INSTITUTO POLITÉCNICO NACIONAL.

CENTRO DE INVESTIGACION EN COMPUTACION

"ARQUITECTURA DE UN CONTROLADOR LÓGICO DIFUSO USANDO LÓGICA COMBINATORIA"

T E S I S

QUE PARA OBTENER EL GRADO DE :

MAESTRO EN CIENCIAS EN INGENIERÍA DE CÓMPUTO
CON OPCIÓN EN SISTEMAS DIGITALES

PRESENTA:

ING. ARTURO TÉLLEZ VELÁZQUEZ

DIRECTORES DE TESIS:
M. EN C. ROMEO URBIETA PARRAZALES
DR. OSCAR CAMACHO NIETO

México, D. F. Junio de 2008

Índice de contenido

Resumen

En esta tesis se presenta el desarrollo de la arquitectura de un *Controlador Lógico Difuso*, denotado a lo largo de este trabajo como FLC, por sus siglas en inglés, para su implementación en cualquier aplicación, haciendo uso sólo de lógica combinatoria y de un dispositivo llamado *Arreglo de Compuertas Programables en Campo*, o FPGA. Esta arquitectura está basada en módulos básicos de construcción programados en un lenguaje de descripción de hardware llamado VHDL, cuyo diseño completamente combinatorio y modular permite escalar al sistema, mediante simple replicación, para las necesidades particulares de alguna aplicación.

Han aparecido muchas implementaciones de FLC desde de su primera implementación en hardware, todas con lógica secuencial, debido a que los circuitos combinatorios solían ser muy costosos en cuanto recursos en hardware y a velocidad de operación. Todas aquéllas implementaciones con circuitería secuencial aún cuando son eficientes, resultan ser demasiado complejas para el diseño y eficientes. Pero con la aparición del FPGA y otras tecnologías de años recientes, es posible el uso de circuitería combinatoria rápida para el diseño de sistemas digitales complejos.

El propósito de esta arquitectura es usar el paralelismo entre sus módulos para incrementar el rendimiento en velocidad del FLC y de ésta manera, aplicarlo a cualquier proceso, aprovechando las ventajas de las nuevas tecnologías en FPGA y facilitando al diseñador el escalamiento de la misma, para incrementar también la exactitud del control. Este trabajo comprende sólo el diseño del FLC a nivel hardware, pero también se hacen las consideraciones de diseño de todo el sistema de control, y de esta manera, se pueda realizar un diseño completo y acertado.

Cabe destacar que, esta arquitectura combinatoria es innovadora, por el hecho de tener facilidades de *adaptación* al tener funciones de membresía dinámicas, es decir, que sus valores de configuración pueden cambiarse *en línea*. Asimismo, como otra aportación de este trabajo, se presenta una metodología de diseño sencilla y detallada para el diseño de FLCs, mediante la elaboración de un FLC de ejemplo, aplicado al control de un servomotor de corriente directa.

Abstract

This work presents the architecture development of a *Fuzzy Logic Controller*, FLC, for its implementation on any application, using combinatorial logic circuits implemented on a *Field Programmable Gate Array*, FPGA. This architecture is based on VHDL programmed basic modules, so combinatorial, that enable to increase and improve the entire system performance, by means of replication, to fulfill the particular application needs.

There have been so many FLC implementations since the first hardware one appeared, but everyone using complex designs with sequential logic circuits. This is because of the high hardware resource and delay time costs about combinatorial logic circuits. But recent FPGA and other technologies goods let us the use of fast combinatorial circuits for system complex designs too.

The purpose of this work is using the parallelism to increase the velocity performance on the FLC and apply it to any process, approaching the new FPGA technologies advantages and easing the scaling to increase the control accuracy. This work only comprises the design of the FLC in hardware terms, but also is made the design considerations of the entire control system to get a complete and correct design.

It is worth highlighting this combinatorial architecture innovates by the fact of having *adaptation* facilities having dynamic membership functions; it means that its configuration may be changed *on line*. Also, the strong way of this work is an easy and detailed FLC design methodology, while elaborating an FLC like an example applied to a DC servo control.

Planteamiento del problema

Recientemente ha habido un gran incremento en el uso de Controladores Lógico Difusos–*Fuzzy Logic Controller* (FLC) en muchos más ámbitos de la ciencia y la industria, así como implementaciones en diferentes tecnologías para lograr éste propósito, desde el uso de microcontroladores difusos hasta la implementación de arquitecturas completas de FLCs en tecnologías reprogramables, como es el caso de los Arreglos de Compuertas Programables en Campo–*Field Programmable Gate Array* (FPGA).

Un algoritmo de un FLC puede ser implementado fácilmente en software y ejecutado en un microprocesador, en un microcontrolador, o bien en una computadora de propósito general. Aunque los FLCs basados en software son mucho más económicos y flexibles suelen presentar dificultades en sistemas de control que requieren procesamiento de altas cantidades de datos en corto tiempo [5].

El uso de los FPGA ha sido rentable desde el punto de vista de la versatilidad para realizar cualquier tipo de diseño digital en términos de costos y tiempos de diseño. Con la aplicación de la lógica difusa en la industria en los sistemas de alta velocidad, en especial en los sistemas de tiempo real, como es el procesamiento de imágenes, la robótica, entre otras, son necesarias implementaciones computacionales más rápidas. Con el desarrollo de los Circuitos Integrados de Aplicación Específica–*Application Specific Integrated Circuit* (ASIC), es posible alcanzar altas velocidades y cualquier diseño desde el más simple hasta el más complejo puede estar contenido en un sólo Circuito Integrado–*Integrated Circuit* (IC). De esta manera, aquéllos sistemas que anteriormente no podían realizarse o que tenían un alto costo para su realización, como la manipulación de los *sistemas no lineales*, pueden ser perfectamente modelados e implementados mostrando altas velocidades de procesamiento [17], [19]. Los FPGAs surgieron a partir de los ASICs como una opción aún más económica y flexible, presentando velocidades de procesamiento competitivas y abriendo una brecha amplia para que los investigadores pudieran elaborar sus propios diseños a la medida en poco tiempo. De esta manera, comenzaron a aparecer FLCs dedicados que realizan operaciones lógicas difusas como parte de otro procesador o bien dedicados completamente a realizar toda la operación del control en estos sistemas reconfigurables.

Un FLC no está basado en un modelo matemático de todo el proceso que se pretende controlar, por lo que es muy efectivo para controlar procesos en los que la función de transferencia de la planta está indefinida, pero la acción de control está basada en la influencia del exterior y en simples decisiones basadas en el conocimiento que se adquiere en base a la experiencia, es decir, la *Base de Conocimiento*, de la misma manera que un ser humano lo haría, esto es, explotando su *Capacidad Heurística*. Los FLCs

permiten integrar cierto razonamiento humano en algoritmos de control en sistemas computacionales para diferentes áreas de la ciencia y de la industria. [7]

El diseño basado en los FPGA permite a los diseñadores crear, probar y modificar de manera rápida y sencilla cualquier cantidad de sistemas digitales e implementar sistemas dedicados que pueden ayudar a agilizar la computación a otros sistemas que no son de tiempo real, como las computadoras personales. Desde esta perspectiva, se han desarrollado gran cantidad de arquitecturas para la implementación de un FLC.

Entre las arquitecturas basadas en FPGA existen las que llevan a cabo su ejecución en forma secuencial, es decir que los datos se propagan por el FLC uno a uno y no puede procesarse otro dato hasta que se haya terminado de procesar el anterior. Esto conlleva a la pérdida de datos y el uso de muchos ciclos de reloj para obtener el resultado final, debido a que el procesamiento puede extenderse demasiado, que mientras se procesa un dato se pueden perder otros. Así también existen arquitecturas que vienen a reducir ampliamente el problema de pérdida de datos y los ciclos de procesamiento, utilizado varias etapas de Segmentación–*Pipelining*, incrementado indudablemente la complejidad de la arquitectura pero mejorando en gran parte el uso de recursos y el tiempo de procesamiento. Operaciones como la *Multiplicación* y la *División* suelen ser las más costosas en términos de tiempo de procesamiento y en consumo de recursos en un sistema, por lo que surgieron arquitecturas que se auxilian de elementos externos como memorias, microcontroladores y computadoras que realizan la tarea de supervisar y establecer la configuración del FLC y realizar de manera anticipada el cálculo de los parámetros del mismo; así que todo el procesamiento se basa en lecturas secuenciales o simultáneas a una o a varias memorias, con lo que el tiempo de procesamiento se reduce críticamente. Sin embargo, en todos estos casos, implica un costo muy alto de recursos externos o internos al FPGA y la complejidad se hace presente a la hora de escalar la arquitectura.

Es difícil mover la balanza para equilibrar el consumo de recursos con el tiempo de ejecución, por lo que es necesario realizar cambios en la forma de diseñar los algoritmos que describen a una arquitectura de un FLC. Entonces, la factibilidad para implementar un FLC en un FPGA depende de la elección de un algoritmo óptimo y sencillo que consuma un tiempo de procesamiento menor, consuma la menor cantidad de recursos posible del dispositivo a utilizar y su escalamiento no incremente la complejidad ni decremente su rendimiento. Ese algoritmo o procedimiento es la base de este trabajo y su pretensión es la de lograr que cualquiera pueda diseñar un FLC de manera sencilla, aprovechando la versatilidad de los FPGAs, con circuitos lógicos digitales sencillos.

Objetivos

El desarrollo de esta tesis se sustenta en el cumplimiento de los siguientes objetivos:

- Diseñar y probar una arquitectura de un *Controlador Lógico Difuso* (FLC) totalmente combinatorio, que proporcione facilidades de adaptación, liberándose así de señales de reloj, cuyas etapas esenciales se ejecuten en forma paralela, permitiendo de manera sencilla, escalar la arquitectura para mejorar el rendimiento, sin perder velocidad de procesamiento ni generalidad.
- Demostrar que el uso de la lógica combinatoria para el diseño de FLCs representa una opción práctica para la implementación de sistemas difusos aplicables a sistemas de tiempo real y sistemas no lineales.
- Reducir tiempo de diseño utilizando una metodología simple para la construcción de FLCs en un FPGA, mediante módulos básicos de construcción.

Herramientas

Para lograr los objetivos antes propuestos es necesario considerar algunos trabajos que se han realizado, recopilándolos y extrayendo las características más relevantes de ellos. Se estudiaron algunos documentos de la biblioteca digital de IEEE, los cuales están enlistados en la sección de *Referencias* [1–27], al final de este trabajo y los cuales son analizados en el Capítulo 2 llamado *Antecedentes* donde se muestra el rendimiento de algunos diseños. Como se ha dicho anteriormente, esta implementación puede escalarse para aplicarse a cualquier sistema. Independientemente de la aplicación, el FLC consta de tres etapas básicas bien conocidas, según Passino [28], como son la Difusificación–*Fuzzification*, la Inferencia Difusa–*Fuzzy Inference* y la Desdifusificación–*Defuzzification*.

Para la implementación se hace uso de Lenguajes de Descripción de Hardware (HDL), en este caso de *Very–High–Speed–Integrated–Circuit Hardware Description Language* (VHDL), las herramientas de Síntesis de *Xilinx ISE 6.3i*, las herramientas de Simulación de Mentor Graphics *Modelsim Xilinx Edition III 6.0a* y la caja de Herramientas *Fuzzy Toolbox–Fuzzy Inference System* (FIS) de MATLAB 2006 para la simulación y comparación del controlador implementado con el controlador simulado por software. Se utiliza el Spartan 3 FPGA Starter Kit de Xilinx, cuyo FPGA es el XC3S200–5FT256 para la implementación del FLC.

El diseño del FLC consta de un conjunto de etapas básicas, que se agrupan en la metodología de diseño propuesta en el *Capítulo 3*, donde se pretende explicar detalladamente el diseño de cada una de las etapas del FLC. La etapa de Difusificación consta de módulos difusificadores llamados *Funciones de Membresía* con los que cada valor real de la variable de entrada al sistema es convertido a un valor de membresía o pertenencia que puede manipular el controlador para realizar la toma de decisiones pertinente. La etapa de Inferencia Difusa consta de interconexiones en estructura de tipo árbol invertido, mediante módulos pequeños llamados MAX y MIN, de acuerdo a la base de conocimiento descrita por el diseñador del FLC. Mientras que la etapa de Desdifusificación transforma el valor difuso obtenido por las etapas anteriores a un valor real de acuerdo a la base de conocimiento. Se implementan estas etapas en este orden en el FPGA, se simulan con el Modelsim una por una, como si cada una fuera a implementarse por separado, se simula el FLC con Fuzzy Logic Toolbox de MATLAB y finalmente se implementan todas juntas en el FPGA para verificar su correcto funcionamiento por completo.

Es necesario aclarar que el FLC es totalmente combinatorio y la velocidad de operación depende de las velocidades de operación de sus elementos internos. La división es el elemento más lento del sistema y se ha implementado en dos algoritmos en

diferentes etapas, esto debido a la diferente respuesta que ofrece de acuerdo al número de bits, los cuales son la División Con Restauración *Restoring Division Algorithm* y la División Sin Restauración *Non–Restoring Division Algorithm*. Cabe resaltar que estos algoritmos pueden sustituirse por otros más rápidos y de esta manera la velocidad del FLC incrementará indudablemente.

Organización de la tesis

Basado en la sección de Recursos, el Capítulo 1, *Introducción*, constituye el Marco Teórico que debe conocer el lector antes de crear un FLC, desde la historia de la lógica difusa, la teoría del control difuso, las tecnologías reprogramables y el FPGA.

El Capítulo 2, *Antecedentes*, constituye el Estado del Arte o el Estado de la Cuestión, donde se presentan todos los FLCs que existen hasta la fecha, sin considerar a las nuevas ramas de la inteligencia artificial como el control neuro–difuso. Finalmente, después de realizar un estudio de los FLCs más relevantes se establecen los puntos que justifican la elaboración de este trabajo.

El Capítulo 3, *Diseño*, describe toda la arquitectura del FLC propuesta y se presenta una metodología de diseño, con un caso de estudio de ejemplo, con el que se pretende explicar a detalle dicha metodología y validar el funcionamiento del FLC implementado en hardware con los resultados del siguiente capítulo, *Resultados*.

El Capítulo 4, Resultados, presenta los resultados de tiempo y recursos utilizados cuando el diseño se implementa en FPGA. También se realiza una simulación en el Fuzzy Toolbox de MATLAB y una simulación del diseño de hardware mediante Modelsim con el fin de comparar los resultados y validar el funcionamiento del mismo. Finalmente el diseño es implementado en el FPGA y se comparan los resultados obtenidos con los previstos.

El Capítulo 5, *Conclusiones*, resume los resultados del capítulo anterior.

Existen 3 anexos importantes. El *Anexo 1*, muestra el diseño de los algoritmos de la división y de la multiplicación, utilizados en los módulos que constituyen a la arquitectura del FLC. El *Anexo 2*, muestra al lector la manera cómo un FPGA implementa un diseño HDL, basándose en las partes que lo constituyen. Y el *Anexo 3*, contiene algunas consideraciones de diseño importantes, tanto para el FLC y la división, como para el diseño en VHDL.

Introducción 1

La lógica difusa es el tópico central de esta tesis, por lo que se describe en esta sección con varios ejemplos sencillos y se hace una especie de recuento del desarrollo de algunas aplicaciones que han aparecido en los últimos años en el mundo y que han convertido al control difuso en una solución popular, por ser bastante viable y económica. Además se muestra al lector el lugar en el que está ubicado el control difuso dentro de la inteligencia artificial, la teoría básica del control difuso y la descripción de la tecnología ASIC para justificar el uso del FPGA como un FLC.

La lógica difusa y su historia

La lógica binaria de las computadoras modernas frecuentemente falla cuando trata de describir la vaguedad del mundo real. La lógica difusa ofrece alternativas más satisfactorias. Las computadoras no razonan como lo hace el cerebro. Las computadoras "razonan" cuando manipulan hechos precisos, que han sido reducidos a cadenas de ceros y unos e instrucciones que pueden ser falsas o verdaderas. El cerebro humano puede razonar a partir de aserciones vagas o afirmaciones que involucran incertidumbres o juicios de valor como son: "El aire está frío", "La velocidad es rápida" o "Ella es joven." A diferencia de las computadoras, lo humanos tienen *Sentido Común*, que les permite razonar en un mundo en donde las cosas son sólo parcialidades de la verdad. La *Lógica Difusa* es una rama de la Inteligencia Artificial que ayuda a las computadoras a representar toda la gama de imágenes del sentido común en un mundo lleno de incertidumbres. Especialistas en Lógica abordaron en 1920 su primer concepto clave: "Todo es una expresión del grado". La lógica difusa manipula conceptos vagos, tales como "tibio" o "seco" y de esta manera ayuda a los ingenieros a construir acondicionadores de aire, lavadoras y otros dispositivos que juzgan qué tan rápido deberían operar o cambiar de una configuración a otra, siempre que el criterio para realizar tales cambios esté bien definido. Cuando los matemáticos carecieron de algoritmos específicos para establecer cómo un sistema debería comportarse a determinadas entradas, la lógica difusa hizo su aparición controlando y describiendo el sistema usando reglas de sentido común para referirse a cantidades indefinidas. Los sistemas difusos frecuentemente deducen sus

reglas a partir de un *Experto*. Cuando no existe un experto que proporcione las reglas, existen sistemas difusos adaptivos que aprenden las reglas, observando cómo una persona regula los sistemas reales.

Figura 1. Vista frontal de un tren del Metro de Sendai en Japón.

Una reciente oleada de productos difusos comerciales, la mayoría de ellos provenientes de Japón, han popularizado a la lógica difusa. En 1980 la firma F. L. Smith and Company, en Copenhagen, usó por primera vez un sistema difuso para supervisar la operación de un cemento llamado Kiln. En 1988 Hitachi revolucionó la era del control utilizando un sistema difuso para controlar un tren en Sendai, Japón. Desde entonces, las compañías japonesas han usado lógica difusa para construir todo tipo de productos electrodomésticos y productos electrónicos, en general.

Las aplicaciones para lógica difusa se extienden más allá de los sistemas de control. Teoremas recientes muestran que en principio la lógica difusa puede ser usada para modelar cualquier sistema continuo, es decir, estar basado en la ingeniería, física, biología o economía. Investigadores de muchos campos de la ciencia pueden encontrar que los modelos de sentido común, son mucho más útiles o certeros de lo que son los modelos matemáticos convencionales.

El punto central de la diferencia entre la lógica clásica y la lógica difusa es algo que *Aristóteles* nombraba el *Argumento del Tercer Hombre*. En la teoría de conjuntos clásica, un objeto pertenece o no pertenece a un conjunto. No hay intermedios: el número cinco pertenece completamente al conjunto de los números impares y no al conjunto de los pares. En tales conjuntos bivalentes, un objeto no puede pertenecer a ambos, sino que a uno, a otro ó a ninguno de ellos. Este principio preserva la estructura de la lógica y evita la contradicción de que un objeto es y no es una cosa al mismo tiempo.

Los conjuntos que son difusos, o multivalentes, rompen el Argumento del Tercer Hombre en alguna forma. Un objeto pertenece solamente en forma parcial a un conjunto

difuso, aunque puede pertenecer a más de un conjunto. Aún para un individuo, el aire en el ambiente puede sentirlo fresco y justo a un lado sentirlo caliente unos pocos grados. Aquéllos límites donde los conjuntos clásicos son exactos, los límites de los conjuntos difusos son curvos o suaves, por lo que esta curvatura crea contradicciones parciales.

Los grados difusos o *Grados de Membresía* son porcentajes de pertenencia a un conjunto difuso, por lo que no son lo mismo que los porcentajes de probabilidad, aunque en alguna forma son muy parecidos. La probabilidad expresa si algo ocurrirá o no ocurrirá. La Vaguedad–*Fuzziness* expresa el grado al cual algo ocurre o alguna condición existe.

La única restricción en la lógica difusa es que el grado de membresía de un objeto en grupos complementarios debe sumar la unidad. De esta manera, la lógica difusa elude la contradicción bivalente de que algo es cien por ciento alguna condición o es cien por ciento lo contrario porque destruiría la lógica formal. De hecho, el Argumento del Tercer Hombre se mantiene como un caso especial en la lógica difusa en el que un objeto pertenece al cien por ciento a un grupo.

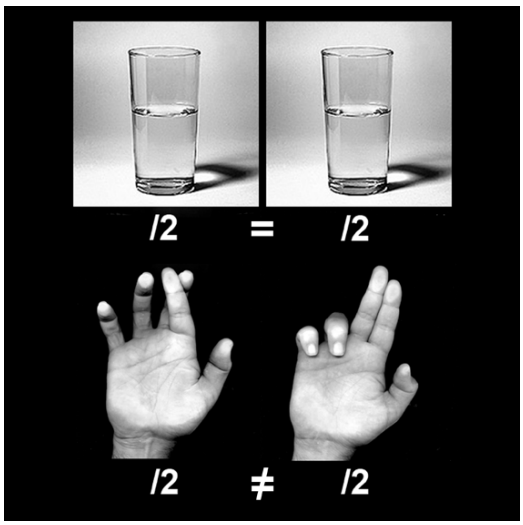

Figura 2. Puede que un vaso medio lleno sea lo mismo que un vaso medio vacío. Pero no es lo mismo media mentira que una verdad a medias [43].

El estudio moderno de la lógica difusa y de las contradicciones parciales tuvo sus orígenes a principios de este siglo, cuando Bertrand Russell encontró la antigua paradoja griega en el centro de la teoría y de la lógica de conjuntos moderna. De acuerdo al viejo enigma, un habitante de Creta aseguraba que todos los cretanos mienten. ¿Pero, siendo él un cretano, cómo saber si mintió? Si miente, entonces dice la verdad y por lo tanto no miente. Si no miente, entonces dice la verdad y por lo tanto miente. Ambos casos nos llevan a una contradicción debido a que la expresión es, en ambos casos, verdadera y falsa. Russell encontró la misma paradoja en la teoría de conjuntos. El conjunto de todos

los conjuntos es un conjunto y por lo tanto es un miembro de sí mismo. Pero el conjunto de todas las manzanas no es un miembro de sí misma porque sus miembros son manzanas y no conjuntos. Tomando en cuenta la contradicción antes explicada, Russell entonces se preguntó, "¿Es el conjunto de todos los conjuntos que no son miembros de ellos mismos un miembro de sí mismo?" Si lo es, no lo es; si no lo es, lo es.

Pero la lógica difusa dice que la respuesta es mitad verdadera y mitad falsa. Es decir, los cretanos mienten el cincuenta por ciento de las veces y la otra mitad no lo hacen. Cuando la membresía es menor que el total, un sistema bivalente puede simplificar el problema redondeándolo a cero o a cien por ciento. Pero el cincuenta por ciento no se redondea hacia abajo.

En 1920, de manera independiente a Russell, un lógico polaco llamado Jan Lukasiewicz trabajó sobre los principios de la lógica multivaluada, en la cual las expresiones pueden tomar valores de verdad fraccionales entre el cero y el uno de la lógica binaria. En un artículo de la Filosofía de la Ciencia en 1937, el filósofo Max Black aplicó la lógica multivaluada a listas o a conjuntos de objetos y realizó las primeras representaciones gráficas de las curvas de conjuntos difusos. Siguiendo la directriz de Russell, Black llamó a estos conjuntos "vagos".

Casi treinta años después el científico Lotfi A. Zadeh, entonces presidente del Departamento de Ingeniería Eléctrica de la Universidad de California en Berkeley, publicó el documento llamado Conjuntos Difusos "*Fuzzy Sets*," artículo que le dio nombre a este campo finalmente. Zadeh aplicó la lógica de Lukasiewicz a cada objeto en un conjunto y construyó el álgebra para los conjuntos difusos. Aún así, los conjuntos difusos no fueron usados hasta mediados de 1970, cuando Ebrahim H. Mamdani, de Queen Mary College en Londres, diseñó un controlador difuso para una máquina de vapor. Desde entonces, el término "*lógica difusa*" ha llegado a significar cualquier sistema matemático o computacional que razona con conjuntos difusos.

Figura 3. Prof. Lotfi A. Zadeh.

La Lógica Difusa está basada en reglas de la forma "*IF…THEN*" que son procesadas luego de convertir las entradas reales en conjuntos difusos. A este paso se le llama

Difusificación. Para construir un sistema difuso, un ingeniero debe empezar con un conjunto de reglas difusas que provengan de un experto. Un ingeniero debe definir los grados y las formas de las funciones de membresía en varios conjuntos de entrada y salida difusos como conjuntos de curvas. La relación entre los conjuntos de entrada y salida podrían entonces ser graficados.

Las reglas de un sistema difuso definen un conjunto de formas traslapadas que relacionan un amplio intervalo de entradas con un amplio intervalo de salidas. En este sentido, los sistemas difusos aproximan a las ecuaciones o funciones matemáticas de la forma *causa y efecto*. Estas funciones son leyes que le dicen a un microprocesador cómo ajustar la potencia de un aire acondicionado o la velocidad de la lavadora en respuesta a alguna medida correspondiente.

Los sistemas difusos pueden aproximar cualquier función matemática continua. El teorema de convergencia afirma que si se proporcionan suficientes funciones de membresía es posible describir cualquier función matemática con la relación entrada/ salida. Este teorema también muestra que podemos elegir el error máximo de aproximación, asegurando que hay un número finito de reglas difusas suficientes para alcanzar este propósito. Un sistema difuso razona, es decir, *infiere*, basado en sus funciones de membresía y sus reglas. A este paso se le llama *Inferencia Difusa*. Dos o más reglas convierten cualquier cantidad de número entrante en un resultado, debido a la forma en que se traslapan sus funciones de membresía.

En su forma difusa, tal curva de salida no puede asistir a los controladores que actúan con instrucciones binarias. Por lo que el paso final es llevar a cabo la *Desdifusificación*, en la cual la curva de salida difusa es convertida en un valor numérico. La técnica más común es computar los resultados del paso de inferencia con el algoritmo de *Centro de la Masa* ó *Centroide*, como el área debajo de la curva. En esta instancia, el centroide de la curva de salida difusa debe corresponder a un valor apropiado que pueda afectar a la salida de manera adecuada y precisa.

Figura 4. Masaki Togai & Hiroyuki Watanabe.

Conforme los sistemas se vuelven más complejos, los antecedentes de las reglas pueden incluir cualquier cantidad de términos unidos por operadores "*and*" o separados por operadores "*or*." Por ejemplo, un acondicionador de aire difuso puede usar una regla que diga "*SI* el aire está fresco *Y* la humedad es alta, *ENTONCES* activa el motor a velocidad media." Los productos difusos usan microprocesadores que ejecutan algoritmos de inferencia difusa y los sensores que miden los cambios en las condiciones de entrada. Los *Chips Difusos* o Controladores Difusos son procesadores diseñados para almacenar y procesar reglas de inferencias difusas. En 1985 *Masaki Togai* y *Hiroyuki Watanabe*, quienes entonces trabajaban en AT & T Bell Laboratories, construyeron el *primer chip difuso digital*. Este procesaba 16 reglas simples en 12.5 microsegundos, a una velocidad de 0.08 millones de inferencias lógico–difusas por segundo (MFLIPS). La compañía Togai InfraLogic, actualmente ofrece chips basados en hardware llamado *Fuzzy Computational Acceleration* que procesa hasta dos millones de reglas por segundo. La mayoría de las firmas de los microprocesadores actualmente tienen proyectos de investigación y desarrollo de chips difusos. Los productos difusos cuentan en su mayoría con microprocesadores estándar que los ingenieros han programado con muy pocas líneas de código de inferencia difusa. La aplicación más famosa de lógica difusa es el controlador del tren del metro usado en Sendai, la cual ha superado tanto a los operadores humanos como a los controladores automáticos convencionales. Los controladores convencionales inician y detienen al tren reaccionando a los marcadores de posición que muestran qué tan cerca está el vehículo de una estación. Debido a que los controladores están rígidamente programados, la marcha solía ser súbita; el control automático aplicaba la misma presión a los frenos cuando el tren estaba, digamos a 100 metros de la estación, aún cuando el tren estuviera andando cuesta arriba o cuesta abajo. A mediados de los 80s los ingenieros de Hitachi usaron reglas difusas para acelerar y frenar el tren lenta y suavemente, aún con más precisión que la que pudieran tener los operadores humanos.

Las reglas cubrían un amplio intervalo de variables acerca del funcionamiento actual del tren, como qué tan frecuentemente, por cuánto su velocidad cambiaba y qué tan cerca la velocidad actual estuvo de la máxima. En pruebas de simulación el controlador difuso superó a una versión automatizada en la medida de la comodidad de la marcha y alcanzó a reducir el 10 por ciento del consumo de energía del tren. Actualmente éste sistema difuso funciona en el metro de Sendai durante las horas pico y también lo usan algunos trenes del metro de Tokio. Los humanos operan el metro durante horas no pico para que no pierdan sus habilidades de conducción.

Compañías en Japón y Corea están construyendo una serie de productos difusos al consumidor que ofrecen un control más preciso que el de los convencionales. Las lavadoras difusas ajustan el ciclo de lavado para cada carga de ropa, cambiando la forma de lavado tan pronto la ropa va quedando limpia. Una lavadora difusa proporciona un lavado más fino que una máquina "torpe" con comandos definidos. Simplemente, un

sensor óptico mide la suciedad o la claridad del agua de lavado y el controlador estima el tiempo que tardará en desmanchar la ropa. Algunas máquinas usan un sensor de carga para poder realizar cambios en la velocidad de agitación o la temperatura del agua que debe usar. Otras disparan burbujas en la tina de la lavadora para disolver las manchas con el detergente. Una lavadora difusa puede usar por lo menos 10 reglas difusas para determinar una amplia variedad de formas de lavado.

En las cámaras digitales y en las cámaras de video, la lógica difusa relaciona los datos de la imagen para elegir entre varias configuraciones de lentes. Una de las primeras grabadoras de video, la Canon H800, que fue puesta en el mercado en 1990, ajusta el auto–enfoque basado en 13 reglas difusas. Los sensores miden la claridad de las imágenes en seis áreas. Las reglas toman aproximadamente un kilobyte de memoria y convierten los datos del sensor para obtener una nueva configuración de lentes. Matsushita agregó más reglas para suprimir los errores de la imagen causadas por los movimientos de la mano en sus pequeñas cámaras de video Panasonic. Las reglas difusas infieren dónde la imagen será desplazada y reajustada. Las reglas consideran los cambios locales y globales en la imagen y entonces realiza una compensación. Por otro lado, los controladores de las cámaras de video basados en modelos matemáticos pueden compensar para pocas situaciones de errores en la imagen, cosa que superan los controladores difusos.

Figura 5. Algunos productos que utilizan lógica difusa.

Los sistemas con controladores difusos frecuentemente son más eficientes con la energía porque calculan de manera más precisa la potencia requerida para hacer su trabajo. Mitsubishi y la Samsung de Corea reportaron que sus aspiradoras difusas

alcanzaron más del 40 por ciento de ahorro de energía a comparación de las aspiradoras no difusas. Este sistema usa leds infrarrojos para medir los cambios en el flujo de polvo y así juzgar si el piso está limpio. Un microprocesador de 4–bits mide el flujo del polvo en el aire aspirado para calcular la potencia de succión apropiada y otras configuraciones de aspiración. La industria automotriz también se beneficia de la lógica difusa. General Motors utiliza una transmisión difusa en su nueva línea de automóviles llamada Saturn. Nissan ha pretendido integrar a sus autos un sistema de frenos difusos anti–derrape, un sistema de transmisión difusa e inyección electrónica difusa. Un conjunto de reglas difusas ajustan el flujo de la gasolina. Los sensores miden la configuración de la marcha, la presión del colector de admisión, la temperatura del agua en el radiador y las revoluciones por minuto del motor. Un segundo conjunto de reglas difusas programa la ignición del motor basado en las revoluciones por minuto, la temperatura del agua y la concentración de oxígeno.

Uno de los sistemas difusos más complejos es un modelo de un helicóptero, diseñado por Michio Sugeno del Instituto de Tecnología de Tokio. Cuatro elementos del elevador de la nave, el alerón, la válvula reguladora y el timón responden a 13 comandos de voz difusos, tales como “elévate,” “aterriza” y “suspéndete.” El controlador difuso puede hacer que la nave se quede suspendida en un lugar, una tarea muy difícil aún para pilotos humanos.

Pocos sistemas difusos manipulan información en relación a la manipulación de dispositivos. Con las reglas de lógica difusa, el conglomerado japonés Omron supervisa cinco bases de datos médicas en un sistema de administración de salud para muchas empresas. Estos sistemas difusos usan 500 reglas para diagnosticar la salud de más de 10,000 pacientes y para elaborarles planes personalizados para ayudarlos a prevenir enfermedades, mantenerse en forma y reducir el estrés.

El talón de Aquiles de un sistema difuso está en sus propias reglas. Casi todos los productos difusos que están actualmente en el mercado dependen de reglas suministradas por un experto. Los ingenieros entonces se dedican a un largo proceso de ajustar ésas reglas y sus conjuntos difusos. Para automatizar este proceso, algunos ingenieros están construyendo *Sistemas Difusos Adaptivos* que usan *Redes Neuronales* u otras herramientas estadísticas para refinar o bien formar ésas reglas iniciales.

Actualmente existen productos electrodomésticos en Japón que usan aprendizaje neuronal supervisado para ajustar las reglas difusas que controlan su operación. Uno de ellos es un horno de micro–ondas de Sanyo y algunas lavadoras de otras compañías. Sharp emplea ésta técnica para modificar las reglas de su refrigerador difuso de manera que el aparato aprende con qué frecuencia su dueño hambriento va a abrir la puerta y de acuerdo a esto ajusta el ciclo de enfriamiento. Estos aparatos se pre–entrenan en el

laboratorio para que el consumidor pueda reprogramarlos. Con esto se espera que los aparatos puedan adaptarse a las necesidades propias de cada usuario.

El control difuso y la inteligencia artificial

El control difuso, como se dijo antes, está relacionado con la inteligencia artificial en una rama que finalmente se estableció como Inteligencia Computacional, una alternativa más entre los sistemas expertos y las redes neurodifusas. Día con día incrementa el uso de tecnología con inteligencia artificial, desde un simple procesador de texto hasta la más complicada base de datos o aparato electrónico para el hogar, tanto que ya forma parte de nuestra vida. Es por eso que es necesario resaltar la herencia que la inteligencia artificial le ha dado al control difuso desde sus orígenes. La definición moderna de la Inteligencia Artificial–*Artificial Intelligence* (AI) es "el estudio y el diseño de agentes inteligentes", donde cada agente es un sistema que percibe su ambiente y toma decisiones que maximizan sus posibilidades de lograr el éxito. John McCarthy, introdujo este término en 1956, definiéndola como la "ciencia y la ingeniería que realiza máquinas inteligentes." Otros nombres para el campo han sido propuestos, tales como *Inteligencia Computacional*, Inteligencia Sintética ó Racionalidad Computacional [40].

El término Inteligencia Artificial es también usado para describir una propiedad de las máquinas o de los programas: la inteligencia que el sistema demuestra. Los rasgos que los investigadores quieren que una máquina exhiba son: razonamiento, conocimiento, planeación, aprendizaje, comunicación, percepción y la habilidad de mover y manipular objetos de tipo real o virtual.

Los investigadores dedicados a AI usan herramientas y enfoques de muchos campos, incluyendo de las ciencias de la computación, psicología, filosofía, neurociencia, ciencias cognitivas, lingüística, investigación de operaciones, economía, teoría de control, probabilidad, optimización y lógica. También se relaciona con otros campos como son la robótica, sistemas de control, programación, minería de datos, logística, reconocimiento de patrones, entre otros.

La inteligencia artificial es una ciencia joven y es aún una colección fragmentada de sub–campos. Hasta ahora, no hay una teoría unificada establecida que enlace los sub–campos en un todo coherente.

El primer período de la Inteligencia Artificial, llamado *sub–simbólico*, data de aproximadamente 1950 a 1965. Este período utilizó representaciones numéricas (o sub–simbólicas) del conocimiento. Aunque la mayor parte de los libros de Inteligencia Artificial

enfatizan el trabajo realizado por Rosenblatt y Widrow con redes neuronales durante este período, la realidad es que otra importante escuela sub–simbólica data también de la misma época y estos son los *Algoritmos Evolutivos*.

La escuela clásica dentro de la Inteligencia Artificial, utiliza representaciones simbólicas basadas en un número finito de primitivas y de reglas para la manipulación de símbolos. El período *simbólico* se considera aproximadamente comprendido entre 1962 y 1975, seguido por un período dominado por los sistemas basados en el conocimiento de 1976 a 1988. Sin embargo, en este segundo período las representaciones simbólicas (por ejemplo, redes semánticas, lógica de predicados, etc.) siguieron siendo parte central de dichos sistemas.

La Programación Lógica tiene sus orígenes más cercanos en los trabajos de J. A. Robinson que propone en 1965 una regla de inferencia a la que llama resolución, mediante la cual la demostración de un teorema puede ser llevada a cabo de manera automática.

En la actualidad, la Inteligencia Artificial empieza a extender sus áreas de investigación en diversas direcciones y trata de integrar diferentes métodos en sistemas a gran escala, tratando de explotar al máximo las ventajas de cada esquema.

La resolución es una regla que se aplica sobre cierto tipo de fórmulas del Cálculo de Predicados de Primer Orden, llamadas *cláusulas* y la demostración de teoremas bajo esta regla de inferencia se lleva a cabo por "*reducción al absurdo*."

Otros trabajos importantes de esa época que influyeron en la programación lógica, fueron los de Loveland, Kowalski y Green, que diseña un probador de teoremas que extrae de la prueba el valor de las variables para las cuales el teorema es válido.

Estos mecanismos de prueba fueron trabajados con mucho entusiasmo durante una época, pero, por su ineficiencia, fueron relegados hasta el nacimiento de *Prolog*, que surge en 1971 en la Universidad de Marsella, Francia.

Actualmente, la programación lógica ha despertado un creciente interés que va mucho más allá del campo de la Inteligencia Artificial y sus aplicaciones. Los japoneses, con sus proyectos de máquinas de la quinta generación, dieron un gran impulso a este paradigma de programación.

La Lógica de *Primer Orden*, es uno de los formalismos más utilizados para representar conocimiento en Inteligencia Artificial. La Lógica cuenta con un lenguaje formal mediante el cual es posible representar fórmulas llamadas *axiomas*, que permiten

describir fragmentos del conocimiento y, además consta de un conjunto de "*reglas de inferencia"* que aplicadas a los axiomas, permiten derivar *nuevo conocimiento* [42].

La inteligencia artificial simbólica

Cuando el acceso a las computadoras digitales fue posible a mediados de los años 50s, los investigadores de inteligencia artificial comenzaron a explorar la posibilidad de que la inteligencia humana pudiera reducirse a una manipulación simbólica. La investigación fue centrada en tres instituciones: CMU, Stanford y el MIT. Estos fueron algunos de los enfoques:

Simulación cognitiva

Los economistas Herbert Simon y Alan Newell estudiaron la habilidad de resolución de problemas humanos e intentaron formalizarla y sus trabajos establecieron las bases del campo de la inteligencia artificial, así como de las ciencias cognitivas, entre otras. Su equipo de investigación realizó experimentos psicológicos para demostrar las similitudes entre la resolución de problemas humanos y los programas que estaban desarrollando.

Inteligencia artificial lógica

A diferencia de Newell y Simon, John McCarthy intuyó que las máquinas no necesitan simular el pensamiento humano, pero en vez de eso deberían tratar de encontrar la esencia del razonamiento abstracto y la resolución de problemas, independientemente de si la gente usó los mismos algoritmos. Su laboratorio en Stanford (SAIL) se enfocó en el uso de la lógica formal para resolver una amplia variedad de problemas, incluyendo la representación del conocimiento, la planeación y el aprendizaje.

Inteligencia artificial simbólica desorganizada

Investigadores en el MIT, tales como Marvin Minsky y Seymour Papert encontraron que resolviendo problemas difíciles en el procesamiento de la visión y el lenguaje natural requerían soluciones *a la medida*. Ellos acordaban que habían balas de plata, es decir, ningún principio simple y general (como en la lógica) que capturara todos los aspectos del comportamiento inteligente. Un logro importante fue admitir que la inteligencia artificial requería de grandes cantidades de conocimientos del sentido común y que este tenía que ser ingeniado en un concepto complicado a la vez. Roger Schank describió su enfoque "anti–lógica"

como un enfoque "desorganizado" y este aún forma parte de la base de las investigaciones del conocimiento del sentido común.

Inteligencia artificial basada en el conocimiento

Cuando las computadoras con amplias memorias estuvieron disponibles cerca de los 70s, los investigadores de estos tres enfoques comenzaron a construir el conocimiento para algunas aplicaciones. Esta revolución del conocimiento llevó al desarrollo y al establecimiento de los *Sistemas Expertos*, la primera forma realmente exitosa de Software Inteligente.

La inteligencia artificial sub–simbólica

Durante los años 60s, los enfoques simbólicos alcanzaron grandes éxitos en la simulación del pensamiento en alto nivel en pequeños programas de demostración. Enfoques basados en la *Cibernética* o en las Redes Neuronales fueron cuestionados y puestos en segundo plano. Cerca de los 80s, sin embargo, los progresos en la inteligencia artificial simbólica parecieron paralizarse y muchos creyeron que los sistemas simbólicos nunca podrían imitar todo el proceso del conocimiento humano, especialmente para la percepción, la robótica, el aprendizaje y el reconocimiento de patrones. Algunos investigadores comenzaron a indagar en un enfoque "sub–simbólico" para problemas específicos:

Inteligencia artificial basada en el comportamiento

Investigadores relacionados con la robótica, como Rodney Brooks, rechazó a la inteligencia artificial simbólica y se enfocó en los problemas de ingeniería básicos que permitieran a los robots moverse y sobrevivir. Su trabajo revivió a la inteligencia artificial con un punto de vista no–simbólico basado en las investigaciones en cibernética recientes e reintrodujeron el uso de la Teoría de Control en la inteligencia artificial. Este enfoque "*bottom–up*" es conocido también como inteligencia artificial basada en el comportamiento, situada, o de *Nouvelle*.

Inteligencia computacional

El interés en las redes neuronales y el "*conexionismo"* fue revivido por David Rumelhart y otros investigadores a mediados de los 80s. Estos y otros enfoques sub–simbólicos, tales como los *Sistemas Difusos* y la *Computación Evolutiva*, son ahora estudiados colectivamente por esta disciplina emergente.

Las nuevas formalidades

En los 90s, los investigadores desarrollaron herramientas matemáticas sofisticadas para resolver sub–problemas específicos. Estas herramientas son realmente científicas, en el sentido de que sus resultados son medibles y verificables y ellos han sido responsables de muchos éxitos recientes. El lenguaje matemático compartido ha permitido también un alto nivel de colaboración con muchos más campos establecidos (como las matemáticas, la economía o la investigación de operaciones). Russel y Norvig describieron a este movimiento como una "revolución" y "la victoria de la formalidad".

El control difuso hereda la resolución mediante reglas de inferencia de la inteligencia artificial a través de la corriente del sub–simbolismo y encuentra a través de la computación una salida para el éxito de las implementaciones de soluciones para la descripción del conocimiento humano.

El control difuso

El control difuso se basa en los conceptos de lógica difusa que instauró Zadeh [28]. Para la formalización de los conceptos de la lógica difusa aplicada al control es necesario conocer la teoría básica del control difuso, usada para comenzar a explicar la arquitectura de un FLC propuesta en este trabajo.

