hay diferencia entre la simulación (*Figura* 4.8 A) y la implementación (*Figura* 4.8 B)

z

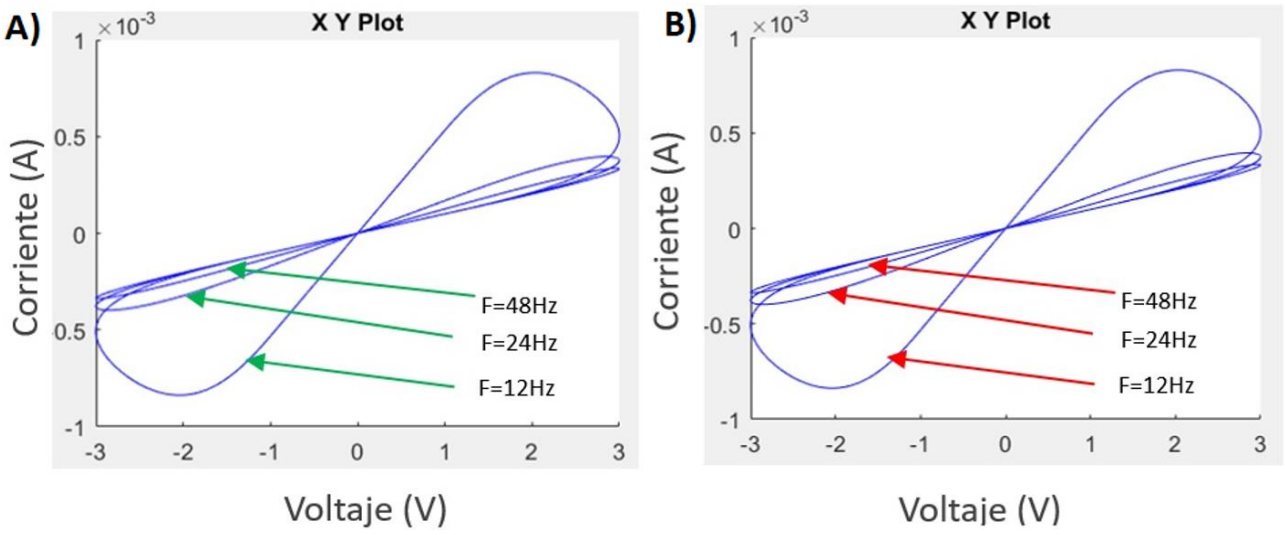

Figura 4.8 Gráficos I-V A) Simulink, B) System Generator y Nexys 4DDR Artyx-7 en co-simulación.

En la *Figura* 4.9, se observan las gráficas A) simulación, B) implementación, del cambio de memristancia dependiendo del voltaje aplicado. Al inicio la resistencia interna se configuro en la máxima posible (10 KΩ), y al tratarse de un modelo decremental, la memristancia comienza a disminuir al polarizar directamente cuando se supere el umbral (V_T); al disminuir el voltaje aplicado por debajo del umbral, la memristancia deja de cambiar, pero al polarizar inversamente y sobrepasar el umbral nuevamente, esta comienza a incrementar hasta llegar a su valor máximo posible.

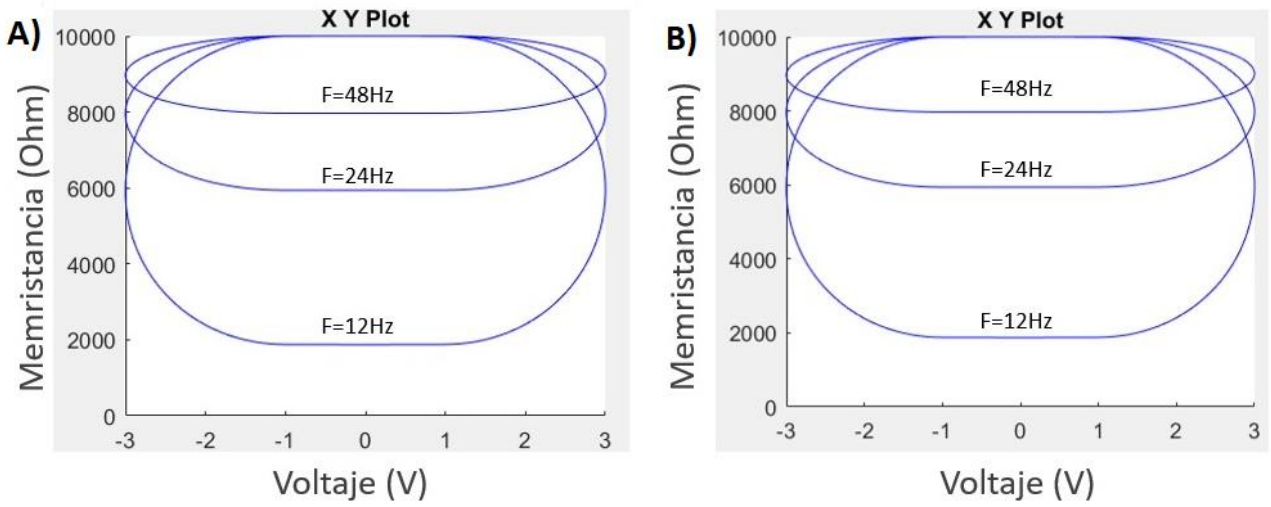

Figura 4.9 Memristancia vs Voltaje A) Simulink, B) System Generator, para distintas frecuencias.

También se observa porqué la histéresis comienza a disminuir al aumentar la frecuencia. Esto es debido a que al ser una señal de voltaje tipo senoidal, el tiempo en que se polariza el memristor directa o inversamente está determinado por la frecuencia, si esta es muy rápida el memristor no tiene suficiente tiempo como para que los parámetros (α, β) afecten de manera significativa a la ecuación (4.3), de esta manera el cambio que pueda tener entre sus valores de memristancia será menor. En el caso de una alta frecuencia, el valor de la memristancia tenderá a ser mono-valuado semejante a una resistencia.

Algo similar ocurre con el voltaje aplicado (*Figura* 4.10); siendo la misma señal de excitación, pero con una amplitud menor la histéresis disminuye, esto debido a que el cambio significativo se produce al superar el umbral, y si la diferencia entre la señal aplicada y el umbral es pequeña, el cambio que sufre la memristancia también es menor.

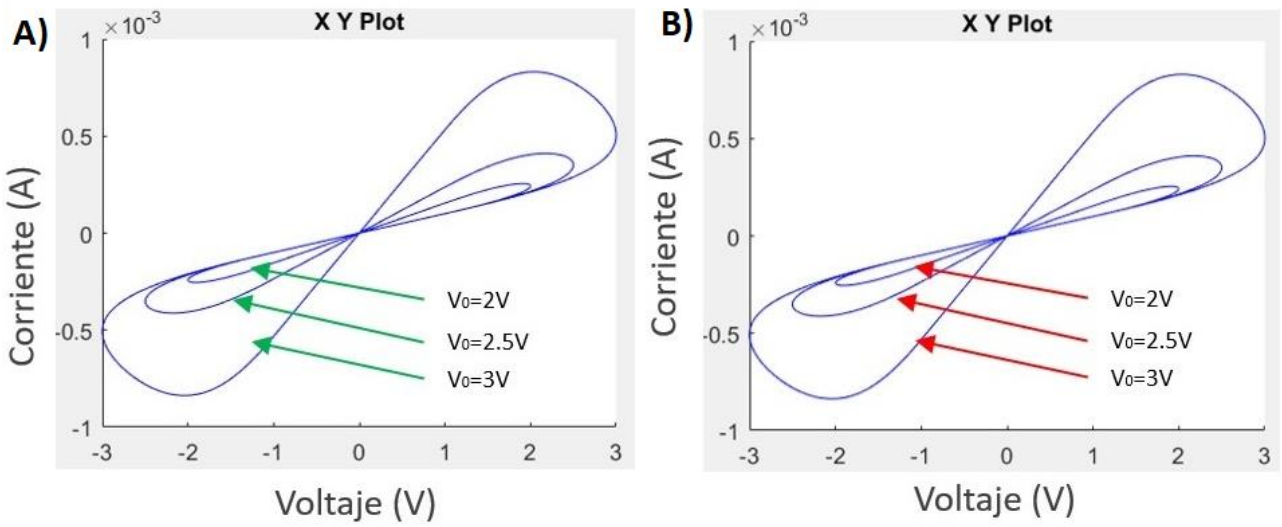

Figura 4.10 Gráficos I-V, A) Simulink, B) System Generator Nexys 4 DDR Artyx-7, para distintas amplitudes de la señal seno donde se observa la disminución de la histéresis a valores cercanos al umbral V_T.

Ahora, para verificar la capacidad de ajuste multi-estado del memristor, se cambia la señal de voltaje aplicada por $\boldsymbol{v(t) = \pm 3 sin^2(100\pi\, t)}$**.** Como se observa en la *Figura* 4.11 A, la señal de voltaje oscila en la parte positiva como en la parte negativa. Los datos en azul son los obtenidos en la simulación con Simulink, mientras que la línea roja punteada son los datos generados por el FPGA en co-simulación.

En la *Figura* 4.11 B, se comprueba la capacidad del memristor de mantener un valor de memristancia interna cuando cesa la aplicación de un voltaje. Siendo un memristor decremental, la oscilación de la parte positiva de la señal de voltaje, hace que la memristancia disminuya; cuando se está por debajo del umbral o en 0v la memristancia no cambia y se mantiene hasta el siguiente ciclo, disminuyendo cada vez más. Una vez que se polariza inversamente el memristor, se puede apreciar como la memristancia comienza a incrementarse y de la misma manera en los intervalos en los que el voltaje aplicado es menor al umbral, la memristancia se mantiene.

En la *Figura* 4.11 C, se aprecia la histéresis que es generada por los cambios de memristancia; recordando que en polarización directa esta disminuye, los lóbulos generados son únicamente los del cuadrante positivo, mientras que su contraparte en el cuadrante negativo es generada al polarizar inversamente. Para entender por qué es que van cambiando de tamaño los lóbulos, hay que recordar que el memristor es inicializado en su valor máximo (10KΩ) por lo que, si bien el cambio siempre es el mismo, no lo es la corriente que se genera al estar disminuyendo la memristancia. Entonces, el lóbulo más cerrado positivo, corresponde al primer cambio de memristancia, mientras que el mayor positivo y negativo corresponden al cambio de memristancia cuando ésta se encuentra más cerca de su valor mínimo, lo que provoca una corriente mayor.

Por último, en la *Figura* 4.11 D, se aprecia el cambio de memristancia vs el voltaje aplicado. En esta gráfica, es evidente que el cambio de memristancia es el mismo, siempre siendo significativo al superar el umbral y se mantiene en un valor menor a este. Las flechas verdes indican el camino que sigue la memristancia; al inicio comienza disminuyendo hasta que la señal de excitación se vuelve negativa; después de esto, la memristancia comienza a subir de manera similar a como disminuyó.

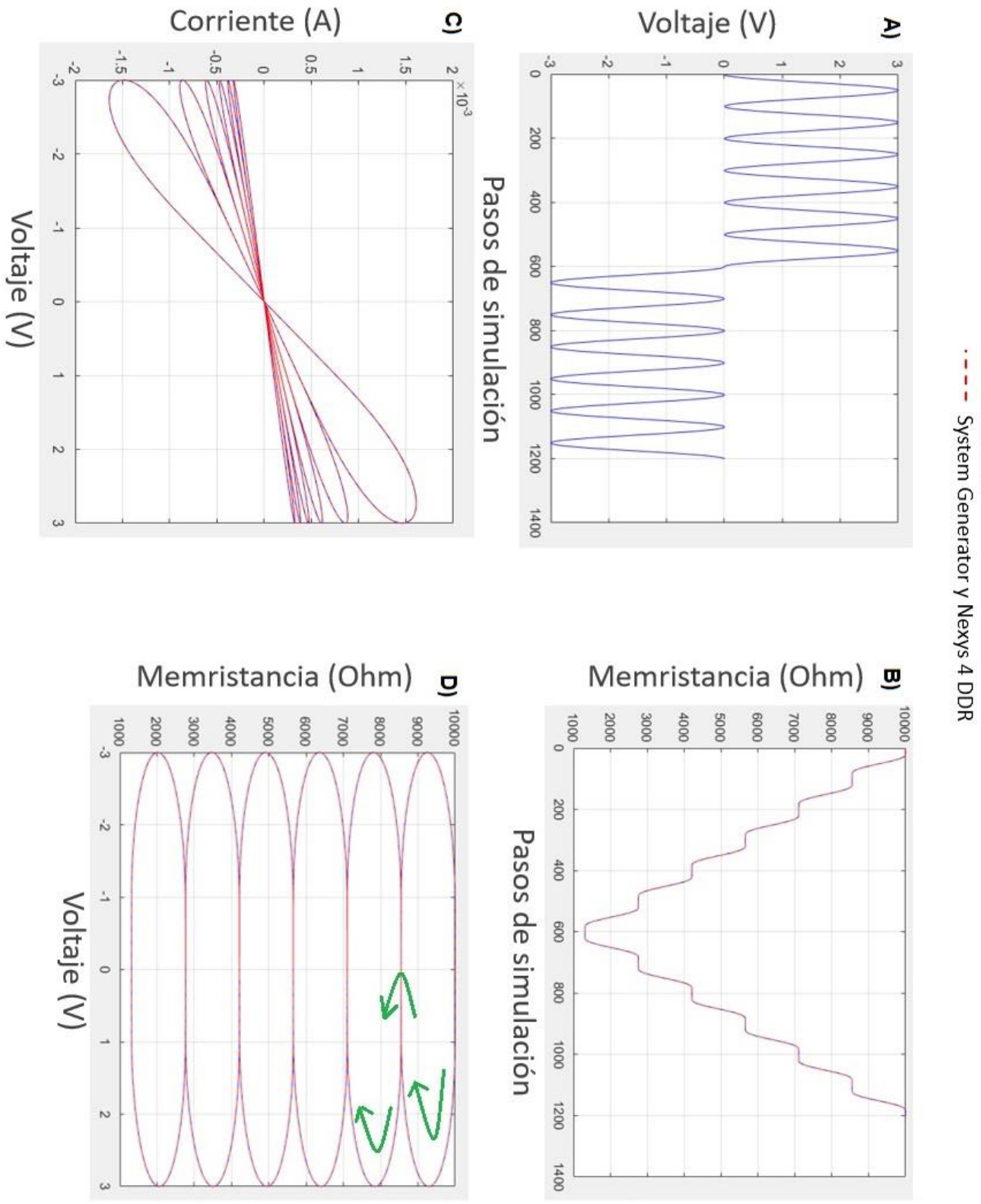

Figura 4.11 Comparación entre Simulink e implementación con System Generator y el FPGA. A) una señal $v(t) = \pm 3sin^2(100\pi\, t)$ aplicada*, B) la capacidad de ajuste multi-estado del memristor, C) mostrando el clásico cruce por cero de la histéresis, D) el cambio de memristancia dependiente del voltaje aplicado.*

Entonces, ya que el modelo del memristor no cambió y solo lo hicieron las señales de voltaje, los recursos de esta implementación se muestran en la *Tabla* 4.1.

Recursos Memristor	Nexys 4DDR Artix-7 Punto flotante a 32bits.	Nexys 4DDR Artix-7 Punto fijo a 18bits.
DSP's	20(8.3%)	18(7.5%)
LUT's	3141(5%)	3487(5.5%)
Registros	38(0.5%)	76(0.5%)

Tabla 4.1 Recurso utilizados por la implementación del módulo memristor en el FPGA

Teniendo en cuenta que los recursos son limitados, y que solo se cuenta con 240 (DSP's), 63 400 (LUT's), y 126 800 (Registros) disponibles, es que se decidió cambiar la resolución de 32 bits en flotante a 18 bits en punto fijo, esto no repercute en la obtención de los datos, ya que los valores máximos y mínimos a alcanzar por el memristor se encuentran entre 100 y 10 000 fue posible realizar esta optimización de recurso. Esto es muy importante, ya que se desea implementar una red neuronal pulsada en el FPGA de manera paralela, como en un sistema biológico. Entonces debido a que la disponibilidad de los multiplicadores (DPS's) es la mayor limitante se optó por realizar una disminución de los recursos.

Por último, se realizó una prueba para verificar la posibilidad de conectar el módulo memristor con otros elementos, en este caso uno de su misma clase como se observa en la *Figura* 4.12, donde un memristor es contactado en serie con otro, pero de manera opuesta y con las resistencias iniciales opuestas.

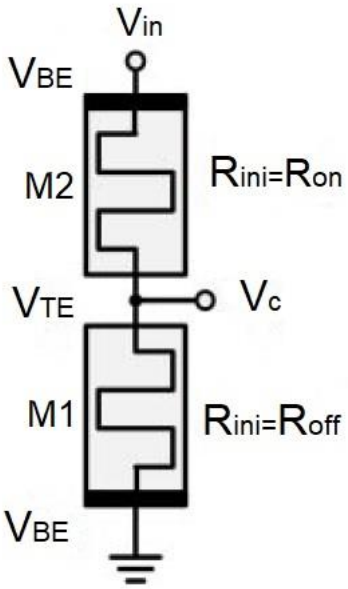

Figura 4.12 CRS (switch resistivo complementario) en anti serie.

Esta configuración es conocida como CRS, existiendo más posibilidades de conexión, como la antiparalelo, serie, paralelo, pudiendo iniciar la resistencia interna del memristor en varios estados, es posible generar varios experimentos a implementar. En este caso, se verifica por medio de simulación e implementación la configuración mostrada en la *Figura* 4.12 y *Figura* 4.13.

Al tratarse de una implantación digital, es necesario conocer el valor del voltaje presente entre los dos memristores, por lo que un divisor de voltaje se encuentra señalado en rojo en la *Figura* 4.12, calculando el voltaje en ese nodo. Entonces, siendo que se encuentran en anti serie (electrodos opuestos en serie), al polarizarse uno en directa, el otro memristor se encontrará polarizado en inversa; al cambiar de polaridad la señal se dará una situación similar, pero con los casos invertidos.

El voltaje aplicado es una señal de tipo triangular, con una amplitud de ±2.5V como se observa en la *Figura* 4.14; las líneas roja y azul corresponden a los voltajes en los memristores 1,2 respectivamente, los valores de los memristores siguen siendo los mismos que en los ejemplos anteriores, a excepción de dt=0.0005s.

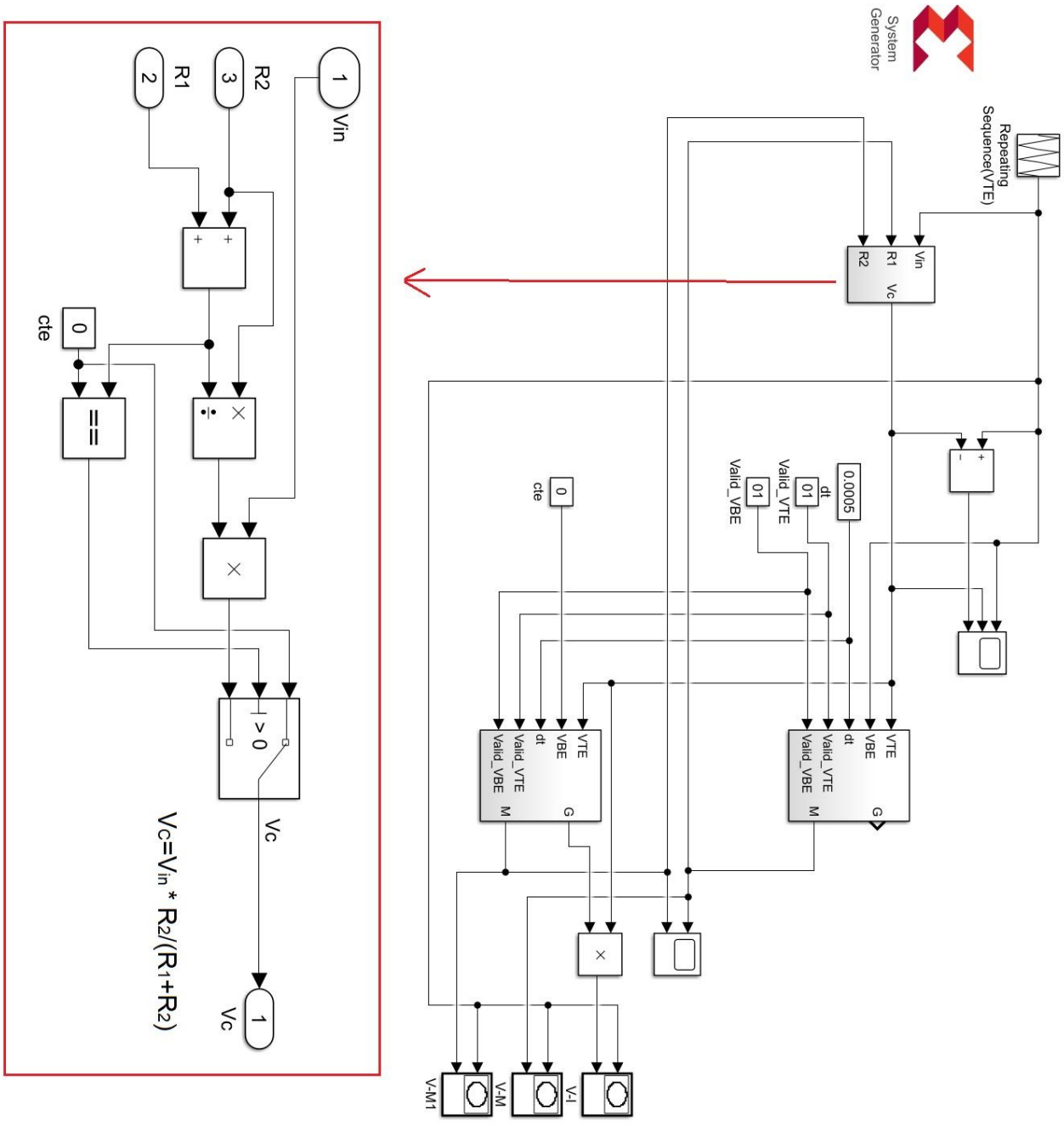

Figura 4.13 Implementación CRS en System Generator y co-Simulación con ayuda de Simulink.

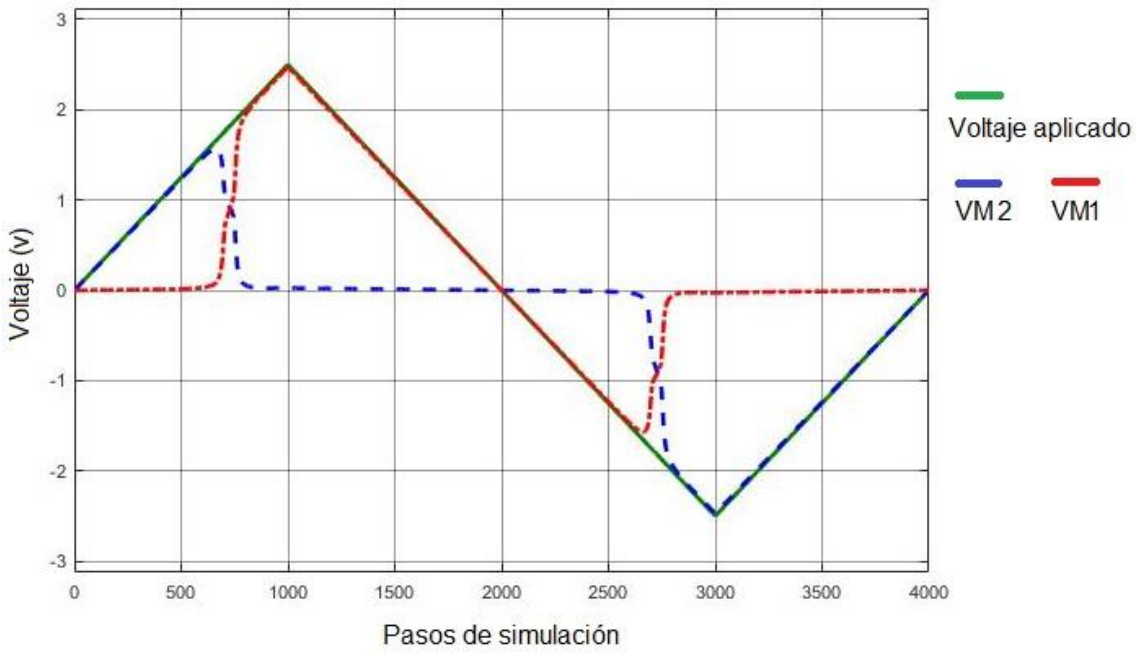

Figura 4.14 Voltaje aplicado (verde), en el memristor 1 (rojo), y en el memristor 2(azul)

En la *Figura* 4.14, se observa que al inicializarse el memristor M2 con Ron(Rmin), con los electrodos opuestos, y el memristor M1 con la una Roff(Rmax), el voltaje al principio está concentrado en M2(10KΩ), pero conforme aumenta la amplitud de la señal aplicada, M1(100Ω) comienza a aumentar su memristancia, así como M2 comienza disminuir, llegando un punto en el que se intercambian; teniendo en M2 el valor menor(100Ω) y en M1 el valor mayor(10KΩ), concentrando así el voltaje en M1. Cuando la señal pasa por la parte negativa este proceso se repite, pero de manera opuesta; alternando así los valores de Rmin y Rmax en los memristores.

Para comprender como es que el voltaje se concentra en un memristor y después en otro, se presenta la *Figura* 4.15, donde se observa como en un principio las resistencias internas de M1 y M2 se encuentran opuestas, conforme la señal de voltaje aumenta uno se polariza directo(M2) y el otro en inverso(M1); esto provoca un crecimiento de memristancia en M1 y una disminución en M2, al invertir la señal los valores de las memristancias se encuentran opuestos, haciendo que ahora en M1 disminuya la memristancia, mientras que en M2 aumente.

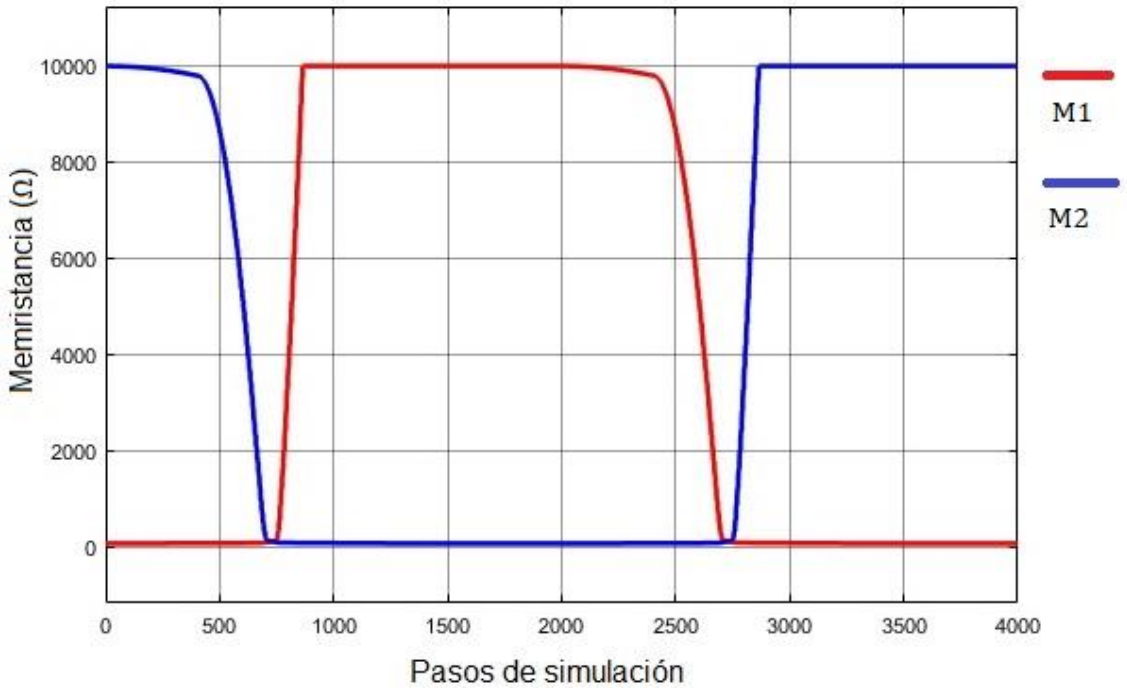

Figura 4.15 Intercambio de memristancia M1(rojo), M2(azul).

