Diseño de un Control Difuso con Adaptación en Chip Backpropagation.

por

José Erasmo Arroyo Huerta

Tesis sometida como requisito parcial para obtener el grado de

MAESTRO EN CIENCIAS EN LA ESPECIALIDAD DE ELECTRONICA

en el

Instituto Nacional de Astrofísica, Óptica y Electrónica
Noviembre 2007
Tonantzintla, Puebla

Supervisada por:

Dr. José Alejandro Díaz Méndez

RESUMEN.

En este trabajo, se realiza la implementación a nivel Circuito Integrado de un controlador difuso tipo Takagi-Sugeno-Kang de orden cero, adaptado mediante el algoritmo Nonlinear Backpropagation. Primero se modela el controlador difuso sin el algoritmo, haciendo simulaciones tanto en MATLAB como en SPICE. Posteriormente, se incluye el algoritmo de adaptación para entrenar al controlador con tres superficies no lineales como objetivo: superficie sinusoidal, superficie cuadrática y superficie Sinc. El parámetro del controlador que es adaptado mediante el algoritmo son los singleton.

A partir de simulaciones, se encuentra el valor inicial óptimo de los singleton para un mejor desempeño en el entrenamiento del sistema adaptable. Se establece la relación entre resolución en la salida del controlador para las superficies objetivo y el costo de implementación, ya sea en hardware o software.

Finalmente, se propone un módulo adicional al sistema para poder controlar el valor de la constante de adaptación, modificando dinámicamente su valor de acuerdo a la señal de error calculada, para garantizar la máxima velocidad de entrenamiento y la convergencia hacia un punto fijo estable del sistema adaptable.

ABSTRACT.

This work presents the implementation of a Takagi-Sugeno-Kang Zero-Order VLSI (Very Large Scale Integration) Adaptive Fuzzy Controller; the fuzzy controller has been trained by Nonlinear Backpropagation algorithm. SPICE and MATLAB's simulations was made to characterize the Fuzyy Controller with and without the algorithm by using three no-linear surfaces: Sinusoidal, quadratic and Sinc functions. The algorithm tunes the values of the singletons to adapt the Fuzzy Controller.

Doing simulations, an optimal value of singleton is found to improve the performance of the adaptive system training. A tradeoff between input resolution and cost of implementation, be it Hardware of Software, are established.

Finally, an additional module to modify the value of Learning Rate is proposed. The Learning Rate is shifted between two values to guarantee and to accelerate the convergence ratio of the adaptive system.

AGRADECIMIENTOS.

A mis hermanos, que mantuvieron su confianza y apoyo hacia mí en esta etapa que concluye.

A mi asesor de tesis, Dr. José Alejandro Díaz Méndez, por su guía y apoyo durante el desarrollo de este trabajo.

A mis amigos del INAOE por su paciencia, apoyo, comprensión, consejos e invaluables aportaciones en mi crecimiento personal y profesional.

Al pueblo de México, que mediante la beca otorgada por el Consejo Nacional de Ciencia y Tecnología, CONACyT, ha permitido la continuación de mis estudios.

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

DEDICATORIA.

A mis padres, Rubén y Lourdes, por su infinito amor, por ser mi razón e inspiración para seguir adelante, por su apoyo incondicional, por su paciencia interminable, por ser la fuente de todo cuanto soy.

ÍNDICE.

PREFACIO.

La complejidad de los sistemas de control actuales está creciendo de manera muy rápida. Los sistemas de control convencionales, tales como PID, ofrecen una solución limitada cuando las plantas que controlan son altamente no lineales y variantes en el tiempo, debido a que no es trivial modelar estos sistemas de forma matemática.

Es por eso que surge una nueva alternativa en el área de control mediante el uso de controladores difusos, los cuales, debido a su capacidad de modelar sistemas de una forma más lingüística que matemática, son ideales para tratar con plantas cuya descripción matemática es difícil de obtener, pero que, sin embargo, obtener una descripción aproximada mediante el conocimiento de un experto es mucho más sencillo. La naturaleza no-lineal de los controladores difusos es aprovechada cuando la planta que se desea controlar presenta este mismo comportamiento no-lineal.

Sin embargo, un controlador difuso por sí mismo no es capaz de controlar plantas cuyas características son variantes en el tiempo, sino que necesitan de un sistema de entrenamiento que modifique sus parámetros para adaptarse a las nuevas exigencias que conlleva un cambio en el punto de operación de la planta que se está controlando. Es aquí donde se hace uso de los Algoritmos de Adaptación, los cuales, de manera muy general, realizan el entonado de ciertos parámetros del controlador mediante un lazo de retro-alimentación del sistema, calculando un parámetro de error al comparar la salida de la planta con una señal deseada. Dos parámetro muy importantes de los sistemas adaptables retro-alimentados son la estabilidad y la velocidad de convergencia, ambos parámetros están determinados directamente por un valor constante llamado "*Constante de Adaptación"* la elección adecuada de éste parámetro determinará el desempeño global del sistema.

Existen muchos algoritmos de adaptación, sin embargo, su implementación en Hardware muchas veces es muy cara, debido a la alta complejidad que estos algoritmos presentan. Es por todo lo anterior que la propuesta del presente trabajo es implementar un Controlador Difuso adaptado mediante un algoritmo con una complejidad relativamente baja al ser implementado en sistemas electrónicos de muy alta integración (VLSI) para hacer la adaptación tanto "en-línea" como "en-chip". También se propone la adición al sistema adaptable de un bloque que permita controlar el valor de la constante de adaptación en base a un criterio bien establecido: un valor grande cuando el error del sistema sea grande, así la velocidad de convergencia aumenta y no se cae en inestabilidad, puesto que cuando el error es pequeño, la constante de adaptación también es pequeña disminuyendo la velocidad de convergencia pero garantizando de esta forma la estabilidad.

La tesis está organizada de la siguiente forma:

En el Capítulo uno se presenta una introducción a los conceptos básicos de Lógica Difusa, Controladores Difusos, Esquemas de Control, y los Sistemas Adaptables.

En el Capítulo dos se presenta la descripción generalizada de los algoritmos de adaptación que utilizan el método del gradiente descendente más comunes. Para cada algoritmo, se presentan sus ventajas, desventajas y modelado matemático.

En el Capítulo tres, se presenta la estructura básica del controlador difuso utilizado, así como de la propuesta de implementación del algoritmo Nonlinear Backpropagation. En este mismo Capítulo, se modelan y simulan en MATLAB los bloques de construcción necesarios para la implementación, tanto del controlador difuso como del algoritmo de adaptación.

En el Capítulo cuatro, se presenta el diagrama a bloques propuesto para la implementación a nivel transistor del sistema adaptable completo. También se diseñan, simulan y caracterizan todos los bloques de

construcción necesarios para la realización electrónica en sistemas VLSI del sistema adaptable. Se presentan de manera gráfica los resultados de las simulaciones realizadas en SPICE y se muestra un resumen en forma de Tabla de las características de cada bloque.

El Capítulo cinco contiene los resultados finales de las simulaciones del sistema adaptable hechas en MATLAB y en SPICE, se muestra de forma gráfica los resultados alcanzados y se comparan con los resultados obtenidos en otros trabajos para medir las mejoras alcanzadas con la presente propuesta.

En el Capítulo seis se presentan las conclusiones obtenidas al finalizar el presente trabajo.

CAPÍTULO 1. INTRODUCCIÓN.

1.1 La Lógica Difusa.

En 1965, Lofti Z. Zadeh hace la primera publicación sobre conjuntos difusos [1] motivado en gran medida por la convicción de que los métodos tradicionales de análisis de sistemas son inadecuados para representar sistemas en los cuales las relaciones entre variables no permiten por sí mismas ser representadas en términos de derivadas o ecuaciones diferenciales. Dichos sistemas son comunes en campos como biología, sociología, economía, y, más generalmente, en campos en los cuales los sistemas tienen una naturaleza más humanística que mecánica. Este primer trabajo constituyó los cimientos de lo que ahora conocemos como Lógica Difusa.

La Lógica Difusa modela de forma adecuada la incertidumbre del pensamiento humano y ofrece un formalismo matemático que emula el esquema de deducción humana. Debido a que la Lógica Difusa formaliza el tratamiento de conocimientos vagos e imprecisos, ésta ha sentado las bases para la implementación de algoritmos innovadores que dan solución a problemas complejos cuya naturaleza, más lingüística que numérica, hace muy difícil su análisis con los métodos tradicionales. Una variable lingüística es aquélla cuyo valor son palabras y no números. El concepto de variable lingüística ha jugado, y continúa jugando un papel clave en las aplicaciones tanto de la Teoría de Conjuntos Difusos como de Lógica Difusa.

A diferencia de la Lógica Booleana, la cual utiliza solamente dos valores o distinciones tajantes (cero y uno) trazando una línea entre los elementos que pertenecen y los que no pertenecen a una determinada clase, la Lógica Difusa (también conocida como Lógica Multi-valuada) extiende el rango de los valores de verdad a todos los números reales en el intervalo entre cero y uno. En la década de los 30's Jan Lukasiewicz, un filósofo

Polaco, utilizó un número de éste intervalo para representar la posibilidad de que cierta sentencia fuese falsa o verdadera. Como una analogía, la Lógica Booleana solamente puede evaluar dos colores: blanco y negro (Figura 1.1 a)), mientras que la Lógica Difusa hace uso de todo el espectro de grises (Figura 1.1 b)), aceptando así, que las cosas pueden ser parcialmente falsas y parcialmente verdaderas al mismo tiempo.

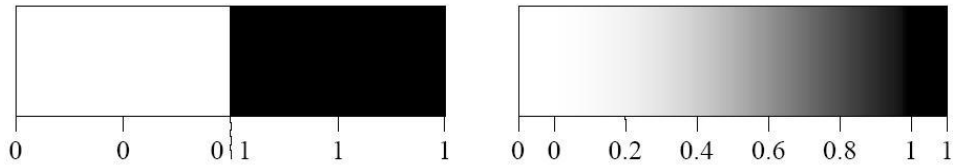

Figura 1.1 Conjuntos a) Lógica Booleana, b) Lógica Multi-Valuada.

Así por ejemplo, considerando una persona con edad de cincuenta años, la Lógica Booleana solamente podría clasificarla cono “vieja” ó “joven”, mientras que la Lógica Multi-valuada la clasificaría como “60% vieja y 40% joven”.

1.2 Controladores Difusos.

Desde sus inicios, la Lógica Difusa encontró su principal campo de aplicación en el área de control debido a su capacidad para expresar el conocimiento en una forma lingüística, permitiendo que cualquier sistema pueda ser descrito por reglas simples que son fácilmente entendibles por el humano. En 1974, el británico Ebrahim Mandami, demuestra la aplicabilidad de la Lógica Difusa en el campo del control, desarrollando el primer sistema de control difuso práctico: la regulación de un motor de vapor [2].

Así como la Lógica Difusa puede ser definida simplemente como “*calculando más con palabras que con números*”, el control difuso puede ser descrito simplemente como “*controlando más con sentencias que con ecuaciones*”, estas sentencias pueden ser simplemente reglas empíricas.

1.2.1 Esquemas De Control.

Los controladores difusos han sido ampliamente utilizados en diversos sistemas de control no lineal, en aplicaciones para productos comerciales (lavadoras, cámaras de video, hornos), en procesos industriales, robots y sistemas de transporte (en 1986 Hitachi pone en operación un sistema de transporte subterráneo basado en control difuso).

Existen varios sistemas de control en los que se utilizan ampliamente controladores difusos. Uno es el *control directo*, mostrado en la Figura 1.2, en el cual la salida del proceso es comparado con una referencia, si existe una desviación (ó error), el controlador toma una acción de acuerdo con la estrategia de control y los datos introducidos por el usuario.

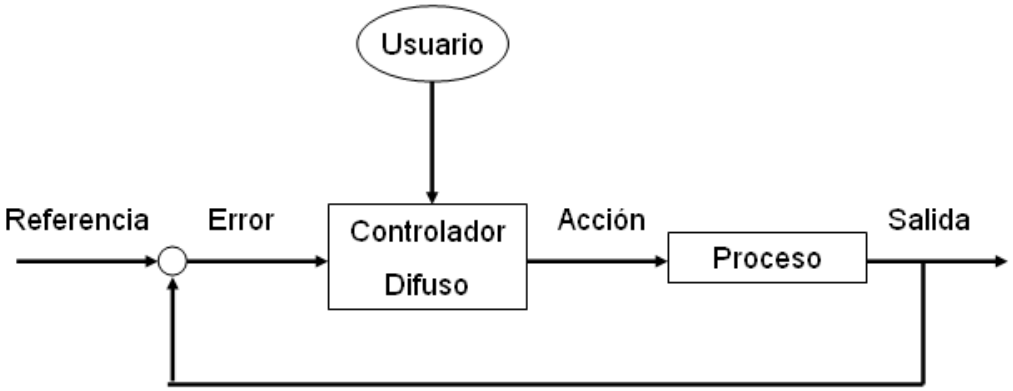

Figura 1.2 Diagrama de Control Directo.

Otro sistema es el control "*feedforward*", en este caso, se debe compensar una perturbación medible. Debido a que éste tipo de sistema de control requiere de un buen modelo, el cual casi siempre es muy complicado o difícil de obtener, entonces se puede utilizar un controlador difuso. La Figura 1.3 muestra un controlador y el compensador difuso. El esquema, omitiendo la entrada de la perturbación, puede ser visto como una colaboración de acciones de control lineales y no lineales; el controlador C puede ser un controlador lineal de tipo PID, mientras que el controlador difuso F es un controlador no lineal suplementario.

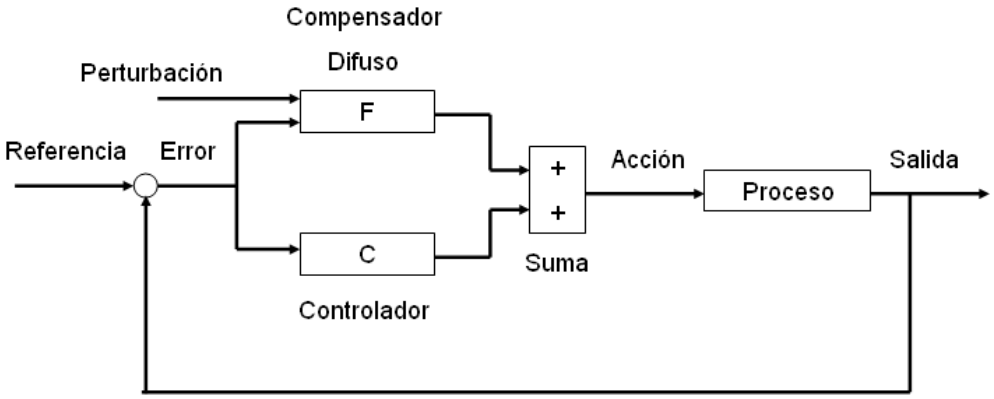

Figura 1.3 Diagrama de control "Feedforward".

Un tercer sistema es de control con parámetros adaptables, mostrado en la Figura 1.4. Si una planta no lineal cambia su punto de operación, éste esquema hace posible el cambio de parámetros del controlador de acuerdo a cada punto de operación. Éste esquema contiene un controlador difuso cuyos parámetros son cambiados como una función del punto de operación en una forma predeterminada.

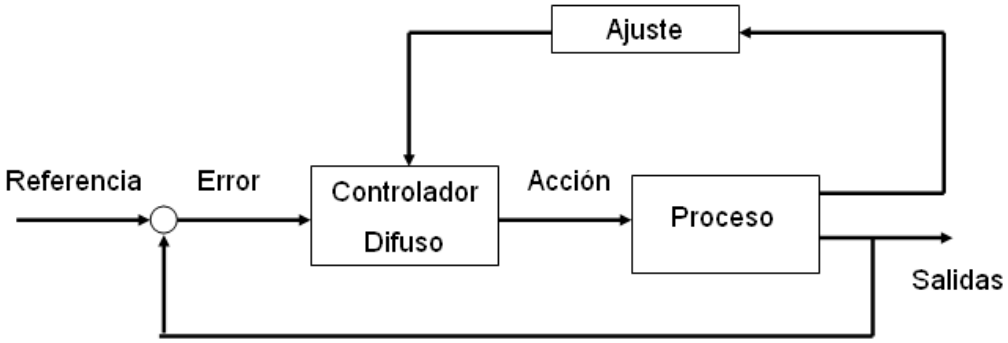

Figura 1.4 Diagrama de Control con Ajuste de Parámetros.

1.2.2 Estructura De Un Controlador Difuso.

Existen componentes característicos de un controlador difuso que soportan un procedimiento de diseño. En la Figura 1.5 se muestra la

estructura básica de un controlador difuso, el cual se encuentra entre bloques de pre-procesamiento y post-procesamiento.

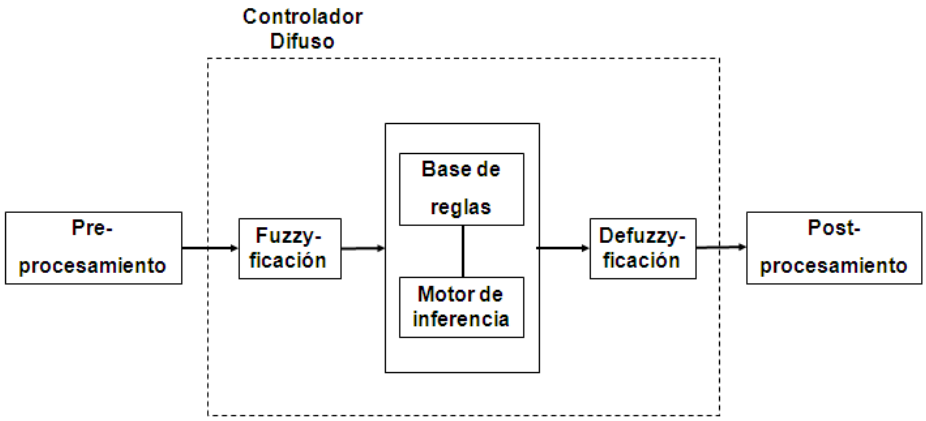

Figura 1.5 Estructura básica de un Controlador Difuso.

Debido a que las entradas a un sistema electrónico son casi siempre mediciones más duras (*crisp)* que lingüísticas, es necesario un bloque de pre-procesamiento que acondicione las mediciones antes de que éstas entren en el controlador. Algunos ejemplos típicos de la acción que realiza el bloque de pre-procesamiento son:

- Normalización o escalamiento dentro de un rango particular.
- Filtrado de ruido.
- Promedios para obtener las tendencias de la señal.
- Diferenciación e integración de sus equivalentes discretos.

En sistemas electrónicos VLSI (Very Large Scale Integration), el bloque de pre-procesamiento se realiza principalmente fuera de chip, de ésta manera, el chip que contiene al controlador difuso recibe en sus terminales las señales de entrada listas para ser procesadas en forma difusa.

El primer bloque dentro del controlador difuso es el de fuzzyficación el cual convierte cada dato de entrada a grados de membresía por medio de

una ó varias funciones de membresía. Es decir, transforma el valor determinístico de la variable de entrada en un conjunto difuso. De ésta forma, el bloque de fuzzyficación acopla los datos de entrada con las condiciones de las reglas para determinar qué tan bien la condición de cada regla se acopla a una instancia de entrada particular. Existe un grado de membresía para cada término lingüístico que aplica para cada variable de entrada:

$$\Phi_i = Fuzzyficador(\varphi_c) \qquad (1.1)$$

Donde $\boldsymbol{\Phi_i}$ es un conjunto difuso y $\boldsymbol{\varphi_c}$ es la variable de entrada al controlador.

Posteriormente, se tiene el motor de inferencia que, junto con la base de reglas, realizan la toma de decisiones que dictarán la forma en que actuará el sistema. El método de inferencia se basa en el grado de pertenencia de los datos de entrada en los conjuntos difusos de los espacios correspondientes, siempre que éstas se den, para tomar una decisión en el espacio de salida. El conjunto de reglas, que son la base de conocimiento, es del tipo antecedente-consecuente; es decir, tiene la forma: *"si <condiciones difusas ó variables de entrada> entonces <acciones de control difuso":*

$$Si\left(\varphi_{j1} = \Phi_{j1}\right) y ... y\left(\varphi_{jq} = \Phi_{jq}\right) Entonces\left(f = \alpha_{j1}\varphi_{j1} + \alpha_{j2}\varphi_{j2} + ... + \alpha_{jq}\varphi_{jq}\right) \quad (1.2)$$

Existen distintos métodos para llevar a cabo la etapa de inferencia difusa: Mínimo-Máximo, Máximo-Producto:

$$\begin{aligned} \mu_j &= \min(\mu_1, \mu_2) \\ \mu_j &= \mu_1 * \mu_2 \end{aligned} \qquad (1.3)$$

El primer método es el más utilizado debido a su sencillez, aunque provoca cierta pérdida de información ya que sólo una de las entradas es considerada (con $\boldsymbol{\mu}$ mínima). La segunda está libre de ésta desventaja, pero

la membresía resultante es desproporcionalmente menor (el producto de dos números menores que uno es menor que el menor de ellos).

El último bloque del controlador difuso convierte los valores difusos de las variables de salida en valores concretos dentro del universo de discurso correspondiente; es decir genera una acción no difusa a partir de la acción difusa resultante del motor de inferencia. Existen varios métodos de defuzzyficación:

- Centro de Gravedad. La variable de salida es la abscisa bajo el centro de gravedad del conjunto difuso:

$$u = \frac{\sum_i \mu(x_i) x_i}{\sum_i \mu(x_i)} \qquad (1.4)$$

Donde x_i es un punto en el universo discreto y $\mu(x_i)$ es el grado de membresía en la función de membresía. La expresión puede ser interpretada como el promedio pesado de los elementos dentro del conjunto difuso. Para aplicaciones en tiempo continuo, se reemplaza las sumatorias por integrales, tal como se muestra en (1.5):

$$u = \frac{\int_i \mu(x_i) x_i}{\int_i \mu(x_i)} \qquad (1.5)$$

Éste método es ampliamente utilizado a pesar de que su complejidad computacional es relativamente alta. El método de Centro de Gravedad también es llamado Centroide de Área.

- Centro de Gravedad por Singleton. Si las funciones de membresía de los consecuentes son singleton, entonces el valor de la salida es:

$$u = \frac{\sum_i \mu(s_i) s_i}{\sum_i \mu(s_i)} \qquad (1.6)$$

Aquí s_i es la posición del singleton ***i*** en el universo y $\mu(s_i)$ es el grado de membresía. Debido a que μ es diferenciable con respecto a los singleton s_i éste método es utilizado en sistemas neuro-difusos.

- Bisector de Área. Este método toma la abscisa de la línea vertical que divide el área bajo la curva en dos mitades iguales. En sistemas en tiempo continuo:

$$u = \left\{ x \middle| \int_{Min}^{x} \mu(x)\,dx = \int_{x}^{Max} \mu(x)\,dx \right\} \qquad (1.7)$$

Donde ***x*** es un punto en el universo, ***μ(x)*** es su grado de membresía, ***Min*** es el valor más pequeño del universo, y ***Max*** es el valor más grande. Su complejidad computacional es relativamente alta y puede ser ambigua.

- Promedio del Máximo. Es una aproximación intuitiva la cual selecciona el punto con la posibilidad más fuerte, esto es, la máxima función de membresía. Este método no toma en cuenta la forma del conjunto difuso, pero su complejidad computacional es relativamente baja.

1.2.3 Tipos De Controladores Difusos.

Dependiendo de las técnicas de diseño empleadas podemos clasificar a los controladores difusos dentro de tres grandes categorías:

- Controladores Difusos Digitales. Este tipo de controlador fue propuesto en 1986 por Togai y Watanabe [3] y su trabajo dio como resultado, la fabricación de algunos chips [4] [5] [6]. En general un sistema difuso digital puede ser un co-procesador difuso [7] o un Circuito Integrado de Aplicación Específica (ASIC) [8], el cual contiene circuitos lógicos para calcular el algoritmo difuso, memorias para almacenar la base de reglas, y generadores o tablas de búsqueda para las funciones de membresía tanto de las variables de entrada

como las de salida. La principal ventaja de éste tipo de controladores es su gran flexibilidad, que su diseño puede ser automatizado, y la facilidad con la que pueden ser programados. Sin embargo, su desventaja es que los controladores difusos digitales son muy complejos requiriendo más área de chip en aplicaciones VLSI y requieren conversiones Analógico-Digital y Digital-Analógico (A/D, D/A) para poder procesar las señales de entrada (provenientes de sensores) y entregar una señal de salida (dirigida a los actuadores) aplicable a un sistema de control, esto debido a la naturaleza analógica del mundo exterior.

- Controladores Difusos Analógicos. La investigación de sistemas difusos analógicos comenzó con Yamakawa, en un trabajo publicado en 1986 [9], y fue continuado por varios investigadores [10] [11] [12]. Debido a la característica no lineal de los dispositivos activos en los circuitos analógicos, los elementos difusos pueden ser implementados con estructuras muy simples. Esto conlleva a una reducción en la complejidad del circuito lo cual implica una mejor velocidad de desempeño y reduce el consumo de área en chip. Su ventaja es la rapidez de procesamiento debido a que no requieren etapas de conversión A/D y D/A; en contraste con los sistemas difusos digitales, los analógicos presentan una muy pobre flexibilidad.
- Controladores Difusos de Señal Mezclada. En este tipo de controladores se saca ventaja de cada una de las partes, tanto de la Digital como de la Analógica. En esta topología, la programación de las características del controlador se hace de forma digital, y el procesamiento de la información se hace en forma analógica [13] [14].

1.3 Controladores Adaptables.

Como se mencionó anteriormente, el principal campo de aplicación de la Lógica Difusa es el control, sin embargo, a medida que los sistemas que se necesita controlar crecen en complejidad, es necesario también que los sistemas expertos que los controlan se vuelvan más complejos y sean capaces de tomar decisiones frente a perturbaciones que se presenten, de tal manera que se garantice siempre un correcto funcionamiento de el sistema en general.

Por su comportamiento en el tiempo, los sistemas pueden clasificarse en:

- Sistemas Estacionarios. Este tipo de sistemas tienen la característica de que su comportamiento es invariante en el tiempo, todos sus parámetros permanecen constantes. Ante la misma entrada en distintos instantes responden igual, ó dicho de otra manera, cuando la entrada ***y(t)*** depende solo de la entrada ***x(t)***, Ecuación 1.8; por lo tanto el sistema controlador también se mantiene invariante y tampoco necesita variar sus parámetros.

$$Si\ \ L\{x(t)\} = y(t)\ \ entonces\ \ L\{x(t-t_0)\} = y(t-t_0) \quad (1.8)$$

  Donde ***L*** representa el sistema físico en cuestión.

- Sistemas Variantes en el Tiempo. En este tipo de sistemas, los parámetros son funciones explícitas del tiempo. Ante la misma entrada en instantes diferentes, la respuesta es diferente. Al no mantenerse fijos los parámetros de éste tipo de sistemas, el sistema controlador deberá también modificar sus parámetros de acuerdo a las nuevas condiciones del sistema o planta mediante un sistema adicional de entrenamiento que adapte al controlador a las nuevas condiciones de operación.

Generalmente, los sistemas variantes en el tiempo, también son sistemas no lineales que presentan retardos en el tiempo y que cambian su comportamiento dinámico en diversas formas, es por eso que en los últimos años, se ha optado por el uso de controladores difusos, los cuáles por su naturaleza no lineal y su capacidad de poder modelar sistemas matemáticamente muy complejos, son adecuados para controlar este tipo de sistemas. Si bien, un controlador difuso no es capaz por sí mismo de adaptarse a los cambios que pueda presentar la planta, se han propuesto diversas técnicas de entrenamiento para éstos controladores [15] [16] [17][18], en los cuales, el sistema de entrenamiento, generalmente es un algoritmo de aprendizaje que modifica ciertos parámetros del controlador mediante el cálculo de una señal de error.

En general, el uso de controladores difusos adaptables se debe a su capacidad de poder aproximar cualquier función real y continua, controlando de manera transparente el grado de precisión deseado mediante la relación de entrada-salida del controlador [19], sin embargo, los datos que ingresan al controlador difuso no siempre tienen una naturaleza lingüística, sino que pueden contener también datos numéricos, este es el caso de muchos problemas prácticos en ingeniería, economía, administración, etc. Los algoritmos de adaptación permiten incorporar ésta información numérica junto con la información lingüística en un mismo sistema difuso [20].

CAPÍTULO 2. EL ALGORITMO DE RETRO-PROPAGACIÓN.

2.1 Introducción.

Un algoritmo de adaptación es básicamente un *Algoritmo de Aprendizaje* el cual opera iterativamente usando muestras tomadas previamente para modificar los parámetros de un sistema adaptable hasta obtener la mejor respuesta en la salida. Los sistemas adaptables usan datos externos para ajustar automáticamente sus parámetros. Esto significa que estos sistemas "conocen" su salida a través de un lazo de retro-alimentación que incluye una función de costo. El lazo de retro-alimentación es utilizado directamente para cambiar los parámetros a través de procedimientos sistemáticos llamados *"reglas de entrenamiento o de aprendizaje"*, así que la salida del sistema mejora con respecto al objetivo deseado, esto es, el error decrece a través del entrenamiento.

Los sistemas adaptables han tenido un gran impacto en la solución de una amplia variedad de problemas prácticos en diversos campos de las ciencias e ingeniería como procesamiento de señales, cancelación de ruido, clasificación y reconocimiento de patrones, identificación de sistemas, y control. También son usados en muchos productos comerciales como módems, procesamiento de imágenes, procesadores e instrumentación biomédica.

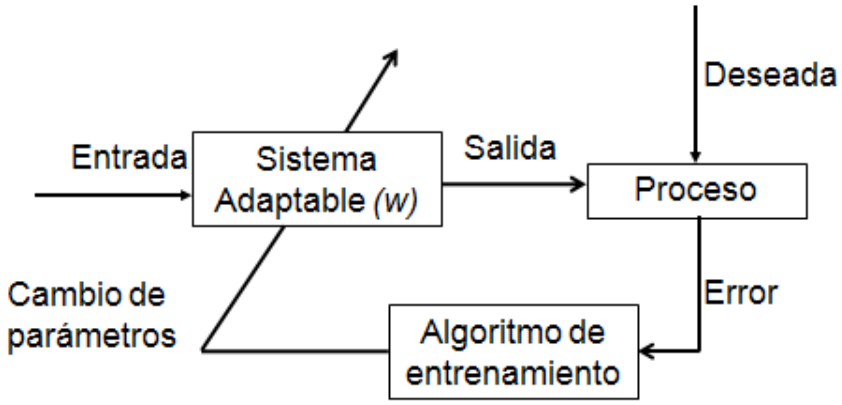

Figura 2.1 Diagrama esquemático de un sistema adaptable.

En la Figura 2.1 se muestra el diagrama a bloques de un sistema adaptable general. El sistema adaptable manda la señal de control hacia la planta o proceso de interés, en este punto, el desempeño del proceso se compara con la respuesta deseada y se genera una señal de error, la cual está dada por:

$$error = salida - deseada \qquad (2.1)$$

Donde ***salida*** es el desempeño actual de la planta y ***deseada*** es la señal objetivo o esperada. El error es recibido por el algoritmo de entrenamiento, éste último genera una señal de corrección para ajustar los parámetros del sistema adaptable y de esta forma aproximar el desempeño actual de la planta al esperado, ó, dicho de otra forma, disminuir la función de error.

2.2 Algoritmos de Adaptación.

Existen diferentes algoritmos de adaptación utilizados en el entrenamiento de sistemas adaptables, los cuales pueden clasificarse en:

- Algoritmos de descenso por gradiente. Estos algoritmos se basan en el cálculo de las derivadas de la *función de costo*. Entendiendo por una *función de costo* aquella que provee una medida cuantitativa para evaluar la calidad del desempeño del sistema adaptable.
- Algoritmos de recocido simulado. Están basados en variaciones estadísticas de los parámetros del sistema.
- Algoritmos Genéticos. Basados en procesos de mutación y cruza entre diferentes realizaciones del sistema.

Este trabajo está basado en algoritmos de descenso por gradiente, entre los que se encuentran: Least Mean Square (LMS), Recursive Least Square (RLS), de Retro-propagación (Backpropagation: BP).

2.2.1 Algoritmo Least Mean Square (LMS).

Es el algoritmo de adaptación más utilizado en sistemas adaptables, fue propuesto por Widrow y Hoff en 1960 [21]. Sus principales características son:

- Baja complejidad. Su implementación en software requiere un bajo costo computacional, y su implementación en hardware es relativamente sencilla.
- No requiere estimaciones de gradiente "fuera-de-línea" o almacenamiento de datos.
- Baja velocidad de convergencia. Es relativamente lento para poder alcanzar un punto de adaptación del sistema.
- Alta sensibilidad al ruido aditivo. Maneja relaciones señal a ruido altas.
- Robustez. No importan las condiciones iniciales de los pesos, la solución del algoritmo siempre converge básicamente hacia el mismo valor.
- Estabilidad. La estabilidad de LMS depende principalmente del valor de la constante de adaptación, la cuál debe ser seleccionada cuidadosamente, debido a que si el valor es muy grande, el entrenamiento es rápido pero el sistema puede ser inestable, de otro modo, si es muy pequeño, el sistema es estable, pero el entrenamiento puede hacerse muy lento.
- Errores de Convergencia. Uno de los principales problemas de LMS es que en el proceso de adaptación, el algoritmo puede engancharse en un punto de mínimo local, y de ésta forma no pueda alcanzar el mínimo de la función de gradiente descendiente.

Debido a lo anterior, el algoritmo LMS es atractivo para aplicaciones de tiempo real que requieren filtros de alto orden operando en ambientes estacionarios o lentamente variantes en el tiempo y altas relaciones señal a

ruido. LMS ha encontrado su mayor campo de aplicación en el diseño de filtros adaptables.

