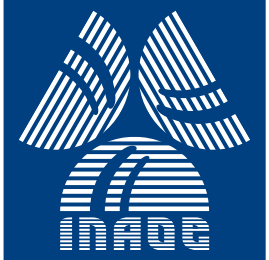


Diseño de Sistemas Inteligentes Híbridos VLSI con base en Técnicas Difuso-Evolutivas

por

M. en C. Juan López Hernández

Tesis sometida como requisito parcial para obtener el grado de

DOCTOR EN CIENCIAS EN LA ESPECIALIDAD DE ELECTRÓNICA

en el

Instituto Nacional de Astrofísica, Óptica y Electrónica

Junio 2008

Tonantzintla, Puebla

Supervisada por:

Dr. J. Alejandro Díaz Méndez, INAOE

Dr. Carlos A. Reyes García, INAOE

Con todo mi amor y agradecimiento a

Kissai

Sara Danae

Ivanka Lorelei

AGRADECIMIENTOS

Al Consejo Nacional de Ciencia y Tecnología CONACYT, por su apoyo durante la realización de mis estudios de Doctorado.

Al Instituto Nacional de Astrofísica, Óptica y Electrónica, INAOE, por brindarme la oportunidad de realizar mis estudios de Doctorado.

Un agradecimiento especial a mis directores de tesis Dr. J. Alejandro Díaz Méndez y Dr. Carlos Alberto Reyes García por su confianza, guía y apoyo durante el desarrollo de mi trabajo.

Finalmente, agradezco a mi esposa e hijas todo su sacrificio durante todo este tiempo.

Resumen

La ingeniería, ciencias computacionales y las ciencias físicas y biológicas están utilizando métodos sofisticados de la Inteligencia Computacional (IC) para la solución de problemas complejos. El incremento asociado al diálogo y a la interconexión entre las tecnologías inteligentes ha llevado a la consolidación de la inteligencia computacional y sus realizaciones prácticas: los Sistemas Inteligentes Híbridos (SIH). Los SIH pueden tener diferentes estructuras, las cuales tienen impacto en la eficiencia y exactitud con la que se resuelve el problema, por esta razón a nivel electrónico es importante optimizar el diseño de la arquitectura. El objetivo del presente trabajo es desarrollar una alternativa a aquellos problemas que requieran una realización electrónica de un SIH, de igual forma se aborda la problemática que existe al realizar estas implementaciones físicas. Se muestra el diseño en modo de corriente de un sistema Difuso-Evolutivo, quien a su vez hace uso del caos como apoyo en el proceso de adaptación evolutiva. El sistema difuso diseñado tiene la característica de poder procesar señales en tiempo continuo, de igual forma los parámetros que definen su comportamiento son programables en tiempo continuo. El sistema de aprendizaje fue diseñado con base en el algoritmo básico de las Estrategias Evolutivas donde se requiere de generadores de señales aleatorias como fuente de posibles soluciones. Es aquí donde se hace uso de la teoría del caos para diseñar generadores de ruido y poder obtener las señales aleatorias requeridas. Los circuitos caóticos se diseñaron haciendo uso de mapas logísticos iterativos. Dado que tanto el algoritmo evolutivo como el sistema caótico, son sistemas iterativos, se diseñaron circuitos que procesarán señales analógicas en modo discreto y poder controlar el flujo del procesamiento. De esta forma, se diseñaron sistemas difusos evolutivos capaces de adaptarse en línea para encontrar la solución requerida.

Abstract

Engineering, Computational, Physical and Biological Sciences are using sophisticated methods of the Computational Intelligence (CI) for the solution of complex problems. The increase associated with the dialogue and inter-connectivity between intelligent technologies has led to the consolidation of computational intelligence practices and their achievements: the Intelligent Systems Hybrids (HIS). The HIS can have different structures, which have an impact on the efficiency and accuracy with which it solves the problem, which is why it is important to optimize to an electronic level the design of the architecture. The objective of the present work is to develop an alternative to those problems which require completion of an electronic HIS, likewise addresses the problem that exists to make these physical implementations. The design in current mode of an Evolutive-Fuzzy System is shown, who in turn uses the chaos as support in the evolutionary process of adaptation. The designed system has the feature to process signals in continuous time, just as the parameters that define their behavior are continuous time programmable. The learning system was designed based on the basic Evolutive-Strategies algorithm which requires random signal generators as a source of possible solutions. It is here where he uses the chaos theory to design noise generators and to obtain the required random signals. The chaotic circuits were designed using iterated logistic maps. Since both the evolutionary algorithm as the chaotic system, are iterative systems, were designed circuits that process signals in analog and discreet mode to control the flow of processing. Thus, evolutive-fuzzy systems were designed able to adapt online to find the required solution.

Índice

Prefacio

Las técnicas de Soft-Computing (SC) o Inteligencia Computacional con la gran variedad de aplicaciones eficientes son muy fascinantes. Los problemas en ingeniería, ciencias computacionales y las ciencias físicas y biológicas están utilizando los métodos cada vez más sofisticados de la Inteligencia Computacional. Debido a los grandes requerimientos interdisciplinarios en la mayoría de las aplicaciones del mundo real, no existe un puente entre las diferentes Tecnologías Inteligentes independientes, llamadas Sistemas Difusos, Redes Neuronales, Algoritmos Evolutivos, Caos y Sistemas Basados en Conocimiento entre otras. El incremento asociado al dialogo y la interconexión entre las Tecnologías Inteligentes ha llevado a la consolidación del SC y sus implementaciones prácticas – los Sistemas Inteligentes Híbridos (SIH). Los SIH que combinan diversas técnicas de SC son necesitados ya que los problemas del mundo real involucran gran complejidad en su planteamiento y solución. Los SIH pueden tener diferentes arquitecturas, las cuales tienen impacto en la eficiencia y exactitud con la que se resuelve el problema, por esta razón es importante optimizar el diseño de la arquitectura. Las arquitecturas pueden combinar de diferentes formas las diferentes técnicas del SC para alcanzar la meta final de resolver el problema en cuestión. En el presente trabajo se muestra una alternativa a aquellos problemas que requieran una implementación en hardware de un SIH. Se analiza la problemática que existe al realizar implementaciones electrónicas de un SIH. Se observará que para obtener un buen desempeño de los sistemas diseñados es conveniente combinar diferentes técnicas del SC. Se muestra el diseño de sistema Difuso-Evolutivo, quien a su vez hace uso del Caos como apoyo en el proceso evolutivo.

Capítulo 1

Sistemas Inteligentes Híbridos

1.1 Introducción.

En la actualidad, se ha encontrado que una gran cantidad de procesos reales al ser analizados, cuentan con no-linealidades debidas a sus propiedades físicas. Este tipo de situaciones traen como consecuencia que los modelos requeridos para su análisis sean muy complicados o a veces imposibles de obtener desde un punto de vista matemático. Aunado a esto, la tarea de modelado se vuelve aun más difícil. No obstante, se ha logrado desarrollar técnicas que han dado solución a algunos de estos problemas [1-4]. Independientemente de la técnica a utilizar, siempre se obtendrá un modelo aproximado de un sistema real, dado que en estas aplicaciones puede existir incertidumbre acerca de las condiciones de operación del sistema. Hoy en día, como alternativa se utilizan un conjunto de técnicas computacionales denominadas como Computación Suave (*Soft-Computing* - SC) como herramientas de diseño y/o control y/o adaptación para estos sistemas. Entre ellas se tiene a las Redes Neuronales (RN), los Sistemas Difusos (SD) y los Algoritmos Evolutivos (AE) como las más representativas. Las RN y los SD, han resultado de gran ayuda en la tarea de modelado de sistemas. Las arquitecturas de las RN, dado que emulan la parte cognitiva del cerebro humano, permiten que los sistemas aprendan del entorno que los rodea, mientras que los SD nos ayudan a trabajar con la incertidumbre propia de entornos dinámicos o en los errores de los datos de muestra [5,6]. Por otro lado, los AE han demostrado ser de gran ayuda en el proceso de optimización de algoritmos de aprendizaje y/o sintonización [7,8] de sistemas muy complejos. El éxito de estas técnicas, no solo se debe a las cualidades y ventajas obtenidas al hacer uso de ellas de manera individual, sino también, a la posibilidad de combinarlas y así obtener sistemas más robustos. A continuación se describirán las principales características de las técnicas antes mencionadas. Además, se incluirá la teoría del Caos en esta investigación, confirmando que mientras más se tenga conocimiento de las diversas técnicas disponibles y haciendo una buena elección de ellas, al crear un sistema híbrido, es posible obtener buenos resultados.

1.2 Computación Suave (Soft-Computing).

La Computación Suave consiste de diversos paradigmas de cómputo, donde la completa integración de estas metodologías forma el núcleo de esta misma. El sinergismo le permite a la SC incorporar conocimiento humano eficientemente, lidiar con la imprecisión y la incertidumbre, y aprender a adaptarse a ambientes desconocidos o cambiantes para obtener un mejor desempeño.

1.2.1 Redes Neuronales.

Las redes neuronales (RN) artificiales son, como su nombre lo indica, redes computacionales que tratan de emular, de forma burda, las redes de celdas (neuronas) del sistema nervioso central biológico (humano). Esta emulación es una emulación de celdas por celdas (neurona por neurona, elemento por elemento) utilizando como base el conocimiento neurofisiológico de las neurona biológicas y sus redes. Así mismo, estas emulaciones difieren de la máquinas de cómputo convencionales (digitales o analógicas) que sirven para reemplazar, mejorar o dar más velocidad a la computación del cerebro humano sin tener en cuenta la organización de los elementos de cómputo y sus conexiones. La red neuronal, de hecho es una arquitectura de cómputo que permite usar operaciones de cómputo muy simples (suma, multiplicación y elementos de lógica fundamental) para resolver problemas definidos matemáticamente de forma compleja, problemas no-lineales o problemas estocásticos. Un algoritmo convencional utilizaría conjuntos complejos de ecuaciones y que se aplicarían solo a un problema dado y no a otros. Las RN son (a) computacionalmente y algorítmicamente muy simples y (b) tienen características auto-organizativas para permitirle resolver una amplia gama de problemas. Otro aspecto de las RN que es diferente y ventajoso con respecto a los algoritmos convencionales, al menos potencialmente, es su alta capacidad de cómputo paralelo. Una computadora digital convencional es una máquina secuencial. Si un solo transistor (de los millones que forman la computadora) falla, la máquina completa se detendrá. En el sistema nervioso central de un humano adulto, miles de neuronas muren cada día, de cualquier forma el funcionamiento del cerebro no es afectado totalmente, salvo cuando las células se encuentran en lugares clave (muy pocos) y mueren en grandes cantidades. Esta insensitividad al daño de cierta cantidad de celdas se debe al alto paralelismo de las redes neuronales biológicas, en contraste con el diseño secuencial de las computadoras convencionales. La

misma característica de redundancia se aplica a las RN artificiales. Sin embargo, ya que la mayoría de las RN son simuladas en computadoras digitales convencionales, este aspecto de insensitividad no se conserva. Aun así, existe una mayor disponibilidad de RN en hardware, que en términos de circuitos integrados consta de cientos e incluso miles de neuronas incluidas en un solo chip [9,10]. Cada vez es más aceptado el uso de las RN, teniendo una nueva comprensión de la forma de simplificar la programación y diseño de algoritmos para un determinado fin.

1.2.1.1 Arquitecturas.

La estructura básica y fundamental de procesamiento en toda red neuronal es la neurona simple. En la Figura 1.1 se muestra una neurona biológica, mientras que en la Figura 1.2 se muestra su contraparte artificial.

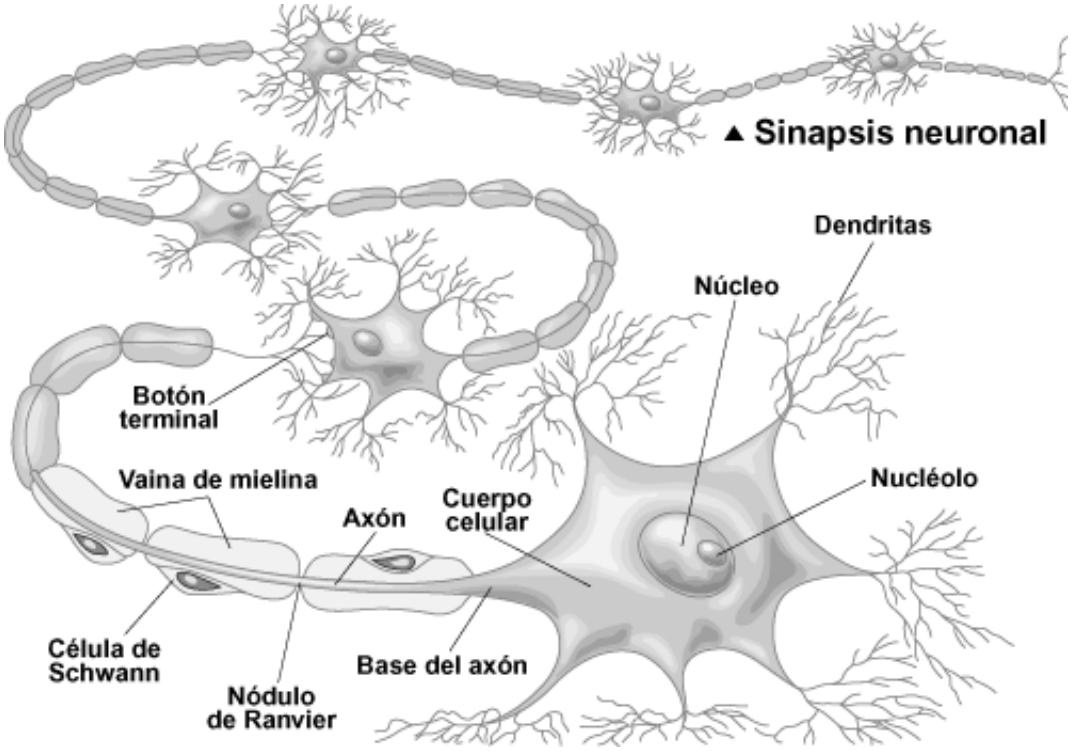


Figura 1.1 Neurona biológica.

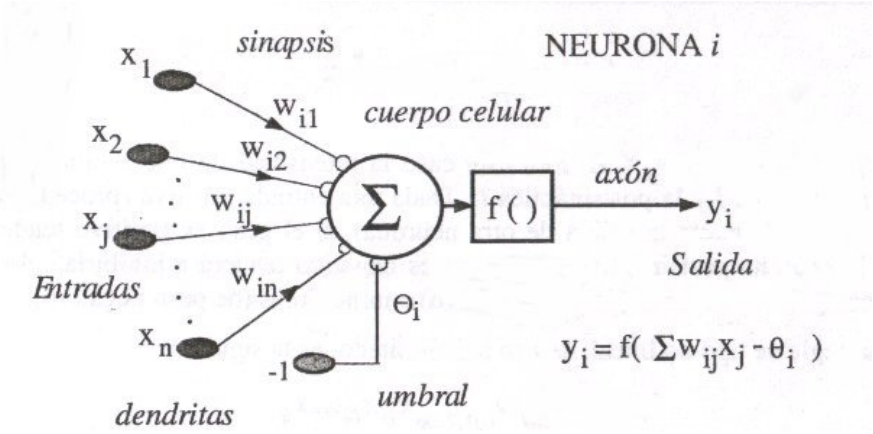


Figura 1.2 Neurona Artificial.

En la neurona artificial se dispone de *n* entradas *x*, las cuales serán pesadas o multiplicadas por sus respectivos pesos *w* emulando el proceso de sinapsis de la neurona biológica. Se suman todas las entradas para ser evaluadas por una función *f*(*) y obtener un valor de salida y_i, emulando al cuerpo celular cuando reacciona a los diferentes estímulos que llegan a el. En la tabla 1.1 se muestran diferentes funciones *f*(*) utilizadas en la simulación y diseño de RN.

Tabla 1.1 Funciones de activación para neuronas artificiales.

	Función	Rango
Identidad	$y = x$	$[-\infty, +\infty]$
Escalón	$y = signo(x)$ $y = H(x)$	{-1, +1} {0, +1}
Lineal a tramos	$y = \begin{cases} -1, & x < -l \\ x, & +l \le x \le -l \\ +1, & x > +l \end{cases}$	[-1, +1]
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	[0, +1] [-1, +1]
Gaussiana	$y = A \cdot e^{-Bx}$	[0, +1]

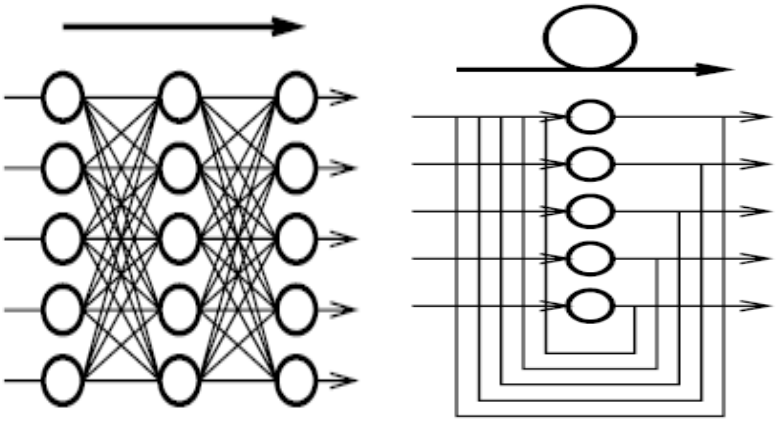
Una arquitectura que consta de una sola neurona es conocida como Perceptrón Simple. Aunque es la arquitectura más simple que existe puede ser útil al resolver algunos problemas. En arquitecturas más complejas se recomiendan dos arquitecturas de acuerdo al tipo de conexión existente entre las neuronas de la red:

Redes de Trayectoria Directa: donde el flujo de datos desde las entradas hasta las salidas es estrictamente hacia delante. El procesamiento de los datos se puede extender sobre múltiples capas de unidades de procesamiento, pero no deben existir conexiones de retroalimentación entre ellas, solo deben existir conexiones entre las salidas de las neuronas de una capa hacia las entradas de las neuronas de la siguiente capa. Este tipo de arquitectura se muestra en la Figura 1.3(a).

Redes Recurrentes: donde existen conexiones de retroalimentación. Contrario a las redes de trayectoria directa, estas cuentan con propiedades dinámicas muy importantes. En algunos casos los valores de activación de las celdas unitarias entran en un proceso de relajación tal que la red evolucionará hasta llegar a un estado estable en el cual los valores de activación no cambiarán más. En otras aplicaciones, el cambio de los valores de activación de las neuronas de salida es significativo, de tal forma que ese comportamiento dinámico constituye la salida de la red. Este tipo de arquitectura se muestra en la Figura 1.3(b).

1.2.1.2 Aprendizaje.

Una RN tiene que ser configurada de tal forma que al aplicarle un conjunto de entradas produzca (ya sea de forma directa o por un proceso de relajación) las salidas deseadas. Existen varios métodos para establecer los pesos entre las conexiones. Una forma es establecer los pesos explícitamente, utilizando conocimiento *a priori*. Otra forma es entrenar la RN aplicándole patrones de entrenamiento y permitir que se ajusten los pesos de acuerdo a algunas reglas de entrenamiento.



(a) (b)

Figura 1.3 Red de a) Trayectoria Directa, b) Recurrente.

Los métodos de aprendizaje se pueden clasificar de dos formas:

Supervisado o Asociativo: la red es entrenada aplicando pares de patrones que incluyen las salidas deseadas para las entradas especificadas. Estos pares de patrones pueden ser provistos por un agente externo o por el sistema que contiene a la red (auto-supervisada).
Sin Supervisión o Auto-Organizado: una unidad de salida es entrenada para responder a agrupaciones de patrones dentro de las entradas. En este paradigma, se supone que el sistema descubrirá iterativamente las características principales de la población de entrada. Contrario al paradigma del aprendizaje supervisado, no hay un conjunto de categorías *a priori* dentro de los cuales los patrones son clasificados; más bien el sistema debe desarrollar su propia representación de los estímulos de entrada.

La regla delta es un algoritmo de aprendizaje para RN de una sola capa. En este método se debe definir una medida del desempeño general de la red y después encontrar una forma de optimizarlo. Los algoritmos deben cambiar los pesos de tal forma que las salidas o^q lleguen a ser más y más similares a la salida deseada y^q para toda q = 1, 2, 3,…, m cuando se le presenta la entrada x^q a la red. Una medida general del desempeño puede ser:

$$E = \sum_{q=1}^{N} E^q = \sum_{q=1}^{N} \left(\frac{1}{2} \sum_{i=1}^{m} \left(y_i^q - o_i^q \right)^2 \right) \tag{1.1}$$

El objetivo es minimizar E con respecto a los pesos w_{ij} de la red, donde w_{ij} es el peso que va del nodo j a la salida i de una neurona. Para utilizar el método del gradiente descendente como método de optimización, se necesita que los w_{ij} sean diferenciables. Esto se reduce a requerir que:

$$o_i^q = f_i \left(\sum_{j=0}^{n} w_{ij} x_j^q \right) \tag{1.2}$$

sea diferenciable. Esto es, la función de activación f_i de la *i-ésima* neurona debe ser elegida tal que sea diferenciable. El error E es una función de las variables w_{ij}, donde i=1, 2, 3,…, m y j=1, 2, 3,…, n. Recuerde que el gradiente ∇E de E en el punto W con las componentes w_{ij} es el vector de las derivadas parciales $\partial E/\partial w_{ij}$. Al igual que la derivada de una función de una

variable, el gradiente siempre apunta en dirección cuesta arriba de la función E. La dirección cuesta abajo (paso descendiente) de E en W es $-\nabla E$. Entonces, para minimizar E, se debe mover proporcionalmente al negativo de ∇E, teniendo que actualizar cada peso w_{jk} como:

$$w_{jk} \rightarrow w_{jk} + \Delta w_{jk} \tag{1.3}$$

donde

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{1.4}$$

y $\eta > 0$, donde η es un número llamado razón de aprendizaje. De esta forma la regla delta nos lleva a buscar un vector de pesos w^* tal que el gradiente de E en w^* sea cero. Para una red simple de una sola neurona como el perceptrón, w^* es el mínimo absoluto de $E(w)$.

Otro algoritmo de aprendizaje popular es el algoritmo de retro-propagación, que es una generalización de la regla delta. Si se observa de cerca la regla delta para redes de una sola capa, se observa que para actualizar el peso w_{ik} cuando el patrón de aprendizaje de entrada x^q es presentado, se necesita:

$$\Delta w_{ik}^q = -\eta \delta_i^q x_k^q \tag{1.5}$$

es decir, se necesita calcular la función:

$$\delta_i^q = \left(o_i^q - y_i^q\right) f'\left(\sum_{j=0}^{n} w_{ij} x_j^q\right) \tag{1.6}$$

y esto es posible ya que se tiene a nuestra disposición el valor y_i^q que es conocido como la salida deseada en la salida del nodo i.

En el caso de redes multicapas, no es posible detectar como es que afecta el desempeño de las neuronas de las capas intermedias en el desempeño total de la red. Dado que no se dispone de patrones de entrenamiento para cada capa de la red, no es posible aplicar directamente la regla delta. Es aquí donde es necesario aplicar el método de retro-propagación. En resumen, este método se puede describir por los siguientes pasos:

1. Inicializar los pesos w_{ik} con valores aleatorios pequeños y seleccionar la razón de aprendizaje η.

2. Aplicar el patrón x^q a la capa de entrada.

3. Propagar x^q desde la entrada hasta la salida utilizando:

$$o_i = f_i\left(\sum_{k=o}^{p} w_{ik} o_k\right) \tag{1.7}$$

4. Calcular el error $E^q(w)$ en la capa de salida utilizando:

$$E^q(w) = \frac{1}{2}\sum_{i=1}^{p}\left(o_i^q - y_i^q\right)^2 \tag{1.8}$$

5. Calcular los valores δ de la capa de salida utilizando:

$$\delta_i^q = f_i'\left(\sum_{k=1}^{n} w_{ik} z_k\right)\left(o_i^q - y_i^q\right) \tag{1.9}$$

6. Calcular los valores δ de las capas ocultas propagando los valores δ hacia atrás, es decir:

$$\delta_i^q = f_i'\left(\sum_{k=0}^{n} w_{ik} x_k^q\right)\sum_{j=1}^{p} \upsilon i_j \delta_j^q \tag{1.10}$$

7. Utilizar para todo w_{ik} de la red:

$$\Delta^q w_{ik} = -\eta \delta_i^q o_k^q \tag{1.11}$$

8. $q \rightarrow q + 1$ y volver al paso 2.

En general, estos dos algoritmos (la regla delta y la retro-propagación) nos ayudan a encontrar una solución donde el error del entrenamiento tiende a ser cero. Sin embargo, dependiendo de la complejidad del problema, el algoritmo podría llevarnos a encontrar una solución donde el error sea un mínimo local y no global.

1.2.2 Sistemas Difusos.

Desde hace algunos años, el uso de la teoría de conjuntos difusos o lógica difusa en sistemas de control ha estado ganando gran popularidad, especialmente en países orientales. Desde mediados de los años 80s, los científicos Japoneses han trabajado en transformar la teoría de la lógica difusa en una realización tecnológica. Hoy en día, los sistemas de control basados en lógica difusa o simplemente *controladores de lógica difusa* (CLD), pueden encontrarse en

una creciente cantidad de productos, desde máquinas de lavado hasta lanchas, desde unidades de aire acondicionado hasta cámaras de bolsillo con auto-enfoque.

El éxito de los CLD se debe principalmente a su habilidad para trabajar con conocimiento representado en forma lingüística en lugar de representaciones matemáticas convencionales. Los ingenieros en control tradicionalmente han dependido de modelos matemáticos para realizar sus diseños. No obstante, mientras más complejo sea el sistema, menos efectivo será modelo matemático. Este es el concepto fundamental que provee motivación al desarrollo de la lógica difusa formulada por Lofti Zadeh [11], el fundador de la teoría de conjuntos difusos. Los problemas del mundo real pueden ser extremadamente complejos y los sistemas complejos son inherentemente difusos. La principal ventaja de los CLD es su habilidad de integrar la experiencia, la intuición y la heurística dentro de un mismo sistema en vez de atenerse a modelos matemáticos. Esto los hace más efectivos en aplicaciones donde los modelos existentes no están bien definidos y no son factibles.

Zadeh [11] propuso las siguientes definiciones de operaciones de conjuntos difusos como una extensión de las operaciones clásicas:

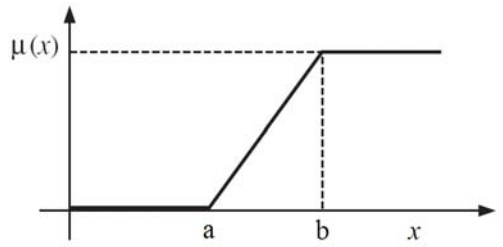
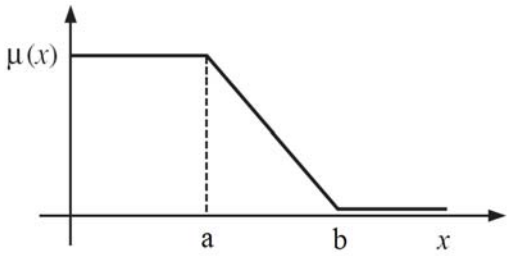
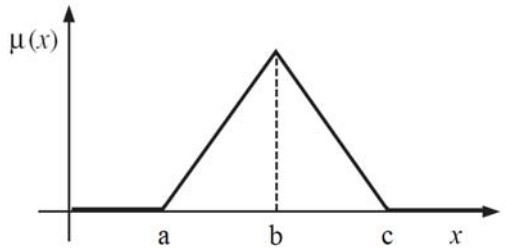
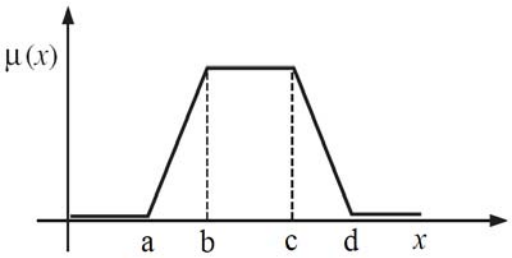
Complemento $\forall x \in X : \mu_{A'}(x) = 1 - \mu_A(x)$ (1.14)

Unión $\forall x \in X : \mu_{A \cup B}(x) = \max[\mu_A(x), \mu_B(x)]$ (1.15)

Intersección $\forall x \in X : \mu_{A \cap B}(x) = \min[\mu_A(x), \mu_B(x)]$ (1.16)

Estas definiciones forman la base de la teoría de la lógica difusa básica. La relación entre un elemento en el universo de discurso y un conjunto difuso esta definida por su función de membresía. La naturaleza exacta de la relación depende de la forma o figura de la función de membresía utilizada. En la tabla 1.2 se muestran algunas de las funciones más utilizadas dada su simplicidad matemática.

Tabla 1.2 Funciones de Membresía.

$\Gamma(x;a,b)=\begin{cases}0 & x<a\\ \dfrac{x-a}{b-a} & a\le x\le b\\ 1 & x>b\end{cases}$	
$L(x;a,b)=\begin{cases}1 & x<a\\ \dfrac{x-b}{a-b} & a\le x\le b\\ 0 & x>b\end{cases}$	
$\Lambda(x;a,b,c)=\begin{cases}0 & x<a\\ \dfrac{x-a}{b-a} & a\le x\le b\\ \dfrac{x-c}{b-c} & b\le x\le c\\ 0 & x>c\end{cases}$	
$\Pi(x;a,b,c,d)=\begin{cases}0 & x<a\\ \dfrac{x-a}{a-b} & a\le x\le b\\ 1 & b\le x\le c\\ \dfrac{x-d}{c-d} & c\le x\le d\\ 0 & x>d\end{cases}$	

Ahora se establecerá el concepto de *variable lingüística* para describir las entradas y las salidas del CLD. Una variable convencional es numérica y precisa, no es capaz de soportar la vaguedad de la teoría de conjuntos difusos. Por definición una variable lingüística esta hecha de palabras, sentencias o lenguaje natural y que son menos precisas que los números. Esto constituye un modo para aproximar la caracterización de fenómenos complejos o no completamente definidos. Por ejemplo, 'EDAD' es una variable lingüística cuyos valores pueden ser los conjuntos difusos 'JOVEN' y 'VIEJO'. Un ejemplo más común se presenta en el control difuso donde se tiene la variable lingüística 'ERROR', la cual puede

tener los *valores lingüísticos* 'POSITIVO', 'CERO' y 'NEGATIVO'. Hasta aquí se ha analizado la base del pensamiento difuso. Ahora se mostrará como es posible utilizar este tipo de razonamiento para controlar el mundo real y procesar los datos provenientes de la percepción de este.

Un *Sistema de Inferencia Difuso* (SID) es una herramienta de cómputo popular basada en los conceptos de la teoría de conjuntos difusos, reglas si-entonces difusas y razonamiento difuso, y que ha encontrado aplicaciones exitosas en una amplia variedad de campos de aplicación, tales como control automático, clasificación de datos, análisis de decisiones, sistemas expertos, predicción de series de tiempo, robótica y reconocimiento de patrones. Debido a su naturaleza multidisciplinaria, los sistemas de inferencia difusa son conocidos por muchos nombres, tales como *sistemas basados en reglas difusas*, *sistemas expertos difusos* [12], *modelo difuso* [13], *memoria asociativa difusa* [14], *control lógico difuso* [15] y/o simplemente *sistema difuso*.

La estructura básica de un SID consiste de 3 componentes conceptuales: una *base de reglas*, la cual contiene una selección de reglas difusas; una *base de datos* (o diccionario), la cual define las funciones de membresía utilizadas en las reglas difusas; y un *mecanismo de razonamiento*, el cual realiza el procedimiento de inferencia considerando las reglas y los hechos para proporcionar una conclusión o salida razonable. Se debe notar que el SID básico puede tomar valores de entrada tanto difusos como duros (que son vistos como *singletons* difusos), pero las salidas que este produce son siempre conjuntos difusos. A veces es necesario obtener una salida dura, especialmente en una situación donde el SID es utilizado como un controlador. Es por esto que se necesita un método de *desdifución* para obtener un valor duro que represente de la mejor manera a un conjunto difuso. En la Figura 1.4 se muestra un SID con salida dura, donde la línea entrecortada representa un SID básico con salida difusa y el bloque de desdifución sirve para transformar un conjunto difuso de salida en un solo valor duro. Tres tipos de sistemas de inferencia difusa han sido utilizados ampliamente, las diferencias entre estos tres tipos de SID se deben a los consecuentes de sus reglas difusas y por lo tanto, dependiendo de estos serán los métodos de agregación y desdifución.

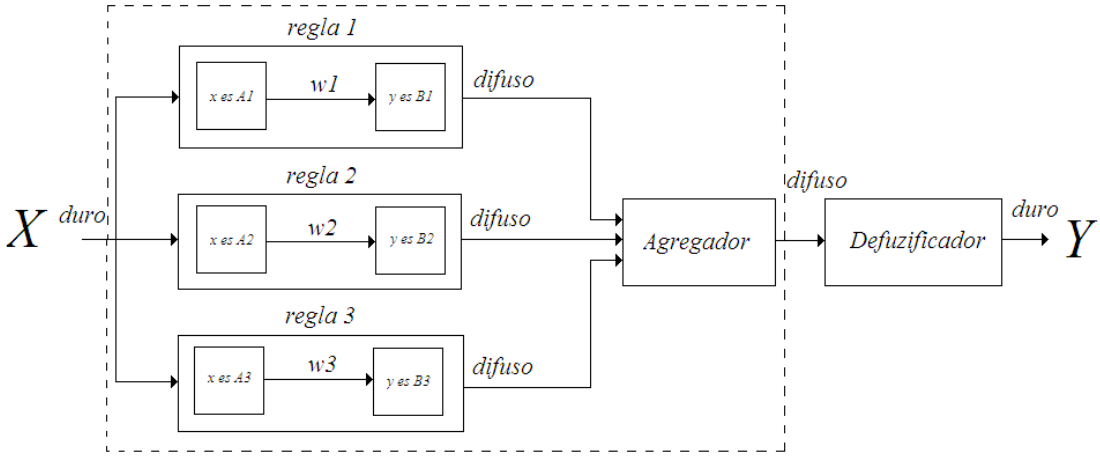


Figura 1.4 Sistema de Inferencia Difuso.

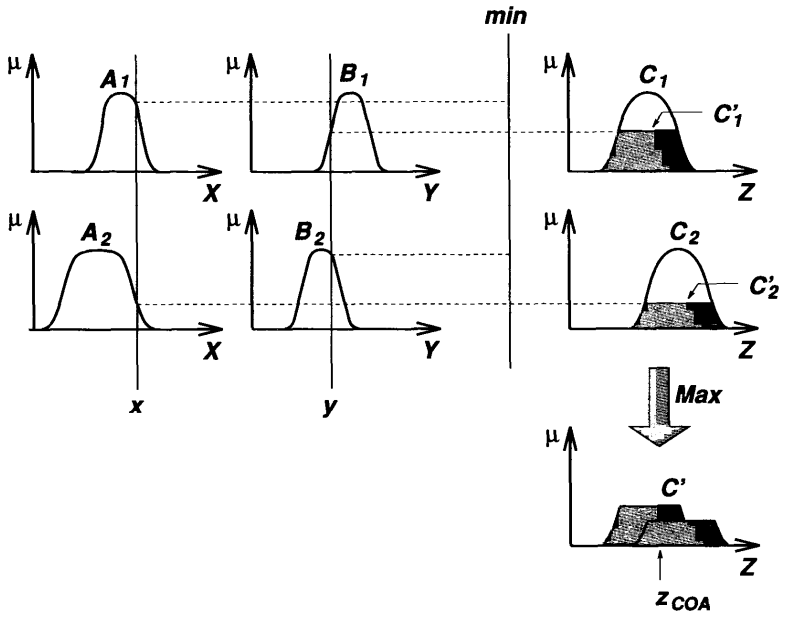


Figura 1.5 Sistema de inferencia Mamdani.

1.2.2.1 Modelos Mamdani.

El sistema de inferencia Mamdani fue propuesto como un primer intento de controlar una máquina de caldera y pistón de combinación mediante reglas de control lingüísticas obtenidas de operadores humanos experimentados. La Figura 1.5 muestra como un SID tipo Mamdani de dos reglas proporciona una salida *z* cuando esta sujeto a dos entradas duras *x* y *y*. En la aplicación de Mamdani, se utilizaron dos SID como controladores para generar el calor de

entrada a la caldera y controlar la abertura de la maquinaria del cilindro respectivamente, para regular la presión de vapor en la caldera y la velocidad de la maquinaria. Ya que la planta solo toma valores duros, es necesario utilizar un desdifusor. Cuando se habla de desdifución se refiere a la forma en que un valor duro es extraído de un conjunto difuso como un valor representativo. En general, hay cinco métodos para defuzificar un conjunto difuso A de un universo de discurso Z. A continuación se dará una breve explicación de cada una de ellas.

- **Centroide de área:**

$$z_{COA} = \frac{\int_Z \mu_A(z) z \, dz}{\int_Z \mu_A(z) \, dz} \tag{1.17}$$

  donde $\mu_A(z)$ es la salida de la función de membresía agregada. Esta es la estrategia de desdifución más adoptada, la cual es tomada del cálculo de los valores normales de las funciones de membresía difusas.

- **Bisector de área:**

$$z_{BOA} = \int_{\alpha}^{z_{BOA}} \mu_A(z) \, dz = \int_{z_{BOA}}^{\beta} \mu_A(z) \, dz \tag{1.18}$$

  donde $\alpha=\min\{z \mid z \in Z\}$ y $\beta=\max\{z \mid z \in Z\}$. Es decir, una línea vertical z_{BOA} particiona la región entre $z=\alpha$, $z=\beta$, y=0 y $y=\mu_A(z)$ en dos regiones con la misma área.