Lo anterior provoca que el voltaje se concentre en el memristor con mayor valor de memristancia, por eso es que los voltajes también se encuentran alternando entre ambos memristores.

En la *Figura* 4.16, se observa que un cambio de corriente significativo es apreciable solo cuando los memristores alternan su memristancia; en el resto de la gráfica se comportan como una resistencia.

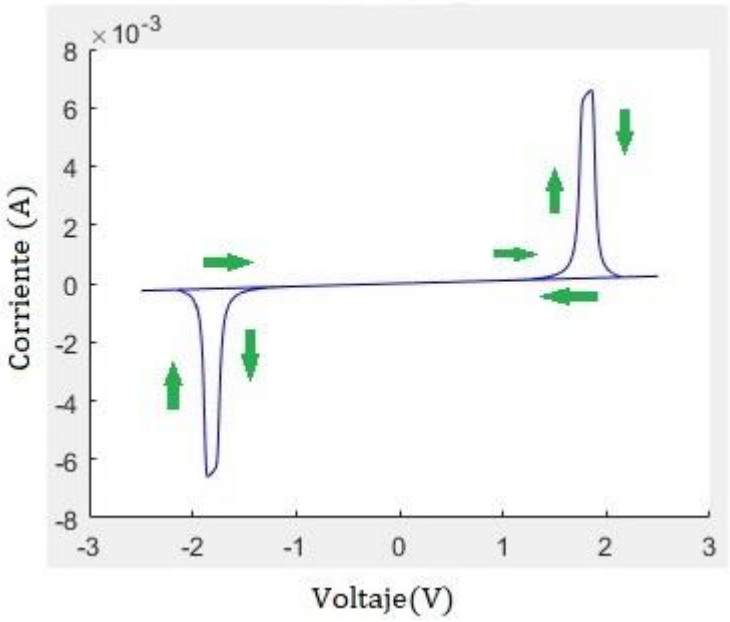

Figura 4.16 Gráfica I-V para el CRS.

Los recursos utilizados solo para implementar los memristores, sin incluir el módulo que calcula el voltaje en el nodo central, están en la *Tabla* 4.2.

Recursos	Nexys 4DDR Artix-7 Punto fijo a 18bits.
DSP's	36(15%)
LUT's	6974(9.9%)
Registros	152(0.5%)

Tabla 4.2 Recursos utilizados por el CRS.

Una vez comprobado el correcto funcionamiento de la implementación en FPGA del módulo del memristor digital con umbral controlado por voltaje, se procede al diseño de una neurona pulsada.

4.4 Implementación de una Neurona Pulsada

Corroborado el funcionamiento del memristor y conociendo los recursos que este utilizara del FPGA, es necesario optar por un modelo de neurona pulsada que pueda ser optimizado, y no consuma demasiados recursos como se hizo en la implementación del memristor. Para este caso, se utilizará un modelo simplificado, que consta de las conexiones observadas en la *Figura* 4.17.

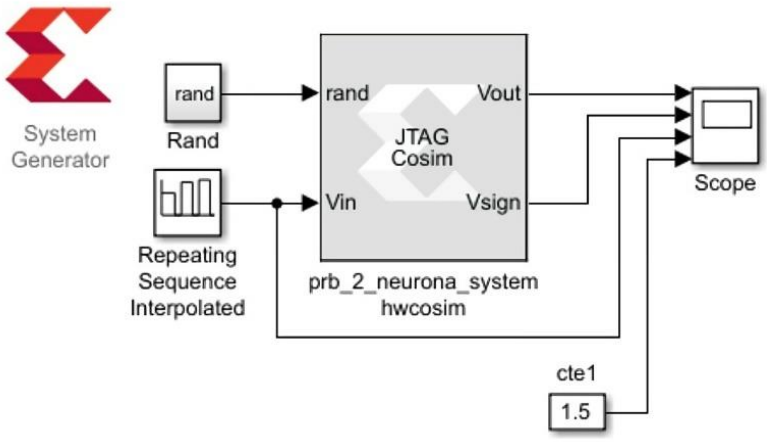

Figura 4.17 Modulo de una neurona pulsada en System Generator a implementar en FPGA.

Como se mencionó en el capítulo 2, existen varios tipos de neuronas las cuales pueden ser simuladas con distintos modelos, pero todas tienen en común tres cosas:

a. Disparan un pulso al superar un determinado umbral.
b. Cuentan con un periodo refractario en el cual no pueden disparar.
c. La forma de los pulsos no lleva información, sino que está determinada por el espacio y rapidez de estos (trenes de pulsos), aunque la separación entre cada pulso no es la misma siempre.

Teniendo en cuenta esto, y dado que un solo modulo del memristor ya ocupa recursos significantes (18 DSP's de 240), se optó por la propuesta presentada en la ref. [3], donde se muestra una neurona pulsada bio-inspirada, la cual está simplificada sin perder alguno de los puntos anteriores.

Dicho modelo se puede observar en la *Figura* 4.18. Para este caso, V_{IN} es el voltaje de entrada a la neurona y si este supera un umbral V_T definido internamente, la neurona comenzará a pulsar siendo reflejado en V_{OUT}. La separación entre los pulsos de salida está definida por la ecuación (4.4), según la ref. [5].

$$\delta t = \tau - \gamma * (V_{in} - V_T) + \lambda(\eta - 5) \qquad (4.4)$$

Donde η es un número aleatorio entre 0-10, λ es un múltiplo que indica cuanto afecta este valor al cálculo del espacio entre pulsos, dando un espacio diferente entre cada pulso generado, pero sin afectar al tren de pulsos en general, algo que es común en los sistemas biológicos, V_T es el umbral a superar, τ es un tiempo total a esperar que puede ser representado en ms y γ es un múltiplo de la diferencia entre V_T y V_{IN}, por lo que una mayor diferencia entre esto, implica que la separación entre cada pulso será menor.

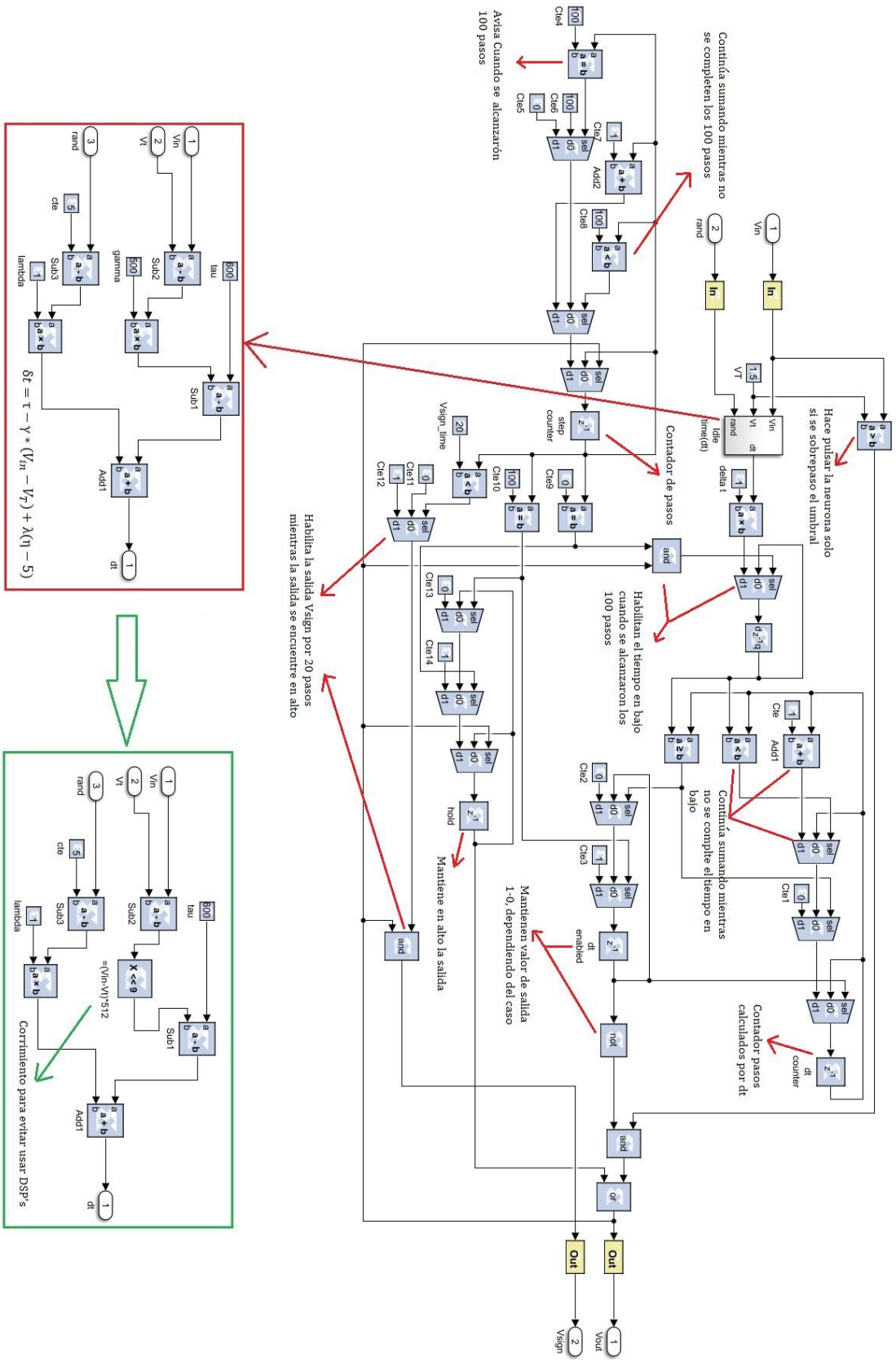

Figura 4.18 Diagrama interno de una Neurona pulsada.

La operación de la neurona está determinada por los contadores que definen el tiempo de excitación de cada pulso, así como la separación entre estos. Para el tiempo en alto se proponen 100 pasos (ciclos de reloj) como se observa en la *Figura* 4.18, que son obtenidos por medio del sumador y el registro indicado. En cuanto al tiempo en bajo, está determinado por la función dt, este valor depende directamente de la diferencia entre el umbral V_T y el valor de entrada de voltaje V_{IN}; una vez calculado, se guarda en un registro y un contador comienza su función hasta el valor indicado, posteriormente la salida en alto es habilitada y el proceso comienza de nuevo.

Adicionalmente, se agrega una salida V_{SIGN} que se habilita durante el tiempo en alto de Vout, durante un tiempo de 20 pasos, la cual será explicada y utilizada posteriormente para la implementación de la red neuronal pulsada.

Una vez construido el módulo de la neurona, este fue simulado y posteriormente implementado en el FPGA. Dado que en el memristor se comprobó que no existe aparente diferencia entre las simulaciones y las implementaciones, solo se mostrarán los resultados de la co-simulación con la placa Nexys 4DDR Artix-7.

En la *Figura* 4.19 se observa una señal del voltaje de entrada (rojo), la cual es utilizada para excitar la neurona, cuando un voltaje de umbral (azul) es configurado en 1.5V.

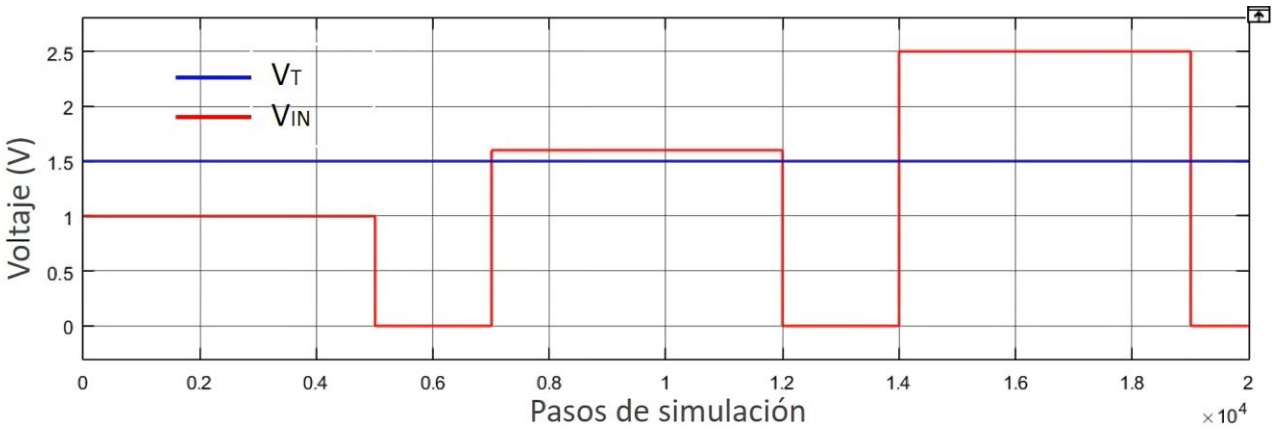

Figura 4.19 Voltaje de entrada (rojo), comparado con el voltaje de umbral establecido (azul)

Mientras que en la *Figura* 4.20 se aprecia cómo es que si la señal de excitación no supera el umbral, la neurona no dispara. Una vez que el voltaje supera el umbral, la neurona comienza a emitir pulsos con una separación determinada entre ellos; cuanto mayor sea la diferencia positiva entre el umbral y el voltaje de entrada, menor será el tiempo entre cada pulso.

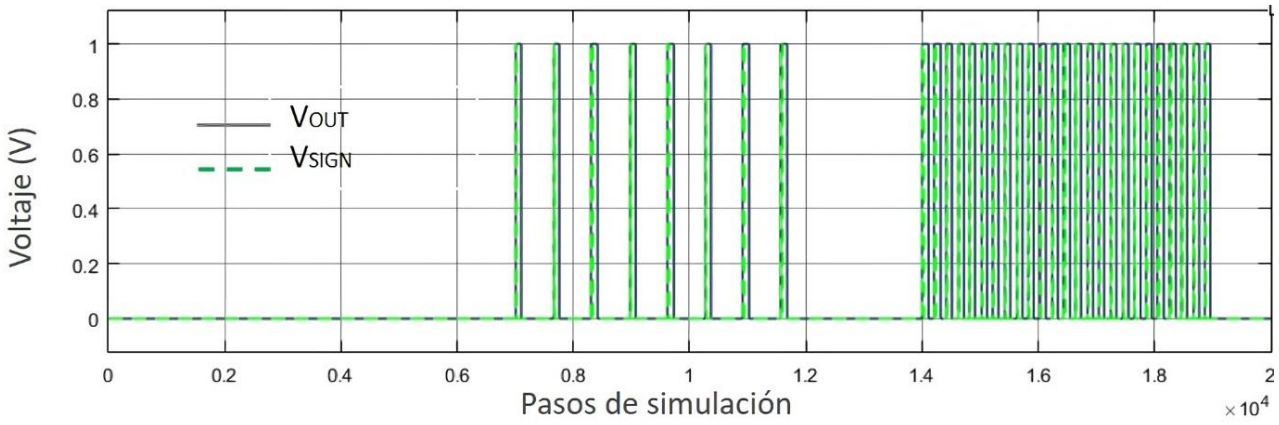

Figura 4.20 Resultados obtenidos de la implementación del módulo neurona en el FPGA

La *Figura* 4.21, es una ampliación del primer pulso generado que se observa en la *Figura* 4.20, donde ahora es posible distinguir el V_{OUT} en 100 pasos efectivamente, así como el V_{SIGN} en 20 pasos.

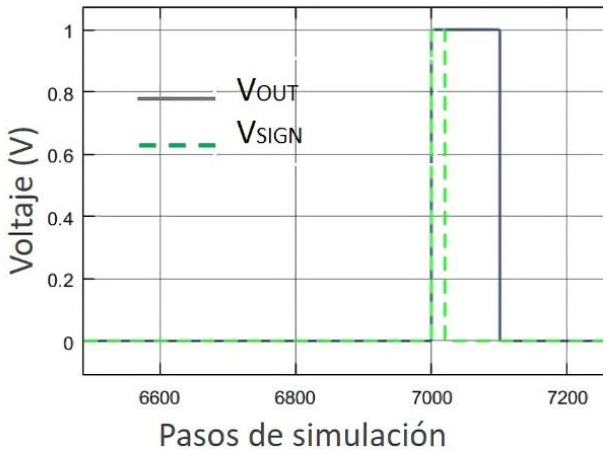

Figura 4.21 Ampliación para poder observar V_{SIGN} y V_{OUT}

Es importante mencionar que esta neurona está diseñada de tal manera que solo puede recibir voltajes de entrada máximos hasta 2.5V, ya que a un valor mayor el tiempo calculado en bajo para la separación entre cada pulso es tan pequeño que comienza a volverse inestable debido al random que incluye la función dt; los valores utilizados fueron $\tau = 600ms,\ \gamma = 500\frac{ms}{V}\ V_T = 1.5V,\ \lambda = 1ms$ y los recursos consumidos por esta implementación, son mostrados en la *Tabla* 4.3.

Recursos Neurona	Nexys 4DDR Artix-7 Punto flotante a 32bits.	Nexys 4DDR Artix-7 Punto fijo a 18bits.
DSP's	9 (4%)	0 (0%)
LUT's	2814 (4.5%)	190 (0.5%)
Registros	98 (0.5%)	52 (0.5%)

Tabla 4.3 Recursos utilizados por la neurona implementada en el FPGA.

Donde se puede ver que fue realizada una optimización de recursos, al cambiar de 32 bits en flotante, a 18 bits en punto fijo; este cambio también se observa en la *Figura* 4.18 donde se utilizó un corrimiento de bits para evitar una multiplicación. Gracias a esto se logró reducir considerablemente los recursos utilizados por neurona pulsada.

4.5 Implementación SNN: Experimento Pavlov's dog

Ya que se han construido, simulado e implementado los módulos del memristor y la neurona pulsada, es posible recrear el famoso experimento conocido como "Pavlov's dog", donde se hará uso del concepto "memoria asociativa"; este experimento puede ser resumido en la *Figura* 4.22.

En este experimento, Pávlov sometió a un perro a diferentes estímulos, intentado asociarlos. Para comenzar se le presentaba al perro un estímulo visual (comida), y se observa como el sujeto comienza a salivar, posteriormente se retiraba el estímulo y se cambiaba por el sonido de una campana, aquí se observaba que el perro no presentaba salivación.

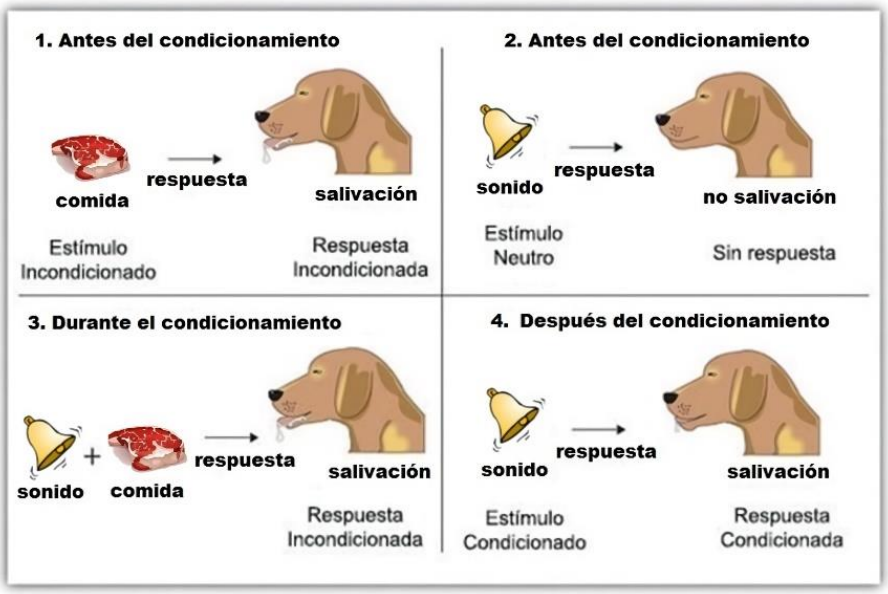

Figura 4.22 Experimento Pavlov's dog.

El objetivo de presentar el estímulo visual junto con el sonoro durante un cierto periodo de tiempo para que el perro asocie el sonido de la campana con la comida. Una vez realizado esto, al presentar solo el sonido de la campana el perro saliva, no con la misma intensidad, pero se ha logrado que lo asocie con la comida.

En la *Figura* 4.23 se presenta la red neuronal a implementar, donde se aprecian 3 neuronas, dos de entrada (comida y sonido) y una de salida para la salivación, donde los memristores trabajarán como la sinapsis entre las neuronas. Además, se observa una resistencia con una conductancia G_0, debido a que, al tratarse de una implementación digital es necesario conocer el valor del voltaje en cada nodo.

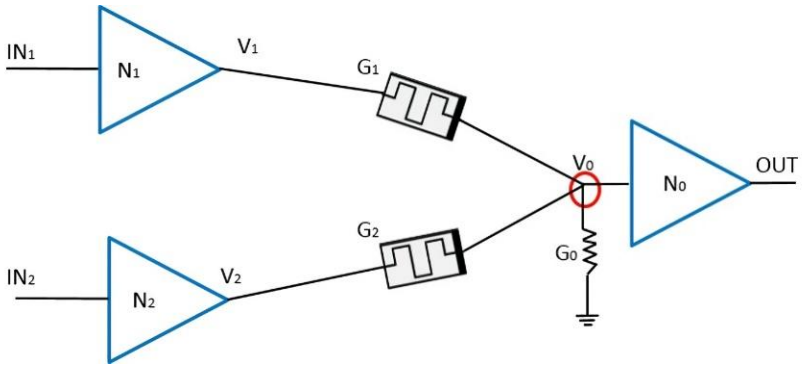

Figura 4.23 Red neuronal a construir.

Recordando que la salida (pulsos) de una neurona es un valor de voltaje binario (0-1), este solo indica si un memristor aporta algo a la siguiente neurona, por lo que es necesario contar con un módulo que calcule el valor del voltaje a ingresar a la neurona de salivación; dicho dispositivo es conocido como KCL (Kirchhoff's current law), el cual hace uso de la ecuación (4.5).

$$V_0 = \frac{V_{pulse}(V_1G_1 + V_2G_2 + \ldots + V_nG_n)}{G_0 + G_1 + G_2 + \cdots + G_n} \qquad (4.5)$$

$$v_0 = V_{pulse}\frac{G_x}{G_y}$$

La implementación de la ecuación (4.5), es mostrad en la Figura 4.24, donde de ser necesario es posible expandir este módulo, para agregar más neuronas. G_0 es determinada en 0.0001 para no afectar el cálculo del voltaje de entrada, y dado que se supone que los estímulos de las neuronas de comida y de sonido están siendo excitadas con 2.5V, este valor se usa como V_{pulse}.

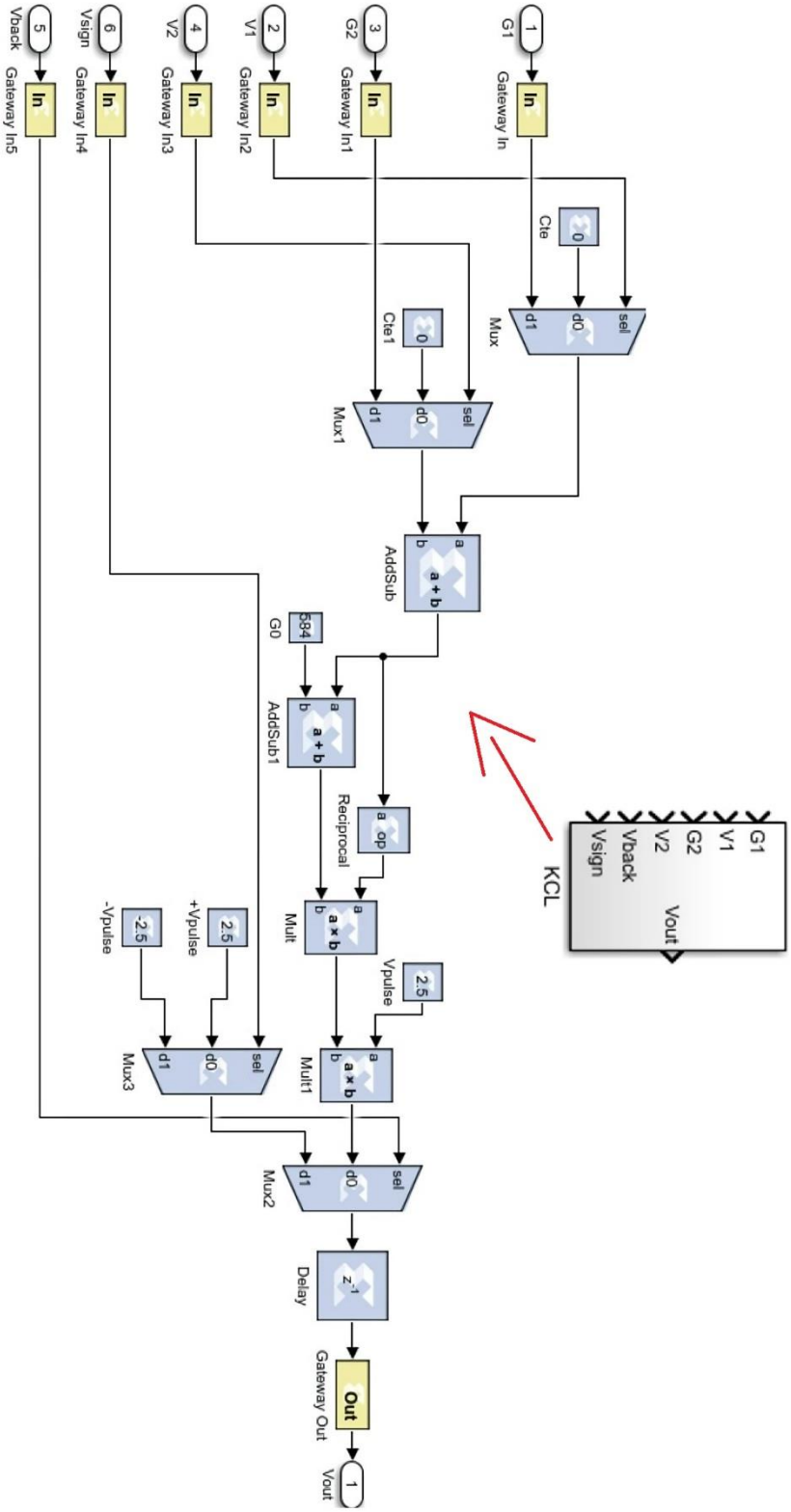

Figura 4.24 KCL usado para calcular el voltaje de entrada en la neurona de salida.

Cuando las neuronas de entrada se encuentran pulsando, los memristores reciben V_{IN} en su terminal V_{TE} y es necesario calcular V_0; cuando las neuronas de entrada se encuentran inactivas los memristores están flotados, lo que se realiza con un módulo de control mostrado en la *Figura* 4.25, haciendo invalida la terminal Valid_VTE; en cambio cuando, la neurona de salida dispara significa que los memristores tienen un potencial en su terminal V_{BE}, lo que significa que hay pulsos de retro-propagación, los cuales serán positivos o negativos dependiendo de V_{sign}, la cual a su vez es una amplitud fija de estos pulsos; para indicar si están presentes se hace uso de $V_{back,}$ que corresponde a la salida de la neurona de salivación.

De esta manera un entrenamiento similar al STDP puede lograrse, con ayuda del pulso V_{sign}, por lo que el memristor podrá ser polarizado en forma directa o inversa dependiendo en cómo se presenten los pulsos en las terminales VTE y VBE.

Entonces, ahora es posible realizar la implementación de la SNN que recreará el experimento “Pavlov’s dog”, como se observa en la *Figura* 4.25, donde las neuronas de entrada son estimuladas con una serie de pulsos. Los valores de salida modificarán la memoria del memristor, infiriendo directamente en la aportación de un pulso a la neurona de salida.

