INGENIERÍA ELECTRÓNICA

Implementación *Hardware* del algoritmo Karnik-Mendel mejorado basada en operadores CORDIC

Linda K. Duran[*§], Miguel A. Melgarejo[*]
**Laboratorio de Automática, Microelectrónica e Inteligencia Computacional*
Universidad Distrital Francisco José de Caldas, Bogotá, Colombia
§ e-mail: lkduran@ieee.org

(Recibido: Noviembre 13 de 2008 - Aceptado: Diciembre 01 de 2009)

Resumen

Este artículo presenta una propuesta de arquitectura para calcular el centroide generalizado de un conjunto difuso tipo dos de intervalo utilizando el algoritmo Karnik-Mendel mejorado. Se adapta la arquitectura para calcular el producto y el cociente basados en el algoritmo CORDIC. Además, se describen dos implementaciones *hardware* de la arquitectura propuesta. La primera utiliza operadores paralelos convencionales, mientras que la segunda incluye los operadores basados en el algoritmo CORDIC. Se llevan a cabo las dos implementaciones sobre tecnología *Field Programmable Gate Array* y se validan calculando el centroide de doce conjuntos difusos tipo dos de intervalo determinísticos. Se estiman algunos índices de desempeño de las implementaciones como área y frecuencia con el fin de establecer un marco de comparación.

Palabras Claves: Huella de incertidumbre, Conjunto difuso tipo uno, Conjunto difuso tipo dos, Algoritmo Karnik-Mendel mejorado, Centroide generalizado, Algoritmo CORDIC.

ELECTRONICAL ENGINEERING

Hardware Implementation of an enhanced Karnik-Mendel algorithm based on CORDIC operators

Resumen

This paper presents an architectural proposal for computing the generalized centroid of an interval type two fuzzy set using the enhanced Karnik Mendel algorithm. The architecture is adapted so that CORDIC based operators can be used to compute multiplications and divisions. In addition, two hardware implementations of the proposed architecture are described. The first one uses conventional parallel operations, while the latter includes CORDIC based operators. Both implementations are achieved over Field Programmable Gate Array technology. They are validated by computing the centroid of twelve deterministic interval type-2 fuzzy sets. Performance indices like area and maximum operation frequency are obtained in order to establish a comparative framework.

Keywords: Foot print of uncertainty, Type one fuzzy set, Type two fuzzy set, Enhanced Karnik-Mendel algorithm, Generalized centroid, CORDIC algorithm.

1. Introducción

Los conjuntos difusos tipo dos (T2-FS) se introdujeron por Zadeh en 1975 como una extensión de los conjuntos difusos tipo uno (T1-FS) Karnik & Mendel (2001), los T2-FS permiten modelar y minimizar los efectos de las incertidumbres en los sistemas de lógica difusa (FLS), Mendel et al. (2006). Los conjuntos difusos tipo dos de intervalo (IT2-FS) son un caso particular de los T2-FS. El cálculo práctico de las operaciones en un T2-FLS es computacionalmente viable si todos los T2-FS son IT2-FS Mendel & Liu (2007).

Utilizar sistemas difusos tipo dos de intervalo (IT2-FLS) permite obtener algunas ventajas que al emplear sistemas de lógica difusa tipo uno (T1-FLS), Hagras (2007), algunas de estas son: (1) Los FLS basados en T2-FS pueden tener mejor rendimiento que los FLS tipo uno (T1-FLS) dado que, los IT2-FS permiten modelar y manejar incertidumbres lingüísticas y numéricas asociadas a las entradas y salidas de los FLS. (2) Utilizar IT2-FS para representar las entradas y salidas de un FLS permite reducir la base de reglas del FLS, con respecto al uso de T1-FLS. (3) Los IT2-FLS permiten obtener salidas que no pueden calcularse al utilizar T1-FLS con el mismo número de funciones de pertenencia.

Algunas de las aplicaciones en las que los IT2-FLS han mostrado obtener mejor desempeño que al emplear T1-FLS son: Biometría multimodal Hidalgo et al. (2008), Clustering, Rhee (2007), Modelamiento de series de tiempo económicas, Hernández & Méndez (2006), Robótica móvil, Hagras (2004), Ecualización de canales, Liang & Mendel (2000) e Información geográfica, Fisher (2007).

La Figura 1 presenta el modelo de un FLS, las entradas puntuales son fusificadas en IT2-FS, donde se le asignan pesos a las entradas de acuerdo con funciones de pertenencia previamente establecidas. Los IT2-FS obtenidos en la fusificación activan la base de reglas y el motor de inferencia, la base de reglas está representada por IT2-FS y constituyen el corazón del FLS, el motor de inferencia combina las reglas activas en IT2-FS de salida. El procesamiento de la salida permite calcular un valor puntual a partir de los conjuntos difusos obtenidos en la etapa de inferencia. En un T2-FLS este procesamiento tiene dos etapas, la primera, denominada reducción de tipo, transforma un conjunto difuso tipo dos (T2-FS) en un conjunto difuso tipo uno (T1-FS). La segunda, calcula la salida puntual del sistema, a partir del T1-FS obtenido en la reducción de tipo.

Existen diferentes métodos para calcular la reducción de tipo, la mayoría de ellos están fundamentados en el cálculo del centroide de un IT2-FS. En Karnik & Mendel (2001) se propone un método exacto para calcular el centroide de un IT2-FS, conocido como algoritmo Karnik-Mendel (KM). En la actualidad este método es ampliamente utilizado en las aplicaciones, Mendel (2007). Por ejemplo en Lynch et al. (2007) y Hagras (2004) se implementa un controlador difuso tipo dos de intervalo (IT2-FLC) en el que la reducción de tipo se calcula utilizando el algoritmo KM. En estas aplicaciones se evidencia que la complejidad computacional del algoritmo KM, debida a su naturaleza iterativa, restringe su uso en aplicaciones en tiempo real y eleva su costo *hardware*. En Wu & Mendel (2002) se presenta el método de las formas cerradas de Wu-Mendel, el cual se caracteriza por ser no iterativo y permite estimar de manera aproximada la incertidumbre contenida en la salida de un T2-FLS, evitando el cálculo de la reducción de tipo. La implementación *hardware* de las formas cerradas de Wu-Mendel puede resultar estructuralmente compleja, de acuerdo con el número de T2-FS presentes a la salida de la etapa de inferencia de un T2-FS Melgarejo & Peña (2007).

Con el fin de atacar la complejidad computacional al calcular el centroide de un IT2-FS se han propuesto algunos métodos entre estos el método iterativo para el cálculo del centroide de un IT2-FS Melgarejo (2007) y el algoritmo Karnik-Mendel mejorado (EKM), Wu & Mendel (2007). El método iterativo permite calcular el centroide de un IT2-FS reutilizando una búsqueda exhaustiva sobre el universo discurso de la función centroide

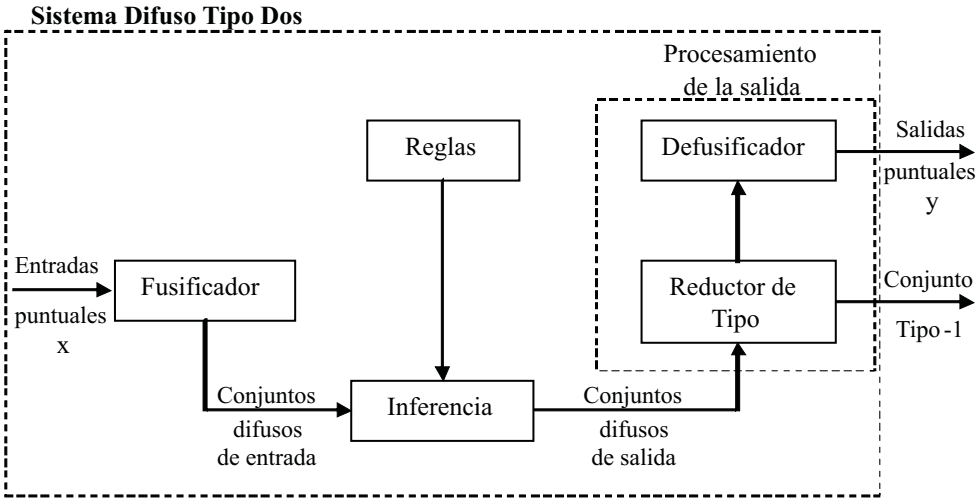


Figura 1. Modelo de un T2-FLS Mendel (2001)

y demuestra ser más rápido que el algoritmo KM. El algoritmo EKM realiza una inicialización óptima del algoritmo KM y al igual que el método iterativo, resulta ser más rápido que su antecesor.