Variables lingüísticas, valores y reglas

Un sistema difuso implica una correspondencia no lineal estática entre sus entradas y sus salidas. Se asume que un sistema difuso tiene entradas $u_i \in U_i$ donde $i = 1, 2, \dots, n$ y salidas $y_i \in Y_i$ donde $i = 1, 2, \dots, m$. Las entradas y las salidas son valores "concretos", esto es, números reales y no son conjuntos difusos. El módulo de Difusificación convierte las entradas concretas en un valor correspondiente de membresía o pertenencia a los conjuntos difusos; el Mecanismo de Inferencia usa la base de reglas difusas para producir conclusiones difusas, esto es, los conjuntos difusos de salida implicados; y el módulo de Desdifusificación convierte esas conclusiones difusas en valores de salida concretos. La siguiente figura muestra un Sistema de Control Difuso de *n* entradas por *m* salida:

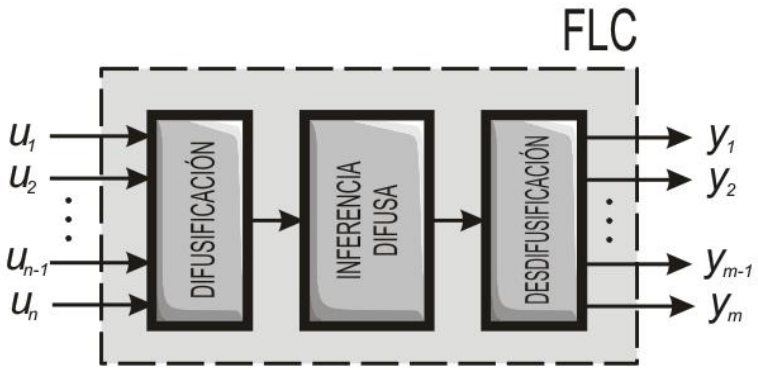

Figura 6. Estructura general del control difuso.

Un FLC por sí solo no tiene sentido hasta que se tienen completamente estudiadas las variables que intervendrán, el sistema o el proceso (la planta) en general y el tipo de control que se realizará. Un FLC funciona para un sistema de control de lazo abierto o para un sistema de control de lazo cerrado perfectamente, de manera similar a un sistema de control convencional, como se muestra en la siguiente figura.

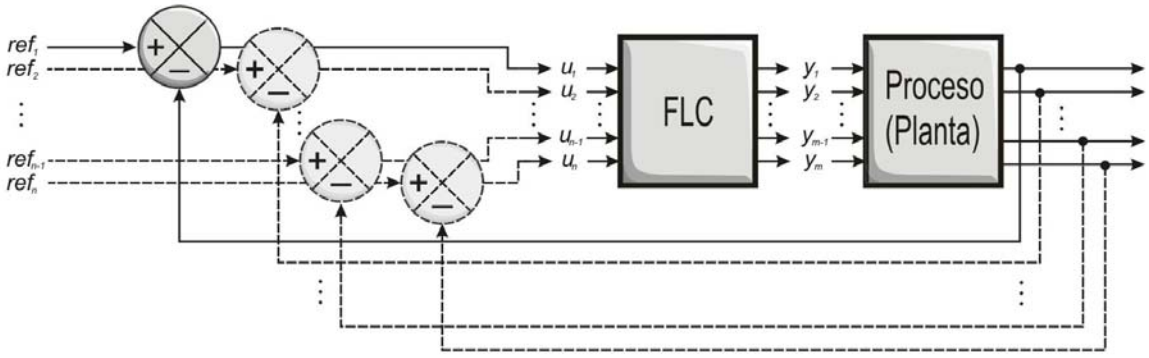

Figura 7. Sistema de control difuso generalizado.

Este es un diagrama generalizado de un sistema de control difuso que puede contener uno o más lazos cerrados anidados o separados, o bien puede tener alguna entrada que no esté retroalimentada.

El universo de discurso

Todos los valores concretos u_i y y_i posibles son llamados “Universo de Discurso” para los conjuntos ordinarios U_i e Y_i, correspondientemente, en otras palabras conforman el dominio de los conjuntos difusos. El “Universo de Discurso Efectivo” representa

aquéllos puntos entre [α, β] donde α y β son puntos en los que las funciones de membresía de los extremos se saturan para los universos de discurso de entrada, o los puntos más allá de los cuales las salidas no afectarán al universo de discurso de salida. Por lo tanto, el "ancho" del universo de discurso está definido por $|\beta - \alpha|$.

Las variables lingüísticas

Para especificar las reglas para la base de conocimiento, el experto usará una "descripción lingüística"; es decir que, son necesarias expresiones lingüísticas para caracterizar las entradas y las salidas. Se hace uso de las "variables lingüísticas" (descripciones simbólicas constantes que son en general cantidades que varían en el tiempo) para describir las entradas y las salidas del sistema difuso. Las variables lingüísticas denotadas por $\tilde{u}_i$ son usadas para describir las entradas u_i. De manera similar, $\tilde{y}_i$ son usadas para describir las salidas y_i.

Los valores lingüísticos

Tan pronto u_i e y_i toman valores a lo largo del universo de discurso U_i e Y_i, respectivamente, las variables lingüísticas $\tilde{u}_i$ e $\tilde{y}_i$ toman *"valores lingüísticos"* que son usados para describir características de las variables. Denotemos a $\tilde{A}_i^j$ como el j–ésimo valor lingüístico de la variable lingüística $\tilde{u}_i$ definida a lo largo del universo de discurso U_i. Si se asume que existen muchos valores lingüísticos definidos en U_i, entonces la variable lingüística $\tilde{u}_i$ adquiere los elementos del conjunto de valores lingüísticos de entrada denotados por

$$\tilde{A}_i = \{\tilde{A}_i^j : j = 1, 2, \dots N_i\}$$

De manera similar $\tilde{B}_i^j$ denota el j–ésimo valor lingüístico de la variable lingüística $\tilde{y}_i$ definida a lo largo del universo de discurso Y_i, entonces la variable lingüística $\tilde{y}_i$ adquiere los elementos del conjunto de valores lingüísticos de salida denotados por

$$\tilde{B}_i = \{B_i^j : j = 1, 2, \dots N_i\}$$

Las reglas lingüísticas

La correspondencia de las entradas con las salidas para un sistema difuso está en parte caracterizada por un conjunto de reglas de acción de la forma

$$\textbf{IF } premise \textbf{ THEN } consequent \qquad (1)$$

Usualmente, las entradas de un sistema difuso son asociadas a partir de la "*premisa*" hasta la "*consecuencia*". Estas reglas **IF–THEN** pueden representarse de varias formas. Por ejemplo, la forma MISO (*Multiple Input–Single Output*) es de la forma

$$\textbf{IF } \tilde{u}_1 \text{ is } \tilde{A}_1^j \textbf{ AND } \tilde{u}_2 \text{is } \tilde{A}_2^k \textbf{ AND}, \dots, \textbf{AND } \tilde{u}_n \text{ is } \tilde{A}_n^l \textbf{ THEN } \tilde{y}_q \text{is } \tilde{B}_q^p \qquad (2)$$

Este es un conjunto completo de reglas lingüísticas que el experto especifica para controlar un sistema. Puede verse fácilmente que la forma MIMO (*Multiple Input–Multiple Output*) para una regla puede descomponerse en varias reglas de la forma MISO. Por ejemplo, una regla MIMO con n entradas y $m = 2$ salidas, como

$$\textbf{IF } \tilde{u}_1 \text{is } \tilde{A}_1^j \textbf{ AND } \tilde{u}_2 \text{is } \tilde{A}_2^k \textbf{ AND}, \dots, \textbf{AND } \tilde{u}_n \text{is } \tilde{A}_n^l \textbf{ THEN } \tilde{y}_1 \text{is } \tilde{B}_1^r \textbf{ AND } \tilde{y}_2 \text{is } \tilde{B}_2^s \qquad (2.1)$$

es lingüísticamente equivalente a las dos reglas siguientes de la forma

$$\textbf{IF } \tilde{u}_1 \text{is } \tilde{A}_1^j \textbf{ AND } \tilde{u}_2 \text{ is } \tilde{A}_2^k \textbf{ AND}, \dots, \textbf{AND } \tilde{u}_n \text{is } \tilde{A}_n^l \textbf{ THEN } \tilde{y}_1 \text{ is } \tilde{B}_1^r \qquad (2.2)$$

$$\textbf{IF } \tilde{u}_1 \text{is } \tilde{A}_1^j \textbf{ AND } \tilde{u}_2 \text{ is } \tilde{A}_2^k \textbf{ AND}, \dots, \textbf{AND } \tilde{u}_n \text{ is } \tilde{A}_n^l \textbf{ THEN } \tilde{y}_2 \text{ is } \tilde{B}_2^s \qquad (2.3)$$

Esto sucede debido a que el operador lógico "and" en la consecuencia de la regla MIMO asegura que ambas consecuencias son válidas. Para la implementación, se deben especificar dos sistemas difusos, uno para la salida y_1 y otra para y_2.

Se asume que hay un total de reglas R en la base de reglas y naturalmente se asume que las reglas en la base de reglas son distintas todas; sin embargo, no siempre tiene que ser así. Por simplicidad se usan duplas

$$(j, k, \dots, l; p, q)_i$$

Para denotar la i–*ésima* regla MISO de la forma dada en la ecuación (2). Cualquiera de los términos asociados con cualquiera de las entradas para cualquier regla MISO puede ser incluida u omitida.

Finalmente, nótese que si todos los términos de las premisas son usados en cada regla y una regla es formada por cada combinación posible de los elementos de la premisa, entonces hay

$$R = \prod_{i=1}^{n} N_i = N_1 \times N_2 \times \dots \times N_n \qquad (3)$$

reglas en la base de reglas, donde R es el número de reglas y N es el número de funciones de membresía de cada una de las entradas. Claramente, en este caso el número de reglas

incrementa exponencialmente conforme incrementan el número de entradas al controlador difuso o el número de funciones de membresía de cada entrada.

Las funciones de membresía

Los conjuntos difusos y la lógica difusa son usados para cuantificar heurísticamente la representación de las variables lingüísticas, de los valores lingüísticos y de las reglas lingüísticas que son especificadas por el "*experto*". El concepto *Conjunto Difuso* es introducido definiendo primero lo que es una "Función de Membresía."

Denotemos a $\mathrm{U_i}$ como un universo de discurso y a $\widetilde{\mathrm{A}}_{\mathrm{i}}^{\mathrm{j}} \in \widetilde{\mathrm{A}}_{\mathrm{i}}$ como un valor lingüístico específico para la variable lingüística $\tilde{\mathrm{u}}_{\mathrm{i}}$. La función $\mu(u_i)$ asociada al valor lingüístico $\widetilde{\mathrm{A}}_{\mathrm{i}}^{\mathrm{j}}$ que corresponde al universo de discurso U_i en [0, 1] es llamada "*función de membresía*." Esta función de membresía describe la "certeza" de que un elemento de U_i, denotado por u_i, con una descripción lingüística $\tilde{\mathrm{u}}_{\mathrm{i}}$, puede ser clasificado lingüísticamente como $\widetilde{\mathrm{A}}_{\mathrm{i}}^{\mathrm{j}}$. Las funciones de membresía son subjetivamente especificadas de una manera heurística a partir de la experiencia o la intuición.

Una función de membresía puede tomar cualquier forma posible (por ejemplo, una forma triangular o una forma trapezoidal), y cada forma proporcionará una medida diferente de los valores lingüísticos que cuantifican. La Figura 8 ilustra la variedad de formas de funciones de membresía.

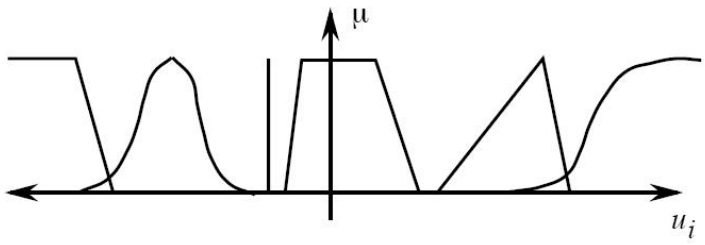

Figura 8. Varias formas de Funciones de Membresía.

Los conjuntos difusos

Dada la variable lingüística $\tilde{\mathrm{u}}_{\mathrm{i}}$ con un valor lingüístico $\widetilde{\mathrm{A}}_{\mathrm{i}}^{\mathrm{j}}$ definida en el universo de discurso U_i y una función de membresía $\mu_{A_i^j}(u_i)$ (función de membresía asociada con el conjunto difuso A_i^j) que corresponde a U_i en los valores [0, 1], un *"conjunto difuso"* denotado por A_i^j está definido por

$$A_i^j = \left\{ \left(u_i,\ \mu_{A_i^j}(u_i) \right) : u_i \in U_i \right\} \quad (4)$$

Nótese que un conjunto difuso es simplemente un conjunto concreto de pares ordenados conformados por el universo de discurso y sus valores de membresía asociados.

La lógica difusa

A continuación se definen algunas operaciones lógicas con las que se puede trabajar con conjuntos difusos. Cabe destacar que no son aplicables a la teoría de conjuntos tradicional, sin embargo son muy parecidas:

El subconjunto difuso

Dados dos conjuntos difusos A_i^1 y A_i^2 asociados con el universo de discurso U_i $(N_i = 2)$, con funciones de membresía denotadas como $\mu_{A_i^1}(u_i)$ y $\mu_{A_i^2}(u_i)$ respectivamente, A_i^1 se define como un subconjunto difuso de A_i^2, denotado por $A_i^1 \subset A_i^2$, si $\mu_{A_i^1}(u_i) \leq \mu_{A_i^2}(u_i)$ para todo $u_i \in U_i$.

El complemento difuso

El complemento "not" de un conjunto difuso A_i^1 con una función de membresía $\mu_{A_i^1}(u_i)$ tiene una función de membresía dada por $\mu_{\neg A_i^1}(u_i) = 1 - \mu_{A_i^1}(u_i)$.

La intersección difusa (AND)

La intersección difusa de los conjuntos difusos A_i^1 y A_i^2, los cuales están definidos en el universo de discurso U_i, es un conjunto difuso denotado por $A_i^1 \cap A_i^2$, con una función de membresía definida por cualquiera de los siguientes dos métodos:

El mínimo (MIN)

El valor mínimo existente entre las intersecciones de dos conjuntos difusos está definido por:

$$\mu_{A_i^1 \cap A_i^2} = min\left\{ \mu_{A_i^1}(u_i),\ \mu_{A_i^2}(u_i) : u_i \in U_i \right\} \quad (5)$$

El producto algebraico

El producto algebraico que corresponde a las intersecciones de dos conjuntos está definido por:

$$\mu_{A_i^1 \cap A_i^2} = \mu_{A_i^1}(u_i)\, \mu_{A_i^2}(u_i) : u_i \in U_i \qquad (6)$$

Suponga que se usa la notación $x * y = min\{x, y\}$, o que algunas otras veces se use esta notación para denotar el producto $x * y = xy$ (el símbolo $*$ es algunas veces llamado "norma triangular"). Por lo tanto es necesario hacer una distinción, entonces la expresión $\mu_{A_i^1}(u_i) * \mu_{A_i^2}(u_i)$ es una representación general para la intersección de dos conjuntos difusos. En la lógica difusa, la intersección es usada para representar a la operación lógica "and". Esta cuantificación de la operación "and" proporciona la justificación fundamental para la representación **AND** en la premisa de una regla.

Figura 9. Representación de la intersección entre dos conjuntos difusos.

La unión difusa (OR)

La unión difusa de los conjuntos difusos A_i^1 y A_i^2, los cuales están definidos en el universo de discurso U_i, es un conjunto difuso denotado por $A_i^1 \cup A_i^2$, con una función de membresía definida por cualquiera de los siguientes dos métodos:

El máximo (MAX)

El valor máximo existente entre las uniones de dos conjuntos difusos está definido por:

$$\mu_{A_i^1 \cup A_i^2} = max\left\{ \mu_{A_i^1}(u_i),\ \mu_{A_i^2}(u_i) : u_i \in U_i \right\} \quad (7)$$

La suma algebraica

La suma algebraica que corresponde a las uniones de dos conjuntos está definido por:

$$\mu_{A_i^1 \cup A_i^2} = \mu_{A_i^1}(u_i) + \mu_{A_i^2}(u_i) - \mu_{A_i^1}(u_i)\mu_{A_i^2}(u_i) : u_i \in U_i \quad (8)$$

Suponga que se usa la notación $x \oplus y = max\{x, y\}$, o que algunas otras veces se use para denotar $x \oplus y = x + y - xy$ (el símbolo $\oplus$ es a veces llamado "co–norma triangular"). Por lo tanto es necesario hacer una distinción, entonces la expresión $\mu_{A_i^1}(u_i) \oplus \mu_{A_i^2}(u_i)$ es una representación general de la unión de dos conjuntos difusos. En la lógica difusa, la unión es usada para representar la operación lógica "or". Esta cuantificación de la operación "or" proporciona la justificación fundamental de que esta operación tiene una relación inherente con las reglas en la base de reglas (nótese que cada elemento de la lista de reglas del sistema difuso debe interpretarse como uniones, operaciones **OR**).

Figura 10. Representación de la unión entre dos conjuntos difusos.

Las difusificación

Los conjuntos difusos son usados para cuantificar la información en la base de reglas y el mecanismo de inferencia opera sobre conjuntos difusos para producir conjuntos difusos; por lo tanto, se debe especificar la manera de cómo el sistema difuso convertirá sus entradas numéricas $u_i \in U_i$ en conjuntos difusos de manera que estos puedan ser usados por el sistema difuso. A este proceso se le llama "Difusificación." Denotemos a U_i^* como el conjunto de todos los conjuntos difusos posibles que pueden ser

definidos en U_i. Dado que $u_i \in U_i$, la Difusificación transforma u_i a un conjunto difuso denotado por $\hat{A}_i^{fuz}$ definido sobre el universo de discurso U_i. Estas transformaciones son producidas por el Operador de Difusificación $\mathcal{F}$ definido por:

$$\mathcal{F}: U_i \longrightarrow U_i^*$$

donde

$$\mathcal{F}(u_i) = \hat{A}_i^{fuz}$$

Frecuentemente es usada la "Difusificación con Singleton", la cual produce un conjunto difuso $\hat{A}_i^{fuz} \in U_i^*$ con una función de membresía definida por

$$\mu_{\hat{A}_i^{fuz}}(x) = \begin{cases} 1, & x = u_i \\ 0, & de\ lo\ contrario \end{cases} \qquad (9)$$

Cualquier conjunto difuso con esta forma por su función de membresía es llamado *"Singleton."* Es una función de membresía que representa un conjunto difuso en donde existe un valor de membresía máximo (uno) en un único valor en su universo de discurso y para el resto del mismo es totalmente excluyente (cero). Es un caso particular de conjunto difuso que se asemeja a un conjunto lógico tradicional, donde un valor sólo puede corresponder a un solo conjunto y no a otro. Su forma es muy similar a la del impulso unitario, descrita en las matemáticas. La Difusificación con Singleton es generalmente usada en las implementaciones debido a que en ausencia de ruido podemos estar absolutamente seguros de que u_i toma su valor correspondiente (y ningún otro valor) y debido a que proporciona ciertos ahorros en las necesidades de cómputo para implementar un sistema difuso (en el desdifusificador).

El mecanismo de inferencia

El mecanismo de inferencia tiene dos tareas básicas: (1) determinar el grado en el cual cada regla es relevante a la situación actual tal como es caracterizado por las entradas u_i, donde $i = 1, 2, \ldots, n$ (a esta tarea se le llama "comparación"); e (2) inferir las conclusiones usando las entradas actuales u_i y la información en la base de reglas (a esta tarea se le llama "paso de inferencia"). Para realizar la comparación note que $A_1^j \times A_1^k \times \ldots \times A_1^l$ es el conjunto difuso que representa la premisa de la i–*ésima* regla $(j, k, \ldots, l; p, q)_i$ (pueden haber más de una regla para una premisa).

A continuación se detallan éstas dos tareas:

La comparación

Suponga que en algún momento se obtienen las entradas u_i, donde $i = 1, 2, \dots, n$, y la Difusificación produce los conjuntos difusos representando las entradas

$$\hat{A}_1^{fuz}, \hat{A}_2^{fuz}, \hat{A}_3^{fuz}, \dots, \hat{A}_n^{fuz}$$

Hay entonces dos pasos básicos para realizar la comparación:

Combinar las entradas con las premisas de la regla

El primer paso de comparación involucra encontrar conjuntos difusos $\hat{A}_1^j, \hat{A}_2^k, \dots, \hat{A}_n^l$ con funciones de membresía

$$\mu_{\hat{A}_1^j}(u_1) = \mu_{A_1^j}(u_1) * \mu_{\hat{A}_1^{fuzz}}(u_1)$$

$$\mu_{\hat{A}_2^k}(u_1) = \mu_{A_2^k}(u_1) * \mu_{\hat{A}_2^{fuzz}}(u_1)$$

.
.
.

$$\mu_{\hat{A}_n^l}(u_1) = \mu_{A_n^l}(u_1) * \mu_{\hat{A}_n^{fuzz}}(u_1)$$

(para toda j, k, . . ., l) que combinen los conjuntos difusos obtenidos de la difusificación con los conjuntos difusos usados en cada término en las premisas de las reglas. Si se usa difusificación con singleton, entonces cada conjunto difuso es un singleton que está delimitado por la función de membresía de la premisa (esto es, $\mu_{\hat{A}_1^j}(u_1) = \mu_{A_1^j}(\overline{u_1})$ para $\overline{u_1} = u_1$ y $\mu_{\hat{A}_1^j}(\overline{u_1}) = 0$ para $\overline{u_1} \neq u_1$). Esto es, con la difusificación con singleton se tiene que $\mu_{\hat{A}_i^{fuzz}}(u_1) = 1$ para toda $i = 1, 2, \dots, n$ para las entradas dadas $u_i$tales que

$$\mu_{\hat{A}_1^j}(u_1) = \mu_{A_1^j}(u_1)$$

$$\mu_{\hat{A}_2^k}(u_1) = \mu_{A_2^k}(u_1)$$

.
.
.

$$\mu_{\hat{A}_n^l}(u_1) = \mu_{A_n^l}(u_1)$$

Se puede ver que cuando se usa la difusificación con singleton, la combinación de conjuntos difusos que fueron creados por el proceso de difusificación para representar las entradas con las funciones de membresía de la premisa para las reglas es particularmente simple. Simplemente reduce la computación de los valores de membresía de los conjuntos difusos para las entradas dadas $u_1, u_2, \dots, u_n$.

Determinar cuáles reglas están activas

En el segundo paso, se deben formar valores de membresía para la i–*ésima* premisa de la regla (la que suele llamarse $\mu_{premise}$) que representa la certeza de que cada premisa de la regla se mantiene para una entrada dada. Se define

$$\mu_i(u_1, u_2, \dots, u_n) = \mu_{\hat{A}_1^j}(u_1) * \mu_{\hat{A}_2^k}(u_2) * \dots * \mu_{\hat{A}_n^l}(u_n) \qquad (10)$$

la cual es simplemente una función de las entradas μ_i. Cuando la difusificación con singleton es usada se tiene

$$\mu_i(u_1, u_2, \dots, u_n) = \mu_{A_1^j}(u_1) * \mu_{A_2^k}(u_2) * \dots * \mu_{A_n^l}(u_n) \qquad (11)$$

Se usa $\mu_i(u_1, u_2, \dots, u_n)$ para representar la certeza de que una regla i se empareje con la información de entrada cuando se usa la difusificación con singleton. Este método representa la certeza de que una premisa de una regla se cumpla y por lo tanto representa el grado al cual una regla en particular se mantiene para un conjunto dado de valores de entrada.

<u>El paso de inferencia</u>

Existen dos alternativas para realizar el paso de inferencia, una que involucra el uso de varios conjuntos difusos implicados y otro que usa un conjunto difuso general implicado.

Determinar los conjuntos difusos implicados

El paso de inferencia es realizado computando, para la i–*ésima* regla, los "conjuntos difusos implicados" que pertenecen a $\hat{B}_q^i$ con función de membresía

$$\mu_{\hat{B}_q^i}(y_q) = \mu_i(u_1, u_2, \dots, u_n) * \mu_{B_q^p}(y_q) \qquad (12)$$

Un conjunto difuso implicado $\widehat{B}_q^i$ especifica la certeza de que la salida sea un valor concreto y específico y_q dentro de su universo de discurso Y_q, tomando a consideración solamente la regla i. Nótese que ya que $\mu_i(u_1, u_2, \ldots, u_n)$ cambiará con el tiempo, de la misma manera la forma de las funciones de membresía $\mu_{\widehat{B}_q^i}(y_q)$ para cada regla lo hará.

Determinar el conjunto difuso general implicado

Alternativamente, un mecanismo de inferencia podría, además, computar el "conjunto difuso general implicado" $\hat{B}_q$ con una función de membresía

$$\mu_{\hat{B}_q}(y_q) = \mu_{\hat{B}_q^1}(y_q) \oplus \mu_{\hat{B}_q^2}(y_q) \oplus \ldots \oplus \mu_{B_q^R}(y_q) \qquad (13)$$

que representa la conclusión alcanzada considerando a todas las reglas de la base de reglas al mismo tiempo (nótese que determinar $\hat{B}_q$ puede, en general, requerir recursos computacionales significativos).

La desdifusificación

Existen varias estrategias para realizar la Desdifusificación. Cada una proporciona una manera de elegir un valor para una salida (la cual se denotará como y_q^{crisp}) basado en los conjuntos difusos implicados o en el conjunto difuso general implicado (dependiendo del tipo de estrategia de inferencia elegida en la sección anterior).

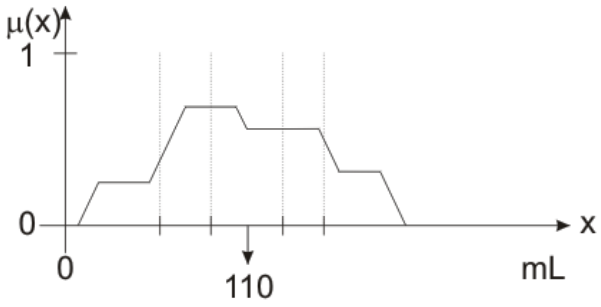

Figura 11. La desdifusificación convierte valores ambiguos en valores concretos.

Desdifusificación: Conjuntos difusos implicados

Como este tipo de desdifusificación es la más común, se especifican las técnicas de desdifusificación para los conjuntos difusos implicados $\widehat{\mathrm{B}}_{\mathrm{q}}^{\mathrm{i}}$:

El centro de gravedad (COG)

Un valor concreto de salida y_q^{crisp} es elegido usando el centro del área y el área de cada conjunto difuso implicado y está dado por

$$y_q^{crisp} = \frac{\sum_{i=1}^{R} b_i^q \int_{y_q} \mu_{\hat{B}_q^i}(y_q)\, dy_q}{\sum_{i=1}^{R} \int_{y_q} \mu_{\hat{B}_q^i}(y_q)\, dy_q}$$

donde R es el número de reglas, b_i^q es el centro del área de la función de membresía de B_i^q asociado con los conjuntos difusos implicados $\hat{B}_q^i$ para la i–*ésima* regla y

$$\int_{y_q} \mu_{\hat{B}_q^i}(y_q)\, dy_q$$

denota el área bajo $\mu_{\widehat{\mathrm{B}}_{\mathrm{q}}^{\mathrm{i}}}(\mathrm{y_q})$. Nótese que el método COG puede ser fácil de computar ya que es frecuentemente sencillo encontrar expresiones de forma cerrada para $\int_{\mathrm{y_q}} \mu_{\widehat{\mathrm{B}}_{\mathrm{q}}^{\mathrm{i}}}(\mathrm{y_q})\, \mathrm{dy_q}$, la cual representa el área bajo la curva que describe la función de membresía. Note que el área debajo de cada conjunto difuso implicado debe ser computable, de tal manera que el área debajo de cada función de membresía de salida (que son usadas en la consecuencia de la regla) debe ser finita (esto es porque no se pueden "saturar" las funciones de membresía en los valores de las orillas del universo de discurso). También, nótese que el sistema difuso debe estar definido de tal manera que

$$\sum_{i=1}^{R} \int_{y_q} \mu_{\hat{B}_q^i}(y_q)\, dy_q \neq 0$$

Este valor será diferente de cero si hay al menos una regla que esté activa para todas las combinaciones posibles de entradas al sistema difuso y los conjuntos difusos consecuentes no serán áreas iguales a cero.

El promedio de los centros

Se elige un valor concreto de salida y_q^{crisp} usando los centros de las funciones de membresía de salida y la certeza máxima de cada una de las conclusiones representadas con los conjuntos difusos implicados y está dado por

$$y_q^{crisp} = \frac{\sum_{i=1}^{R} b_i^q sup_{y_q}\left\{\mu_{\hat{B}_q^i}(y_q)\right\}}{\sum_{i=1}^{R} sup_{y_q}\left\{\mu_{\hat{B}_q^i}(y_q)\right\}}$$

donde "sup" denota "*supremum*" (esto es, el límite superior que puede ser expresado como el valor máximo). Por lo tanto, $sup\{\mu(x)\}$ es el valor más grande que puede alcanzar $\mu(x)$. También, b_i^q es el centro de la función de membresía B_i^q asociado con los conjuntos difusos implicados $\hat{B}_q^i$ para la i–ésima regla. Nótese que el sistema difuso debe cumplir con

$$\sum_{i=1}^{R} sup_{y_q}\left\{\mu_{\hat{B}_q^i}(y_q)\right\} \neq 0$$

para todos los valores de u_i. También, note que $sup_{y_q}\left\{\mu_{\hat{B}_q^i}(y_q)\right\}$ es frecuentemente muy fácil de computar debido a que si $\mu_{\hat{B}_q^i}(y_q) = 1$ para al menos de un valor de y_q (la cual es la manera de definir las funciones de membresía consecuentes), entonces para muchas estrategias de inferencia, usando la Ecuación (12), se tiene que

$$sup_{y_q}\left\{\mu_{\hat{B}_q^i}(y_q)\right\} = \mu_i(u_1, u_2, \ldots, u_n)$$

Entonces, la fórmula de Desdifusificación por promedio de los centros queda de la siguiente manera:

$$y_q^{crisp} = \frac{\sum_{i=1}^{R} b_i^q \mu_i(u_1, u_2, \ldots, u_n)}{\sum_{i=1}^{R} \mu_i(u_1, u_2, \ldots, u_n)} \qquad (14)$$

donde se debe asegurar que $\sum_{i=1}^{R} \mu_i(u_1, u_2, \ldots, u_n) \neq 0$ para todos los valores de u_i. También nótese que esto implica que *la forma* de las funciones de membresía de los conjuntos de salida *no importan*; por lo tanto, pueden usarse simplemente singletons centrados en las posiciones

apropiadas en las funciones de membresía de salida en vez de usar las formas de las funciones de membresía completas.

Desdifusificación: Conjunto difuso general implicado

El criterio del máximo

Se elige una salida concreta y_q^{crisp} como el punto en el universo de discurso Y_q para la cual el conjunto difuso general implicado $\hat{B}_q$ alcanza un valor máximo, esto es

$$y_q^{crisp} \in \left\{ \arg sup_{y_q} \left\{ \mu_{\hat{B}_q}(y_q) \right\} \right\}$$

Aquí, "$\arg sup_x \{\mu(x)\}$" devuelve el valor de x que resulta de la operación supremum que la función $\mu(x)$ haya alcanzado. Algunas veces el supremum puede ocurrir en más de un punto del universo de discurso Y_q. Es por eso que esta estrategia de desdifusificación es evitada debido a esta ambigüedad; sin embargo, el siguiente método de desdifusificación ofrece una solución viable.

La media del máximo

Es elegida una salida concreta y_q^{crisp} para representar el valor medio de todos los elementos cuyas funciones de membresía en el conjunto difuso general implicado $\hat{B}_q$ sean un máximo. Se define a b_q^{max} como el supremum de la función de membresía de $\hat{B}_q$ sobre el universo de discurso Y_q. Además, se define un conjunto difuso $\hat{B}_q^* \in Y_q$ cuya función de membresía está definida como

$$\hat{B}_q^* = \begin{cases} 1, & \mu_{\hat{B}_q}(y_q) = b_q^{max} \\ 0, & de\ lo\ contrario \end{cases}$$

Entonces el valor concreto de salida, usando el método de la media del máximo está definido como

$$y_q^{crisp} = \frac{\int_{y_q} y_q \mu_{\hat{B}_q^*}(y_q)\, dy_q}{\int_{y_q} \mu_{\hat{B}_q^*}(y_q)\, dy_q} \qquad (15)$$

donde el sistema difuso debe estar definido de tal manera que $\int_{y_q} \mu_{\hat{B}_q^*}(y_q)\, dy_q \neq 0$ para todo u_i. Nótese que las integrales de la Ecuación (15) deben computarse a cada instante de tiempo ya que dependen de $\hat{B}_q$, que cambia con el tiempo. Esto puede requerir excesivos recursos computacionales para universos de discurso continuos. Para algunos tipos de funciones de membresía, ideas simples basadas en la geometría pueden usarse para simplificar los cálculos; sin embargo, algunas elecciones respecto a la forma de las funciones de membresía, pueden contener muchos valores máximos dispersos en varios sub intervalos a lo largo del universo de discurso. En estos casos puede resultar difícil computar el valor desdifusificado a menos que las funciones de membresía estén discretizadas. Estas son algunas de las complicaciones a las que se puede enfrentar un diseñador de controladores difusos.

El centro del área

Es elegido un valor concreto de salida y_q^{crisp} como el centro del área de la función de membresía del conjunto difuso general implicado en $\hat{B}_q$. Para universos de discurso de salida continuos Y_q, la salida de la estrategia del centro del área está denotada por

$$y_q^{crisp} = \frac{\int_{y_q} y_q \mu_{\hat{B}_q}(y_q)\, dy_q}{\int_{y_q} y_q \mu_{\hat{B}_q}(y_q)\, dy_q}$$

Este sistema difuso debe estar definido tal que $\int_{y_q} y_q \mu_{\hat{B}_q}(y_q)\, dy_q \neq 0$ para todo u_i. Nótese que, de manera similar al método de la media de los máximos, este enfoque puede ser computacionalmente costoso. Además, la computación del área del conjunto difuso general implicado no cuantifica el área de los conjuntos que se traslapan dos veces; por lo tanto, el área del conjunto difuso general implicado puede ser mucho más difícil de computar en tiempo real.

Es importante notar que cada ecuación antes descrita para la desdifusificación actualmente proporciona una cuantificación matemática de la operación de todo el sistema difuso, proporcionando en cada uno de los términos una descripción definida completa.

De manera general, puede verse que el uso del conjunto difuso general implicado para la desdifusificación es frecuentemente indeseable por dos razones: primero, el conjunto difuso general implicado $\hat{B}_q$ es en sí difícil de computar y segundo, las técnicas de desdifusificación basadas en un mecanismo de inferencia que proporciona $\hat{B}_q$ también son difíciles de computar. Esta es la razón de que en la mayoría de los controladores difusos existentes usen las técnicas de desdifusificación basadas en varios conjuntos difusos implicados.

La tecnología ASIC

Un controlador difuso puede implementarse en muchas tecnologías existentes, desde un microcontrolador hasta una PC, pero es conveniente migrar a una tecnología que sea capaz de responder de manera adecuada en tiempo real [30]. Así fue como los controladores difusos comenzaron a diseñarse en la tecnología VLSI, migrando posteriormente a los ASIC que en esta sección se pretende describir.

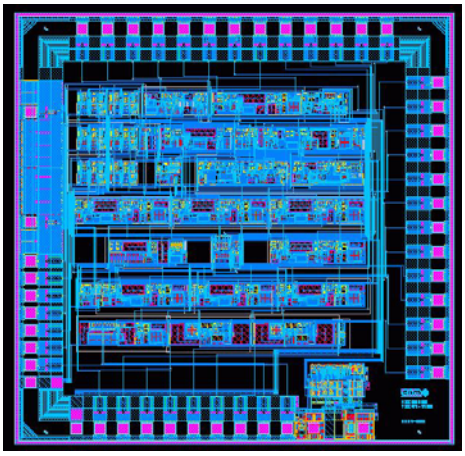

Figura 12. Fotografía de un circuito ASIC para aplicaciones de audio.

Un ASIC–*Aplication Specific Integrate Circuit* ó Circuito Integrado de Aplicación Específica, es un circuito integrado configurable diseñado para un propósito u aplicación específica, para un producto electrónico específico.

Con los últimos avances en las tecnologías de miniaturización y las herramientas de diseño, la complejidad máxima, y por ende la funcionalidad, en un ASIC ha crecido desde 5,000 compuertas lógicas hasta más de 100 millones. Los ASIC modernos a menudo incluyen otros elementos prediseñados tales como:

- Procesadores de n–bit.
- Bloques de memoria RAM, ROM, EEPROM y memoria flash.
- DSP.
- Amplificadores analógicos.
- Otros tipos de módulos caracterizados por el consumidor tales como interfaces o codificadores.

Este tipo de ASIC frecuentemente es llamado Sistema en un solo Chip, o SoC por sus siglas en inglés (System–on–a–Chip). Los diseñadores de ASIC digitales usan lenguajes de descripción de hardware (HDL), tales como Verilog o VHDL, para describir la funcionalidad de estos dispositivos. Los niveles de configuración de un ASIC pueden estar en el campo de lo físico (construcción del hardware) o de lo lógico (configuración por software). Ello depende del subconjunto o tipo de ASIC que se emplee.

Desde los años 70, se ha llevado a cabo el desarrollo de la microelectrónica creando nuevas tecnologías evolutivas para lograrlo, pero no fue sino hasta 1980 que los ingenieros de la compañía *Ferranti*, exploraron las ventajas del diseño de un IC configurable o adaptable para un sistema o aplicación en particular, más allá de usar IC estándar. La microelectrónica crea el paso en el cual la implementación de un IC estándar puede lograrse utilizando funciones lógicas con uno o más IC configurables. Como la tecnología VLSI (Very Low Scale Integration) hace posible la construcción de un sistema con muchos componentes más pequeños se pueden combinar muchos IC estándares dentro de un IC configurable.

Ferranti, empresa de Ucrania, fue la primera en producir los primeros arreglos de compuertas. La adaptación del arreglo se produce cuando es cambiada la máscara de interconexión metálica. Las ULA (Uncommitted Logic Array) consideradas como uno de los primeros IC semi configurables desarrolladas, en principio tenían unos cientos de compuertas para luego extender la gamma y hacer otros modelos que incluyen elementos de memoria RAM.

Estos IC están hechos sobre una oblea de silicio de algunos microns de grosor, cada oblea mantiene unos cientos de IC llamados muertos. Los transistores y el cableado están hechos de muchas capas cuyo número está entre unas 10 y 15 todas distintas entre sí, dispuestas una sobre la otra e interconectadas según los requerimientos. Cada capa tiene un patrón que es definido utilizando una máscara similar a una diapositiva de

fotografía. La primera mitad de las capas definen a los transistores y la segunda mitad a las interconexiones entre ellos.

A continuación se describen los diferentes ASIC en orden de aparición, las ventajas y desventajas de cada una de estas tecnologías:

ASIC semi configurable

Para el caso de los ASIC más utilizados tenemos a los semi configurables. En estos las celdas lógicas ya han sido pre configuradas y solo pueden alterarse las configuraciones de todas las máscaras de interconexión. Al utilizar este método, el trabajo del diseñador se hace mucho más fácil. Aun para esta clase de ASIC existen dos categorías las cuales son:

Celdas estándar (Standard–Cell–Based).

En Japón es un término conocido coloquialmente como *CBIC* pronunciado como "sea–bick". Utiliza celdas lógicas prediseñadas tales como compuestas AND, compuertas OR, multiplexores y flip–flops, se les conoce como *"Celdas Estándar"*.

Las áreas de las CBIC conocidas como bloques flexibles están compuestos por columnas de celdas estándar como una pared de ladrillos. El área de las celdas estándar puede utilizarse en combinación con celdas mucho más grandes o quizás con microcontroladores o microprocesadores conocidos como mega–celdas. Las mega–celdas también son llamadas mega–funciones, bloques completamente configurables, macros de nivel de sistemas (SLM), bloques fijos, núcleos, o bloques de funcionalidad estándar (FSB).

Los diseñadores de estos ASIC solo definen el lugar de las celdas estándar y la interconectividad dentro de un CBIC. Sin embargo, la celda estándar puede ser ubicada en cualquier lugar de la pastilla de silicio; esto permite que todas las mascaras de un CBIC puedan ser configurables para un consumidor en particular. La ventaja de CIBIC es que los diseñadores ahorran tiempo y reducen el riesgo al utilizar librerías de celdas pre–caracterizadas y probadas, diseñadas utilizando las técnicas de una celda completamente configurable. Adicionalmente cada celda estándar puede ser optimizada individualmente. Durante el diseño de cada celda de la librería, cada transistor ha sido elegido para maximizar la velocidad y el área que ocupa en el IC. La desventaja es el tiempo o costo de diseño o la

compra de la librería de celdas estandarizadas y el tiempo que requiere fabricar todas las capas del ASIC para el nuevo diseño.

El diseño de celdas estándar permite la automatización del proceso de ensamble de un ASIC. Grupos de estas celdas pueden acomodarse en forma de columnas, las columnas forman pilas verticales para formar a su vez bloques flexibles rectangulares. Puede interconectarse a otro bloque de celdas estándar de otro bloque o con otros bloques completamente configurables. Por ejemplo, puede desearse incluir una interfaz específica o un microcontrolador junto con algo de memoria. El bloque del microcontrolador puede ser una mega celda fija, a partir de los bloques puede generarse memoria utilizando un compilador de memoria y un controlador de memoria personalizado que puede construirse dentro de un bloque de celdas estándar.

Arreglos de compuertas (Gate Array).

En un ASIC basado en arreglo de compuertas los transistores están predefinidos en una oblea de silicio. Los patrones de definición de los transistores de un arreglo de compuertas y el elementos mas pequeño es replicado para hacer la base del arreglo, así como los dibujos de la porcelana en el piso, a este diseño primario se le llama la C*elda Primitiva*. Solo la capa superior tiene definida las interconexiones entre los transistores. Para distinguir este tipo de arreglo de compuertas de otros tipos de arreglos de compuertas este frecuentemente es llamado *Máscara de Arreglo de Compuertas* o *MGA* por sus siglas en ingles. El diseñador elige de una librería de arreglos de celdas pre–caracterizadas o prediseñadas. Las celdas lógicas de la librería de arreglo de compuertas frecuentemente son llamadas *Macros*. La razón de esto es que, el diseño de la celda base es el mismo para todas y la interconexión entre ellas es lo que puede configurarse libremente.

Puede hacerse la difusión entre varias obleas de silicio de varios consumidores según sea necesario. Utilizando las obleas de silicio prefabricadas se reducen los tiempos de metalización requeridos para hacer un MGA. Algunos de los diferentes tipos de ASIC basados en arreglo de compuertas existentes de acuerdo a su modalidad de construcción son:

- Arreglo de compuertas acanalados.
- Arreglo de compuertas sin acanalado.

- Arreglo de compuertas estructurado.

Los ASIC basados en arreglo de compuertas y los basados en celdas utilizan celdas predefinidas, pero la diferencia es que en una celda estándar puede cambiarse el tamaño de los transistores para optimizar el desempeño y la velocidad, pero el tamaño de los componentes en un arreglo de compuertas es fijo. Esto puede resultar en una disyuntiva entre el área de un arreglo de compuertas en el silicio.

ASIC completamente configurable

Un ASIC completamente configurable tiene probablemente todos los elementos lógicos configurables y adicionalmente todas sus capas son configurables. Un microprocesador es un ejemplo de un circuito integrado completamente configurable, en él los diseñadores invierten muchas horas de trabajo para configurar completamente una sección de no más de una micra cuadrada.