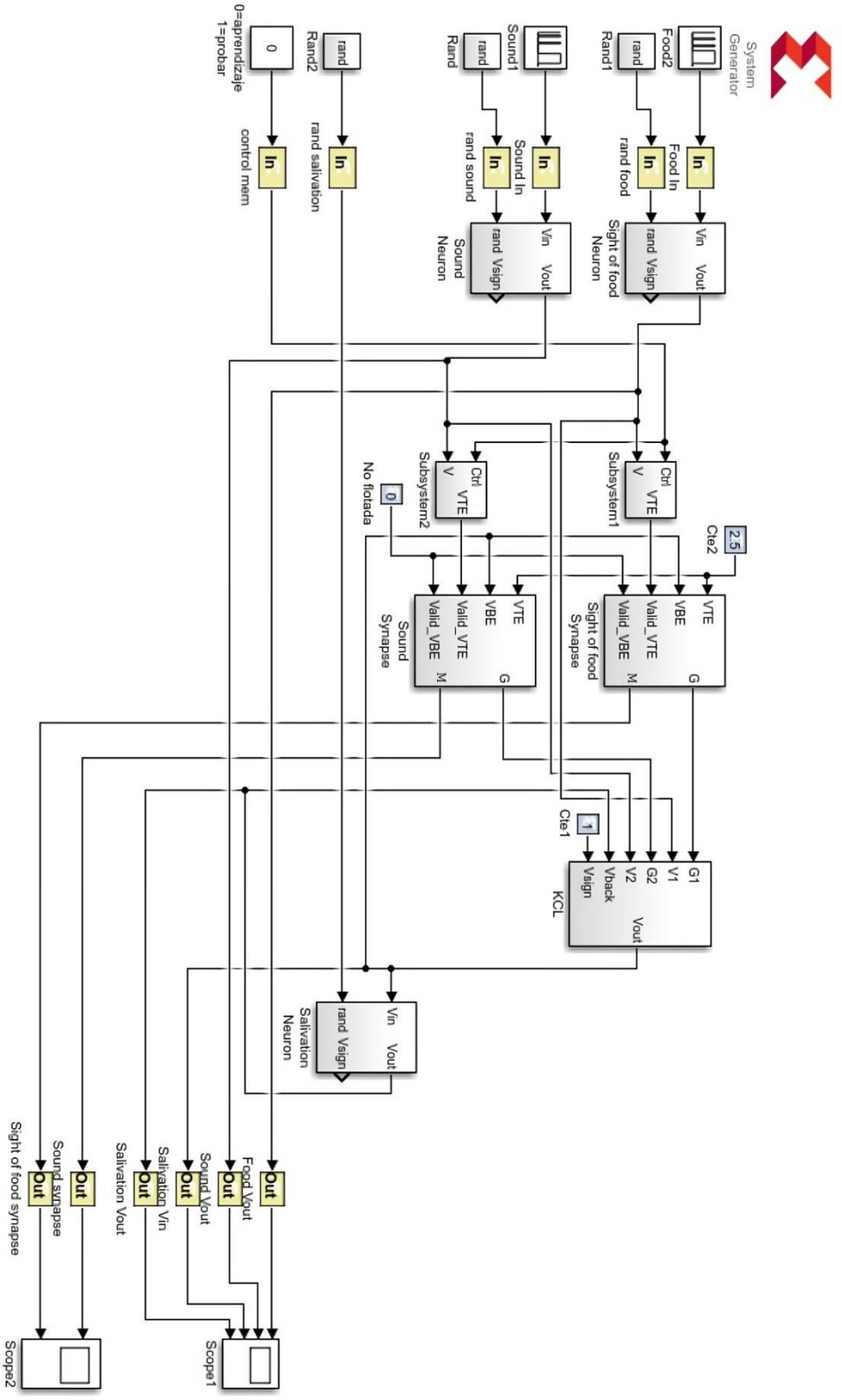

Figura 4.25 SNN reproduciendo el experimento Pavlov's dog", realizada en System Generator a implementar en el FPGA.

En este caso, el memristor asociado a la vista de comida es inicializado en un valor bajo de memristancia (675Ω), permitiendo a la neurona de salida pulsar; en cambio, el memristor del estímulo sonoro es inicializado en un nivel alto (10KΩ), evitando que la neurona de salivación pulse cuando la neurona de entrada es excitada. El V_{sign} es colocado a 1 en el módulo KCL ya que en este caso solo se requieren pulsos negativos de acción, mientras que los módulos de control están sincronizados para flotar los memristores después del entrenamiento y que no exista un cambio de memristancia.

Entonces, en esta co-simulación todo lo que se muestra entre los bloques amarrillos se encuentra implementado en el FPGA, mientras que los estímulos de las neuronas son enviados a través de la computadora con ayuda de Simulink. Los resultados obtenidos pueden ser visualizados en la *Figura* 4.26. Los parámetros utilizados para este ejemplo son: $a = 0\frac{\Omega}{V \cdot s}$, $\mathrm{b} = -15\,000\frac{\Omega}{V \cdot s}$, $\Delta t = 0.0001s$, $V_T = 4V$, $R_{min} = 675\Omega$, $R_{max} = 10K\Omega$, $R_{init\ sonido} = R_{max}$, $R_{init\ comida} = R_{min}$.

Como se observa en la *Figura* 4.26, al principio se realiza una prueba, donde se estimula la neurona del sonido (rojo), observando una pulsación en su salida, pero esto no provoca un estímulo suficientemente fuerte a la entrada de la neurona de salivación (naranja), como para que ésta (verde) pulse a su salida . Ahora, cuando la neurona de la vista de comida es estimulada, a su salida de esta también hay pulsaciones (azul), pero debido a que la sinapsis que transmite estos pulsos a la siguiente neurona tiene mayor fuerza, se puede observar a la entrada de la neurona de salivación (naranja) un estímulo suficiente para hacer pulsar su salida (verde); este proceso es repetido una vez más para corroborar que no existe cambio alguno en los memristores.

Posteriormente, es presentado el estímulo de la vista de comida haciendo pulsar la neurona de salivación; sin retirar este estimulo, es excitada la neurona de sonido durante un lapso de tiempo equivalente a $2x10^4$ ciclos de reloj, lo que modifica la memristancia asociada a la sinapsis del sonido. Una vez terminado el entrenamiento se procede a realizar los estímulos separados como en un principio.

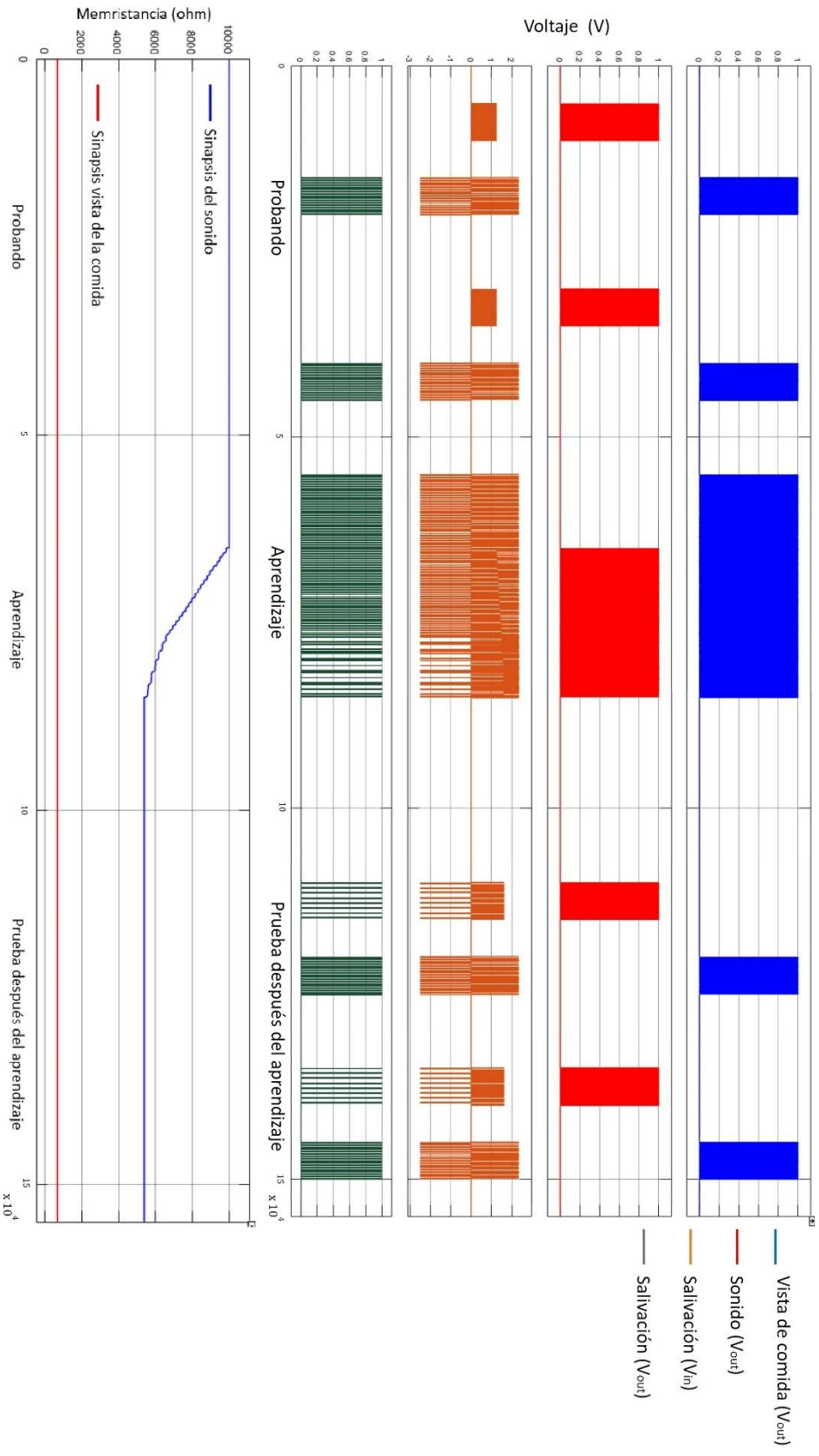

Figura 4.26 Resultados SNN implementada en el FPGA.

En esta ocasión, al haber disminuido la memristancia asociada al sonido, como se observa en la *Figura* 4.26, el estímulo que se genera en la entrada de la neurona de salivación provoca que a la salida de esta se presente una pulsación constante, no tan juntos los pulsos como al estimular la neurona de la vista de comida, pero ahora ya se ha logrado asociar un estímulo con otro. En este ejemplo, se sincronizó el control para flotar el memristor una vez terminado el entrenamiento, aunque este también se puede colocar a 1, permitiendo que la red decida cuando flotar los memristores.

Esto provoca que después del entrenamiento, la memristancia asociada a la sinapsis del sonido pueda seguir cambiando, como se muestra en la *Figura* 4.27.

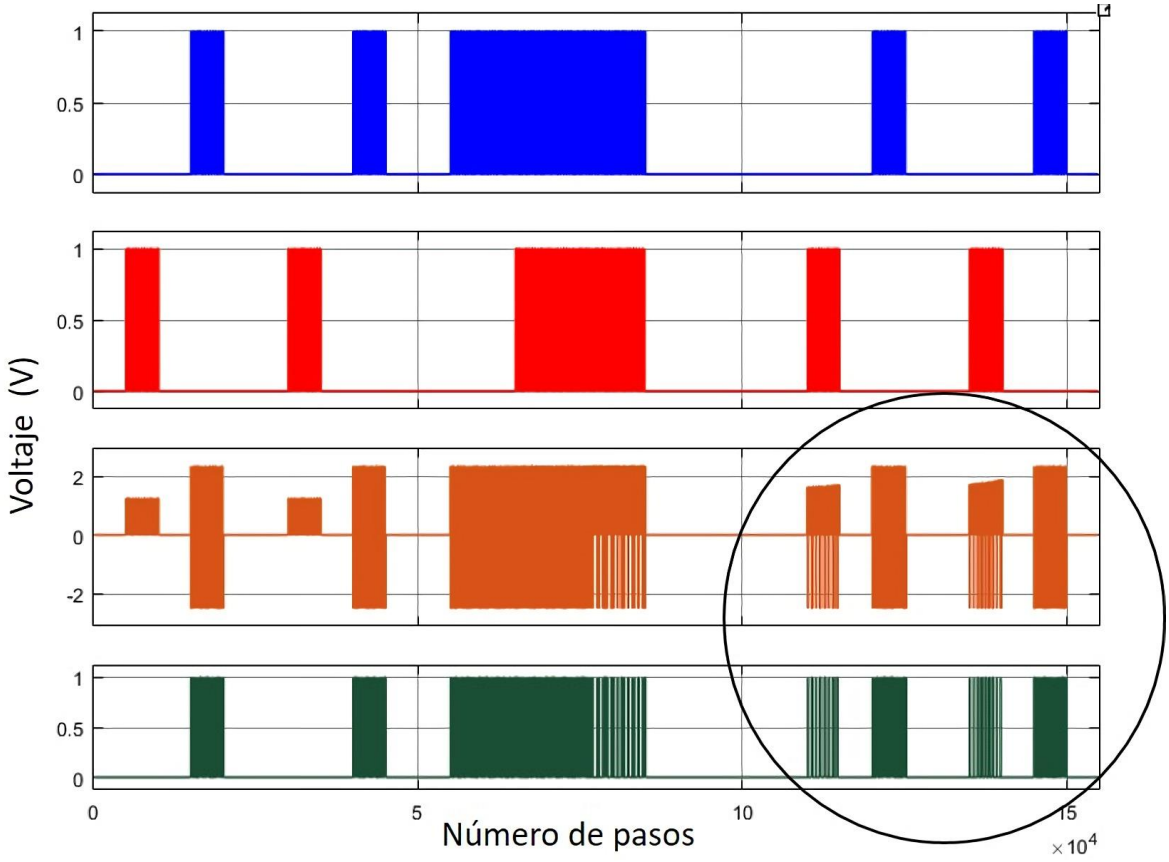

Figura 4.27 SNN implementada, dejando que el control para flotar los memristores sea decidido por la red.

En este ejemplo, aún después de haber finalizado el entrenamiento, la sinapsis del sonido continúa cambiando cuando un estímulo es presentado.

En la *Figura* 4.28, se observa una ampliación de la zona indicada en la *Figura* 4.27, donde es evidente un aumento en el número de pulsos a la salida de la neurona de salivación (8 a 9 pulsos), debido a que la memristancia asociada continúa disminuyendo al presentar solo el estímulo del sonido.

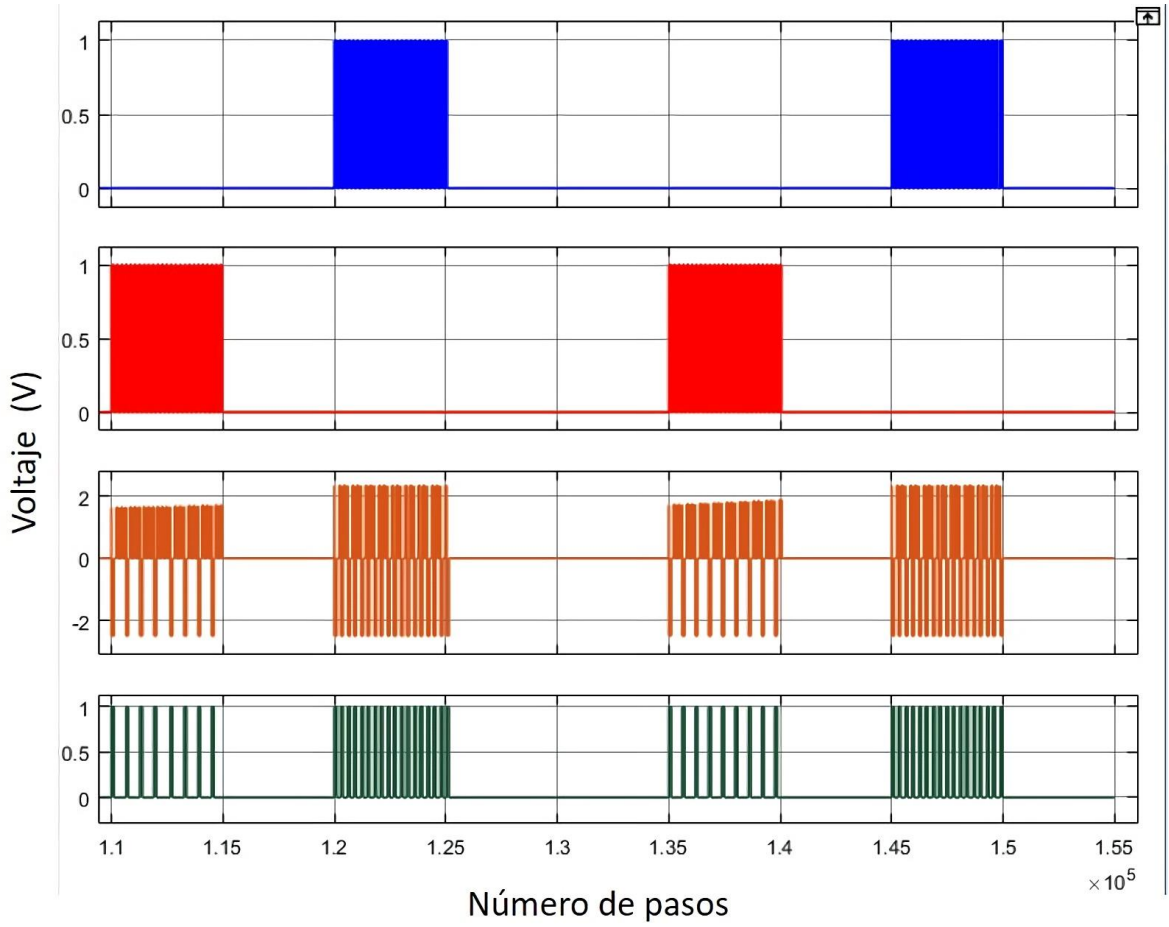

Figura 4.28 Ampliación para observar el cambio de la sinapsis asociada al sonido

Esto es consecuencia del módulo KCL, donde el voltaje de entrada a la neurona de salivación es constantemente actualizado (como se observa en los pulsos naranjas), debido a que los pulsos de retro propagación son generados cada vez que es superado el umbral de dicha neurona, provocando un cambio de memristancia en el siguiente ciclo. De manera similar ocurre al estimular la neurona de la vista, pero como está ya se encuentra en su valor mínimo de memristancia, no es posible observar una disminución.

Por último, se observa en la *Figura* 4.29 el caso donde el entrenamiento y las pruebas son llevadas al extremo, provocando que a la salida de la neurona de salivación los pulsos generados por la vista de comida sean los mismos que los generados por el estímulo sonoro, ya que la memristancia asociada a la sinapsis del sonido se encuentra en el mismo valor que el de la vista.

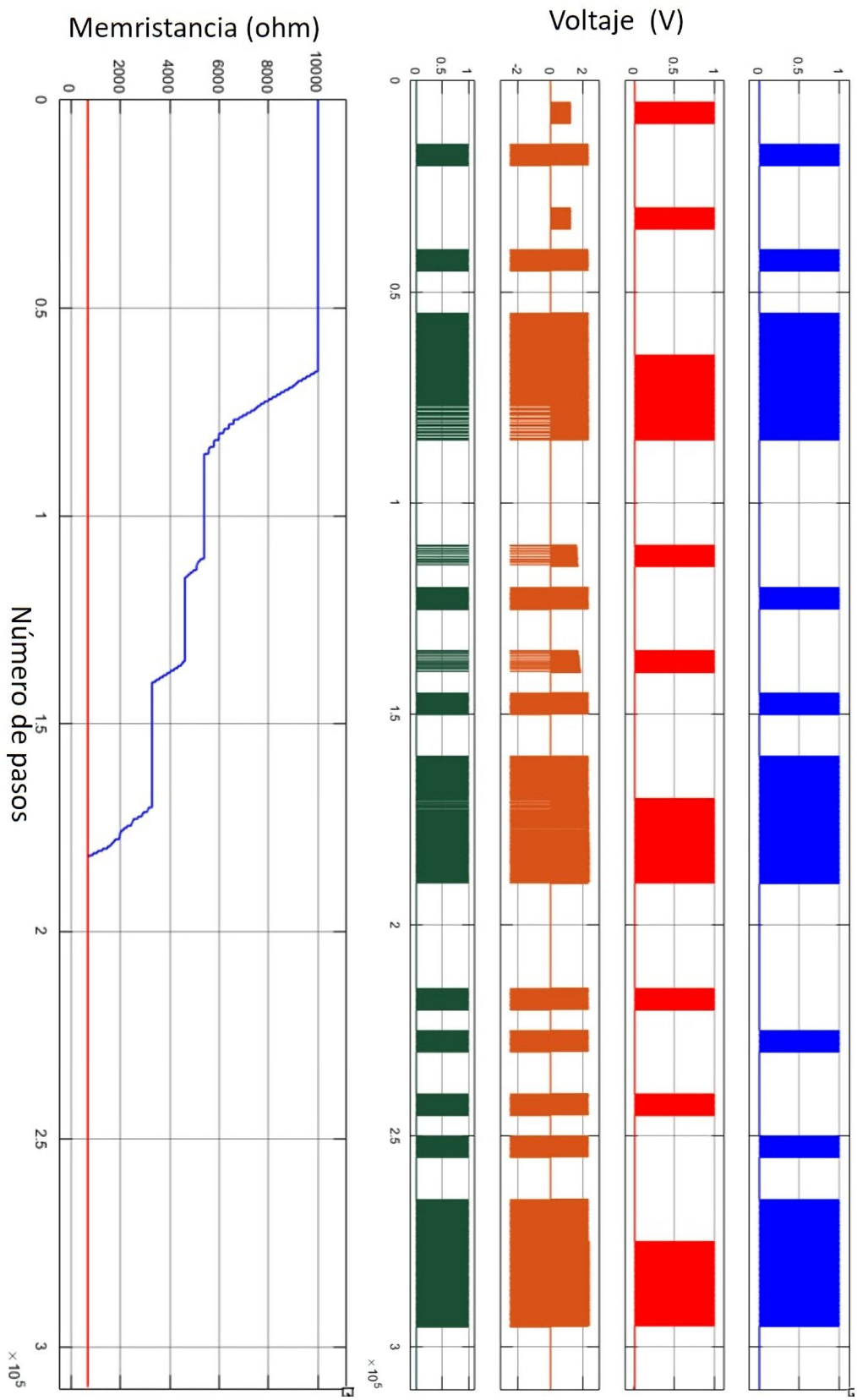

Figura 4.29 SNN entrenada varias veces.

Los recursos consumidos por esta implementación a 32 bits en punto flotante son dados en la *Tabla* 4.4.

Módulos	Cantidad	DSP's.	LUT's	Registros
Memristores	2	40	5889	93
Neuronas	3	27	8390	294
KCL	1	14	1043	32
Total		**81(34%)**	**15322(24%)**	**149(0.5%)**

Tabla 4.4 Recursos consumidos por la implementación a 32 bits en flotante.

Entonces, se realizó una optimización de recursos, por lo que los resultados similares fueron obtenidos con neuronas a 18 bits en punto fijo, siendo éste el mínimo alcanzado, según la *Tabla* 4.5.

Módulos	Cantidad	DSP's.	LUT's	Registros
Memristores	2	40	5919	90
Neuronas	3	0	694	168
KCL	1	14	1045	32
Total		**54(23%)**	**7658(12%)**	**290(0.5%)**

Tabla 4.5 Recursos consumidos por la implementación a 18 bits en punto fijo.

4.6 Implementación SNN de una Matriz Dinámica de reconocimiento de Caracteres

Ya realizada la construcción y verificación de la SNN recreando el experimento Pavlov's dog, se procede a generar una implementación donde sea posible visualizar el funcionamiento del algoritmo STDP implícito en la red neuronal pulsada. En esta ocasión se ampliará la red para poder realizar el reconocimiento de caracteres de una matriz dinámica.

En la *Figura* 4.30, se observa un alfabeto de 3x3 pixeles monocromáticos, de donde se tomarán algunos caracteres para la prueba de la red. En esta implementación, cada pixel es representado por una neurona y un memristor, por lo que son necesarias 9 neuronas para toda la matriz más 1 neurona de salida, además de los 9 memristores trabajando como sinapsis, y un módulo KCL para calcular el voltaje en la neurona de salida, así como los pulsos de retro-propagación.

Figura 4.30 Alfabeto y números 3x3 pixeles

Como se observa en la *Figura* 4.31, el número de neuronas necesarias aumentó, por lo que el módulo KCL deberá ser expandido también.

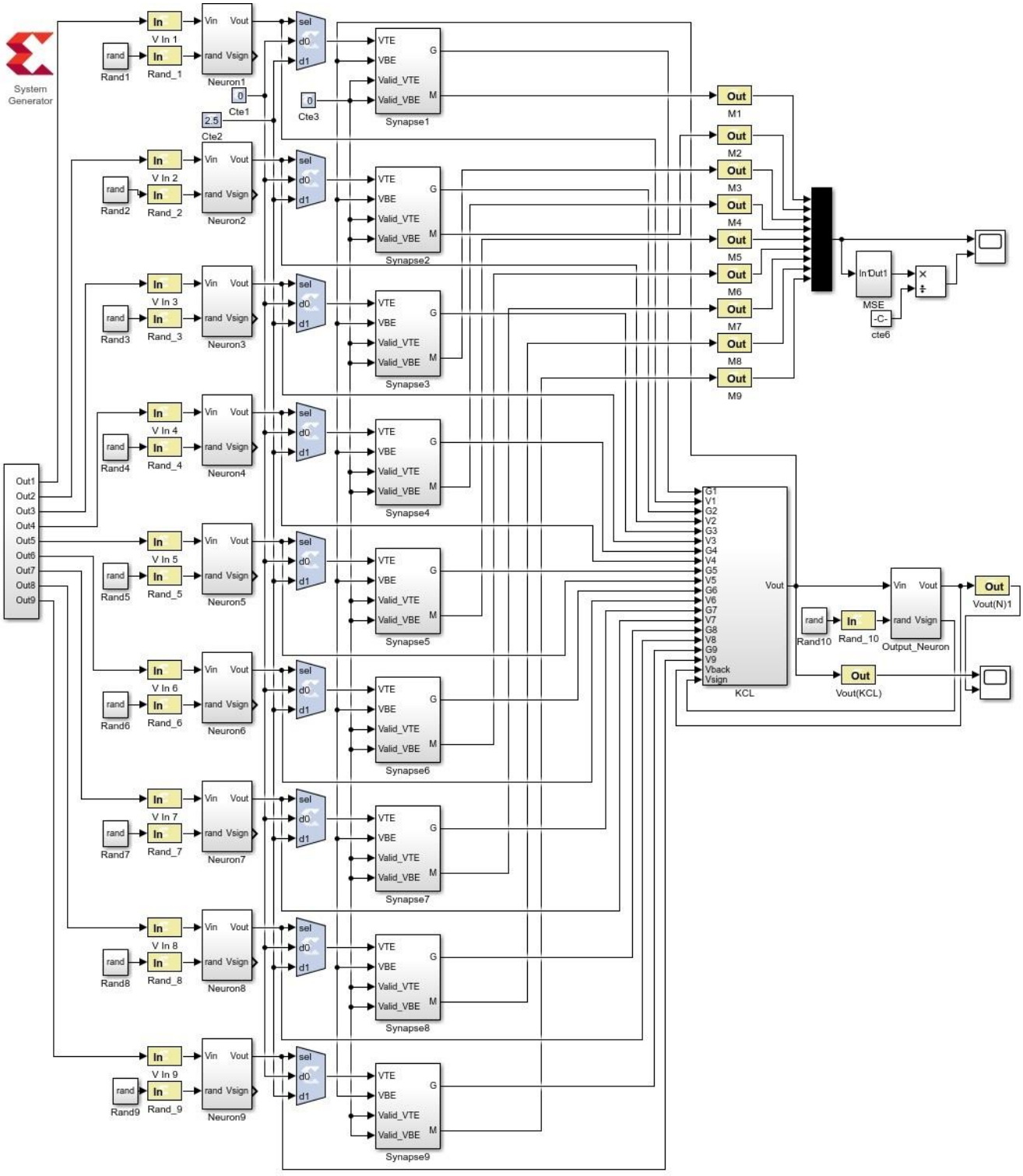

Figura 4.31 Implementación SNN para el uso de una matriz dinámica de caracteres.

En la *Figura* 4.32, se aprecia el interior del KCL, donde los valores Vpulse fueron elegidos para obtener una co-simulación más rápida, ya que, en este caso al tener un número significativo de ciclos de reloj, la limitación es la velocidad de trasferencia entre el pc y el FPGA.