Este artículo presenta una propuesta arquitectural para la implementación *hardware* del cálculo del centroide de un IT2-FS empleando el algoritmo EKM. Esta propuesta arquitectural se implementa sobre *Field Programmable Gate Array* (FPGA) utilizando operadores aritméticos paralelos, para mejorar el desempeño en área de la implementación se explora el cálculo del producto y el cociente utilizando el algoritmo *Coordinate Rotation Digital Computer* (CORDIC) en coordenadas lineales, Volder (1959). La implementación se realiza a resolución de 16 bits en los datos de entrada y salida sobre la FPGA Virtex-E 600 de Xilinx® y manejando hasta 16 puntos de discretización del universo discurso para el cálculo del centroide.

2. Conjuntos difusos tipo dos

A continuación se presenta un resumen de algunos elementos conceptuales acerca de la teoría de los conjuntos difusos tipo dos.

2.1 Conjunto difuso tipo dos

De acuerdo con Mendel (2001), un conjunto difuso tipo dos, que se denota como Ã, se caracteriza por medio de una función de pertenencia tipo dos $\mu_{\tilde{A}}(x,u)$, donde $x \in X$ y $u \in J_x \subseteq [0,1]$.

$$\tilde{A} = \left\{ \left((x,u), \mu_{\tilde{A}}(x,u) \right) \middle| \forall x \in X, \forall u \in J_x \subseteq [0,1] \right\} \quad (1)$$

en donde $0 \leq \mu_{\tilde{A}}(x,u) \leq 1$.

2.2 Función de pertenencia secundaria

Para cada valor de x, $x = x'$, el plano 2-D cuyos ejes son u y $\mu_{\tilde{A}}(x',u)$ se denomina como una porción vertical de $\mu_{\tilde{A}}(x,u)$. Una función de pertenencia secundaria es una porción vertical de $\mu_{\tilde{A}}(x,u)$. Esto es $\mu_{\tilde{A}}(x = x',u)$ para todo $x \in X$ y $\forall u \in J_{x'} \subseteq [0,1]$

$$\mu_{\tilde{A}}(x = x', u) \equiv \mu_{\tilde{A}}(x) = \int_{u \in J_{x'}} f_{x'}(u) / u \quad J_{x'} \subseteq [0,1] \quad (2)$$

En donde $0 \leq f_{x'}(u) \leq 1$.

La función de pertenencia tipo dos mostrada en la Figura 2 tiene cinco divisiones verticales, para cada valor de x diferente de cero. La función de pertenencia secundaria en $x = 4$ es

$$\mu_{\tilde{A}}(4) = a/0 + b/0.2 + c/0.4 + d/0.6 + e/0.8$$

La unión de las cinco funciones de pertenencia secundarias en x= 1,2,3,4,5 es $\mu_{\tilde{A}}(x,u)$. Las funciones de pertenencia primarias son $J_1 = J_2 = J_4 = J_5 = \{0,0.2,0.4,0.6,0.8\}$ y $J_3 = \{0.6,0.8\}$.

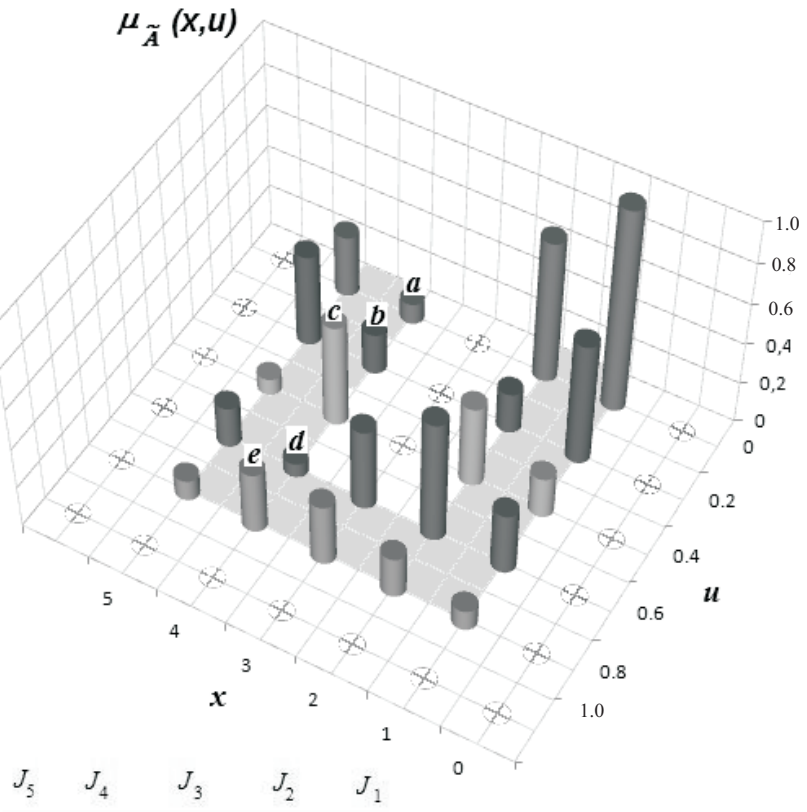


Figura 2. Conjunto difuso tipo dos

2.3 Conjunto difuso tipo dos de intervalo

La amplitud de una función de pertenencia secundaria se conoce como grado secundario. En Ec. (2), $f_x(u)$ corresponde al grado secundario, mientras que en Ec. (1), $\mu_{\tilde{A}}(x',u')(x' \in X, u' \in J_x)$ es un grado secundario. En la Figura 2 cada una de las líneas verticales representa a $\mu_{\tilde{A}}(x,u)$ para un par específico (x,u) y su amplitud corresponde al grado secundario.

Se dice entonces que un conjunto difuso tipo dos es de intervalo (IT2-FS) cuando todos sus grados de pertenencia secundarios son iguales a uno. Un IT2-FS queda completamente caracterizado por su huella de incertidumbre FOU, por sus siglas en inglés. La FOU corresponde a una región en dos dimensiones, tal como lo ilustra la Figura 3, que está acotada por una función de pertenencia superior $\overline{f}_{(x)}$ y una función de pertenencia inferior $\underline{f}_{(x)}$ En el interior de la FOU se encuentran todos los conjuntos difusos que conforman el IT2-FS, los cuales se denominan conjuntos difusos tipo uno empotrados, Mendel et al. (2006).

2.4 Centroide de un conjunto difuso tipo dos de intervalo

Sea Ã un IT2-FS, su centroide $c(\tilde{A})$ es un intervalo dado por Mendel (2001), Mendel (2007):

$$c\left(\tilde{A}\right) = 1/\left[y_l\left(\tilde{A}\right), y_r\left(\tilde{A}\right)\right] = 1/[y_l, y_r] \quad (3)$$

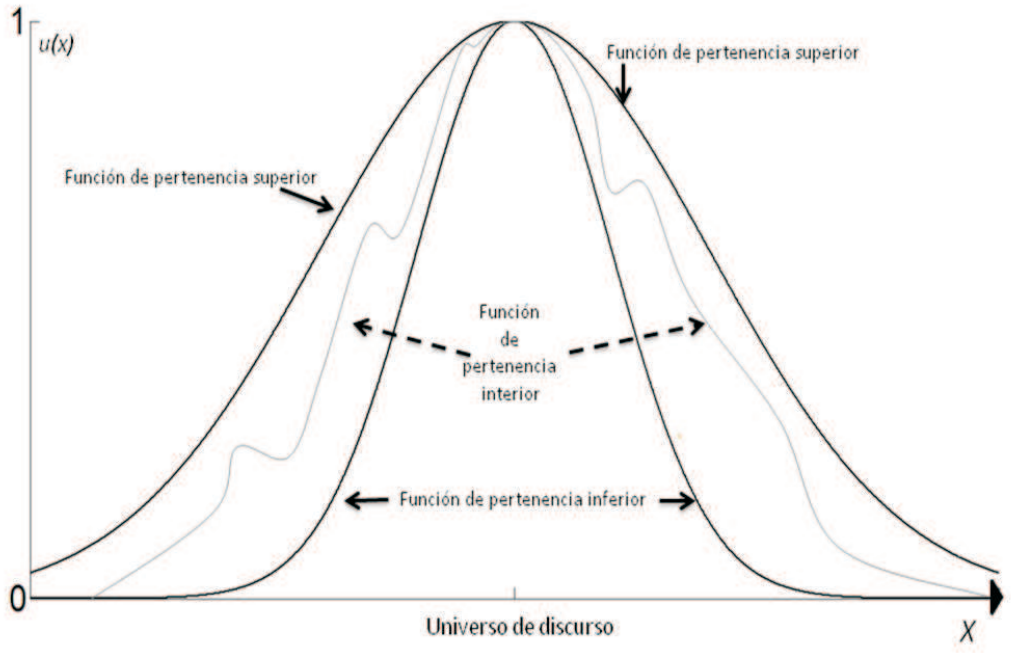


Figura 3. Huella de incertidumbre, FOU.

siendo y_l e y_r respectivamente las soluciones de los siguientes problemas de optimización:

$$y_l = \min_{\forall f_i \in [\underline{f}_i, \overline{f}_i]} \frac{\sum_{i=1}^{N} x f_i}{f_i} \quad (4)$$

$$y_r = \max_{\forall f_i \in [\underline{f}_i, \overline{f}_i]} \frac{\sum_{i=1}^{N} x f_i}{f_i} \quad (5)$$

donde el universo de discurso X al igual que las funciones de pertenencia superior e inferior han sido discretizadas en N puntos.