En general, un filtro adaptable está basado en una estructura conocida como “Combinador lineal adaptable” (CLA). Mostrado en la Figura 2.2

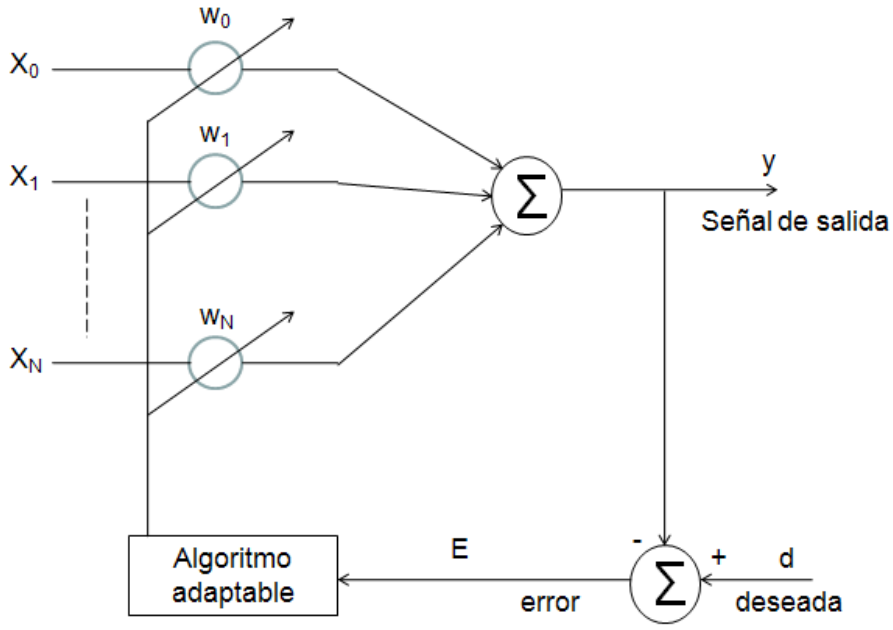

Figura 2.2 El Combinador Lineal Adaptable

La estructura CLA, consta de un conjunto de *N* entradas, dadas por el vector $\boldsymbol{x= [x_0 \ \ x_1 \ \dots \ x_N]^T}$, un vector de ganancias (también llamadas pesos) ajustables $\boldsymbol{w=[w_0 \ \ w_1 \ \dots \ w_N]^T}$, dos bloques sumadores, una señal de salida única “***y***” y una señal de entrada deseada “***d***”. La salida de la estructura CLA es una combinación lineal de los componentes de entrada para un conjunto fijo de pesos, entendiendo por combinación lineal una suma de los múltiplos de los elementos de un vector:

$$y = x_0 * w_0 + x_1 * w_1 + \dots + x_N * w_N = \sum_{i=1}^{n} x_i * w_i \quad (2.2)$$

Para la actualización de los pesos, es necesario calcular una señal de error mediante la diferencia entre la salida del sistema y la señal de entrada

deseada, y la cual es retro-alimentada hacia el algoritmo de adaptación quien se encarga de establecer el valor de cambio adecuado para la modificación de los pesos del CLA. Sin embargo, cuando los pesos se encuentran en proceso de adaptación no solo están en función de la respuesta del algoritmo, sino que también son función de los componentes de entrada, por lo tanto, la salida del CLA deja de ser una función lineal de la entrada. Es decir, su operación se vuelve no-lineal.

La actualización de los pesos, según el algoritmo LMS es:

$$W_{(n)} = W_{(n-1)} + 2\alpha e_{(n)} X_{(n)} \qquad (2.3)$$

Donde $\boldsymbol{W_{(n)}}$ es el vector de coeficientes de los pesos del sistema en la iteración actual, $\boldsymbol{W_{(n-1)}}$ es el vector de coeficientes de los pesos del sistema en la iteración anterior, $\boldsymbol{\alpha}$ es el factor que controla la estabilidad y velocidad de adaptación del algoritmo, conocida como *Constante de Adaptación*, $\boldsymbol{e_{(n)}}$ es el error dado por:

$$e_{(n)} = d(n) - W^T X_{(n)} \qquad (2.4)$$

Donde $\boldsymbol{d_{(n)}}$ representa la salida deseada del sistema, $\boldsymbol{W^T}$ es la traspuesta del vector de pesos y $\boldsymbol{X_{(n)}}$ es el vector de los coeficientes de entrada al sistema.

De acuerdo con (2.3), es evidente porque el algoritmo LMS es tan popular, ya que hace uso solamente de operaciones algebraicas básicas: suma y multiplicación, las cuales son fácilmente implementadas en Software o Hardware.

Una representación gráfica del proceso iterativo para un Filtro Adaptable por medio de su superficie de desempeño se muestra en la Figura 2.3.

La forma paraboloide de la gráfica de la Figura 2.3 se debe a que, para un Filtro Adaptable, el algoritmo es una función cuadrática de los parámetros $\boldsymbol{p_i}$. El sub-índice $\boldsymbol{i}$ denota el número de iteraciones que se han realizado, iniciando en $\boldsymbol{w_0}$ y acercándose lo más posible al punto de error

mínimo representado por $\boldsymbol{w_k}$. Cada paso en la iteración del sistema es realizado en dirección contraria al gradiente.

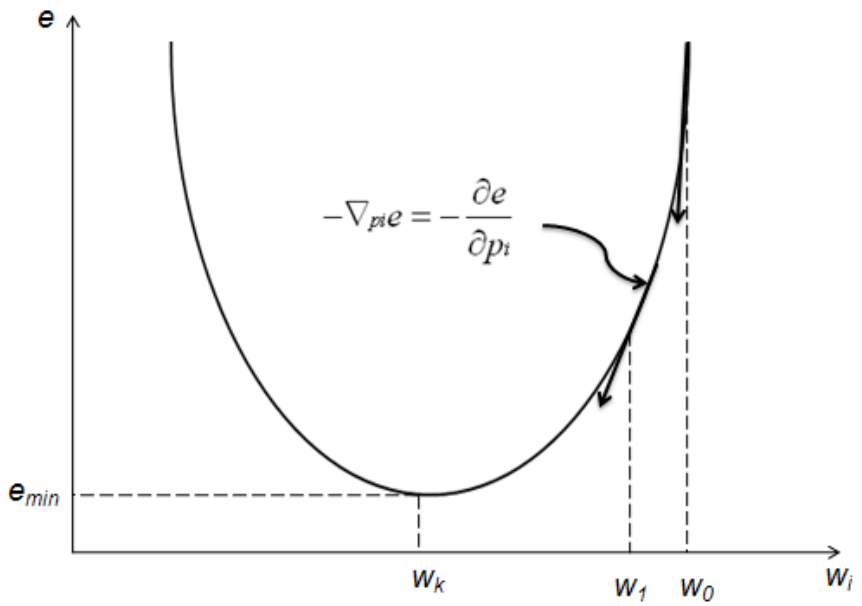

Figura 2.3 Comportamiento del algoritmo con respecto a un parámetro en un Filtro Adaptable.

La estimación del gradiente puede ser ruidosa, sin embargo, debido a que LMS estima el error de la muestra actual (iteración actual), y considerando que el proceso de adaptación no encuentra un mínimo en un paso, sino que requiere de un número determinado de iteraciones para converger hacia el punto mínimo de la superficie de desempeño, el ruido en el gradiente es filtrado. Es por esto que con el algoritmo LMS no se necesita tomar en cuenta las perturbaciones y promedios para estimar apropiadamente el gradiente en cada iteración; es el proceso iterativo el que mejora la estimación del gradiente. Entonces, se puede agregar algún tipo de ruido a la respuesta deseada y encontrar que los parámetros a adaptar permanecen sin cambios significativos. Esta robustez es una de las

principales razones de la implementación de LMS en problemas del "mundo real", en donde el ruido es "omnipresente".

2.2.2 Algoritmo Recursive Least Square (RLS)

Otro de los algoritmos de adaptación comúnmente usados en filtrado adaptable es el algoritmo Recursive Least Square (RLS) [22] [23], en el cual se minimiza directamente el valor cuadrático medio de la señal de error por medio de una inversión de matriz. Sus principales características son:

- Alta complejidad computacional. Debido a que la inversión de matrices es un trabajo computacionalmente muy caro.
- Alta velocidad de convergencia. Es más rápido al encontrar el mínimo de la función de desempeño comparado con LMS.
- Baja sensibilidad al ruido aditivo. Maneja relaciones señal a ruido bajas.

Debido a sus características, RLS es atractivo en aplicaciones en tiempo real donde el sistema debe operar en ambientes no estacionarios con relaciones señal a ruido bajas, en el caso de aplicaciones de filtrado, el orden de los filtros adaptados mediante RLS es también bajo.

La Figura 2.4 muestra la estructura de un filtro transversal adaptable, utilizado en el cálculo del algoritmo RLS.

Considerando la salida del filtro de respuesta a impulso infinito adaptable mostrado en la Figura 2.4, la cuál está dada por:

$$y_{(n)} = W^T X_{(n)} = X^T{}_{(n)} W \tag{2.5}$$

Donde ***X*** y ***W*** son vectores dados por:

$$X_{(n)} = \left[x_{(n)}, x_{(n-1)}, \ldots, x_{(n-N+1)} \right]^T \tag{2.6}$$

Y

$$W = \left[w_0, w_1, w_2, \ldots, w_{N-1} \right]^T \tag{2.7}$$

Entonces, el vector de coeficientes óptimo se calculará de manera tal que el error cuadrático medio sea minimizado. El error cuadrático medio esta dado por:

$$E\left[e^2{}_{(n)}\right] = E\left[\left(d_{(n)} - y_{(n)}\right)^2\right] \qquad (2.8)$$

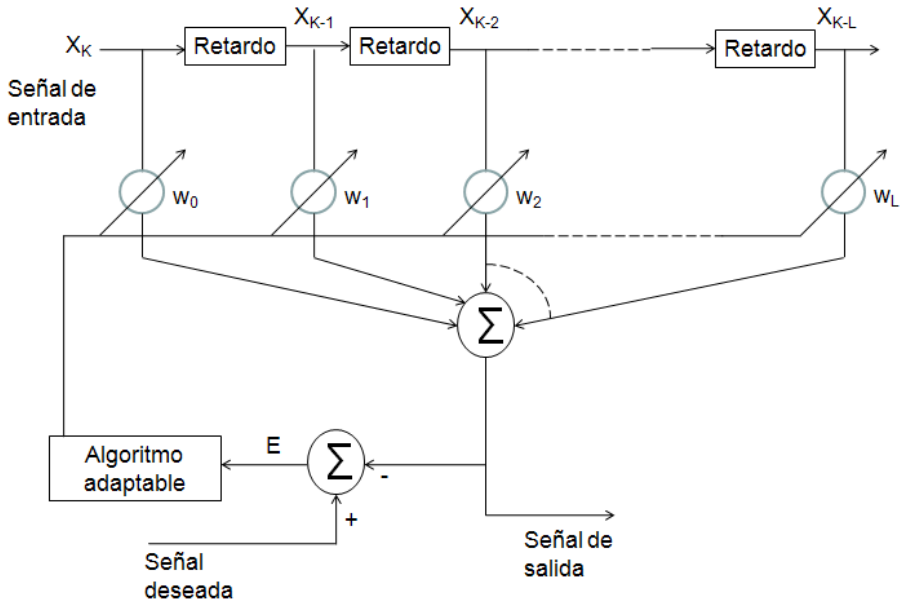

Figura 2.4 Filtro Transversal Adaptable.

Para minimizar el error cuadrático medio, se puede usar el principio de ortogonalidad en la estimación de mínimos cuadrados. Esto es, se selecciona el vector de coeficientes de tal forma que el error de salida llegue a ser ortogonal al vector de entrada, de tal forma que:

$$E\left[X_{(n)}\left(d_{(n)} - X^T{}_{(n)}W\right)\right] = 0 \qquad (2.9)$$

De donde:

$$E\left[X_{(n)}X^T{}_{(n)}W\right] = E\left[X_{(n)}d_{(n)}\right] \qquad (2.10)$$

Si se asume que el vector de coeficientes es no correlacionado con el vector de entrada, entonces:

$$E\left[X_{(n)}X^{T}{}_{(n)}\right]W = E\left[X_{(n)}d_{(n)}\right] \qquad (2.11)$$

Lo cual puede re-escribirse como:

$$RW = P \qquad (2.12)$$

Donde **P** es el vector de correlación entre la señal deseada $\boldsymbol{d_{(n)}}$ y el vector de entradas $\boldsymbol{X_{(n)}}$ y **R** es la matriz de auto-correlación de la señal de entrada.

Por lo tanto, la actualización del vector de pesos ***W*** está dado por:

$$W_{(n)} = W_{(n-1)} + \alpha R^{-1} e_{(n)} X_{(n)} \qquad (2.13)$$

Lo cual se conoce como Algoritmo Recursive Least Square (RLS), donde la matriz de auto-correlación ***R*** es estimada "en-línea". A diferencia del algoritmo LMS, el algoritmo RLS utiliza un conjunto de pesos que es siempre el óptimo de los datos dados al sistema.

Como ya se mencionó, el costo computacional del algoritmo RLS es alto, sin embargo, al hacer uso de información de segundo orden (información de la función de correlación) para actualizar los pesos la convergencia es mucho más rápida y los valores finales más precisos. Por otra parte, el algoritmo RLS tiene problemas en ambientes no-estacionarios, debido a que la ganancia **α** tiende a cero cuando el algoritmo converge. Si las propiedades de los datos cambian, el algoritmo no puede actualizar los pesos, porque **α** es cero ó muy pequeño. Es por esto que Haykin [24] propone aplicar el algoritmo RLS con una ventana con decaimiento exponencial en la estimación de **α**.

2.2.3 Algoritmo de Retro-Propagación (Backpropagation: BP)

El algoritmo de Retro-propagación es un algoritmo de gradiente descendiente utilizado para entrenar sistemas con varias capas ocultas, conocido como *perceptrón*. El *Perceptrón multicapas* consiste, principalmente de tres capas:

- Capa de Entrada. Cuyo número de nodos coincide con el número de elementos del patrón de entrada.

- Capa Oculta. Consiste de una o varias capas, las cuales se encuentran entre la Capa de Entrada y la Capa de Salida.
- Capa de Salida. Su número de nodos depende de la aplicación en particular.

En la Figura 2.5 se muestra la estructura de un perceptrón multicapas, en el que existen *N* nodos en la capa de entrada, correspondiente a las *N* entradas del sistema, la capa oculta se muestra de manera generalizada, considerando que esta capa puede contener más de una capa interna, la capa de salida tiene *K* nodos, cada uno para conectar las *K* señales de salida.

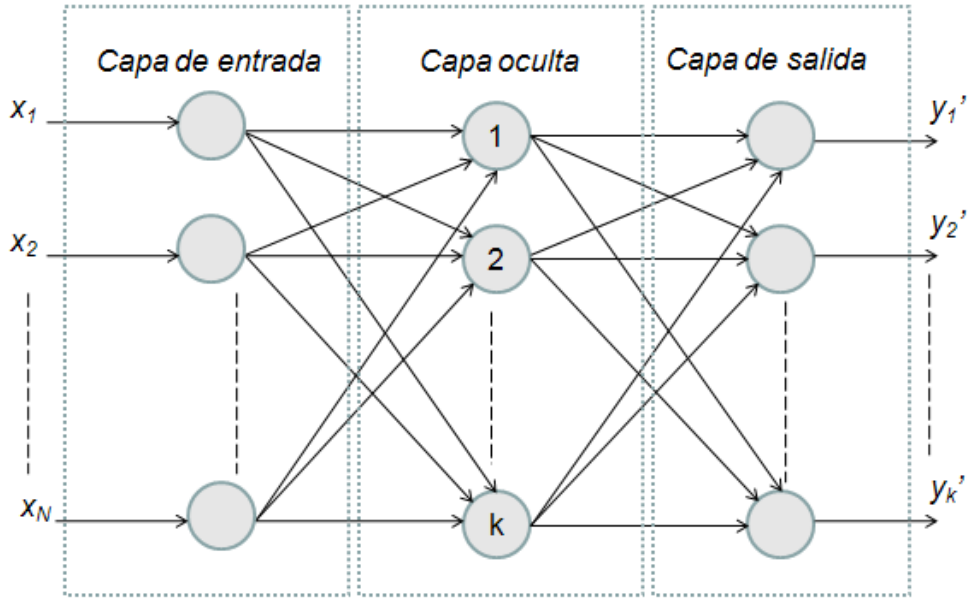

Figura 2.5. Perceptrón Multicapas.

Por ejemplo en un perceptrón multicapas aplicado al reconocimiento de dígitos manuscritos, el número de nodos de la capa de entrada es igual al número de pixeles usados para representar los dígitos, y el número de nodos de la capa de salida debe ser 10, cada uno de los cuales coincide con uno de

los posibles resultados de la red; sin embargo, no existe un criterio específico para determinar el número de nodos en las capas ocultas. Por lo tanto, el número de nodos de las capas de entrada y salida está determinado automáticamente por la aplicación, mientras que el número de capas ocultas así como el número de nodos de cada capa oculta debe decidirse a partir de varias pruebas.

El perceptrón multicapa es entrenado con "*aprendizaje por corrección de error*", lo cual significa que la respuesta deseada del sistema debe ser conocida, tal es el caso de problemas de reconocimiento de patrones, debido a que los datos de entrada están etiquetados, esto es, se conoce qué dato pertenece a qué clase. Para el entrenamiento del perceptrón multicapa, se puede hacer uso de la Regla Delta, la cual esta definida como:

$$w_i(n+1) = w_i(n) + \alpha e_p(n) x_{ip}(n) f'(net_p(n)) \qquad (2.14)$$

Donde $\boldsymbol{w_i}$ es el i-ésimo peso que se va a adaptar, $\boldsymbol{\alpha}$ es la constante de adaptación, $\boldsymbol{e_p}$ es el error del sistema, ***f ' (.)*** representa la derivada parcial del argumento y $\boldsymbol{net_p}$ es la función de activación del sistema.

Sin embargo, si se desea utilizar la Regla Delta para entrenar un perceptrón, se debe conocer cómo calcular el error en cada capa del perceptrón, lo cual requiere la habilidad de conocer la respuesta deseada para cada capa. Un error explícito no está disponible en las capas ocultas de un perceptrón multicapa. Esto es conocido como "*El problema de asignación de crédito*" y representa el tropiezo que terminó con la primera era conexionista en los años 70's. La solución a éste problema fue el descubrimiento de un método para el entrenamiento del perceptrón multicapa conocido como "*Retro-propagación*" (*Backpropagation*) ó Regla *Delta Generalizada*. Este método fue reinventado por muchos investigadores que trabajaron en el cálculo de sensibilidad a través de una red [25] [26] .

Para la derivación del algoritmo de Retro-propagación primero se explicará la base del algoritmo conocida como la Regla de la Cadena, la cual

indica básicamente como calcular la derivada parcial de una variable con respecto a otra cuando están relacionadas por medio de una función:

$$y = f(x) \tag{2.15}$$

El objetivo es, entonces, calcular:

$$\frac{\partial y}{\partial x} \tag{2.16}$$

Lo cual define la *sensibilidad* de ***y*** con respecto a ***x***. Asumiendo que ***f*** es diferenciable, entonces:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial f}\frac{\partial f}{\partial x} \tag{2.17}$$

La ecuación (2.17) calcula cuánto un cambio en ***x*** es reflejado en ***y,*** esto es, cuán sensible es ***y*** a los cambios en ***x***. En algoritmos adaptables, la aplicación de la regla de la cadena es transparente, ya que la idea obvia es distribuir los ajustes de acuerdo a la sensibilidad de la salida a cada peso. Si se desea minimizar el error de salida, se deben hacer cambios grandes en los pesos que afecten principalmente el valor de la salida, lo cual es medido por medio de la sensibilidad. Esto es lo que los algoritmos de gradiente descendiente hacen, tal como LMS.

Para modificar un peso, se propaga hacia atrás el error de salida hacia puntos intermedios en la topología, y es escalado conforme es propagado, de acuerdo a las funciones de transferencia elementales que se encuentren, tal como se muestra en la Figura 2.6

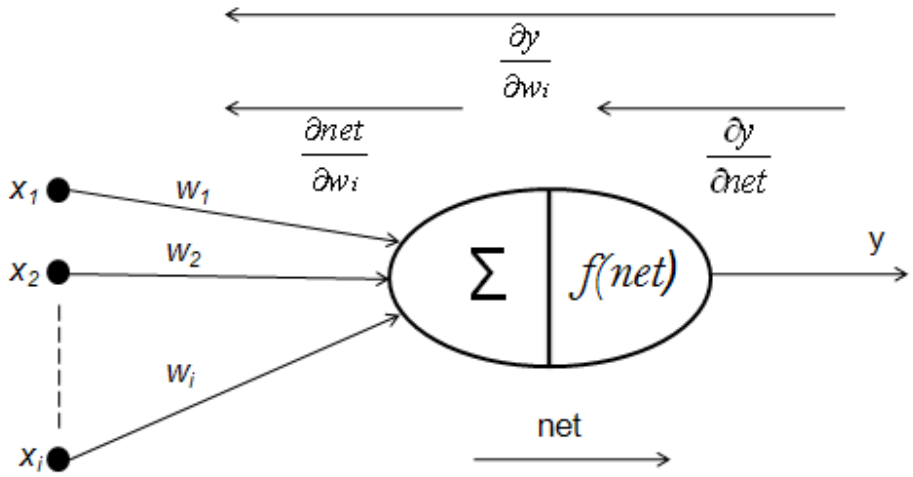

Figura 2.6 Ilustración del cálculo de la sensibilidad a través de un perceptrón no lineal.

La metodología mostrada en la Figura 2.6 es muy poderosa, debido a que no es necesario conocer explícitamente el error en los puntos intermedios, tales como ***net***. La Regla de la Cadena deriva automáticamente la contribución del error, calculando la sensibilidad desde el nodo de salida y propagándolo hacia atrás hasta llegar a todos los puntos ocultos.

Ahora, considerando un Perceptrón con una sola capa oculta, mostrado en la Figura 2.7

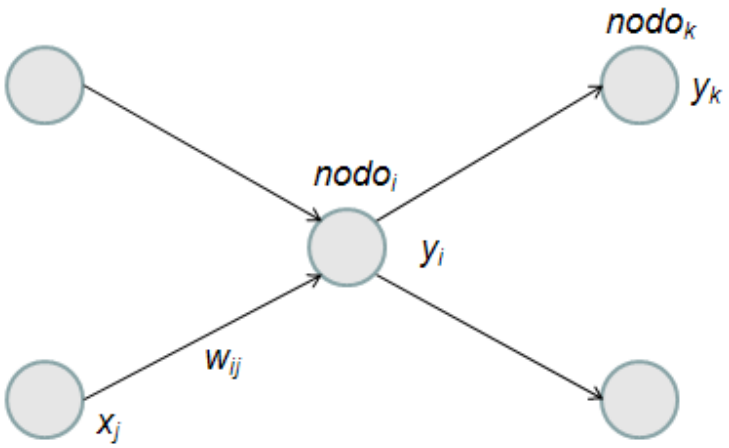

Figura 2.7. Red con una sola capa oculta.

El objetivo es adaptar los pesos que están conectados al ***nodo$_i$***. La respuesta deseada en ese punto no es conocida, debido a que dicho nodo no esta conectado al "mundo externo" (es un nodo interno). Para adaptar sus pesos, se aplica la siguiente metodología:

- En lugar de usar una respuesta deseada explícita para calcular el error en el ***nodo$_i$***, se usa un error derivado de la capa de salida.
- El error es el error de salida propagado y escalado.
- La sensibilidad es calculada automáticamente por la Regla de la Cadena.

Entonces, la actualización del peso usando el algoritmo De Retro-propagación es:

$$w_{ij}(n+1) = w_{ij}(n) + \alpha f'(net_i(n))\left(\sum_k e_k(n) f'(net_k(n)) w_{ki}(n)\right) y_j(n) \quad (2.18)$$

La ventaja del entrenamiento usando el algoritmo De Retro-propagación es que es un procedimiento sistemático (paso-a-paso) que puede ser aplicado independientemente de la topología de la red y las dimensiones de la entrada. Redes más grandes y complejas pueden requerir mayor tiempo adaptación, pero el algoritmo De Retro-propagación continúa siendo adecuado para su entrenamiento. En general BP puede ser aplicado a cualquier topología retro-alimentada, esto es porque el algoritmo actúa localmente en cada nodo oculto, no importando donde esté localizado en la topología.

Existe un flujo intrínseco para el algoritmo De Retro-propagación: el error local empieza a ser calculado en la salida de la red y se propaga hacia la entrada. Primero, un dato de muestra es enviado a través de la red para obtener una salida y calcular un error, tal como se muestra en la Figura 2.8. Entonces se calcula el error inyectado en la capa de salida y se refleja a la entrada del nodo de salida. Después, los errores en las capas anteriores pueden ser calculados por medio de (2.18), y así sucesivamente, hasta alcanzar la capa de entrada. Durante el proceso de retro-propagación de los

errores se usan los pesos “viejos” (pesos de las iteraciones anteriores). Una vez que todos los errores locales son hallados, se calcula la actualización para el peso de salida, y posteriormente la actualización para los pesos de las capas ocultas.

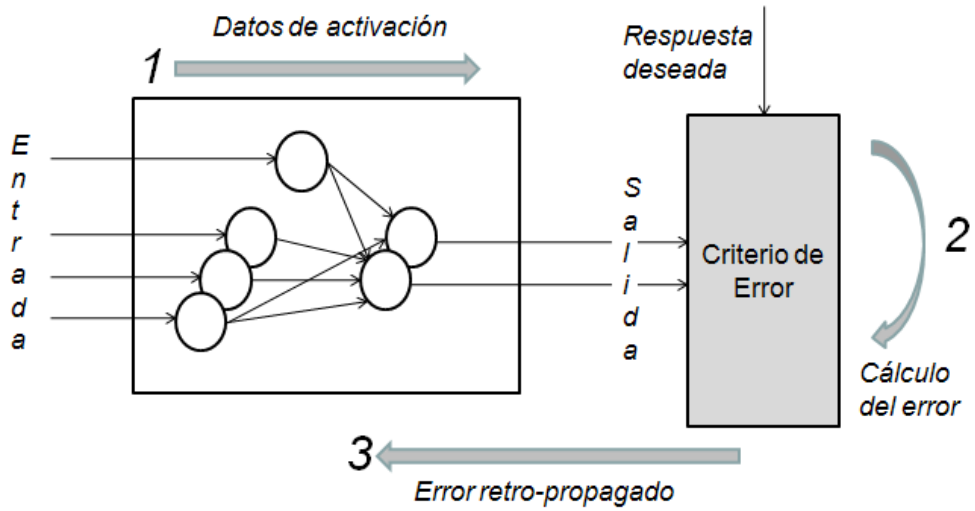

Figura 2.8. Flujo de operaciones del algoritmo De Retro-propagación.

Varias implementaciones del algoritmo BP han sido reportadas en la literatura: Eldredge y Hutchings [27] implementaron el algoritmo BP en un FPGA (Field Programmable Gate Array) Xilinx XC3090, dividiendo BP en tres etapas de ejecución secuencial: 1)”Feed-Fodward”, 2) algoritmo de Retro-propagación, 3) Actualización de pesos. Beuchat y Sánchez [28] diseñaron una Red de Computadoras Reconfigurable (RENCO) para implementar el algoritmo BP en cuatro FPGA’s Flex 10K130 y un micro-procesador Motorola 68360. En [29] Lysagh et. al. combinan ideas de diseño previamente reportadas para implementar el algoritmo en una Red Neuronal Artificial (ANN) para la función XOR. Un co-diseño Hardware/Software para la implementación del algoritmo BP es reportado en [30].

2.3 Algoritmo "Nonlinear Backpropagation"

Debido a que el algoritmo De Retro-propagación hace uso de derivadas parciales para el cálculo de la actualización de los pesos en una red con capas ocultas, su implementación en hardware, sea en Circuitos Integrados de Aplicación Específica (ASIC) o Dispositivos Programables (FPGA) no es tan sencilla, debido a que se requieren circuitos electrónicos muy complejos para poder realizar la derivada parcial de una función. En [31] Hertz et. al. proponen una nueva familia de algoritmos llamados Nonlinear Backpropagation (NLBP) a partir de una sencilla re-escritura del algoritmo BP estándar, (2.18). Se le denomina "No-lineal" porque la retro-propagación del error se hace a través de unidades no-lineales. Una característica de la familia de algoritmos no-lineales, es que para constantes de adaptación infinitesimalmente pequeñas, su comportamiento comienza a ser idéntico al del BP estándar. Esto significa que para constantes de adaptación pequeñas, el desempeño de NLBP es comparable a BP estándar, mientras que para constantes de adaptación grandes su desempeño es mucho mejor.

Considerando una red general retro-alimentada con N neuronas. La actividad de las neuronas esta denotado por V_i y el peso de la conexión de la neurona j a la neurona i esta denotado por w_i. Los valores de umbral están fijados por medio de la polarización de la neurona $i=0$. La salida de la unidad i en la red es:

$$v_i = g(h_i), \qquad h_i = \sum_j w_{ij} v_j + \xi_i \tag{2.19}$$

Donde g es la función de activación y ξ_i es la salida externa de la unidad. La ecuación (2.19) se aplica repetidamente para todas las neuronas hasta que el estado de actividad converja hacia un punto fijo.

El error de la red esta definido como:

$$\varepsilon_k = \varsigma_k - v_k \tag{2.20}$$

Donde ς_k es el valor deseado de las unidades de salida cuando la entrada es ξ_i *y* v_k es la salida actual. Para las unidades que no son de salida, $\varepsilon_i = 0$.

Podemos definir la función de *activación hacia atrás* como:

$$y_i = g\left(y_i + \frac{\eta_i}{\alpha_i}\left[\sum_k \frac{\eta_k}{\alpha_k}\left(y_k - v_k \right) w_{ki} + \varepsilon_i \right] \right) \qquad (2.21)$$

Donde $\boldsymbol{\alpha}$ y $\boldsymbol{\eta}$ son constantes equivalentes a la constante de adaptación en BP estándar y cuya selección de valores conlleva a obtener diferentes familias de algoritmos. Para unidades de salida, debido a que la sumatoria es vacía y el error es cero la ecuación anterior tiene una solución simple:

$$y_i = v_i \qquad (2.22)$$

El error está dado, en general, por:

$$\varepsilon_k = \begin{cases} \varsigma_k - v_k \; para\ unidades\ de\ salida \\ 0 \qquad para\ las\ demás\ unidades \end{cases} \qquad (2.23)$$

Entonces, Haciendo $\boldsymbol{\alpha_i}$ muy grande comparada con $\boldsymbol{\eta_i}$, el comportamiento de NLBP indistinguible del BP estándar, por lo tanto se debe esperar una alta estabilidad numérica, debido a que $\boldsymbol{y_i}$ debe tener un valor muy cercano a $\boldsymbol{v_i}$ y, la ecuación para la actualización del peso (2.24), requerirá una alta precisión.

$$\Delta w_{ij} = \alpha_i \left(y_i - v_i \right) \qquad (2.24)$$

De otra manera, para $\boldsymbol{\alpha}$ muy pequeña el algoritmo se aleja del comportamiento de gradiente descendiente. Por estas razones, es interesante considerar el rango en los que $\boldsymbol{\alpha}$ toma valores entre $\boldsymbol{\eta}$ y $\boldsymbol{1}$, esto es:

$$\eta_i \le \alpha_i \le 1 \qquad (2.25)$$

El límite donde $\boldsymbol{\alpha_i = \eta_i}$, es el más estable numéricamente, mientras que el otro límite, en donde $\boldsymbol{\alpha_i = 1}$, es el de mayor rapidez para hallar convergencia en el método del gradiente descendiente. Si se considera que la razón $\boldsymbol{\lambda = \eta_i / \alpha_i}$ es la misma para todas las capas ocultas del sistema, entonces las ecuaciones toman una forma muy simple:

$$y_i = g\left(y_i + \sum_k \left(y_k - v_k \right) w_{ki} + \lambda \varepsilon_i \right) \qquad (2.26)$$

Y finalmente:

$$\Delta w_{ij} = \alpha_i \left(y_i - v_i \right) v_j \qquad (2.27)$$

Por otro lado, si se considera una arquitectura de red con una sola capa oculta, las ecuaciones para la actualización de los pesos toman una forma diferente al hacer uso del método de *Representaciones Internas*, presentado por Krogh et. al. [32]. Al aplicar este método, el cambio para el peso de la unidad oculta ***j*** a la unidad de salida ***i*** está dado por:

$$\Delta w_{ij} = \eta \left[\varsigma_i - g\left(\sum_k w_{ik} v_k \right) \right] v_j \qquad (2.28)$$

En donde el error de retro-propagación está dado por (2.20), pero en donde:

$$v_i = g\left(\sum_j w_{ij} v_j \right) \qquad (2.29)$$

Donde $\boldsymbol{v_i}$ es obtenido por medio de la propagación de los objetivos específicos de las capas ocultas hacia los pesos de salida.

CAPÍTULO 3. MODELADO EN MATLAB.

3.1 Introducción.

El uso de herramientas de simulación a nivel comportamental permite conocer de una manera rápida y clara la respuesta de sistemas complejos. Normalmente, para modelar un sistema a nivel comportamental, se divide el sistema complejo dentro de bloques de construcción elementales y cada bloque se modela de acuerdo a su función de transferencia. Estos bloques son vistos siempre como una caja negra, es decir un elemento del cual solo se conocen sus entradas, salidas y la función que las relaciona, pero no se sabe que existe dentro de la caja y lo cual permite llevar a cabo dicha función.