En este tipo de ASIC se pueden diseñar una o todas las celdas lógicas, la circuitería específicamente para un ASIC. Esta posibilidad permite al diseñador dejar de lado la facilidad de usar celdas probadas y pre caracterizadas para todo o parte del diseño. Esto es de provecho solo en caso de que las celdas lógicas existentes en las librerías no tengan propiedades deseables tales como la velocidad de cálculo o si la celda es muy grande y consume mucha energía; puede darse el caso de que simplemente ninguno de los diseños disponibles de las celdas de los archivos o librerías sirvan para el propósito deseado. Cada vez menos IC completamente configurables son diseñados puesto que existen problemas con ciertas partes especiales del ASIC que son muy difíciles de manejar.

Históricamente la tecnología bipolar ha sido utilizada para la precisión en funciones analógicas. La razón fundamental de ello es que en todos los circuitos integrados el apareamiento de las características entre los componentes de distintos IC es malo pero entre los componentes de un mismo IC es excelente. Para mejorar la diferencia entre ellos se procesan obleas de silicio por lotes donde se producen varios miles de IC al mismo tiempo con mínimas diferencias de apareamiento entre sí.

El apareamiento entre los transistores es crucial para la operación de un circuito. Para el diseño de IC se deben localizar pares de transistores uno junto al otro. La física del dispositivo dicta que un par de transistores bipolares podría

siempre aparearse mucho más que los transistores de tecnología CMOS del mismo tamaño.

La tecnología bipolar es empleada para el diseño de los ASIC completamente configurables analógicos porque proporciona mejor precisión. Aunque la realidad es otra, a pesar de las malas propiedades del uso de la tecnología CMOS para la electrónica analógica su empleo se ha incrementado, por dos razones:

Es la tecnología más difundida en el mercado para fabricar IC, por lo que muchos ASIC están construidos con esta tecnología.

Permite mucho mayor nivel de integración ya que se requieren funciones analógicas y digitales dentro de un mismo IC.

Por este motivo lo diseñadores de ASIC han encontrado maneras de implementar funciones analógicas utilizando tecnología CMOS con técnicas que aprovechan la exactitud de los diseños analógicos bipolares (BiCMOS).

Dispositivos lógicos programables (PLD)

Son IC estándar de la familia de los ASIC que están disponibles en configuraciones estándar desde catálogos de partes y se venden en grandes volúmenes a muchos consumidores. Sin embargo, los PLD pueden configurarse o programarse para crear partes configurables para una aplicación específica, los PLD utilizan diferentes tecnologías para permitir la programación del dispositivo. Entre las principales características de los PLD se puede destacar:

- No poseen máscaras, capas, ni celdas lógicas configurables.
- Rápido diseño.
- Ocupa un solo gran bloque de interconexiones programables.
- Poseen una matriz de macro–celdas lógicas que usualmente consiste de un arreglo programable lógico seguido por un Flip–Flop o Latch.

Arreglo de compuertas programables en campo (FPGA)

Cuando la complejidad del arreglo de compuertas programable se incrementa entonces a este ASIC se le denomina FPGA, y esta es la única diferencia entre los PLD y el GA. De hecho algunas de las compañías que

fabrican FPGA denominan a sus productos PLD complejos. El FPGA es uno de los más nuevos miembros de la familia de los ASIC, su importancia creció rápidamente reemplazando al uso de la familia TTL en los sistemas de microelectrónica. Algunas de las características de este tipo de ASIC son:

- Ninguna de las máscaras son configurables por el consumidor.
- Se emplea un método para programar las interconexiones y las celdas lógicas básicas.
- El núcleo es un arreglo regular de celdas lógicas básicas que puede implementarse como una lógica secuencial a base de Flip–Flops.
- Una matriz de interconexiones programables rodea a las celdas lógicas.
- Las celdas programables rodean al núcleo del dispositivo.
- El diseño solo tarda algunas horas.

Así pues, un circuito integrado dedicado a realizar tareas específicas suele responder a las necesidades reales de manera óptima. Muchos diseñadores implementan los FLCs en los FPGA por ser una tecnología que ofrece muchas ventajas bastante obvias respecto a las antes descritas. Actualmente, los FPGA más recientes pueden alojar grandes diseños con una respuesta en tiempo que jamás se pensó que se lograría alcanzar en los principios de la tecnología ASIC. Muchas empresas utilizan los FPGA como el móvil para diseñar circuitos dedicados que son propuestos para formar parte de un sistema computacional más grande, como lo han hecho las empresas Intel y AMD con la integración de algunas unidades funcionales en los nuevos procesadores, por ejemplo.

Los beneficios de un FPGA

Un FPGA, Arreglo de Compuertas Programables en Campo, es un dispositivo semiconductor de la familia de los ASIC que contiene *"bloques lógicos"* cuya interconexión y funcionalidad se pueden programar [38]. Se dice que un dispositivo electrónico o sistema integrado (embebido) es *"programable en campo"* si su *firmware* (almacenado en memoria no volátil, como una ROM) puede ser modificado *en el lugar o sitio donde se ocupe* sin desensamblar el dispositivo o regresarlo a su manufacturero para alguna modificación [39]. Una jerarquía de interconexiones programables permite a los bloques lógicos de un FPGA ser interconectados según la necesidad del diseñador del sistema. Estos bloques lógicos e interconexiones pueden ser programados después del proceso de

manufactura por el diseñador, así pues un FPGA puede desempeñar cualquier función lógica necesaria. Históricamente, los FPGAs surgen como una evolución de los PLDs y fue la compañía llamada Xilinx quien lanzó por primera vez al mercado en 1985 el primer FPGA de 1000 compuertas equivalentes, el XC2064, en manos de sus fundadores Ross Freeman, Bernie Vonderschmitt y Jim Barnett [44].

El uso de los FPGA en las industrias se debe al hecho de que el FPGA combina lo mejor de los ASICs y de los sistemas basados en procesador. Un FPGA proporciona fiabilidad y rentabilidad a diferencia del diseño personalizado ASIC, sin alcanzar la velocidad de un ASIC; sin embargo, la relación entre el costo y el beneficio es favorable y es por eso que conviene diseñar sistemas digitales en un FPGA. También presenta cierta flexibilidad comparado con los diseños de software que son ejecutados en un procesador, pues no está limitado al número de núcleos de procesamiento disponibles. A diferencia de los procesadores, los FPGAs son completamente paralelos por naturaleza, de tal manera que diferentes operaciones no tienen que competir por los mismos recursos. Cada tarea de procesamiento independiente está asignada a una sección dedicada en el chip y puede funcionar de manera autónoma sin alguna influencia de otros bloques lógicos. Como resultado, el rendimiento de una parte de la aplicación no se ve afectado cuando se agregan más procesamientos.

Los FPGA presentan, en general, cinco grandes beneficios [36]:

1. *Rendimiento.* Aprovechando el paralelismo, los FPGAs exceden la potencia de cómputo de los *Procesadores Digitales de Señales* (DSPs) rompiendo el paradigma de la ejecución secuencial y logrando realizar más procedimientos por ciclo.
2. *Tiempo.* La tecnología FPGA ofrece gran flexibilidad y capacidades de realizar prototipos rápidos para enfrentar las necesidades y demandas del mercado. Es decir, que esta tecnología te permite probar una idea o concepto y verificarlo en hardware sin llevarlo por el largo proceso de fabricación de diseños ASIC personalizados. Con esto es posible realizar cambios al diseño y asegurar su funcionamiento en algunas horas en vez de semanas. Otro tipo de hardware puede conectarse a algunas terminales del chip FPGA, ya sea para configuración, para pruebas con otros dispositivos o simplemente para aprendizaje.
3. *Costos.* La naturaleza del silicón programable conlleva a eliminar los costos de fabricación o los largos tiempos de ensamble. Si hay alguna necesidad de modificar el diseño, los costos para realizarlos son realmente insignificantes comparados con los que implica un ASIC.
4. *Fiabilidad.* Mientras que las herramientas de software proporcionen el ambiente de programación adecuado, la circuitería del FPGA es realmente

una implementación en “duro” de la ejecución de un programa. Es decir que a cualquier diseño que se implemente en un FPGA se le puede tener la confianza de que operará siempre de la misma manera como el programa indica.

5. *Mantenimiento a largo plazo.* Como se dijo antes, los chips FPGA son modificables en campo y no requieren el gasto en tiempo y dinero involucrados con el rediseño ASIC. Algunos protocolos de comunicación, por ejemplo, tienen ciertas especificaciones que pueden cambiar con el tiempo; así pues una interfaz basada en ASIC no permite el mantenimiento del diseño y crea ciertos retos de compatibilidad. Los FPGA son capaces de mantenerse abiertos a cualquier modificación futura que pueda ser necesaria. Por lo que, pueden realizarse mejoras funcionales sin gastar tiempo en el rediseño del hardware.

Antecedentes 2

Los controladores difusos existentes

En esta sección, se hace un resumen del estado de los FLCs implementados en un sistema digital reprogramable, como es el uso de los FPGA, hasta el momento. También, se habla brevemente de algunos FLCs que fueron las primeras implementaciones en hardware y dieron cabida a su posterior implementación en FPGA.

Algunos de los primeros FLCs implementados en hardware están enlistados al principio de la sección llamada *Referencias*, entre los que destacan los diseños VLSI de Togai y Watanabe [1], también una implementación con electrónica analógica y electrónica digital en un mismo chip realizada por Miki y Yamakawa [2] y también la implementación segmentada de Watanabe, Dettloff y Yount [3]. Está claro que la efectividad de un FLC es la velocidad de procesamiento, es decir, la velocidad con la que se propaga una señal en una trayectoria en un sistema digital; este concepto está ligado al término *Retardo de Propagación* [35], que es el período de tiempo que inicia cuando un dato estable y válido pasa a través de una entrada a un sistema y termina cuando en la salida de ese sistema el resultado de la acción en la entrada es también un dato estable y válido. Las primeras implementaciones, como era de esperarse, no realizaban grandes cantidades de procesamiento debido a las limitaciones tecnológicas existentes entre los años 1986 y 1992.

Aunque el FPGA en este período ya había sido inventado por Xilinx, [36] y [37], fue a partir de 1992, que las implementaciones VLSI de FLCs quedaron atrás debido al uso de los FPGA, entre los que destacan [4], [5], [10], [23], [26] y [27]. Debido a la gran flexibilidad de los FPGA, los FLCs comienzan a obtener velocidades mayores al millón de inferencias lógico difusas por segundo (1 MFLIPS), esto debido al uso de técnicas computacionales como las *Look–Up Tables* (LUT) o Tablas de Mapeo [32], que guardan información pre–procesada en memorias internas o externas; además, comienzan a aplicarse algunas técnicas que se utilizan en las arquitecturas de computadoras de esa época, como es la Segmentación [33] y el Paralelismo [34]. Muchas de las arquitecturas de la época combinaron el procesamiento serie con el paralelismo, aunque todas estas implementaciones son demasiado complejas y algunas limitadas dependiendo del dispositivo FPGA y de la técnica empleados para su implementación.

En general, los FLCs implementados en FPGA hasta el momento se pueden dividir en dos enfoques [32]:

1. La Computación en Tiempo de Ejecución o RTC (Runtime Computation)
2. La Computación por Búsqueda o LUC (Lookup Computation)

La ingeniería de hardware hereda estos métodos de la ingeniería de software.

La RTC realiza el procesamiento de la información en tiempo de ejecución (en línea), llevando a cabo el cálculo del proceso en el momento en el que en alguna de sus entradas existe algún cambio; lo cual conlleva a un consumo de tiempo relativamente alto para obtener un resultado en alguna de sus salidas. La LUC reduce su procesamiento a simples búsquedas en algún dispositivo de almacenamiento como puede ser una memoria; esto ahorra tiempo de procesamiento elevando la velocidad del sistema en general.

Ambos enfoques tienen ventajas uno sobre el otro. La LUC eleva el tiempo de procesamiento porque todo el cálculo se reduce a accesos a memoria por medio de una LUT; la RTC toma más tiempo en obtener un resultado concreto porque tiene que realizar nuevos cálculos para obtener un nuevo resultado. La LUC, por tener todos los resultados posibles precalculados no puede realizar de manera práctica cambios en los parámetros para realizar un nuevo cálculo, por lo que tiene que auxiliarse de un sistema supervisorio o inteligente para poder modificar y recalcular los nuevos valores que contendrá la misma; la RTC se limita sólo a pequeños cambios de valores en algunos registros para realizar un nuevo cálculo y obtener un resultado correcto.

El estudio realizado a varias arquitecturas de FLCs antiguos y recientes ha hecho posible establecer ciertas categorías y a su vez ha permitido sustentar el desarrollo de este proyecto y de este trabajo. La tecnología de hoy en día beneficia a las implementaciones surgidas de las ideas de los investigadores en control difuso en el mundo, incluso a aquéllas que computacionalmente eran imposibles para su implementación, con lo que diseños más simples y comparablemente rápidos pueden realizarse en FPGA.

La siguiente tabla muestra las categorías de los artículos revisados, enlistados en las *Referencias*, de algunos investigadores, en la que cada referencia puede estar caracterizada en varias categorías a la vez. La mayoría de los FLCs analizados manejan resoluciones fijas desde 3 bits hasta 8 bits, lo que suele ser bajo pero se justifica debido al gran costo computacional que este implica. Todos tienen sincronía con reloj, en cualquiera de sus tipos de procesamientos, ya sea usando paralelismo, procesamiento serie, la combinación de ambos o la segmentación. El tipo de procesamiento más utilizado es el paralelo y el siguiente es la segmentación. El número de entradas y salidas, número y formas de las funciones de membresía son en general fijas. Aquéllos trabajos en los que hacen énfasis en el número máximo de funciones de membresía traslapadas es 2, realizan

reducción de reglas debido a que en cada ciclo de reloj ejecutan sólo una regla activa, el resto mantiene una estructura de árbol de módulos MAX–MIN. El método de desdifusificación más usado es el de Centroide o COG y la máquina de inferencia más utilizada es la Mamdani. La forma de implementación del difusificador y del desdifusificador se deriva en los 2 enfoques antes descritos (la RTC y la LUC), así como la combinación de ambos, no muy común, catalogada como *Distribuido*, porque distribuye en todo el sistema módulos que contienen estructuras basadas en estos dos enfoques, lo cual fue desarrollado por Kim en [13] y [25]. Finalmente, el tipo algoritmo de la división utilizado depende de si se hace uso del enfoque RTC, por lo que para el enfoque LUC el algoritmo de la división resulta irrelevante.

Tabla 1. Clasificación de FLCs existentes implementados en hardware (FPGA).

Resolución	**General**	**Específico**			**Indefinido**
	[2], [10]	[1], [3], [4], [5], [6], [7], [8], [9], [11], [12], [13], [14], [16], [17], [18], [19], [22], [25], [26]			[15], [20], [21], [23], [24], [27]
Electrónica	**Analógica**	**Digital**			
	[2]	[1], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27]			
Tecnología usada	**VLSI**	**FPGA**			
	[1], [2], [3]	[4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27]			
Sincronía con reloj	**No**	**Sí**			
	[2]	[1], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27]			
Tipo de procesamiento	**Indefinido**	**Serie–Paralelo**	**Segmentación**	**Paralelo**	
	[21]	[1], [8]	[3], [5], [7], [11], [12], [15]	[2], [4], [6], [9], [10], [13], [14], [16], [17], [18], [19], [20], [22], [23], [24], [25], [26], [27]	
Número de entradas/salidas	**General**	**Específico**			
	[1], [2], [10], [26]	[3], [4], [5], [6], [7], [8], [9], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [27]			
Número de funciones de membresía por entrada/salida	**Indefinido**	**General**	**Específico**		
	[15], [16]	[1], [2], [3], [10], [12], [25], [26], [27]	[4], [5], [6], [7], [8], [9], [11], [13], [14], [17], [18], [19], [20], [21], [22], [23], [24]		
Formas de funciones de membresía usadas	**Indefinido**	**General**	**Específico**		
	[1], [2], [3], [13], [16], [19]	[4], [5], [10], [12], [25], [26], [27]	[6], [7], [8], [9], [11], [14], [15], [17], [18], [20], [21], [22], [23], [24]		
Número máximo de funciones de membresía traslapadas	**2**			**Indefinido**	
	[4], [5], [6], [7], [11], [13], [14], [15], [17], [20], [22], [23], [24], [25]			[1], [2], [3], [8], [9], [10], [12], [16], [18], [19], [21], [26], [27]	
Número de reglas difusas válidas	**Indefinido**	**General**	**Específico**		
	[8], [15], [21], [27]	[1], [2], [3], [10], [12], [25], [26]	[4], [5], [6], [7], [9], [11], [13], [14], [16], [17], [18], [19], [20], [22], [23], [24]		
Método de desdifusificación	**Indefinido**	**MOM**	**COG**		
	[1]	[12], [13]	[2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27]		
Tipo de máquina de inferencia	**TSK**	**Indefinido**	**Mamdani**		
	[9], [11], [24]	[8], [19], [20], [21]	[1], [2], [3], [4], [5], [6], [7], [10], [12], [13], [14], [15], [16], [17], [18], [22], [23], [25], [26], [27]		
Implementación del difusificador	**Distribuido**	**Indefinido**	**RTC**	**LUC**	
	[13], [25]	[1], [18], [19], [20], [21]	[9], [11], [14], [15], [22], [23], [24]	[2], [3], [4], [5], [6], [7], [8], [9], [10], [12], [14], [15], [16], [17], [25], [26], [27]	
Implementación de la máquina de inferencia	**Distribuido**	**Indefinido**	**RTC**	**LUC**	
	[13], [25]	[9], [19], [20], [21], [23]	[4], [5], [10], [11], [14], [22], [26], [27]	[1], [2], [3], [6], [7], [8], [12], [15], [16], [17], [18], [24]	
Implementación del desdifusificador	**Distribuido**	**Indefinido**	**RTC**	**LUC**	
	[13], [25]	[1], [9], [10], [20], [21], [23], [26], [27]	[3], [6], [7], [8], [11], [15], [16], [17], [18], [22], [24]	[4], [5], [11], [12], [14], [19]	
Algoritmo de división usado	**Restas sucesivas**	**Con restauración**	**Sin restauración**	**Indefinido**	**No usado**
	[8], [12]	[6], [24]	[3], [7], [15]	[9], [16], [18], [20], [21], [25]	[1], [4], [5], [10], [11], [13], [14], [17], [19], [22], [23], [26], [27]

Los FLCs antes mencionados tienen un alto nivel de complejidad y la posibilidad de expandir su rendimiento resulta una tarea bastante difícil. También, la tendencia de los controladores difusos muestra que la generalidad de un sistema puede disminuirse al buscar velocidad. Es por esto que, si se necesitaran cambiar algunos parámetros del controlador como el número, la forma de las funciones de membresía o la resolución, ninguno de estos controladores sería capaz de responder a tales expectativas. Por lo tanto deben buscarse ciertas alternativas de diseño que permitan equilibrar la generalidad con la rapidez en un sistema difuso. Sin embargo, ningún diseño presentado cuenta con un equilibrio destacado. La fuerte dependencia de la sincronía con reloj de un sistema digital secuencial limita la facilidad de expandir el sistema para mejorar su rendimiento.

De la Tabla 1 se han extraído sólo aquéllos artículos que reportan la velocidad en unidades MFLIPS y recopilado en la Tabla 2; todos ellos están ordenados de acuerdo a la velocidad. Cabe destacar que es importante observar el año de la publicación y el dispositivo FPGA utilizado. Como comparativo y para darles su importancia a los pioneros, se han agregado a la Tabla 2 las primeras implementaciones en hardware, reconocidas por los diseñadores de FLCs en todo el mundo.

Tabla 2. Comparación de implementaciones de FLCs en hardware basada en la velocidad.

MFLIPS:	Dispositivo utilizado:	Referencia:	Año de publicación:	Autor:
50.00	Indefinido Xilinx XC3020 Xilinx XC3000	[27] [26] [10]	1991 1992 1995	MANZOUL *et al.*
50.00	Indefinido	[20]	2000	LUND *et al.*
15.38	Xilinx XC3S1500	[11]	2005	DELIPARASCHOS *et al.*
10.00	Xilinx XC3042 y XC3064	[25]	1997	KIM *et al.*
9.00	Xilinx XC4008	[5]	1994	HUNG
8.00	Altera EPF8820	[12]	1997	ARANGUREN *et al.*
5.56	Xilinx XC3S200	[14]	2007	GONZÁLEZ *et al.*
3.13	Xilinx XC4013	[17]	2000	KIM
3.13	Xilinx XC3S1000	[15]	2007	SÁNCHEZ *et al.*
2.00	Xilinx XCV600E	[16]	2003	GAONA *et al.*
1.74	Xilinx XC4005	[4]	1993	HUNG *et al.*
1.41	Xilinx XC2S300E	[6]	2005	JOY *et al.*
0.92	Xilinx XC4010E	[18]	2000	RAMOS *et al.*
0.60	VLSI	[2]	1995	MIKI *et al.*
0.58	VLSI	[3]	1990	WATANABE *et al.*
0.53	Altera EPF81188	[13]	1996	KIM *et al.*
0.08	VLSI	[1]	1986	TOGAI *et al.*

Estos son algunos de los diseños de FLC más importantes, estudiados en este trabajo. Existen muchas implementaciones, pero la mayoría de ellas no muestran los resultados de velocidad, ni los recursos utilizados en el FPGA. Algunos de ellos no muestran ni explican de manera detallada el funcionamiento de su diseño, por lo que suelen existir inconvenientes a la hora de realizar comparaciones justas entre los FLCs de otros diseñadores. Cabe destacar que todos estos diseños están sincronizados con reloj.

La implementación del FLC de MANZOUL *et al*. [26, 27 y 10]. Es una de las que promete ser más rápidas. Su diseño consiste en la generación de ecuaciones booleanas a partir de un programa alterno que le permite realizar reducción de términos, que a su vez, le permite implementarlo en las LUTs de un FPGA. Su diseño exhibe una velocidad muy alta, sin embargo esto no va de acuerdo con la tecnología existente en los años de su publicación, ya que otros diseñadores de su época utilizan dispositivos FPGA con mucha más capacidad. Otra implementación que promete sobrepasar las 50 MFLIPS es la de LUND et al. [20]. Su diseño está basado en uno anterior en donde utiliza una PAL y hardware adicional para realizar las etapas del FLC. Los resultados de su diseño no son expuestos, ni se especifica el tipo de dispositivo utilizado.

Los resultados de DELIPARASCHOS *et al.* [11] son mucho más convincentes, para la tecnología de su época, al prometer 15.38 MFLIPS, debido a que sí especifica el dispositivo utilizado y muestra algunos resultados de los recursos utilizados en el FPGA. Su diseño segmentado (con 13 etapas) y haciendo uso de una máquina de inferencia TSK, le permiten obtener excelentes resultados. KIM *et al.* [25] prometen 10 MFLIPS utilizando dos FPGA para que cada uno realice diferentes etapas del FLC. Aunque los dispositivos que utilizan son lentos, la combinación de sus elementos de procesamiento mediante una red llamada KAFA, les permite alcanzar estos resultados. El resto de los diseños no sobre pasan los 10 MFLIPS, pero describen arquitecturas interesantes que pueden tomarse en cuenta para implementarlas o para mejorar el diseño de alguna arquitectura diseñada en el FPGA.

Todas las implementaciones VLSI mostradas al final de la Tabla 2, son las primeras implementaciones que existieron a partir de 1986, poco después de la invención del FPGA. A partir de ellas se ha popularizado el uso de FPGA para realizar diseños de FLCs e implementarlos para aplicaciones en tiempo real, ya que su rapidez es competitiva, comparado con muchos diseños ASIC.

Justificación

Este proyecto surge de la idea de crear un FLC de fácil diseño para su implementación con circuitos combinatorios en un FPGA, ya que en la actualidad no existe una metodología sencilla de diseño ni alguna arquitectura combinatoria fácilmente escalable que compita con el rendimiento de los FLC actuales.

Basado en el análisis de los antecedentes, es necesario resaltar ciertos puntos que justifiquen la elaboración de este proyecto.

- El uso de sincronía por reloj en todas las arquitecturas antes estudiadas, sobre todo en aquéllas que utilizan la segmentación, imposibilita el diseño y el rediseño práctico de FLCs, debido a las dependencias con las unidades de control y máquinas de estados que poseen.
- No existe un FLC completamente diseñado con lógica combinatoria, debido a que cuando se implementó por primera vez en FPGA, el diseño combinatorio resultaba bastante costoso en cuanto a recursos (hardware), sobre todo si se quería llevar a cabo la RTC. Actualmente la tecnología en FPGA permite la elaboración de diseños más costosos, debido a que cuentan con recursos muy amplios y velocidades altas. Con lo cual, la elaboración del presente proyecto representa una mejora en la forma de construir sistemas difusos, pues al no depender de la sincronía por reloj permite que el sistema pueda expandirse hasta los límites del dispositivo FPGA utilizado. Con esto, diseños más simples y bastante eficientes son posibles haciendo uso del paralelismo.
- La idea del presente trabajo es demostrar que el uso de sistemas digitales combinatorios para el diseño de FLCs representa una opción bastante práctica para la implementación de sistemas difusos que tienen rendimiento comparable con los ya existentes, beneficiándose de la tecnología actual. Así pues, un FLC totalmente combinatorio puede utilizarse en sistemas de tiempo real.
- Debido a la necesidad de realizar prototipos rápidos, característico en los diseños sobre FPGA, esta arquitectura modular diseñada con lógica combinatoria permite crear prototipos FLC rápidos, adecuándolos de manera sencilla a cada sistema que se requiera controlar (planta), mediante una metodología de diseño. Además, no existe una metodología clara y sencilla que permita al usuario crear FLCs para su propia aplicación específica, con la cual se ahorraría tiempo de diseño.

Diseño 3

Metodología de diseño

Las siguientes secciones de este capítulo describen la implementación de cada una de las etapas que se necesitan para realizar el control difuso. Como se dijo anteriormente, el diseño inicialmente se realiza con ayuda de MATLAB&SIMULINK y su caja de herramientas llamada *Fuzzy Logic Toolbox*, pero toda la implementación se realiza en VHDL; cabe aclarar que el código VHDL no está detallado en este trabajo, por lo que sólo están disponibles para consulta en el disco compacto que viene adjunto.

El propósito de este capítulo es mostrar una metodología de diseño sencilla y rápida para construir FLCs eficientes, cuyas ocho etapas, permiten que fácilmente puedan ser llevados a la implementación en un FPGA. Independientemente de la metodología, los aspectos más importantes que tienen que tomarse en cuenta para el diseño de un FLC en un FPGA son los siguientes:

- Aplicación a la que se destina.
- La capacidad de procesamiento, generalmente expresada en MFLIPS.
- La cantidad de recursos utilizados para tal propósito.

Las ventajas del diseño de hardware, haciendo uso de los HDLs, permiten llevar cualquier sistema a su elaboración en poco tiempo. A continuación se presentan las etapas requeridas para el diseño de cualquier FLC, llevando de la mano al lector a través de un caso de estudio, donde se diseña un FLC para llevar a cabo el control de un servomotor de corriente directa. Todos estos pasos están distribuidos en todo este capítulo.

1. Establecer lo que se quiere controlar y a partir de qué parámetros.
2. Definir el número de entradas y salidas del FLC basándose en el paso anterior.
3. Definir el número de funciones de membresía o conjuntos difusos por cada entrada y salida, basándose en el paso anterior. Además, definir la forma de los mismos basándose en las características del proceso y en el intervalo de operación del FLC (universo de discurso).

4. Establecer la configuración del FLC mediante las reglas de inferencia difusa de acuerdo a la forma deseada de operación y basado en el conocimiento que tiene el *experto* acerca del proceso.
5. Construir el difusificador a partir de funciones de membresía sencillas (trapezoidal, triangular, S, Z, etc.).
6. Construir la máquina de inferencia basándose en la etapa 4, mediante estructuras de *árbol invertido* de módulos MIN y módulos MAX.
7. Construir el desdifusificador mediante módulos de división y multiplicación bajo la filosofía del paralelismo.
8. Implementar el diseño en FPGA.

ETAPA 1. El sistema puede ser tan grande como la cantidad de recursos disponibles en el FPGA. Así que podemos hablar de un FLC que contenga un número cualquiera de entradas y un número cualquiera de salidas, pero estaríamos exagerando pues así se use el dispositivo más grande que haya en el mundo no se conseguiría implementar un sistema tan generalizado. Así pues, para comenzar, es necesario saber qué se pretende controlar y basado en qué, para luego establecer un número de entradas y un número de salidas concretos. Como se hace referencia en el Capítulo 1, en la sección llamada Control Difuso, existen sistemas difusos MIMO y MISO, que son los más comunes y los sistemas SISO y SIMO que no son frecuentes pero existen. Además, la ecuación para sistemas MIMO (2.1) puede representarse en varios sistemas difusos MISO como se muestra en las ecuaciones (2.2) y (2.3). El diseño de sistemas difusos se reduce a los sistemas MISO, ya que resulta conveniente para cada salida de un sistema MIMO diseñar un controlador MISO [28].

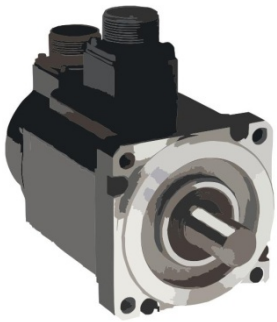

Figura 1 Servomotor de corriente directa Sure–Servo SVL210.

ETAPA 2. Suponga un sistema de control difuso MISO para un servomotor de corriente directa que posee un codificador de posición como el de la Figura 13. El sistema posee 2 entradas $n = 2$ y una salida $m = 1$, en donde u_1 representa la variable de entrada del error que existe entre la posición actual del rotor y la posición deseada del mismo y u_2 representa el cambio en este error; finalmente la variable de salida y_1

representa el voltaje que debe aplicársele al servomotor para conseguir la posición deseada [45]. Renombremos a partir de este momento a la primera variable de entrada u_1 como el "error de posición," denotándola como *eP*. Y a la otra variable de entrada u_2 como el "cambio en el error de posición," denotándola como *cP*. Por último, la variable de salida y_1 es el "voltaje aplicado," denotado como *V*. Así pues el sistema basado en la Figura 7 queda reducido como se muestra en la Figura 14:

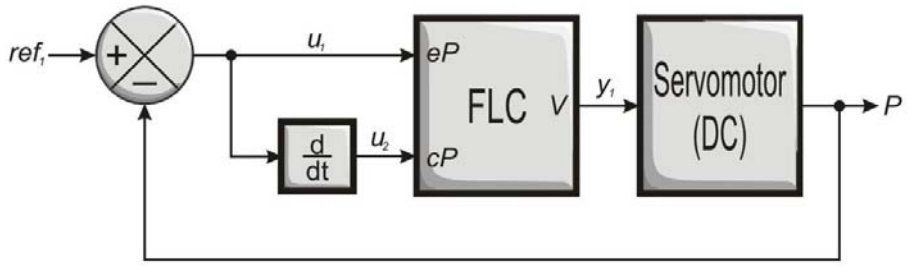

Figura 2. Sistema de control difuso para un servomotor de corriente directa.

En la Figura 14 puede apreciarse un sistema de control difuso de lazo cerrado gobernado por un FLC de dos entradas por una salida. Nótese que la segunda variable de entrada al controlador depende solamente de una de las variables características del sistema, que en este caso, es la medición de la posición actual en el rotor, y se obtiene a partir de la derivación del error de posición resultante del módulo restador característico del sistema de control de lazo cerrado. La salida del controlador es llevada a un circuito que sirve de interfaz entre el servomotor y el controlador (no especificado en el diagrama porque el propósito de este trabajo es sólo el diseño del FLC, pero se debe incluir dentro del módulo del proceso a controlar, en este caso el servomotor), el cual interpreta el voltaje de corriente directa que debe aplicarle a las terminales del servomotor (que sirve a su vez de actuador sobre el medio, que en este caso es el rotor) para mover al rotor a la posición deseada. El servomotor posee un codificador de posición que entrega mediciones continuas de la posición actual del rotor, el cual está representado con la letra *P*. De esta manera el lazo cerrado (de retroalimentación) se logra con el uso del restador donde llegan ambas medidas, tanto la de posición actual como de la posición deseada, que es la referencia (ref_1).

ETAPA 3. Antes de elegir el número de funciones de membresía (conjuntos difusos) y las etiquetas lingüísticas (variables lingüísticas) que tendrá el FLC es necesario establecer el tamaño del universo de discurso para cada una de las variables de entrada. Siendo el caso el sistema de control antes propuesto, para la variable de entrada *eP* (error de posición), cuya medición está basada en los valores que se obtienen de *P* (posición actual), los valores deben estar expresados en radianes (*rad*), pues el rotor puede girar a lo largo de 2π radianes y por lo tanto, los valores que expresan el error de posición pueden estar

entre $-\frac{\pi}{2}rad$ y $\frac{\pi}{2}rad$. Con esta configuración propuesta, el error máximo permitido puede estar por lo menos en $\frac{1}{4}$ de circunferencia de la posición de referencia o posición deseada en ambas direcciones. A su vez, para la variable *cP* (cambio en el error de posición) sus valores pueden estar entre $-\frac{\pi}{6}rad/s$ y $\frac{\pi}{6}rad/s$, lo que equivale a valores de pendientes entre $-30°/s$ y $30°/s$. Para ambos casos estos intervalos de trabajo para los universos de discurso de cada variable de entrada dependen del criterio del diseñador.

El número de funciones de membresía que debe tener cada variable de entrada al FLC depende de los recursos del sistema computacional en donde se vaya a implementar y también de la precisión que el diseñador requiera para el proceso que requiera controlar. Para el sistema propuesto, se requiere evaluar todo el sistema de control para obtener los requerimientos mínimos para su funcionamiento.

Para la entrada *eP*, se requiere que el error sea mínimo o nulo y que si es muy grande, tanto positiva como negativamente, el sistema sea capaz de reducirlo o anularlo completamente. Por lo tanto, el módulo restador obtiene el error de posición a partir de *ref* y de *P*.

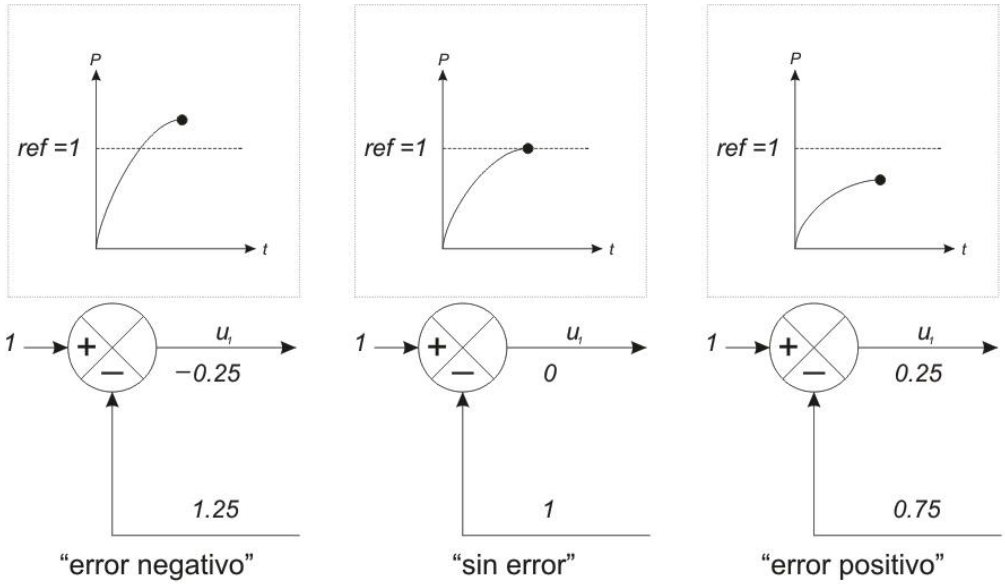

Figura 3. Asignación de etiquetas lingüísticas a partir de las características del error de posición.

De la Figura 15, surgen tres etiquetas lingüísticas (conjuntos): *Error Negativo* (NE), *Sin Error* (ZE) y *Error Positivo* (PE). Sin embargo, todas estas etiquetas lingüísticas anteriores como tales siguen siendo valores concretos y no difusos, esto debido a que cualquier valor *P* que esté por encima de *ref* siempre provocará un error negativo y sólo pertenecerá al conjunto "error negativo," lo mismo sucede para el conjunto "error positivo," quedando uno y sólo un valor para el conjunto "sin error" que es cuando *P* y *ref* son exactamente iguales. Debido a esto, es necesario convertir tales conjuntos tradicionales a conjuntos difusos dentro del intervalo de trabajo, es decir, su universo de discurso. A este paso se le ha llamado *Difusificación*.

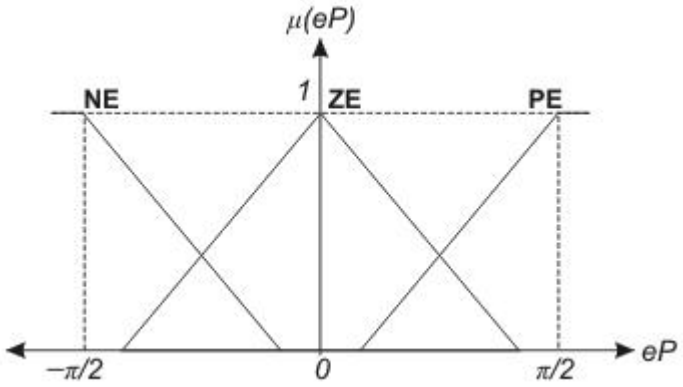

Figura 4. Conjuntos difusos para la variable "error de posición."

La Figura 16 muestra una distribución uniforme de tres conjuntos difusos dentro del universo de discurso para la variable "error de posición." Entonces, para cada valor (en radianes) de la variable *eP*, existe por lo menos un valor de membresía y hasta dos como máximo, es decir que la pertenencia no se limita a un solo conjunto.

La entrada *cP*, se puede traducir como "la pendiente" o la "velocidad de cambio del error de posición" que describe el proceso al aplicársele la acción de control en un instante dado. La Figura 14 muestra que *cP* es la *derivada* del error de posición con respecto al tiempo, lo cual representa una velocidad de error. Para esta variable, de igual manera se requiere que el cambio en el error antes descrito sea mínimo o nulo, es decir que entre más cerca esté de cero, existe la posibilidad de que el proceso esté controlado para la acción de control que se le esté aplicando en ese instante.

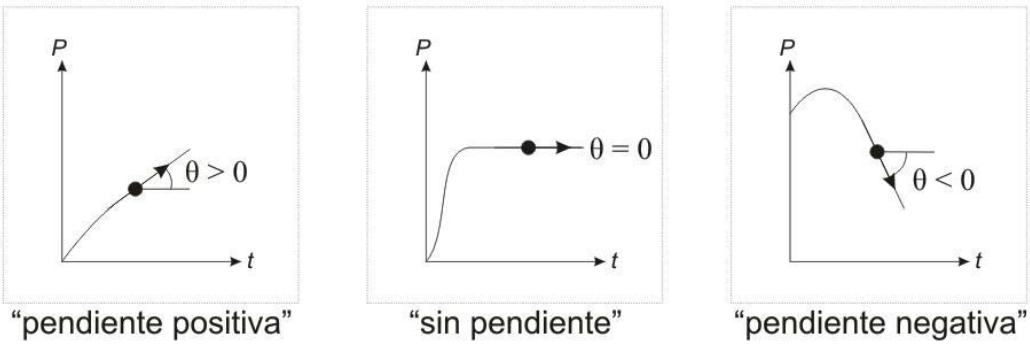

Figura 5. Asignación de etiquetas lingüísticas a partir de las características del cambio en el error de posición.

De la Figura 17, surgen tres etiquetas lingüísticas (conjuntos): Pendiente Negativa (NC), Sin Pendiente (ZC) y Pendiente Positiva (PC). De la misma manera que para la variable *eP*, es necesario convertir estos conjuntos tradicionales a conjuntos difusos por medio de la difusificación. Derivado de lo anterior, los conjuntos difusos resultantes se pueden apreciar en la Figura 18:

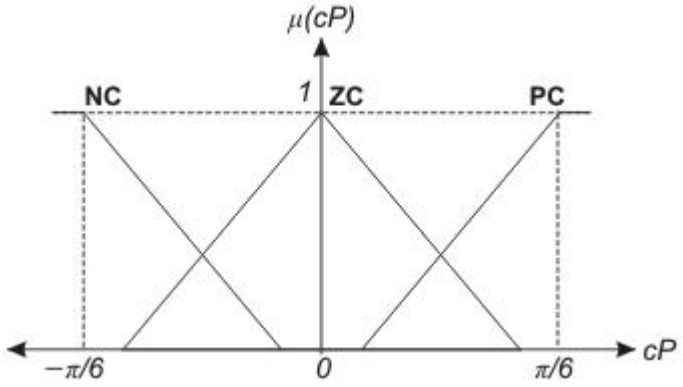

Figura 6. Conjuntos difusos para la variable "cambio en el error de posición."

Finalmente la salida *V* es el voltaje aplicado al servomotor. Los servomotores de corriente directa pueden mover su rotor en ambas direcciones y mantenerlo en una posición de manera estable, dependiendo de la polaridad del voltaje aplicado a sus terminales [46]. Supóngase un servomotor cuya alimentación acepta voltajes entre –5 y 5 volts, entonces existen mínimo tres acciones de control posibles, que a su vez representan las etiquetas lingüísticas: Voltaje Negativo (NV), Sin Voltaje (ZV) y Voltaje Positivo (PV). Haciendo el mismo procedimiento que se hizo con cada una de las variables de entrada, los conjuntos difusos para la variable V quedan como se muestran en la Figura 19:

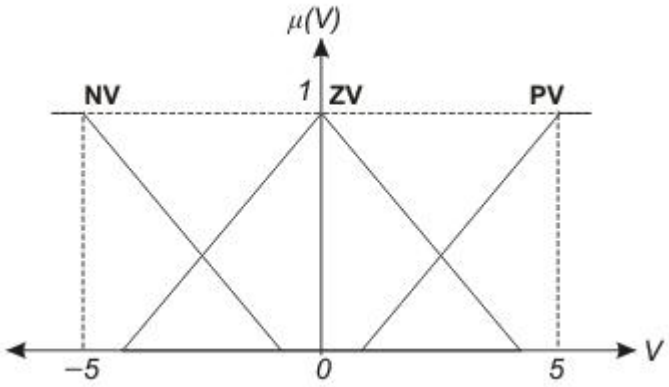

Figura 7. Conjuntos difusos para la variable "voltaje aplicado."

La distribución y la forma de las funciones de membresía a lo largo de los universos de discurso de cada una de las variables tanto de entrada como de salida están sujetas al criterio y al conocimiento acerca del proceso por parte del "experto," que en este caso es el diseñador.

El siguiente paso del diseñador es conocer el número de reglas con las que puede contar a partir de las características de la entrada. Según la ecuación (3), el número de entradas y de funciones de membresía por cada entrada afecta al número de reglas que puede haber en el FLC. Por lo que después de resolver la ecuación (3) y tomando en cuenta que cada entrada posee $N_i = 3$ funciones de membresía entonces existen 9 reglas que pueden estar activas en el FLC.

$$R = \prod_{i=1}^{3} N_i = N_1 \times N_2 = (3) \times (3) = 9$$

ETAPA 4. Basándose en el conocimiento que el experto tiene acerca del proceso, se establecen las reglas con las que el controlador lo opera y controla. Las reglas son finalmente la configuración del controlador y las que permiten inferir, por medio de la *Inferencia Difusa*, una salida a partir de las premisas, es decir, las entradas al controlador. Para elegir la configuración del FLC ideal para llevar a cabo el control es necesario realizar un análisis al proceso para cubrir todos los posibles valores que pueden ocurrir en las entradas del FLC y así asegurar que en su salida exista un valor concreto y exacto que promueva a una acción de control correcta.