Una forma más sencilla de entender el centroide un IT2-FS es considerarlo como el intervalo formado por todos los centroides de los conjuntos difusos tipo uno empotrados contenidos en la huella de incertidumbre, Mendel (2007). Claramente, un intervalo queda completamente caracterizado por los puntos máximo y mínimo que lo limitan.

2.5 Algoritmo Karnik-Mendel Mejorado

Al calcular el centroide de todos los T1-FS empotrados en la FOU se obtendría un conjunto de puntos en un intervalo. No obstante, el número de T1-FS contenidos en la FOU es no contable, por esta razón, calcular el centroide de cada uno de estos conjuntos sería computacionalmente inviable. Sin embargo, existen dos conjuntos difusos tipo uno especiales empotrados en la FOU, uno de ellos provee el centroide mínimo y_l y el otro el centroide máximo y_r,. Por tanto, calcular centroide de un IT2-FLS consiste en encontrar solo los valores extremos del conjunto reducido, y_l y y_r . En Mendel (2001) se muestra que los centroides y_l y y_r pueden expresarse según Ec. (6) y Ec. (7):

$$y_l = \frac{\sum_{i=1}^{L} x_i \overline{f}_i + \sum_{i=L+1}^{M} x_i \underline{f}_i}{\sum_{i=1}^{L} \overline{f}_i + \sum_{i=L+1}^{M} \underline{f}_i} \quad (6)$$

$$y_r = \frac{\sum_{i=1}^{R} x_i \underline{f}_i + \sum_{i=R+1}^{M} x_i \overline{f}_i}{\sum_{i=1}^{R} \underline{f}_i + \sum_{i=R+1}^{M} \overline{f}_i} \quad (7)$$

donde $\overline{f}_i$ y $\underline{f}_i$ son las funciones de pertenencia superior e inferior de la FOU y x_i son los puntos de discretización del universo de discurso. L es el punto de cambio entre $\overline{f}_i$ y $\underline{f}_i$ y R es el punto de cambio entre $\overline{f}_i$ y $\underline{f}_i$. La idea clave es encontrar L y R, los cuales pueden calcularse independientemente. Sin embargo, no existen fórmulas cerradas para hallarlos. Con el fin de dar mayor claridad sobre las Ec.(6) y Ec.(7), la Figura 4 muestra un ejemplo de los conjuntos difusos tipo uno empotrados para calcular los centroides mínimo y máximo al igual que los respectivos puntos de corte L y R.

El algoritmo Karnik-Mendel (KM), Mendel (2001), se propuso para determinar los puntos de cambio L y R, su costo computacional es alto dado que se trata de un método iterativo y además necesita de dos a seis iteraciones para converger Mendel & Liu (2007). Para sopesar el costo computacional del algoritmo KM, se propuso posteriormente el algoritmo Karnik-Mendel Mejorado (EKM), Wu & Mendel (2007), el cual consiste en utilizar puntos óptimos de inicialización para el algoritmo KM que permitan reducir el número de iteraciones necesarias para convergencia. Utilizar los puntos óptimos de inicialización le permite converger al algoritmo en una iteración con una probabilidad de 97%, cuando se tienen entre tres y cien puntos de discretización, Wu & Mendel (2007).

A continuación se presenta el algoritmo EKM para el cálculo del centroide mínimo.

1. Hacer $k = [M/2.4]$, donde k el entero más cercano a M / 2.4 y M corresponde al número de puntos de discretización con los que es representada la FOU y calcular:

$$a = \sum_{i=1}^{k} \underline{x}_i \overline{f}_i + \sum_{i=k+1}^{M} \underline{x}_i \underline{f}_i \quad (8)$$

$$b = \sum_{i=1}^{k} \overline{f}_i + \sum_{i=k+1}^{M} \underline{f}_i \quad (9)$$

$$y = a / b \quad (10)$$

2. Encontrar $k' \in [1, M-1]$ tal que $\underline{x}_{k'} \le y \le \underline{x}_{k'+1}$

3. Verificar si $k' = k$. Si esto se cumple, parar, asignar $y_l = y$ y llamar k a L. Si no se cumple continuar.

4.Calcular $s = sign(k' - k)$

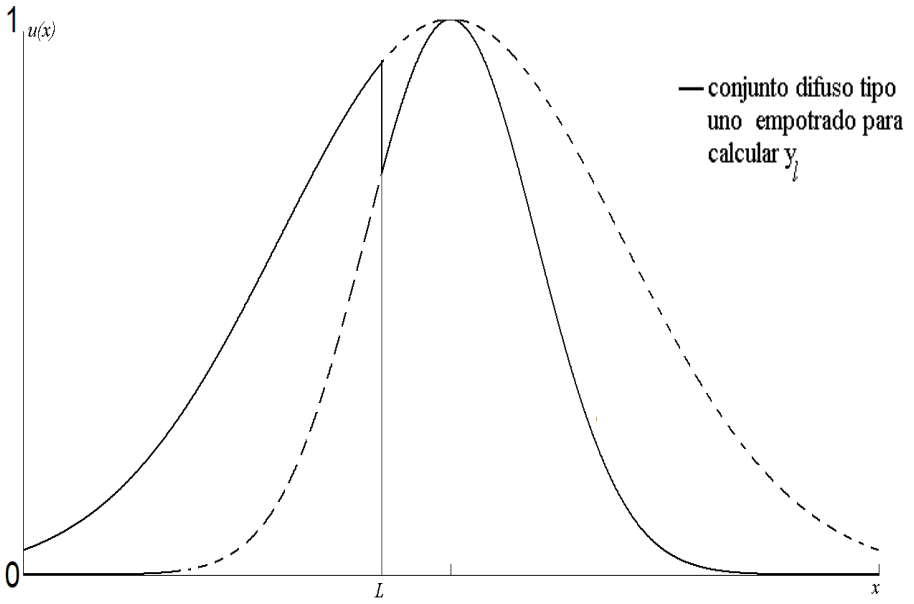
$$a' = a + s \sum_{i=\min(k,k')+1}^{\max(k,k')} \underline{x}_i \left(\overline{f}_i - \underline{f}_i \right) \quad (11)$$

$$b' = b + s \sum_{i=\min(k,k')+1}^{\max(k,k')} \left(\overline{f}_i - \underline{f}_i \right) \quad (12)$$