Un modelado adecuado conlleva a la simulación individual de cada bloque y modificarlo, si es necesario, hasta obtener una respuesta idealizada; posteriormente se pueden unir bloques pequeños para formar bloques más grandes y complejos, los cuales se entienden como cajas negras conteniendo cajas negras. En la figura 3.1 se representa la construcción de bloques a partir de bloques más pequeños.

Finalmente, se unen los bloques complejos hasta tener un modelado completo del sistema. Este estilo de modelado ofrece al diseñador la posibilidad de detectar problemas a nivel arquitectura y resolverlos antes de llevar su sistema a niveles de diseño más bajos, como la implementación en hardware del sistema, sea en circuitos dedicados (ASIC) o en dispositivos de hardware reconfigurable (FPGA). En electrónica existen varias herramientas de simulación comportamental: Verilog, VHDL, MATLAB. Cada una con sus propias bondades y limitaciones.

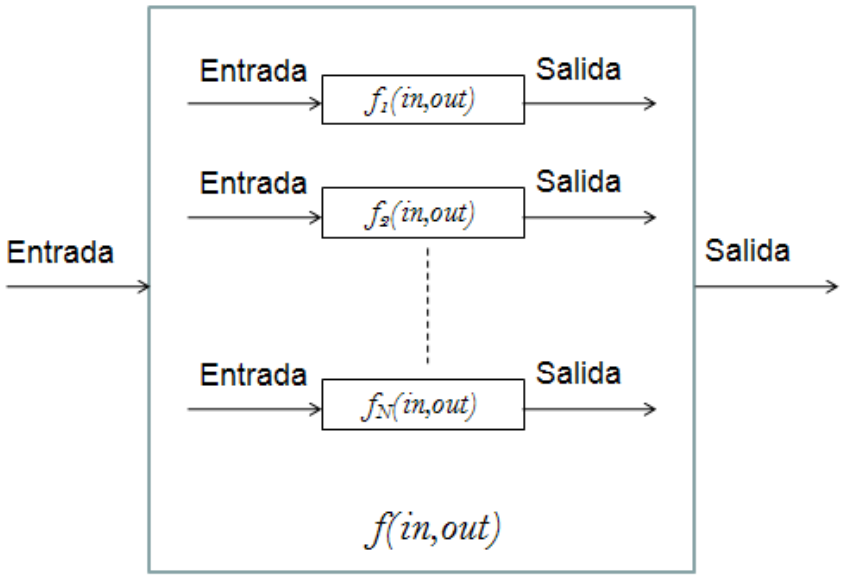

Figura 3.1 Encapsulado de bloques básicos para generar bloques más complejos en modelado comportamental.

3.2 Arquitectura del Controlador Difuso con el Algoritmo "Nonlinear Backpropagation".

El controlador difuso que va a ser adaptado mediante el Algoritmo "Nonlinear Backpropagation" tiene las siguientes características:

- Tipo Takagi-Sugeno-Kang de orden cero.
- Dos entradas.
- Una salida.
- 3 Funciones de membresía de tipo Triangular-Trapezoidal por entrada.
- El grado de traslape de las funciones de membresía es dos.
- 9 reglas.

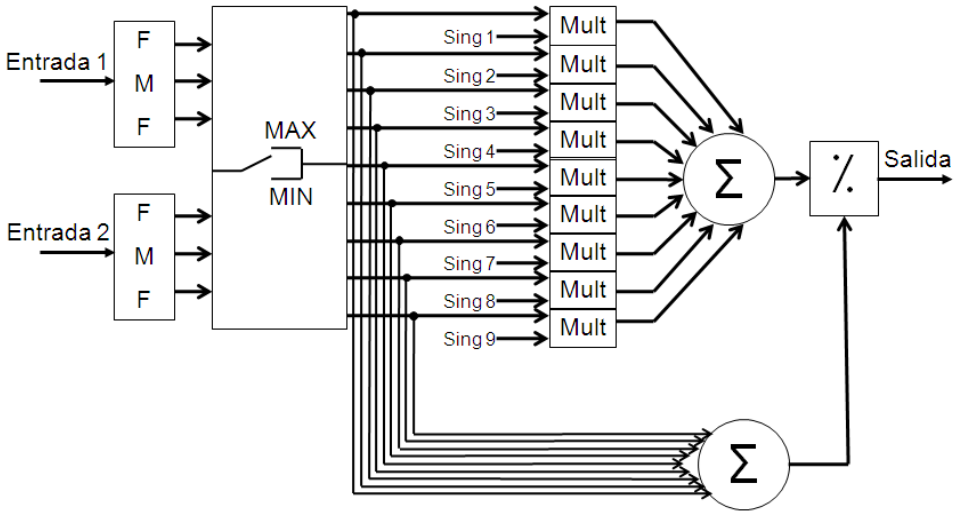

Figura 3.2 Estructura básica del controlador difuso.

La Figura 3.2 muestra el diagrama a bloques del controlador difuso utilizado para la simulación en MATLAB. Cada una de las dos entradas ingresan a los bloques donde se encuentran las funciones de membresía (FMF), cada bloque FMF contiene tres tipos de funciones de membresía: Tipo “S”, Tipo “Z” y Tipo Triangular, generando así tres salidas, cada una conteniendo el grado de pertenencia del valor de entrada a una de las tres funciones. Los tres grados de membresía por cada entrada pasan entonces al bloque de inferencia, en donde es posible seleccionar el tipo de operador a utilizar: a)MAX, ó b)MIN. Las reglas se forman mediante todas las combinaciones posibles de los conjuntos difusos de entrada, y debido a que cada conjunto contiene tres elementos, entonces el número de reglas es nueve (3 x 3). La salida de las nueve reglas se utiliza para la defuzzyficación empleando el método de Centro de Gravedad: por un lado se realiza la sumatoria de todas las reglas, en paralelo a esto se hace la multiplicación de cada regla por el valor del singleton correspondiente, el resultado de las nueve multiplicaciones también es sumado. Finalmente se realiza el cociente entre las dos sumatorias y se obtiene la salida del controlador difuso.

En cuanto al algoritmo de adaptación, se implementó el algoritmo "Nonlinear Backpropagation" (NLBP). Como ya se mencionó en el capítulo 2, este algoritmo es usado para adaptar sistemas con varias capas ocultas mediante la propagación hacia atrás del error calculado desde la capa de salida, sin embargo, en el presente trabajo la meta es adaptar solamente el valor de los singleton, por lo cual, el sistema puede verse como un sistema con una sola capa oculta, tal como se muestra en la Figura 3.3 y entonces, haciendo uso de *Representaciones Internas*, el algoritmo está dado por (2.28).

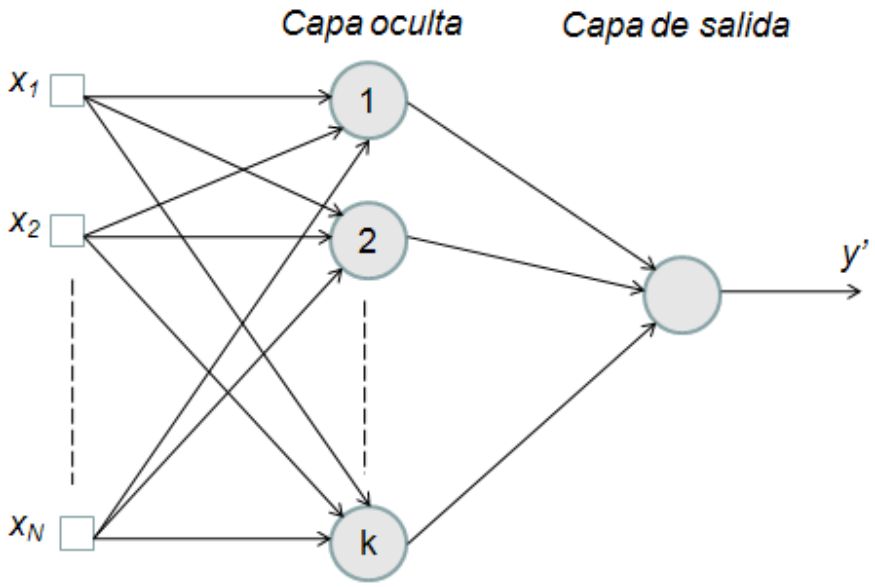

Figura 3.3 Sistema adaptable con una sola capa oculta.

En el diagrama de la Figura 3.3 la capa oculta esta representada por los singleton del controlador difuso, y la capa de salida está representando al método de defuzzyficación. Por lo tanto, el error necesario para el cálculo del algoritmo, es calculado directamente de la salida del controlador y se propaga hacia atrás hasta la capa oculta de los singleton para ajustar sus valores de acuerdo con una señal deseada.

Debido a la naturaleza misma del algoritmo de Retro-propagación, el error obtenido en la capa de salida, se propaga hacia atrás para justar el

valor de los singleton, después se vuelve a propagar hacia la capa oculta anterior y se ajustan tanto la base de reglas como el motor de inferencia, finalmente, se puede propagar hacia la primera capa oculta y modificar las funciones de membresía; de esta forma, se tendría un controlador difuso totalmente adaptable, en el que las correcciones de todos sus parámetros conlleva a una convergencia más rápida y estable.

3.2 Generación de las funciones de membresía.

La parte de "fuzzyficación" en un controlador difuso, es realizada por las funciones de membresía, las cuales convierten cada dato de entrada en grados de membresía.

Las funciones de membresía generadas en MATLAB tienen forma Triangular-Trapezoidal y están distribuidas uniformemente sobre todo el discurso de entrada, el cual está delimitado por los valores de voltaje de alimentación positiva y negativa propios de la implementación electrónica del circuito (-1.65V, 1.65V). Así mismo, el valor pico de la función (100μA), representa la corriente máxima con la que opera el circuito, es decir, el "1" difuso del sistema.

Cada función de membresía fue generada mediante la instrucción:

```
trapmf(x,[a b c d])
```

Donde *x* es el vector sobre el cual actúa la función, y *a, b, c, d* son escalares que indican los puntos de corte de la función de membresía tal como se muestra en la Figura 3.4

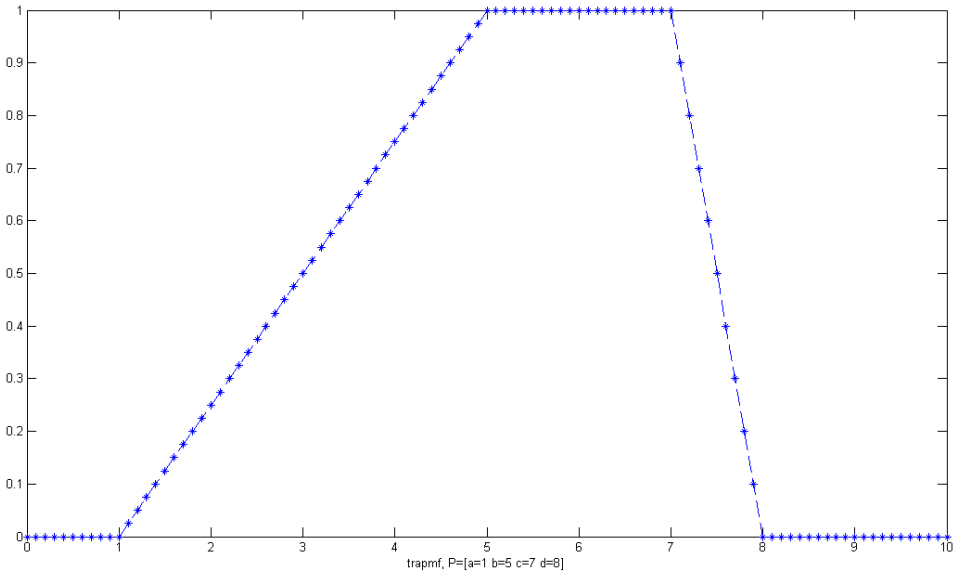

Figura 3.4. Función de membresía generada con la instrucción *trapmf*.

En la Figura 3.4, el valor los argumentos de la función *trapmf* son:

- X es un vector que va desde 0 hasta 10 en incrementos de 0.1
- a=1
- b=5
- c=7
- d=8

Puede verse, entonces, que ***a*** indica el punto en donde la función empieza a subir, ***b*** es el punto en donde la función alcanza el máximo valor, ***c*** es el valor en el cual la función empieza a descender y ***d*** es el valor en el que la función regresa a su punto mínimo.

Se utilizaron tres instrucciones *trapmf* para generar las tres funciones de membresía deseadas. El resultado se presenta en la Figura 3.5

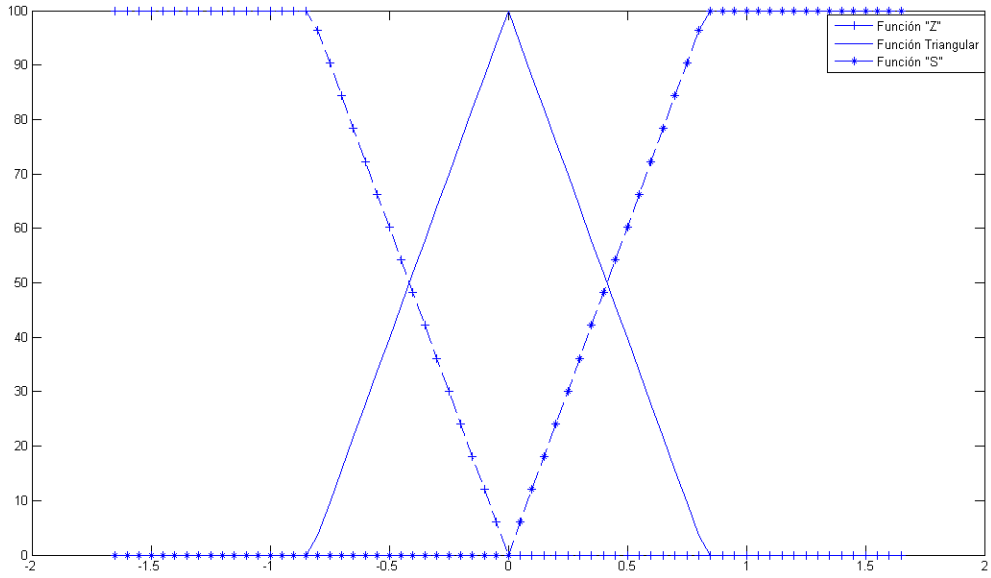

Figura 3.5 Funciones de membresía tipo “S”, “Z” y Triangular generadas en MATLAB.

3.3 Base de Reglas.

Como ya se mencionó anteriormente, es posible elegir el tipo de operador que se desea utilizar en el motor de inferencia. Esto es realizado mediante la comparación del valor de una variable a través de la instrucción condicional:

```
if expresión 1
    operación 1
else
    operación 2
end
```

En *expresión 1* se compara el valor de la variable, la cual es de naturaleza booleana, si el valor de la variable es “True”, entonces el programa realiza la inferencia difusa mediante el operador MIN, si el valor de la variable es “False” el programa utiliza el operador MAX para hacer la inferencia.

En el caso de que se elija el operador MIN como método de inferencia, se utiliza la instrucción:

```
min(A,B)
```

Esta instrucción regresa un “Array” con las mismas dimensiones de *A y B* con los elementos más pequeños de *A* ó *B*. Las dimensiones de *A y B* deben ser las mismas, ó, como en este caso, deben ser números escalares.

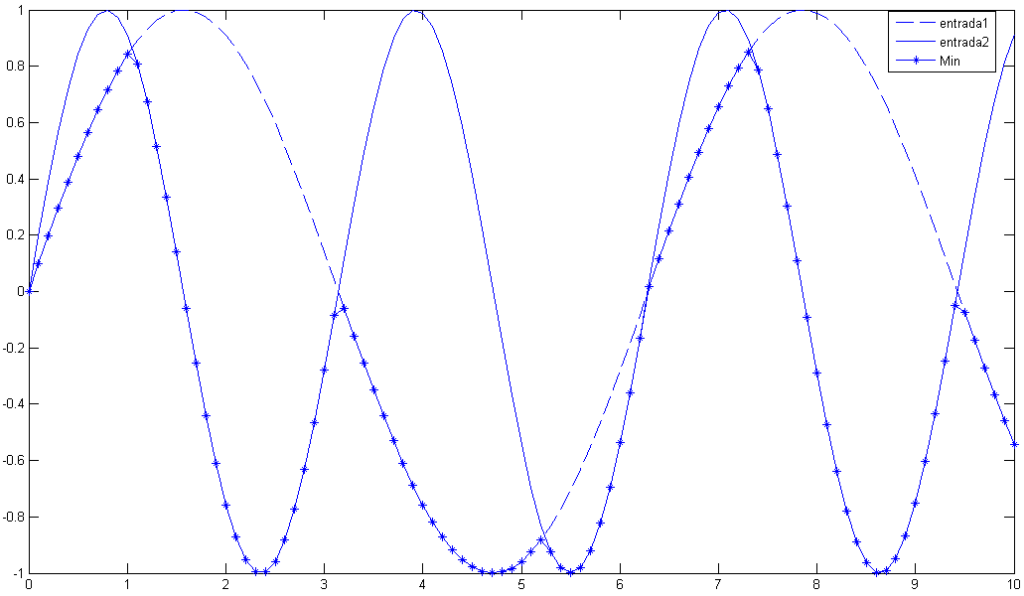

Figura 3.6. Simulación del operador MIN.

En la Figura 3.6 se muestra el resultado del operador MIN aplicado sobre dos señales de tipo sinusoidal. La línea punteada representa la entrada 1, La línea continua representa la entrada dos y la línea marcada con “*” es el valor que arroja el operador.

Si el operador elegido es el MAX, entonces se usa la instrucción:

```
max(A,B)
```

Al igual que la instrucción “*min”*, la instrucción *“max”* regresa un array, pero que ahora contiene al elemento más grande de *A y B*, cuando *A y B* son escalares.

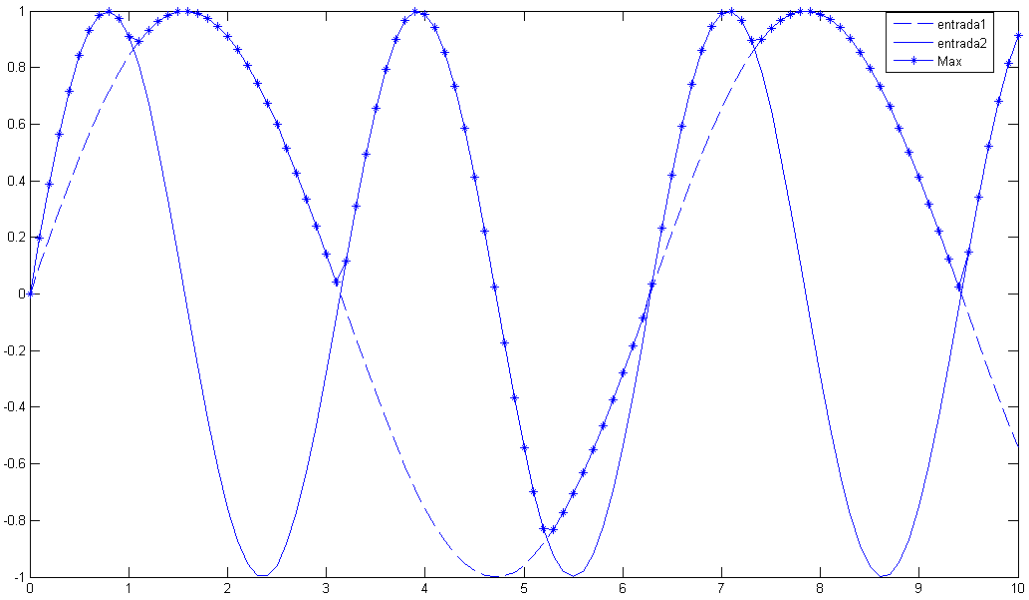

Figura 3.7. Simulación del operador MAX.

En la Figura 3.7 se muestra el resultado del operador MAX aplicado sobre dos señales de tipo sinusoidal. La línea punteada representa la entrada 1, La línea continua representa la entrada dos y la línea marcada con "*" es el valor que arroja el operador.

3.4 Defuzzyficación.

Como puede observarse en la Figura 3.2, después de que el motor de inferencia genera los grados de activación para cada una de las nueve reglas, el siguiente paso es la defuzzyficación por medio del método de centro de gravedad. El cual está dado por:

$$y' = \frac{\sum_{i=1}^{n} y_i * \mu_B(y_i)}{\sum_{i=1}^{n} \mu_B(y_i)} \qquad (3.1)$$

Donde ***y'*** es la salida del controlador difuso, y_i es el valor del *i-ésimo* singleton, *y* $\mu_B(y_i)$ es el grado de activación de la *i-ésima* regla.

Para calcular el numerador, primero multiplicamos cada singleton por el grado de activación de la regla, esto se hace por medio del segmento de código:

```
producto(:,k)=sing(:,k).*regla(:,k);
```

Donde `producto(:,k)` es el vector que contiene el valor del singleton escalado por el grado de activación de la regla correspondiente a la iteración ***k***; `sing(:,k)` es el vector que contiene el valor del singleton en la iteración ***k***; y `regla(:,k)` es el vector que contiene el valor del grado de activación de la regla en la iteración ***k***.

Una vez hecho el producto del singleton por el grado de activación de la regla, se realiza la suma de los nueve singleton escalados mediante el siguiente código:

```
numerador(k)=sum(producto(:,k));
```

Donde `numerador(k)` es el vector que contiene la suma de los singleton escalados en la iteración ***k*** *y* `sum(x)` es una instrucción de MATLAB que realiza la sumatoria de los elementos del vector **x**.

Para calcular el denominador, se realiza la sumatoria del valor de los nueve singleton a través del código:

```
denominador(k)=sum(regla(:,k));
```

donde `denominador(k)` es el vector que contiene la suma de los nueve grados de activación de las reglas.

Finalmente, se hace el cociente para obtener la salida del sistema:

```
salida(k)=numerador(k)/denominador(k);
```

donde `salida(k)` es el vector que contiene la salida defuzzyficada del sistema en la iteración ***k***.

3.5 Cambio de constante de adaptación.

Como se mencionó anteriormente, la constante de adaptación es el valor por el cual se multiplica el resultado del algoritmo para hacer que la convergencia hacia un punto estable sea más rápida o más lenta, pero al mismo tiempo, para garantizar que se llegue ó no a la estabilidad del algoritmo. Por lo mismo, en el presente trabajo se propone hacer un cambio en el valor de la constante de adaptación, para acelerar la convergencia, pero al mismo tiempo, garantizar la estabilidad.

Al iniciar la adaptación del controlador, el error es, en general, muy grande, dependiendo de las condiciones iniciales del sistema, por lo tanto se elige un valor grande para la constante de adaptación, lo cual hace que el algoritmo tenga una velocidad rápida de convergencia, conforme el controlador se va adaptando, el error disminuye, si la constante de adaptación continuara con un valor grande, entonces el sistema podría volverse inestable y nunca lograría converger hacia un punto fijo. Es por eso que al llegar a un valor dado, se hace un cambio en el valor de la constante de adaptación; ahora se elige un valor pequeño, para que las variaciones en el sistema sean más suaves y se garantice la convergencia del sistema en un punto de estabilidad. La constante de adaptación está dada por:

$$\alpha = \begin{cases} 0.1 if\,(error > 2.5) \\ 0.01 if\,(error < 2.5) \end{cases} \qquad (3.2)$$

Donde **α** es la constante de adaptación y ***error***, es el error del sistema dado por:

$$error = salida - deseada \qquad (3.3)$$

Donde ***salida*** es la salida actual del controlador difuso y ***deseada*** es la salida deseada del controlador.

El cambio de la constante de adaptación se lleva a cabo mediante el segmento de código:

```
if error(k)>2.5
     alpha=0.1;
elseif error(k)<=2.5
     alpha=0.01;
end
```

3.6 Implementación del Algoritmo.

Finalmente, se aplica el algoritmo "Nonlinear Backpropagation" para ajustar el valor de los singleton. Esto es realizado por el segmento de código:

```
singleton(:,k+1)=singleton(:,k)+(alpha*NLBP(k));
```

Donde `singleton(:,k+1)` es el vector que contiene el valor de los singleton para la iteración siguiente ***(k+1)***, `singleton(:,k)` es el vector de los valores de los singleton actuales, alpha es el valor de la constante de adaptación obtenido como se explico en la sección 3.5, y *NLBP(k)* es el vector que contiene el resultado de aplicar el algoritmo NLBP en la salida del controlador difuso y el cual está dado por:

```
NLBP(k)=(deseada(k)-numerador(k))*regla(:;k)/denominador(k)
```

Donde *deseada(k)* es el valor esperado en la salida del controlador difuso para la iteración ***k***, *numerador(k)* y *denominador(k)* son los valores calculados tal como se explica en la sección 3.4 y *regla(:;k)* es el vector que contiene los grados de activación de las reglas en la iteración ***k*** .

De esta manera, el valor del singleton para la siguiente iteración está dado por su valor actual, más el valor de cambio generado por NLBP y escalado por la constante de adaptación.

En el Capítulo cinco se presentan los resultados del modelo completo, tanto del controlador difuso, como del sistema adaptable completo.

CAPÍTULO 4. DISEÑO ELECTRÓNICO DE LOS BLOQUES DE CONSTRUCCIÓN.

4.1 Introducción.

En el diseño de circuitos electrónicos VLSI se deben satisfacer compromisos entre las diferentes características del circuito:

- Área: el consumo de área para aplicaciones VLSI es crítico, por lo cual se busca siempre reducirlo al máximo, principalmente para sistemas portátiles.
- Consumo de Potencia: en aplicaciones portátiles se busca también reducir al máximo el consumo de potencia, ya que de ello depende el tiempo de vida de la batería.
- Velocidad de operación: A pesar de que la velocidad de operación depende de la aplicación específica del circuito, generalmente siempre se trata de incrementar al máximo, aunque esto signifique un mayor consumo de área y de potencia.
- Fuente de alimentación: los niveles de voltaje necesarios para la operación de un circuito han disminuido gracias a la aplicación de técnicas conocidas como “de bajo voltaje”, como trabajar en la región de *sub*-umbral [33], utilizar compuertas flotantes [34] ó la utilización de fuentes flotantes para correr los niveles de voltaje necesarios para que un determinado circuito funcione.

Tomando en cuenta estos tópicos, se diseñaron los bloques de construcción que conforman el controlador difuso adaptable. El diseño se dividió en dos partes principales:

a) Controlador Difuso. En esta etapa se diseñaron, probaron y caracterizaron los circuitos que conforman el Controlador Difuso. Los bloques utilizados fueron:

- Función de membresía tipo “S”.
- Función de membresía tipo “Z”
- Función de membresía tipo Triangular.
- Operadores Máximo y Mínimo.
- Circuito Normalizador.
- Multiplicador de cuatro cuadrantes.
- Operación de suma

La Figura 4.1 muestra el diagrama a bloques del controlador difuso utilizado:

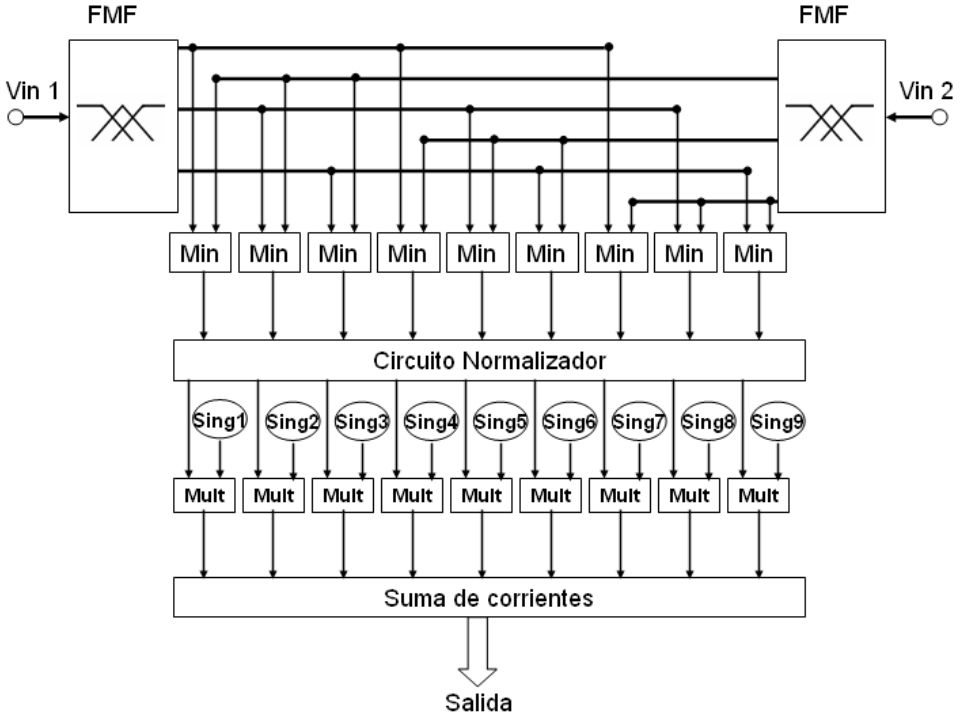

Figura 4.1 Diagrama del Controlador Difuso con dos entradas, una salida y nueve reglas.

Las entradas del controlador difuso son en modo voltaje, la función de los circuitos generadores de las funciones de membresía (FMF) es recibir el voltaje de entrada y entregar su grado de pertenencia a tres diferentes funciones de membresía: función tipo “S”, función tipo “Z” y función tipo Triangular, las cuales se encuentran distribuidas uniformemente en todo el discurso de entrada. Este circuito de función de membresía conforma el bloque fuzzyficador del controlador.

Las salidas de los dos circuitos de membresía se combinan de tal manera que forman grupos de tres reglas asociadas a cada singleton. A cada par de funciones de membresía se le aplica el operador **Min**. Esto conforma los bloques de base de reglas y motor de inferencia.

Para la implementación del bloque defuzzyficador existen tres principales topologías:

- Usando una extensión del circuito de agregación propuesto por Mead, mostrado en la Figura 4.2 [35]. Esta topología, mostrada en la Figura 4.2, hace uso de un concepto elegante de circuito para implementar una versión no lineal del método de centro de gravedad dando una salida en modo voltaje. Sin embargo, debido a la retroalimentación, su respuesta transitoria no es óptima; así mismo, debido a que la corriente aplicada al Amplificador de Transconductancia (OTA) es de gran señal, el rango de operación lineal y la respuesta transitoria son altamente no-homogéneas sobre el universo de discurso; finalmente, en esta topología se requiere incorporar Memorias y Convertidores Digital-Analógico para dar programabilidad a los singleton.

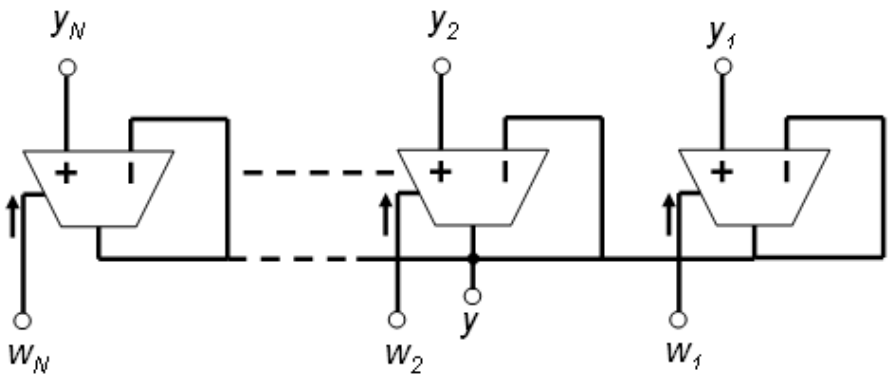

Figura 4.2 Circuito de defuzzyficación *"Follower- aggregation"*

- Usando circuitos divisores [36] [37]. Esta topología, mostrada en la Figura 4.3, permite una programación digital transparente de los singleton. Sin embargo requiere la aplicación de divisores en modo corriente con un amplio rango lineal, por lo cual es más sensible a errores de acoplamiento (mismatching), la respuesta transitoria es altamente dependiente del nivel de señal, además de que la compensación de errores en el divisor no es trivial, por lo cual se requiere del uso de divisores muy exactos. Finalmente otra desventaja de esta topología es la necesidad de hacer réplicas de corrientes para los puntos de suma.

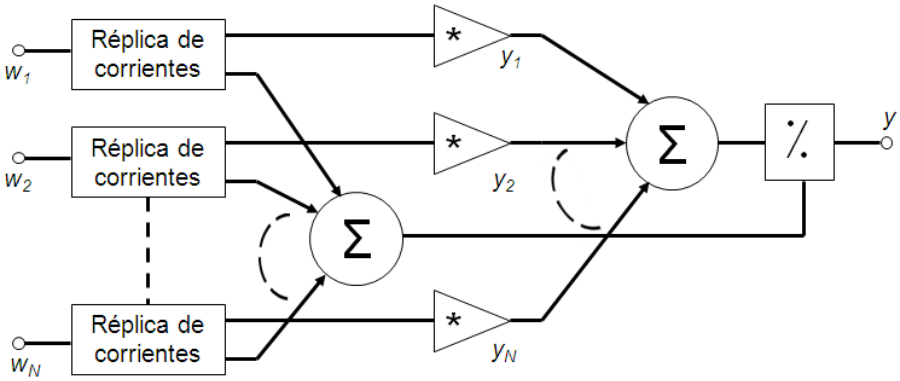

Figura 4.3 Circuito de defuzzyficación *utilizando divisor en modo corriente.*

- Usando la técnica de normalización. La Figura 4.4 muestra la topología con normalización, la cual esta basada en el circuito translineal BJT (Bipolar Junction Transistor) de Gilbert [38]. Dentro de las ventajas de esta topología son que no existen lazos globales de retroalimentación, por lo tanto, su respuesta dinámica es mucho mejor que las topologías anteriores. Al igual que la topología con divisores, la topología con normalizadores presenta una programación de los singleton transparente. Además, esta técnica tiene un esquema modular, debido a que se puede hacer N réplicas de la celda básica por cada par entrada-salida, y solamente se debe agregar un circuito común formado por un transistor y una fuente de corriente.