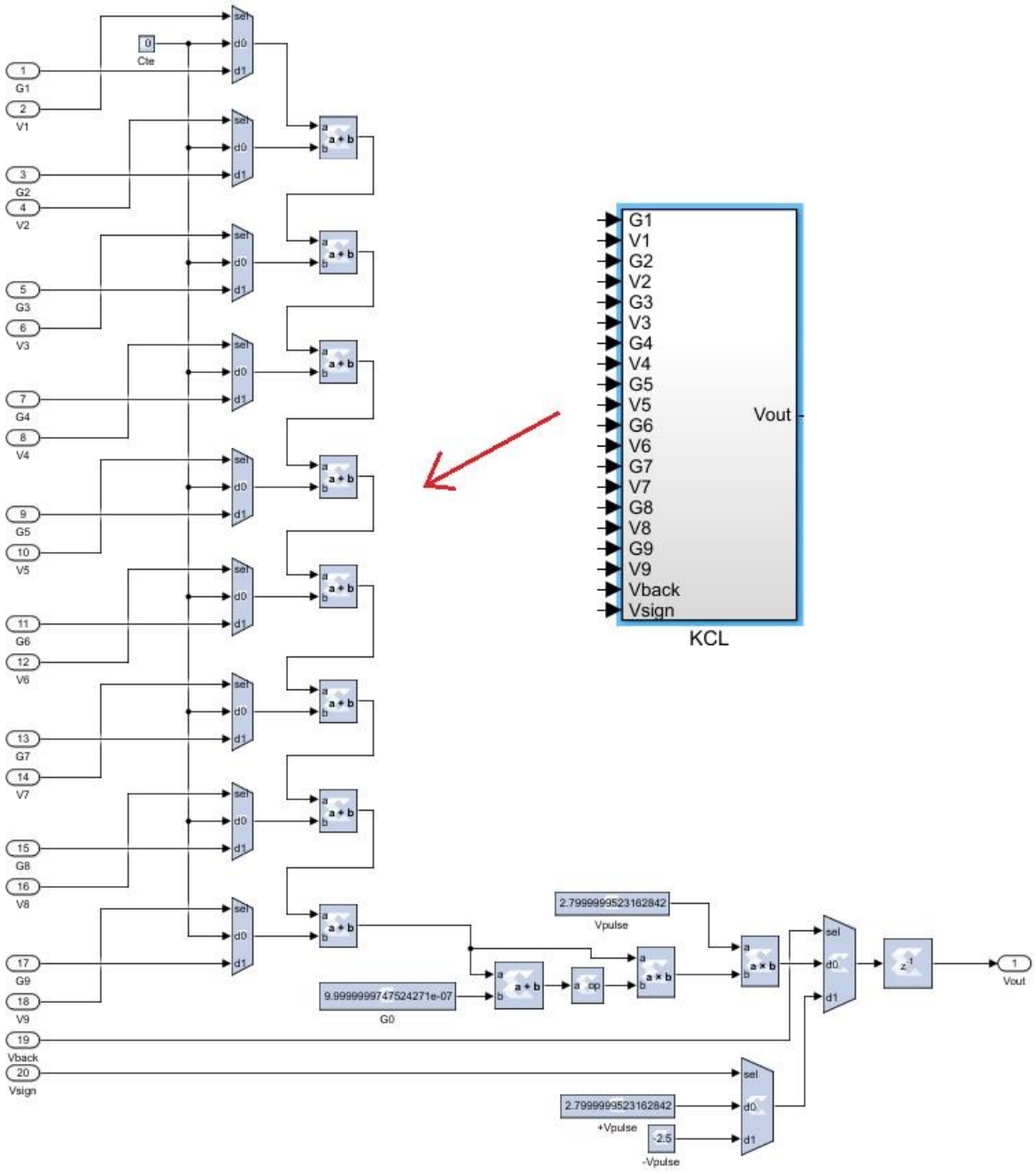

Figura 4.32 Modulo KCL ampliado para 9 Neuronas.

En este caso, para realizar las pruebas se tomaron las letras D, U, T, H; los memristores son inicializados en un valor intermedio (5KΩ ±40Ω), utilizando los parámetros: $a = -800\frac{\Omega}{V \cdot s}$, $\mathrm{b} = -12800\frac{\Omega}{V \cdot s}$, $V_T = 3V$, $R_{min} = 675\Omega$, $R_{max} = 10K\Omega$. Además, es necesario conectar el Vsing de la neurona de salida al módulo KCL, ya que se necesitarán los pulsos de retro-propagación positivos y negativos.

Cada pixel será representado por una neurona y un memristor, enumerados como en la *Figura* 4.33.

Figura 4.33 Numeración utilizada para las neuronas de entrada.

Lo que se realiza, es colocar una serie de patrones asociados a cada letra, como en la *Figura* 4.34; de esta forma, se excitan con 2.5V las neuronas que forman dicha letra(negro) y a 0V el resto de pixeles (blanco). Así, para formar la letra D se excitan todas las neuronas a excepción de 5, 7, 9, como se observa en la *Figura* 4.34. Además, al ser una matriz dinámica, los caracteres son presentados solo por intervalos de 100, 000 ciclos de reloj y después se cambia a la siguiente letra.

Cuando estas neuronas de entrada se encuentran pulsado (V_{IN}=2.5V), el voltaje en la entrada de la neurona de salida es calculado, esto la hace pulsar y gracias al Vsign solo los memristores que corresponden a las sinapsis de la letra presentada se polarizan directamente, disminuyendo así su memristancia; mientras, en las

neuronas no excitadas, los memristores se consideran a tierra en lugar de estar flotados por medio del control Valid_VTE y VBE, así, al presentarse los pulsos de retro -propagación estos se encontrarán polarizados de manera inversa, aumentando la memristancia.

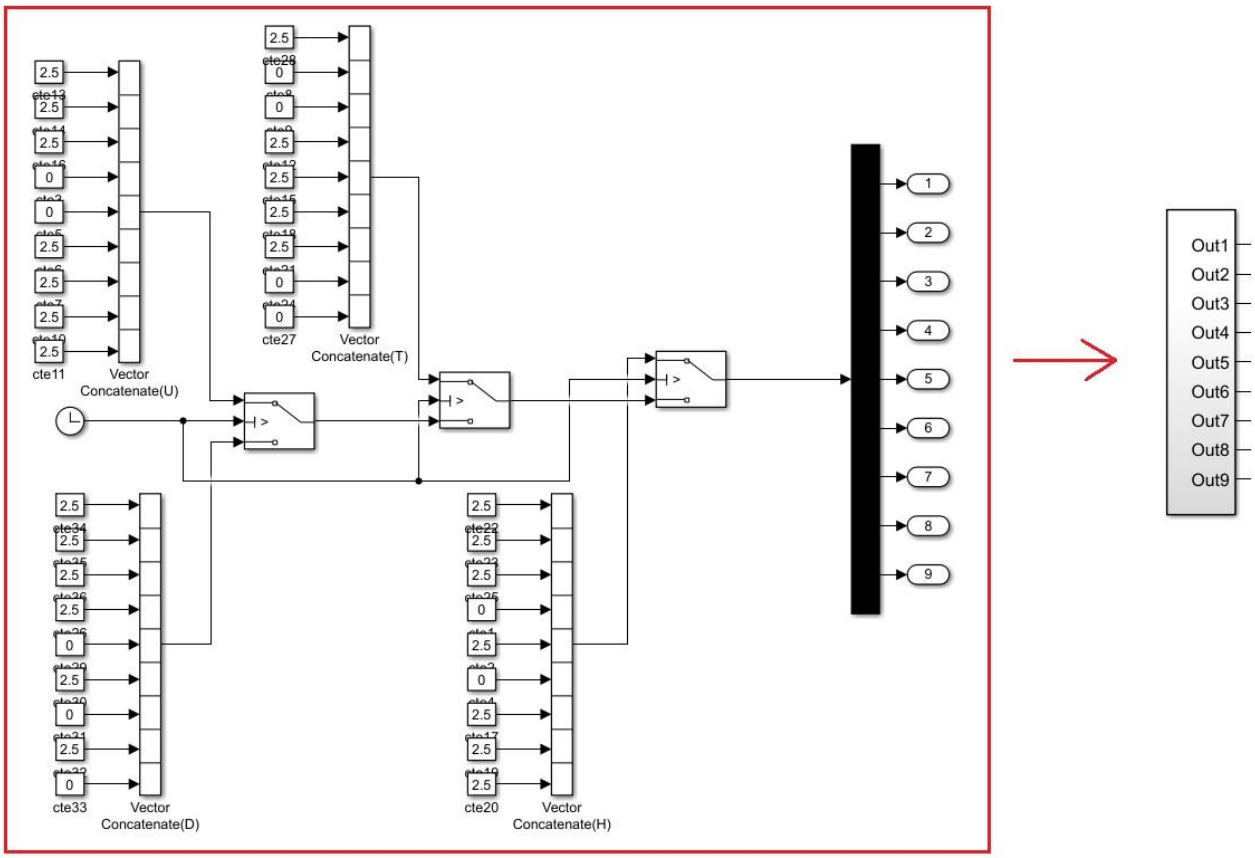

Figura 4.34 Patrones para excitar las neuronas de entrada.

Para observar la evolución del sistema, se calcula el error cuadrático medio con la ayuda del módulo MSE. El interior de este, se puede observar en la *Figura* 4.35, donde la obtención del error está sincronizada con el carácter presentado; así, los resultados obtenidos pueden ser visualizados en la *Figura* 4.36 y en la *Figura* 4.37.

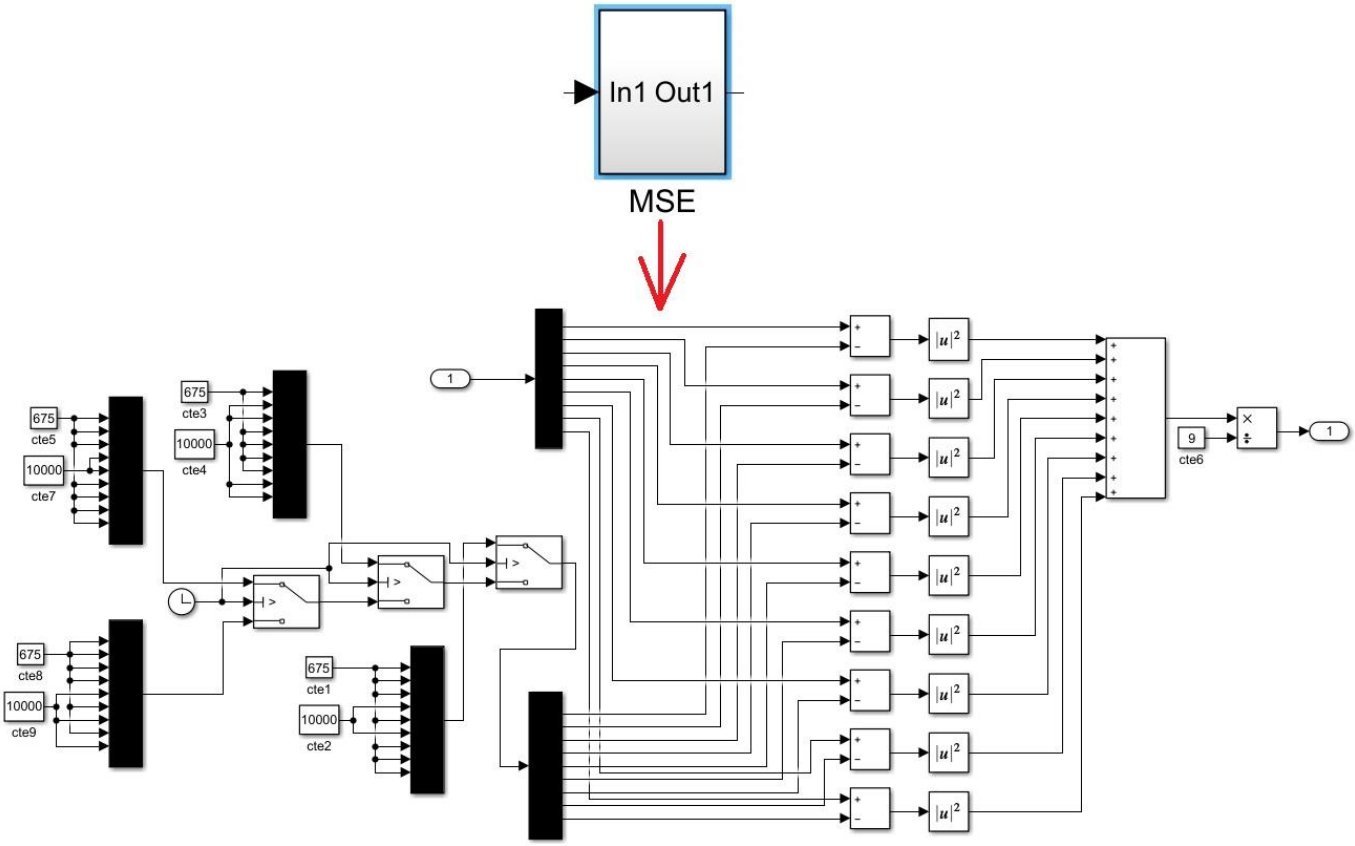

Figura 4.35 Obtención del error cuadrático medio (MSE).

Hay que resaltar que la obtención del MSE es llevada a cabo en Simulink, y no en el FPGA, esto debido a que al ser una implementación más compleja se quiso evitar el uso completo de los DSP's, lo cual podría provocar problemas de compilación, por lo que solo se tomaron los valores de memristancia.

En la *Figura* 4.36 A) se observan las nueve sinapsis inicializadas alrededor de 5KΩ, mientras que en B) se observa el error cuadrático medio. Al presentar la primera letra (D), la memristancia de cada sinapsis comienza a cambiar, disminuyendo en las neuronas excitadas, provocando una mayor contribución a la neurona de salida, y aumentando en las neuronas no excitadas, minimizando así la aportación de estas en la siguiente neurona.

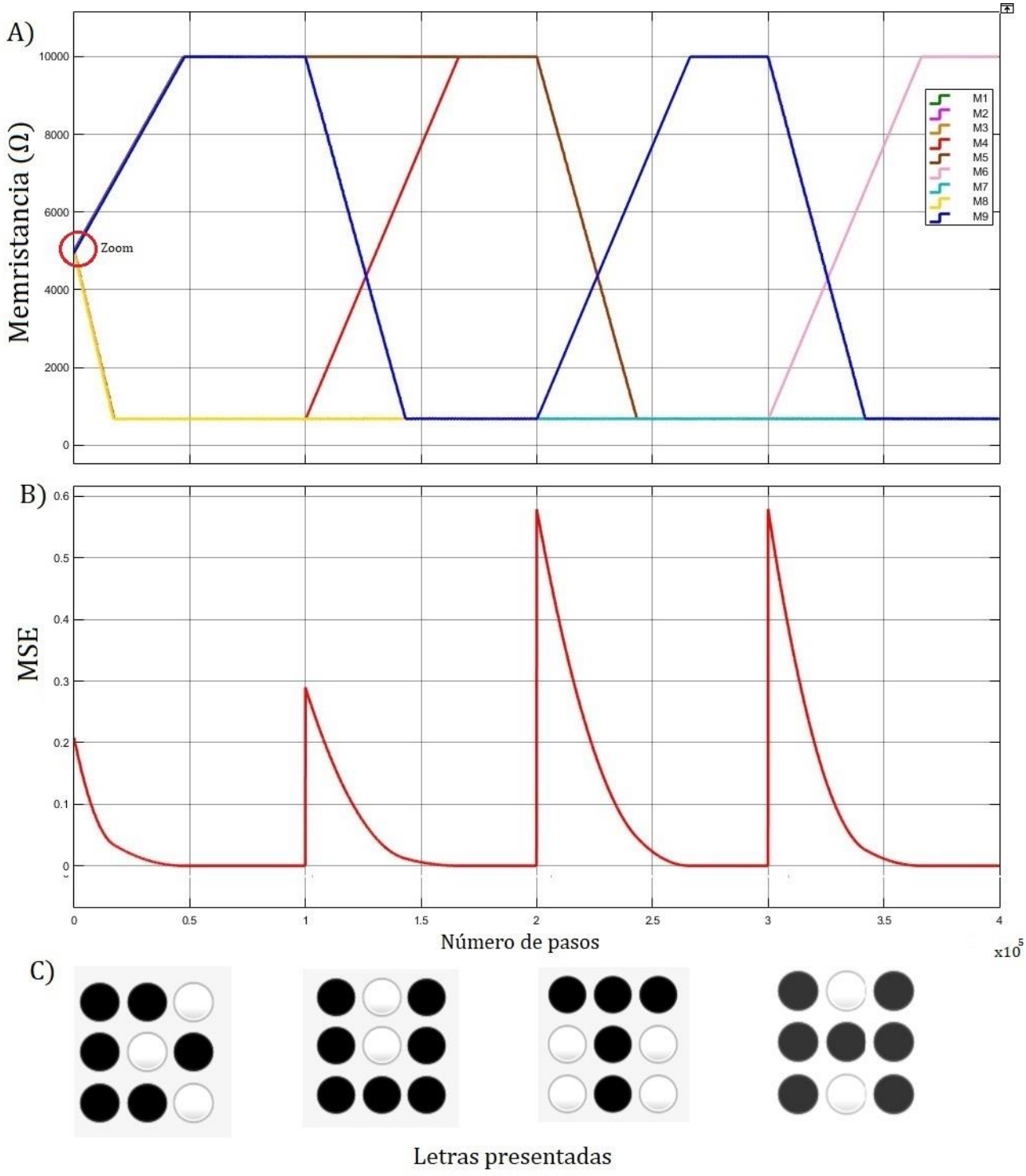

Figura 4.36 Resultados implementación SNN en aprendizaje continuo para una matriz dinámica de caracteres, A) cambio de las memristancias dependiendo del patrón presentado, B) evolución del MSE conforme el patrón es aprendido, C) patrones presentados.

Este proceso es repetido durante un intervalo de tiempo conocido como aprendizaje.

Mientras esto sucede, se puede observar como el MSE va disminuyendo hasta llegar a cero, esto indica que la red aprendió a reconocer esa letra ya que ahora presentará un patrón especifico de pulsos a su salida. Otra forma de verificar el aprendizaje de la letra es por medio de la sinapsis, ya que ella se puede apreciar, por medio de los estados Roff y Ron, la letra almacenada.

Cuando otra letra es presentada, el MSE es calculado de nuevo, por lo que mientras más difiera una nueva letra de la anterior, mayor será el MSE inicial; esto puede ser comprobado con las letras T y H ya que una es casi la opuesta (pixeles) de la otra.

Cada vez que la letra cambia, los valores de las memristancias lo hacen también, aumentando o disminuyendo según sea requerido, asociando la letra nueva hasta disminuir a cero el MSE, por lo que este sistema se encuentra en un continuo aprendizaje.

En la *Figura* 4.37, se observa una ampliación de los primeros ciclos de reloj correspondientes a los resultados presentados en la *Figura* 4.36, donde es posible distinguir las nueve sinapsis inicializadas alrededor de los 5KΩ, donde al presentar el primer patrón, solo las memristancias 5, 7 y 9 aumentan, mientras las otras disminuyen.

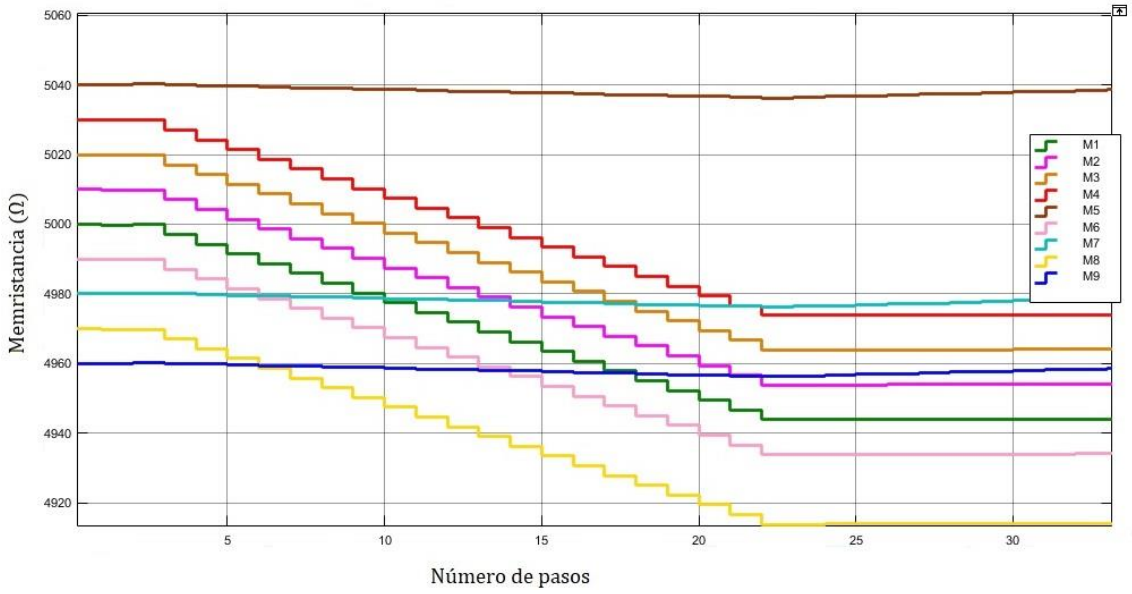

Figura 4.37 Ampliación para observar los 9 memristores inicializados en 5kΩ (±40Ω).

Los pulsos generados por la neurona de salida si bien tienen un patrón, este no especifica de que letra se trata, ya que una letra la cual tenga el mismo número de pixeles encendidos puede generar el mismo patrón de salida, por lo que para lograr asociar los pulsos de una neurona con una determinada letra es necesario contar con más neuronas de salida y módulos KCL.

En la *Figura* 4.38, se observa el patrón de salida regular generado al presentar la letra D, después de entrenar la red (antes de $10x10^4$ pasos); después, al cambiar la letra, se observa como el patrón de salida cambia por uno no diferenciable.

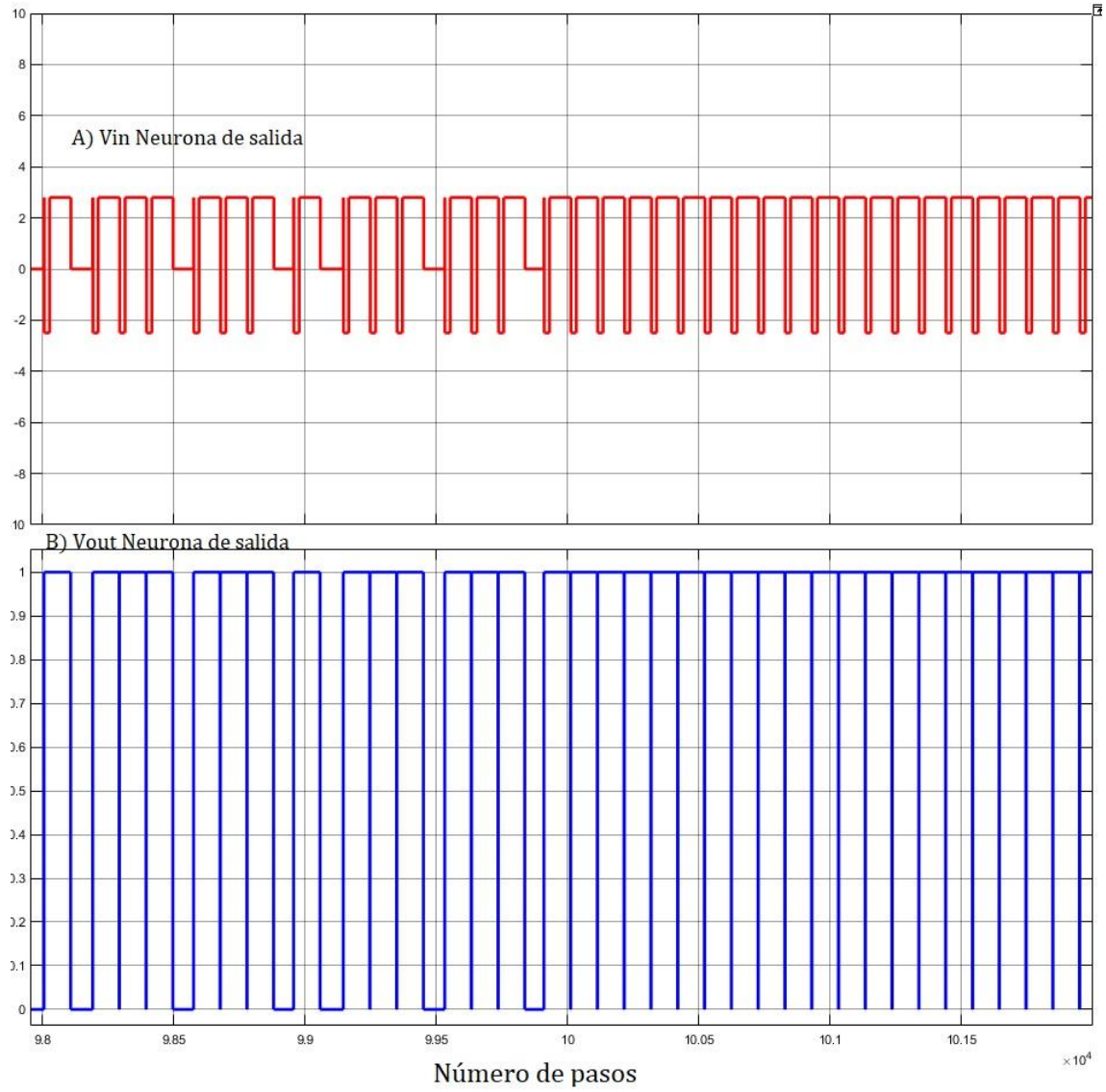

Figura 4.38 Patrón de salida después del aprendizaje de la primera letra (D).

Siendo que la mayor limitante son los recursos consumidos por los memristores y el módulo de suma KCL, la implementación máxima en este FPGA es una matriz 3x3, los recursos utilizados por esta implementación con las neuronas a 18 bits en punto fijo y el memristor en 32 bits a punto flotante son mostrados en la *Tabla* 4.6.

Módulos	Cantidad	DSP's.	LUT's	Registros
Memristores	9	180	27169	397
Neuronas	10	0	1997	560
KCL	1	14	3781	32
Total		**194(81%)**	**32947(52%)**	**989(0.85%)**

Tabla 4.6 Recursos consumidos por esta implementación a 18 bis en punto fijo.

Como se aprecia en la *Tabla* 4.6, los multiplicadores (DSP's) que se requieren para implementar este ejemplo están cerca del máximo (240) en punto fijo, por lo que en punto flotante no son suficientes los recursos. Es por esto que solo fue posible implementar caracteres de una resolución 3x3, logrando mantener el objetivo del uso de un FPGA (ejecución paralela).

4.7 Implementación del Algoritmo de Hormigas con Memristores

Otra forma de abordar los posibles usos del memristor como se planteó en el capítulo anterior, es hacer un análogo al algoritmo de hormigas. Una vez demostrado que efectivamente las redes de memristores pueden asemejar la optimización de caminos llevada a cabo por las hormigas, si se agrega un parámetro de relajación en la ecuación de estado del memristor, se plantea la posibilidad de resolver arreglos mucho más complejos. En este caso, se implementará en el FPGA el arreglo de memristores mostrado en la *Figura* 4.39, donde se observan varios posibles caminos a elegir.

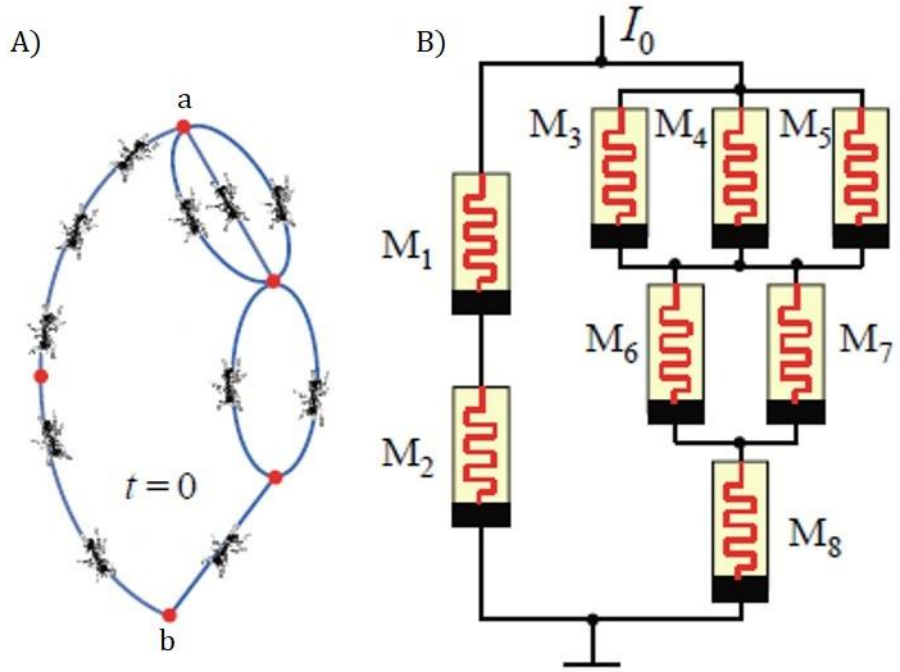

Figura 4.39 A) Diagrama con hormigas a resolver con B) arreglo de memristores equivalente.