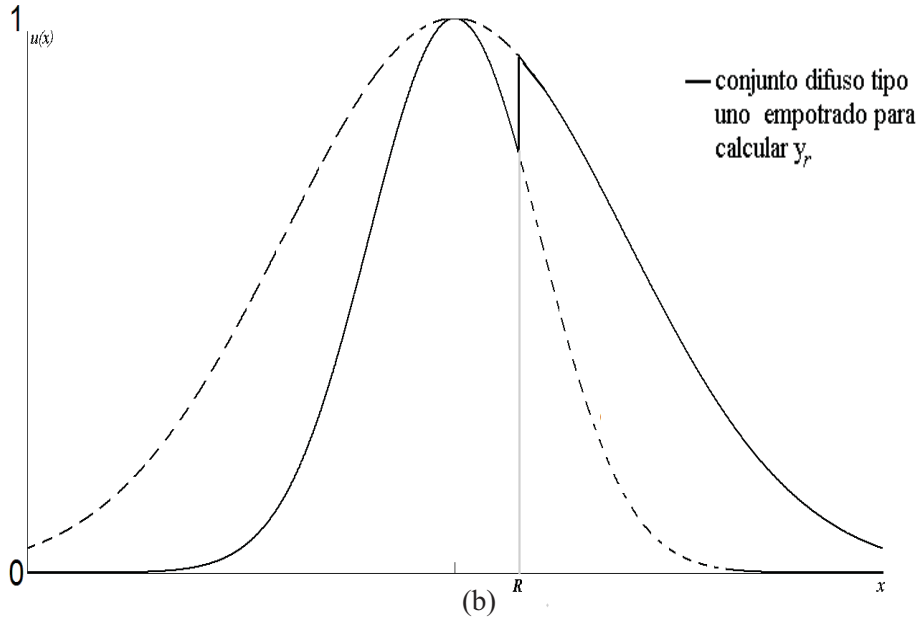
$$y' = a'/b' \quad (13)$$

Asignar $y = y'$, $a = a'$, $b = b'$ y $k = k'$. Ir al paso 2

Se debe tener en cuenta que el soporte de la FOU $\underline{x}_i (i = 1, \ldots, M)$ debe estar organizado en forma ascendente tal que $\underline{x}_1 \le \underline{x}_2 \le \cdots \le \underline{x}_M$ asegurando que cada peso de las funciones de pertenencia superior e inferior, $\overline{f}_i$ y $\underline{f}_i$ corresponda a su respectivo $\underline{x}_i$ Los pasos uno a tres calculan la primera iteración del algoritmo, si la condición del paso tres se cumple, se ha encontrado el punto de cambio L, lo cual indica que se ha calculado satisfactoriamente el centroide. Si la condición del paso tres no se



(a)



(b)

Figura 4. Ilustración de los conjuntos difusos tipo uno empotrados para calcular (a) y_l, (b) y_r

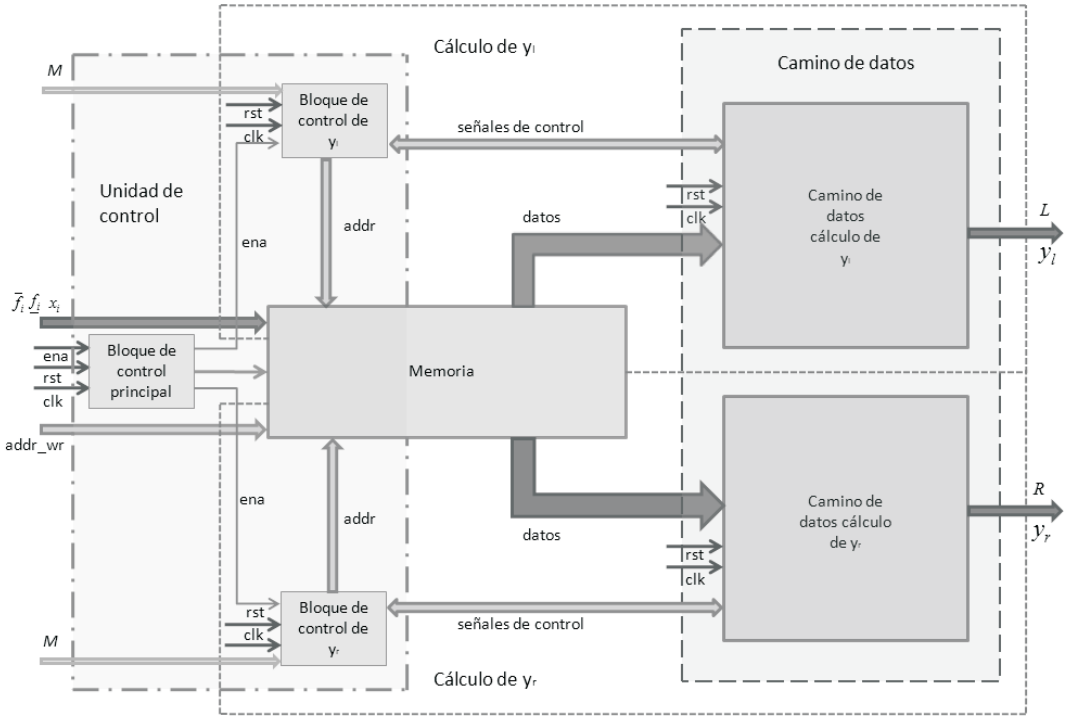


Figura 5. Arquitectura hardware propuesta para el cálculo del centroide de un IT2-FLS.

cumple, es necesario re-calcular el centroide, variando el punto de cambio auxiliar *k,* hasta encontrarlo. El algoritmo para calcular el centroide máximo y_r es similar al presentado en Ec. (8) a (13), pero se tiene en cuenta la Ec. (7) y se inicializa el algoritmo utilizando el entero más cercano a M/ 1.7, Wu & Mendel (2007).

3. Arquitectura propuesta para el cálculo del centroide de un IT2-FS utilizando el algoritmo Karnik-Mendel mejorado

Para realizar la propuesta arquitectural del algoritmo EKM, se considera que este permite calcular el centroide mínimo y máximo en paralelo, Wu & Mendel (2007). Además, se aprovecha la similitud de las ecuaciones del modelo computacional del algoritmo EKM para compartir recursos *hardware* mejorando así el rendimiento en área de la implementación de la arquitectura. En la 5 se presenta la arquitectura propuesta para el cálculo del EKM. En ella se tienen tres unidades que son, el camino de datos, la unidad de control y la memoria. Las entradas corresponden al soporte x, y a los valores de incertidumbre mínimo y máximo de la FOU o de los niveles de disparo de las reglas activas $\overline{f}_i$ y $\underline{f}_i$. M corresponde a los puntos de discretización de la FOU. Las salidas corresponden a y_l y y_r y a los puntos de cambio L y R.

3.1 Camino de datos

El camino de datos se divide en dos unidades, el camino de datos para calcular y_l y el camino de datos para calcular y_r. Se hace esta distinción, dado que las ecuaciones para calcular el centroide mínimo y máximo son similares. Entonces, es posible proponer un camino de datos único y duplicarlo para obtener y_l y y_r, modificando las entradas y el direccionamiento de la memoria para el cálculo de las sumatorias. En la Figura 6 se presenta el camino de datos propuesto. En el bloque uno se calcula la sumatoria de productos de la izquierda de a y la sumatoria de productos de la derecha de a', en el bloque dos se calcula la sumatoria productos de la derecha de a. El bloque tres calcula la sumatoria de la izquierda de b y la

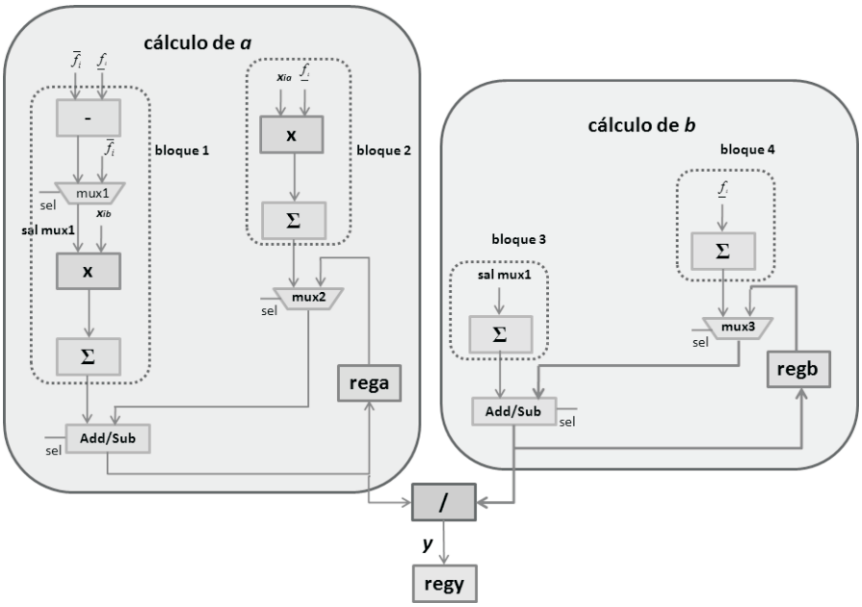


Figura 6. Camino de datos para el cálculo del centroide.

Tabla 1 . Organización de las sumatorias para el cálculo del centroide sobre el camino de datos.