Teniendo en cuenta lo anterior, tenemos nueve combinaciones entre las entradas y por lo tanto nueve casos distintos que analizar a continuación:

1. Cuando el error de posición es negativo y la pendiente es negativa,
2. Cuando el error de posición es negativo y la pendiente es nula,
3. Cuando el error de posición es negativo y la pendiente es positiva,
4. Cuando el error de posición es nulo y la pendiente es negativa,
5. Cuando el error de posición es nulo y la pendiente es nula,
6. Cuando el error de posición es nulo y la pendiente es positiva,
7. Cuando el error de posición es positivo y la pendiente es negativa,
8. Cuando el error de posición es positivo y la pendiente es nula y
9. Cuando el error de posición es positivo y la pendiente es positiva.

Ahora bien, todos estos casos tienen una representación gráfica que ilustra cada una de las circunstancias que pueden ocurrir para este controlador, mismas que se muestran en la Figura 20.

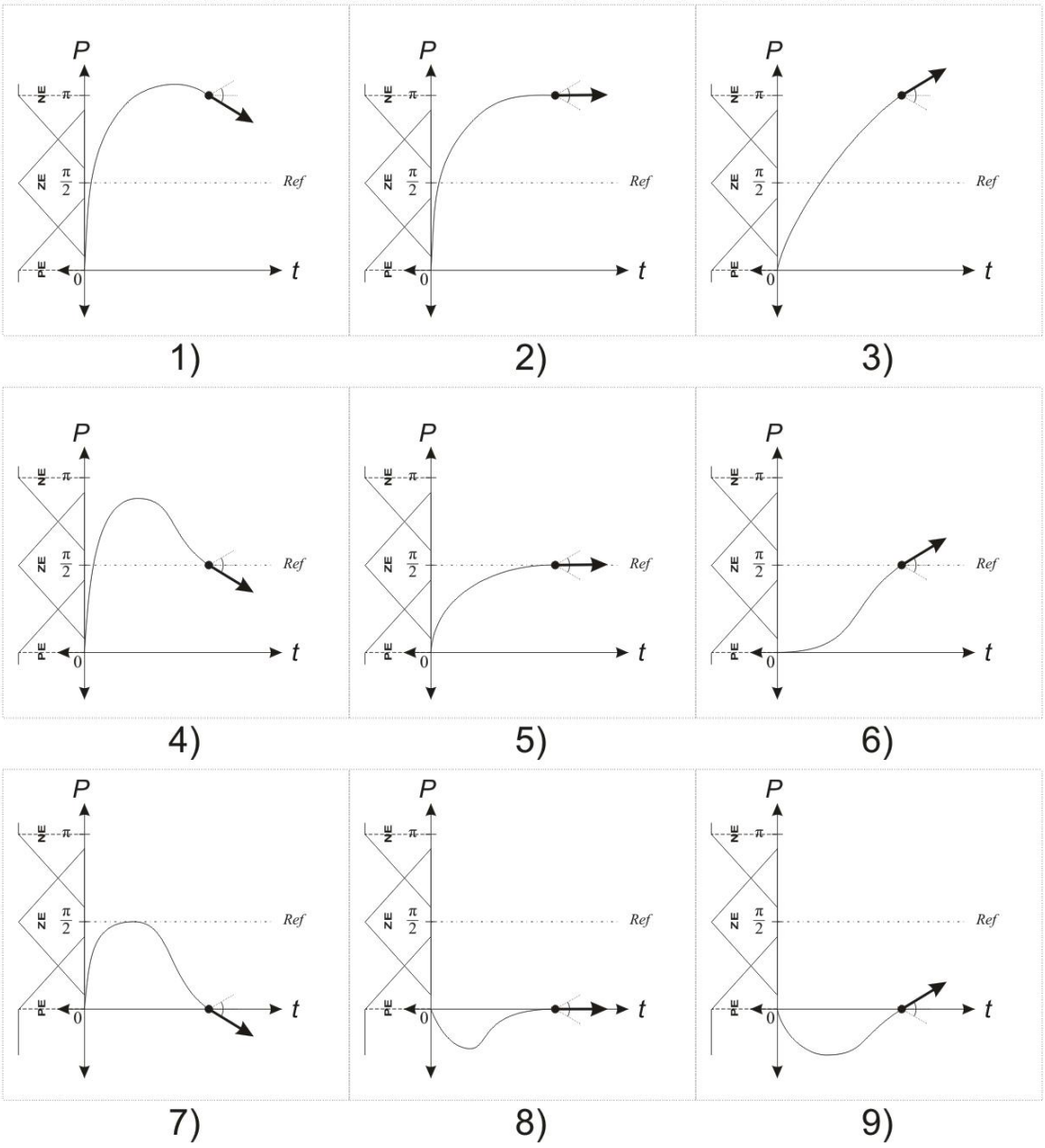

Figura 8. Representación gráfica de cada uno de los casos.

Como puede notarse en la Figura 20, el valor de referencia, es decir, la Posición Deseada *Ref*, se encuentra en $\frac{\pi}{2}$ y se han elegido para todos los casos valores de error de posición para los que en cada función de membresía no existe ningún traslape con el fin de lograr que ocurra cada caso a la vez. Puede decirse entonces que cada caso representa una regla de inferencia con la que el FLC decidirá lo que debe hacer en el momento en que

ocurra y para cada caso que ocurra se activarán las reglas pertinentes de acuerdo a los valores de las entradas implicadas. Cabe destacar, que por el hecho de que existan nueve reglas, no todas ellas tienen que estar activas forzosamente.

En cada uno de los gráficos de la Figura 20 están presentes las dos entradas del FLC. En el eje de las ordenadas *P*, se ha traslapado con el universo de discurso de la variable de entrada *eP*, a partir del centro del conjunto difuso ZE, coincidiendo con la referencia Ref. Asimismo, el universo de discurso de la variable *cP* es representado en cada gráfico con un intervalo de ángulo que va desde $-30°$ a $30°$ y el valor de la pendiente que describe la lectura de la Posición Actual es representado por una flecha de mayor grosor, que aparece en cada gráfico.

El diseñador, es decir el experto, decide cómo quiere que el FLC reaccione para cada uno de estos casos. Es por eso, que se necesita llegar a una expresión lingüística que describa de manera sencilla la acción de control que necesita el FLC para funcionar adecuadamente:

"Si la posición actual está por encima de la posición deseada entonces aplica un nivel de voltaje al motor que permita acercar al rotor a la posición deseada; si la posición actual está por debajo de la posición deseada entonces aplica un voltaje contrario al anterior que permita acercar al rotor a la posición deseada; y finalmente si la posición actual está justo o muy cerca de la posición deseada entonces deja de aplicar voltaje."

Como se ha dicho anteriormente, el control difuso es un método de aproximación que permite calcular heurísticamente la acción de control que debe aplicarse. Basado en esto y en la expresión anterior, las reglas de inferencia, que representan la base de conocimiento, según la ecuación (2) son las siguientes:

1. **IF** *eP* is NE **AND** *cP* is NC **THEN** *V* is NV
2. **IF** *eP* is NE **AND** *cP* is ZC **THEN** *V* is NV
3. **IF** *eP* is NE **AND** *cP* is PC **THEN** *V* is NV
4. **IF** *eP* is ZE **AND** *cP* is NC **THEN** *V* is NV
5. **IF** *eP* is ZE **AND** *cP* is ZC **THEN** *V* is ZV
6. **IF** *eP* is ZE **AND** *cP* is PC **THEN** *V* is PV
7. **IF** *eP* is PE **AND** *cP* is NC **THEN** *V* is PV
8. **IF** *eP* is PE **AND** *cP* is ZC **THEN** *V* is PV
9. **IF** *eP* is PE **AND** *cP* is PC **THEN** *V* is PV

Según la Figura 20, sólo una regla podría estar activa a la vez y podría observarse el funcionamiento del FLC.

Las reglas 1 a 4 provocan que el FLC aplique al motor un voltaje negativo; las reglas 6 a 9 provocan que el FLC aplique al motor un voltaje positivo; finalmente sólo la regla 5

no aplica voltaje al motor, puesto que en este punto se podría estar cerca de la posición deseada.

La precisión de un FLC depende directamente del número de funciones de membresía que existan en el universo de discurso de cada una de sus entradas y de la forma en que el experto las distribuya. Esto debido a que el número de reglas posibles en el FLC es la combinación del número de funciones de membresía de cada una de sus entradas, ecuación (3).

Para verificar el funcionamiento del FLC propuesto, es necesario realizar una simulación mediante el una herramienta muy útil llamada Fuzzy Toolbox que viene incluido en las herramientas de MATLAB, la cual permite construir y simular FLCs de manera fácil y rápida, y realizar una simulación final del FLC en SIMULINK, que es un ambiente gráfico que viene incluido en MATLAB también y permite simular cualquier sistema.

Todos los archivos utilizados para la realización de este trabajo se encuentran en el CD que viene adjunto, incluyendo aquéllos generados por MATLAB y sus herramientas. Por el momento, sólo se muestran las capturas de pantalla del FLC construido con el Fuzzy Toolbox y la simulación del Sistema de Control Difuso completo con SIMULINK, ya que en la sección de Resultados se hablará a detalle de su funcionamiento, los pasos a seguir y la comparación entre el sistema difuso generado con MATLAB, el cual es llamado Fuzzy Inference System, ó FIS, y el diseñado con VHDL, es decir, el FLC.

A continuación se habla de la arquitectura propuesta para un FLC, haciendo uso del enfoque RTC, su diseño mediante VHDL y la metodología para construir el FLC propuesto. Se habla del esquema de un FLC generalizado que contiene las siguientes etapas, frecuentemente mencionadas con anterioridad:

- El Difusificador,
- La Máquina de Inferencia Difusa y
- El Desdifusificador.

El difusificador

En esta sección se describe el diseño completo e implementación de un difusificador general usando el lenguaje de descripción de hardware llamado VHDL. Además, se diseña el FLC propuesto para el sistema de control difuso anteriormente descrito.

La sección de Controladores Difusos Existentes, en los Antecedentes, muestra la tendencia de diseño de FLCs mediante el enfoque LUC. Sin embargo, debido a que este enfoque muestra cierta dificultad cuando los parámetros del controlador cambian, especialmente cuando se requiere mover las funciones de membresía dentro de su universo de discurso, el diseño propuesto es elaborado mediante el enfoque RTC.

Esto conlleva a diseñar módulos que realicen los cálculos aritméticos necesarios para llevar a cabo la computación en tiempo de ejecución. Aunque en realidad, el término "en tiempo de ejecución" no es el mismo que se concibe para un programa en ingeniería de software, ya que la ejecución en hardware implica un tiempo de propagación a través de un circuito hasta obtener un resultado.

Debido a la simplicidad de una función de membresía trapezoidal simétrica, cuya ecuación se muestra a continuación, se pueden ahorrar bastantes recursos en un FPGA y por ende, es una de las funciones de membresía más convenientes para esta arquitectura.

$$y(x) = \begin{cases} \frac{\mu_{max}}{a}(x - c_1 + a), & c_1 - a < x < c_1 \\ -\frac{\mu_{max}}{a}(x - c_2 - a), & c_2 < x < c_2 + a \\ \mu_{max}, & c_1 < x < c_2 \\ 0, & de\ lo\ contrario \end{cases} \quad (16)$$

Donde a es la apertura del trapecio donde el valor de membresía decae, c_1 y c_2 son los extremos del trapecio donde el valor de membresía comienza y termina siendo μ_{max}, que es el valor de membresía máximo, que según [28] es uno.

La otra función de membresía práctica usada bastante en el diseño de FLC en hardware, es la función triangular simétrica, es decir, isósceles. Su ecuación es bastante parecida a la anterior:

$$y(x) = \begin{cases} \frac{\mu_{max}}{a}(x - c + a), & c - a < x < c \\ -\frac{\mu_{max}}{a}(x - c - a), & c < x < c + a \\ \mu_{max}, & x = c \\ 0, & de\ lo\ contrario \end{cases} \quad (17)$$

Donde a es la apertura del triángulo isósceles, y c es el centro del mismo. En este caso la ecuación se redujo puesto que c_1 y c_2 son exactamente los mismos, es decir c.

Así también, las funciones S y Z que son singularidades de un trapecio. Suelen usarse como las funciones de membresía que están en los extremos del universo de discurso de una variable de entrada.

La función S tiene la siguiente ecuación:

$$y(x) = \begin{cases} \frac{\mu_{max}}{a}(x - c + a), & c - a < x < c \\ \mu_{max}, & x \geq c \\ 0, & de\ lo\ contrario \end{cases} \quad (18)$$

Asimismo, la función Z tiene la siguiente ecuación:

$$y(x) = \begin{cases} -\frac{\mu_{max}}{a}(x - c - a), & c < x < c + a \\ \mu_{max}, & x \leq c \\ 0, & de\ lo\ contrario \end{cases} \quad (19)$$

Por lo tanto, estos módulos básicos son creados en VHDL, ya que por su simplicidad no consumen gran cantidad de hardware, aunque cabe mencionar que no son los únicos que pueden implementarse en FPGA. Su representación gráfica es la siguiente:

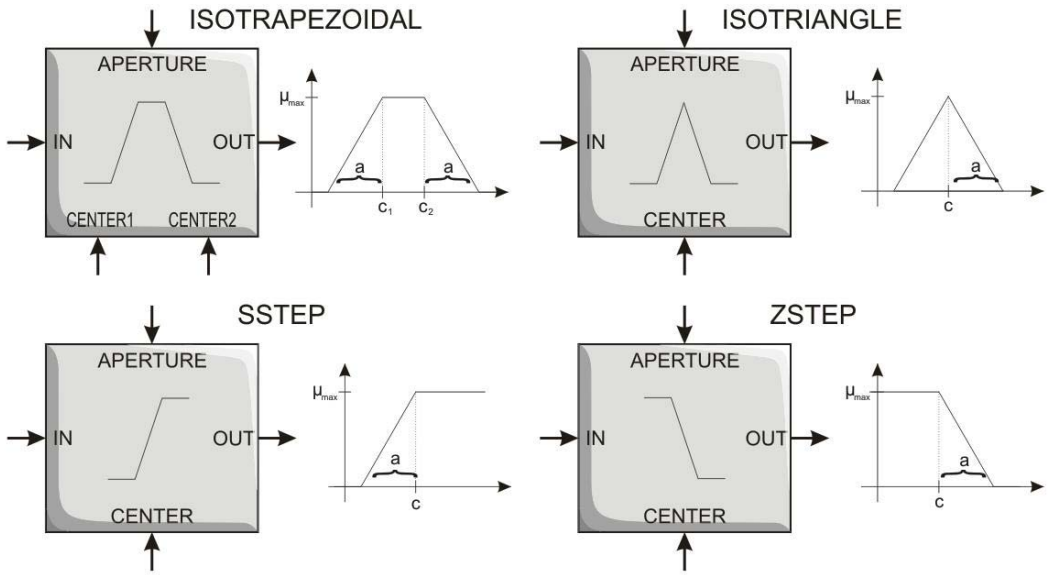

Figura 9. Funciones de membresía prácticas implementables en hardware.

Donde μ_{max} es igual a uno y en todos los casos a es la apertura de la función. Puesto que los valores de membresía según [28] están entre [0, 1], una representación numérica sencilla en un sistema computacional no es posible sin ocupar muchos recursos, por lo que es necesario *Normalizar* los valores de membresía que van de cero a uno con un equivalente numérico binario.

Debido a que el universo de discurso es lineal, la normalización es directa, haciendo coincidir el valor 1 con el valor máximo representable sin signo en b bits en un sistema computacional y el valor 0 con el valor mínimo representable sin signo. De tal manera que si $b = 8$ los valores de membresía pueden ir de 00000000 a 11111111 [00H, FFH]. Por lo tanto, $\mu_{max} = FFH$.

Como puede observarse, las ecuaciones (16–19) utilizan la operación de división para calcular la pendiente. Por lo que es necesario implementar un algoritmo de división. Un algoritmo adecuado para cada una de las funciones de membresía es la división de una constante entre una variable utilizando una modificación del algoritmo de la *División Con Restauración*. Para el cálculo de la división en la etapa de Desdifusificación es mucho más conveniente la división *Sin Restauración*, para el cálculo del recíproco. Debido a que las pendientes de los lados de la función de membresía triángulo son multiplicadas por el valor de entrada, es necesario implementar también el algoritmo de la multiplicación. Todos estos algoritmos fueron modificados, es decir hechos a la medida del diseño del FLC, basándose en los algoritmos de división y multiplicación existentes. Todos ellos están detallados en el Anexo 1 e incluidos en el CD adjunto a este trabajo.

Por simplicidad, sólo se utilizan las funciones: triangular, S y Z. El circuito digital que describe una función de membresía triangular es el siguiente:

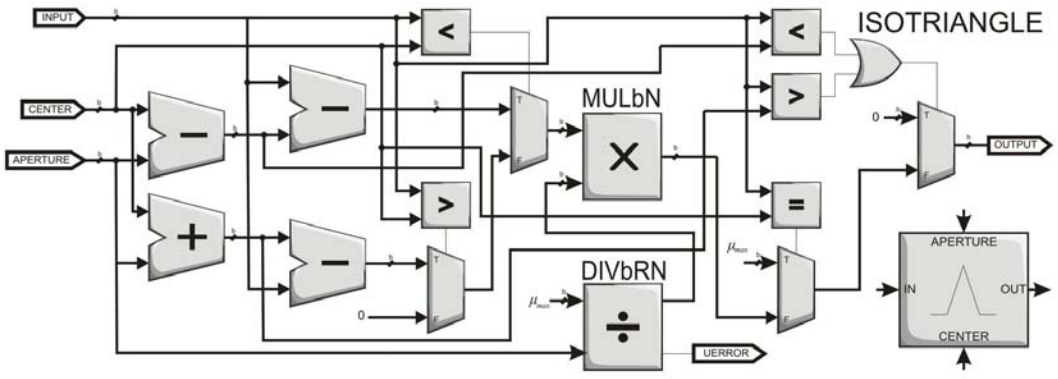

Figura 10. Diseño digital de una función de membresía triangular isósceles.

Como puede observarse en la Figura 22, los módulos más complejos y costosos son MULbN y DIVbN. Ambos módulos fueron implementados por la siguiente razón: para la

multiplicación sólo existe un número finito de multiplicadores dedicados en un dispositivo FPGA y el número de funciones de membresía que utilizan una multiplicación puede exceder este número; para la división no existe algún divisor dedicado en el FPGA. En la parte inferior derecha de la figura anterior está la representación de una función de membresía triangular isósceles, denotado como ISOTRIANGLE.

Figura 11. Función de membresía general.

La Figura 23 representa una función de membresía general, misma que puede ser cualquiera de las antes descritas. Las funciones S y Z son particularidades de la función triangular, por lo que se consideran en este trabajo, mas no se describen a detalle, ya que también se incluyen en el CD adjunto al mismo.

Con varias funciones de membresía es posible crear un difusificador para una variable y con varios difusificadores, si el sistema de control requiere un FLC de más de una variable, es posible crear un difusificador completo que contenga varios difusificadores independientes. De manera que para construir un difusificador para una variable es necesario hacerlo con varias funciones de membresía conectadas en paralelo, como se muestra a continuación:

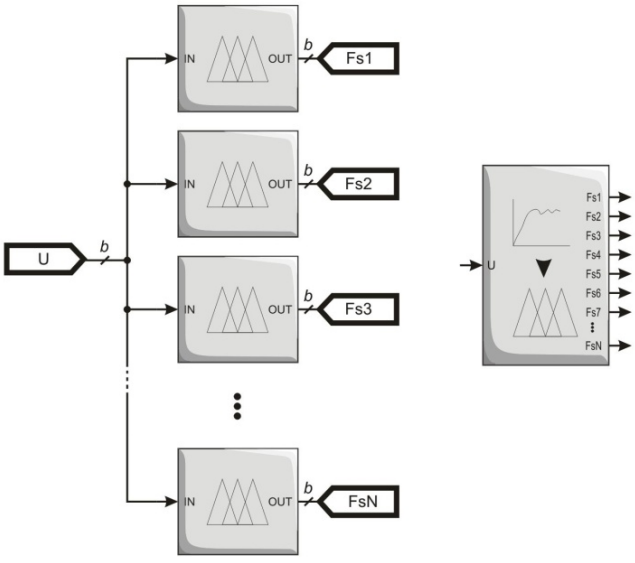

Figura 12. Difusificador para una variable.

Se trata de un difusificador de una variable que contiene N funciones de membresía. Este módulo se construye mediante N replicaciones, suponiendo que todas fueran funciones de membresía iguales. Todas ellas reciben la misma entrada y cada una se encarga de convertir un valor concreto en un valor difuso. El módulo de la derecha representa a un difusificador para una sola variable.

Para crear un difusificador completo, es necesario conocer el número de entradas al sistema, esto es n. Así, un difusificador de n entradas tiene la siguiente forma:

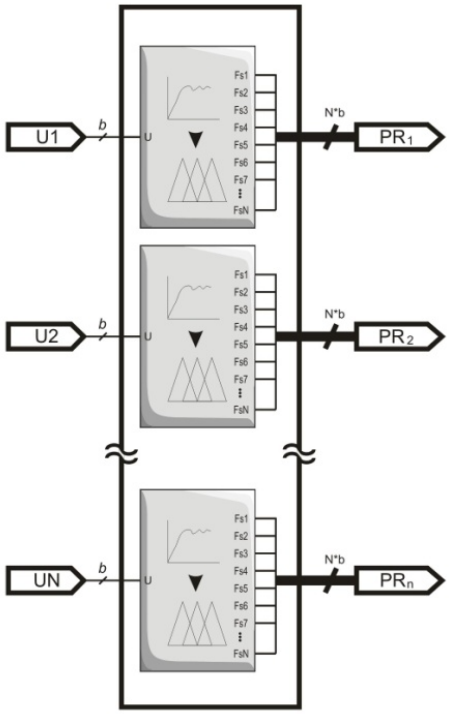

Figura 13. Difusificador completo.

La figura anterior muestra el paralelismo que existe entre los módulos de un difusificador completo. Cada uno de los difusificadores de una variable que contiene este módulo actúa independientemente de los demás. Todos los buses de salida tienen N vectores de b bits. Como se ha venido diciendo, el número entradas y el número de funciones de membresía posibles para este FLC, están en función del tamaño del dispositivo FPGA donde se quiera implementar.

ETAPA 5. Es momento entonces de construir el difusificador basándose en el sistema de control propuesto para el servomotor. Para esto, necesitamos definir el número de bits que el FLC procesará. En el estudio realizado en la sección Controladores

Difusos Existentes de los Antecedentes se determina que la mayoría de los FLCs realizados es de 8 bits, esto es $b = 8$.

Por otro lado, el sistema de control para el servomotor consta de un número de entradas al sistema, que es 2, o sea $n = 2$ y un número de salidas, que es 1, o sea m= 1. El número de funciones de membresía por entrada es 3, esto es $N = 3$, aunque no necesariamente las dos entradas deben tener el mismo número de conjunto difusos. Las funciones de membresía que se utilizan en el ejemplo son triangular, S y Z.

Hasta el momento sólo hace falta re–escalar los valores del universo de discurso de las variables de entrada y de salida a valores que un sistema computacional puede comprender, es decir, binario (o hexadecimal), para poder implementar el FLC en el FPGA, es decir, preparar a los valores de entrada para ser procesados. Este procedimiento es sencillo, puesto que sólo se toman los valores mínimo y máximo del universo de discurso y se sobreponen en los valores mínimo y máximo de los valores representables con 8 bits. Puesto que los valores posibles para el error de posición del rotor del motor van desde $-\frac{\pi}{2}rad$ a $\frac{\pi}{2}rad$ y los valores posibles para el cambio en el error de posición van desde $-\frac{\pi}{6}rad/s$ a $\frac{\pi}{6}rad/s$, los universos de discurso discretizados de las dos variables de entrada basados en las Figuras 16 y 18 quedan así:

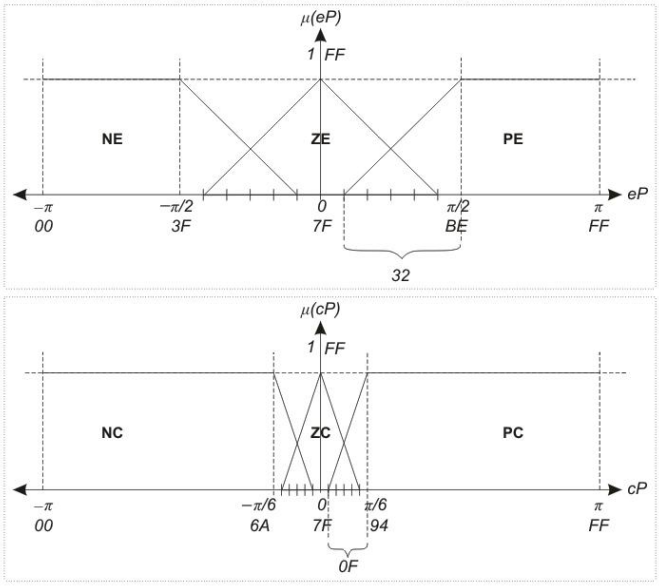

Figura 14. Universos de discurso discretizado de las variables de entrada para el FLC del servomotor.

En la Figura 26 puede observarse que ambos universos de discurso quedan escalados finalmente de $-\pi\, rad$ a $\pi\, rad$. Independientemente de la posición inicial del

rotor la posición deseada está únicamente en el intervalo $[-\pi, \pi]$ debido a que una circunferencia es una función periódica de π. Esto se aplica a las dos variables de entrada del FLC puesto que ambos dependen de la posición y es por eso que el error de posición *eP* y el cambio en este error *cP* no pueden estar más allá de este intervalo. De esta manera, los universos de discurso equivalentes en binario quedan bien definidos. También puede observarse que todas las funciones de membresía de *eP* tienen $a = 32H$ y todas las funciones de membresía para *cP* tienen $a = 0FH$. Los centros de las funciones de membresía de cada variable de entrada quedan distribuidos uniformemente en los valores hexadecimales mostrados, por ejemplo, el centro del conjunto difuso PE, $c = \frac{\pi}{2}$ se encuentra en el valor BE hexadecimal. Por lo tanto, se puede decir que cada valor del universo de discurso de cada variable de entrada tiene una resolución de $\frac{\pi}{128}$.

Finalmente, se debe hacer el mismo procedimiento para la variable de salida.

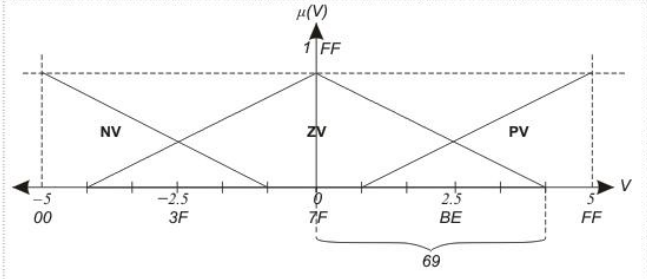

Figura 15. Universo de discurso discretizado de la variable de salida para el FLC del servomotor.

A diferencia de los universos de discurso de las variables de entrada, las funciones de membresía están distribuidas de manera no uniforme, ya que las funciones S y Z que se encuentran en los extremos no deben extenderse a más allá de ±5 volts. En su representación hexadecimal, las funciones S y Z sólo poseen un valor de membresía máximo en un solo valor de su universo de discurso. En la Figura 27, todas las funciones de membresía tienen una apertura de $a = 69H$.

Con todos estos valores, es posible construir el difusificador completo. Un difusificador como el propuesto en la Figura 24 construido bajo el enfoque RTC con funciones de membresía como los que se muestran en la Figura 21, puede aceptar cualquier valor de apertura que se encuentre entre $(00H, FFH]$, ya que un valor de apertura igual a cero provocaría una indeterminación, y puede aceptar cualquier valor de centro entre $[00H, FFH]$. Por lo tanto, los parámetros para construir un difusificador con los módulos de la Figura 21 son los siguientes:

Tabla 1. Parámetros de difusificador para funciones de membresía mediante enfoque RTC implementados en FPGA.

Variable	N (HEX)		Z (HEX)		P (HEX)	
	c	a	c	a	c	a
eP (E)	3F	32	7F	32	BE	32
cP (C)	6A	0F	7F	0F	94	0F
V (V)	01	69	7F	69	FE	69

La Tabla 3 muestra los valores de centro y apertura de las funciones de membresía que representan a los conjuntos difusos NE, ZE, PE, NC, ZC, PC, NV, ZV y PV, en sus correspondientes universos de discurso.

Denotemos a C y A como los *registros* que mantienen el valor de centro y apertura, c y a correspondientemente, y a su vez mantienen la configuración de las funciones de membresía ISOTRIANGLE, SSTEP y ZSTEP, que son la función triángulo isósceles, la función S y la función Z, mencionadas en la Figura 21, entonces los módulos difusificadores basados en la Tabla 3 y en la Figura 25, quedarían de la siguiente manera:

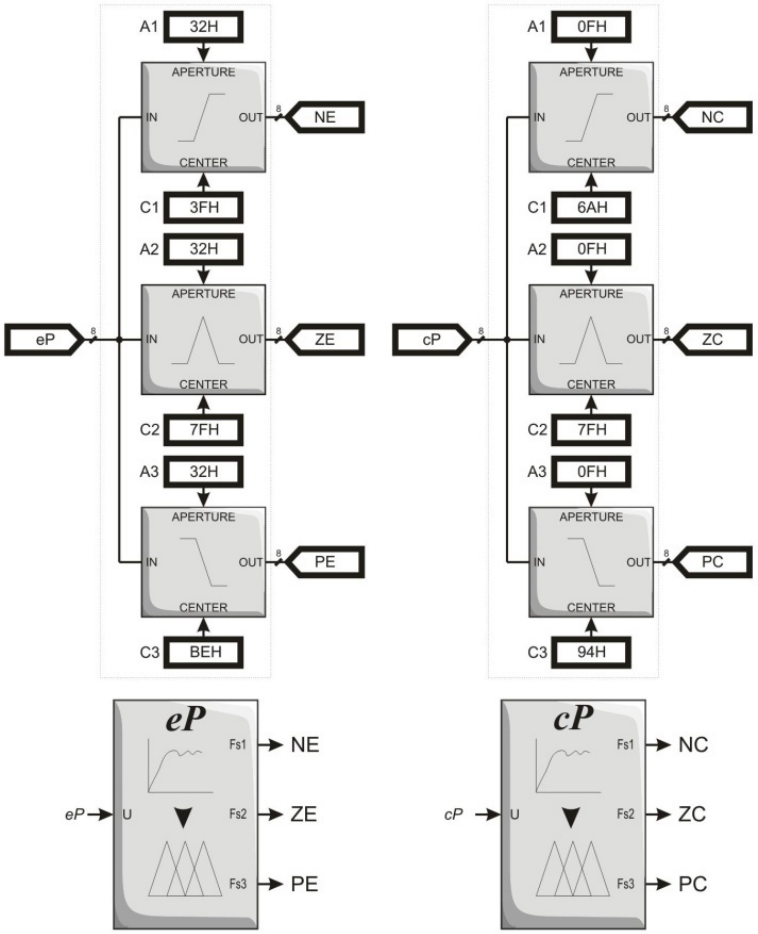

Figura 16. Difusificadores para las variables de entrada del FLC propuesto para el servomotor.

La variable de salida no requiere difusificarse debido a que a partir de las variables de entrada se inferirá un valor del universo de discurso de la variable de salida, basándose en los centros de sus funciones de membresía. Esto se debe a que se hará uso de la ecuación (14) *Centroide*, la cual calcula la salida concreta a partir de los centros y no de la forma de los conjuntos de la variable difusa a la salida. Este paso se ve más adelante en la sección del *Desdifusificador*.

Finalmente, la etapa de Difusificación termina juntando los difusificadores independientes de cada variable en el diseño de forma paralela. Entonces, el diseño del difusificador completo queda como se muestra en la Figura 29:

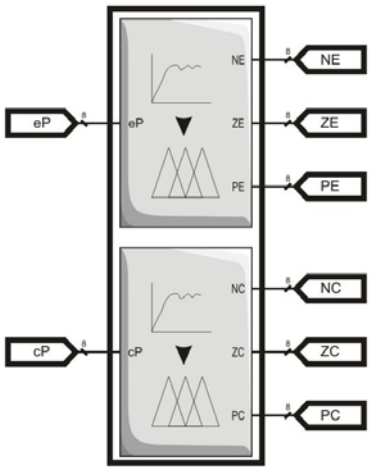

Figura 17. Difusificador completo del FLC propuesto para el servomotor.

El diseño del difusificador del FLC propuesto para el servomotor termina hasta aquí. A continuación se explica de manera detallada el diseño de la *Máquina de Inferencia Difusa* basada en el método de implicación llamado Mamdani que hará uso de los valores de membresía que se obtienen de la etapa de *Difusificación* para realizar una o varias conclusiones y entregar esos resultados a la etapa de Desdifusificación.

La máquina de inferencia difusa

El procedimiento de la inferencia difusa tiene alcance en la tercera Ley de Newton [47]:

"A toda acción corresponde una reacción en igual magnitud y dirección pero de sentido opuesto."

El propósito de la teoría de control en general, es la de encontrar precisamente una acción contraria (reacción) que responda a una acción o a un conjunto de acciones de tal manera que contrarreste su efectos en el proceso, es decir, que controle al proceso. Por ende, esa acción o acciones que inician el desbalance son las llamadas *premisas* y la reacción o reacciones son las *consecuencias.* Una inferencia es un método que nos permite resolver un problema (consecuencia lógica) basado en las premisas [48].

En general, la consecuencia lógica es una relación entre un conjunto de oraciones que funcionan como premisas, y otra oración que es sostenida como conclusión de esas premisas [49]. Tales oraciones, como se ha dicho en los Antecedentes, se encuentran en las Reglas de Inferencia Difusa y son la base del control difuso.

El método de implicación Mamdani, realiza inferencias precisamente a partir de las premisas implicadas. Esas inferencias calculan, mediante operaciones MIN y MAX, es decir la implicación y la agregación, los valores de membresía que deben tener los conjuntos difusos pertenecientes a la variable de salida del FLC, a partir de los valores de membresía de las variables de entrada implicadas.

Por lo antes mencionado, una Máquina de Inferencia Difusa que usa el Método de Implicación Mamdani opera de la siguiente manera [50]:

a) Identificar las reglas que tengan consecuencias en común dentro del conjunto de reglas de inferencia difusa.
b) Dentro del subconjunto de reglas obtenidas en el paso anterior, relacionar todas las premisas con módulos **MIN** en el mismo orden en que aparecen. Así pues a cada operación **AND** le corresponde un módulo **MIN** que realiza la operación de implicación. Si el conjunto de reglas fuera para un sistema SISO este paso no se aplica.
c) De todos los valores resultantes de los módulos **MIN**, encontrar el valor máximo de todos ellos mediante módulos **MAX** que realizan la operación de agregación, es decir, aplicar la operación **OR** para agregar todas las implicaciones del paso anterior. Este es entonces el valor que corresponde a la consecuencia. Si el conjunto de reglas fuera para un sistema SISO se deben conectar módulos **MAX** para llegar a la consecuencia.

Las premisas son los valores de membresía resultantes de la etapa de Difusificación. Los módulos MIN y MAX son simples multiplexores de b bits operados por comparadores *menor que* y *mayor que*, correspondientemente.

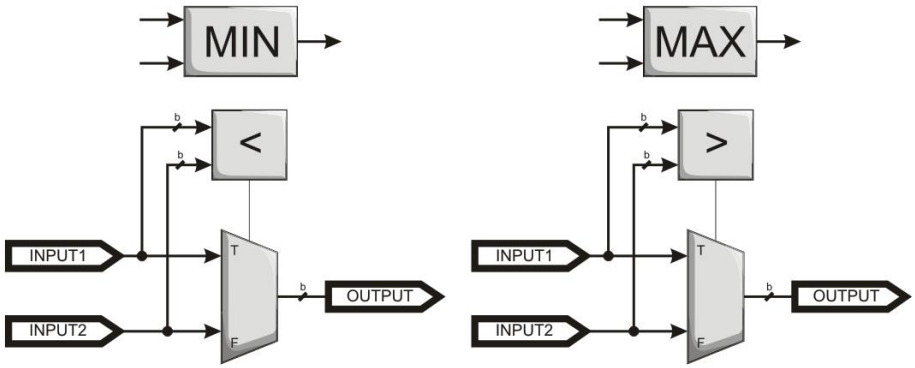

Figura 18. Módulos MIN y MAX para la construcción de la máquina de inferencia Mamdani.

Entonces, en un FLC en general, para encontrar una consecuencia se deben conectar las premisas con los módulos MIN y MAX de la manera correcta como se ha descrito en los pasos anteriores, lo cual se muestra en la Figura 31:

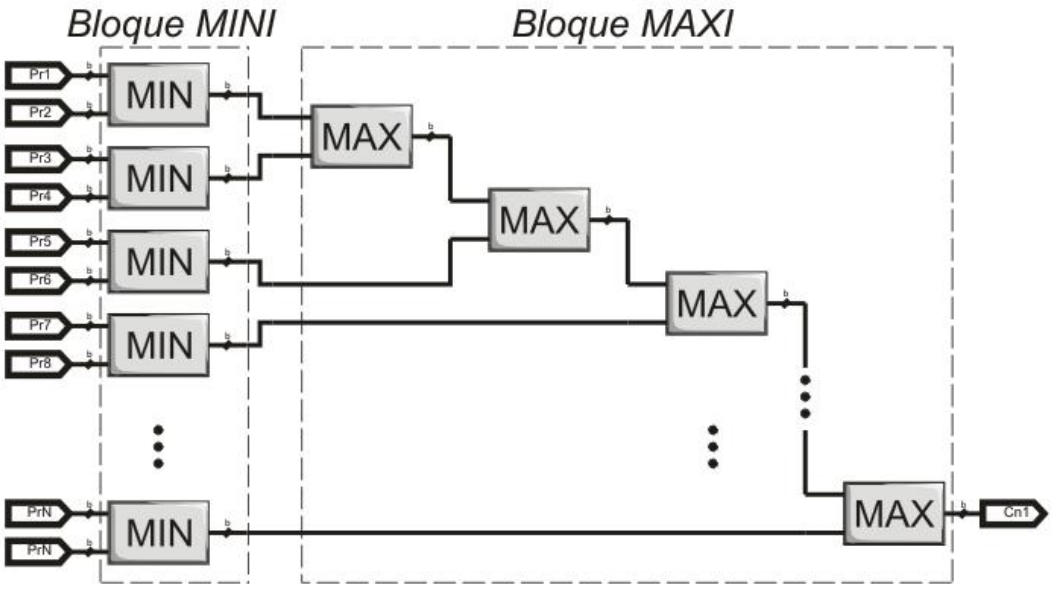

Figura 19. Interconexión general de módulos MIN y MAX para la elaboración de una máquina de inferencia difusa.

La estructura que se debe construir para obtener una consecuencia a partir de una o varias premisas es el *Árbol Invertido* y existen dos bloques que deben destacarse: El bloque MINI está conformado de varios módulos MIN conectados en paralelo; y el bloque MAXI está conformado de módulos MAX en cascada (en serie) y esta última, resulta ser la parte que más tiempo de propagación desarrolla. Estructuras de este tipo pueden existir en una máquina de inferencia difusa y una sola estructura puede representar varias reglas difusas a la vez. Además, la Figura 31 sólo representa una estructura de árbol invertida para una sola consecuencia de la variable de salida basada en la ecuación (2), para un sistema de control difuso MIMO; por lo que pueden existir varias estructuras como esta para cada consecuencia, es decir, para cada conjunto difuso de salida.

ETAPA 6. Para entender mejor este procedimiento, es necesario continuar con el diseño del FLC propuesto para el control del servomotor de la sección anterior. Ahora, el paso a) para el diseño de la máquina de inferencia difusa requiere analizar el conjunto de reglas difusas:

1. **IF** eP is NE **AND** cP is NC **THEN** V is NV ☆
2. **IF** eP is NE **AND** cP is ZC **THEN** V is NV ☆
3. **IF** eP is NE **AND** cP is PC **THEN** V is NV ☆
4. **IF** eP is ZE **AND** cP is NC **THEN** V is NV ☆
5. **IF** eP is ZE **AND** cP is ZC **THEN** V is ZV
6. **IF** eP is ZE **AND** cP is PC **THEN** V is PV
7. **IF** eP is PE **AND** cP is NC **THEN** V is PV
8. **IF** eP is PE **AND** cP is ZC **THEN** V is PV
9. **IF** eP is PE **AND** cP is PC **THEN** V is PV

Como puede observarse, las reglas 1, 2, 3 y 4 tienen una consecuencia en común y son el subconjunto de reglas que se analizan por ahora. Como se dice en el paso a), el conjunto difuso NV de la variable *V* es común en este subconjunto. Según el paso b), las premisas deben relacionarse con módulos MIN, es decir:

1. **IF** eP is NE **AND** cP is NC $\rightarrow min(\text{NE}, \text{NC})$.
2. **IF** eP is NE **AND** cP is ZC $\rightarrow min(\text{NE}, \text{ZC})$.
3. **IF** eP is NE **AND** cP is PC $\rightarrow min(\text{NE}, \text{PC})$.
4. **IF** eP is ZE **AND** cP is NC $\rightarrow min\ (\text{ZE}, \text{NC})$.

Entonces existen 4 módulos MIN como se muestra en la Figura 32:

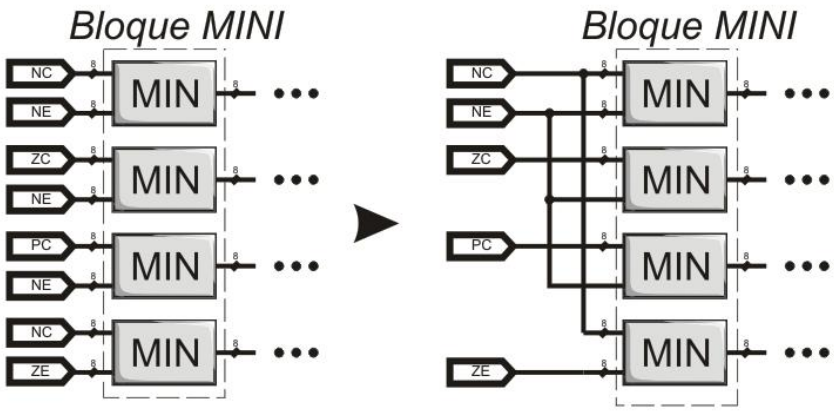

Figura 20. Conexión de módulos MIN.

Como puede observarse en la figura anterior, entran 5 premisas a la sección MIN donde NE se compara en cada módulo MIN con NC, ZC y PC; también ZE se compara con NC. Al final de estos módulos MIN se obtienen 4 valores mínimos u operaciones de implicación, de los cuales debe obtenerse el más grande de todos ellos para obtener el valor de membresía de la consecuencia correspondiente, como se dice en el paso c), es decir que existen 3 operaciones de agregación que contribuyen a la misma consecuencia.