En esta ocasión, se mantiene el mismo modelo de memristor mostrado en los ejemplos anteriores, a excepción de agregar un término extra en la ecuación que describe el cambio de la memristancia.

$$\dot{M} = b * v + \frac{1}{2}(\alpha - b)(|V + V_T| - |V - V_T|)\,\theta(R - R_{ON})\,\theta(R_{OFF} - R) + \Gamma M \qquad (4.6)$$

La ecuación (4.6) fue utilizada en los ejemplos anteriores, el término extra Γ conocido como parámetro de relajación, es propuesto en la ref. [6] $\Gamma = 50$, esto, para lograr una semejanza con el algoritmo de hormigas, donde Γ representa la evaporación artificial de las feromonas.

En nuestro caso Γ tiene la función del regresar al memristor a su valor máximo de memristancia, logrando de esta manera una evolución diferente de los memristores si la corriente repartida no es igual, por lo que para esta implementación fue necesario modificar el memristor para trabajar con una corriente de entrada como se muestra en la *Figura* 4.40

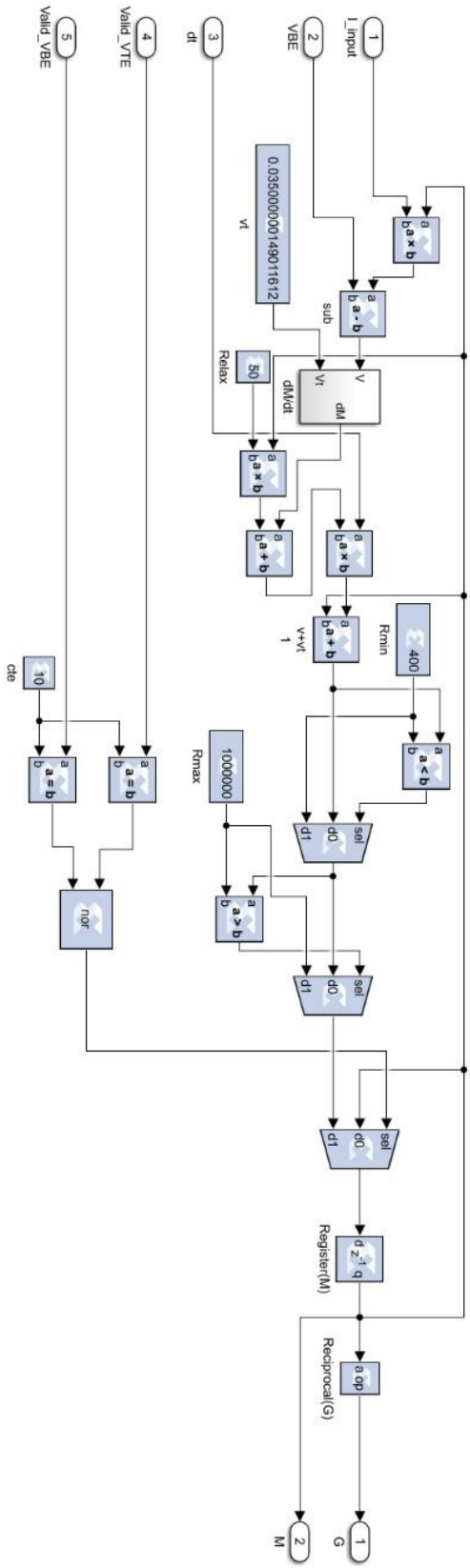

Figura 4.40 Módulo memristor arreglado para trabajar con corriente y el parámetro de relajación

Una vez modificado el módulo del memristor incluyendo el parámetro de relajación, se procede a generar el arreglo de la Figura 4.39 en System Generator, como se muestra en la *Figura* 4.41.

Pero de manera similar como se presentó en el ejemplo del CRS (switch resistivo complementario), al ser una implementación digital es necesario conocer el valor del voltaje o corriente en todos los nodos, por lo que es necesario construir pequeños módulos encargados de calcular la corriente en cada nodo del arreglo.

Estos módulos se apoyan del memristor, ya que este es capaz de entregar memristancia o memductancia; sin embargo, las operaciones que se realizarán no serán binarias como en el caso del KCL, por lo que ocuparán recursos adicionales a los contemplados solo por los memristores.

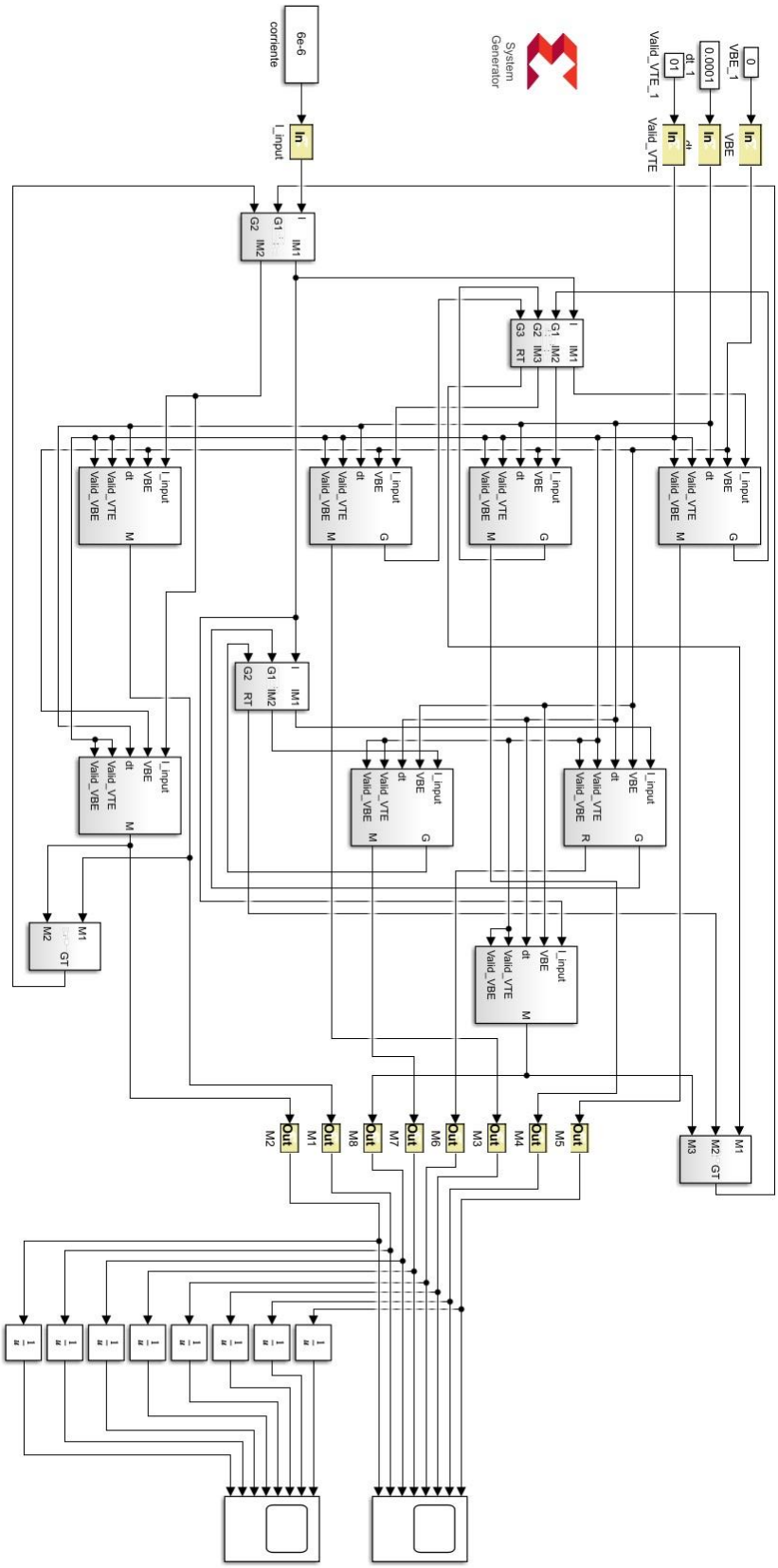

Figura 4.41 Red de memristores

En la *Figura* 4.42, se observa un módulo para calcular las corrientes en un arreglo de memristores en paralelo, esta estructura puede ser ampliada o disminuida según se requiera; además debido a que la corriente en serie será la misma, este módulo solo es necesario en 3 ocasiones.

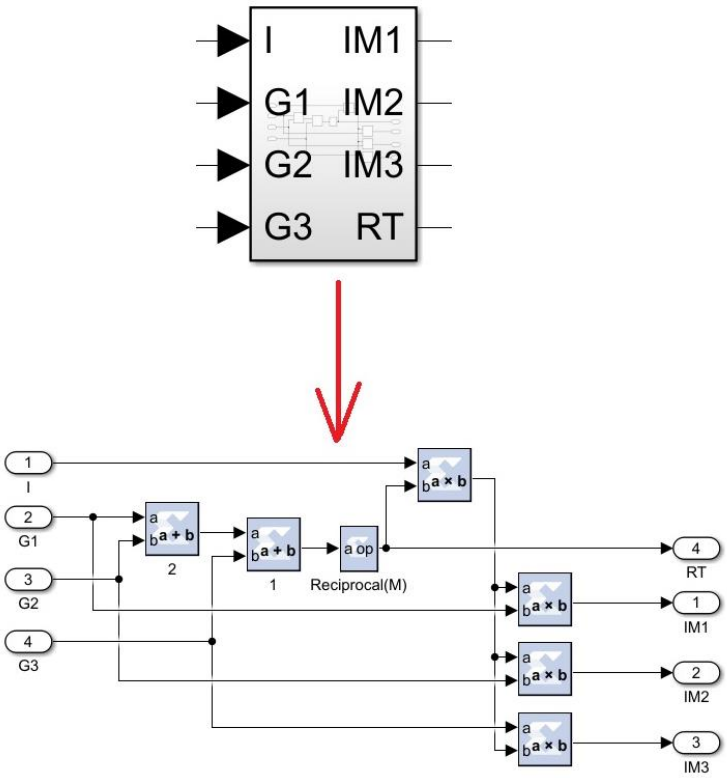

Figura 4.42 Módulo para calcular corrientes en paralelo.

Los parámetros utilizados para este ejemplo son: $a = 0\frac{\Omega}{V \cdot s}, \mathrm{b} = -190000\frac{\Omega}{V \cdot s}, \Delta t = 0.0001s, V_T = 0.035V, R_{min} = 400\Omega,\ R_{max} = 1M\Omega,\ R_{init} = 100K\Omega$. El V_T es pequeño, debido a que se desea un cambio apenas se aplique un voltaje, además como a representa el cambio de memristancia antes del umbral, este parámetro pierde relevancia.

En la *Figura* 4.41, se observa la implementación llevada a cabo en System Generator, esta es equivalente a la mostrada en la *Figura* 4.39 B donde los memristores se encuentran inicializados con valores iguales iguales, solo que

debido al arreglo en el que se encuentran, la corriente inyectada se distribuye de manera diferente.

Cada memristor necesita una corriente de entrada calculada por los módulos en cada nodo; como la corriente en serie es la misma, no es necesario hacer uso del VBE, por lo que es colocado a 0. Entonces, cuando la corriente atraviesa un memristor se genera un voltaje de entrada, este modifica la memristancia conforme la ecuación (4.6). Sin embargo, debido a la naturaleza del arreglo, la corriente en cada memristor difiere, por lo que los memristores no son polarizados con el mismo voltaje.

Esto se ve reflejado en la *Figura* 4.43, donde se observa como al principio todos los memristores se encuentran inicializados en 100KΩ, al estar polarizados directamente tendrían que disminuir su memristancia, pero dado que tienen voltajes diferentes, el parámetro de relajación Γ afecta más a los memristores con menor voltaje. Entonces, en un inicio la memristancia tiende a disminuir, pero conforme transcurre el tiempo esta cambia más rápido en unos memristores que en otros; para este caso, en los memristores 1, 2 el voltaje que los polariza es mayor que en los otros, ya que la memristancia total en ese camino es menor, provocando un mayor flujo de corriente.

Conforme aumenta el número de pasos (tiempo), los memristores 1, 2 se ven con más beneficios al recibir mayor corriente; este proceso se repite a lo largo del tiempo provocando que, después de 400 ciclos de reloj, casi la totalidad de corriente fluya por este camino, en cambio, los otros memristores aumentan su memristancia al verse afectados por Γ. En la *Figura* 4.44, se aprecia la evolución de la memductancia, verificando que, efectivamente el camino con mayor memductancia son los memristores 1, 2 correspondientes al mejor camino posible.

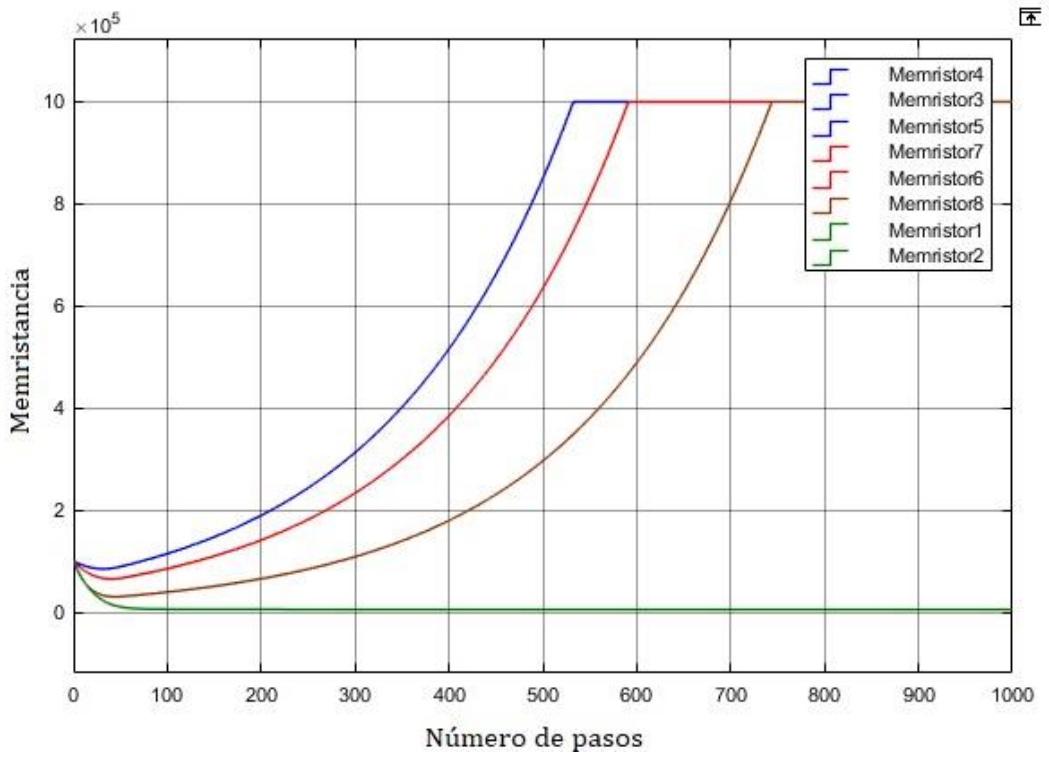

Figura 4.43 Evolución de la memristancia.

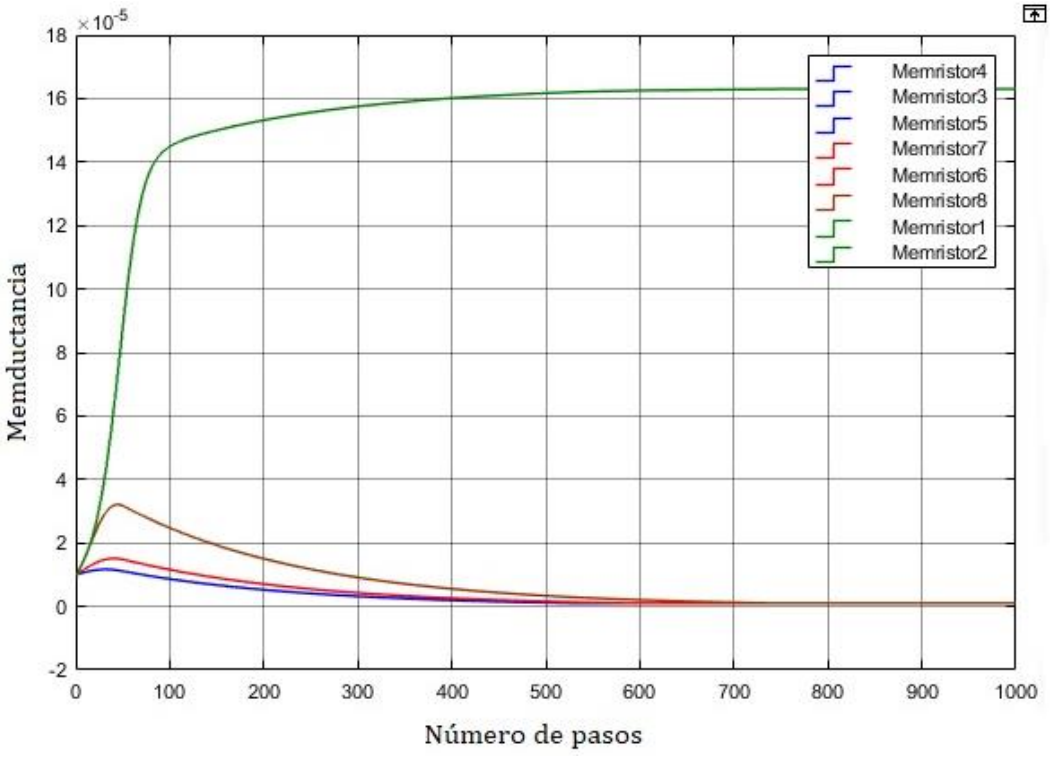

Figura 4.44 Evolución memductancia.

Otros ejemplos siguiendo la metodología anterior fueron simulados y programados en el FPGA, System Generator y Matlab; éstos se encuentran en el apéndice B.

Los recursos utilizados para esta implementación se encuentran en la *Tabla* 4.7; para este caso se utilizó el módulo del memristor a 32 bits en punto flotante, ya que el intervalo en él que varía la memristancia va desde 400Ω hasta 1MΩ.

Módulos	Cantidad	DSP's.	LUT's	Registros
Memristores	8	178	29049	638
Obtención de I	3	40	3511	0
Obtención de G	2	22	1605	0
Total		**240(100%)**	**34165(54%)**	**638(0.5%)**

Tabla 4.7 Recursos utilizados para la implementación (red de 8 memristores).

4.8 Procesamiento de Imágenes con Memristores

Como último uso, presentado en esta tesis, se abordará el procesamiento de imágenes. Ya que en las referencias [7] y [8] se plantea el algoritmo de hormigas como un detector de bordes, además de que en la ref. [6] se propone el uso del memristor como análogo del algoritmo de hormigas, se aborda un ejemplo comparando ambos métodos según la ref. [9].

El proceso para detección de bordes en imágenes, es un problema computacionalmente intensivo, se basa en la búsqueda de cambios bruscos de brillo en la imagen, identificando y dejando solo un conjunto de líneas conectadas como se observa en la *Figura* 4.45 para un caso ideal.

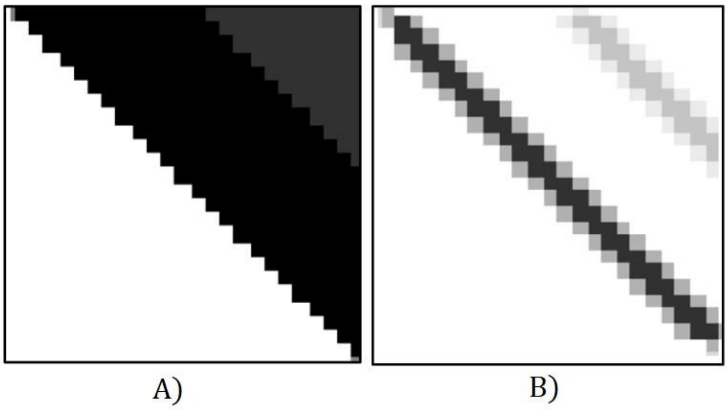

Figura 4.45 A) Imagen con dos bordes, B) Contrastando los bordes de A), tomada de a ref. [9].

Sin embargo, en la práctica, los objetos presentes en una imagen no siempre tienen un contraste óptimo, lo cual puede conducir a la detección de bordes falsos, esto requiere un post procesado mayor.

Existiendo excelentes algoritmos enfocados directamente a la detección de bordes como Canny, los cuales observan la imagen como un conjunto, el objetivo de esta implementación es presentar el memristor como una alternativa que pueda ser utilizada en forma de Hardware en un futuro, en este caso la red de memristores observaría la imagen pixel a pixel, para procesarla de manera secuencial.

Para ambos métodos (hormigas y memristores), es necesario inicializar la imagen en una escala de grises; posteriormente, a cada pixel se le asigna un valor conocido como la heurística asociada, cuyo valor está definido por los pixeles vecinos, en la ref. [9] se presenta la fórmula:

$$\eta_{(i.j)} = \frac{1}{I_{MAX}}[|I(i,j-1) - I(i,j+1)| + |I(i-1,j) - I(i+1,j)|] \qquad (4.7)$$

Donde $I(i,j)$ representa la intensidad del pixel en el lugar (i,j), I_{MAX} es la intensidad máxima de un pixel, que se utiliza para normalizar la heurística asociada $\eta_{(i.j)}$. Sin embargo, la ecuación (4.7) presenta una discontinuidad al solo observar un cambio de menor a mayor, por lo que se optó por modificarla.

$$\eta_{(i.j)} = \frac{1}{I_{MAX}}[|I(i,j-1) - I(i,j+1)| + |I(i-1,j) - I(i+1,j)| + |I(i,j+1) - I(i,j-1)| + |I(i+1,j) - I(i-1,j)|] \quad (4.8)$$

La ecuación (4.8) similar a la ecuación (4.7) ve los cambios en la imagen de menor a mayor, así como los cambios de mayor a menor, esto asigna una heurística asociada, mejor calculada a cada pixel, por medio de sus vecinos.

Una vez que se tienen los valores heurísticos de cada pixel en la imagen, se simulan posibles caminos a recorrer por una colonia de hormigas que explora en busca del camino más corto.

Suponiendo que la colonia se encuentra en un pixel (i, j), los caminos que las hormigas pueden elegir serán definidos dada la complejidad que se desee, como en la *Figura* 4.46.

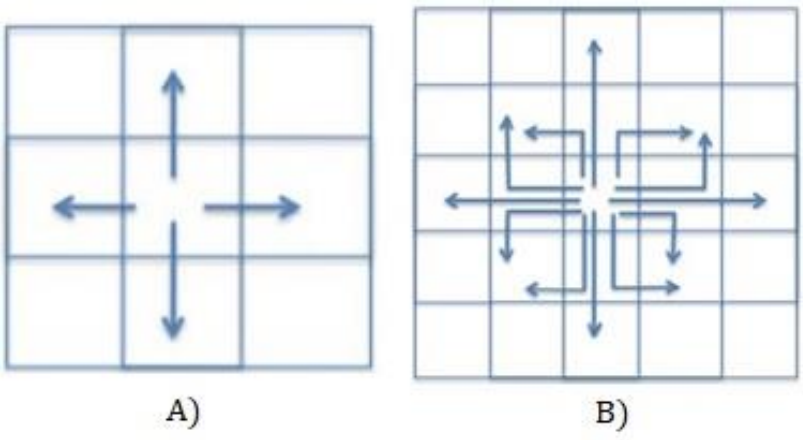

Figura 4.46 Posibles caminos a simular, presentados en la ref. [9].

Donde los caminos pueden ser solo de dos pixeles en las 4 direcciones básicas (Figura 4.46 A)), o mucho más complejos y largos como en la *Figura* 4.46 B). Para nuestro caso, se elige simular caminos de solo dos pixeles de longitud, pero en 8 direcciones, como en la *Figura* 4.47, donde las hormigas partirán del pixel $I(i,j)$.

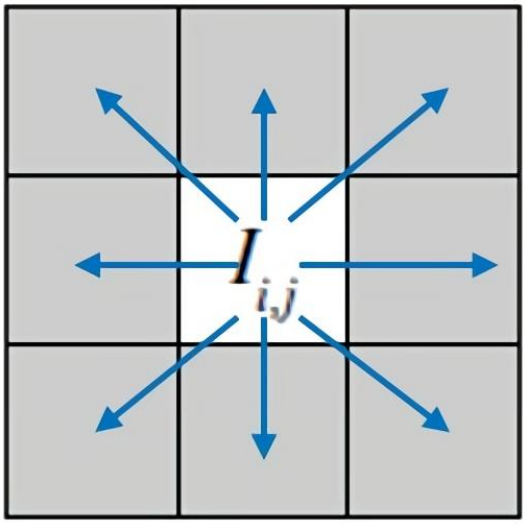

Figura 4.47 Propuesta de caminos a simular.

Una vez determinado que serán 8 caminos a simular, se hace uso de las ecuaciones y definiciones del algoritmo de optimización de hormigas presentado en el capítulo 3, donde la ecuación (4.9) representa la probabilidad de elegir un camino, en función de la feromona $\tau_{i,j}$, y la longitud del camino Le_{path_m} :

$$p_{path_m} = \frac{\prod_{(i_t,j_t)\in path_m} \tau(i_t,j_t)^{\alpha} \left(\frac{1}{Le_{path_m}}\right)^{\beta}}{\sum_{f=1}^{M} \prod_{(i_t,j_t)\in path_f} \tau(i_t,j_t)^{\alpha} \left(\frac{1}{Le_{path_f}}\right)^{\beta}} \qquad (4.9)$$

La longitud de un camino está definida por la ecuación (4.10), por medio de la heurística asociada a cada pixel.

$$Le_{path_m} = \sum_{(i_t,j_t)\in path_m} \eta^{-1}_{(i_t,j_t)} \qquad (4.10)$$

La actualización de las feromonas está definida en la ecuación (4.11), en función de la longitud del camino Le_{path_m} y un parámetro de evaporación de feromonas ρ.

$$\tau_{(i,j)}(k+1) = (1-\rho)\tau_{(i,j)} + \frac{vQ}{Le_{path_m}} \qquad (4.11)$$

Una vez elegido el camino más corto, este se actualiza conforma a la ecuación anterior (4.11). En cambio, los otros caminos también se actualiza su feromona, pero sin incluir el termino $\frac{vQ}{Le_{path_m}}$, por lo que el parámetro de evaporación los comenzará a afectar.

Entonces, teniendo la heurística asociada a cada pixel en escala de grises, junto con los 8 caminos a los cuales se les obtendrá la longitud, así como la probabilidad de ser elegidos y posteriormente actualizada su feromona, es posible generar el código para la detección de bordes por medio del algoritmo de hormigas. Este junto con otros programas, se encuentran en el apéndice C, por lo que para propósitos generales solo se mostraran los resultados obtenidos y los parámetros utilizados.

Hay que resaltar que en las ecuaciones presentadas (4.9), (4.10) y (4.11), el único parámetro que se ve constantemente actualizado es la feromona, por lo que en un inicio todas serán inicializadas en $\tau = 0.001$, pero, conforme evolucione el sistema, la imagen comenzará a visualizarse con esa matriz, aunque dado que esta cambiará de una manera lenta, será necesario mapear los valores a una escala uint8 para poder visualizar la imagen correctamente.

Para el primer ejemplo se utiliza la siguiente imagen mostrada en la *Figura* 4.48 :

Figura 4.48 Pimientos, formato TIFF en resolución 512x512.

Esta imagen es tomando por Matlab y convertida en escala de grises, como se observa en la *Figura* 4.49; posteriormente, los valores de la heurística asociada a cada pixel son calculados por medio de la ecuación (4.8).

Figura 4.49 Conversión a escala de grises.