Bloque	y_l	y_r
	$\sum_{i=1}^{k} \underline{x}_i \overline{f}_i$	$\sum_{i=1}^{k} \overline{x}_i \underline{f}_i$
1	$\sum_{i=\min(k,k')+1}^{\max(k,k')} \underline{x}_i \left(\overline{f}_i - \underline{f}_i\right)$	$\sum_{i=\min(k,k')+1}^{\max(k,k')} \overline{x}_i \left(\overline{f}_i - \underline{f}_i\right)$
2	$\sum_{i=k+1}^{M} \underline{x}_i \underline{f}_i$	$\sum_{i=k+1}^{M} \overline{x}_i \overline{f}_i$
	$\sum_{i=1}^{k} \overline{f}_i$	$\sum_{i=1}^{k} \underline{f}^i$
3	$\sum_{i=\min(k,k')+1}^{\max(k,k')} \left(\overline{f}_i - \underline{f}_i\right)$	$\sum_{i=\min(k,k')+1}^{\max(k,k')} \left(\overline{f}_i - \underline{f}_i\right)$
4	$\sum_{i=k+1}^{M} \underline{f}_i$	$\sum_{i=k+1}^{M} \overline{f}_i$

sumatoria de la derecha de b', finalmente el bloque cuatro calcula la sumatoria de la derecha de b. Los bloques Add/Sub calculan una suma o una resta, según el valor del signo s, si es necesario realizar el paso cuatro del algoritmo. Finalmente y y y' se calculan en el divisor, representado con (/) en la Figura 6. La resume la organización del cálculo de las sumatorias sobre el camino de datos de acuerdo con las Ec. (8) a (13).

3.2 Unidad de control

La unidad de control de la arquitectura propuesta, presenta una estructura de control jerárquica Micheli (1994). Este tipo de estructura realiza operaciones sobre un circuito digital, indicando cuando deben ser calculadas, desde un bloque de control raíz y relegando a bloques de control individuales la ejecución de dichas operaciones,

como lo ilustra la Figura 7. En la arquitectura propuesta para el cálculo del centroide, se tiene un bloque de control raíz y dos bloques de control secundarios que actúan sobre el camino de datos (Figura 5). El bloque raíz de control se encarga de realizar la escritura en memoria de los datos de entrada para el cálculo del centroide (y_l y y_r), habilitar el cálculo de y_l y y_r e indicar cuándo se ha calculado el centroide. Este bloque de control raíz podría además controlar el funcionamiento de un IT2-FLS, en el cual es posible utilizar la arquitectura propuesta para calcular la reducción de tipo.

Los bloques de control secundarios activan el cálculo de y_l y y_r, su funcionamiento es idéntico, la máquina de estados finita (FSM), que se ilustra en la Figura 8, consta de nueve estados en los que se calcula ya sea y_l o y_r. Se necesitan dos bloques de control secundario operando en paralelo dado que los puntos óptimos de inicialización del algoritmo EKM son diferentes para hallar y_l y y_r, Wu & Mendel (2007), lo cual cambia el direccionamiento de la memoria.

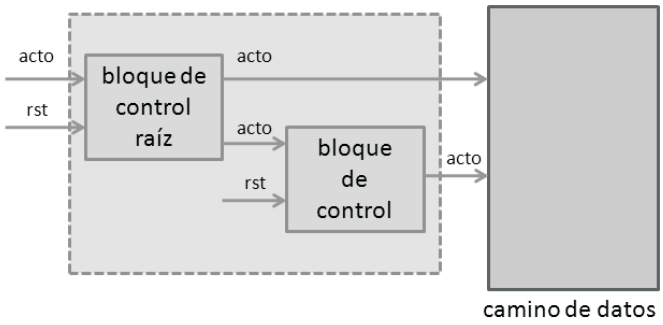


Figura 7. Estructura de control jerárquica.

3.3 Memoria

La memoria se utiliza para almacenar las entradas del algoritmo , x, $\overline{f}_i$ y $\underline{f}_i$ que pueden corresponder a los niveles de disparo de M reglas activas en un IT-FLS, o a la discretización de la FOU con M puntos. Se asume que en la memoria, se encuentran organizados todos los datos tal que se cumpla $\underline{x}_1 \leq \underline{x}_2 \leq \cdots \leq \underline{x}_N$ y $\overline{x}_1 \leq \overline{x}_2 \leq \cdots \leq \overline{x}_N$ haciendo que cada x_i corresponda a su respectivo peso f_i. La configuración de la memoria se ilustra en la Figura 9, la cual tiene buses de entrada y salida separados, un bus de direccionamiento de escritura y un bus de direccionamiento de lectura.

La escritura de la memoria se realiza externamente con el fin de brindar flexibilidad en el cálculo del centroide, al utilizar la arquitectura propuesta, en un IT2-FLS, permitiéndole a la unidad de control principal del IT2-FLS escribir la memoria mientras realiza el cálculo de la inferencia. Para calcular las Ecs.(8) a (13), se direcciona la memoria desde los bloques secundarios de control. Cada bloque secundario accede a la memoria independientemente según la dirección de memoria que el bloque necesite para el cálculo del centroide. En el direccionamiento de la memoria se agrupan las sumatorias como se presenta en la Tabla 1.

La arquitectura presentada en esta sección (Figura 5) es general, dado que permite resolver el problema del cálculo del centroide para un número M de reglas activas en la salida del motor de inferencia de un IT2-FLS, ó M puntos de discretización de la FOU que describe un IT2-FS. Permite además, calcular los dos centroides, y_l y y_r, en paralelo, dado que cada uno se calcula sobre un camino de datos como el presentado en la Figura 6.

4. Algoritmo CORDIC para el cálculo del producto y el cociente

El algoritmo CORDIC está fundamentado en la rotación de un vector sobre un plano, el vector (x_{in},y_{in}) es rotado un ángulo θ, produciendo el vector (x_R,y_R). La rotación se describe por medio de Ec. (14), Ercegovac & Lang (2004), Andraka (1998), Volder (1959).

$$x_R = M_{in}\cos(\beta+\theta) = x_{in}\cos\beta\cos\theta - y_{in}\sin\beta\sin\theta$$
$$y_R = M_{in}sen(\beta+\theta) = x_{in}\sin\beta\cos\theta + y_{in}\cos\beta\sin\theta \quad (14)$$

El algoritmo CORDIC realiza la rotación por medio de micro rotaciones de ángulos elementales α_j y de la descomposición del ángulo θ en la suma de ángulos elementales. Expresando el algoritmo

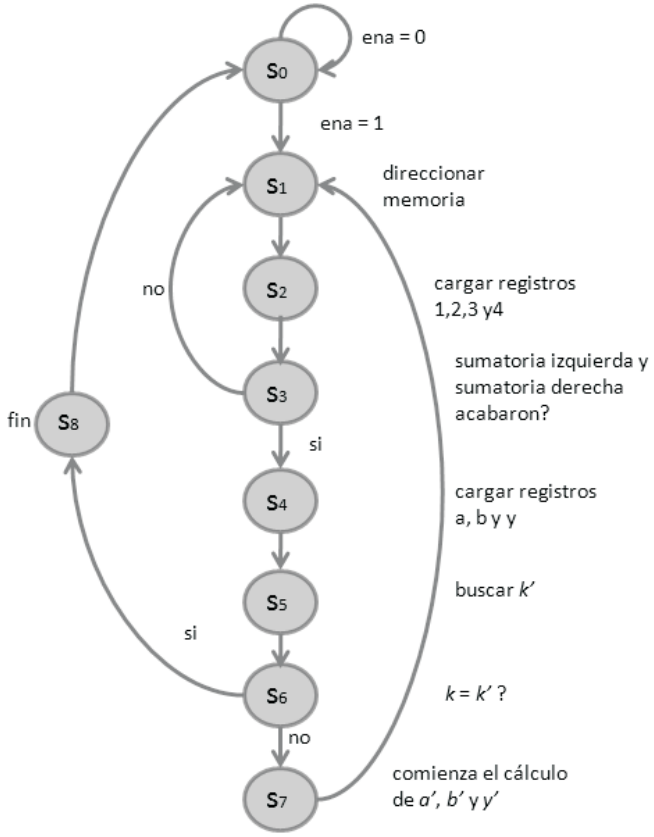


Figura 8. Diagrama de estados de la FSM del bloque secundario de control.