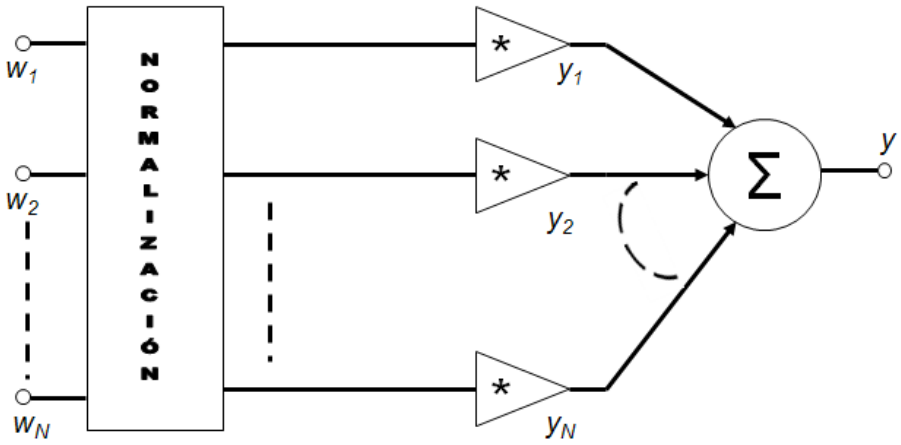

Figura 4.4 Circuito de defuzzyficación *utilizando la técnica de normalización.*

b) Algoritmo “Nonlinear Backpropagation”. En esta segunda etapa, fueron diseñados, probados y caracterizados todos los bloques de construcción necesarios para la implementación electrónica del Algoritmo. Dichos bloques fueron:

- Comparador de Corrientes.

- Multiplicador de cuatro cuadrantes.
- Punto de resta de Corrientes.
- Espejos PWL
- Generadores de Retardo

Adecuado a una implementación electrónica, el algoritmo “Nonlinear Backpropagation” está dado por:

$$\Delta s_i(t) = \alpha \left[\varsigma_i - g\left(\sum_k w_{ik} V_k \right) \right] V_j \qquad (4.1)$$

La imagen 4.5 muestra el diagrama a bloques de la implementación electrónica del Algoritmo “Nonlinear Backpropagation”.

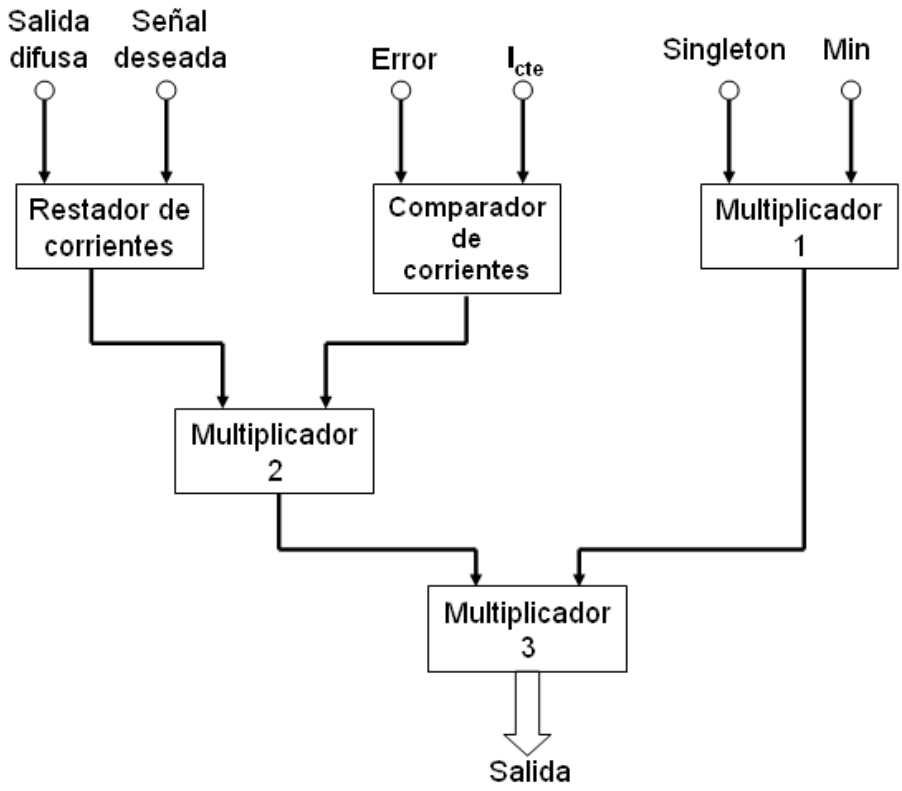

Figura 4.5 Diagrama a bloques de la implementación electrónica del algoritmo “Nonlinear Backpropagation”.

La función del circuito que implementa el algoritmo es generar la señal de adaptación para los singleton, por lo cual se requiere un bloque de este circuito para cada singleton. En este trabajo, el Controlador Difuso consta de nueve singleton, por lo cual se requieren nueve bloques del algoritmo.

El circuito recibe como entradas la señal de salida del sistema difuso, la señal deseada o esperada del controlador, una corriente constante para comparación, el valor del singleton correspondiente y la salida del operador **Min**. Mediante un punto de resta de corrientes, se calcula el error existente en el tiempo ***(t)*** el cual está dado por:

$$I_e(t) = I_{salida}(t) - I_{deseada}(t) \qquad (4.2)$$

Donde $I_e(t)$ es la corriente de error en el tiempo (t), $I_{salida}(t)$ es la corriente de salida del controlador difuso en el tiempo (t) e $I_{deseada}(t)$ es la corriente que se espera tenga el controlador en el mismo tiempo (t). Posteriormente, ésta señal de error es comparada con una corriente de referencia para hacer el cambio de valor de constante de adaptación: cuando el error sea mayor que un valor esperado entonces la constante de adaptación tomará un valor grande para disminuir rápidamente el error, conforme el sistema realiza las iteraciones, el error disminuye hasta ser menor que el valor esperado, es entonces cuando el bloque de comparación de corrientes cambia su valor, haciéndolo mas pequeño, lo cual significa que la constante de adaptación es más pequeña y con esto se asegura la convergencia del algoritmo hacia un punto estable. Finalmente, mediante los multiplicadores 1, 2 y 3 se realiza la multiplicación de todas las señales adecuadas para obtener la señal de cambio en la iteración actual.

Una vez calculado el cambio en el peso de cada singleton, el siguiente paso es llevar este cambio a un bloque en el cual el valor del singleton sea actualizado de acuerdo con este cambio. Este procedimiento es realizado por un circuito formado por espejos cuya característica de transferencia es de la forma Lineal a Trozos (PWL), un generador de retardos, también basado en espejos de corriente y un punto de suma de corrientes. El diagrama a bloques de este circuito es mostrado en la Figura 4.6.

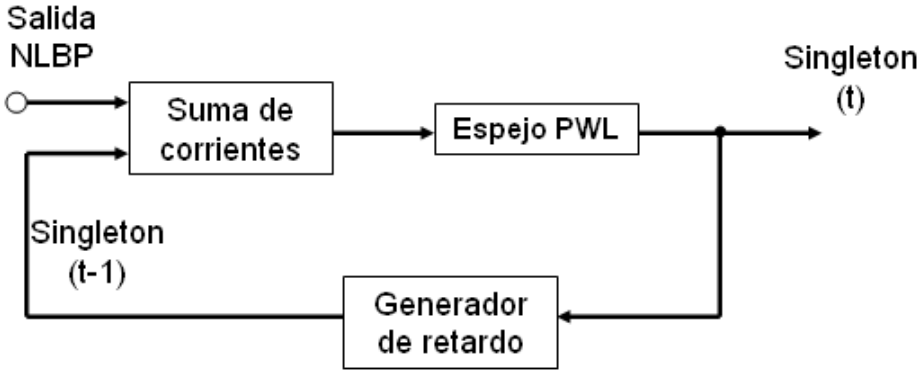

Figura 4.6. Actualización del valor del Singleton.

Este circuito recibe como única señal de entrada, la salida del bloque que calcula el algoritmo “Nonlinear Backpropagation” y retroalimenta el valor actual del singleton hacia un punto de suma de corrientes, pasando antes por un generador de retardo, lo cual permite hacer cada iteración en un tiempo específico. La función del espejo PWL es acotar la señal de salida entre los valores de “0” y “1” difusos (eléctricamente correspondientes a 0uA y 100uA) además de ofrecer la posibilidad de programar el valor inicial de los singleton mediante una etapa adicional de ganancia.

4.2 Funciones de Membresía.

Existen circuitos para generar funciones de membresía en diferentes modos (voltaje-voltaje, corriente-corriente y voltaje-corriente) y formas (triangular, trapezoidal, gaussiana, etc.)

Debido a que generalmente las señales de entradas a un Circuito Integrado son en modo voltaje y de que en el presente trabajo todo el procesamiento se realiza en modo corriente, se optó por elegir un circuito que reciba como señal de entrada un voltaje y a la salida entregue una señal en modo corriente. Se implementó un circuito por cada una de las formas “S”, “Z” y Triangular.

- Función de membresía tipo “S” y “Z”. Para la realización de las funciones de membresía de formas “S” y “Z” se utilizó un par diferencial, el cuál es el bloque de construcción más básico de la electrónica en general. Este circuito cumple la característica de manejar un voltaje como señal de entrada y entregar una corriente como señal de salida. El circuito esquemático del par diferencial se muestra en la Figura 4.7

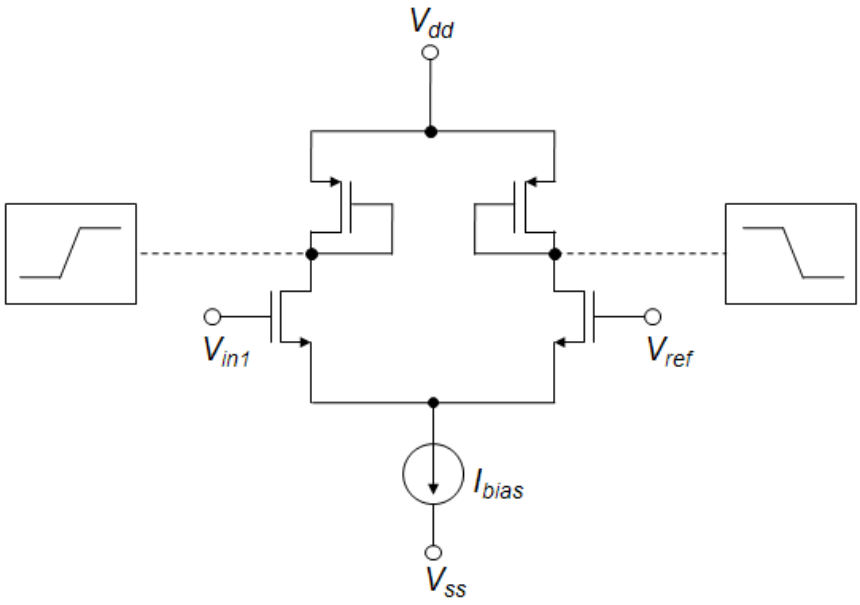

Figura 4.7. Circuito para generar las funciones de membresía tipo “S” y “Z”.

Cuando V_{in1} en la Figura 4.7 es menor que V_{ref} la corriente que fluye es cero en una rama del par diferencial, mientras que en la otra es igual a I_{bias}, conforme V_{in1} aumenta, la corriente empieza a balancearse en ambas ramas, aumentando el flujo en una rama y disminuyendo en la otra, considerando que la suma de las corrientes a través de ambas ramas siempre es igual a I_{bias}. Cuando V_{in1} es mayor que V_{ref} entonces toda la corriente fluye sobre la otra rama. Dependiendo de en que rama se tome la señal de salida, se puede obtener, ya sea la función "S" ó la función "Z", tal como se muestra en la Figura 4.7. Por razones de acoplamiento, en éste trabajo se utilizó un par diferencial para generar la función "S" y otro para la función "Z".

- Función de membresía tipo Triangular. Se han propuesto diferentes circuitos para la generación de funciones de membresía tipo Triangular, Ahmadi et. al. [39] utiliza aproximaciones PWL con fuentes de corriente, Ota y Wilamowski [40] usa una topología en la que, por cada función de membresía, se requieren dos pares diferenciales, así mismo, ambos pares diferenciales deben ser polarizados por fuentes de corriente idénticas. La figura 4.8 muestra el diagrama esquemático del circuito utilizado para la generación de la función de membresía tipo Triangular.

  Como se puede observar en la Figura 4.8, el circuito es capaz de generar los tres tipos de funciones de membresía, sin embargo, los puntos de quiebre de cada una de las corrientes de salida I_{o1}, I_{o2}, I_{o3}, están condicionados por los voltajes de referencia V_{ref1} y V_{ref2} lo cual impide una programación independiente de cada una de las señales, es por ésta razón que el circuito mostrado solo se utiliza para la generación de la función de membresía tipo Triangular.

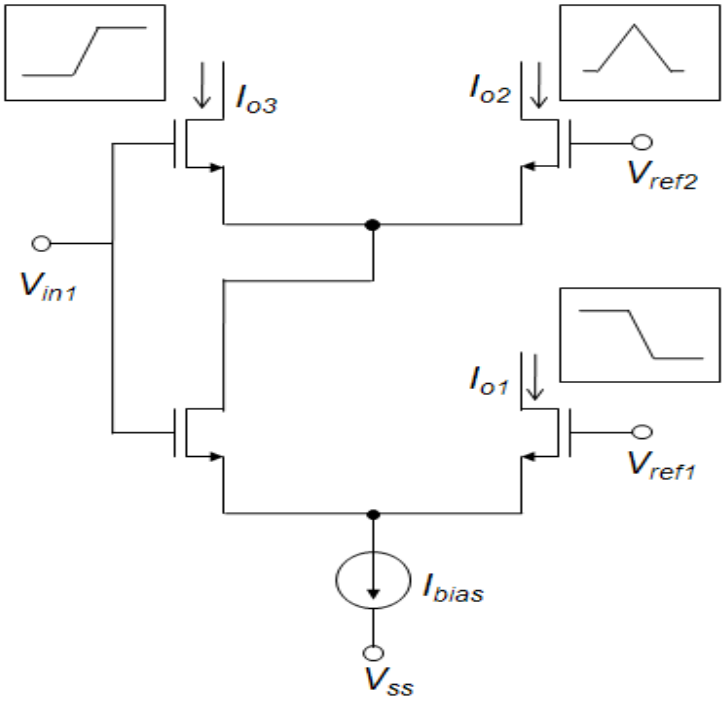

Figura 4.8. Circuito para generar la función de membresía tipo Triangular.

Los tres circuitos para generar las tres funciones de membresía necesarias fueron encapsulados en un sub-circuito [41] y simulados utilizando una tecnología de AMS 0.35um. La Tabla 4.1 muestra un resumen de las características del circuito al ser llevado al simulador:

Tabla 4.1 Características de los circuitos para las tres funciones de membresía.

	Valor	Unidades
$(W/L)_{Todos}$	7	---
I_{bias}	100	µA
V_{DD}	1.65	V
V_{SS}	-1.65	V
V_{in}	-1.65 – 1.65	V
I_o	0-100	µA

El rango de entrada para las funciones de membresía va desde -1.65 hasta 1.65V, lo cual es equivalente al valor de las fuentes de alimentación positiva y negativa, respectivamente. La salida está entre 0 y 100 µA, valores que equivalen al "0" y "1" difusos del sistema. La respuesta del circuito en el simulador es presentada en la Figura 4.9

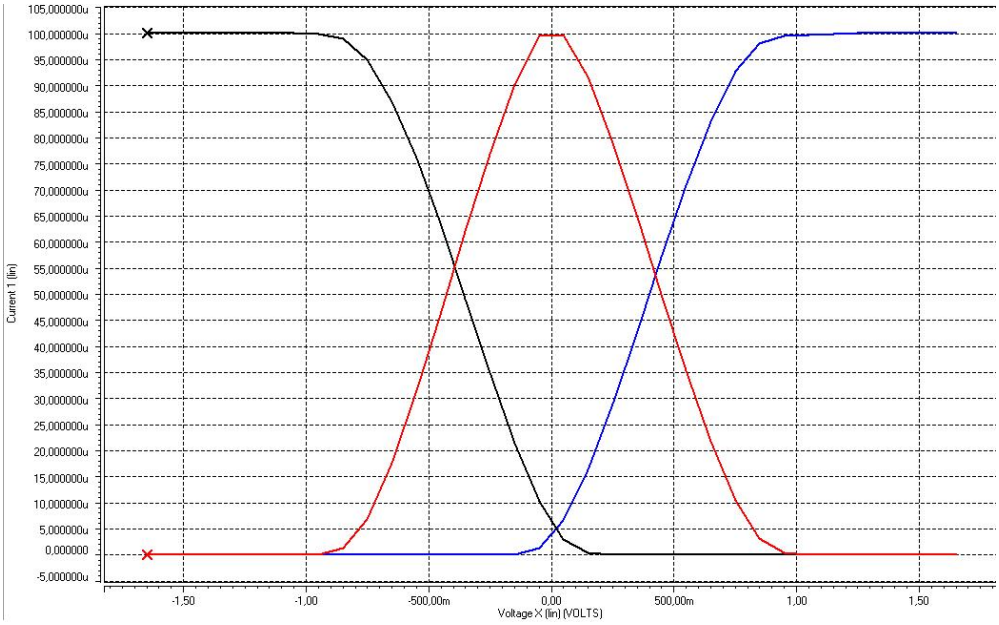

Figura 4.9. Simulación en SPICE de las funciones de membresía.

4.3 Operadores Max y Min.

El motor de inferencia de un controlador difuso está dado por las operaciones de unión e intersección de los conjuntos difusos[42]. Dichas operaciones son realizadas por los operadores ***Max*** y ***Min.***

4.3.1 Operador Max.

La Figura 4.10 muestra el circuito para implementar el operador Max, propuesto por Lazzaro en 1989 [43].

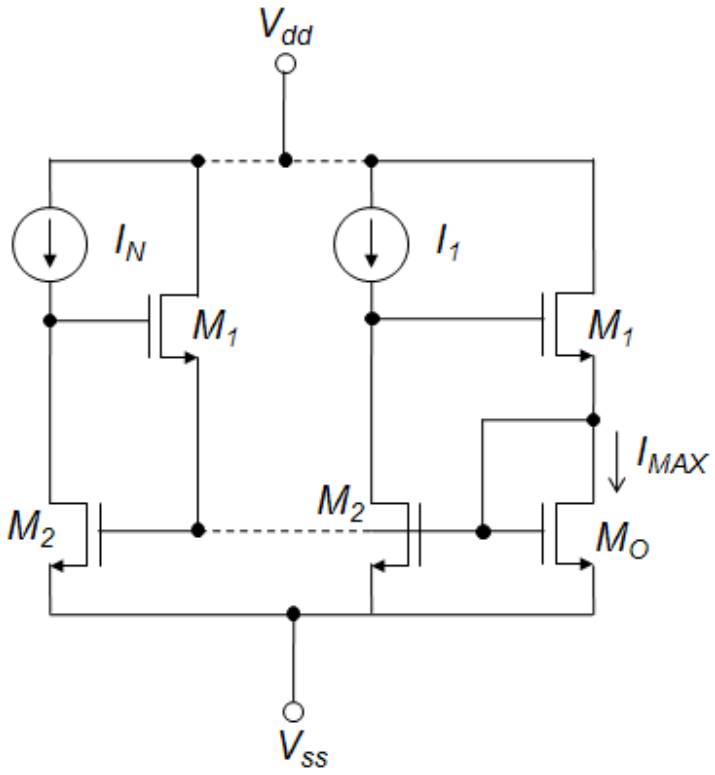

Figura 4.10. Circuito MAX de Lazzaro.

El circuito de la Figura 4.10 consta de *N* fuentes de voltaje controladas por corriente (Transistores M_1 y M_2) conectadas en paralelo. Considerando primero cada fuente de voltaje controlada por separado y asumiendo que todos los transistores trabajan en saturación, entonces el voltaje controlado es el voltaje en la fuente del transistor M_1 y su valor es proporcional a la raíz cuadrada de su fuente de corriente (esto es $I_1, \ldots, I_N$). Sin embargo, la fuente de los transistores M_1 está atada a un transistor común conectado como diodo M_O, el cual es de las mismas dimensiones de los transistores M_2 de las *N* celdas. De esta manera, todas las fuentes de voltaje controladas por corriente están conectadas en paralelo y compiten por imponer su propio voltaje en el nodo común. El que logra mantener su voltaje gana y los transistores M_1 de las demás celdas permanecen apagados. Así, la celda ganadora junto con el transistor de salida común M_O permanecen configurados como un espejo de corriente tipo Wilson, el cual replica la

corriente de la celda ganadora (esto es, la corriente de entrada máxima) en M_O. Como resultado, los transistores M_2 de las celdas perdedoras permanecen en la región de tríodo.

En 2003, Dualibe, et al. [44] proponen una versión modificada del circuito de Lazzaro, esta versión es mostrada en la Figura 4.11.

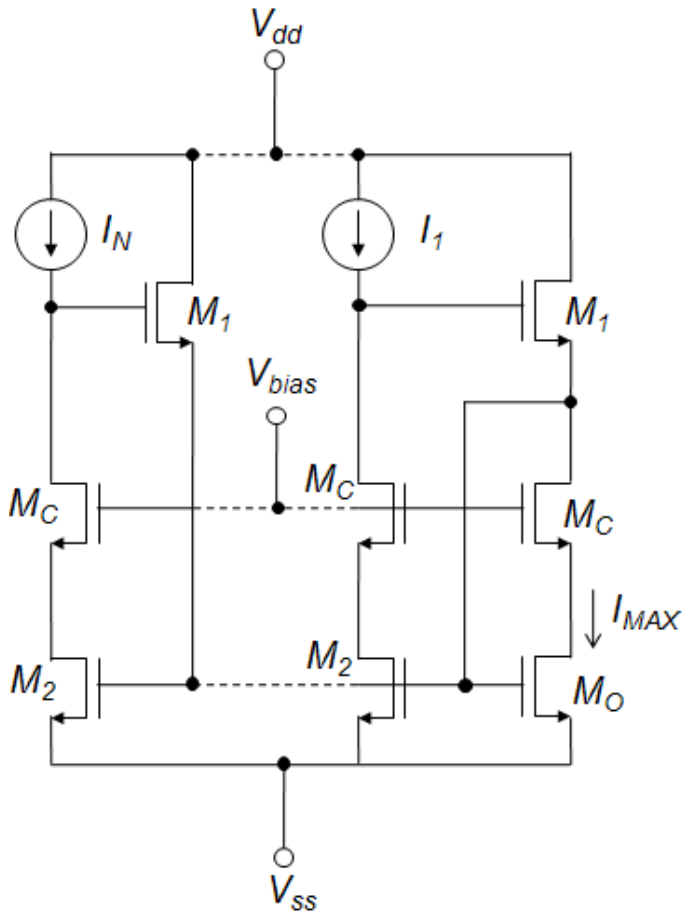

Figura 4.11. Circuito MAX modificado.

En el circuito de la Figura 4.11, los transistores M_2 han sido configurados como cascode por medio de los transistores M_C. Estos están polarizados por medio del voltaje V_{bias} con la finalidad de garantizar la saturación de los transistores M_2 y M_O. De esta manera, la corriente ganadora se refleja con mayor precisión hacia la salida. A medida que más celdas van siendo agregadas al circuito, la carga de salida en el nodo común

(el drenador de M_2) se incrementa debido a que mas capacitancias compuerta-fuente (C_{gs}) de los transistores M_2 están siendo conectadas en paralelo. Así, la velocidad del circuito puede ser influenciada considerablemente por el número de entradas. Para nuestro caso en particular, el número de entradas del circuito **Max** es dos, por lo tanto la carga es la mínima que se puede presentar y la velocidad de operación del circuito es la máxima posible.

La Tabla 4.2, muestra el resumen de las características del circuito MAX para la simulación en SPICE.

Tabla 4.2 Características del Circuito MAX.

	Valor	Unidades
$(W/L)_{Todos}$	15	---
V_{DD}	1.65	V
V_{SS}	-1.65	V
I_{in}	0 – 100	µA
I_o	0-100	µA

El circuito MAX simulado consta de dos celdas básicas y el transistor común de salida solamente, mostrado en la Figura 4.11. El rango de la corriente de entrada corresponde a los valores del “0” y “1” difusos, es decir va desde 0 hasta 100 µA, la salida del circuito, por lo tanto, tiene los mismos rangos. Se realizó una simulación en DC y otra en transitorio para probar el desempeño del circuito **Max**, los resultados son mostrados en las Figuras 4.12 y 4.13, respectivamente.

Para la simulación en DC, se barrieron las dos corrientes de entrada desde 0 hasta 100 µA, la primera con incrementos de 1 µA y la segunda con incrementos de 10 µA.

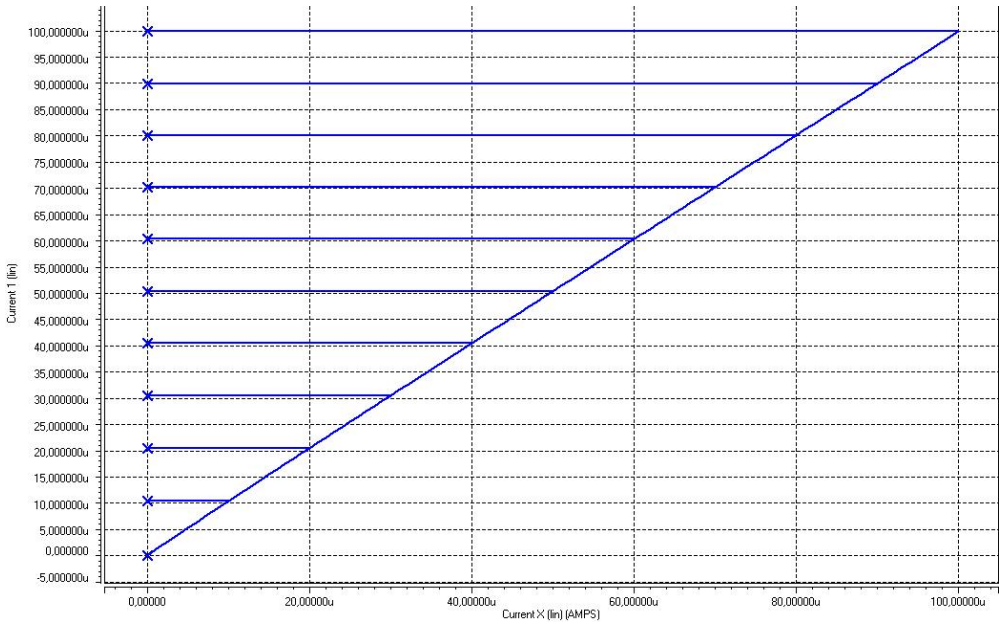

Figura 4.12. Respuesta en DC del Circuito MAX.

Para la simulación en transitorio, se emplearon dos señales sinusoidales centradas en 50 µA y con picos de 50 µA, la primera con una frecuencia de de 10MHz y la segunda con una frecuencia de 20MHz.

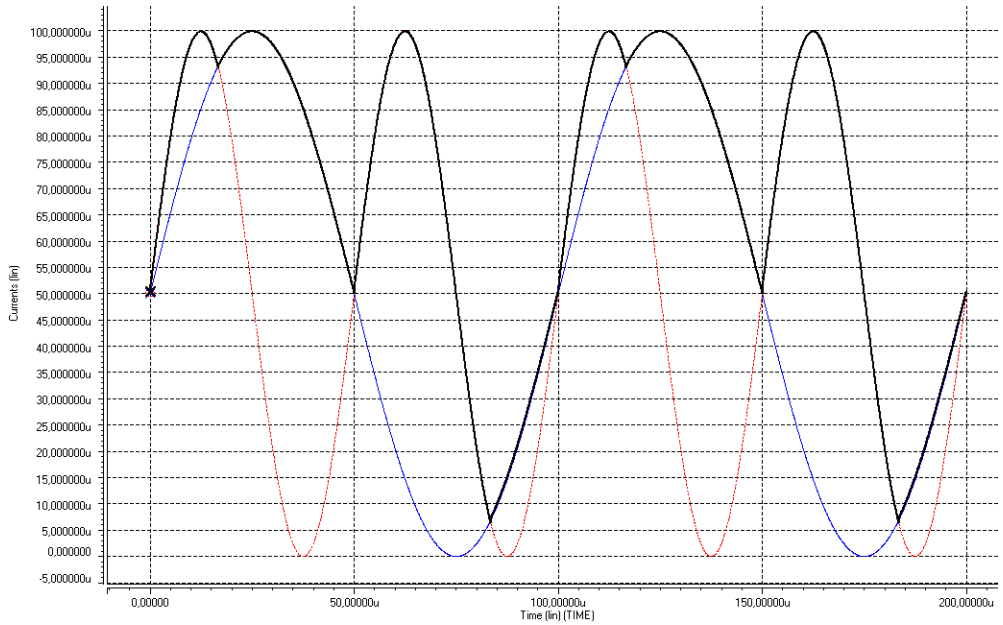

Figura 4.13. Respuesta en transitorio del circuito MAX.

4.3.2 Operador MIN.

La figura 4.14 muestra el diagrama esquemático del oeprador **MIN**, propuesto en [44].

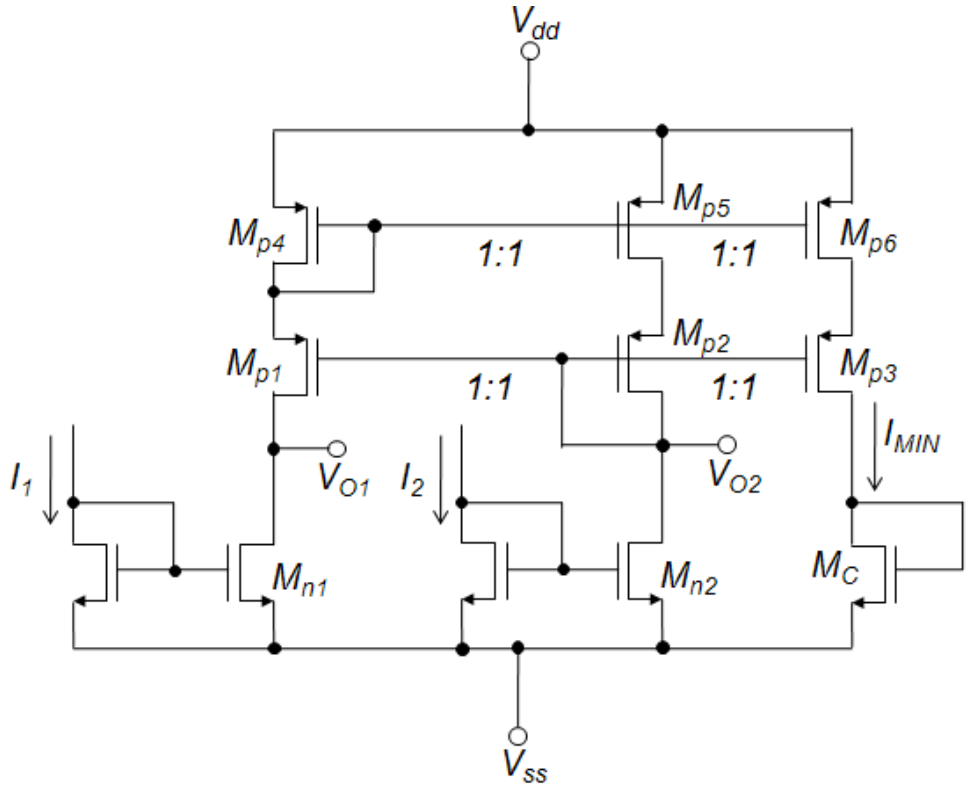

Figura 4.14. Diagrama del circuito MIN.

En este circuito, el grupo formado por los transistores M_{p4}, M_{p5}, M_{p6} tienen las mismas dimensiones, al igual que los grupos formados por M_{p1}, M_{p2}, M_{p3} y M_{n1}, M_{n2}. La rama de salida abarca los transistores PMOS M_{p3}, M_{p6} y el transistor NMOS conectado como diodo M_C. El circuito realiza una comparación entre las corrientes de entrada I_1 e I_2. La corriente de entrada mas pequeña es reflejada hacia la otra rama de salida, cuyo transistor correspondiente, M_n, opera en la región de tríodo. La corriente de entrada mínima es también reflejada hacia la rama de salida y puede ser recuperada a través del transistor NMOS conectado como diodo M_C.

El circuito MIN puede ser transformado en un circuito multi-entrada agregando ramas con transistores apilados M_P. Sin embargo el número máximo de entradas esta limitado por el valor de V_{DD}.

En [44] se propone una versión simplificada del circuito **Min**, el cual desarrolla solamente la operación **Min** (el circuito original realiza la operación **Min** y la de un circuito Looser-take-all), mientras se mantienen una complejidad reducida del circuito.