- **Media de los máximos:**

$$z_{MOM} = \frac{\int_{Z'} z \, dz}{\int_{Z'} dz} \tag{1.19}$$

  donde $Z'=\{z \mid \mu_A(z)= \mu^*\}$. En particular, si $\mu_A(z)$ tiene un solo máximo en $z=z^*$, entonces $z_{MOM}=z^*$. Sin embargo, si $\mu_A(z)$ alcanza su máximo en $z \in [z_{izq},z_{der}]$, entonces $z_{MOM}=(z_{izq} + z_{der})/2$.

- **El más pequeño de los máximos:** z_{SOM} es el mínimo (en términos de magnitud) de los máximos de z.

- **El más grande de los máximos:** z_{LOM} es el máximo (en términos de magnitud) de los máximos de z.

1.2.2.2 Modelos Sugeno.

El modelo difuso Sugeno (también conocido como **modelo difuso TSK**) fue propuesto por Takagi, Sugeno y Kang en un esfuerzo por desarrollar una aproximación sistemática para generar reglas difusas a partir de un conjunto de datos de entrada y salida dados. Una regla difusa típica en un modelo Sugeno tiene la forma:

SI x es A **Y** y es B **ENTONCES** $z=f(x,y)$

donde A y B son conjunto difusos en el antecedente, mientras que $z=f(x,y)$ es una función dura en el consecuente. Usualmente $f(x,y)$ es un polinomio de las variable x y y, pero esta puede ser cualquier función siempre y cuando pueda describir apropiadamente la salida del modelo dentro de la región difusa especificada por el antecedente de la regla. Cuando $f(x,y)$ es un polinomio de primer orden el SID es llamado modelo difuso Sugeno de primer orden. Cuando f es una constante, se tiene un modelo difuso Sugeno de orden cero, el cual se puede ver como un caso especial de un SID Mamdani, en el cual cada consecuente de las reglas se especifica por un singleton difuso (o un consecuente pre-defuzificado) o un caso especial de modelo Tsukamoto, en el cual cada consecuente de las reglas se especifica por una función de membresía de una función escalón centrada en la constante. Un ejemplo de un modelo difuso Sugeno de una sola entrada puede ser expresarse como:

SI	x es pequeño	**ENTONCES**	y = x + 6
SI	x es mediano	**ENTONCES**	y = 2x + 1
SI	x es grande	**ENTONCES**	y = x - 6

1.2.2.3 Modelos Tsukamoto.

En los modelos difusos Tsukamoto, el consecuente de cada regla si-entonces difusa es representado por un conjunto difuso con una función de membresía monotónica como se muestra en la Figura 1.6. Como resultado, la salida inferida de cada regla es definida como un valor duro inducido por la activación directa de la regla. La salida total se toma como el promedio ponderado de la salida de cada regla. Ya que cada regla infiere una salida dura, este modelo agrega cada salida de las reglas mediante el método de promedio ponderado y así evita el consumo de tiempo debido al proceso de desdifución. Aun así, los modelos Tsukamoto no son de uso común debido a que no son tan transparentes como los modelos Mamdani o Sugeno.

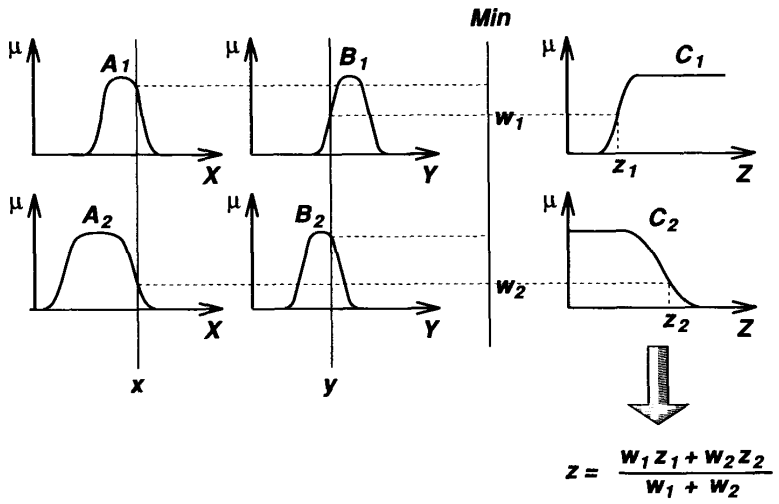


Figura 1.6 Sistema de inferencia Tsukamoto.

1.2.3 Algoritmos Evolutivos.

Los Algoritmos Evolutivos (AE) son métodos estocásticos que se han aplicado satisfactoriamente en muchos problemas de búsqueda, optimización y aprendizaje en maquinas. La estructura general de estos algoritmos es la que se muestra a continuación:

$t = 0$;
inicilización($P(t)$);
evaluación($P(t)$);
while not(*solución*)
    $P'(t)$ = reproducción($P(t)$);
    evaluación($P'(t)$);
    $P(t+1)$ = selección($P(t)$, $P'(t)$);
    $t = t + 1$;
end while;

Estos métodos emulan el proceso evolutivo de los seres vivos. Es decir, sobre la base de una población $P(t)$ de N individuos o posibles soluciones, genera y evoluciona a nuevas poblaciones mediante algún método de reproducción. La solución óptima se encuentra mediante algún mecanismo de evaluación y selección. Los individuos o soluciones generalmente son llamados cromosomas C. En ellos se encuentran codificados todos los parámetros que afectan al comportamiento del problema a optimizar, donde cada parámetro o variable es denominado como gen. Una representación gráfica de estos términos se muestra en la Figura 1.7. La codificación de estos parámetros puede ser utilizando códigos binarios o utilizando valores numéricos reales.

x_1	x_2	X_3	…	x_n	C_i	
0000	0101	1111	…	1101	C_1	binario
0100	1000	0001	…	1010	C_2	
…	…	…	…	…	…	
0010	1101	0110	…	1001	C_N	
0.836	0.123	0.057	…	0.999	C_1	real
0.295	0.106	1.00	…	0.386	C_2	
…	…	…	…	…	…	
0.234	0.111	0.001	…	0.981	C_N	

Figura 1.7 Representación genética del problema.

La evaluación de estas poblaciones normalmente se realiza utilizando una función de error o costo. El resultado de esta evaluación no solo sirve para determinar cual es la mejor solución encontrada, además, también se puede utilizar para implementar algún método de reproducción basado en elitismo. Otro mecanismo es el de reproducción sexual o de cruza, donde se intercambia información entre cromosomas padres y se heredan a los hijos. Por otro lado, al igual que todo proceso evolutivo, también se tiene un mecanismo de mutación. Este mecanismo altera la información de alguno o algunos de los genes de cada cromosoma para crear nuevas poblaciones. Aquí hay que hacer una distinción en la forma que adoptan estos mecanismos de reproducción al aplicarse a cromosomas con codificación binaria o real. Tratándose de códigos binarios, los mecanismos están bien definidos, se intercambian bits o se utilizan operadores lógicos como el de inversión para crear máscaras y alterar los bits de cada cromosoma. Sin embargo, cuando se habla de valores reales, no hay nada establecido, es posible utilizar los mismos operadores utilizados en datos binarios o utilizar incluso operadores matemáticos. Para el mecanismo de cruza se puede pensar en calcular el promedio de dos cromosomas y el resultado puede pasar como hijo a la siguiente generación. Para la mutación también se puede pensar en una simple multiplicación por algún número aleatorio. Por último, la evaluación se realiza con una función de costo o función de aptitud. Dicha función evalúa a cada uno de los cromosomas en cada generación con el fin de determinar que individuo es el más apto para transmitir su información o sus genes a las siguientes

poblaciones. Esta función tampoco esta predeterminada, el diseñador puede optar por cualquier función que necesite o se adapte a sus necesidades. Sin embargo, siempre se recomienda que la función esté formulada de tal modo que las malas soluciones sean penalizadas severamente arrojando valores muy bajos de aptitud, mientras que a las soluciones muy buenas se les otorgue un valor de aptitud muy cercana o igual al máximo valor de aptitud que se pueda alcanzar. Los AE pueden clasificarse también de acuerdo al tipo de operadores de reproducción y el número de cromosomas por población que utilicen. Existen tres principales categorías: Estrategias Evolutivas (EE), Algoritmos Genéticos (AG) y Programación Evolutiva (PE) [16-19].

1.2.3.1 Estrategias Evolutivas.

Las estrategias evolutivas fueron desarrolladas en 1964 en Alemania para resolver problemas hidrodinámicos de alto grado de complejidad por un grupo de estudiantes de ingeniería encabezado por Ingo Rechenberg [16]. La versión original (1+1)-EE usaba un solo padre y con él se generaba un solo hijo. Este hijo se mantenía si era mejor que el padre, o de lo contrario se eliminaba (a este tipo de selección se le llama *extintiva*, porque los peores individuos obtienen una probabilidad de ser seleccionado de cero). En la (1+1)-EE, un individuo nuevo es generado usando:

$$x(t+1) = x(t) + N(0,\sigma) \tag{1.20}$$

donde t se refiere a la generación (o iteración) en la que se encuentra, y $N(0,\sigma)$ es un vector de números Gaussianos independientes con una media de cero y desviaciones estándar σ. Rechenberg [20] introdujo el concepto de población, al proponer una estrategia evolutiva llamada (μ+1)-EE, en la cual hay μ padres y se genera un solo hijo, el cual puede reemplazar al peor padre de la población (selección extintiva). Schwefel [21] introdujo el uso de múltiples hijos en las denominadas (μ+λ)-EEs y (μ,λ)-EEs. La notación se refiere al mecanismo de selección utilizado:

- En el primer caso, los μ mejores individuos obtenidos de la unión de padres e hijos sobreviven.

- En el segundo caso, sólo los μ mejores hijos de la siguiente generación sobreviven.

Rechenberg formuló una regla para ajustar la desviación estándar de forma determinística durante el proceso evolutivo de tal manera que el procedimiento converja hacia el óptimo. Esta regla se conoce como la "regla del éxito 1/5", y en palabras dice: La razón entre mutaciones exitosas y el total de mutaciones debe ser 1/5. Si es mayor, entonces debe incrementarse la desviación estándar. Si es menor, entonces debe decrementarse, formalmente:

$$\sigma(t) = \begin{cases} \dfrac{\sigma(t-n)}{c} & p_s > \dfrac{1}{5} \\ \sigma(t-n)\cdot c & p_s < \dfrac{1}{5} \\ \sigma(t-n) & p_s = \dfrac{1}{5} \end{cases} \tag{1.21}$$

donde n es el número de dimensiones, t es la generación, p_s es la frecuencia relativa de mutaciones exitosas medida sobre intervalos de (por ejemplo) 10n individuos, y c=0.817 (este valor fue derivado teóricamente por Schwefel [22]). $\sigma(t)$ se ajusta cada n mutaciones. En las estrategias evolutivas se evoluciona no sólo a las variables del problema, sino también a los parámetros mismos de la técnica (es decir, las desviaciones estándar). A esto se le llama *auto-adaptación*. Los padres se mutan usando las siguientes ecuaciones:

$$\begin{aligned} \sigma'(i) &= \sigma(i)\cdot e^{(\tau' N(0,1)+\tau N_i(0,1))} \\ x'(i) &= x(i) + N(0,\sigma'(i)) \end{aligned} \tag{1.22}$$

donde τ y τ' son constantes de proporcionalidad que están en función de n. Los operadores de recombinación de las estrategias evolutivas pueden ser:

- **Sexuales**: el operador actúa sobre 2 individuos elegidos aleatoriamente de la población de padres.
- **Panmíticos**: se elige un solo padre al azar, y se mantiene fijo mientras se elige al azar un segundo padre (de entre toda la población) para cada componente de sus vectores.

Las estrategias evolutivas simulan el proceso evolutivo al nivel de los individuos, por lo que la recombinación es posible. Asimismo, usan normalmente selección determinística.

1.2.3.2 Algoritmos Genéticos.

Los algoritmos genéticos (denominados originalmente "planes reproductivos genéticos") fueron desarrollados por John H. Holland a principios de los 1960s [23], motivado por resolver problemas de aprendizaje de máquina. El algoritmo genético enfatiza la importancia de la cruza sexual (operador principal) sobre el de la mutación (operador secundario), y usa selección probabilística. El algoritmo básico es el siguiente:

- Generar (aleatoriamente) una población inicial.
- Calcular aptitud de cada individuo.
- Seleccionar (probabilísticamente) en base a aptitud.
- Aplicar operadores genéticos (cruza y mutación) para generar la siguiente población.
- Ciclar hasta que cierta condición se satisfaga.

La representación tradicional es la binaria, tal y como se ejemplifica en la Figura 1.8. A la cadena binaria se le llama *cromosoma*. Una o más posiciones de la cadena son denominadas como *gen* y al valor dentro de esta posición se le llama *alelo*. Cada uno de los genes es la codificación de un parámetro del sistema modelado, que a su vez es el *genotipo* que corresponde a un a solución del problema (*fenotipo*). Para poder aplicar el algoritmo genético se requiere de los 5 componentes básicos siguientes:

- Una representación de las soluciones potenciales del problema.
- Una forma de crear una población inicial de posibles soluciones (normalmente un proceso aleatorio).
- Una función de evaluación que juegue el papel del ambiente, clasificando las soluciones en términos de su *aptitud*.
- Operadores genéticos que alteren la composición de los hijos que se producirán para las siguientes generaciones.

- Valores para los diferentes parámetros que utiliza el algoritmo genético (tamaño de la población, probabilidad de cruza, probabilidad de mutación, número máximo de generaciones, etc.)

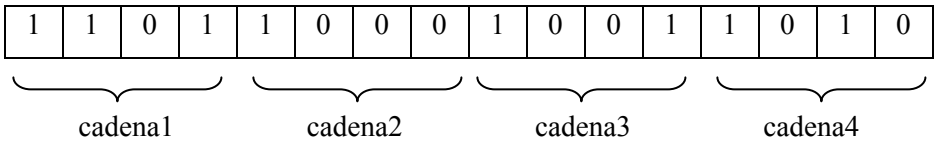


Figura 1.8 Codificación en los AG.

El AG usa selección probabilística al igual que la Programación Evolutiva, y en contraposición a la selección determinística de las Estrategias Evolutivas. El AG usa representación binaria para codificar las soluciones a un problema, por lo cual se evoluciona el genotipo y no el fenotipo como en la Programación Evolutiva o las Estrategias Evolutivas. El operador principal en el AG es la cruza, y la mutación es un operador secundario. En la Programación Evolutiva, no hay cruza y en las Estrategias Evolutivas es un operador secundario. Ha sido demostrado [24] que el AG requiere de elitismo (o sea, retener intacto al mejor individuo de cada generación) para poder converger al óptimo. Los AG no son, normalmente, auto-adaptativos, aunque el uso de dicho mecanismo es posible, y ha sido explorado extensivamente en la literatura especializada [7, 25, 26].

1.2.3.3 Programación Evolutiva.

Lawrence J. Fogel propuso en los 60s una técnica denominada programación evolutiva, en la cual la inteligencia se ve como un comportamiento adaptativo [27]. La programación evolutiva enfatiza los nexos de comportamiento entre padres e hijos, en vez de buscar emular operadores genéticos específicos (como en el caso de los algoritmos genéticos). El algoritmo básico de la programación evolutiva es el siguiente:

- Se genera aleatoriamente una población inicial.
- Se aplica mutación.
- Se calcula la aptitud de cada hijo y se usa un proceso de selección mediante torneo (normalmente estocástico) para determinar cuáles serán las soluciones que se retendrán.

La programación evolutiva es una abstracción de la evolución al nivel de las especies, por lo que no se requiere el uso de un operador de recombinación (diferentes especies no se pueden cruzar entre sí). Asimismo, usa selección probabilística. En la Tabla 1.3 se muestra un resumen con las principales características de los tres principales paradigmas de la Computación Evolutiva:

Tabla 1.3 Algoritmos Evolutivos.

	Estrategias Evolutivas	**Programación Evolutiva**	**Algoritmos Genéticos**
Representación	Real	Real	Binaria
Función de Aptitud	Valor de la función objetivo	Valor de la función objetivo ajustada	Valor de la función objetivo ajustada
Auto-Adaptación	Desviaciones estándar y ángulos de rotación	-Ninguna -Varianzas -Coef. de correlación	Ninguna
Mutación	Gaussiana, como operador principal	Gaussiana, como operador único	Inversión de bits, operador secundario
Recombinación	-Discreta e intermedia -Sexual y panmítica, importante para la auto-adaptación	Ninguna	-Cruza de 2 puntos -Cruza uniforme, únicamente sexual, operador principal
Selección	Determinística, extintiva o basada en la preservación	Probabilística, extintiva	Probabilística, basada en la preservación
Restricciones	Restricciones arbitrarias de desigualdad	Ninguna	Límites simples mediante el mecanismo de codificación
Teoría	-Velocidad de convergencia para casos especiales, (1+1)EE, (1+λ)EE -Convergencia global para (μ+λ)EE	-Velocidad de convergencia para casos especiales, (1+1)PE -Convergencia global para meta-PE	-Teoría de los Esquemas -Convergencia global para la versión elitista

1.2.4 Caos.

Debido a la gran cantidad de fenómenos que generan sistemas dinámicos no lineales es difícil, si no imposible, cubrir todos los conceptos necesarios acerca de estos en un solo capítulo. En este apartado, tomando como referencia el mapa logístico más simple, se muestran algunos pero importantes conceptos básicos y algunos esenciales relacionados con los sistemas dinámicos no lineales.

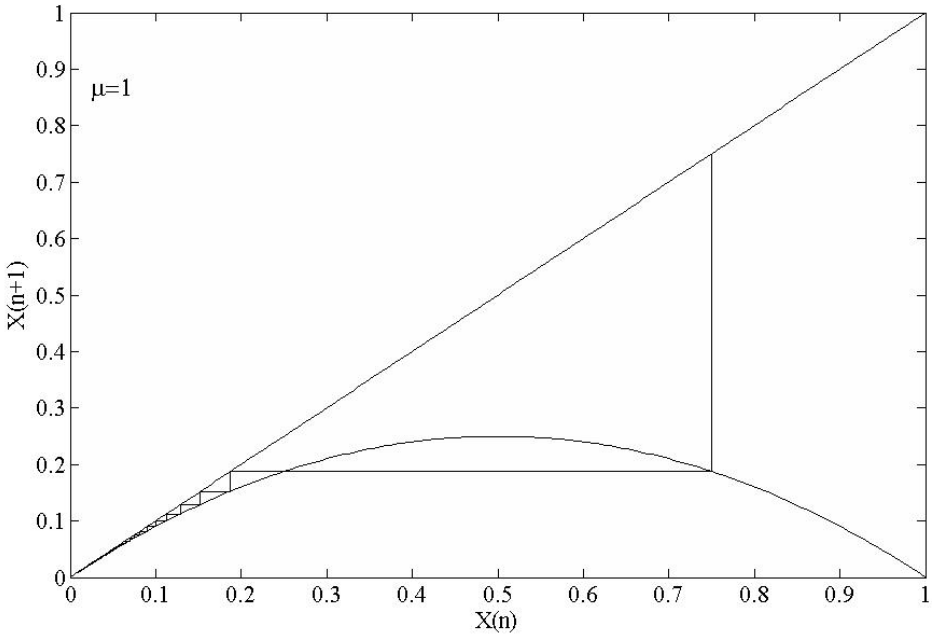
1.2.4.1 Mapa Logístico.

Analizando un *Mapa Logístico* se pueden introducir algunos conceptos básicos tales como bifurcaciones, orbitas periódicas estables y no estables, ventanas periódicas, comportamientos ergódicos y de mezclado, conexiones homoclínicas, orbitas caóticas y cierto tipo de universalidad. Un mapa logístico se denota como una ecuación de diferencias:

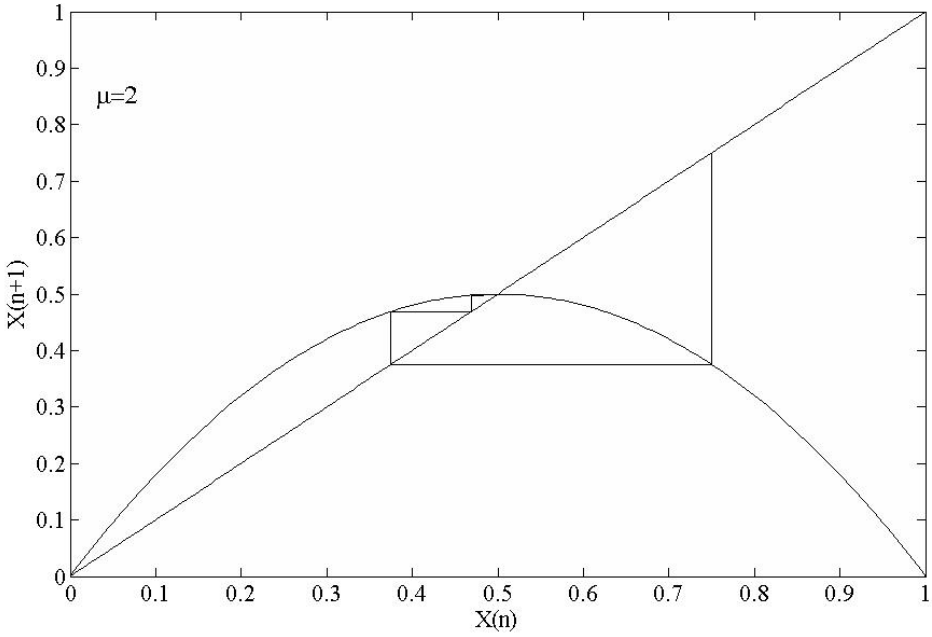
$$x_{n+1} = f(\mu, x_n) = \mu x_n (1 - x_n) \tag{1.23}$$

La ecuación 1.23 fue propuesta por Robert May y Mitchell Feigenbaum en 1976 como un modelo que describe la población biológica [28]. Este modelo representa a la población como una fracción, x_n, de una población máxima que puede ser soportada por su hábitat. μ representa la razón de crecimiento de población de las especies. Por lo tanto, $x_n \in [0,1]$ y $0 \leq \mu \leq 4$. Si $\mu > 4$, la ecuación 1.23 no será valida en el intervalo [0,1].

Resolviendo la ecuación $x=f(\mu,x)$ para encontrar los puntos fijos se pueden derivar los puntos fijos de nuestro mapa. Cuando $0 < \mu < 1$, $x_1 = 0$ es el único punto fijo ($x_2 = (\mu - 1)/\mu$ esta fuera de [0,1]). Ya que $|f'(\mu,x_1)| = |f'(\mu,0)| = \mu < 1$, el punto fijo $x_1 = 0$ es estable y es un punto atractor, por lo que todos los puntos convergen a cero a través de las iteraciones de f sin importar donde inicien los puntos, lo que se muestra en la Figura 1.9a. Cuando $\mu \geq 1$, existen dos puntos fijos $x_1 = 0$ y $x_2 = (\mu - 1)/\mu$. Debido a que $|f'(\mu,0)| = \mu \geq 1$, el punto fijo x_1 se vuelve inestable así como se crea el punto fijo x_2. El segundo punto fijo x_2 es estable para $1 < \mu < 3$, ya que $|f'(\mu,x_2)| = |f'(\mu,(\mu - 1)/\mu)| = |1 - \mu| < 1$, entonces todos los puntos convergen a x_2, como se muestra en la Figura 1.9b. Posteriormente, ¿qué sucede con los dos puntos fijos?, ¿se vuelven inestables para $\mu > 3$?.



(a)



(b)

Figura 1.9 Mapa Logístico para a) $0 < \mu \leq 1$ y b) $1 < \mu \leq 3$.

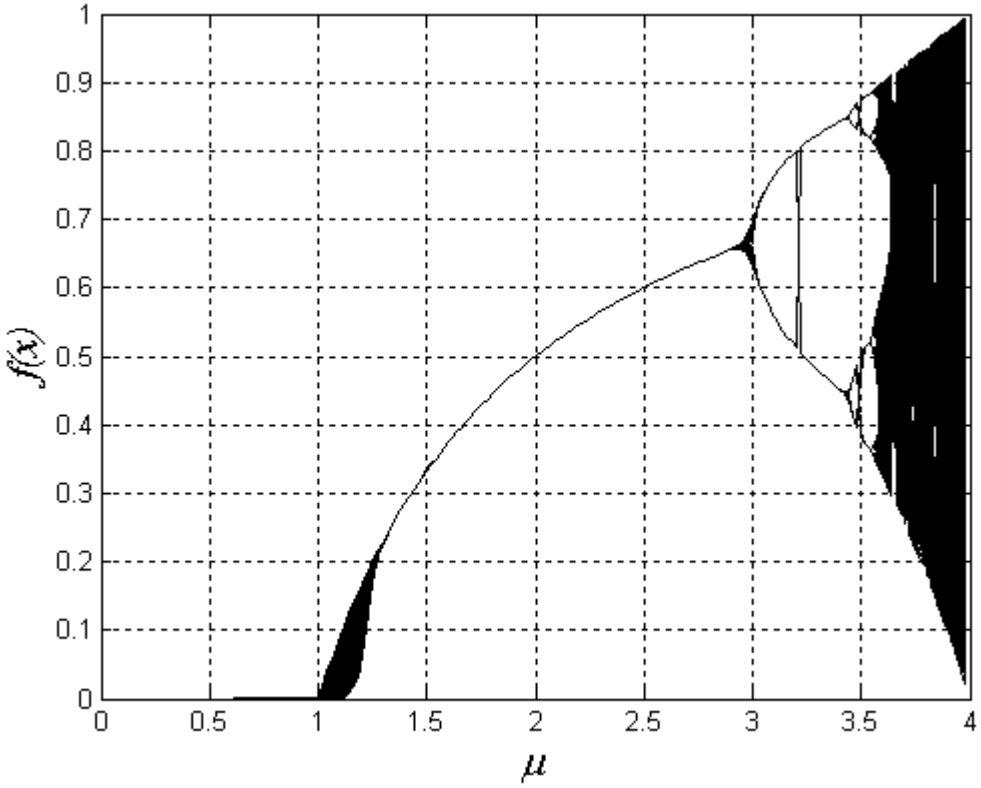


Figura 1.10 Bifurcaciones de $f(x)=\mu x(1-x)$.

Es claro que la estabilidad de los puntos fijos cambia conforme varía el parámetro μ. Para observar esto, se grafica el conjunto de atractores de f como una función de μ para $0<\mu<4$. En la Figura 1.10 se muestra la grafica resultante de la simulación numérica del modelo. Claramente se observa que hasta antes de $\mu \approx 3$, el punto fijo x_2 aun es un atractor o punto estable. Sin embargo, conforme se incrementa el valor de μ se pueden observar algunas características de la función derivadas de la solución asintótica del mapa logístico. El fenómeno observado es mejor conocido como *bifurcación*. Se puede ver que el conjunto de atractores inicialmente consiste de un solo punto que tiene una bifurcación en dos puntos alrededor de $\mu \approx 3$. Subsecuentemente, estos dos puntos vuelven a bifurcar en 4 puntos, los cuales bifurcan en ocho y así sucesivamente. Los intervalos de cambio en μ entre bifurcaciones va decreciendo eventualmente hasta que aparece un conjunto de atractores caóticos. La región caótica aparece intercalada en bandas llamadas *ventanas periódicas*, las cuales consisten solamente de un número pequeño de puntos.

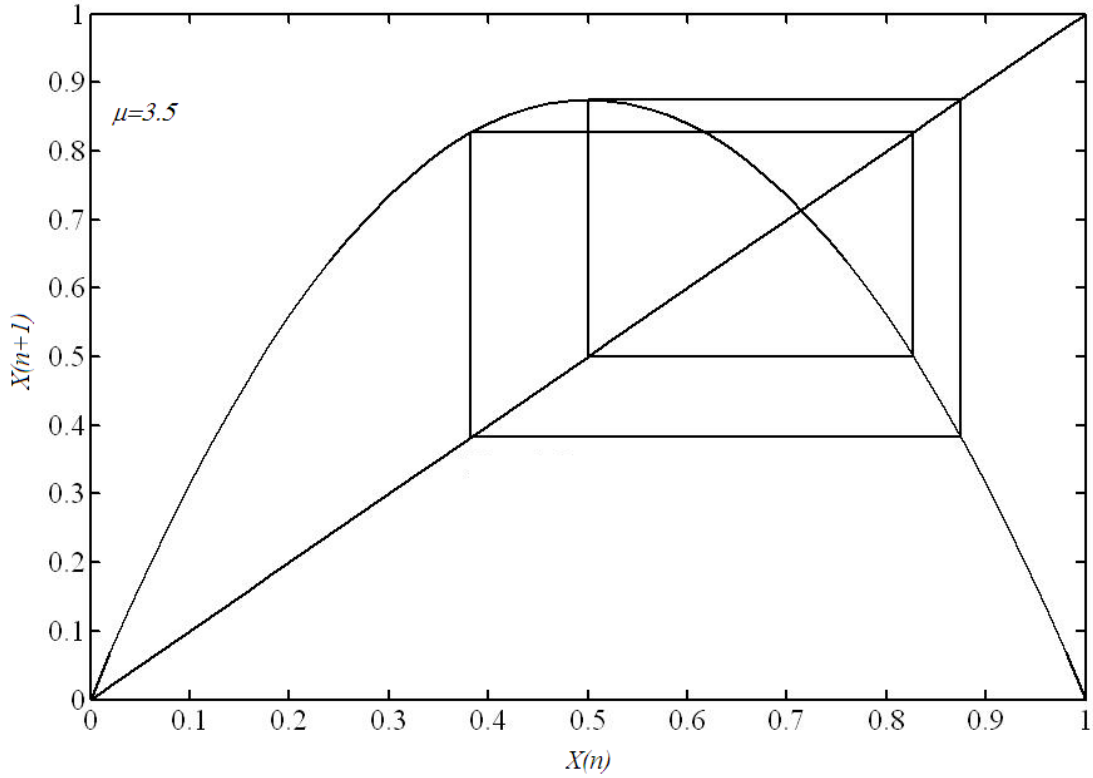


Figura 1.11 Mapa Logístico para $\mu = 3.5$.

Ahora se verá a detalle el mecanismo detrás de este fenómeno. Como se pudo observar anteriormente, la primera bifurcación se presenta cuando $\mu = 1$, en la cual el punto fijo $x_1 = 0$ se vuelve inestable y aparece el punto fijo $x_2 = (\mu - 1)/\mu$ volviéndose estable. La segunda bifurcación toma lugar en $\mu \approx 3$, donde el punto fijo x_2 se vuelve inestable y se crea un ciclo de dos puntos (*2-cycle*) de atracción como se muestra en la Figura 1.11. Similar a la derivación de los puntos fijos del mapa logístico, los punto del ciclo de dos puntos x^*_1 y x^*_2 pueden ser obtenidos resolviendo las ecuaciones $f(\mu,f(\mu,x))=x$. Esto nos da una ecuación de orden cuatro:

$$-\mu^3 x^4 + 2\mu^3 x^3 - \left(\mu^2 + \mu^3\right)x^2 + \left(\mu^2 - 1\right)x = 0 \qquad (1.24)$$

Teniendo conocimiento de los dos puntos fijos en la iteración, la ecuación 1.24 puede reducirse a una ecuación de tercer orden con $x_1 = 0$:

$$-\mu^3 x^3 + 2\mu^3 x^2 - \left(\mu^2 + \mu^3\right)x + \left(\mu^2 - 1\right) = 0 \tag{1.25}$$

dividiendo esta expresión entre ($x - (\mu - 1)/\mu$) se obtiene:

$$-\mu^3 x^2 + \left(\mu^2 + \mu^3\right)x - \left(\mu^2 + \mu\right) = 0 \tag{1.26}$$

y obteniendo las raíces de la ecuación anterior se tiene:

$$x_{1,2}^{*} = \frac{1 + \mu \pm \sqrt{\mu^2 - 2\mu - 3}}{2\mu} \tag{1.27}$$

los cuales forman un ciclo de 2 puntos para el mapa logístico f cada uno con un punto fijo de $f^2 = f+f$. Asumiendo que los puntos $x_1, x_2, \ldots, x_p$ denotan los puntos de un ciclo de p puntos (*p-cycle*) del mapa logístico, donde $f(x_i) = x_{i+1}$ y $f(x_p) = x_1$. Cada uno de los puntos x_i es un punto fijo de f^p. Por medio de la diferenciación utilizando la regla de la cadena se tiene:

$$\left.\frac{df^p}{dx}\right|_{x_i} = f'(x_1) f'(x_2) \cdots f'(x_p) = \prod_{j=1}^{p} f'(x_j), \; para\ toda\ i = 1,2,\ldots,p \tag{1.28}$$

y por lo tanto:

$$\left|\left.\frac{df^p}{dx}\right|_{x_i}\right| = \prod_{j=1}^{p} \left| f'(x_j) \right|, \; para\ toda\ i = 1,2,\ldots,p \tag{1.29}$$

aplicando el criterio de estabilidad para el caso de $f^2(\mu,x)$ se tiene:

$$-1 < \mu^2 (1 - 2x_1)(1 - 2x_2) < 1 \tag{1.30}$$

Sustituyendo los puntos $x^*_{1,2}$ en las ecuaciones 1.27 y 1.30 se obtiene que $3 < \mu < 1+\sqrt{6} \approx 3.44949$. En $\mu = 1+\sqrt{6}$ el ciclo de dos puntos pierde su estabilidad y se bifurca en un ciclo de 4 puntos y así sucesivamente. La secuencia de las bifurcaciones se vuelve infinita, y resulta bastante complejo calcularlas todas analíticamente. Tomando a μ_n como el valor del parámetro en el cual aparece un periodo de un ciclo de 2 puntos y denotando el valor de convergencia como μ_∞. Asumiendo convergencia geométrica, la diferencia entre los valores μ_∞ y μ_n es:

$$\mu_\infty - \mu_n = \frac{c}{\delta^n} \tag{1.31}$$

donde c y $\delta < 1$ son constantes. Aplicando una transformación algebraica se tiene:

$$\delta = \frac{\mu_n - \mu_{n-1}}{\mu_{n+1} - \mu_n} \tag{1.32}$$

Con esto se puede calcular el número de bifurcaciones en μ_1=3.0, μ_2=3.449490…, μ_3=3.54409…, μ_4=3.564407…, μ_5=3.568759…, μ_6=3.569692…, μ_7=3.569891… y μ_8=3.569934…, que se puede estimar que convergen a μ_∞=3.5699456… mediante la ecuación 1.31. Utilizando estos valores de μ, se puede obtener una aproximación del valor de δ = 4.669201609102990…, valor llamado como constante de *Feigenbaum* después de su descubrimiento. De la ecuación 1.31 también se puede encontrar que c = 2.637…. Se debe hacer notar que la constante de Feigenbaum es universal, por ejemplo, esta aparece en cualquier sistema dinámico que tiene un comportamiento caótico mediante *bifurcaciones de doblamiento de periodos* y tienen un solo máximo cuadrático.

Se puede ver claramente en la Figura 1.10 que el intervalo entre μ_∞ y 4 contiene un gran número de ventanas pequeñas donde el conjunto de atractores es un ciclo periódico estable. La ventana más grande corresponde al periodo estable de 3 ciclos (*3-cycle*). Este aparece alrededor de μ = 3.828427… y es estable en un intervalo de μ relativamente grande. Fuera de esta ventana el mapa parece caótico [29].

1.2.4.2 Bifurcaciones.

Como se discutió anteriormente, una bifurcación es un cambio cualitativo en la dinámica de un sistema que ocurre cuando un parámetro de control es variado. Típicamente una bifurcación ocurre cuando se vuelve inestable. Las bifurcaciones son clasificadas de acuerdo a como pierden estabilidad. En esta sección se muestran tres tipos de bifurcaciones llamadas *saddle node*, *pitchfork* y *transcrítica* [30].

Considerando mapas en la forma $f : \mathtt{R} \times \mathtt{R} \to \mathtt{R}$, donde $(\mu,x) \mapsto f(\mu,x) \in \mathtt{R}$. La estabilidad de los puntos fijos es determinada por el valor de $\partial f/\partial x$. El valor de μ para el cual el punto fijo se vuelve inestable creando una bifurcación es llamado el valor de bifurcación de μ. Algunos ejemplos para distinguir estos tipos de bifurcaciones se presentan a continuación. El sistema unidimensional:

$$\dot{x} = f(\mu, x) = \mu x - x^2 \tag{1.33}$$

tiene dos puntos fijos $x_1 = 0$ y $x_2 = \mu$. Si se varía μ entonces existirán dos curvas de puntos fijos como se muestra en la Figura 1.12, donde las curvas sólidas se refieren a un equilibrio estable y las curvas punteadas se refieren a unas inestables. Ya que el Jacobiano (***J***) de un sistema unidimensional es simplemente $J = \partial f/\partial x|_{x_1} = \mu$, se verá que para $\mu < \mu_0 = 0$, el punto fijo $x_1 = 0$ es estable, pero para $\mu > \mu_0 = 0$ este se vuelve inestable. Por lo tanto a $(x_1,\mu_0) = (0,0)$ es llamado punto de bifurcación. De forma similar (x_2,μ_0) también es un punto de bifurcación. Debido a la naturaleza del punto de bifurcación (0,0) en el plano x – μ es llamada bifurcación transcrítica. Para el sistema unidimensional:

$$\dot{x} = f(\mu, x) = \mu - x^2 \tag{1.34}$$

se tiene un punto fijo $x_1 = 0$ en $\mu_0 = 0$ y puntos fijos $x_2 = \pm\sqrt{\mu}$ para $\mu > 0$, donde $x_2 = \sqrt{\mu}$ es estable y $x_2^* = -\sqrt{\mu}$ es inestable para $\mu > 0$. Por lo tanto, debido a la naturaleza de la bifurcación $(x,\mu) = (0,0)$ en el plano x – μ es llamada bifurcación Saddle-Node, lo que se muestra en la Figura 1.13.

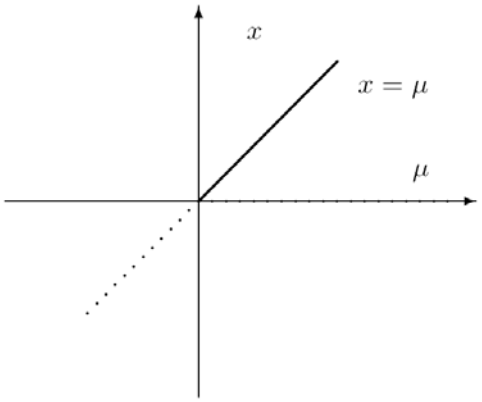


Figura 1.12 Bifurcación Transcrítica.