Así pues, la conexión final de los módulos MIN–MAX para la consecuencia NV se presenta en la Figura 33:

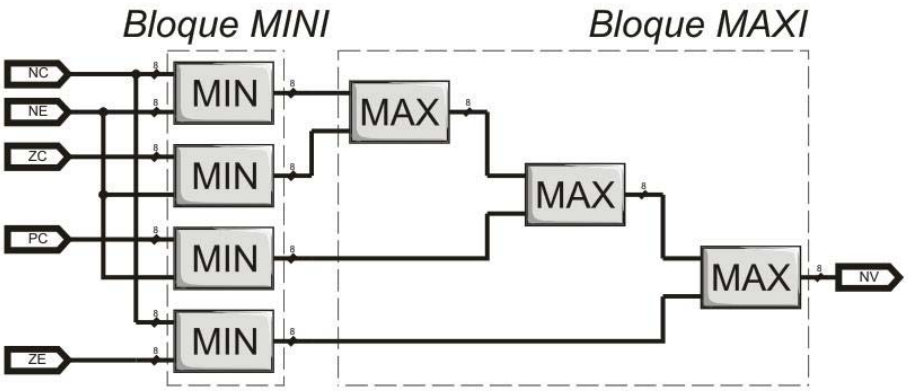

Figura 21. Conexión final de módulos MIN–MAX para la consecuencia NV.

De la misma manera se procede ahora con el siguiente subconjunto de reglas para el paso a):

1. **IF** *eP* is NE **AND** *cP* is NC **THEN** *V* is NV
2. **IF** *eP* is NE **AND** *cP* is ZC **THEN** *V* is NV
3. **IF** *eP* is NE **AND** *cP* is PC **THEN** *V* is NV
4. **IF** *eP* is ZE **AND** *cP* is NC **THEN** *V* is NV
5. **IF** *eP* is ZE **AND** *cP* is ZC **THEN** *V* is ZV ☆
6. **IF** *eP* is ZE **AND** *cP* is PC **THEN** *V* is PV
7. **IF** *eP* is PE **AND** *cP* is NC **THEN** *V* is PV
8. **IF** *eP* is PE **AND** *cP* is ZC **THEN** *V* is PV
9. **IF** *eP* is PE **AND** *cP* is PC **THEN** *V* is PV

Sólo la regla 5, (resaltada en color rojo) tiene una consecuencia, el conjunto difuso ZV de la variable *V*; y los conjuntos difusos implicados son ZE y ZC. Según el paso b), las premisas deben relacionarse con módulos MIN, es decir:

5. **IF** eP is ZE **AND** cP is ZC $\rightarrow min(\mathrm{ZE}, \mathrm{ZC})$.

Entonces, existe 1 módulo MIN, debido a que no existe ninguna otra premisa que contribuya hacia la misma consecuencia como se sugiere en el paso c), es decir, no existe ninguna operación de agregación; así pues, la conexión de módulos MIN–MAX se simplifica y queda de la siguiente forma:

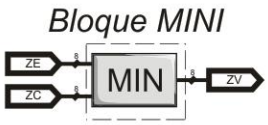

Figura 22. Conexión final de módulos MIN–MAX para la consecuencia ZV.

Por último, las reglas restantes forman un árbol invertido para la consecuencia PV, muy parecido al realizado para la consecuencia NV. El subconjunto de reglas para el paso a) es el siguiente:

1. **IF** eP is NE **AND** cP is NC **THEN** V is NV
2. **IF** eP is NE **AND** cP is ZC **THEN** V is NV
3. **IF** eP is NE **AND** cP is PC **THEN** V is NV
4. **IF** eP is ZE **AND** cP is NC **THEN** V is NV
5. **IF** eP is ZE **AND** cP is ZC **THEN** V is ZV
6. **IF** eP is ZE **AND** cP is PC **THEN** V is PV ☆
7. **IF** eP is PE **AND** cP is NC **THEN** V is PV ☆
8. **IF** eP is PE **AND** cP is ZC **THEN** V is PV ☆
9. **IF** eP is PE **AND** cP is PC **THEN** V is PV ☆

Las reglas 6, 7, 8 y 9 coinciden en la consecuencia PV de la variable V; y los conjuntos difusos implicados son NC, ZC, PC, ZE y PE. Según el paso b), las premisas deben relacionarse con módulos MIN, es decir:

6. **IF** eP is ZE **AND** cP is PC $\rightarrow min(\text{ZE}, \text{PC})$.
7. **IF** eP is PE **AND** cP is NC $\rightarrow min(\text{PE}, \text{NC})$.
8. **IF** eP is PE **AND** cP is ZC $\rightarrow min(\text{PE}, \text{ZC})$.
9. **IF** eP is PE **AND** cP is PC $\rightarrow min(\text{PE}, \text{PC})$.

Entonces existen 4 módulos MIN como se muestra en la Figura 35:

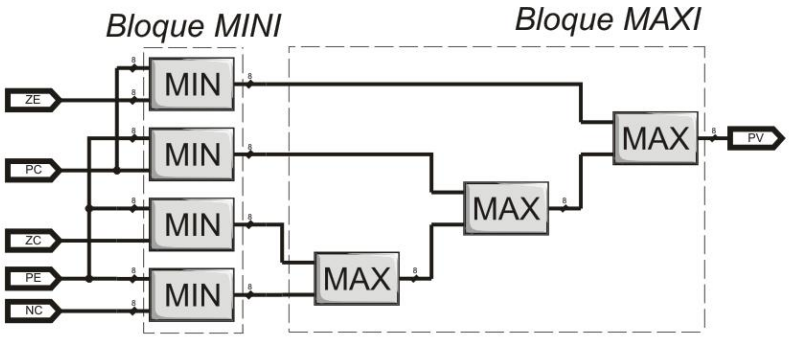

Figura 23. Conexión final de módulos MIN–MAX para la consecuencia PV.

Nótese que en cada árbol invertido siempre existen tantos módulos MAX como la cantidad de módulos MIN que hay, pero disminuido en uno. Finalmente, la máquina de inferencia para el FLC propuesto para el servomotor queda de la siguiente manera. Ver Figura 36.

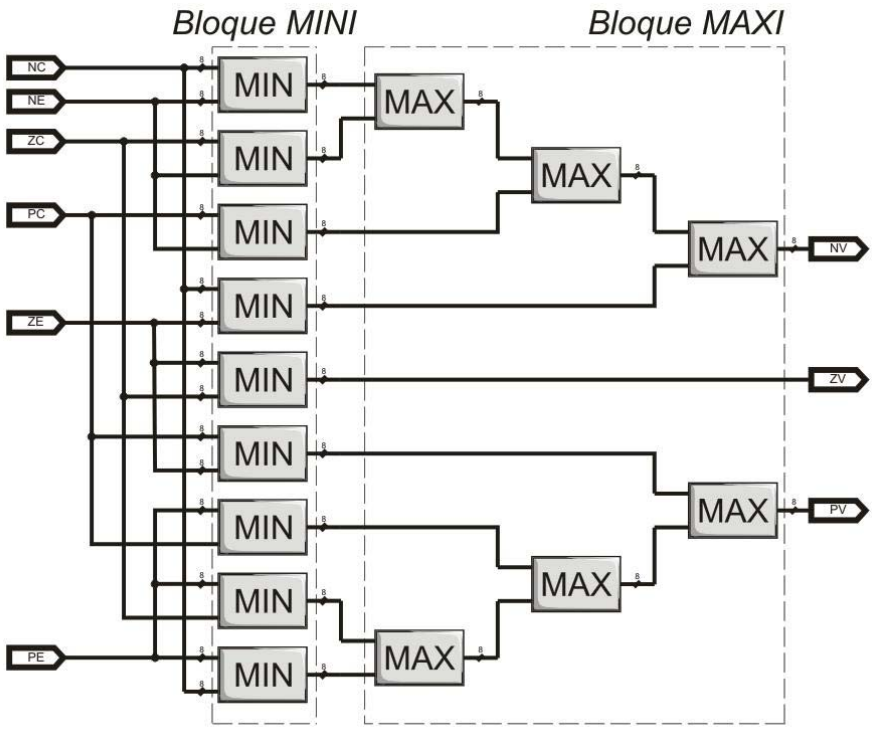

Figura 24. Máquina de inferencia difusa del FLC propuesto para el servomotor.

Cabe aclarar, que toda máquina de inferencia creada bajo este enfoque no es general y cada máquina de inferencia deberá ser configurada para cada proceso, de acuerdo al procedimiento realizado en esta y en la anterior sección. Los valores de membresía de todas las consecuencias resultantes de la máquina de inferencia deben pasar por la última etapa del control difuso: la Desdifusificación.

El desdifusificador

El desdifusificador es la última etapa del control difuso donde todos los valores de membresía de los conjuntos difusos de la variable de salida son transformados en valores reales y concretos. Existen muchas técnicas para obtener este valor concreto que finalmente será el que lleve la acción de control a cabo sobre el actuador y por ende sobre el proceso. La técnica más usada para desdifusificar, según el estudio realizado en la

sección Controladores Difusos Existentes en los Antecedentes, es la de *Centroide* y es la que se utiliza en la arquitectura de este FLC. También es conocida como el *Promedio de los Centros*, *el Centro de Gravedad (COG)*, etc.

Bajo el enfoque RTC esto implicaría utilizar la multiplicación y la división basándose en la ecuación (14).

$$y_q^{crisp} = \frac{\sum_{i=1}^{R} b_i^q \mu_i(u_1, u_2, \ldots, u_n)}{\sum_{i=1}^{R} \mu_i(u_1, u_2, \ldots, u_n)}$$

Esta ecuación realiza la multiplicación de cada valor de membresía de las consecuencias $\mu_i(u_1, u_2, \ldots, u_n)$ resultante de la máquina de inferencia con el valor de su centro correspondiente b_i^q. El resultado de cada multiplicación es acumulado y este resultado se divide entre la suma de todas los valores de membresía de las consecuencias. Es por esto que se dice en la sección *El Difusificador,* que la forma de las funciones de membresía que representan a los conjuntos difusos de las consecuencias no importa. Lo único que importa es la posición del centro, dentro del universo de discurso de la variable de salida del FLC, pues estos representan a los *pesos* o las *contribuciones* de las consecuencias al multiplicarse con los valores de membresía de cada consecuencia correspondiente para obtener un valor concreto y_q^{crisp}.

La utilización de los algoritmos de la multiplicación y de la división conlleva a dos problemas: la pérdida de velocidad de procesamiento y/o al incremento de recursos utilizados en el FPGA. También es claro que la velocidad de un FLC bajo el enfoque RTC está en función de estos algoritmos. Es necesario entonces lidiar con estos dos problemas y buscar la manera adecuada de mejorar el rendimiento y el ahorro de recursos. Es por esto que se proponen dos maneras de realizar el desdifusificador:

1. Utilizando la *División Sin Restauración* (utilizado en el Difusificador).
2. Utilizando la *División Sin Restauración* modificada para calcular el recíproco.

Un desdifusificador general para la primera propuesta se muestra en la Figura 37:

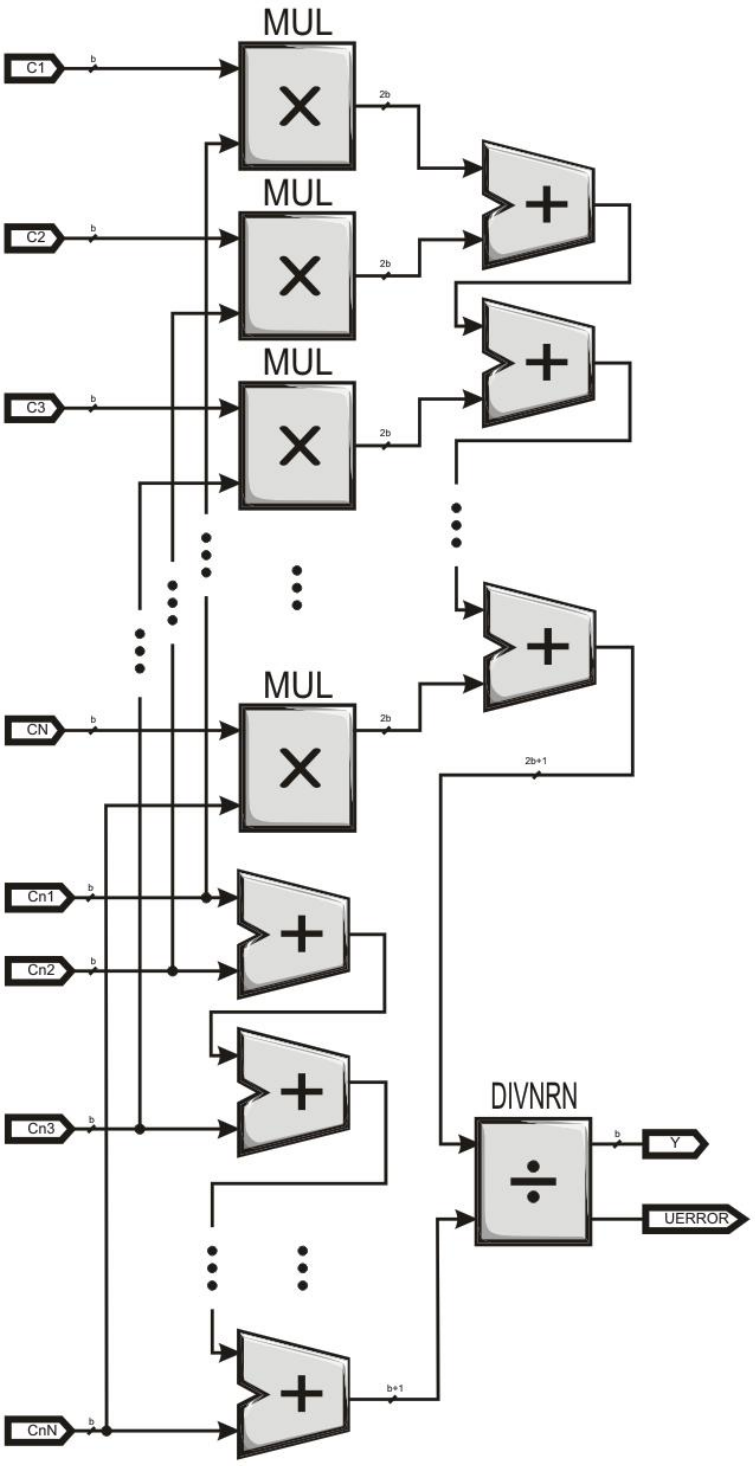

Figura 25. Diseño de un desdifusificador centroide general.

Como puede observarse en la Figura 37, la parte del denominador se encuentra en la parte de abajo, donde cada consecuencia *Cn* es acumulada mediante operaciones de suma en cascada. Debido a que cada par de entradas al desdifusificador es sumado y el resultado tiene un bit excedente para el acarreo, es decir que el tamaño de todos los sumadores es de $b+1$ bits. Esto es porque el número de funciones de membresía traslapadas es de 2 y por lo tanto, las consecuencias resultantes de la máquina de inferencia tienen valores diferentes de cero en sólo 2 de ellas. Así, el resultado acumulado que funge del divisor en la operación de la división sólo es de $b+1$. La parte del numerador es más complicada porque el cálculo de este resultado implica la multiplicación del centro por su consecuencia correspondiente. Se puede observar que en la mitad superior de la figura existen módulos MUL que realizan la multiplicación de b bits y su resultado es del doble de bits, $2b$. El numerador de la fracción o el dividendo para la división es el resultado de acumular mediante operaciones de suma en cascada de cada uno de las consecuencias ponderadas resultantes de cada multiplicación. Finalmente, es necesario realizar la división mediante el algoritmo *Sin Restauración* de $2b$ bits, detallado en el Anexo 1.

Este desdifusificador tiene la inconveniencia de tener operandos de $2b$ bits en el módulo DIVNRN, lo que lo hace un algoritmo lento y costoso. Una manera de mejorar esto sería: ocupar módulos para la división DIVNRN de b bits, lo cual ayuda a reducir el tiempo de propagación de este diseño combinatorio. Esto se muestra en la segunda propuesta a continuación:

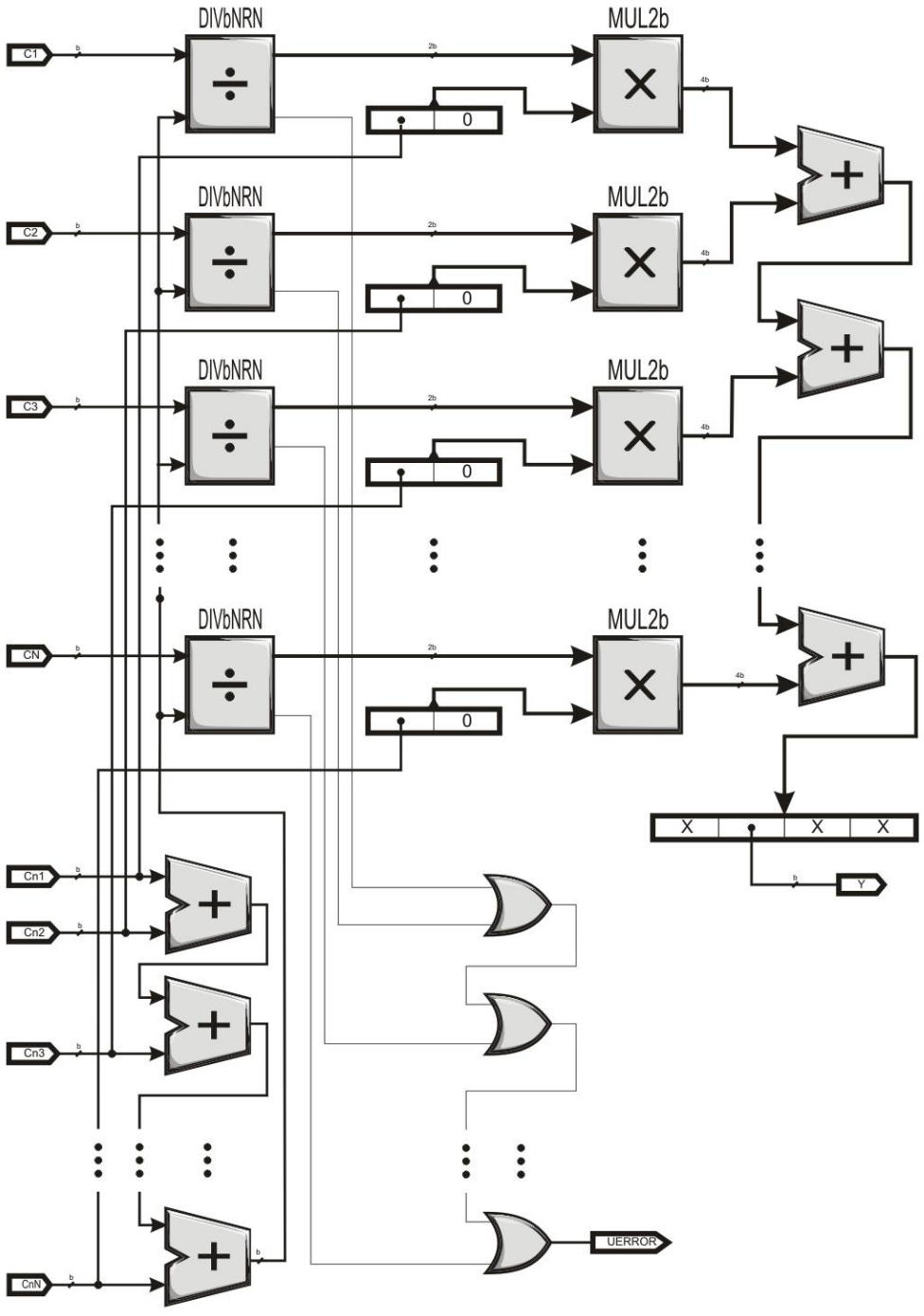

Figura 26. Diseño de un desdifusificador centroide general mejorado en tiempo.

Como puede observarse en la Figura 38, de manera general el diseño del desdifusificador es un poco más complejo y relativamente requiere de más recursos. Esto implica un módulo DIVbNRN por cada consecuencia en la primera etapa, realizando la división sin restauración modificada de b bits del centro de la consecuencia entre la suma de todos los valores de membresía de todas las consecuencias, que es equivalente a obtener el recíproco de la suma de todos los valores de membresía de todas las consecuencias multiplicado por el centro de una de las consecuencias. Esta división recibe dos valores de b bits y obtiene como resultado un valor de $2b$ bits en formato binario fraccional. Los b bits más significativos representan la parte entera y los b bits menos significativos la parte fraccionaria, esto debido a que la suma de todos los valores de membresía de las consecuencias siempre es mayor o igual al valor del centro, entonces el resultado siempre es menor a uno, esta es la primera etapa del desdifusificador.

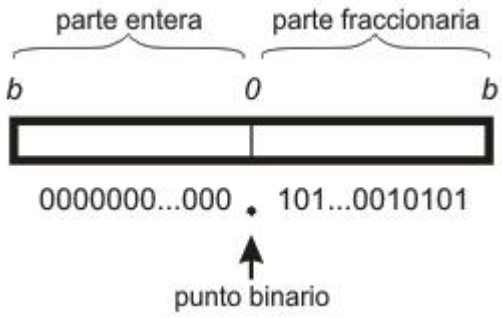

Figura 27. Representación de un registro con punto binario fijo.

La siguiente etapa se encarga de multiplicar el resultado de cada división por cada uno de los valores de membresía de su consecuencia correspondiente, concatenado con una cadena de b bits que representa un cero en la parte fraccionaria, donde cada módulo multiplicador MUL2b recibe valores de $2b$ bits y entrega valores de $4b$ bits. Aunque los valores obtenidos son mucho más grandes que en el enfoque anterior, se puede justificar pues una multiplicación siempre es menos costosa en recursos y en tiempo que una división. La última etapa se encarga de acumular todos los resultados de cada multiplicación y entregar el resultado a la salida. Debido a que este resultado es demasiado grande, longitud de $4b$ bits, sólo los b bits menos significativos de la parte entera contienen el valor concreto que se busca conseguir con este desdifusificador.

En ambos casos, si no existiera la restricción del número de funciones de membresía máximo que pueden estar traslapadas a dos, entonces existirían más de dos consecuencias con valores diferentes de cero y esto acarrearía el uso de más recursos para incrementar el número de bits que debe tener el divisor en el primer caso, pues cada suma que se realice no garantiza que el valor obtenido no sobrepase a la cantidad máxima que puede representar, además de perder velocidad. En el segundo enfoque, también

existiría el mismo problema a la hora de acumular todos los resultados, sin embargo, presenta menos dificultad, pues al módulo de división no se le tiene que incrementar el número de bits, lo cual lo hace más eficiente, pero más costoso en hardware.

El número de módulos de división y multiplicación depende del número de conjuntos difusos que el diseñador haya elegido para la salida. Cada conjunto difuso para la variable de salida constituye un grado de libertad para la acción de control, aunque esto implique mayor uso de recursos para implementarlo en el FPGA.

ETAPA 7. A continuación se diseña el desdifusificador del FLC propuesto para el control del servomotor. Un desdifusificador se diseña de acuerdo al número de conjuntos difusos que se requieren a la salida, para llevar a cabo el control (consecuencias) y al número de conjuntos difusos implicados (premisas) a través de la máquina de inferencia difusa. Así pues, todas las premisas están relacionadas según el conjunto de reglas de inferencia difusa y el resultado son tres acciones de control diferentes: aplicar voltaje positivo, aplicar voltaje negativo y no aplicar voltaje.

Los valores de estos tres conjuntos difusos entran al desdifusificador y calculan el valor de salida correcto de voltaje que debe aplicarse al servomotor para llegar a la posición deseada mediante la obtención del promedio ponderado por los centros de los esos conjuntos difusos. El primer enfoque del desdifusificador queda de la siguiente manera:

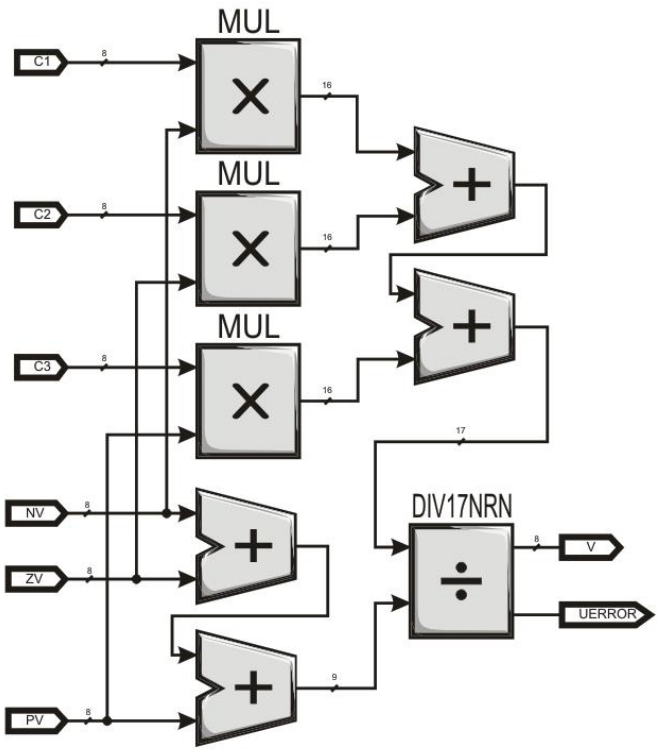

Figura 28. Desdifusificador del FLC propuesto para el servomotor.

El segundo enfoque, es mejor en cuanto se refiere a velocidad porque utiliza un módulo de división de 8 bits, en vez de uno de 16 bits como se muestra en el primer enfoque de la Figura 40. Se realiza un análisis de tiempos de todos los módulos diseñados para este FLC, al implementarlo en FPGA, en el capítulo de *Resultados*. El circuito del segundo enfoque queda de la siguiente manera:

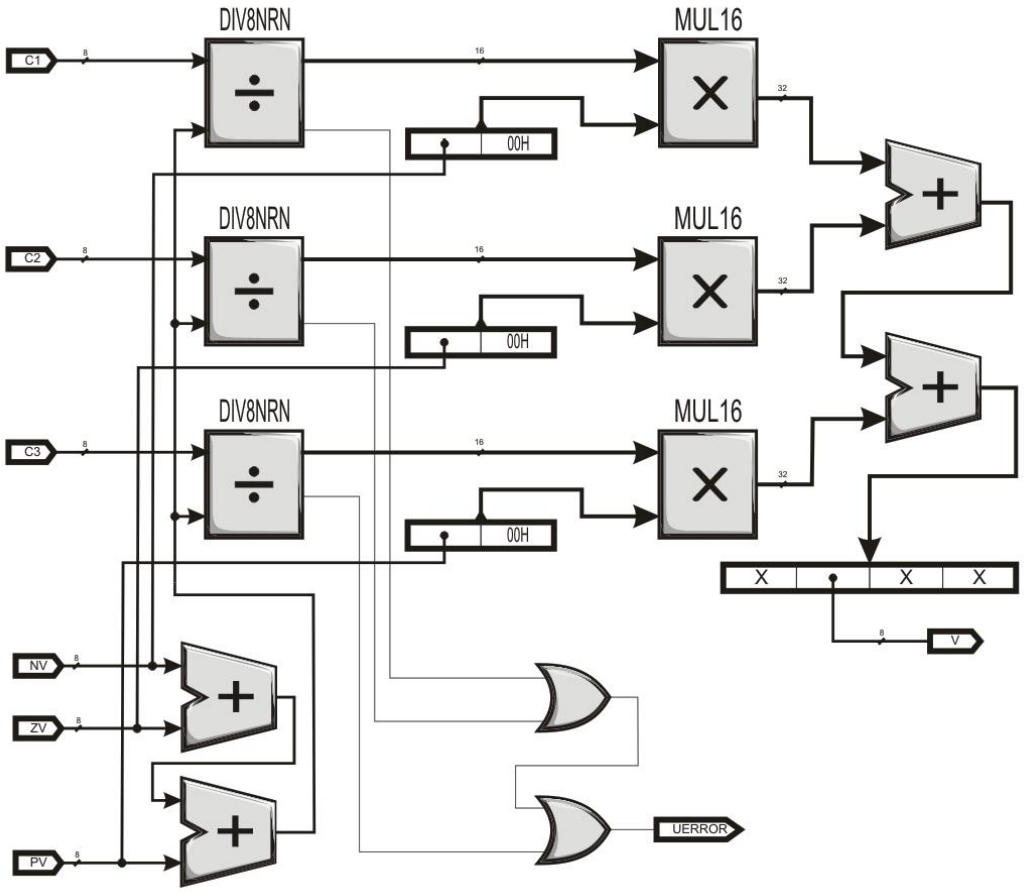

Figura 29. Desdifusificador mejorado del FLC propuesto para el servomotor.

Cabe destacar que el resultado de 8 bits, que se obtiene de la Figura 41, puede ser ampliado con los primeros 8 bits correspondientes a la parte fraccionaria para tener un resultado más preciso de 16 bits. Como puede observarse en el diseño del FLC a lo largo de todo este capítulo, todos los módulos muestran el alto nivel de *Paralelismo* que existe, característico de diseños lógicos combinatorios; así también la *Replicación* se hace ver claramente en todos los módulos, lo que convierte a este diseño en uno generalizado hasta cierto punto y con posibilidades de realizar adaptación en tiempo real. La siguiente figura muestra al FLC conectado a un proceso que es supervisado por un sistema inteligente, el cual realiza la adaptación al modificar los parámetros de las funciones de membresía:

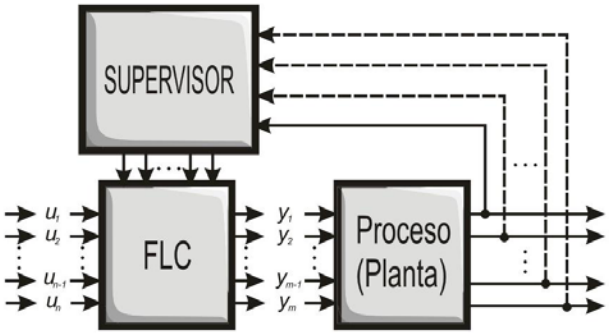

Figura 30. Capacidades de adaptación FLC, mediante un sistema supervisorio inteligente.

Hasta este punto, en este trabajo se termina el diseño de las tres etapas principales de la arquitectura de un FLC. El siguiente capítulo contiene los resultados obtenidos tras la simulación y la implementación de los módulos descritos en este capítulo y en los programas mencionados en la sección *Herramientas*. Por último, la ***ETAPA 8*** del diseño del FLC se encuentra distribuida en todo el capítulo siguiente, llamado *Resultados*, en donde se muestran los resultados de tiempos y costos en hardware, al implementarlo en FPGA.

Resultados 4

Este capítulo contempla la verificación del funcionamiento del FLC y cada uno de sus módulos descritos en el capítulo anterior, implementados en un FPGA y realizando una comparación con los resultados de la simulación. Como se ha dicho antes en la sección Metodología del Capítulo 1, se realiza una comparación del diseño del FLC en VHDL mediante la simulación en Modelsim y las herramientas de Xilinx con la simulación realizada del FIS en el Fuzzy Toolbox de MATLAB. Además se realiza un análisis de rendimiento del FLC para verificar los tiempos en el diseño digital combinatorio y las ventajas de utilizar el paralelismo. También se muestran los resultados de los procesos de Síntesis y Mapeo del ISE de Xilinx para obtener los tiempos aproximados de procesamiento y los recursos que ocuparía cada módulo si cada uno fuera implementado por separado en el FPGA; y de esa misma manera para el FLC completo.

El algoritmo de la división resulta ser el más lento y el más costoso de todos los módulos que componen al FLC, es decir, al difusificador, a la máquina de inferencia y al desdifusificador, como se muestra en la Tabla 4. Su importancia en el enfoque RTC es crucial junto con las operaciones de suma, resta y multiplicación para representar cualquier ecuación matemática de mayor complejidad. Por ello es necesario elegir el algoritmo más rápido y/o menos costoso para obtener mejores resultados en cuanto a rendimiento a la hora de integrarlo a un sistema como el propuesto en este trabajo.

Para ello, haciendo uso de las herramientas de *Síntesis* y *Mapeo* de ISE de Xilinx, es posible conocer de una manera muy precisa los recursos utilizados en el FPGA y de manera aproximada el retardo de propagación de cada uno de los módulos implementados y del diseño completo en general. La siguiente sección muestra los resultados de todos los módulos en cuanto a recursos y en cuanto a tiempos.

Resultados generales

ETAPA 8. La Tabla 4 resume el conteo de recursos y los tiempos de retardo de propagación obtenidos con las herramientas de ISE, después de realizar el proceso de *Síntesis y Mapeo* e implementar el diseño en el kit de desarrollo *Spartan 3*–FPGA de Xilinx. En la primera columna llamada *Algoritmo* se da una breve descripción de los módulos

diseñados para el FLC propuesto para el control del servomotor. La siguiente columna llamada *Módulo VHDL* contiene los nombres de los archivos *.vhd que se encuentran y están a disposición del lector en el CD adjunto a este trabajo. La tercera columna contiene los tiempos de retardo de propagación de cada uno de estos módulos en nanosegundos (ns) si se implementaran cada uno por separado en el FPGA o también todo el diseño como se muestra en las últimas dos filas de la tabla. Con estos tiempos es posible obtener el número de operaciones por segundo (MOPS) de cada módulo por separado. Las MOPS de los módulos de las últimas dos filas son equivalentes al número de inferencias lógico difusas por segundo (MFLIPS) mencionado con anterioridad. El resto de las columnas contienen los recursos utilizados en el FPGA según los procesos de síntesis y mapeo de ISE, es decir, la cantidad de *Look–Up Tables* o tablas de búsqueda utilizadas para implementar el diseño y por ende la cantidad de rebanadas o *Slices* de los elementos lógicos o *CLB*s que constituyen al FPGA. Así también el número de pines que utilizaría el FPGA mediantes los *IOB*s o módulos de entrada y salida si se implementara cada módulo por separado.

Tabla 1. Resultados de recursos utilizados y retardos de propagación en FPGA usando las herramientas de Xilinx ISE.

Algoritmo	Módulo VHDL	Retardo de propagación (ns)	MOPS	LUT	Slices	IOB	Número de compuertas equivalentes
División sin restauración de 16 bits	DIV16NRN	48.50	20.62	644	343	49	6852
División sin restauración de 8 bits modificada	DIV8NRN	28.83	34.68	208	107	25	1863
División con restauración alternativa de 8 bits	DIV8RN	28.84	34.67	124	67	17	831
Multiplicación de 8 bits	MUL8N	13.17	75.93	36	20	24	320
Función de membresía triángulo isósceles	ISOTRIANGLE	36.70 14.51	27.25 68.92	251	134	33	1994
Función de membresía escalón–S	SSTEP	36.70	27.25	249	133	34	1920
Función de membresía escalón–Z	ZSTEP	36.70	27.25	251	134	34	1922
Difusificador	FUZZIFIER	37.42	26.72	755	404	81	5844
Desdifusificador 1	DEFUZZIFIER1	54.86	18.23	681	370	57	19290
Desdifusificador 2	DEFUZZIFIER2	41.49	24.10	677	346	57	18240
Máquina de inferencia Mamdani	MAMDANI	19.32	51.75	242	122	72	1800
Operaciones MIN y MAX	MAX/MIN	9.36	106.84	16	8	24	120
Controlador Lógico Difuso (1)	**FLC1**	**94.99**	**10.53**	**2433**	**1298**	**145**	**32754**
Controlador Lógico Difuso (2)	**FLC2**	**84.01**	**11.90**	**2689**	**1428**	**145**	**32721**

Cabe aclarar que todos los valores de retardo de propagación son valores estimativos, ya que para el programa ISE calcula todos los diseños como si ellos fueran a implementarse por separado en el FPGA, por lo que agrega retrasos de tiempo que usa para los búferes que se encuentran en los módulos IOB para conectarse con el exterior del mismo. Los valores finales de los 2 FLCs propuestos, el primero con el desdifusificador con división de 16 bits y el segundo con el desdifusificador alterno con división de 8 bits, se encuentran en las últimas dos filas de la Tabla 4, para verificar la velocidad de todo el diseño y los recursos utilizados del FPGA. Estos valores son comparables con los diseños mostrados en la Tabla 2, ya que este diseño realiza poco más de 10 MFLIPS utilizando lógica combinatoria. Como se explica en los Antecedentes, todos los diseños que reportan la velocidad de su FLC son dependientes de una señal de reloj lo cual puede beneficiar en el rendimiento y/o perjudicar en la complejidad del diseño cuando se trata de escalarlo. Sólo existen tres diseños que sobrepasan el rendimiento en velocidad del FLC propuesto, basado en la arquitectura de diseño del capítulo *Diseño*; esto debido al tipo de procesamiento de datos utilizado, como puede ser la segmentación. La ventaja del FLC propuesto es la modularidad que exhibe al escalar el diseño y por la tanto, la facilidad de crear diseños de FLC de manera rápida y sencilla mediante la replicación y el paralelismo.

Como puede observarse en la Tabla 4, los módulos DIV16NRN, DIV8NRN, DIV8RN, MUL8N, MIN y MAX fueron diseñados con el propósito de construir el resto de los módulos que se muestran en la tabla que son parte medular del FLC propuesto para el control del servomotor. Cabe aclarar que en este capítulo sólo se muestran los resultados del FLC para este sistema de control sin implementar toda la aplicación (control del servomotor), con el fin de mostrar al lector la facilidad para implementar cualquier sistema de control utilizando un FLC basado en estos módulos.

Los módulos de la división son más lentos a diferencia de la multiplicación y esto puede observarse en la Tabla 4. También puede observarse que a mayor número de bits utilizado mayor es el tiempo que se lleva para procesar la información. El módulo DIV16NRN, perteneciente al desdifusificador 1, es la división sin restauración de 16 bits modificada, por eso lleva el posfijo NRN. El módulo DIV8NRN es la división sin restauración de 8 bits, que a diferencia de DIV16NRN, no sólo tiene como diferencia el número de bits, sino que se encarga de realizar la división de un número cualquiera entre otro número, que necesariamente debe ser mayor que el anterior, obteniendo como resultado un valor menor o igual que uno. Esto es equivalente a multiplicar un número cualquiera por el recíproco de otro número cualquiera. Este algoritmo estrega al desdifusificador 2 un valor en formato fraccional en binario para su posterior procesamiento. Finalmente, el módulo DIV8RN es el algoritmo de la división con restauración de 8 bits modificado; este algoritmo es implementado mediante otros módulos que ejecutan cada iteración de esta división de una manera diferente. Esto permite una mejora en tiempo respecto a los otros. La diferencia entre DIV8RN Y DIV8NRN radica en que DIV8NRN está diseñado para realizar el

recíproco multiplicado y devuelve un valor menor que uno; mientras que DIV8RN realiza la división de dos enteros, por lo que su resultado es un entero mayor que uno. DIV8RN es más rápido que su contraparte (división sin restauración) de 8 bits, debido a que realiza la división de un entero constante entre otro entero contenido en un registro, es decir en un entero variable.

Las siguientes figuras muestran la ruta crítica de los diseños realizados con VHDL que conforman las 3 etapas del FLC propuesto para el control del servomotor, que son: la Difusificación, la Inferencia Difusa y la Desdifusificación. Esto con el propósito de mostrar la trayectoria que describen los datos al entrar al módulo lógico y justificar el tiempo que tarda en recorrerlo hasta obtener un valor correcto a la salida. El Difusificador está basado en los módulos ISOTRIANGLE, SSTEP y ZSTEP. Por simplicidad se muestra en la Figura 43 sólo el módulo ISOTRIANGLE, pues los otros dos módulos están basados en este primero y además tardan un poco menos de tiempo en procesar la información que este módulo, como puede verse en la Tabla 4, esto quiere decir que ISOTRIANGLE es la función de membresía diseñada más lenta y como puede observarse en la Figura 28, al ponerse en paralelo con todas las demás ISOTRIANGLE resulta ser el módulo que limita la velocidad del difusificador.

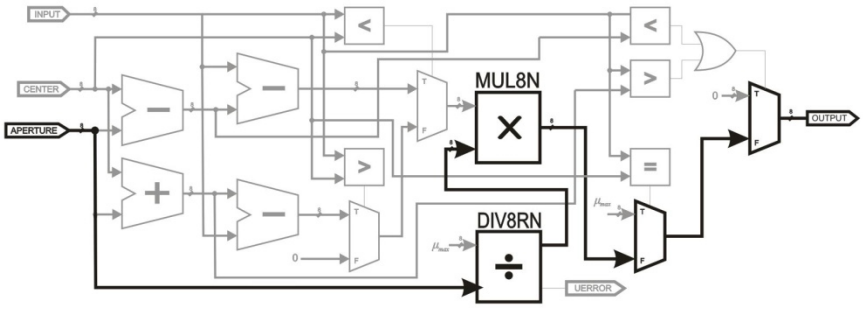

Figura 1. Ruta crítica del módulo ISOTRIANGLE.

Como puede observarse en la Figura 43, el cálculo de la pendiente mediante la operación de la división y la multiplicación del valor de entrada por esta pendiente, representan las operaciones más lentas del módulo ISOTRIANGLE, es decir provocan un retardo de propagación que afecta a todo el difusificador. Esto puede comprobarse en la Tabla 4, donde la diferencia entre los retardos de propagación de los módulos ISOTRIANGLE y FUZZIFIER es de menos de 1ns. Sin embargo, si la apertura del triángulo no fuera un valor que cambia constantemente en el diseño, salvo para modificar la configuración del FLC, la ruta crítica no pasaría directamente por el módulo DIV8RN y sí solamente por el módulo MUL8N. Esto debido a que si la apertura del triángulo permanece constante, la pendiente es constante igualmente y el cálculo del valor de membresía a la salida sólo pasaría por el módulo MUL8N al multiplicar la pendiente

constante por el valor de entrada modificado. Esto ahorraría el tiempo que utiliza para calcular la pendiente a través del módulo de división y agilizaría a todo el difusificador como se muestra en la Figura 44:

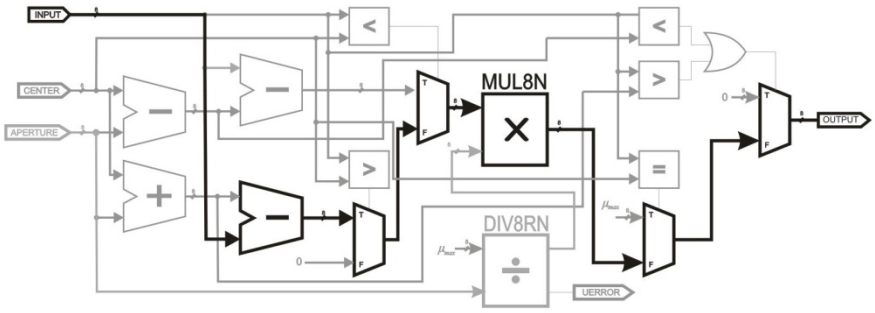

Figura 2. Ruta crítica para cálculo de valor de membresía del módulo ISOTRIANGLE.

Entonces el tiempo de retardo de propagación de la función triángulo isósceles es el siguiente:

$$t_d(ISOTRIANGLE) = t_d(\text{SUB}) + 2t_d(\text{MUX}) + t_d(\text{MUL}) \qquad (20)$$

Esta es la ruta crítica más importante del módulo ISOTRIANGLE, debido a que cualquier valor que cambie en la entrada es totalmente independiente del cálculo de la pendiente del triángulo isósceles. La Figura 43 es importante también pues si se requiere ajustar la apertura del triángulo para realizar alguna adaptación del FLC, se llevaría mucho más tiempo procesar toda la información, pues en este caso sí haría uso del módulo de división. Por lo tanto, la ruta crítica de la Figura 44 mostraría un retardo de propagación aproximado de 14.51ns. Para esto se modificó el módulo ISOTRIANGLE eliminando el módulo de la división y haciendo constante al valor de la pendiente. Este resultado es mostrado en la misma Tabla 4, debajo del tiempo de retardo de propagación del módulo ISOTRIANGLE. Esto mismo sucede para los módulos SSTEP y ZSTEP.

De manera general, el tiempo de retardo de propagación del difusificador está en función del tiempo de retardo de propagación de sus funciones de membresía que conforman a los conjuntos difusos para determinada variable. Así pues la Figura 45 muestra el paralelismo que existe en los módulos de las funciones de membresía del difusificador, del módulo FUZZIFIER, y el tiempo de retardo de propagación del mismo está en función del retardo del módulo ISOTRIANGLE.