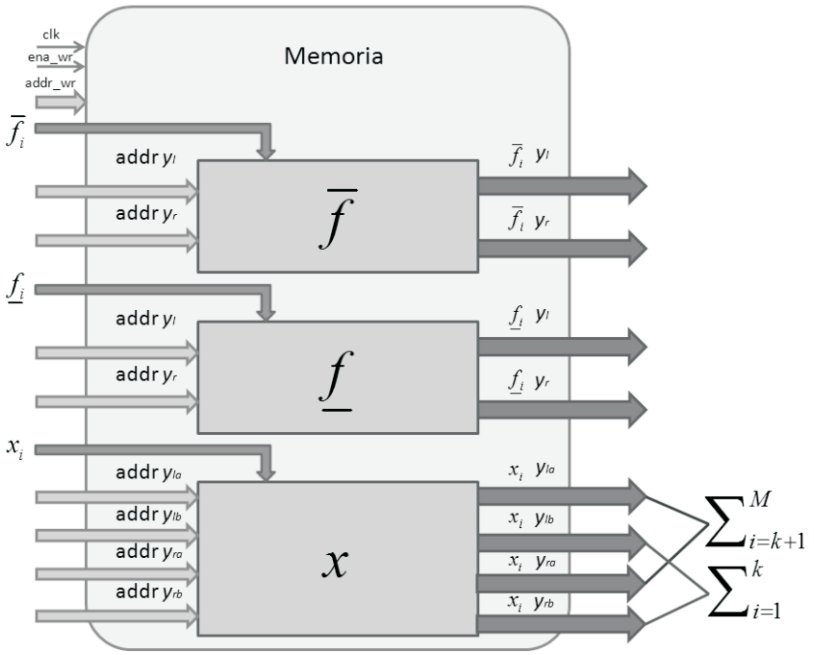


Figura 9. Configuración de la memoria.

de tal forma que no se realicen multiplicaciones, la iteración resultante, denominada micro-rotación CORDIC se presenta en Ec. (15), Ercegovac & Lang (2004).

$$\left\{\begin{array}{l} x[j]=x[j]-\sigma_j^{-2j}\,y[j] \\ y[j]=y[j]+\sigma_j^{-2j}\,x[j] \\ z[j]=z[j]-\sigma_j^{-2j}\tan^{-1}\left(2^{-j}\right) \end{array}\right\} \qquad (15)$$

Las operaciones aritméticas del camino de datos propuesto en la pueden realizarse utilizando operadores aritméticos basados en el algoritmo CORDIC, el cual es de naturaleza iterativa y su extensión a coordenadas lineales se realiza por medio de la rotación de un vector tal como lo muestra la Figura 10, Ercegovac & Lang (2004), que es:

$$\left\{\begin{array}{l} x_R = x_{in} \\ y_R = y_{in} + x_{in} z_{in} \end{array}\right\} \qquad (16)$$

donde x_{in} y y_{in} corresponden a las coordenadas *x*-*y* del punto inicial en coordenadas lineales y zi_n corresponde al ángulo con el cual se va a rotar el punto, x_R e y_R, son las coordenadas finales del punto rotado en un ángulo z_{in}. Una iteración CORDIC en coordenadas lineales se calcula utilizando la Ec. (17):

$$\left\{\begin{array}{l} x[j+1]=x[j] \\ y[j+1]=y[j]+\sigma_j 2^{-j} x[j] \\ z[j+1]=z[j]+\sigma_j 2^{-j} \end{array}\right\} \qquad (17)$$

donde *j* corresponde a la *j*-ésima iteración del algoritmo.

Existen dos modos en los que opera el algoritmo CORDIC, modo vector y modo rotación. Una vez el algoritmo converge, los valores finales para el modo rotación permiten calcular una suma-producto con la Ec. (18) y para modo vector una suma-cociente con la Ec. (19), Ercegovac & Lang (2004). Para la convergencia del algoritmo en coordenadas lineales es necesario cumplir la condición de Ec. (20) para el modo rotación, y la condición de Ec. (21) para el modo vector Hu et al. (1991).

$$\left\{\begin{array}{l} x_f = x_i \\ y_f = y_i + x_i z_i \\ z_f = 0 \end{array}\right\} \qquad (18)$$

$$\left\{\begin{array}{l} x_f = x_i \\ y_f = 0 \\ z_f = z_i + \dfrac{y_i}{x_i} \end{array}\right\} \qquad (19)$$

$$\left|z_i\right| \le 1 \qquad (20)$$

$$\left|y_i / x_i\right| \le 1 \qquad (21)$$

4.1 Cálculo del producto y cociente con el algoritmo CORDIC

Utilizando el modo rotación del algoritmo CORDIC es posible calcular el producto de dos números, *x* y *z*, si $y_i = 0$, cumpliendo la condición de Ec. (20) necesaria para la convergencia del algoritmo.

De la Ec. (19), se observa que utilizando el algoritmo CORDIC en modo vector es posible calcular el cociente entre dos números y_i y x_i, si z_i =0. Para que el algoritmo converja, es necesario cumplir la condición de Ec. (21).

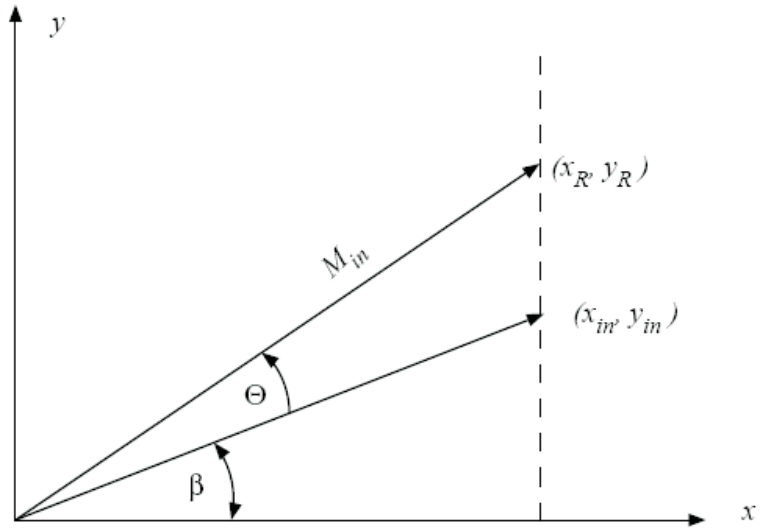


Figura 10. Rotación de un vector en coordenadas lineales Ercegovac & Lang (2004).

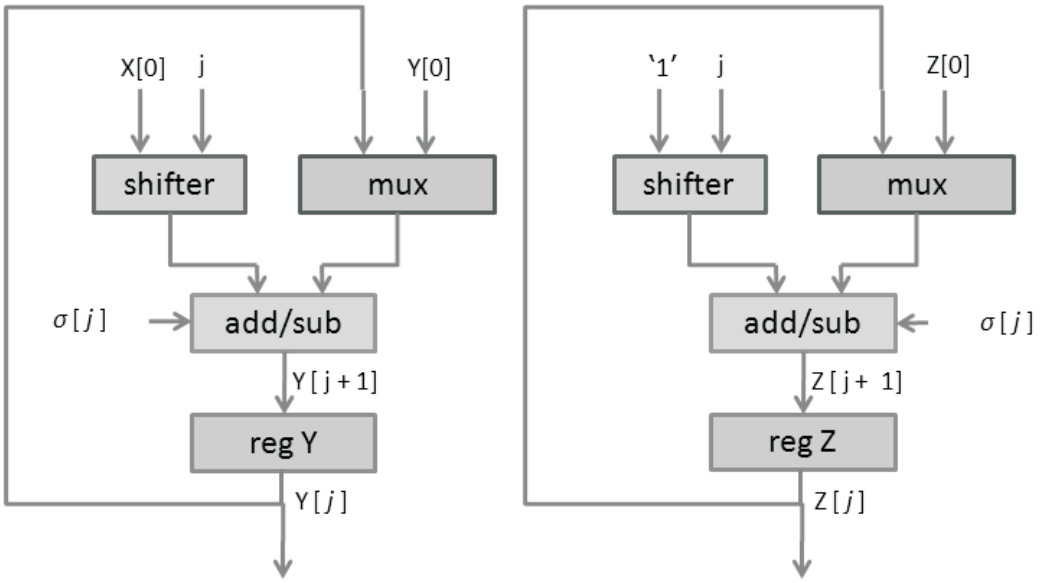


Figura 11. Implementación palabra-serial del algoritmo CORDIC en coordenadas lineales Ercegovac & Lang (2004).

4.2 Implementación *hardware* de la multiplicación y la división utilizando el algoritmo CORDIC

La implementación del algoritmo CORDIC se puede llevar a cabo mediante un esquema de palabra serial o un esquema de palabra paralela segmentado. En la Figura 11 se presenta la implementación palabra-serial para realizar la multiplicación o la división, por su naturaleza serial, permite reutilizar el *hardware* en el cálculo de cada iteración del algoritmo. Otras posibles implementaciones con orientación a tecnología FPGA pueden consultarse en Andraka (1998).