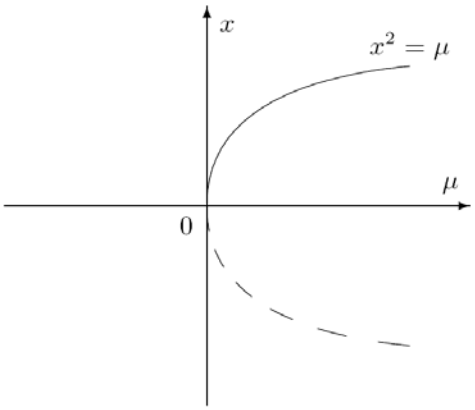


Figura 1.13 Bifurcación Saddle-Node.

En el sistema unidimensional:

$$\dot{x} = f(\mu, x) = \mu x - x^3 \tag{1.35}$$

se tiene un punto fijo $x_1 = 0$ en $\mu_0 = 0$ y puntos fijos en $x_2 = \pm\sqrt{\mu}$ para $\mu > 0$, ya que $x_0 = 0$ es inestable para $\mu > \mu_0 = 0$ y estable para $\mu < \mu_0 = 0$, y $x_2 = \pm\sqrt{\mu}$ es estable para $\mu > 0$ debido a que el Jacobino $J = -2\mu$. Este tipo de bifurcación es llamada Pitchfork y se muestra en la Figura 1.14.

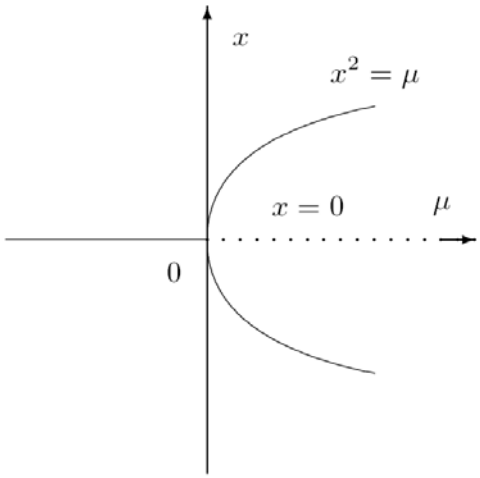


Figura 1.14 Bifurcación Pitchfork.

1.2.4.3 Exponentes de Lyapunov.

Una característica distintiva del caos es la extrema sensibilidad a las condiciones iniciales (o parámetros) del sistema dinámico, en el sentido de que dos trayectorias que inician una cercana de la otra en el espacio de fase se alejarán exponencialmente una de la otra. Otras características del caos incluyen la existencia de un conjunto denso de órbitas periódicas inestables en este régimen [31], exponentes de Lyapunov positivos o una entropía finita de Kolmogorov-Sinai [32], espectros de potencia continuos [32], ergodicidad [32], así como otras propiedades limitantes.

De entre todas las características del caos, los exponentes de Lyapunov positivos son la forma más simple de verificar y poder medir el comportamiento de un sistema caótico. Una prueba cuantitativa del comportamiento caótico puede a veces distinguirlo del comportamiento ruidoso debido al azar o influencias externas. Con una medida cuantitativa del grado de caoticídad se podrá observar como cambia el caos de acuerdo a la variación de los parámetros. Comenzando con dos valores iniciales muy cercanos x_0 y y_0 se tiene:

$$x_k = f\left(\mu, x_{k-1}\right) = \cdots = f^{\,k}\left(\mu, x_0\right) \quad y \quad y_k = f^{\,k}\left(\mu, y_0\right) \tag{1.36}$$

si x_k y y_k son separadas exponencialmente rápidamente durante las iteraciones, entonces:

$$\left|y_k - x_k\right| = \left|y_0 - x_0\right| e^{k\lambda} \quad \left(\lambda > 0\right) \tag{1.37}$$

y

$$\frac{1}{k}\ln\left|y_k - x_k\right| \to \lambda \text{ así como } k \to \infty \tag{1.38}$$

En el caso de que el movimiento sea hacia una región limitada, el fenómeno de separación exponencial no podrá ocurrir a menos que los valores iniciales estén lo suficientemente cerca uno del otro. Por lo tanto, se deja que $|y_0 - x_0| \to 0$ antes de tomar el límite $k \to \infty$ con el fin de definir la constante:

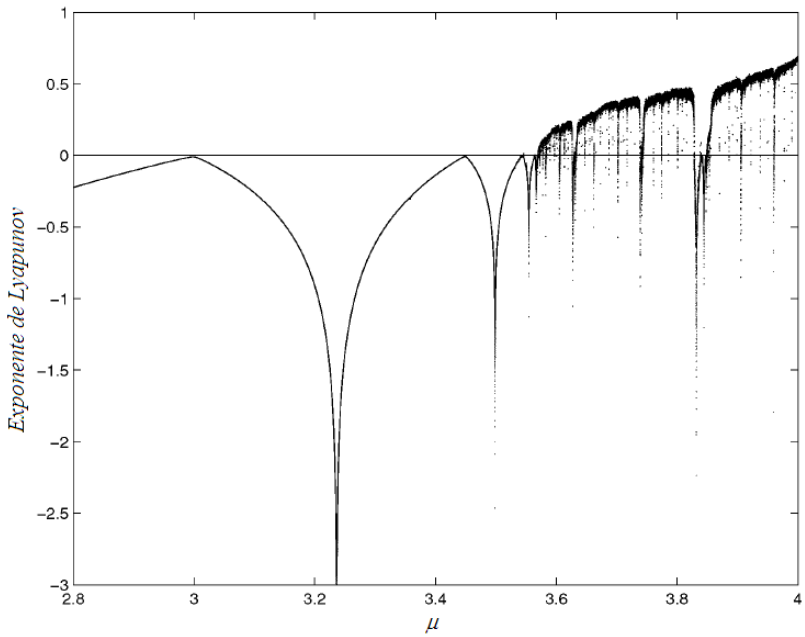


Figura 1.15 Exponente de Lyapunov para el mapa logístico en función de μ.

$$\begin{aligned}\lambda &= \lim_{k\to\infty}\frac{1}{k}\lim_{|y_0-x_0|\to 0}\ln\left|\frac{y_k-x_k}{y_0-x_0}\right| \\ &= \lim_{k\to\infty}\frac{1}{k}\lim_{|y_0-x_0|\to 0}\ln\left|\frac{f^k(\mu,y_0)-f^k(\mu,x_0)}{y_0-x_0}\right| \\ &= \lim_{k\to\infty}\frac{1}{k}\ln\left|\frac{df^k(\mu,x_0)}{dx_0}\right| \\ &= \lim_{k\to\infty}\frac{1}{k}\sum_{i=0}^{k-1}\ln\left|\frac{df(\mu,x_i)}{dx_i}\right|\end{aligned} \tag{1.39}$$

que es llamada el exponente de Lyapunov de la trayectoria $x_k = f^k(x_0)$ para k = 0, 1, …. En el caso del mapa logístico se tiene:

$$\begin{aligned}\lambda(x_0) &= \lim_{k\to\infty}\frac{1}{k}\sum_{i=0}^{k-1}\ln|\mu-2\mu x_i| \\ &= \ln\mu+\lim_{k\to\infty}\frac{1}{k}\sum_{i=0}^{k-1}\ln|1-2x_i|\end{aligned} \tag{1.40}$$

Debe notarse especialmente que $x = 0$ es un punto fijo y por lo tanto se tiene que $\lambda(0) = \ln(\mu)$ (y también $\lambda(1) = \ln(\mu)$). En la Figura 1.15 se muestra el exponente de Lyapunov para el mapa logístico como función de μ. El exponente de Lyapunov es independiente del valor inicial x_0. Se puede observar que para valores de μ para los cuales λ toma valores negativos en la Figura 1.15 corresponde a las regiones de comportamiento periódico evidente en la Figura 1.10.

Las bifurcaciones vistas hasta ahora son conocidas como *bifurcaciones de doblamiento de periodo*, las cuales se acentúan conforme aumenta μ. La solución del doblamiento de periodo aparece en el diagrama de bifurcación no como una curva, sino como una región – las *ventanas* (*periódicas*). La secuencia de valores del parámetro, $\{\mu_k\}$, converge en el número μ_∞. En este punto la curva se vuelve aperiódica. Más allá de este punto empiezan a aparecer ciclos impares de límites periódicos así como orbitas caóticas. En $\mu = 4$ el movimiento es formalmente ergódicos en el intervalo unitario (0,1). Sin embargo, la bifurcación de doblamiento de periodo no es la única forma en que los sistemas pueden llegar a ser caóticos. Otro fenómeno que ocurre frecuentemente es el llamado *intermitencia*. La intermitencia establece que el sistema se comporta de tal forma que las oscilaciones periódicas y regulares en relación a la variable observada son interrumpidas abruptamente por "ráfagas" después de que el parámetro es variado ligeramente. Conforme más se varíe el parámetro las ráfagas ocurren con más frecuencia y la duración de las oscilaciones regulares decrece.

El mapa logístico también muestra una transición intermitente hacia el caos. En la Figura 1.16 se muestra la secuencia de iteraciones para una μ en la vecindad de $\mu_c = 1 + 2\sqrt{2} = 3.828\ldots$, un valor para el inicio de una larga banda de periodos de tres ciclos. Conforme μ se desvía de μ_c, las iteraciones del mapa logístico hacen la transición de un periodo estable de tres ciclos al caos. Para un valor de μ cerca de μ_c, donde el mapa logístico muestra intermitencia como se muestra en la parte baja de la Figura 1.16, se grafica cada tercer punto en la iteración del mapa logístico en la Figura 1.17. Los tramos, de cien iteraciones de duración, donde cada tercer punto regresa al mismo valor, muestra las fases laminares del mapa. Las ráfagas caóticas son las regiones que conectan estas fases laminares.

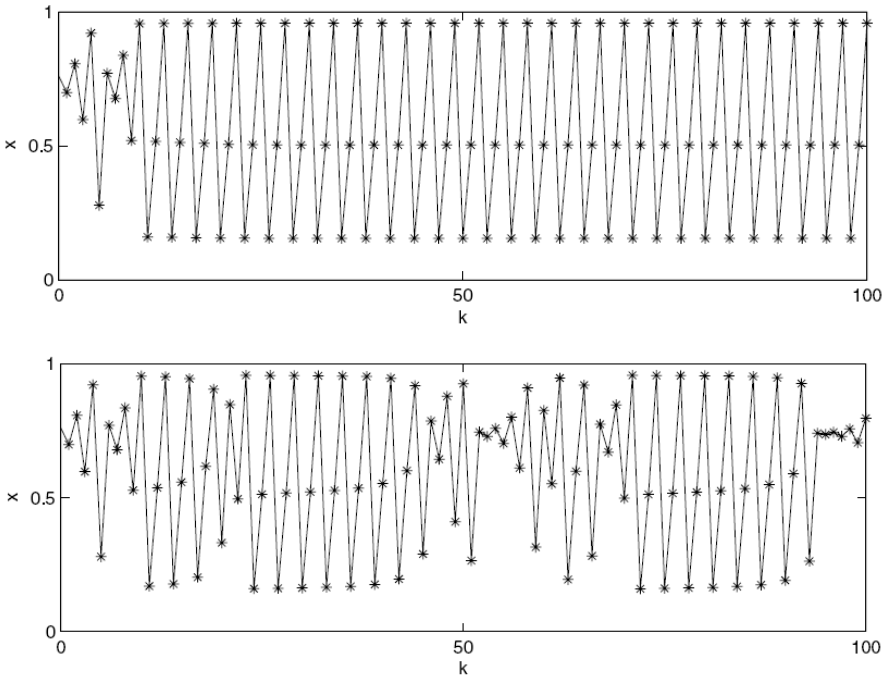


Figura 1.16 Transición a la intermitencia en el mapa logístico para $\mu = 3.8304$ (arriba) y $\mu = 3.8264$ (abajo).

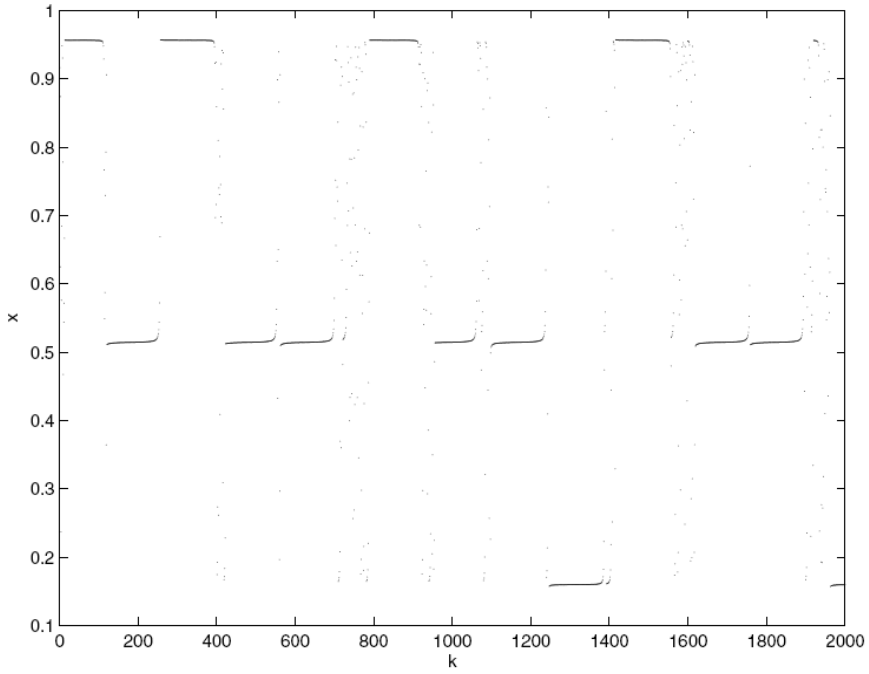


Figura 1.17 Fases laminares en el mapa logístico donde es graficada cada tercer iteración para $\mu = 3.828422$.

1.3 Sistemas Híbridos.

Poco después del nacimiento de la inteligencia computacional a principios de los 90's, los investigadores tomaron gran interés en estudiar el comportamiento sinergístico de las herramientas de cómputo emergentes. Inspirado por el sinergismo de las herramientas de cómputo en 1992 el Prof. Zadeh acuñó el nombre de Computación Suave para esta disciplina emergente. En las palabras del mismo Zadeh:

> *Computación Suave es un nuevo enfoque de la computación el cual asemeja la notable capacidad de la mente humana de razonar y aprender en un ambiente de incertidumbre e imprecisión* [33].

Las características de las herramientas de la SC revelan que cada herramienta tiene sus ventajas y desventajas. La lógica difusa, por ejemplo, es buena con el razonamiento aproximado, pero no tiene mucho alcance en aplicaciones de aprendizaje-máquina u optimización. Por otro lado, las redes neuronales tienen un importante rol en el aprendizaje-máquina. Los algoritmos genéticos pueden ser utilizados en búsquedas inteligentes, optimización y algunas aplicaciones de aprendizaje máquina. Una síntesis cooperativa de estas herramientas debe dar lugar a mejores modelos computacionales que pueden complementar las limitaciones de una herramienta mediante el uso cuidadoso de otras. Dentro de los principales paradigmas sinérgicos se pueden mencionar los siguientes:

a) **Neural-Difuso:** En un híbrido de este tipo es necesario hacer un uso conjunto de las redes neuronales y la lógica difusa en una plataforma común. Una mezcla de estas técnicas le brinda al usuario un sistema con las siguientes capacidades: aprendizaje-máquina de datos aproximados y/o razonamiento aproximado utilizando el conocimiento adquirido a través del aprendizaje-máquina. Este sinergismo puede también ser incorporado dentro de un sistema de adquisición de conocimiento en línea para la adquisición automática del conocimiento y/o sus parámetros, tales como factores de certeza, distribución de membresías o probabilidades condicionales, mientras se esta razonando.

b) **Neural-Genético:** Las redes neuronales y los algoritmos genéticos pueden dar lugar a muchos híbridos. Muchos de estos se centran en la adaptación de los pesos de la red neuronal utilizando los algoritmos genéticos. Un algoritmo genético puede utilizarse para determinar los pesos de una red de trayectoria directa. La suma del error cuadrático medio de las neuronas de la capa de salida, bajo estas circunstancias puede ser utilizada como la función de aptitud. El entrenamiento genético aplicado a la red neuronal ocasionalmente superará al típico algoritmo de retro-propagación y su posibilidad de quedar atrapado en un mínimo local.

c) **Difuso-Genético:** Formas comunes de este tipo de híbrido incluyen la optimización de los parámetros de un sistema difuso utilizando un algoritmo genético. Los sistemas difusos emplean fuzificadores para mapear las amplitudes de las señales del mundo real dentro del rango [0,1]. Esto es usualmente realizado por una curva especial llamada función de membresía. La elección de la distribución de estas funciones tiene un impacto importante en el funcionamiento del sistema difuso. Esta distribución puede ser adaptada por un algoritmo genético optimizando el desempeño del sistema difuso.

d) **Neural-Difuso-Genético:** Este híbrido puede ser configurado de muchas formas. Una configuración simple incluye una red neuronal como un clasificador de patrones, entrenada con distribuciones de funciones de membresías, las cuales han sido pre-ajustadas por un algoritmo genético. Entre otras configuraciones, el algoritmo genético puede emplearse junto con sistema neural-difuso para optimizar sus parámetros.

e) **Caótico-Difuso:** Aunque la relación entre la lógica difusa y la teoría del caos no está aún completamente comprendido en la actualidad, el estudio de sus interacciones se ha llevado a cabo durante más de dos décadas, por lo menos con respecto a los siguientes aspectos: el control difuso del caos, sistemas difusos adaptables de series de tiempo caóticas, relaciones teóricas entre la lógica difusa y la teoría del caos, modelado difuso de sistemas caóticos con propiedades asignadas, modelos difusos Takagi-Sugeno castificados, y criptografía difusa-caótica.

Capítulo 2

Sistemas Inteligentes VLSI

2.1 Implementación VLSI.

Cuando se necesita implementar un sistema VLSI, independientemente de la aplicación, siempre se toman en cuenta varios factores, algunos de diseño y otros de costo. Al realizar el diseño se deben considerar aspectos tales como la velocidad de respuesta, el consumo de potencia, consumo de área en silicio y actualmente se necesita que sean sistemas reconfigurables o reprogramables. Dado esto, es muy importante elegir la técnica de diseño de circuitos a utilizar: digital, analógica, tiempo continuo o discreto, etc. Es por esto, que es muy importante tener en cuenta que tipo de bloques u operaciones se requieren en el procesamiento de la señal. Por ejemplo, para implementar una RN o un sistema difuso convencional los bloques básicos de procesamiento que se necesitan son de suma, resta, multiplicación o división, y dependiendo de la función de activación puede ser una función exponencial o tangencial. La complejidad de un circuito analógico para realizar estas operaciones es mínima, dado que la naturaleza de los dispositivos permite obtener funciones exponenciales incluso con un solo transistor, mientras que un procesador digital (secuencial o embebido) solo para realizar la suma de números binarios de un bit necesita varios transistores [34]. Para poder realizar operaciones exponenciales o tangenciales el consumo de transistores es mucho mayor, lo que lleva a un consumo de área en silicio mucho mayor. Se habla de una gran precisión en sistemas digitales, sin embargo, esta precisión tiene un precio muy caro en lo que respecta a costos según lo anterior. Los multiplicadores utilizados por las RN en un sistema digital son diseñados a partir de sumadores, mientras que los multiplicadores analógicos son circuitos compactos que necesitan pocos transistores, o si se considera que estos pesos son una especie de válvula que regula el flujo sináptico, este multiplicador se puede reducir a un transistor de paso. Otro aspecto a considerar es que la velocidad de los sistemas digitales se ve limitada a su arquitectura. Si la arquitectura es secuencial y el algoritmo muy complejo, el sistema puede llegar a ser muy lento. Una solución es hacer uso de arquitecturas paralelas, donde la arquitectura será siempre limitada por el consumo de área y potencia debida al uso masivo de transistores conmutando continuamente. Es por esto que con circuitos analógicos, que tienen una respuesta instantánea

y son compactos, se han diseñado e implementado arquitecturas neuronales y difusas con un desempeño igual o mayor que el obtenido con arquitecturas digitales [34-36]. Así mismo, se ha logrado diseñar arquitecturas o celdas analógicas con capacidad de reconfiguración. El siguiente paso a nivel hardware es la integración de los mecanismos de adaptación basados en los AE. En [37, 38] se ha demostrado que es posible la adaptación de sistemas inteligentes utilizando sistemas digitales y los AG, dada su representación binaria. Ahora se verá con un poco más de detalle los diseños VLSI de cada paradigma, así como las limitantes derivadas de la forma en que están implementadas.

2.2 Redes Neuronales.

Debido a la similitud entre las neuronas biológicas y artificiales, el hardware analógico parece una buena opción para implementarlas teniendo como resultado implementaciones más baratas que operan a altas velocidades. Por otro lado las aproximaciones digitales ofrecen por mucho, mayor flexibilidad y gran exactitud. Así mismo, los chips digitales se pueden diseñar sin la necesidad de conocimiento avanzado de los circuitos utilizando sistemas de ayuda asistida por computadora (CAD), no siendo así para el diseño de chips analógicos los cuales requieren el conocimiento de la física del transistor al igual que la experiencia. Una ventaja de las implementaciones analógicas de redes neuronales sobre las digitales, es su gran similitud a las leyes físicas presentes en las redes neuronales biológicas. Por ejemplo, los pesos sinápticos pueden ser codificados por un solo elemento analógico (resistencia o transistor), contrariamente a los sistemas digitales donde muchos elementos son requeridos. Además, reglas muy simples como las leyes de Kirchhoff pueden ser utilizadas para la agregación de señales de entrada adicionales.

Los chips ideados por el grupo de Carver Mead en la Universidad de Caltech [39] son redes neuronales inspiradas en sistemas biológicos. Mead trato de construir chips neuronales analógicos que copiaran todo lo posible a las redes biológicas, y que además tuvieran un extremo bajo consumo de potencia, circuitos completamente analógicos y un procesamiento en tiempo continuo. Los circuitos deben ser capaces de abarcar las representaciones de los datos usados por las neuronas en las diferentes partes del cerebro. En un sistema neuronal, la computación muchas veces es una serie de transformaciones de una representación a otra, así como las salidas más importantes se transmiten a un siguiente nivel. Un estímulo activa cierto

número de neuronas que responden a un rango limitado de entradas, pero cuyas regiones de sensibilidad se superponen. Como resultado, se utilizan patrones espacio-temporales de actividad de una población de celdas para representar los datos. Con estos principios en mente fue posible convertir los circuitos diseñados en sistemas. Como ejemplo se puede mencionar un sistema diseñado para procesamiento de imágenes. En la Figura 2.1 se muestra la retina artificial diseñada. Varios experimentos muestran que la retina de silicio funciona de manera similar a la retina biológica. Se muestran similitudes entre las sensibilidades de intensidad, el tiempo de respuesta para una sola salida cuando destellos de luz son las entradas, así como la respuesta al contraste de bordes.

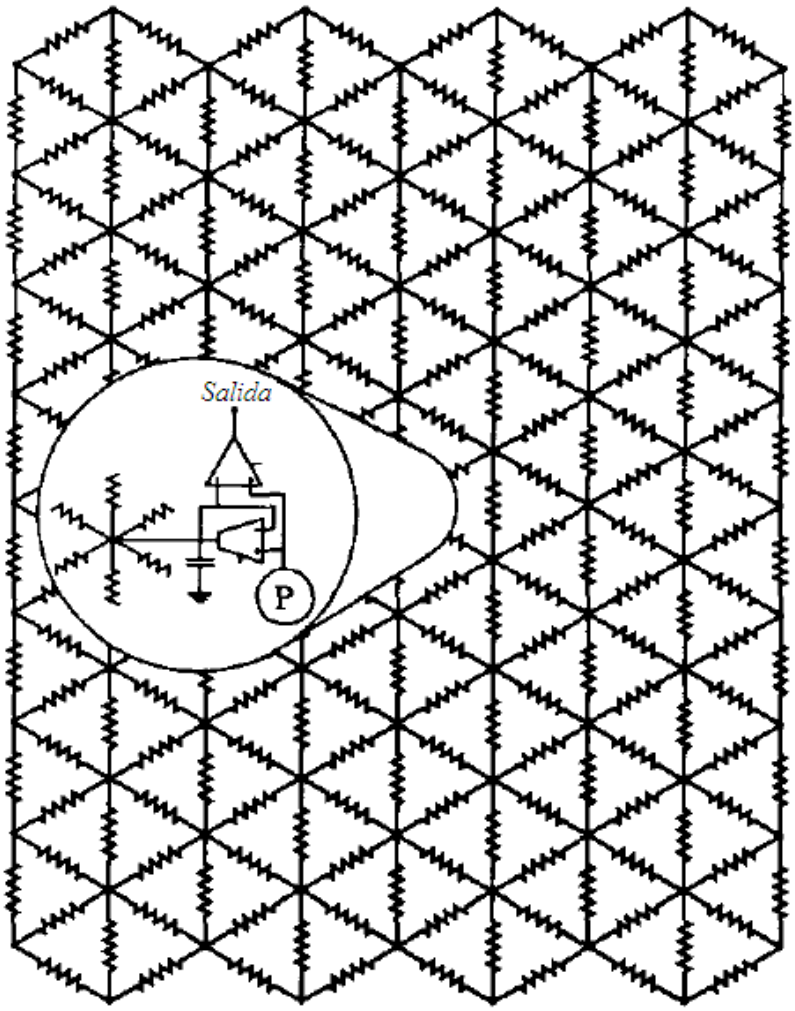


Figura 2.1 Retina Artificial.

Una aproximación radicalmente diferente es el chip LNeuro desarrollado en los Laboratorios de Electrónica de Philips (LEP) en Francia [40]. Mientras que la mayoría de los chips neuronales implementan redes Hopfield o en algunos casos redes Kohonen, este neuro-chip digital puede ser configurado para integrar cualquier regla de aprendizaje o topología de la red. En la Figura 2.2 se muestra la arquitectura del chip LNeuro, donde por claridad solo se muestra una arquitectura de 4 neuronas. Este chip consta de una parte de multiplicación y suma, y otra parte para aprendizaje. El chip LNeuro 1.0 cuenta con un nivel de paralelismo de 16. Los pesos son de 8 bits de longitud en fase de relajación y de 16 bits en fase de aprendizaje. La unidad aritmética lógica (alu) tiene una entrada externa que permite la acumulación de productos parciales externos. Esto puede utilizarse para construir redes grandes, estructuradas o de alta precisión. Los productos parciales son almacenados y sumados en un árbol de sumadores, por lo que la cantidad de operaciones de cómputo se incrementa linealmente con el número de neuronas (en vez de incrementarse cuadráticamente como en simulaciones o máquinas seriales). Por razones de flexibilidad, las funciones de activación permanecen fuera del chip. Los resultados de los cálculos de las sumas ponderadas salen del chip serialmente y el resultado debe ser escrito de nuevo en los registros de estado neural.

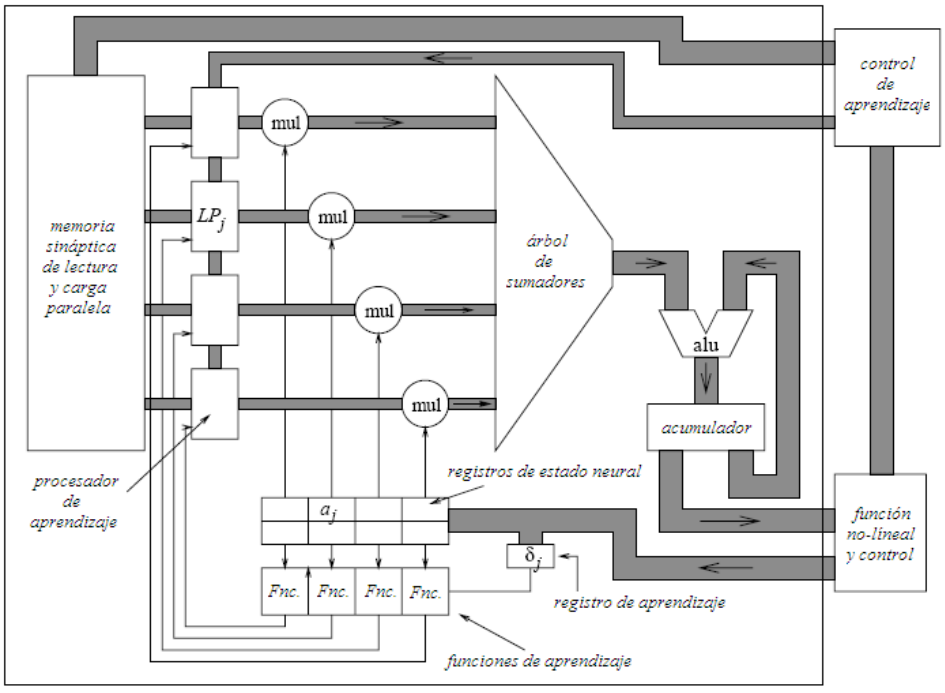


Figura 2.2 Chip LNeuro.

Para integrar redes neuronales de gran tamaño o muchas capas, es necesario conectar varios chips entre ellos. Cuando solo unas pocas neuronas deben ser conectadas, los chips pueden conectarse en renglones subsecuentes como en redes de trayectoria directa y esto no es problema. Pero en otros casos, cuando la cantidad de neuronas de un chip a conectarse con las neuronas de otro chip es considerable hay ciertos problemas:

- el diseño con empaques para chips con un gran número de pins de entrada o salida,
- la capacidad de salida del chip (fan-out), ordinariamente cada chip tiene la capacidad de enviar señales a un número pequeño de otros chips. Debido a esto, se requieren amplificadores, los cuales son costosos en disipación de potencia y consumo de área,
- el ruteo, siendo este uno de los problemas más serios.

Como se ha visto, la gran fortaleza de las RN es su capacidad de aprendizaje. Mientras que una red con valores de pesos pre-calculados y fijos podría tener cierto mérito en aplicaciones industriales, la adaptabilidad en línea sigue siendo un objetivo de diseño para la mayoría de los sistemas neuronales.

2.3 Sistemas Difusos.

Las implementaciones de SD vía software son fácilmente obtenibles y proveen alta flexibilidad ya que soportan sistemas con un número arbitrario de reglas, sin limitaciones en el número y tipo de funciones de membresía y un amplio rango de mecanismos de inferencia. Por otro lado, las aproximaciones por software no son adecuadas para aplicaciones que demandan SD de tamaño pequeño, bajo consumo de potencia y alta velocidad de inferencia. Para cumplir con los requerimientos impuestos por este tipo de aplicaciones se debe optar por aproximaciones a nivel hardware. Una opción es emplear microprocesadores de propósito general, ocasionalmente mejorados con nuevas instrucciones o nuevos circuitos. Otra opción es diseñar hardware dedicado, es decir circuitos integrados de aplicación específica (ASIC), en este caso optimizados para sistemas de inferencia basados en lógica difusa. En la tabla 2.1 se resumen cuatro tipos de implementación de sistemas difusos.

Tabla 2.1 Opciones de implementación de Sistemas Difusos

Implementación	Características	Velocidad
Completamente Software	Simple, bajo costo	Baja
CPU incluyendo instrucciones dedicadas difusas	Operaciones difusas de alta velocidad, costo moderado	Mediana
CPU apoyado por un coprocesador difuso	Operaciones difusas de alta velocidad, costo moderado pero el tiempo de manejo de datos limita la velocidad total de operación	Alta
ASIC de inferencia difusa	Alto costo, poca generalización	Muy alta

El desarrollo del primer controlador difuso se relaciona con dos grupos de investigación: Togai-Watanabe [41] por un procesador difuso digital y Yamakawa-Miki [42] por un procesador difuso analógico. Ambos desarrollos fueron los primeros en una serie de arquitecturas de hardware que utilizan técnicas analógicas y digitales. Mientras que el primer chip digital implemento 16 reglas y tuvo una velocidad estimada de 80,000 FLIPS (inferencias lógicas difusas por segundo) sin la desdifución, su contraparte analógica usó funciones de membresía triangulares y necesito un área más pequeña que su contraparte, pero fue diseñado con componentes discretos no integrados. La velocidad de inferencia fue de 1μs independientemente del número de reglas. La etapa de desdefución no formó parte de las mediciones ya que el método del centro de gravedad necesitó un tiempo relativamente largo debido al circuito que realizaba la división matemática. Estas primeras implementaciones mostraron algunas desventajas que necesitan ser mejoradas. La solución para estas desventajas entre las diferentes implementaciones, se basa en los siguientes criterios:

- reducción de área, especialmente para arquitecturas digitales,
- unidades de procesamiento rápidas para todos los pasos del árbol del control difuso: fuzificación, inferencia, desdifución, medidos en FLIPS,

- generación automática de los bloques difusos básicos,
- flexibilidad en el intercambio de bases de conocimiento: términos difusos, base de reglas.

La mayoría de las realizaciones digitales existentes consideraron principalmente funciones de membresía triangulares o trapezoidales, que pueden ser estimadas fácilmente por tablas de búsqueda. No solamente los términos lingüísticos son almacenados en bloques de memoria, sino también las bases de reglas. Muchas aproximaciones utilizan circuitos de modo mixto combinando técnicas digitales y analógicas. Otra clase de hardware difuso es el sintetizable. Esta flexibilidad se alcanza a través de la generación automática de un controlador difuso ASIC basado en especificaciones VHDL o a través de una clase de bloques funcionales difusos que pueden ser generados automáticamente dependiendo de la aplicación [43]. Los desarrollos analógicos pueden subdividirse en controladores de modo corriente [42,44-47], controladores de modo voltaje [48,49] y controladores de modo transconductancia (voltaje-corriente) [36,50]. Dependiendo de la implementación llegan a tener velocidades de inferencia de 1MFLIPS [51] y 10MFLIPS [52] en un proceso CMOS de 1.2μm.

Las técnicas analógicas de tiempo continuo, ofrecen una buena relación entre consumo de área y velocidad de inferencia. Esto es logrado por un lado con la eliminación de señales de control y por otro lado, explotando completamente la potencialidad del elemento básico de un circuito integrado CMOS (el transistor MOS).

La ventaja de utilizar estas técnicas es que las operaciones requeridas por cada regla pueden ser realizadas también en paralelo con circuitos que admiten múltiples entradas, bajo consumo de potencia y ocupan pequeñas áreas de silicio. Desde un punto de vista de la arquitectura, una característica relevante de los sistemas difusos es su paralelismo inherente. Por un lado, varias reglas son activadas para cada combinación de entradas. Por otro lado, varios datos pueden ser procesados en paralelo por cada regla, tales como los antecedentes y los puntos discretos del universo de salida. Para evitar la necesidad de hacer un barrido de los puntos del universo de salida, la mayoría de los procesadores analógicos implementan sistemas Takagi-Sugeno de orden cero. Además del paralelismo, otra característica

importante a nivel sistema es la programabilidad del chip. Los parámetros que pueden ser programados en un chip difuso son los siguientes: el número de conjuntos difusos para cubrir los antecedentes de las reglas y los parámetros que definen sus funciones de membresía, el número de conjuntos difusos para los consecuentes de las reglas y los parámetros que definen sus funciones de membresía y los parámetros que conforman la base de reglas. Si el procesador difuso no es programable, todos estos parámetros son fijados durante el proceso de fabricación, de otra manera se necesitarían circuitos de programación con lo cual se incrementa la complejidad de acuerdo al número de parámetros programables.

Las técnicas analógicas en tiempo discreto, trabajan con señales analógicas muestreadas en el tiempo. Dos técnicas ampliamente conocidas son los capacitores conmutados y las corrientes conmutadas, que son técnicas en modo voltaje y modo corriente respectivamente. Las técnicas de capacitores conmutados pueden considerarse como una solución intermedia entre fiabilidad, flexibilidad, proceso de diseño altamente automatizado para realizaciones digitales, bajo consumo de área, bajo consumo de potencia y un proceso de diseño manual de realizaciones analógicas en tiempo continuo. Los elementos básicos de los diseños de capacitores conmutados son interruptores controlados por señales de reloj y capacitores lineales, que usualmente se fabrican con dos capas de polisilicio. Otros elementos empleados son los amplificadores operacionales, amplificadores de transconductancia y comparadores. Una desventaja de los capacitores conmutados es que no son adecuados para operar con voltajes de alimentación pequeños. Estas limitaciones nos llevan al desarrollo de técnicas de corriente conmutada. Los diseños de corriente conmutada también emplean interruptores, pero sus otros componentes básicos son más simples, tales como espejos de corriente o simplemente un transistor. Por otro lado, los capacitores utilizados no necesitan ser lineales permitiendo el uso de los capacitores parásitos existentes en las compuertas de los transistores. De esta forma el diseño simple de los circuitos de corriente conmutada los hace más compactos. Una desventaja podría ser la baja precisión que tienen los circuitos de corriente conmutada comparados con los circuitos de capacitores conmutados.

Mientras que las técnicas analógicas en tiempo continuo son adecuadas para el procesamiento paralelo de señales, las técnicas en tiempo discreto son adecuadas para el procesamiento secuencial de señales ya que los datos pueden ser almacenados en algunos bloques y transmitirse a otros bloques de acuerdo a las señales de control. Las arquitecturas

secuenciales generalmente emplean menos circuitos que las arquitecturas completamente paralelas, ya que un mismo bloque puede implementar varias operaciones requeridas en diferentes intervalos de tiempo y por lo tanto llegan a ser más efectivas en términos de área y consumo de potencia. Por otro lado, el tiempo que les lleva calcular la salida es mayor debido a que el procesamiento no se desarrolla en un solo paso. Como consecuencia, el elegir una arquitectura completamente paralela, completamente secuencial o mixta dependerá de las restricciones de área y velocidad. Cuando se seleccione el tipo de arquitectura, el diseñador debe considerar si una arquitectura completamente paralela puede implementarse en un solo chip, ya que si esta requiere conectar varios chips, los retardos introducidos pueden dar como resultado una arquitectura serial, haciendo esta menos efectiva.