Utilizando los parámetros $\alpha = 1, \beta = 1, \tau_{ini} = 0.01$, se obtienen las probabilidades de elegir el mejor camino (ecuación(4.9)) con base en la longitud y la feromona inicial del pixel (i, j). Las feromonas iniciales son todas iguales, así que sólo la longitud afecta en la primera iteración.

Una vez obtenido el mejor camino, se actualizan las feromonas como en la ecuación (4.11), utilizando $\rho = 0.001, v = 1,\ Q = 1$; se prosigue con el siguiente pixel (i+1, j) y así, hasta terminar con toda la imagen (512x512 =262 144 pixeles). Al finalizar este proceso, se tiene la primera iteración completada. En todos los casos que se mostrarán, a menos que se indique lo contrario, las imágenes fueron tratadas con 8 iteraciones.

Cuando las iteraciones fueron completadas, la imagen es mapeada a valores de 0-255 para poder ser visualizada; posteriormente, es procesada por medio de un umbral que se utiliza para obtener un mejor contraste en los bordes, El umbral puede ser modificado a conveniencia de la imagen a tratar, pero para mantener un desempeño homogéneo, este es usado siempre con los mismos parámetros, los cuales pueden ser visualizados en el apéndice C.

Los resultados obtenidos para el ejemplo anterior, al utilizar 8 caminos actualizando las feromonas solo en el mejor, pueden ser visualizados en la *Figura* 4.50.

Para explorar más opciones de este método, fueron realizados programas con 4 y 8 caminos posibles, ambos con una longitud de dos pixeles, en los que la feromona se actualiza en el mejor camino y también donde se actualiza en todos los casos.

Figura 4.50 A) Imagen al finalizar las iteraciones antes del umbral, B) después de aplicar el umbral.

Esto comprueba que, efectivamente, el algoritmo de hormigas puede utilizarse para la detección de bordes en imágenes y, como se mostró en la implementación anterior, el memristor puede utilizarse en la optimización de caminos.

Por lo que, como se plantea en la ref. [9], si se coloca una fuente de corriente en un pixel el cual tiene varios caminos a elegir representados por memristores en serie, en donde cada valor de memristancia está determinado en un inicio por la heurística asociada a cada pixel, es posible generar un ejemplo multiplexado, que recorra toda la imagen, donde cada pixel es sustituido por un memristor y una fuente de corriente colocada para que la corriente sea distribuida entre los caminos. Esto modificará la memristancia interna, y provocará una detección de bordes como en el ejemplo anterior, pero en este caso solo simulando la evolución de los memristores.

En la *Figura* 4.51 se muestra un ejemplo de cómo interpretar esta idea. Dos caminos con una longitud de 4 pixeles son representados por 4 memristores en serie cada uno, la fuente de corriente es conectada en el pixel inicial y, dependiendo de la memductancia que cada arreglo genere, la corriente será distribuida de manera diferente, eligiendo el mejor camino como las hormigas.

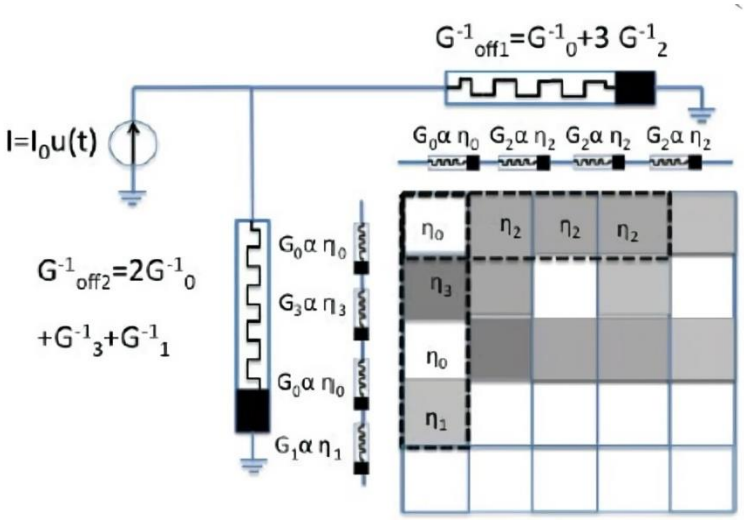

Figura 4.51 Ejemplo de caminos en el pixel inicial i=1, j=1.

La inicialización de las heurísticas asociadas será la misma que se utilizó con el ejemplo de las hormigas, solo que para inicializar las memductancias de los memristores se tomara la consideración:

$$G_{ini(i,j)} \propto \eta_{(i,j)}{}^{-1} \qquad (4.12)$$

Para este caso, se consideran 4 posibles caminos, con una longitud de dos pixeles como en la *Figura* 4.46 A), solo que para poder obtener los 4 caminos disponibles desde un principio, las líneas de los bordes no serán actualizadas, por lo que el ejemplo comenzará desde el pixel i=2, j=2, hasta i=n-1, j=n-1.

Este ejemplo, al requerir una cantidad considerable de memristores (1 por cada pixel), necesita ser multiplexado, además, debe mantener una matriz recordando todos las memristancias que serán mapeadas posteriormente a una imagen visible.

Así, al perder la característica paralela otorgada por el FPGA, los resultados que se mostrarán fueron generados solo en Matlab. Adicionalmente, un ejemplo con 4 caminos y 8 memristores es presentado en Symtem Generator y mostrado en el apéndice C, sección 4.

Para fines prácticos como ya se realizó, solo los parámetros serán indicados, mientras que los códigos de programación pueden ser consultados más adelante.

Tomando el conocido modelo del memristor ya trabajado en las implementaciones anteriores, pero ahora con los siguientes parámetros $a = 0\frac{\Omega}{V \cdot s}$, b $= -196\,000\,000\frac{\Omega}{V \cdot s}, \Delta t = 0.0001s, V_T = 0.035V, R_{min} = 400\Omega,\ R_{max} = 1M\Omega,\ \Gamma = 50,$ y utilizando una fuente de corriente de I=3uA, que se aplica durante un tiempo de 0.01s, todo esto en una única iteración, los resultados que se obtienen al procesar la misma imagen, pero con una red de memristores pueden visualizarse en la *Figura* 4.52.

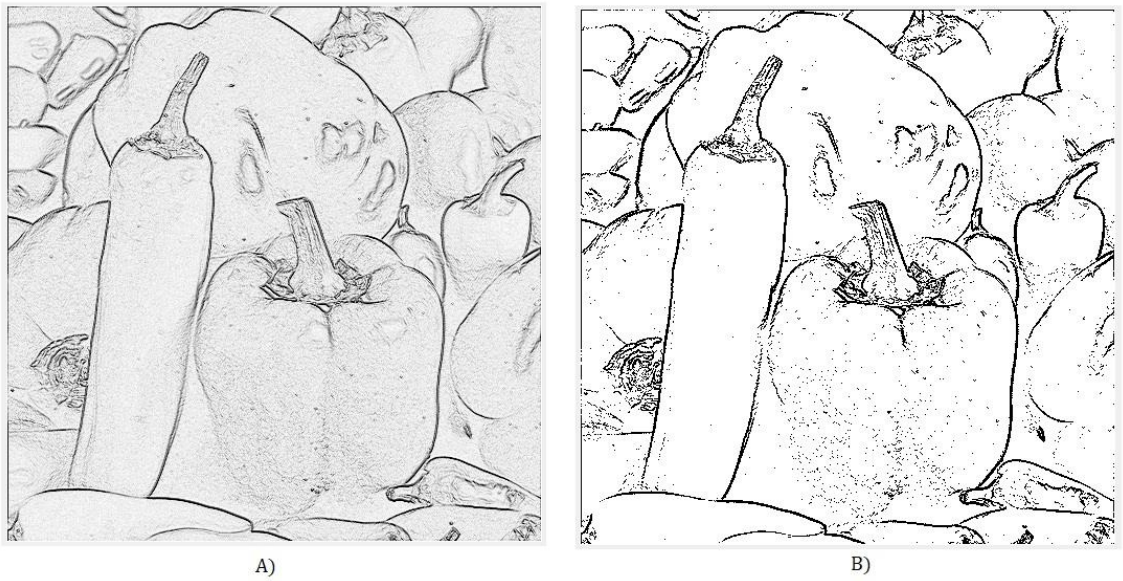

Figura 4.52 A) imagen procesada con los memristores antes del umbral, B) imagen después del umbral

Donde se observa que los resultados de la *Figura* 4.52, son considerablemente mejores con respecto a la *Figura* 4.50; sin embargo, el tiempo que lleva el proceso de la imagen también es mayor, ya que en el algoritmo de hormigas procesa una imagen de 512x512 pixeles toma un tiempo de 6.71s, mientras que el llevado a cabo por los memristores es de 108s.

Para comparar de una manera más justa, fueron modificados los programas incluyendo otros ejemplos, en donde, con el algoritmo de hormigas solo se tienen 4

caminos simples a elegir, además de actualizar todas las feromonas de la misma manera y no solo en el mejor camino, obteniendo resultados que valen la pena ser resaltados.

Figura 4.53 A) procesado con el algoritmo de hormigas utilizando 4 caminos actualizando todos los caminos, B) post-procesado de la imagen A) con un umbral.

En la *Figura* 4.53 se aprecia el ejemplo utilizando 4 caminos en vez de 8 para el procesado con el algoritmo de hormigas; en este caso, se obtiene un mejor contraste que en el procesado con memristores, pero perdiendo detalles. Para indagar más a fondo, fueron procesadas otras imágenes con mayor y menor cantidad de detalles, obteniendo un abanico de resultados para comparar de manera objetiva.

Para obtener resultados variados, se optó por procesar una imagen simple como la presentada en la *Figura* 4.54; esta imagen se encuentra en formato JPG, tiene una resolución de 450x253 pixeles y un peso de 24 KB.

Figura 4.54 A) imagen original, B) imagen en escala de grises

Siguiendo el mismo procedimiento que en los ejemplos pasados, primero se transforma la imagen a una escala de grises; posteriormente, se procesa con 4 programas distintos, 3 con el algoritmo de hormigas y 1 con memristores; se obtiene así una matriz de resultados (feromonas), que es mapeada para obtener una imagen visible y finalmente, se aplica un umbral para obtener mejor contraste.

En la *Figura* 4.55, se observan los resultados para:

A) El algoritmo de hormigas. utilizando 4 caminos como en la *Figura* 4.46A) y actualizando las feromonas de igual manera que el mejor camino, por lo que solo influye la longitud. Los índices I) y II) indican la imagen obtenida solo con el proceso y después de un umbral, respectivamente. Ya que en ese caso es una figura simple, se observa que este resultado es el que mejor contraste obtiene al aplicar el umbral. El tiempo de proceso de la imagen es de 2.24s.

B) El algoritmo de hormigas utilizando 4 caminos, pero a diferencia de A) en este caso el mejor camino actualiza las feromonas de forma distinta, obteniendo una imagen más tenue y a trazos, esto, debido a elegir solo 2 pixeles de longitud. El tiempo para procesar la imagen es de 2.25s.

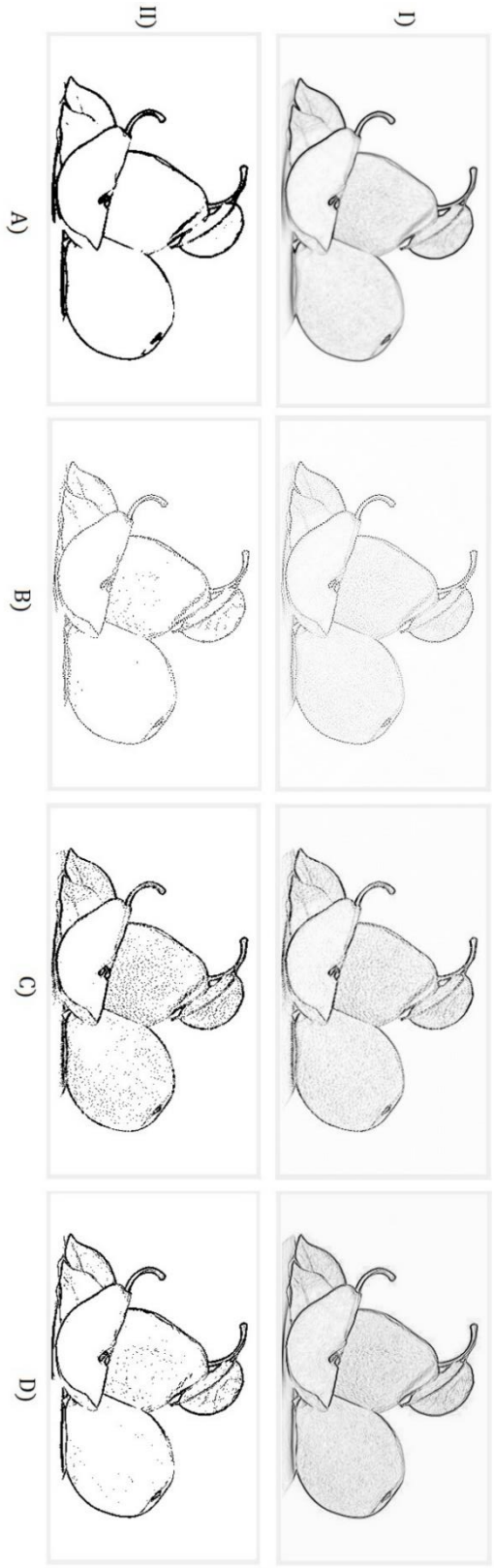

Figura 4.55 Resultados con imagen de peras en distintos casos (A, B, C, D), I) Al finalizar el proceso, II) Después de un umbral.

C) El algoritmo de hormigas utilizando 8 caminos posibles, donde para actualizar las feromonas se favorece al mejor camino. En este caso la imagen cuenta con más detalle, solo que al usar una longitud de dos pixeles este favorece a cualquier cambio significativo de este tamaño, por lo que presenta un poco de ruido en la imagen con umbral. El tiempo total para procesar la imagen es de 4.22s.

D) La identificación de bordes con la red de memristores a 4 caminos; como en los otros casos, también se utiliza una longitud de 2 pixeles, pero a diferencia del algoritmo de hormigas, este favorece aún más el detalle en la imagen, por lo que no tiene tanto contraste como en A), pero si es más detallada. El tiempo para procesar toda la imagen es de 50.24s.

Otra de las pruebas se realizó con la *Figura* 4.56, donde en A) se observa la imagen a color, la cual se encuentra en formato PNG, con una resolución de 220X220 pixeles, y un tamaño de 88KB, los resultados obtenidos para los 4 se pueden observar en la *Figura* 4.57.

Figura 4.56 Lenna A) imagen original, B) imagen en escala de grises.

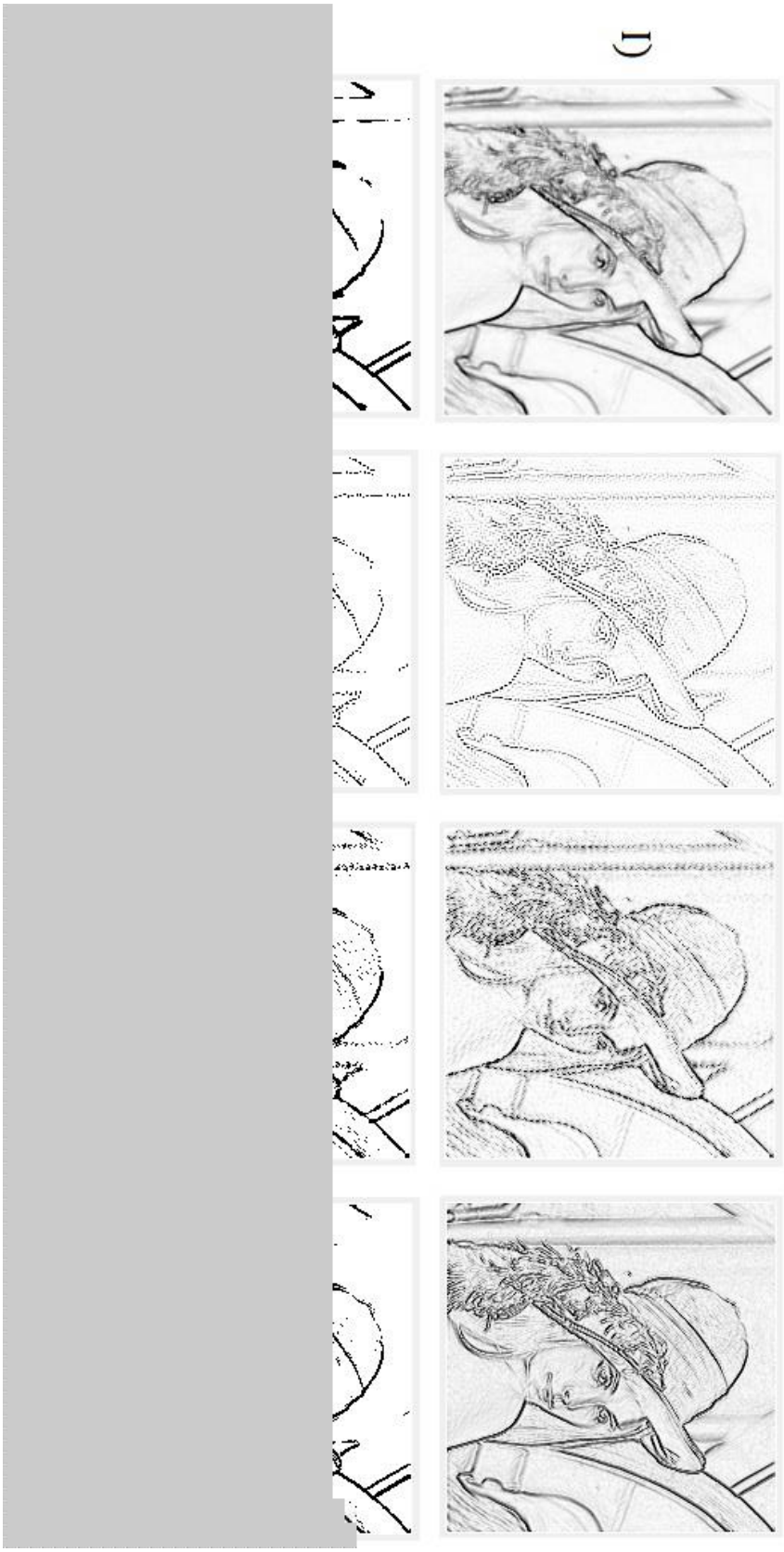

Figura 4.57 Resultados Lenna en distintos casos (A,B,C,D),I) Al finalizar el proceso, II) Después de un umbral.

Como en el ejemplo anterior, en la *Figura* 4.57 se observan los casos:

A) Algoritmo de hormigas a 4 caminos posibles con dos pixeles de longitud, actualizando las todas feromonas de la misma manera, con un tiempo de procesado de 1.51s, donde la imagen tiene tanto contraste que se torna borrosa y poco detallada.
B) Algoritmo de hormigas con 4 caminos a dos pixeles de longitud, donde la actualización de las feromonas favorece al mejor camino; el proceso es culminado en 1.76s. Para este caso la imagen favorece caminos cortos, por lo que se visualiza a trazos.
C) Algoritmo de hormigas utilizando 8 caminos posibles con una longitud de dos pixeles, actualizando las feromonas favoreciendo al mejor camino; el tiempo de procesado de la imagen es de 3.25s. Se obtienen una imagen detallada, pero con demasiado ruido.
D) La detección de bordes utilizando memristores, simulando 4 caminos con 2 pixeles de longitud; el tiempo de procesado es de 21.78s. Este caso presenta la mejor imagen obtenida antes y después del umbral, ya que el detalle es máximo sin tornar borrosa la imagen al dar contraste.

Como último ejemplo, es presentada la *Figura* 4.58, la cual se encuentra en formato JPG, con una resolución de 510x341 pixeles, un tamaño de 56 KB y donde se aprecia que el fondo se encuentra desenfocado.

Figura 4.58 A) Imagen a color con fondo desenfocado, B) imagen en escala de grises

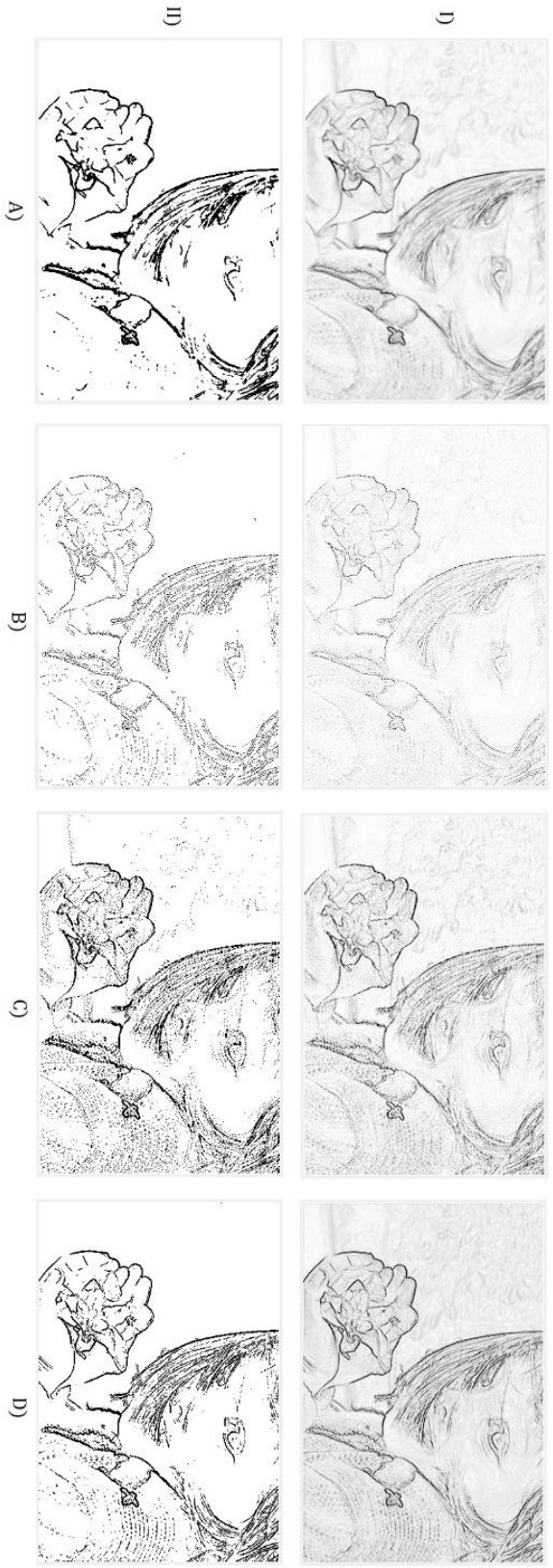

Figura 4.59 Resultados de identificación de bordes para los casos (A, B, C, D), I) al finalizar el proceso, II) después de un umbral.

En la *Figura* 4.59, se aprecian los resultados para los 4 casos en el mismo orden que el ejemplo anterior:

A) Algoritmo de hormigas 4 caminos, actualizando feromonas sin distinguir el mejor camino, procesado en. 3.28s.
B) Algoritmo de hormigas 4 caminos, diferenciando el mejor camino al momento de actualizar las feromonas, procesado en 3.49s.
C) Algoritmo de hormigas utilizando 8 caminos, donde se favorece al mejor camino al actualizar las feromonas, donde el tiempo utilizado para procesar la imagen es de 5.47s.
D) Detección de bordes utilizando 4 caminos de dos pixeles de longitud, simulado con memristores, el proceso es terminado en 78.6s, obteniendo el mejor resultado.

Estos ejemplos demuestran que efectivamente, los arreglos de memristores pueden ser utilizados en la detección de bordes como se hace con el algoritmo de hormigas, además de que los mejores resultados en detalle son logrados solo con 4 caminos simulados con memristores; en cambio, el mejor contraste se obtiene con el algoritmo de hormigas, pero solo en figuras simples, en imágenes muy detalladas como Lenna o la imagen desenfocada, los memristores obtienen siempre un mejor resultado.

Los parámetros que se utilizaron para los ejemplos son los mismos presentados al inicio de esta sección. El tiempo de procesado puede variar dependiendo de las especificaciones de la computadora utilizada; en este caso los programas fueron compilados en Matlab R2018a, con un procesador i7 920 a 2.67GHz y 8GB en RAM.

4.9 Resumen

En este capítulo, se detalló el modelo del memristor a utilizar en las distintas aplicaciones, además de comprobar su correcto funcionamiento a través de la verificación de los 3 fingerprints. También, se comprobó el correcto desempeño de la neurona pulsada, optando por un modelo simple, con bajo impacto en el consumo de recursos.

Utilizando estos dos módulos, se realizaron cuatro ejemplos: SNN al reproducir el experimento de Pávlov y SNN para el reconocimiento de caracteres en una matriz dinámica, donde se comprobó el buen comportamiento del memristor al sustituir la sinapsis y el algoritmo STDP en una rede neuronal pulsada (SNN), también, se comparó el desempeño de las redes memristivas vs el algoritmo de hormigas, al optimizas rutas, y en el procesamiento de imágenes, obteniendo resultados favorables para los memristores. Verificando así, la capacidad del modelo para adaptarse según lo requieran las implementaciones.

4.10 Conclusiones

El modelo del memristor a 32 bits en punto flotante, consume alrededor del 10% de los recursos del FPGA, por lo que debe ser optimizado para implementar varios memristores.

La neurona pulsada prácticamente no consume recurso del FPGA, pero es un modelo muy simple, por lo que hay te tenerlo en cuenta al momento de diseñar una aplicación especifica.

La equivalencia del algoritmo de hormigas con las redes memristivas se logra a través de un parámetro extra en la ecuación del memristor, por lo que se concluye que este modelo cuenta con cierto grado de plasticidad.

El reconocimiento de caracteres solo fue simulado, ya que al necesitar una gran cantidad de memristores los recursos del FPGA no son suficientes. Una alternativa seria multiplexar datos en el FPGA, pero se perdería la ventaja del paralelismo.

4.11 Referencias

[1] T. Saigusa, A. Tero, T. Nakagaki, and Y. Kuramoto, "*Amoeba Anticipate Periodic Events*" Phys. Rev. Lett. **100**, 018101, 2008.

[2] Yu. V. Pershin, S. La Fontaine and M. Di Ventra*," Memristive Model of Amoeba's learning*" Phys. Rev. E, vol. 80, pp. 021926/1-6, 2009.

[3] V. Ntinas, I. Vourkas, A. Abusleme, G. C. Sirakoulis and A. Rubio, "*Experimental Study of Artificial Neural Networks Using a Digital Memristor Simulator*", in IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 10, pp. 5098-5110, 2018.*

[4] C. Yang, H. Choi, S. Park, M. P. Sah, H. Kim, and L. O. Chua, "*A memristor emulator as a replacement of a real memristor*", *Semicond. Sci. Technol.*, vol. 30, no. 1, p. 015007, 2014.

[5] Pershin, Yuriy & Di Ventra, Massimiliano. (2009), "*Experimental demonstration of associative memory with memristive neural networks"*, Nature Precedings. 4. 10.1038/npre.2009.3258.1.

[6] Y. V. Pershin and M. Di Ventra, "*Memcomputing and swarm intelligence*," arXiv preprint arXiv: 1408.6741, 2014.

[7] Anna V. Baterina and C. Oppus. 2010, "*Image edge detection using ant colony optimization"*, WSEAS Trans. Sig. Proc. 6, pp. 58-67, 2010.

[8] J. Tian, W. Yu and S. Xie, "*An ant colony optimization algorithm for image edge detection*," 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, Hong Kong, 2008, pp. 751-756, 2008.*

[9] Z. Pajouhi and K. Roy, "*Image Edge Detection Based on Swarm Intelligence Using Memristive Networks*", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 9, pp. 1774-1787, 2018.

Capítulo 5 Conclusiones Generales

Con base en los ejemplos presentados en el capítulo 4, así como los resultados obtenidos, se concluye que el modelo del memristor bipolar con umbral controlado por voltaje, es viable para utilizarse en otras implementaciones, se demostró que es capaz de ajustarse a los distintos usos que se le dieron.

Gracias a la versatilidad del modelo, resulta útil modificarlo a conveniencia para reproducir resultados de modelos más complejos. Los recursos que utiliza pueden optimizarse dependiendo de la aplicación a realizar, esto se demostró en la implementación del modelo en System Generator en el FPGA.

Implementar el modelo del memristor en distintas aplicaciones fue un éxito, este modelo puede ser utilizado en diseños futuros que requieran ser utilizados en un FPGA.