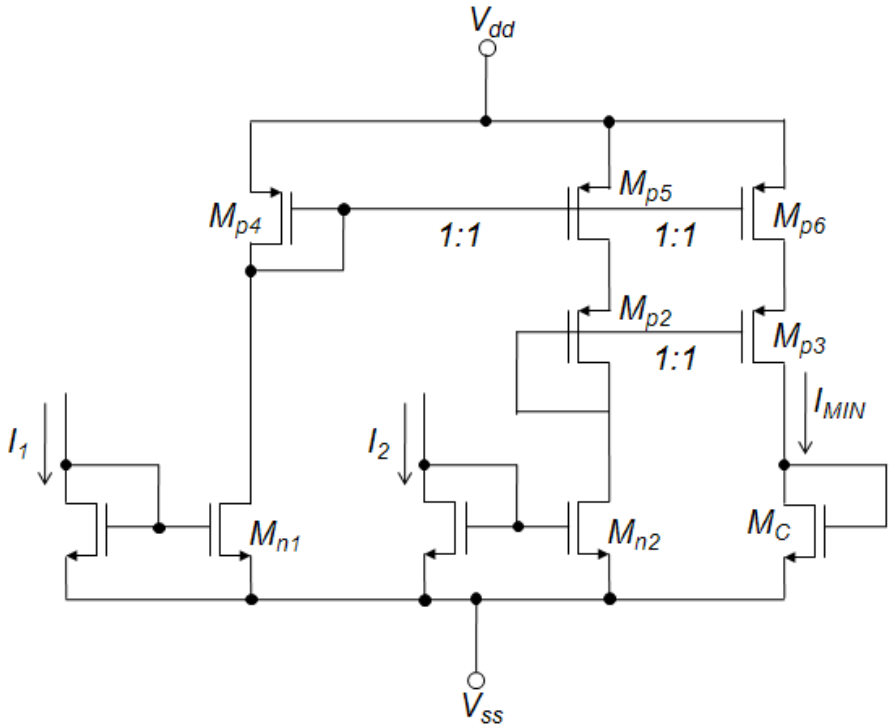

Figura 4.15. Circuito MIN con complejidad reducida.

En ambos circuitos, el desempeño en términos de exactitud y velocidad dependen de cuantas entradas sean consideradas. Además, debido a que el circuito realiza una comparación exhaustiva entre todas sus entradas simultáneamente, se mantiene un modo de procesamiento paralelo. Para una carga de entrada (*Fan-In*) pequeña, lo cual es el caso para controladores difusos, el uso de este circuito es adecuado.

La Tabla 4.3, muestra el resumen de las características del circuito MIN de la Figura 4.15 para la simulación en SPICE.

Tabla 4.3 Características del Circuito MIN.

	Valor	Unidades
$(W/L)_N$	5	---
$(W/L)_P$	7	---
V_{DD}	1.65	V
V_{SS}	-1.65	V
I_{in}	0 – 100	μA
I_o	0-100	μA

Al igual que el circuito **Max**, el circuito **Min** simulado consta de dos celdas básicas y el transistor común de salida solamente. El rango de la corriente de entrada corresponde a los valores del “0” y “1” difusos, es decir va desde 0 hasta 100 μA, la salida del circuito, por lo tanto, tiene los mismos rangos. Se realizó una simulación en DC y otra en transitorio para probar el desempeño del circuito MIN, los resultados son mostrados en las Figuras 4.16 y 4.17, respectivamente.

Para la simulación en DC, se barrieron las dos corrientes de entrada desde 0 hasta 100 μA, la primera con incrementos de 1 μA y la segunda con incrementos de 10 μA.

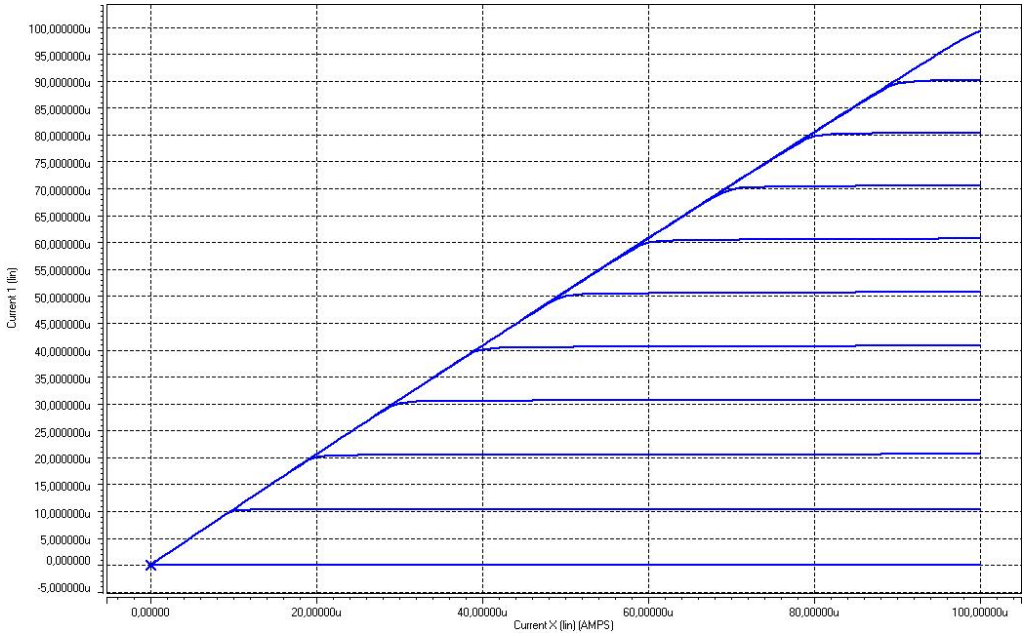

Figura 4.16. Respuesta en DC del circuito MIN.

Para la simulación en transitorio, se emplearon dos señales sinusoidales centradas en 50 µA y con picos de 50 µA, la primera con una frecuencia de de 10MHz y la segunda con una frecuencia de 20MHz.

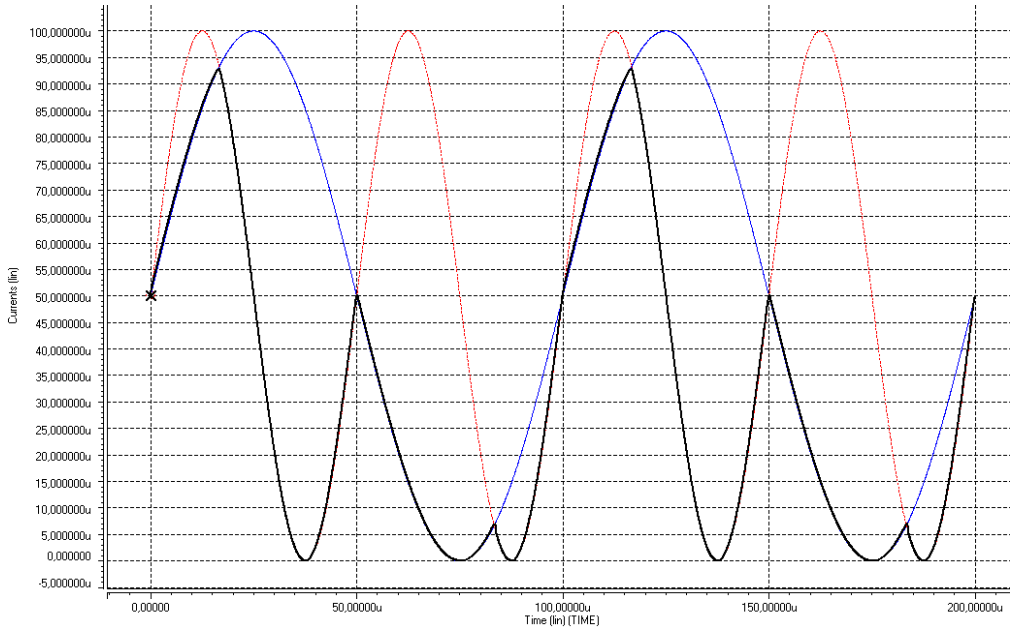

Figura 4.17. Respuesta en transitorio del Circuito MIN.

4.4 Normalizador.

La Figura 4.18 muestra el diagrama esquemático del circuito normalizador utilizado:

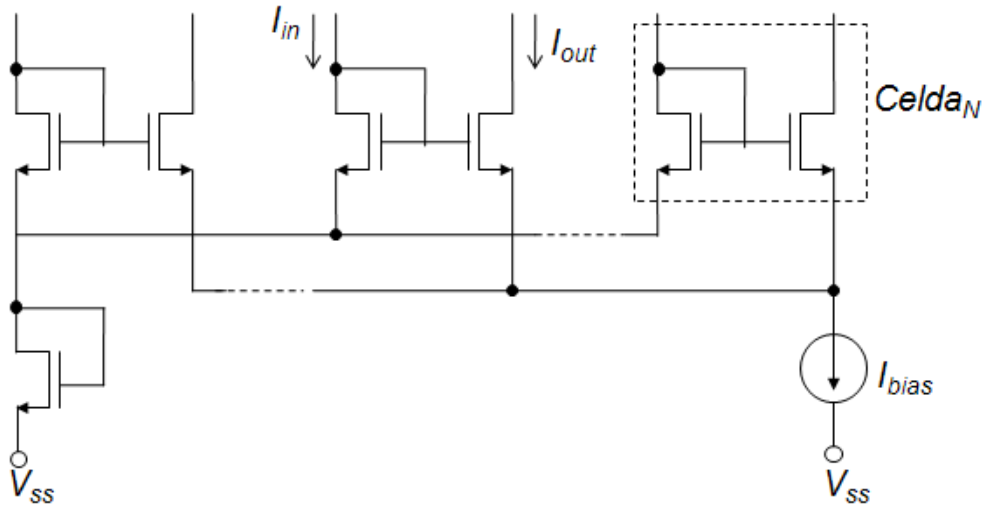

Figura 4.18. Diagrama del circuito normalizador.

El circuito normalizador consta de una estructura modular, y puede ser construido para procesar *N* señales mediante la repetición de su celda básica, mostrada en la Figura 4.18 como $Celda_N$. La señal de entrada para cada celda, proviene de la etapa de inferencia difusa del controlador, en éste caso de la salida de los operadores ***Min*** utilizados, la corriente de salida se obtiene en el drenador del otro transistor de la celda. Los transistores que reciben la señal de entrada comparten un nodo común, cuyo voltaje está fijado por medio de un transistor conectado como diodo; los transistores que entregan la señal de salida, se conectan en un punto de suma de corrientes, cuyo valor esta dado por la corriente de polarización $\boldsymbol{I_{bias}}$. El circuito normalizador presenta las siguientes características:

- La suma de todas las corrientes de salida es constante e igual al valor de $\boldsymbol{I_{bias}}$.

- Para cada celda, la transformación entrada-salida es monotónica, esto es, a la entrada más alta, le corresponde la salida más alta, manteniendo así el peso relativo de cada antecedente. Por lo tanto, a pesar de que el circuito no realiza la función de normalización ideal, éste mantiene las características esenciales y necesarias para el proceso de defuzzyficación.
- El circuito tiene una función de transferencia no lineal, lo cual no es un problema debido a que el controlador difuso completo es altamente no lineal.
- El circuito tiene tres principales fuentes de error sistemático:
  a) La impedancia finita de I_{bias} lo cual se puede disminuir realizando la fuente con espejos de tipo cascode.
  b) El desacoplo de los voltajes en los nodos de entrada, este error puede reducirse al usar transistores tipo P en la etapa anterior, esto es, en el operador **Min.**
  c) El desacoplo de voltajes en corriente directa (DC) de los nodos de entrada y salida. El cual se puede atenuar utilizando transistores cascode.

Se construyó un circuito normalizador para procesar 9 señales, provenientes de las nueve reglas del controlador, cuyas características están resumidas en la Tabla 4.4:

Tabla 4.4 Características del circuito normalizador

	Valor	Unidades
$(W/L)_{Todos}$	7.5	---
I_{bias}	100	µA
V_{DD}	1.65	V
V_{SS}	-1.65	V
I_{inN}	0-100	µA
I_{oN}	0-100	µA

Debido a que las señales de entrada del normalizador provienen de los circuitos Min, éstas están acotadas entre el “0” y “1” difusos: 0-100 μA. Así mismo, cada señal de salida esta dentro del mismo rango para que la siguiente etapa pueda procesarlas. La simulación en SPICE se hizo variando solamente dos de las entradas, haciendo un barrido de 0-100 μA en intervalos de 0.1 μA de la entrada 1, y un barrido de 0-100 μA en intervalos de 50 μA para la entrada 2. En la Figura 4.19, se muestra el comportamiento de las dos señales de salida del circuito normalizador.

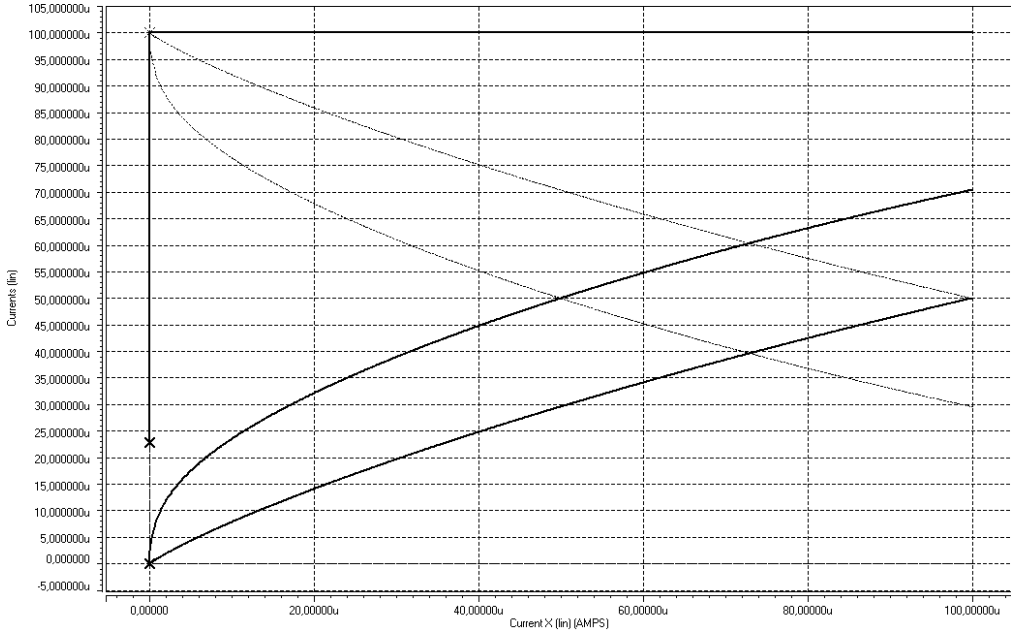

Figura 4.19. Respuesta en DC del normalizador variando solamente dos entradas.

4.5 Constante de Adaptación.

Debido a que el valor de la constante de adaptación varía de acuerdo con (3.2), se utilizó un circuito comparador de corrientes para su implementación electrónica. El diagrama del comparador es mostrado en la Figura 4.20, el circuito consta de 3 etapas principales:

- Punto de Resta. Compuesto por un espejo de corriente tipo N y otro espejo de corriente tipo P, esta etapa recibe como entradas las señales a comparar: el error, dado por la ecuación (3.3), y el valor

constante programado; si el error es mayor que el valor deseado, el voltaje en el nodo de resta es llevado a V_{DD}, si el error es menor que el valor deseado, entonces el voltaje en el nodo de resta cambia a V_{SS}.

- Buffer de Voltaje. Esta etapa intermedia consta de dos inversores digitales conectados en cascada, el primero de ellos recibe como entrada el voltaje en el nodo de resta y lo invierte. La función del segundo inversor es mejorar la respuesta del circuito, al hacer más sensible el circuito a pequeñas diferencias entre el error y el valor deseado.
- Compuertas de Transmisión. Esta es la etapa final del circuito y es la que se encarga de entregar el valor adecuado para la constante de adaptación. Esta compuesta por dos compuertas de transmisión conectadas de tal forma que cuando una de ellas se cierra, la otra se abre, así cuando el error es más grande que el valor deseado, el sistema de compuertas presenta a la salida, α_{out}, el valor más grande de la constante de adaptación, llamado α_1 y cuando el error es más pequeño, entonces a la salida se obtiene el valor más pequeño de la constante de adaptación, representado en la Figura 4.20 por α_2

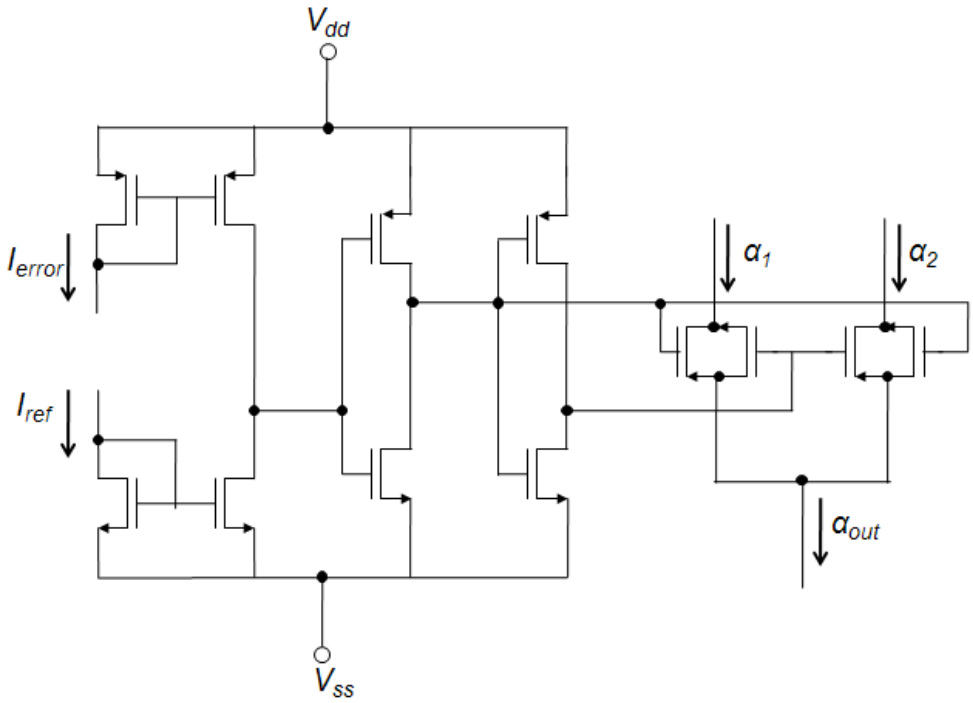

Figura 4.20. Diagrama del generador de la constante de adaptación.

La tabla 4.5 muestra un resumen de las características del circuito para su simulación en SPICE.

Tabla 4.5 Características del circuito de la constante de adaptación.

	Valor	Unidades
$(W/L)_{espejoP}$	10	---
$(W/L)_{espejoN}$	6	---
$(W/L)_{inversorN}$	1	---
$(W/L)_{inversorP}$	3	---
$(W/L)_{compuerta}$	10	---
I_{bias}	100	µA
V_{DD}	1.65	V
V_{SS}	-1.65	V
α_{out}	0.1 ó 1	µA

La Figura 4.21, muestra la simulación en SPICE del circuito generador de la constante de adaptación

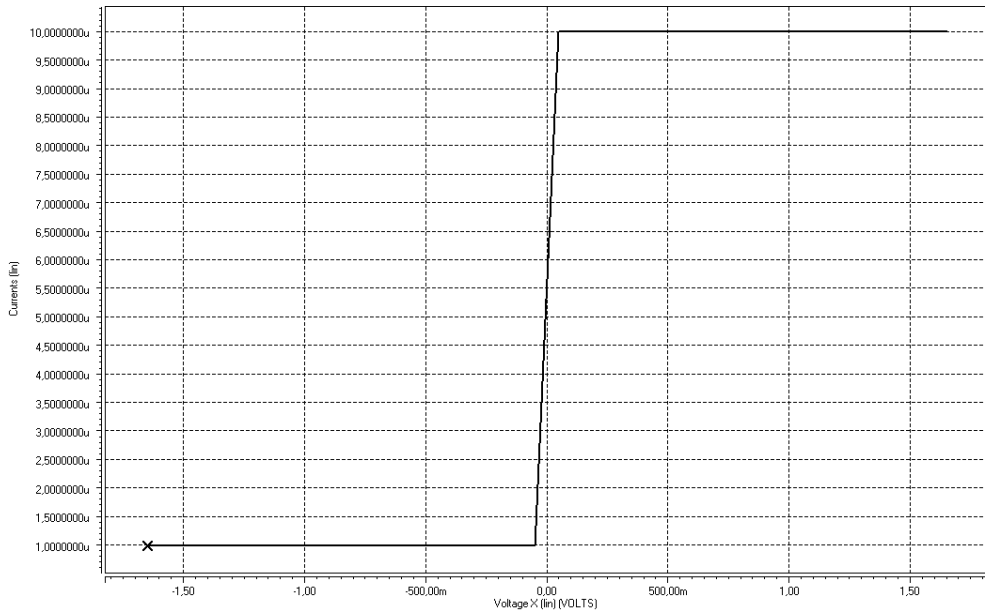

Figura 4.21 Respuesta en DC del comparador de corrientes.

Para la simulación en DC del comparador de corrientes, se barrió la entrada del error de -1.65V hasta 1.65V, y la entrada de la corriente de referencia se mantuvo fija en 0V, debido a esto, la corriente de salida se encuentra en su valor bajo para todo el rango de valores negativo, cuando $v_1 < v_2$, mientras que cuando $v_1 > v_2$ la salida toma su valor más alto.

4.6 Multiplicador de cuatro cuadrantes.

La multiplicación es una de las operaciones más usadas en el presente trabajo, debido a que es la única que está presente tanto en el algoritmo como en el sistema difuso. Para realizar esta operación se utilizó un multiplicador de cuatro cuadrantes basado en el principio translineal propuesto por Wiegerink y Seevinck [45]. El multiplicador translineal en saturación esta compuesto de dos celdas que realizan la función cuadrática,

es decir, su salida es una función del cuadrado de las entradas. La figura 4.22 muestra el diagrama esquemático del multiplicador:

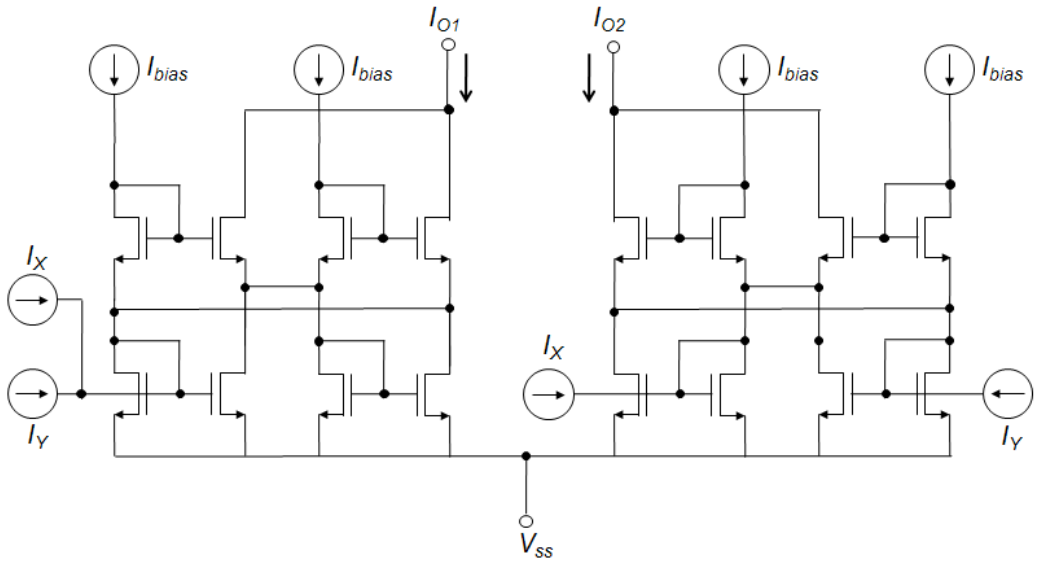

Figura 4.22 Diagrama esquemático del multiplicador de cuatro cuadrantes

De la figura 4.22 se tiene que:

$$I_{O1} = 2I_B + \left(\frac{(I_Y + I_X)^2}{8I_B} \right) \qquad (4.3)$$

Y

$$I_{O2} = 2I_B + \left(\frac{(I_Y - I_X)^2}{8I_B} \right) \qquad (4.4)$$

Por lo tanto, si hacemos que la corriente de salida del multiplicador sea la diferencia de (4.3.) y (4.4), tenemos que:

$$I_O = I_{O1} - I_{O2} = \frac{I_X \bullet I_Y}{2I_B} \qquad (4.5)$$

La corriente de polarización $\boldsymbol{I_B}$ puede ser usada para ajustar la ganancia del multiplicador, o bien para obtener la función de División. El rango de la corriente de entrada del multiplicador es:

$$|I_X| + |I_Y| \leq 4I_B \qquad (4.6)$$

Para hacer la resta de las dos corrientes de salida, solo se refleja, a través de un espejo tipo P, la corriente I_{O1} hacia la rama donde circula la corriente I_{O2} y de este mismo nodo se toma la corriente de salida I_O.

La tabla 4.6 presenta un resumen de las características del multiplicador de cuatro cuadrantes para su simulación en SPICE.

Tabla 4.6 Características del multiplicador de cuatro cuadrantes.

	Valor	Unidades
$(W/L)_{Todos}$	12.5	---
I_{bias}	50	µA
V_{DD}	1.65	V
V_{SS}	-1.65	V
I_{inN}	0-100	µA
I_{oN}	0-100	µA

El multiplicador fue diseñado para tener un rango de entrada de 0-100uA y un rango de salida de 0-100uA, compatibles con el “0” y “1” difusos del sistema en general. En las figuras 4.23 y 4.24 se muestra la respuesta en DC y Transitorio del multiplicador, respectivamente.

Para la simulación en DC del multiplicador, se hizo un barrido de 0µA hasta 100µA en incrementos de 0.01µA para una entrada, y un barrido 0µA hasta 100 µA en incrementos de 10 µA para la otra.

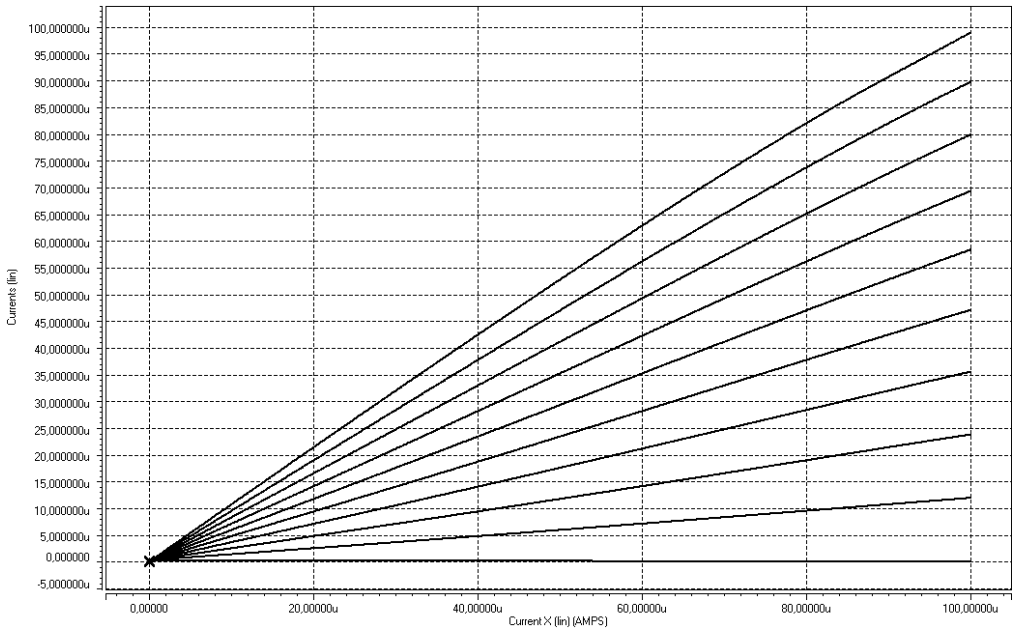

Figura. 4.23 Respuesta en DC del multiplicador.

Para la simulación en transitorio, se utilizaron como entrada, dos señales sinusoidales, centradas en 50μA y con picos de 50μA, una diez veces más lenta que la otra.

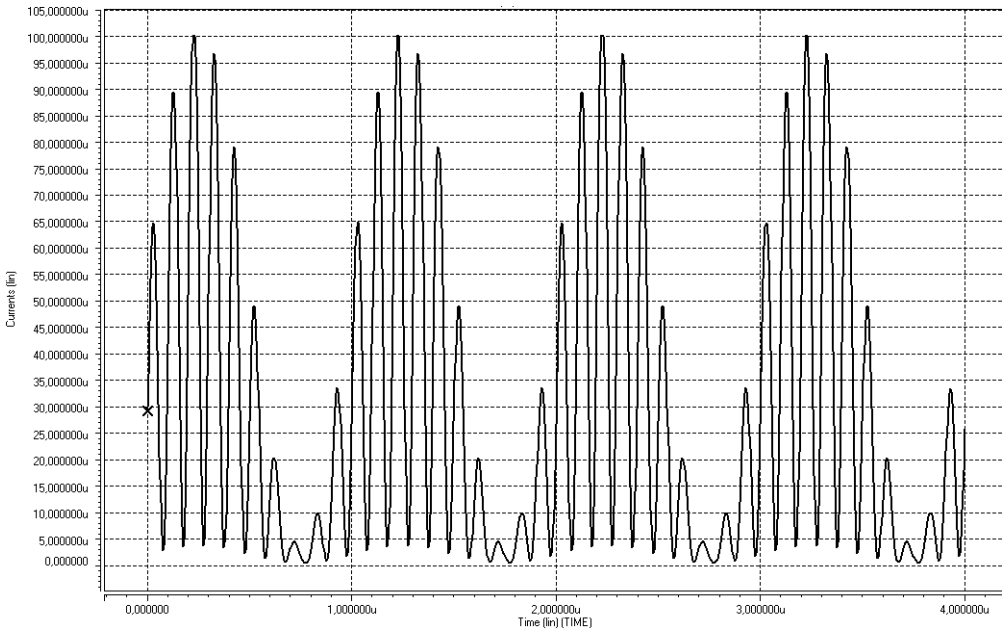

Figura 4.24. Respuesta transitoria del Multiplicador para señales de entrada sinusoidales de amplitud 0-100μA y frecuencias de 100MHz y 10MHz.

4.7 Divisor en modo corriente.

La operación de división es una de las operaciones básicas más complicadas de realizar a nivel electrónico, debido a que la operación presenta un comportamiento exponencial. Diversos trabajos han reportado circuitos divisores en modo corriente trabajando en sub-umbral [46], sin embargo, este tipo de circuitos presentan la desventaja de que el rango de entrada y salida es muy pequeño, por lo cual se necesita realizar un pre- y post-escalamiento [33] para que la señal pueda ser procesada, esto conlleva a mayor consumo de área y potencia, así como a errores sistemáticos debidos al desacoplo entre los espejos de corriente utilizados para realizar el escalamiento. Por otro lado, han sido propuestos también circuitos divisores trabajando en modo voltaje [47] los cuales hacen uso de técnicas de bajo voltaje para disminuir el consumo de potencia, pero que hacen que el circuito presente una menor frecuencia de operación.

En [48] se presenta un circuito divisor trabajando en señal mezclada, sus entradas son corrientes y la salida es en modo voltaje. La principal desventaja de este circuito es que el rango de salida esta limitado a un V_{DS}, el Voltaje Drenador-Fuente del transistor de salida. En la Figura 4.25, se muestra el circuito divisor de señal mezclada propuesto en [49].

En el circuito de la Figura 4.25, la división es realizada por los transistores $\boldsymbol{M_1}$, $\boldsymbol{M_2}$, $\boldsymbol{M_3}$, en la parte inferior del circuito, todos ellos están polarizados para trabajar en la región de tríodo. Los transistores $\boldsymbol{M_4}$, $\boldsymbol{M_5}$, $\boldsymbol{M_6}$, cuyo voltaje de compuerta es idéntico, acoplan los voltajes de Drenador-Fuente ($\boldsymbol{V_{DS}}$) de los transistores $\boldsymbol{M_1}$, $\boldsymbol{M_2}$, $\boldsymbol{M_3}$. Debido a que $\boldsymbol{V_{b1}}$ y $\boldsymbol{V_{bO}}$ son voltajes de polarización fijos, el voltaje de compuerta del transistor $\boldsymbol{M_3}$, $\boldsymbol{V_{out}}$ es “auto-ajustable”.

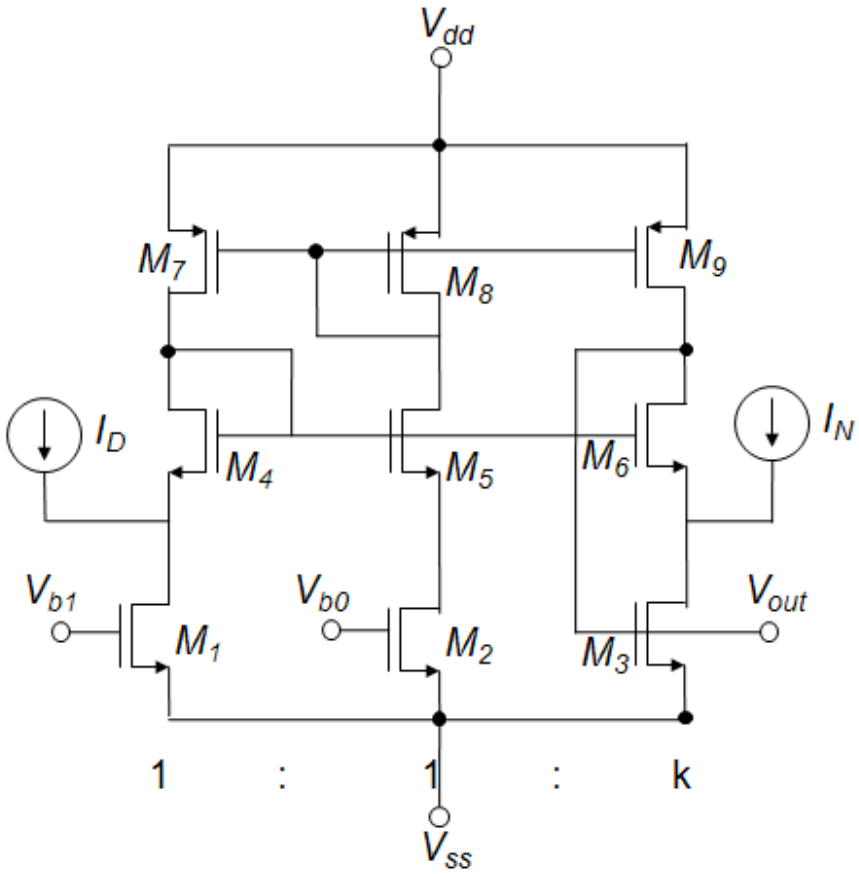

Figura 4.25. Circuito Divisor.