En las Figuras 2.3a a 2.3d se muestran los bloques requeridos para implementar un sistema difuso con técnicas mixtas [36] y lograr cumplir con el requerimiento de programabilidad. Se puede observar que el procesamiento de las señales se logra mediante celdas de arquitectura analógica en tiempo continuo. Sin embargo, con el objetivo de obtener la programabilidad de dichas celdas, se modifican algunos bloques básicos, como los espejos de corrientes, agregándoles o sustituyéndolos por convertidores digitales-analógicos. Los convertidores ayudarán en la tarea de programar los parámetros del sistema difuso de forma sistemática y controlada digitalmente. Sin embargo, esto nos lleva a incrementar el número de pins en el encapsulado agregando n-bits (n-pins) por parámetro. Si no se dispone de los pins necesarios, será necesario agregar un sistema de multiplexado dentro del chip más algunas líneas de control en el encapsulado, incrementando el consumo de área. En las Figuras 2.4a a 2.4d se muestran los bloques utilizados para implementar un sistema difuso con técnicas completamente analógicas en tiempo continuo [47]. A diferencia del sistema anterior, en este sistema se logra tener una programación de los parámetros del sistema difuso en tiempo continuo haciendo uso de celdas también completamente analógicas. La ventaja obtenida con este tipo de diseño es el ahorro en consumo de área al evitar el uso de convertidores digitales-analógicos, además de la reducción en el uso de pins en el encapsulado del chip.

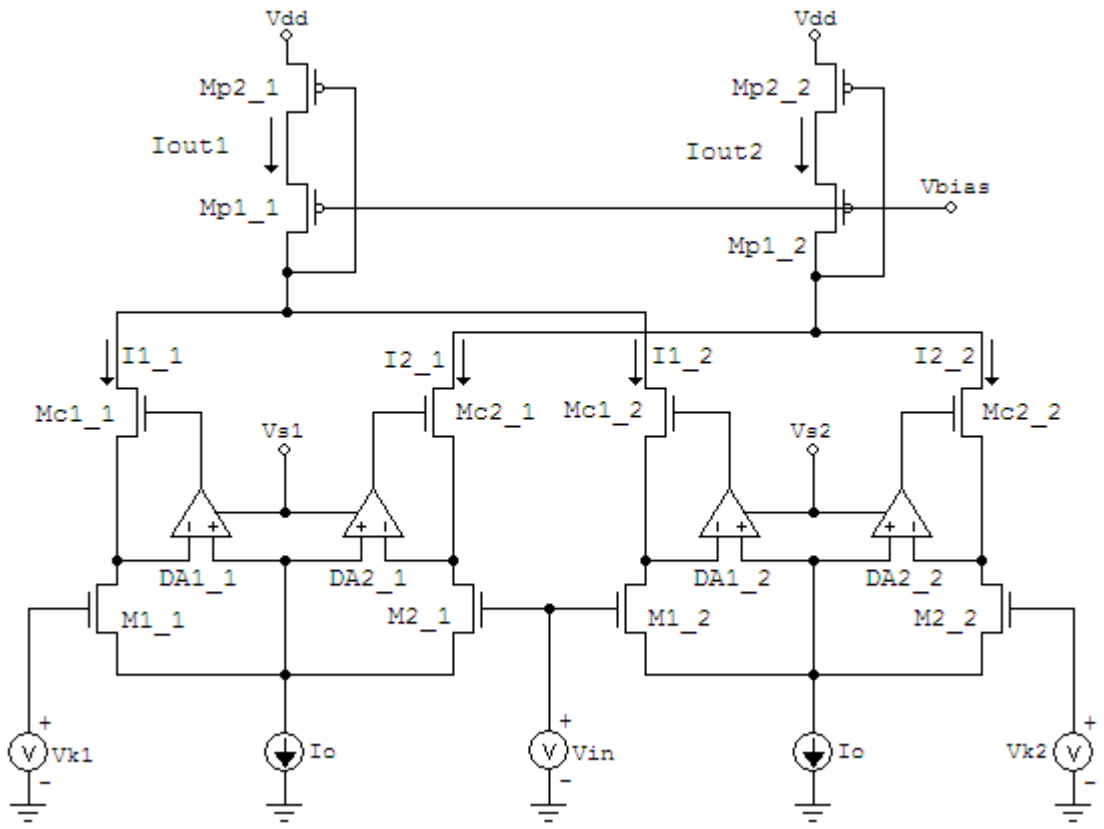


Figura 2.3 a) Circuito para una función de membresía programable.

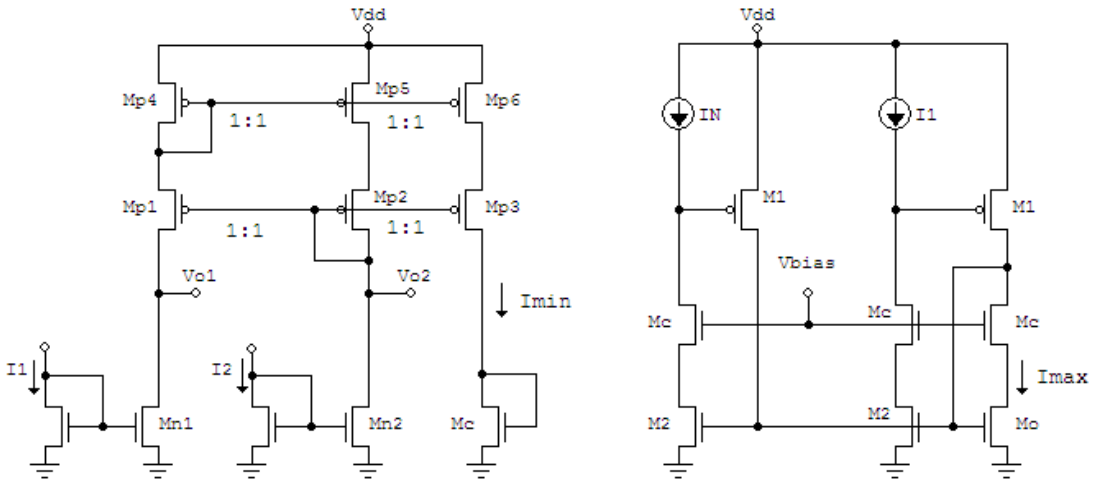


Figura 2.3 b) Circuitos para los operadores min-max.

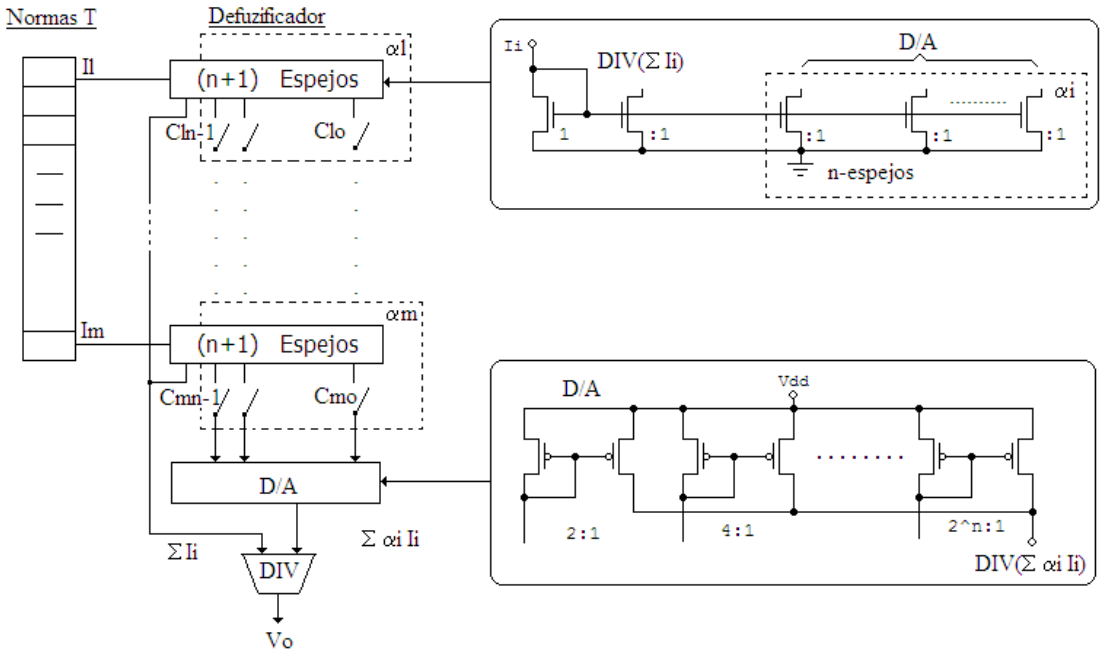


Figura 2.3 c) Circuito defuzificador.

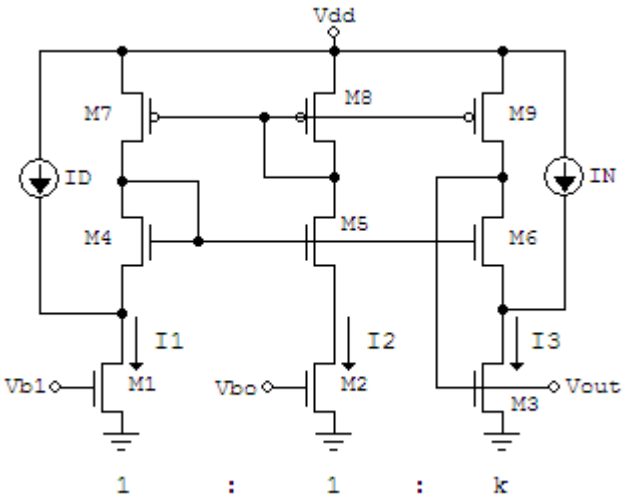


Figura 2.3 d) Circuito divisor.

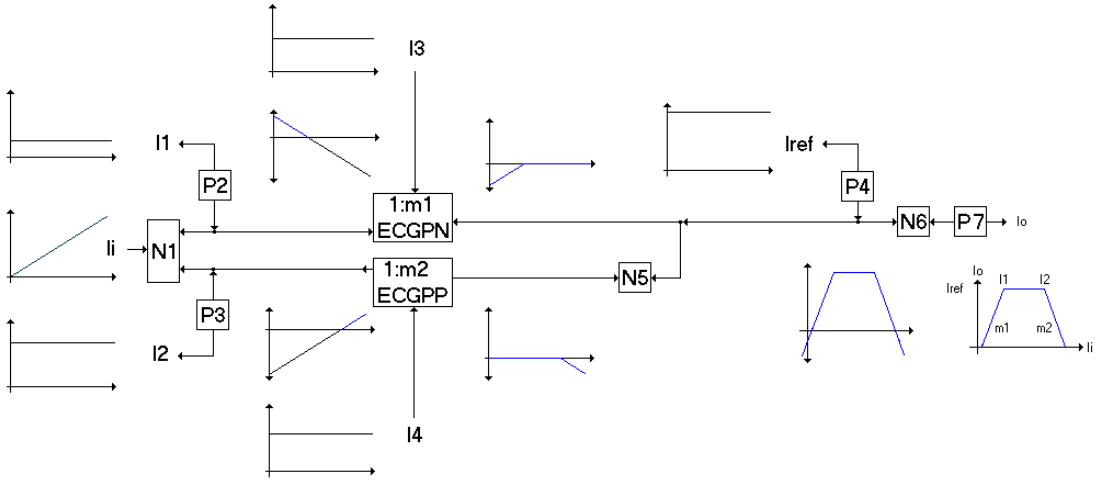


Figura 2.4 a) Circuito en modo corriente para el cómputo de la función de membresía programable, donde cada bloque N o P son espejos de corriente.

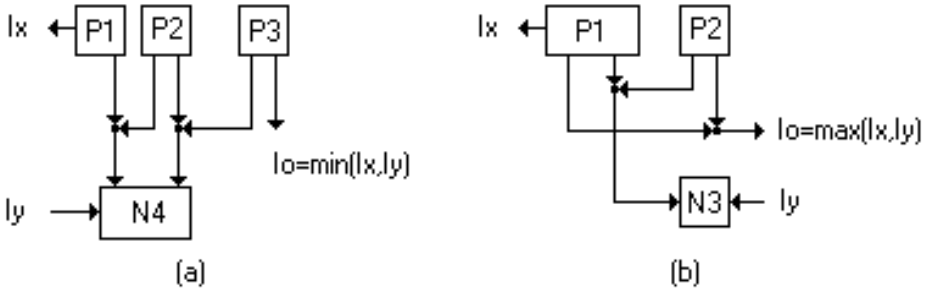


Figura 2.4 b) Circuitos para los operadores min-max.

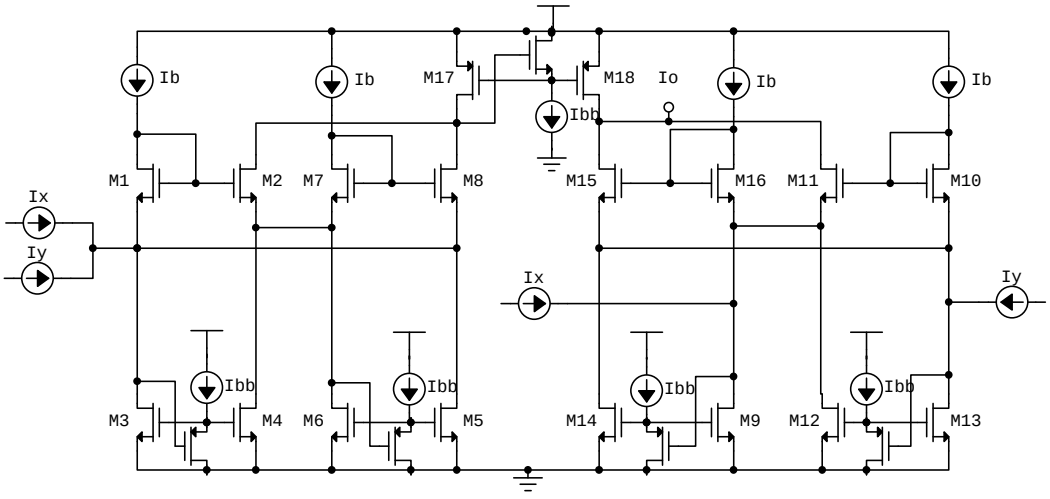


Figura 2.4 c) Circuito multiplicador.

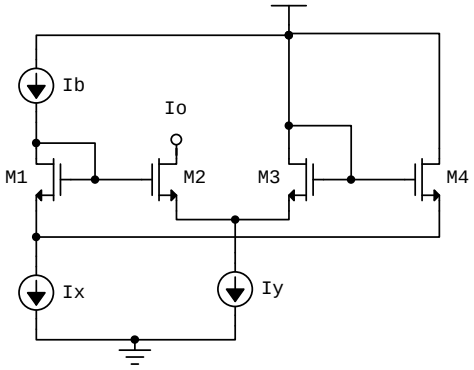


Figura 2.4 d) Circuito divisor.

2.4 Algoritmos Evolutivos

En años recientes ha surgido un gran interés por la realización en hardware de algoritmos evolutivos [53-55]. El objetivo original de estos sistemas evolutivos fue el de ayudar en la optimización o adaptación dentro del mismo circuito integrado de otros sistemas. Esto dio lugar a la investigación y desarrollo de un nuevo hardware con características evolutivas. El Hardware Evolutivo (HE) es considerado como la integración de la Computación Evolutiva y dispositivos programables. El principal objetivo del HE es la reconfiguración automática de la estructura del hardware y mejorar su desempeño. La capacidad de reconfiguración automática del HE lo hace diferente del hardware común fundamentalmente porque este último queda incapacitado para cambiar la función de su hardware una vez fabricado. No obstante, existe la posibilidad de fabricar sistemas con hardware evolucionado. Es decir, hardware que se ha creado a través de un proceso de continuo refinamiento haciendo uso de técnicas evolutivas y detenido cuando se obtenga un diseño que cumpla con los requerimientos establecidos.

Los métodos para crear HE abarcan técnicas digitales, analógicas o mixtas. Cuando se habla de síntesis de circuitos, el objetivo es diseñar circuitos que satisfagan un conjunto de especificaciones funcionales y de tiempo de procesamiento. Se debe contar con información completa sobre el entorno físico de operación. Normalmente no existen prototipos o hardware preliminar. Básicamente, los diseños se realizan desde cero, los problemas en la síntesis de circuitos tienen tres cosas en común que el HE puede explotar.

Primero, virtualmente no hay restricciones de recursos de cómputo, el algoritmo evolutivo puede ejecutarse en cualquier sistema de cómputo, desde una computadora portátil hasta un sistema multiprocesador paralelo de procesamiento masivo. La escasez de memoria no tiene relevancia. Segundo, es posible una evolución extrínseca o intrínseca. No solo es posible medir el desempeño del sistema mediante simulaciones del sistema, también pueden realizarse mediciones físicas utilizando osciloscopios sofisticados de almacenamiento digital o analizadores de espectros y tomarlas como guía en el proceso de síntesis. Tercero, los problemas de síntesis basados en HE son resueltos en ambientes de laboratorio. Bajo estas condiciones hay pocas restricciones en tiempos de simulación o el número de generaciones que un algoritmo evolutivo puede ejecutar. Los métodos de síntesis son considerados métodos de evolución fuera de línea (*off-line*).

Cuando se habla de hardware adaptable, se habla de hardware que siempre se adapta *in-situ* (en su lugar). Este tipo de ambiente operacional presenta una serie de enormes desafíos: los algoritmos evolutivos tienen que ejecutarse con recursos de cómputo limitados, tales como microcontroladores de baja velocidad y memoria limitada; el algoritmo evolutivo muchas veces tiene un tiempo de ejecución máximo predefinido; no es posible instrumentar una prueba concreta de la evolución del hardware; y la evolución es realizada con poca o nula intervención humana o supervisión.

Si bien una parte fundamental en la adaptación de HE son los algoritmos evolutivos, no se puede dejar de mencionar al hardware que hace posible la realización de estos algoritmos. En el caso de HE digital, básicamente se cuenta con dos tipos de dispositivos programables: los dispositivos lógicos programables (PLD) y los campos de arreglos de compuertas programables (FPGA). Mientras que el hardware digital cada vez se vuelve más y más potente, existen algunos problemas que requieren de circuitos analógicos. Como por ejemplo los sensores siempre utilizan una arquitectura analógica para medir una cantidad física del mundo analógico, filtros analógicos o a veces de circuitos de procesamiento de señal analógicos. En el caso del HE analógico también se cuenta con dos tipos de dispositivos programables: los campos de arreglos analógicos programables (FPAA) y los campos de arreglos de transistores programables (FPTA). Este último tipo de dispositivo es el que cuenta con la mayor flexibilidad de diseño, pudiendo obtener sistemas analógicos e incluso sistemas digitales o mixtos.

Debe hacerse notar que independientemente del dispositivo programable, sea digital o analógico, los mecanismos evolutivos o algoritmos evolutivos han sido implementados de forma digital. Los algoritmos genéticos han sido una alternativa de mucha demanda ya que sus codificaciones binarias le permiten una adaptación directa y simple en hardware digital. En la Figura 2.5 se muestra una celda básica para la implementación de algoritmos genéticos en hardware digital.

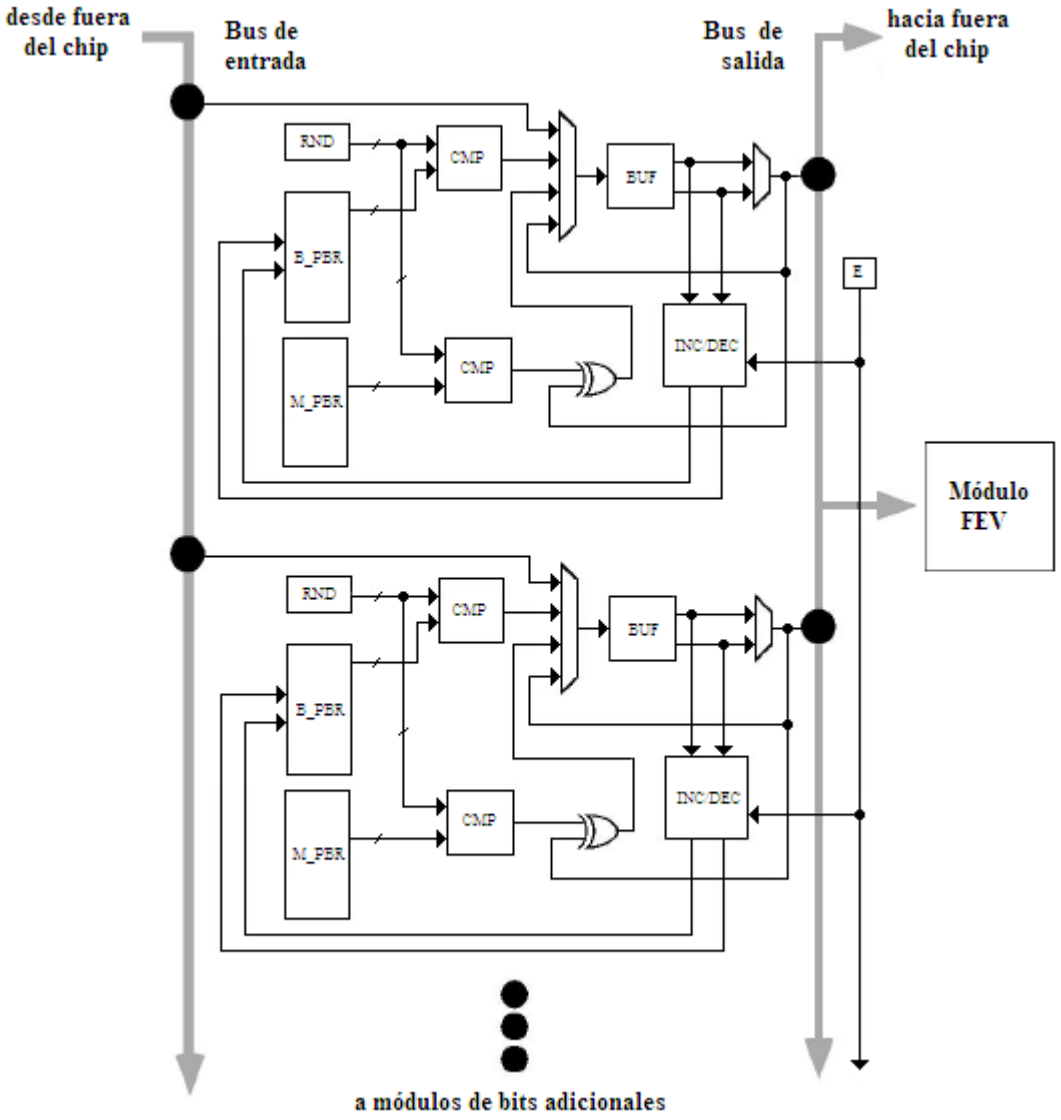


Figura 2.5 Data Path completo para implementación de Algoritmos Genéticos en hardware.

2.5 Sistemas Caóticos

En el Capítulo 1 se introdujeron conceptos necesarios para la comprensión de sistemas caóticos. Si bien los conceptos vistos se analizaron para modelos basados en mapas logísticos, cuando se habla de realizaciones en hardware existen otros modelos que pueden llegar a tener una implementación electrónica. Desde 1983, L.O. Chua se cuestionó sobre temas concernientes al caos y la sincronización de señales, obteniendo aproximaciones de estos sistemas mediante circuitos eléctricos con funciones de transferencia por trazos lineales [56]. El circuito de Chua para generar señales caóticas se puede considerar como una evolución del modelo de Van der Pol al cual se le agregó un circuito RC como el que se muestra en la Figura 2.6. De forma similar, este cuenta con un circuito resonante y una resistencia no lineal. En la Figura 2.7 se muestra de forma gráfica la función de transferencia (corriente vs. voltaje) requerida para la resistencia no lineal.

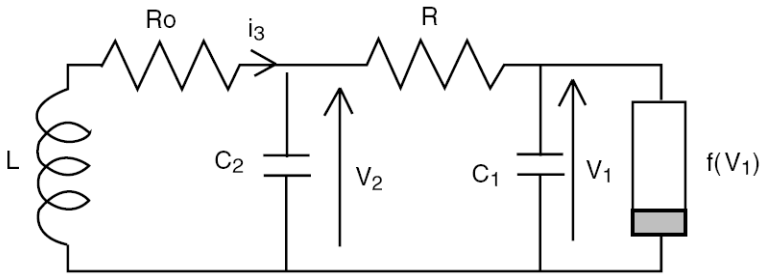


Figura 2.6 Circuito de Chua.

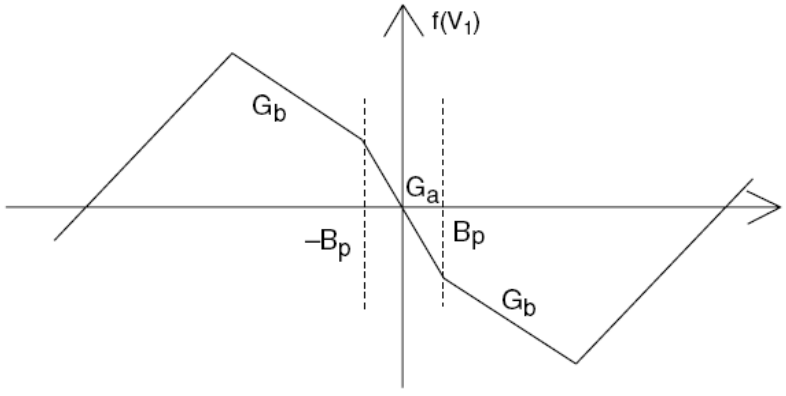
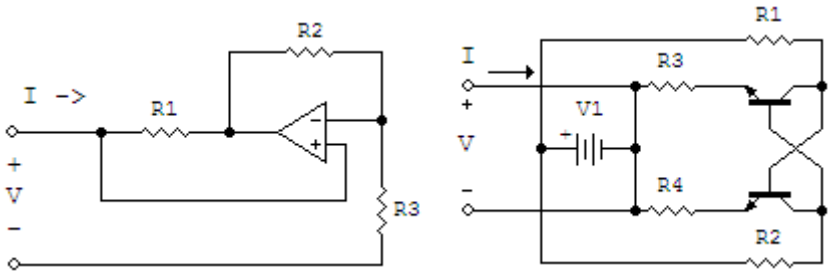


Figura 2.7 Función de transferencia de la resistencia no lineal.

Tomando como referencia el circuito de Chua, se puede observar que a simple vista el circuito parece ser simple y de fácil implementación. Sin embargo, a nivel vlsi se sabe que es posible realizar capacitores con buena precisión, no siendo así para el caso de las resistencias y las inductancias. Las variaciones en los valores de las resistencias dependerán del tipo de material que se disponga en el proceso de fabricación, y considerando que en este tipo de diseños un parámetro de ajuste y control es el valor de dichas resistencias, es importante para el diseñador tomar una buena decisión respecto al valor y la forma en que se va a implementar este componente. Algo similar sucede con las inductancias, generalmente hechas con un conductor enrolladas sobre un núcleo. Este tipo de inductancias no son posibles en diseños vlsi dado que solo es posible realizar estructuras planas. Este tipo de situaciones hace que el factor de calidad Q de las inductancias requerido para el diseño no sea fácil de conseguir. Así mismo, todos los componentes deben diseñarse de tal forma que los valores requeridos no sean muy grandes, de lo contrario esto implicará un incremento considerable en el consumo de área.

No obstante lo anterior, esto no es considerado el principal problema en la realización de sistemas caótico en base al circuito de Chua. El principal inconveniente es la necesidad de una resistencia no lineal, que además requiere que sea capaz de tomar valores negativos. Físicamente no existe un dispositivo que tenga este tipo de respuesta por si solo. Esto ha llevado a la búsqueda de circuitos que puedan cumplir con los requisitos de la resistencia no lineal. En la Figura 2.8 se muestran algunas realizaciones de la resistencia no lineal [57].



(a) (b)

Figura 2.8 Resistencia no lineal hecha con a) un amplificador y b) transistores bipolares.

Como se puede observar en la Figura 2.8a, la implementación de la resistencia no lineal nos obliga a utilizar dispositivos como lo son los amplificadores (de voltaje, corriente, etc.) incrementando el consumo de área. En el caso de la Figura 2.8b se hace uso de transistores bipolares, los cuales comúnmente no están disponibles en una tecnología CMOS convencional. Esto implicaría utilizar tecnologías especializadas y costosas. De ahí que sea preferible una implementación con base en amplificadores si la aplicación lo amerita, aún si el consumo de área se incrementa.

Los sistemas caóticos como el circuito de Chua, se pueden considerar como generadores de señales caóticas en tiempo continuo. Existen realizaciones en tiempo discreto basadas en mapas logísticos [58-62], que parten de suponer que se tiene un modelo iterativo de la forma $X(n+1) = f(\ X(n)\ )$. La idea detrás de estos sistemas es simple, obtener un circuito con una función de transferencia que represente algún mapa logístico para posteriormente hallar la forma de retroalimentarlo en tiempo discreto. Es decir, tener un bloque de muestreo y retención que conecte la salida del circuito hasta su entrada como se muestra en la Figura 2.9. Se puede observar que el circuito de muestreo y retención puede ser tan simple como utilizar dos interruptores, dos capacitores como elementos de memoria y un buffer controlados por señales de reloj. El circuito del buffer dependerá de que tipo de señal se este procesando, en el caso de corrientes puede ser solo un espejo de corriente o un amplificador de ganancia unitaria en el caso de voltajes. En las Figuras 2.10a a 2.10d se muestran algunas implementaciones de mapas logísticos.

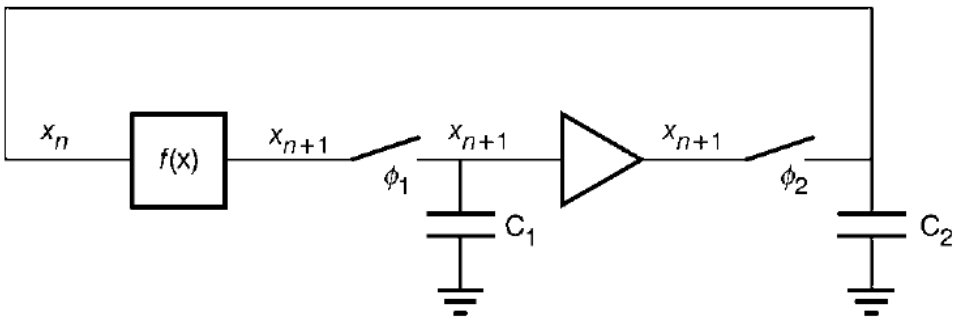


Figura 2.9 Retroalimentación del mapa logístico.

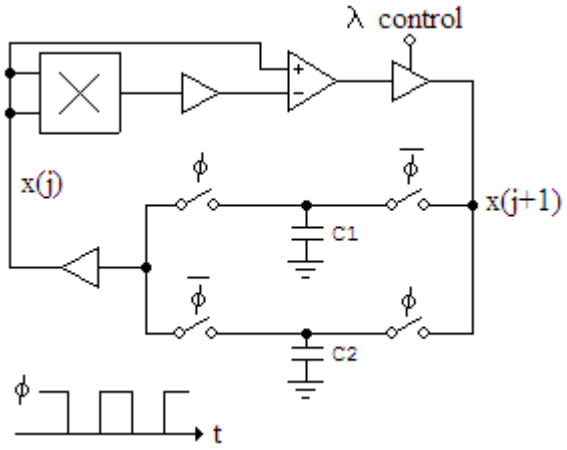


Figura 2.10 a) Mapa Logístico para la generación de ruido [60].

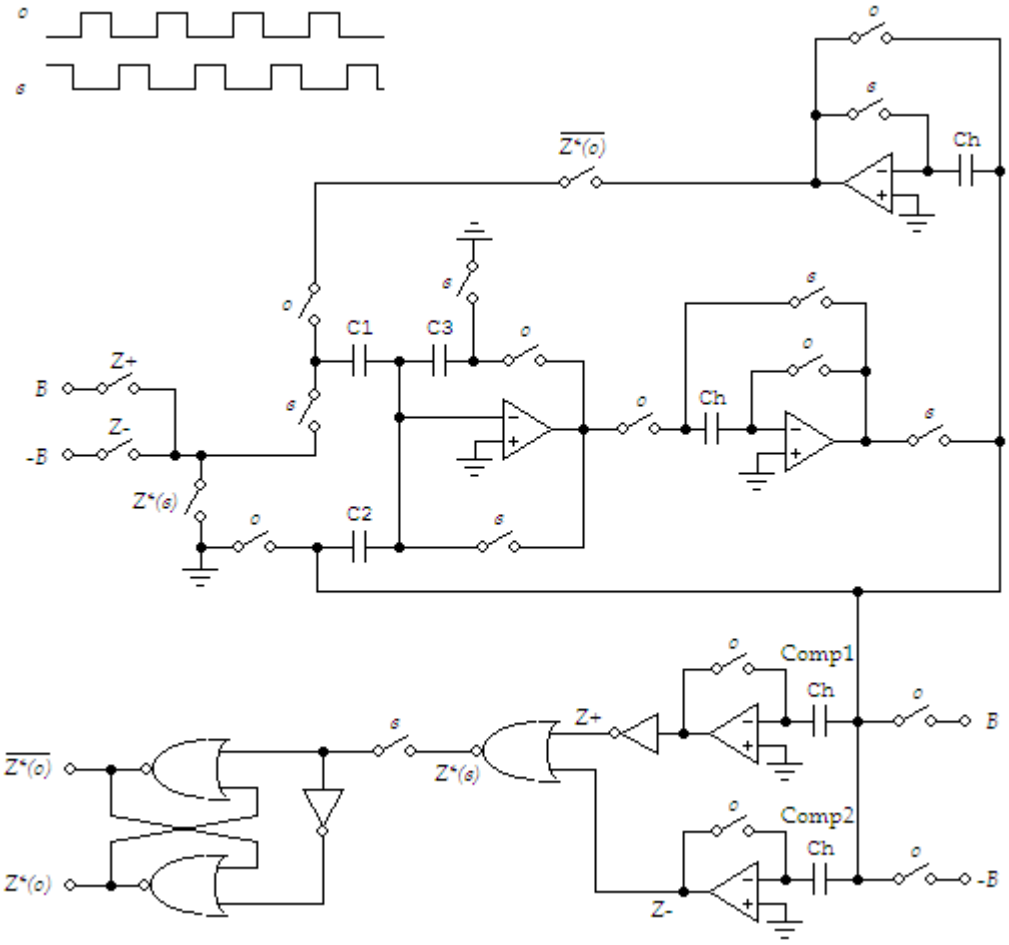


Figura 2.10 b) Mapa Logístico para la generación de ruido 1/f [61].

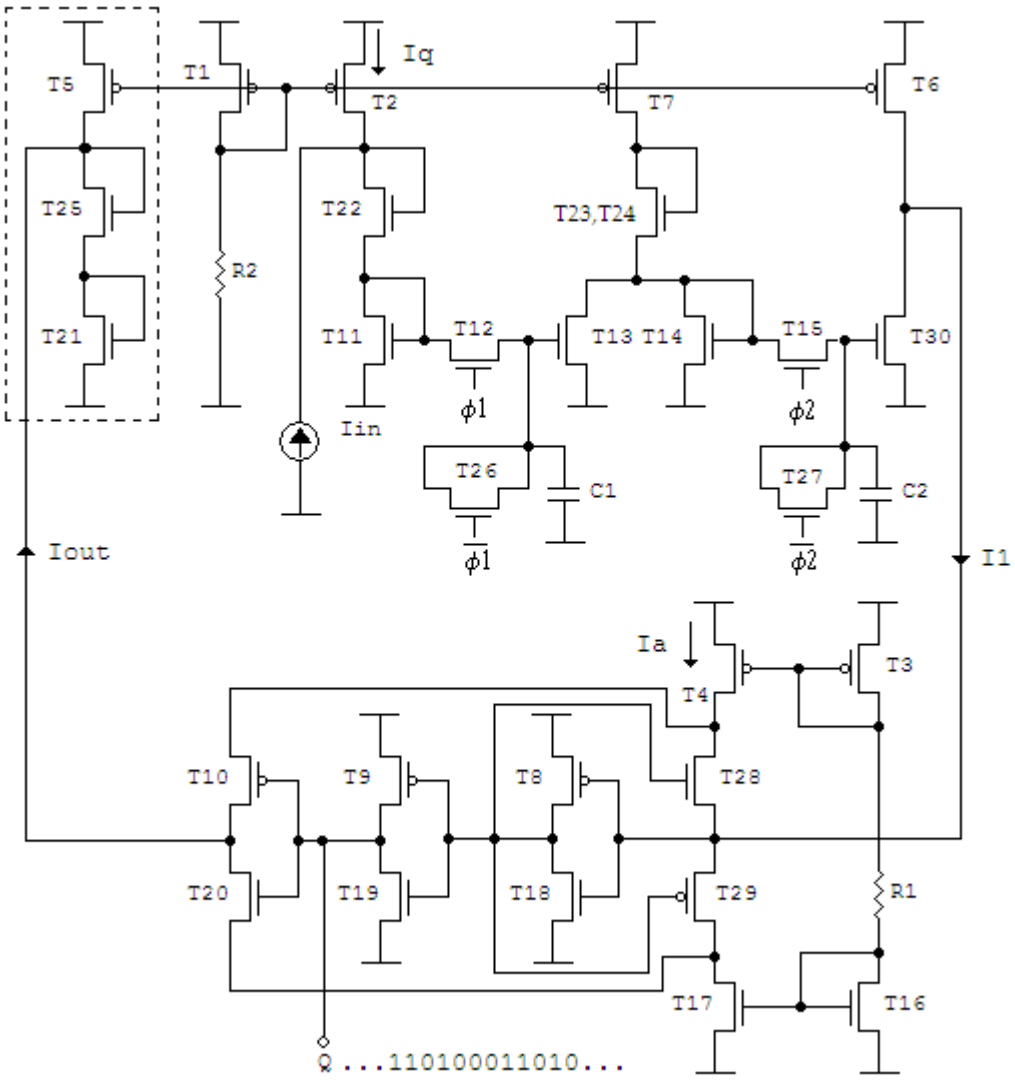


Figura 2.10 c) Mapa Logístico para la generación de números aleatorios [62].