La neurona pulsada simplificada funciono correctamente, además esta fue optimizada de tal manera que los recursos consumidos sean mínimos. Esto junto con el memristor, permitió realizar 3 implementaciones en el FPGA (Pavlov´s dog, Matriz dinámica, optimización de caminos).

Adicionalmente, se comprobó la propuesta que presenta el memristor como un análogo del algoritmo de hormigas, por medio de simulaciones numéricas en Matlab y las implementaciones en el FPGA.

Finalmente, se puede concluir que el procesamiento de imágenes con redes memristivas es mejor que con el clásico algoritmo de hormigas, pero hay que tener en cuenta que el tiempo también es mayor, alrededor de 10-15 veces mas tardado, esto dependiendo de las características del equipo en donde se realicen las simulaciones.

Anexos

Apéndice A, modelos del memristor simulados.

Sección: 1 modelo "HP".

Modelo del memristor propuesto por los investigadores de Hewlett-Packard.

```
clc, clear, close all;

t = 0:0.0001:0.6;           %time
w0=2*pi*5;
v=zeros(1,length(t));
for a=1: length(t)
    if(a<=round(length(t)/2))
    v(a) = 1.*(sin(w0*t(a))).^2;
    else
    v(a) = -1.*(sin(w0*t(a))).^2;
    end
 end

%plot(v);

Roff=38000;
Ron=100;
Rin=11000;
D=10*(10.^-9);              % 10nm = 10*10^-7 cm
uv=10.^-14;                 % cm^2 s^-1 v^-1

DR=Roff-Ron;
k=(uv*Ron)/(D^2);
p=10;
%
X0=(Roff-Rin)/(DR);
Rm=Ron*X0+Roff*(1-X0);
I=v(1)/Rm;
wi=X0*D;

x=zeros(1,size(t,2));
x(1)=X0;

M=zeros(1,size(t,2));
M(1)=Rm;

i=zeros(1,size(t,2));
i(1)=I;

dw=zeros(1,size(t,2));
dw(1)=wi;
w=dw;

fx=zeros(1,size(t,2));
fx(1)=1-(2*X0-1).^(2*p);
```

```
for j=2:size(t,2);
x(j)=w(j-1)/D;
M(j)=Ron*x(j)+Roff*(1-x(j));
i(j)=v(j)/M(j);
fx(j)=1-(2*x(j)-1).^(2*p);
dw(j)=k*D*i(j)*fx(j);
w(j)=(dw(j)*(t(2)-t(1)))+w(j-1);

end

[tt,vv,ii] = plotyy(t,v,t,i);                          %[AX,H1,H2] =
plotyy(...)
 title('Memristor Voltaje y Corriente vs. Time')
 xlabel('Time')
 set(vv,'color','b');
 set(ii,'color','g');
 axes(tt(1));
 ylabel('Voltaje')
 axes(tt(2));
 axis ([ t(1) t(end) -0.00021 0.00021]);
 set(tt(2),'YColor','r');
 ylabel('Corriente','color','r')

figure, plot(t,(w/D));
xlabel('time');
ylabel('w/D')
title('X')

%plot(v,i);
figure, plot(v,i);
xlabel('Voltage');
ylabel('Current')
title('Memristor Corriente vs. Voltaje')
```

Sección 2: modelo usado para la emulación de la ameba.

Modelo del memristor con umbral controlado por voltaje, utilizado en todas las aplicaciones.

```
clc, clear, close all;

t =0:0.0001:2;          %time
w0=2*pi*1;              % VELOCIDAD ANGULAR DE LA ONDA

vte=0.1.*sin(w0*t);
vbe=0;
v =vte-vbe;

%v=-1.2:0.01:1.2;
vt=0.035;
```

```
a=0;
b=-19.6e7;

roff=1e6;
ron=400;

dr=b.*v+0.5*(a-b)*(abs(v+vt)-abs(v-vt));

r_old=1e6;

for n=1: size(dr,2)
    m=(dr(n)*0.0001)+r_old;
    if(m>=roff)
        r(n)=roff;
    elseif(m<=ron)
        r(n)=ron;
    else
        r(n)=m;
       r_old=r(n);
    end

    i(n)=v(n)/r(n);
end

% plot(v,dr,'r');
% grid on;

%plot(v,i);
figure, plot(v,i);
grid on;
xlabel('Voltage');
ylabel('Current')
title('Memristor Corriente vs. Voltaje')
```

Apéndice B, simulación redes de memristores.

Sección 1: simulación 2 caminos.

Simulación, elección de mejor camino de dos posibles, utilizando memristores.

```
clc, clear, close all;

t =0:0.0001:0.1;        %time
i=6e-6;                 %corriente aplicada

vbe1=0;                 %voltaje electrodo inferior
vt=0.035;               %voltaje de umbral
a=0;                    %alfa
b=-19.6e7;              %beta
dt=t(2);                %tiempo de integración;
```

```
roff=1e6;                %memristancia máxima
ron=400;                 %memristancia mínima

r_old1=100e3;            %R inicial 1
r_old2=100e3;            %R inicial 2
r_old3=101e3;            %R inicial 3

for n=1: size(t,2)

    G1(n)=1/r_old1;          %memductancia 1
    G2(n)=1/r_old2;          %memductancia 2
    G3(n)=1/r_old3;          %memductancia 3

    Gs=(1/(r_old2+r_old3));

    RT=1/(G1(n)+Gs);         %R total
    v=RT*i;                  %V aplicado

    i1=v*G1(n);              %corriente 1
    i2=v*Gs;                 %corriente 2
    i3=i2;                   %corrientre para resistencias en serie

    v1=r_old1*i1;            %voltaje en el memristor 1
    v2=r_old2*i2;            %voltaje en el memristor 2
    v3=r_old2*i3;            %voltaje en el memristor 3

    dr1=(b.*v1+0.5*(a-b)*(abs(v1+vt)-abs(v1-vt)))+(50*r_old1);
%cambio de resistencia 1
    dr2=(b.*v2+0.5*(a-b)*(abs(v2+vt)-abs(v2-vt)))+(50*r_old2);
%cambio de resistencia 2
    dr3=(b.*v3+0.5*(a-b)*(abs(v3+vt)-abs(v3-vt)))+(50*r_old3);
%cambio de resistencia 3

    m1(n)=(dr1*dt)+r_old1;                  %memristancia 1
    if(m1(n)>=roff)                         %limite superior
        m1(n)=roff;
    elseif(m1(n)<=ron)                      %limite inferior
        m1(n)=ron;
    else
    end

    m2(n)=(dr2*dt)+r_old2;                  %memristancia 2
    if(m2(n)>=roff)                         %limite superior
        m2(n)=roff;
    elseif(m2(n)<=ron)                      %limite inferior
        m2(n)=ron;
    else
    end

    m3(n)=(dr3*dt)+r_old3;                  %memristancia 3
    if(m3(n)>=roff)                         %limite superior
        m3(n)=roff;
    elseif(m3(n)<=ron)                      %limite inferior
```

```
        m3(n)=ron;
    else
    end

    r_old1=m1(n);                               %actualización r1
    r_old2=m2(n);                               %actualización r2
    r_old3=m3(n);                               %actualización r3
end

figure, plot(m1);
hold on;
grid on;
plot(m2);
plot(m3);
xlabel('iteraciones');
ylabel('resistencia');
title('Memristores cambio de resistencias internas')

figure, plot(G1);
hold on;
grid on;
plot(G2);
plot(G3);
xlabel('iteraciones');
ylabel('conductancia');
title('Memristores cambio de resistencias internas');
```

Sección 2: simulación 4 caminos.

Simulación red de memristores eligiendo el mejor camino entre 4 posibles.

```
clc, clear, close all;

t =0:0.0001:0.1;        %time
it=6e-6;                %corriente aplicada

vbe1=0;                 %voltaje electrodo inferior
vt=0.035;               %voltaje de umbral
a=0;                    %alfa
b=-19.6e7;              %beta
dt=t(2);                %tiempo de integración;

roff=1e6;               %memristancia máxima
ron=400;                %memristancia mínima

%        1     2     3     4     5     6     7     8
r_old=[100e3 101e3 100e3 150e3 100e3 200e3 100e3 250e3];            %R
inicial 1,2,3,4,5,6,7,8

for n=1: size(t,2)

    for x=1: size(r_old,2)                      %obtención de las
siguientes conductancias
```

```
        G(x,n)=1/r_old(x);                          %g=1/res 1,2,3,4,5,6,7,8
    end

    y=1;
    for x=1: size(r_old,2)/2
        Gs(x)=(1/(r_old(y)+r_old(y+1)));            %conductancias en serie
Gs 1,2,3,4
        y=y+2;
    end

    RT=1/(Gs(1)+Gs(2)+Gs(3)+Gs(4));                 %R total
    v=RT*it;                                        %V aplicado

    y=1;
    for x=1: size(r_old,2)/2
        i(y)=v*Gs(x);                               %corriente 1,3,5,7
        i(y+1)=v*Gs(x);                             %corriente 2,4,6,8 para
resistencias en serie(i1=12,i3=i4)
        y=y+2;
    end

    for x=1: size(r_old,2)
     v(x)=r_old(x)*i(x);                            %voltaje en el memristor
1,2,3,4,5,6,7,8
    end

    for x=1: size(r_old,2)
    dr(x)=(b.*v(x)+0.5*(a-b)*(abs(v(x)+vt)-abs(v(x)-vt)))+(50*r_old(x));
%cambio de resistencia 1,2,3,4,5,6,7,8
    end

    for x=1: size(r_old,2)
        m(x,n)=(dr(x)*dt)+r_old(x);                 %memristancia
1,2,3,4,5,6,7,8
        if(m(x,n)>=roff)                            %limite superior
            m(x,n)=roff;
        elseif(m(x,n)<=ron)                         %limite inferior
            m(x,n)=ron;
        else
        end
    end

    for x=1: size(r_old,2)
    r_old(x)=m(x,n);                                %actualización
r1,r2,r3,r4,r5,r6,r7,r8
    end
end

figure, plot(m(1,:));                               %plot memristancias
hold on;
grid on;
for x=2: size(r_old,2)                              %plot 7 restantes
    plot(m(x,:));
end
xlabel('iteraciones');
```

```
ylabel('resistencia');

figure, plot(G(1,:));                          %plot memductancias
hold on;
grid on;
for x=2: size(r_old,2)                         %plot 7 restantes
    plot(G(x,:));
end
xlabel('iteraciones');
ylabel('conductancia');
```

Apéndice C, procesamiento de imágenes.

Sección 1: algoritmo de hormigas a 4 caminos.

Algoritmo de hormigas a 4 caminos con dos pixeles de longitud cada uno, donde las feromonas son actualizadas de la misma manera.

```
clc, clear, close all;

%p=imread('pimientos.jpg');
%p=imread('frutos.jpg');
%p=imread('peras.jpg');
%p=imread('elefante.jpg');
%p=imread('manzana.jpg');
%p=imread('Lenna.png');
%p=imread('Peppers.tiff');
%p=imread('cuadros.jpg');
%p=imread('cinvestav.jpg');
%p=imread('c3.png');
p=imread('t3.jpg');
gray=rgb2gray(p);

figure
imshow(p);                          % 1 Mostrar figura a color

figure
imshow(gray);                       % 2 Mostrar figura en escala de
grises

fil=size(gray,1);
col=size(gray,2);

edge=zeros(fil,col)+0.0039;         %para que los pixeles de los
bordes(tengan un valor) +0.0039   ||||||
 ini=zeros(fil,col);
for f=2:1:fil-1
    for c=2:1:col-1
```

```
        %edge(f,c)=abs(gray(f,c-1)-gray(f,c+1))+abs(gray(f-1,c)-
gray(f+1,c));        %////inicialización
        edge(f,c)=abs(gray(f,c-1)-gray(f,c+1))+abs(gray(f-1,c)-
gray(f+1,c))+abs(gray(f,c+1)-gray(f,c-1))+abs(gray(f+1,c)-gray(f-1,c));
        %edge(f,c)=abs(gray(f-1,c-1)-gray(f+1,c+1))+abs(gray(f-1,c)-
gray(f+1,c))+abs(gray(f-1,c+1)-gray(f+1,c-1))+abs(gray(f,c-1)-
gray(f,c+1));
        ini(f,c)=255-edge(f,c);
           if(edge(f,c)==0)
%||||||
           edge(f,c)=(edge(f,c)+1)/255;      %evitando la división entre
cero               %||||||
           else
%||||||
            edge(f,c)=(edge(f,c))/255;          %normalizando los valores
de la inicialización
           end
%||||||

    end
end


L=2;                                %////////////// Lonfitud del recorrido
tau=zeros(fil,col)+0.01;               %////////////// Valores iniciales
de las feromonas(1 ó 0.01)

a=1;        %alfa
b=1;        %beta
p=0.001;    %parametro de olvido de feromonas
Q=1;
v=1;

for iteracion=1:1:8
    for f=2:1:fil-1             %limitando filas para no llegar a los
bordes
        for c=2:1:col-1         %evitando llegar a los bordes de las
columnas

            Lepath_u=1/edge(f,c);             %/////////////camino arriba
            Lepath_d=1/edge(f,c);             %/////////////camino abajo
            Lepath_r=1/edge(f,c);             %/////////////camino
derecha
            Lepath_l=1/edge(f,c);             %/////////////camino
izquierda


            for step=0:1:L-1        %

                Lepath_u=(1/edge(f-step,c))+Lepath_u;
%/////simulación de los caminos horizontales y verticales
                Lepath_d=1/edge(f+step,c)+Lepath_d;         %Aquí
calculamos la longitud hacia los vecinos desde un pixel
                Lepath_r=1/edge(f,c+step)+Lepath_r;         %longitud
hasta el pixel de la derecha
```

```
                Lepath_l=1/edge(f,c-step)+Lepath_l;          %longitud
hasta el pixel de la izquierda
            end

            den=((tau(f-
1,c)^a)*(1/Lepath_u)^b)+((tau(f+1,c)^a)*(1/Lepath_d)^b)+((tau(f,c+1)^a)*(
1/Lepath_r)^b)+((tau(f,c-1)^a)*(1/Lepath_l)^b);     %denominador para
todos según la ecuacion 3

            Ppath_u=((tau(f-1,c)^a)*(1/Lepath_u)^b)/den;     %probabilidad
arriba
            Ppath_d=((tau(f+1,c)^a)*(1/Lepath_d)^b)/den;     %probabilidad
abajo
            Ppath_r=((tau(f,c+1)^a)*(1/Lepath_r)^b)/den;     %probabilidad
derecha
            Ppath_l=((tau(f,c-1)^a)*(1/Lepath_l)^b)/den;     %probabilidad
izquierda

            tau(f-1,c)=(1-p)*tau(f-1,c)+((v*Q)/Lepath_u);
            tau(f+1,c)=(1-p)*tau(f+1,c)+((v*Q)/Lepath_d);
            tau(f,c+1)=(1-p)*tau(f,c+1)+((v*Q)/Lepath_r);
            tau(f,c-1)=(1-p)*tau(f,c-1)+((v*Q)/Lepath_l);


        end
    end
end

figure
imshow(uint8(ini));                         % 3 Mostar inicializacion


tau=255-tau*37.5;                           %tau=tau*31.68;
figure
imshow(uint8(tau));                         % 4 Mostar feromonas mapeadas

for i=1:1:fil
    for j=1:1:col
       if(tau(i,j)<200)
           tau(i,j)=0;
       else
           tau(i,j)=255;
       end
    end
end

 figure
 imshow(uint8(tau));                        % 5 Mostrar feromonas mapeadas
despues del umbral
```

Sección 2: algoritmo de hormigas a 4 caminos, modificado.

Algoritmo de hormigas con 4 caminos dos pixeles de longitud actualizando las feromonas del mejor camino de manera distinta.

```
clc, clear, close all;

%p=imread('pimientos.jpg');
%p=imread('frutos.jpg');
%p=imread('peras.jpg');
%p=imread('elefante.jpg');
%p=imread('manzana.jpg');
%p=imread('Lenna.png');
%p=imread('Peppers.tiff');
p=imread('t3.jpg');
gray=rgb2gray(p);

figure
imshow(p);                              % 1 Mostrar figura a color

figure
imshow(gray);                           % 2 Mostrar figura en escala de
grises

fil=size(gray,1);
col=size(gray,2);

edge=zeros(fil,col)+0.0039;             %para que los pixeles de los
bordes(tengan un valor) +0.0039   ||||||
 ini=zeros(fil,col);
for f=2:1:fil-1
    for c=2:1:col-1
        %edge(f,c)=abs(gray(f,c-1)-gray(f,c+1))+abs(gray(f-1,c)-
gray(f+1,c));        %////inicialización
        edge(f,c)=abs(gray(f,c-1)-gray(f,c+1))+abs(gray(f-1,c)-
gray(f+1,c))+abs(gray(f,c+1)-gray(f,c-1))+abs(gray(f+1,c)-gray(f-1,c));
        %edge(f,c)=abs(gray(f-1,c-1)-gray(f+1,c+1))+abs(gray(f-1,c)-
gray(f+1,c))+abs(gray(f-1,c+1)-gray(f+1,c-1))+abs(gray(f,c-1)-
gray(f,c+1));
        ini(f,c)=255-edge(f,c);
           if(edge(f,c)==0)
%||||||
           edge(f,c)=(edge(f,c)+1)/255;      %evitando la división entre
cero               %||||||
           else
%||||||
            edge(f,c)=(edge(f,c))/255;          %normalizando los valores
de la inicialización
           end
%||||||

    end
end
```

```
L=2;                                %////////////// Lonfitud del recorrido
tau=zeros(fil,col)+0.01;               %////////////// Valores iniciales
de las feromonas(1 ó 0.01)

a=1;        %alfa
b=1;        %beta
p=0.001;    %parametro de olvido de feromonas
Q=1;
v=1;

for iteracion=1:1:8
    for f=2:1:fil-1             %limitando filas para no llegar a los
bordes
        for c=2:1:col-1         %evitando llegar a los bordes de las
columnas

            Lepath_u=0;             %//////////////camino arriba
            Lepath_d=0;             %//////////////camino abajo
            Lepath_r=0;             %//////////////camino derecha
            Lepath_l=0;             %//////////////camino izquierda


            for step=0:1:L-1        %

                Lepath_u=(1/edge(f-step,c))+Lepath_u;
%/////simulación de los caminos horizontales y verticales
                Lepath_d=1/edge(f+step,c)+Lepath_d;         %Aquí
calculamos la longitud hacia los vecinos desde un pixel
                Lepath_r=1/edge(f,c+step)+Lepath_r;         %longitud
hasta el pixel de la derecha
                Lepath_l=1/edge(f,c-step)+Lepath_l;         %longitud
hasta el pixel de la izquierda
            end

            den=((tau(f-
1,c)^a)*(1/Lepath_u)^b)+((tau(f+1,c)^a)*(1/Lepath_d)^b)+((tau(f,c+1)^a)*(
1/Lepath_r)^b)+((tau(f,c-1)^a)*(1/Lepath_l)^b);    %denominador para
todos según la ecuacion 3

            Ppath_u=((tau(f-1,c)^a)*(1/Lepath_u)^b)/den;    %probabilidad
arriba
            Ppath_d=((tau(f+1,c)^a)*(1/Lepath_d)^b)/den;    %probabilidad
abajo
            Ppath_r=((tau(f,c+1)^a)*(1/Lepath_r)^b)/den;    %probabilidad
derecha
            Ppath_l=((tau(f,c-1)^a)*(1/Lepath_l)^b)/den;    %probabilidad
izquierda

            if(Ppath_u>Ppath_d)&&(Ppath_u>Ppath_r)&&(Ppath_u>Ppath_l)
%actualización de las feromonas(arriba)
                tau(f-1,c)=(1-p)*tau(f-1,c)+((v*Q)/Lepath_u);  %arriba
                tau(f+1,c)=(1-p)*tau(f+1,c);   %abajo
                tau(f,c+1)=(1-p)*tau(f,c+1);   %derecha
                tau(f,c-1)=(1-p)*tau(f,c-1);   %izquierda
```

```
            elseif(Ppath_d>Ppath_u)&&(Ppath_d>Ppath_r)&&(Ppath_d>Ppath_l)
%actualización de las feromonas(abajo)
                tau(f-1,c)=(1-p)*tau(f-1,c);    %arriba
                tau(f+1,c)=(1-p)*tau(f+1,c)+((v*Q)/Lepath_d);   %abajo
                tau(f,c+1)=(1-p)*tau(f,c+1);    %derecha
                tau(f,c-1)=(1-p)*tau(f,c-1);    %izquierda

            elseif(Ppath_r>Ppath_u)&&(Ppath_r>Ppath_d)&&(Ppath_r>Ppath_l)
%actualización de las feromonas(derecha)
                tau(f-1,c)=(1-p)*tau(f-1,c);    %arriba
                tau(f+1,c)=(1-p)*tau(f+1,c);    %abajo
                tau(f,c+1)=(1-p)*tau(f,c+1)+((v*Q)/Lepath_r);   %derecha
                tau(f,c-1)=(1-p)*tau(f,c-1);    %izquierda


            elseif(Ppath_l>Ppath_u)&&(Ppath_l>Ppath_d)&&(Ppath_l>Ppath_r)
%actualización de las feromonas(izquierda)
                 tau(f-1,c)=(1-p)*tau(f-1,c);    %arriba
                 tau(f+1,c)=(1-p)*tau(f+1,c);    %abajo
                 tau(f,c+1)=(1-p)*tau(f,c+1);    %derecha
                 tau(f,c-1)=(1-p)*tau(f,c-1)+((v*Q)/Lepath_l);
%izquierda

            else
                tau(f-1,c)=(1-p)*tau(f-1,c)+((v*Q)/Lepath_u);
                tau(f+1,c)=(1-p)*tau(f+1,c)+((v*Q)/Lepath_d);
                tau(f,c+1)=(1-p)*tau(f,c+1)+((v*Q)/Lepath_r);
                tau(f,c-1)=(1-p)*tau(f,c-1)+((v*Q)/Lepath_l);

            end

        end
    end
end

figure
imshow(uint8(ini));                     % 3 Mostar inicializacion


tau=255-tau*37.5;
figure
imshow(uint8(tau));                     % 4 Mostar feromonas mapeadas

for i=1:1:fil
    for j=1:1:col
       if(tau(i,j)<200)
           tau(i,j)=0;
       else
           tau(i,j)=255;
       end
    end
end
```

```
 figure
 imshow(uint8(tau));                    % 5 Mostrar feromonas mapeadas
despues del umbral
```

Sección 3: algoritmo de hormigas a 8 caminos.