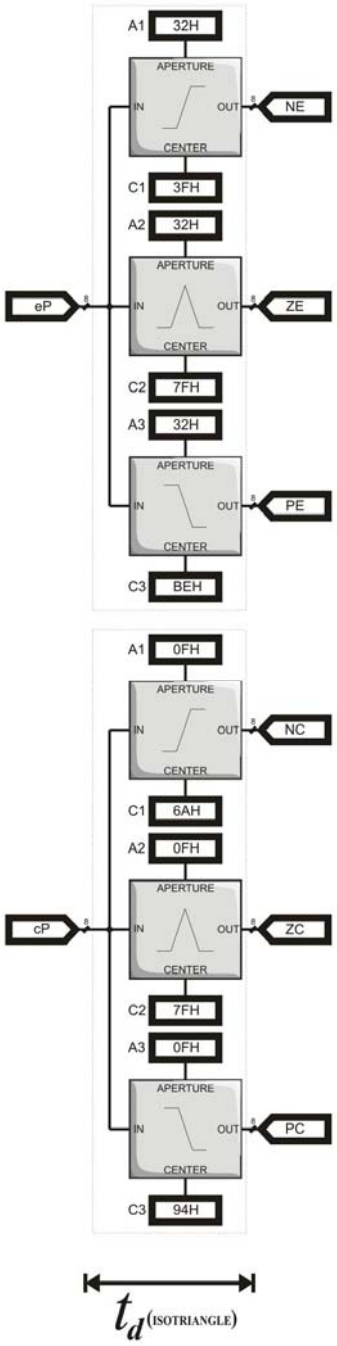

Figura 3. Retardo de propagación de los difusificadores.

Entonces el retardo de propagación del difusificador del FLC propuesto para el control del servomotor es el siguiente:

$$t_d(FUZZIFIER) = t_d(ISOTRIANGLE) \quad (21)$$

El retardo de propagación de la máquina de inferencia difusa, está basado en el retardo de propagación de los módulos MIN y MAX. Sin embargo, la manera como se conectan estos módulos puede provocar que el retardo de propagación crezca de manera considerable al incrementar el número de entradas, al incrementar el número de conjuntos difusos por cada entrada y por lo tanto al considerar todas las reglas de inferencia, es por ello la importancia de las *Etapas 1–4* de la Metodología Propuesta. Como puede observarse en la Figura 36, en cada ramificación de la máquina de inferencia existen por cada módulo MIN en paralelo un módulo MAX menos en serie, es decir, si hay 4 módulos MIN en paralelo, hay entonces 3 módulos MAX en serie. Para todos los módulos MIN de la máquina de inferencia el tiempo de propagación es igual porque todos ellos se encuentran en paralelo. El problema sucede con los módulos MAX, pues los retardos de propagación de cada uno de ellos se suman porque se encuentran en serie. Aunque el impacto de este problema es considerable para un número muy grande de entradas y/o de conjuntos difusos, la consideración de diseño de FLCs de tipo MISO ayuda a reducir este problema, teniendo varios FLCs con diferentes máquinas de inferencia procesando datos diferentes. La Figura 46 muestra lo antes explicado:

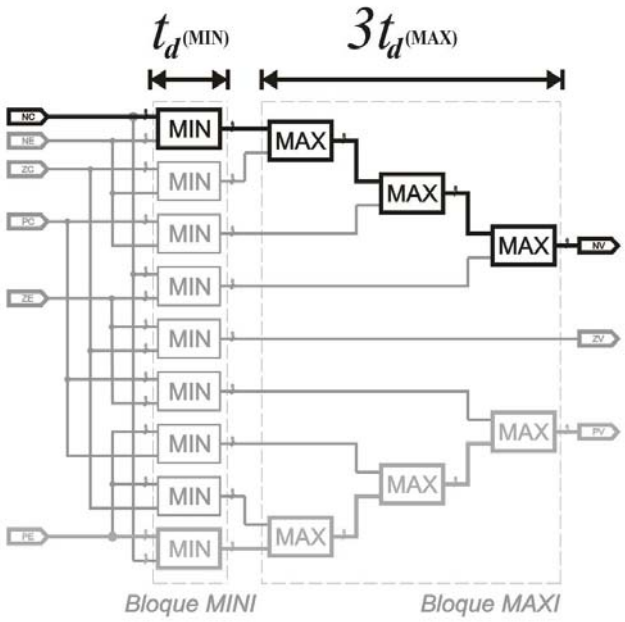

Figura 4. Retardo de propagación de la máquina de inferencia difusa.

Por lo tanto, el retardo de propagación de la máquina de inferencia difusa del FLC propuesto para el control del servomotor es el siguiente:

$$t_d(MAMDANI) = t_d(MIN) + 3t_d(MAX) \quad (22)$$

Finalmente, la última etapa, el desdifusificador es el más lento de todo el FLC, ya que posee uno o varios módulos de división y de multiplicación que se encuentran calculando el centroide todo el tiempo. Del mismo modo, también se muestran los resultados de ambos enfoques del desdifusificador.

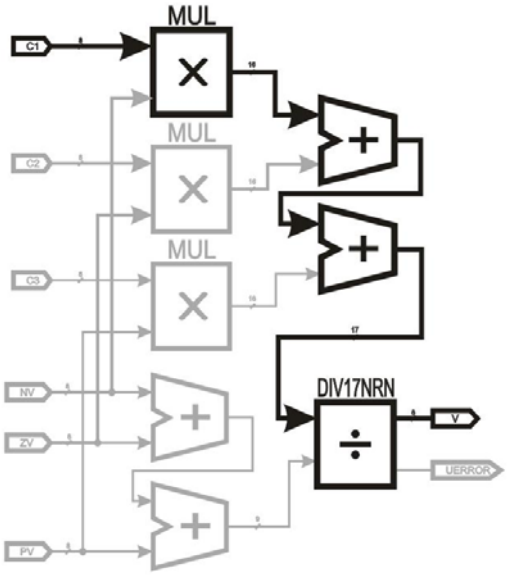

Figura 5. Ruta crítica del desdifusificador usando un solo módulo de división.

El elemento de división en este primer desdifusificador es más lento que los módulos de división usados en el resto del diseño, debido a que el número de bits es del doble. Aunque el módulo advierte al diseñador que el algoritmo de la división es de 17 bits, en realidad puede usarse el módulo DIV16NRN, si se restringe el número de funciones de membresía traslapadas a dos. Aún así, esta división sin restauración es más rápida que su contraparte con restauración de 16 bits. Como puede observarse en la Figura 47 la ruta crítica comprende la obtención de la multiplicación entre el centro del primer conjunto difuso de la salida y su valor de membresía obtenido de la inferencia difusa, pasando por la suma de los resultados de todas las multiplicaciones y por último realizando la división. Todas las operaciones de multiplicación se encuentran en paralelo por lo que sólo el retardo de propagación de una representa la de todas. Nótese que por cada consecuencia que recibe el desdifusificador existe una operación de suma menos,

tanto en la obtención del dividendo como en la obtención del divisor. El retardo de propagación del desdifusificador del FLC propuesto para el control del servomotor es el siguiente:

$$t_d(DEFUZZIFIER1) = t_d(MUL8N) + 2t_d(ADD) + t_d(DIV16NRN) \qquad (23)$$

El otro enfoque del desdifusificador implica el uso de un módulo de división de 8 bits por cada consecuencia, dividiendo el centro de la consecuencia entre la suma de todos los valores de membresía obtenidos de la inferencia difusa. La ruta crítica es la siguiente:

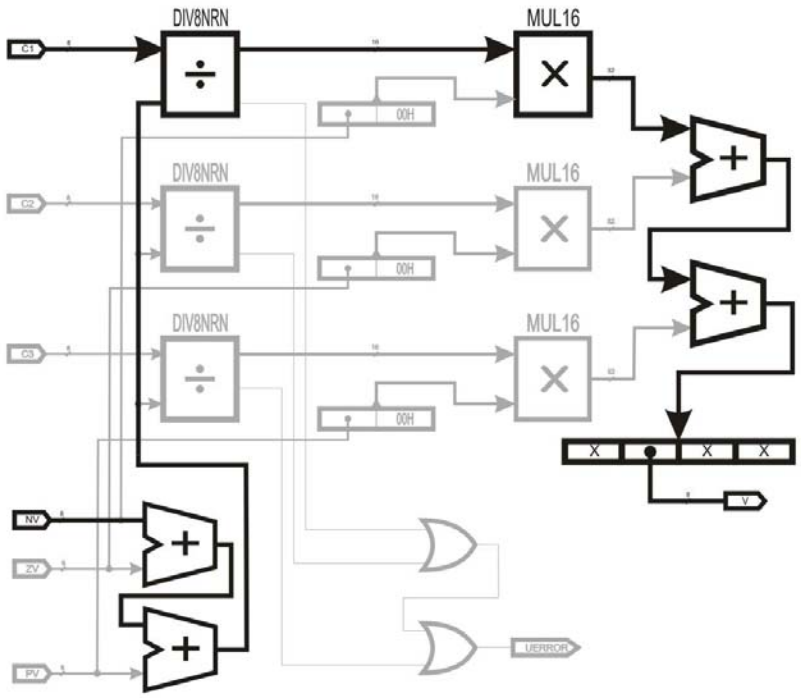

Figura 6. Ruta crítica del desdifusificador usando un módulo de división por cada consecuencia.

No importa qué cantidad de módulos DIV8NRN existan, porque todos ellos se encuentran en paralelo y su retardo de propagación no impacta de manera importante en el desempeño del desdifusificador, aunque la complejidad del diseño incrementa. La ruta crítica comprende la obtención del divisor de la división realizando la suma de todos los valores de membresía de las consecuencias, pasando por la realización de la división, la multiplicación y por último la suma de los resultados de todas las multiplicaciones. Por cada consecuencia que recibe el desdifusificador existe una operación de división; de igual

manera que el enfoque anterior, por cada consecuencia que recibe el desdifusificador existe una operación de suma menos, tanto en la obtención del divisor común que tienen los módulos de división como en la acumulación de los resultados de las multiplicaciones. Este método utiliza mayor cantidad de bits en las multiplicaciones pero ahorra bastante tiempo al utilizar la división de 8 bits en vez de 16 bits como en el enfoque anterior. La penalización más grande se encuentra en las multiplicaciones que tienen que ser de 16 bits en punto binario y cuyos resultados están dados en 32 bits en formato de punto binario. El retardo de propagación del desdifusificador alternativo del FLC propuesto para el control del servomotor es el siguiente:

$$t_d(DEFUZZIFIER2) = 2t_d(ADD8) + t_d(DIV8NRN) + t_d(MUL16N) + 2t_d(ADD32) \quad (24)$$

Este desdifusificador alternativo incrementa la velocidad de FLC en general, por lo que en el empeño de mejorar el rendimiento del desdifusificador se encuentra la mejora general del FLC. Asimismo, es importante tomar en cuenta a aquéllos algoritmos basados en el enfoque RTC que sean mucho más rápidos y confiables para la obtención de un mismo resultado.

Finalmente, el tiempo de retardo total del FLC propuesto con el primer desdifusificador para el control del servomotor es el siguiente:

$$t_d(FLC1) = t_d(FUZZIFER) + t_d(MAMDANI) + t_d(DEFUZZIFIER1) \quad (25)$$

Y el tiempo de retardo total del FLC propuesto con el desdifusificador alternativo para el control del servomotor es el siguiente:

$$t_d(FLC2) = t_d(FUZZIFER) + t_d(MAMDANI) + t_d(DEFUZZIFIER2) \quad (26)$$

A continuación se muestran los resultados de la simulación que se obtuvieron con las herramientas de Mentor Graphics Modelsim y de Mathworks MATLAB, con el fin de comparar los resultados interpretándolos de manera adecuada, según la metodología antes planteada.

Resultados de simulación en MATLAB

Un FLC en el Fuzzy Toolbox de MATLAB es denotado como *FIS* por sus siglas Fuzzy Inference System, o Sistema de Inferencia Difuso. Esta caja de herramientas diseñada por Roger Jang y Ned Gulley en 1995 para el programa MATLAB de la corporación The MathWorks [54], es una interfaz gráfica de usuario (GUI) que facilita la elaboración de sistemas lógico–difusos, mediante una colección de funciones disponibles en el ambiente de cómputo numérico de MATLAB, para integrarlos o combinarlos con herramientas de simulación como Simulink y/o programas en C elaborados por el mismo usuario [55].

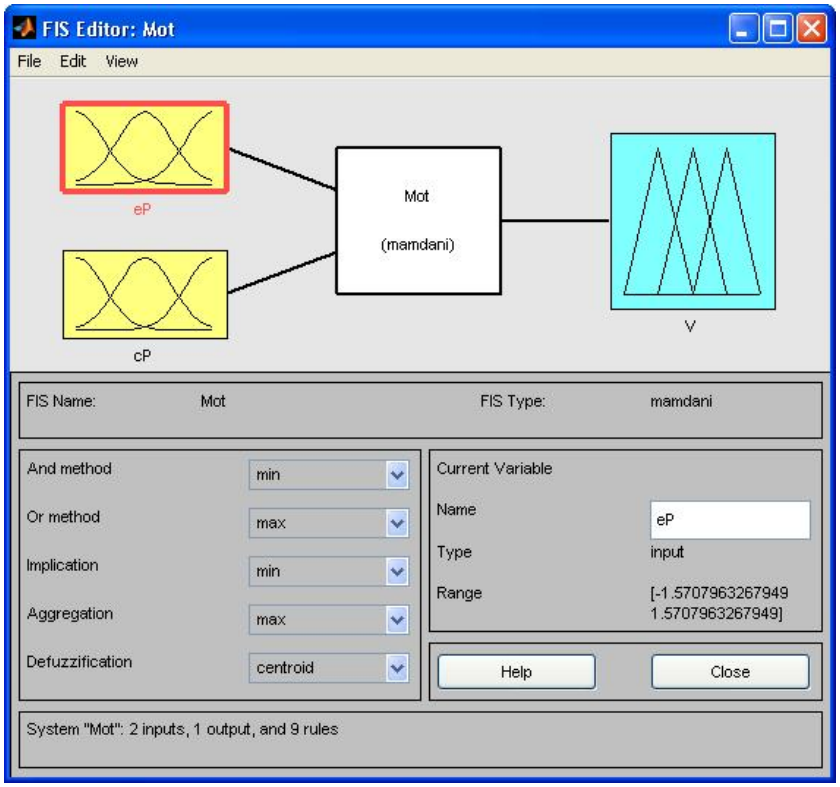

Figura 7. Diseño del FIS basado en el FLC propuesto para el control del servomotor usando la Interfaz gráfica del Fuzzy Toolbox de MATLAB.

Un FIS elaborado con esta herramienta posee una ventana que le permite al usuario definir el número de entradas y salidas que en nuestro caso es de 2 entradas por 1 salida, el tipo de desdifusificación que en nuestro caso es *centroide* y la máquina de inferencia que en nuestro caso es *mamdani*, entre otros parámetros de manera sencilla y rápida, como la que se muestra en la Figura 49. Cada uno de los módulos que corresponden a las variables de entrada y salida tienen una configuración que puede modificarse fácilmente al hacer doble clic alguno de ellos, como se muestra a continuación:

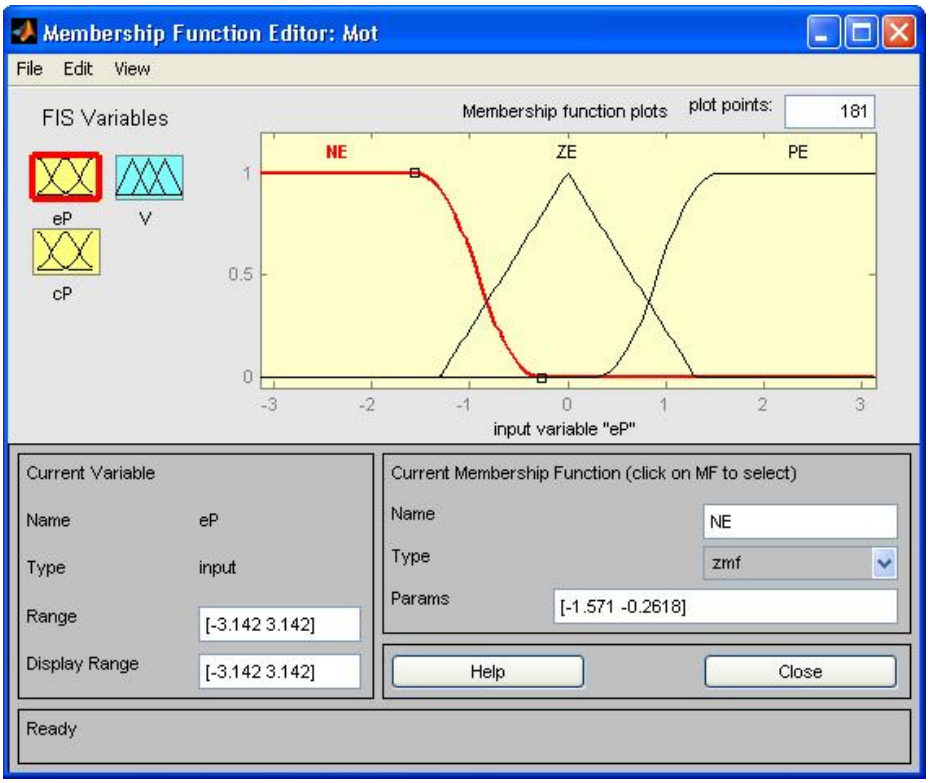

Figura 8. FIS: Funciones de membresía de las variables del FLC propuesto para el control del servomotor.

La facilidad de modificar las funciones de membresía es evidente, basta con hacer clic en un nodo y arrastrarlo hasta el lugar deseado dentro del intervalo de la gráfica para cada una de las variables; pero también es posible modificar sus parámetros de manera exacta modificando los valores del vector llamado *Params*. También es posible seleccionar la función de membresía para darle la forma al conjunto difuso modificando la selección de la lista desplegable llamada *Type* y asignarle un nombre *Name* para de cada una de las variables. La Figura 50 muestra la interfaz gráfica que utiliza el FIS para crear, agregar y modificar las funciones de membresía para cada una de las variables tanto de entrada como de salida.

La creación de las reglas de inferencia difusa es también sencilla. Basta con seleccionar el nombre asignado al conjunto difuso de determinada entrada y relacionarlo con el nombre del otro conjunto difuso de la otra variable y determinar el conjunto difuso de salida seleccionándolo. Con el botón *Add rule* se agrega una regla dependiendo de las selecciones realizadas antes de presionarlo. Entonces el conjunto de reglas de inferencia difusa del FIS para el control del servomotor queda de la siguiente manera:

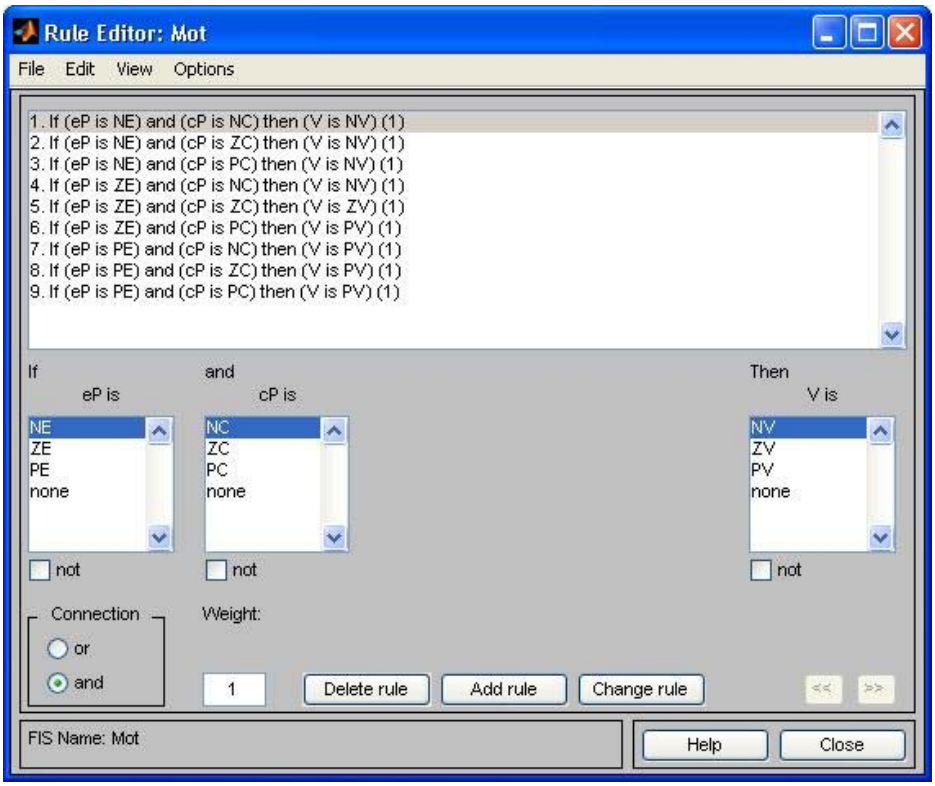

Figura 9. FIS: Conjunto de reglas de inferencia del FLC propuesto para el control del servomotor.

Todas las configuraciones del FIS quedan grabadas en un archivo que describe mediante el lenguaje de MATLAB, la configuración completa del FIS. Este archivo es suficiente y necesario para realizar simulaciones mediante Simulink. Como el FIS se ha llamada Mot, entonces el archivo se llama `Mot.fis`, que se incluye en el CD que viene adjunto con este trabajo y que se muestra a continuación:

```
[System]
Name='Mot'
Type='mamdani'
Version=2.0
NumInputs=2
NumOutputs=1
NumRules=9
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='eP'
Range=[-3.14159265358979 3.14159265358979]
NumMFs=3
MF1='NE':'zmf',[-1.5707963267949 -0.261799387799149]
MF2='ZE':'trimf',[-1.30899693899575 0 1.30899693899575]
MF3='PE':'smf',[0.261799387799149 1.5707963267949]

[Input2]
Name='cP'
Range=[-3.14159265358979 3.14159265358979]
NumMFs=3
MF1='NC':'zmf',[-0.523598775598299 -0.0872664625997165]
MF2='ZC':'trimf',[-0.436332312998582 0 0.436332312998582]
MF3='PC':'smf',[0.0872664625997165 0.523598775598299]

[Output1]
Name='V'
Range=[-8.8 8.8]
NumMFs=3
MF1='NV':'zmf',[-5 -1]
MF2='ZV':'trimf',[-4 0 4]
MF3='PV':'smf',[1 5]

[Rules]
1 1, 1 (1) : 1
1 2, 1 (1) : 1
1 3, 1 (1) : 1
2 1, 1 (1) : 1
2 2, 2 (1) : 1
2 3, 3 (1) : 1
3 1, 3 (1) : 1
3 2, 3 (1) : 1
3 3, 3 (1) : 1
```

Código 1. Mot.fis: Configuración del sistema difuso FIS para el control del servomotor usado en MATLAB.

Hasta este momento todos los parámetros de configuración del FIS deben estar definidos para obtener el FIS deseado, para poder realizar una simulación que pueda compararse con el FLC diseñado en VHDL. La simulación puede verse claramente desde dos puntos de vista: mediante la *Vista de Reglas*, modificando valores en la entrada, observando las reglas que se activan y obteniendo el comportamiento a la salida; o bien mediante la *Vista de Superficie de Control*. Estas dos vistas de simulación se encuentran en el menú View/Rules y en View/Surface, respectivamente.

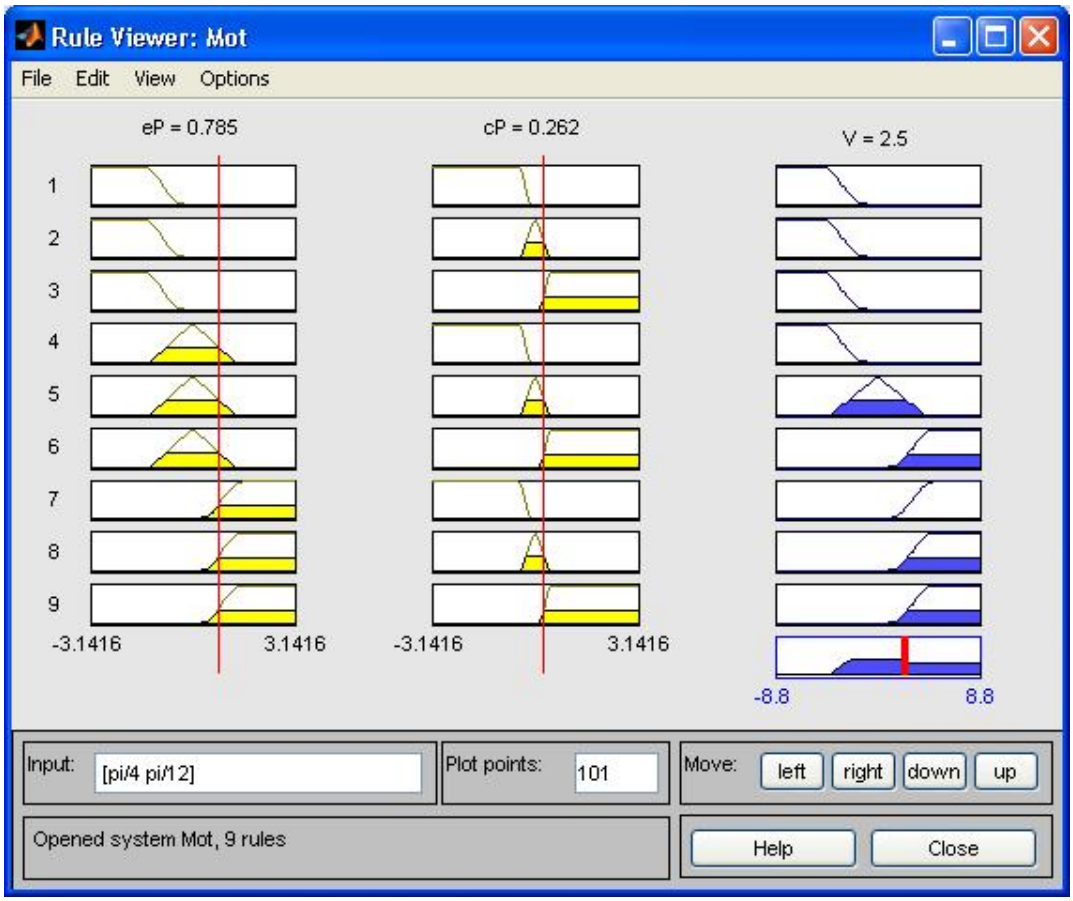

Figura 10. FIS: Vista de reglas del FLC propuesto para el control del servomotor.

La vista de reglas de la Figura 52 permite la modificación de los valores de entrada arrastrando un cursor (línea roja) sobre las gráficas de cada una de ellas, o bien modificando los valores del vector de entrada *Input*. Después de cada modificación al vector de entrada se calcula el valor que le corresponde a la salida del FIS, pasando por las 3 etapas del control difuso para obtener el resultado concreto.

Cada uno de estos resultados tiene un valor equivalente dentro del universo de discurso escalado del FLC propuesto para el control del servomotor, como en la Figura 27.

Un valor mínimo de –5 volts corresponde a un valor hexadecimal en hardware de 00. Un valor máximo de 5 volts corresponde a un valor hexadecimal en hardware de FF. El cero corresponde a un valor hexadecimal de 7F. Si se quiere tener una precisión de 0.1 volts dentro de un universo de discurso de 8 bits es necesario reajustar el universo de discurso de la Figura 27 a los valores que se pueden representar. Si a 5 volts le corresponde la mitad de la cantidad que se puede representar con 8 bits (127 valores) excluyendo al cero entonces cada:

$$\frac{0.1v}{5v}(127) = 2.54$$

valores del universo de discurso equivalen a recorrer 0.1 volts. Sin embargo, 2.54 no puede representarse con esta cantidad de bits, por lo que lo más adecuado es redondear este valor a uno que permita que pueda representarse los 5 volts, ya sea hacia arriba a 3 o hacia abajo a 2. Si 0.1 volts equivale a 3 valores del universo de discurso, entonces 1 volt tiene 30 valores y por lo tanto 5 volts equivalen a $5 \times 30 = 150$ valores. Esto no es adecuado debido a que en 127 valores deben ser suficientes para representar a los 5 volts y $150 > 127$. Esto nos lleva a redondear este valor hacia abajo, es decir a 2. Y Si 0.1 volts equivale a 2 valores del universo de discurso, entonces 1 volt tiene 20 valores y por lo tanto 5 volts equivalen a $5 \times 20 = 100$ valores. Por lo tanto el valor que corresponde a los 5 volts dentro del universo de discurso de la variable de salida es de 100, o 64 en hexadecimal.

Para que exista simetría en el universo de discurso de la variable de salida *V*, voltaje, deben existir otros 100 valores por debajo de 127 para representar –5 volts. Entonces el universo de discurso de la variable de salida *V* debe modificarse de la siguiente manera:

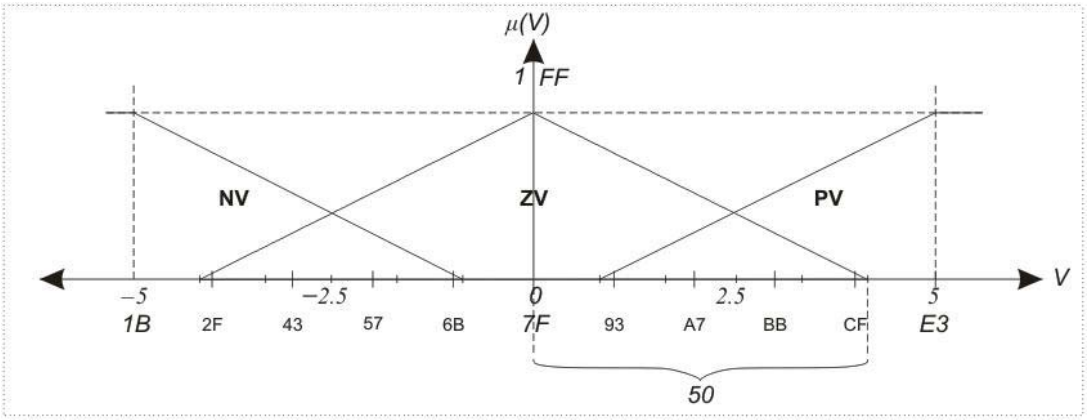

Figura 11. Ajuste del universo de discurso de la variable de salida del FLC para el control del servomotor.

Entonces la Tabla 3 es modificada de la siguiente manera:

Tabla 2. Modificación a los parámetros de los conjuntos difusos del FLC para la variable de salida V.

Variable	N (HEX)		Z (HEX)		P (HEX)	
	c	a	c	a	c	a
eP (E)	3F	32	7F	32	BE	32
cP (C)	6A	0F	7F	0F	94	0F
V (V)	1B	50	7F	50	E3	50

Como se dijo anteriormente, los valores de apertura de las funciones de membresía para la variable de salida *V*, no se utilizan para el cálculo del valor de salida, pues el método de desdifusificación de centroide discrimina la forma de estas funciones.

Usando esta vista de reglas del FIS se realiza una tabla que contiene los 25 casos más significativos que pueden ocurrir en las entradas del FLC ya implementado en hardware. También se expresa su valor correspondiente de la salida, en hexadecimal, escalado según la Figura 53 y de esta misma manera los valores equivalentes correspondientes, en hexadecimal, de las entradas mostradas en la Figura 26. Entonces los valores de la Tabla 6 de cada columna corresponden a los valores representativos para el FIS y para el FLC en hexadecimal en cada variable. Estos valores son usados en la sección Resultado de Simulación en Modelsim para verificar el funcionamiento del FLC implementado en hardware. La última columna con los valores correspondientes al FLC son aquéllos valores que deben coincidir con aquéllos que se observen en la simulación.

Tabla 3. Resultados del FIS para el control del servomotor utilizando los valores de entrada más representativos.

Caso	eP		AND	cP		THEN	V	
	FIS	FLC		FIS	FLC		FIS	FLC
1	$-\frac{\pi}{2}$	3F	∧	$-\frac{\pi}{6}$	6A	→	-5.89	1B
2	$-\frac{\pi}{2}$	3F	∧	0	7F	→	-5.89	1B
3	$-\frac{\pi}{2}$	3F	∧	$\frac{\pi}{6}$	94	→	-5.89	1B
4	0	7F	∧	$-\frac{\pi}{6}$	6A	→	-5.89	1B
5	0	7F	∧	0	7F	→	0	7F
6	0	7F	∧	$\frac{\pi}{6}$	94	→	5.89	E3
7	$\frac{\pi}{2}$	BE	∧	$-\frac{\pi}{6}$	6A	→	5.89	E3
8	$\frac{\pi}{2}$	BE	∧	0	7F	→	5.89	E3
9	$\frac{\pi}{2}$	BE	∧	$\frac{\pi}{6}$	94	→	5.89	E3
10	$-\frac{\pi}{4}$	60	∧	$-\frac{\pi}{12}$	75	→	-2.5	4D
11	$-\frac{\pi}{4}$	60	∧	$\frac{\pi}{12}$	89	→	0	7F
12	$\frac{\pi}{4}$	9E	∧	$-\frac{\pi}{12}$	75	→	0	7F
13	$\frac{\pi}{4}$	9E	∧	$\frac{\pi}{12}$	89	→	2.5	B1
14	$-\frac{\pi}{4}$	60	∧	0	7F	→	-2.5	4D
15	0	7F	∧	$-\frac{\pi}{12}$	75	→	-2.5	4D
16	$\frac{\pi}{4}$	9E	∧	0	7F	→	2.5	B1
17	0	7F	∧	$\frac{\pi}{12}$	89	→	2.5	B1
18	$-\frac{\pi}{4}$	60	∧	$-\frac{\pi}{6}$	6A	→	-5.53	1B
19	$-\frac{\pi}{4}$	60	∧	$\frac{\pi}{6}$	94	→	0.59	8B
20	$\frac{\pi}{4}$	9E	∧	$-\frac{\pi}{6}$	6A	→	-0.59	73
21	$\frac{\pi}{4}$	9E	∧	$\frac{\pi}{6}$	94	→	5.53	E3
22	$-\frac{\pi}{2}$	3F	∧	$-\frac{\pi}{12}$	75	→	-5.53	1B
23	$-\frac{\pi}{2}$	3F	∧	$\frac{\pi}{12}$	89	→	-5.53	1B
24	$\frac{\pi}{2}$	BE	∧	$-\frac{\pi}{12}$	75	→	5.53	E3
25	$\frac{\pi}{2}$	BE	∧	$\frac{\pi}{12}$	89	→	5.53	E3

Aquéllos valores que sobrepasaban los voltajes extremos no pueden representarse en 8 bits, por lo que son interpretados como el número máximo permitido, es decir que -5.89 o -5.53 volts en realidad debe ser interpretado como -5 volts y el valor hexadecimal que le corresponde en su universo de discurso es 1B.

Los valores de entrada de cada uno de los casos son representativos pues provocan que el FIS active sus reglas en diferentes circunstancias, de esta manera puede verificarse el correcto funcionamiento del mismo. Los casos 1 al 9 representan a las circunstancias del conjunto de reglas de inferencia difusas, en donde no existe ningún traslapamiento entre las funciones de membresía de cada una de las variables de entrada, por lo tanto sólo existen a lo mucho 2 reglas activas. Los casos 10 al 13 comprenden a las circunstancias donde existen traslapamiento entre 2 funciones de membresía en las dos entradas, es decir, en donde existen 4 reglas activas que contribuyen a la salida. Los casos 14 al 17 comprenden aquéllas circunstancias en donde al menos una variable se encuentra sólo en su conjunto difuso central y la otra se encuentra con algún traslapamiento y viceversa. Los casos 18 al 25 comprenden a aquéllas circunstancias en que al menos una variable se encuentra sólo en algún conjunto difuso de los extremos y la otra se encuentra con algún traslapamiento.

Los valores de entrada de esta tabla son los valores de prueba que se ingresan a las entradas del FLC implementado en FPGA y validado en la siguiente sección por los resultados de simulación mediante Modelsim. Los valores de la última columna de la Tabla 6 son comparados con los resultados de la simulación para cada uno de los casos.

La vista de superficie de control de la Figura 54 ofrece un panorama más general del comportamiento del FIS a diferentes situaciones que el programa elije basándose en los universos de discurso de todas las variables. Sin embargo, para comprobar el funcionamiento del FLC en base al FIS es mucho más útil la vista de reglas.

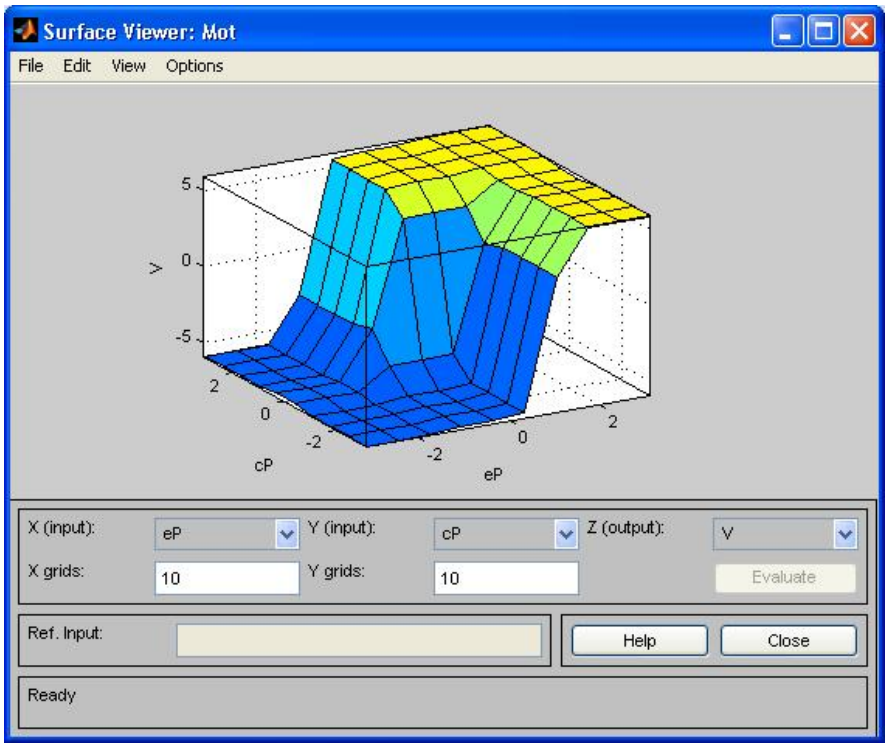

Figura 12. FIS: Vista de superficie de control del FLC propuesto para el control del servomotor.

Finalmente, el funcionamiento del FIS puede comprobarse utilizando una herramienta muy poderosa de MATLAB: El Simulink. Por medio del Simulink es posible simular cualquier sistema de control difuso haciendo uso del archivo de configuración FIS que se produce mediante el Fuzzy Toolbox mencionado anteriormente.

Ahora, supóngase que sí se conoce la función de transferencia de la planta que en este caso es el servomotor y que es la siguiente:

$$F(s) = \frac{0.5}{s^3 + 13.6667s^2 + 23.348s}$$

Entonces se requiere un FLC que sea capaz de mover al rotor a la posición deseada, según la Figura 55:

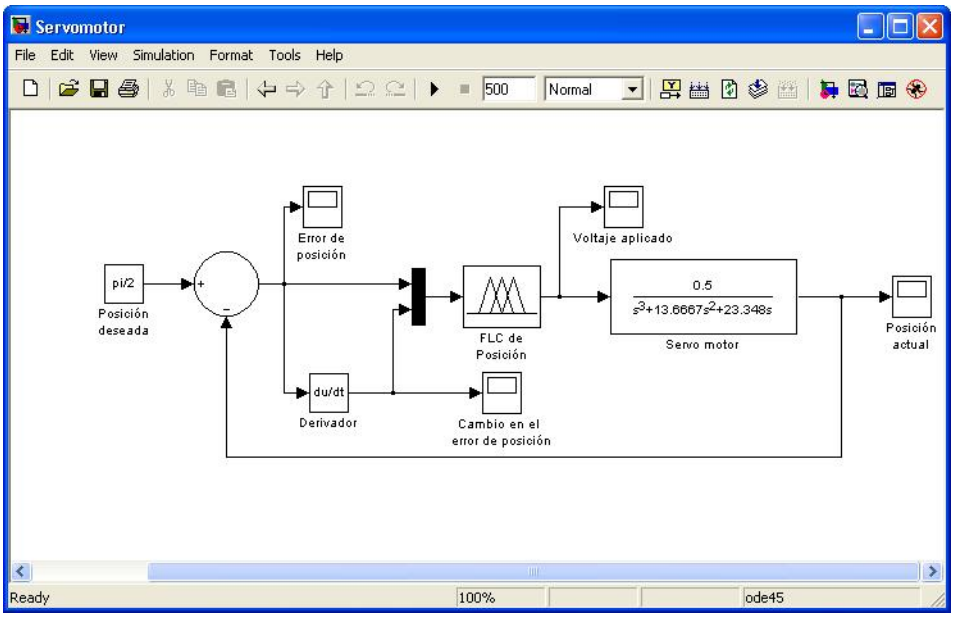

Figura 13. Sistema de control difuso para el servomotor usando SIMULINK.

En la Figura 55 se muestra un sistema de control de lazo cerrado para un servomotor, cuya función de transferencia está definida. El lazo de cerrado permite conocer el valor de posición actual y mediante el restador puede conocerse la diferencia entre el valor de posición actual y el valor de posición deseado, es decir el error de posición. Este error de posición es una de las variables de entrada del controlador. La otra variable de entrada es el cambio en el error de posición que se obtiene derivando respecto al tiempo al error. El controlador aplica un voltaje a la entrada del proceso y este responde a su salida con un valor de posición en radianes.

Para que el FIS diseñado anteriormente pueda funcionar, es necesario exportarlo hacia el Workspace desde el Fuzzy Toolbox como una variable de entorno con algún nombre, por ejemplo `Mot`.