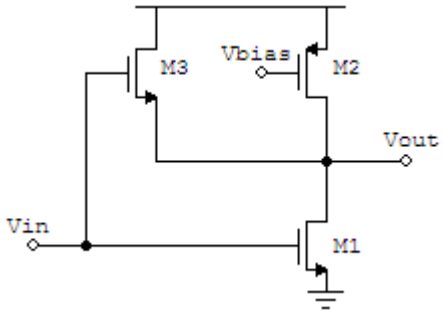


Figura 2.10 d) Mapa Logístico para la generación de señales caóticas [63].

2.6 Sistemas Híbridos.

Como se ha visto anteriormente, cuando se trata de diseñar sistemas híbridos se debe hacer un análisis de que es útil y que no de cada uno de los paradigmas involucrados. También se ha visto que en implementaciones en software de sistemas híbridos para simulación o síntesis de sistemas inteligentes no hay muchas restricciones, puede ser posible la integración de tantos paradigmas como se desee. Así también, se ha visto que esto no se cumple cuando se trata de realizar una implementación en hardware.

Sistemas híbridos muy utilizados son los Difuso-Neuronal y Neuronal-Difuso, donde principalmente se explota la capacidad de aprendizaje de las redes neuronales y se aplica en la adaptación de un sistema difuso o se explota la capacidad de la lógica difusa para procesar datos imprecisos integrando bloques difuso dentro de la arquitectura de la red neuronal. Como se ha visto, el hardware de procesamiento, sea difuso o neuronal, no tiene muchos inconvenientes en poder implementarse. Sin embargo, cuando se piensa en como integrar un método de aprendizaje como el de retro-propagación se empieza a tener inconvenientes. Analizando al algoritmo, se observa que hay que realizar cálculos matemáticos sobre derivadas parciales. Este tipo de cálculo se puede resolver de forma analítica con la ayuda de software. A nivel hardware es simple pensar en calcular derivadas con respecto al tiempo sin ningún problema, algo que es inherente en todo hardware. Sin embargo, cuando se habla de derivadas parciales, se habla de variaciones de una variable con respecto a variaciones de otra variable, lo que implica que en algún momento se debe realizar una división numérica.

Los circuitos que realizan la división digitalmente tienen el problema de que se construyen en base a multiplicaciones o sumas y restas secuenciales bajando con esto el desempeño en velocidad. Un circuito divisor analógico puede ser más compacto y veloz que su contraparte digital, pero no evita compartir con el divisor digital el problema de la división por un valor muy pequeño que tienda a cero donde se espera que el resultado sea infinito. Entonces se tiene que es posible integrar sistemas híbridos de este tipo siempre y cuando factores como la velocidad digital o la imprecisión analógica no sean factores determinantes en el diseño.

Los sistemas híbridos difusos-evolutivos o neuronales-evolutivos pueden ser realizados sin muchas complicaciones en hardware digital o mixto, no siendo así para hardware analógico. Como se vio anteriormente, los algoritmos genéticos son muy utilizados para la implementación digital de los algoritmos evolutivos. Una característica de estos algoritmos es la necesidad de generar múltiples soluciones y evaluarlas. Si se implementa para procesar de forma paralela todas las soluciones, es de esperarse que el consumo de área se incremente directamente proporcional al número de soluciones generadas y evaluadas. De lo contrario se necesitaría un arquitectura serial la cual se volvería más lenta conforme se incremente el número de soluciones por generar y evaluar.

Ahora bien, al comparar el desempeño de los tres principales paradigmas del soft-computing de acuerdo a los datos de la Tabla 2.2. Respecto a la adaptabilidad, se ve que los AE y las RN tienen buen desempeño, no así para los SD al igual que con la habilidad de aprendizaje. Analizando el resto de las comparaciones, se puede deducir que un sistema híbrido con RN y SD o AE y SD puede cubrir satisfactoriamente todos los parámetros de desempeño. Esto corrobora todo lo visto anteriormente, solo queda decidir cual de estos dos es más apropiado para su implementación en hardware. Si se considera que uno de los parámetros que más importancia tiene cada día es el de velocidad de respuesta, y en base a todo lo expuesto anteriormente sobre el desempeño de los diferentes tipos de hardware, la mejor opción es realizar los sistemas en hardware analógico.

Tabla 2.2 Comparativa de los tres paradigmas del Soft-Computing.

	RN	SD	AE
Representación de Conocimiento	M	**B**	m
Tolerancia a Incertidumbre	**B**	**B**	**B**
Tolerancia a imprecisión	**B**	**B**	**B**
Adaptabilidad	**B**	m	**B**
Habilidad de Aprendizaje	**B**	M	**B**
Capacidad de Interpretación	M	**B**	m
Descubrimiento de Conocimiento	**B**	m	b
Capacidad de Mantenimiento	**B**	b	b

Malo	M	Algo malo	m	Algo bueno	b	Bueno	**B**

Si se opta por realizar un aprendizaje mediante RN será difícil eliminar el problema de la división numérica. Solo nos queda optar por el uso de AE como método de aprendizaje. En la tabla 1.3 se puede observar que las EE tienen la capacidad de utilizar representaciones de datos reales y no binarios como en los AG. Esto es favorable pensando en que los sistemas analógicos por su naturaleza también procesan datos reales y no binarios. Otro punto que se debe observar en la tabla 1.3 es el de la velocidad de convergencia. En el caso de las EE existe la posibilidad de implementar algoritmos compactos ((1+1)EE) teniendo buen desempeño en velocidad. Si se considera que el algoritmo solo requiere generar una solución en cada iteración, se puede deducir que el hardware requerido será menor comparado con el requerido por otros algoritmos, teniendo como resultado un consumo menor de área en silicio.

Una característica de los AE es que generan soluciones de forma heurística utilizando valores aleatorios. En el caso de los sistemas electrónicos analógicos existen dos tipos de señales con características aleatorias. El primer tipo son señales ruidosas, que generalmente se obtienen del ruido térmico u otro tipo de ruido generado en los dispositivos electrónicos. El segundo tipo son las señales caóticas generadas con circuitos caóticos como los vistos anteriormente. De esta forma se ve que es factible el diseño de sistemas inteligentes híbridos vlsi analógicos de buen desempeño. Una vez más se confirma que haciendo un análisis de los diferentes paradigmas del soft-computing, es posible tener un sinergismo entre las diferentes disciplinas y obtener una solución al problema en cuestión. En el siguiente capítulo se abordará el diseño de una arquitectura analógica para la implementación de sistemas Difusos-Evolutivos VLSI.

Capítulo 3
Diseño del Sistema Difuso-Evolutivo

3.1 Arquitectura del Sistema Difuso-Evolutivo.

Como se ha visto anteriormente, existen muchas formas para implementar sistemas difusos electrónicamente. También se ha visto que el uso de circuitos analógicos puede permitir que se obtenga un buen desempeño en velocidad y bajo consumo de área al diseñar un sistema difuso. Se sabe que existen técnicas que procesan voltajes o corrientes, en tiempo continuo o discreto. Si lo que se busca es disminuir la complejidad de la arquitectura del sistema difuso, se debe optar por una técnica que permita realizar operaciones básicas como la suma, resta y multiplicación con el mínimo de circuitos. En este caso se optará por el diseño de bloques de procesamiento difusos diseñados para procesar datos en modo corriente en tiempo continuo. Sin embargo, las entradas y salidas del circuito integrado que se utilizarán aceptan voltajes como señal eléctrica. Esto implica que se deberán utilizar bloques de conversión voltaje-corriente y corriente voltaje.

Se debe considerar que el sistema difuso será diseñado para procesar corrientes en tiempo continuo, por lo que los parámetros adaptables también serán corrientes. Esto implica que los bloques evolutivos deberán generar las soluciones en forma de corriente. Dado que el algoritmo evolutivo como tal es un proceso iterativo o discreto, como primera aproximación electrónica del sistema evolutivo, se diseñarán circuitos de procesamiento en modo corriente y tiempo discreto. En la Figura 3.1 se muestra la arquitectura del sistema difuso-evolutivo. Como se puede observar, se debe contar con dos sistemas difusos, generadores de números aleatorios, memorias, sumadores, restadores y un comparador. El objetivo es crear una arquitectura basada en las estrategias evolutivas. Un sistema difuso servirá para evaluar la solución padre o mejor adaptada, el otro servirá para probar las soluciones hijo. La salida de ambos sistemas será restada a una referencia para obtener el error de ambos sistemas. Los errores serán comparados para saber cual solución es la más apta. El comparador generará una señal *k* con la cual se deberá activar el almacenamiento en memoria de la solución hijo en caso de ser la mejor adaptación al problema. Las soluciones hijo serán generadas mediante la suma de un dato aleatorio a la solución padre.

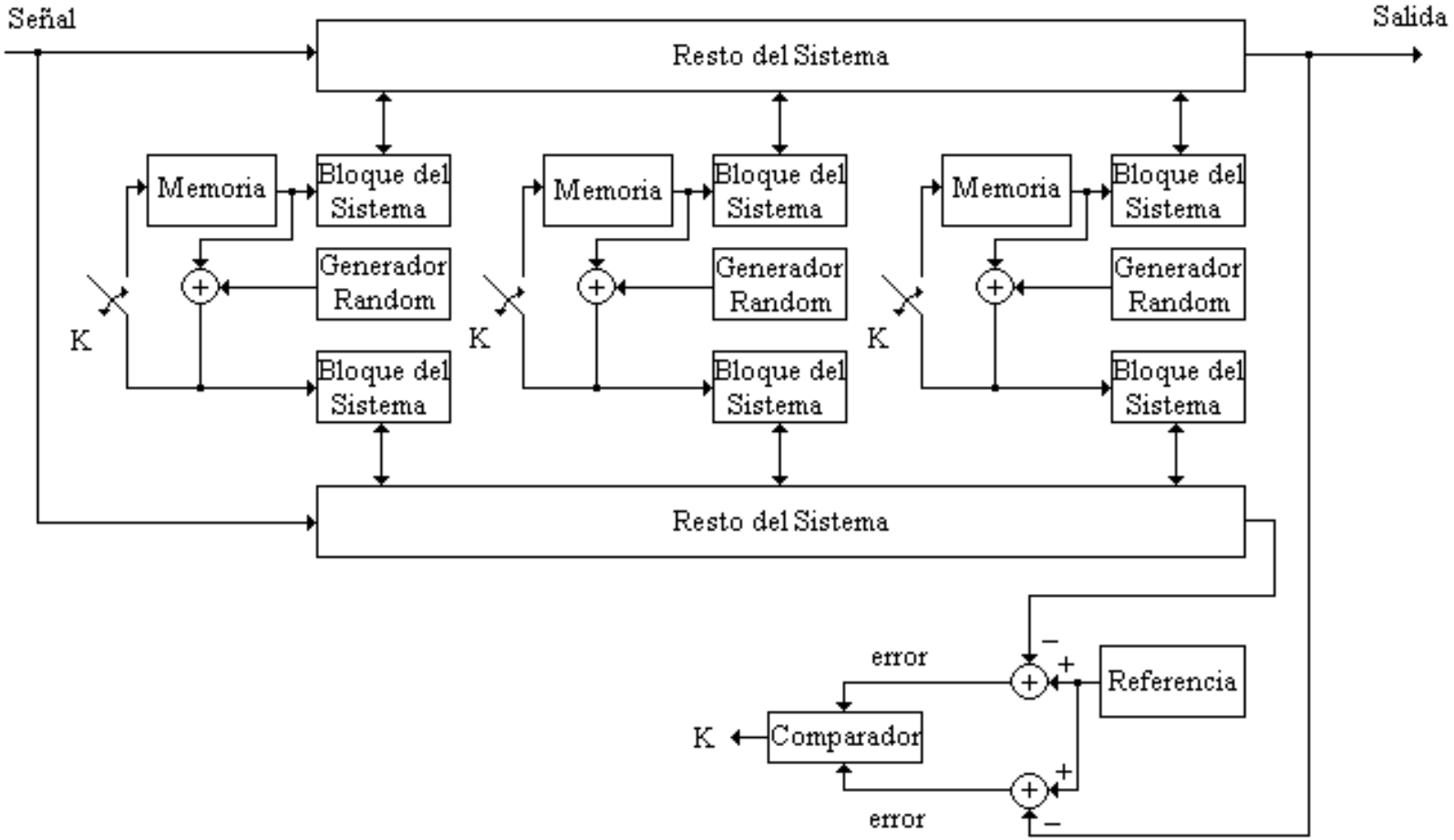


Figura 3.1 Arquitectura del sistema difuso-evolutivo.

Si bien los bloques de procesamiento necesarios son simples, se puede ver que se requerirá de dos sistemas difusos con parámetros ajustables, además de un número de generadores aleatorios dependiendo del número de parámetros a ajustar. Se tiene una arquitectura de procesamiento paralelo, esto permitirá tener un buen desempeño en velocidad de respuesta pero será necesario pagar el costo en consumo de área. Ahora se verá el diseño de los bloques más a detalle.

3.2 Bloques funcionales difusos.

Como se mencionó anteriormente, es necesario un bloque que convierta los voltajes en corrientes. Un circuito que realiza esta tarea es el amplificador operacional de transconductancia (OTA). En las Figuras 3.2a a 3.2c se muestran el circuito del OTA utilizado, la respuesta en DC y la transconductancia del amplificador respectivamente. De estas gráficas se puede obtener el rango de señal de entrada en el cual se tiene una respuesta lineal, es decir, la corriente de salida es directamente proporcional al voltaje diferencial de entrada.

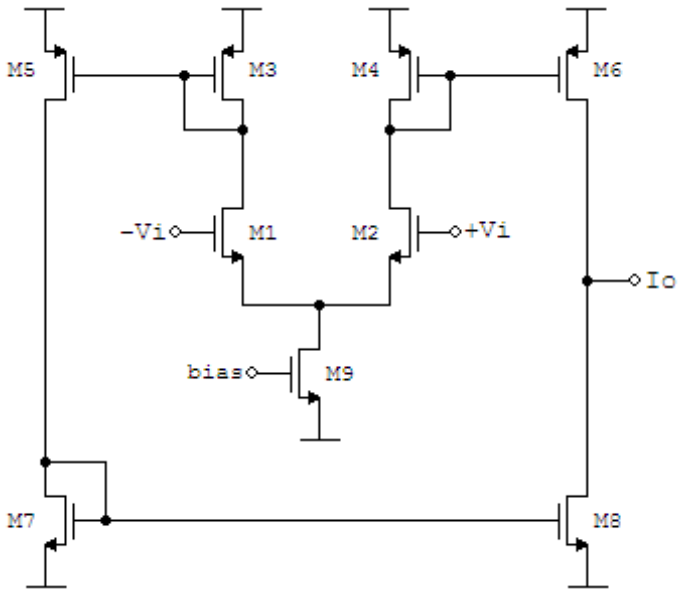


Figura 3.2a Amplificador Operacional de Transconductancia.

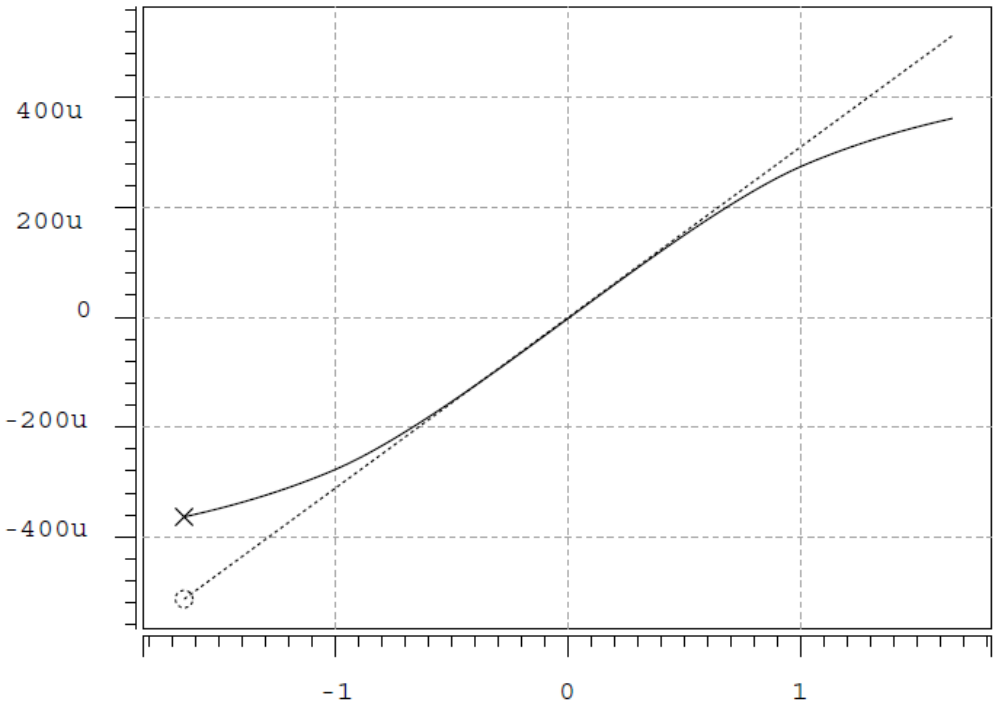


Figura 3.2b Corriente de salida vs. Voltaje diferencial de entrada.

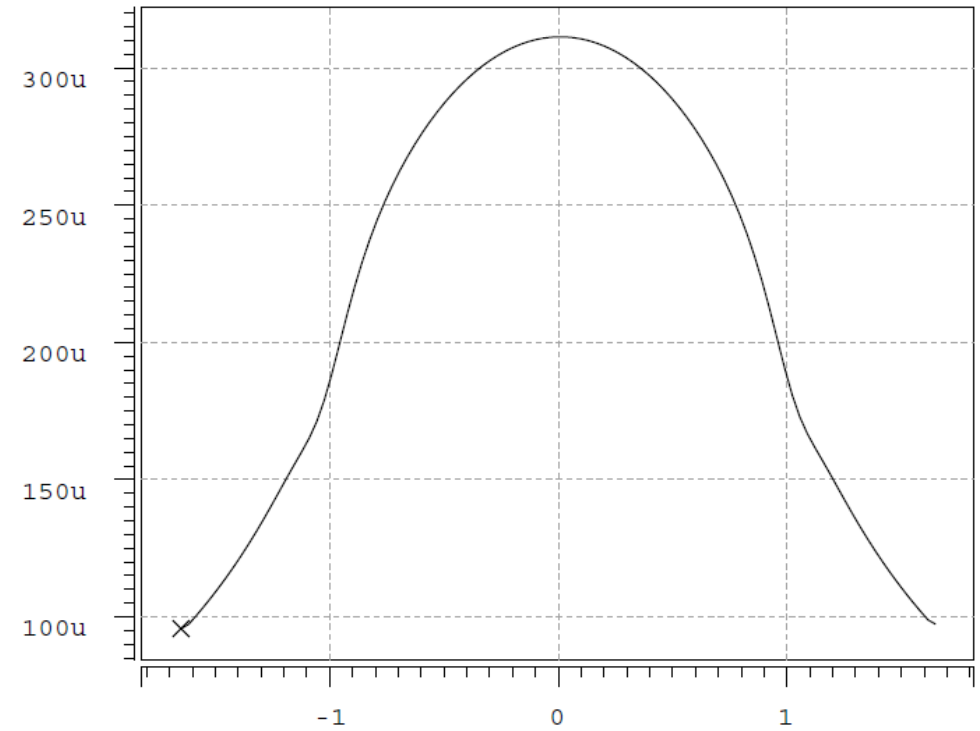


Figura 3.2c Transconductancia (g_m) del amplificador.

Al analizar la Figura 3.2c, se observa que la transconductancia no es constante. Es decir, no se tendrá una respuesta completamente lineal. Sin embargo, en un pequeño rango alrededor de -0.5V a +0.5V, la variación de g_m ya no es tan grande comparada con la variación presente en todo el rango de entrada. Si se observa la Figura 3.2b en ese mismo rango, se verá que la corriente de salida (línea continua) no se desvía considerablemente de la línea recta. Y es aquí donde precisamente se espera que el procesamiento difuso absorba la imprecisión presente en la conversión de la señal.

Se ha mencionado que el procesamiento difuso se realizará en modo corriente. También se mencionó que se debe convertir la señal resultante en voltaje con el fin de poder medirla en la salida del circuito integrado. Para lograr esto es necesario hacer uso de un amplificador operacional de transimpedancia. Una forma simple de construir un amplificador de este tipo es la que se muestra en la Figura 3.3. En esta figura se muestra al mismo amplificador de la Figura 3.2a, solo que esta vez cuenta con un lazo de retroalimentación negativa. Este tipo de conexión hace que el amplificador se comporte como una resistencia. Al inyectar una corriente, esta se vera reflejada como un voltaje en las terminales del amplificador.

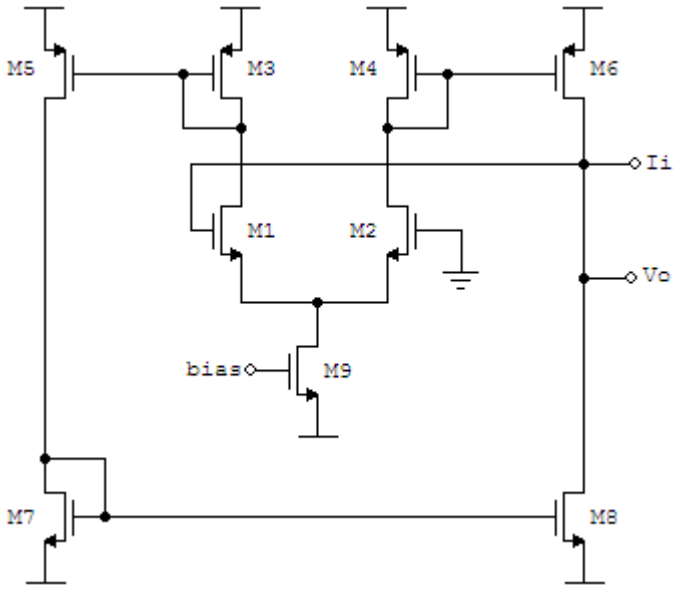


Figura 3.3 Amplificador operacional de transconductancia conectado como resistor.

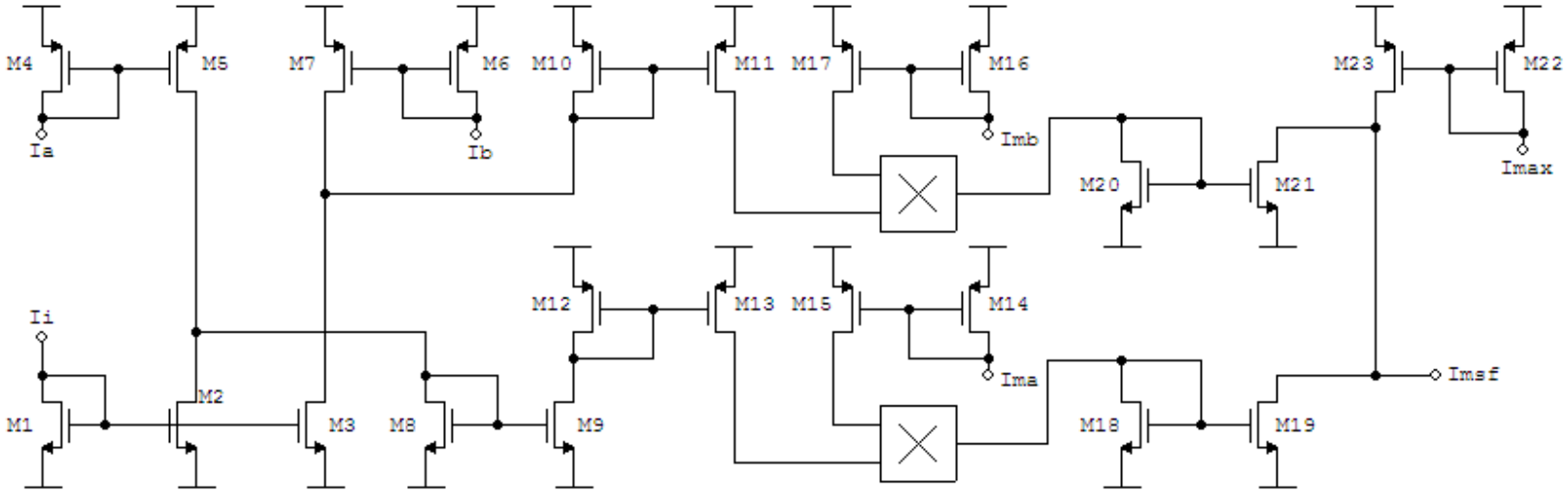


Figura 3.4 Circuito generador de la función Π.

Como es de esperarse, esta resistencia activa hereda las características no lineales del amplificador de transconductancia. Dado que el nodo de entrada en forma de corriente es el mismo nodo de salida pero en forma de voltaje, se debe tener cuidado de no exceder los rangos de respuesta lineal permitidos por el amplificador de transconductancia. Ahora se verá cuales son los bloques de procesamiento difusos.

3.2.1 Fuzificación.

De ahora en adelante todos los bloques serán circuitos que procesan señales en modo corriente. En el caso de la fuzificación, se requiere de un circuito que genere funciones de membresía. En modo corriente es posible diseñar circuitos que aproximan funciones por trazos. Se puede diseñar un circuito que genere una función tipo $\prod$.

Recordando de la tabla 1.2, esta función cuenta con 4 parámetros ajustables *a*, *b*, *c* y *d* para determinar su forma final. Si se igualan *b* y *c* se obtendrá una función tipo Λ. Si *b* es igual al valor mínimo que pueda tomar la variable de entrada se tendrá una función tipo L. Si *c* es igual al valor máximo que pueda tomar la variable de entrada se tendrá una función tipo Γ. De esta forma se puede utilizar el circuito de la Figura 3.4 donde se muestra un circuito que genera una función tipo $\prod$ con base en el circuito propuesto en [63].

Es posible integrar este mismo circuito tantas veces se requiera una función de membresía, si el sistema de adaptación logra ajustar sus parámetros, los circuitos realizarán las funciones de membresía correspondientes. Si ya se tiene en mente un sistema con funciones tipo Λ, Γ o L, es posible rediseñar el circuito y disminuir el número de transistores para obtener celdas más compactas. Como se puede observar, solo se utilizan espejos de corriente y multiplicadores, lo que permite que el rediseño sea simple.

Si se requiere una función tipo Λ se debe igualar I_a y I_b, se podría eliminar M6 y conectar la compuerta de M7 a la compuerta de M5. Si se requiere una función tipo Γ, se deberán eliminar los transistores M6, M7, M10, M11, M16, M17, M20, M21 y el multiplicador asociado a estos transistores. Si se requiere una función tipo L, se deberán eliminar los transistores M4, M5, M8, M9, M12, M13, M14, M15, M18, M19 y el multiplicador asociado a estos transistores. De igual forma, si se requieren funciones con

pendientes simétricas, se podría eliminar M16 y conectar la compuerta de M17 a la compuerta de M15.

Nótese que el uso de espejos de corriente nos da gran flexibilidad en el diseño de este tipo de bloques difusos. En este caso, se hace uso de multiplicadores para manipular las pendientes de la función de forma lineal. El circuito multiplicador se analizará en detalle más adelante. En la Figura 3.5 se muestra la simulación de una función de membresía tipo Π programable con diferentes puntos de quiebre y pendientes. En la Figura 3.6 se muestra la simulación de un conjunto de funciones de membresía tipo L, Λ, Π, Γ, simétricas y asimétricas.

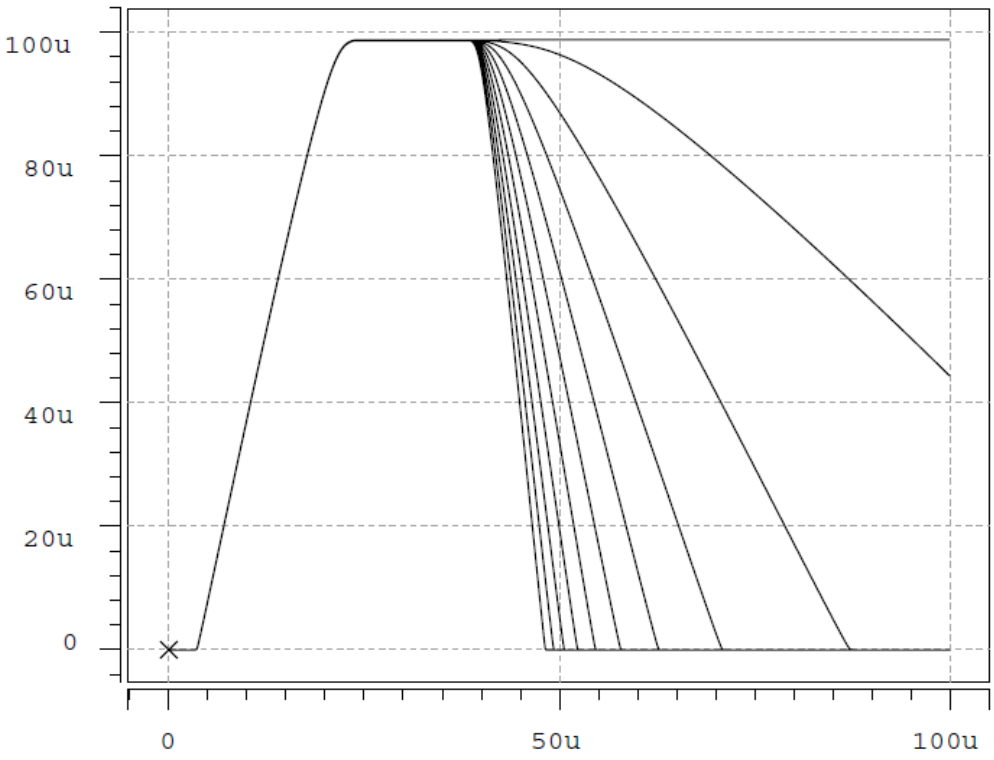


Figura3.5 Función de membresía programable en modo corriente.

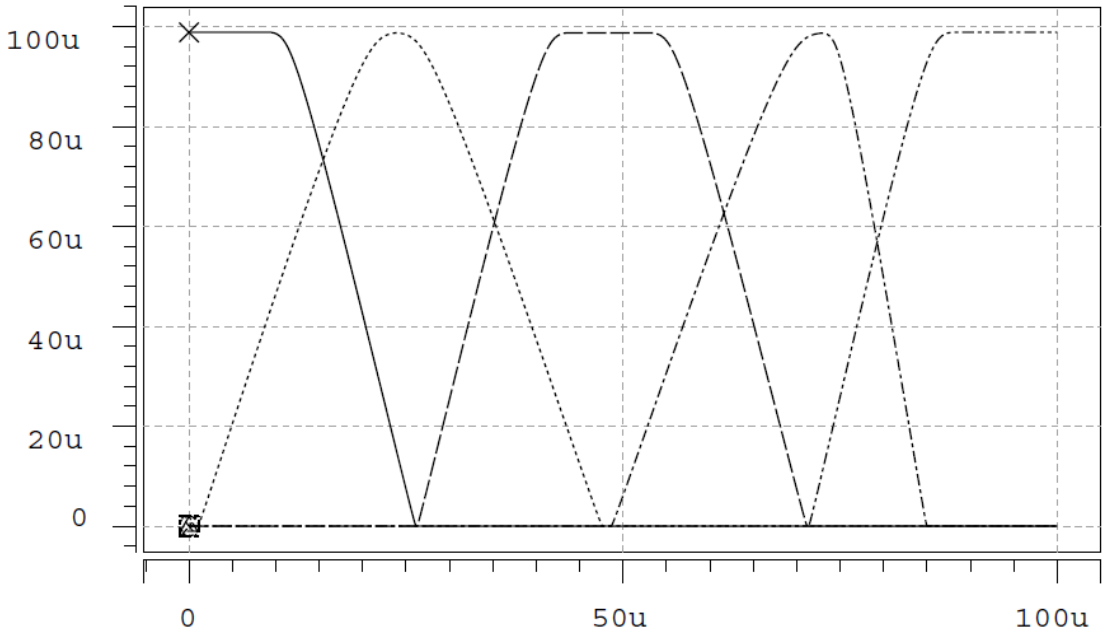
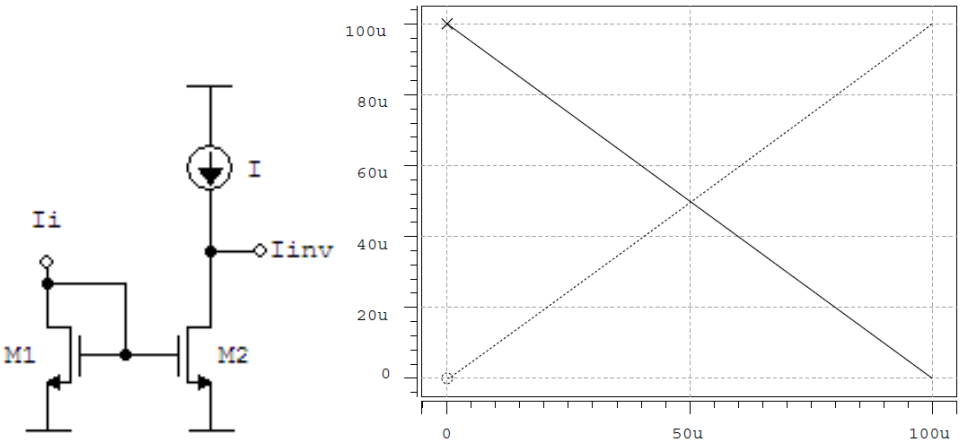


Figura 3.6 Conjunto de funciones de membresía en modo corriente.

3.2.2 Inferencia.

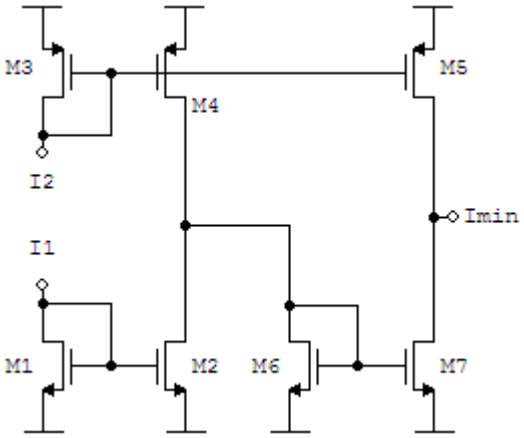
Para realizar la inferencia se podrán utilizar los tres operadores lógicos básicos de los sistemas de inferencia difusos: inversión, mínimo y máximo. En el caso de la operación de inversión solo se requiere restar la señal a un valor máximo *I* como se muestra en la Figura 3.7a. En la Figura 3.7b se muestra la simulación de este circuito.



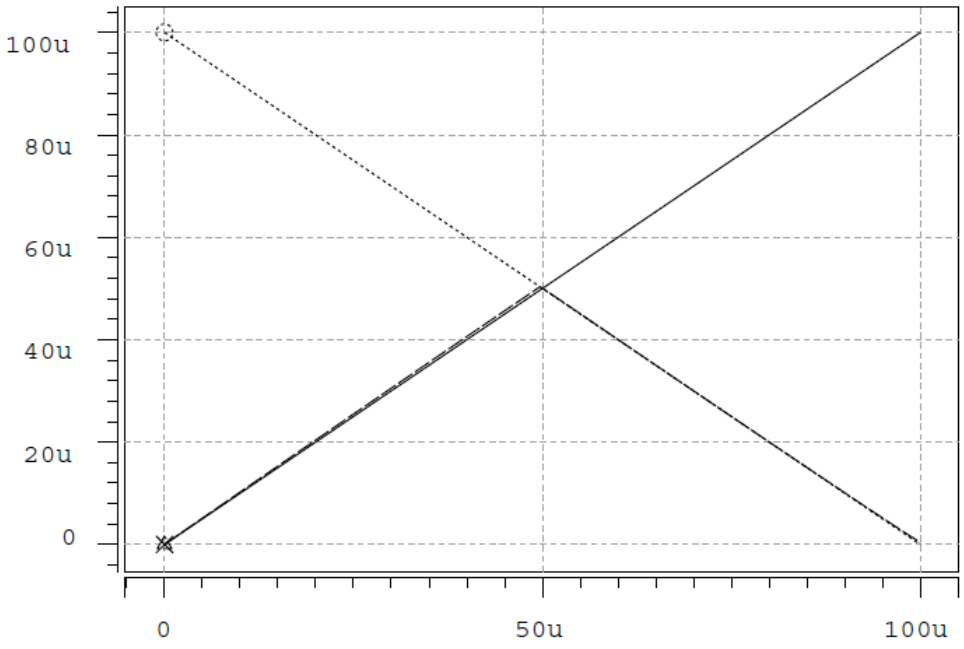
(a) (b)

Figura 3.7 a) Circuito Inversor y su b) simulación.

En la Figura 3.8a se muestra el circuito para realizar la operación mínimo utilizando solo espejos de corriente [64]. En la Figura 3.8b se muestra la simulación del circuito para el operador mínimo. En la Figura 3.9a se muestra el circuito para realizar la operación máximo utilizando solo espejos de corriente [64]. En la Figura 3.9b se muestra la simulación del circuito para el operador máximo.