Algoritmo de hormigas utilizando 8 caminos a dos pixeles de longitud, donde las feromonas son actualizadas diferenciando el mejor camino.

```
clc, clear, close all;

%p=imread('pimientos.jpg');
%p=imread('frutos.jpg');
%p=imread('peras.jpg');
%p=imread('elefante.jpg');
%p=imread('manzana.jpg');
%p=imread('Lenna.png');
%p=imread('Peppers.tiff');
p=imread('t3.jpg');
gray=rgb2gray(p);

figure
imshow(p);                             % 1 Mostrar figura a color

figure
imshow(gray);                          % 2 Mostrar figura en escala de
grises

fil=size(gray,1);
col=size(gray,2);

edge=zeros(fil,col)+0.0039;            %para que los pixeles de los
bordes(tengan un valor) +0.0039   ||||||
 ini=zeros(fil,col);
for f=2:1:fil-1
    for c=2:1:col-1
        %edge(f,c)=abs(gray(f,c-1)-gray(f,c+1))+abs(gray(f-1,c)-
gray(f+1,c));         %////inicialización
        edge(f,c)=abs(gray(f,c-1)-gray(f,c+1))+abs(gray(f-1,c)-
gray(f+1,c))+abs(gray(f,c+1)-gray(f,c-1))+abs(gray(f+1,c)-gray(f-1,c));
        %edge(f,c)=abs(gray(f-1,c-1)-gray(f+1,c+1))+abs(gray(f-1,c)-
gray(f+1,c))+abs(gray(f-1,c+1)-gray(f+1,c-1))+abs(gray(f,c-1)-
gray(f,c+1));
        ini(f,c)=255-edge(f,c);
           if(edge(f,c)==0)
%||||||
           edge(f,c)=(edge(f,c)+1)/255;      %evitando la división entre
cero               %||||||
           else
%||||||
            edge(f,c)=(edge(f,c))/255;          %normalizando los valores
de la inicialización
```

```
          end
%||||||

    end
end


L=2;                                %////////////// Lonfitud del recorrido
tau=zeros(fil,col)+0.01;               %////////////// Valores iniciales
de las feromonas(1 ó 0.01)

a=1;        %alfa
b=1;        %beta
p=0.001;    %parametro de olvido de feromonas
Q=1;
v=1;

for iteracion=1:1:8
    for f=2:1:fil-1             %limitando filas para no llegar a los
bordes
        for c=2:1:col-1         %evitando llegar a los bordes de las
columnas

            Lepath_u=0;             %//////////////camino arriba
            Lepath_d=0;             %//////////////camino abajo
            Lepath_r=0;             %//////////////camino derecha
            Lepath_l=0;             %//////////////camino izquierda
            Lepath_ur=0;            %//////////////camino arriba-derecha
            Lepath_ul=0;            %//////////////camino arriba-izquierda
            Lepath_dr=0;            %//////////////camino abajo-derecha
            Lepath_dl=0;            %//////////////camino abajo-izquierda

            for step=0:1:L-1        %

                Lepath_u=(1/edge(f-step,c))+Lepath_u;
%/////simulación de los caminos horizontales y verticales
                Lepath_d=1/edge(f+step,c)+Lepath_d;          %Aquí
calculamos la longitud hacia los vecinos desde un pixel
                Lepath_r=1/edge(f,c+step)+Lepath_r;          %longitud
hasta el pixel de la derecha
                Lepath_l=1/edge(f,c-step)+Lepath_l;          %longitud
hasta el pixel de la izquierda
                Lepath_ul=1/edge(f-step,c-step)+Lepath_ul;   %longitud
hasta el pixel de la arriba-izquierda
                Lepath_ur=1/edge(f-step,c+step)+Lepath_ur;   %longitud
hasta el pixel de la arriba-derecha
                Lepath_dr=1/edge(f+step,c+step)+Lepath_dr;   %longitud
hasta el pixel de la abajo-derecha
                Lepath_dl=1/edge(f+step,c-step)+Lepath_dl;   %longitud
hasta el pixel de la abajo-izquierda
            end

            den=((tau(f-
1,c)^a)*(1/Lepath_u)^b)+((tau(f+1,c)^a)*(1/Lepath_d)^b)+((tau(f,c+1)^a)*(
1/Lepath_r)^b)+((tau(f,c-1)^a)*(1/Lepath_l)^b)+((tau(f-1,c-
```

```
1)^a)*(1/Lepath_ul)^b)+((tau(f-
1,c+1)^a)*(1/Lepath_ur)^b)+((tau(f+1,c+1)^a)*(1/Lepath_dr)^b)+((tau(f+1,c
-1)^a)*(1/Lepath_dl)^b);    %denominador para todos según la ecuacion 3

            Ppath_u=((tau(f-1,c)^a)*(1/Lepath_u)^b)/den;
%probabilidad arriba
            Ppath_d=((tau(f+1,c)^a)*(1/Lepath_d)^b)/den;
%probabilidad abajo
            Ppath_r=((tau(f,c+1)^a)*(1/Lepath_r)^b)/den;
%probabilidad derecha
            Ppath_l=((tau(f,c-1)^a)*(1/Lepath_l)^b)/den;
%probabilidad izquierda
            Ppath_ul=((tau(f-1,c-1)^a)*(1/Lepath_ul)^b)/den;
%probabilidad arriba-izquierda
            Ppath_ur=((tau(f-1,c+1)^a)*(1/Lepath_ur)^b)/den;
%probabilidad arriba-derecha
            Ppath_dr=((tau(f+1,c+1)^a)*(1/Lepath_dr)^b)/den;
%probabilidad abajo-derecha
            Ppath_dl=((tau(f+1,c-1)^a)*(1/Lepath_dl)^b)/den;
%probabilidad abajo-izquierda


if(Ppath_u>Ppath_d)&&(Ppath_u>Ppath_r)&&(Ppath_u>Ppath_l)&&(Ppath_u>Ppath
_ul)&&(Ppath_u>Ppath_ur)&&(Ppath_u>Ppath_dr)&&(Ppath_u>Ppath_dl)
%actualización de las feromonas(arriba)
                tau(f-1,c)=(1-p)*tau(f-1,c)+((v*Q)/Lepath_u);  %arriba
                tau(f+1,c)=(1-p)*tau(f+1,c);       %abajo
                tau(f,c+1)=(1-p)*tau(f,c+1);       %derecha
                tau(f,c-1)=(1-p)*tau(f,c-1);       %izquierda
                tau(f-1,c-1)=(1-p)*tau(f-1,c-1);   %arriba-izquierda
                tau(f-1,c+1)=(1-p)*tau(f-1,c+1);   %arriba-derecha
                tau(f+1,c+1)=(1-p)*tau(f+1,c+1);   %abajo-derecha
                tau(f+1,c-1)=(1-p)*tau(f+1,c-1);   %abajo-izquierda


elseif(Ppath_d>Ppath_u)&&(Ppath_d>Ppath_r)&&(Ppath_d>Ppath_l)&&(Ppath_u>P
path_ul)&&(Ppath_u>Ppath_ur)&&(Ppath_u>Ppath_dr)&&(Ppath_u>Ppath_dl)
%actualización de las feromonas(abajo)
                tau(f-1,c)=(1-p)*tau(f-1,c);       %arriba
                tau(f+1,c)=(1-p)*tau(f+1,c)+((v*Q)/Lepath_d);  %abajo
                tau(f,c+1)=(1-p)*tau(f,c+1);       %derecha
                tau(f,c-1)=(1-p)*tau(f,c-1);       %izquierda
                tau(f-1,c-1)=(1-p)*tau(f-1,c-1);   %arriba-izquierda
                tau(f-1,c+1)=(1-p)*tau(f-1,c+1);   %arriba-derecha
                tau(f+1,c+1)=(1-p)*tau(f+1,c+1);   %abajo-derecha
                tau(f+1,c-1)=(1-p)*tau(f+1,c-1);   %abajo-izquierda


elseif(Ppath_r>Ppath_u)&&(Ppath_r>Ppath_d)&&(Ppath_r>Ppath_l)&&(Ppath_u>P
path_ul)&&(Ppath_u>Ppath_ur)&&(Ppath_u>Ppath_dr)&&(Ppath_u>Ppath_dl)
%actualización de las feromonas(derecha)
                tau(f-1,c)=(1-p)*tau(f-1,c);       %arriba
                tau(f+1,c)=(1-p)*tau(f+1,c);       %abajo
                tau(f,c+1)=(1-p)*tau(f,c+1)+((v*Q)/Lepath_r);  %derecha
                tau(f,c-1)=(1-p)*tau(f,c-1);       %izquierda
                tau(f-1,c-1)=(1-p)*tau(f-1,c-1);   %arriba-izquierda
```

```
                tau(f-1,c+1)=(1-p)*tau(f-1,c+1);    %arriba-derecha
                tau(f+1,c+1)=(1-p)*tau(f+1,c+1);    %abajo-derecha
                tau(f+1,c-1)=(1-p)*tau(f+1,c-1);    %abajo-izquierda


elseif(Ppath_l>Ppath_u)&&(Ppath_l>Ppath_d)&&(Ppath_l>Ppath_r)&&(Ppath_u>P
path_ul)&&(Ppath_u>Ppath_ur)&&(Ppath_u>Ppath_dr)&&(Ppath_u>Ppath_dl)
%actualización de las feromonas(izquierda)
                tau(f-1,c)=(1-p)*tau(f-1,c);        %arriba
                tau(f+1,c)=(1-p)*tau(f+1,c);        %abajo
                tau(f,c+1)=(1-p)*tau(f,c+1);        %derecha
                tau(f,c-1)=(1-p)*tau(f,c-1)+((v*Q)/Lepath_l);
%izquierda
                tau(f-1,c-1)=(1-p)*tau(f-1,c-1);    %arriba-izquierda
                tau(f-1,c+1)=(1-p)*tau(f-1,c+1);    %arriba-derecha
                tau(f+1,c+1)=(1-p)*tau(f+1,c+1);    %abajo-derecha
                tau(f+1,c-1)=(1-p)*tau(f+1,c-1);    %abajo-izquierda


elseif(Ppath_ul>Ppath_u)&&(Ppath_ul>Ppath_d)&&(Ppath_ul>Ppath_r)&&(Ppath_
ul>Ppath_l)&&(Ppath_ul>Ppath_ur)&&(Ppath_ul>Ppath_dr)&&(Ppath_ul>Ppath_dl
)   %actualización de las feromonas(arruba-izquierda)
                tau(f-1,c)=(1-p)*tau(f-1,c);        %arriba
                tau(f+1,c)=(1-p)*tau(f+1,c);        %abajo
                tau(f,c+1)=(1-p)*tau(f,c+1);        %derecha
                tau(f,c-1)=(1-p)*tau(f,c-1);        %izquierda
                tau(f-1,c-1)=(1-p)*tau(f-1,c-1)+((v*Q)/Lepath_ul);
%arriba-izquierda
                tau(f-1,c+1)=(1-p)*tau(f-1,c+1);    %arriba-derecha
                tau(f+1,c+1)=(1-p)*tau(f+1,c+1);    %abajo-derecha
                tau(f+1,c-1)=(1-p)*tau(f+1,c-1);    %abajo-izquierda


elseif(Ppath_ur>Ppath_u)&&(Ppath_ur>Ppath_d)&&(Ppath_ur>Ppath_r)&&(Ppath_
ur>Ppath_l)&&(Ppath_ur>Ppath_ul)&&(Ppath_ur>Ppath_dr)&&(Ppath_ur>Ppath_dl
)   %actualización de las feromonas(arruba-derecha)
                tau(f-1,c)=(1-p)*tau(f-1,c);        %arriba
                tau(f+1,c)=(1-p)*tau(f+1,c);        %abajo
                tau(f,c+1)=(1-p)*tau(f,c+1);        %derecha
                tau(f,c-1)=(1-p)*tau(f,c-1);        %izquierda
                tau(f-1,c-1)=(1-p)*tau(f-1,c-1);    %arriba-izquierda
                tau(f-1,c+1)=(1-p)*tau(f-1,c+1)+((v*Q)/Lepath_ur);
%arriba-derecha
                tau(f+1,c+1)=(1-p)*tau(f+1,c+1);    %abajo-derecha
                tau(f+1,c-1)=(1-p)*tau(f+1,c-1);    %abajo-izquierda


elseif(Ppath_dr>Ppath_u)&&(Ppath_dr>Ppath_d)&&(Ppath_dr>Ppath_r)&&(Ppath_
dr>Ppath_l)&&(Ppath_dr>Ppath_ul)&&(Ppath_dr>Ppath_ur)&&(Ppath_dr>Ppath_dl
)   %actualización de las feromonas(abajo-derecha)
                tau(f-1,c)=(1-p)*tau(f-1,c);                %arriba
                tau(f+1,c)=(1-p)*tau(f+1,c);                %abajo
                tau(f,c+1)=(1-p)*tau(f,c+1);                %derecha
                tau(f,c-1)=(1-p)*tau(f,c-1);                %izquierda
```

```
                tau(f-1,c-1)=(1-p)*tau(f-1,c-1);            %arriba-
izquierda
                tau(f-1,c+1)=(1-p)*tau(f-1,c+1);            %arriba-
derecha
                tau(f+1,c+1)=(1-p)*tau(f+1,c+1)+((v*Q)/Lepath_dr);
%abajo-derecha
                tau(f+1,c-1)=(1-p)*tau(f+1,c-1);            %abajo-
izquierda



elseif(Ppath_dl>Ppath_u)&&(Ppath_dl>Ppath_d)&&(Ppath_dl>Ppath_r)&&(Ppath_
dl>Ppath_l)&&(Ppath_dl>Ppath_ul)&&(Ppath_dl>Ppath_ur)&&(Ppath_dl>Ppath_dr
)    %actualización de las feromonas(abajo-izquierda)
                tau(f-1,c)=(1-p)*tau(f-1,c);        %arriba
                tau(f+1,c)=(1-p)*tau(f+1,c);        %abajo
                tau(f,c+1)=(1-p)*tau(f,c+1);        %derecha
                tau(f,c-1)=(1-p)*tau(f,c-1);        %izquierda
                tau(f-1,c-1)=(1-p)*tau(f-1,c-1);    %arriba-izquierda
                tau(f-1,c+1)=(1-p)*tau(f-1,c+1);    %arriba-derecha
                tau(f+1,c+1)=(1-p)*tau(f+1,c+1);    %abajo-derecha
                tau(f+1,c-1)=(1-p)*tau(f+1,c-1)+((v*Q)/Lepath_dl);
%abajo-izquierda

            else
                tau(f-1,c)=(1-p)*tau(f-1,c)+((v*Q)/Lepath_u);
%arriba
                tau(f+1,c)=(1-p)*tau(f+1,c)+((v*Q)/Lepath_d);
%abajo
                tau(f,c+1)=(1-p)*tau(f,c+1)+((v*Q)/Lepath_r);
%derecha
                tau(f,c-1)=(1-p)*tau(f,c-1)+((v*Q)/Lepath_l);
%izquierda
                tau(f-1,c-1)=(1-p)*tau(f-1,c-1)+((v*Q)/Lepath_ul);
%arriba-izquierda
                tau(f-1,c+1)=(1-p)*tau(f-1,c+1)+((v*Q)/Lepath_ur);
%arriba-derecha
                tau(f+1,c+1)=(1-p)*tau(f+1,c+1)+((v*Q)/Lepath_dr);
%abajo-derecha
                tau(f+1,c-1)=(1-p)*tau(f+1,c-1)+((v*Q)/Lepath_dl);
%abajo-izquierda

            end

        end
    end
end

figure
imshow(uint8(ini));                     % 3 Mostar inicializacion


tau=255-tau*37.5;
figure
imshow(uint8(tau));                     % 4 Mostar feromonas mapeadas
```

```
for i=1:1:fil
    for j=1:1:col
       if(tau(i,j)<200)
           tau(i,j)=0;
       else
           tau(i,j)=255;
       end
    end
end

 figure
 imshow(uint8(tau));                    % 5 Mostrar feromonas mapeadas
despues del umbral
```

Sección 4: red de memristores a 4 caminos.

Detección de bordes utilizando una red de memristores a 4 caminos con una longitud de dos pixeles.

```
clc, clear, close all;
%////////////////////////////////////////////////////////////////////////
/////
%Parte inicializacíon
%////////////////////////////////////////////////////////////////////////
/////
%p=imread('frutos.jpg');
%p=imread('elefante.jpg');
%p=imread('peras.jpg');
%p=imread('Peppers.tiff');
%p=imread('Lenna.png');                 %Imagen a obtener bordes
p=imread('t3.jpg');
figure
imshow(p);                              % 1 Mostrar figura a color


gray=rgb2gray(p);                       %Conversión a B y N
figure                                  %Para una ventana independiente
imshow(gray);                           %Visualización de la imagen en BN

fil=size(gray,1);                       %Número de filas
col=size(gray,2);                       %Número de columnas

edge=zeros(fil,col)+0.0039;             %Para que los pixeles de los
bordes(tengan un valor) +0.0039   ||||||
ini=zeros(fil,col);                     %matriz para ver la
inicialización

for f=2:1:fil-1
    for c=2:1:col-1
```

```
        %edge(f,c)=abs(gray(f,c-1)-gray(f,c+1))+abs(gray(f-1,c)-
gray(f+1,c));            %////inicialización
        edge(f,c)=abs(gray(f,c-1)-gray(f,c+1))+abs(gray(f-1,c)-
gray(f+1,c))+abs(gray(f,c+1)-gray(f,c-1))+abs(gray(f+1,c)-gray(f-1,c));
        %edge(f,c)=abs(gray(f-1,c-1)-gray(f+1,c+1))+abs(gray(f-1,c)-
gray(f+1,c))+abs(gray(f-1,c+1)-gray(f+1,c-1))+abs(gray(f,c-1)-
gray(f,c+1));
        ini(f,c)=255-edge(f,c);                 %para poder visualizar la
matriz en BN 0-255
        if(edge(f,c)==0)
%||||||
            edge(f,c)=(edge(f,c)+1);            %evitando la división
entre cero       %||||||
        else
%||||||
            edge(f,c)=(edge(f,c));              %normalizando los valores
de la inicialización 0-1
        end
%||||||
    end
end

figure                                          %ventana indepndiente
imshow(uint8(ini));                             %matriz inicializada
vista de 0-255


%////////////////////////////////////////////////////////////////////////
/////
%Parte memristores
%////////////////////////////////////////////////////////////////////////
/////
tau=zeros(fil,col);                 %matriz para valores finales de la
imagen

t =0:0.0001:0.01;       %time
it=3e-6;                %corriente aplicada

vbe1=0;                 %voltaje electrodo inferior
vt=0.035;               %voltaje de umbral
a=0;                    %alfa
b=-19.6e7;              %beta
dt=t(2);                %tiempo de integración;

roff=1e6;               %memristancia máxima
ron=400;                %memristancia mínima


for f=2:1:fil-1                     %limitando filas para no llegar a los
bordes
        for c=2:1:col-1             %evitando llegar a los bordes de las
columnas

%////////////////////////////////////////////////////////////////////////
/////
```

```
            %Para para pasar la inicialización de los edge a las
memristancias iniciales

%////////////////////////////////////////////////////////////////////////
/////
            %          1          2          3          4         5
6         7          8
            r_old=[edge(f,c) edge(f+1,c) edge(f,c) edge(f-1,c) edge(f,c)
edge(f,c+1) edge(f,c) edge(f,c-1)];            %R inicial 1,2,3,4,5,6,7,8
            r_old=r_old*10000;

            for n=1: size(t,2)

                for x=1: size(r_old,2)                          %obtención de
las siguientes conductancias
                    G(x)=1/r_old(x);                            %g=1/res
1,2,3,4,5,6,7,8
                end

                y=1;
                for x=1: size(r_old,2)/2
                    Gs(x)=(1/(r_old(y)+r_old(y+1)));
%conductancias en serie Gs 1,2,3,4
                    y=y+2;                                      %para poder
sumar r1+r2, r3+r4...
                end

                RT=1/(Gs(1)+Gs(2)+Gs(3)+Gs(4));                 %R total
                v=RT*it;                                        %V aplicado

                y=1;
                for x=1: size(r_old,2)/2
                    i(y)=v*Gs(x);                               %corriente
1,3,5,7
                    i(y+1)=v*Gs(x);                             %corriente
2,4,6,8 para resistencias en serie(i1=12,i3=i4)
                    y=y+2;                                      %para generar
las 8 corrientes
                end

                for x=1: size(r_old,2)
                 v(x)=r_old(x)*i(x);                            %voltaje en
el memristor 1,2,3,4,5,6,7,8
                end

                for x=1: size(r_old,2)
                dr(x)=(b.*v(x)+0.5*(a-b)*(abs(v(x)+vt)-abs(v(x)-
vt)))+(50*r_old(x));    %cambio de resistencia 1,2,3,4,5,6,7,8
                end

                for x=1: size(r_old,2)
                    m(x)=(dr(x)*dt)+r_old(x);                   %memristancia
1,2,3,4,5,6,7,8
                    if(m(x)>=roff)                              %limite
superior
```

```
                        m(x)=roff;
                    elseif(m(x)<=ron)                          %limite
inferior
                        m(x)=ron;
                    else
                    end
                end

                for x=1: size(r_old,2)
                r_old(x)=m(x);
%actualización r1,r2,r3,r4,r5,r6,r7,r8
                end
            end
            tau(f+1,c)=m(2);                                   %asignación
de los nuevos valores a la matriz de la imagen down
            tau(f-1,c)=m(4);                                   %asignación
pixel up
            tau(f,c+1)=m(6);                                   %asignación
pixel right
            tau(f,c-1)=m(8);                                   %asignación
pixel left
        end
end
tau=255-tau*2.55e-4;                                           %inversión de
colores y mapeo de 0-255

figure
imshow(uint8(tau));
%visualización de la imagen final

for i=1:1:fil
    for j=1:1:col
       if(tau(i,j)<180)
           tau(i,j)=0;
       else
           tau(i,j)=255;
       end
    end
end

figure
imshow(uint8(tau));
%visualización de la imagen final despues del umbral
```

Apéndice D, comparación procesamiento de imágenes.

Sección 1: detección de bordes para distintas imágenes.

En las siguientes imágenes, se muestran en la parte superior, la imagen original a color, y su versión en escala de grises.

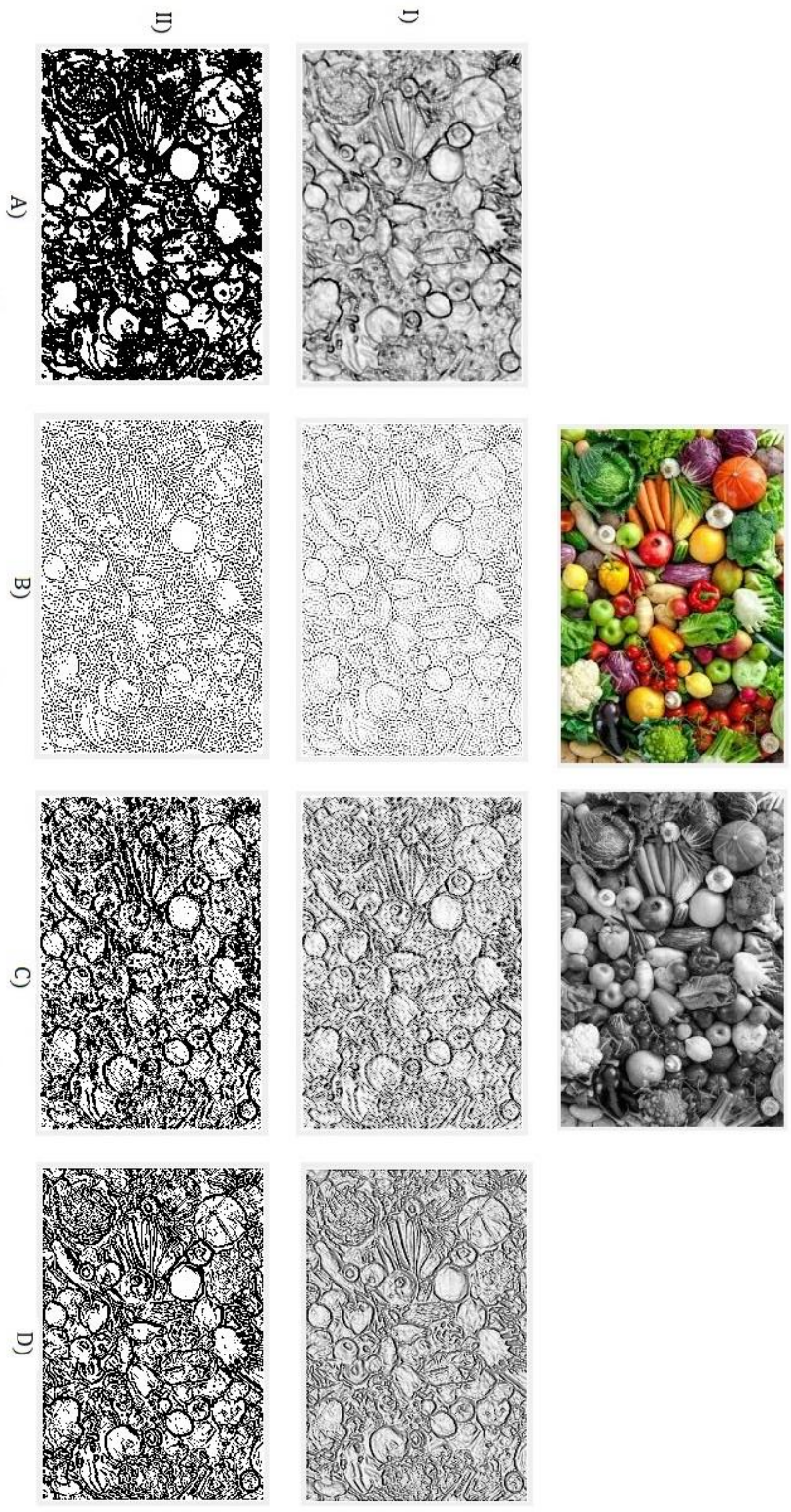

Figura: Frutos y verduras, en formato JPG.

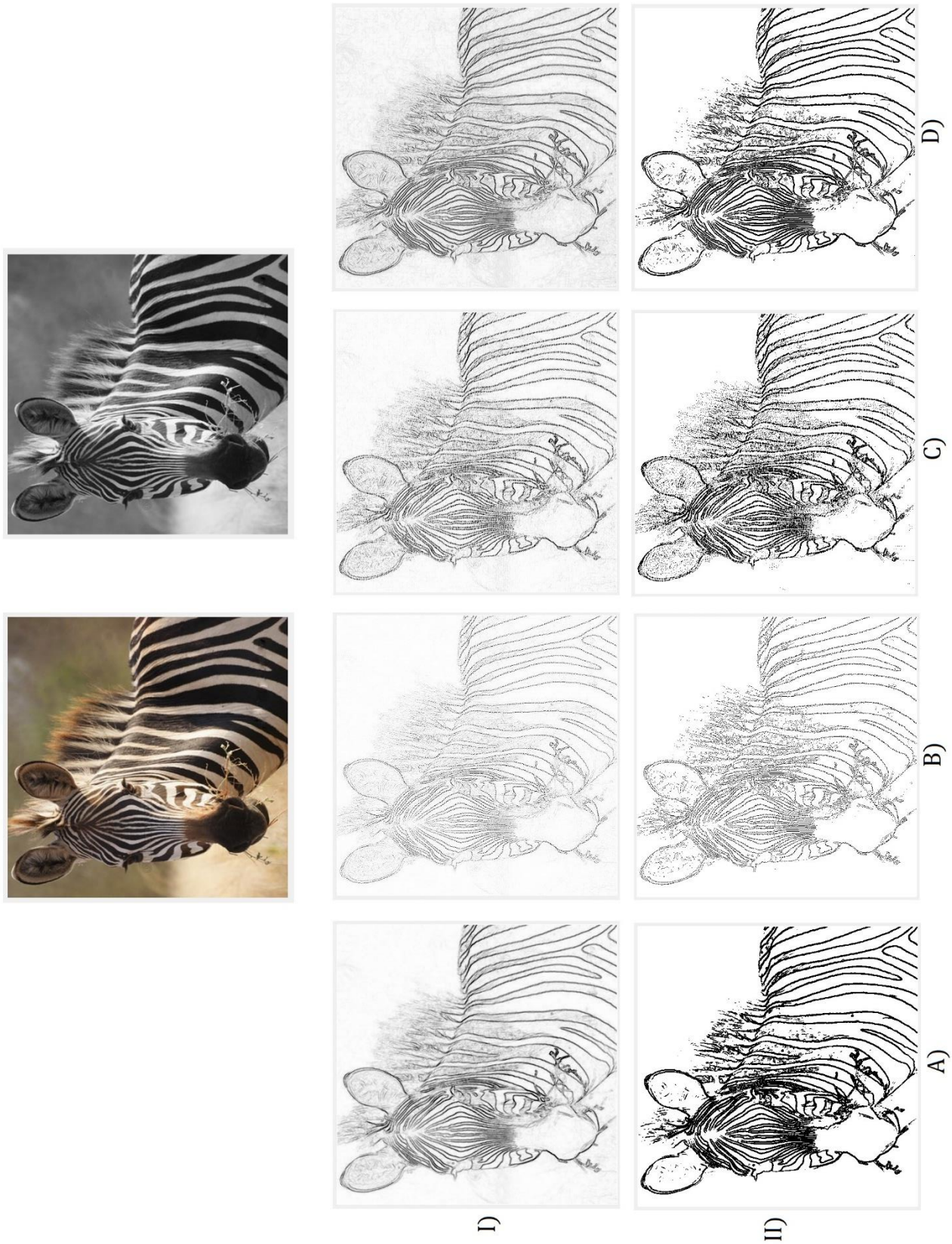

Figura: Cebra con fondo desenfocado, en formato JPG.

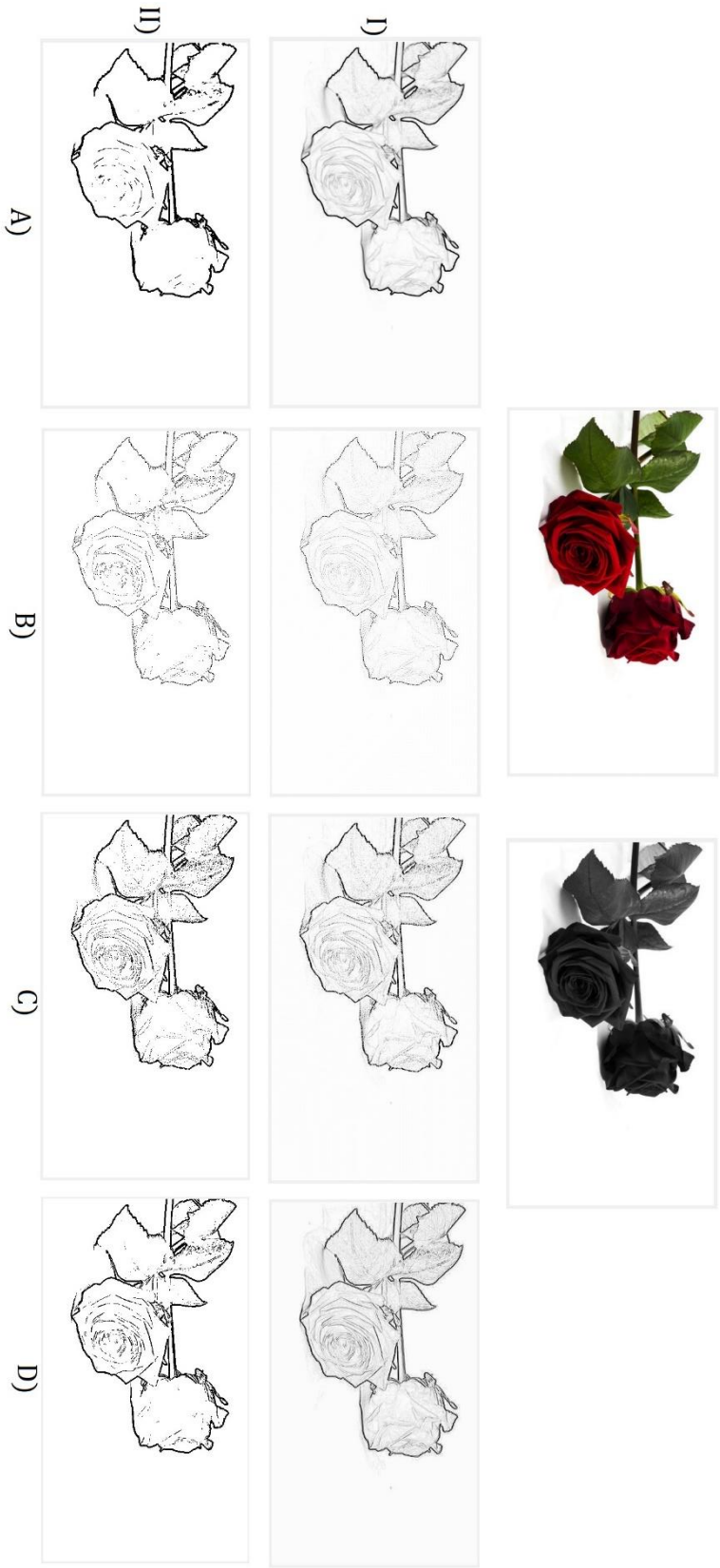

Figura: Rosa con fondo blanco, en formato JPG.

En todas las imágenes anteriores, como en las presentadas en el capítulo 4:

I) Indica los resultados al finalizar el procesamiento.

II) Indica los resultados después de aplicar el umbral al proceso previo.

A) Son los resultados con el algoritmo de hormigas, utilizando 4 caminos a 2 pixeles de longitud, actualizando las feromonas sin elegir el mejor.

B) Muestra los resultados con el algoritmo de hormigas, utilizando 4 caminos a dos pixeles de longitud, donde el mejor camino actualiza las feromonas de manera distinta.

C) Son los resultados del algoritmo de hormigas, utilizando 8 caminos a dos pixeles de longitud, actualizando las feromonas del mejor camino de manera distinta.

D) Son los resultados utilizando una red de memristores, donde se usaron 4 caminos a dos pixeles de longitud.