Para calcular la multiplicación o la división, es necesario utilizar las condiciones iníciales de Ec. (18) y Ec. (19), así como calcular σ_j de acuerdo con:

$$\sigma_j = \begin{cases} 1 & si \quad z[j] \geq 0 \\ -1 & si \quad z[j] < 0 \end{cases} \tag{22}$$

$$\sigma_j = \begin{cases} 1 & si \quad z[j] < 0 \\ -1 & si \quad z[j] \geq 0 \end{cases} \tag{23}$$

donde σ determina la operación que se va a realizar, para calcular la multiplicación σ se calcula con la Ec. (22) y para calcular la división con la Ec. (23). Suponga que se va a calcular la multiplicación, retomando la Ec. (18), correspondiente al modo rotación del algoritmo, se hace que y_i sea igual a cero, mientras que x_i y z_i, corresponderán a los operandos que se busca multiplicar. Para calcular una iteración, Ec. (17), se calcula σ_j, con la Ec. (22) y se realiza la multiplicación por 2^{-j} en los registros de corrimiento (*shifter*), donde j corresponde a la iteración que se está calculando, en los bloques ADD/SUB se realiza la suma o resta según sea el caso y los registros REGY y REGZ permiten almacenar el valor de la iteración calculada.

El esquema de implementación de la Figura * es de palabra-serial Ercegovac & Lang (2002), en la cual se procesan los datos de forma paralela, pero las iteraciones del algoritmo se calculan de manera serial. De esta forma para calcular ya sea un producto o un cociente, se necesitan $2n + 1$ ó $n + 1$ iteraciones (ciclos de reloj) respectivamente, de modo que el algoritmo CORDIC converja, donde n representa la resolución de los operandos.

El error en el cálculo del algoritmo CORDIC, Hu & Bass (1993), depende del número de iteraciones y de la resolución de los operandos. Este error se puede contrarrestar si se incrementa el tamaño *hardware* de la implementación del algoritmo Hu & Naganathan (1990) y Walter (1971):

$$l(n) = n + \log_2 n \tag{24}$$

donde n para la multiplicación corresponde al número de bits del producto y para la división n es el tamaño del cociente.

El número de iteraciones necesarias para convergencia, en el esquema de implementación palabra-serial del algoritmo CORDIC resulta similar al caso de implementación de la aritmética serial convencional Ercegovac & Lang (2004). La conveniencia de utilizar este tipo de implementación palabra-serial está sujeta a las restricciones de la aplicación. Además, de las Ec. (16) y (17) y la Figura * queda abierta la exploración de la implementación de la multiplicación y la división sobre el mismo hardware, teniendo en cuenta las condiciones de las Ec. (20) y (21), posibilidad que no ofrece la aritmética serial convencional.

5. Implementación *hardware* de la arquitectura propuesta para el cálculo del centroide

La implementación de la arquitectura propuesta está restringida a la capacidad en área de la tecnología sobre la cual se empotrará. Esta restricción se ve reflejada en el número de conjuntos o puntos de discretización de la FOU que pueda manejar la implementación y en la resolución de los datos de entrada y de salida. La implementación se realiza sobre la FPGA Virtex-E 600 de XILINX©, a 16 bits de resolución de los datos de entrada y salida, con M = 16 puntos de

discretización de la FOU, utilizando representación numérica no redundante en complemento a dos. Se realiza una descripción estructural en Lenguaje de Descripción de *Hardware* (VHSIC Hardware Description Language, VHDL). La simulación se lleva a cabo utilizando ModelSim® XE III/Starter 6.2g de Mentor Graphics Corporation©. Las etapas de síntesis, ubicación e interconexión del diseño, se llevan a cabo con la herramienta ISE™ Project Navigator 9.2i de Xilinx®.

5.1 Implementación *hardware* con operadores aritméticos paralelos

La primera propuesta de implementación consiste en tomar la arquitectura general de la Sección 4 e implementar el producto y el cociente utilizando operadores paralelos en aritmética convencional complemento a dos. El bloque de control raíz y los bloques de control secundarios son idénticos a los bloques propuestos en la Figura 8. El dimensionamiento del camino de datos se realiza teniendo en cuenta que el error en el cálculo del centroide se deba únicamente al error debido a la cuantificación de los datos y a su propagación, sin sobredimensionar su tamaño. Los sumadores de los bloques uno a cuatro del camino de datos de la 5 se implementan bajo el esquema MAC (*Multiplier Accumulator*), donde la i-ésima suma es acumulada en un registro hasta completar el cálculo de la sumatoria. Para calcular el cociente x/d se emplea el algoritmo NPD (Non Performing Divide), Ercegovac & Lang (2004), implementado en forma paralela. La memoria implementada es una SRAM utilizando la configuración presentada en la Figura 9.

5.2 Implementación *hardware* con operadores aritméticos basados en el algoritmo CORDIC

Al igual que la implementación paralela, se utiliza la arquitectura presentada en la Sección 3. La diferencia con la implementación paralela son los operadores para calcular el producto y el cociente, operaciones aritméticas involucradas en el cálculo del centroide, las cuales se implementan con referencia al cálculo de la multiplicación y la división presentadas en la Sección 4. Utilizar este tipo de operadores modifica la FSM de la unidad de control, dado que se utiliza la implementación palabra-serial del algoritmo CORDIC para cada operador aritmético, la cual debe asegurar el cálculo del producto y el cociente, teniendo en cuenta la naturaleza iterativa del algoritmo CORDIC..

6. Validación de la implementación

El objetivo de la validación sobre el dispositivo programado es verificar que la implementación de la arquitectura propuesta para el cálculo del centroide de un IT2-FS funciona en la realidad, así como conocer el comportamiento del error en el cálculo del centroide al muestrear la FOU. Para la validación se generan 12 FOUS con 1000 puntos de discretización, cuya naturaleza se presenta en la , se escogen este tipo de FOUs dado que tienen la naturaleza de las FOUs reportadas en Wu & Mendel (2007) para determinar puntos óptimos de inicialización del algoritmo EKM, representando un punto de referencia para explorar el cálculo de los centroides. Se generan FOUs cargadas hacia la derecha o hacia la izquierda con el fin de explorar la implementación en valores extremos de su resolución, y FOUs centradas sobre el universo discurso. Es decir que por cada tipo de FOUs se calculan tres centroides diferentes.

Luego de generar las FOUs se sigue el procedimiento a continuación:

• Muestrear la FOU con $M = 16$ y cuantificarla a resolución de 16 bits.

• Describir en VHDL una memoria ROM desde una aplicación en MATLAB® utilizando las FOUs cuantificadas y programar el dispositivo.

• Implementar además una función en MATLAB® para calcular el centroide de un IT2-FS utilizando el algoritmo EKM.

• Calcular el centroide c_i, correspondiente al centroide de la FOU completa (representada con

1000 puntos de discretización) y calcular el centroide de FOU muestreada (c_{cS}) utilizando una función implementada en MATLAB®.

• Verificar el cálculo de los centroides en *hardware* (c_H) se utiliza la herramienta ChipScope™ Analizer 9.2i de Xilinx® y calcular los centroides obtenidos en software con los centroides obtenidos en *hardware*.

Inicialmente se halla el error que se tiene al calcular el centroide de la FOU cuando esta se muestrea con $M = 16$:

$$e_{cS} = abs\left(\frac{c_i - c_m}{c_i}\right)\text{x}100 \qquad (25)$$

donde c_i es el centroide de la FOU completa y c_m el centroide de la FOU muestreada, ambos calculados con la implementación software. Luego se halla el error al calcular el centroide con la FOU completa y la FOU muestreada, este último calculado con la implementación software:

$$e_{cH} = abs\left(\frac{c_i - c_{mH}}{c_i}\right)\text{x}100 \qquad (26)$$

donde c_i es el centroide de la FOU completa calculado en software y c_m el centroide de la FOU muestreada calculado en *hardware*. Finalmente se obtiene la diferencia entre ec_S y ec_H:

$$d = abs\left(\frac{c_{cS} - c_{sH}}{c_{cS}}\right) \qquad (27)$$

El resultado al calcular el error de Ec. (25) y Ec. (26) y la diferencia entre ellos Ec. (27) se presenta en la Tabla 2, los cuales son aproximadamente iguales para las dos implementaciones realizadas, paralela y CORDIC. En particular, el error que se tiene al calcular el centroide en *hardware* a resolución de 16 bits, respecto al cálculo en una plataforma como MATLAB® con $M = 16$, al y $n = 16$ bits con FOUS de naturaleza gaussiana, está entre $2.35\text{x}10^{-2}$ % y $1.36\text{x}10^{-2}$ % aproximadamente. Para las FOUs de naturaleza triangular, el error obtenido está entre 5.04x^{-3}% y 1.75x^{-3}%. De esta forma se valida la implementación de la arquitectura propuesta en la Sección 3 utilizando operadores aritméticos basados en el algoritmo CORDIC.

7. Desempeño de la implementación

Los índices de desempeño de la implementación *hardware* se resumen en la para las dos implementaciones realizadas, la máxima frecuencia de operación se determina por medio de simulación, utilizando el modelo generado por la herramienta ISE™ Project Navigator de Xilinx® en la etapa de ubicación e interconexión de la implementación. Con la frecuencia máxima de operación se estima la potencia máxima consumida por la implementación utilizando Xpower™ de Xilinx®.

El desempeño de la implementación CORDIC es mejor que el de implementación paralela para resolución de 16 bits de los datos de entrada y de salida, la implementación CORDIC es 2.9 veces más pequeña que la implementación paralela y su máxima frecuencia de operación es 20.6 veces mayor que para la implementación paralela. El consumo de potencia total (dinámica y estática) estimada es 1.1 veces menor para la implementación paralela que para la implementación CORDIC, teniendo en cuenta que la frecuencia máxima de operación de la propuesta CORDIC a la que es hallada la potencia consumida es 18.18 veces mayor que la frecuencia máxima de la implementación paralela.

8. Conclusiones

Se presentó una propuesta de arquitectura para la implementación del cálculo del centroide de un IT2-FS utilizando el algoritmo Karnik-Mendel mejorado. Se exploró además el uso del algoritmo CORDIC para el cálculo del producto y el cociente del modelo computacional del algoritmo EKM. Se realizó la primera implementación *hardware* sobre tecnología FPGA para el cálculo del centroide de un IT2-FS con el algoritmo EKM, así como la validación de la arquitectura propuesta utilizando la propuesta aritmética CORDIC, para una resolución de 16 bits y para máximo 16 puntos

de discretización de la FOU. Se presentó una comparación de implementación utilizando operadores aritméticos paralelos, ya que no se disponía de un referente de implementación *hardware* del algoritmo EKM reportado en la literatura y se obtuvo que la implementación CORDIC, para la resolución explorada, tiene mejor desempeño en área y frecuencia que la implementación paralela.

Tabla 2. Error en el cálculo del centroide de diferentes FOUs, M =16 y n = 16.

y_l			y_r		
ec_S (%)	ec_H (%)	d (%)	ec_S (%)	ec_H (%)	d (%)
23.129	23.112	2.25E-2	14.116	14.105	1.36E-2
0.360	0.357	3.21E-3	0.260	0.262	2.93E-3
2.816	2.818	1.30E-3	1.977	1.979	1.80E-3
11.069	11.059	1.04E-2	5.535	5.532	3.67E-3
0.256	0.251	4.89E-3	0.159	0.161	1.88E-3
1.806	1.808	2.86E-3	2.223	2.225	1.76E-3
10.219	10.217	2.14E-3	0.194	0.199	5.04E-3
0.051	0.049	1.75E-3	0.026	0.030	3.85E-3
0.090	0.088	2.17E-3	1.724	1.727	3.19E-3
7.012	7.009	2.85E-3	0.004	0.000	4.07E-3
0.138	0.135	2.32E-3	0.100	0.103	3.61E-3
0.456	0.454	2.29E-3	1.808	1.812	3.16E-3

Tabla 3. Índices de desempeño de la implementación sobre la FPGA VIRTEX-E 600.

Implementación	**Paralela**	**CORDIC**
Índice	Valor	Valor
Máxima frecuencia de operación (PAR)	2MHz	41.205MHz
Máxima frecuencia de operación (Simulación PAR)	2.2MHz	40MHz
Slices	5891 (85%)	2031 (29%)
Máxima potencia consumida	295mW	324mW

9. Referencias bibliográficas

Andraka, R. (1998). *A survey of CORDIC algorithms for FPGA based computers*. Proceedings of the 1998 ACM/SIGDA – Sixth International Symposium on Field Programmable Gate Arrays, 191-200.

Ercegovac, M. D. & Lang, T. (2004). *Digital Arithmetic*. Morgan Kaufmann Publishers.

Fisher, P. (2007). What is where? type-2 fuzzy sets for geographical information [research frontier]. *IEEE Computational Intelligence Magazine* 2(1), 9–14.

Hagras, H. (2004). *A Type-2 Fuzzy Logic Controller for Autonomous Mobile Robots*. In The 2004 IEEE International Conference on Fuzzy Systems, Budapest, Hungary, (2), 965 – 970.

Hagras, H. (2007). Type-2 FLCs: A New Generation of Fuzzy Controllers. *IEEE Computational Intelligence* 2(1), 30–43.

Hernandez & Mendez (2006). *Modeling and Prediction of the MXNUSD Exchange Rate Using Interval Singleton Type-2 Fuzzy Logic Systems*. In *Proc. IEEE International Conference on Fuzzy Systems, pages 2305–2308.*

Hidalgo, D., Castillo, O., & Melin, P. (2008). *Optimization with Genetic Algorithms of Modular Neural Networks using Interval Type-2 Fuzzy Logic for Response Integration: The Case of Multimodal Biometry.* In Proc. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on Neural Networks IJCNN 2008,738–745.

Hu, X., Harber, R. G.,& Bass, S. C. (1991). Expanding the Range of Convergence of the CORDIC Algorithm. *IEEE Transaction on Computers* 40(1), 13–21.

Karnik, N. & Mendel, J. M. (2001). Centroid of a Type-2 Fuzzy Set. *Information Scienciens* 132(1-4), 195–220.

Hu, X. & Bass, S.C. (1993). A Neglected Error Source in the CORDIC Algorithm. *IEEE Transactions on Acoustics, Speech and Signal Processing* 38(2), 352–354.

Hu, Y & Naganathan, A. (1990). Novel Implementation of a Chirp <e1>Z</e1>-Transform Using a CORDIC Processor. *IEEE Transactions on Acoustics, Speech and Signal Processing* 38(2), 352–354.

Liang, Q. & Mendel, J. (2000). Overcoming time-varying co-channel interference using type-2 fuzzy adaptive filters. *IEEE Transactions on Fuzzy Systems* 47(12), 1419–1428.

Lynch, C., Hagras, H., & Callaghan, V. (2007). *Parallel Type-2 Fuzzy Logic Co Processors for Engine Management.* University of Florida Research Exchange on intelligent environments.

Melgarejo, M. (2007). *A Fast Recursive Method to Compute the Generalized Centroid of an Interval Type-2 Fuzzy Set.* In Proc. of NAFIPS'07 Annual Meeting of the North American Fuzzy information processing society 2007, San Diego.

Melgarejo, M. & Peña, C. (2007). Implementing Interval Type 2 Fuzzy Processors. *IEEE Computational Intelligence Magazine* 2(1), 63–71.

Mendel, J. M. (2001). *Rule-Based Fuzzy Logic Systems: Introductions and New Directions Upper Saddel River*. Prentice-Hall, NJ.

Mendel, J. M. (2007). Type-2 Fuzzy Sets and Systems: An Overview. *IEEE Computational Intelligence* 2(1), 20–29.

Mendel, J. M., John, R., & Liu, F. (2006). Interval Type-2 Fuzzy Logic Systems Made Simple. *IEEE Transactions on Fuzzy Systems* 14(6), 808–821.

Mendel, J. M. & Liu, F. (2007). Super-Exponential Convergence of the Karnik-Mendel Algorithms for Computing the Centroid of an Interval Type-2 Fuzzy Set. *IEEE Transactions on Fuzzy Systems* 15(2), 309–320.

Micheli, G. (1994). *Synthesis and optimization of digital circuits*. McGraw-Hill Series in Electrical and Computer Engineering, Electronics and VLSI Circuits.

Rhee, F. C.-H. (2007). Uncertain Fuzzy Clustering, Insights and Recommendations. *IEEE Computational Intelligence* 2(1), 44–56.

Volder J.E. (1959). The CORDIC Trigonometric Computing Technique. *IRE Transactions on Electronic Computers* (8), 330-334.

Walter, J.S. (1971). A Unified Algorithm for Elementary Functions. *Spring Joint Computer Conference* (38), 379–385.

Wu, D. & Mendel, J. M. (2007). *Enhanced Karnik-Mendel Algorithms.* In Proc. of NAFIPS'07 Annual Meeting of the North American Fuzzy information processing society 2007, San Diego.

Wu, H. & Mendel, J. (2002). Uncertainty Bounds and their Use in the Design of Interval Type-2 Fuzzy Logic Systems. *IEEE Transactions on Fuzzy Systems* 10(5), 622–639.