Si la posición que se desea es $\frac{\pi}{2} = 1.5707$ radianes, entonces el FIS debe ser capaz de mover al motor a la posición deseada aplicándole el voltaje necesario. Esto puede observarse claramente en las gráficas que arrojan los medidores después de 500 iteraciones en la figura siguiente:

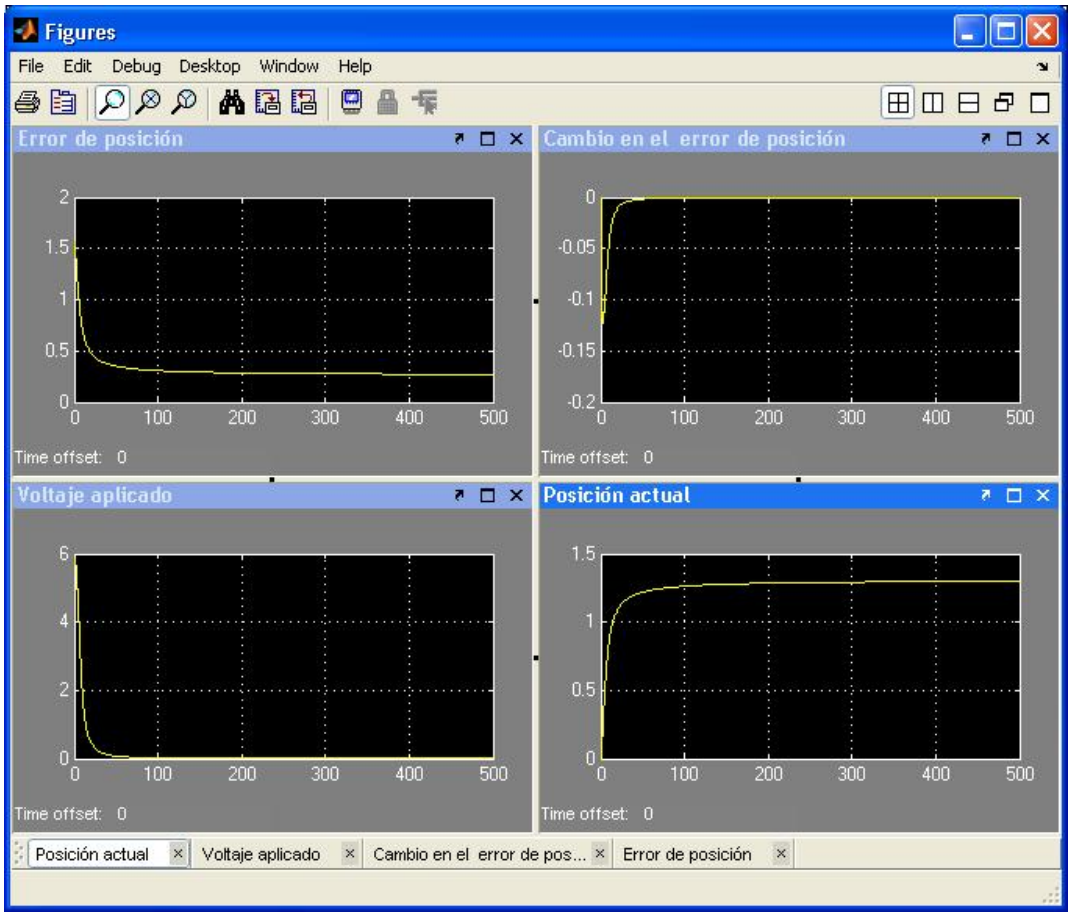

Figura 14. Resultados del FIS en un sistema de control de lazo cerrado elaborado en Simulink.

En la Figura 56 puede observarse que la gráfica de la posición actual está por debajo de la posición deseada, y el motor se estabiliza en 1.2557 radianes. Esto no quiere decir que el FIS este mal diseñado o no sea eficiente. Esto se debe realmente a que el número de funciones membresía por entrada es bajo y esto limita los grados de libertad (número de reglas de inferencia difusa) que tiene el FIS para poder actuar adecuadamente. Sin embargo es una buena aproximación al valor deseado y por lo tanto es válido el controlador.

A continuación se compara el funcionamiento del FIS con el funcionamiento del FLC implementado en FPGA con ayuda de VHDL. Esta comparación es validada gracias a los resultados de la simulación del hardware mediante Modelsim y a la implementación en FPGA de este diseño.

Resultados de simulación en Modelsim

Modelsim es un programa que proporciona un ambiente amigable de simulación y depuración de diseño de circuitos lógicos digitales usando lenguajes de descripción de hardware para ASIC y FPGA. Además existen muchas versiones dedicadas a un tipo de fabricante específico, como en nuestro caso la versión hecha para diseño digital en los FPGAs de Xilinx.

En esta sección se pretende verificar el funcionamiento del diseño del FLC propuesto para el control de un servomotor, con la única intención de mostrar al lector la facilidad de construir FLCs de manera rápida y sencilla para cualquier aplicación haciendo uso de la lógica combinatoria y aprovechando el paralelismo que exhibe. El diseño es elaborado en VHDL en su totalidad para implementarlo en un FPGA.

La Tabla 6 contiene los valores que se obtienen de la simulación mediante el Fuzzy Toolbox de MATLAB y también contiene los valores de la variable de la salida correctos, de acuerdo al universo de discurso ajustado de la variable de salida mostrado en la Figura 53.

Con las herramientas de Xilinx es posible crear un archivo de prueba *Testbench Waveform* que nos permite asignar valores a las variables de entrada del diseño cada determinado tiempo y nos permite observar el resultado de las variables de salida mediante el simulador que el mismo Xilinx pone a disposición de los usuarios, tal como se muestra en la Figura 57. Sin embargo, con este mismo archivo es posible ver más a detalle el funcionamiento y realizar una depuración del diseño cuando se hace uso del Modelsim. Para realizar una simulación en cualquier simulador es necesario por lo menos realizar el proceso de Síntesis o *Synthesize*, como aparece en Xilinx ISE, aunque esto depende del tipo de simulación que se requiera realizar.

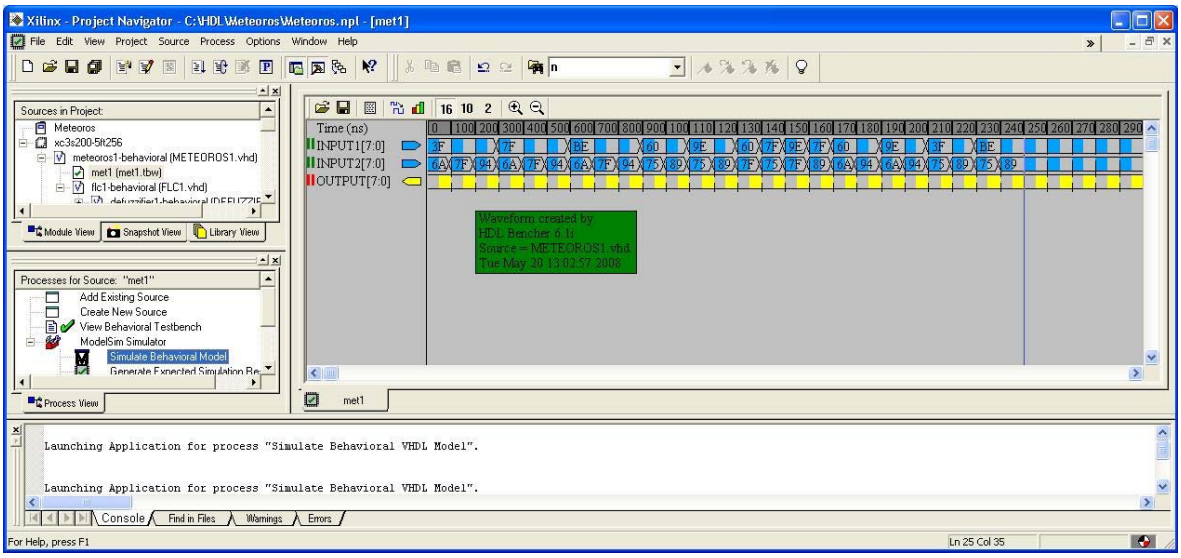

Figura 15. Creación del archivo de prueba (testbench waveform) del FLC propuesto para el control del servomotor.

La Figura 58 muestra los resultados de simulación del módulo `FLC1.vhd` integrado en otro archivo llamado `METEOROS1.vhd`, ambos disponibles en el CD que viene adjunto este trabajo. El módulo METEOROS1 sólo sirve para contener al módulo FLC1 y para asignarle valores a los parámetros de las funciones de membresía de las variables de entrada, ya que el paso de difusificación depende de estos valores; y asignarle valores a los centros de las funciones de membresía de la variable de salida, ya que de estos valores depende el funcionamiento del paso de la desdifusificación. Estos valores constantes pueden cambiarse por registros, los cuales le dan la capacidad de adaptación, es decir, la manera como el FLC definirá sus etiquetas lingüísticas. El módulo FLC1 tiene la capacidad de modificar en tiempo real los parámetros de las funciones de membresía; sólo se necesita que en el módulo METEOROS1 se especifique un registro para cada uno de estos valores para así modificarlo si es necesario.

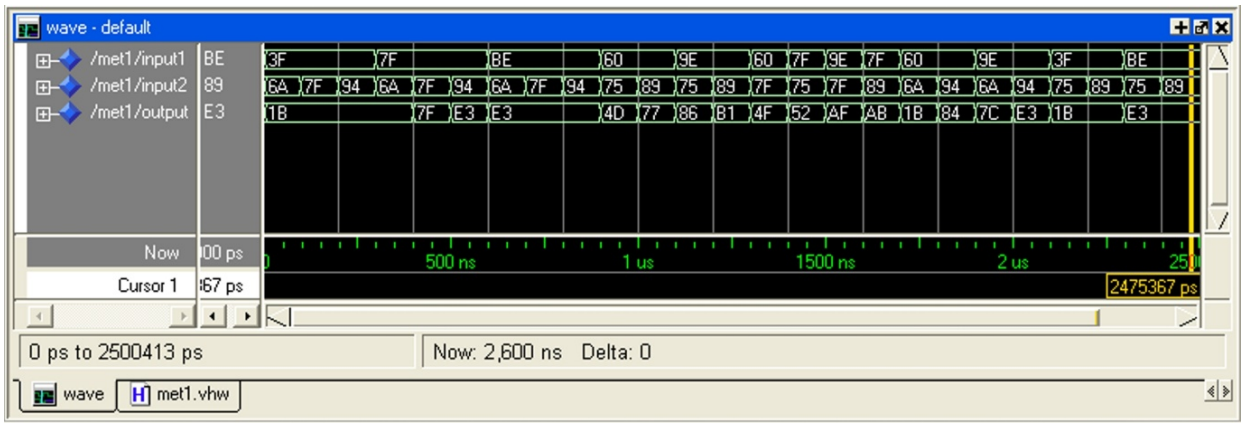

Figura 16. Resultados obtenidos mediante la simulación en Modelsim del FLC propuesto para el control del servomotor, usando los 25 valores más representativos a las entradas.

Los valores en el puerto OUTPUT contienen los resultados después de aplicarle los valores en hexadecimal de las variables de entrada de la Tabla 6. Estos valores son comparados con los obtenidos en la Figura 58, en la siguiente tabla:

Tabla 4. Comparación de los resultados deseados con los resultados obtenidos.

Caso	eP	cP	V deseado	V obtenido	Error
1	3F	6A	1B	**1B**	0
2	3F	7F	1B	**1B**	0
3	3F	94	1B	**1B**	0
4	7F	6A	1B	**1B**	0
5	7F	7F	7F	**7F**	0
6	7F	94	E3	**E3**	0
7	BE	6A	E3	**E3**	0
8	BE	7F	E3	**E3**	0
9	BE	94	E3	**E3**	0
10	60	75	4D	**4D**	0
11	60	89	7F	**77**	–8
12	9E	75	7F	**86**	+7
13	9E	89	B1	**B1**	0
14	60	7F	4D	**4F**	+2
15	7F	75	4D	**52**	+5
16	9E	7F	B1	**AF**	–2
17	7F	89	B1	**AB**	–6

18	60	6A	1B	**1B**	0
19	60	94	8B	**84**	–7
20	9E	6A	73	**7C**	+9
21	9E	94	E3	**E3**	0
22	3F	75	1B	**1B**	0
23	3F	89	1B	**1B**	0
24	BE	75	E3	**E3**	0
25	BE	89	E3	**E3**	0

Los resultados obtenidos de la Tabla 6 a partir de los 25 casos, son resumidos en la Tabla 7. Como puede observarse existen ciertos errores debido al uso de la división de enteros, es decir que los operandos son enteros y el resultado también es entero, lo cual provoca que en la división existan truncamientos de valores, sobre todo en la etapa de desdifusificación por centroide. Si el FLC trabajara con valores en punto binario entonces podrían apreciarse mejores resultados, aunque la mayoría de estos resultados son muy cercanos a los deseados. También el resultado suele ser impreciso debido al número de funciones de membresía utilizadas por cada una de las entradas, ya que el número de reglas disponibles depende directamente de esto y por ende limita la precisión del FLC.

Si modificamos los valores de las funciones de membresía de la variable cP, por ejemplo ampliando los triángulos con los siguientes valores:

Tabla 5. Modificación a los parámetros de los conjuntos difusos del FLC para la variable de entrada cP.

Variable	N (HEX)		Z (HEX)		P (HEX)	
	c	a	c	a	c	a
***eP* (E)**	3F	32	7F	32	BE	32
***cP* (C)**	3F	32	7F	32	BE	32
***V* (V)**	1B	50	7F	50	E3	50

los resultados a la salida del FLC cambian de la siguiente manera:

Tabla 6. Comparación de los resultados deseados con los resultados obtenidos, modificando los conjuntos difusos de la variable de entrada cP.

Caso	eP	cP	V deseado	V obtenido	Error
1	3F	3F	1B	**1B**	0
2	3F	7F	1B	**1B**	0
3	3F	BE	1B	**1B**	0
4	7F	3F	1B	**1B**	0
5	7F	7F	7F	**7F**	0
6	7F	BE	E3	**E3**	0
7	BE	3F	E3	**E3**	0
8	BE	7F	E3	**E3**	0
9	BE	BE	E3	**E3**	0
10	60	60	4D	**4F**	+2
11	60	9E	7F	**80**	+1
12	9E	60	7F	**80**	+1
13	9E	9E	B1	**AF**	–2
14	60	7F	4D	**4F**	+2
15	7F	60	4D	**4F**	+2
16	9E	7F	B1	**AF**	–2
17	7F	9E	B1	**AF**	–2
18	60	3F	1B	**1B**	0
19	60	BE	8B	**84**	–7
20	9E	3F	73	**7C**	+9
21	9E	BE	E3	**E3**	0
22	3F	60	1B	**1B**	0
23	3F	9E	1B	**1B**	0
24	BE	60	E3	**E3**	0
25	BE	9E	E3	**E3**	0

Se puede observar en la Tabla 9 que al realizar el movimiento de los triángulos, de la variable *cP*, se pudo reducir el error que había en los resultados de salida del FLC mostrado en la Tabla 7. Esta característica le permitiría al FLC realizar una *adaptación*, apoyándose de un sistema inteligente. La adaptación es una manera de extender o reducir el intervalo en el que el FLC decide el momento en el que realizará una decisión al inferir una consecuencia, sin intervención del humano. Esto puede ser bastante útil, pues de esta manera se puede forzar al FLC a comportarse de determinada manera, colocando a un

sistema computacional (inteligente) supervisor que cambie los valores de estos registros cuando la acción de control no sea la adecuada para controlar al proceso.

Tabla 7. Equivalencias de resultados en los experimentos en el FLC.

Caso	eP	AND	cP	THEN	V FIS	V FLC	FLC (adaptado)
1	$-\frac{\pi}{2}$	∧	$-\frac{\pi}{6}$	⟶	-5.89	-5	-5
2	$-\frac{\pi}{2}$	∧	0	⟶	-5.89	-5	-5
3	$-\frac{\pi}{2}$	∧	$\frac{\pi}{6}$	⟶	-5.89	-5	-5
4	0	∧	$-\frac{\pi}{6}$	⟶	-5.89	-5	-5
5	0	∧	0	⟶	0	0	0
6	0	∧	$\frac{\pi}{6}$	⟶	5.89	5	5
7	$\frac{\pi}{2}$	∧	$-\frac{\pi}{6}$	⟶	5.89	5	5
8	$\frac{\pi}{2}$	∧	0	⟶	5.89	5	5
9	$\frac{\pi}{2}$	∧	$\frac{\pi}{6}$	⟶	5.89	5	5
10	$-\frac{\pi}{4}$	∧	$-\frac{\pi}{12}$	⟶	-2.5	-2.5	-2.4
11	$-\frac{\pi}{4}$	∧	$\frac{\pi}{12}$	⟶	0	-0.4	0
12	$\frac{\pi}{4}$	∧	$-\frac{\pi}{12}$	⟶	0	0.3	0
13	$\frac{\pi}{4}$	∧	$\frac{\pi}{12}$	⟶	2.5	2.5	2.4
14	$-\frac{\pi}{4}$	∧	0	⟶	-2.5	-2.4	-2.4
15	0	∧	$-\frac{\pi}{12}$	⟶	-2.5	-2.3	-2.4
16	$\frac{\pi}{4}$	∧	0	⟶	2.5	2.4	2.4
17	0	∧	$\frac{\pi}{12}$	⟶	2.5	2.2	2.4
18	$-\frac{\pi}{4}$	∧	$-\frac{\pi}{6}$	⟶	-5.53	-5	-5
19	$-\frac{\pi}{4}$	∧	$\frac{\pi}{6}$	⟶	0.59	0.2	0.2
20	$\frac{\pi}{4}$	∧	$-\frac{\pi}{6}$	⟶	-0.59	-0.2	-0.2
21	$\frac{\pi}{4}$	∧	$\frac{\pi}{6}$	⟶	5.53	5	5
22	$-\frac{\pi}{2}$	∧	$-\frac{\pi}{12}$	⟶	-5.53	-5	-5
23	$-\frac{\pi}{2}$	∧	$\frac{\pi}{12}$	⟶	-5.53	-5	-5
24	$\frac{\pi}{2}$	∧	$-\frac{\pi}{12}$	⟶	5.53	5	5
25	$\frac{\pi}{2}$	∧	$\frac{\pi}{12}$	⟶	5.53	5	5

La Tabla 10 muestra, finalmente, el resumen de todos los experimentos y la equivalencia entre los valores hexadecimales y los valores de voltaje correspondientes a la salida, de cada uno de los experimentos realizados en el FIS, el FLC y el FLC adaptado (al que se le modificaron las funciones de membresía de la variable *cP*):

Finalmente el diseño integrado en el archivo METEOROS.vhd fue bajado al FPGA mediante el proceso *Generar Archivo de Programación* del ISE, con ayuda del programa llamado PACE, incluido dentro de las herramientas del ISE, para asignar los puertos del diseño a los pines del FPGA; y con ayuda de otro programa del ISE llamado iMPACT, para bajar el archivo de configuración al FPGA. Los resultados obtenidos fueron exactamente los mismos que los obtenidos tras la simulación del diseño con ayuda de Modelsim.

Con estos resultados puede confirmarse que es posible realizar cualquier FLC haciendo uso de los módulos combinatorios presentados en este trabajo y obtener resultados satisfactorios dependiendo del seguimiento de la metodología mostrada en este trabajo. En un FPGA es posible integrar cualquier diseño digital y ésta ventaja proporciona la facilidad de manipular cualquier variable de entrada en un universo de discurso adecuado, tal como se hace para el control del servomotor, en donde los valores de las variables de entrada son extendidos en un nuevo intervalo de trabajo para que el FLC funcione de acuerdo a su contexto y entregue valores, en ese mismo contexto, que pueden ser interpretados como comandos que realicen una acción de control determinada. Por ejemplo, un Convertidor Digital a Analógico puede interpretar la palabra digital de salida como un cambio en el voltaje aplicado a las terminales del servomotor, así cuando la salida sea 1B, el voltaje que debe aplicar al servomotor es –5 volts.

Conclusiones 5

Se diseñó una arquitectura de un controlador lógico difuso, usando módulos aritméticos básicos implementados con lógica combinatoria. Para llevar a cabo este propósito, se le proporcionó al diseñador una metodología de diseño, que consiste de ocho etapas básicas, donde paso a paso se diseñó el FLC, tomando en cuenta todas sus consideraciones y usando una aplicación como caso de estudio (control de un servomotor) con el fin de mostrar al lector la facilidad de diseño y la posibilidad de ahorrar tiempo de diseño. Derivado de lo anterior, se desarrollaron los módulos en VHDL para implementarlos en un FPGA y con ayuda de Modelsim y MATLAB, se simuló y verificó el correcto funcionamiento tanto de los módulos desarrollados como del FLC en general, obteniéndose resultados que compiten con los desarrollos propuestos hasta la fecha.

Con el análisis realizado a esta propuesta, se concluye que el paralelismo que existe en los módulos que constituyen al FLC, favoreció al crecimiento del rendimiento respecto a los FLCs actualmente reportados. Con lo que esta arquitectura tiene la característica de crecer de forma modular, adaptándose a las necesidades del diseñador sin presentar mayor dificultad a la hora de escalar el sistema. También se demostró que el FLC tiene capacidades de adaptación al tener registros que se pueden modificar en línea, siendo estos los parámetros de las funciones de membresía, con lo cual se obtuvo mayor precisión en los resultados.

Derivado de los experimentos y del análisis de los resultados del FLC propuesto, se demostró que los circuitos lógicos combinatorios son una opción práctica y factible para el diseño de FLCs obteniéndose resultados comparables a los mejores FLCs actualmente reportados, beneficiándose de la tecnología en FPGA actual.

Referencias

[1] Togai M.; Watanabe H.; *"Expert system on a chip: An engine for real–time approximate reasoning;"* IEEE Expert Syst. Mag., 1986, pp. 55–62, Volume 1.

[2] Miki, T.; Yamakawa, T.; *"Fuzzy inference on an analog fuzzy chip;"* Micro, IEEE, August 1995, pp. 8–18, Volume 15, Issue 4.

[3] Watanabe, H.; Dettloff, W.D.; Yount, K.E.; *"A VLSI Fuzzy Logic Controller with Reconfigurable, Cascadable Architecture;"* Solid–State Circuits, IEEE Journal, April 1990, pp. 376–382, Volume 25, Issue 2.

[4] Hung, D.; Zajac, W.; *"Implementing a fuzzy inference engine using FPGA;"* ASIC Conference and Exhibit, 1993. Proceedings, Sixth Annual IEEE International, 27 September–1 October 1993, pp. 349–352.

[5] Hung, D.L.; *"Custom design of a hardware fuzzy logic controller;"* Fuzzy Systems. IEEE World Congress on Computational Intelligence, Proceedings of the Third IEEE Conference, 26–29 June 1994, pp. 1781–1785, Volume3.

[6] Vasantha Rani, S.P.J.; Kanagasabapathy, P.; Sathish Kumar, A.; *"Digital Fuzzy Logic Controller using VHDL;"* INDICON, 2005 Annual IEEE, 11–13 December 2005, pp. 463–466.

[7] Singh, S.; Rattan, K.S.; *"Implementation of a fuzzy logic controller on an FPGA using VHDL;"* Fuzzy Information Processing Society, 2003. NAFIPS 2003. 22nd International Conference of the North American 24–26 July 2003, pp. 110–115.

[8] Masmoudi, N.; Hachicha, M.; Kamoun, L.; *"Hardware Design of Programmable Fuzzy Controller on FPGA;"* Fuzzy Systems Conference Proceedings, 1999. FUZZ–IEEE '99. 1999 IEEE International, 22–25 August 1999, pp. 1675–1679, Volume3.

[9] Masmoudi, M.S.; Insop Song; Karray, F.; Masmoudi, M.; Derbel, N.; *"Hardware/Software Approach for FPGA Implementation of a Fuzzy Logic Controller;"* Design and Test of Integrated Systems in Nanoscale Technology, 2006. DTIS 2006. International Conference on 2006, pp. 419–423.

[10] Manzoul, M.A.; Jayabharathi, D.; *"FPGA for fuzzy controllers;"* Systems, Man and Cybernetics, IEEE Transactions, 1995, pp. 213–216, Volume 25, Issue 1.

[11] Deliparaschos, K.M.; Nenedakis, F.I.; Tzafestas, S.G.; *"A fast digital fuzzy logic controller: FPGA design and implementation;"* Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference, 19–22 September 2005, Volume 1.

[12] ---▶ Aranguren, G.; Barron, M.; Arroyabe, J.L.; Garcia–Carreira, G.; "*A Pipe–line Fuzzy Controller in FPGA;"* Fuzzy Systems, 1997. Proceedings of the Sixth IEEE International Conference, 1–5 July 1997, pp. 635–640, Volume2.

[13] ---▶ Youngdal Kim; Hyung Lee–Kwang; *"An Architecture of Fuzzy Logic Controller with Parallel Defuzzification;"* Fuzzy Information Processing Society, 1996. NAFIPS. 1996 Biennial Conference of the North American, 19–22 June 1996, pp. 497–501.

[14] ---▶ Gonzalez, J.L.; Castillo, O.; Aguilar, L.T.; *"FPGA as a Tool for Implementing Non–fixed Structure Fuzzy Logic Controllers;"* Foundations of Computational Intelligence, 2007. FOCI 2007. IEEE Symposium, 1–5 April 2007, pp. 523–530.

[15] ---▶ Sanchez–Solano, S.; Cabrera, A.J.; Baturone, I.; Moreno–Velo, F.J.; Brox, M.; *"FPGA Implementation of Embedded Fuzzy Controllers for Robotic Applications;"* Industrial Electronics, IEEE Transactions, August 2007, pp. 1937–1945, Volume 54, Issue 4.

[16] ---▶ Gaona, A.; Olea, D.; Melgarejo, M.; *"Sequential Fuzzy Inference System Based on Distributed Arithmetic;"* Computational Intelligence for Measurement Systems and Applications, 2003. CIMSA '03. 2003 IEEE International Symposium, 29–31 July 2003, pp. 125–129.

[17] ---▶ Daijin Kim; *"An Implementation of Fuzzy Logic Controller on the Reconfigurable FPGA System;"* Industrial Electronics, IEEE Transactions, June 2000, pp. 703–715, Volume 47, Issue 3.

[18] ---▶ Ramos, R.; Roset, X.; Manuel, A.; *"Implementation of Fuzzy Logic Controller for DC/DC Converters Using FPGA;"* Instrumentation and Measurement Technology Conference, 2000. IMTC 2000. Proceedings of the 17th IEEE, 1–4 May 2000, pp. 160–163, Volume 1.

[19] ---▶ McKenna, M.; Wilamowski, B.M.; *"Implementing a Fuzzy System on a Field Programmable Gate Array;"* Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference, 15–19 July 2001, pp. 189–194, Volume 1.

[20] ---▶ Lund, T.; Torralba, A.; Carvajal, R.G.; *"The Architecture of an FPGA–Style Programmable Fuzzy Logic Controller Chip;"* Computer Architecture Conference, 2000. ACAC 2000. 5th Australasian, 31 January–3 February 2000, pp. 51–56.

[21] ---▶ Khatra, A.P.; Rattan, K.S.; *"Implementation of a Multi–Layered Fuzzy Controller on an FPGA;"* Fuzzy Information Processing Society, 2006. NAFIPS 2006. Annual meeting of the North American, 3–6 June 2006, pp. 420–425.

[22] ---▶ Vuong, P.T.; Madni, A.M.; Vuong, J.B.; *"VHDL Implementation For a Fuzzy Logic Controller;"* World Automation Congress, 2006. WAC '06 24–26, July 2006, pp. 1–8.

[23] ---▶ Xin Yuan; Hong Xiang Lan; *"VLSI Design of Fuzzy Logic Controller;"* Solid–State and Integrated Circuit Technology, 1995 4th International Conference, 24–28 October 1995, pp. 691–693.

[24] ---▶ Chia–Feng Juang; Jung–Shing Chen; *"Water Bath Temperature Control by a Recurrent Fuzzy Controller and Its FPGA Implementation;"* Industrial Electronics, IEEE Transactions, June 2006, pp. 941–949, Volume 53, Issue 3.

[25] ---▶ Young Dal Kim; Hyung Lee–Kwang; *"High Speed Flexible Fuzzy Hardware for Fuzzy Information Processing;"* Systems, Man and Cybernetics, Part A, IEEE Transactions, January 1997, pp. 45–56, Volume 27, Issue 1.

[26] ---▶ Manzoul, M.A.; Jayabharathi, D.; *"Fuzzy Controller on FPGA Chip;"* Fuzzy Systems, 1992., IEEE International Conference, 8–12 March 1992, pp. 1309–1316.

[27] ---▶ Manzoul, M.A.; Jayabharathi, D.; *"Implementation of Fuzzy Controllers using Combinatorial Circuits;"* Fuzzy Systems, 1992, Proceedings of NAFIPS'91 Workshop, May 1991, pp. 163–167.

[28] ---▶ Passino K.; Yurkovich S.; *"Fuzzy Control;"* Addison Wesley Longman, Inc. 1998, pp. 51–69.

[29] ---▶ Ogata K.; *"Ingeniería de Control Moderna;"* 4ª. Edición, Pearson Educación, 2003.

[30] ---▶ Marlinda E.; Guevara J.; López J. *"Blogspot: Circuitos Integrados Configurables ASIC;"* http://icprgm–asic.blogspot.com/search?updated–min=2007–01–01T00%3A00%3A00–08%3A00&updated–max=2008–01–01T00%3A00%3A00–08%3A00&max–results=14

[31] ---▶ Wikipedia Search: *"ASIC;"* http://en.wikipedia.org/wiki/Application–specific_integrated_circuit

[32] ---▶ Wikipedia Search: *"Lookup Table;"* http://en.wikipedia.org/wiki/Lookup_table

[33] ---▶ Wikipedia Search: *"Pipeline;"* http://en.wikipedia.org/wiki/Pipeline_%28computing%29

[34] ---▶ Wikipedia Search: *"Parallel Computing;"* http://en.wikipedia.org/wiki/Parallel_computing

[35] ---▶ Wikipedia Search: *"Propagation Delay;"* http://en.wikipedia.org/wiki/Propagation_delay

[36] ---▶ National Instruments™ Tutorial: *"Introduction to FPGA Technology: Top Five Benefits;"* http://zone.ni.com/devzone/cda/tut/p/id/6984?metc=mt6dhp

[37] ---▶ Kaczynski J.; *"The Challenges of Modern FPGA Design Verification;"* ALDEC. FPGA Journal; http://www.fpgajournal.com/articles/20040727_aldec.htm

[38] ---▶ Wikipedia Search: *"FPGA;"* http://en.wikipedia.org/wiki/Field–programmable_gate_array

[39] ---▶ Wikipedia Search: *"Field Programmability;"* http://en.wikipedia.org/wiki/Field–programmable

[40] ---▶ Wikipedia Search: "*Artificial Intelligence;*" http://en.wikipedia.org/wiki/Artificial_intelligence

[41] ---▶ Kosko B., Isaka S.; "Fuzzy Logic;" http://www.fortunecity.com/emachines/e11/86/fuzzylog.html

[42] ---▶ Monografías Search: *"Inteligencia Artificial;"* http://www.monografias.com/trabajos12/inteartf/inteartf.shtml

[43] ---▶ http://elestantedelfondo.blogspot.com/2007/03/no–es–lo–mismo.html

[44] ---▶ http://www.xilinx.com/company/history.htm

[45] ---▶ Nguyen, H. T.; Prasad, N. R.; Walker, C. L.; Walker, E. A.; *"A First Course in Fuzzy and Neural Control;"* Champman & Hall/CRC Editions, 2003; pp. 146–157.

[46] ---▶ Wikipedia Search: "*Servomotor;*" http://es.wikipedia.org/wiki/Servomotor

[47] ---▶ Monografías Search: *"Leyes de Newton;"* http://www.monografias.com/trabajos30/leyes–newton/leyes–newton.shtml

[48] ---▶ Wikipedia Search: "*Inferencia;*" http://es.wikipedia.org/wiki/Inferencia

[49] ---▶ Novoa, G. J. F. *"La relación de consecuencia lógica;"* http://www.uv.mx/cienciahombre/revistae/vol20num3/articulos/consecuencia/index.html

[50] ---▶ Patyra, M. J.; Mlynek, D.M.; "*Fuzzy logic: implementation and applications;*" Wiley; 1996.

[51] ---▶ Wikipedia Search: "*Division Algorithm;*" http://en.wikipedia.org/wiki/Division_algorithm

[52] ---▶ Oberman, S. F.; Flynn, M. J.; *"Division Algorithms and Implementations;"* IEEE Transactions on Computers; Aug 1997; Vol 46, No. 8; pp. 833–854.

[53] ---▶ National Instruments™ Tutorial: *"FPGAs: Under the Hood;"* http://zone.ni.com/devzone/cda/tut/p/id/6983

[54] ---▶ Roger Jang's Publication List: http://140.114.76.148/jang/research/publication/list.asp?sepField=Type

[55] ---▶ The MathWorks™; "*MATLAB. Fuzzy Logic Toolbox. User's guide;*" Release 2008a; version 2.

[56] ---▶ Wikipedia Search: "*Division (Digital);*" http://en.wikipedia.org/wiki/Division_%28digital%29

[57] ---▶ Altuim Design Summer 08 Learning Guides: *"Altium VHDL Language Reference," "Altium Synthesis Reference;"* http://www.altium.com/community/support/learningguides

[58] ---▶ Bhasker, J; *"Verilog HDL Synthesis: A Practical Primer;"* Lucent Technologies; Star Galaxy Publishing; 1998.

Anexo 1

El algoritmo de la división

El algoritmo de la división es un teorema de las matemáticas el cual expresa precisamente del proceso de fraccionamiento de enteros. El nombre de algoritmo es un término equivocado ya que es un teorema y no un algoritmo, aunque el algoritmo de la división puede ser usado para obtener el máximo común divisor de dos enteros [51].

Debe hacerse notar que el término "algoritmo de la división" en el estudio del álgebra es comúnmente aplicado de manera general a una variante de este teorema. Además el término "algoritmo de la división" se debe a que han existido a lo largo de la historia de la computación diversas maneras de obtener el mismo resultado al implementarlo en hardware [52].

En los años recientes implementaciones de la división dedicadas a diversas aplicaciones han incrementado su complejidad computacional; asimismo, aquéllos sistemas que incluyen la división basan su velocidad de operación en la velocidad de este algoritmo, debido a que tal complejidad lo hace una de las implementaciones en hardware más lentas. En consecuencia, los diseñadores de microprocesadores de propósito general han tenido que poner mucha atención en la implementación de sus unidades de punto flotante. Por ejemplo, sistemas de renderización de gráficos de alto rendimiento requieren hardware dedicado de alta velocidad que realice divisiones de punto flotante. Aunque en realidad se ha puesto poca atención en la mejora del rendimiento de la división a lado de la suma, la resta y la multiplicación fue necesario realizar un estudio para obtener la división mediante uno o varios algoritmos que fueran capaces de incrementar el rendimiento obteniendo los mismos resultados.

Inicialmente los operandos de los algoritmos de la división eran enteros y la división se expresaba mediante un cociente y un residuo; pero la división se extendió a los límites de la precisión y la división de enteros no fue suficiente, es por ello que los operandos de la división son de punto flotante o punto fijo en las unidades de procesamiento de punto flotante y por ende el cociente es de punto flotante o punto fijo.

Todos los algoritmos son de tipo iterativo y cada iteración se encarga de obtener un resultado nuevo para la iteración siguiente. El algoritmo de la división más sencillo es el de *Restas Sucesivas*, pero es uno de los más costosos. Puede consumir muchos recursos en hardware si se implementa con lógica combinatoria o de manera secuencial puede llevarse muchos ciclos de reloj hasta conseguir el cociente. Este algoritmo es apto para la división de enteros.

Otro algoritmo de la división apto para operandos enteros es la *División Con Restauración*. A diferencia del algoritmo de restas sucesivas este algoritmo obtiene un dígito del cociente en cada iteración y realiza también restas sucesivas. Es un algoritmo muy sencillo que obtiene en cada iteración un dígito del cociente a partir de los resultados de las restas de divisores desplazados. Esta es una variante del algoritmo original y es uno de los algoritmos utilizados en este trabajo el cual se describe detalladamente a continuación en este anexo.

Según Oberman [52], existen diversos algoritmos de la división dentro de los cuales clasifican a los algoritmos de *Recurrencia de Dígito*. Dentro de esta clasificación se encuentran las divisiones *SRT, Sin Restauración* e incluso el algoritmo *Con Restauración* mencionado anteriormente. Estos algoritmos son mucho más convenientes para la realización de divisiones de punto flotante o punto fijo, pero también son compatibles hasta cierto punto con operaciones de enteros. En todos estos algoritmos se realizan operaciones de resta en cada iteración y con base en los resultados de estas restas se selecciona un dígito mediante una *función de selección de dígito*, dentro de un conjunto de símbolos numéricos o dígitos (dependiendo de la base de los operandos, es decir base–2, base–4, base–10, etc), formando así en cada iteración el cociente resultante (de esa misma base). Todos estos algoritmos tienen en común la función de selección de dígito y cada uno de ellos realiza de manera diferente esa selección. Esto los hace diferentes entre ellos, sin embargo tales diferencias pueden afectar al rendimiento general de un sistema computacional. De ahí la importancia de conocerlos para saber cuál de ellos es el más conveniente para determinada aplicación.

En este anexo también se hace uso del Algoritmo de *División Sin Restauración*, el cual se describe de manera detallada a continuación.

Algoritmo con restauración

Originalmente este algoritmo se basa en la obtención del dígito del cociente en cada iteración mediante la realización de restas entre el *residual* y el divisor y la *Restauración* del residual dependiendo del resultado de cada resta. El residual es resultado entre cada resta que sirve para seguir dividiendo el dividendo entre el divisor. Dicha restauración es realizada si el resultado de la resta es negativo, así cada vez que el

residual se hace negativo, este se restaura con el valor residual positivo anterior. En cada iteración el residual es desplazado a la izquierda mediante un registro de corrimiento, antes de realizar la resta.

A continuación se muestran los siguientes pasos para realizar en hardware el algoritmo de la División Con Restauración:

1. Definir el tamaño de los operandos, denotado por n. Este es el número total de iteraciones.
2. Inicializar el residual inicial con el valor del dividendo.
3. En cada iteración, respaldar el residual inmediato en una variable que guarde el residual anterior. La primera iteración comienza obteniendo el MSB del cociente y termina con el LSB.
4. Desplazar el residual a la izquierda una posición.
5. Restar al residual actual el valor del divisor.
6. El bit del cociente en dicha iteración será el bit de signo negado del resultado de la resta del paso anterior.
7. Si el bit de signo del resultado de la resta del paso 5 es uno, entonces debe restaurarse el valor del residual actual con el valor respaldado del residual anterior. De lo contrario, no restaurar.
8. El proceso continúa hasta que se haya procesado el último bit del cociente.