La ecuación que caracteriza al circuito divisor de la Figura 4.25 es:

$$\left(V_{out}-V_{b0}\right)=\frac{\left(V_{b1}-V_{b0}\right)}{k}\frac{I_N}{I_D} \qquad (4.7)$$

Así, el voltaje de salida $\boldsymbol{V_{out}}$ esta referenciado a $\boldsymbol{V_{b0}}$ por lo que el circuito es un divisor de dos cuadrantes. Dados los rangos de corrientes deseados para $\boldsymbol{I_N}$ e $\boldsymbol{I_D}$ el rango del voltaje de salida esta definido por la diferencia entre los voltajes de polarización $\boldsymbol{V_{b1}}$ y $\boldsymbol{V_{b0}}$ y por el factor de escalamiento $\boldsymbol{k}$, el cual debe ser seleccionado adecuadamente.

La Tabla 4.7, muestra un resumen de las características del circuito divisor para la simulación en SPICE.

Tabla 4.7 Características del circuito divisor.

	Valor	Unidades
$(W/L)_{1,2,3}$	3.5	---
$(W/L)_{4,5,6}$	9	---
$(W/L)_{7,8,9}$	8	---
V_{b1}	0.5	V
V_{b0}	0	V
V_{DD}	1.65	V
V_{SS}	-1.65	V
$I_{N,D}$	0-100	µA
V_{out}	0 – 0.5	V

El circuito se diseñó para que el rango de las corrientes de entrada coincidiera con el “0” y “1” difuso del sistema, así mismo la salida se fijó entre los rangos 0V y 500mV. La respuesta en DC del divisor se muestra en la Figura 4.26.

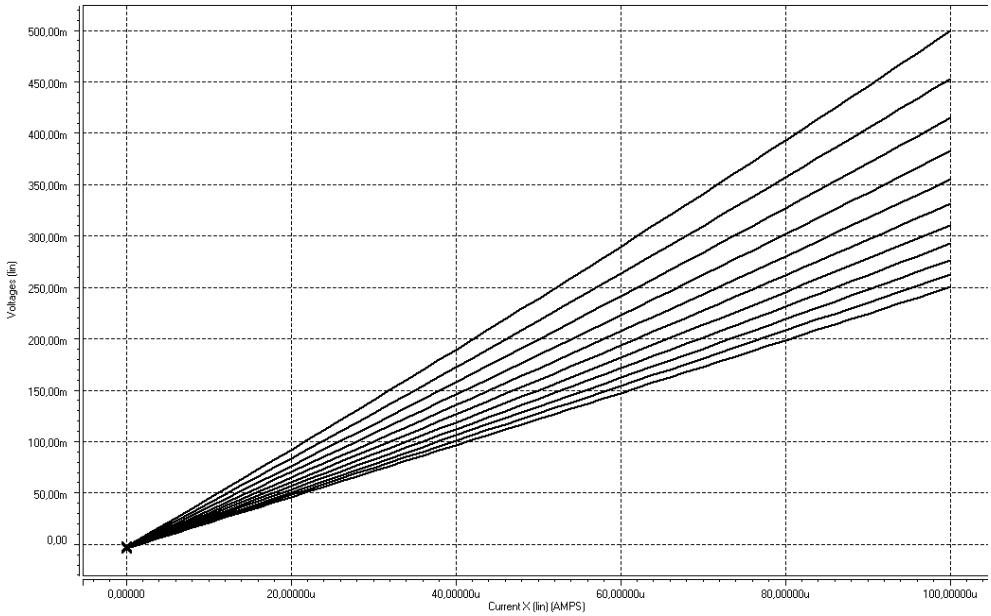

Figura 4.26. Respuesta en DC del circuito divisor.

Debido a que, como ya se ha mencionado, el controlador procesa toda la información en modo corriente, fue necesario implementar un Amplificador de Transconductancia (OTA), que convirtiera el voltaje de salida en una corriente. El circuito utilizado fue un OTA simétrico, debido a que presenta una relativa facilidad de diseño y no es necesario compensarlo con técnicas como la del capacitor Miller. La figura 4.27, muestra el diagrama esquemático del OTA.

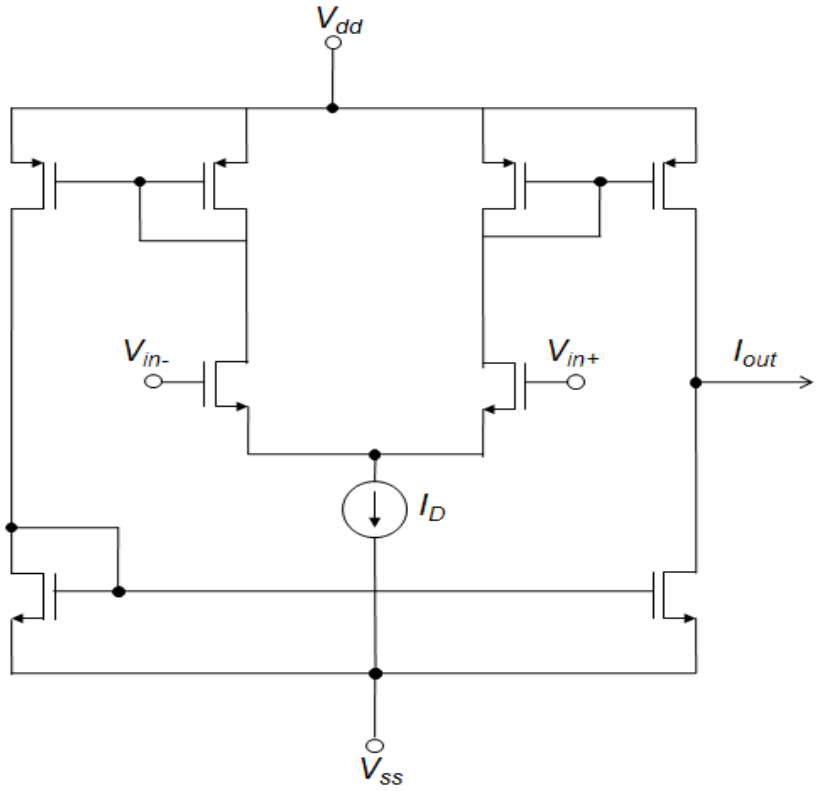

Figura 4.27. Diagrama esquemático de un OTA simétrico.

La ecuación característica de un amplificador de transconductancia esta dada por:

$$I_O = gm(V_{in+} - V_{in-}) \qquad (4.8)$$

Donde I_O es la corriente de salida del OTA, V_{in+} y V_{in-} son los voltajes en las terminales de entrada positiva y negativa respectivamente, y ***gm*** es la Transconductancia del amplificador, la cual depende directamente de la transconductancia del par diferencial de entrada.

Como se mencionó anteriormente, entre las ventajas de este OTA está su diseño relativamente fácil, su simplicidad y uso de poca área, su ganancia (**gm**) es mucho más baja que un Amplificador Operacional (OPAM) de dos etapas, tiene un solo polo dominante, el cual esta dado por la capacitancia de carga en el nodo de alta impedancia (nodo de salida), por lo cual tampoco necesita ser compensado con redes RC, como resultado, al hacer uso solamente de transistores CMOS, es una alternativa muy "atractiva" para la integración en sistemas VLSI.

La Tabla 4.8, muestra un resumen de las características del OTA para la simulación en SPICE.

Tabla 4.8 Características del OTA

	Valor	Unidades
$(W/L)_{Todos}$	8.5	---
I_{bias}	150	µA
V_{DD}	1.65	V
V_{SS}	-1.65	V
V_{in+}	0-500	mV
V_{in-}	0	V
I_O	0-100	µA

El OTA fue diseñado para un rango de entrada de 0mV a 500mV (que es el rango de salida del circuito divisor), y un rango de salida de 0 µA a 100 µA que corresponden al "0" y "1" difusos del sistema. La salida del circuito divisor se conecta directamente a la entrada positiva del OTA, mientras que

la entrada negativa está fija a un voltaje de referencia: 0V (correspondiente a la tierra de señal).

La respuesta en DC del OTA es mostrada en la Figura 4.28.

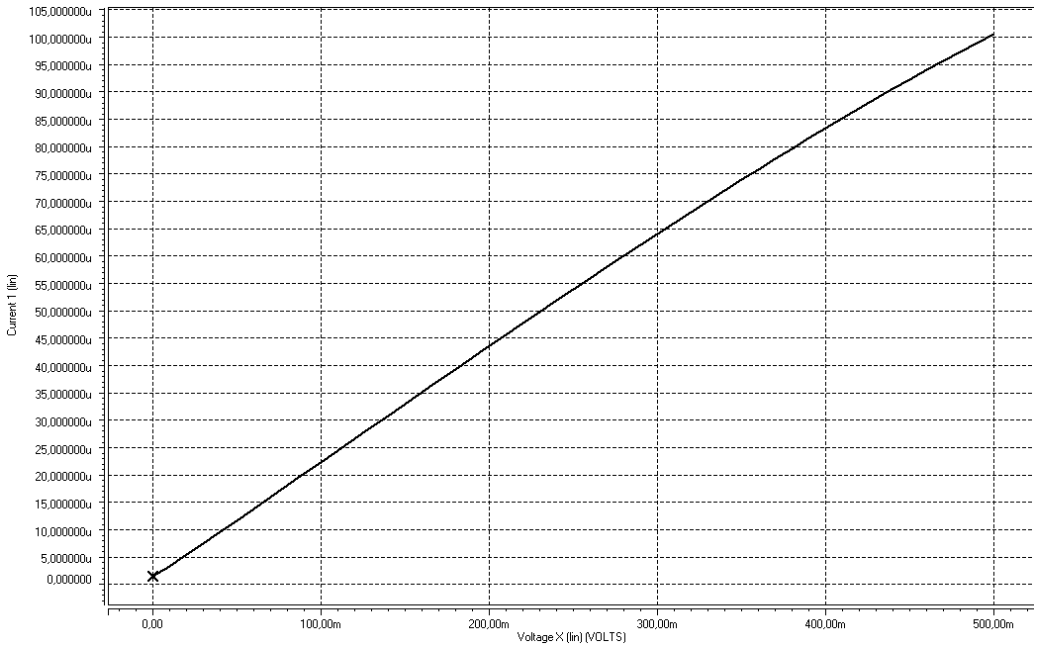

Figura 4.28. Respuesta en DC del OTA simétrico.

4.8 Espejos PWL.

Los espejos con una salida de tipo Lineal a Trozos (PWL) son útiles cuando se requiere acotar la señal de salida dentro de un rango bien definido. Debido a que el algoritmo “NonLinear Backpropagation” requiere de una retroalimentación en modo corriente, el bloque utilizado se diseñó basado en espejos de corriente con ésta característica. La Figura 4.29 muestra una combinación de espejos de corriente con salida PWL:

La salida del espejo de corriente de la Figura 4.29, es de la forma PWL, su valor mínimo está dado por el valor de la fuente de polarización del espejo tipo P, mientras que el punto de quiebre de la función PWL esta definido por el valor de la fuente de alimentación del espejo tipo N, la

pendiente del segundo segmento de recta está dada por la relación entre las dimensiones de los transistores que conforman el par de espejos.

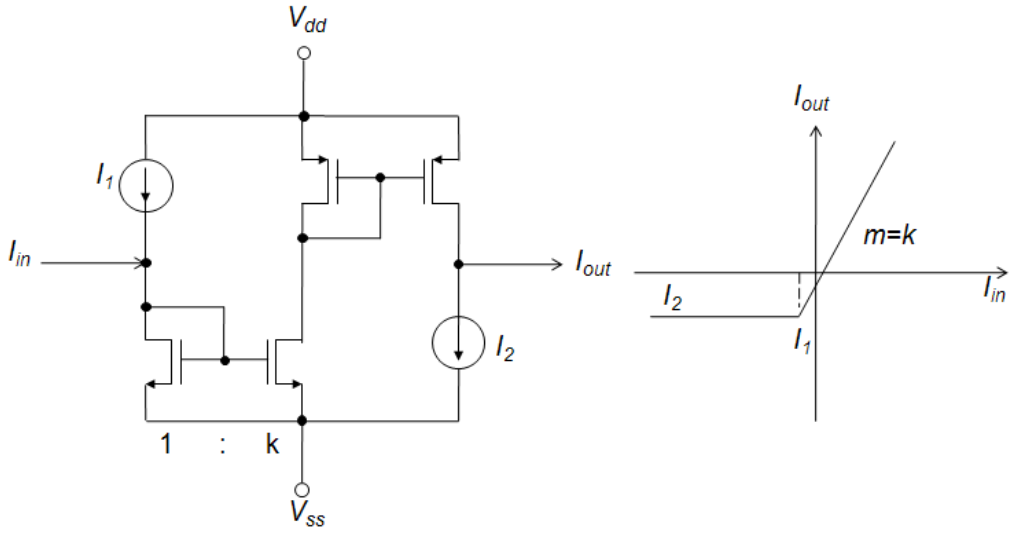

Figura 4.29. Espejo de corriente con salida PWL acotada en la parte inferior.

Una desventaja de este arreglo de espejos, es que solamente se define el acotamiento inferior de la corriente de salida, pues no hay forma de recortar la señal de salida en la parte superior de la función. Otro arreglo de espejos ofrece la posibilidad de acotar el nivel superior de la función:

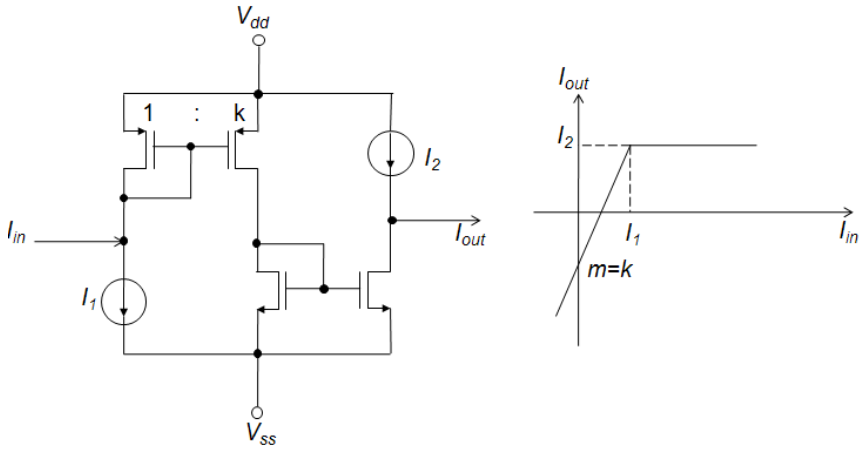

Figura 4.30. Espejo de corriente con salida PWL acotada en la parte superior.

El espejo de la Figura 4.30, cuenta con el acotamiento en la parte superior, sin embargo, no existe forma de acotar la salida en la parte inferior. Para solucionar el problema del acotamiento superior e inferior, se hizo una combinación de ambos espejos, de tal forma que se aproveche el acotamiento inferior del espejo de la Figura 4.29 y el acotamiento superior del espejo de la Figura 4.30:

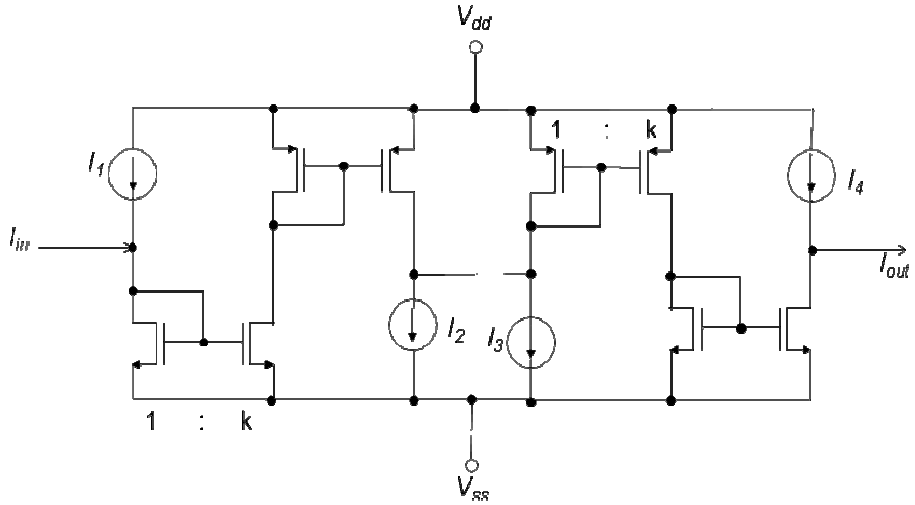

Figura 4.31, combinación de espejos PWL con salida acotada en ambos extremos.

Al poner en cascada un espejo PWL con acotamiento inferior y un espejo PWL con acotamiento inferior, se logra tener una salida con acotamiento en ambos niveles y se sigue manteniendo el control sobre el comportamiento de la salida.

Como se observa en la Figura 4.32, la respuesta de los espejos conectados en cascada está delimitada por las cuatro corrientes de alimentación del circuito de la Figura 4.32, las cuales ofrecen al diseñador control total sobre la respuesta en la salida, pudiendo modificar los niveles inferior y superior, por medio de las corrientes de polarización I_2 e I_4 respectivamente, y definiendo los puntos de quiebre mediante las corrientes I_1 e I_3, así mismo, la pendiente de la recta intermedia puede ser controlada

por medio de la razón entre los transistores que conforman cada espejo individual.

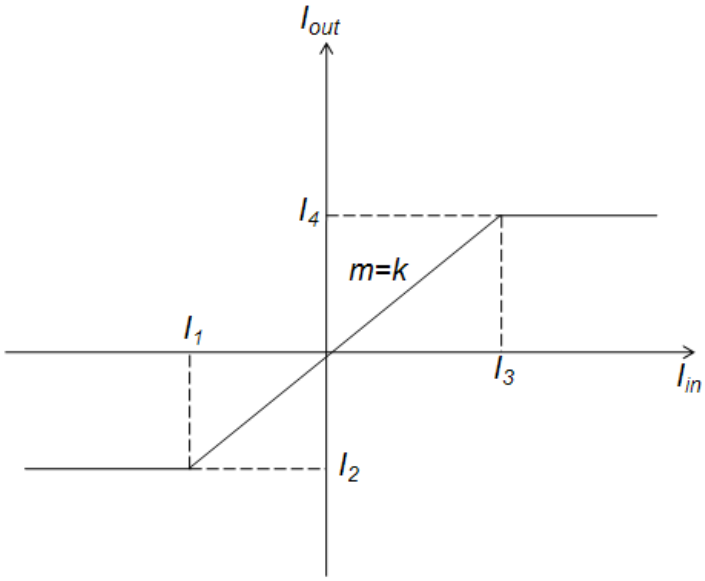

Figura 4.32. Respuesta PWL con acotamiento inferior y superior.

La tabla 4.9 muestra un resumen de las características del espejo PWL con acotamiento superior e inferior utilizadas en la simulación en SPICE:

Tabla 4.9 Características del espejo PWL

	Valor	Unidades
$(W/L)_{Todos}$	11	---
I_1	-50	µA
I_2	0	µA
I_3	50	µA
I_4	100	µA
V_{DD}	1.65	V
V_{SS}	-1.65	V
I_{in}	-100 - 100	µA
I_O	0 - 100	µA

La respuesta en DC del espejo se muestra en la Figura 4.33.

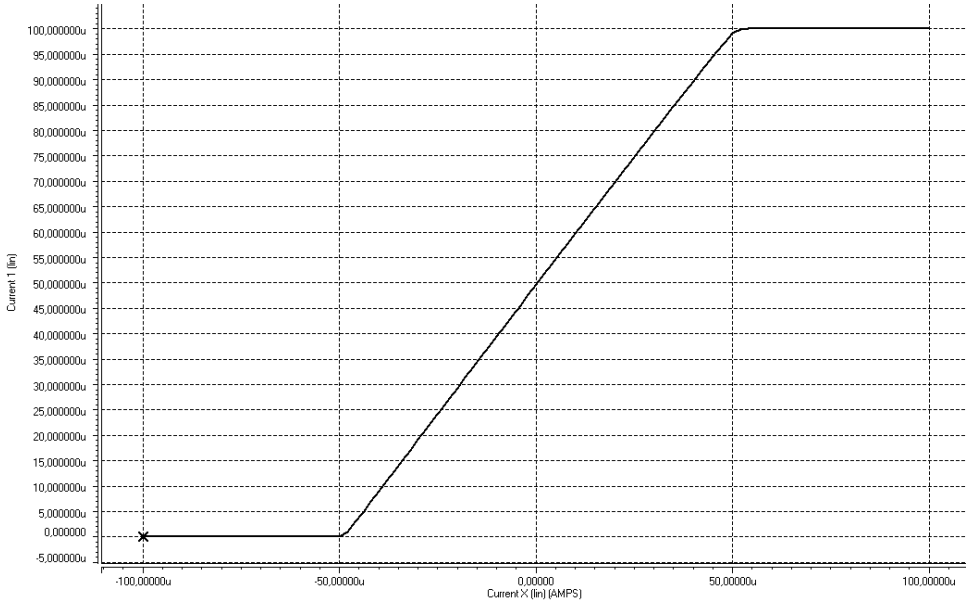

Figura 4.33. Respuesta en DC del espejo PWL.

4.9 Generador de Retardo.

Para generar el retardo necesario para que el sistema pueda efectuar las iteraciones en el tiempo, se utilizó una celda de memoria analógica switcheada en modo corriente (SI). Diferentes topologías de SI han sido propuestas [50] en los últimos años.

El funcionamiento de las celdas SI está basado en la habilidad del transistor MOS de mantener su corriente de drenador I_D con la compuerta en circuito abierto mediante la carga almacenada en su óxido de compuerta [51]. El uso de dispositivos MOS como medio de almacenamiento fue propuesto originalmente para aplicaciones de fotodiodos en 1972 [52] La Figura 4.34 muestra la estructura de una celda SI básica de primera generación.

Las fuentes de corriente del circuito sirven para polarizar a los transistores, de tal modo que la corriente de entrada pueda ser de valor negativo o positivo. Así, mientras la corriente de entrada no exceda el valor

de la fuente de corriente (en el sentido negativo) los transistores permanecerán en saturación y la celda funcionará correctamente. El switch que separa las compuertas de ambos transistores está controlado por una señal de reloj Φ.

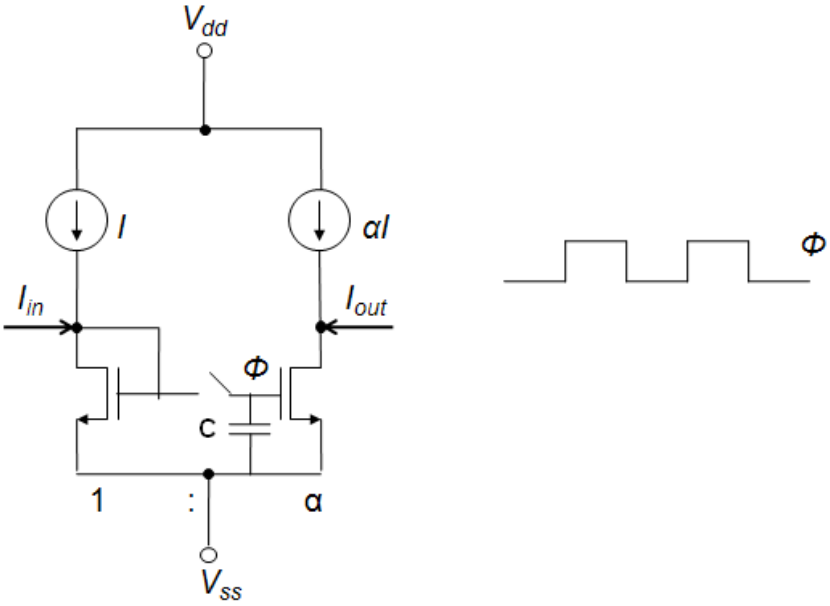

Figura 4.34. Celda de memoria de primera generación

Cuando ***Φ*** está en “Alto” la función del circuito es la de un espejo de corriente y la corriente de entrada I_{in} fluye sobre el transistor conectado como diodo sumada a la corriente de polarización. Ambos transistores comparten el mismo voltaje de compuerta, sin embargo en el segundo transistor fluye la misma corriente que en el primero multiplicada por el factor de escalamiento **α.** Cuando **Φ** pasa a “Bajo” el switch se abre y el último valor del voltaje de compuerta es almacenado en el capacitor **C**, manteniendo el valor de la corriente de drenador del segundo transistor y, por lo tanto, manteniendo el valor de la corriente de salida $\boldsymbol{I_{out}}$.

En esta configuración, el retardo de la señal es de medio ciclo de **Φ**, por lo tanto, se utilizó una configuración en cascada para obtener un retardo

total igual al ciclo completo de **Φ**. El diagrama esquemático para el retardo total se muestra en la Figura 4.35.

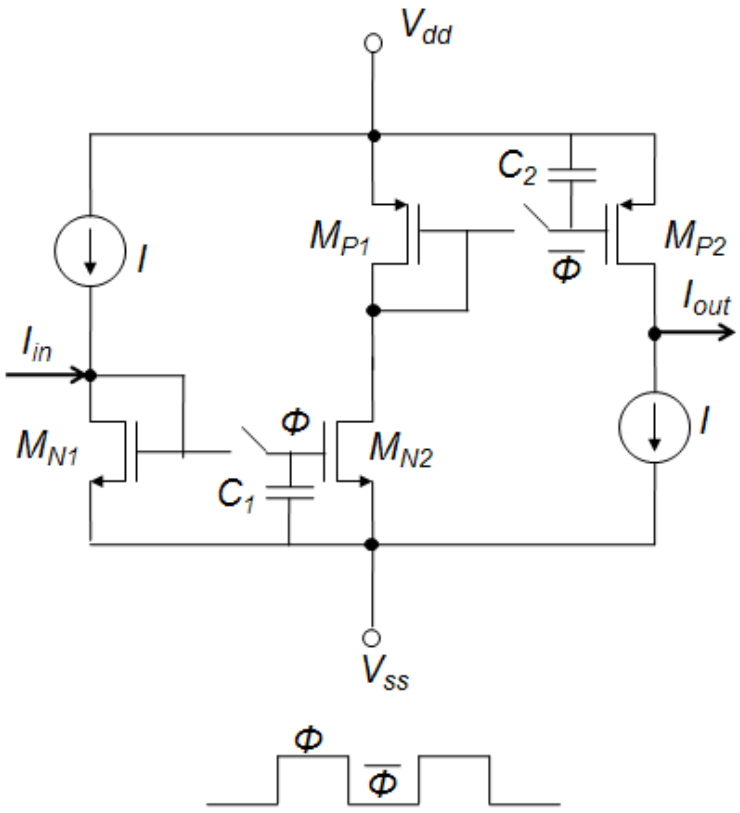

Figura 4.35. Celda de memoria en cascada con retardo total igual al ciclo completo de reloj.

En el circuito de la Figura 4.36, cuando ***Φ*** esta en alto, el switch del espejo tipo N está cerrado, mientras que el switch del espejo tipo P se encuentra abierto, durante este periodo, la corriente de entrada fluye sobre M_{N1} haciendo que el capacitor C_1 se cargue al valor de voltaje de compuerta. Cuando **Φ** está en bajo, el switch del espejo tipo N se abre y el switch del espejo tipo P se cierra, durante este periodo, la corriente que fluye sobre M_{N2} y M_P es mantenida por el voltaje almacenado en C_1 y se refleja en la rama de salida de la celda, y el capacitor C_2 se carga al valor de voltaje de compuerta de M_{P1}. Cuando llega un nuevo nivel alto de ***Φ***, la corriente de salida de la

celda se mantiene constante gracias al voltaje almacenado en C_2 durante el ciclo anterior. De esta manera, la señal necesita un ciclo completo de reloj desde que entra al primer espejo hasta que es reflejada en el segundo espejo.

La tabla 4.10 resume las características de la celda de la Figura 4.35 para la simulación en SPICE.

Tabla 4.10 Características de la celda de retardo.

	Valor	Unidades
$(W/L)_{Espejos\ N,\ P}$	7	---
$(W/L)_{Switches}$	1	---
C_1	300	fF
C_2	900	fF
Φ	250	KHz
I_{bias}	100	μA
V_{DD}	1.65	V
V_{SS}	-1.65	V
I_{in}	0-100	μA
I_{out}	0-100	μA

La celda presenta un retardo total de 4μs, para retardos más pequeños se puede hacer uso de la capacitancia compuerta-fuente (C_{GS}) parásita de los transistores CMOS. El comportamiento transitorio de la celda se muestra en la Figura 4.36.

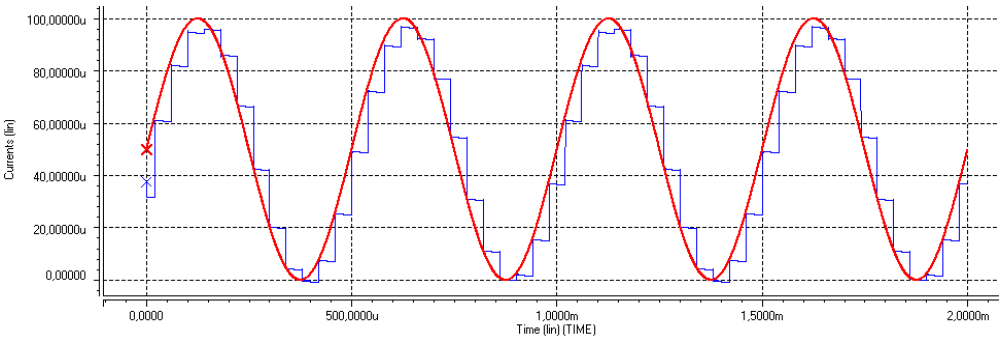

Figura 4.36. Respuesta en transitorio de la celda de retardo.

4.10 Operaciones Básicas: Suma y Resta de Corrientes.

Una de las ventajas de trabajar en modo corriente, es que las operaciones básicas de Suma y Resta se pueden aplicar de forma transparente, al aplicar la Ley de Corriente de Kirchoff (KCL), la cual especifica:

"La suma total de las corrientes que entran y salen de un nodo es igual a cero".

Entonces, basta con hacer incidir sobre un mismo nodo las corrientes que se desean sumar y/o restar con el sentido adecuado para realizar la operación.

CAPÍTULO 5. RESULTADOS.

En el presente capítulo se presentan los resultados obtenidos de las simulaciones hechas tanto en MATLAB como en SPICE del cotrolador difuso adaptado mediante el algoritmo “Nonlinear Backpropagation”.

Las simulaciones hechas en MATLAB fueron:

- Simulación de un controlador difuso tipo Takagi-Sugeno-Kang de orden cero con dos entradas, una salida, tres funciones de membresía por entrada, nueve reglas, nueve singleton, mediante la herramienta “***FIS editor”*** del ***Fuzzy Logic Toolbox*** para copiar tres funciones objetivo no lineales: Función Cuadrática, Función Sinusoidal, Función Sinc.
- Simulación de un controlador difuso tipo Takagi-Sugeno-Kang de orden cero con dos entradas, una salida, cinco funciones de membresía por entrada, veinticinco reglas, y veinticinco singleton, entrenado mediante el algoritmo “Nonlinear Backpropagation”, utilizando tres superficies no lineales como funciones objetivo, y variando el valor inicial de los singleton para hallar el mejor desempeño del entrenamiento.
- Simulación de un controlador difuso tipo Takagi-Sugeno-Kang de orden cero con dos entradas, una salida, tres funciones de membresía por entrada, nueve reglas, y nueve singleton, entrenado mediante el algoritmo “Nonlinear Backpropagation”, utilizando tres superficies no lineales como funciones objetivo.

Las simulaciones hechas en SPICE fueron:

- Simulación de un controlador difuso tipo Takagi-Sugeno-Kang de orden cero con dos entradas, una salida, tres funciones de membresía por entrada, nueve reglas, nueve singleton, para copiar tres funciones objetivo no lineales.

- Simulación de un controlador difuso tipo Takagi-Sugeno-Kang de orden cero con dos entradas, una salida, tres funciones de membresía por entrada, nueve reglas, y nueve singleton, entrenado mediante el algoritmo “Nonlinear Backpropagation”, utilizando tres superficies no lineales como funciones objetivo.