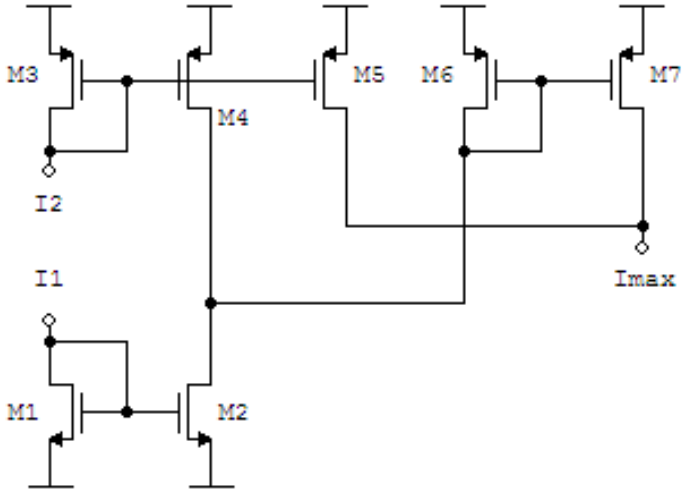


(a)

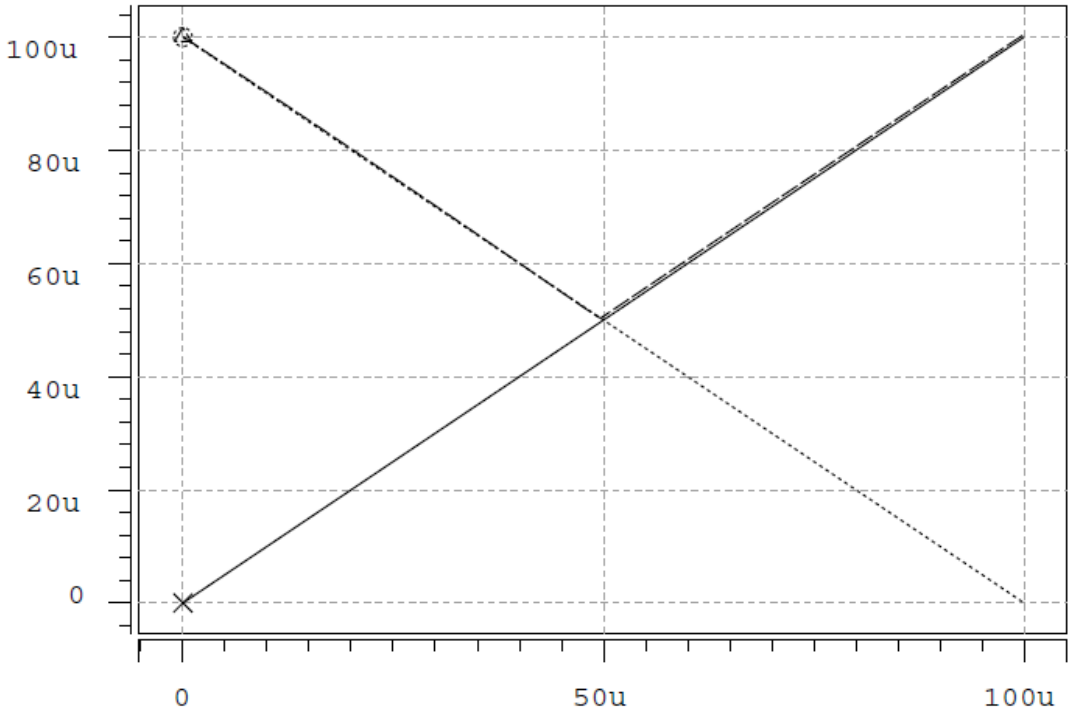


(b)

Figura 3.8 a) Circuito MIN y su b) simulación.



(a)



(b)

Figura 3.9 a) Circuito MAX y su b) simulación.

3.2.3 Desdifución.

Como se ha visto anteriormente, existen diversas formas de realizar el proceso de desdifución. En el presente trabajo se optará por desarrollar sistemas tipo Sugeno. El método de desdifución a utilizar será el de sumas ponderadas. También se ha visto que este tipo de desdifución implica el uso de la división matemática.

En la Figura 3.10a se muestra el esquema convencional de este tipo de desdifución. Se requiere que los consecuentes $w_1,w_2,\ldots,w_n$ sean pesados (multiplicados) por el singletons correspondientes $y_1,y_2,\ldots,y_n$ y agregados (sumados) posteriormente. Más adelante, esta agregación debe dividirse entre la suma de todos los consecuentes y ponderar el resultado. No obstante, también se ha visto que el uso de la división tiene sus inconvenientes. Es por esto que se utilizará un esquema alterno que proporciona el mismo resultado.

En la Figura 3.10b se muestra un esquema de desdifución que solo utiliza sumas, multiplicaciones y la normalización propuesto en [65]. La ventaja de este esquema radica en que existe un circuito capaz de realizar la normalización sin hacer un uso explicito de la división, donde obviamente son eliminados los inconvenientes inherentes al bloque divisor. Ya se ha visto que la suma y resta en modo corriente es muy simple de implementar, solo se requiere unir las fuentes en un nodo con la dirección de flujo apropiada.

Anteriormente se hizo mención del circuito multiplicador como parte del circuito para realizar funciones de membresía. En la Figura 3.11a se muestra el circuito propuesto en [66] para realizar la multiplicación de dos corrientes. En la Figura 3.11b se muestra la simulación del circuito multiplicador. De igual forma que todos los circuitos anteriores, el bloque básico de diseño es el espejo de corriente. El circuito encargado de normalizar las señales se basa en el circuito normalizador de Gilbert [67]. En la Figura 3.12a se muestra el circuito normalizador y en la Figura 3.12b la simulación del mismo.

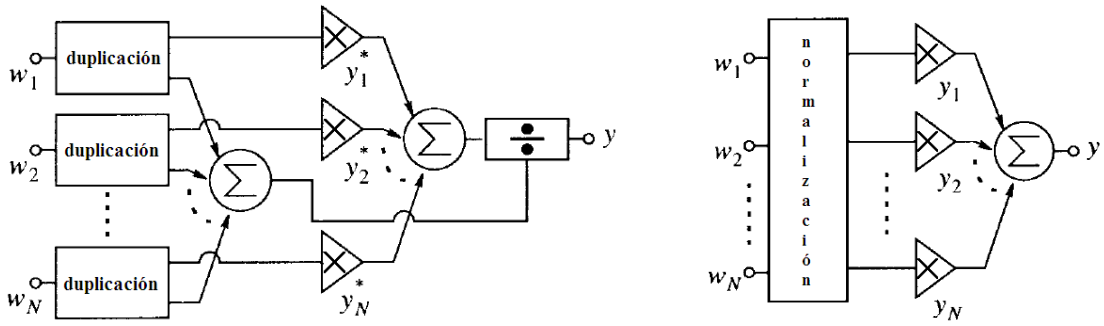


Figura 3.10 Esquema de desdifución a) multiplicación-suma-división, b) normalización-multiplicación-suma.

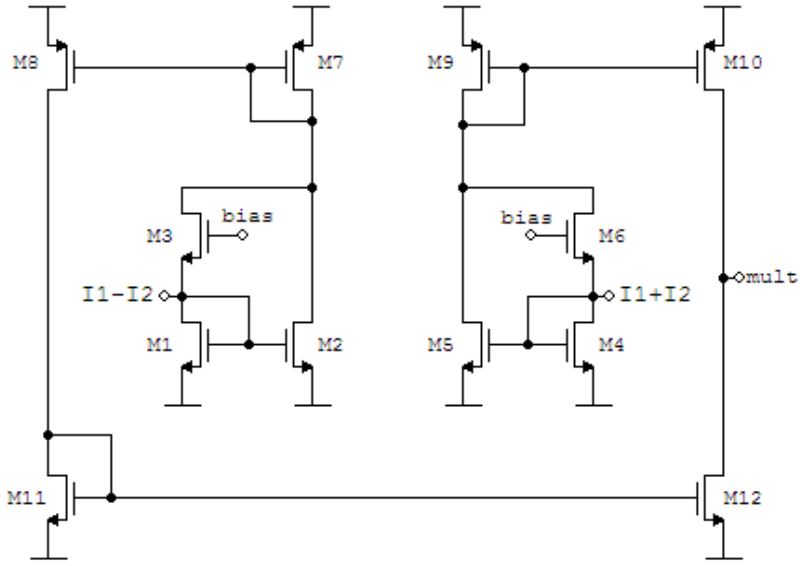


Figura 3.11 a) Circuito multiplicador

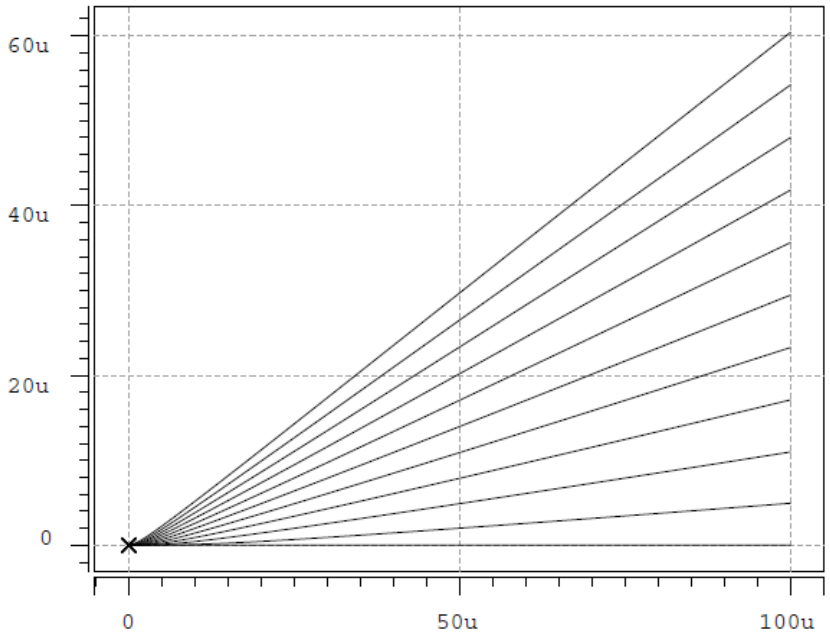


Figura 3.11 b) Respuesta del circuito multiplicador.

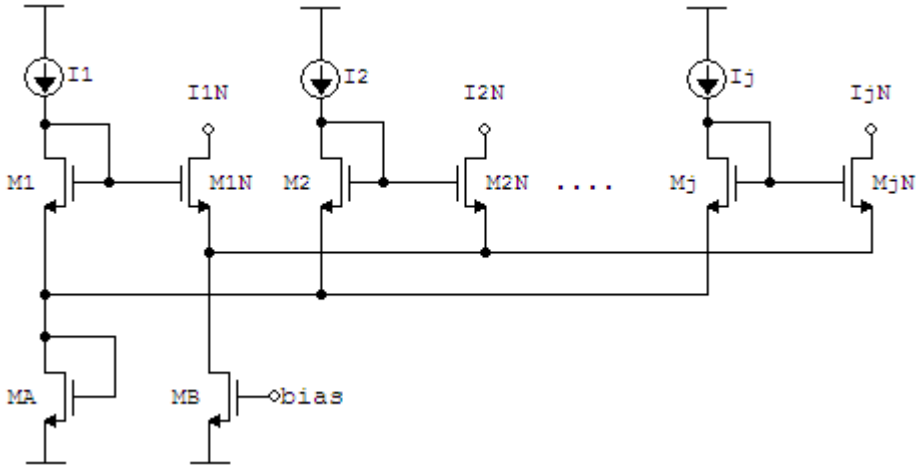


Figura 3.12 a) Circuito normalizador.

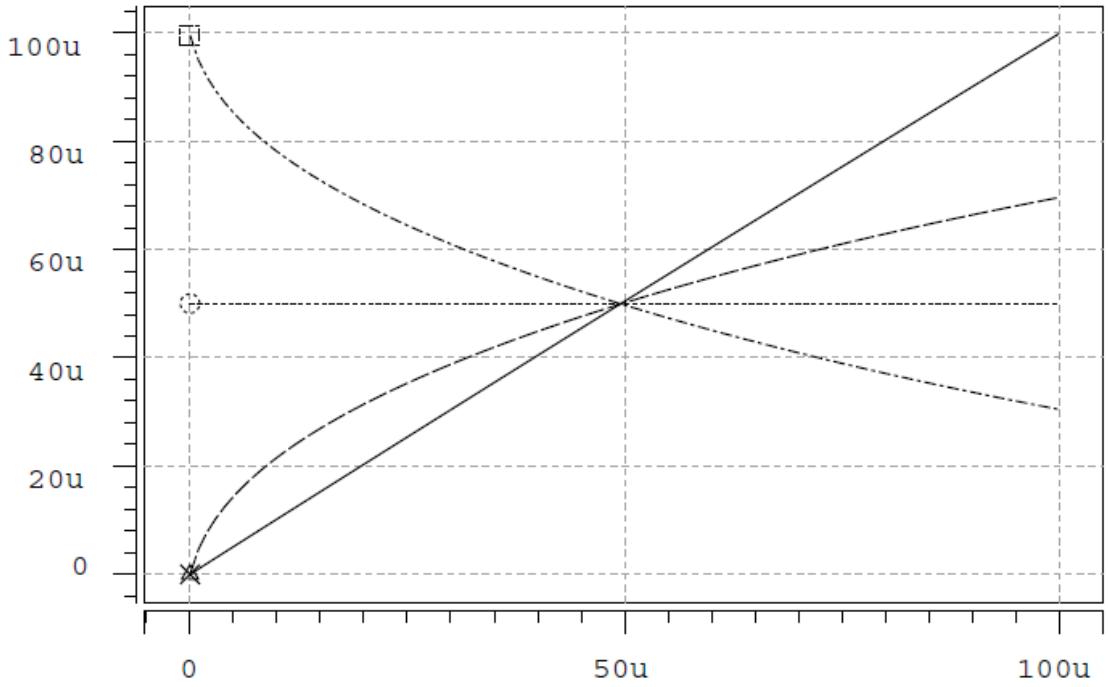


Figura 3.12 b) Respuesta del circuito normalizador.

Hasta aquí se han presentado los bloques analógicos para la implementación de un sistema de inferencia difuso. Debe resaltarse la simplicidad de diseño de estos sistemas en modo corriente al utilizar como bloque principal al espejo de corriente. También debe resaltarse que todos los bloques pueden ser programados en tiempo continuo.

3.3 Bloques funcionales evolutivos.

En capítulos anteriores se ha visto que los algoritmos evolutivos involucran procesos secuenciales. Esto implica que de alguna forma se tendrá que implementar una forma de control secuencial. Una forma simple de hacer esto es mediante el uso de una señal de reloj. Sin embargo, se busca que los bloques sean diseñados de tal forma que el uso de circuitos digitales sea minimizado. Además, se deben diseñar bloques que sean compatibles con los del sistema de inferencia difuso en modo corriente. Tomando en cuenta estas consideraciones, se optará por diseñar el esquema evolutivo haciendo uso de bloques que manipulen corrientes de forma conmutada.

3.3.1 Memoria.

Los elementos de memoria analógicos siempre se ven asociados con el uso de capacitores. El capacitor almacena energía o cargas eléctricas en forma de voltajes. Si se están utilizando corrientes es necesario convertirlas a voltajes. En la Figura 3.13 se muestra un circuito en modo corriente para el almacenamiento de datos analógicos. En este circuito la corriente entrante es convertida en voltaje por la impedancia vista a través de los transistores M1 y M3. Cuando la señal de reloj activa las compuertas de transmisión el voltaje generado es almacenado en los capacitores. Cuando la señal de reloj deshabilita las compuertas de transmisión los transistores M2 y M4 continúan convirtiendo en corriente el último voltaje almacenado en los capacitores. De forma similar se operan los transistores tipo P.

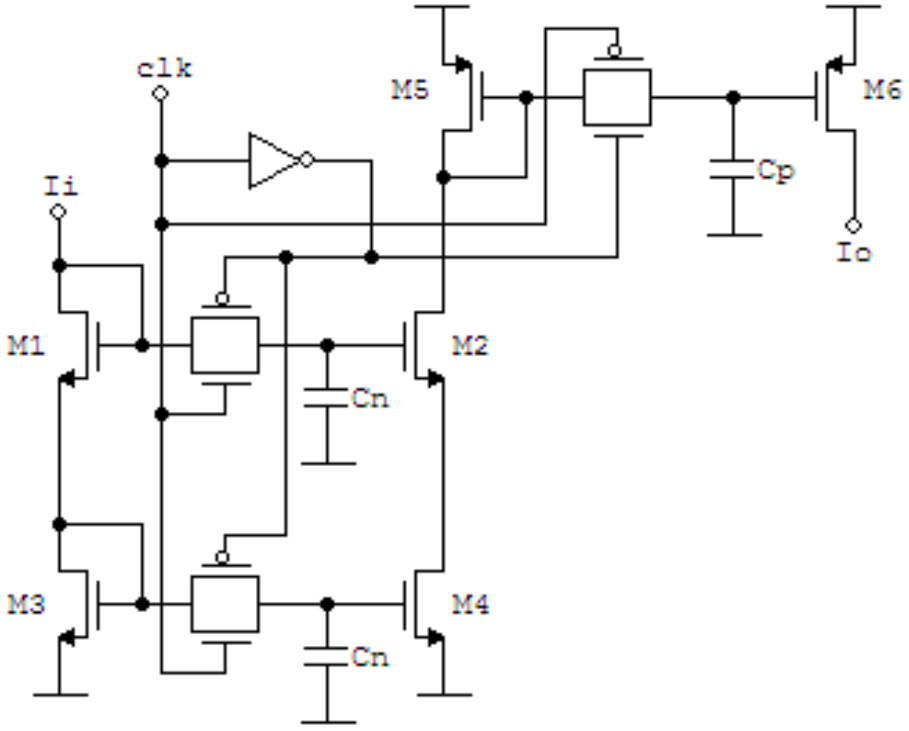
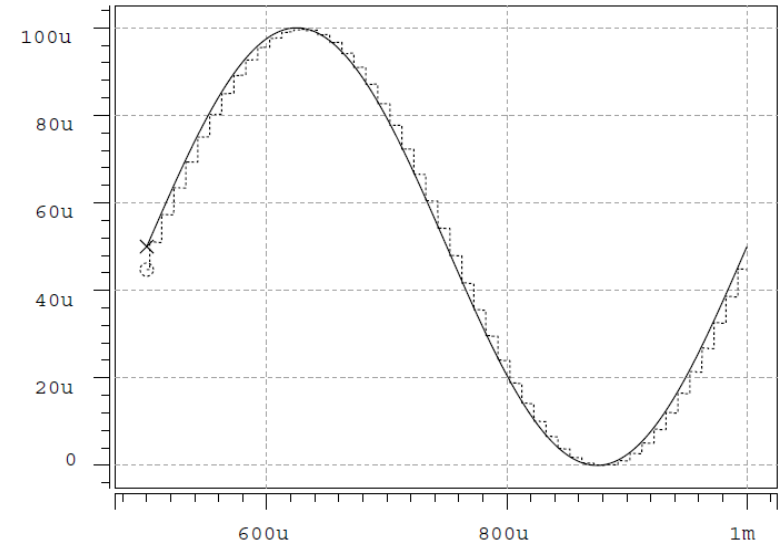
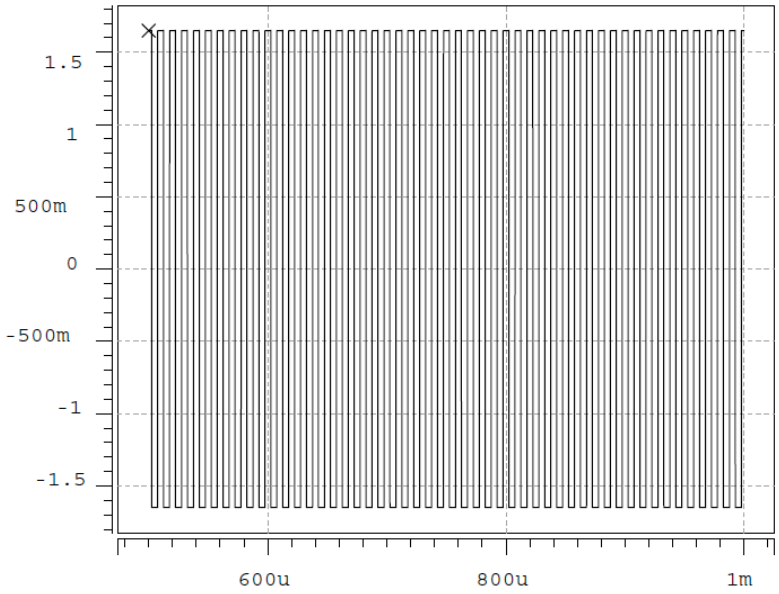


Figura 3.13 Memoria analógica en modo corriente.

En la Figura 3.14 se muestra la simulación de la memoria analógica en modo corriente. Como se puede ver, en cada ciclo de reloj es retenido el valor muestreado en la entrada del circuito.



(a)



(b)

Figura 3.14 Simulación de la memoria analógica en modo corriente: a) señal de entrada y salida del circuito y b) señal de control (reloj).

3.3.2 Evaluación de aptitud.

Anteriormente se mencionó que es necesario evaluar la aptitud o el desempeño del sistema que se este optimizando. Una forma simple de evaluar el desempeño de un sistema electrónico es mediante la medida del error absoluto generado por dicho sistema. Es decir, teniendo una referencia (pudiendo ser otro sistema o un simple valor electrónico), se debe encontrar la diferencia entre este valor y la salida del sistema que se este optimizando. En este caso se deberán calcular los errores de un sistema padre e_p y un sistema hijo e_h. Los valores generados de una resta pueden ser valores positivos o negativos. Si estos errores son comparados directamente, y considerando e_p = -0.5 y e_h = 0.1, el error con menor valor numérico es e_p, sin embargo e_p no es el que menos se desvía de la referencia. Es por esto que se requiere que la comparación de los errores sea de acuerdo a sus valores absolutos y así determinar cual error se desvía lo menos posible de la referencia. Bajo estas condiciones será necesario implementar circuitos que calculen el valor absoluto de una corriente así como un circuito comparador. El encontrar el valor absoluto de una señal eléctrica se puede ver como un simple proceso de rectificación de la señal. Es decir, independientemente de la polaridad de la señal que entre al circuito rectificador, esta saldrá con una misma polaridad conservando la magnitud con la que entro. Para esto solo se requiere de espejos de corriente y tener duplicada la señal de error. En la Figura 3.15a se muestra el circuito rectificador para el cálculo del valor absoluto de una corriente y en la Figura 3.15b se muestra la simulación de dicho circuito.

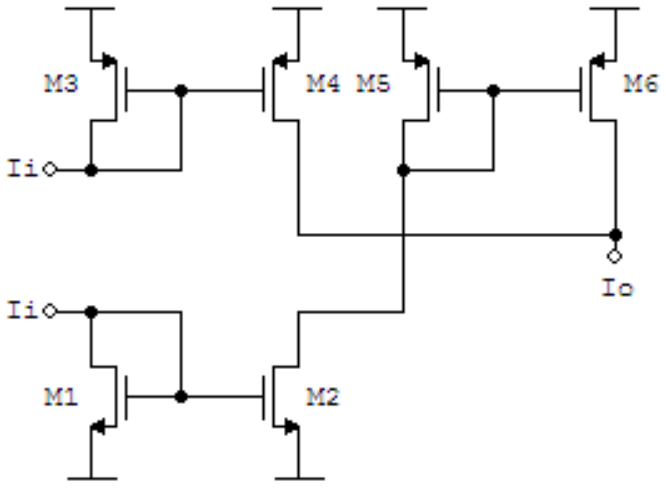


Figura 3.15 a) Circuito para el cálculo del valor absoluto de una corriente.

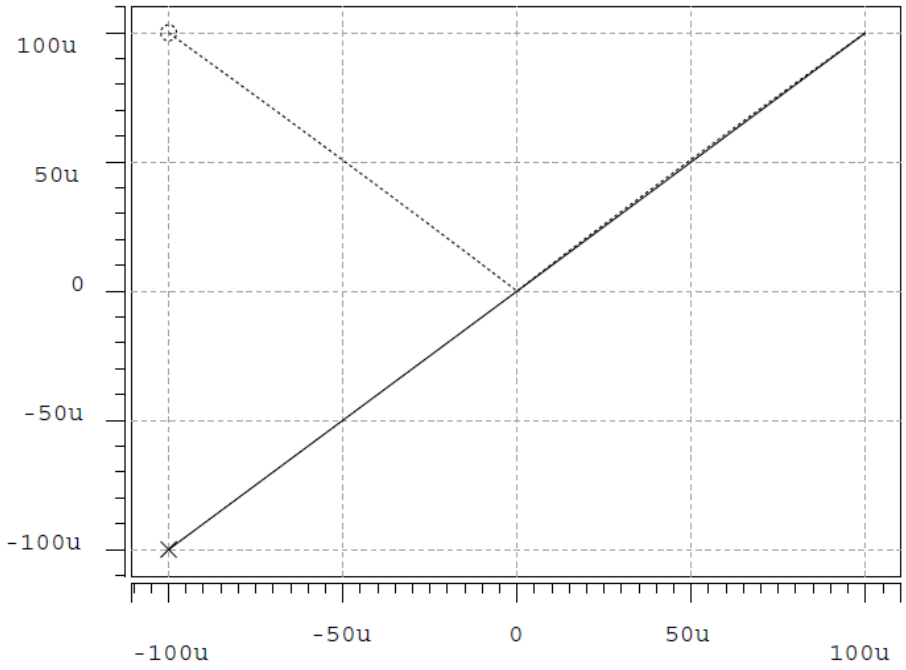


Figura 3.15 b) Respuesta del circuito para el cálculo del valor absoluto de una corriente.

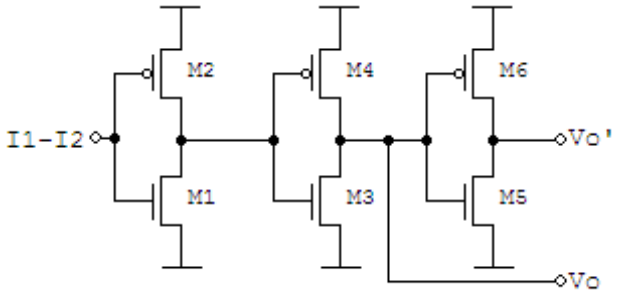


Figura 3.16 a) Circuito comparador.

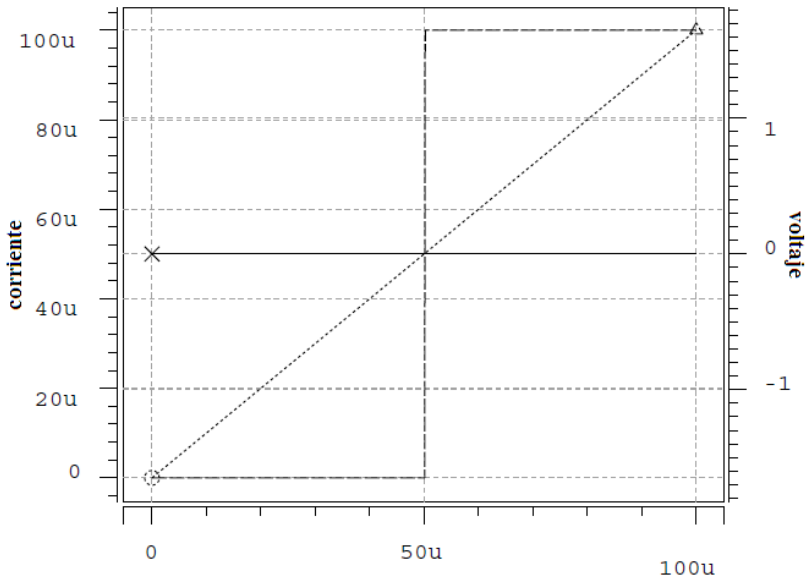


Figura 3.16 b) Respuesta del circuito comparador.

Para realizar la comparación de dos corrientes se utilizará el circuito de la Figura 3.16a. Como se puede ver solo se utilizarán inversores digitales que pueden ser vistos como amplificadores de señal. Al encontrarse o restarse dos corrientes en la entrada del comparador, estas almacenarán cargas en la capacitancia de entrada del circuito. Si una de las dos corrientes es mayor que la otra, se generará un voltaje en la entrada que tenderá a alcanzar el valor de una de las fuentes de polarización. Como consecuencia el comparador conmutará su salida digital de un 1 a 0 lógico o viceversa cuando se detecte una diferencia entre las corrientes de entrada. En la Figura 3.16b se muestra la simulación del comparador.

3.3.3 Generación de soluciones.

En los algoritmos evolutivos es de vital importancia tener una fuente confiable de números aleatorios, ya que de esta depende el éxito de encontrar la solución del sistema a optimizar. Una primera aproximación analógica para la generación de números aleatorios es el uso de sistemas caóticos. Ya se ha visto que los mapas logísticos ofrecen una solución práctica en la implementación de osciladores caóticos. Estos osciladores requieren que el mapa logístico sea iterado o retroalimentado, donde dicha iteración puede ser sincronizada por un pulso de reloj. Esto permitirá la fácil adaptación del resto del sistema considerando que el esquema evolutivo deber ser controlado de forma secuencial o iterativa también.

En la Figura 2.9 se muestra el esquema general para la iteración de mapas logísticos y generar oscilaciones caóticas. Si se observa, el sistema requiere de un elemento de memoria (conformado por interruptores y capacitores) controlado por una señal de reloj. Este elemento de memoria será implementado por el circuito de la Figura 3.13. Ahora solo se requiere implementar un circuito en modo corriente que genere un mapa logístico.

El circuito propuesto en [68] y que se muestra en la Figura 3.17 es diseñado bajo el principio translineal cuadrático teniendo la siguiente función de transferencia:

$$I_o' = 2I_b + \frac{I_i^2}{8I_b} \tag{3.1}$$

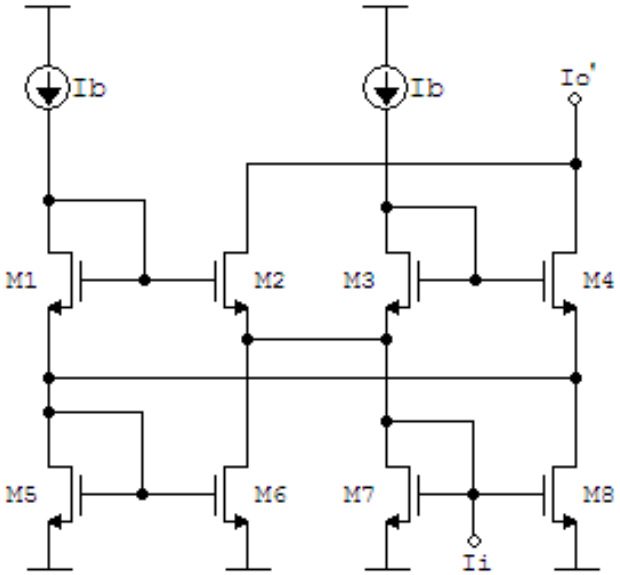


Figura 3.17 Circuito translineal cuadrático.

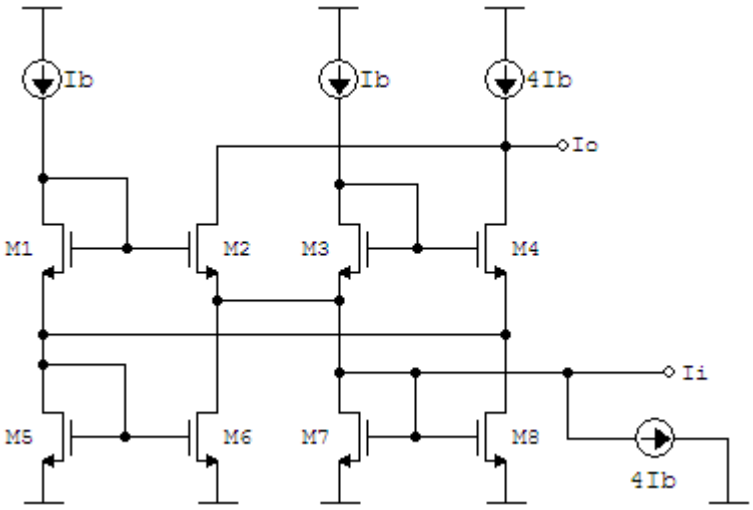


Figura 3.18 Circuito translineal para la implementación del mapa logístico.

Al modificar esta función de transferencia agregando una corriente de offset en la salida y la entrada del circuito como se muestra en la Figura 3.18. Agregando estas corrientes de offset en la ecuación 3.1 y aplicando algebra:

$$I_o = 4I_b - I_o' = 4I_b - \left(2I_b + \frac{(I_i - 4I_b)^2}{8I_b} \right) \tag{3.2}$$

$$I_o = 2I_b - \frac{(I_i - 4I_b)^2}{8I_b} \tag{3.3}$$

$$I_o = 2I_b - \frac{I_i^2}{8I_b} + \frac{8I_iI_b}{8I_b} - \frac{16I_b^2}{8I_b} \tag{3.4}$$

$$I_o = Ii - \frac{I_i^2}{8I_b} \tag{3.5}$$

$$I_o = \frac{1}{8I_b}\left(8I_iI_b - I_i^2\right) \tag{3.6}$$

$$I_o = \frac{1}{8I_b} I_i \left(8I_b - I_i\right) \tag{3.7}$$

De la ecuación 3.7 se puede observar su similitud con la ecuación 1.23 del mapa logístico analizado en el primer capitulo de este trabajo. Si se considera que $8I_b$ = 1, se tendría la misma función sin contar con el parámetro de control μ. De hecho lo que se tiene es un circuito que representa el mismo mapa logístico, pero con un factor de escalamiento igual a $8I_b$. En la Figura 3.19 se muestra la simulación del circuito generador del mapa logístico.

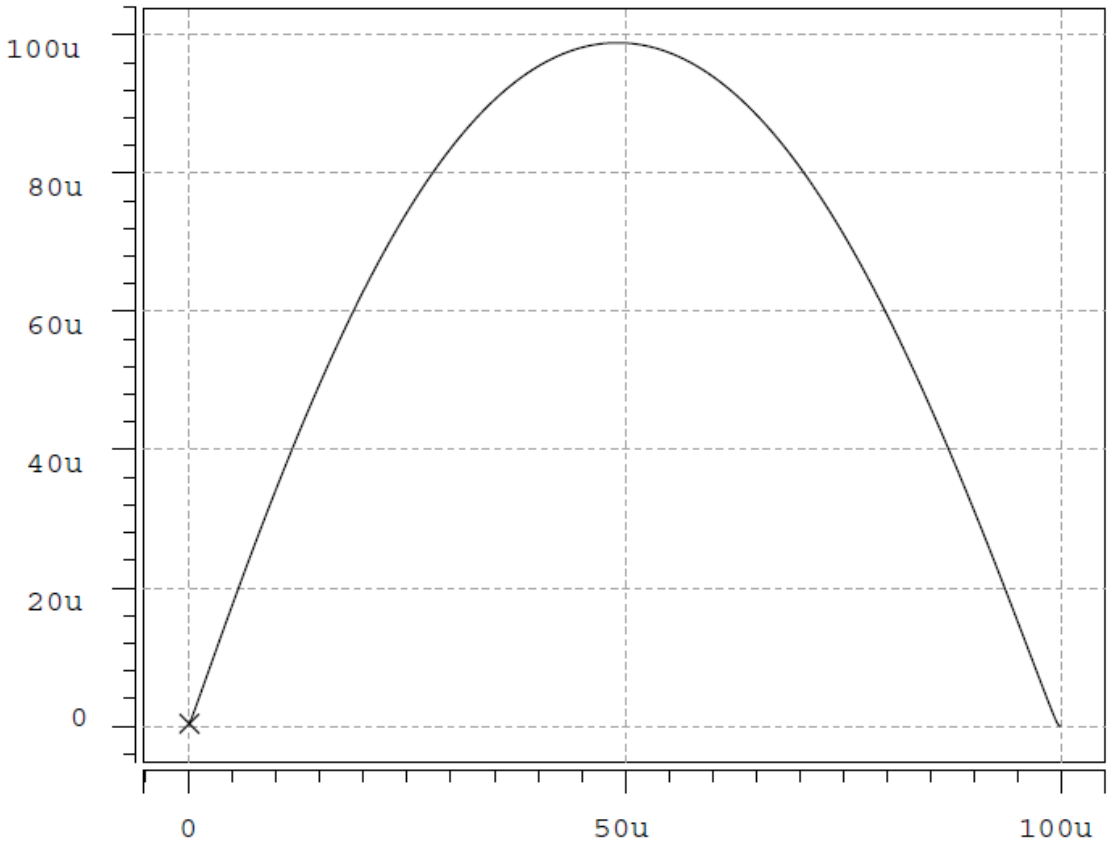


Figura 3.17 Simulación del circuito generador del mapa logístico.

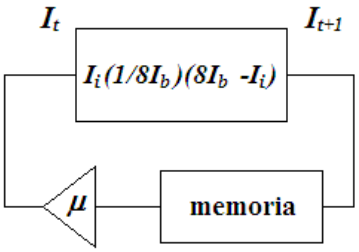


Figura 3.18 Configuración del circuito caótico.

Ahora se tienen los bloques necesarios para construir un oscilador caótico. En la Figura 3.18 se muestra el esquema que se debe utilizar para conformar el circuito caótico. La forma de controlar las oscilaciones del circuito se realiza con el ajuste del parámetro μ, que puede ser considerado como un bloque de ganancia. En este caso se puede aprovechar la estructura del circuito memoria de la Figura 3.13. Considerando que esta conformado por espejos de corriente, se pueden ajustar las dimensiones de los transistores M2 y M4 o M6 y controlar la ganancia de los espejos. El circuito completo del oscilador caótico se muestra en la Figura 3.19. La simulación transitoria del oscilador caótico y el espectro en frecuencia de la señal generada son mostrados en las Figuras 3.20a y 3.20b respectivamente. De estas gráficas se puede observar que el circuito es capaz de generar señales del tipo aleatorias. El espectro obtenido es similar al de señales ruidosas, señales que muchas veces son consideradas como fuentes de secuencias aleatorias.

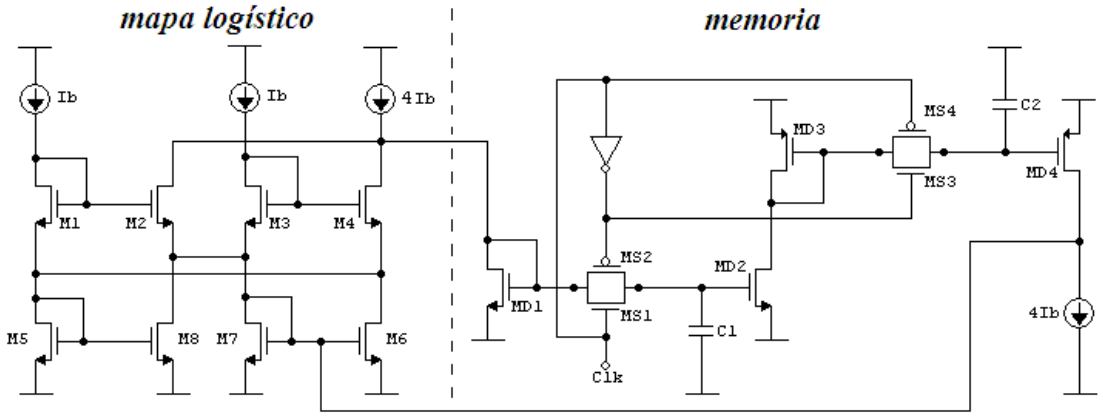
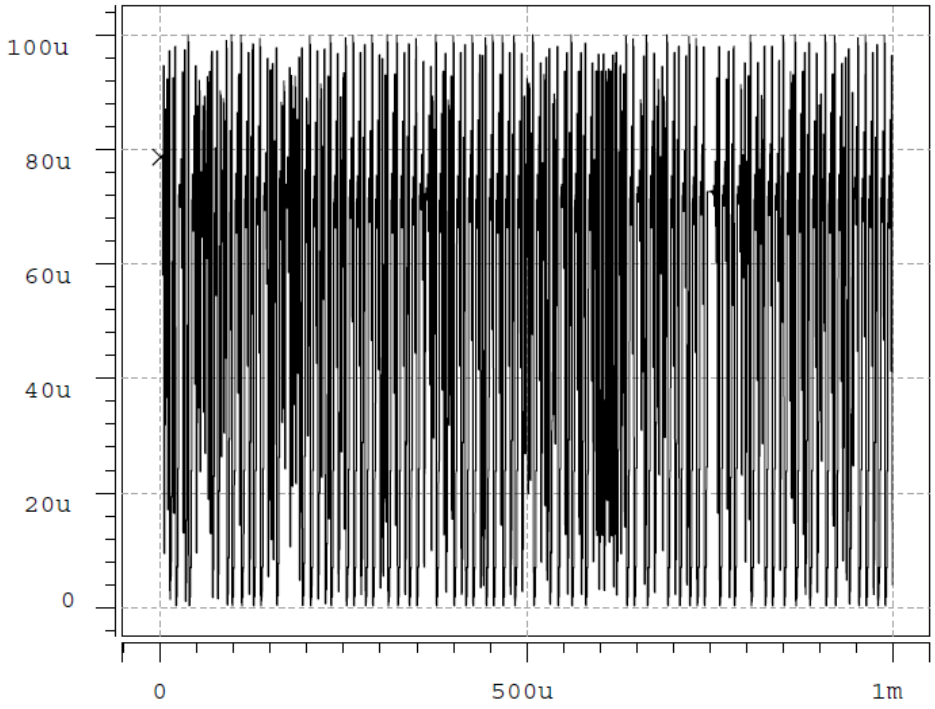
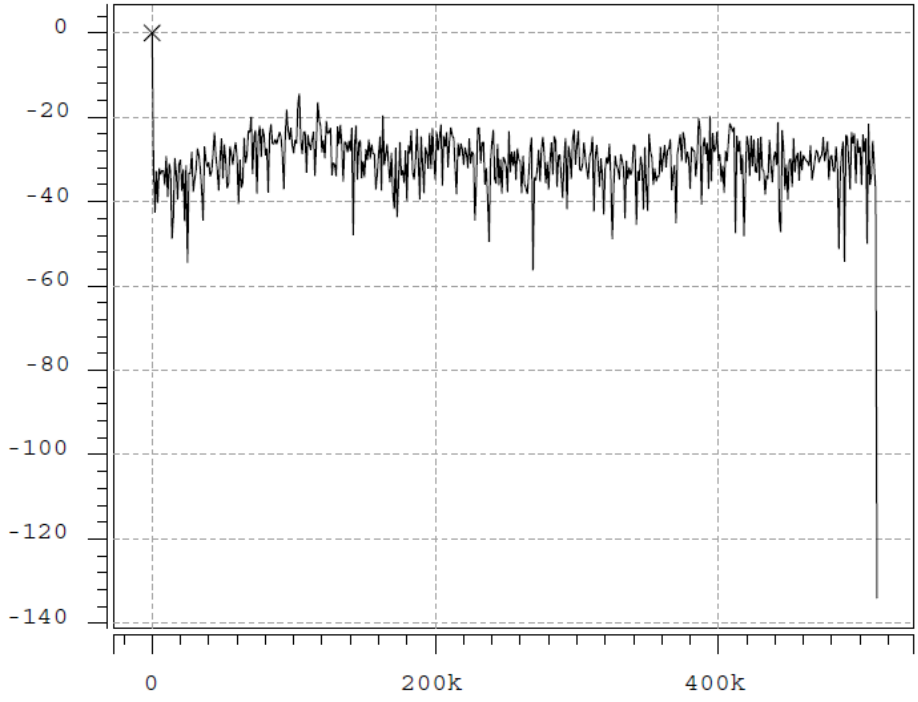


Figura 3.19 Oscilador caótico en modo corriente.



(a)



(b)

Figura 3.20 Simulación del oscilador caótico, a) transitorio, b) espectro en frecuencia.

Para corroborar el correcto funcionamiento de los osciladores caóticos se realizó su fabricación en una tecnología de 0.8μm de AustriaMicrosystems. En la Figura 3.21 se muestra la instrumentación requerida en la prueba del circuito integrado fabricado. En la Figura 3.22 se muestra la señal aleatoria obtenida de forma experimental. En esta figura se puede observar que la señal obtenida no tiene una secuencia definida. Así en la Figura 3.23 se muestra el espectro en frecuencia obtenido de la señal caótica, comprobándose que el espectro es similar al de una señal ruidosa o que no tiene una secuencia determinada.

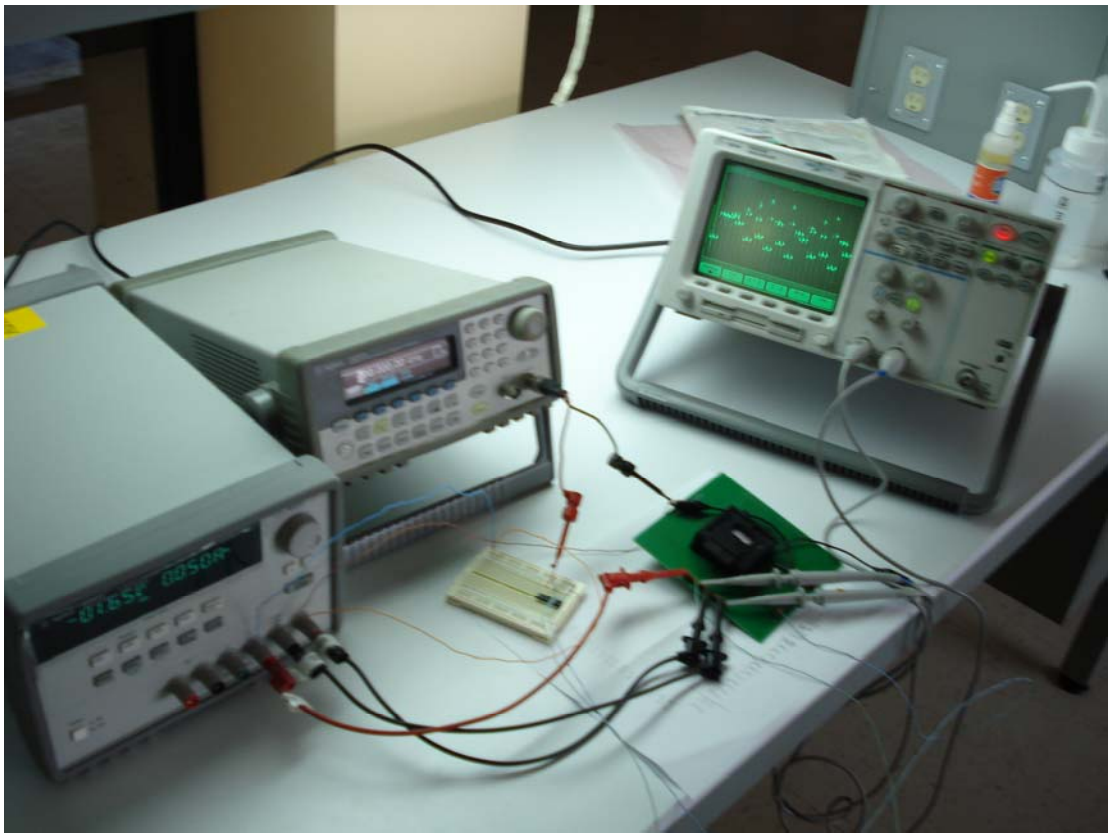


Figura 3.21 Instrumentación para la prueba del circuito integrado fabricado.

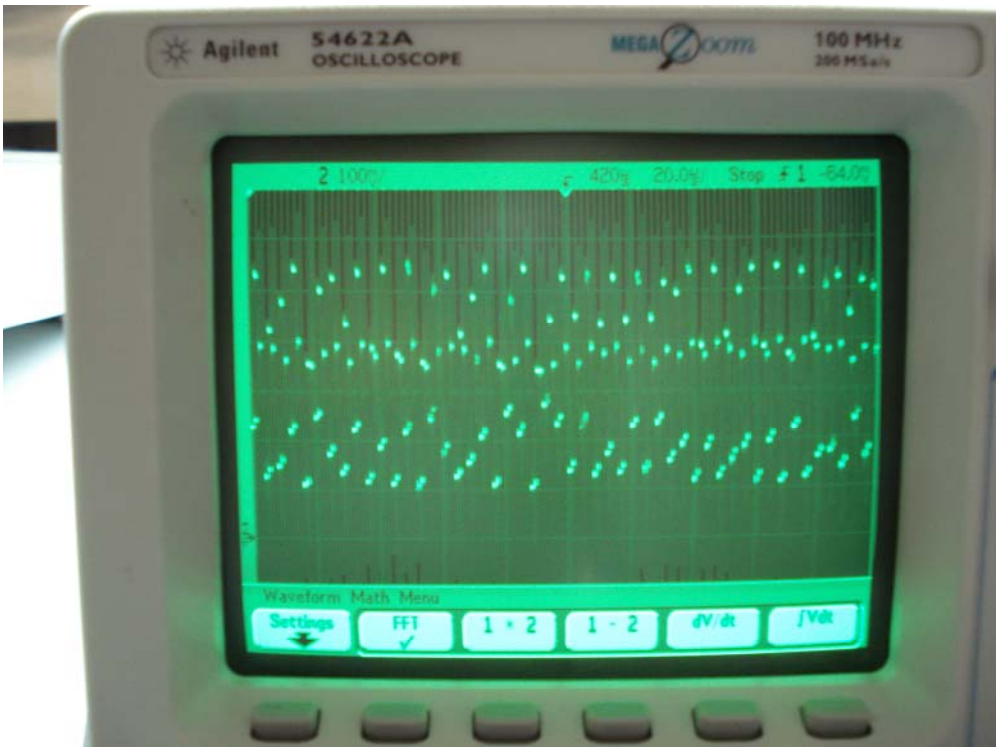


Figura 3.22 Señal caótica obtenida experimentalmente.

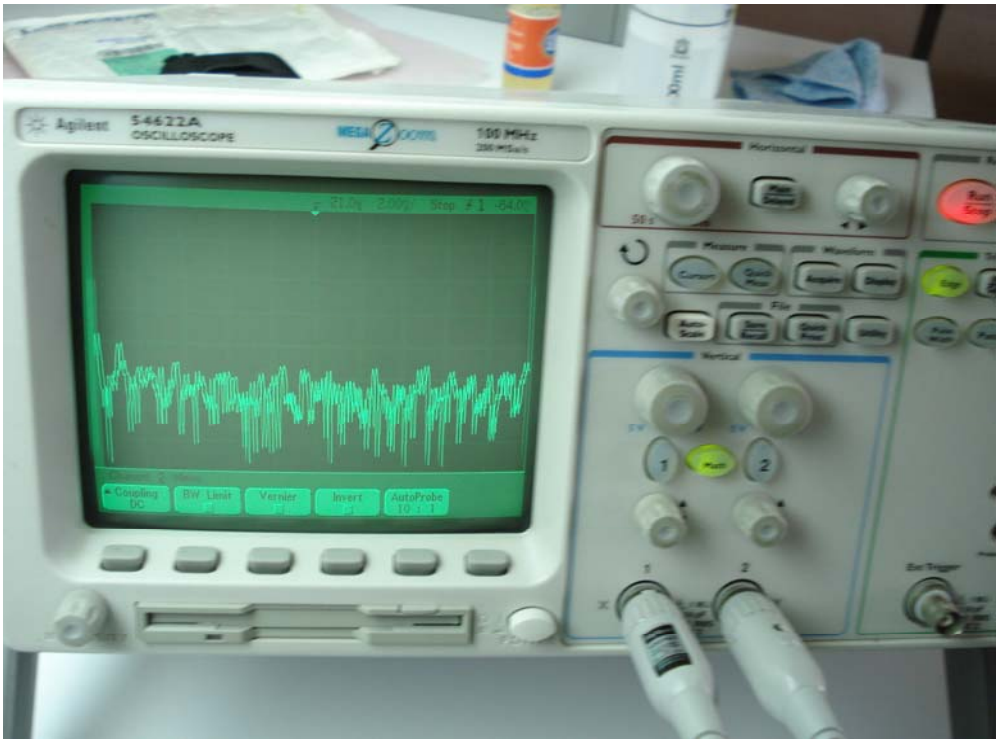


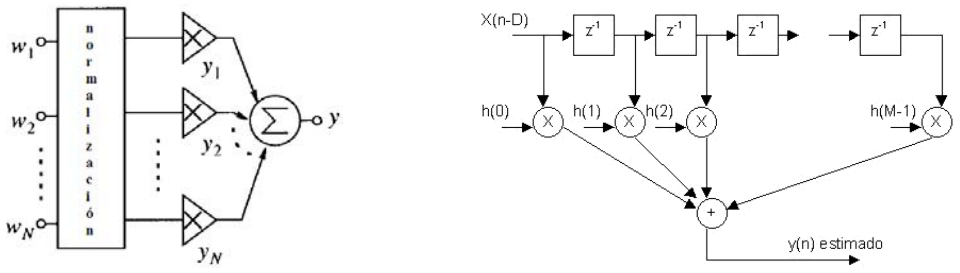
Figura 3.23 Espectro en frecuencia de la señal caótica.

Capítulo 4

Adaptación Evolutiva del Sistema Difuso VLSI

4.1 Adaptación Evolutiva de Sistemas Difusos.

La forma o modo en que la adaptación de sistemas difusos debe realizarse depende del tipo de tipo de sistema por adaptar. En sistemas VLSI son ampliamente utilizados los sistemas difusos de tipo Sugeno. Este tipo de sistemas al igual que los otros esta conformado por bloques que generan Funciones de Membresía, bloques para la Inferencia de las reglas y bloques de Defuzificación. La defuzificación en los sistemas Sugeno de orden cero consta de multiplicar los antecedentes por singletons (constantes) y a su vez agregarlos o acumularlos para obtener una salida. Este tipo de estructuras son iguales a las presentes en los filtros FIR donde las salidas de los retardos son equivalentes a las salidas de los antecedentes. En la Figura 4.1 se puede observar esta equivalencia. Esto implica que un simple algoritmo de adaptación LMS puede resolver determinar los valores necesarios de los singletons. No obstante, en un sistema difuso no solo los singletons son los parámetros que se pueden o deben adaptar. También puede ser necesario adaptar los parámetros de las funciones de membresía y/o la base de reglas. La adaptación de estos parámetros rebasa las capacidades de un algoritmo convencional como el LMS y es aquí donde se requiere de un esquema de adaptación que garantice la solución de una solución. En este capítulo se muestran los resultados obtenidos al adaptar sistemas difusos haciendo uso de una arquitectura evolutiva. Primero se observará el desempeño del esquema al adaptar sistemas de una sola entrada y una sola salida (SISO). Posteriormente se observará el desempeño en la adaptación de un sistema de múltiples entradas y una salida (MISO).



(a) (b)

Figura 4.1 Equivalencia entre la (a) defuzificación de un Sistema Difuso y (b) un Filtro FIR.

4.2 Adaptación Evolutiva de Sistemas Difusos SISO.

Los sistemas difusos de una sola entrada y una sola salida generalmente tienen la estructura que se muestra en la Figura 4.2. La señal es fuzificada en los circuitos que generan funciones de membresía (FM1 a FM3). Estas funciones de membresía son normalizadas y escaladas por los singletons (Sing1 a Sing3) para posteriormente ser sumadas y obtener una salida. En este caso la arquitectura evolutiva deberá minimizar el error de adaptación del sistema. En la Figura 4.3 se muestran las funciones de membresía que se utilizaran como referencia. Los valores de los singletons serán Sing1 = Sing3 = 0μA y Sing2 = 100μA, donde se tendrá una equivalencia de 100μA para el '1 lógico' y 0μA para el '0 lógico'. En la Figura 4.4 se muestra la respuesta esperada del Sistema Difuso.

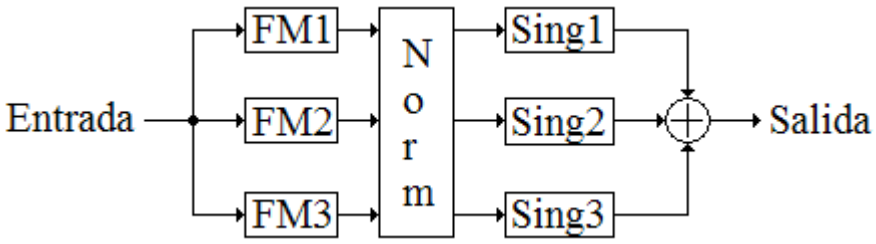


Figura 4.2 Sistema Difuso SISO.

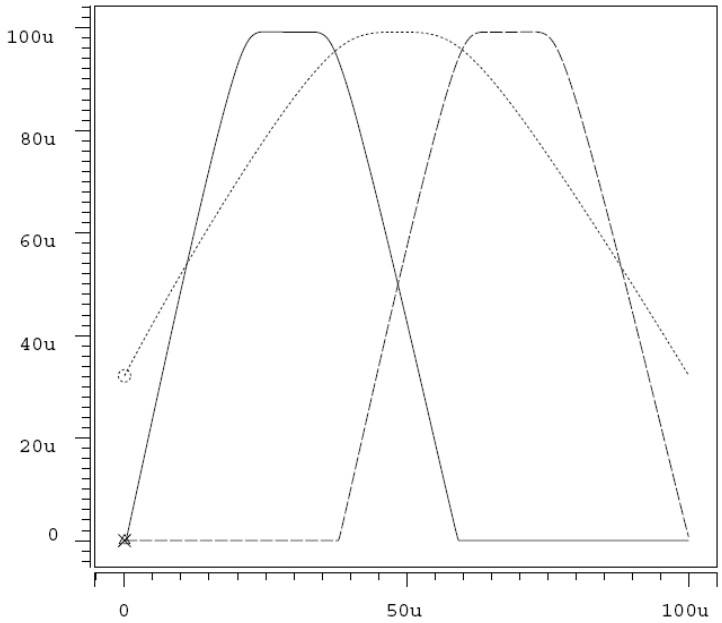


Figura 4.3 Funciones de membresía para el Sistema Difuso SISO.

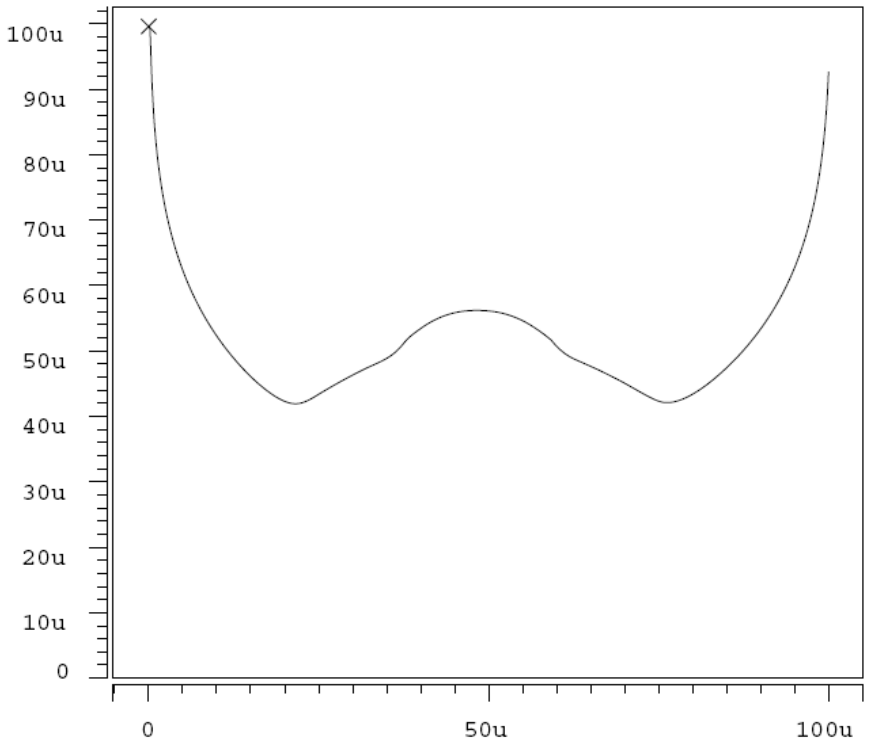


Figura 4.4 Función de transferencia del Sistema Difuso SISO.

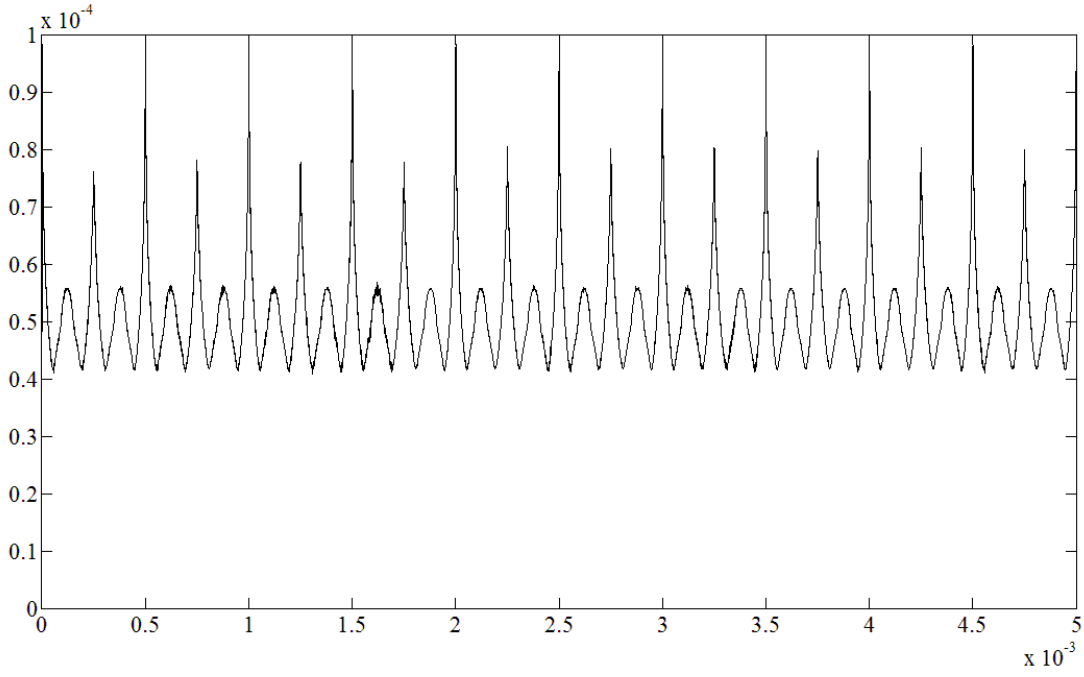


Figura 4.5 Respuesta transitoria del Sistema Difuso SISO de referencia.

La arquitectura evolutiva utilizará 15 osciladores caóticos para encontrar la mejor solución posible y adaptar los valores de los 4 parámetros de las funciones de membresía (puntos de quiebre y pendientes) y los 3 singletons. Se realizará una adaptación donde solo habrá 5,000 iteraciones. En las Figuras 4.5 y 4.6 se muestra la respuesta transitoria del sistema difuso utilizado como referencia y del sistema difuso adaptable respectivamente. En la Figura 4.7 se muestra el error cuadrático medio del sistema difuso adaptable padre durante el proceso de adaptación.

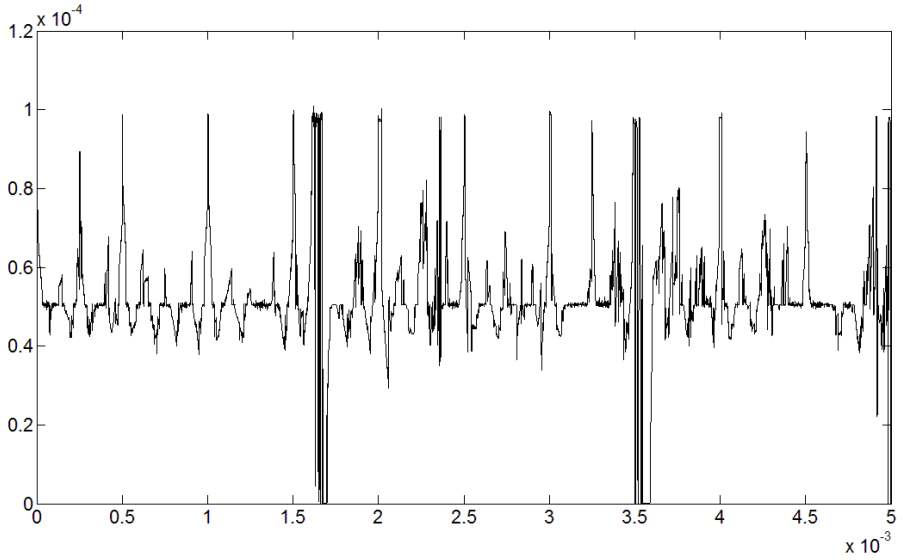


Figura 4.6 Respuesta transitoria del Sistema Difuso SISO adaptable.

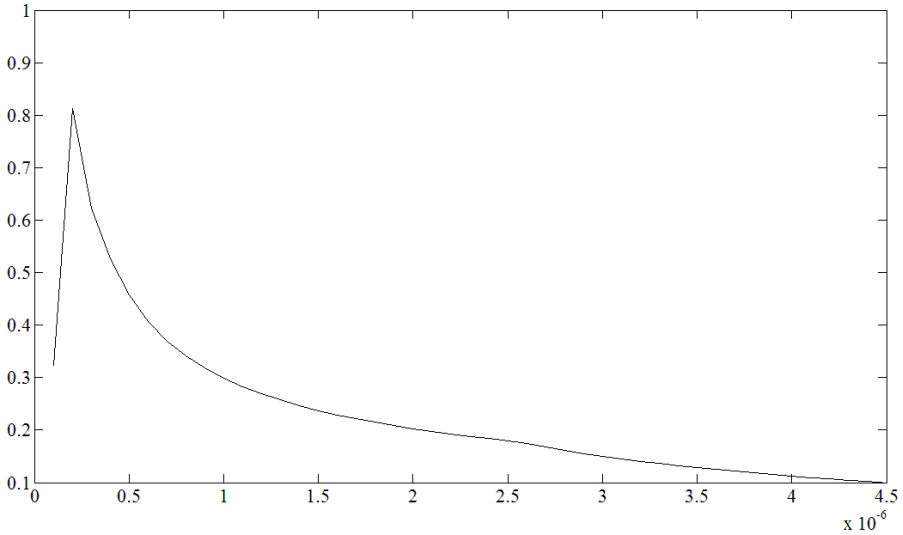


Figura 4.7 Error cuadrático medio del Sistema Difuso SISO adaptable padre.

En la Figura 4.6 se puede observar como el sistema de adaptación trata de seguir la respuesta del sistema de referencia. Debe observarse como el sistema adaptable tiene un comportamiento '*errático*' debido a la naturaleza del algoritmo. Es de esperarse que la respuesta del sistema de adaptación obedezca a la forma en que son generadas las soluciones a partir de un sistema caótico, donde las soluciones se buscan de forma heurística. Sin embargo, observando la Figura 4.7 se puede ver como realmente el sistema de adaptación tiende a encontrar una solución. El error de adaptación no llega a ser cero, propiedad característica de los sistemas basados en adaptaciones a través de estrategias evolutivas. En la Figura 4.8 se hace un acercamiento al comportamiento del error del sistema. En esta figura se puede observar como hay intervalos de tiempo en los que el error se incremente y después vuelve a decrecer teniendo una tendencia a minimizar el error.

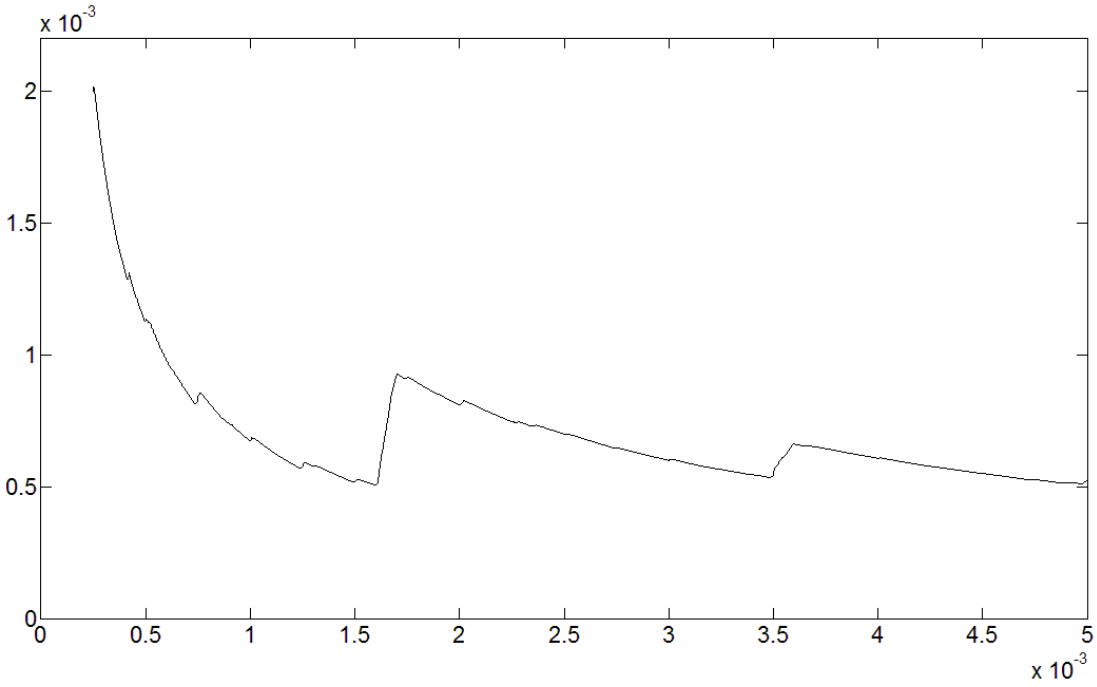


Figura 4.8 Acercamiento del error cuadrático medio del Sistema Difuso SISO adaptable padre.

4.3 Adaptación Evolutiva de Sistemas Difusos MISO.

Ahora se verá el comportamiento del sistema evolutivo al adaptar un Sistema Difuso de 2 entradas y una salida. En la Figura 4.9 se muestra la estructura del Sistema Difuso que servirá de referencia para la adaptación de otro sistema. De la misma forma, en la Figura 4.10 se muestra la superficie de dicho sistema.

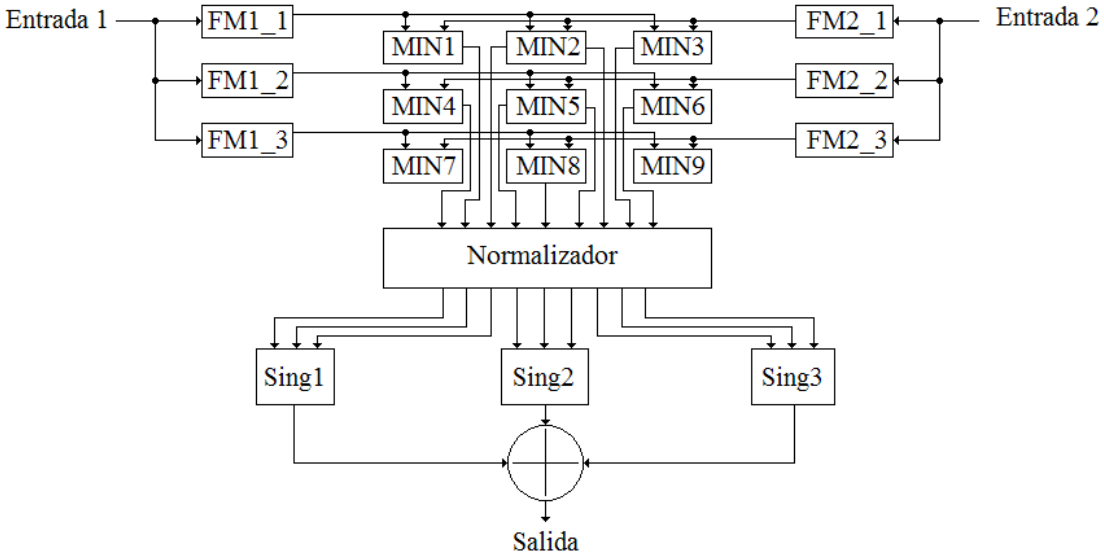


Fig. 4.9 Sistema Difuso MISO (2 entradas – 1 salida).

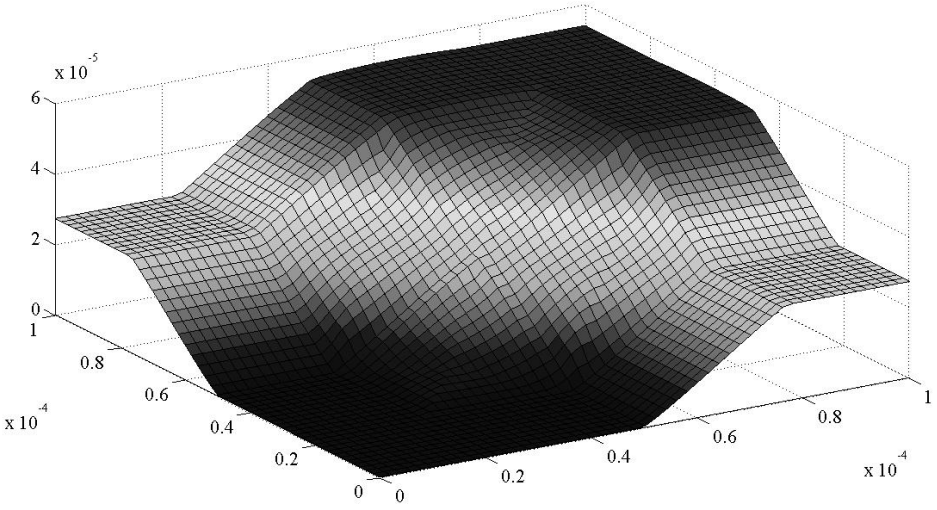


Fig. 4.10 Superficie de respuesta del Sistema Difuso MISO.

La arquitectura evolutiva utilizará 27 osciladores caóticos para encontrar la mejor solución posible y adaptar los valores de los 4 parámetros de las funciones de membresía (3 funciones por entrada) y los 3 singletons. Se realizará una adaptación donde solo habrá 5,000 iteraciones. En las Figuras 4.11 y 4.12 se muestra la respuesta transitoria del sistema difuso utilizado como referencia y del sistema difuso adaptable respectivamente. En la Figura 4.13 se muestra el error cuadrático medio del sistema difuso adaptable padre durante el proceso de adaptación.

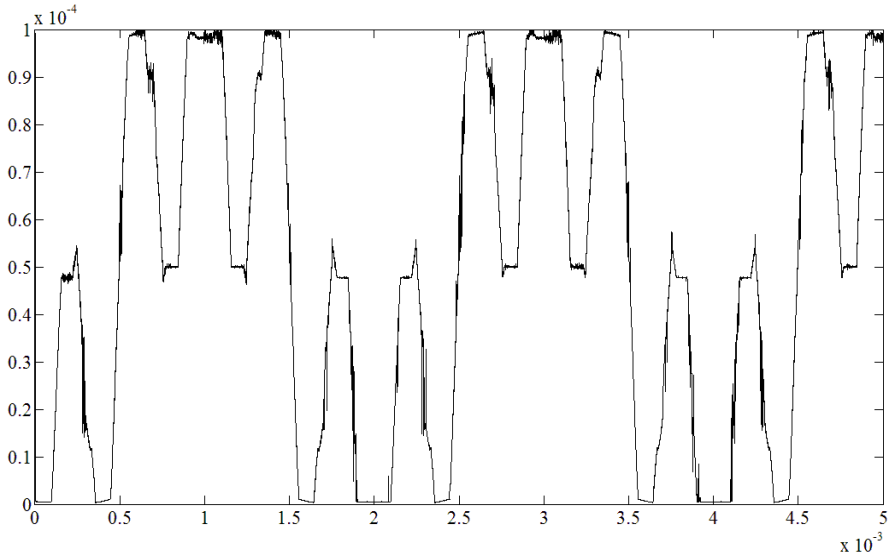


Figura 4.11 Respuesta transitoria del Sistema Difuso MISO de referencia.

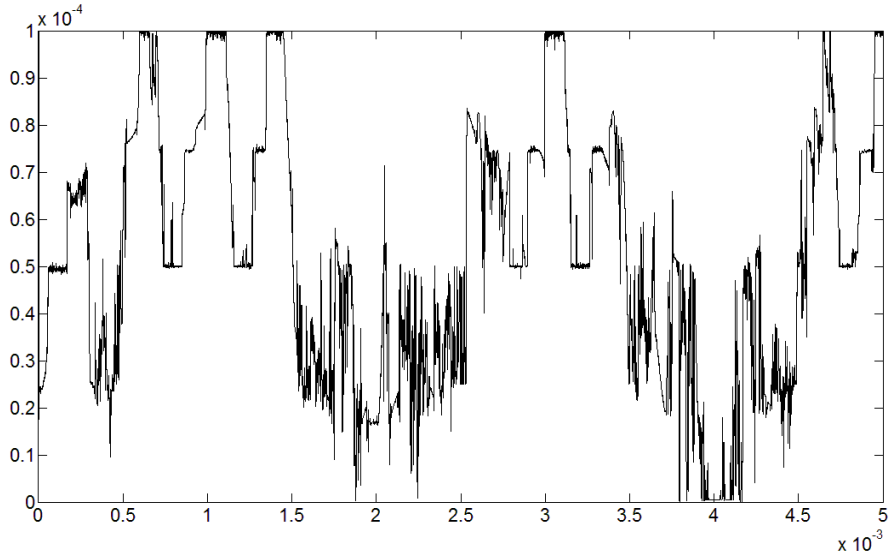


Figura 4.12 Respuesta transitoria del Sistema Difuso MISO adaptable.

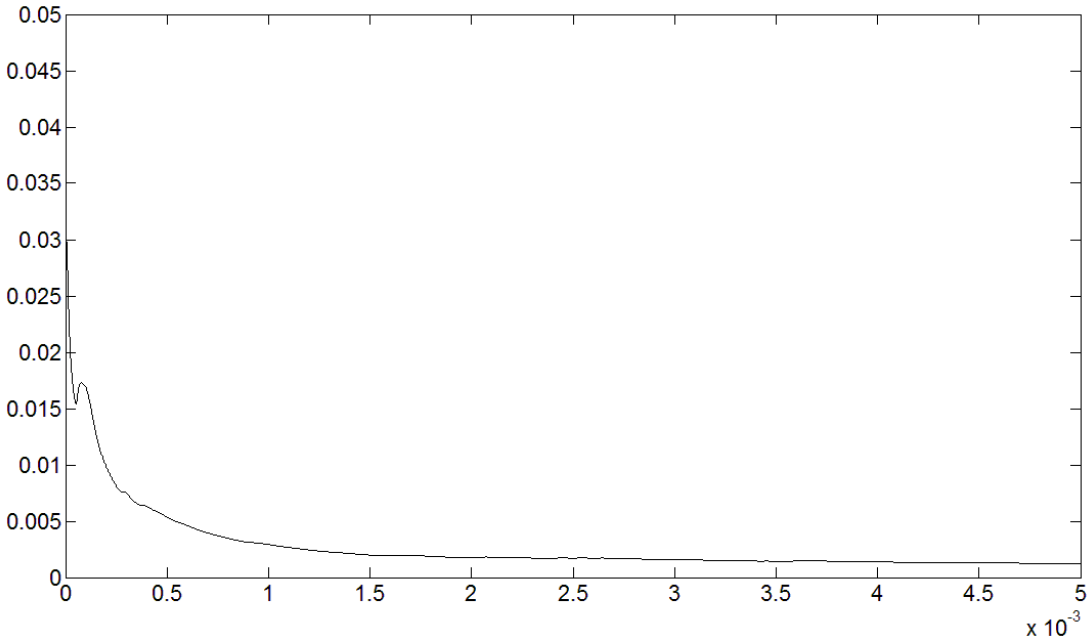


Figura 4.13 Error cuadrático medio del Sistema Difuso MISO adaptable padre.

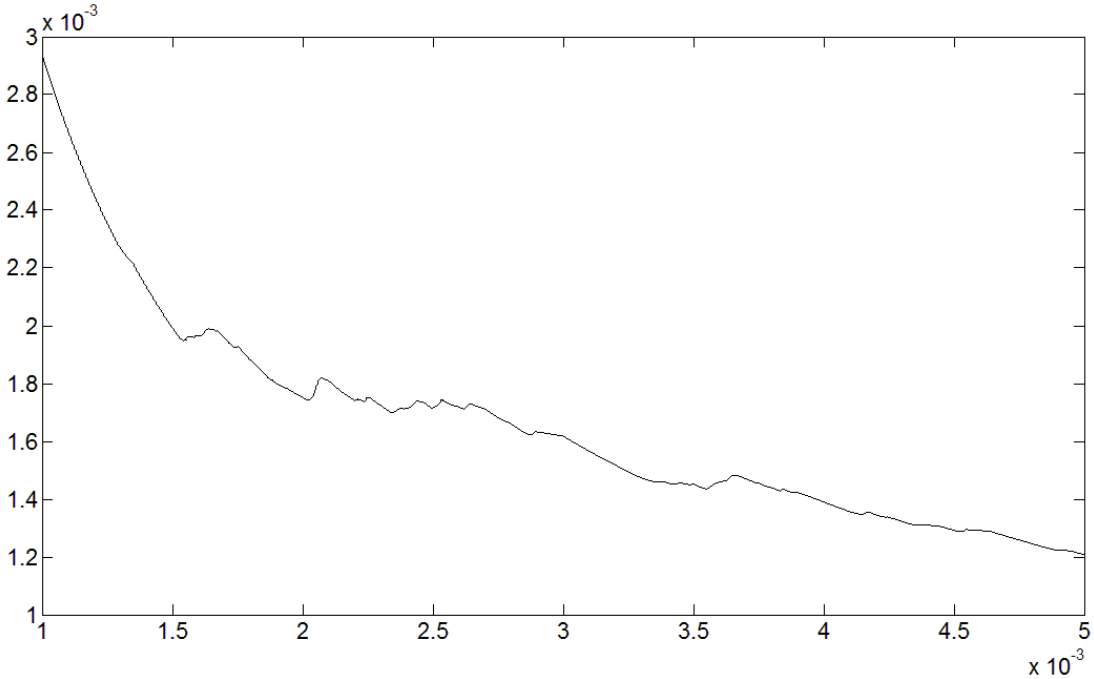


Figura 4.14 Acercamiento del error cuadrático medio del Sistema Difuso MISO adaptable padre.

De igual forma que en el caso del sistema de adaptación del Sistema Difuso SISO, en la Figura 4.12 se puede observar como el sistema de adaptación trata de seguir la respuesta del sistema de referencia. Debe observarse como el sistema adaptable MISO también tiene un comportamiento errático debido a la naturaleza del algoritmo. Sin embargo, observando la Figura 4.13 se puede ver como realmente el sistema de adaptación tiende a encontrar una solución. El error de adaptación tampoco llega a ser cero. En la Figura 4.14 se hace un acercamiento al comportamiento del error del sistema y también se puede observar como hay intervalos de tiempo en los que el error se incremente y después vuelve a decrecer teniendo una tendencia a minimizar el error.

Conclusiones

La adaptación de sistemas electrónicos es una tarea que requiere de herramientas y técnicas de diseño que puedan lidiar con la complejidad de los sistemas y de los modelos que los caracterizan. El uso de herramientas de software ha ayudado en la síntesis de sistemas electrónicos. De igual forma cuando se requiere de un sistema de identificación electrónico es posible sintetizar un circuito mediante el uso de herramientas de software. Sin embargo, esto implica que se deben tener muestras del comportamiento del sistema a modelar y tratarlas computacionalmente para obtener la síntesis del circuito. Con la introducción de técnicas computacionales como la Computación Suave en la tarea de síntesis se han obtenido mejores resultados. Como se ha mencionado anteriormente, en estos casos se dice que el tipo de adaptación se realiza es '*adaptación fuera de línea*'. Este tipo de adaptación es aceptable cuando los costos en tiempo y recursos computacionales son aceptables.

Una solución al problema de no disponer del tiempo de síntesis requerido o un sistema de cómputo que pueda manejar los complejos modelos matemáticos es la denominada '*adaptación en línea*'. La adaptación en línea nos da la posibilidad de poder sintetizar un sistema electrónico tomando directamente la señal de referencia y ajustar la arquitectura o los parámetros del sistema electrónico adaptable. Esto implica que el circuito adaptable debe tener la propiedad o característica de ser un sistema programable y/o reconfigurable.

Si bien se ha logrado el diseño de sistemas electrónicos capaces de reconfigurarse o programarse, también es necesario que el sistema de adaptación pueda interactuar en línea con el circuito adaptable. Los sistemas de cómputo son de gran ayuda en esta labor, pero dependiendo del tipo de circuito – ya sea analógico, digital o mixto – es necesario diseñar una interfaz compatible entre ambos sistemas. Si el sistema adaptable es digital el diseño de la interfaz puede resultar en un diseño simple. Cuando se trata de circuitos adaptables analógicos o que los bloques de adaptación son analógicos no solo se requiere de la interfaz con el sistema de cómputo, también es necesario utilizar convertidores de señal analógica a digital y/o digital a analógica. Algunas veces esto tiene el inconveniente de no permitir una adaptación en tiempo real.

Los procesos de conversión aunados al de la interfaz y procesamiento computacional pueden ralentizar el proceso de adaptación. Debido a esto, es preferible tener un mecanismo de adaptación dentro del mismo sistema electrónico. Esto permitirá que el sistema de adaptación interactúe directamente con el circuito adaptable. El circuito de adaptación puede ser analógico, digital o mixto, dependiendo del tipo que sea el circuito adaptable y además el tipo de técnica o algoritmo de adaptación utilizado. Ahora, se ha dicho que las técnicas de Computación Suave son de gran ayuda en el proceso de adaptación a través de una herramienta de software, por lo que el siguiente paso es su diseño e implementación en hardware. Se ha mencionado que la Computación Suave esta conformado por diversas técnicas como los son las Redes Neuronales, Sistemas Difusos, Algoritmos Evolutivos y Caos entre otros. Todos estos paradigmas tienen ciertas propiedades y cualidades que ayudan en el proceso de adaptación.

No obstante, se ha demostrado que la unión o combinación de estos paradigmas permite el diseño de sistemas de adaptación más robustos. Estos sistemas denominados híbridos, pueden incluir tantas técnicas como el diseñador crea pertinente incluir. La selección de estas técnicas debe ser aun más minuciosa cuando se desea implementarlas en hardware. En el presente trabajo se realizo un análisis y se optó por utilizar tres paradigmas de la Computación Suave: Sistemas Difusos, Estrategias Evolutivas y Caos. Los circuitos adaptables que fueron diseñados son circuitos para Sistemas Difusos debido a que estos sistemas nos permiten modelar otros sistemas de mayor complejidad donde exista incertidumbre o ambigüedad. Los circuitos diseñados son del tipo analógico, ya que este tipo de circuitos nos permiten circuitos compactos y veloces.

Respecto al sistema de adaptación en línea, se decidió diseñar un sistema basado en estrategias evolutivas. Estas estrategias forman parte de un grupo de algoritmos evolutivos como lo son los algoritmos genéticos y la programación evolutiva. Aunque estas dos últimas han demostrado se eficientes la implementación en hardware de estos algoritmos resulta costosa debido a la cantidad de soluciones y evaluaciones que deben generarse para encontrar la solución óptima. Debe notarse que estos algoritmos son estrictamente iterativos o discretos, lo que llevo al diseño de un sistema analógico en tiempo discreto. También debe observarse que al hacer uso de las estrategias evolutivas se busco que el procesamiento fuese paralelo.

Esto implica que se debe contar solo con dos sistemas difusos para poder evaluar el desempeño de un sistema padre y un sistema hijo. Si se desea minimizar el consumo de área y utilizar un solo sistema difuso, se deberá modificar la arquitectura y hacerla secuencial reduciendo la velocidad de adaptación. Si bien la estructura evolutiva es iterativa o discreta, los bloques de procesamiento también son analógicos. Y es aquí donde es necesario encontrar una fuente de soluciones (señales aleatorias) que sea compatible con la estructura evolutiva analógica. Los Mapas Logísticos son una opción muy compatible con la arquitectura evolutiva diseñada. Estos mapas logísticos igualmente son sistemas estrictamente iterativos o discretos. También tienen las propiedades necesarias para generar señales aleatorias con las propiedades estadísticas requeridas de forma controlada. Mediante el ajuste de la ganancia de uno de sus bloques es posible tener un sistema oscilante el cual puede tener un comportamiento caótico.

De esta forma se logró el diseño de un sistema híbrido difuso-evolutivo. Se realizó la adaptación de dos sistemas difusos, uno SISO (una entrada-una salida) y otro MISO (dos entradas-una salida), donde en ambos casos se puede observar como el arquitectura evolutiva trata de adaptar y minimizar el error de adaptación. También se observa que la respuesta del sistema adaptable tiene un comportamiento errático en la búsqueda de la solución óptima. Este comportamiento se debe a la naturaleza del algoritmo implementado, el cual realiza una búsqueda heurística de las soluciones. También se observo que el error no llega a ser cero, esto también de acuerdo a las características del algoritmo EE (1+1). Donde será necesario buscar implementar algún algoritmo de mayor complejidad (EE(1+λ) o EE(μ+λ)).

Trabajo Futuro

Para continuar con la optimización de la arquitectura propuesta en este trabajo, así como de los bloques necesarios, se deberá realizar:

1) **Analizar el comportamiento dinámico de los circuitos generadores de mapas logísticos**. De las mediciones realizadas al circuito fabricado, pudo observarse que variando la frecuencia de reloj puede variar la secuencia generada. Por consecuencia, las señales generadas podrían no ser caóticas y no tener las propiedades estadísticas requeridas para la obtención de soluciones en el esquema evolutivo.

2) **Diseñar circuitos generadores de mapas logísticos controlables**. Es decir, que las propiedades estadísticas de las señales generadas puedan ser controladas por un(os) parámetro(s) externo(s).

3) **Diseñar una arquitectura para una Estrategia Evolutiva (1+1) auto-adaptable**. Una arquitectura donde la desviación estándar de las soluciones generadas pueda variar de forma determinística y mejorar la convergencia hacia el error mínimo global.

4) **Diseñar una arquitectura para una Estrategia Evolutiva (1+λ)**. Analizar la posibilidad de incrementar la complejidad de la arquitectura sin perder el paralelismo en el procesamiento y mejorar el desempeño del sistema difuso-evolutivo.

5) **Incluir un mecanismo de paro en el proceso de adaptación**. Si se obtiene una eficiencia minima en el desempeño del sistema difuso se deberá detener el proceso de adaptación.

6) **Integración del Sistema Difuso-Evolutivo**. Se deberá integrar la arquitectura difuso-evolutiva a nivel VLSI y demostrar experimentalmente su funcionamiento.

Referencias

[1] P. E. Wellstead, *Self-Tuning Systems – Control and Signal Processing*. Baffins Lane, Chichester: John Wiley and Sons Ltd., 1991.

[2] C. M. Roadknight, G. R. Balls, G. E. Mills, and D. P. Brown, "Modeling Complex Environmental Data" *IEEE Trans. Neural Networks,* vol. 8, pp. 852-862, July 1997.

[3] M. Iatrou, T. W. Berger, and V. Z. Marmarelis "Modeling of Nonlinear Nonstationary Dynamic Systems with a Novel Class of Artificial Neural Networks" *IEEE Trans. Neural Networks,* vol. 10, pp. 327-339, March 1999.

[4] R. K. Ursem, T. Krink, M. T. Jensen, and Z. Michalewicz, "Analysis and Modeling of Control Tasks in Dynamic Systems" *IEEE Trans. Evolutionary Computation,* vol. 6, pp. 378-389, August 2002.

[5] S. K. Pal, S. Mitra, *Neuro-Fuzzy Pattern Recognition – Methods in Soft-Computing*. John Wiley and Sons Ltd., 1999.

[6] V. Kecman, *Learning and Soft-Computing*. Cambridge, Massachusetts: The MIT Press, 2001.

[7] T. Bäck, *Evolutionary Algorithms in Theory and Practice*. New York, New York: Oxford University Press, 1996.

[8] A. Kuri, J. Galaviz, *Algoritmos Genéticos*. México, México: Publicaciones del IPN, 2002.

[9] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[10] M. A. Jabri, R. J. Coggins, B. G. Flower. *Adaptive Analog VLSI Neural Systems*. London: Chapman and Hall, 1996.

[11] L.A. Zadeh, "Fuzzy Sets", en *Information and Control*, vol. 8, 1965, pp. 338–353.

[12] A. Kandel, *Fuzzy expert systems*. CRC Press, Inc., Boca Raton, FL, 1992.

[13] M. Sugeno, G. T. Kang, "Structure identification of fuzzy model", en *Fuzzy Sets and Systems*, vol. 28, pp. 15-33, 1988.

[14] B. Kosko. *Neural networks and fuzzy systems: a dynamical systems approach*. Prentice Hall, Upper Saddle River, NJ, 1991..

[15] E. H. Mamdani, S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller", en *Inter. Journal of Man-Machine Studies*, vol. 7A, pp. 1-13, 1975.

[16] C. A. Coello, *Introducción a la Computación Evolutiva*. Notas de Curso, LANIA, México, 2000.

[17] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. New York, New York: Springer, 1996.

[18] L. David, *Handbook of Genetic Algorithms*. New York, New York: Van Nostrand Reinhold, 1991.

[19] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann–Holzboog, Stuttgart, Alemania, 1973.

[20] Hans-Paul Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Birkh¨auser, Basel, Alemania, 1977.

[21] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chichester, UK, 1981.

[22] John H. Holland. Concerning efficient adaptive systems. En M. C. Yovits, G. T. Jacobi, G. D. Goldstein, *Self-Organizing Systems—1962*, pp. 215–230. Spartan Books, Washington, D.C., 1962.

[23] Günter Rudolph, "Convergence Analysis of Canonical Genetic Algorithms", en *IEEE Transactions on Neural Networks*, vol. 5, pp. 96–101, 1994.

[24] J. David Schaffer, AmyMorishima. An Adaptive Crossover Distribution Mechanism for Genetic Algorithms. En John J. Grefenstette, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pp. 36–40. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1987.

[25] Lawrence Davis. Adapting Operator Probabilities In Genetic Algorithms. En J. David Schaffer, *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 61–69, SanMateo, California, 1989. Morgan Kaufmann Publishers.

[26] T. Bäck. Self-adaptation in genetic algorithms. En F.J. Varela. P. Bourgine, *Proceedings of the First European Conference on Artificial Life*, pp. 263–271, Cambridge, Massachusetts, 1992. MIT Press.

[27] Lawrence J. Fogel. *Artificial Intelligence through Simulated Evolution*. JohnWiley, New York, 1966.

[28] R. M. May, and G. F. Oster, "Bifurcations and Dynamic Complexity in Simple Ecological Models," *American Naturalist*, vol. 110, pp. 573–599, 1976.

[29] S. N. Rasband, *Chaotic Dynamics of Nonlinear Systems*. Wiley, New York, NY, 1997.

[30] G. Chen, J.L. Moiola and H.O. Wang, “Bifurcation control: theories, methods, and application”, en *Int. J. Bifurcation Chaos*, Vol. 10, No. 3, pp. 511–548, 2000.

[31] H.-O. Peitgen, H. Jürgens, D. Saupe, *Chaos and fractals – New frontiers of science*, Springer, 2004.

[32] E. A. Jackson, *Perspectives of nonlinear dynamics, Vols. 1 y 2*. Cambridge university Press, 1990.

[33] L. A. Zadeh, “*Fuzzy logic, neural networks and soft-computing*”, página del curso CS294-4, Universidad de California, Berkeley, Noviembre, 1992.

[34] A. Kandel, G. Langholz, *Fuzzy Hardware – Architectures and Applications*. Norwell, Massachussets: KAP, 1998.

[35] I. Baturone, Á. Barriga, S. Sánchez, C. J. Jiménez, D. R. López, *Microelectronic Design of Fuzzy Logic-Based Systems*. Boca Raton, Florida: CRC Press, 2000.

[36] C. Dualibe, P. Jespers, M. Verleysen, *Design of Analog Fuzzy Logic Controllers in CMOS Technology*. Kluwer Academic Publishers, 2003.

[37] J. Qiu, M. Walters, “A GA-based Learning Algorithm for the Learning of Fuzzy Behaviour of a Mobil Robot Reactive Control System”, en *Proceedings of the International Conference on Genetic Algorithms in Engineering Systems*, September 1997, pp. 251-258.

[38] J. C. Gallagher, S. Vigraham, and G. Kramer, “A Family of Compact Genetic Algorithms for Intrinsic Evolvable Hardware” en *IEEE Trans. Evolutionary Computation*, vol. 8, pp. 111-126, 2004.

[39] C. Mead, *Analog VLSI and Neural Systems*. Addison-Wesley, 1989.

[40] J.B. Theeten, M. Duranton, N. Mauduit, J. A. Sirat, "*The LNeuro chip: A digital VLSI with on-chip learning mechanism*", en Proc. of the Inter. Conference on Neural Networks, vol. 1, pp. 593-596,1990.

[41] M. Togai, H. Watanabe, “A VLSI Implementation of a Fuzzy Inference Engine: Toward an Expert System on a Chip”, en *Information Science*, vol. 38, pp. 147-163, 1986.

[42] T. Yamakawa, T. Miki, “The Current Mode Fuzzy Logic Integrated Circuits Fabricated by the Standard CMOS Process”, en *Computers, IEEE Trans*, vol. C-35, no. 2, pp. 161-167, 1986.

[43] M. Patyra et al., “Digital Fuzzy Logic Controller: Design and Implementation”, en *Fuzzy Systems, IEEE Trans.*, vol. 4, no. 4, pp. 439-459, 1996.

[44] M. Sasaki, F. Ueno, “A Fuzzy Function Generator (FLUG) Implemented with Current Mode CMOS Circuits”, en *21st Inter. Symp. on Multiple Valued Logic*, pp. 356-362, 1991.

[45] J.L. Huertas, S. Sanchez-Solano, A. Barriga, I. Baturone, “Serial Architecture for Fuzzy Controllers: Hardware Implementation Using Analog Digital VLSI Techniques”, en *2nd Inter. Conf. on Fuzzy Logic and Neural Networks*, pp. 353-358, 1992.

[46] O. Ishizuka, K. Tanno, Z. Tang, H. Matsumoto, “Design of a Fuzzy Controller with Normalization Circuit”, en *IEEE Inter. Conf. on Fuzzy Systems*, pp. 1303-1308, 1992.

[47] J. López-Hernández, *Diseño Orientado a Síntesis de Circuitos Analógicos de Bajo Voltaje para Aplicaciones de Lógica Difusa*. Tesis de Maestría, INAOE, México, 2003.

[48] L. Peters, S. Guo, R. Camposano, “A Novel Analog Fuzzy Controller for Intelligent Sensors”, en *Fuzzy Sets and Systems 70*, pp. 235-247, 1995.

[49] T. Miki, T. Yamakawa, “Fuzzy Inference on an Analog Fuzzy Chip”, en *IEEE Micro*, vol. 15, no. 4, pp. 58-66, 1995.

[50] T. Kettner, K. Schumacher, K. Goser, “Realization of a Monolithic Analog Fuzzy Logic Controller”, en *20th European Solid State Circuit Conf.*, pp. 66-69, 1993.

[51] T. Miki et al., “Silicon Implementation for a Novel High-Speed Inference Engine: Mega-FLIPS Analog Fuzzy Processor”, en *J. Intelligent & Fuzzy Systems*, vol. 1, no. 1, pp. 27-42, 1993.

[52] L. Lemaitre, M.J. Patyra, D. Mlynek, “Analysis and Design of Fuzzy Logic Controller in Current Mode”, en *Solid State Circuits, IEEE Journal*, vol. 29, no. 3, pp. 317-322, 1994.

[53] T. Higuchi, Y. Liu, X. Yao, *Evolvable Hardware*. New York: Springer, 2006.

[54] G.W. Greenwood, A.M. Tyrrell, *Introduction to Evolvable Hardware - A Practical Guide for Designing Self-Adaptive Systems*. New Jersey: IEEE-Wiley, 2007.

[55] N. Nedjah, L. Mourelle, *Evolvable Machines - Theory & Practice*. Alemania: Springer, 2005.

[56] Y.S. Tang, A.I. Mees, L.O. Chua, "Synchronization and Chaos", en *IEEE Trans. Circuits & Syst.*, vol. 30, no. 9, 1983.

[57] L.O. Chua, G.Q. Zhong, "Negative Resistance Curve Tracer", *Circ. & Syst., IEEE Trans.*, vol. 32, no. 6, 1985.

[58] G. McGonigal, M. Elmasry, " Generation of noise by electronic iteration of the logistic map", en *Circ. & Syst., IEEE Trans.*, vol. 34, no. 8, 1987.

[59] M. Delgado, A. Rodriguez, S. Espejo, J.L. Huertas, "A Chaotic Switched-Capacitor Circuit for 1/F Noise Generation", en *Circ. & Syst. I, IEEE Trans.*, vol. 39, no. 4, 1992.

[60] T. Stojanovski, J. Pihl, L. Kocarev, "Chaos Based Random Number Generators PART II: Practical Realization", en *Circ. & Syst. I, IEEE Trans.*, vol. 48, no. 3, 2001.

[61] P. Dudek, V.D. Juncu, "Compact discrete-time chaos generator circuit", en *Electronics Letters*, vol. 39, no. 20, 2003.

[62] G. Cauwenberghs, "Delta-sigma cellular automata for analog VLSI random vector generation", en *Circ. & Syst. II, IEEE Trans.*, vol. 46, no. 3, 1999.

[63] T. Yamakawa, T. Miki, F. Ueno, "The design and fabrication of the current mode fuzzy logia semi-custom IC in Standard CMOS IC technology", en *Proc. 15th Int. Sympusium on Multiple-Valued Logic*, 1985, pp. 73-82.

[64] M.J. Patyra, J.E. Long, "Synthesis of Current Mode Building Blocks for Fuzzy Logic Control Circuits", en Proc. ISCAS'94, London, 1994, pp. 283–286.

[65] T. Yamakawa, "A fuzzy inference engine in nonlinear analog mode and its application to a fuzzy logic control", en *IEEE Trans. Neural Networks*, vol. 4, pp. 496-522, 1993.

[66] K. Bult, H. Wallinga, "A Class of Analog CMOS Circuits Based on Square-Law Characteristic of an MOS Transistor in Saturation", en *IEEE Journal of Solid-State Circuits*, vol. 22, no. 3, pp. 357-365, 1987.

[67] B. Gilbert, "A monolithic 16-channel analog array normalizer", en *IEEE J. Solid-State Circuits*, vol. 19, pp. 956-963, 1984.

[68] R. J. Wiegerink, *Analysis and Synthesis of MOS Translinear Circuits*. Norwell, Massachussets, Kluwer Academic Publishers, 1993.

Apéndice - Layouts

En este apéndice se proporcionan patrones geométricos de los bloques funcionales requeridos para la implementación del sistema difuso-evolutivo.

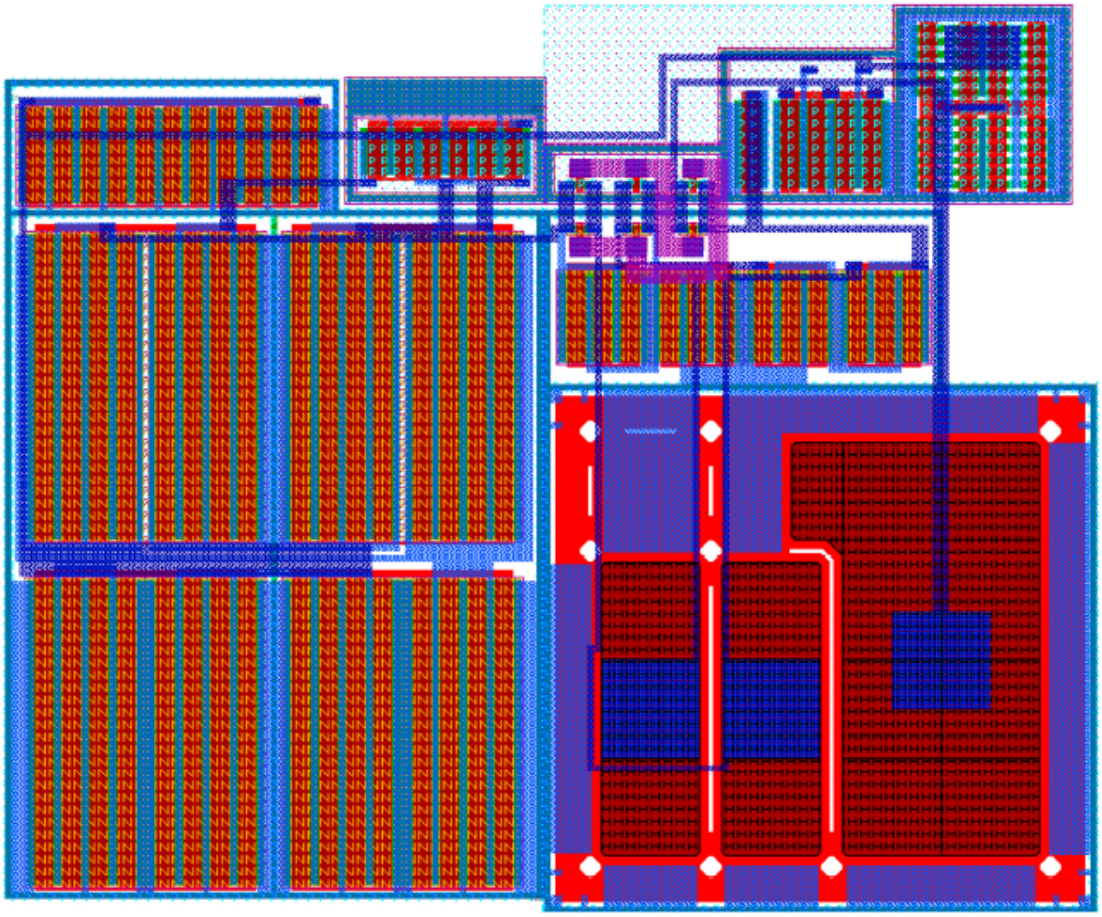


Figura A.1 Oscilador Caótico (Mapa Logístico iterado).

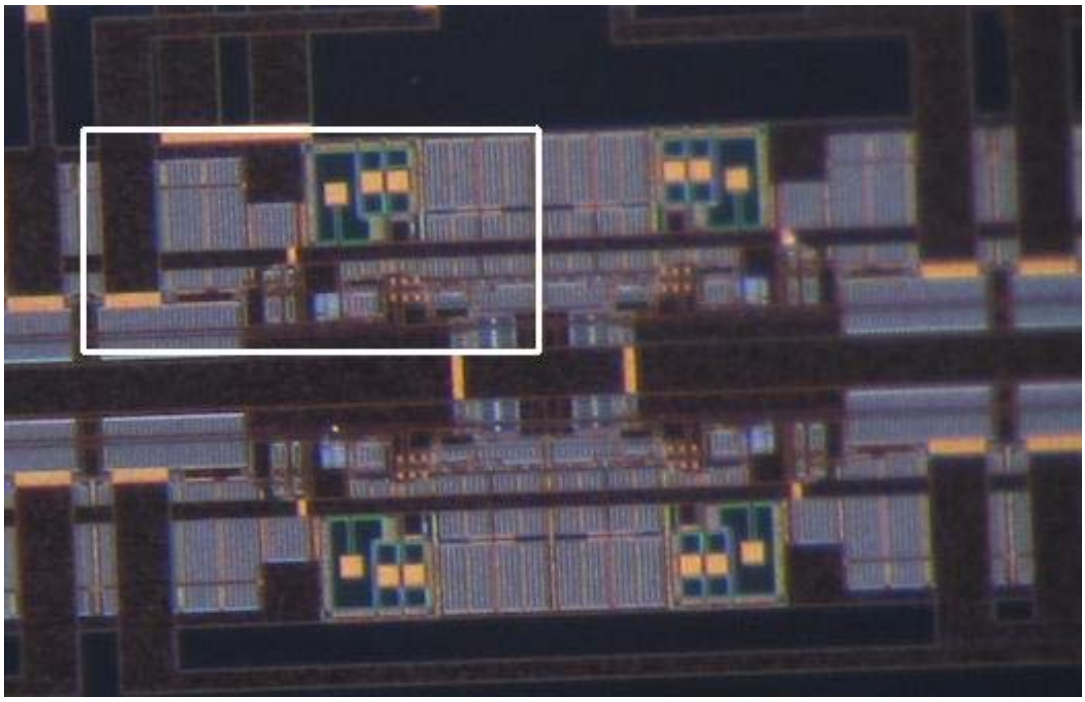


Figura A.2 Oscilador Caótico fabricado.

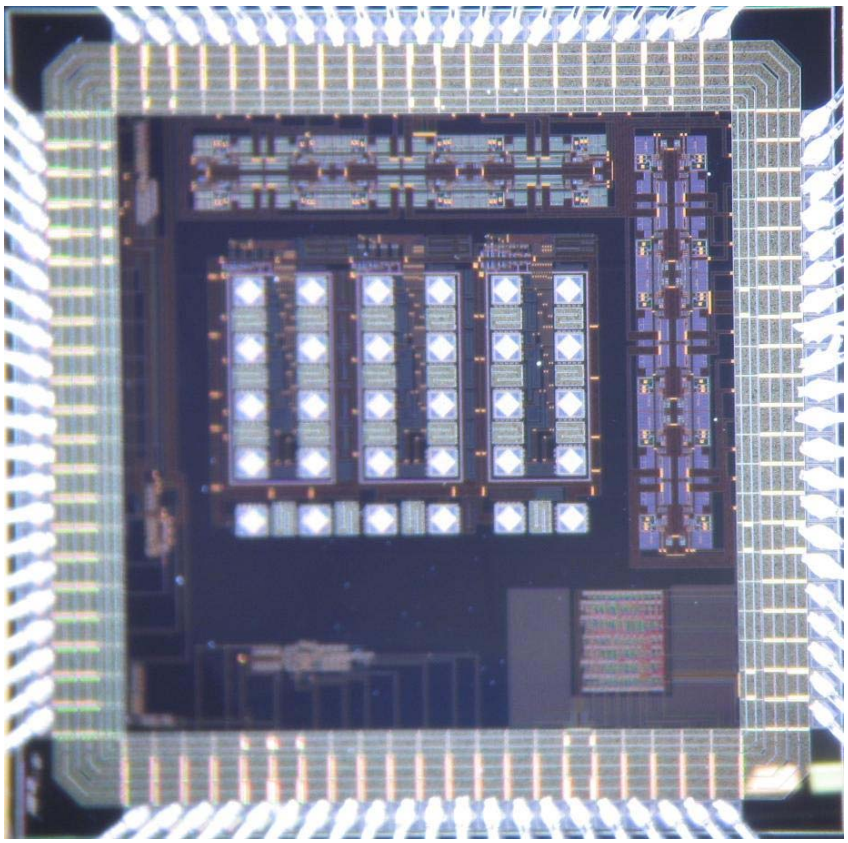


Figura A.3 Vista completa del circuito integrado fabricado.

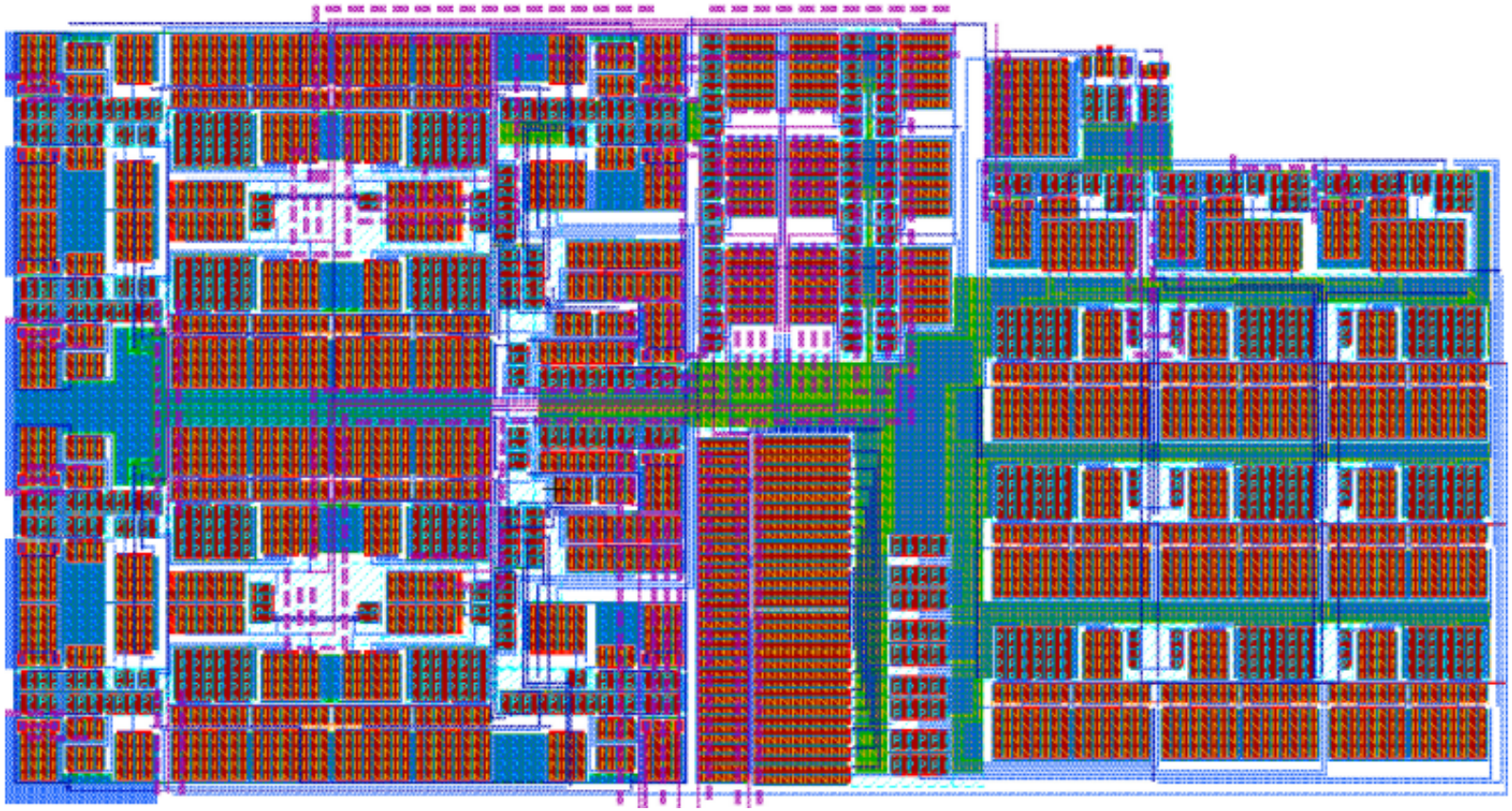


Figura A.4 Sistema Difuso de 2-entradas 1-salida y base de reglas de 3X3.