Este diseño fue implementado en VHDL para fines de prueba y el código es el siguiente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DIVR is
	generic(n: integer := 8);
	Port(DVND : in std_logic_vector(n-1 downto 0);
		DVSR : in std_logic_vector(n-1 downto 0);
		QUOT : out std_logic_vector(n-1 downto 0);
		UERROR : out std_logic);
end DIVR;

architecture Behavioral of DIVR is
begin
	process(DVND, DVSR)
		variable getquot : std_logic_vector(n-1 downto 0);
		variable divisor, last_residual, residual : std_logic_vector(2*n downto 0);
		begin
			divisor := '0' & DVSR & conv_std_logic_vector(0, n);
			residual := (conv_std_logic_vector(0, n+1) & DVND);
			for i in n-1 downto 0 loop
				last_residual := residual;
				residual := (residual(2*n-1 downto 0) & '0') -divisor;
				getquot(i) := not residual(2*n);
				if residual(2*n) = '1' then
					residual := (last_residual(2*n-1 downto 0) & '0');
				end if;
			end loop;
			QUOT <= getquot;
	end process;
	UERROR <= '1' when DVSR = conv_std_logic_vector(0, n) else '0';
end Behavioral;
```

Código 1. Implementación genérica en VHDL del algortimo de la división con restauración.

Como puede observarse en la estructura *for*, se encuentran la mayoría de los pasos de la división con restauración, y cada una de ellas representa una serie etapas idénticas en cascada que se entregan a su etapa consecuente un residual que sirve para calcular los siguientes dígitos del cociente. El residual se recorre hacia la izquierda a partir de un valor inicial que es el dividendo. Así pues cada etapa realiza una resta, calcula un dígito del cociente y restaura si es necesario. El bit de signo es el MSB del residual y es el que se verifica en cada iteración.

Este tipo de división es óptima para valores enteros y fraccionarios de sus operandos y el cálculo del residuo es directo como puede observarse en la siguiente tabla. Supóngase un módulo de división de 8 bits cuyo dividendo es FFH y el divisor es 0CH:

Tabla 1. Resultados de la división con restauración de 8 bits.

Iteration	Last residual	Residual	Get Quotient							
–	X XXXX	0 00FF	X	X	X	X	X	X	X	X
7	0 00FF	1 F5FE ⟶ 0 01FE	0	X	X	X	X	X	X	X
6	0 01FE	1 F7FC ⟶ 0 03FC	0	0	X	X	X	X	X	X
5	0 03FC	1 FBF8 ⟶ 0 07F8	0	0	0	X	X	X	X	X
4	0 07F8	0 03F0	0	0	0	1	X	X	X	X
3	0 03F0	1 FBE0 ⟶ 0 07E0	0	0	0	1	0	X	X	X
2	0 07E0	0 03C0	0	0	0	1	0	1	X	X
1	0 03C0	1 FB80 ⟶ 0 0780	0	0	0	1	0	1	0	X
0	0 0780	0 0300	0	0	0	1	0	1	0	1
Remainder:			**03**							
Quotient:			**15**							

Como puede observarse en el código, no existe un puerto que muestre el resultado del residuo, pues no se necesita para este trabajo. Además, este algoritmo permite realizar divisiones entre operandos fraccionarios realizando modificaciones sencillas al código. Sin embargo existe una manera de realizar este mismo algoritmo de la división, pero ahora recorriendo el divisor hacia la derecha.

En este algoritmo alternativo existe un divisor desplazado por cada etapa y el residual es calculado de acuerdo a un bit de habilitación. Calculando todos los valores posibles del divisor desplazado para cada etapa y con el bit de habilitación es posible realizar el cálculo del siguiente residual mediante una resta, o bien, la restauración dejando pasar el residual anterior a la siguiente etapa. El bit de habilitación es calculado realizando operaciones OR entre cada bit del divisor. La siguiente figura lo muestra con el algoritmo de la división de 8 bits:

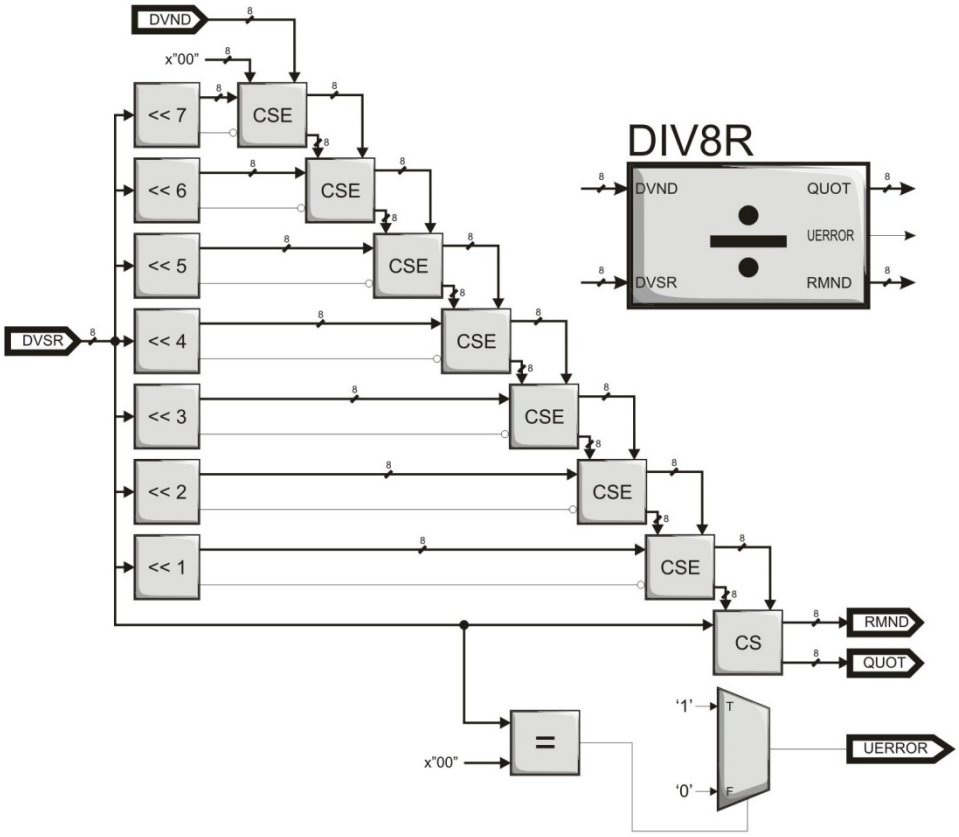

Figura 1: División con restauración de 8 bits.

Este algoritmo está implementado con módulos CSE y CS que se encargan de realizar el proceso de la división. Los módulos de desplazamiento a la izquierda calculan todos los valores de desplazamiento del divisor y entregan un bit de habilitación y un valor de divisor desplazado diferente. Los módulos CSE y CS están compuestos de un restador y varios multiplexores como se muestra a continuación:

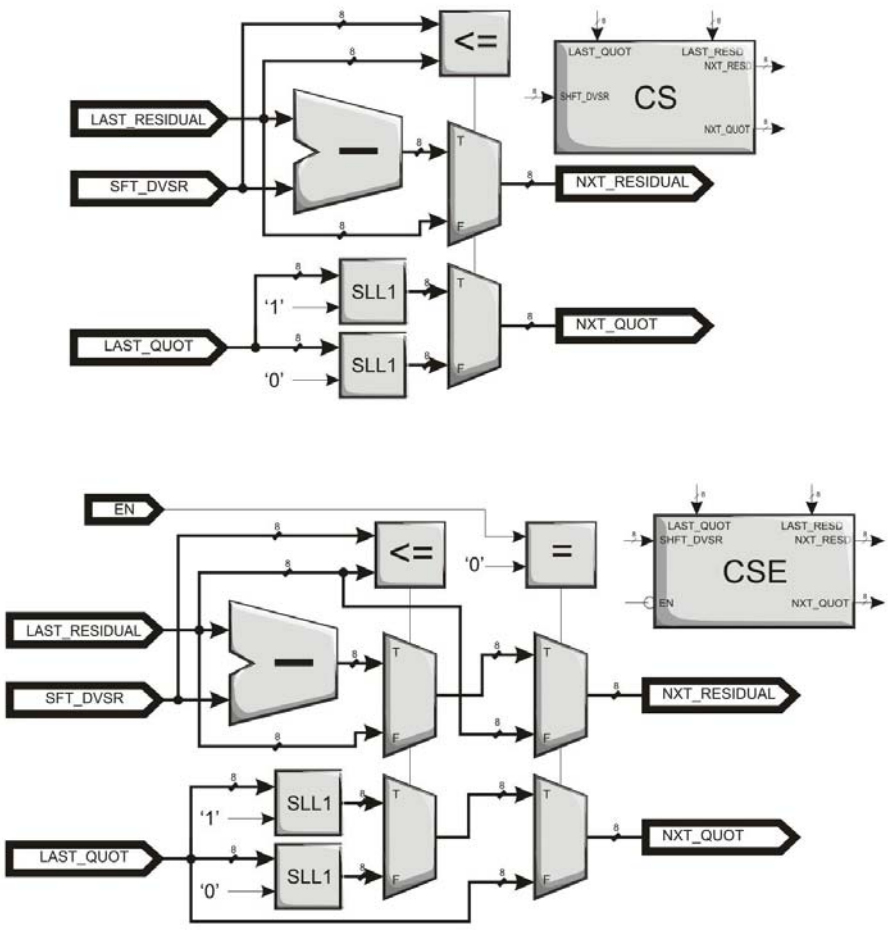

Figura 2. Módulos CSE y CS que forman parte de la división con restauración alternativa.

Como puede observarse en la anterior, en el módulo CSE uno de los multiplexores se encarga de comparar (menor o igual que) el valor de residual que recibe con el divisor desplazado correspondiente y de esta manera restarlos entre sí o de lo contrario dejar pasar el residual anterior a la siguiente etapa, es decir, restaurarlo. A su vez, otro multiplexor se encarga de insertar por la derecha un uno o un cero al registro de corrimiento SLL1 que recibe al cociente anterior, desplazándolo a la izquierda y entregando a la siguiente etapa la modificación del cociente resultante. Dependiendo de esta comparación deja pasar el cociente desplazado con uno o el cociente desplazado con cero. El bit de habilitación EN deja pasar los resultados de los multiplexores anteriores. Para el módulo CS no tiene los multiplexores del bit de habilitación con lo que permite entregar como resultado el cociente y el residuo final. Este modelo es perfecto para operadores entre enteros.

De igual manera se muestra una tabla que contiene los resultados después de cada iteración de la división de 8 bits de FFH entre 0CH:

Tabla 2. . Resultados de la división con restauración alternativa de 8 bits.

Iteration	EN	Shifted Divisor	Residual	Next Quotient
–	X	XX	FF	00
1	1	00	FF	00
2	1	00	FF	00
3	1	80	FF	00
4	0	C0	3F	01
5	0	60	3F	02
6	0	30	0F	05
7	0	18	0F	0A
8	0	0C	03	15
Remainder:				**03**
Quotient:				**15**

Estas son dos maneras diferentes de realizar la división con restauración. Sin embargo, para los fines de este trabajo es más conveniente el uso de este último enfoque.

Algoritmo sin restauración

Este algoritmo también se basa en la obtención del dígito del cociente en cada iteración mediante la realización de restas y también de sumas entre el *residual* y el divisor. La restauración, en este caso, no es realizada debido a que en cada iteración si el resultado es negativo se suma al residual el divisor y si el resultado es positivo se resta al residual el divisor, por lo que si el resultado es negativo no es necesario restaurar el residual al valor anterior. En cada iteración el residual es desplazado a la izquierda mediante un registro de corrimiento, antes de realizar una suma o una resta.

A continuación se muestran los siguientes pasos para realizar en hardware el algoritmo de la División Sin Restauración:

1. Definir el tamaño de los operandos, denotado por n. Este es el número total de iteraciones.
2. Inicializar el residual inicial con el valor del dividendo y restarle el divisor.
3. En cada iteración, desplazar el residual a la izquierda una posición. Verificar el bit de signo del residual después de cada suma o resta.

4. Si el residual es negativo sumar el divisor al residual. Si es positivo restar el divisor al residual.
5. El bit del cociente en dicha iteración será el bit de signo negado del resultado del paso 4.
6. El proceso continúa hasta que se haya procesado el último bit del cociente.

Este diseño fue implementado en VHDL para implementación como parte de este trabajo y el código es el siguiente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DIVNR is
	generic(n: integer := 8);
	Port(DVND : in std_logic_vector(n-1 downto 0);
		DVSR : in std_logic_vector(n-1 downto 0);
		QUOT : out std_logic_vector(n-1 downto 0);
		UERROR : out std_logic);
end DIVNR;

architecture Behavioral of DIVNR is
begin
	process(DVND, DVSR)
		variable getquot : std_logic_vector(n-1 downto 0);
		variable divisor, residual : std_logic_vector(2*n downto 0);
		begin
			divisor := '0' & DVSR & conv_std_logic_vector(0, n);
			residual := (conv_std_logic_vector(0, n+1) & DVND) -divisor;
			for i in n-1 downto 0 loop
				residual := residual(2*n-1 downto 0) & '0';
				if residual(2*n) = '1' then
					residual := residual + divisor;
				else
					residual := residual -divisor;
				end if;
				getquot(i) := not residual(2*n);
			end loop;
			QUOT <= getquot;
	end process;
	UERROR <= '1' when DVSR = conv_std_logic_vector(0, n) else '0';
end Behavioral;
```

Código 2. Implementación genérica en VHDL del algortimo de la división sin restauración.

De igual manera se muestra una tabla que contiene los resultados después de cada iteración de la división de 8 bits de FFH entre 0CH:

Tabla 3. Resultados de la división sin restauración de 8 bits con residuo correcto.

Iteration	Residual	Get Quotient							
–	1 F4FF	X	X	X	X	X	X	X	X
7	1 F5FE ⟶ ADD	0	X	X	X	X	X	X	X
6	1 F7FC ⟶ ADD	0	0	X	X	X	X	X	X
5	1 FBF8 ⟶ ADD	0	0	0	X	X	X	X	X
4	0 03F0 ⟶SUB	0	0	0	1	X	X	X	X
3	1 FBE0 ⟶ ADD	0	0	0	1	0	X	X	X
2	0 03C0 ⟶ SUB	0	0	0	1	0	1	X	X
1	1 FB80 ⟶ ADD	0	0	0	1	0	1	0	X
0	0 0300	0	0	0	1	0	1	0	1
Remainder:					**03**				
Quotient:					**15**				

Este algoritmo tiene una ventaja y una desventaja. La ventaja consiste en que aunque los operandos sean enteros el resultado puede expresarse en formato fraccionario en binario incrementando el número de bits en el módulo de salida y por ende el número de iteraciones del algoritmo. La desventaja consiste en que si se requiere utilizar el residuo para cualquier propósito, el residual no es suficiente por sí solo para expresarlo para todos los valores de sus operandos, pues al no tener una restauración puede arrojar valores de residuos negativos que conllevan a valores incorrectos, aunque el cociente sea correcto.

Por ejemplo, para la división de 8 bits de FFH entre 0EH, los resultados son los siguientes:

Tabla 4. Resultados de la división sin restauración de 8 bits con residuo incorrecto.

Iteration	Residual	Get Quotient							
–	1 F2FF	X	X	X	X	X	X	X	X
7	1 F3FE ⟶ ADD	0	X	X	X	X	X	X	X
6	1 F5FC ⟶ ADD	0	0	X	X	X	X	X	X
5	1 F9F8 ⟶ ADD	0	0	0	X	X	X	X	X
4	0 01F0 ⟶SUB	0	0	0	1	X	X	X	X
3	1 F5E0 ⟶ ADD	0	0	0	1	0	X	X	X
2	1 F9C0 ⟶ ADD	0	0	0	1	0	0	X	X
1	0 0180 ⟶ ~~SUB~~	0	0	0	1	0	0	1	X
0	1 ~~F5~~00	0	0	0	1	0	0	1	0
Remainder:		**incorrecto: F5, correcto: 03**							
Quotient:		**12**							

Antes de realizar la última resta en la iteración uno el residual es 0 0180, por lo que sólo se debe realizar el desplazamiento en la iteración cero para obtener el residuo correcto quedando 0 0300, sin embargo este problema tiene solución agregando hardware que deshabilite la última operación de resta antes de que el residual final se haga negativo, pues para determinados valores de sus operandos sobra una operación de resta. Este problema no ocurre en el algoritmo de la división con restauración como pudo verse en la sección anterior.

El algoritmo de la multiplicación

Esta sección muestra el algoritmo de la multiplicación básica, que es similar al algoritmo que se enseña en la educación básica. Esta sección es breve por el hecho de que existen algoritmos de multiplicación mucho más rápidos que este y el algoritmo de la división es el de mayor interés por ser el más lento de las cuatro operaciones básicas a nivel computacional. El propósito es mostrar al lector la sencillez para diseñarlo e integrarlo en cualquier sistema que lo requiera sin ocupar los multiplicadores dedicados integrados de un FPGA.

La base de este algoritmo es acumular en una variable el valor desplazado de uno de los factores si alguno de los bits del otro factor es uno, dependiendo del bit analizado en cada iteración.

A continuación se muestran los siguientes pasos para realizar en hardware el algoritmo de la multiplicación básica:

1. Definir el tamaño de los operandos, denotado por n. Este es el número total de iteraciones. El tamaño del resultado será la suma de los bits de cada operando.
2. Inicializar el acumulador a cero y la variable desplazadora con el primer factor.
3. En cada iteración, verificar el valor de los bits del segundo factor a partir del LSB. Si el bit es uno, entonces acumula el valor de la variable desplazadora. De lo contrario, no acumules.
4. Desplazar a la variable desplazadora una posición a la izquierda.
5. El proceso continúa hasta que se haya procesado el último bit del segundo factor.

Este diseño fue implementado en VHDL para implementación como parte de este trabajo y el código es el siguiente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MUL is
	generic( n : integer := 8);
	Port( FACT1 : in std_logic_vector(n-1 downto 0);
		FACT2 : in std_logic_vector(n-1 downto 0);
		PROD : out std_logic_vector(2*n-1 downto 0));
end MUL;

architecture Behavioral of MUL is
begin
	process( FACT1, FACT2)
	variable wresult, accumulate, wshfact : std_logic_vector( 2*n-1 downto 0 );
	begin
		wshfact := conv_std_logic_vector(0, n) & FACT2;
		accumulate := conv_std_logic_vector(0, 2*n);
		for i in 0 to n-1 loop
			if FACT1(i) = '1' then
				accumulate := accumulate + wshfact;
			end if;
			wshfact := wshfact(2*n-2 downto 0) & '0';
		end loop;
		PROD <= accumulate;
	end process;
end Behavioral;
```

Código 3. Implementación genérica en VHDL del algortimo de la multiplicación básica.

Anexo 2

El FPGA, bajo la lupa

Existen algunas herramientas de diseño de alto nivel para tecnología FPGA, que ofrecen facilidades a aquéllos ingenieros y científicos que tienen o no experiencia en diseño de hardware digital. Sin embargo, si la persona tiene experiencia en programación gráfica, C o VHDL, puede preguntarse cómo es que un bloque de silicón configurable puede ejecutar un programa.

Cada FPGA está elaborado de un número finito de recursos predefinidos con interconexiones programables para implementar un circuito digital.

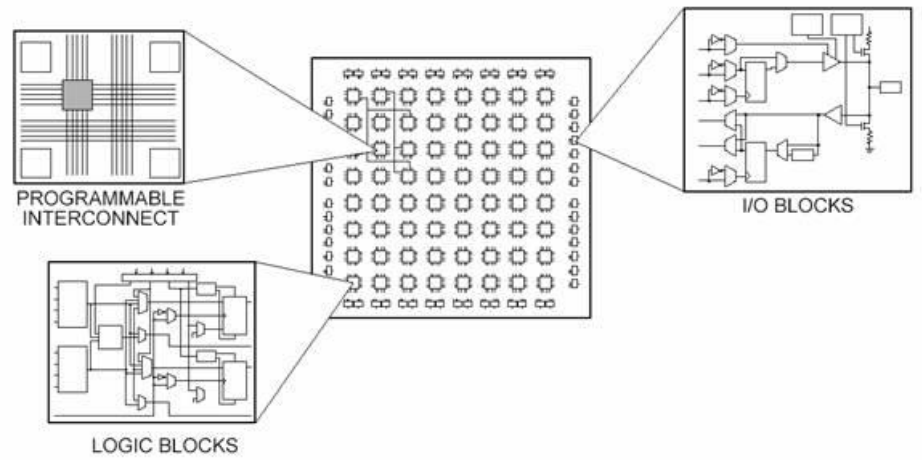

Figura 3. Partes de un FPGA.

Las especificaciones de un FPGA incluyen la cantidad de bloques lógicos configurables, el número de bloques lógicos para funciones dedicadas, tales como multiplicadores; y el tamaño de recursos de memoria tales como bloques de memoria RAM dedicada. Existen muchas otras partes de un FPGA, pero éstas son típicamente las más importantes cuando se va a seleccionar y a comparar FPGAs para una aplicación en particular.

En el más bajo nivel, los bloques lógicos configurables, llamados *Slices* o *Celdas Lógicas*, están hechas de dos estructuras básicas: los *Flip–Flops* (FF) y las *Look Up Tables* (LUT). Es importante notar existen muchas familias de FPGA en el mercado, todas ellas difieren entre sí en la manera de cómo vienen empacados los FFs y las LUTs. Por ejemplo, el FPGA Virtex–II de Xilinx contiene celdas con dos LUTs y dos FFs, en cambio Virtex–5 posee cuatro LUTs y cuatro FFs. A su vez, la arquitectura de una LUT en sí, puede variar de cuatro o cinco entradas entre familias.

La siguiente tabla enlista las especificaciones de algunos FPGAs de Xilinx para motivos de comparación. El número de compuertas equivalentes ha sido tradicionalmente una manera de comparar a la tecnología FPGA de los ASIC, pero esta cantidad no describe realmente el número de componentes individuales dentro del FPGA. Es por eso que Xilinx no especifica el número de compuertas equivalentes para la nueva familia Virtex–5.

Tabla 5. Algunas especificaciones de diferentes familias de FPGA de Xilinx.

	Virtex–II 1000	Virtex–II 3000	Spartan–3 1000	Spartan–3 2000	Virtex–5 LX30	Virtex–5 LX50	Virtex–5 LX85
Compuertas	1 M	3 M	1 M	2 M	———	———	———
Flip–Flops	10,240	28,672	15,360	40,960	19,200	28,800	51,840
LUTs	10,240	28,672	15,360	40,960	19,200	28,800	51,840
Multiplicadores	40	96	24	40	32	48	48
Bloques RAM (kb)	720	1,728	432	720	1,152	1,728	3,456

Para entender estas especificaciones mejor, es necesario considerar que todo código es sintetizado en un circuito digital. La *Síntesis* es el proceso de traducir un lenguaje de alto nivel en una verdadera implementación en hardware. A todo código sintetizable dado le corresponde un circuito que describe cómo un bloque lógico debe ser interconectado.

Ahora examinemos un circuito lógico digital obtenido a partir de un código de descripción de hardware. La Figura 62 muestra un ejemplo de cinco señales booleanas de entrada que fungen una función que calcula el valor de un bit a la salida.

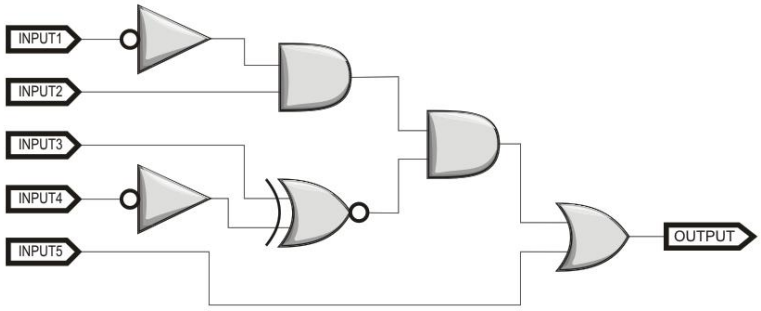

Figura 4. Circuito de una función simple booleana.

Bajo condiciones normales, el circuito, para un FPGA, que resulta de la Figura 62 es el siguiente:

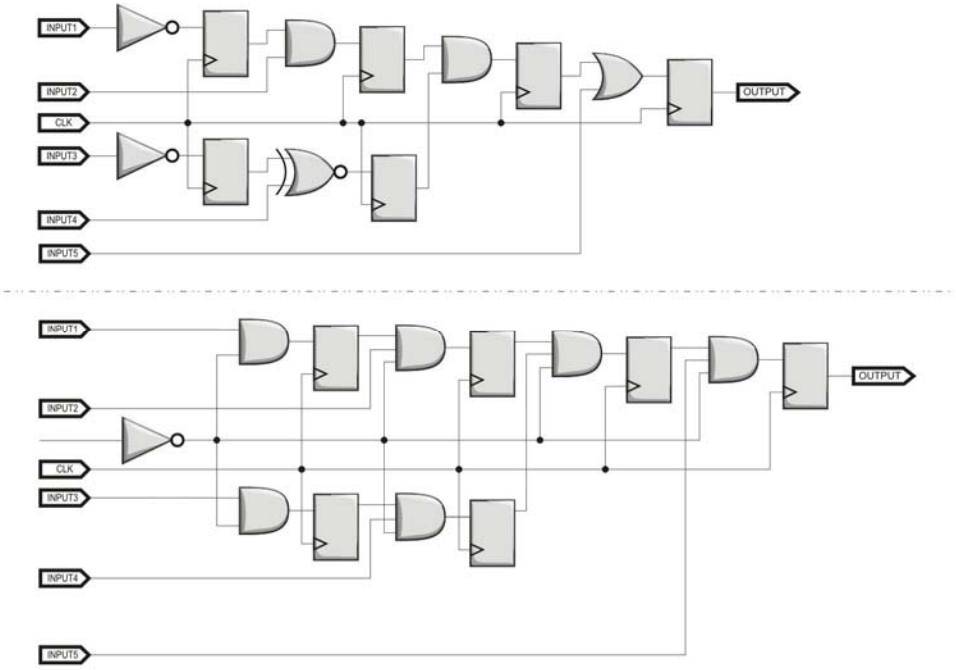

Figura 5. Circuito equivalente resultante de la Figura 62.

Puede ser difícil ver, pero existen dos ramas de circuitería que son creadas. Las primeros cinco señales de entrada alimentan a la rama superior, en la cual se agregan un FF entre cada operación booleana. Las otras cinco señales de entrada alimentan a la rama inferior con el mismo número de FFs con el fin de llevar la cuenta del número de ciclos necesarios para propagar el dato a lo largo del circuito digital. En total, son usados 12 FFs

y 12 LUTs para implementar este circuito. La rama superior y cada componente son analizados en la sección siguiente.

El Flip–Flop

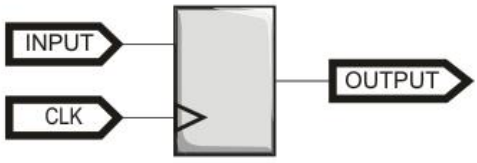

Figura 6. Símbolo para el Flip–Flop.

Los *Flip–Flops* son registros de corrimiento usados para sincronizar la lógica y para guardar estados lógicos entre cada ciclo de reloj. En cada flanco de reloj, un flip–flop almacena un valor de uno o cero (FALSO o VERDADERO) a partir de sus entradas y retiene ese valor constante hasta el siguiente flanco de reloj. Bajo condiciones normales, un programa de síntesis puede colocar un flip–flop entre cada operación para maximizar el tiempo de propagación disponible para cada operación a ejecutar. La excepción a esta regla es cuando un código determinado es colocado dentro de una estructura de control *for*, por ejemplo. En una estructura *for*, que es sintetizada como un replicador, los flip–flops son agregados al inicio y al final del bucle y es responsabilidad del programador considerar el tiempo de retraso que esto puede provocar. La Figura 65 muestra a la rama superior de la Figura 63, con los FFs remarcados con línea roja.

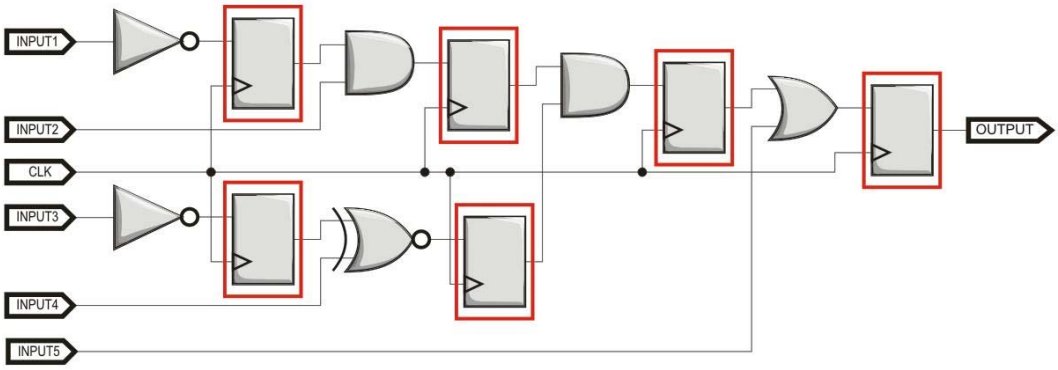

Figura 7. Identificación de los flip–flops de la Figura 63.

La Look Up Table

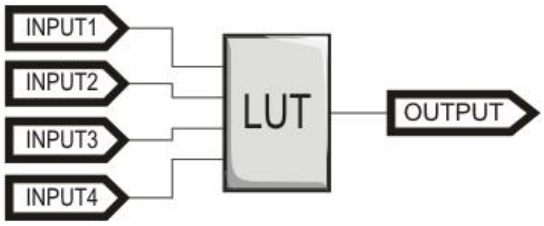

Figura 8. Look Up Table de 4 entradas.

La lógica restante en el circuito de la Figura 65 es implementada usando muy pequeñas cantidades de memoria RAM en la forma de tablas de búsqueda o tablas de mapeo. Es fácil asumir que el número de compuertas en un FPGA se refiere al número de compuertas NAND y compuertas NOR en un chip particular, pero en realidad, toda la *lógica combinatoria* (todas las compuertas AND, OR, NAND, NOR, XOR) es implementada como tablas de verdad dentro de memoria LUT. Una tabla de verdad es una lista predefinida de las salidas posibles por cada combinación de entradas.

Por ejemplo, para la operación booleana AND de la Figura 67, existe una tabla de verdad de dos entradas y una salida como la mostrada en la Tabla 16:

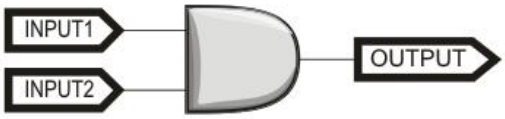

Figura 9. Operación booleana AND.

Tabla 6. Tabla de verdad para la operación booleana AND.

Entrada 1	Entrada 2	Salida
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

También puede verse a las entradas como a un índice numérico para todas las entradas posibles, como en un arreglo, como se muestra en la Tabla 17:

Tabla 7. Implementación de la LUT de la tabla de verdad para la operación booleana AND.

índice de la LUT	Salida
0(00)	**0**
1(01)	**0**
2(10)	**0**
3(11)	**1**

Los FPGA Virtex–II y Spartan–3 tienen LUTs de cuatro entradas para implementar tablas de verdad de hasta 16 combinaciones de cuatro señales entradas. Por ejemplo, la Figura 68 es la implementación de un circuito que posee 4 entradas:

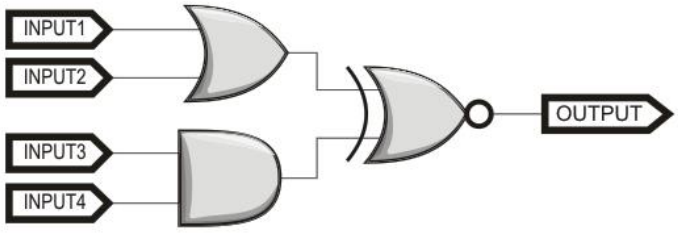

Figura 10. Circuito de 4 señales de entrada.

La Tabla 18 muestra la correspondiente tabla de verdad que se implementaría en una LUT de cuatro entradas:

Tabla 8. Implementación de la LUT para el circuito de la Figura 68.

índice de la LUT	Salida
0(0000)	**1**
1(0001)	**1**
2(0010)	**1**
3(0011)	**0**
4(0100)	**0**
5(0101)	**0**
6(0110)	**0**
7(0111)	**1**
8(1000)	**0**
9(1001)	**0**
10(1010)	**0**

11(1011)	**1**
12(1100)	**0**
13(1101)	**0**
14(1110)	**0**
15(1111)	**1**

La familia Virtex–5 usa LUTs de seis entradas, las cuales implementan tablas de verdad de hasta 64 combinaciones de seis diferentes señales de entrada. Esto incrementa de manera importante los recursos usados del FPGA cuando el circuito es replicado por una instrucción HDL de control de flujo, también la lógica combinatoria entre los FFs puede volverse mucho más compleja.

Las multiplicadores dedicados y las celdas DSP (DSP Slices)

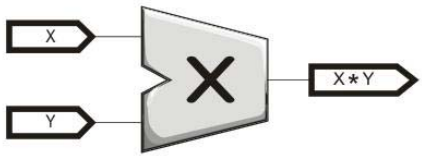

Figura 11. Módulo de multiplicación dedicada.

La simple tarea de multiplicar dos entradas puede incrementar la complejidad y el uso de los recursos a la hora de implementarlo en un circuito digital. Para proporcionar un marco de referencia, la Figura 70 es el dibujo esquemático de una manera de implementar un multiplicador de 4 bits por 4 bits usando lógica combinatoria:

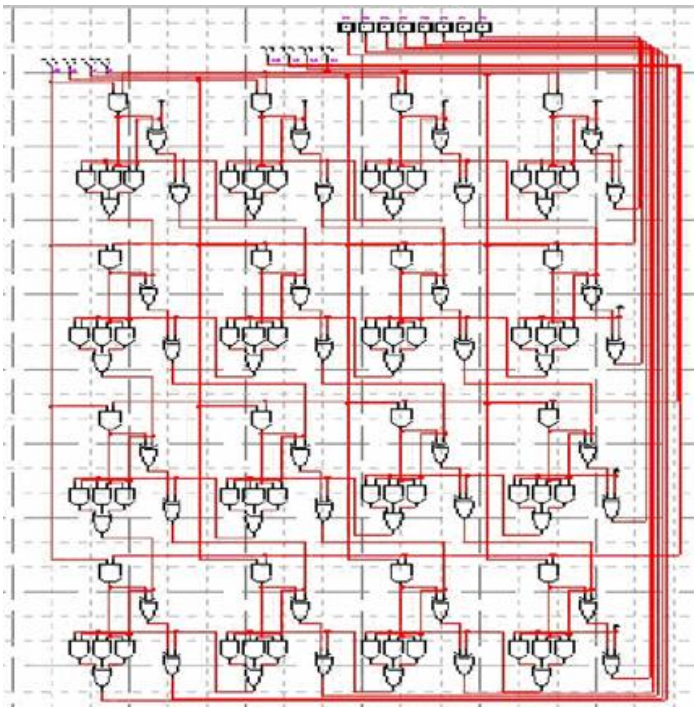

Figura 12. Circuito de un multiplicador de 4 por 4 bits.

Ahora imagine multiplicar dos números de 32 bits, lo cual se llevaría más de 2000 operaciones para una sola multiplicación. Debido a esto, el FPGA tiene circuitería dedicada de multiplicadores prediseñados para ahorrar la utilización de recursos en LUTs y FFs para aplicaciones matemáticas y para procesamiento digital de señales. Los FPGA Virtex–II y Sparta–3 tienen multiplicadores de 18 por 18 bits, así pues multiplicar dos números de 32 bits requeriría de tres multiplicadores para una sola operación. Muchos algoritmos de procesamiento de señales involucran acumular el resultado de varias multiplicaciones, como resultado, FPGAs de alto rendimiento como los Virtex–5 incluyen circuitería que realizan multiplicación–acumulación. Esto bloques prediseñados, también conocidos como celdas DSP48, ó DSP48 slices., integran un multiplicador de 25 bits por 18 bits con un sumador. La Tabla 19 muestra los recursos de varias familias de FPGA.

Tabla 9. Recursos de multiplicadores para diferentes familias de FPGA de Xilinx.

	Virtex–II 1000	Virtex–II 3000	Spartan–3 1000	Spartan–3 2000	Virtex–5 LX30	Virtex–5 LX50	Virtex–5 LX85
Multiplicadores	40	96	24	40	32	48	48
Tipo	18x18	18x18	18x18	18x18	Celdas DSP48	Celdas DSP48	Celdas DSP48

Los bloques RAM

Los recursos de memoria son otra especificación clave a considerar cuando se selecciona un FPGA. La memoria RAM definida por el usuario, distribuida en todo el chip FPGA, es útil para almacenar conjuntos de datos o pasar valores entre bucles paralelos. Dependiendo de la familia FPGA, es posible configurar los bloques de RAM en bloques de 16 o 32 kilobits. También existe la opción de implementar esos conjuntos de datos usando FFs, sin embargo, estos grandes arreglos pueden convertirse rápidamente en costos realmente altos en cuanto a recursos. Por ejemplo, un arreglo de 100 elementos de números de 32 bits podría consumir más del 30% de los FFs de un Virtex–II ó tomar menos del 1% de los bloques de RAM dedicada. Algunos algoritmos de procesamiento de señales frecuentemente necesitan almacenar un gran número de datos, tales como coeficientes de una ecuación compleja y sin tener a la mano memoria disponible, por lo que muchas funciones son posibles gracias al uso de estos bloques de memoria RAM.

Es posible también usar bloques de memoria para retener datos de formas de onda periódicas para la generación de señales en el mismo chip almacenando un período completo como una tabla de valores e ingresando a sus valores mediante índices de manera secuencial. La frecuencia de la señal de salida está determinada por la velocidad en que sus valores sean accedidos; entonces es posible usar este método para cambiar dinámicamente la frecuencia de salida sin introducir transiciones en la forma de onda.

La ejecución paralela inherente de los FPGAs permite usar diferentes señales de reloj para acceder a todas las piezas de hardware de manera independiente. Pasar datos entre diferentes bloques de ejecución puede ser difícil y la memoria dedicada es usada frecuentemente para suavizar esta transferencia por medio de búferes de tipo FIFO (first input, first output). Es posible crear un búfer FIFO para diferentes tamaños y asegurar que los datos no se pierdan entre las partes asíncronas del FPGA. La Tabla 20 muestra la cantidad de bloques de memoria RAM dedicada in varias familias de FPGA.

Tabla 10. Recursos de memoria para diferentes familias de FPGA de Xilinx.

	Virtex–II 1000	Virtex–II 3000	Spartan–3 1000	Spartan–3 2000	Virtex–5 LX30	Virtex–5 LX50	Virtex–5 LX85
RAM total (kb)	1728	720	432	720	1152	1728	3456
Tamaño del bloque (kb)	16	16	16	16	36	36	36

El uso de la tecnología FPGA continúa incrementando tanto como las herramientas de alto nivel evolucionan y más se abstraen los conceptos descritos en este anexo. También es importante examinar al FPGA y apreciar lo que está sucediendo cuando el lenguaje de descripción de hardware se compila y ejecuta en el silicón. Comparando y seleccionando el hardware basándose en los flip–flops, las LUTs, los multiplicadores y los bloques de memoria, es la mejor estrategia para elegir el chip FPGA adecuado para determinada aplicación. La utilización de recursos es extremadamente útil durante el desarrollo, especialmente cuando se optimiza el diseño para mejorar la velocidad o la cantidad.

Estos bloques de construcción fundamentales no son todos los recursos disponibles en un FPGA, ni los únicos entre una familia de un fabricante, mucho menos entre FPGA de diferentes fabricantes, ya que existen otras partes que son importantes y útiles, las cuales no son discutidas en este documento.

Anexo 3

Consideraciones para el diseño de un FLC en un FPGA

El diseño del FLC usando un lenguaje de descripción de hardware nos permite realizar replicaciones del mismo módulo con el fin de incrementar la precisión del mismo.

Lo que hay que tener en cuenta en el diseño de FLCs es primeramente llevar a cabo las etapas 1–4 de la metodología de diseño para una arquitectura general de un FLC, detalladas en el Capítulo 3. Posteriormente se debe elegir un FPGA adecuado de acuerdo al número de entradas al FLC, el número de funciones de membresía de cada entrada y de cada salida.

El diseño en FPGA, por medio de un kit de desarrollo, permite probar el funcionamiento de un sistema digital, en este caso, de un FLC, de manera sencilla, pero también es posible implementar este diseño en cualquier FPGA que tenga los recursos suficientes. Por medio de la Tabla 4, es posible realizar un conteo aproximado del uso de recursos por cada función de membresía usada para un determinado conjunto difuso; también es posible determinar el uso de recursos del desdifusificador de acuerdo al número de funciones de membresía de salida, usando el desdifusificador 2. De esta manera, se puede obtener un aproximado del número de compuertas equivalentes que utiliza un determinado diseño y con base en esto, el diseñador puede decidir el FPGA que más le conviene utilizar.

Una vez que se decide qué dispositivo FPGA se usará, es necesario modificar el código en VHDL de los siguientes módulos (que viene en el CD adjunto a este trabajo):

- FUZZIFIER
- MAMDANI
- DEFUZZIFIER

Antes de realizar la modificación es necesario crear un proyecto en el ISE 6.3i y agregar los módulos DIV8R, DIV8NRN, DIV16NRN, ISOTRIANGLE, SSTEP, ZSTEP, MIN y MAX (todos con extensión ***.vhd**, mismos que vienen en el CD), esto con el fin de incluir los módulos básicos para construir cualquier FLC. Las conexiones de los módulos antes

descritos dependen del diseño realizado en las primeras cuatro etapas de la metodología de diseño propuesto, sobre todo en la máquina de inferencia MAMDANI.vhd, proceso que se explica en todo el Capítulo 3.

El módulo que se encargará de contener a los módulos FIZZIFIER, MAMDANI y DEFUZZIFIER, puede tomar el nombre de la aplicación a la que se destine. Por ejemplo, en el CD hay un archivo llamado METEOROS.vhd, que se encarga de contener a estos módulos y que puede contener cualquier otra aplicación en el mismo diseño, lo cual depende del diseñador.

Una vez modificado el código, el diseño puede Sintetizarse, Implementarse y Bajarse al FPGA que el diseñador haya decidido, mediante las herramientas que Xilinx proporciona en el ISE.

Consideraciones para el diseño de la división

A lo largo de todo el trabajo se ha dicho que la velocidad de cada etapa del control difuso, en un FLC, bajo el enfoque RTC, depende de la velocidad del algoritmo de la división implementado, ya que suele ser el módulo más lento del sistema. Los módulos de división utilizados en este trabajo, en su versión genérica, se encuentran en el CD adjunto a esta tesis.

Los algoritmos descritos en el Anexo 1, es decir, la División Con restauración y la División Sin Restauración, son considerados métodos lentos [56], así como el algoritmo SRT; todos los Algoritmos por Recurrencia de Dígito son considerados lentos dependiendo de la base utilizada; entre más alta sea la base, resultados más rápidos pueden obtenerse hasta cierto punto, ya que la manera de determinar el dígito se complica cuando el número de la base es alto, lo cual puede provocar un deterioro en el rendimiento. También existen métodos rápidos como el de Newton Raphson y el de Goldschmidt, que pueden representar una opción bastante interesante para incrementar el rendimiento del FLC, ya que no existe alguno que utilice estos métodos.

Lo que debe considerarse cuando se quiere implementar el algoritmo de la división para una aplicación determinada, es la cantidad de recursos que utilizará y por supuesto, el tiempo que tarda en procesar dicha división. Para un sistema que requiere realizar pocas divisiones, un algoritmo rápido que consume muchos recursos de un FPGA puede ser suficiente para obtener el resultado deseado, justificando el alto uso de recursos con la disminución tiempo. Para un FLC con la arquitectura propuesta en este trabajo, el uso de un algoritmo que consuma muchos recursos del FPGA no es conveniente debido a que

un módulo de división es usado por cada función de membresía en el difusificador y por cada consecuencia que entra por el desdifusificador, si se utiliza en desdifusificador 2. De modo que si se requiere que el sistema crezca para aumentar su precisión, incrementando el número de funciones de membresía por cada entrada y salida, entonces se agotarían rápidamente los recursos del FPGA utilizado.

Por lo tanto, es necesario elegir inteligentemente el tipo de división que se usará, si es que se quiere mejorar el rendimiento de determinado sistema. Con mayor razón, cuando se trata de un sistema que puede atender señales que varían en el orden de nanosegundos, es decir sistemas en tiempo real, como es el caso de un FLC.

Como conclusión, es necesario poner en la balanza el consumo de recursos y la velocidad de procesamiento para una aplicación, donde se quiera realizar una división.

Consideraciones para el diseño en VHDL

Existen muchos trabajos y libros dedicados a instruir al programador a realizar código VHDL eficiente que *sintetice*, independientemente del programa de Síntesis y del FPGA de cada fabricante. En ésta sección se describen sólo aquéllos inconvenientes que se fueron encontrando a lo largo del desarrollo de la arquitectura del FLC.

Inicialmente el FLC fue diseñado en gran parte con circuitos secuenciales, pero la división siempre representó un problema para implementarla, ya que el algoritmo utilizado obtenía un dígito de la división por cada ciclo de reloj. Esto complicó el procesamiento.

Se sugiere que cuando se usen circuitos secuenciales, cada proceso posea la menor cantidad de hardware posible, ya que esto imposibilita el incremento de la frecuencia de reloj para obtener determinado resultado. Debido a esto, el diseño se reorientó hacia el uso de circuitos combinatorios.

En VHDL, los *procesos* no son exclusivos de los circuitos secuenciales, aunque frecuentemente se utilice en su forma más sencilla, un Flip–Flop. A partir de esto, el algoritmo de la división implementado sufrió grandes cambios, pues los procesos síncronos que utilizaba se cambiaron por un proceso asíncrono, es decir, un circuito combinatorio. Por ejemplo, cualquiera de los dos códigos VHDL, mostrados a continuación, son absolutamente válidos:

A

```
architecture Behavioral of SUM is
begin
        process(CLK, RESET, A, B) begin
                if RESET = '1' then
                        C<= conv_std_logic_vector(0, n);
                elsif rising_edge(CLK) then
                        C<= A + B;
                end if;
        end process;
end Behavioral;
```

B

```
architecture Behavioral of SUM is
begin
        process(A, B) begin
                C<= A + B;
        end process;
end Behavioral;
```

Código 4. Proceso síncrono y proceso asíncrono.

La diferencia entre el código 5A y el 5B consiste en que el circuito 5A, síncrono, realiza la operación de suma, cada vez que existe un flanco de subida en el reloj CLK, por medio de un Flip–Flop. El código 5B es válido también, sin embargo, el proceso asíncrono podría o no estar en el código, ya que existe una sola instrucción dentro de él. Si existieran más instrucciones dentro del proceso asíncrono se asegura que, en la simulación, se ejecuten de manera ordenada cada una de las instrucciones y se pueda observar el comportamiento del circuito paso por paso, es decir, facilita la depuración del circuito. Pero si se implementa en un FPGA, el procesamiento no se ejecuta de manera ordenada, sino dependiendo del tiempo de retardo de propagación de cada elemento que infiera el compilador.

Como puede observarse en los Códigos 2–4, todos los procesos son asíncronos y poseen muchas instrucciones dentro de él. Todos usan la instrucción **`for-loop`**, para replicar el código que hay dentro de ella y si el resultado anterior de una variable es usado después de cada iteración, entonces esas replicaciones en hardware son conectadas en cascada, una con otra, lo cual incrementa el tiempo de procesamiento considerablemente, a diferencia de los circuitos secuenciales; con lo que se necesita un solo circuito secuencial para realizar las operaciones del bucle **`for`** de cualquiera de los códigos antes mencionados, para obtener el mismo resultado, pero utilizaría *n* iteraciones para lograrlo.

También pudo observarse que el uso de *variables* dentro de cada proceso asíncrono, sí *sintetizan* de manera adecuada, por lo menos para el cálculo de la división y para procesos similares donde se requiere que el resultado de una iteración se utilice en la siguiente.

Se recomienda, si se hace uso de expresiones condicionales, tales como **`if-elsif`**, **`with-when`**, **`when-else`** y **`with-select-when`**, se consideren

absolutamente todos los casos posibles, pues de lo contrario el compilador inferirá un *latch* a la salida del conjunto de multiplexores, automáticamente, sin tomar en cuenta la decisión del diseñador. Por lo que, si el programador sólo espera que se cumpla una sola condición, entonces debe evitar el uso de la sentencia **`elsif`** y terminar la instrucción con un **`end if`** (sólo dentro de un proceso, porque fuera de él provoca un error). El compilador interpreta que es una instrucción condicional, en la que si se cumple la condición, entonces se ejecuta la instrucción; y si no se cumple, entonces no se infiere un *latch* a la salida de este multiplexor, porque ese resultado no será utilizado. Esto no sucede con las instrucciones restantes, en las que es forzoso terminar la ejecución condicional con un **`else`**, ya que si no se hace esto, entonces el compilador inferirá un *latch* a la salida de este conjunto de multiplexores. Para el caso de la instrucción **`with–select–when`**, es necesario utilizar la instrucción **`others`** para considerar el caso no esperado y evitar que se infiera un *latch*.

Si el usuario no está familiarizado con el lenguaje VHDL, la referencia [57] proporciona información importante acerca de lo que se puede hacer con esta herramienta, desde conocer las instrucciones básicas hasta entender el funcionamiento en hardware que implica, así como conocer de manera general en qué sintetiza (su equivalente en hardware) una ó varias instrucciones de un diseño.

El diseño del FLC de este trabajo, así como de la división, no está limitado al uso de VHDL. El diseño puede realizarse en cualquier lenguaje de descripción de hardware, como el reconocido *Verilog*, ya que el algoritmo que se sigue para el diseño es el mismo. La referencia [58] proporciona mayor información al diseñador acerca de la síntesis de un programa determinado, para describir un sistema digital, independientemente del dispositivo FPGA utilizado.