  Todas las simulaciones, se hicieron basadas en la aplicación del modelado adaptable, mostrado en la Figura 5.1

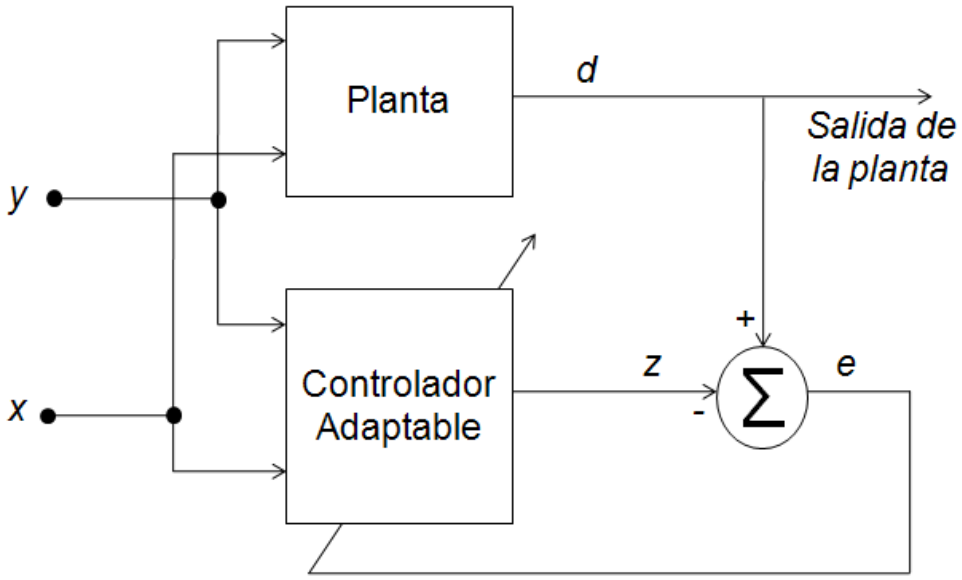

Figura 5.1 Modelado adaptable de una planta don dos entradas y una salida

En la Figura 5.1 ***x*** e ***y*** son las señales de entrada tanto de la planta como del controlador difuso adaptable, ambas señales son un vector que contiene ruido aleatorio uniformemente distribuido en el rango de entrada (-1.65, 1.65).

Para todas las simulaciones realizadas, se trabajó con tres superficies no lineales como funciones objetivo:

- Sinusoidal. Esta superficie, está dada por:

$$Salida=((\sin((Ent1))/.75)*(\sin((Ent2)/.75)))*1.65 \quad (5.1)$$

Donde ***Ent1*** e ***Ent2*** son las entradas del sistema. La superficie es mostrada en la Figura 5.2

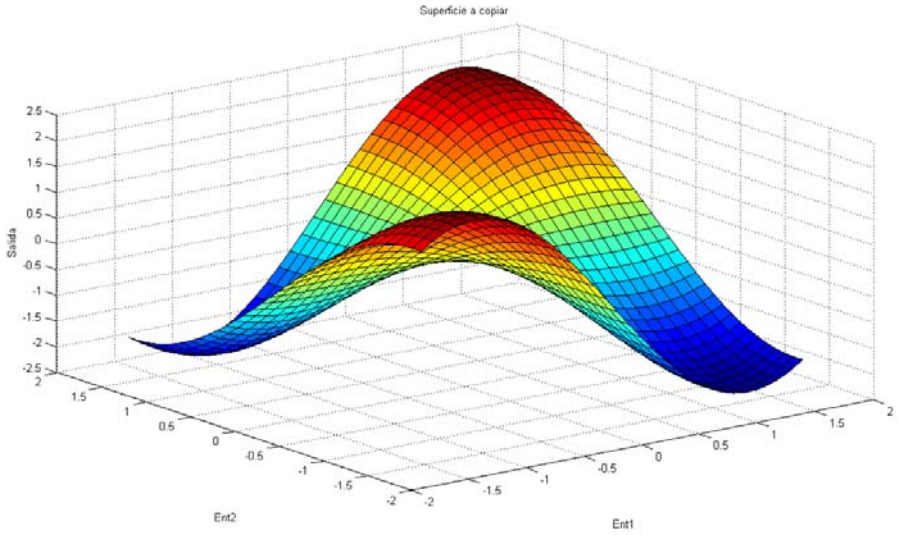

Figura 5.2 Superficie objetivo sinusoidal.

- Cuadrática. Esta superficie, está dada por:

$$Salida=((Ent1.\hat{}2)+(Ent2.\hat{}2))/3.3 \quad (5.2)$$

La superficie es mostrada en la Figura 5.3

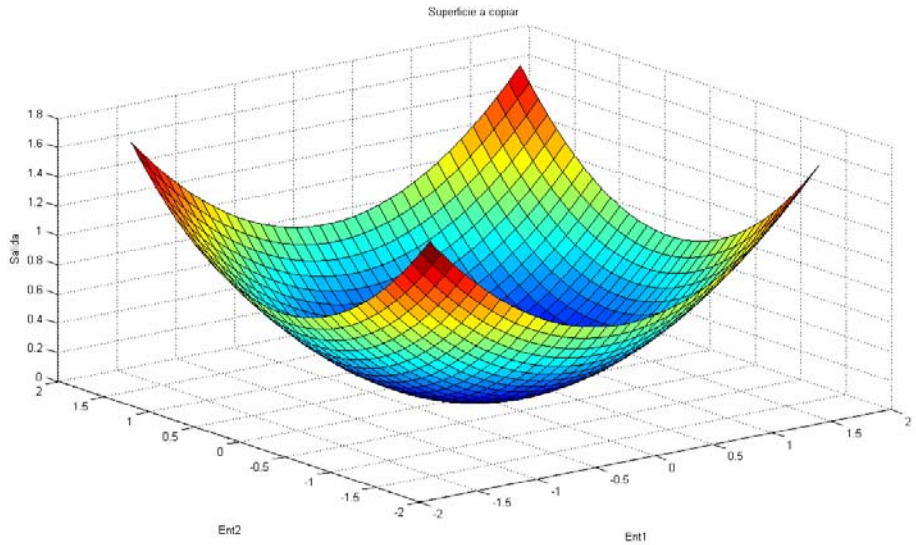

Figura 5.3 Superficie objetivo cuadrática.

- Sinc. Esta superficie, está dada por:

$$Salida=(sinc(Ent1)*sinc(Ent2.))/3.3 \quad (5.3)$$

La superficie es mostrada en la Figura 5.4

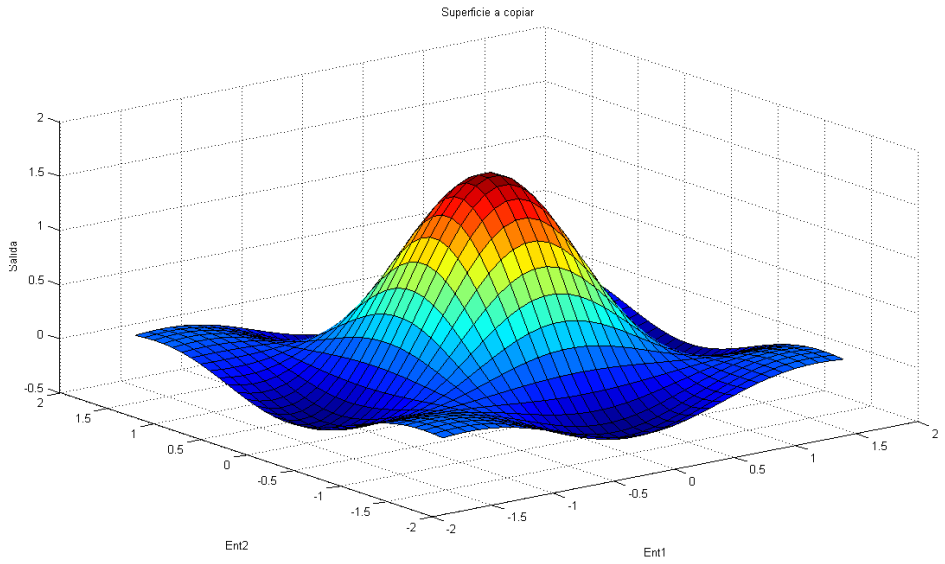

Figura 5.4 Superficie objetivo Sinc

. Para la primera simulación en MATLAB, como se indicó anteriormente, se utilizó el Editor Gráfico ***FIS Editor***, las tres funciones de membresía por entrada y la base de reglas se programaron tal como se mencionó en el Capítulo 3. La posición de los singleton fue determinada y programada para cada superficie en particular. La Tabla 5.1 muestra la posición de los singleton para copiar cada una de las tres superficies.

Tabla 5.1 Valores de los nueve singleton para las tres superficies objetivo.

	S1	S2	S3	S4	S5	S6	S7	S8	S9
Sinusoidal	100	50	0	50	50	50	0	50	100
Cuadrática	100	0	100	0	0	0	100	0	100
Sinc	5	4	5	4	100	4	5	4	5

Los resultados para las tres superficies, se muestran en las Figuras 5.5, 5.6 y 5.7.

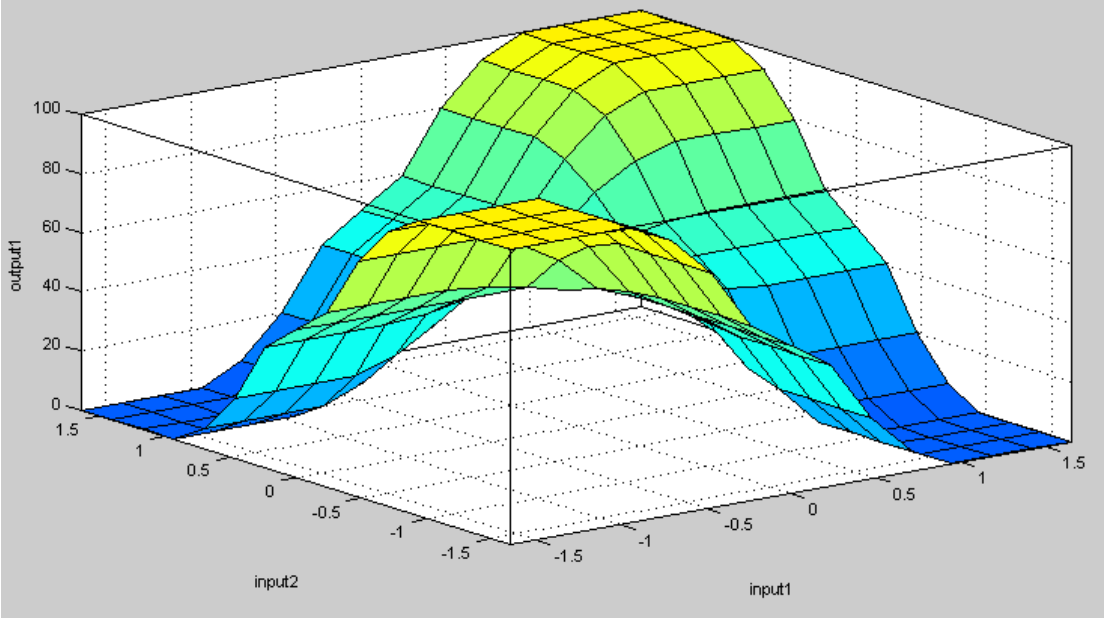

Figura 5.5 Superficie sinusoidal copiada utilizando FIS Editor.

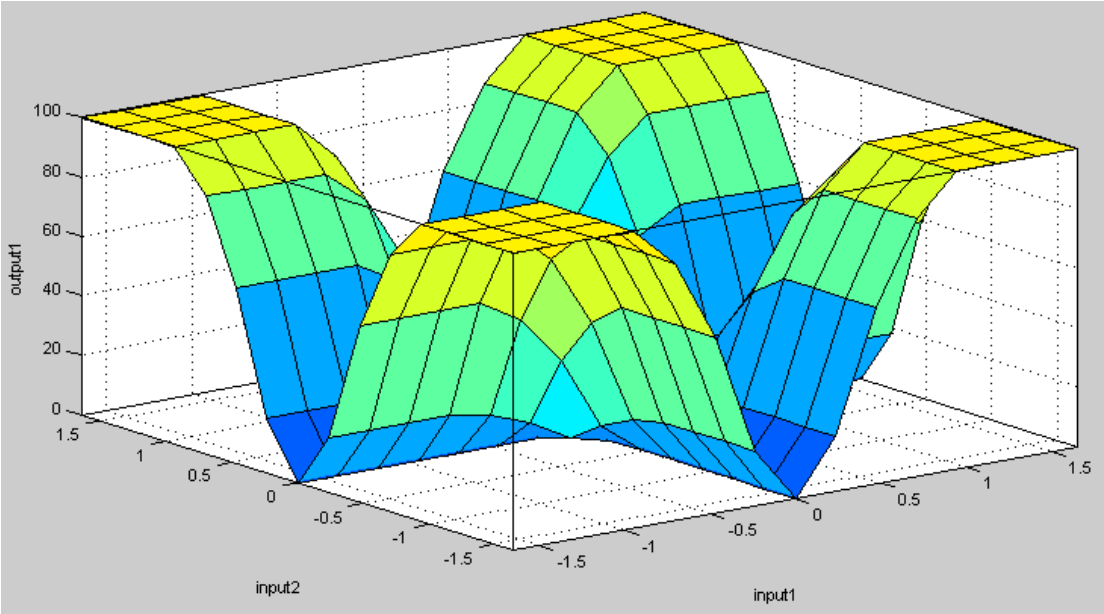

Figura 5.6 Superficie cuadrática copiada utilizando FIS Editor.

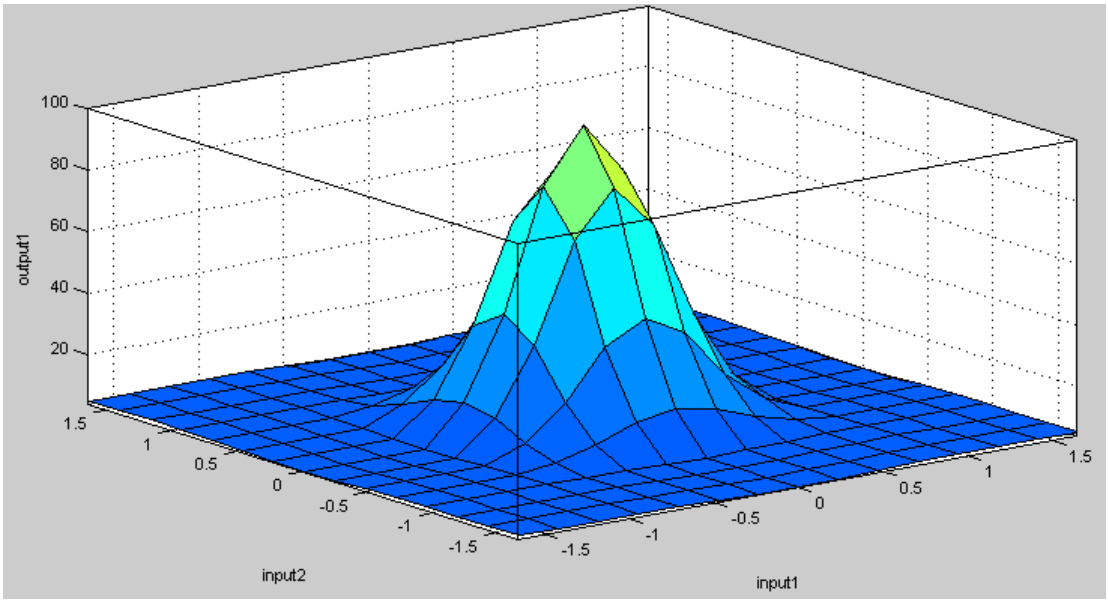

Figura 5.7 Superficie Sinc copiada utilizando FIS Editor.

Como puede observarse en las Figuras 5.5, 5.6 y 5.7 la resolución en la salida del controlador es baja, debido a que sólo se cuenta con nueve grados de libertad, equivalente a los nueve singleton con los que cuenta el controlador. Por lo tanto, para aplicaciones, en las que la resolución no sea un parámetro crítico, éste sistema es adecuado debido a su sencillez y bajo costo de implementación, sea en Hardware o Software.

El aumento en la resolución de la salida implica aumentar el número de singleton, lo que a su vez conlleva a aumentar el número de reglas y el número de funciones de membresía por entrada, provocando un mayor consumo de recursos en la implementación del controlador.

Para la segunda simulación en MATLAB, se implementan ahora cinco funciones de membresía y veinticinco reglas, pero se mantiene la característica de que están distribuidas en todo el discurso. La distribución de las funciones de membresía se muestra en la Figura 5.8

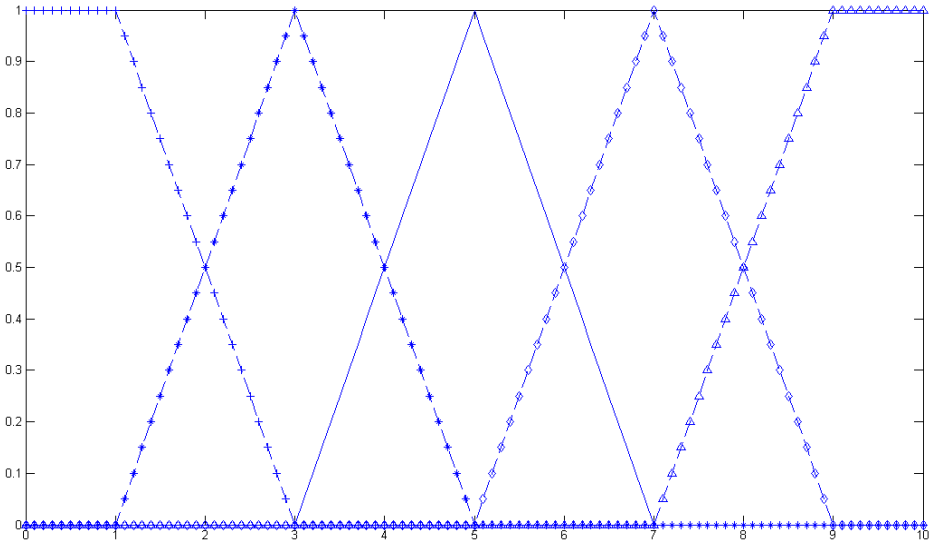

Figura 5.8 Distribución de las cinco funciones de membresía.

Para ésta simulación, se implementa el algoritmo de Retro-propagación como sistema de entrenamiento del controlador difuso. Los veinticinco singleton se inicializaron en 50, equivalente a la mitad del discurso, a partir de ahí el controlador difuso fue entrenado durante 3,000 iteraciones. Debido a que con 5 singleton y veinticinco reglas, el controlador tiene más grados de libertad para poder modelar la superficie objetivo, éstas fueron modificadas sólo en ésta simulación y se muestran junto con los resultados en las Figuras 5.9, 5.10 y 5.11.

En la parte inferior derecha de la Figura 5.9 se muestra la función sinusoidal objetivo para el controlador, en la parte superior derecha se muestra la superficie que el controlador copio después de 3,000 iteraciones, en la parte superior izquierda se muestra el comportamiento del error cuadrático medio (MSE) del sistema y en la parte inferior izquierda se muestra la posición final de cada uno de los veinticinco singleton. Como se puede observar, al contar con más singleton, reglas y funciones de membresía, el controlador es capaz de copiar superficies más complejas con un error relativamente bajo, puesto que ahora, el espacio es particionado en 25 secciones, equivalentes a 25 grados de libertad en la salida del controlador.

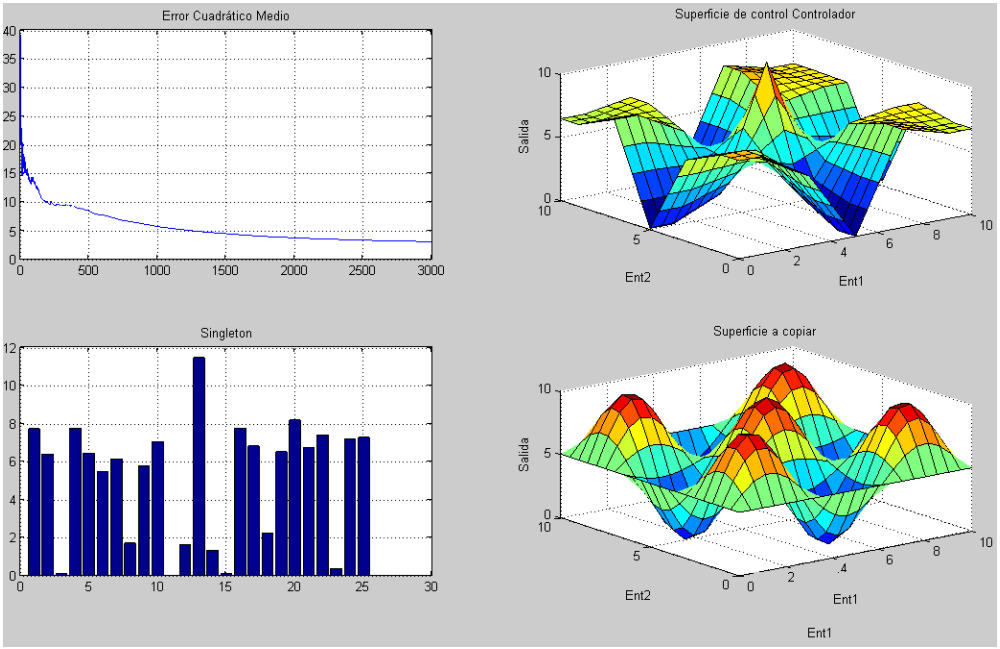

Figura 5.9 Resultados del entrenamiento del controlador con veinticinco reglas para una función sinusoidal.

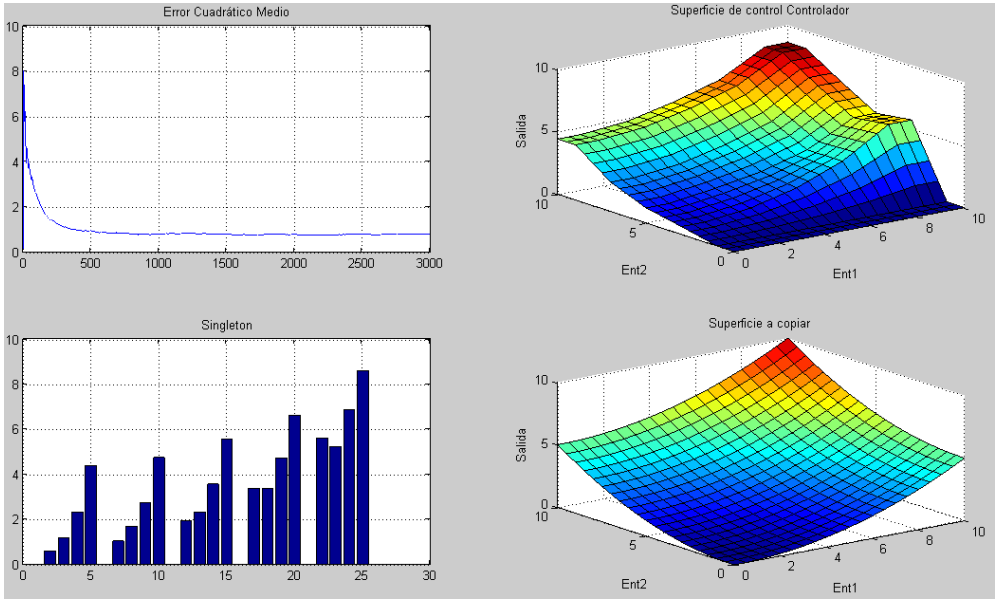

Figura 5.10 Resultados del entrenamiento del controlador con veinticinco reglas para una función cuadrática.

En el caso de la función objetivo cuadrática, puede verse que, al ser más sencilla, el controlador la puede copiar con un error muy pequeño, lo cual puede ser comprobado en la gráfica del error cuadrático medio en la parte superior izquierda de la Figura 5.10 o por inspección visual de la superficie copiada en la parte superior derecha de la misma Figura. Más aún, se puede observar un patrón bien definido en la distribución cuasi-uniforme de la posición de los singleton después del entrenamiento.

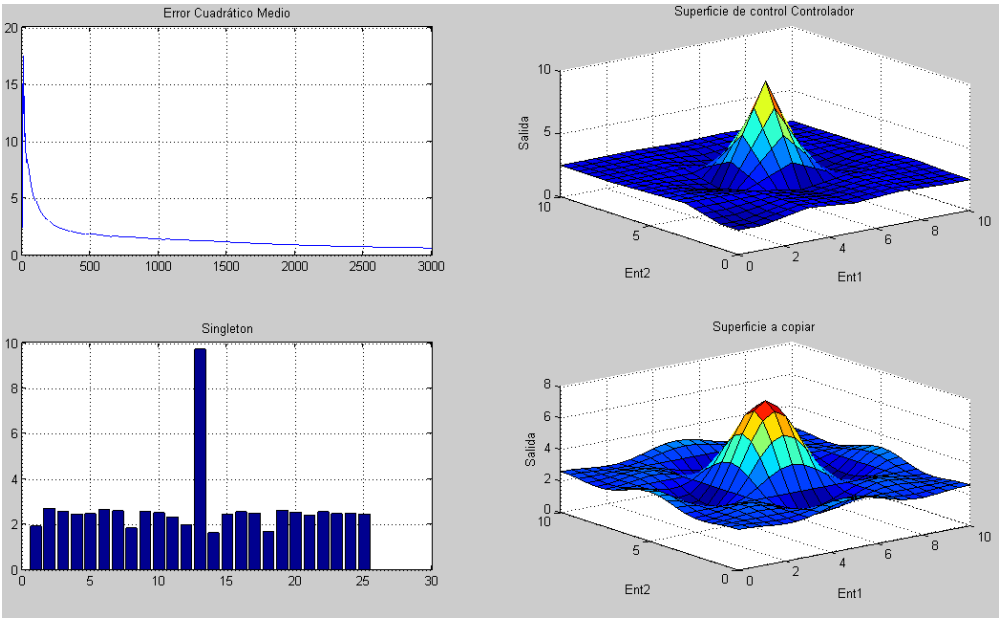

Figura 5.11 Resultados del entrenamiento del controlador con veinticinco reglas para una función Sinc.

En la Figura 5.11 puede observarse nuevamente el comportamiento del controlador entrenado con una superficie objetivo relativamente sencilla, el error decrece rápidamente desde las primeras iteraciones, al igual que los singleton que alcanzan un valor estable después de un número pequeño de iteraciones.

La rapidez en la adaptación, y la estabilidad del sistema se deben principalmente al cambio en el valor de la constante de adaptación. En [53] se presenta un controlador difuso con las mismas características pero

entrenado mediante el algoritmo LMS, el trabajo aquí presentado demostró ser cinco veces más rápido al adaptar el sistema y con un error cuadrático medio menor. La tabla 5.2 muestra una comparación entre este trabajo y [53].

Tabla 5.2 Comparación del desempeño del presente trabajo con [53].

Función	Trabajo	MSE	Número de iteraciones
Sinusoidal	Este trabajo	2	3,000
	[53]	2.7	15,000
Sinc	Este trabajo	0.28	3,000
	[53]	0.3	15,000

Para comprobar que la posición inicial de los singleton afecta el proceso de adaptación se hicieron las siguientes simulaciones para el mismo controlador difuso con cinco funciones de membresía y veinticinco reglas:

- Todos los singleton inicializados en “0” difuso.
- Todos los singleton inicializados en “1” difuso.
- Todos los singleton distribuidos en el discurso.
- Se hicieron cinco grupos de cinco singleton cada uno y los grupos fueron distribuidos en el discurso.

La tabla 5.3 muestra la comparación entre los resultados obtenidos para cada uno de los casos.

Tabla 5.3 Comparación del desempeño del controlador difuso cambiando la condición inicial de los singleton

Singleton	MSE sinusoidal	MSE cuadrática	MSE Sinc
Cero	6.9	3.2	2.3
Uno	3.2	0.8	0.6

Distribuidos	2.4	0.15	0.4
Agrupados	3.2	0.4	0.6
mitad	2	0.05	0.28

Como puede observarse en la Tabla 5.3, el peor caso es en el que los singleton se inicializan en cero, y el mejor caso es en el que los singleton se encuentran en la mitad del discurso.

Para la tercera simulación en MATLAB se vuelve a hacer uso de las funciones objetivo mostradas en las Figuras 5.2, 5.3 y 5.4. Ahora el controlador difuso tiene tres funciones de membresía por entrada y nueve singleton cuyas posiciones iniciales se fijaron en la mitad del discurso. Los resultados para las tres funciones son mostradas en las Figuras 5.12, 5.13 y 5.14.

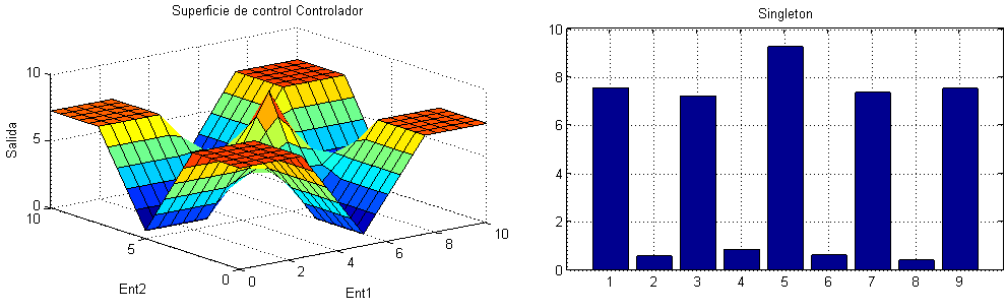

Figura 5.12. Superficie sinusoidal copiada y posición final de los singleton.

La Figura 5.12 muestra los resultados del entrenamiento del controlador difuso después de 3,000 iteraciones, ahora los grados de libertad del sistema son solamente nueve, que son el número que resulta de sumar los valles y crestas de la función copiada, cuando la función objetivo era de tipo sinusoidal.

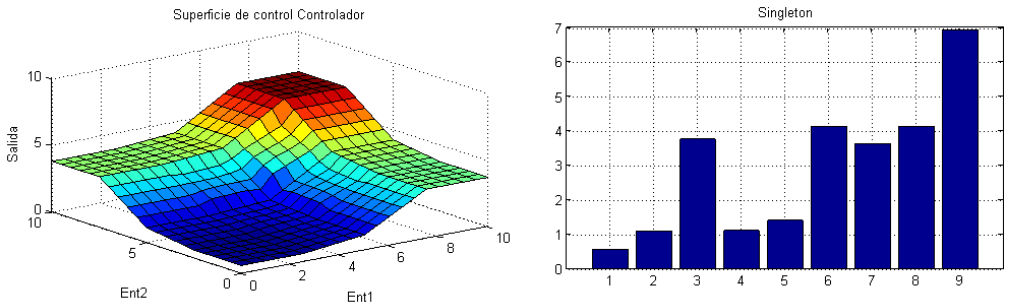

Figura 5.13. Superficie cuadrática copiada y posición final de los singleton.

Tal como se esperaba, la función cuadrática no presenta la misma pendiente suave de la mostrada en la Figura 5.10, los cambios abruptos que se perciben en la Figura 5.13 se debe que se tienen menos grados de libertad, la analogía de este comportamiento puede hacerse con la resolución de imágenes digitalizadas, mientras más pixeles contenga la imagen, mejor calidad presenta, pero el peso computacional es más caro.

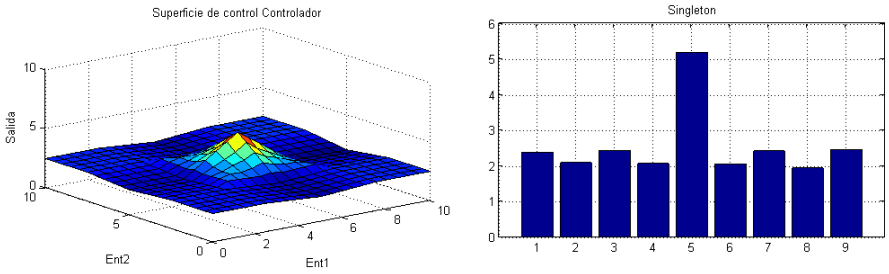

Figura 5.14. Superficie Sinc copiada y posición final de los singleton.

En la Figura 5.14 se presentan los resultados del entrenamiento del controlador, cuando la función objetivo es una función Sinc, a pesar de la pobre resolución en la salida del controlador, puede observarse un patrón establecido en la posición final de los singleton.

La tabla 5.4 muestra un resumen del error cuadrático medio normalizado obtenido para cada una de las tres funciones objetivo:

Tabla 5.4 Error Cuadrático Medio para las tres funciones objetivo

MSE sinusoidal	MSE cuadrática	MSE Sinc
0,24	0.04	0.07

Para la primera simulación en SPICE, se utilizaron las tres funciones objetivo de las Figuras 5.2, 5.3 y 5.4, así mismo, los singleton se fijaron en los valores finales obtenidos de la tercera simulación de MATLAB y permanecieron constantes durante toda la simulación. En este caso, se realizó una simulación en DC haciendo un barrido de la Entrada 1 de -1.65V a 1.65V en intervalos de 0.01V y de la Entrada 2 un barrido de -1.65V a 1.65V en intervalos de 0.65, los resultados se presentan a continuación:

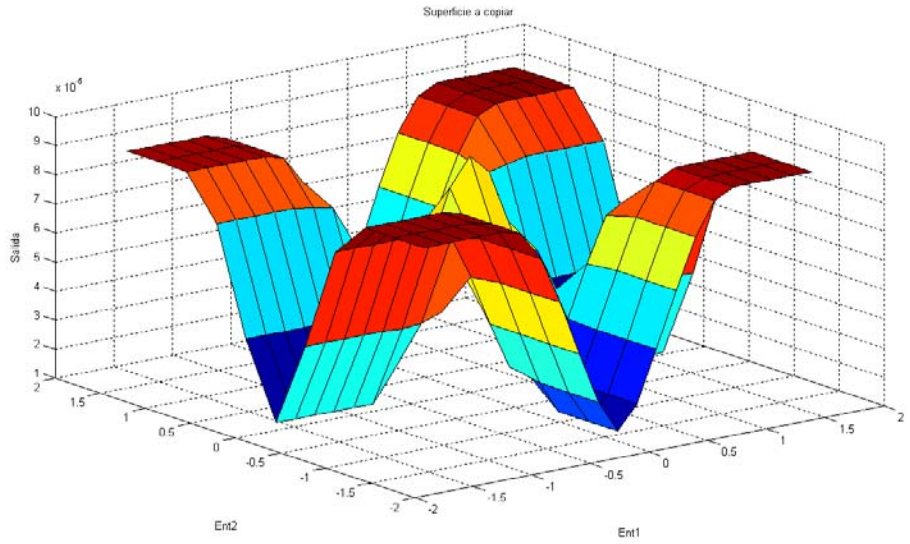

Figura 5.15. Superficie sinusoidal copiada por el controlador en SPICE.

Como puede observarse, los resultados obtenido con el controlador implementado a nivel transistor y simulado en SPICE (Figura 5.15) son comparables a los obtenidos con la simulación en MATLAB (Figura 5.12), en donde, por supuesto, durante la simulación todo se modela de forma ideal, no existen, como existen en SPICE, efectos no-ideales propios de los dispositivos. Debe notarse, sin embargo, que las escalas de las Figuras son diferentes, esto es porque en SPICE, el “1” tiene dimensiones de “micras”, es

decir, está multiplicado por un factor de (10^{-6}), debido a que la señal es procesada en modo corriente y el valor máximo para el que fue diseñado el controlador fue 100μA. De la misma forma, en los resultados en SPICE, el rango de las entradas es el que coincide con los voltajes de alimentación del circuito electrónico, es decir de -1.65V a 1.65V.

En la Figura 5.16 se muestra el resultado del controlador diseñado en SPICE cuando se tiene como función objetivo una función cuadrática, al igual que con la función sinusoidal, el desempeño es comparable al ideal, obtenido en MATLAB.

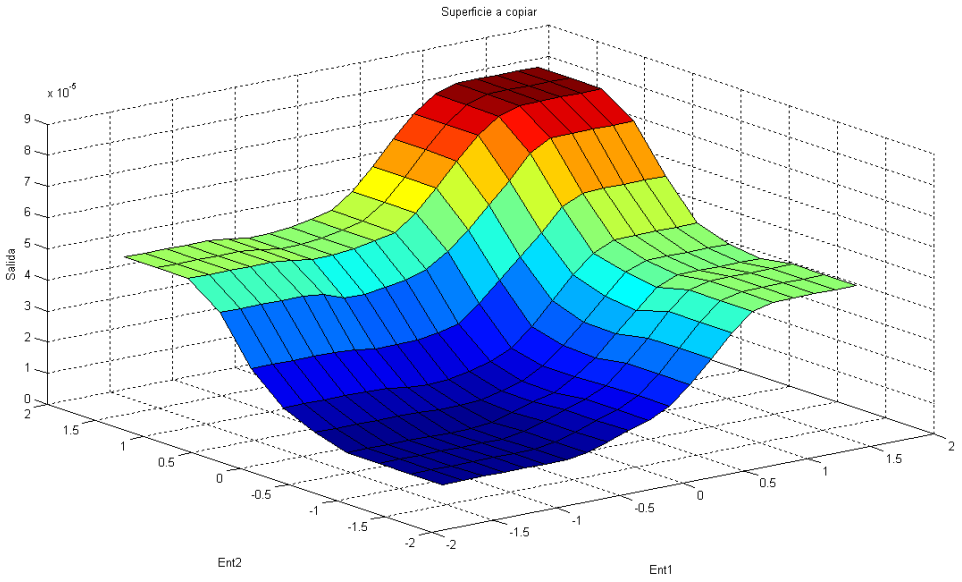

Figura 5.16. Superficie cuadrática copiada por el controlador en SPICE.

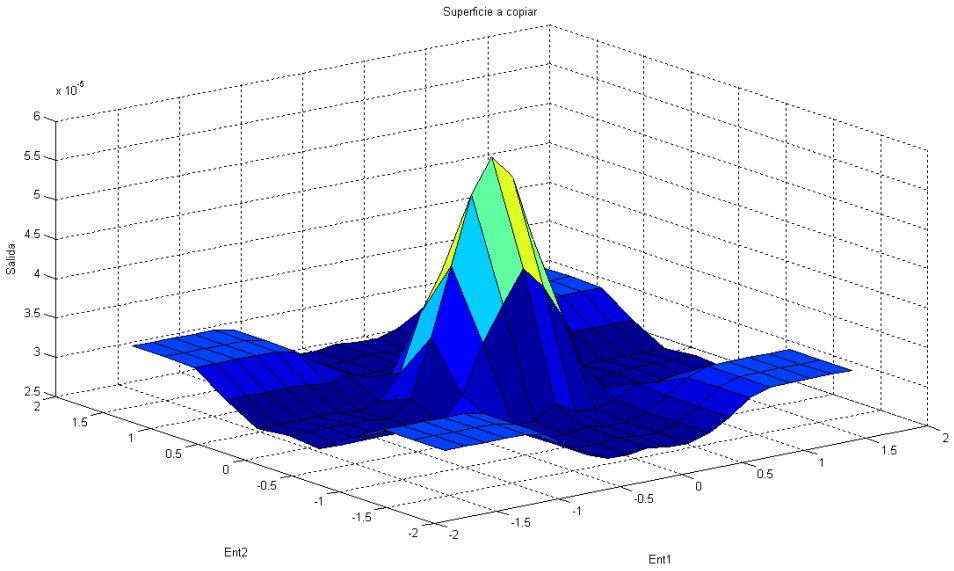

Figura 5.17. Función Sinc copiada por el controlador en SPICE.

En la Figura 5.17 se muestra la superficie copiada por el controlador en SPICE, cuando la función objetivo es de tipo Sinc.

Finalmente, se realizó la segunda simulación en SPICE, la cual consiste en entrenar al controlador difuso por medio de la implementación a nivel transistor del algoritmo Nonlinear Backpropagation. En este caso, se realizó una simulación en transitorio para poder modelar el comportamiento del sistema en el tiempo, al igual que en MATLAB, las entradas al sistema adaptable son dos fuentes de voltaje PWL que modelan el vector de ruido aleatorio necesario para la adaptación. Las funciones objetivo para este caso son las mostradas en las Figuras 5.2, 5.3 y 5.4.

La Figura 5.18 muestra el proceso de adaptación de los singleton en el tiempo.

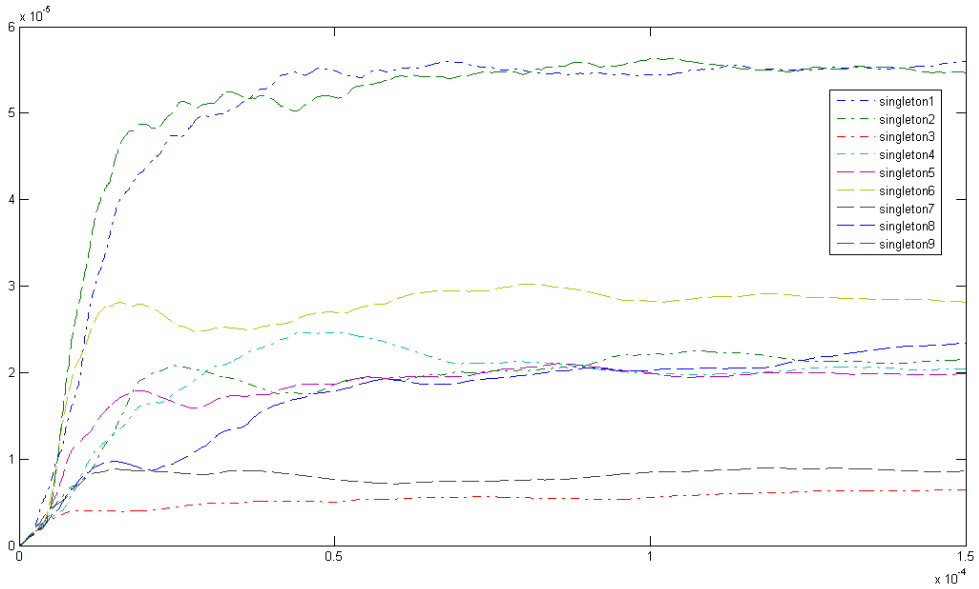

Figura 5.18 Adaptación de los singleton en el tiempo, para la función objetivo sinusoidal en SPICE.

Como puede observarse en la Figura 5.18, los singleton fueron inicializados para mostrar el peor caso, es decir, en cero. También puede observarse que el sistema llega a un punto cuasi-estable en aproximadamente 50μs.

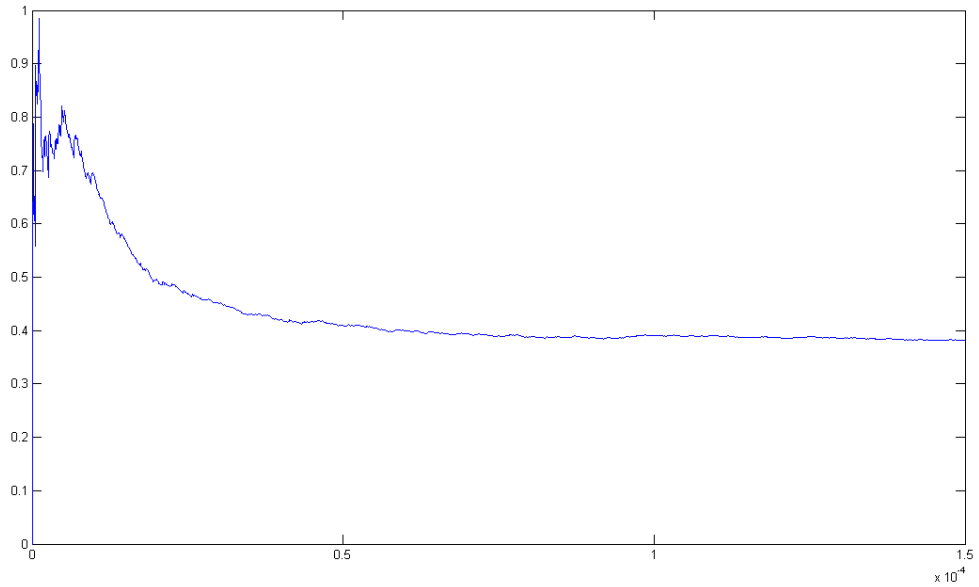

Figura 5.19 Error normalizado calculado en la adaptación del controlador difuso en SPICE para la función sinusoidal.

La Figura 5.19 muestra el comportamiento en el tiempo del error cuadrático medio normalizado cuando se tiene como objetivo la función sinusoidal.

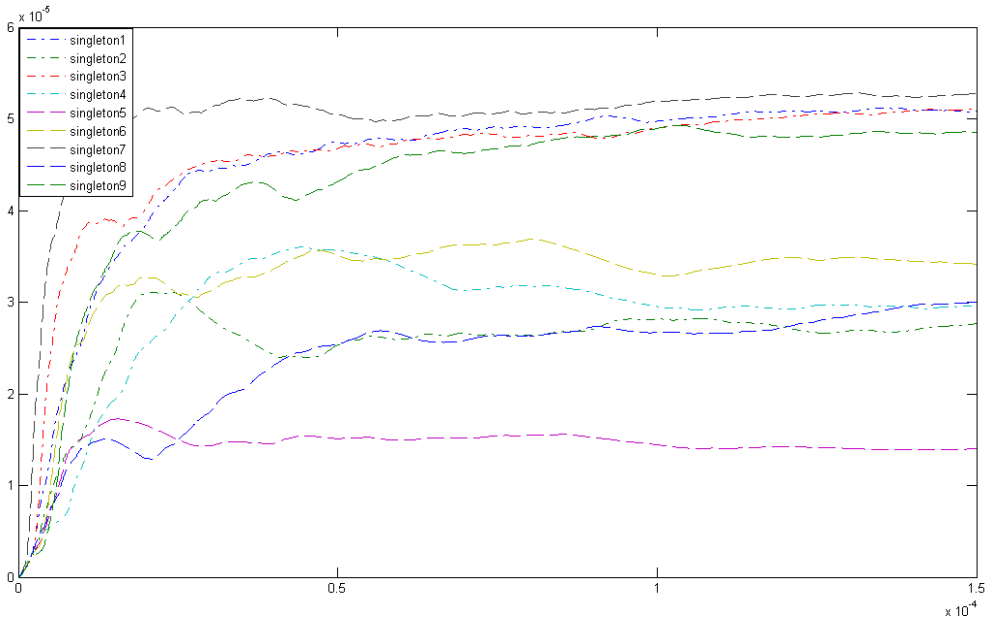

Figura 5.20 Adaptación de los singleton en el tiempo, para la función objetivo cuadrática en SPICE.

La Figura 5.20 muestra el comportamiento de los singleton en el transcurso del tiempo cuando la función objetivo es una función cuadrática, se puede ver que el tiempo en el que se alcanza el estado cuasi-estable es aproximadamente el mismo.

El error cuadrático medio normalizado para la función cuadrática es mostrado en la Figura 5.21.

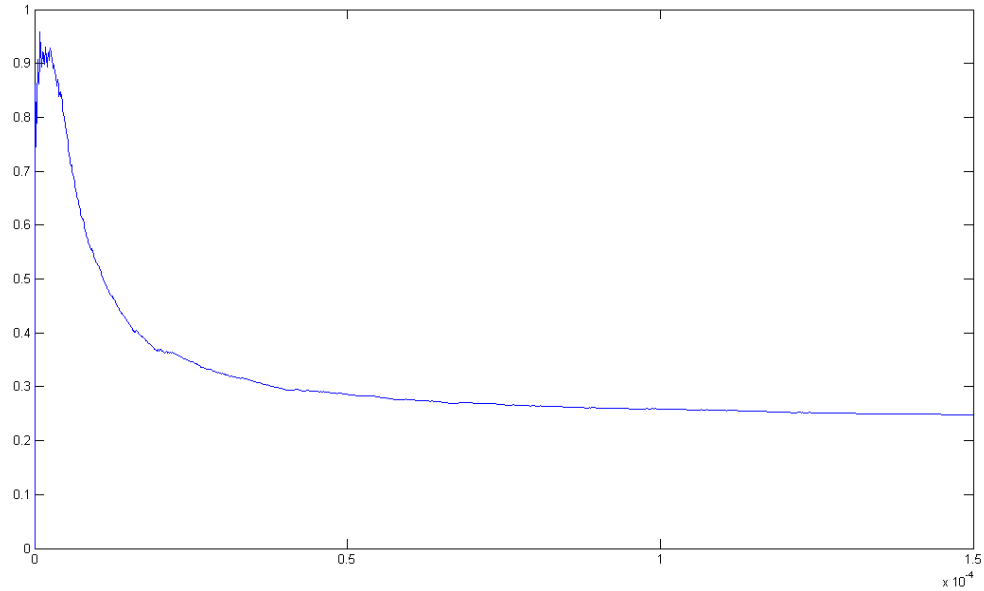

Figura 5.21 Error normalizado calculado en la adaptación del controlador difuso en SPICE para la función cuadrática

En la Figura 5.22 se muestra la adaptación de los singleton en el tiempo para el controlador difuso en SPICE cuando la función objetivo es la función Sinc. El error cuadrático medio normalizado para ésta función se muestra en la Figura 5.23

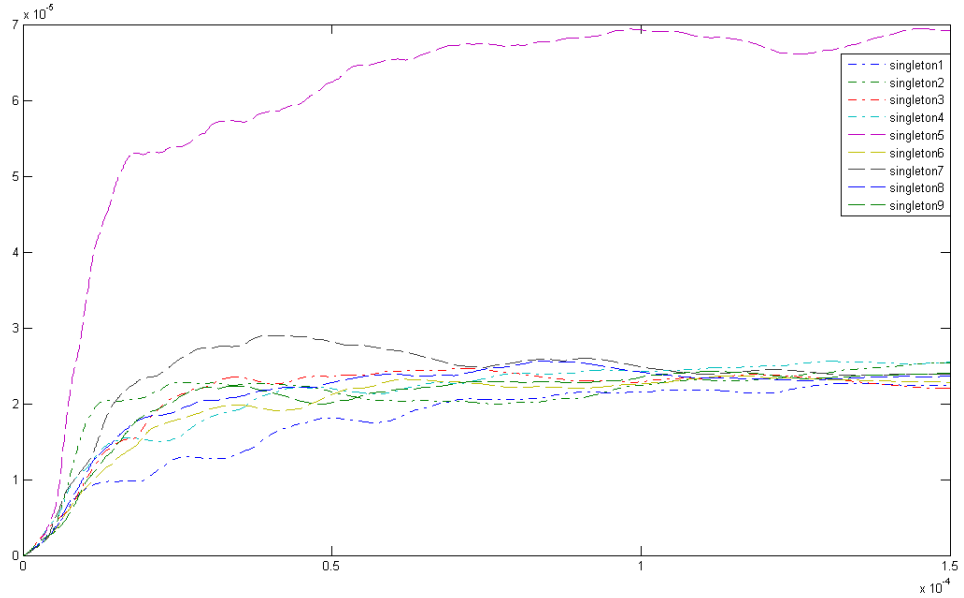

Figura 5.22 Adaptación de los singleton en el tiempo, para la función objetivo sinc en SPICE.

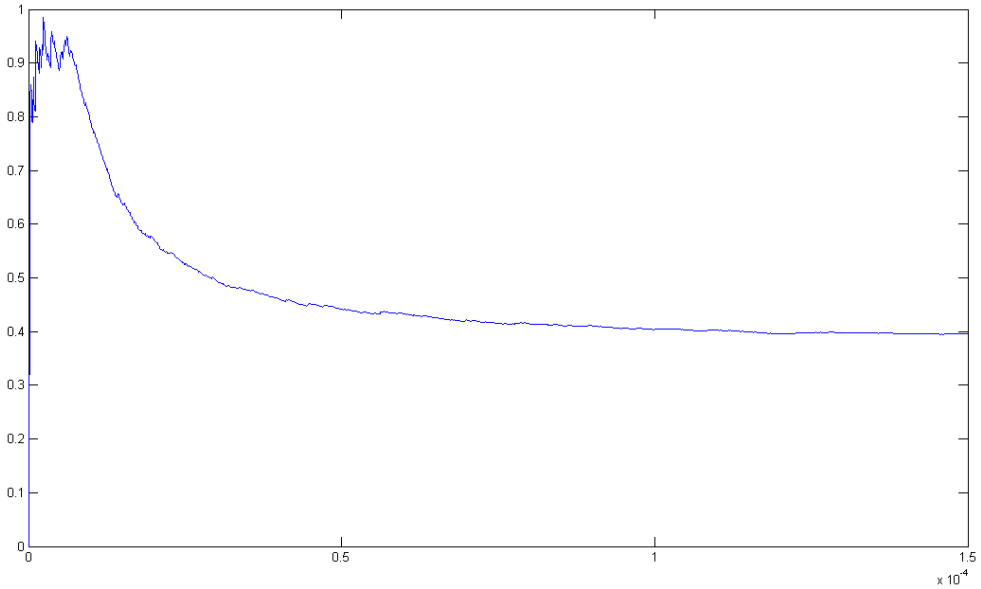

Figura 5.23 Error normalizado calculado en la adaptación del controlador difuso en SPICE para la función Sinc

Finalmente, en la Tabla 5.5 es mostrado un resumen comparativo de los resultados obtenidos, bajo las mismas condiciones, en SPICE y MATLAB.

Tabla 5.5 Comparación de resultados obtenidos en MATLAB y SPICE.

	S1	S2	S3	S4	S5	S6	S7	S8	S9	MSE
Sinusoidal MATLAB	57	18	6	21	33	21	5	21	58	0.24
Sinusoidal SPICE	48	22	5	22	32	22	5	22	51	0.39
Cuadrática MATLAB	48	35	43	34	32	35	45	32	50	0.04
Cuadrática SPICE	45	33	43	33	32	36	45	30	47	0.25
Sinc MATLAB	24	21	24	21	54	21	24	21	24	0.07
Sinc SPICE	22	24	22	24	69	23	24	24	25	0.4

En la Tabla 5.5 puede observarse que el desempeño del controlador a nivel transistor es comparable al del controlador modelado en MATLAB, el error que se presenta en la última columna de la tabla es el error cuadrático medio normalizado.

El desempeño del controlador difuso implementado a nivel transistor es de 256MFLIPS.

CAPÍTULO 6. CONCLUSIONES.

Los objetivos de éste trabajo fueron alcanzados satisfactoriamente. En el primer capítulo se hizo un análisis de los algoritmos más comunes y se comprobó que la implementación del algoritmo Nonlinear Backpropagation es el adecuado para entrenar sistemas difusos con una sola capa oculta en sistemas electrónicos VLSI, debido a que no es necesario calcular la derivada parcial de la función de costo del sistema, lo cual no es trivial en aplicaciones Hardware .

En el capítulo 3, se modelaron en MATLAB de forma ideal los bloques básicos que conforman tanto al controlador difuso tipo Takagi-Sugeno-Kang de orden cero como al algoritmo Nonlinear Backpropagation. Los bloques modelados fueron también caracterizados y se presentó de manera gráfica su comportamiento individual. En éste mismo capítulo, se proponen las estructuras completas y generalizadas tanto del controlador como del algoritmo de entrenamiento.

En el capítulo 4, se diseñaron, probaron y caracterizaron todos los bloque de construcción a nivel electrónico para la futura implementación, tanto del controlador difuso, como del algoritmo de entrenamiento, todo esto utilizando una tecnología de AMS 0.35µm. Se hizo un análisis de cada bloque y se justificó la elección de una u otra topología en cada caso. Se mostraron Tablas con las características de la simulación en SPICE, así como las gráficas resultantes de las simulaciones, algunas en el dominio del tiempo, otras en DC. Se demostró también que el uso de un bloque defuzzyficador utilizando circuitos normalizadores, presenta un desempeño adecuado y evita el uso de circuitos divisores, los cuáles requieren un gran esfuerzo de diseño para ser implementados.

Finalmente en el Capítulo 5, se presentan los resultados finales obtenidos tanto en MATLAB como en SPICE. Se presentan de manera

gráfica y resumida en forma de Tabla, todos y cada uno de los resultados obtenidos en ambos simuladores, se hizo una comparación con otro trabajo y se demostró que el desempeño del trabajo aquí propuesto fue mejor. Así mismo se hicieron varias simulaciones para comparar y encontrar el punto inicial adecuado para obtener el mejor desempeño del sistema adaptable. Se mostró también que la implementación del controlador difuso junto con el algoritmo Nonlinear Backpropagation en sistemas VLSI presenta un desempeño comparable al desempeño ideal esperado, y que es capaz de controlar plantas con comportamiento altamente no lineal, adaptando sus parámetros en intervalos de tiempo pequeños (50μs). El desempeño del controlador difuso implementado a nivel transistor es de 256MFLIPS.

Otra de las propuestas de éste trabajo fue implementar un sistema que permitiera hacer el cambio de valor de la constante de adaptación, haciendo que el algoritmo presente una rápida convergencia y, al mismo tiempo, una gran estabilidad numérica al alcanzar un punto fijo de operación. Ésta característica fue implementada tanto en MATLAB como en SPICE obteniendo los resultados esperados en ambos casos.

REFERENCIAS.

[1] Zadeh, L., “Fuzzy sets”, *Information and Control 8*, 1965, pp. 338-353.

[2] Mamdani A. “An Experiment in linguistic synthesis with a fuzzy logic controller”, *Int. J. Man-Machine Studies*, 7, 1975, 1-13.

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

[4] Patyra M. J., Grantner J. L., Koster K., “Digital Fuzzy Controller: Design and Implementation”, *IEEE Transactions on Fuzzy Systems,* Vol. 4, November 1996.

[5] Gabrielli A., Gandolfi E., “A Fast Digital Fuzzy Processor”, *IEEE MICRO,* January/February 1999, Vol. 19, Nol. 1, pp. 68-79.

[6] Rojas I., Pelayo F. J., Ortega O., Prieto A., “A CMOS implementation of fuzzy controllers based on adaptive membership function ranges”, 5th International Conference on Microelectronics for Neural Networks and Fuzzy Systems (MicroNeuro '96) p. 317.

[7] K. Shimizu, M. Osumi and F. Imae, Digital Fuzzy Processor FP-5000, in Proceedings of the 2nd International Conference on Fuzzy Logic and Neural Networks, Iizuka, Japan, pp. 539-542, 1992.

[8] H. Eichfeld, M. Löhner and M. Müller, Architecture of a Fuzzy Logic Controller with optimized memory organisation and operator design, in Proceedings of the 1st IEEE International Conference on Fuzzy systems, pp. 1317-1323, San Diego, 1992.

[9] T. Yamakawa and T. Miki, The Current Mode Fuzzy Logic Integrated Circuits Fabricated by the Standard CMOS Process, IEEE Transaction on Computers, vol. C-35, no. 2, pp. 161-167, 1986.

[10] Catania V., Puliafito A., Russo M., Vita L., "A VLSI Fuzzy Inference Processor Base don a Discrete Analog Approach", IEEE Transactions on Fuzzy Systems, Vol. 2, No. 2, May 1994.

[11] Çilingiroglu U., Pamir B., Günay S., Dülger F., "Sampled-Analog Implementation of Application-Specific Fuzzy Controllers", IEEE Transactions on Fuzzy Systems, Vol. 5, No. 3, August 1997.

[12] Manaresi N., Rovatti R., Franchi E., Guerrieri R., Baccarani G., "A Silicon Compiler of Analog Fuzzy Controllers: From Behavioral Specifications to Layout", IEEE Transactions on Fuzzy Systems, Vol. 4, No. 4, November 1996.

[13] Baturone, I., Sánchez-Solano S., Barriga A., Huertas J., "Implementation of CMOS Fuzzy Controllers as Mixed-Signal Integrated Circuits", IEEE Transactions on Fuzzy Systems, Vol. 5, No. 1, February 1997.

[14] Dulibe C., Jespers P., Verleysen M., “On designing Mixed-Signal Programmable Fuzzy Logic Controllers as Embedded Subsystems in Standard CMOS technologies”, SBCCI’2001 proceedings – 14th Symposium on Integrated Circuits and System Design, Pirenopolis (Brazil), 10-15 September 2001, pp. 194-200.

[15] Tong S., Li Y., “Direct Adaptive Fuzzy Backstepping Control for Nonlinear Systems”, First International Conference on Innovative Computing, Information and Control - Volume II (ICICIC'06), 2006, pp. 623.

[16] Phan P., Gale T., “Direct Adaptive Fuzzy Control with Less Restrictions on the Control Gain”, International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA'06), 2006 p. 168.

[17] Ji W., Li Q., Xu B., “Design Study of Adaptive Fuzzy PID Controller for LOS Stabilized System”, Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06), 2006 pp. 336-341.

[18] Azuara A., “Algoritmo LMS aplicado a la Adaptación en Línea de Controladores Analógicos Difusos en VLSI”, Tesis de Maestría, INAOEP, 2007.

[19] L. X. Wang y J. M. Mendel, “Fuzzy Basis Functions, Universal Approximation, and Orthogonal Least-Squares Learning”, IEEE Trans. Neural Networks, vol. 3, no. 5, pp. 807-814, 1992.

[20] L. X. Wang y J. M. Mendel, “Fuzzy Adaptive Filters, with Application to Nonlinear Channel Equalization”, IEEE Trans. Fuzzy Systems, vol. 1, no. 3, pp. 161-170, 1993.

[21] B. Widrow y M. E. Hoff, “Adaptive Switching Circuits”, IRE WECON Conv., vol. 4, pp. 94-104, 1960.

[22] Haykin, S., “Adaptive Filter Theory”, Prentice Hall, Englewood Cliff NJ, 1991.

[23] Proakis J., Rader M., Ling F., Nikias Ch., “Advanced Signal Processing”, McMillian Publishing Co., Singapure.

[24] Haykin, S., “Adaptive Filter Theory”, Prentice-Hall, Englewood Cliffs, NJ, 1986.

[25] werbos P., “Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences”, Ph.D. dissertation, Harvard, 1974.

[26] Rumelhart, D., Hinton, G., Williams R., “Learning representations by backpropagation erors”, *Nature,* 323:533-536, 1986.

[27] J. Eldredge and B. Hutchings. A Hardware Implementation of the Backpropagation Using Reconfigurable FPGA’s. In *IEEE World Conference on Computational Intelligence*, pages 77–80, June 1994.

[28] J. H. J.L. Beuchat and E. Sánchez. Hardware Reconfigurable Neural Networks. In *5th Reconfigurable Architectures Workshop (RAW’98)*, pages 91–98, Orlando, Florida, USA, 1998.

[29] P. Lysaght, J. Stockwood, J. Law, and D. Girma. Artificial Neural Network Implementation on a Fine-Grained FPGA. In R. Hartenstein and M. Z. Servit, editors, *Field- Programmable Logic: Architectures, Synthesis and Applications. 4th International Workshop on Field-Programmable Logic and Applications*, pages 421–431, Prague, Czech Republic, 1994. Springer-Verlag.

[30] X. Li and S. Areibi. A Hardware/Software Co-design Approach for Face Recognition. In *16th International Conference on Microelectronics*, pages 67–70, Tunis, Tunisia, December 2004.

[31] Hertz J., Krogh A., Lautrup B., Lehman T, "Nonlinear Backpropagation: Doing Backpropagation without derivatives of the Activation Function", IEEE Transactions on Neural Networks, Vol. 8, No. 6, November 1997.

[32] Krogh A., Thorbergsson G. I., Hertz J. A., "A cost function for internal representations", Advances in Neural Inform. Processing Syst., D. S. Touretzky, Ed., Denver, CO, 1989, Vol. 2, pp. 733-740.

[33] López J., "Diseño Orientado a Síntesis, de Circuitos Analógicos de Bajo Voltaje para Aplicaciones de Lógica Difusa", Tesis de Maestría, INAOEP, 2004.

[34] Marshall G., Collins S., "Fuzzy Logic Architecture Using Subthreshold Analogue Floating-Gate Devices", IEEE Transactions on Fuzzy Systems, Vol. 5, No.1, February 1997.

[35] Mead C., "Analog VLSI and Neural Systems" Addison Wesley, 1989.

[36] Kettner T., Heite C., Schumacher K. " analog CMOS realization of fuzzy logic membership functions", IEEE J. Solid-State Circuits, Vol. 28, pp. 857-861, July 1993.

[37] Huertas J. L., Sánchez-Solano S., Baturone I., Barriga A., "Integrated circuit implementation of fuzzy controllers", IEEE J. Solid-State Circuits, Vol. 31, pp. 1051-1058, July 1996.

[38] Gilbert B., "Current-mode circuits from a translinear view point: A tutorial", Analogue IC Design: in The Current-Mode Approach, Toumazou C., Lidgey F. J., Haigh D. G., Eds. London, UK: Peter Peregrinus Ltd., 1990.

[39] *Ahmadi* S., L. **Sellami,** and R **W.** Newcomb, A CMOS PWL Fuzzy Membenhip Fundon, IEEE *Ituemwnal Symposium on Circuits and Sr:nems,* Seattle WA, vol. 3, pp., 2321-2324, May 1995.

[40] Ota Y., B.M.Wilamowski, Analog Hardware Implementation of a Voltage-Mode Fuzzy Min-Max Controller, *Journal of* Circuits, *Systems, and Computers,* Vol. 6, No.2, pp. 171-184, 1996.

[41] Sandler S., Hymowitz C., "SPICE Circuit Handbook", McGraw-Hill.

[42] Watanabe H., Chen D., Konuri S., "Evaluation of Min/Max Instructions for Fuzzy Information Processing", IEEE Transactions on Fuzzy Systems, Vol. 4, No. 3, August 1996.

[43] Lazzaro J., Ryckebusch S., Mohawald M.A. and Mead C., "Winner-take-all networks of O(n) complexity", in Adavances in Neural Information Processing Systems, Vol. 1, D.S. Touretzky, Ed. Los Altos, CA: Morgan Kaufmann, 1989, pp. 703-711.

[44] Dualibe C., Verleysen M., Jespers P., “Design of Analog Fuzzy Logic Controllers in CMOS Technologies Implementation, Test and Application”, Kluwer Academic Publishers, Boston/Dordrecht/London, 2003, pp.66-74.

[45] Wiegerink R., “Analysis and Synthesis of MOS Translinear Circuits”, Kluwer Academic Publishers, Boston/Dordrecht/London, pp. 110-123.

[46] Gilbert B., “Translinear Circuits: A proposed Classification”, en Electronic Letters,11(1), pp. 14-16, 1975, errata 11(6), p. 136.

[47] Weihsing L., Shen-Iuan L. “Low-Voltage CMOS Voltage-Mode Divider and Its Application.

[48] Weihsing Liu, Shen-Iuan Liu*, Senior Member, IEEE*, and Shui-Ken Wei, “CMOS Current-Mode Divider and Its Applications” IEEE Transactions On Circuits And Systems—II: Express Briefs, Vol. 52, No. 3, March 2005.

[49] Dualibe C., Verleysen M., Jespers P., “Two-quadrant CMOS analogue divider”, Electronics Letters, 11th june 1998, Vol. 34, No.12”.

[50] Wilcock R., “A CAD Methodology for Switched Current Analogue IP Cores”, Ph.D. dissertation, University of Southampton, UK, 2003.

[51] J. B. Hughes, "Analogue Techniques for Very Large Scale Integrated Circuits," in *Electronics and Computer Science*. Southampton: University of Southampton, 1992.

[52] S. Matsuzaki and I. Kondo, "Information holding apparatus." UK: 1359105, 1972], pero fue utilizado por primera vez para filtros en modo corriente en 1987[N. C. Bird, "Storing sampled analogue electrical currents." UK: 2209895, 1987.

[53] Azuara A., “LMS Adaptation Scheme for Analog Controllers Based on Fuzzy Logic”, 2006 International Conference on Electronic Design. 2006

ÍNDICE DE FIGURAS.

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS.