UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TESIS DOCTORAL

DISEÑO DE CONTROLADORES

DEDICADOS A LA LÓGICA DIFUSA

AUTOR: HÉCTOR NELSON ACOSTA

DIRECTOR: JAVIER GARRIDO SALAS

OCTUBRE DE 2006
MADRID – ESPAÑA

TRIBUNAL:

PRESIDENTE: DR.
PROFESOR

VOCALES: DR.
PROFESOR

DR.
PROFESOR

DR.
PROFESOR

SECRETARIO: DR.
PROFESOR

SUPLENTES: DR.
PROFESOR

DR.
PROFESOR

CALIFICACIÓN:

FECHA DE LECTURA:

A Florencia, Giuliano, Antonela, Guido y Lucia :

TABLA DE CONTENIDOS

INDICE DE TABLAS

INDICE DE FIGURAS

AGRADECIMIENTOS

Este trabajo se inició de forma efectiva durante 1995, después de haber gastado algún tiempo en el estudio previo de otros temas que no llegaron a cuajar lo suficiente. Cuando un trabajo se alarga tanto en el tiempo, la lista de personas que merecen ser destacadas en estas líneas aumenta, así como la probabilidad de cometer olvidos que espero me sean disculpados.

En primer lugar deseo expresar mi más profundo y sincero agradecimiento a tres personas: *Eduardo Boemo*, *Javier Garrido* y *Jean-Pierre Deschamps*, maestros y compañeros. Sin el apoyo, capacidad y paciencia, probablemente este trabajo no se habría acabado.

Asimismo, a *Gery Bioul*, *Hugo Curti*, *Gustavo Sutter*, *Elías Todorovich*, y *Marcelo Tosini*, integrantes del grupo de *Arquitectura de Computadoras*, con los que después de tanto tiempo y tantas horas de divertido trabajo y aprendizaje conjunto me une, en muchos casos, una relación más allá de lo estrictamente profesional.

A los compañeros de la UAM, tanto de la Escuela de Informática como a los de la Facultad de Ciencias, que me han asistido tanto en lo académico como en lo personal.

A las autoridades de la Facultad de Ciencias Exactas de la UNICEN, quienes también pusieron su granito de arena para facilitar el desarrollo de esta tarea.

A todos los compañeros del Departamento de Computación y Sistemas de la UNCPBA que en algún momento me han preguntado: "¿Qué? ¿Y...? ¿Cuándo la terminas?"

Finalmente, quiero agradecer a mis padres y amigos por haber sido tan pacientes conmigo.

RESUMEN

Los controladores difusos (FLC) pueden ser materializados en distintas plataformas de acuerdo a las prestaciones que se requiera, por hardware estándar con varios niveles de software, por microcontroladores dedicados o por circuitos de aplicación específica microprogramados. El diseño de aplicaciones en circuitos a medida hace que las tareas de diseño sean muy costosas. Con el objetivo de reducir el tiempo de desarrollo de los sistemas de control basados en lógica difusa, se proponen dos arquitecturas y un conjunto de herramientas que asisten al diseñador en la materialización. Se comienza con una descripción de alto nivel que especifica el controlador, y se generan automáticamente ambas arquitecturas, tanto en software como en hardware.

Las arquitecturas propuestas cubren dos enfoques de implementación. Una arquitectura propone un reducido consumo de recursos, utilizando una única unidad de cálculo con operaciones básicas de ocho bits, donde la aplicación se especifica por medio de un microprograma secuencial. La otra arquitectura utiliza múltiples unidades de cálculo con anchos de palabra adaptados a la aplicación, y cuyo motor de inferencia se basa en reglas activas para obtener mejores prestaciones.

Los FLC software pueden usarse como prototipo para ajustar el diseño o como materialización definitiva. De esta manera, se asiste al diseñador desde la especificación hasta el circuito en VHDL con su correspondiente microprograma. Se presenta un análisis de espacio y velocidades sobre plataformas FPGA.

ABSTRACT

Fuzzy logic controllers can be implemented using different platforms; standard hardware with several software levels, dedicated microcontroller or microprogrammed specific application circuits. To design applications using customized circuits makes the implementation too expensive.

Two architectures and a set of tools to aid the designer are proposed to reduce the FLC developing time. A high level specification file is used for the automatic generation of both architectures: software and hardware.

The proposed architectures cover two different approaches for implementation . One of them proposes a reduced consumption of resources, based on one calculus unit with 8-bit basic operations, where the aplication is specified by a secuential microprogram. The other one uses multiple computer units with word length defined by the application, with the inference motor based on active rules to obtain a better performance.

FLC software can be used as prototype to adjust the application design or as definitive implementation. Thus, the desginer is assisted from the controller specification until the VHDL circuit, with its microprogram is complete. An area and speed analysis over different platforms is also presented.

1 - INTRODUCCIÓN

1.1 - DESCRIPCIÓN DEL PROBLEMA

El enfoque tradicional de desarrollo de sistemas de control dedicados a una aplicación concreta, consiste en elegir una plataforma hardware y un sistema operativo, y en generar programas de aplicación ejecutables en la configuración escogida. Es una solución cómoda dada la disponibilidad tanto de plataformas hardware estándar (tarjetas con procesadores, memorias, interfaces con buses normalizados, puertos de entrada y salida, etc.) como de herramientas software (sistemas operativos, compiladores, depuradores, etc.). Sin embargo padece del inconveniente de que los programas de aplicación tienen que adecuarse a la plataforma hardware y al sistema operativo que hayan elegido.

Una solución más personalizada consiste en desarrollar nuevas tarjetas basadas en microprocesadores y protocolos de conexión estándares. Puede ser una solución impuesta por varios motivos: compatibilidad con señales de entrada y salida externas, prestaciones, costo, tamaño, etc.; aunque no modifica el hecho de que el desarrollo de los programas de aplicación siga siendo una tarea costosa. Lo descrito anteriormente añade el inconveniente de que el desarrollo de la plataforma hardware tiene que preceder al desarrollo de los programas; por lo tanto el habitual e iterativo ciclo de desarrollo mostrado en la Fig. 1.1 incluye la fabricación de sucesivos prototipos de la plataforma hardware.

Técnicas recientes, tales como simulación al nivel de sistemas, ingeniería concurrente [Ver97], co-diseño hardware/software [Sta97, Tod99], permiten desarrollar las plataformas hardware y los programas de forma concurrente sin tener que (por lo menos durante las primeras etapas) fabricar prototipos del hardware.

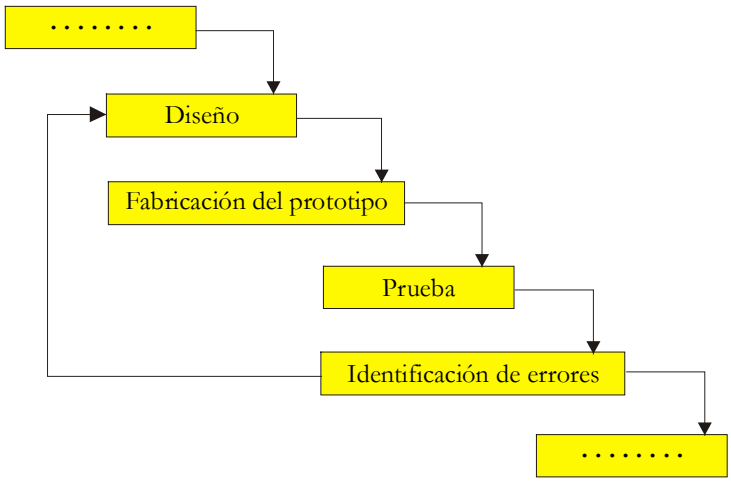


Fig. 1.1 – Ciclo de diseño

En lugar de diseñar tarjetas basadas exclusivamente en componentes estándares existe también la posibilidad de utilizar componentes a medida: ASIC[1], FPGA[2] o PLD[3]. Las herramientas informáticas disponibles hoy en día (bibliotecas de células, compiladores de macro-células, programas de síntesis, simuladores, etc.) hacen que el diseño de un ASIC o de una FPGA sea una tarea cuyo nivel de dificultad se asemeje al del desarrollo de programas de complejidad relativamente baja. Partiendo de la observación de que los productos propuestos por los fabricantes de semiconductores (ASIC, FPGA, bibliotecas, compiladores) y por los vendedores de programas de desarrollo (sintetizadores, simuladores, etc.) permiten diseñar componentes a medida con un esfuerzo moderado, surge la idea de sustituir el conjunto: "*computadora / sistema operativo / programas de aplicación específicos*", por un conjunto de procesadores a medida ejecutando (micro)programas específicos.

Algunos de los motivos para la utilización de componentes a medida tipo ASIC o FPGA son entre otros, el bajo costo unitario en medianos y grandes volúmenes de producción, el tamaño físico pequeño (es posible integrar en un solo chip la lógica de un problema), la posibilidad de obtener grandes prestaciones, y la seguridad de evitar la copia no autorizada del diseño. Existen en el mercado muchos proveedores de este tipo de tecnología, con una gran variedad de productos.

[1] ASIC: Application Specific Integrated Circuits

[2] FPGA: Field Programmable Gate Arrays

[3] PLD: Programmable Logical Devices

Este trabajo se ha llevado a cabo, fundamentalmente, en las instalaciones del Grupo de "*Hardware y Arquitectura de Computadoras*" del *INstituto de Tecnología Informática Aplicada* (INTIA) en Tandil. El grupo tiene como principales actividades la investigación y la transferencia de tecnologías en el campo de los ASIC o FPGA. Dichas actividades se vinculan con este estudio en el marco de tres proyectos: ASICTAN (*Aplicaciones eSpecíficas Integradas en Circuitos de Tecnología AvaNzada*), ALIASEMEH (*ALgoritmos de Inteligencia Artificial para Sistemas Empotrados: MEtodologías y Herramientas*) y AEIOU (*Algoritmos Empotrados Inteligentes de Orientación Universal*) han tenido como objetivo el desarrollo de una metodología de diseño de sistemas dedicados (sistemas empotrados[4]). Ambos proyectos han sido financiados por la Secretaría de Ciencia y Técnica de la UNICEN[5], mientras que IBERCHIP (programa de la Unión Europea para el fomento de la μ-electrónica en América latina) ha facilitado gran parte del material.

Dentro del estudio metodológico del diseño de sistemas dedicados de los proyectos, este trabajo emprende el desarrollo de metodologías y herramientas de síntesis de controladores difusos a medida sobre plataformas ASIC y FPGA. Dicha elección se justifica en el hecho de que los algoritmos de control difusos pertenecen a una clase particular de algoritmos, los esquemas de cálculo, cuya materialización es más sencilla que la de algoritmos tradicionales que incluyen saltos y bifurcaciones. Por otra parte existe una demanda efectiva para este tipo de controlador que tiene que procesar un número alto de reglas en un intervalo de tiempo reducido.

1.2 - LÓGICA DIFUSA

La lógica difusa puede ser una solución alternativa a la creación de modelos matemáticos de sistemas de control de procesos, permitiendo emular en cierto sentido el pensamiento humano. No sólo existe un gran número de realizaciones concretas de dicho tipo de controlador publicadas en congresos y revistas especializadas, sino que además existe gran demanda e interés de la industria.

[4] Embedded systems.

[5] UNICEN: Universidad Nacional del Centro de la Provincia de Buenos Aires, (7000) Tandil, Provincia de Buenos Aires, Argentina.

"¿Porqué utilizar control difuso?". Considerando las discusiones en la literatura, en los *newsgroup*[6] y en los *mailing-list*[7] las justificaciones encontradas se pueden resumir en cuatro razones o mitos que son analizados en esta misma sección:

1) El control difuso es una tecnología nueva y puede usarse para eliminar los reclamos de patentes de soluciones similares para problemas técnicos basadas en otra técnica diferente.

2) Se han realizado estudios que determinan que los consumidores quieren lógica difusa, porque la tratan como un sinónimo de alta tecnología.

3) El desarrollo de controladores difusos es más fácil de aprender y requiere menos personal que el desarrollo de controladores convencionales, lo que resulta en una producción más barata.

4) Los controladores difusos brindan mayor robustez que el control convencional.

Las razones 1 y 2 responden a estrategias y comportamientos de mercado, lo cual las aleja del aspecto técnico y metodológico al cual esta tesis se enfoca, por lo tanto no se analizan.

Los controladores difusos se describen por reglas *IF-THEN*, por lo cual brindan una representación del conocimiento sencilla y comprensible. Se pueden ver como un lenguaje de (muy) alto nivel, representado por reglas, que un compilador toma para generar un algoritmo de control. Es una ventaja que estrategias de control complejas, conocidas por operadores o ingenieros en forma de experiencia, puedan ser programadas y mantenidas de una forma sencilla y comprensible.

Por lo general se dice que los controladores difusos proveen mayor robustez, aunque no hay resultados de investigación que prueben que son más robustos que los

[6] El principal grupo de noticias sobre sistemas difusos es: comp.ai.fuzzy.

[7] La principal lista de discusión en el dominio de sistemas difusos es: fuzzy-mail@vexpert.dbai.tuwein.at.

convencionales. Cuando la variación de parámetros de un proceso es parcialmente conocida, un controlador difuso puede ser diseñado para ser menos sensible a los cambios de tales parámetros, y así se obtiene un controlador más robusto que uno convencional [Men95]. De esta forma, *son más robustos* debe ser interpretado como *pueden ser más robustos*.

1.3 - JUSTIFICACIÓN DE LA TESIS

El lector de esta tesis podría preguntarse: "¿Porqué otro trabajo sobre control difuso?". Considerando la gran cantidad de libros, conferencias, actas de congresos, esta pregunta puede parecer legítima. Aunque, por un lado, hay varios artículos y libros que son muy matemáticos y no se enfocan hacia el hardware para el control [Kan93, Mun94, Jan95, Hat96, Lin96, Zen96, Cas99, Kas99, Ori99, Sch99, Dra00, Bog01, Cha01, GNM01, FSu03, Hua03, Muc03, Zho03]. Por el otro lado, algunos autores enfocan la problemática como simples problemas de control o aplicaciones, y son incompletos con respecto al diseño de la plataforma hardware que materialice la aplicación eficientemente [Kos92, Hec96, Hil96, Wan96, Kan98, Kov99, Pat99, Set99, Kar00, Mor00, Num00, Son00, Bar01, Pir01, MLi01, OdM01, YHL01, YuJ01, Yia01, FLa02, JIH02, KCT02, Uan02, VGo02, JYi02, Ma03, Wai03, Xu03, Zim03, Bon03, Bie03].

En la bibliografía e incluso en algunas aplicaciones comerciales que tratan la definición de ambientes para el desarrollo de controladores difusos que sólo llegan hasta la definición del software para algún controlador [Mokon, Aptro, Archi, Moo96, MotFL, Mur98, Mas94, Bat98e, Son99, Ma00, ShT02, GAr02, CWL03, Sar03, JLe03], o que asisten al diseñador en la selección de parámetros y reglas mediante simuladores o herramientas de inteligencia artificial [Kos92, Mas94, Car98, Klo99, Paw99, Pit99, Yon99, Kov00, Teo00, Pir01, GuG00, Bar01, HPH02, AlK03, Los03].

Este trabajo se enfoca al desarrollo de plataformas hardware que materialicen controladores difusos, propone una metodología de diseño que se apoya en dos arquitecturas dedicadas y un conjunto de herramientas de asistencia. Se proponen arquitecturas sintetizables descritas en VHDL con una o múltiples unidades de

cálculo, que materializan la inferencia difusa utilizando todas las reglas o por medio de reglas activas. Para analizar la propuesta se describe la realización de dos controladores concretos. El objetivo es doble, sirve para desarrollar y probar las herramientas útiles para la síntesis de controladores basados en lógica difusa, y es un banco de prueba para la metodología propuesta. El primero de los controladores forma parte de un sistema de conducción automática de un vehículo, mientras que el segundo está destinado a controlar un sistema en equilibrio inestable de un péndulo invertido. Numerosas realizaciones concretas y teóricas de dichos controladores se han publicaron en congresos y revistas especializadas por consiguiente constituye un interesante banco de prueba: [Ngu89, Kos92, Bat98e] para el primero y [Can67, Son99, Set99, Son00, Yia01, Cha01, JYi02] para el segundo. Por otra parte, en el Apéndice B se muestran las materializaciones de 9 ejemplos adicionales.

1.4 - OBJETIVOS DE LA TESIS

El objetivo general de esta tesis es el estudio de la metodología de diseño, análisis y síntesis automática de arquitecturas para resolver de forma eficiente la materialización de controladores difusos digitales. A tal fin se proponen modelos arquitecturales como referencia para el establecimiento de una guía metodológica orientada a este tipo de materializaciones. Este objetivo general se concreta en los siguientes objetivos particulares.

1.4.1 - METODOLOGÍA GENERAL DE SÍNTESIS

La metodología se desarrolla y valida en paralelo con el diseño de las arquitecturas, y sus pasos y consideraciones se ilustran con un problema particular pero es extensible a problemas genéricos. El objetivo básico de esta metodología es acortar el ciclo de diseño de controladores, reduciendo costes sin disminuir la calidad de los resultados, para poder mantener la competitividad impuesta por el mercado.

La metodología se basa principalmente en tres enfoques:

- Un conjunto de herramientas que asisten al diseñador para, especificar el comportamiento del controlador en un lenguaje muy cercano al del usuario, y transformar esa especificación inicial en el microprograma que controla la ruta de datos de la arquitectura. En esta etapa, se realizan también optimizaciones de las operaciones aritméticas y reticulares, al igual que la asignación óptima de registros buscando minimizar la memoria necesaria.

- La especificación VHDL de la arquitectura, flexible y fácilmente modificable, con el objeto de poder ser configurada y adaptada con un mínimo esfuerzo a la aplicación.

- Un procedimiento de simulación y validación basado en herramientas comerciales con módulos VHDL que faciliten la depuración del microprograma y del diseño.

1.4.2 - ARQUITECTURA PASIVA

Proponer una arquitectura que permita materializar controladores difusos microprogramados basados en una *única* unidad de cálculo realizada a medida de la aplicación. Esta arquitectura se basa en un controlador hardware común a todas las posibles aplicaciones, que es adaptado a la aplicación por medio del (micro)programa.

1.4.3 - ARQUITECTURA ACTIVA O DE ALTAS PRESTACIONES

Proponer una arquitectura que permita materializar controladores difusos microprogramados de altas prestaciones utilizando *múltiples* unidades de cálculo realizadas a medida de la aplicación. Esta arquitectura debe optimizar la evaluación de reglas utilizando reglas activas, de tal forma que se evalúen sólo las reglas que aportan información relevante al cálculo de la función de salida.

1.4.4 – ANÁLISIS

Analizar, discutir y comparar soluciones arquitecturales y requerimientos del algoritmo, en cuanto al número y tipo de operaciones realizadas, y al flujo y estructura de datos.

1.5 - PRINCIPALES CONTRIBUCIONES

Los resultados obtenidos en esta tesis han dado origen a contribuciones en tres áreas, la arquitectura de controladores difusos con dos propuestas (una pasiva y otra activa o de altas prestaciones), y la fusificación por segmentos. También se desea destacar como una contribución el diseño de un circuito integrado de aplicación específica que materializa la arquitectura de una de las propuestas.

1.5.1 - ARQUITECTURAS PARA CONTROLADORES DIFUSOS

Se diseñan y presentan minuciosamente dos arquitecturas para controladores difusos. En la primera propuesta se mantiene el hardware adaptándose sólo el tamaño de palabra de la ruta de datos, mientras que la aplicación se materializa por medio de un microprograma. En la segunda propuesta, el microprograma sufre muy pocas variaciones para adaptarse a la aplicación, mientras que el hardware es totalmente dedicado a la aplicación. Ambas constan de una ruta de datos y un único nivel de programación el (micro)programa.

- La primera arquitectura propuesta se basa en una única unidad de cálculo controlada por un microprograma. La unidad de cálculo tiene un conjunto de operaciones que permite ejecutar un algoritmo de control difuso.

- La segunda arquitectura propuesta se basa en múltiples unidades de cálculo (dedicadas y con un ancho de palabra suficiente como para realizar operaciones de múltiples anchos de palabra en un solo ciclo) controlada por un microprograma. Este diseño materializa un motor de inferencias difusas optimizado, que trabaja por reglas activas o dirigido por reglas, donde sólo se

evalúan las reglas que aportan información para el cálculo de la salida.

1.5.2 - FUSIFICACIÓN POR SEGMENTOS

Se propone un esquema de cálculo para funciones de pertenencia de variables de entrada, que permite sólo calcular las funciones cuyo valor no sea nulo. Este esquema facilita el diseño de controladores de alta velocidad, aunque se requieran funciones de pertenencia complejas realizadas con un número variable de segmentos.

1.5.3 - DISEÑO DE UN PRIMER INTEGRADO

Se realiza la materialización de un circuito integrado de aplicación específica (ASIC) con el diseño de una arquitectura básica que trabaje por reglas pasivas. Este acontecimiento, de haberse realizado en una facultad de electrónica, hubiera pasado desapercibido pero, se realizó en una Facultad de Ciencias Exactas, con una carrera orientada a los sistemas, y principalmente a la parte informática de dichos sistemas. El haber diseñado un integrado fortaleció al grupo, fomentando los temas de microelectrónica, el diseño de sistemas digitales y el diseño de aplicaciones específicas (ASIC / FPGA) en la Facultad de Ciencias Exactas de la UNCPBA (Tandil, Argentina). Por otra parte, indirectamente ha impulsado la inclusión de algunos cursos de grado optativos, tales como: "Técnica de FL en Control Automático" dictado por 3 años con un promedio de 10 alumnos, "Algoritmos Hardware" dictado por 3 años con 20 alumnos inscriptos promedio, y "Programación Hardware" dictado por 2 años con un promedio de 15 alumnos.

1.6 - ORGANIZACIÓN DE LA TESIS

El trabajo realizado no queda reflejado cronológicamente en el orden de exposición de esta Memoria. Muchas de las tareas han sido desarrolladas en paralelo, pero se ha intentado organizar su exposición de forma que la lectura sea fácilmente comprensible por el lector.

Esta tesis puede ser dividida en tres grandes bloques para su lectura. Un bloque formado por el capítulo 2, que resume los estudios realizados, sustenta

teóricamente la tesis en el área del control difuso, con un resumen de los principios de la lógica difusa, su aplicación al control y las posibles arquitecturas para las materializaciones hardware. El segundo bloque propone el diseño de arquitecturas hardware de controladores difusos; el capítulo 3 define las bases sobre las cuales se trabaja en la parte experimental de esta tesis; los capítulos 4 y 5 muestran el diseño de una arquitectura hardware a ser adaptada mediante el microprograma; y los capítulos 6 y 7 una arquitectura hardware cuya ruta de datos y unidades de cálculo son realizadas de manera automática y completamente a medida de la aplicación lo que simplifica el microprograma. El tercer bloque, formado por el capítulo 8 y 9, muestran el análisis de materializaciones de las arquitecturas y describen las conclusiones de esta tesis.

El capítulo 2 es una introducción a la lógica difusa, que sin ser exhaustiva, recorre las bases para la materialización hardware de controladores difusos. En el capítulo 3 se plantean los objetivos a tener en cuenta por los controladores a desarrollar, con respecto al conjunto de instrucciones, al cálculo de las funciones de pertenencia, y al paralelismo.

En el capítulo 4 se analiza y diseña una arquitectura que materializa un controlador de forma completa. Se define el conjunto de instrucciones. El capítulo 5 describe las herramientas desarrolladas para la asistencia en el desarrollo de una arquitectura básica trabajando con reglas pasivas.

En el capítulo 6 se muestra una arquitectura de altas prestaciones, que hace uso del paralelismo inherente en el algoritmo de control difuso. Además, utiliza un motor de inferencias difusas que trabaja por reglas activas, donde sólo se evalúan las reglas que contribuyen a hallar la solución. El capítulo 7 describe las herramientas que asisten al diseñador que utiliza la arquitectura de altas prestaciones.

El capítulo 8 describen y analizan varias materializaciones de las arquitecturas propuestas, utilizando como plataforma a la familia Spartan 2 de Xilinx.

En el capítulo 9 se realiza un análisis y se comentan las conclusiones del trabajo desarrollado. También se proponen líneas de trabajo que permitirán continuar con el

trabajo de esta tesis a futuro.

Esta tesis se complementa con cuatro apéndices que presentan información que ayuda a definir el contexto de la tesis y una materialización concreta. El apéndice A presenta una breve descripción de las herramientas que asisten al diseñador de FLC. El apéndice B presenta varios ejemplos desarrollados usando las herramientas propuestas. El apéndice C muestra los esquemas Viewlogic de todos los circuitos de la arquitectura materializada en el capítulo 4. El apéndice D describe detalladamente el funcionamiento de las arquitecturas propuestas mediante el análisis de diagramas de señales durante una simulación funcional.

2 – LÓGICA DIFUSA

2.1 - INTRODUCCIÓN A LA LÓGICA DIFUSA

En un artículo escrito en 1961, *Lofti A. Zadeh* menciona que se necesita una técnica nueva, un tipo especial de matemática, que considere valores lógicos multivaluados, pero hubo que esperar a que en 1965 publicara el primer artículo de lógica difusa [Per95].

Lógica difusa[1] (FL) es una lógica multi-valuada que permite (por medio de conjuntos de pertenencia) una forma más práctica de enfocar los problemas como se ven en el mundo real. Al contrario que la información binaria (si/no), la lógica difusa emula la habilidad de razonamiento y hace uso de datos aproximados para encontrar soluciones precisas.

Se puede configurar un sistema difuso para mapear entradas en salidas, con el mismo propósito que cualquier otro sistema de computación [Aco97]. Básicamente, consiste de 3 etapas: fusificación, evaluación de reglas y defusificación.

Fusificación, es un proceso de traducción para obtener la representación difusa a partir de los valores actuales o *crisp* (por ejemplo, *temperatura*), para lo cual utiliza las funciones de pertenencia. Evaluación de reglas o inferencia difusa, es la forma de producir respuestas numéricas difusas a partir de reglas lingüísticas aplicadas a los valores difusos de entrada. Defusificación, es un proceso de traducción que permite obtener un valor numérico representativo de todas las salidas a partir de la información difusa que produce la evaluación de reglas.

2.1.1 - VENTAJAS DE LA LÓGICA DIFUSA

- No requiere construcciones matemáticas complejas.

[1] Lógica difusa: Fuzzy Logic (FL).

- Uso de lenguaje natural
- Facilidad de configuración
- Si bien la inferencia se realiza por medio de lógica difusa, se obtienen resultados precisos
- Fácil adaptación a trabajos en colaboración con otras técnicas

2.1.2 - DESVENTAJAS DE LA LÓGICA DIFUSA

- Se debe entender y ser capaz de definir el problema
- Se deben evaluar y ajustar los resultados

2.2 - CONOCIMIENTO DEL PROBLEMA

Muchos problemas tienen distintas formas de conocimiento: objetivo y subjetivo. 1) El *conocimiento objetivo* es usado para la formulación de problemas de ingeniería, basado en modelos matemáticos. 2) El *conocimiento subjetivo* representa información lingüística, que normalmente es imposible cuantificar para ser usada en modelos matemáticos tradicionales, tales como reglas, información de expertos, requerimientos de diseño.

El conocimiento subjetivo es usualmente ignorado al comienzo de los diseños de ingeniería, pero se usa frecuentemente para evaluar tales diseños. Ambos tipos de conocimiento pueden y deberían ser usados para resolver problemas reales. Las dos formas de conocimientos pueden ser coordinadas de forma lógica usando FL.

2.3 - SISTEMA DIFUSO

En general un *sistema basado en lógica difusa* (FLS) es un mapeo no lineal de datos de entrada en una salida escalar. En este capítulo se muestra como interpretar el mapeo no lineal de un FLS geométricamente, como es hecho

comúnmente en la literatura de control difuso, y también como escribir una fórmula detallada para la relación de entrada y salida.

Un sistema de control difuso mapea entradas *crisp* en salidas *crisp*. Contiene 4 componentes: reglas, fusificador, motor de inferencia y defusificador. Una vez que las reglas han sido establecidas, un FLS puede ser visto como un mapeo que puede ser expresado cuantitativamente como $\boldsymbol{Y} = f(\boldsymbol{X})$.

Las *reglas* pueden ser provistas por expertos o extraídas de datos numéricos. En todo caso, las reglas son expresadas como una colección de sentencias IF-THEN. De las reglas se debe conocer:

1) Variables lingüísticas en contraposición a los valores numéricos de una variable (*muy_calido* VS 36 grados centígrados). El valor numérico 36º C es interpretado de acuerdo a funciones de pertenencia asociadas a cada variable lingüística, y su semántica permite representar con un valor difuso cuanto pertenece el valor numérico a cada variable lingüística.

2) La cuantificación de variables lingüísticas que es realizada usando funciones de pertenencia (la variable u_1 debe tomar un valor de pertenencia al término lingüístico asociado, en el rango *muy_calido* a *muy_frio*). Si la variable temperatura tiene las variables lingüísitcas: *muy_calido*, *calido*, *frio* y *muy_frio*; el valor 36º puede tomar los valores *muy_calido*=1, *calido*=0.3, *frio*=0 y *muy_frio*=0.

3) Los conectores lógicos para variables lingüísticas (and, or, ...).

4) Las implicaciones (IF *a* THEN *b*).

5) Como combinar un conjunto de reglas.

El *fusificador* mapea valores numéricos *crisp* en conjuntos difusos. Es necesario para activar reglas, que están en términos de variables lingüísticas y tienen asociados conjuntos difusos. Si la variable temperatura toma el valor 36º C, al fusificar el valor se obtiene la pertenencia a todos los conjuntos difusos, de tal forma que puede ser válido que *muy_calido*=1, *calido*=0.3, *frio*=0 y *muy_frio*=0.

El *motor de inferencia* de un FLS mapea conjuntos difusos en conjuntos difusos. Maneja la forma en la cual las reglas son combinadas. Como los humanos usan distintos tipos de procesos de inferencia, para entender cosas o tomar decisiones, un FLS puede usar distintos procedimientos de inferencia difusa. Solo se usan unos pocos procedimientos en aplicaciones FL.

El *defusificador* mapea conjuntos de salida difusos en números *crisp*. En aplicaciones de control, tales números corresponden a la acción de control a tomar.

2.4 - CONJUNTOS

2.4.1 - CONJUNTOS CRISP

Llamamos conjunto *crisp A*, en el universo de discurso *U*, al conjunto de valores permitidos que puede tomar una variable; puede ser definido por la enumeración de todos sus miembros o identificando los elementos $x \subset A$. Así *A* puede ser definido como: $A = \{ x \mid x$ cumpla una condición $\}$. Alternativamente podemos introducir una función característica (o de pertenencia) para *A*, denotada $\mu_A(x)$, tal que $A \Rightarrow \mu_A(x) = 1$, si $x \in A$, y $\mu_A(x) = 0$, si $x \notin A$. El subconjunto *A* es matemáticamente equivalente a su función de pertenencia $\mu_A(x)$, ya que conociendo $\mu_A(x)$ se conoce *A*.

Si *A* y *B* son dos subconjuntos de *U*, la unión de *A* y *B*, denotada por $A \cup B$, contienen todos los elementos de *A* y de *B*. La intersección de *A* y *B*, denotada como $A \cap B$, contiene todos los elementos que están simultáneamente en *A* y *B*. Si $\neg A$ denota el complemento de *A*, contiene todos los elementos que no están en *A*. De estos hechos se desprende que:

$$\begin{aligned} A \cup B \Rightarrow \mu_{A\cup B}(x) &= \max[\, \mu_A(x), \mu_B(x) \,] \\ A \cap B \Rightarrow \mu_{A\cap B}(x) &= \min[\, \mu_A(x), \mu_B(x) \,] \\ \neg A \Rightarrow \mu_{\neg A}(x) &= 1 - \mu_A(x) \end{aligned}$$

Las fórmulas $\mu_{A\cup B}(x)$, $\mu_{A\cap B}(x)$ y $\mu_{\neg A}(x)$ son muy usadas para probar otras propiedades de los conjuntos. Notese también que, *max* y *min* no son sólo formas

de describir $\mu_{A\cup B}(x)$ y $\mu_{A\cap B}(x)$, aunque estas fórmulas no son usualmente usadas en la teoría convencional de conjuntos, son esenciales [Kli88].

Las operaciones de unión e intersección *crisp* son: conmutativas [$A\cup B = B\cup A$], asociativas [$(A\cup B)\cup C = A\cup(B\cup C)$], y distributivas [$(A\cup B)\cap C = (A\cap C)\cup(B\cap C)$ y $(A\cap B)\cup C = (A\cup C)\cap(B\cup C)$]. Estas propiedades pueden ser probadas por diagramas de Venn o usando operaciones con funciones de pertenencia.

Las leyes de Morgan para conjuntos *crisp* son: $\neg[A\cup B] = \neg B\cap\neg A$ y $\neg[A\cap B] = \neg B\cup\neg A$. Estas leyes, que son muy usadas para probar operaciones más complicadas sobre conjuntos, pueden ser probadas por diagramas de *Venn* o usando funciones de pertenencia.

Las dos leyes fundamentales (Aristotélicas) de conjuntos *crisp* son:

- Ley de Contradicción: $A\cup\neg A = U$, y
- Ley de la Exclusión del Medio: $A\cap\neg A=\varnothing$.

2.4.2 - CONJUNTOS DIFUSOS

Un conjunto difuso F definido sobre un universo de discurso U es caracterizado por una función de pertenencia $\mu_F(x)$, que toma valores en el intervalo [0,1]. Un conjunto difuso es una generalización de un conjunto ordinario (cuyas funciones de pertenencia toman valores 0 ó 1). Una función de pertenencia brinda una medición del grado de similitud de un elemento en U al subconjunto difuso. En FL un elemento puede residir en más de un conjunto con diferentes grados de similitud, lo que no puede ocurrir en conjuntos *crisp*.

Por ejemplo si se analizan líquidos potables, se puede considerar que el *agua mineral* pertenece el 100% a dicho conjunto, la *cerveza* el 95%, mientras que el *ácido sulfúrico* el 0%. Así, la pertenencia a un conjunto puede interpretarse como que el *agua mineral* es similar a un líquido potable el 100%, la *cerveza* el 95% y el *ácido* el 0% [Men95].

Un conjunto difuso F en U puede representarse como un conjunto de pares ordenados de la siguiente forma:

$$F = \{ (x, \mu_F(x)) \mid x \in U \}, \qquad \text{siendo } \mu_F(x) > 0.$$

- Si U es continuo, $F = \int_U \mu_F(x) / x$.
- Si U es discreto, $F = \Sigma_U \mu_F(x) / x$.

Notación: los símbolos '∫' y 'Σ' denotan la colección de todos los puntos $x \in U$, y '/' denota la asociación de los elementos en U con sus grados de pertenencia

En FL, las operaciones de unión, intersección y complemento se definen en términos de sus funciones de pertenencia. Si los conjuntos difusos A y B son descritos por sus funciones de pertenencia $\mu_A(x)$ y $\mu_B(x)$, la definición de la unión difusa es la siguiente función de pertenencia:

$$A \cup B \Rightarrow \mu_{A \cup B}(x) = MAX[\mu_A(x), \mu_B(x)]$$

y la definición de la intersección difusa:

$$A \cap B \Rightarrow \mu_{A \cap B}(x) = MIN[\mu_A(x), \mu_B(x)].$$

Adicionalmente, la función de pertenencia para el complemento difuso es:

$$\neg A \Rightarrow \mu_{\neg A}(x) = 1 - \mu_A(x).$$

Obviamente, las tres definiciones están inspiradas en sus correspondientes funciones de pertenencia de conjuntos *crisp*. Aunque ambas funciones de pertenencia se vean iguales, debemos recordar que:

- Los últimos son conjuntos difusos mientras que los primeros son conjuntos *crisp*;
- Los conjuntos difusos pueden ser caracterizados sólo por sus funciones de pertenencia, mientras que los conjuntos *crisp*

> pueden ser caracterizados por sus funciones de pertenencia, por la descripción de sus elementos, o por la lista de sus elementos.

Los conjuntos difusos no cumplen las leyes de Contradicción y de Exclusión del Medio: $A \cup \neg A \neq U$, y $A \cap \neg A \neq \varnothing$. De hecho, una de las formas de describir diferencias entre teoría de conjuntos *crisp* y difusos es explicando que esas dos leyes no se cumplen.

2.4.3 - VARIABLES LINGÜÍSTICAS

Si u denota el nombre de una variable lingüística (ej. *temperatura*), los valores numéricos de la variable lingüística u son denominados x, donde $x \in U$. Algunas veces x y u son usados indistintamente, especialmente cuando una variable lingüística es una letra conocida en ingeniería. Una variable lingüística es usualmente descompuesta en conjuntos de términos T(u), los cuales cubren el universo de discurso.

Por ejemplo, el dominio de la variable lingüística *temperatura* (en el rango de 0 a 100 grados centígrados) puede ser representado por un conjunto de cuatro términos: *muy_cálido*, *cálido*, *frío* y *muy_frío*. Cada uno de los términos será evaluado por medio de una función de pertenencia, obteniéndose para cada valor de temperatura la similitud (o pertenencia) a cada una de las variables (*muy_cálido*, *cálido*, *frío* y *muy_frío*).

2.4.4 - FUNCIONES DE PERTENENCIA

En aplicaciones FL de ingeniería, las funciones de pertenencia $\mu_F(x)$ son asociadas con términos que aparecen en antecedentes o consecuentes de reglas, o en frases. Así las funciones de pertenencia definen el comportamiento de cada término.

Las formas de funciones de pertenencia más comúnmente usadas son: triangular, trapezoidal, lineal por segmentos (*piecewise*) [Aco98b, Aco99b] y Gausiana. Basado en su experiencia, el usuario puede seleccionar arbitrariamente dichas formas, así pueden tomar distintos valores.

Se obtiene mayor resolución al representar el dominio con mayor cantidad de funciones de pertenencia; el inconveniente es que se produce un incremento en la cantidad de reglas, lo que implica un aumento de la complejidad de cálculo.

Una de las ventajas de FL es que las funciones de pertenencia pueden ser diseñadas para solaparse, de tal forma que pueda expresarse que *una botella puede estar parcialmente llena o parcialmente vacía al mismo tiempo*. Así, es posible distribuir nuestra decisión sobre más de una clase de entrada, lo cual ayuda a hacer FLS robustos.

2.5 - SISTEMAS DE LÓGICA DIFUSA

Los cuatro elementos que permiten escribir una fórmula matemática que relaciona la salida de un FLS con sus entradas son:

- Las **reglas** definen el comportamiento del controlador.
- El **fusificador** es el módulo responsable de la traducción a valores difusos de los valores de las variables de entrada.
- El **motor de inferencia** se encarga de realizar el cálculo o aplicación de todas las reglas.
- El **defusificador** es el encargado de la traducción del valor difuso de salida, que genera el motor de inferencia, a su valor *crisp* de salida.

2.5.1 - REGLAS

Una base de reglas consiste en una colección de reglas IF-THEN, la cual puede ser expresada como:

$$R^{(L)}\textbf{: IF } u_1 \textbf{ is } F^L{}_1 \textbf{ AND } u_2 \textbf{ is } F^L{}_2 \textbf{ AND } \ldots \textbf{ AND } u_p \textbf{ is } F^L{}_p \textbf{ THEN } v \textbf{ is } G^L$$

donde, $L = 1, 2, \ldots, M$, y $F^L{}_i$ y G^L son conjuntos difusos en $U_i \subset R$ y $V \subset R$, respectivamente (R denota el conjunto de números reales), u_i representa a cada

variable de entrada del sistema (u_1, ..., u_p) $\in$ U_1 x ... x U_p, y $v \in V$ y representa una variable de salida del sistema. Sus valores numéricos son $x \in U$ e $y \in V$, respectivamente. Una regla puede tomar en consideración uno o múltiples antecedentes para determinar un consecuente.

Las reglas difusas son generadas a partir de los datos realizando los siguientes tres pasos:

1) Determinar el grado (valores de funciones de pertenencia) de los elementos de $x^{(j)}$.

2) Asignar cada variable a la región con máximo grado.

3) Obtener una regla de un par de datos de entrada/salida deseado.

Como puede haber gran cantidad de datos, puede ser que haya reglas un poco conflictivas (ej. reglas con los mismos antecedentes, pero diferentes consecuentes). Esto se resuelve asignando un grado de certeza, a cada regla y aceptando sólo la regla que tenga mayor grado de un grupo en conflicto.

Una regla de múltiples antecedentes y múltiples consecuentes puede siempre ser considerada como un grupo de reglas con múltiples entradas y una simple salida; donde:

$$p \Rightarrow (q_1 \wedge q_2 \wedge q_3) \Leftrightarrow (p \Rightarrow q_1) \wedge (p \Rightarrow q_2) \wedge (p \Rightarrow q_3).$$

2.5.2 - FUSIFICADOR

El fusificador mapea valores crisp

$$x = \{ x_1, x_2, ..., x_n \} \in U$$

en un conjunto difuso A^* en U. El fusificador más ampliamente usado es el *singleton*, por ejemplo, A^* es un *singleton* difuso con base x' si,

$$\mu_{A^*}(x) = 1, \qquad \text{para } x = x', \text{ y}$$

$$\mu_{A^*}(x) = 0, \qquad \text{para otro } x \in U \text{ con } x \neq x'.$$

La fusificación *singleton* no siempre será adecuada, especialmente cuando los datos están corruptos por medición de ruido. La fusificación *non-singleton* provee bases para manejar tales incertidumbres de acuerdo con el marco de trabajo de los FLS. Un fusificador *non-singleton* es uno para el cual

$$\mu_{A*}(x') = 1, \text{ y}$$

$\mu_{A*}(x)$ decrece de una unidad a medida se aleja de x'.

En esta fusificación, x' es mapeado en un número difuso; por ejemplo una función de pertenencia difusa es asociada con el fusificador, tal función puede ser Gausiana, triangular, etc.. A mayor amplitud de esas funciones, mayor es la incertidumbre acerca de x'.

2.5.3 - MOTOR DE INFERENCIA DIFUSA

Los principios de FL se usan para combinar reglas difusas IF-THEN, de la base de reglas, en un mapeo de conjuntos de entrada difusos $U = U_1 \times U_2 \times ... \times U_p$ en conjuntos difusos de salida en V. Cada regla es interpretada como una implicación difusa. Con referencia a la ecuación,

$$R^{(L)}: \textbf{IF } u_1 \textbf{ is } F^L_1 \textbf{ AND } u_2 \textbf{ is } F^L_2 \textbf{ AND } ... \textbf{ AND } u_p \textbf{ is } F^L_p \textbf{ THEN } v \textbf{ is } G^L$$

si

$$\begin{aligned} F^L_1 \times F^L_2 \times ... \times F^L_p &\equiv A \\ G^L &\equiv B \end{aligned}$$

entonces,

$$R^{(L)}: F^L_1 \times F^L_2 \times ... \times F^L_p \rightarrow G^L = A \rightarrow B.$$

El motor de inferencias, como sistema, es un bloque que mapea conjuntos de entrada difusos en conjuntos difusos de salida: $\mu_F(x) \rightarrow B(x,y)$. La Fig. 2.1 muestra el diagrama, donde la entrada es un vector permitiendo que las reglas tengan (potencialmente) múltiples antecedentes [Men95].

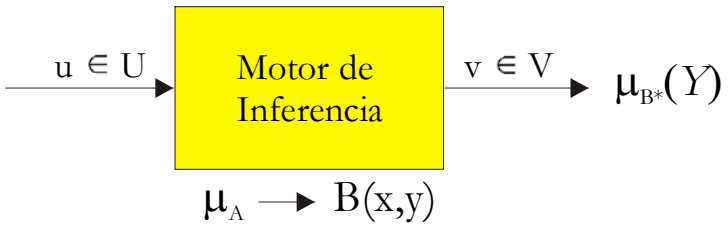


Fig. 2.1 - Diagrama del motor de inferencia

Se asume que nuestro universo de discurso es discreto, y que cada U_i (con i = 1, 2, 3, ..., p), y G son finitos, así $R^{(L)}$ está dado por una función de pertenencia discreta y multivariable $\mu_{R(L)}(x,y)$, descrita por

$$\mu_{R(L)}(x, y) = \mu_{A\rightarrow B}(x, y),$$

donde x es una de las variables de entrada ($x_1, x_2, ..., x_p$). Consecuentemente,

$$\mu_{R(L)}(x, y) = \mu_{R(L)}[x_1, x_2, ..., x_p, y],$$

y,

$$\mu_{R(L)}(x, y) = \mu_{A\rightarrow B}(x, y) = \mu_{FL1}(x_1) \barwedge \mu_{FL2}(x_2) \barwedge ... \barwedge \mu_{FLp}(x_p) \barwedge \mu_{GL}(y)$$

donde múltiples antecedentes son conectados por el operador $\barwedge$, que representa el AND y sólo el producto o mínimo es utilizado para su evaluación. La entrada p-dimensional a $R^{(L)}$ es dada por el conjunto difuso A_x, cuya función de pertenencia es:

$$\mu_{Ax}(x) = \mu_{x1}(x_1) \barwedge ... \barwedge \mu_{xp}(x_p)$$

donde $x_k \subset U_k$ (con k = 1, 2, ..., p) son los conjuntos difusos que describen las entradas. Cada regla $R^{(L)}$ determina un conjunto difuso (de salida) $B^L = A_x \barvee R^{(L)}$ en R. El conjunto de las variables lingüísticas de salida representa a todas las reglas $R^{(L)}$ que aportan información al mismo consecuente B^L. La unión de múltiples reglas es establecida por medio del operador $\barvee$, que representa el OR y sólo la suma o máximo son utilizados para su evaluación.

$$\mu_{BL}(y) = \mu_{Ax} \barvee R^{(L)} = \sup_{x\in Ax}[\mu_A(x) \barwedge \mu_{A\rightarrow B}(x, y)]$$

Esta ecuación es la relación entrada/salida entre el conjunto difuso, que excita a una regla del motor de inferencia, y el conjunto difuso en la salida del motor.

Algunas aplicaciones de ingeniería requieren implicaciones y conectivos que exigen otro tipo de relación entre las reglas. Por ejemplo, una alternativa muy utilizada es la combinación de reglas *aditivamente*. Bart Kosko denomina a tales sistemas (con un apropiado fusificador y defusificador) como FLS aditivos [Kos92].

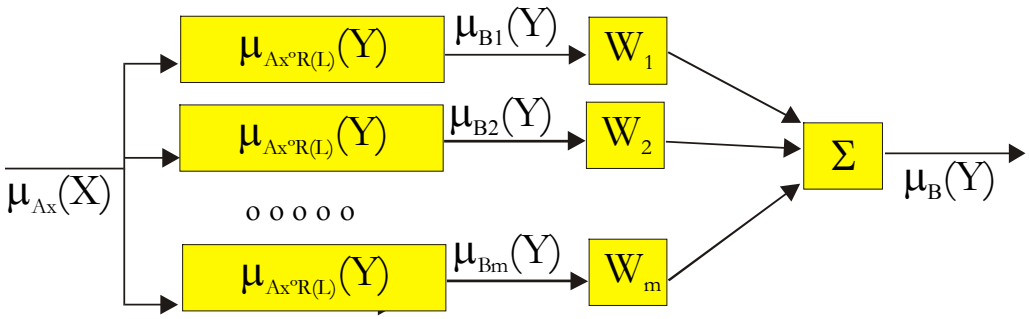


Fig. 2.2 - Combinador aditivo

La Fig. 2.2 muestra un combinador aditivo que usa un filtro adaptativo cuyas entradas son salidas de conjuntos difusos. Los pesos w_1, w_2, ..., w_M del combinador pueden ser vistos como grados de verdad de cada regla. Es posible que se conozca que algunas reglas pueden ser más fidedignas que otras; a tales reglas se les asignará un peso mayor que a las menos fidedignas [Men95]. Si tal información fuera desconocida al comienzo del sistema, se pueden definir todos los pesos iguales a la unidad, o usar un procedimiento de entrenamiento para enseñar los valores óptimos de los pesos.

2.5.4 - DEFUSIFICADOR

En un FLS el proceso de defusificación produce una salida *crisp*, a partir de la información difusa que sale del motor de inferencia. Han sido propuestos en la literatura muchos defusificadores, sin embargo, no hay bases científicas (ninguno de ellos ha sido derivado de un principio tal como "la maximización de la información difusa" o "la entropía"); consecuentemente, la defusificación es más un arte que una ciencia [Men95].

Para desarrollar aplicaciones de FL, el criterio más difundido para la selección del defusificador es la simplicidad computacional. De acuerdo con este criterio, se muestran los dos defusificadores más utilizados.

2.5.4.1) CENTRO DE GRAVEDAD (COG):

Este defusificador determina el centro de gravedad (COG, Center of Gravity) $\underline{y}$ de B, y usa ese valor como salida del FLS. Del cálculo, resulta que:

$$\underline{y} = [\int_S y \ \mu_B(y) \ dy] \ / \ [\int_S \mu_B(y) \, dy],$$

donde S representa el dominio de $\mu_B(y)$.

2.5.4.2) CENTROIDE:

Este defusificador es una simplificación realizada sobre el cálculo del COG. Como frecuentemente S es discreto, entonces $\underline{y}$ puede ser aproximado reemplazando con un sumatorio el cálculo de integrales. En lugar de determinar el centro de gravedad, se establece un punto aproximado denominado *centroide* $\underline{y}$ de B, y usa ese valor como salida del FLS.

$$\underline{y} = [\int_S y \ \mu_B(y) \ dy] \ / \ [\int_S \mu_B(y) \, dy] \cong [\ \Sigma \ y_i \, \mu_B(y_i)] \ / \ [\Sigma \, \mu_B(y_i)]$$

con $i = 1, ..., I$. Este método es la forma más ampliamente usada de defusificación [Kos92, Men95, Kan98].

2.6 - MODELOS DIFUSOS

A principio de los setenta, el modelado difuso fue tratado como un importante problema de la teoría de sistemas difusos. La identificación de un modelo difuso, directamente de los datos de entrada y salida o de la investigación del comportamiento del operador del proceso, es requerida para aplicaciones prácticas de control difuso. Los modelos ayudan a ver el comportamiento y la estructura básica de procesos complejos [Ton77].

Hay una variedad de modelos difusos de acuerdo a la forma de las reglas *IF-THEN* involucradas en los modelos difusos, pero pueden ser clasificados en dos principales modelos: el propuesto por *Mamdani* en [Mam75] y el propuesto por *Takagi-Sugeno* [Tak85].

2.6.1 - MODELO DIFUSO DE 'MAMDANI'

La configuración del modelo difuso de *Mamdani* [Mam93] se muestra en la Fig. 2.3. En esta clase de modelos difusos, las reglas difusas *IF-THEN* son de la forma:

$$R^i: \textbf{IF } x_1 \text{ is } A^i_1 \textbf{ and } x_2 \text{ is } A^i_2 \textbf{ and } .... \textbf{ and } x_n \text{ is } A^i_n \textbf{ THEN } y \text{ is } B^i$$

Es también una característica que la materialización del motor de inferencias difusas, fusificador y defusificador se benefician de la libertad de elección del tipo de operaciones.

Las principales ventajas del modelo difuso de *Mamdani* se especifican a continuación. Primero, su simplicidad en la representación de las reglas difusas, tanto las premisas como los consecuentes tienen forma de conjunto difuso lo que facilita su interpretación. Segundo, su flexibilidad en la materialización debido a la posibilidad de seleccionar las operaciones del motor de inferencia, del fusificador o defusificador.

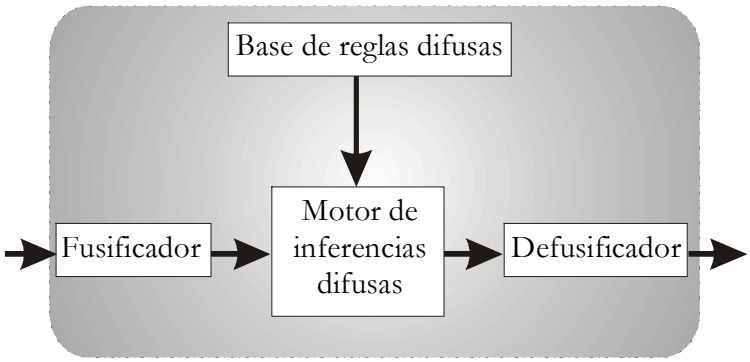


Fig. 2.3 - Configuración del Modelo de Mamdani

La principal desventaja de este modelo difuso es que para sistemas complejos no lineales usualmente se requieren muchas reglas difusas *IF-THEN*, lo cual hace más compleja su materialización.

2.6.2 - MODELO DIFUSO DE 'TAKAGI-SUGENO'

La configuración del modelo difuso de *Takagi-Sugeno* se muestra en la Fig. 2.4. En esta clase de modelos difusos, las reglas difusas *IF-THEN* son de la forma:

$$R^i: \textbf{IF } \boldsymbol{x}_1 \text{ is } \boldsymbol{A}^i_1 \textbf{ and } \boldsymbol{x}_2 \text{ is } \boldsymbol{A}^i_2 \textbf{ and } .... \textbf{ and } \boldsymbol{x}_n \text{ is } \boldsymbol{A}^i_n \textbf{ THEN}$$

$$\boldsymbol{y}^i = \boldsymbol{p}^i_0 + \boldsymbol{p}^i_1 \boldsymbol{x}_1 + \boldsymbol{p}^i_2 \boldsymbol{x}_2 + ... + \boldsymbol{p}^i_n \boldsymbol{x}_n$$

Donde, $\boldsymbol{A}^i_k$ son conjuntos difusos; $\boldsymbol{p}^i_h$ son coeficientes del i-ésimo consecuente lineal (con $\boldsymbol{h}$=0, 1, 2, ..., $\boldsymbol{n}$); y además $\boldsymbol{y}^i$ es la salida de la i-ésima regla *IF-THEN* difusa.

La principal ventaja de este modelo difuso es su poderosa capacidad de representar relaciones complejas y no lineales en un conjunto con un pequeño número de reglas difusas *IF-THEN*. En suma, los consecuentes en el modelo de Mamdami, con reglas difusas *IF-THEN*, son reemplazados por consecuentes en forma de relaciones lineales de Takagi-Sugeno.

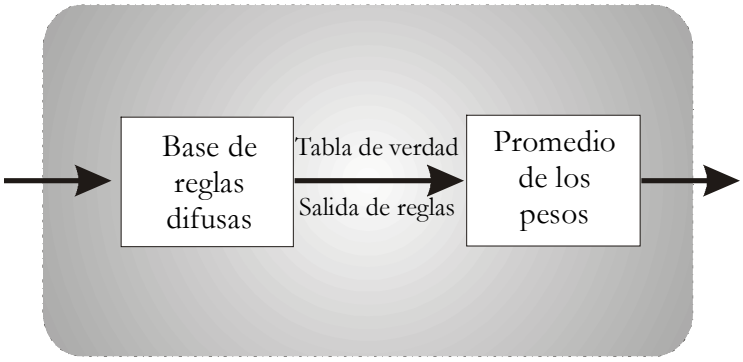


Fig. 2.4 - Configuración del modelo difuso de Takagi-Sugeno

2.7 - COMPARACIÓN DE MÁQUINAS DIFUSAS

Las decisiones arquitecturales adoptadas generalmente difieren en gran medida entre las distintas máquinas, por lo cual es imposible hacer una descripción detallada de los cambios propuestos por las arquitecturas innovadoras. Por esa causa, se comparan los diseños desarrollados y se destacan algunos puntos importantes desde el punto de vista de la materialización.

Una de las formas de comparar las arquitecturas propuestas es por la velocidad máxima a la que pueden trabajar, siendo esta uno de los parámetros siempre presentes en la bibliografía. Se utilizan dos unidades para medir la velocidad de los procesadores o controladores difusos:

- **FIPS:** *Fuzzy Inference Per Second* (inferencias difusas por segundo), y
- **FRPS**: *Fuzzy Rules Per Second* (reglas difusas por segundo).

Algunas arquitecturas están orientadas a evaluar reglas difusas, entonces el tiempo requerido es proporcional al número de reglas, por lo cual es imposible dar su velocidad en FIPS ya que depende del número de reglas por inferencia, lo cual cambia de acuerdo a la aplicación.

Por supuesto, la velocidad no es el único parámetro importante para juzgar la calidad de un procesador difuso, también es importante el número máximo de variables de entrada y salida, su precisión, el máximo número de reglas, el consumo de energía, tamaño / área, etc..

2.8 - PERSPECTIVA HISTÓRICA DEL HARDWARE

La Tabla 2.1 muestra el desarrollo de las primeras aplicaciones de lógica difusa, desde la perspectiva histórica propuesta por [Rus98].

Año	Detalle
1961	*Lofti A. Zadeh* menciona que se necesita de una nueva técnica, un tipo de matemáticas difusas ("*fuzzy*" *mathematics*) [Sur95].
1965	*Lofti A. Zadeh* introduce el concepto de conjunto difuso.
1969	En *Duke University* se realiza la primera investigación sobre materializaciones hardware de lógica difusa.
1972	*Sugeno M*. presenta la idea de mediciones difusas.
1974	*Mamdani E. H*. presentan una aplicación de control difuso.
1980	*Yamakawa T*. construye el primer circuito difuso con componentes bipolares discretos
1982	*Linkman*, es la primer aplicación industrial de una aplicación difusa, en una fábrica de cemento en Alemania.
1984	*Togai M*. y *Watanabe H*. presentan la primera materialización de un VLSI difuso.

1986	*Hitachi*, luego de 8 años de estudio pone en funcionamiento un sistema de metro cuyo control es realizado por algoritmos difusos. El controlador difuso usado redujo la distancia de frenado 2.5 veces, redujo el consumo en un 10% y dobló el índice de confort.
1987	*Yamakawa T*. presenta el primer controlador difuso analógico.
1988	*Togai M*. presenta '*Digital fuzzy processors*'
1990	Se funda en Japón el *LIFE* (*Laboratory for International Fuzzy Engineering Research*)
1992	Se realiza la primera conferencia internacional de la *IEEE* sobre sistemas difusos
1993	Sale el primer número de la *IEEE Transactions on Fuzzy Systems*

Tabla 2.1 – Desarrollo de aplicaciones de FL, perspectiva histórica [Rus98]

En 1984, *Togai* y *Watanabe* [Tog85] desarrollan la primer máquina de inferencia difusa para *AT&T Bell Laboratories*. Se diseña un chip VLSI digital con una entrada y una salida, que ejecuta 250000 FRPS sin etapa de defusificación.

En 1987, *Yamakawa* [Yam87] presenta el primer controlador difuso analógico, que integra algunos circuitos dedicados bipolares analógicos. Utiliza circuitos específicos para el cálculo de las operaciones MIN (*mínimo*) y MAX (*máximo*). Este integrado es capaz de evaluar 1MFIPS incluyendo la etapa de defusificación. Adopta como método de defusificación el cálculo del COG (*Center Of Gravity*). El mismo diseño, pero sin la etapa de defusificación alcanza la velocidad de 10MFIPS.

En 1988, *Togai Infralogic* presenta el primer procesador difuso en el mercado, el FC110. Por supuesto este procesador es más lento que los procesadores de los avanzados grupos de investigación, pero es de propósito mucho más general. El FC110 corriendo a 10Mhz ejecuta 28600 FIPS.

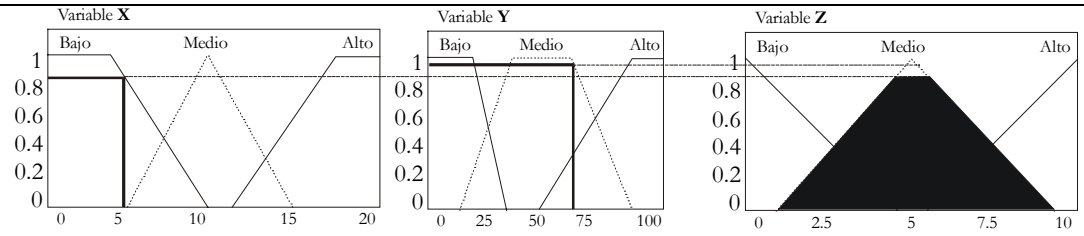
En 1990, *Watanabe* [Wat90] presenta un controlador difuso mejorado que toma ventajas de la alta velocidad y capacidad de integración de los VLSI. Adopta la suma lógica (mínimo) y el producto lógico (máximo) como operadores de unión e intersección. El chip opera a 36 Mhz, con un rendimiento de 580000 FIPS y con hasta 102 reglas por inferencia. El universo de discurso fue cuantificado en 64 valores distintos y los grados de pertenencia divididos en 16 valores. Las

entradas son registradas cada 64 ciclos de reloj y usadas como dirección de comienzo de las expresiones almacenadas. La incertidumbre o imprecisión de las entradas es relativa al ancho de las funciones de fusificación almacenadas. El diseñador puede adoptar el COG como método de defusificación.

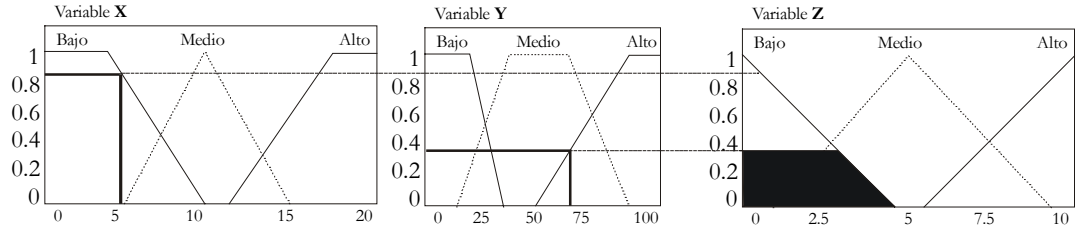
La característica principal de todos estos procesadores, es que cada uno trabaja con conjuntos difusos de salida. Por ello, gran cantidad de hardware está dedicado a determinar la conclusión de las reglas difusas. En las arquitecturas modernas este concepto es obsoleto, los nuevos procesadores generalmente trabajan con *singletons* de salida. Esto significa que por cada salida, para cada regla hay solo un valor *crisp* que es ponderado con el grado de verdad de la regla para obtener el valor defusificado final. Todos estos procesadores usan el método de inferencias MIN-MAX. Con excepción de la etapa de defusificación, puede verse fácilmente que todo el cálculo está basado en operaciones elementales: mínimo (MIN) y máximo (MAX).

El cálculo de una regla difusa expresada de acuerdo al método MIN-MAX puede dividirse en los siguientes pasos.

- asignación de los valores de pertenencia α para cada antecedente
- asignación de los valores de pertenencia θ para la premisa
- cálculo de la función de pertenencia de salida mediante la aplicación del método de inferencia (mediante las operaciones de truncar y unir en el caso del MIN-MAX)
- defusificación



a) **IF** (***x*** is BAJO) **and** (***y*** is MEDIO) **THEN** ***z*** is MEDIO



b) **IF** (***x*** is BAJO) **and** (***y*** is ALTO) **THEN** ***z*** is BAJO

de (a) y (b) ENTONCES el centroide será el promédio ponderado de MEDIO y BAJO

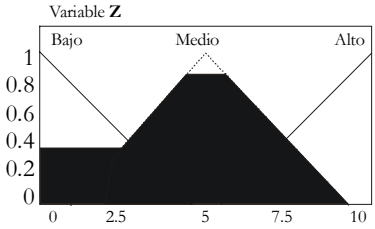


Fig. 2.5 - Inferencia difusa usando el método MIN-MAX.

En la Fig. 2.5 se muestra el cálculo geométrico aplicando el método MIN-MAX de la aplicación de dos reglas, suponiendo que los valores de entrada son: ***x*** = 5 e ***y*** = 70. Ambas variables tienen tres conjuntos difusos: *bajo*, *medio* y *alto*, definidos según las funciones de pertenencia. Entonces, $\mu_{bajo}(\boldsymbol{x})$= 0.8, $\mu_{medio}(\boldsymbol{x})$= 0, $\mu_{alto}(\boldsymbol{x})$=0, $\mu_{bajo}(\boldsymbol{y})$= 0, $\mu_{medio}(\boldsymbol{y})$= 0.95, y además $\mu_{alto}(\boldsymbol{y})$= 0.4. Al evaluar:

- La regla (a), "**IF** (***x*** is BAJO) **and** (***y*** is MEDIO) **THEN** ***z*** is MEDIO", tiene una operación ***and***, que se debe evaluar utilizando el operador mínimo, entonces ***z*** = min(***x*** is *bajo*, ***y*** is *medio*)= min(0.8, 0.95)= 0.8, lo que significa que ***z*** pertenece a la función *medio* en un 0.8.

- La regla (b), "**IF** (***x*** is BAJO) **and** (***y*** is ALTO) **THEN** ***z*** is BAJO", entonces ***z*** = min(***x*** is *bajo*, ***y*** is *alto*)= min(0.8, 0.4) = 0.4, lo que significa que ***z*** pertenece a la función *bajo* en un 0.4.

- Ambas reglas deben ser combinadas utilizando el operador máximo, entonces se provoca que las funciones consecuentes se corten a la altura que fue calculada anteriormente, para luego calcular el *centro de gravedad* como si formaran una sola función. La solución de la aplicación del algoritmo difuso es el punto del centro de gravedad.

- Para defusificar el valor difuso obtenido de la inferencia, se debe calcular $z = (\Sigma\ def^i * valor\_difuso^i) / (\Sigma\ def^i) = (bajo_z * def_{bajo} + medio_z * def_{medio} + alto_z * def_{alto}) / (bajo_z + medio_z + alto_z) = (0.4 * 1 + 0.8 * 5 + 0 * 0) / (0.4 + 0.8 + 0) = (0.4 + 4) / (1.2) = 4.4 / 1.2 = 3.66$.

En 1991 *Ikeda* [Ike91] aplica el concepto de reglas activas al hardware difuso. Aunque la idea es muy simple, marcó un gran paso en la investigación de procesadores difusos. De hecho, permite que la evaluación de inferencias difusas sea acelerada con muy poco incremento de área. Para que una regla, que contiene ***I*** antecedentes en la premisa relacionados por el operador AND (evaluado con el operador MIN o producto), tenga un grado de activación nulo θ, es suficiente que sólo uno de los antecedentes tenga un grado de pertenencia nulo α. Las reglas activas son sólo aquellas que tienen premisas con un θ positivo. Una vez que el valor de una variable (difuso o CRISP) ha sido fijado, éste tiene una intersección no nula con un limitado número de valores lingüísticos ***O*** en su conjunto de términos. Es posible mostrar [Asc95] que el porcentaje de reglas activas *FActive* comparado con el número total es igual a:

$$FActive = 100\ (\ \boldsymbol{O}\ /\ \boldsymbol{T}\ )\ \hat{}\ \boldsymbol{I}\ \%$$

donde se asume que el factor de solapamiento y el número de elementos de cada conjunto de términos de todas las variables de entrada ***I*** son siempre ***O*** y ***T***

respectivamente. Por ejemplo, con 4 entradas ($\boldsymbol{I} = 4$), cada una con un conjunto de 4 términos ($\boldsymbol{T} = 4$) y un factor 2 de solapamiento ($\boldsymbol{O} = 2$), ***Factive*** = 6.25%. Si, por el otro lado se tiene que, $\boldsymbol{I} = 4$, $\boldsymbol{T} = 8$, y $\boldsymbol{O} = 2$, solo 256 de 65536 posibles reglas podrían ser activadas, ***FActive*** = 0.39 %.

El número de reglas activas para variaciones de los valores de entrada será siempre una pequeña fracción del total de reglas. Una consecuencia lógica de procesar todas las reglas es la ineficiencia, primero porque esas reglas no contribuyen al resultado final y segundo, porque gastan tiempo de cálculo, lo cual es un factor importante. Reglas activas son utilizadas ampliamente en varios diseños [Asc95, Asc96, Aco98].

En 1992, se presentó el procesador difuso WARP (*Weight Associative Rule Processor*), diseñado e implementado por *SGS-Thompson Microelectronic*[2], que alcanza los 10 MFRPS.

En 1992, *OMRON Corporation* presenta el procesador difuso FP-5000. Tiene un desempeño de 10 MFRPS pero es mucho más general, de hecho, puede evaluar hasta 128 entradas y 128 salidas (de 8 bits cada una), y almacenar hasta 32000 reglas difusas.

En 1992, se presenta una idea para la compresión de memoria dedicada al almacenamiento de funciones de pertenencia, sin restringir la forma de la figura [Chi92]. Uno de los primeros métodos para almacenar las funciones de pertenencia es la construcción de un vector que contiene todos los valores de las funciones de pertenencia de los correspondientes elementos en el universo de discurso. Esta solución desperdicia mucha memoria, ya que la mayoría de las posiciones contiene un valor nulo. Explorando el concepto de solapamiento de las posiciones de memoria ocupadas se pueden comprimir dichos valores [Asc96].

Si asumimos que el conjunto de términos W($\boldsymbol{x}$) de la variable difusa $\boldsymbol{x}$ tiene una cardinalidad igual a $\boldsymbol{N}_x$ y que el solapamiento es igual a $\boldsymbol{O}_m$, significa que para

[2] SGS-Thompson Microelectronic: *http://www.st.com*

cada elemento del universo de discurso hay sólo $\boldsymbol{O}_m$ diferentes funciones de pertenencia con valores no nulos de pertenencia.

Si $\boldsymbol{C}_j$ ($1 <= \boldsymbol{j} <= \boldsymbol{O}_m$) es una partición de W($\boldsymbol{x}$):

$$\boldsymbol{U}\ (\boldsymbol{j}=1,\ \boldsymbol{O}_m)\ \boldsymbol{C}_j = W(\ \boldsymbol{x}\ )$$

$$\boldsymbol{C}_j \cap (\boldsymbol{j}\ !=\boldsymbol{k}\ )\ \boldsymbol{C}_k = \phi$$

para cada conjunto $\boldsymbol{C}_j$ hay todos los $\boldsymbol{c}_j$ que no interceptan funciones de pertenencia que pertenecen a todo el conjunto de términos. De esta forma, es posible almacenar solo $\boldsymbol{O}_m$ vectores en lugar de los $\boldsymbol{N}_x$ vectores que se almacenaban antes. De hecho, en cada vector es posible almacenar todas las funciones de pertenencia que pertenecen a un conjunto. Cada elemento del universo de discurso es asociado solo con el grado de pertenencia no nulo en ese conjunto.

Un ejemplo se da en la Fig. 2.6. El primer gráfico muestra un conjunto de términos con tres elementos. Obviamente el solapamiento $\boldsymbol{O}_m$ es igual a dos. Los dos gráficos siguientes muestran el *particionado* del conjunto de términos en dos conjuntos $\boldsymbol{C}_1$ (almacenado en el banco *A*) y $\boldsymbol{C}_2$ (almacenado en el banco *B*). Con esta solución la cantidad de memoria requerida para almacenar todas las funciones de pertenencia de $\boldsymbol{W}(\boldsymbol{x})$ se reduce de $\boldsymbol{N}_x$ a $\mathbf{O}_m$. En este ejemplo se pasa de necesitar tres bancos por variable a dos bancos, pero normalmente puede suceder que de siete u ocho bancos se reduzca a dos. Se debe tener cuidado, ya que el acceso a esta memoria no da la información acerca de a cual función de pertenencia corresponde el valor obtenido.

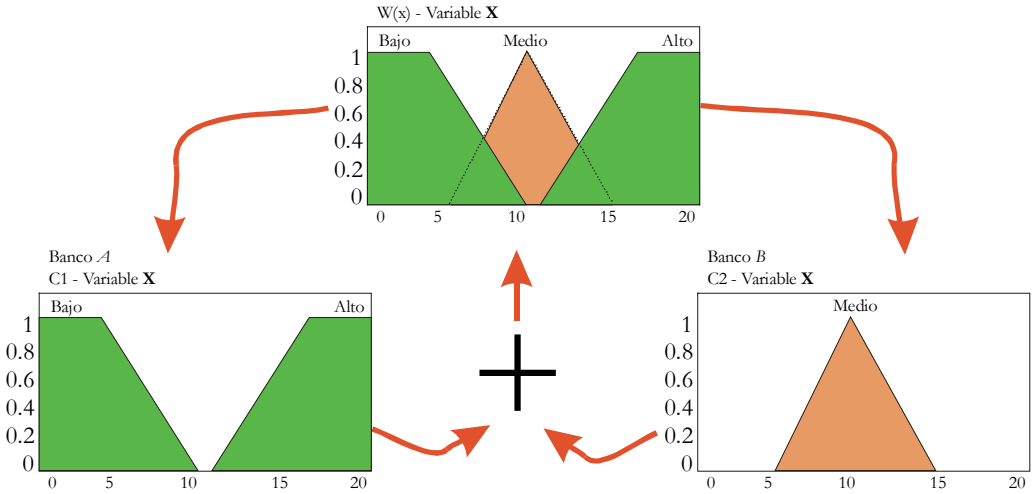


Fig. 2.6 - Particionado del conjunto de términos

Con esta organización no sólo se ahorra memoria, también se facilita el proceso de fusificación, permitiendo la fusificación de todas las variables lingüísticas en sólo $\boldsymbol{O}_{\mathrm{m}}$ accesos a memoria [Chi92]. Además, permite el ahorro de memoria, sin restringir la forma de la función de pertenencia.

Existen otros métodos que permiten el ahorro de área de silicio, esencialmente basados en generadores de funciones de pertenencia. Estos circuitos pueden generar varias figuras, pero en un número limitado (formadas por triángulos, trapecios, campanas de Gauss, etc.). Esta limitación permite que solo unos pocos parámetros necesiten ser almacenados, los cuales son usados luego por el circuito para reconstruir la función de pertenencia.

En 1992, se presenta un procesador difuso basado en un superconductor llamado *Josephson Fuzzy Inference Engine*. En sus resultados de simulación se estima que para 4 reglas y 16 niveles de entradas puede alcanzar los 0.79 Giga FIPS.

En 1992 comienzan a presentarse en congresos la materialización de FLC en FPGAs [Sur92]. Otra experiencia es FAM (*Fuzzy Automata Machine*) [How92], que se basa en un modelo de autómata celular de grano fino, *síncrono* y *sistólico*, implementado en lógica reconfigurable.

En 1993, OMRON presenta una mejora del procesador de *Yamakawa* describiendo un chip de reglas (FP-9000) y un chip defusificador (FP-9001). El FP-9000 se implementó en tecnología de 2μm *BiCMOS* y el FP-9001 en tecnología de 3μm bipolar. Alcanzan una velocidad de cálculo de 1.4 MFIPS sin defusificador, y 625 KFIPS incluyendo la etapa de defusificación. La velocidad de inferencia es independiente del número de reglas porque puede ser conectado en paralelo. El defusificador usa un divisor analógico.

En 1993, *Mitsubishi Electronic Corporation* presenta un procesador difuso [Nak93] más lento, pero con evaluación difusa más precisa. La velocidad es de 200 KFIPS con un reloj a 20 Mhz. El procesador tiene una resolución de 12 bits para las entradas, y de 16 bits para las salidas.

En 1993, *Sasaki* [Rus98] presenta la implementación de una arquitectura altamente paralela, basada en una SIMD (*Single Instruction Multiple Data*) y una estructura de lógica en memoria (*Logic-in-memory*). La velocidad de inferencia incluyendo defusificación alcanza los 7.5 MFIPS.

En 1993, *Ungering* y *Goser* [Ung93] proponen una arquitectura de controlador de lógica difusa usando la técnica de PWM (*Pulse With Modulation*) y una estructura de *pipeline*. *Tombs*, *Torralba* y *Franquelo* [Tom96] materializan una arquitectura mixta, con un fusificador analógico y un circuito de división híbrido.

En 1993, Gupta y Siewiorek [Gup93] presentan un conjunto de técnicas de síntesis de alto nivel para circuitos de cálculo. La herramienta se basa en una base de conocimiento definida mediante reglas. Clasifica los tipos de circuitos básicos, y usa moldes (*templates*) para mantener su información.

Durante 1994 se presenta XFuzzy versión 1.1, un ambiente para la simulación y generación de código C de sistemas de control basados en lógica difusa. Este sistema fue desarrollado en el Centro Nacional de Microelectrónica, sede Sevilla, España [Mas94]; dicho proyecto se prolonga hasta 1998 [Bat98e].

En 1995, *A. Ruiz*, *J. Gutierrez* y *F. Fernandez* [Rui95] proponen un método de defusificación que evita la división, calculando el *centro de área* en lugar de un centro de gravedad.

En 1995, *Ascia*, *Catania*, *Giacalone*, *Russo* y *Vita* [Asc95] proponen un procesador que optimiza el tiempo de cálculo, realizando un pre – proceso que reduce el número de reglas a ser procesadas, calculando los grados de activación de las reglas activas. Los mismos autores proponen un procesador difuso con reglas activas [Asc96] en una arquitectura mixta, analógica y digital, altamente paralela mediante segmentación.

En 1996, *J. Moore* y *M. Manzoul* [Moo96] presentan un sistema de generación de controladores difusos cuya descripción inicial se basa en un lenguaje. Se generan arreglos sistólicos VLSI para aplicaciones de control en tiempo real.

En 1996, *A. Costa*, *A. De Gloria* y *M. Olivieri* [Cos96] proponen una metodología de diseño para materializar FLC en una arquitectura asíncrona (*self-time*), utilizando segmentación con etapas no simétricas en el tiempo. Este diseño es muy efectivo para sistemas con una complejidad muy alta, logrando consumos muy bajos.

En 1997, *I. Baturone*, *S. Sánchez-Solano*, *A. Barriga* y *J. L. Huertas* [Bat97] presentan técnicas para la materialización de funciones de pertenencia materializadas en circuitos analógicos. Proponen métodos para evitar la división en la defusificación y, justifican el uso de conversores (A/D y D/A) por corriente en lugar de voltaje.

En 1997, Hollstein, Kirschbaum y Glesner [Hol97] presentan un ambiente para realizar prototipos de controladores difusos genéricos. El sistema se basa en una librería de módulos VHDL, que una vez definidos los parámetros del FLC, se realiza la materialización en FPGAs.

En 1997 y 1998, *Carvajal*, *Aguirre*, *Torralba* y *Franquelo* [Car97, Car98] presentan AFAN, una herramienta que permite realizar la síntesis de alto nivel de una arquitectura difusa, incluyendo sistemas difusos y neuronales. Los

requerimientos de velocidad y el costo (en área de silicio) forman parte de los datos iniciales para la generación del VHDL.

En 1998, *A. Sanz* y *J. Falco* proponen un ambiente para el diseño de controladores complejos denominado IDEA, basado en un lenguaje de descripción llamado EDA [San98]. La descripción de alto nivel puede materializarse en software usando una librería C++ o en hardware en un circuito electrónico.

Entre 1995 y 1998, *Salvador-Carrasco* y *J. Gutierrez-Ríos* [Sal95, Sal98] muestran las ventajas de usar arquitecturas serie para materializar algoritmos difusos, permitiendo implementar arquitecturas sistólicas. Esta arquitectura se orienta a sistemas de altas prestaciones, cuyo desempeño es independiente del tamaño del modelo difuso, permitiendo manejar gran cantidad de reglas sin perder eficiencia.

En 1998, se proponen arquitecturas para procesadores difusos que operen directamente sobre los conjuntos difusos en lugar de hacerlo sobre elementos del dominio [Rui98], debiéndose entonces, calcular pocas funciones más complejas, en lugar de muchas funciones más simples.

Desde 1999 los aportes se concentran principalmente en dos grandes áreas: el desarrollo de algunas técnicas innovadoras orientadas a satisfacer las necesidades de velocidad de cálculo o el diseño de aplicaciones industriales (o del mundo real) que normalmente necesitan de gran potencia de cálculo.

La lista de artículos que se mencionan a continuación, no pretende ser exhaustiva, pero intenta describir el movimiento tecnológico que a juicio del autor describe la situación actual de la FL. Para esta descripción se ha clasificado en 6 grupos de acuerdo al dominio de aplicación y al área de interés del autor. A saber: A) Aplicaciones que utilizan FL, B) propuestas para mejorar los algoritmos de cálculo difuso, C) aplicaciones de control tradicional basado en FL, D) análisis de patrones en señales e imágenes, E) planificación y selección de estrategias en robots móviles, y F) control de manipuladores y robots.

La literatura en los últimos años de FL presenta un número muy grande de aplicaciones altamente destacables tanto por el dominio como por la solución propuesta. Son ellas: [CCo98] propone un esquema adaptativo para el control de un robot aplicado a la agricultura, [Pit99] y [Kas99] organizan el tráfico en redes de datos evitando colisiones, [Zul99] determina el comportamiento de motor sincrono con magnetos permanentes garantizando la mejor onda sinusoidal posible, [Paw99, Kov00, Mor00] controlan servos eléctricos y neumáticos, [Bog01] y [Cot00] controlan bucles cerrados de convertidores de potencia DC-DC paralelo y conmutado, [Dra00] ajusta antenas de sistemas de radio-frecuencia, [Kar00] controla la mezcla de combustibles de una caldera, [Pas00] controla motores de combustión interna, [CTL00] controla la temperatura del agua del baño usando auto-aprendizaje, [Hen00] asigna el tráfico en una red de autopistas, [Pap00] organiza las rutas de líneas de ferrocarril, [Nii00] controla de señales de tránsito, [Son00] propone una optimización al problema del péndulo invertido, [XiT00] estabiliza el consumo usando tiristores, [MLi01] administra tareas en un sistema operativo donde los tiempos son especificados con números difusos, [OdM01] descubre información en grandes bases de datos, [YHL01] controla una máquina hidráulica, [YuJ01] presenta un algoritmo para la ubicación de disipadores en sistemas multi-chip, [Yia01] estabiliza el péndulo invertido usando módulos dinámicamente conectados, [FLa02] controla la temperatura de un invernadero, [JIH02] comenta aplicaciones de control industrial en procesos biológicos del Japón, [KCT02] estabiliza una plataforma en tiempo real, [Uan02] analiza un FLC adaptativo para sistemas de misiles, [VGo02] propone un modelo físico difuso de clima variable para predecir la radiación solar, [JYi02] estabiliza el doble péndulo invertido con un motor de inferencia paralelo, [JMa03] asiste al conductor de un automóvil evitando las colisiones, [Wai03] controla un servo motor usando FL y NN, [Xu03] realiza el seguimiento repetitivo incorporando características de aprendizaje para minimizar el error, [Zim03] organiza tareas maximizando la cantidad que son atendidas, [Bon03] coordina el comportamiento de agentes autónomos en tareas de tiempo real, y [Bie03] realiza el reconocimiento facial y emocional para el guiado de una silla de ruedas y para la coordinación de señales biológicas.

Los artículos que proponen mejoras en los algoritmos de cálculo en FLC, son: [Sch99] presenta un método de inferencia difusa basado en la interpolación de reglas y lo aplica al control de dos tanques de fluido, [DaK99] propone el cálculo del COG preciso y de bajo costo operacional basado en el momento de equilibrio, [Ma00] propone un defusificador basado en la distancia métrica entre números difusos, [ShT02] presenta el seguimiento de una variable con alta ganancia, [GAr02] propone una arquitectura con pipelining secuencial que brinda gran flexibilidad y un alto desempeño, [CWL03] propone una función difusa para la inferencia con base radial, [Sar03] propone el control predictivo basado en el modelo dinámico del proceso a controlar, y [JLe03] trata datos vagos en un clasificador inmune al ruido.

Algunos diseñadores enfocan las aplicaciones de control tradicional materializando el controlador por medio de FL. Entre ellos: [KHa99] propone un método para el diseño de PID lineares y no-lineares, [Kov99] ajusta un FLC utilizando redes neuronales**,** [Car00] realiza un análisis de estabilidad de lo que llama un PID difuso, [ZWW00] presenta un PID para estabilizar el seguimiento de funciones, [RkM00] presenta un PI con auto-ajuste robusto, [SAh01] controla servos con un esquema de retroalimentación de error en diferentes puntos de la planta, [WTa01] controla brazos de robot con un PI para el seguimiento de trayectorias, mientras que [Pir01] y [Pat99] utilizan FL en PID con características especiales.

Muchas son las aplicaciones de FL dedicadas a la clasificación y búsqueda de patrones en señales y particularmente en imágenes. Las seleccionadas son: [BSh98] y [GCr02] reconocen patrones en tiempo real, [Num00] y [Klo99] analizan diferentes métodos de inferencia para detectar bordes en imágenes de grupos de piezas conocidas, [Teo00] diseña un clasificador lineal orientado a reconocer caracteres chinos, [JDY01] propone un modelo de recuperación de imágenes indexadas difusas, [LaK02] procesa una imagen multi-canal, y [Dee03] y [JHH02] analizan los parámetros de un clasificador difuso.

Entre las aplicaciones dedicadas a la planificación y selección de estrategias en robots móviles usando FL, se destacan: [GaJ99] realiza la navegación de un

robot móvil en un ambiente desconocido en tiempo-real, [Cas99] y [Orfi99] analizan métodos difusos de inferencia para el sistema de navegación de un robot móvil, [Slu99] utiliza (re)configuración dinámica de reglas en el control de vuelo de una aeronave, [GuG00] controla vehículos sumergidos imitando al operador humano, [Bar01] propone una arquitectura distribuida orientada al control de robots móviles, [HPH02] elige una estrategia en fútbol de robots usando un árbol de decisión difuso, [AlK03] planifica los movimientos de un robot móvil en un ambiente dinámico, y [Los03] conduce un automóvil evaluando métodos de caracterización cuantitativa de datos.

Entre las aplicaciones dedicadas al control de manipuladores y brazos robóticos se destacan: [SKR00] controla punto-a-punto sistemas mecánicos de segundo orden para el control de un brazo de robot, [ALL01] controla manipuladores robóticos administrando la ganancia con auto-ajuste, [GNM01] guía un robot móvil usando un enfoque híbrido (NN y FL), [FSu03] usa NN para el control de trayectorias con un manipulador cuya dinámica es pobremente conocida, [Hua03] presenta el modelo de impedancia de contacto entre la herramienta de un manipulador y superficies, [Muc03] describe un controlador para el seguimiento de paredes sobre un robot móvil Nomad200 proponiendo un método para incorporar el tiempo como variable, y [Zho03] simplifica el control de equilibrio y caminata para un robot bípedo mediante aprendizaje.

En la actualidad hay investigadores dedicados a analizar las implicaciones sociales, filosóficas y éticas de la FL; como se denota en el discurso presentado por el Profesor Zadeh en su discurso de Mackay [Zad00]: "*... al comienzo del nuevo milenio, FL abre un nuevo desafío en el proceso de información, relacionando la idea de realizar cálculos basándonos en las percepciones humanas ... Estamos en la víspera del nacimiento de una teoría con capacidad de procesar información basándose en la percepción. Su efectividad ha sido ya demostrada a través de las implementaciones industriales utilizando una tecnología centrada en FL ...*".

Por otra parte, las aplicaciones industriales tienen una marcada tendencia hacia campos tales como la robótica, análisis de imágenes, visión artificial, análisis de

señales, sistemas de arreglos de radares, vehículos autoguiados, etc.; todas las áreas donde se aplica y aplicará FL tienen un requerimiento común: VELOCIDAD.

3 – CONSIDERACIONES DEL DISEÑO

3.1 - INTRODUCCIÓN

La teoría de conjuntos difusos, establecida por Zadeh [Per95] hace casi cuatro décadas, ha sido el punto de inicio de investigaciones en varios campos de la ingeniería. En el dominio de la ingeniería del control, se ha notado en las últimas dos décadas una gran influencia de los conjuntos difusos. Al principio de los años setenta, los controladores difusos fueron tenidos en consideración y se logró aplicarlos en el control de algunos procesos industriales. Desde esa época, una nueva rama de la ingeniería del control había llegado, conocida como **control difuso**.

El control difuso se basa en la lógica difusa, una lógica que está más cerca en espíritu del pensamiento humano y del lenguaje natural que la lógica tradicional. El control difuso significa principalmente el poder convertir automáticamente una política de control lingüística del conocimiento de expertos en un algoritmo de control. El hecho que el control difuso sea aceptable y atractivo se basa en que se ha aplicado fructíferamente en aplicaciones complejas de control o vagamente definidas, tales como procesos químicos o metalúrgicos. El control difuso puede manejar esos procesos satisfactoriamente por su habilidad de interpretar sentencias lingüísticas acerca de los procesos bajo control, y de inferir consecuentes en una manera calificada sin tener datos precisos y modelos de los procesos. Por el contrario, es bien conocido que las técnicas convencionales de control usualmente tienen gran dificultad con procesos complejos. La experiencia muestra que el control difuso puede generar resultados por lo menos semejantes a los obtenidos con algoritmos de control convencional [Ton77, Tog96, Mam93]. El control difuso ya no está confinado a los estudios en laboratorios; como un método tecnológico, sino que ha entrado en el mercado comercial bajo la forma de lavadoras, cámaras de vídeo y fotos, frigoríficos, etc..

Aunque la técnica del control difuso ha sido utilizada ampliamente y satisfactoriamente en el control de procesos industriales y productos comerciales, no hay en este momento un procedimiento sistemático para el diseño de controladores difusos hardware. En otras palabras, las aplicaciones actuales de controladores difusos son muy flexibles y dependientes del diseñador. Quizás este problema podría ser visto como una característica de los controladores difusos, pero es realmente difícil para los diseñadores sin una rica experiencia obtener un buen controlador difuso cuando se trata de sistemas complejos.

La forma más fácil y general de implementar un sistema difuso, es sobre un procesador clásico de propósito general, materializando el sistema completamente en software. Por supuesto, este procedimiento es el más económico, pero generalmente produce un controlador más lento. Mediante la utilización de algoritmos adoptados en el diseño de procesadores difusos es posible incrementar significativamente la velocidad de las materializaciones software [Sur95].

Otra alternativa de materialización es adaptar los procesadores clásicos para que ejecuten algunas instrucciones difusas dedicadas [Wat93, Wat96]. Si el diseñador adopta esta solución puede contar con todas las posibilidades de la alternativa software, para lograr sistemas difusos mucho más rápidos. Esta solución es quizás la mejor en términos de balance entre velocidad y generalidad.

Otra solución es separar las operaciones difusas del conjunto de instrucciones clásico y utilizar hardware externo dedicado, llamado muchas veces co-procesador, para ejecutar las operaciones difusas especializadas. Este enfoque alcanza gran velocidad en las operaciones difusas; pero la entrada/salida, entre el co-procesador y el procesador, puede limitar la velocidad de operación de todo el sistema.

Un último enfoque consiste en utilizar un hardware dedicado al proceso difuso. Esto permite alcanzar una velocidad de operación muy alta, pero obtenida a un costo muy alto y usualmente con una gran perdida de generalidad.

Este trabajo se ubica dentro de las propuestas de metodologías de diseño

de FLC hardware asistida por herramientas. Se proponen y analizan dos soluciones a la pérdida de generalidad del hardware dedicado. La primer solución se basa en una arquitectura genérica parametrica, donde las distintas aplicaciones son (micro)programadas y hay muy poca diferencia en el hardware de diferentes aplicaciones (principalmente el tamaño de la ruta de datos). La otra opción propuesta se basa en un hardware totalmente a medida de la aplicación y la generalidad se logra porque las herramientas son las que realizan la generación del hardware de forma automática.

3.2 - CRÍTICAS AL HARDWARE DIFUSO

Una de las principales críticas del uso de procesadores difusos es la de *Surmann* y *Ungering* [Sur95]. Ellos afirman que el uso de los mismos conceptos adoptados en materializaciones hardware; tales como tablas de *look-up* y proceso de optimizado de reglas, permiten que procesadores de propósito general produzcan soluciones mejores que muchos procesadores especiales difusos. Realmente la mayoría de las comparaciones entre realizaciones hardware y software reportadas en la literatura son entre procesadores difusos bien diseñados y código sin optimizar. Por ello, la decisión de adoptar un procesador difuso debe hacerse con cuidado.

Uno de los desarrollos más interesantes en el campo es la introducción de instrucciones difusas especializadas en los procesadores de propósito general. En 1993 y 1996, *Watanabe* [Wat93, Wat96] muestra que se puede obtener una mejora de la velocidad de cálculo en un factor de 2.5 de un programa para control difuso agregando sólo 2 instrucciones: MIN y MAX, a un RISC (*Reduced Instruction Set Computer*). De todos modos, hay gran cantidad de aplicaciones que sólo pueden ejecutarse en un procesador difuso dedicado para cumplir con los requerimientos de tiempo real duro.

Como contrapartida a los circuitos de cálculo, se han aplicado tablas de *look-up* para almacenar los valores de las funciones de pertenencia, de esta forma, cualquier forma de función es permitida, pero se requiere una gran área

de silicio para la materialización de dichas memorias.

Es posible estimar que el número de aplicaciones difusas industriales y comerciales será aproximadamente el doble cada año [Mun94], por lo cual en el año 2005 habrá probablemente 1000000 aplicaciones en el mundo. Por ello, no hay duda que la demanda de hardware dedicado se incrementará, como ha sucedido en la historia del microprocesador. Por el momento es muy difícil predecir que tipo de hardware será el más importante. Se ve que el mercado está dividido en 3 grandes grupos:

- procesadores difusos muy rápidos y muy caros
- coprocesadores difusos rápidos y moderadamente caros
- procesadores clásicos con un conjunto de instrucciones orientado al proceso de algoritmos difusos

3.3 - FUNCIONES DE PERTENENCIA

Uno de los problemas para el diseño de arquitectura de controladores difusos de alta velocidad es el fusificador. El fusificador calcula las funciones de pertenencia de cada variable. Estas funciones de pertenencia se asocian con términos que aparecen en antecedentes o consecuentes de las reglas. Las figuras más usadas para definir funciones de pertenencia son las triangulares y trapezoidales. Hay dos estrategias básicas para la materialización de circuitos que calculen el valor de las funciones de pertenencia:

- Orientadas a memoria. El valor del grado de pertenencia es evaluado fuera de línea y almacenado en una memoria. El desempeño del sistema no se ve afectado por la complejidad de la figura de la función.

- Orientadas al cálculo. La evaluación del grado de pertenencia

es realizada en línea. Una memoria es usada para almacenar los parámetros para calcular dichas funciones. La figura de las funciones tiene un impacto grande sobre el costo (tiempo y área).

Tombs [Tom96] y Baturone [Bat97] presentan arquitecturas analógicas para procesadores difusos de alta velocidad. La propuesta de Tombs se basa en la utilización de modulación por ancho de pulso para la representación de los valores difusos. Ascia [Asc96] realiza la materialización analógica de un controlador que trabaja por reglas activas. Chung y Lee [Chu96] usan redes neuronales para determinar el conjunto de reglas activas en un sistema adaptativo al agregar un peso a cada regla, pero limitan la figura de la función de pertenencia a triángulos que cubran todo el dominio de la variable.

3.4 – FUSIFICADOR: ANÁLISIS DE IMPACTO

El circuito de fusificación puede ser analizado de diferentes puntos de vista. En esta sección se presentan cuatro enfoques que analizan respectivamente: la cantidad de instrucciones necesarias para calcular todas las funciones; el tipo de figuras que pueden tener las funciones de pertenencia; la reutilización de la lógica del multiplicador y el paralelismo. Donde los dos últimos enfoques plantean características especiales de la materialización.

3.4.1 - REDUCCIÓN DE LA CANTIDAD DE INSTRUCCIONES

Durante la etapa de fusificación, el valor crisp de cada variable de entrada debe ser traducido a su representación difusa en cada una de las funciones de pertenencia. Por otra parte, sólo los valores difusos cuyo valor sea diferente de cero son utilizados para definir el conjunto de reglas aplicables, y normalmente son un bajo porcentaje del número total de conjuntos difusos y por ende de las reglas.

Desde el punto de vista del diseñador de FLC se pueden utilizar diferentes enfoques, cuyas prestaciones dependen de la complejidad y tamaño del

circuito, y por ende con diferentes capacidades y costos. Así, los enfoques son:

- En un fusificador de mínima área, el circuito calcula el valor de las funciones de pertenencia para todos los conjuntos difusos, de a uno por vez. Es necesario el uso de muchos registros y ejecutar el circuito tantas veces como funciones de pertenencia se hayan definido.

```
FOR i:=1 TO v DO
   FOR j:=1 TO n[i] DO
      reg[i, j]:=calcula_funcion(i, j, x);

Donde: x es el valor de la variable de entrada, v
representa el número de variables de entrada, n[i] es
el número de funciones de pertenencia de la variable
i.
```

- Una primera optimización busca almacenar sólo los valores no nulos, lo cual reduce la cantidad de registros sin incrementar significativamente el área. El circuito calcula todas las funciones de pertenencia de a una por vez, pero sólo se almacenan en registros los valores diferentes de cero. El número máximo de registros es definido por el máximo número de funciones de pertenencia que se superponen. Debe ejecutarse el circuito, tantas veces como funciones de pertenencia se hayan definido.

```
FOR i:=1 TO v DO
   FOR j:=1 TO n[i] DO
      reg[k]:=calcula_funcion(i, j, x);
      IF reg[k]>0 THEN inc(k);

Donde: x es el valor de la variable de entrada, v
representa el número de variables de entrada, n[i] es
el número de funciones de pertenencia de la variable
i, mientras que k representa un registro auxiliar.
```

- Una optimización adicional busca no calcular las funciones de pertenencia cuyo valor sea cero. El número total de registros y el número de funciones a calcular es igual al máximo número de funciones de pertenencia que

se superponen (definido como ***O*** en 2.8). Se necesita una función que inicializa el circuito de cálculo retornando un *selector* a la primer función de pertenencia cuyo resultado es diferente de cero. A continuación se realizan los ***O*** cálculos de las funciones de pertenencia, al calcular una función se prepara el *selector* para realizar el cálculo de la siguiente. Esta funcionalidad produce un incremento del área necesaria para la fusificación, pero reduce el número de registros necesarios.

```
FOR i:=1 TO v DO
   s:=comienzo(i, x);
   FOR j:=1 TO o[i] DO
      reg[i, j]:=calcula_funcion(s, i, j, x);

Donde: x es el valor de la variable de entrada, v
representa el número de variables de entrada, o[i] es
el número de funciones de pertenencia de la variable
i que se superponen, mientras que s representa un
registro auxiliar. La función comienzo retorna la
primer función de pertenencia de la variable v cuyo
valor es diferente de cero.
```

Analizando la optimización con respecto a las instrucciones, en un esquema de fusificación orientado a memoria [Chi92, Asc96], se obtiene la fusificación de todas las variables lingüísticas en sólo O_m accesos a memoria, pero esta optimización no está muy difundida. En un esquema orientado al cálculo, se debe agregar la lógica necesaria para detectar que funciones se activan y cuales no, lo que produce una instrucción más lenta.

3.4.2 - FIGURA DE LAS FUNCIONES DE PERTENENCIA

En un esquema de fusificación orientado a memoria no es necesario restringir la figura de las funciones de pertenencia, ya que los valores son calculados fuera de línea y almacenados en la memoria; así el fusificador trabaja en un tiempo de respuesta constante, independientemente de la forma de la figura.

La figura de las funciones de pertenencia más ampliamente utilizada para aplicaciones de control es la triangular. En materializaciones hardware de fusificadores orientados al cálculo, por lo general, se las limita a una función

del tipo triangular o trapezoidal. Los parámetros necesarios para un circuito de cálculo de las funciones en línea son:

- Triangular. Cada función es descrita por tres puntos: *izquierdo-abajo*, *medio-arriba* y *derecha-abajo*, y los dos ángulos (para evitar su cálculo).

- Trapezoidal. Cada función es descrita por cuatro puntos: *izquierdo-abajo*, *izquierda-arriba*, *derecha-arriba*, y *derecha-abajo*, y los dos ángulos (para evitar su cálculo).

- Polinomio lineal de ***N*** puntos. Cada función es descrita por los siguientes puntos: $(\boldsymbol{x}_1, \boldsymbol{y}_1)$, $(\boldsymbol{x}_2, \boldsymbol{y}_2)$, $(\boldsymbol{x}_3, \boldsymbol{y}_3)$, ..., $(\boldsymbol{x}_n, \boldsymbol{y}_n)$;, y los $\boldsymbol{n}$-1 ángulos (para evitar su cálculo).

Algunos problemas de control necesitan funciones de pertenencia más complejas que la triangular o trapezoidal. En estos casos, por lo general, se usa el polinomio lineal fijo, donde el tamaño de palabra de memoria soporta un máximo número de puntos. Otra alternativa es diseñar un circuito que permita la definición de polinomios lineales con un número arbitrario de puntos [ver Aco98b, Aco99b y capítulo 6 de ésta tesis]. El enfoque polinomial con tamaño fijo desperdicia memoria y tiempo de cálculo, mientras que la cantidad variable de puntos requiere lógica adicional.

3.4.3 - USO DEL MULTIPLICADOR

En un controlador difuso cuyo fusificador y defusificador se basen en memoria, el bloque de mayor tamaño es sin duda la ROM donde se almacenan todos los valores de todas las funciones de pertenencia previamente calculados. La forma de reducir considerablemente el tamaño del bloque de memoria y por ende de todo el FLS, es mediante el uso de un fusificador y defusificador basado en cálculo.

Para la tabla de defusificación puede aplicarse el método *singleton*, que permite un ahorro considerable de memoria. Con este esquema se plantean dos alternativas principales:

a) Basado en memoria. Cuando las funciones de pertenencia se implementan por memoria, se debe utilizar un bloque por cada función. Los valores son definidos por el producto entre el centro de gravedad de la función de decodificación y el peso calculado en el motor de inferencia (el valor difuso de entrada).

b) Basado en cálculo. Con el método *singleton* sólo el centro de gravedad de la función es almacenado en memoria, dado que la multiplicación es calculada en línea. Este método requiere que la operación de multiplicación sea hecha para cada conjunto difuso de salida.

Durante la fusificación de la variable, en un esquema orientado al cálculo, se requiere también de una operación de multiplicación para cada conjunto difuso de entrada. Como la materialización de multiplicadores es sumamente costosa (tanto en área como en tiempo) [Aco00], el circuito de cálculo de las funciones de pertenencia deben contemplar una ruta de datos que permita el reuso del multiplicador, aún a costa de complicar la ruta de datos.

3.4.4 - PARALELISMO

La materialización de funciones de pertenencia puede realizarse por diversas técnicas para calcular en paralelo el esquema, independientemente del enfoque implementado (memoria o circuito de cálculo). Este paralelismo es aplicado mediante replicación lógica para reducir el número de instrucciones y el tiempo de cálculo [Aco00b]. Por ejemplo, para cada instrucción, el circuito de cálculo de las funciones de pertenencia puede producir los siguientes resultados:

- Una función de una determinada variable.

- Todas las funciones de una determinada variable.

- Todas las funciones de pertenencia de todas las variables.

- Una función de pertenencia diferente de cero de una variable del sistema.

- Todas las funciones de pertenencia diferentes de cero de una variable del sistema.

- Todas las funciones de pertenencia diferentes de cero de todas las variables del sistema.

Además de la tecnología para cada una de las materializaciones se debe tener en cuenta que: a) en las últimas tres opciones debe identificarse la función de pertenencia que genera el valor diferente de cero, lo que implica un espacio adicional de memoria; por otro lado, b) el grado de paralelismo tiene un impacto significativo sobre la complejidad y costo del circuito, ya que se trata de paralelismo por replicación de hardware.

3.5 – SISTEMAS DE DESARROLLO DE FLC

Se han hecho numerosos intentos en desarrollar herramientas de asistencia para la definición de controladores difusos. Se han empleado tres enfoques diferentes: desde el punto de vista de sistemas expertos, como un programador de controladores o procesadores convencionales y materializando el controlador en un lenguaje convencional. A mediados de los años noventa se comienza a proponer un enfoque en el que se genera la arquitectura del procesador a medida de la aplicación, junto con su programa o microprograma.

A continuación se presenta la revisión de algunos sistemas comerciales y

académicos; si bien la lista no es exhaustiva, muestra algunas de las iniciativas más difundidas.

MEDAL. Sistema en el cual se definen variables lingüísticas y se les asignan los polígonos de las funciones de pertenencia y decodificación. Las reglas son especificadas con una sintaxis tipo C. Permite seleccionar el método de defusificación. Genera código para los microcontroladores de Motorola.

ARCHIMEDES. Entorno de desarrollo en lenguaje C de la compañía Archimedes [Archi]. Tiene herramientas de depuración gráficas. Genera código para la familia de microcontroladores de Motorola.

FUDGE. Ambiente de desarrollo de aplicaciones en lógica difusa para controladores Motorola, desarrollado y comercializado por Motorola [MotFL]. Es de libre distribución.

MicropFPL. Paquete para controladores difusos de Hitachi, desarrollado y comercializado por Hitachi [HitFL]. Permite configurar la optimización tiempo/espacio, y se programa en C. Tiene herramientas para la materialización de hardware (ASIC's, placas, y circuitos varios).

OMROM. Ambiente de entrenamiento y compilación para una placa con un controlador difuso para PC desarrollada y comercializada por Omrom [MokOM].

TOGAI. La firma Togai Infralogic [Togai] presenta en el mercado una serie de productos de desarrollo propio, que cubren un amplio espectro del mercado de diseñadores de controladores difusos a medida. Entre sus productos se cuenta con: a) TILShell: herramienta de CASE[1] gráfica para el desarrollo de sistemas expertos en lógica difusa. Tiene interfaces al software TIL y a plataformas hardware. b) Fuzzy-C: conjunto de herramientas que convierte un sistema experto de lógica difusa en un lenguaje de programación C standard. c) MicroFPL: conjunto de herramientas que convierte un sistema experto de lógica difusa en código ejecutable por algún controlador del mercado. d) TILGen: herramienta para la

[1] CASE: Computer Aided System Engineering.

construcción de sistemas expertos difusos a partir de muestras de datos de un sistema existente. Usa una técnica descrita por Dr. Bart Kosko como "*Differential Competitive Learning*" (DCL). e) Fuzzy Computational Acceleration: núcleo de diseño de un ASIC que puede ser configurado desde 8 a 32 bits de precisión, y requiere un área muerta muy pequeña.

FIDE. Ambiente que genera código para controladores de Motorola desarrollado y comercializado por Aptronix [Aptro]. Define un lenguaje de especificación de reglas (FIL), un generador de código, y una herramienta para optimizar el código de ejecución para tiempo real.

XFUZZY. Ambiente para la simulación y generación de código C de sistemas de control basados en lógica difusa, desarrollado en el Centro Nacional de Microelectrónica, sede Sevilla, España [Mas94, Bat98e].

3.6 - BASES DEL PROYECTO

El objetivo de ésta tesis es desarrollar una metodología para el diseño de controladores difusos a medida de la aplicación. En los primeros capítulos se ha mostrado el estado del arte en el área del control difuso. En lo que resta de esta memoria, se proponen y aplican dos metodologías para el diseño de FLC.

a) La primera se basa en la arquitectura pasiva (presentada en el capítulo 4) de un controlador con sólo una unidad de cálculo orientada al proceso de operaciones dedicadas a los FLC. Su grado de parametrización hardware es el tamaño de palabra, interno de la ALU y de la ruta de datos. Su adaptación a la aplicación concreta se realiza por medio del microprograma de control. El desarrollo de una aplicación utilizando esta técnica se apoya en un conjunto de herramientas (presentadas en el capítulo 5) que permiten la generación automática del microprograma de control.

b) La segunda metodología se basa en una arquitectura activa o de altas prestaciones, con un motor de inferencias difusas dirigido por reglas (presentado en el capítulo 6). El grado de adaptación a la aplicación es total, ya que el conjunto de herramientas (presentada en el capítulo 7) genera tanto el hardware dedicado como el microprograma de control. El hardware se basa en múltiples unidades de cálculo generadas a medida de la aplicación.

Como ejemplo de la primer metodología se diseña un ASIC, usando tecnología ES2 de una micra (Apéndice C). Esta primer experiencia permitió conocer los detalles del diseño de circuitos de estas características [Sut97]; aunque por coste y escasez de recursos se optó por analizar las materializaciones (presentadas en el capítulo 8) utilizando como plataformas FPGAs de Xilinx.

Ambas metodologías se orientan a asistir al diseñador de FLC en todo el ciclo de diseño y se apoyan en las herramientas informáticas propuestas. Las herramientas parten de un archivo de definición de controladores difusos, cuyo lenguaje ha sido fuertemente influenciado por FIL (*Fuzzy Inference Language*) definido por Aptronix [Aptro]. La arquitectura pasiva se conforma por un conjunto de módulos VHDL genéricos (que utilizan estructuras iterativas para definir diferentes tamaños de palabra) y, en ese contexto, las herramientas a partir del archivo de definición de controladores obtienen el microprograma que controla la arquitectura. En la arquitectura por reglas activas, las herramientas analizan el archivo con la definición del FLC y se generan tanto los módulos VHDL como el microprograma que los controla. Cabe destacar que la descripción del controlador para ambos casos se basa en el mismo lenguaje de especificación (Fig. 3.1), así una misma definición de FLC puede ser sintetizada para cualquiera de las dos arquitecturas.

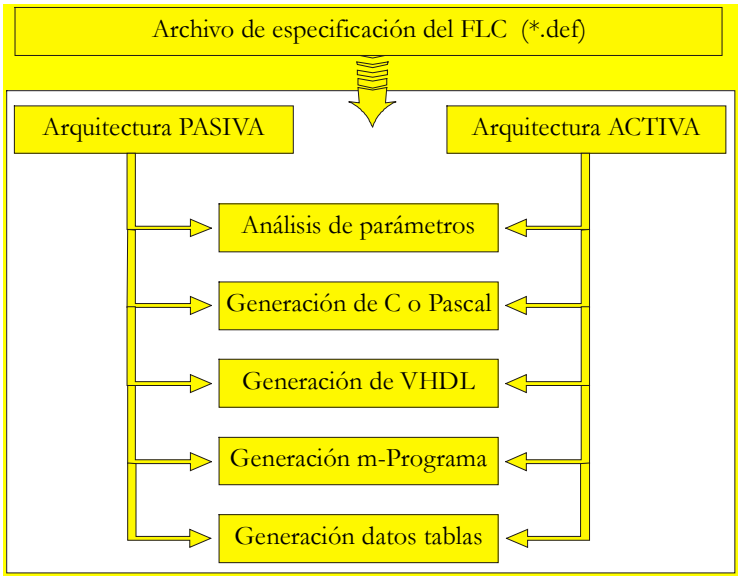


Fig. 3.1 – Diagrama de la herramienta de desarrollo

Las metodologías para el diseño de FLC propuestas están sujetas a las siguientes consideraciones de diseño:

- Se utiliza el modelo difuso de Mamdani [Mam75] (descrito en 2.6.1), debido a que ofrece mayor flexibilidad para la selección de técnicas de cálculo.

- Si bien al momento del diseño o prototipación se ha utilizado indistintamente captura esquemática o VHDL, el objetivo es obtener código VHDL con descripciones de los controladores que puedan ser simuladas y sintetizadas.

- Ambos controladores utilizan fusificación calculando en línea las funciones y el método del *centroide* para defusificar (sección 2.5.4.2). Las funciones de pertenencia calculadas se basan en "*polinomios lineales con cantidad variable de puntos*" (descrito en 3.4.2). Las funciones calculadas pueden ser reemplazadas por tablas con valores precalculados (sección 3.3).

- Los controladores materializan el algoritmo propuesto en la Fig. 2.5, utilizando una única unidad de cálculo la arquitectura pasiva, mientras que los basados en reglas activas tienen múltiples unidades de cálculo.

- Las rutas de datos deben permitir la reutilización de recursos (registros, multiplicador, operaciones de la ALU, ...), de tal forma que la unidad de control o el microprograma pueda seleccionar que dato sea ingresado al circuito (descrito en la sección 3.4.3).

- En ambas propuestas se busca reducir la cantidad de instrucciones del microprograma (sección 3.4.1). Como se busca la mayor velocidad de proceso, siempre se trata que el ciclo de instrucción no crezca demasiado como para reducir la velocidad de cálculo de todo el sistema.

- Sólo en la segunda propuesta se sacrifica coste económico, buscando mejorar el desempeño del FLC; para lo cual se utilizan algunas de las técnicas de paralelismo (sección 3.4.4).

3.7 – EJEMPLOS PROPUESTOS

En esta tesis se propone una metodología de diseño de controladores difusos, y un conjunto de herramientas que asisten al diseñador en su tarea. Para demostrar la aplicación de la metodología y probar el correcto funcionamiento de las herramientas, se necesita de un problema práctico. Este ejemplo debe estar lo suficientemente *bien* definido como para no generar problemas inherentes al ejemplo y no a las herramientas o a la metodología. Teniendo en cuenta estas consideraciones, se organizan los ejemplos en tres fases:

- Una aplicación base que se utiliza para demostrar paso a paso todas las etapas de las metodologías;

- Una aplicación secundaria muy bien documentada que permite analizar otro ejemplo en detalle; y

- Otra serie de ejemplos no tan documentados en este informe, que permiten considerar algunas características diferentes a las dos aplicaciones principales.

Se decide utilizar como aplicación base un ejemplo propuesto por D. Nguyen y B. Widrow en 1989 [Ngu89] y rescatado por Bart Kosko en su libro [Kos92], que ya es un clásico en la literatura de controladores difusos [Des95, Gal95, Kan98]. En este apartado se define completamente el ejemplo. Se define el problema para el guiado automático en el estacionamiento marcha atrás de un camión en una dársena de carga. El camión se encuentra sobre una playa de estacionamiento sin obstáculos en alguna posición sobre los ejes *x*-*y*, con un ángulo sobre la vertical ϕ (*phi*) dado. El algoritmo de guiado lo efectúa un controlador difuso formando parte de un sistema de guiado automático propuesto por *Bart Kosko* [Kos92].

La aplicación secundaria seleccionada se basa en el modelo propuesto por Robert H. Cannon en 1967 [Can67]: "*el péndulo invertido*". Este controlador es un clásico de sistemas con modelo matemático no lineal lo suficientemente complejo como para justificar un control no basado en el modelo [Set99, Son99, Son00, Cha01, Yia01, JYi02]. El ambiente a controlar incluye un servo que mueve un carro sobre una plataforma mecánica, tratando de equilibrar (sin que caiga) una barra rígida, cuyo eje está sobre el carro y tiene un codificador angular solidario al péndulo.

Los ejemplos adicionales se realizan adaptando los FLC propuestos por Aptronix [Aptro] como demostraciones para su sistema de desarrollo sobre micro-controladores estándares del mercado. Sus características se resumen

en éste capítulo y se presentan detalladamente en el Apéndice B.

3.7.1 - APLICACIÓN BASE

En la Fig. 3.2 se describe el problema a resolver: un vehículo tiene que moverse (Fig. 3.2.a) de tal manera que su parte trasera (punto *T*) coincida con el centro (punto *C*) de un muelle de descarga (Fig. 3.2.b), siendo el movimiento del vehículo hacia atrás. El objetivo es que el camión arribe a la posición (x,y) coincidente con la posición (x_m,y_m) del muelle de estacionamiento y que la posición final del extremo del vehículo tenga un ángulo $\phi_f = 0$.

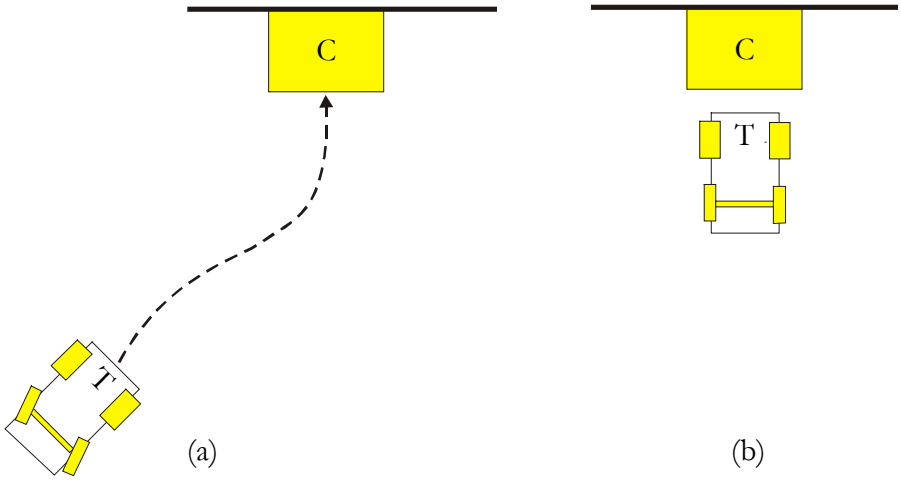


Fig. 3.2 – Enunciado del problema

El sistema de guiado completo consta de:

- Sensores cuya lectura permite calcular las coordenadas x,y del punto T y la inclinación ϕ del vehículo con respecto al eje y (Fig. 3.3);

- Un controlador difuso que calcula el ángulo de giro ideal θ de las ruedas delanteras (Fig. 3.3) en función de x y ϕ (el valor de y es irrelevante si se supone que, inicialmente, el vehículo está a cierta distancia mínima del muelle);

- Un sistema de regulación del ángulo de las ruedas delanteras: en función de la diferencia entre el ángulo ideal, calculado por el controlador difuso, y el ángulo real; se

modifica este último;

- Un sistema de regulación de la velocidad: mientras *y* es superior a cierto valor mínimo *d* la velocidad es constante, por ejemplo *v* m/s; a partir de la distancia mínima *d* la velocidad se reduce hasta la parada del vehículo en la posición deseada.

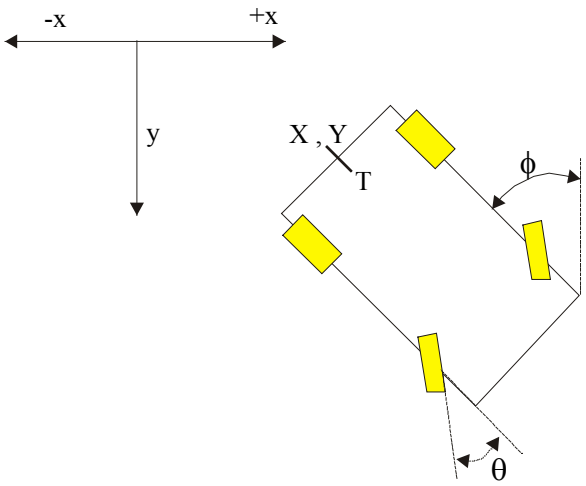


Fig. 3.3 - Definición de las coordenadas del vehículo

En la Fig. 3.4 se puede ver un esquema de bloques del sistema. (controlador, sensores, elementos de acción, etc.)

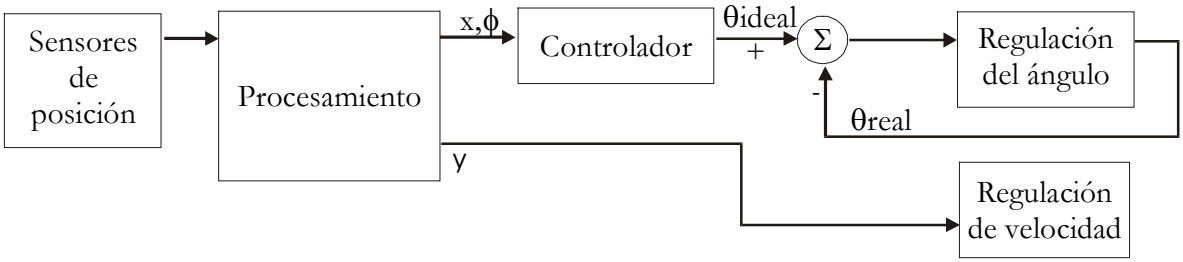


Fig. 3.4 - Diagrama de bloques

ESPECIFICACIÓN DEL CONTROLADOR DIFUSO

Para la simulación se utiliza un modelo discreto simplificado del sistema *controlador* ↔ *vehículo*. Se supone que los pasos discretos de simulación poseen un período de muestreo que es igual a *t* segundos. Con esto la distancia recorrida en ese periodo es igual a $r = v.t$. Si se supone que el tiempo de reacción del elemento de control (regulador del ángulo) es despreciable con respecto al

periodo t, es decir, el tiempo en llevar el ángulo de las ruedas del vehículo del ángulo real al ángulo ideal entregado por el controlador es muy pequeño

$$tiempo\ que\ demora\ = \theta_{real}\ =\ \theta_{ideal}\ <<\ \text{periodo}\ t.$$

Entonces, el cálculo de la nueva posición del vehículo con respecto a la anterior se resume al siguiente grupo de ecuaciones:

$$\phi_{nuevo} = \phi + \theta,$$

$$x_{nuevo} = x - r\ .\ \text{sen}\ \phi_{nuevo}$$

$$y_{nuevo} = y - r\ .\ \cos\ \phi_{nuevo}$$

Donde, $\boldsymbol{\theta}$ es el valor generado por el controlador difuso, $\boldsymbol{\phi}$ el ángulo del vehículo, $\boldsymbol{x}$ e $\boldsymbol{y}$ las coordenadas del vehículo respecto del muelle de estacionamiento. En la Fig. 3.5 se puede ver la construcción geométrica correspondiente.

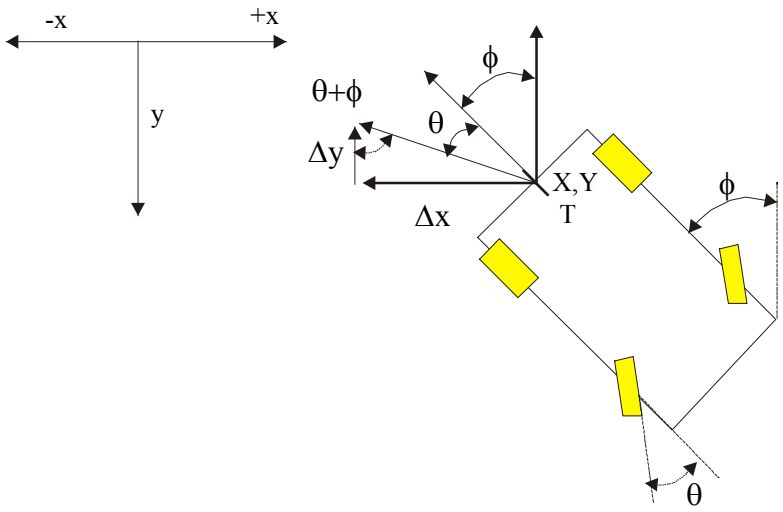


Fig. 3.5 - Cálculo de la trayectoria

Entonces, en un período de t segundos, el vehículo recorre $r = v.t$ metros en una dirección inclinada en θ grados con respecto a la dirección inicial del eje del vehículo ϕ. Con esto los movimientos en $\boldsymbol{x}$ e $\boldsymbol{y}$ se pueden escribir como:

$$\Delta x = -r \cdot \text{sen}(\phi + \theta),$$

$$\Delta y = -r \cdot \cos(\phi + \theta).$$

El rango de las diferentes variables es el siguiente:

- La distancia x está comprendida entre -50 y 50 metros y se mide con una precisión de 0,1 metros; el centro del muelle está en x = 0;
- La distancia y comprendida entre 0 y 100 metros se mide también con una precisión de 0,1 metros; el centro del muelle está en y = 0;
- El ángulo ϕ está comprendido entre -180 y 180 grados y se mide con una precisión de 1 grado.
- El ángulo θ está comprendido entre -30 y 30 grados y se mide con una precisión de 1 grado.

El objetivo es que, sea cual fuese la posición inicial del vehículo, siempre y cuando $y_{inicial}$ sea mayor que cierto valor mínimo, su estado final sea: $x = y = 0$ metros con $\phi = 0$ grados.

ELEMENTOS DEL ALGORITMO DE CONTROL DIFUSO

Las funciones de pertenencia de la variable x son las siguientes:

LE : Left (izquierda)
LC : Left Center (izquierda – centro)
CE : Center (centro)
RC : Right Center (derecha - centro)
RI : Right (derecha)

En la Fig. 3.6 se puede ver una representación gráfica de las funciones de pertenencia.

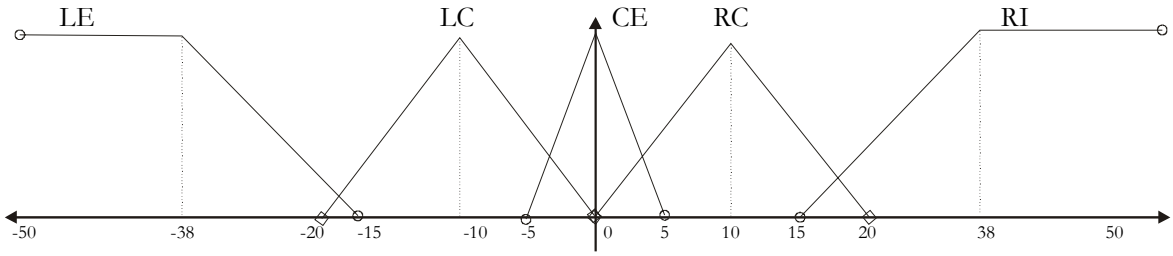


Fig. 3.6 – Funciones de pertenencia de X

Las funciones de pertenencia de la variable ϕ son:

RB : Right Below (derecha – abajo)
RU : Right Upper (derecha – arriba)
RV : Right Vertical (derecha – vertical)
VE : Vertical (vertical)
LV : Left Vertical (izquierda – vertical)
LU : Left Upper (izquierda – arriba)
LB : Left Bellow (izquierda – abajo)

Obsérvese que los arcos correspondientes a los ángulos ϕ se miden desde el eje vertical hasta el costado del vehículo, girando en el sentido contrario al de las agujas del reloj para los valores positivos. Las funciones de pertenencia de ϕ se representan en la Fig. 3.7.

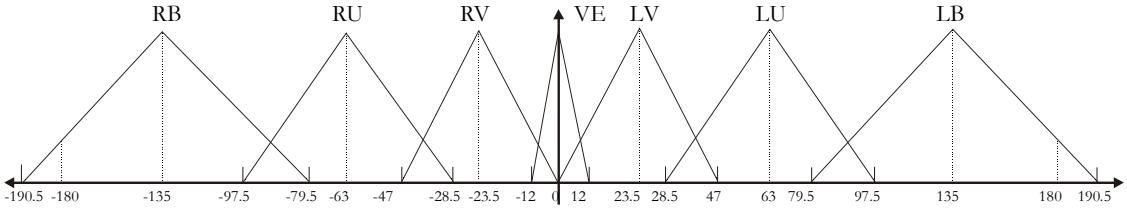


Fig. 3.7 – Funciones de pertenencia de ϕ

Las funciones de decodificación de la variable θ son:

NB : Negative Big (negativo - grande)
NM : Negative Medium (negativo - medio)
NS : Negative Small (negativo - pequeño)
ZE : Zero (cero)
PS : Positive Small (positivo - pequeño)
PM : Positive Medium (positivo - medio)
PB : Positive Big (positivo - grande)

Las funciones de decodificación de la variable θ se representan en la Fig. 3.8, en este caso los valores positivos de θ corresponden al giro de las ruedas hacia la derecha (como en la Fig. 3.3).

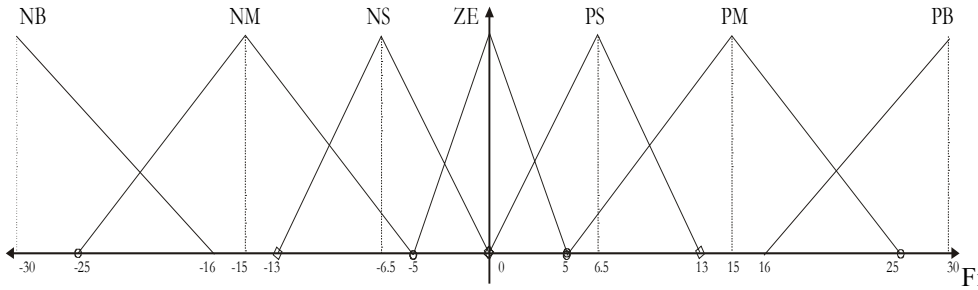


Fig. 3.8 – Funciones de pertenencia de θ

Las 35 reglas de inferencia se describen en la Tabla 3.1.

ϕ \ X	LE	LC	CE	RC	RI
RB	[1] **PS**	[2] **PM**	[3] **PM**	[4] **PB**	[5] **PB**
RU	[6] **NS**	[7] **PS**	**PM**	**PB**	**PB**
RV	**NM**	**NS**	**PS**	**PM**	**PB**
VE	**NM**	**NM**	[18] **ZE**	**PM**	**PM**
LV	**NB**	**NM**	**NS**	**PS**	**PM**
LU	**NB**	**NB**	**NM**	**NS**	**PS**
LB	**NB**	**NB**	**NM**	**NM**	[35] **NS**

Tabla 3.1 - Reglas de inferencia

Este cuadro debe interpretarse de la siguiente manera: cada casilla intersección de un conjunto en x y otro en y da por resultado una salida perteneciente al conjunto indicado en dicha casilla. Dicho en notación **IF-THEN** se lee:

- casilla 1, **IF** x **is** LE **AND** ϕ **is** RB **THEN** θ **is** PS,
- casilla 2: **IF** x **is** LC **AND** ϕ **is** RB **THEN** θ **is** PM,

- casilla 18: **IF** x **is** CE **AND** ϕ **is** VE **THEN** θ **is** ZE, etc.

3.7.2 – APLICACIÓN SECUNDARIA

Estabilizar un péndulo invertido es un simple problema de control que involucra un sistema inherentemente inestable descrito por ecuaciones diferenciales de segundo orden. El sistema de control considerado se muestra en la Fig. 3.9, y consiste del modelo matemático para el péndulo invertido, del cual el ángulo ***a*** es periódicamente leído por el registro y la velocidad angular ***v*** es calculada por diferencias entre consecutivas lecturas del ángulo. Los valores del ángulo y velocidad (angular) son pasados como entradas al FLC, el cual calcula la fuerza de control ***f*** a ser aplicada al péndulo invertido.

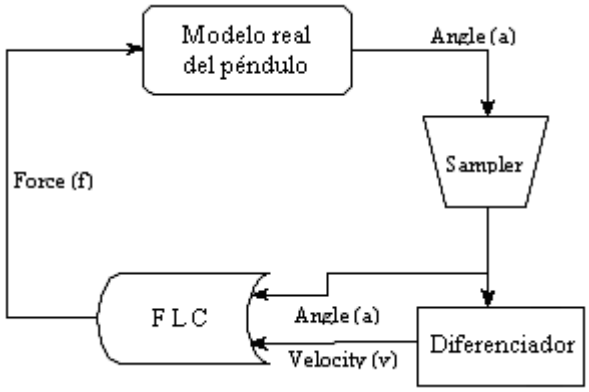


Fig. 3.9 – FLC del péndulo invertido

El FLC es especificado en alto nivel, las reglas definen las relaciones entre las variables difusas: ***ángulo***, ***velocidad*** y ***fuerza***.

MODELO DEL PÉNDULO INVERTIDO

El modelo del péndulo invertido es mostrado en la Fig. 3.10 y consiste de una viga de longitud ***l*** que puede pivotar sobre un carro que se mueve sobre el plano vertical. A tal fin se considera una versión simplificada del modelo completo, asumiendo que el carro no tiene masa y que la masa de la viga ***m*** es concentrada en su centro. Sobre ***m*** actúan dos fuerzas, gravedad (aceleración gravitacional ***g***) y la fuerza de control externa ***f*** la que es responsable de estabilizar el sistema. Otras variables del sistema son el ángulo de inclinación a la perpendicular ***a***, la

velocidad angular ***v*** y la aceleración angular ***j***.

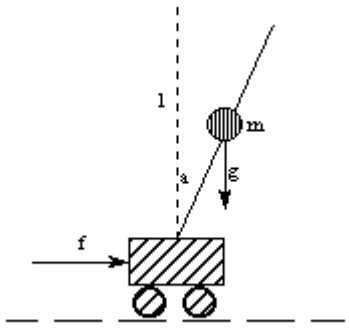


Fig. 3.10 – Sistema del péndulo invertido

En cualquier momento, el péndulo es gobernado por las siguientes ecuaciones:

$$da = v \cdot dt + \frac{1}{2} \cdot j \cdot dt^2$$

$$dv = j \cdot dt$$

$$j = \frac{3 \cdot g}{2 \cdot l} \cdot \sin(a) + \frac{6}{m \cdot l} \cdot f \cdot \cos(a)$$

De la última ecuación se deducen los valores límites para la fuerza de control ***f*** para equiparar la aceleración gravitacional (para cierto ángulo ***a***) y llevar al péndulo a la posición de equilibrio.

$$\left|\frac{6}{m \cdot l} \cdot f \cdot \cos(a)\right| > \left|\frac{3 \cdot g}{2 \cdot l} \cdot \sin(a)\right| \quad \text{ó} \quad |f| > \frac{g}{4} \cdot |\tan(a)|$$

$$|f| > 2.85 \cdot g$$

Entonces ***f*** debe ser mayor que 2.85 ***g*** para cuando el ángulo ***a*** = 85° (cerca del peor caso, cuando 90°).

VARIABLES DIFUSAS

Se deben especificar las variables de entrada Angulo (***a***), Velocidad (***v***) y la variable de salida Fuerza (***f***). Se concentran las funciones de pertenencia

difusas cerca del deseado punto de equilibrio para proveer una mayor precisión para el proceso de estabilización. Las funciones de pertenencia para las variables ángulo (***a***), velocidad (***v***) y fuerza (***f***) son las mismas:

NL : Negative Large (negativo grande)
NM : Negative Medium (negativo medio)
NS : Negative Small (negativo pequeño)
Z : Zero (cero)
PS : Positive Small (positivo pequeño)
PM : Positive Medium (positivo medio)
PL : Positive Large (positivo grande)

Para el ángulo (***a***) se especifica un rango de dominio entre -1.57 (-pi/2) y 1.57 (+pi/2) para el mínimo y máximo valor respectivamente. Ese rango se divide en 7 funciones de pertenencia con un 50% de superposición (Fig. 3.11).

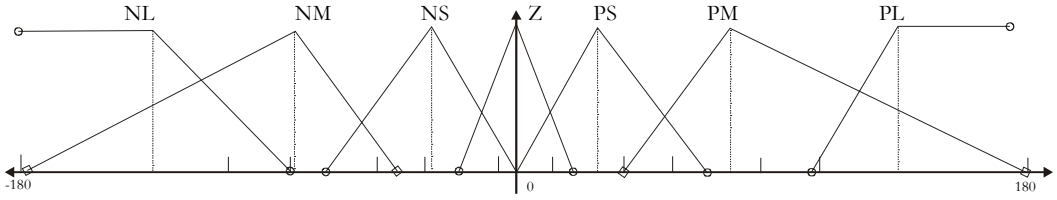


Fig. 3.11 – Variable difusa ANGULO (***a***)

Para la velocidad (***v***) se especifica un rango de dominio entre -30 y +30 para el mínimo y máximo valor respectivamente. Ese rango se divide en 7 funciones de pertenencia con un 50% de superposición (Fig. 3.12).

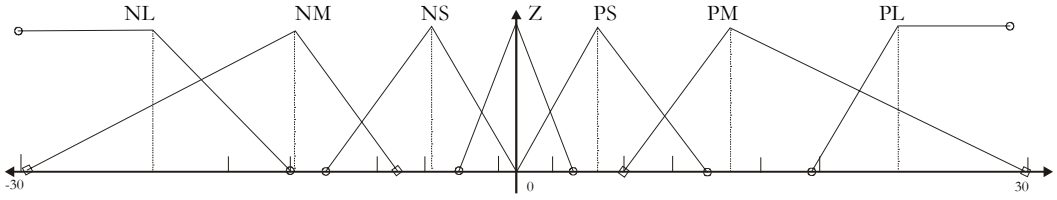


Fig. 3.12 – Variable difusa velocidad angular (***v***)

Para la variable fuerza (***f***) se especifica un rango de dominio entre -30 y +30 para el mínimo y máximo valor respectivamente. Ese rango se divide en 7 funciones de pertenencia con un 50% de superposición (Fig. 3.13).

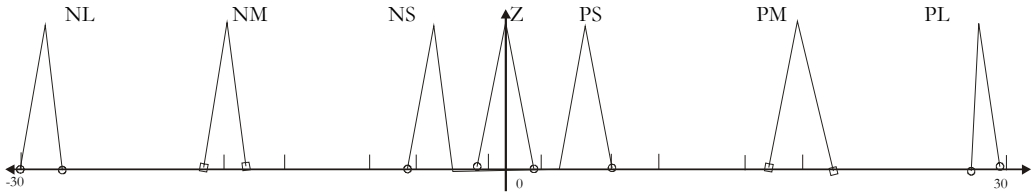


Fig. 3.13 – Singleton de la variable difusa fuerza (***f***)

REGLAS DE CONTROL

Para las variables (***a***) y (***v***) se debe definir la relación sobre la variable de salida (***f***) como términos de reglas. Basándose en el sentido común, intuición y experiencia se definen las guías para controlar el péndulo: "si ángulo y velocidad son del mismo signo, entonces la fuerza debe ser de signo opuesto", con algunas variaciones menores cerca del punto de equilibrio. La Tabla 3.2 muestra la matriz de reglas.

		VELOCIDAD						
		NL	**NM**	**NS**	**Z**	**PS**	**PM**	**PL**
ANGULO	**NL**	PL	PL	PM	PM	PS	PS	Z
	NM	PL	PM	PM	PS	PS	Z	NS
	NS	PM	PM	PS	PS	Z	NS	NS
	Z	PM	PS	PS	Z	NS	NS	NM
	PS	PS	PS	Z	NS	NS	NM	NM
	PM	PS	Z	NS	NS	NM	NM	NL
	PL	Z	NS	NS	NM	NM	NL	NL

Tabla 3.2 – Tabla de reglas del Péndulo Invertido

3.7.3 – OTRAS APLICACIONES

En este apartado se presenta un resumen de las características de los controladores difusos, principalmente definidos como demostración por la empresa Aptronix [Aptro]. Dicha empresa comercializa un conjunto de herramientas, bajo la denominación comercial de FIDE, para desarrollar FLC sobre plataformas de microcontroladores[1] de Motorola (68HC05, 6805, 68HC08, 68HC11, 68HC12, 68HC33x,), Intel (80C196 y 80C296), Siemens Components (SAE81C99A) y Omron Electronics (FP-3000, FP-5000).

[1] Actualización de productos a Marzo de 2001 [Aptro].

Las aplicaciones seleccionadas, explicadas completamente en el Apéndice B, son resumidas a continuación:

- **Auto Focus**. Controla el sistema de enfoque de una cámara fotográfica Se compone de 3 variables de entrada, 3 de salida, 19 funciones de pertenencia y 18 reglas.

- **Servomotor**. Define el controlador del motor de un servo. Se compone de 2 variables de entrada, 1 de salida, 23 funciones de pertenencia y 64 reglas.

- **Péndulo Invertido Doble**. Sistema de control de un péndulo invertido con dos tramos. Se compone de 2 variables de entrada, 1 de salida, 17 funciones de pertenencia y 30 reglas.

- **Temperatura de fundición de vidrio**. Define un controlador de temperatura para el trabajo de vidrio. Se compone de 2 variables de entrada, 1 de salida, 17 funciones de pertenencia y 24 reglas.

- **Temperatura de fundición de vidrio con manejo del error**. Define un controlador de temperatura para el trabajo de vidrio que ofrece mejores prestaciones. Se compone de 2 variables de entrada, 1 de salida, 19 funciones de pertenencia y 21 reglas.

- **Temperatura de reactor**. Define un controlador de temperatura para el trabajo de vidrio. Se compone de 3 variables de entrada, 2 de salida, 32 funciones de pertenencia y 234 reglas.

- **Assembly**. Controlador de un sistema de enfoque para el ensamblado de un láser. Se compone de 2 variables de entrada, 2 de salida, 14 funciones de pertenencia y 20 reglas.

- **Lavadora automática**. Define un controlador de una lavadora de ropa automática. Se compone de 2 variables de entrada, 1 de salida, 11 funciones de pertenencia y 9 reglas.

- **Sistema de navegación de un vehículo**. Define un controlador de navegación de un vehículo, que le permite esquivar obstáculos durante el desplazamiento. Se compone de 4 variables de entrada, 2 de salida, 20 funciones de pertenencia y 16 reglas.

4 – ARQUITECTURA PASIVA

4.1 - INTRODUCCIÓN

Para desarrollar un sistema digital materializando un esquema de cálculo una posible solución consiste en utilizar una herramienta clásica de síntesis de alto nivel, basada en algoritmos de planificación (*scheduling*), asignación de recursos y memorias y en una herramienta de síntesis lógica. En el caso de controladores de altas prestaciones, en los que se requiere el funcionamiento en paralelo de varias unidades aritmético-lógicas, los resultados obtenidos son satisfactorios [Ike91, How92, Rai93, Asc96, Chu96]. En cambio, para controladores que procesan un número relativamente reducido de reglas, y no requieren una frecuencia de muestreo (*throughput*) muy alta - caso de la mayoría de los controladores - los resultados fueron decepcionantes [Wat90, Tog96, Des97].

Por ese motivo se adopta una estrategia diferente, más sencilla y que a la postre, resulta ser más eficiente en el caso de controladores de complejidad media/baja. El sistema consta de una ruta de datos parametrizada y de una unidad de control que ejecuta un microprograma sin bifurcaciones. El diseño de la ruta de datos se reduce a la colocación (creación de las instancias) e interconexión de dos bloques parametrizados descritos en VHDL (unidad aritmético-lógica y puertos de salida) y a dos memorias compiladas (las tablas de las funciones de pertenencia y el banco de registros de doble puerto). La unidad de control contiene un generador de fases (descrito en VHDL e independiente de la aplicación), un contador de microinstrucciones (bloque VHDL parametrizado) y una memoria de microprograma (ROM compilada). El diseño de la unidad de control se reduce prácticamente a la generación del contenido de la ROM del microprograma.

Para la generación del contenido de la ROM del microprograma se han desarrollado una serie de herramientas descritas en el capítulo 5. Para la síntesis lógica de los bloques descritos en VHDL (unidad aritmético-lógica, puertos de

salida, generador de fases, contador de microinstrucciones) se puede utilizar una herramienta comercial. Sin embargo dichos bloques tienen una estructura iterativa (*bit-slice*) con lo cual la relación (número total de puertas) / (número de puertas realmente utilizadas) es muy alta. Por tanto la síntesis lógica puede también ser realizada de forma manual en un tiempo razonable.

Las principales etapas del diseño son las siguientes:

- Definición del esquema de cálculo y generación del microprograma de control; al término de esta etapa habrán sido definidos los parámetros de los bloques y los contenidos de las ROM compiladas.
- Síntesis lógica de los bloques descritos en VHDL.
- Compilación de las macro-células (memorias).
- Ensamblaje de todos los bloques.

4.2 - ESTRUCTURA GENERAL

La estructura general del circuito se muestra en la Fig. 4.1. A continuación se describe brevemente la función de cada uno de los principales bloques. El bloque *tablas* contiene las funciones de pertenencia (funciones de las variables de entrada) y las funciones de decodificación (funciones de la variable interna *w*). La variable de control *tabla* selecciona una función particular. El número de bits *m* elegido para codificar el intervalo [0,1] define el tamaño de las variables (palabras) internas.

El *banco de registros* almacena todas las variables internas del esquema de cálculo. Su organización interna y la definición de las señales de sincronización se diseñan para que en un mismo ciclo del microprograma se pueda leer el contenido de dos registros *R(j)* y *R(k)* y escribir en un registro *R(i)*. La señal de control *t/ual* permite seleccionar el origen del dato que se escribe en el registro *R(i)* puede ser: el valor de una función de pertenencia, el valor de una función de decodificación

(*t/ual* = 1), o el resultado de un cálculo efectuado por la unidad aritmético-lógica (*t/ual* = 0).

La *unidad aritmético-lógica* contiene tres biestables *cent, xent* y *s/r* (sumar/restar) controlados por las variables de control *ff(1:0)* y *sh*. Las variables de control *func(3:1)* definen la función realizada por la unidad aritmético-lógica (sumar, restar, etc.). La salida *serie* es utilizada por el algoritmo de división.

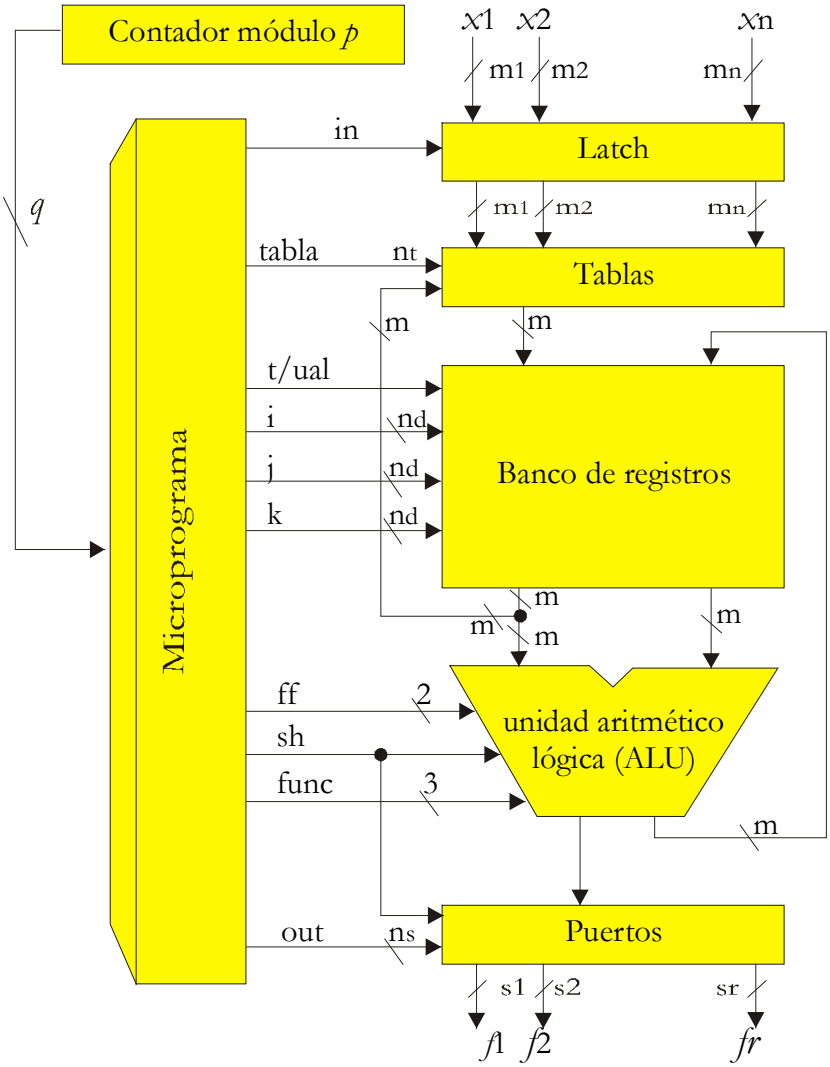


Fig. 4.1 - Estructura general del circuito

El bloque *puertos* contiene un registro de desplazamiento que convierte el resultado de una división de serie en paralelo, y tantos *latches* como funciones de salida. La señal de control *sh* se utiliza para añadir un nuevo bit del cociente al contenido actual del registro de desplazamiento, y la señal de control *out* para

transferir un dato del registro de desplazamiento al puerto de salida seleccionado. Cuando *out = 00...0* no se realiza ninguna transferencia.

La *memoria del microprograma* genera todas las señales de control:

i:	dirección de escritura
j:	dirección del primer operando
k:	dirección del segundo operando
tabla:	selección de una función de pertenencia o de decodificación
t/ual:	origen del dato que se escribe
ff:	control de los *flip-flops* de la unidad aritmético-lógica
sh:	control de los *flip-flops* de la unidad aritmético-lógica y del registro de desplazamiento del bloque *puertos*
func:	operación de la unidad aritmético-lógica
in:	control del *latch* de entrada
out:	control de los *latches* de salida

La especificación completa del circuito se reduce a la definición de los parámetros, a la generación del contenido de la memoria del microprograma, y a la organización interna y programación del bloque *tablas*. Los parámetros a definir son:

n_d :	número de bits con la dirección de los registros
n_t :	número de bits de selección de las funciones de pertenencia y de decodificación
n:	número de entradas
$m_1, m_2, \ldots, m_n$:	número de bits de las entradas
m:	número de bits de las variables internas
r:	número de salidas
$s_1, s_2, \ldots, s_r$:	número de bits de las salidas
n_s:	número de bits de selección de las salidas
p:	número de estados del contador de microprograma
q:	número de bits del contador de microprograma

4.3 - SINCRONIZACIÓN DE LAS SEÑALES

La ejecución de una microinstrucción se descompone en cuatro fases:

- Búsqueda de una nueva microinstrucción (incrementando el contador de programa),

- Lectura de los operandos *R(j)* y *R(k)*,
- Cálculo o lectura de una tabla,
- Escritura del resultado de la microinstrucción en *R(i)* y en el registro de desplazamiento de salida, actualización de los biestables de la unidad aritmético-lógica, actualización del contador de microprograma.

Para ello se generan tres señales de sincronización *ph0, ph1* y *ph2*. En la Fig. 4.2 se muestra un gráfico que corresponde a la ejecución de las transferencias funcionales siguientes:

$R(i) <= F[R(j), R(k)]$	siendo *F* una función de la unidad aritmético-lógica
$R(i) <= A(x_l)$	siendo *A* una función de pertenencia
$R(i) <= B(w)$	siendo *B* una función de decodificación

El primer flanco de subida de *ph0* cierra el *latch* de la dirección *j* en el decodificador de direcciones del banco de registros, y el segundo flanco cierra el *latch* de la dirección *i*. El flanco de subida de *ph2* cierra el *latch* de la dirección *k*. El flanco de bajada de *ph1* controla la escritura en todos los registros controlados por flanco (registro *R(i)* del banco de registros, contador de microprograma, *flip-flops* de la unidad aritmético-lógica, registro de desplazamiento de salida).

Se supone que el registro *R(0)* del banco de registros contiene el valor *0* y es necesario para algunas operaciones, por ejemplo para alinear los operandos en caso de cálculo de un cociente. Una de las primeras microinstrucciones consiste en escribir *0* en *R(0)*, utilizando la operación *cero* de la unidad aritmético-lógica. Por tanto para ejecutar una microinstrucción que no modifique el estado del banco de registros basta con escribir *0* en *R(0)*, razón por la cual no es necesario que el banco tenga una señal de control de la escritura.

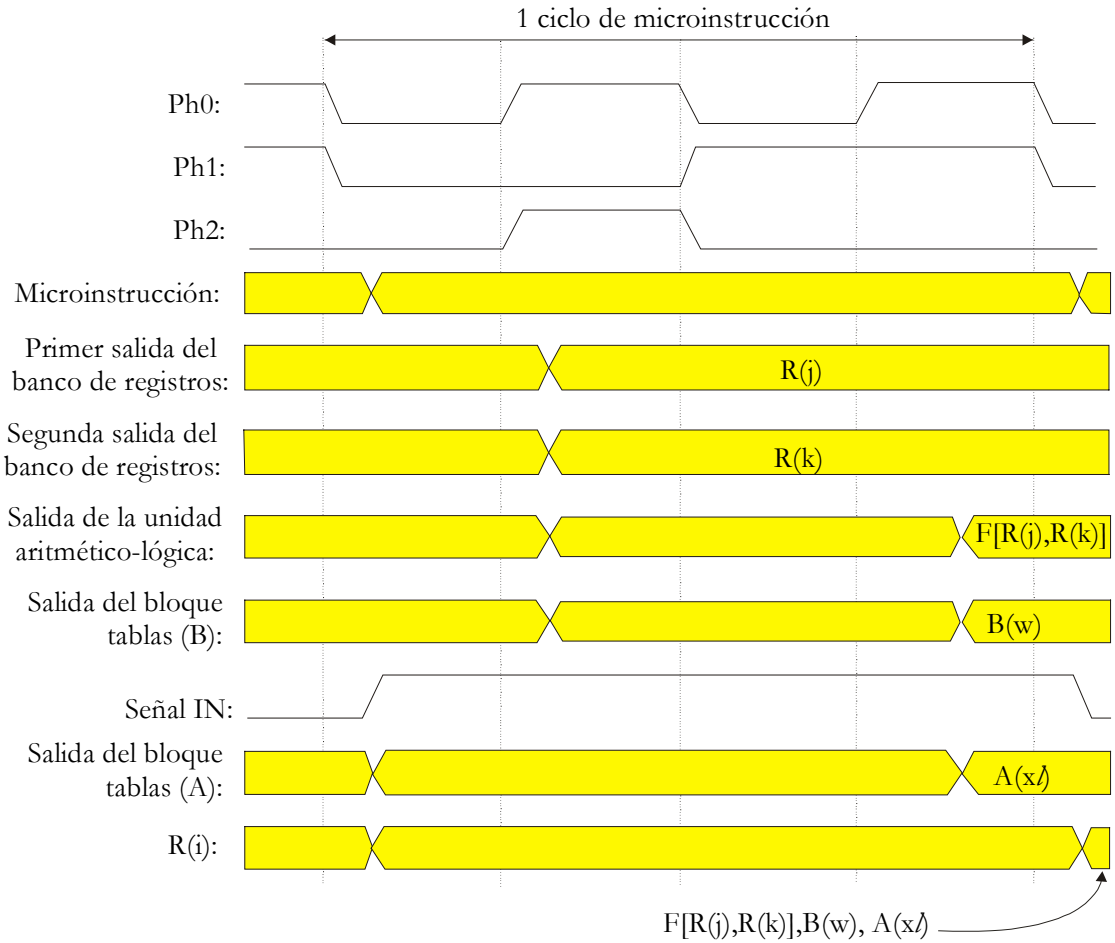


Fig. 4.2 – Diagrama de señales

4.4 - ESTRUCTURA DE LAS TABLAS

En la Fig. 4.3 se muestra la estructura interna del generador de funciones de pertenencia y de decodificación. Cada función (o tabla) ocupa una serie de posiciones contiguas de la memoria. El decodificador asocia a cada número de la tabla la dirección inicial de la misma, y el número de la variable a utilizar. Por tanto el decodificador es un circuito combinacional que tiene que ser sintetizado en cada caso. Con esta arquitectura las tablas pueden tener tamaños diferentes (en función del número de bits de la variable correspondiente) y una misma tabla puede ser compartida por varias variables (puede darse el caso de que una misma función de pertenencia sirva para varias variables de entrada). La memoria puede ser una ROM compilada o una PROM externa (en especial si se integra el controlador en una FPGA).

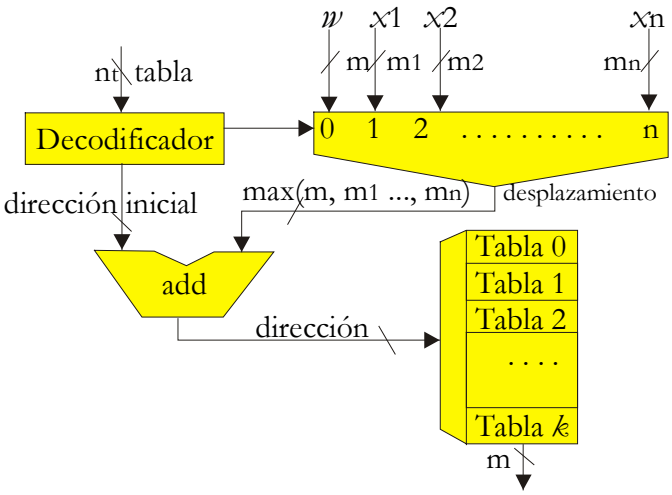


Fig. 4.3 - Diagrama de bloques para las Tablas

La ventaja de esta solución es su sencillez: la memoria es una macro-célula compilada; basta con definir su contenido; el sumador y el *multiplexor* tienen estructuras regulares y pueden ser definidos como bloques VHDL adaptables mediante el uso de parámetros; el decodificador es un circuito combinacional que podría ser sintetizado por un programa de síntesis lógica (caso de una FPGA) o integrado en una ROM compilada (caso de un ASIC).

El inconveniente de la solución propuesta es que, a veces, la memoria es muy grande. Una solución alternativa consiste en sintetizar un circuito de cálculo de las funciones de pertenencia y de decodificación en lugar de almacenar sus valores en una memoria [Aco96].

4.5 - FUNCIONES DE PERTENENCIA

Muchas veces las funciones de pertenencia tienen un gran volumen de información para ser almacenadas en una memoria ROM y además poseen una forma regular, suelen tener una forma trapezoidal (Fig. 4.4). En lugar de almacenar los valores de $A_i(x)$ en una memoria, se pueden calcular. Bastaría con almacenar los valores de *a, b, c, d* así como los coeficientes angulares k_1 y k_2 que corresponden a los intervalos *[a,b]* y *[c,d]*.

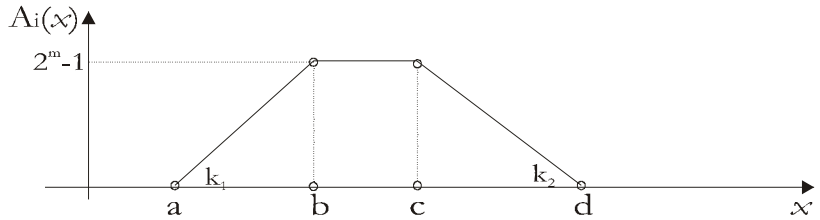


Fig. 4.4-Función de pertenecia por trapecios

Por otra parte, si se utiliza un método simplificado de decodificación (para las funciones de pertenencia de salida) en el cual cada B_j tiene un valor constante (*singleton method*; [Kos92]), entonces, en lugar de almacenar los valores de $C_j(w) = w \bullet B_j$ (donde $\bullet$ representa la operación de multiplicación) dichos valores pueden ser calculados. En resumen, el circuito *tablas* tiene que ejecutar el siguiente cálculo:

IF *función_de_pertenencia_i* **THEN**
  IF $x \le a_i$ **or** $x \ge d_i$ **THEN** $A_i = 0$;
  ELSIF $a_i \le x \le b_i$ **THEN** $A_i = (x - a_i) \bullet k_{i1}$;
  ELSIF $c_i \le x \le d_i$ **THEN** $A_i = (d_i - x) \bullet k_{i2}$;
  ELSE $A_i = 2^m - 1$;
  END IF;
ELSIF *función_de_decodificación_j* **THEN** $C_j = w \bullet B_j$;
END IF;

El esquema teórico de cálculo correspondiente se muestra en la Fig. 4.5. El decodificador selecciona $\boldsymbol{w}$ o una de las entradas $\boldsymbol{x}_i$. Una ROM almacena el valor de los parámetros $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{d}$, $\boldsymbol{k}_1$ (para las funciones de pertenencia), B (para las funciones de salida) y $\boldsymbol{k}_2$. Para las funciones de salida el decodificador selecciona $\boldsymbol{x} = \boldsymbol{w}$ y la memoria contiene los valores siguientes (de acuerdo a la Fig. 4.4):

$$\boldsymbol{a} = \boldsymbol{d} = 0, \quad \boldsymbol{b} = \boldsymbol{c} = 2^m\text{-}1.$$

Por tanto $\boldsymbol{a} \le \boldsymbol{x} \le \boldsymbol{b}$ con lo cual el valor calculado por el multiplicador es

$$(\boldsymbol{x} - \boldsymbol{a}) \bullet \boldsymbol{K}_1 / \boldsymbol{B} = \boldsymbol{w} \bullet \boldsymbol{B}.$$

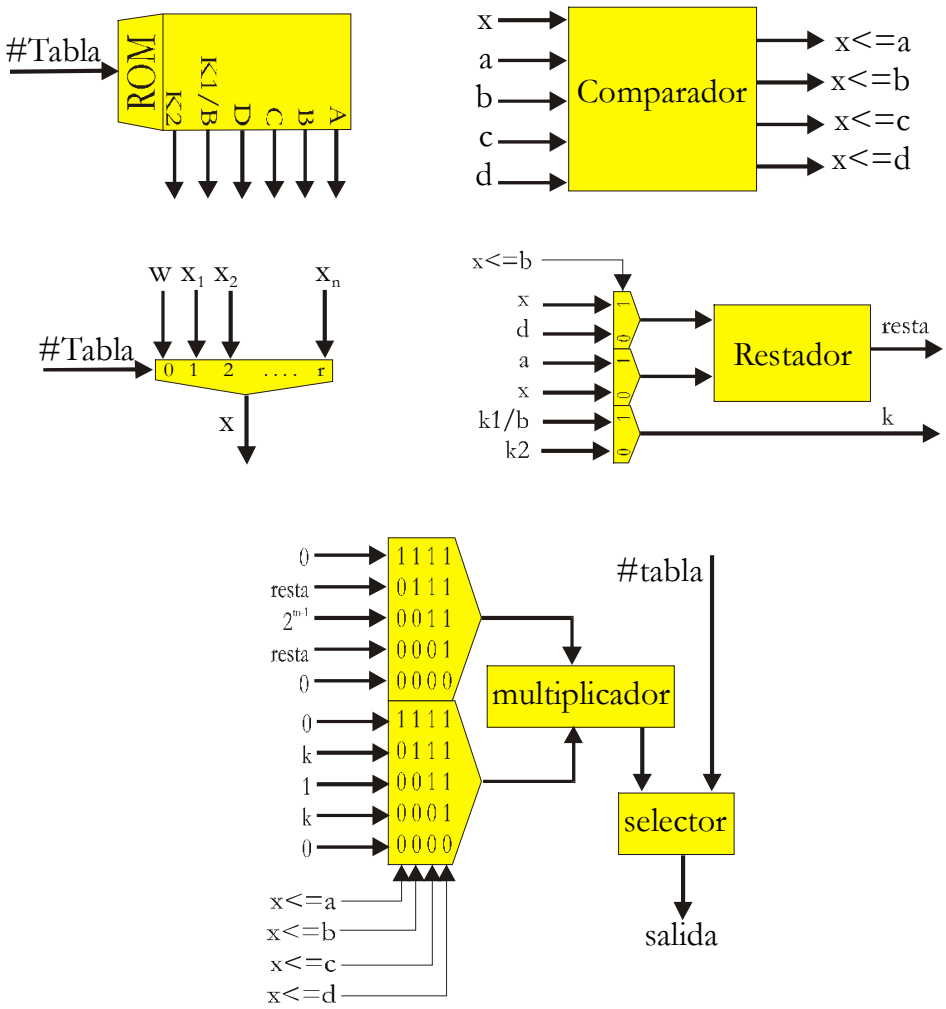


Fig. 4.5 – Esquema de cálculo teórico de las funciones de pertenencia

El selector de salida sirve de interfaz entre la salida del multiplicador y el banco de registros. En el caso de las funciones de pertenencia los operandos de la multiplicación son números reales expresados con cierta precisión y el resultado final (salida del circuito) tiene que ser un entero comprendido entre 0 y 2^m-1; el circuito de interfaz selecciona los ***m*** bits del resultado que corresponden a la parte entera.

Para las funciones de decodificación los operandos de la multiplicación son ***w*** (entero comprendido entre 0 y 2^m-1) y ***B*** (número real expresado con cierta precisión); el resultado es un número real expresado con (posiblemente) precisión múltiple; el circuito de interfaz permite seleccionar sucesivamente la primera palabra, la segunda palabra, etc. del resultado; ello significa que si ***w•B*** se representa con ***k*** palabras, a la función de decodificación ***B*** corresponden ***k***

números de tablas. Todas las filas de la ROM que corresponden a las tablas de una misma función de decodificación B tienen el mismo contenido.

La ROM y el multiplicador son macro células compiladas. Basta con definir el contenido de la ROM y el tamaño del multiplicador. El circuito de comparación, el de cálculo de la resta, los multiplexores y el selector (formado por un conjunto de multiplexores) tienen estructuras regulares y pueden ser definidos como bloques VHDL adaptables mediante el uso de parámetros. El decodificador es un circuito combinacional que puede ser sintetizado por un programa de síntesis lógica o integrado en otra ROM compilada.

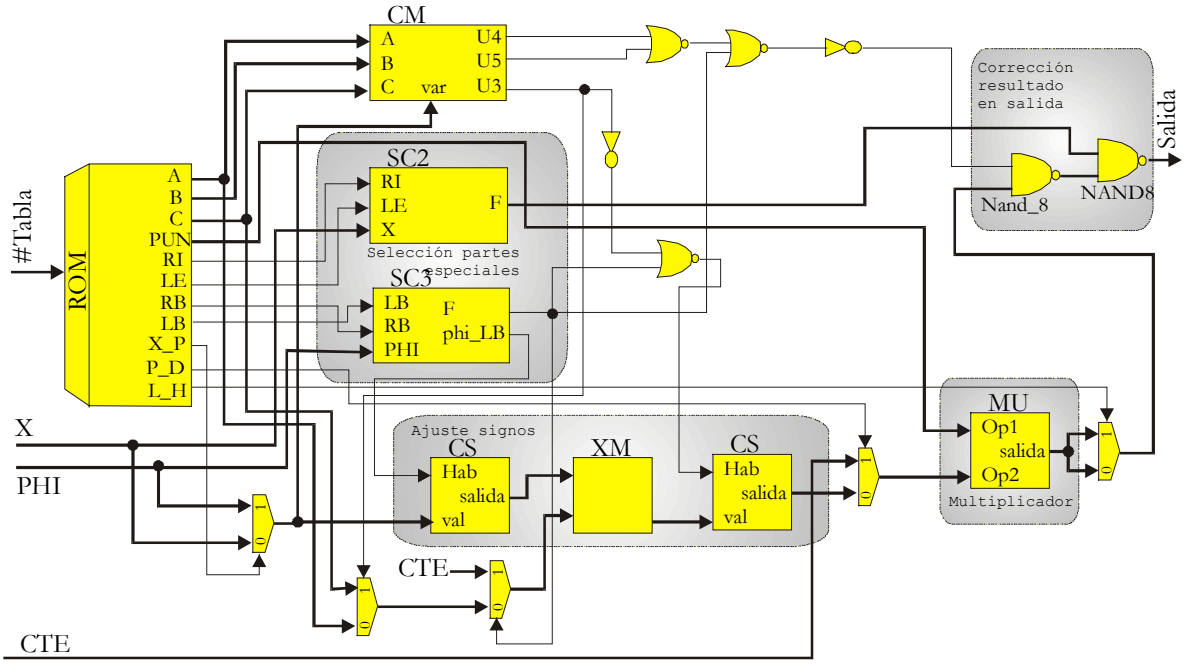


Fig. 4.6 – Diagrama del circuito de cálculo de funciones de pertenencia

En la Fig. 4.6 se muestra un diagrama completo del circuito que materializa el cálculo de las funciones de pertenencia. En este diagrama, además de los multiplexores de control de la ruta de datos y algunas puertas de lógica adicional, se pueden distinguir 6 bloques cuyo funcionamiento se describe a continuación.

4.5.1 - ROM

La ROM es accedida por ***#TABLA*** y contiene los datos para el control de la ruta de datos y los parámetros para el cálculo. Así, ***A***, ***B***, ***C*** es la posición de la

abscisa de cada variable, donde ***A*** es el primer punto de la figura, ***B*** es el segundo y ***C*** el tercer punto. ***PUN*** representa el ángulo o pendiente del segmento, así es posible calcular cualquier punto de la abscisa, teniendo el punto de origen del segmento y luego el ángulo del mismo. ***RI***, ***LE*** informan si se han activado la primera o última función de pertenencia de la variable ***x***. ***RB***, ***LB*** informan si se han activado la primera o última función de pertenencia de la variable ***phi***. ***X_P*** determina si se selecciona la entrada ***x*** o ***phi***. ***P_D*** determina el funcionamiento para la etapa de fusificación o defusificación. ***L_H*** determina que byte sale por los 8 bits de salida del circuito, la parte alta o baja.

4.5.2 - CM

El bloque ***CM*** es el encargado de la detección de la parte activa de la función, de acuerdo al valor actual de la variable de entrada ***VAR***. El valor de ***U3***, ***U4*** y ***U5*** es determinado dependiendo de si ***VAR*** es menor o igual que ***A***, menor igual que ***B*** o menor igual que ***C***, respectivamente.

4.5.3 – SELECCIÓN DE PARTES ESPECIALES

Es el módulo encargado de la "*selección de las partes especiales*" de las funciones de pertenencia, destinada a tratar los casos especiales de la variable ***x*** y ***phi***. Así, SC2 trata el caso particular de los segmentos horizontales de la variable ***x***; sus entradas ***RI***, ***LE*** y ***x*** son utilizados para determinar ***F***, vector cero o uno, de acuerdo a que ***x*** sea menor o igual que –380 o que ***x*** sea mayor o igual de +380. Por otra parte, SC3 trata el caso especial de los segmentos que comienzan con ángulo negativo y terminan con ángulo positivo (dado la continuidad de los ángulos, 360° = 0°) en las abscisas de ***phi***. ***LB***, ***RB*** y ***phi*** son utilizados para determinar ***F*** y ***PHI_LB***, señales utilizadas para realizar los ajustes de signo necesarios.

4.5.4 – AJUSTE DE SIGNOS

El módulo de "*ajuste de signos*" se compone de dos módulos: CS y XM. El módulo CS realiza el cambio de signo del valor ***VAL*** cuando la señal de control

HAB lo especifica. El módulo XM realiza el cálculo de la operación resta entre sus operandos.

4.5.5 - MU

El módulo de multiplicación, MU, es utilizado tanto para el cálculo de las funciones de pertenencia como de las funciones de defusificación por *singleton*. La señal ***SALIDA*** es el resultado de ***OP1*** multiplicado por ***OP2***.

4.5.6 – CORRECCIÓN DE RESULTADOS DE SALIDA

El módulo de "*corrección de resultados de salida*" trata el caso en que los resultados estén fuera del rango que corresponde o en el caso de las partes especiales de las funciones de pertenencia. Es el encargado de validar los resultados, permitiendo la salida del resultado del cálculo o reemplazándolo por el vector cero o uno, de acuerdo a lo que determinen las señales ***U3***, ***U4***, ***U5*** y ***F***.

4.6 - BANCO DE REGISTROS

El banco de registros consta principalmente de una memoria de doble puerto (DPRAM) compilada y de algunos circuitos adicionales (Fig. 4.7). El puerto *A* es de sólo lectura; la dirección correspondiente (*j*) se fija con la señal *ph2*. El puerto *B* es de lectura y escritura; las direcciones correspondientes se fijan con *ph0*; la dirección de lectura (*k*) se selecciona cuando *ph1* = 0 y la dirección de escritura (*i*) cuando *ph1* = 1. Se necesita un latch adicional que almacene *R*(*k*) cuando la dirección del puerto *B* pasa de *k* (lectura) a *i* (escritura), es decir, cuando *ph1* pasa de 0 a 1. La escritura se produce con el flanco de bajada de *ph1*.

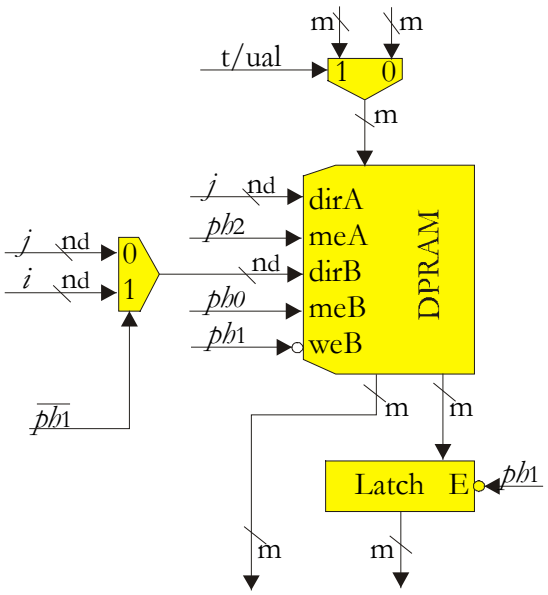


Fig. 4.7 - Banco de Registros

4.7 - UNIDAD ARITMÉTICO-LÓGICA

Para la ejecución de un algoritmo de control difuso se necesitan las siguientes operaciones: *max*(*x*, *y*), *min*(*x*, *y*), $x + y$, $x - y$, $2 \bullet x + y$, $2 \bullet x - y$. Las dos primeras se utilizan para calcular los pesos asociados a las funciones de decodificación. La suma y la resta sirven para calcular los numeradores y denominadores cuyos cocientes son las funciones de salida; en el caso de una función única:

$$N = w_1 \bullet B_1(w_1) + w_2 \bullet B_2(w_2) + \ldots ;$$

$$D = w_1 + w_2 + \ldots .$$

Para el cálculo de N puede que no sea necesario disponer de la multiplicación: en lugar de almacenar en tablas las funciones de decodificación $B_1(w)$, $B_2(w)$, ..., se almacenan directamente, con doble precisión, los productos $C_1(w) = w \bullet B_1(w)$, $C_2(w) = w \bullet B_2(w)$, ...; por tanto, el cálculo de N se reduce a la lectura de las tablas C_1, C_2, ... , y al cálculo de una suma. Además, si existen ciertas simetrías entre las funciones de decodificación puede ser útil disponer de la resta [Aco96]. En el caso de los diseños de las Fig. 4.5 y 4.6 se utiliza un único circuito de

multiplicación tanto para calcular las funciones de pertenencia como para el cálculo de *N*.

Las dos últimas operaciones sirven para calcular un cociente. Para ello se utiliza el algoritmo de división sin restauración [Aco96, Dav83], que permite calcular el cociente N/D de dos enteros N y D siempre y cuando se cumpla la relación $-D \leq N < D$. En el caso del algoritmo de control difuso se sabe que N/D es el valor de la función de salida. Si el rango de la misma es $[a,b]$, se sabe de antemano que

$$a \bullet D \leq N \leq b \bullet D$$

(donde D es siempre positivo). Para poder utilizar el algoritmo de división puede ser necesario sustituir D por $D^* = 2^p.D$ de tal forma que $a/2^p \geq -1$ y $b/2^p < 1$. El algoritmo genera el cociente $N/D^* \cong \overline{q}_0 , q_1 q_2 \ldots q_{l-1}$. El valor de f es:

$$f \cong \overline{q}_0 q_1 q_2 \ldots q_p , q_{p+1} q_{p+2} \ldots q_{l-1} .$$

Prácticamente, el cálculo se realiza como sigue:

1) Alinear los operandos, es decir, sustituir D por $D^* = 2^{k.m} \bullet D$ siendo m el número de bits de las variables (palabras) internas. Para poder realizar dicha operación se añade una operación a la unidad aritmético-lógica: generar la palabra de m bits *cero = 00 ... 0.* La multiplicación por $2^{k.m}$ es equivalente a la concatenación de D con k palabras *cero*.

2) Calcular $r := x +$ *cero*.

3) Ejecutar l veces:

   SI (el signo del resultado de la última operación es negativo)

   ENTONCES $q := 0$ y $r := 2 \bullet r + y$;

SI NO, $q := 1$ y $r := 2 \bullet r - y$; entrar q en el registro de desplazamiento de salida ($Q := 2 \bullet Q + q \bullet s^{1-n}$).

Func(3:1)	*s/r*	*cent*	*z*
00-	-	-	0
010	1	1	*mín(x,y)*
100	1	1	*máx(x,y)*
110	0	-	$x+y+cent$
110	1	-	$x+\bar{y}+cent$
111	0	-	$2 \bullet x+xent+y+cent$
111	1	-	$2 \bullet x+xent+\bar{y}+cent$

Tabla 4.1 - Núcleo de la UAL

El núcleo de la unidad aritmético-lógica es un circuito combinacional cuyo funcionamiento se describe en la Tabla 4.1. Los *m* módulos (*slices*) idénticos (Fig. 4.8) que forman la unidad completa se conectan como lo indica la Fig. 4.9. Las salidas *xsal* y *csal* del circuito completo son las salidas *xsgu* y *csigu* del último módulo (bit *m-1).* Las entradas *xent* y *cent* del circuito completo son las entradas *xant* y *cant* del primer módulo (bit *0*). Obsérvese que para calcular

- $\min(x, y)$,
- $\max(x, y)$,
- $x - y$ ($\equiv x + \bar{y} + 1$), y
- $2 \bullet x - y$ ($\equiv 2 \bullet x + \bar{y} + 1$)

es necesario que las entradas *xent* y *s/r* estén a *1*. Mientras que para calcular

- $x + y$, y
- $2 \bullet x + y$

estas mismas entradas tienen que estar a *0*. Además, para el cálculo de

- $2 \bullet x + y$, y

- $2 \bullet x - y$

la entrada *xent* tiene que ser igual a *0*.

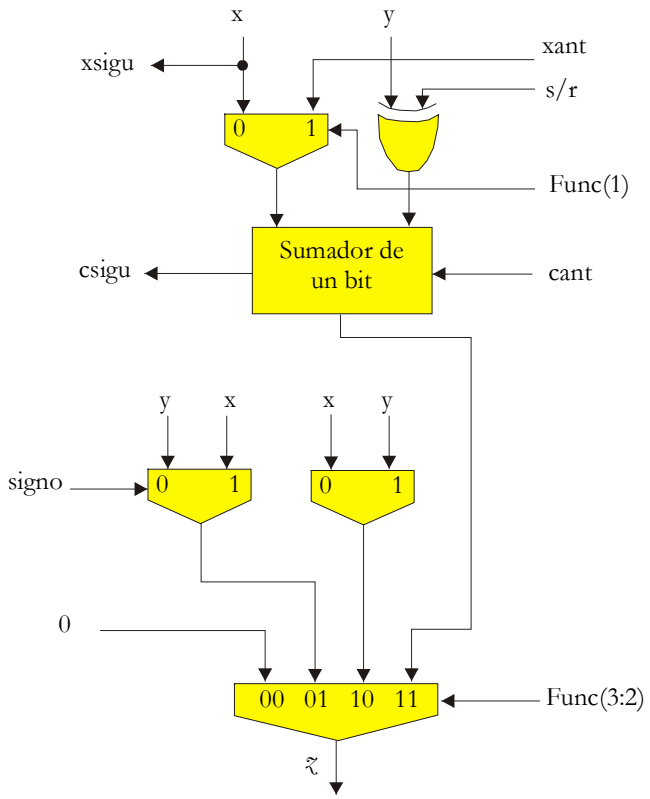


Fig. 4.8 - ALU de un bit

Para poder realizar operaciones de precisión múltiple, así como poder elegir entre suma y resta en función del resultado de la operación anterior, se necesitan tres biestables que almacenan *xent, cent* y *s/r*. En la Tabla 4.2 se define la manera en que se actualizan los tres biestables en función del valor de las variables de control *ff(1:0)* y *sh*.

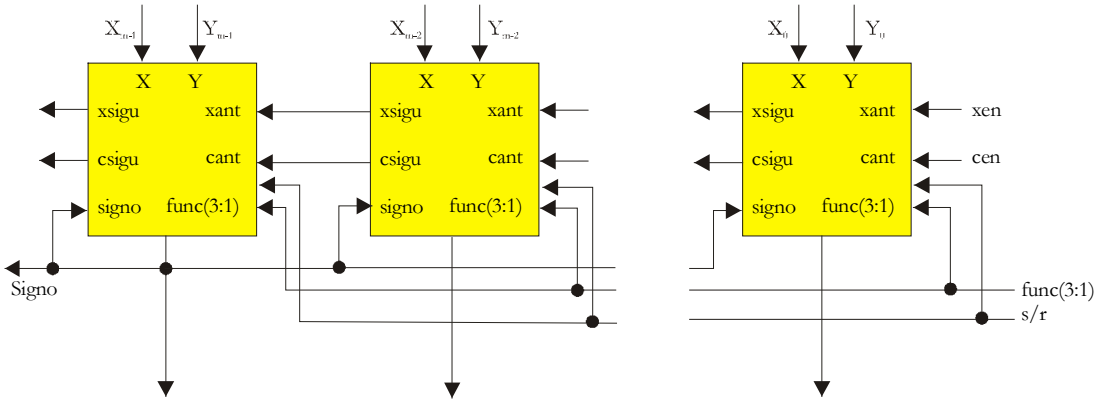


Fig. 4.9 - ALU de 8 bits construida con la ALU de un bit

ff(1:0)	sh	xent	cent	s/r
00	0	*Xent*	*cent*	*s/r*
01	0	0	0	0
10	0	0	1	1
11	0	*Xsal*	*csal*	*s/r*
10	1	0	*/signo*	*/signo*

Tabla 4.2-Comandos para operación de múltiples operandos

La primera línea corresponde a microinstrucciones que no modifican el estado de los biestables. La segunda a la generación del estado inicial para realizar una suma. La tercera a la generación del estado inicial para realizar una resta. La cuarta corresponde a la generación del estado inicial para la segunda, tercera, etc., parte de operaciones de precisión múltiple. La quinta a la generación del estado inicial para realizar las operaciones $2 \bullet x+y$ o $2 \bullet x-y$ en función del signo de la operación anterior. El circuito que permite el manejo de los *flip-flop* se muestra esquematizado en la Fig. 4.10.

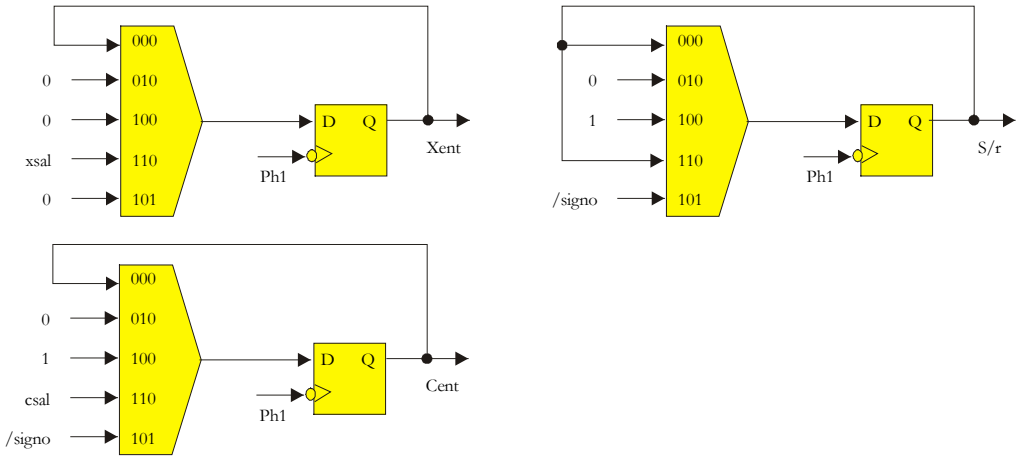


Fig. 4.10 - Circuito para el manejo de los *flip-flops*

Véase un ejemplo en la Tabla 4.3. El siguiente microprograma controla la ejecución de la división de *N* por *D*, suponiendo que *N* consta de tres palabras almacenadas en *R(3), R(2)* y *R(1)*, y *D* de dos palabras almacenadas en *R(5)* y *R(4)*. Para alinear los operandos es necesario multiplicar *D* por 2^m. Por tanto el cálculo que se lleva a cabo es *R(3) R(2) R(1) / R(5) R(4) R(0)*:

Nº	i	j	k	tabla	t/ual	ff	Sh	Func	in	out
0:	0	-	-	-	0	01	0	00-	0	0...00
1:	3	3	0	-	0	10	1	110	0	0...00
2:	1	1	0	-	0	11	0	111	0	0...00
3:	2	2	4	-	0	11	0	111	0	0...00
4:	3	3	5	-	0	10	1	111	0	0...00
5:	1	1	0	-	0	11	0	111	0	0...00
6:	2	2	4	-	0	11	0	111	0	0...00
7:	3	3	5	-	0	10	1	111	0	0...00
8:	1	1	0	-	0	11	0	111	0	0...00
9:	2	2	4	-	0	11	0	111	0	0...00
10:	3	3	5	-	0	10	1	111	0	0...00
11:	1	1	0	-	0	11	0	111	0	0...00
12:	2	2	4	-	0	11	0	111	0	0...00
13:	3	3	5	-	0	10	1	111	0	0...00
14:	0	-	-	-	0	00	0	00-	0	0...01

Tabla 4.3 - Ejemplo: microprograma de división

Al término de la ejecución de este microprograma se habrán calculado cinco bits del cociente, se los habrá introducido en serie en el registro de desplazamiento de salida y se habrá transferido el resultado al puerto de salida nº 1. La microinstrucción nº 0 sirve para poner *cent* y *s/r* a *0*. La microinstrucción nº 1 para sumar *R(3)* con *0* y generar así el bit de signo del cociente; dicho bit se introduce en el registro de desplazamiento de salida (*sh = 1*) y los biestables *cent* y *s/r* toman el valor de */signo*. La microinstrucción nº 2 realiza la transferencia

$$R(1) <= 2 \bullet R(1) \pm R(0)$$

manteniendo *s/r* en su valor anterior y transfiriendo *xsal* y *csal* a los biestables *xent* y *cent*. La microinstrucción nº 3 realiza la transferencia

$$R(2) <= 2 \bullet R(2) \pm R(4)$$

controlando los *flip-flops* de la misma manera que la anterior. La microinstrucción nº 4 realiza la transferencia

$$R(3) <= 2 \bullet R(3) \pm R(5)$$

y actualiza *cent* y *s/r* en función del resultado; al mismo tiempo se añade un bit del cociente al registro de desplazamiento de salida. Con las

microinstrucciones Nº 5 a 7, 8 a 10 y 11 a 13 se generan otros tres bits del cociente. La última microinstrucción transfiere el contenido del registro de desplazamiento al puerto de salida seleccionado.

4.8 - PUERTOS DE SALIDA

Según el algoritmo descrito en la sección 4.7 el valor de una función particular *f* se obtiene en la forma

$$f = \bar{q}_0\ q_1 q_2 \ldots q_p\ ,\ q_{p+1} q_{p+2} \ldots q_{l-1}$$

siendo *l* el número de bits generados por el microprograma. Por otra parte se supone que en las especificaciones del controlador se define el formato deseado de la función *f* ; por ejemplo:

$$f = z_t z_{t-1} \ldots z_0\ ,\ z_{-1} z_{-2} \ldots z_{-(s-t-1)}\ .$$

Por tanto se tienen que cumplir las siguientes desigualdades:

$$p \geq t\ ,\ \text{y}$$

$$l - p \geq s - t.$$

La primera se cumple eligiendo para *p* el primer múltiplo

$$k \bullet m \quad \text{de} \quad m \quad \text{tal que} \quad p = k \bullet m \geq t.$$

Una vez elegido el valor de *p*, el valor de *l* se deduce de la segunda desigualdad:

$$l = p + s - t.$$

En conclusión, el número de bits que tiene que generar el microprograma, para la función de salida f_i, es igual a

$$l_i = p_i + s_i - t_i.$$

El número de bits del registro de desplazamiento es igual a $máx(s_1, s_2, ..., s_r)$.

Observación: en caso de que $p_i = t_i$, el número l_i de bits generados por el microprograma es igual al número s_i de bits de la representación deseada. En este caso el primer bit (el de signo) tiene que ser invertido ($z_t = \overline{q_0}$). Al contrario, si $p_i > t_i$ el número de bits generados por el microprograma es mayor que el número de bits de la representación deseada. Ello significa que se tienen que descartar los $p_i - t_i$ primeros bits, en particular el bit q_0,.

4.9 - INTERFAZ CON CIRCUITOS EXTERNOS

Tal como se ve en la Fig. 4.11 el controlador tiene dos modos de funcionamiento: **modo autónomo** (*modo = 0*) y **modo periférico** (*modo = 1*). En ambos casos la señal *inicio* pone a *0* el contador de microinstrucciones y el generador de fases. Cuando *inicio* pasa de *1* a *0* el microprograma empieza a ejecutarse y sigue ejecutándose de forma cíclica hasta que otra vez se active la señal *inicio*. La diferencia entre los dos modos radica en la forma en que se realizan las muestras de los valores de entrada:

- En modo autónomo las entradas se registran cada vez que el microprograma genera la señal *in*;
- En modo periférico se registran cuando la señal *inicio* está activa. En este segundo caso se habilita también una señal de salida *fin* que pasa de *0* a *1* cuando se hayan transferido los resultados a todos los puertos de salida; luego el microprograma sigue ejecutándose pero con los mismos valores de entrada con lo cual generará siempre los mismos valores de salida.

Observación. Para que el controlador funcione correctamente en modo periférico se tienen que sondear las entradas (*in = 1*) al principio del microprograma (por ejemplo en la primera microinstrucción) y transferir el último resultado a su puerto de salida (*out = número del puerto*) al final. No se puede

utilizar la primera microinstrucción para, a la vez, transferir el último resultado a su puerto y sondear nuevas entradas.

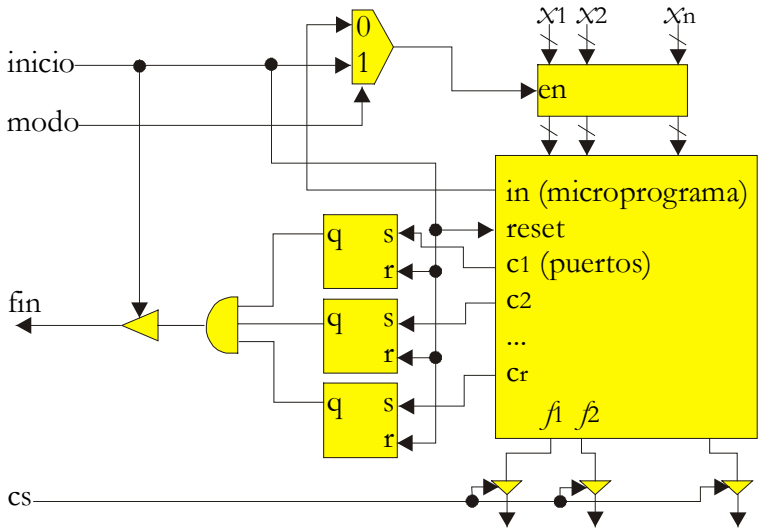


Fig. 4.11 - Interfaz con circuitos externos

4.10 - EJEMPLO DE CONTROLADOR

Se utiliza esta arquitectura adaptable mediante el uso de parámetros en el diseño del circuito para el sistema de guiado automático del vehículo descrito oportunamente en el capítulo 2. Recordemos que la versión implementada contiene 12 funciones de pertenencia, 7 funciones de decodificación y 35 reglas de inferencia. El microprograma generado con las herramientas descritas en el capítulo 5 contiene 125 microinstrucciones.

Las funciones de pertenencia y de decodificación son calculadas y no almacenadas en una ROM [Aco96]. Las macro-células compiladas son las siguientes:

- Un multiplicador de 14 bits por 10 bits.
- Una ROM de 125 palabras de 28 bits (microprograma).
- Una ROM de 31 palabras de 51 bits (vértices de los triángulos y trapecios, y centros de gravedad).

- Una RAM de doble puerto de 32 palabras de 8 bits.

Los bloques sintetizados son:

- Un contador de 8 bits.
- Una unidad aritmético-lógica de 8 bits.
- Un circuito que realiza las restas con 10 bits de precisión.
- Cuatro circuitos que comparan magnitudes de 10 bits .
- Un puerto de salida de 6 bits.

La compilación de las macro-células y el trazado físico se llevaron a cabo en el Centro Nacional de Microelectrónica, España.

4.11 - SÍNTESIS DE LOS BLOQUES DEL CIRCUITO

Si bien existen sintetizadores automáticos de circuitos, los que transforman una especificación VHDL en circuitos lógicos con compuertas estándar, no siempre dan la síntesis óptima. Los métodos de síntesis automática de circuitos son utilizados preferentemente en aquellos casos en los que el tamaño del problema hace muy tediosa la síntesis manual.

Dada la arquitectura propuesta, la síntesis es trivial ya que todos los bloques no son más que estructuras iterativas de compuertas elementales. Las memorias ROMs y RAMs cuentan con compiladores provistos por el fabricante de semiconductores y son incluidas en el diseño como macro-células.

Se utiliza la tecnología *ECPD10* de una micra de la empresa ES2 (European Silicon Structures), por tanto se utiliza la librería de células (biblioteca de compuertas) de esta empresa. Se puede ver los detalles de las librerías en los manuales de especificación de la empresa [Ess92a, Ess92b, Ess92c].

4.11.1 - COMPILACIÓN DE MACRO-CÉLULAS

La compilación de las macro-células se efectúa en el CNM puesto que se necesitan licencias del software para estaciones de trabajo, no disponibles localmente en el INCA (Grupo de *INvestigación en Computación Aplicada*, Tandil). Las macro-células compiladas fueron:

- Multiplicador de 14 por 10 bits. Utilizado en el bloque tablas para el cálculo de las funciones de pertenencia.
- ROM de 31 palabras por 51 bits. Utilizado en el bloque tablas para el cálculo de las funciones de pertenencia.
- ROM de 125 palabras de 28 bits. La ROM del microprograma.
- DPRAM de 32 palabras de 8 bits. El banco de registros.

4.11.2 - BLOQUES REGULARES SINTETIZADOS

Los bloques regulares sintetizados a partir de su definición VHDL son:

- Un contador de 8 bits. Contador del microprograma.
- Una ALU de 8 bits.
- Un circuito que resta valores de 10 bits, utilizado en el módulo tablas.
- Cuatro circuitos que comparan magnitudes de 10 bits.
- Un registro de desplazamiento de 9 bits, para almacenar el resultado de la división.
- Un *latch* de 9 bits (entrada *phi*).
- Un *latch* de 10 bits (entrada *x*).
- Un *latch* de 6 bits (salida *tetha*).

4.11.3 - LÓGICA ADICIONAL

Existen naturalmente diversos selectores, multiplexores, *pads* de entrada salida, *pads* de alimentación, etc. que completan el circuito. La lógica adicional está por debajo de las 100 puertas.

4.12 - RESUMEN

En este capítulo se presenta una arquitectura de controladores difusos, basada en una ruta de datos controlada por un microprograma, que usa una única unidad de proceso. Se analiza: el conjunto de señales y su sincronización, la estructura de las tablas que implementan las funciones de pertenencia y de decodificación, los bancos de registros, y por último, la unidad aritmético – lógica. La ALU es analizada en la ejecución de varios ejemplos de microcódigo, siendo sus operaciones básicas:

- $min\ (x, y)$
- $max\ (x, y)$
- $x + y + cent$
- $x + \text{not}(y) + cent$
- $2 \bullet x + xent + y + cent$
- $2 \bullet x + xent + \text{not}(y) + cent$

El circuito consta de una ruta de datos, adaptable mediante el uso de parámetros, controlada por un microprograma sin saltos. Se describe principalmente su estructura y el funcionamiento de la ruta de datos. Con el diseño propuesto se fabricó un ASIC usando tecnología de ES2 de una micra, totalizando una equivalencia a 18858 transistores.

5 - HERRAMIENTAS DE LA ARQUITECTURA PASIVA

5.1 - INTRODUCCIÓN

Como ambiente de desarrollo de controladores difusos es deseable contar con herramientas que permitan configurar todo el sistema, tanto el software como el hardware, para de tal forma obtener asistencia durante todo el proceso de producción.

Este capítulo describe el estudio y materialización de herramientas para el desarrollo de controladores basados en lógica difusa, cuya interfaz se describe en el Apéndice A. Entre ellas, se plantea:

- Compilador de reglas y de funciones de pertenencia y decodificación. Genera un esquema de cálculo o un programa de simulación del controlador, y los valores para las tablas de las funciones. Por otra parte es útil contar con un sistema que permita representar gráficamente las funciones de pertenencia y decodificación, para verificar sus parámetros.

- Transformador de esquemas de cálculo. Realiza la optimización reticular.

- Generador de programa. Realiza la asignación de registros a las variables del esquema de cálculo, transformándolo en un programa.

- Generador de programa para la simulación del controlador, o microprograma de control del controlador hardware.

- Generador de valores de prueba y analizador de las simulaciones para la verificación de la materialización del sistema.

5.2 - ALGORITMO DE FL MATERIALIZADO

En esta sección se describe un algoritmo para el cálculo de m funciones $f_0, f_1, ..., f_{m-1}$ de n variables $x_0, x_1, ..., x_{n-1}$. El valor de x_i pertenece a un conjunto S_i, y el valor de f_k al conjunto de números reales R. A cada variable x_i corresponden p_i funciones de pertenencia:

$$A_{ij}: S_i \rightarrow [0,1], \quad 0 \le j \le p_i\text{-}1;$$

a cada función f_k corresponde q_k funciones de decodificación:

$$B_{kl}: [0,1] \rightarrow R, \quad 0 \le l \le q_k\text{-}1.$$

El cálculo de f_k se basa en sentencias o reglas de inferencia, tales como:

"**IF** x_0 **is** $A_{0\,j0}$ **AND** x_1 **is** $A_{1\,j1}$ **AND** ... **AND** x_{n-1} **is** $A_{n-1\,jn-1}$ **THEN** f_k **is** B_{kl}".

Se define un conjunto r de coeficientes:

$$r(j_0, j_1, ..., j_{n-1}, k, l) = 1$$

si las reglas son aplicadas; si no

$$r(j_0, j_1, ..., j_{n-1}, k, l) = 0.$$

El algoritmo de cálculo de las funciones f_k, que materializa la propuesta de la Fig. 4.1 (descrito más detalladamente en 5.3.3: esquema de cálculo) es:

$\forall i$:0 ≤ i ≤ n-1, $\forall j$:0 ≤ j ≤ p_i-1, calcular:
$y_{ij} = A_{ij}\ (\ x_i\ )$
$\forall k$:0 ≤ k ≤ m-1,
$\forall l$:0 ≤ l ≤ q_k-1, calcular:
$w_{kl} = \vee r\ (j_0, j_1, \ldots, j_{n-1}, k, l)\ \ y_{0\ j0} \wedge y_{1\ j1} \wedge \ldots \wedge y_{n-1\ jn-1}, \forall\ j_0, j_1, \ldots, j_{n-1}$
$v_{kl} = B_{kl}\ (\ w_{kl}\ )$
donde el símbolo "∧" representa el operador mínimo,
y el símbolo "∨" el máximo.
$N_k = v_{k\ 0} + v_{k\ 1} + \ldots + v_{k\ qk-1}$
$D_k = w_{k\ 0} + w_{k\ 1} + \ldots + w_{k\ qk-1}$
$f_k = N_k\ /\ D_k$

5.3 - DESCRIPCIÓN DEL SISTEMA DE GENERACIÓN

El sistema de generación intenta cubrir la brecha entre la especificación del problema (basado en reglas) y su realización por un circuito específico (controlado por un microprograma). Se define un lenguaje de especificación usando reglas y conjuntos difusos para definir el problema. Del archivo de especificación, el sistema produce un primer esquema de cálculo. Mediante la aplicación de varias transformaciones (minimizar las operaciones aritméticas y reticulares, asignar los registros de forma óptima, realizar las operaciones de múltiples palabras en simple palabra, etc.), el sistema produce el microprograma que dirige el hardware del controlador. El ciclo de desarrollo propuesto [Aco97, Des97] para un controlador difuso se muestra en la Fig. 5.1.

1) Definición de un archivo de texto con la especificación del controlador, usando reglas y definiendo las funciones de pertenencia y decodificación. Se usan dos herramientas para verificar la validez de la especificación: un sistema que muestre los gráficos de las funciones de pertenencia y decodificación, y un generador del ejecutable que simule el controlador.

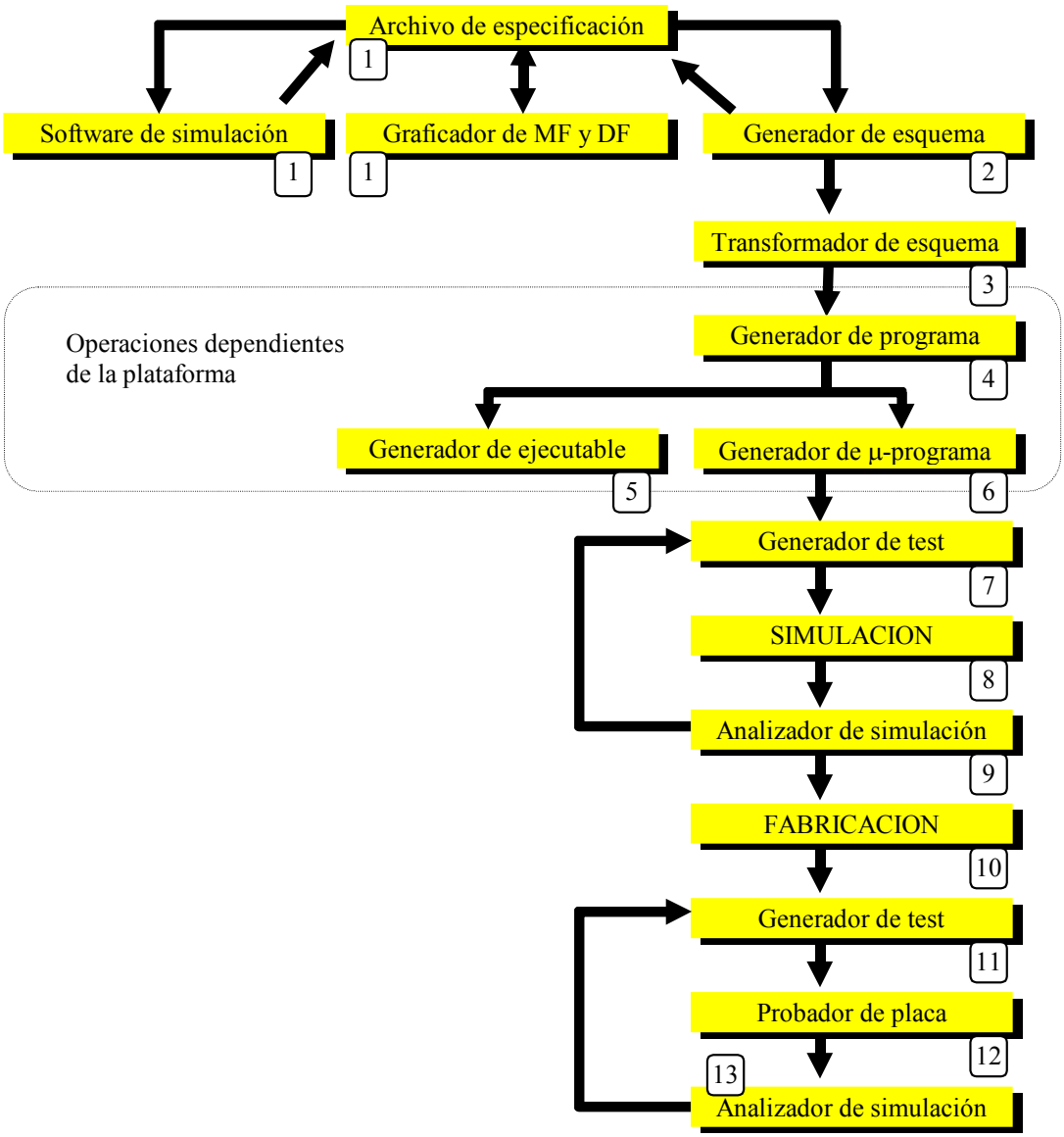


Fig. 5.1-Ciclo de diseño de un FLS basado en la arquitectura pasiva

2) Generación del esquema de cálculo a partir de las reglas que definen el comportamiento del controlador.

3) Transformación del esquema de cálculo, por medio de la optimización reticular.

4) Generación del programa, realizando la asignación óptima de registros.

5) Generación del ejecutable para la simulación o ejecución en plataformas 80x86. Este ejecutable tiene minimizadas las operaciones reticulares y la cantidad de memoria que usa es mínima. También es posible realizar el código optimizado para obtener máxima velocidad de ejecución.

6) Generación del microprograma de acuerdo a la plataforma hardware sobre la cual se ejecutará el controlador (cantidad de ALUs, ancho de palabra, ...).

7) Generación de casos de prueba simulando la plataforma hardware.

8) Simulación de la plataforma hardware y su microprograma en un simulador.

9) Análisis de la simulación. Pueden usarse los ejecutables generados con anterioridad para verificar los resultados.

10) Fabricación de la plataforma hardware (FPGA y ASIC).

11) Generación de casos de prueba para verificar el chip.

12) Prueba del integrado en un banco de pruebas conectado a una PC donde el software hace ejecutar el controlador y recolecta los resultados.

13) Análisis de la ejecución del chip sobre el banco de pruebas: permite comparar resultados con las simulaciones hardware o software.

Se generan ejecutables durante las etapas: (1) código resultado de la traducción directa (sin optimizaciones) y (5) código con optimización reticular y que usa una cantidad mínima de registros. Ambos ejecutables pueden ser usados para simular por software el controlador de tal forma que se puedan comparar los

resultados de las simulaciones de las distintas etapas.

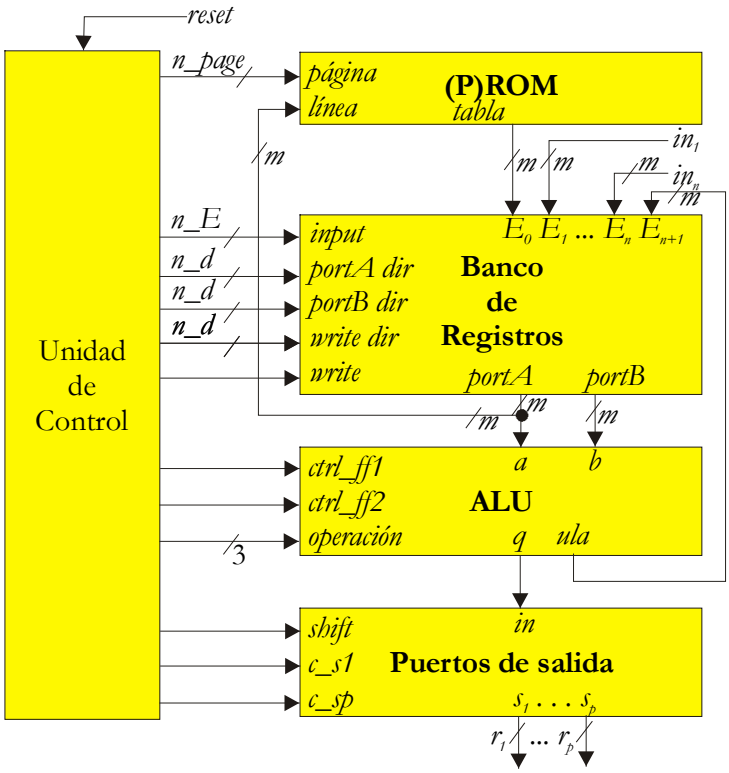


Fig. 5.2-Diagrama de bloques de la arquitectura pasiva

Durante las etapas de generación de programa (4) y generación de microprograma (6), se debe tener en cuenta la plataforma hardware (ver Fig. 5.2) sobre la que se ejecuta el proceso de control. Durante la etapa (4), se realiza la asignación de registros (cantidad variable por plataforma), y durante la etapa (6), se genera el microprograma que controla el flujo de los datos (cantidad de registros, cantidad de ALUs, rutas de datos, etc.).

5.3.1 - LENGUAJE DE ESPECIFICACIÓN

El lenguaje de definición de controladores difusos ha sido fuertemente influenciado, por un lado, por FIL (*Fuzzy Inference Language*) de *Aptronix* [Aptro], y por el otro, por la realización del primer prototipo, para la cual se utilizó el lenguaje *Prolog*. Así el lenguaje es una secuencia de "hechos" *Prolog* que definen: variables de entrada, funciones de pertenencia, variables de salida, funciones de decodificación, y reglas.

A las variables se les define un dominio de trabajo,

$$y = f(x),$$

donde $x \in$ [*Xmínimo*, *Xmáximo*] e $y \in$ [*Ymínimo*, *Ymáximo*], expresado como la cuádrupla: [*Xmínimo*, *Xmáximo*, *Ymínimo*, *Ymáximo*]. Las funciones de pertenencia y decodificación trabajan en el dominio definido para su variable, y por polígonos.

La sintaxis es la siguiente:

```
invar( "NomInVar",NroInVar, MinX, MinY, MaxX, MaxY  )
f_p(   "NombreFP",NroFP, "NomInVar", Puntos_Polígono )
outvar("NomOutVar",NroOutVar,MinX, MinY, MaxX, MaxY )
f_d(   "NombreFD",NroFD, "NomOutVar", Puntos_Polígono )
si([is("NomInVar","NombreFP"), ..], is("NomOutVar","NombreFD"))
```

Donde:

- **f_d( ):** definición de una función de decodificación.
- **f_p( ):** definición de una función de pertenencia.
- **invar( ):** definición de una variable de entrada.
- **is( ):** especificación de una condición.
- ***Puntos_Polígono*:** define los *n* puntos de las funciones de pertenencia o decodificación. Es una secuencia de puntos: *X*, *Y*, que describen los valores de la función sobre el eje *X* e *Y*, respectivamente. Concatenando puntos se forman los polígonos que describen la función de pertenencia o decodificación.

- ***MinX*, *MinY*, *MaxX*, *MaxY*:** definen el área sobre el cual trabajará una variable (de entrada o salida). Siendo *MinX* y *MinY* el vértice inferior-izquierdo y *MaxX* y *MaxY* el vértice superior-derecho de la función.

- ***NombreFP*:** nombre o alias de la función de pertenencia (variable lingüística).

- ***NombreFD*:** nombre o alias de la función de decodificación (variable lingüística).

- ***NomInVar*:** nombre o alias de la variable de entrada (variable lingüística).

- ***NomOutVar*:** nombre o alias de la variable de salida (variable lingüística).

- ***NroFD*:** número que identifica la función de decodificación.

- ***NroFP*:** número que identifica la función de pertenencia.

- ***NroInVar*:** número que identifica la variable de entrada.

- ***NroOutVar*:** número que identifica la variable de salida.

- **outvar( ):** definición de una variable de salida.

- **si( ):** definición de una regla de comportamiento del controlador.

A continuación y como ejemplo, se muestra el archivo de definición del problema planteado por el Dr. Bart Kosko en [Kos92], conocido en la literatura como "*Truck backer-upper control systems*" (ver Capítulo 3, sección 7 para mayores detalles). El controlador guía un camión para que estacione en un muelle de carga marcha atrás. Las variables de entrada son: X (posición en el eje X en el rango de -500 a 500 centímetros) y PHI (ángulo del camión con respecto a la

vertical en el rango -180 a 180 grados), mientras que la de salida es: THETA (el ángulo de las ruedas del camión en el rango de -30 a 30 grados). Se consideran 5 funciones de pertenencia para X (LE, LC, CE, RC, RI) y 7 para PHI (RB, RU, RV, VE, LV, LU, LB). En éste caso todas las funciones de pertenencia y de decodificación se representan por polígonos de 3 puntos. Nótese que todas las variables y sus funciones están representadas con 8 bits, y se muestran las 35 reglas que definen el problema.

```
invar("x",0,0,0,255,255)
invar("phi",1,0,0,255,255)
f_p("LE",0,"x",0,255,31,255,97,0)
f_p("LC",1,"x",76,0,101,255,127,0)
f_p("CE",2,"x",115,0,128,255,140,0)
f_p("RC",3,"x",129,0,154,255,180,0)
f_p("RI",4,"x",158,0,224,255,255,255)
f_p("RB",5,"phi",0,0,32,255,71,0)
f_p("RU",6,"phi",59,0,83,255,107,0)
f_p("RV",7,"phi",93,0,110,255,127,0)
f_p("VE",8,"phi",119,0,128,255,136,0)
f_p("LV",9,"phi",129,0,146,255,163,0)
f_p("LU",10,"phi",148,0,172,255,196,0)
f_p("LB",11,"phi",184,0,223,255,255,0)
outvar("teta",0,0,0,60,255)
f_d("NB",0,"teta",0,255,0,255,14,0)
f_d("NM",1,"teta",5,0,15,255,25,0)
f_d("NS",2,"teta",17,0,24,255,30,0)
f_d("ZE",3,"teta",25,0,30,255,35,0)
f_d("PS",4,"teta",30,0,36,255,43,0)
f_d("PM",5,"teta",35,0,45,255,55,0)
f_d("PB",6,"teta",46,0,60,255,60,255)
si([is("x","LE"),is("phi","RB")],is("teta","PS"))
si([is("x","LC"),is("phi","RB")],is("teta","PM"))
si([is("x","CE"),is("phi","RB")],is("teta","PM"))
si([is("x","RC"),is("phi","RB")],is("teta","PB"))
si([is("x","RI"),is("phi","RB")],is("teta","PB"))
si([is("x","LE"),is("phi","RU")],is("teta","NS"))
si([is("x","LC"),is("phi","RU")],is("teta","PS"))
si([is("x","CE"),is("phi","RU")],is("teta","PM"))
si([is("x","RC"),is("phi","RU")],is("teta","PB"))
si([is("x","RI"),is("phi","RU")],is("teta","PB"))
si([is("x","LE"),is("phi","RV")],is("teta","NM"))
si([is("x","LC"),is("phi","RV")],is("teta","NS"))
si([is("x","CE"),is("phi","RV")],is("teta","PS"))
si([is("x","RC"),is("phi","RV")],is("teta","PM"))
si([is("x","RI"),is("phi","RV")],is("teta","PB"))
si([is("x","LE"),is("phi","VE")],is("teta","NM"))
si([is("x","LC"),is("phi","VE")],is("teta","NM"))
si([is("x","CE"),is("phi","VE")],is("teta","ZE"))
si([is("x","RC"),is("phi","VE")],is("teta","PM"))
si([is("x","RI"),is("phi","VE")],is("teta","PM"))
si([is("x","LE"),is("phi","LV")],is("teta","NB"))
si([is("x","LC"),is("phi","LV")],is("teta","NM"))
si([is("x","CE"),is("phi","LV")],is("teta","NS"))
```

```
si([is("x","RC"),is("phi","LV")],is("teta","PS"))
si([is("x","RI"),is("phi","LV")],is("teta","PM"))
si([is("x","LE"),is("phi","LU")],is("teta","NB"))
si([is("x","LC"),is("phi","LU")],is("teta","NB"))
si([is("x","CE"),is("phi","LU")],is("teta","NM"))
si([is("x","RC"),is("phi","LU")],is("teta","NS"))
si([is("x","RI"),is("phi","LU")],is("teta","PS"))
si([is("x","LE"),is("phi","LB")],is("teta","NB"))
si([is("x","LC"),is("phi","LB")],is("teta","NB"))
si([is("x","CE"),is("phi","LB")],is("teta","NM"))
si([is("x","RC"),is("phi","LB")],is("teta","NM"))
si([is("x","RI"),is("phi","LB")],is("teta","NS"))
```

5.3.2 - GENERADOR DE FUNCIONES

Las tablas son generadas de acuerdo a las funciones de pertenencia y de decodificación definidas con el lenguaje de especificación. Se define cada conjunto difuso con polígonos, donde el sistema realiza la interpolación automáticamente. Es posible definir el conjunto de variables, de entrada y salida, con diferentes tamaños de palabra (precisión), tanto para las abscisas como para las ordenadas de las funciones (rango de valores que adopta). El conjunto de valores de las tablas puede ser expresado en binario puro, en complemento a dos, separando en diferentes archivos la parte alta de la baja, etc.; características que se definen modificando los parámetros del sistema.

Las funciones de decodificación, especifican una tabla que contiene los centros de gravedad de la función cortada por el valor de ingreso (el peso calculado). Opcionalmente, dichos valores son multiplicados por el peso, de tal forma que el microprograma no necesite realizar dicha operación.

El sistema que realiza los gráficos de las funciones de pertenencia y de decodificación, permite la visualización de cada variable (de entrada o de salida) con todas las funciones. El gráfico permite la comparación de todas las funciones de una variable, para detectar posibles errores de definición.

Además, se puede contar con herramientas de síntesis capaces de generar un circuito que calcule las funciones [Des97], a partir del análisis de las tablas. Esto es necesario, dado que no se puede enfocar un problema grande considerando

tablas, ya que el tamaño usado para memorias, haría imposible su materialización ASIC o FPGA.

5.3.3 - ESQUEMA DE CÁLCULO

Por medio de un lenguaje de especificación de reglas, se realiza la definición (planteo) del problema. El compilador de reglas, debe producir un algoritmo general, a ser materializado por las distintas configuraciones hardware y/o software [Aco96].

El compilador de reglas, procesa el archivo con la definición de reglas para generar un esquema de cálculo capaz de realizar las inferencias difusas definidas. Así se obtiene un esquema (o algoritmo) del motor de inferencia a materializar sobre cualquier plataforma de cálculo.

El algoritmo de generación del esquema de cálculo es:

```
a) Consultar todas las funciones de pertenencia.
b) PARA CADA variable de salida VAR
c)     PARA CADA función de decodificación FUN, de VAR
d)       Generar para cada FUN y VAR una expresión:
e)          mínimo si es usado un conector AND y
            máximo si es usado un conector OR (u otra regla)
f)     Consultar todas las funciones de decodificación
g)     Calcular el centro de gravedad de VAR
```

Donde: a) Para cada variable de entrada, consultar todas las funciones de pertenencia. b-c-d) Para cada VAR y cada FUN realizar la inferencia difusa y defusificación. e) Para cada función de pertenencia de la variable de salida VAR, agrupar todas las reglas con el mismo consecuente (se utiliza el operador mínimo para agrupar las condiciones AND y máximo para agrupar las diferentes reglas). f) Consultar las funciones de pertenencia de salida de VAR. g) Realizar el cálculo del COG o centroide, cálculo del *numerador*, *denominador* y división.

Cada resultado de una operación es asignado a una variable interna (Fig. 5.1, etapa 2). El generador del esquema produce la siguiente salida:

```
y[0][0]=A0(x0);
y[0][1]=A1(x0);
y[0][2]=A2(x0);
y[0][3]=A3(x0);
y[0][4]=A4(x0);
y[1][5]=A5(x1);
y[1][6]=A6(x1);
y[1][7]=A7(x1);
y[1][8]=A8(x1);
y[1][9]=A9(x1);
y[1][01]=A01(x1);
y[1][11]=A11(x1);
w[0][0]=y[0][0].y[1][9] |  y[0][0].y[1][01] |
      | y[0][1].y[1][01] |  y[0][0].y[1][11] |
      | y[0][1].y[1][11];
w[0][1]=y[0][0].y[1][7] |  y[0][0].y[1][8] |
      | y[0][1].y[1][8] |  y[0][1].y[1][9] |
      | y[0][2].y[1][01] |  y[0][2].y[1][11] |
      | y[0][3].y[1][11];
w[0][2]=y[0][0].y[1][6] |  y[0][1].y[1][7] |
      | y[0][2].y[1][9] |  y[0][3].y[1][01] |
      | y[0][4].y[1][11];
w[0][3]=y[0][2].y[1][8];
w[0][4]=y[0][0].y[1][5] |  y[0][1].y[1][6] |
      | y[0][2].y[1][7] |  y[0][3].y[1][9] |
      | y[0][4].y[1][01];
w[0][5]=y[0][1].y[1][5] |  y[0][2].y[1][5] |
      | y[0][2].y[1][6] |  y[0][3].y[1][7] |
      | y[0][3].y[1][8] |  y[0][4].y[1][8] |
      | y[0][4].y[1][9];
w[0][6]=y[0][3].y[1][5] |  y[0][4].y[1][5] |
      | y[0][3].y[1][6] |  y[0][4].y[1][6] |
      | y[0][4].y[1][7];
v[0][0]=B[0] (w[0][0]);
v[0][1]=B[1] (w[0][1]);
v[0][2]=B[2] (w[0][2]);
v[0][3]=B[3] (w[0][3]);
v[0][4]=B[4] (w[0][4]);
v[0][5]=B[5] (w[0][5]);
v[0][6]=B[6] (w[0][6]);
N[0]=v[0][0]+v[0][1]+v[0][2]+v[0][3]+v[0][4]+
            +v[0][5]+v[0][6];
D[0]=w[0][0]+w[0][1]+w[0][2]+w[0][3]+w[0][4]+
            +w[0][5]+w[0][6];
f[0]=N[0] / D[0];
```

Para realizar la inferencia difusa, de las reglas mostradas en la Tabla 3.1 de acuerdo a las funciones de pertenencia de θ de la Fig. 3.8, se selecciona una función de pertenencia de salida. Se mostrará con dos funciones: PS y PB. En primer lugar, se seleccionan las reglas cuyo consecuente es PS o PB.

	IF	X is	&	ϕ is	THEN	θ is
1		LE		RB		PS
2		LC		RU		PS
3		CE		RV		PS
4		LV		RC		PS
5		LU		RI		PS
	IF	**X is**	**&**	**ϕ is**	**THEN**	**θ is**
6		RC		RB		PB
7		RI		RB		PB
8		RC		RU		PB
9		RI		RU		PB
10		RI		RV		PB

Así, tomando las líneas 1 a 5 tenemos que el peso de la función de salida PS será calculado en función del máximo de (mínimo de LE y RB, mínimo de LC y RU, mínimo de CE y RV, mínimo de LV y RC, mínimo de LU y RI). Por otra parte, PB se calcula como el máximo entre (mínimo de RC y RB, mínimo de RI y RB, mínimo de RC y RU, mínimo de RI y RU, mínimo de RI y RV). Los valores difusos obtenidos en la defusificación están almacenados en:

LE en y[0][0]	RU en y[1][6]
LC en y[0][1]	RV en y[1][7]
CE en y[0][2]	VE en y[1][8]
RC en y[0][3]	LV en y[1][9]
RI en y[0][4]	LU en y[1][10]
RB en y[1][5]	LB en y[1][11]

Así, se reemplazan los nombres de las funciones de pertenencia por los registros que contienen dicho valor, y entonces se obtienen las nuevas ecuaciones de cálculo:

```
PB   =  max( min(LE,RB), min(LC,RU), min(CE,RV),
             min(LV,RC), min(LU,RI) )

W[0][4] = max( min(y[0][0],y[1][5]), min(y[0][1],y[1][6]),
               min(y[0][2],y[1][7]), min(y[1][9],y[0][3]),
               min(y[1][10],y[0][4]) )
```

```
PS   = max( min(RC,RB), min(RI,RB), min(RC,RU),
            min(RI,RU), min(RI,RV) )

W[0][6] = max( min(y[0][3],y[1][5]), min(y[0][4],y[1][5]),
               min(y[0][3], y[1][6]), min(y[0][4], y[1][6]),
               min(y[0][4], y[1][7]) )
```

5.3.4 - TRANSFORMACIÓN DEL ESQUEMA

El esquema de cálculo está expresado en operaciones de múltiples operandos, y debe ser representado utilizando funciones (+, -, *, /, *min* y *max*) de 2 operandos. Antes de realizar la transformación, se debe realizar primero una optimización del esquema de cálculo reticular del problema [Des96].

El intervalo [0,1], con las operaciones *MAX* ($\vee$, suma) y *MIN* ($\bullet$, producto) es un retículo distributivo. El conjunto *L*, con las operaciones $\vee$ (suma) y $\bullet$ (producto) es un retículo distributivo si se cumplen las siguientes relaciones (axiomas):

Idempotencia:	$a \vee a = a$	$a.a = a$
Conmutatividad:	$a \vee b = b \vee a$	$a.b = b.a$
Absorción:	$a \vee a \bullet b = a$	$a \bullet (a \vee b) = a$
Distributividad:	$a \bullet (b \vee c) = a \bullet b \vee a \bullet c$	$a \vee b \bullet c = (a \vee b) \bullet (a \vee c)$

Dos expresiones reticulares son equivalentes si representan la misma función f: $[0,1]^n \rightarrow [0,1]$. Una expresión tipo suma de productos, de la cual no se puede quitar ningún producto por aplicación de los axiomas, es una expresión canónica disyuntiva. Una expresión tipo producto de sumas, de la cual no se puede quitar ninguna suma por aplicación de los axiomas, es una expresión canónica conjuntiva.

El algoritmo en cada etapa analiza la posibilidad de descomponer cada una de las funciones, bien como producto (basándose en la correspondiente descomposición de su expresión conjuntiva), o bien como suma (expresión disyuntiva). La selección de una u otra solución se basa en un criterio de coste local. En caso de que uno de los factores (o términos) de la descomposición ya hubiera sido calculado antes, se le asigna un coste nulo. En conclusión, se obtiene un esquema de cálculo, no necesariamente óptimo (selección basada

en un criterio de coste local), al cual corresponde una estructura que puede incluir reconvergencias (*reconvergent fan out*). Se selecciona la ecuación disyuntiva o conjuntiva de acuerdo a cual tenga el mínimo coste; mientras que la división de la ecuación en todas las ecuaciones posibles, genera por *backtracking* las listas de ecuaciones equivalentes a analizar.

Cabe destacar la importancia del proceso de desarrollo del controlador, ya que una reducción del número de operaciones repercute de forma positiva no sólo en el tiempo de cálculo, sino también en el número de variables auxiliares y, por lo tanto, en el tamaño del circuito (número de registros). Este último punto es importante, en especial en el caso de una materialización mediante un ASIC.

Algoritmo del módulo de optimización reticular en pseudo-código:

```
carga todas las ecuaciones reticulares en una lista L
HACER
   PARA CADA ecuación i de L
      Divide ecuación i en j, k, ...
      PARA CADA lista de ecuaciones que componen a i
         nj := selecciona disyuntiva o conjuntiva( j )
         nk := selecciona disyuntiva o conjuntiva( k )
         ..................................
         SI (costo(nj)+costo(nk)+...<costo(i)) ∧
                                    ∧ (costo=mínimo)
            ENTONCES
            reemplaza ecuación i por nj, nk, ... en lista L
MIENTRAS  ( hay cambio en L );
```

Donde: L es una lista de todas las ecuaciones reticulares necesarias para el cálculo de una función difusa. Cada una de las ecuaciones i (que forman L) es dividida en todas las posibles ecuaciones j, k, ...; a cada una de éstas subecuaciones se las transforma en conjuntiva o disyuntiva, según corresponda. El costo de las expresiones n_j, n_k, ... es comparado con el costo de la ecuación original i, seleccionándose la ecuación de menor costo. Nótese que si la ecuación n_k ya existe en L, su costo será nula debido a que ya ha sido contabilizado.

El esquema de cálculo obtenido como salida de esta etapa es el siguiente:

```
y[0][0]=A0(x0);
y[0][1]=A1(x0);
y[0][2]=A2(x0);
y[0][3]=A3(x0);
y[0][4]=A4(x0);
y[1][5]=A5(x1);
y[1][6]=A6(x1);
y[1][7]=A7(x1);
y[1][8]=A8(x1);
y[1][9]=A9(x1);
y[1][01]=A01(x1);
y[1][11]=A11(x1);
w[0][0]=ec8 . ec9
w[0][1]=ec10 | ec11
w[0][2]=ec12 | ec13
w[0][3]=y[0][2] . y[1][8]
w[0][4]=ec16 | ec17
w[0][5]=ec18 | ec19
w[0][6]=ec20 | ec21
ec8=ec22 | ec23
ec9=y[0][0]|y[0][1]
ec10=y[0][3].y[1][11]
ec11=ec24 | ec25
ec12=y[0][4].y[1][11]
ec13=ec26 | ec27
ec16=y[0][4].y[1][01]
ec17=ec28 | ec29
ec18=y[0][4].y[1][9]
ec19=ec30 | ec31
ec20=ec32 . ec33
ec21=y[0][4].y[1][7]
ec22=y[1][01]|y[1][11]
ec23=y[1][9].y[0][0]
ec24=y[0][0].y[1][7]
ec25=ec34 | ec35
ec26=y[0][0].y[1][6]
ec27=ec36 | ec37
ec28=y[0][0].y[1][5]
ec29=ec38 | ec39
ec30=ec40 | ec41
ec31=ec42 | ec43
ec32=y[0][4]|y[0][3]
ec33=y[1][6]|y[1][5]
ec34=ec44 | ec45
ec35=ec22 . y[0][2]
ec36=y[0][2].y[1][9]
ec37=ec46 | ec47
ec38=y[0][2].y[1][7]
ec39=ec48 | ec49
ec40=y[0][3].y[1][7]
ec41=ec32 . y[1][8]
ec42=y[0][1].y[1][5]
ec43=ec33 . y[0][2]
ec44=ec9 . y[1][8]
ec45=y[0][1].y[1][9]
ec46=y[0][1].y[1][7]
ec47=y[0][3].y[1][01]
ec48=y[0][1].y[1][6]
ec49=y[0][3].y[1][9]
v[0][0]=B[0] (w[0][0] );
v[0][1]=B[1] (w[0][1] );
v[0][2]=B[2] (w[0][2] );
v[0][3]=B[3] (w[0][3] );
v[0][4]=B[4] (w[0][4] );
v[0][5]=B[5] (w[0][5] );
v[0][6]=B[6] (w[0][6]);
N[0]=v[0][0]+v[0][1]+
        +v[0][2]+v[0][3]+
        +v[0][4]+v[0][5]+
        +v[0][6];
D[0]=w[0][0]+w[0][1]+
        +w[0][2]+w[0][3]+
        +w[0][4]+w[0][5]+
        +w[0][6];
f[0]=N[0] / D[0];
```

5.3.5 - PROGRAMA DE CÁLCULO

La generación del programa es totalmente dependiente de la arquitectura sobre la cual se realice la materialización definitiva. Por ejemplo, si se desea ejecutar sobre un ASIC con 4 ALU's comunicadas con un sistema de *pipelinig*, el generador del programa debe conocerlo, para poder aprovechar todas las características que brinda el hardware.

La ruta de datos para ejecutar el cálculo está formada por los siguientes bloques (Fig. 5.2): (1) una (P)ROM para almacenar las tablas de las funciones de

pertenencia y decodificación, (2) un banco de registros, (3) una unidad aritmética – lógica, y (4) un puerto de salida. Esta ruta de datos es controlada por un microprograma almacenado en otra (P)ROM. A cada bloque le corresponde una descripción adaptable por medio de parámetros de VHDL. Para integrar un controlador particular, es necesario definir los parámetros (*n*, *p*, *m*, *n_page*, *n_E*, y *n_d*) y generar los archivos de datos con el contenido de las (P)ROMs.

La traducción del último esquema en un microprograma es hecha por las tres fases siguientes (Fig. 5.1, etapa 4) [Kan98]:

a) La asignación óptima de registros, para ejecutar el programa en una mínima cantidad de memoria.

b) Un programa basado en las siguientes instrucciones:

INPUT i, j	R(j) <= INi
OUTPUT i	OUTi <= registro de desplazamiento *Q* (salida división)
TABLE i, j, k	R(i) <= fila número *R*(*k*) de la tabla número *j*
ZERO i	R(i) <= 0
MIN i, j, k	R(i) <= R(j) . R(k)
MAX i, j, k	R(i) <= R(j) v R(k)
ADD i1, ..., ip; j1, ..., jp; k1, ..., kp	R(i1)&....&R(ip) <=R(j1)&....&R(jp) + R(k1)&....&R(kp) (dos operandos de *p* palabras)
DIV i1,..., ip; j1,..., jp; k	Q(k-1:0) <= R(i1)&....&R(ip) / R(j1)&....&R(jp) (división de operandos de *p* palabras, con *k* bits de precisión; el cociente es almacenado en un registro de desplazamiento en el bloque de salida).

c) Asignación de identificadores de las tablas.

El *generador de programa* produce el siguiente programa:

0	INPUT 0,0	13	MAX 0,5,4
1	INPUT 1,1	14	MAX 4,2,6
2	INPUT 2,2	15	MIN 2,0,3
3	TABLE 3,2,1	16	MIN 0,4,1
4	TABLE 4,3,1	17	TABLE 4,6,0
5	TABLE 5,4,2	18	TABLE 5,7,0
6	TABLE 6,5,2	19	TABLE 1,8,2
7	MAX 1,3,5	20	TABLE 3,9,2
8	MIN 2,3,5	21	ZERO 7
9	MAX 3,4,6	22	ADD 5,4,3;7,3,1;7;5,4
10	MIN 5,4,6	23	ADD 2,0;7,2;7,0
11	TABLE 4,0,0	24	DIV 5,4,3;2,0,7
12	TABLE 6,1,0	25	OUTPUT 0

La asignación de identificadores de tablas para las funciones de pertenencia y decodificación es:

0	$A0_{(07:0)}$	5	$A5_{(07:0)}$
1	$A1_{(07:0)}$	6	$B0_{(07:0)}$
2	$A2_{(07:0)}$	7	$B0_{(15:8)}$
3	$A3_{(07:0)}$	8	$B1_{(07:0)}$
4	$A4_{(07:0)}$	9	$B1_{(15:8)}$

Nótese que las tablas de decodificación son leídas en precisión simple, implementando la doble precisión a través de un doble acceso a la memoria.

Para la generación del programa, se debe trabajar con direcciones (o identificadores de registros) reales; a los identificadores del algoritmo de cálculo se les asigna un tamaño (en palabras) y una dirección (que nos define el ancho del bus de direcciones de la plataforma). Es recomendable poder reducir el número de registros, haciendo su reutilización máxima.

Este problema de asignación de la mínima cantidad de registros, es un clásico problema NP-completo por lo que aquí se utilizan diferentes estrategias para resolverlo (*Greedy*), quedando a criterio del diseñador cual de las asignaciones finales tomar.

La precedencia de las operaciones forma un grafo dirigido sin ciclos, respetando esta relación de precedencia, el objetivo es minimizar la

cantidad de registros totales a ser utilizados. En la Fig. 5.3, como ejemplo, se ve un archivo de entrada y el grafo de precedencia que se genera.

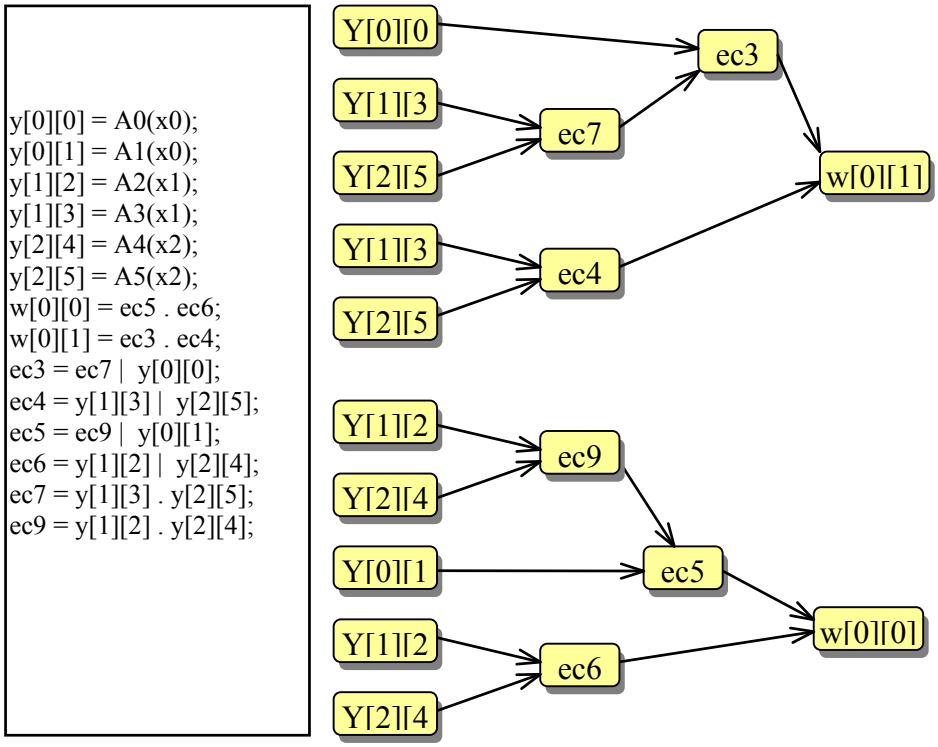


Fig. 5.3-Archivo y grafo de precedencia de operaciones

El programa comienza con un análisis léxico y sintáctico del archivo de entrada generando estructuras de datos para que luego los diferentes algoritmos de asignación operen sobre estas. Algunas de ellas son:

- **tabla_de_símbolos:** es la tabla de símbolos generada por el *parser*, contiene también, para cada entrada, la cantidad de referencias y el registro que está ocupando actualmente.

- **lista_operación:** contiene la lista de operaciones binarias que hay que hacer. Cada operación contiene los índices en la tabla de símbolos del resultado, sus 2 operandos y la operación a realizar.

- **lista_funciones:** es la lista con operaciones de lecturas a tablas.

- **registros:** contiene el estado de los registros.

Estas estructuras de datos son en general clases C++, con lo que se oculta su representación y comportamiento, brindando funciones de accesos para su consulta y modificación.

LAS ESTRATEGIAS DE ASIGNACIÓN

Estrategia uno: En ella se lee una función de entrada, y luego se trata de efectuar todas las operaciones que sean posibles (es decir que sus operandos estén en memoria). Cuando no quedan más operaciones que se puedan llevar a cabo, se lee una nueva entrada, y se vuelve a iterar hasta lograr que la lista de operaciones este vacía. A continuación se puede ver el pseudo-código correspondiente.

```
Asignar_registros1()
   Hasta vaciar la lista de operaciones
      Leer una entrada
      Ver en todas las operaciones
         SI sus operandos están en registros
             Hacer la operación
             Liberar los registros que se puedan
```

Estrategia dos: En esta estrategia se hacen todas las lecturas a las tablas primero y luego se tratan de hacer las operaciones conforme sus dos operandos están en registros. No hay ningún criterio en particular para decidir que operación tomar. Esta estrategia en contra de lo intuitivo en la mayoría de los casos mejora a la primera.

```
Asignar_registros2()
   Leer todas las entradas
   Hasta vaciar la lista de operaciones
      Ver en todas las operaciones
          SI sus operandos están en registros
              Hacer la operación
              Liberar los registros que se puedan
```

Estrategia tres: Aquí, al igual que en la estrategia anterior se hacen todas las lecturas a las tablas primero, pero para elegir que operación hacer se busca aquella en la que sus operandos tengan la menor cantidad de referencias (con el objeto de liberar los registros cuanto antes). Se realiza esto hasta no tener más operaciones por hacer. Esta es la estrategia que mejores resultados da en general, aunque se han encontrado ejemplos en contra.

```
Asignar_registros_3()
   Leer todas las entradas
   Hasta vaciar la lista de operaciones
      Ver en todas las operaciones
          De las operaciones cuyos operandos están
          en registros elegir la mínima cantidad de
          referencias
                Hacer la operación
                Liberar los registros que se puedan
```

Estrategia cuatro: Se podría considerar como una mezcla de las estrategias uno y tres, ya que aquí se leen las entradas según sean necesarias, y además para decidir que función tomar se elige aquella operación en la que sus operandos tengan la menor cantidad de referencias. Para tomar una nueva entrada se elige aquella que tenga también menor cantidad de referencias.

```
Asignar_registros_4()
   Leer una entrada
   Hasta vaciar la lista de operaciones
       De las operaciones cuyos operandos están en registros
       Elegir la de mínima cantidad de referencias
             Hacer la operación
             Liberar los registros que se puedan
   Leer otra entrada
```

Las cuatro estrategias de asignación de registros analizadas utilizan un proceso de búsqueda de la operación con mínima cantidad de referencias para realizar las operaciones que liberan registros lo más pronto posible. De esta manera se pueden utilizar los registros para otra operación y reducir la cantidad de memoria necesaria para el cálculo. A continuación se muestra el programa de cálculo del ejemplo, obtenido siguiendo la tercer estrategia.

```
R0 <- Reservado
R1 <- A11(X1)   ;        Y[1][11] = A11(X1)
R2 <- A01(X1)   ;        Y[1][10] = A01(X1)
R3 <- R2 | R1  ;         EC22 = Y[1][10] | Y[1][11]
R4 <- A8(X1)    ;        Y[1][8] = A8(X1)
R5 <- A6(X1)    ;        Y[1][6] = A6(X1)
R6 <- A5(X1)    ;        Y[1][5] = A5(X1)
R7 <- R5 | R6  ;         EC33 = Y[1][6] | Y[1][5]
R8 <- A9(X1)    ;        Y[1][9] = A9(X1)
R9 <- A7(X1)    ;        Y[1][7] = A7(X1)
R10 <- A4(X0)   ;        Y[0][4] = A4(X0)
R11 <- R10 . R9  ;       EC21 = Y[0][4] . Y[1][7]
R12 <- R10 . R8  ;       EC18 = Y[0][4] . Y[1][9]
R13 <- R10 . R2  ;       EC16 = Y[0][4] . Y[1][10]
R14 <- R10 . R1  ;       EC12 = Y[0][4] . Y[1][11]
R15 <- A3(X0)   ;        Y[0][3] = A3(X0)
R16 <- R15 . R8  ;       EC49 = Y[0][3] . Y[1][9]
R17 <- R15 . R2  ;       EC47 = Y[0][3] . Y[1][10]
R18 <- R15 . R9  ;       EC40 = Y[0][3] . Y[1][7]
R19 <- R10 | R15  ;      EC32 = Y[0][4] | Y[0][3]
R20 <- R19 . R7  ;       EC20 = EC32 . EC33
R21 <- R15 . R1  ;       EC10 = Y[0][3] . Y[1][11]
R22 <- R20 | R11  ;      W[0][6] = EC20 | EC21
R1 <- A2(X0)    ;        Y[0][2] = A2(X0)
R2 <- R7 . R1  ;         EC43 = EC33 . Y[0][2]
R10 <- R19 . R4  ;       EC41 = EC32 . Y[1][8]
R11 <- R1 . R9  ;        EC38 = Y[0][2] . Y[1][7]
R15 <- R1 . R8  ;        EC36 = Y[0][2] . Y[1][9]
R20 <- R3 . R1  ;        EC35 = EC22 . Y[0][2]
R23 <- R18 | R10  ;      EC30 = EC40 | EC41
R24 <- R1 . R4  ;        W[0][3] = Y[0][2] . Y[1][8]
R1 <- A1(X0)    ;        Y[0][1] = A1(X0)
R2 <- R1 . R5  ;         EC48 = Y[0][1] . Y[1][6]
R7 <- R1 . R9  ;         EC46 = Y[0][1] . Y[1][7]
R10 <- R1 . R8  ;        EC45 = Y[0][1] . Y[1][9]
R11 <- R1 . R6  ;        EC42 = Y[0][1] . Y[1][5]
R15 <- R2 | R16  ;       EC39 = EC48 | EC49
R18 <- R7 | R17  ;       EC37 = EC46 | EC47
R19 <- R11 | R2  ;       EC31 = EC42 | EC43
R20 <- R11 | R15  ;      EC29 = EC38 | EC39
R25 <- R15 | R18  ;      EC27 = EC36 | EC37
R26 <- R23 | R19  ;      EC19 = EC30 | EC31
R27 <- R12 | R26  ;      W[0][5] = EC18 | EC19
R17 <- A0(X0)   ;        Y[0][0] = A0(X0)
R2 <- R17 . R6  ;        EC28 = Y[0][0] . Y[1][5]
R7 <- R17 . R5  ;        EC26 = Y[0][0] . Y[1][6]
R10 <- R17 . R9  ;       EC24 = Y[0][0] . Y[1][7]
R11 <- R8 . R17  ;       EC23   = Y[1][9] . Y[0][0]
```

```
R12 <- R2 | R20  ;       EC17 = EC28 | EC29
R15 <- R7 | R25  ;       EC13 = EC26 | EC27
R16 <- R17 | R1  ;       EC9 = Y[0][0] | Y[0][1]
R17 <- R3 | R11  ;       EC8 = EC22 | EC23
R18 <- R13 | R12  ;      W[0][4] = EC16 | EC17
R19 <- R14 | R15  ;      W[0][2] =EC12 | EC13
R20 <- R17 . R16  ;      W[0][0] = EC8 . EC9
R1 <- R16 . R4  ;        EC44 = EC9 . Y[1][8]
R2 <- R1 | R10  ;        EC34 = EC44 | EC45
R3 <- R2 | R20  ;        EC25 = EC34 | EC35
R5 <- R10 | R3  ;        EC11 = EC24 | EC25
R6 <- R21 | R5  ;        W[0][1] = EC10 | EC11
;Los resultados están en los siguientes registros:
;  R20 -> W[0][0]
;  R6 -> W[0][1]
;  R19 -> W[0][2]
;  R24 -> W[0][3]
;  R18 -> W[0][4]
;  R27 -> W[0][5]
;  R22 -> W[0][6]
;La cantidad máxima de Registros usados fue 28.
;Es decir de R0 a R27
; Suma del Denominador D
(R5, R4) <- (R0, R20) + (R0, R6)
(R5, R4) <- (R5, R4) + (R0, R19)
(R5, R4) <- (R5, R4) + (R0, R24)
(R5, R4) <- (R5, R4) + (R0, R18)
(R5, R4) <- (R5, R4) + (R0, R27)
(R5, R4) <- (R5, R4) + (R0, R22)
; Consulta de tablas con pesos
R9  <- B(12, R18) ;
R10 <- B(13, R18) ;
R11 <- B(14, R6) ;
R12 <- B(15, R6) ;
R13 <- B(16, R24) ;
R14 <- B(17, R24) ;
R17 <- B(20, R19) ;
R2  <- B(21, R19) ;
R25 <- B(22, R27) ;
R21 <- B(23, R27) ;
R26 <- B(24, R20) ;
R23 <- B(25, R20) ;
; Suma del numerador N
(R16, R15) <-  (R2, R17) + (R21, R25) ;
(R16, R15) <- (R16, R15) + (R23, R26) ;
(R3, R16, R15) <- (R0, R16, R15) + (R0, R10, R9) ;
(R3, R16, R15) <- (R3, R16, R15) + (R0, R12, R11) ;
(R3, R16, R15) <- (R3, R16, R15) + (R0, R14, R13) ;
; División F[0] = N / D
F[0] = (R3, R16, R15) / (R0, R5, R4)
```

5.3.6 - GENERACIÓN DEL MICROPROGRAMA

Durante esta etapa del diseño del controlador, se transforma el programa de cálculo en el correspondiente microprograma a ejecutarse en el procesador difuso. El sistema para la generación del microprograma supone una arquitectura

determinada (conocida por el generador) sobre la cual se ejecutará.

El sistema de generación del microprograma trabaja como un sistema de reemplazo de macros definidas previamente, de acuerdo a la plataforma seleccionada. Se deben tener en cuenta además las características especiales de la plataforma, por ejemplo la definición del estado de los *flip-flops* internos, ver [Des97]. Hay clases de operaciones que presuponen un estado inicial, por lo cual de acuerdo a la secuencia de operaciones se debe o no definir el estado inicial de los *flip-flops*. Se contempla también el caso que para minimizar las operaciones que definen el valor inicial de los *flip-flops*, se deba intercambiar de orden un grupo de instrucciones; minimizando su cantidad se puede obtener la máxima frecuencia de funcionamiento.

El algoritmo se describe a continuación:

```
lee línea de programa
Verifica secuencia de Operaciones Convenientes
DE ACUERDO A línea de programa leída EXPANDIR
       lectura de tablas de codificación
       lectura de tablas de decodificación
       cálculo de máximos o mínimos
       sumatoria de pesos
       sumatoria centros de gravedad
       división
```

La función de verificación de secuencia de operaciones analiza las *n* operaciones a realizar, para determinar el orden en que es conveniente realizarlas, para minimizar la cantidad de instrucciones de inicialización. El número de operaciones a analizar por cada instrucción a generar es definido externamente. Las sumas (de pesos y centros de gravedad) y la división generan los múltiples pasos del microprograma que realizan dichas operaciones, de acuerdo a la plataforma hardware de soporte.

Para generar el microprograma es necesario partir las operaciones de múltiple precisión y la división en operaciones elementales (suma y resta de una simple palabra) ejecutadas en un ciclo de reloj (Fig. 5.1, etapas 5 y 6).

Por ejemplo, el bloque ADD 2,0;7,2;7,0 produce las siguientes tres microinstrucciones:

a) x <= 0, c <= 0, S/R <= 0;

b) R(0) <= ADD/SUBTRACT(R(2), R(0)),

x <= xsigu, c <= csigu, S/R <= S/R;

c) R(2) <= ADD/SUBTRACT(R(7), R(7));

Donde *x*, *c* y *S/R* son *flip-flops* internos de la UAL, mientras que *xsigu* y *csigu* son salidas de la ALU, ver Fig. 4.1.

A continuación se muestra y explica el código del microprograma de control generado por el sistema.

Nro	Dir_I	Dir_J	Dir_K	Tabla	T_v	FF	sh	Fun	out
0	00000	00000	00000	00000	0	10	0	001	1
1	00001	00000	00000	01011	1	00	0	000	0
2	00010	00000	00000	01010	1	00	0	000	0
3	00011	00010	00001	00000	0	00	0	100	0
4	00100	00000	00000	01000	1	00	0	000	0
5	00101	00000	00000	00110	1	00	0	000	0
6	00110	00000	00000	00101	1	00	0	000	0
7	00111	00101	00110	00000	0	00	0	100	0
8	01000	00000	00000	01001	1	00	0	000	0
9	01001	00000	00000	00111	1	00	0	000	0
10	01010	00000	00000	00100	1	00	0	000	0
11	01011	01010	01001	00000	0	00	0	010	0
12	01100	01010	01000	00000	0	00	0	010	0
13	01101	01010	00010	00000	0	00	0	010	0
14	01110	01010	00001	00000	0	00	0	010	0
15	01111	00000	00000	00011	1	00	0	000	0
16	10000	01111	01000	00000	0	00	0	010	0
17	10001	01111	00010	00000	0	00	0	010	0
18	00010	01111	01001	00000	0	00	0	010	0
19	10010	01010	01111	00000	0	00	0	100	0
20	01010	10010	00111	00000	0	00	0	010	0
21	10011	01111	00001	00000	0	00	0	010	0
22	00001	01010	01011	00000	0	00	0	100	0

23	01010	10010	00100	00000	0	00	0	010	0
24	01011	00010	01010	00000	0	00	0	100	0
25	00010	00000	00000	00010	1	00	0	000	0
26	01010	00111	00010	00000	0	00	0	010	0
27	00111	00010	01001	00000	0	00	0	010	0
28	01111	00010	01000	00000	0	00	0	010	0
29	10010	00011	00010	00000	0	00	0	010	0
30	10100	00010	00100	00000	0	00	0	010	0
31	00010	00000	00000	00001	1	00	0	000	0
32	10101	00010	00101	00000	0	00	0	010	0
33	10110	00010	01001	00000	0	00	0	010	0
34	10111	00010	01000	00000	0	00	0	010	0
35	11000	00010	00110	00000	0	00	0	010	0
36	11001	10101	10000	00000	0	00	0	100	0
37	10000	10110	10001	00000	0	00	0	100	0
38	10001	11000	01010	00000	0	00	0	100	0
39	01010	00111	11001	00000	0	00	0	100	0
40	00111	01111	10000	00000	0	00	0	100	0
41	01111	01011	10001	00000	0	00	0	100	0
42	01011	01100	01111	00000	0	00	0	100	0
43	01100	00000	00000	00000	1	00	0	000	0
44	01111	01100	00110	00000	0	00	0	010	0
45	00110	01100	00101	00000	0	00	0	010	0
46	00101	01100	01001	00000	0	00	0	010	0
47	01001	01000	01100	00000	0	00	0	010	0
48	01000	01111	01010	00000	0	00	0	100	0
49	01010	00110	00111	00000	0	00	0	100	0
50	00110	01100	00010	00000	0	00	0	100	0
51	00010	00011	01001	00000	0	00	0	100	0
52	00011	01101	01000	00000	0	00	0	100	0
53	00111	01110	01010	00000	0	00	0	100	0
54	01000	00010	00110	00000	0	00	0	010	0
55	00010	00110	00100	00000	0	00	0	010	0
56	00100	00010	10111	00000	0	00	0	100	0
57	00010	00100	10010	00000	0	00	0	100	0
58	00100	00101	00010	00000	0	00	0	100	0
59	00010	10011	00100	00000	0	00	0	100	0

Sumatoria de todos los Wi (calculados)

$(D_1\ D_0) = (0\ w_0) + (0\ w_1)$; $(D_1\ D_0) = (D_1\ D_0) + (0\ w_2)$; $(D_1\ D_0) = (D_1\ D_0) + (0\ w_3)$

$(D_1\ D_0) = (D_1\ D_0) + (0\ w_4)$; $(D_1\ D_0) = (D_1\ D_0) + (0\ w_5)$; $(D_1\ D_0) = (D_1\ D_0) + (0\ w_6)$

Donde : D_1 = R4 (00100) D_0 = R5 (00101)

Nro	Dir_I	Dir_J	Dir_K	Tabla	T_v	FF	sh	Fun	out
60	00000	00000	00000	00000	0	01	0	001	0
61	00100	00001	00010	00000	0	11	0	110	0
62	00101	00000	00000	00000	0	01	0	110	0

63	00100	00100	00011	00000	0	11	0	110	0
64	00101	00101	00000	00000	0	01	0	110	0
65	00100	00100	00111	00000	0	11	0	110	0
66	00101	00101	00000	00000	0	01	0	110	0
67	00100	00100	01000	00000	0	11	0	110	0
68	00101	00101	00000	00000	0	01	0	110	0
69	00100	00100	01011	00000	0	11	0	110	0
70	00101	00101	00000	00000	0	01	0	110	0
71	00100	00100	10100	00000	0	11	0	110	0
72	00101	00101	00000	00000	0	01	0	110	0

Lectura de todas las tablas Bxx L y H
Se lee la parte baja primero y luego la alta

Nro	Dir_I	Dir_J	Dir_K	Tabla	T_v	FF	sh	Fun	out
73	00110	01000	00000	01100	1	00	0	000	0
74	01001	01000	00000	01101	1	00	0	000	0
75	01010	00010	00000	01110	1	00	0	000	0
76	01100	00010	00000	01111	1	00	0	000	0
77	01101	00111	00000	10000	1	00	0	000	0
78	01110	00111	00000	10001	1	00	0	000	0
79	10001	00011	00000	10100	1	00	0	000	0
80	10010	00011	00000	10101	1	00	0	000	0
81	10011	01011	00000	10110	1	00	0	000	0
82	10101	01011	00000	10111	1	00	0	000	0
83	10110	00001	00000	11000	1	00	0	000	0
84	10111	00001	00000	11001	1	00	0	000	0

Suma de los pesos de las tablas de decodificación

$(N_1\ N_0) = (Z_{0H}\ Z_{0L}) + (Z_{1H}\ Z_{1L})$; $(N_1\ N_0) = (N_1\ N_0) + (Z_{2H}\ Z_{2L})$

$(N_2\ N_1\ N_0) = (N_2\ N_1\ N_0) + (0\ Z_{3H}\ Z_{3L})$; $(N_2\ N_1\ N_0) = (N_2\ N_1\ N_0) + (0\ Z_{4H}\ Z_{4L})$

$(N_2\ N_1\ N_0)=(N_2\ N_1\ N_0)+(0\ Z_{5H}\ Z_{5L})$; $(N_2\ N_1\ N_0)=(N_2\ N_1\ N_0)+(0\ Z_{6H}\ Z_{6L})$

donde : N_2 = R26(11010) ; N_1 = 25(11001) ; N_0 = 24(11000)

Nro	Dir_I	Dir_J	Dir_K	Tabla	T_v	FF	Sh	Fun	out
85	00000	00000	00000	00000	0	01	0	001	0
86	11000	10001	10011	00000	0	11	0	110	0
87	11001	10010	10101	00000	0	01	0	110	0
88	11000	11000	10110	00000	0	11	0	110	0
89	11001	11001	10111	00000	0	10	0	110	0
90	11000	11000	00110	00000	0	11	0	110	0
91	11001	11001	01001	00000	0	11	0	110	0
92	11010	00000	00000	00000	0	10	0	110	0
93	11000	11000	01010	00000	0	11	0	110	0
94	11001	11001	01100	00000	0	11	0	110	0
95	11010	11010	00000	00000	0	10	0	110	0
96	11000	11000	01101	00000	0	11	0	110	0
97	11001	11001	01110	00000	0	11	0	110	0
98	11010	11010	00000	00000	0	10	0	110	0

División									
Nro	Dir_I	Dir_J	Dir_K	Tabla	T_v	FF	sh	Fun	out
99	00000	00000	00000	00000	0	01	0	001	0
100	11010	11010	00000	00000	0	10	1	110	0
101	11000	11000	00000	00000	0	11	0	111	0
102	11001	11001	00100	00000	0	11	0	111	0
103	11010	11010	00101	00000	0	10	1	111	0
104	11000	11000	00000	00000	0	11	0	111	0
105	11001	11001	00100	00000	0	11	0	111	0
106	11010	11010	00101	00000	0	10	1	111	0
107	11000	11000	00000	00000	0	11	0	111	0
108	11001	11001	00100	00000	0	11	0	111	0
109	11010	11010	00101	00000	0	10	1	111	0
110	11000	11000	00000	00000	0	11	0	111	0
111	11001	11001	00100	00000	0	11	0	111	0
112	11010	11010	00101	00000	0	10	1	111	0
113	11000	11000	00000	00000	0	11	0	111	0
114	11001	11001	00100	00000	0	11	0	111	0
115	11010	11010	00101	00000	0	10	1	111	0
116	11000	11000	00000	00000	0	11	0	111	0
117	11001	11001	00100	00000	0	11	0	111	0
118	11010	11010	00101	00000	0	10	1	111	0
119	11000	11000	00000	00000	0	11	0	111	0
120	11001	11001	00100	00000	0	11	0	111	0
121	11010	11010	00101	00000	0	10	1	111	0
122	11000	11000	00000	00000	0	11	0	111	0
123	11001	11001	00100	00000	0	11	0	111	0
124	11010	11010	00101	00000	0	10	1	111	0

El microprograma del control de estacionamiento del vehículo se compone de las 125 microinstrucciones. Cada palabra de control contiene 28 bits. Las 2 primeras columnas contienen el número de orden de cada microinstrucción en decimal.

5.4 – SIMULACIÓN POR SOFTWARE

El código generado para simular el controlador puede ser definido en los siguientes lenguajes de programación: C o Delphi. En todos los casos el software se limita a materializar el algoritmo propuesto de tal forma que se pueda

utilizar este programa para poner a punto el controlador, mas sólo con un delicado trabajo de optimización éste código podría transformarse en versión definitiva a ser materializada por software sobre una computadora. A continuación se muestra el código del bloque principal de cálculo del controlador.

```
PROGRAM camion;
VAR y : ARRAY [0..1,0..11] OF BYTE;
    w : ARRAY [0..1,0..1] OF BYTE;
    v : ARRAY [0..1,0..6] OF WORD;
    n, d : ARRAY [0..1,0..1] OF WORD;
    F0 : BYTE;
    x : ARRAY [0..1] OF BYTE;
BEGIN
Writeln('ingrese X0 ?'); Readln(x0);
Writeln('ingrese X1 ?'); Readln(x1);
{ carga valores de funciones de pertenencia }
y[0][0]:=A0(x0);  {LE}
y[0][1]:=A1(x0);  {LC}
y[0][2]:=A2(x0);  {CE}
y[0][3]:=A3(x0);  {RC}
y[0][4]:=A4(x0);  {RI}
y[1][5]:=A5(x1);  {RB}
y[1][6]:=A6(x1);  {RU}
y[1][7]:=A7(x1);  {RV}
y[1][8]:=A8(x1);  {VE}
y[1][9]:=A9(x1);  {LV}
y[1][10]:=A10(x1);  {LU}
y[1][11]:=A11(x1);  {LB}
{ Calcula las expresiones reticulares }
w[0][0]:=max( min(y[0][0],y[1][9]),  min(y[0][0],y[1][10]),
              min(y[0][1],y[1][10]), min(y[0][0],y[1][11]),
              min(y[0][1],y[1][11])  );
w[0][1]:=max( min(y[0][0],y[1][7]),  min(y[0][0],y[1][8]),
              min(y[0][1],y[1][8]),  min(y[0][1],y[1][9]),
              min(y[0][2],y[1][10]), min(y[0][2],y[1][11]),
              min(y[0][3],y[1][11])  );
w[0][2]:=max( min(y[0][0],y[1][6]),  min(y[0][1],y[1][7]),
              min(y[0][2],y[1][9]),  min(y[0][3],y[1][10]),
              min(y[0][4],y[1][11])  );
w[0][3]:= min(y[0][2],y[1][8]);
w[0][4]:=max( min(y[0][0],y[1][5]),  min(y[0][1],y[1][6]),
              min(y[0][2],y[1][7]),  min(y[0][3],y[1][9]),
              min(y[0][4],y[1][10])  );
w[0][5]:=max( min(y[0][1],y[1][5]),  min(y[0][2],y[1][5]),
              min(y[0][2],y[1][6]),  min(y[0][3],y[1][7]),
              min(y[0][3],y[1][8]),  min(y[0][4],y[1][8]),
              min(y[0][4],y[1][9])   );
w[0][6]:=max( min(y[0][3],y[1][5]),  min(y[0][4],y[1][5]),
              min(y[0][3],y[1][6]),  min(y[0][4],y[1][6]),
              min(y[0][4],y[1][7])   );
```

```
{ carga valores de funciones de decodificacion }
v[0][0]:=B0( w[0][0] );  {NB}
v[0][1]:=B1( w[0][1] );  {NM}
v[0][2]:=B2( w[0][2] );  {NS}
v[0][3]:=B3( w[0][3] );  {ZE}
v[0][4]:=B4( w[0][4] );  {PS}
v[0][5]:=B5( w[0][5] );  {PM}
v[0][6]:=B6( w[0][6] );  {PB}
{ calcula numeradores }
n[0]:=v[0][0]+v[0][1]+v[0][2]+v[0][3]+v[0][4]+v[0][5]+v[0][6];
{ calcula denominadores }
d[0]:=w[0][0]+w[0][1]+w[0][2]+w[0][3]+w[0][4]+w[0][5]+w[0][6];
{ calcula COG }
F0:=n[0] div d[0];
Writeln(f0);
END.
```

El generador de funciones de pertenencia puede ser realizado a) por una memoria que contenga los valores, o b) por un conjunto de funciones que las calcule en línea. A continuación se muestra el código de la función de pertenencia ***LE*** de la variable ***X***.

```
FUNCTION A0( X : INTEGER ) : INTEGER;
BEGIN
     IF (X <= -380) THEN
     A0:=255
ELSE IF (X < -120) THEN
     A0:=(X+120)*(-0.9807692307692)
ELSE A0:=0;
END;
```

5.5 - RESUMEN

En este capítulo se analiza el algoritmo de cálculo de un controlador basado en lógica difusa, para luego plantear una metodología (Fig. 5.1) para su diseño. La metodología se apoya en varias herramientas para la generación automática del microprograma y del contenido de las memorias de las funciones de pertenencia y decodificación. Entre tales herramientas (ver Apéndice A) se describen las que asisten para los siguientes propósitos:

- Lenguaje de especificación de controladores difusos.

- Generación de simuladores software (programas) que ejecutan el algoritmo de control difuso.
- Generación de un esquema de cálculo básico.
- Esquema de cálculo modificado mediante la optimización reticular y aritmética.
- Generación de un programa con asignación óptima de registros de memoria.
- Generación del microprograma para control de la ruta de datos que implementa dicho algoritmo.
- Generación de datos de prueba; y finalmente.
- Comparación de las simulaciones hardware y software.

Las características de la arquitectura propuesta son:

- Necesita un alto número de registros.
- Calcula todas las operaciones reticulares.
- Puede reducirse el tamaño de la memoria mediante la asignación óptima de registros.
- No usa conocimiento alguno del mecanismo de inferencia difusa.
- Necesita una gran cantidad de puertos de entrada/salida para las líneas de control del microprograma.
- No usa el paralelismo implícito del algoritmo.

- Debe realizar las operaciones de múltiple precisión con múltiples operaciones de simple precisión.
- Necesita la transformación de operaciones reticulares y aritméticas a operaciones equivalentes de dos operandos.

Con las herramientas definidas en éste capítulo, las funciones de pertenencia deben ser materializadas en memoria, o se debe hacer el circuito a medida.

6 - ARQUITECTURA DIRIGIDA POR REGLAS

6.1 - INTRODUCCIÓN

En este capítulo se presenta una arquitectura digital para implementar un controlador difuso de altas prestaciones. En primer lugar se estudia una arquitectura que realiza el cálculo de las funciones de pertenencia siguiendo polinomios lineales en un esquema enfocado al cálculo. Este circuito, para el cálculo de funciones de pertenencia, soporta un método selectivo de inferencia para determinar el conjunto de reglas que se activan, de acuerdo a los valores fusificados. En segundo lugar, se describe una arquitectura totalmente a medida para implementar el motor de inferencias difusas, en un esquema completamente dirigido por reglas, lo que genera una utilización de los recursos muy eficaz. Por último, la defusificación se tiene en cuenta en la ruta de datos de altas prestaciones; no así la división, que se realiza utilizando un algoritmo combinacional genérico.

Las características buscadas del fusificador son:

- Reducción del número de ciclos de reloj necesarios para calcular todas las funciones de pertenencia de la aplicación.
- Soporte de polinomios lineales definidos por un número arbitrario de puntos.
- Reducción del área de control al permitir el uso de algoritmos lineales (sin bifurcaciones).
- Reducción del ancho de la palabra de control.
- Reducción del área del circuito.

- Reducción de la memoria necesaria mediante el uso de variables cuya representación sea a medida de la aplicación.
- Reducción de la cantidad de instrucciones.

6.2 - REGLAS DE INFERENCIA ACTIVAS

Si A_0, A_1, ... , A_{n-1}, son las funciones de pertenencia de la variable de entrada x. Dado un valor particular x_0 de x, el número de valores diferentes de cero entre: $A_0(x_0)$, $A_1(x_0)$, ... , $A_{n-1}(x_0)$, es mucho menor que n. Suponiendo que existe una función $b:L\rightarrow\{ 0, 1, ..., n\text{-}1 \}$, donde L representa el dominio de x, y s un valor constante tal que:

$$A_0(x) = A_1(x) = ... = A_{b(x)-1}(x) = 0,$$

$$A_{b(x)+s}(x) = A_{b(x)+s+1}(x) = ... = A_{n-1}(x) = 0,$$

los únicos valores distintos de cero están entre:

$$A_{b(x)}(x), A_{b(x)+1}(x), ... , A_{b(x)+s-1}(x).$$

Un ejemplo es el mostrado en la Fig. 6.1. Los resultados de las funciones de pertenencia analizadas son resumidos en la Tabla 6.1.

	Funciones no - cero
$x \le x_1$	A_0
$x_1 < x \le x_2$	$A_0\ A_1$
$x_2 < x \le x_3$	A_1
$x_3 < x \le x_4$	$A_1\ A_2$
$x_4 < x \le x_5$	A_2
$x_5 < x \le x_6$	$A_2\ A_3$
$x_6 < x \le x_7$	$A_2\ A_3\ A_4$
$x_7 < x \le x_8$	$A_3\ A_4$
$x_8 < x \le x_9$	$A_3\ A_4\ A_5$
$x_9 < x \le x_{10}$	$A_4\ A_5$
$x_{10} < x$	A_5

Tabla 6.1 - Resultados del primer ejemplo

Por lo tanto, $\boldsymbol{s} = 3$ y $\boldsymbol{b}$ pueden ser definidos como:

$$\begin{aligned} \boldsymbol{x} < \boldsymbol{x}_5 &: \quad \boldsymbol{b}(\boldsymbol{x})=0, \\ \boldsymbol{x}_5 < \boldsymbol{x} \le \boldsymbol{x}_8 &: \quad \boldsymbol{b}(\boldsymbol{x})=2, \\ \boldsymbol{x}_8 < \boldsymbol{x} &: \quad \boldsymbol{b}(\boldsymbol{x})=3, \end{aligned}$$

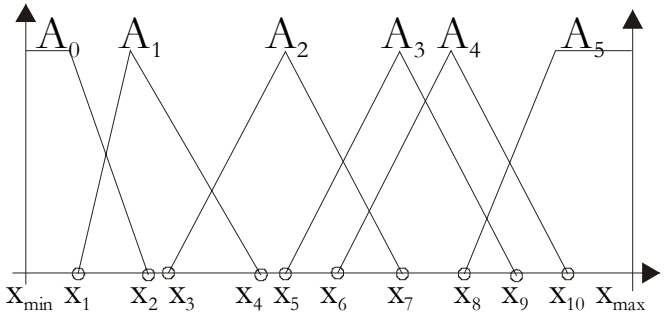


Fig. 6.1 - Ejemplo de funciones de pertenencia

La definición de $\boldsymbol{b}$ y $\boldsymbol{s}$ plantea un problema de optimización: las funciones de pertenencia deben ser numeradas de tal forma que minimice el valor de $\boldsymbol{s}$. Esto puede plantear un problema difícil; en varias aplicaciones prácticas una solución óptima obvia puede ser deducida por la inspección visual de las funciones de pertenencia.

La ruta de datos incluye un bloque MBF (Fig. 6.2) que calcula o almacena los valores de las funciones de pertenencia. El microprograma de control debe incluir las siguientes instrucciones:

$$\begin{aligned} \boldsymbol{y}_{(0)} &\leftarrow \boldsymbol{A}_0(\boldsymbol{x}); \\ \boldsymbol{y}_{(1)} &\leftarrow \boldsymbol{A}_1(\boldsymbol{x}); \\ &\ldots\ldots \\ \boldsymbol{y}_{(n-1)} &\leftarrow \boldsymbol{A}_{n-1}(\boldsymbol{x}); \end{aligned}$$

donde $\boldsymbol{y}_{(0)}, \boldsymbol{y}_{(1)}, \ldots, \boldsymbol{y}_{(n-1)}$ son registros de la ruta de datos.

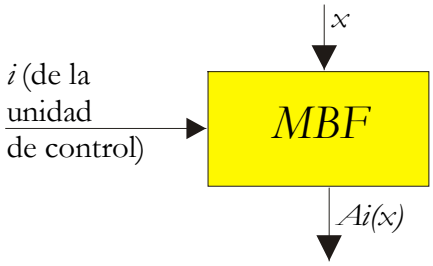


Fig. 6.2 - Bloque de cálculo de funciones de pertenencia

Agregando un sumador y una tabla conteniendo los valores de ***b*** (Fig. 6.3), el número total de microinstrucciones puede ser reducido:

$$y_{(0)} \leftarrow A_{b(x)}(x), \qquad z_{(0)} \leftarrow b(x);$$

$$y_{(1)} \leftarrow A_{b(x)+1}(x), \qquad z_{(1)} \leftarrow b(x) + 1;$$

$$\bullet\bullet\bullet\bullet\bullet$$

$$y_{(s-1)} \leftarrow A_{b(x)+s-1}(x), \qquad z_{(s-1)} \leftarrow b(x) + s - 1;$$

los registros de la ruta de datos $z_{(0)}, z_{(1)}, \ldots, z_{(s-1)}$, almacenan los índices de las funciones de pertenencia correspondientes a los valores de pertenencia almacenados en $y_{(0)}, y_{(1)}, \ldots, y_{(s-1)}$.

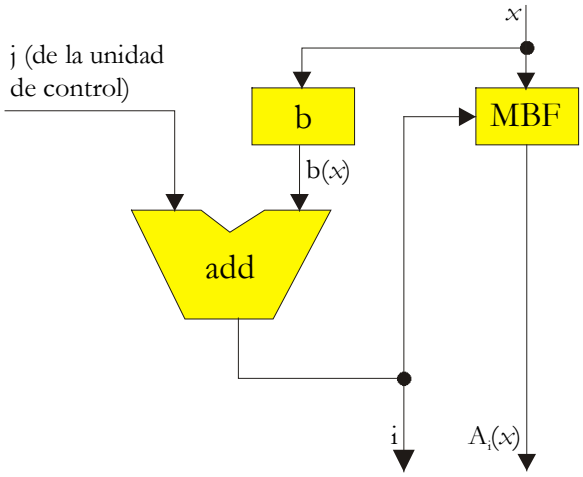


Fig. 6.3 - Cálculo de funciones de pertenencia no nulas

Considerando el caso general, donde hay ***m*** variables de entrada. Los cálculos previos pueden ser realizados para cada variable $x_0, x_1, \ldots, x_{m-1}$ a los que corresponde las funciones: $b_0(x_0)$, $b_1(x_1)$, ... $b_{m-1}(x_{m-1})$, y los valores

constantes s_0, s_1, ... , s_{m-1}. Un ejemplo de tal diseño se implementa y se muestra en la Fig. 6.4.

Una vez que los valores:

$$y_1(0),\ y_1(1),\ y_1(2),\ \ldots\ , y_1(s_1\text{-}1),\ z_1(0),\ z_1(1),\ \ldots\ ,\ z_1(s_1\text{-}1),$$

$$y_2(0),\ y_2(1),\ y_2(2),\ \ldots\ ,\ y_2(s_2\text{-}1),\ z_2(0),\ z_2(1),\ \ldots\ ,\ z_2(s_2\text{-}1),$$

$$\ldots\ldots$$

$$y_m(0),\ y_m(1),\ y_m(2),\ \ldots\ ,\ y_m(s_m\text{-}1),\ z_m(0),\ z_m(1),\ \ldots\ ,\ z_m(s_m\text{-}1),$$

han sido calculados y almacenados, los pesos asociados con las funciones de decodificación pueden ser calculados. Para tal propósito la siguiente expresión reticular:

$$y_1(\,i_1\,) \wedge y_2(\,i_2\,) \wedge \ldots. \wedge y_m(\,i_m\,),$$

debe ser calculada para cada combinación de (i_1, i_2, ..., i_m) de los índices pertenecientes a:

$$\{0, 1, \ldots , s_1\text{-}1\} \text{ x } \{0, 1, \ldots , s_2\text{-}1\} \text{ x } \ldots \text{ x } \{0, 1, \ldots , s_m\text{-}1\}.$$

Por otro lado, debe obtenerse el índice k del consecuente de la regla:

$$\textbf{IF } x_1 \textbf{ is } A_{1\ z1(i1)} \text{ and } \ldots \text{ and } x_m \textbf{ is } A_{m\ zm(im)} \textbf{ THEN } f \text{ is } C_k.$$

El conjunto de reglas puede ser almacenado en una memoria difusa asociativa (FAM, *Fuzzy Associative Memory* [Chu96]). Se supone que el banco de registros v está asociado con las salidas de las funciones de decodificación; entonces, los pesos calculados previamente deben ser acumulados (operación máximo) con el contenido del registro k de v:

$$v(\,k\,) \leftarrow v(\,k\,) \vee (\,y_1(i_1) \wedge y_2(i_2) \wedge \ldots. \wedge y_m(i_m)\,).$$

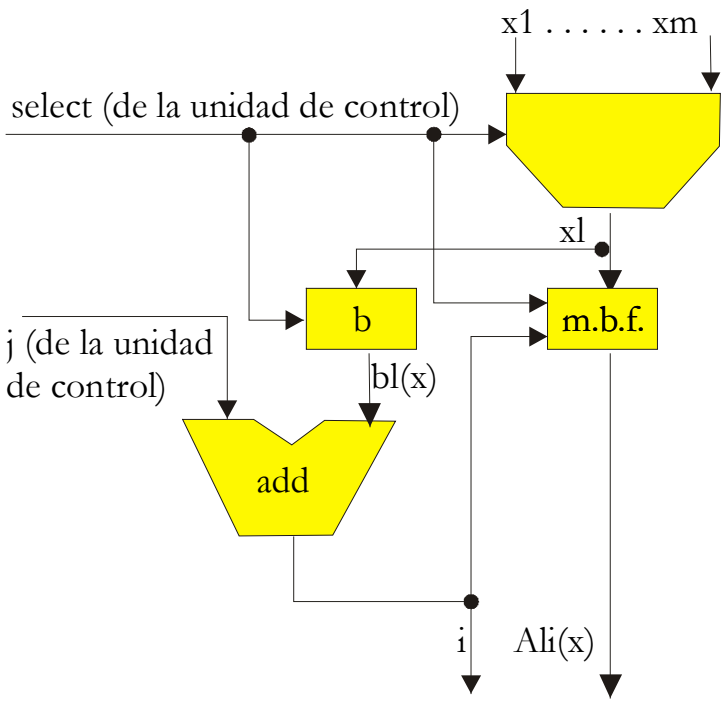


Fig. 6.4 - Cálculo de funciones de pertenencia (con m variables de entrada)

Resumiendo, después del cálculo de todos los valores fusificados se deben realizar las siguientes operaciones:

```
v(0) ←0;
v(1) ←0;
... ;
v(m-1) ←0;
PARA  cada_combinación_de_índices (i1, i2, ...., im)  REPETIR
   v(FAM(z1(i1), ...., zm(im))) ← v(FAM(z1(i1), ...., zm(im))) ∨
                                  ∨ (y1(i1) ∧ y2(i2) ∧ .... ∧ ym(im));
FIN  REPETIR;
```

Un posible circuito es mostrado en la Fig. 6.5.

Después del cálculo de los pesos $v(0)$, $v(1)$, $v(2)$, ... , falta calcular:

$$f = [\, B_0(v(0)) + B_1(v(1)) + ... \,] / [\, v(0) + v(1) + ... \,],$$

donde B_0, B_1, ... son las funciones de decodificación. El circuito de la Fig. 6.5 puede completarse con el circuito de la Fig. 6.6.

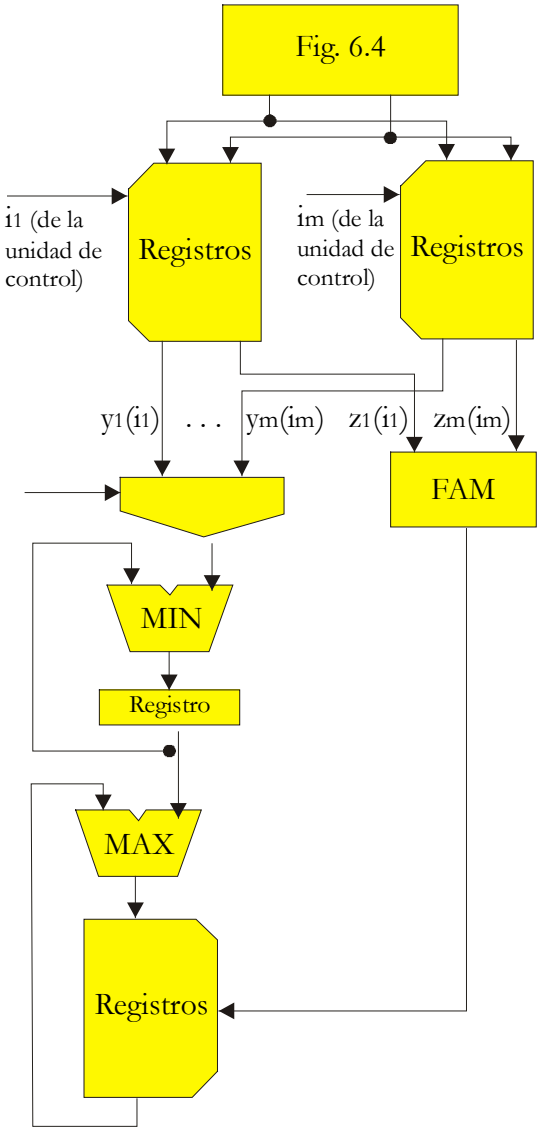


Fig. 6.5 - Cálculo de las reglas de inferencia

El correspondiente microprograma contiene:

- $s_1 + s_2 + ... + s_m$ microinstrucciones para el cálculo de los valores de las funciones de pertenencia.

- $(m * s_1 * s_2 * .... * s_m) + p$ microinstrucciones para realizar las operaciones *min* y *max* (donde p representa el número de funciones de pertenencia de salida).

Por otro lado, deben añadirse las instrucciones para el cálculo del cociente N/D.

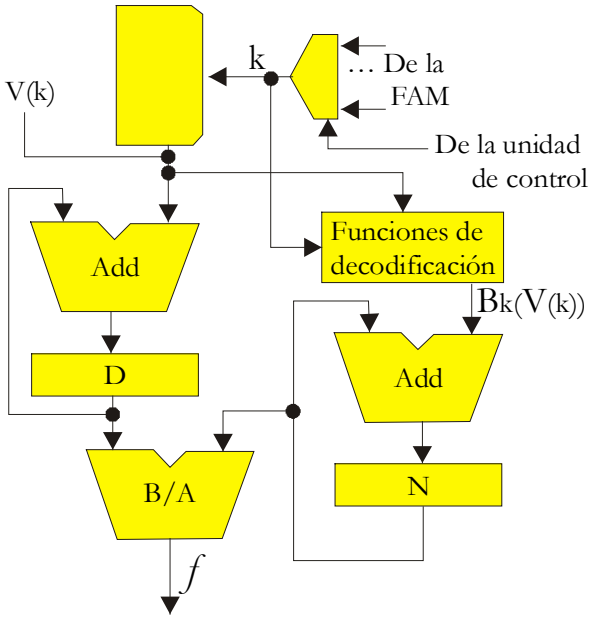


Fig. 6.6 - Cálculo de la función de salida

6.3 - ALGORITMO DE DIVISIÓN

Para la materialización completa del algoritmo difuso, se realiza un circuito dedicado al cálculo de la operación división. Dicho circuito realiza la división de dos números, el dividendo ***N*** y el divisor ***D***. Esta operación genera como resultado un número en binario puro con tantos bits de precisión como etapas tenga el divisor, donde el tamaño del dividendo se encuentra limitado a la suma de los tamaños del divisor y el cociente. El circuito de la Fig. 6.7 materializa el algoritmo de división con restauración.

Cada bit del divisor, antes de llegar al ***full-adder*** es pasado a través de una puerta XOR, usada para hacer su complemento en forma programable. Los 8 bits más significativos de los 15 del dividendo (***N***[14:7]) son alineados con los 8 del complemento del divisor (***D***[7:0]), para alimentar el ***full-adder***[7:0] en la parte superior del circuito. El cable de carry de entrada del ***full-adder*** es conectado a nivel lógico alto, así la resta es realizada como una suma en complemento a dos.

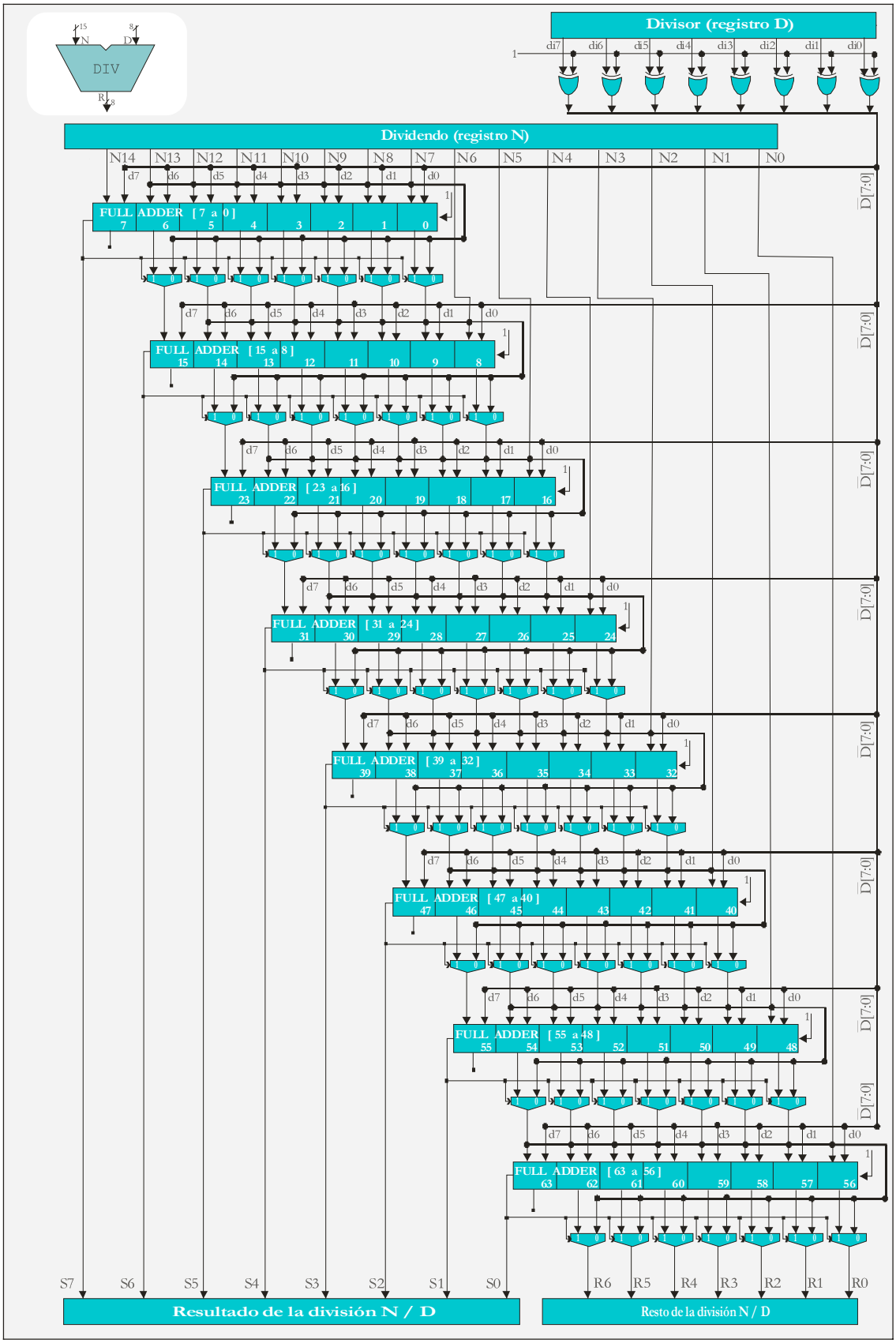


Fig. 6.7 – Circuito de división

Cada bit de salida del ***full-adder*** es conectado a un multiplexor, usado para realizar la restauración. La conexión que realiza el control de la restauración es el bit de ***carry*** de salida del ***full-adder***. En este caso si el ***carry*** de salida genera un valor lógico alto la resta puede ser realizada, en caso contrario se realiza la restauración. Un ***carry*** generado en esta suma de complemento a dos equivale a un "*no-borrow*" generado en una resta del divisor del dividendo. Así en función del signo del resultado de la resta se calcula ($2 \bullet \boldsymbol{r} + \boldsymbol{d}$) ó ($2 \bullet \boldsymbol{r} - \boldsymbol{d}$).

Los dígitos del ***carry*** generados en este proceso de división, con restauración, son directamente los dígitos del cociente. La salida de los multiplexores con los restos parciales son enviados al ***full-adder***[15:8] en el segundo nivel, junto con el siguiente dígito del dividendo (***N***[6]). En la parte inferior del circuito (Fig. 6.7) el resto es generado por el ***full-adder***[63:56].

6.4 - ALGORITMO PROPUESTO

El "*banco de reglas*" o "*base de reglas*" es frecuentemente denominado en la literatura como FAM (por *Fuzzy Associative Memory*). La asociación difusa del par (***A***, ***B***) vincula la salida del conjunto difuso ***B*** (del valor de control) con la entrada del conjunto difuso ***A*** (de los valores de entrada), donde el par representa la asociación como antecedente – consecuente de una cláusula *IF-THEN*.

Cada valor de una variable de entrada es fusificado mediante la aplicación de las funciones de pertenencia. Esos valores de salida son diferentes de cero sólo para unos pocos conjuntos difusos. Por otro lado, sólo los valores diferentes de cero disparan reglas de la FAM, de modo que la mayoría de los consecuentes de salida estarán vacíos. Por la aplicación de este principio, es posible desarrollar una arquitectura que sólo calcule los consecuentes de las reglas disparadas (las no vacías). Con este principio se logra reducir drásticamente el número total de operaciones, y consecuentemente, el tiempo de cálculo.

La materialización propuesta se compone utilizando los circuitos cuyos diagramas se han mostrado en, la Fig. 6.4 para el cálculo de las funciones de pertenencia, la Fig. 6.5 para el cálculo que realiza el motor de inferencia (más detallada en la Fig. 6.6), y la Fig. 6.7 el circuito de división. En esta sección se describe detalladamente el motor de inferencia y el circuito de defusificación.

En el ejemplo de B. Kosko, para estacionar un camión en un muelle de carga presentado en la sección 3.7.1 [Kos92], el sistema usa un máximo de cuatro reglas al mismo tiempo.

Se introduce la siguiente notación:

- ***n*** y ***m*** representan el número de variables de entrada y salida respectivamente.
- ***i*** y ***k*** identifican las variables de entrada y salida respectivamente.
- ***p*** y ***r*** representan el número de funciones de pertenencia y funciones de decodificación respectivamente.
- ***j*** es la identificación de una función de pertenencia de una variable de entrada.
- ***q*** es el número máximo de reglas simultáneamente activadas (por ciclo).

Este algoritmo trabaja con una única señal de ***RESET*** y varios registros o bancos de registros distribuidos en el diseño. Los bancos de registros son: $\boldsymbol{REG}_1$, $\boldsymbol{REG}_2$, ..., $\boldsymbol{REG}_n$, $\boldsymbol{IDF}_1$, $\boldsymbol{IDF}_2$, ..., $\boldsymbol{IDF}_n$, $\boldsymbol{SOP}_1$, $\boldsymbol{SOP}_2$, ..., $\boldsymbol{SOP}_m$; mientras que ***ADF***, ***N***, y ***D*** son los registros. Nótese que el ancho de estos registros depende de varios factores: $\boldsymbol{REG}_i$ de la precisión de las funciones de pertenencia; $\boldsymbol{IDF}_i$ de la cantidad de funciones de pertenencia; ***ADF*** de la cantidad de funciones de decodificación; $\boldsymbol{SOP}_i$ de la precisión de las funciones de pertenencia; ***N*** del tamaño y cantidad de funciones de decodificación (con la operación de

multiplicación incluida); y por último ***D*** de la precisión de las funciones de pertenencia y la cantidad de funciones de decodificación. El algoritmo de cálculo de las f_k funciones de salida es:

```
∀i:0 ≤ i ≤ n-1, ∀j:0 ≤ j ≤ p_i-1:
     IF A_ij(x_i) ≠ 0 THEN
          REG_i[ ptr_i ] = A_ij( x_i );
          IDF_i[ ptr_i ] = A_IDF(i j)( x_i );
Para todo k,
Para toda combinación posible de
                 (ind_1, ind_2, ..., ind_n),
donde ind_i apunta a una de las funciones de pertenencia,
cuyo valor es diferente de cero, de x_i; y
Para cada función de decodificación con identificador
adf:
ADF = FAM_k (       IDF_1[ind_1]+
                    IDF_2[ind_2]+
               ..............  +
                    IDF_n[ind_n]     );
SOP[ ADF ] = max( SOP[ ADF ],
              min(  REG_1[ind_1],
                    REG_2[ind_2],
               ..................,
                    REG_n[ind_n]   )  );
∀ind: 0 ≤ ind ≤ q-1, calcular
      VVV_ind = B( ADF, SOP[ ind ] );
      N_k = N_k + VVV_kh;
      D_k = D_k + SOP_kh;
      F_k = N_k / D_k;
```

6.5 – APLICACIÓN DEL CONTROLADOR PROPUESTO

Esta sección describe el circuito que materializa el algoritmo de control difuso (propuesto por Kosko descrito en 3.7.1) utilizando el motor de inferencia por reglas activas, mientras que se realiza el cálculo de las funciones de pertenencia en paralelo para ambas variables (***x*** y ***phi***), con los bloques ***MF_X*** y ***MF_PHI***. A continuación se describen los bloques y las señales de control que componen la arquitectura. La estructura general del circuito se muestra en la Fig. 6.8.

La primera característica que se destaca del diseño es que no utiliza un único banco de registros, en su lugar se utilizan pequeños bancos de registros configurados a medida de la información a almacenar. Esta

característica simplifica la ruta de datos, reduce el tamaño de la palabra de control y permite ajustar el ancho de palabra al necesario. Los bancos utilizados son: a) $\boldsymbol{REG}_x$ y $\boldsymbol{REG}_\phi$ almacenan los valores de las funciones de pertenencia de las variables de entrada, y los identificadores de las funciones de pertenencia activadas. b) ***SOP*** almacena las salidas del motor de inferencia difusa. c) ***RegIO*** almacena los valores crisp de entrada y salida. d) ***N*** almacena el valor del numerador. e) ***D*** almacena el valor del denominador.

Como segunda característica que se destaca en el diseño es la utilización de múltiples unidades de cálculo configuradas totalmente a medida de las necesidades de la aplicación. Por ejemplo hay varios bloques que realizan la operación sumar. Esto se realiza para mejorar la velocidad de cálculo, permitiendo realizar operaciones de acuerdo al ancho de palabra necesario, y además, se utiliza en gran medida el paralelismo inherente al algoritmo.

Entre los bloques que forman el circuito están: a) ***MF***(x) y ***MF***(ϕ) son circuitos de cálculo (o bloques de memoria) de las funciones de pertenencia de cada variable de entrada, mientras que ***MF***(x) es también usado para el cálculo de las funciones de defusificación. b) ***MIN*** y ***MAX*** calculan los valores mínimos y máximos necesarios para realizar la evaluación del conjunto de reglas. c) ***FAM_theta*** es un bloque de memoria que almacena el conjunto de reglas a aplicar por el algoritmo, cuyo direccionamiento se realiza en función de los identificadores de las funciones de pertenencia de cada variable. d) ***SumaD*** y ***SumaN*** se utilizan para realizar las operaciones de suma, almacenando los resultados en los acumuladores de diferentes anchos ***D*** y ***N*** respectivamente. e) ***Divisor*** es el circuito que realiza el cálculo de la división entre ***D*** y ***N***.

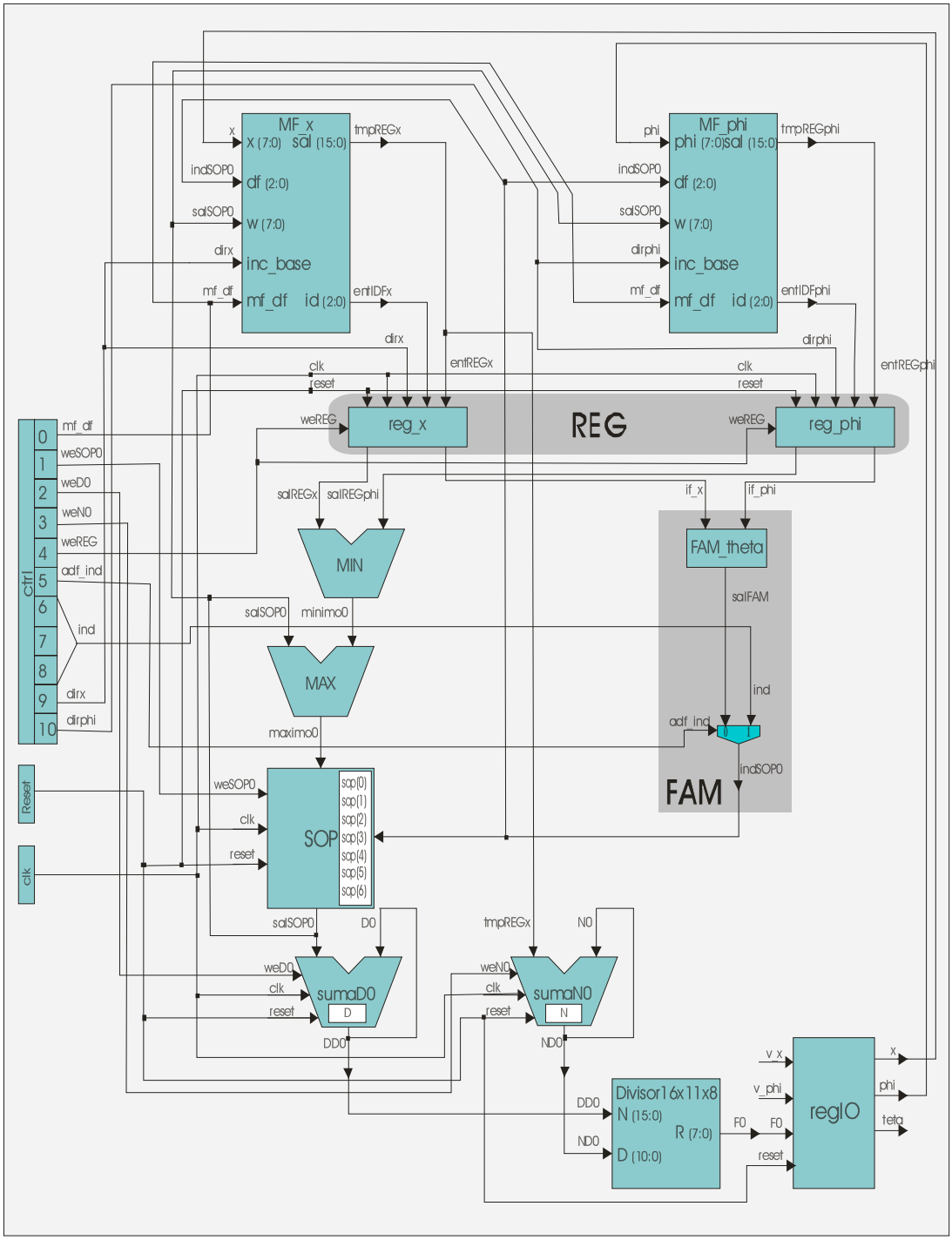


Fig. 6.8 - Diagrama de la arquitectura del ejemplo de Kosko

Cabe destacar además que cada variable de salida debe contar con sus propios bloques: ***MAX***, ***MIN***, ***SumaD***, ***SumaN***, ***FAM***, ***Divisor*** y ***SOP***. Esta característica permite lograr un alto desempeño, ya que todas las salidas del FLC son también calculadas en paralelo.

Por último, completan el diagrama de la arquitectura dirigida por

reglas las señales de control. Estas señales determinan la zona del circuito activa y forman parte del microprograma que controla el circuito. Las señales de control son: a) ***mf_df*** determina si los módulos ***MF_x*** y ***MF_phi*** deben realizar una fusificación o defusificación. b) ***weSOP*** determina si se puede escribir en el registro ***SOP***. c) ***weD0*** determina si se puede escribir en el registro ***D***. d) ***weN*** determina si se puede escribir en el registro ***N***. e) ***weREG*** determina si se puede escribir en el registro ***REG***. f) ***IND*** contiene la dirección del banco de registros ***SOP*** durante la etapa de defusificación. g) ***ADF_IND*** selecciona la dirección para el registro ***SOP***, por medio de ***IND*** durante la defusificación y utilizando ***FAM_theta*** durante la inferencia. h) ***dirX*** y ***dirPHI*** son las direcciones para acceder a los bloques de memoria ***REGx*** y ***REGphi***. Nótese que las señales de ***RELOJ*** y ***RESET*** no se muestran en el diagrama de la arquitectura.

6.6 - DESCRIPCIÓN POLINÓMICA LINEAL

En el capítulo 2 se describieron los bloques principales que forman parte de un controlador difuso. Entre ellos se menciona el proceso de fusificación y defusificación de las variables de entrada y salida, respectivamente.

La fusificación es un proceso de traducción de una variable *crisp* en difusa; proceso que se logra aplicando una función de pertenencia al valor de cada variable de entrada. La defusificación es un proceso de traducción de una variable difusa a su correspondiente valor *crisp*; proceso que se ha simplificado en gran medida al representar las funciones de pertenencia por un único valor (*singleton*). En esta sección se propone una técnica para diseñar los circuitos que realizan el cálculo de funciones de pertenencia genéricas.

Cada función de pertenencia puede ser implementada por una aproximación polinómica lineal. Un ejemplo puede verse en la Fig. 6.9.

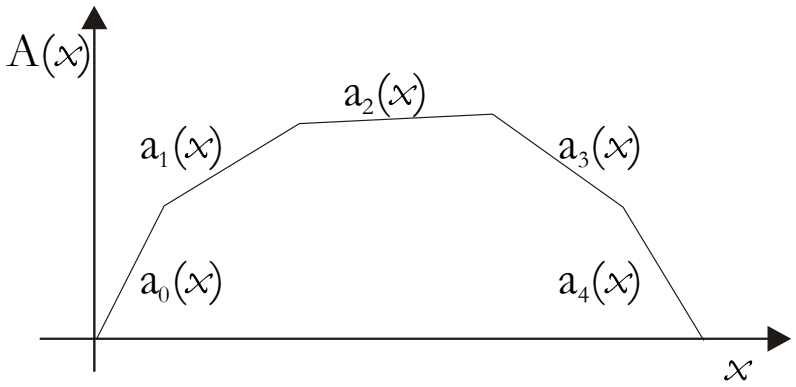


Fig. 6.9 - Aproximación lineal polinómica

Cada función ($A(x)$ en la Fig. 6.9) puede ser sustituida por varios segmentos de funciones lineales ($a_1(x)$ a $a_5(x)$ en la Fig. 6.9). Uno de ellos $a_2(x)$ es definido en la Fig. 6.10, como:

IF ($x_0 \leq x \leq x_1$) **THEN** $a_2(x) = y_0 + k * (x - x_0)$
ELSE $a_2(x) = 0$,
donde $k = (y_1 - y_0) / (x_1 - x_0)$.

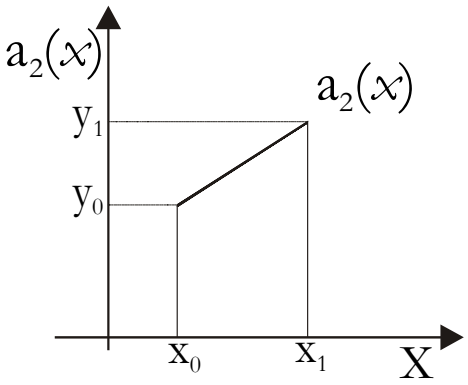


Fig. 6.10 - Función segmento

Se propone el siguiente método: el conjunto de *n* funciones de pertenencia $\{A_0, A_1, A_2, \ldots, A_{n-1}\}$ es sustituido por el conjunto de funciones segmento $\{a_0, a_1, \ldots, a_{r-1}\}$. Se define una función para aplicar las reglas de inferencia:

$$c : \{0, 1, ..., r-1\} \rightarrow \{0, 1, ..., n-1\};$$

la cual asocia con todos los índices de segmentos el correspondiente índice de la función de pertenencia.

Un posible circuito es el mostrado en la Fig. 6.11. El correspondiente microprograma es:

$$y_{(0)} \leftarrow a_0(x), \quad z_{(0)} \leftarrow c(0);$$

$$y_{(1)} \leftarrow a_1(x), \qquad z_{(1)} \leftarrow c(1);$$

. . . . . .

$$y_{(r-1)} \leftarrow a_{r-1}(x), \qquad z_{(r-1)} \leftarrow c(r-1);$$

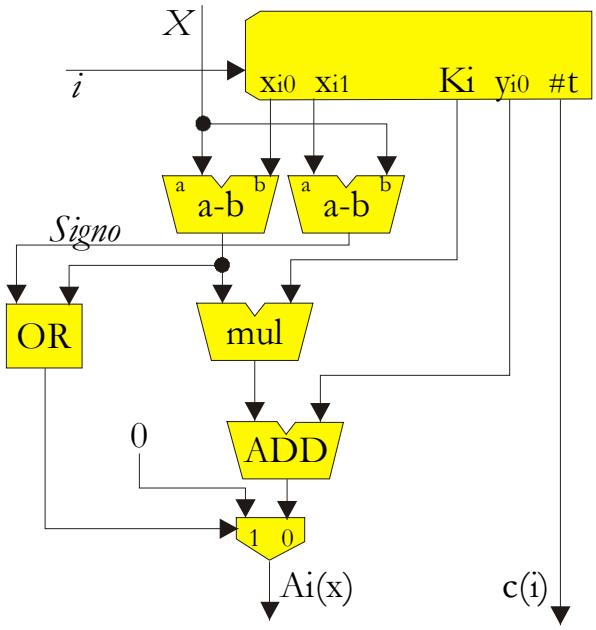


Fig. 6.11 - Cálculo de la función polinómica lineal

6.7 - POLINOMIO LINEAL Y REGLAS ACTIVAS

Suponga que todas las funciones de pertenencia $A_0, A_1, A_2, \ldots, A_{n-1}$, de la variable de entrada x han sido sustituidas por las funciones del polinomio lineal $a_0, a_1, \ldots, a_{r-1}$, y que puede hallarse una función:

$$b : L \rightarrow \{ 0, 1, \ldots, r\text{-}1 \},$$

y una constante s tal que:

$$\boldsymbol{a}_0(\boldsymbol{x}) = \boldsymbol{a}_1(\boldsymbol{x}) = \ldots = \boldsymbol{a}_{b(x)-1}(\boldsymbol{x}) = 0,$$

$$\boldsymbol{a}_{b(x)+s}(\boldsymbol{x}) = \boldsymbol{a}_{b(x)+s+1}(\boldsymbol{x}) = \ldots = \boldsymbol{a}_{r-1}(\boldsymbol{x}) = 0,$$

Así, las etapas de fusificación e inferencia por reglas pueden ser implementadas por el circuito de la Fig. 6.4 en el cual el bloque MBF es el circuito de la Fig. 6.11 con una señal adicional de entrada *select* al decodificador de direcciones de la memoria. Obsérvese que la salida $\boldsymbol{i}$ de la Fig. 6.4 debe ser reemplazada por $\boldsymbol{c}(\boldsymbol{i})$. El microprograma correspondiente es:

$$\boldsymbol{y}_{(0)} \leftarrow \boldsymbol{a}_{b(x)}(\boldsymbol{x}), \qquad \boldsymbol{z}_{(0)} \leftarrow \boldsymbol{c}(\boldsymbol{b}(\boldsymbol{x}));$$

$$\boldsymbol{y}_{(1)} \leftarrow \boldsymbol{a}_{b(x)+1}(\boldsymbol{x}), \qquad \boldsymbol{z}_{(1)} \leftarrow \boldsymbol{c}(\boldsymbol{b}(\boldsymbol{x}) + 1);$$

. . . . . .

$$\boldsymbol{y}_{(s-1)} \leftarrow \boldsymbol{a}_{b(x)+s-1}(\boldsymbol{x}), \qquad \boldsymbol{z}_{(s-1)} \leftarrow \boldsymbol{c}(\boldsymbol{b}(\boldsymbol{x}) + \boldsymbol{s}\text{-}1);$$

Si los segmentos pueden ser numerados de tal forma que:

$$\boldsymbol{a}_i(\boldsymbol{x}) \neq 0,$$

$$\forall \boldsymbol{x} \text{ y } \forall \boldsymbol{i}, \text{ tal que } \boldsymbol{b}(\boldsymbol{x}) \leq \boldsymbol{i} \leq \boldsymbol{b}(\boldsymbol{x}) + \boldsymbol{s} - 1,$$

no se hace necesaria la verificación de rango aplicada por la condición:

$$\boldsymbol{x}_{i0} \leq \boldsymbol{x} \leq \boldsymbol{x}_{i1}.$$

El circuito de la Fig. 6.11 puede ser simplificado ya que uno de los circuitos encargados de realizar la resta ($\boldsymbol{x}_{i1} - \boldsymbol{x}$), la puerta *OR* y el *multiplexor* de salida pueden ser eliminados.

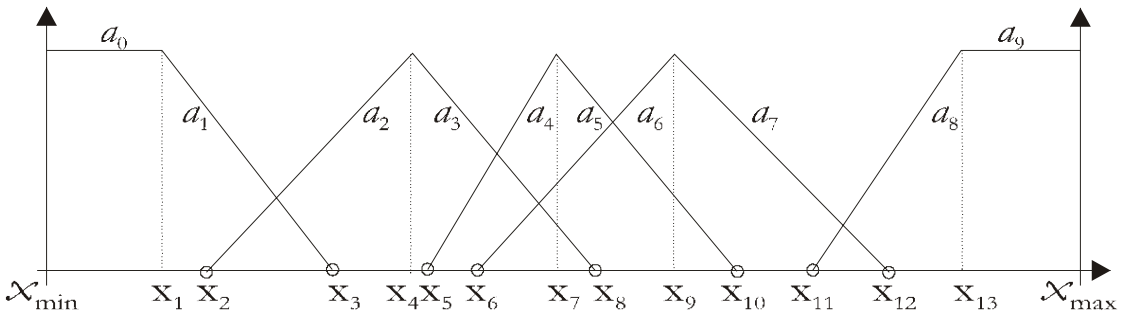


Fig. 6.12 - Funciones segmento: primer ejemplo

Un primer ejemplo puede verse en la Fig. 6.12. Los resultados del análisis de las funciones segmento son resumidos en la Tabla 6.2.

	Segmentos no - cero
$x \le x_1$	a_0
$x_1 < x \le x_2$	a_1
$x_2 < x \le x_3$	$a_1\ a_2$
$x_3 < x \le x_4$	a_2
$x_4 < x \le x_5$	$a_2\ \ a_3$
$x_5 < x \le x_6$	$a_3\ a_4$
$x_6 < x \le x_7$	$a_3\ a_4\ a_5$
$x_7 < x \le x_8$	$a_4\ a_5\ a_6$
$x_8 < x \le x_9$	$a_5\ a_6$
$x_9 < x \le x_{10}$	$a_6\ a_7$
$x_{10} < x \le x_{11}$	a_7
$x_{11} < x \le x_{12}$	$a_7\ a_8$
$x_{12} < x \le x_{13}$	a_8
$x_{13} < x$	a_9

Tabla 6.2 - Funciones segmento del primer ejemplo

En este caso, $s = 3$ y b pueden ser definidos de la siguiente forma:

$x < x_4$: $b(x)=0$,
$x_4 < x \le x_6$: $b(x)=2$,
$x_6 < x \le x_7$: $b(x)=3$,
$x_7 < x \le x_8$: $b(x)=4$,
$x_8 < x \le x_{11}$: $b(x)=5$,
$x_{11} < x$: $b(x)=7$.

Obsérvese que la condición de verificación de rango (3) no es satisfecha.

Otro ejemplo es representado en la Fig. 6.13. En este caso, $s = 2$, y b es definido como:

$x \le x_1$: $b(x)=0$,
$x_1 < x \le x_2$: $b(x)=2$,
$x_2 < x$: $b(x)=4$,

y la condición es satisfecha.

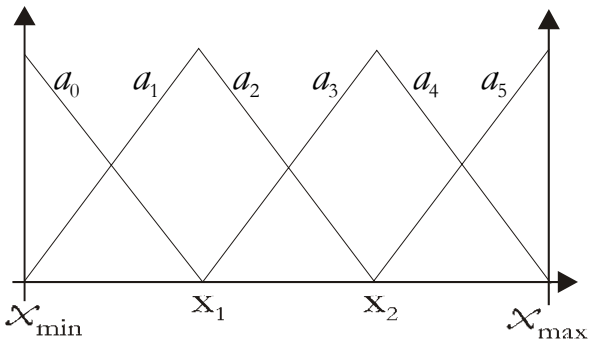


Fig. 6.13 - Funciones segmento, segundo ejemplo

Aún cuando la condición de verificación de rango no se cumple, la versión simplificada del circuito de la Fig. 6.11 puede ser usada si se agregan algunos segmentos inútiles. Como un ejemplo que ilustre tales segmentos, la Fig. 6.12 puede ser numerada como se muestra en la Fig. 6.14. Los segmentos $\boldsymbol{b}_0$, $\boldsymbol{b}_1$, $\boldsymbol{b}_3$, $\boldsymbol{b}_4$, $\boldsymbol{b}_6$, $\boldsymbol{b}_8$, $\boldsymbol{b}_{13}$, $\boldsymbol{b}_{15}$, $\boldsymbol{b}_{17}$, $\boldsymbol{b}_{19}$, y $\boldsymbol{b}_{20}$, son segmentos inútiles iguales a cero. El análisis de los segmentos de la Fig. 6.14 produce los resultados mostrados por la Tabla 6.3.

	Funciones segmento
$\boldsymbol{x} \le \boldsymbol{x}_1$	$\boldsymbol{b}_0\ \boldsymbol{b}_1\ \underline{\boldsymbol{b}_2}$
$\boldsymbol{x}_1 < \boldsymbol{x} \le \boldsymbol{x}_2$	$\boldsymbol{b}_3\ \boldsymbol{b}_4\ \underline{\boldsymbol{b}_5}$
$\boldsymbol{x}_2 < \boldsymbol{x} \le \boldsymbol{x}_3$	$\underline{\boldsymbol{b}_5}\ \boldsymbol{b}_6\ \boldsymbol{b}_7$
$\boldsymbol{x}_3 < \boldsymbol{x} \le \boldsymbol{x}_4$	$\boldsymbol{b}_6\ \underline{\boldsymbol{b}_7}\ \boldsymbol{b}_8$
$\boldsymbol{x}_4 < \boldsymbol{x} \le \boldsymbol{x}_5$	$\underline{\boldsymbol{b}_7}\ \boldsymbol{b}_8\ \underline{\boldsymbol{b}_9}$
$\boldsymbol{x}_5 < \boldsymbol{x} \le \boldsymbol{x}_6$	$\boldsymbol{b}_8\ \underline{\boldsymbol{b}_9}\ \underline{\boldsymbol{b}_{10}}$
$\boldsymbol{x}_6 < \boldsymbol{x} \le \boldsymbol{x}_7$	$\underline{\boldsymbol{b}_9}\ \underline{\boldsymbol{b}_{10}}\ \underline{\boldsymbol{b}_{11}}$
$\boldsymbol{x}_7 < \boldsymbol{x} \le \boldsymbol{x}_8$	$\underline{\boldsymbol{b}_{10}}\ \underline{\boldsymbol{b}_{11}}\ \underline{\boldsymbol{b}_{12}}$
$\boldsymbol{x}_8 < \boldsymbol{x} \le \boldsymbol{x}_9$	$\underline{\boldsymbol{b}_{11}}\ \underline{\boldsymbol{b}_{12}}\ \boldsymbol{b}_{13}$
$\boldsymbol{x}_9 < \boldsymbol{x} \le \boldsymbol{x}_{10}$	$\underline{\boldsymbol{b}_{12}}\ \boldsymbol{b}_{13}\ \underline{\boldsymbol{b}_{14}}$
$\boldsymbol{x}_{10} < \boldsymbol{x} \le \boldsymbol{x}_{11}$	$\boldsymbol{b}_{13}\ \underline{\boldsymbol{b}_{14}}\ \boldsymbol{b}_{15}$
$\boldsymbol{x}_{11} < \boldsymbol{x} \le \boldsymbol{x}_{12}$	$\underline{\boldsymbol{b}_{14}}\ \boldsymbol{b}_{15}\ \underline{\boldsymbol{b}_{16}}$
$\boldsymbol{x}_{12} < \boldsymbol{x} \le \boldsymbol{x}_{13}$	$\boldsymbol{b}_{15}\ \underline{\boldsymbol{b}_{16}}\ \boldsymbol{b}_{17}$
$\boldsymbol{x}_{13} < \boldsymbol{x}$	$\underline{\boldsymbol{b}_{18}}\ \boldsymbol{b}_{19}\ \boldsymbol{b}_{20}$

Tabla 6.3 - Funciones segmento (con segmentos inútiles)

Los segmentos no inútiles son los subrayados.

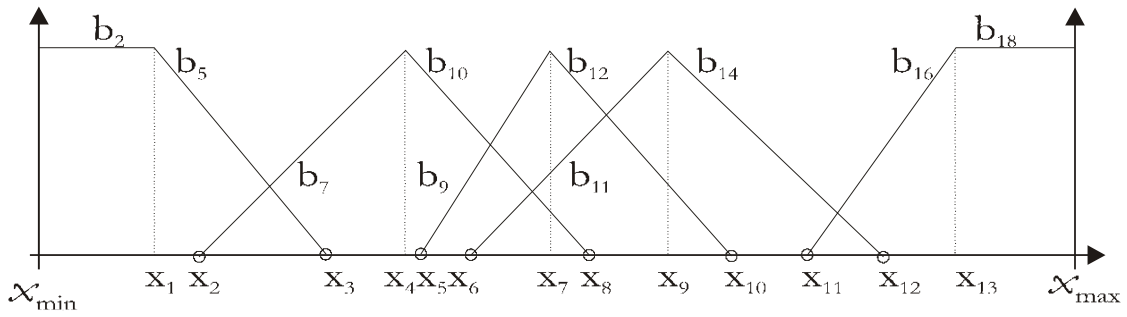


Fig. 6.14 - Agregado de segmentos "inútiles"

$x \le x_1$: $b(x)$=0,
$x_1 < x \le x_2$: $b(x)$=3,
$x_2 < x \le x_3$: $b(x)$=5,
$x_3 < x \le x_4$: $b(x)$=6,
$x_4 < x \le x_5$: $b(x)$=7,
$x_5 < x \le x_6$: $b(x)$=8,
$x_6 < x \le x_7$: $b(x)$=9,
$x_7 < x \le x_8$: $b(x)$=10,
$x_8 < x \le x_9$: $b(x)$=11,
$x_9 < x \le x_{10}$: $b(x)$=12,
$X_{10} < x \le x_{11}$: $b(x)$=13,
$X_{11} < x \le x_{12}$: $b(x)$=14,
$X_{12} < x \le x_{13}$: $b(x)$=15,
$x_{13} < x$: $b(x)$=18.

A cada segmento inútil le corresponden los siguientes valores de parámetros (Fig. 6.11):

$$x_{i0} = \textit{valor\_correspondiente};$$
$$k_{i0} = 0,$$
$$c(i) = \textit{cualquier\_valor}.$$

6.8 - FPS: CIRCUITO DE CÁLCULO

Para demostrar el método de síntesis, se han desarrollado varios ejemplos (descritos en la sección 3.7 y el Apéndice B), uno de ellos es el de B. Kosko [Kos92]. En éste ejemplo, la especificación inicial consta de 12 funciones de pertenencia, 7 funciones de decodificación y 35 reglas (ver Fig. 6.8, Fig. 6.15 y la Tabla 6.4). Las tres variables ϕ, x e y determinan la posición del camión. ϕ especifica el ángulo horizontal del camión. El par (x, y) especifica la posición

central trasera del camión en el plano. θ es el ángulo de las ruedas de dirección.

		X				
		LE	LC	CE	RC	RI
ϕ	RB	PS	PM	PM	PB	PB
	RU	NS	PS	PM	PB	PB
	RV	NM	NS	PS	PM	PB
	VE	NM	NM	ZE	PM	PM
	LV	NB	NM	NS	PS	PM
	LU	NB	NB	NM	NS	PS
	LB	NB	NB	NM	NM	NS

Tabla 6.4 - Reglas del ejemplo de

Una comparación basada en el conjunto de instrucciones del ejemplo del controlador de Kosko, puede verse a continuación:

- Enfoque estándar. El sistema necesita fusificar todas las variables, por lo cual se necesita ejecutar: 5 veces para la variable x, 7 veces para la variable ϕ y 14 veces para la variable de salida θ (7 accesos de doble precisión). Así, el controlador necesita de 26 instrucciones.

- Enfoque por reglas activas. El sistema necesita fusificar todas las variables de entrada: 5 para x y 7 para ϕ. Además, sólo los valores difusos de salida diferentes de cero son necesarios, por lo cual se requieren 8 para θ (cuatro funciones de doble precisión). En total el controlador necesita de 20 instrucciones.

- Enfoque con reglas activas y segmentos. El sistema necesita acceder O funciones para las variables de entrada: 2 llamadas para x y 2 para ϕ. Las salidas diferentes de cero del conjunto difuso son 8 para θ (cuatro accesos a funciones de doble precisión). Así el controlador necesita sólo 12 instrucciones.

Descripción del circuito (mostrado en la Fig. 6.15):

- Variables de entrada. Las variables para fusificar son $\boldsymbol{x}$ y $\boldsymbol{\phi}$; $\boldsymbol{w}$ es la salida difusa del motor de inferencia a obtener del centro de gravedad.

- Salidas. Almacenan los identificadores de las funciones de pertenencia cuyo valor es diferente de cero.

- Palabra de control.

    1) ***X_φ***: selecciona la variable de entrada (0:$\boldsymbol{x}$, 1:$\boldsymbol{\phi}$).

    2) ***MF_DF***: selecciona el acceso a la función de pertenencia o función de decodificación.

    3) ***#DF***: el circuito calcula la función de decodificación identificada por ***#DF*** cuando ***MF_DF*** está a nivel lógico alto.

    4) ***INC_BASE***: el circuito calcula la función de pertencia del registro ***BASE*** más el valor del campo ***INC_BASE***.

    5) ***L_H***: permite obtener las partes bajas (0) o altas (1) de las palabras de doble precisión. Los cálculos de simple precisión son retornados por la parte baja.

- Bloques de cálculo.

    1) ***Selector de índice***: selecciona la dirección de ***INDICE-ROM*** comparando el valor de la variable con el primer punto de cada segmento, para determinar cual es el primer segmento que produce un valor no nulo.

    2) ***ADD***, ***SUB***, ***MUL***: estos circuitos realizan las operaciones aritméticas suma, resta y multiplicación respectivamente.

- Bloques de memoria.

  1) ***INDICE-ROM***: contiene la dirección ***BASE*** para acceder a ***ROM-PARAMETROS***.

  2) ***ROM-PARAMETROS***: contiene los parámetros para el cálculo de las funciones. X_0 es el punto de la abscisa del primer segmento, Y_0 es el valor de la función de pertenencia para el punto X_0, ***CONSTANT*** es el ángulo del segmento, y ***#TAB*** es el identificador de la función o tabla a la que pertenece el segmento.

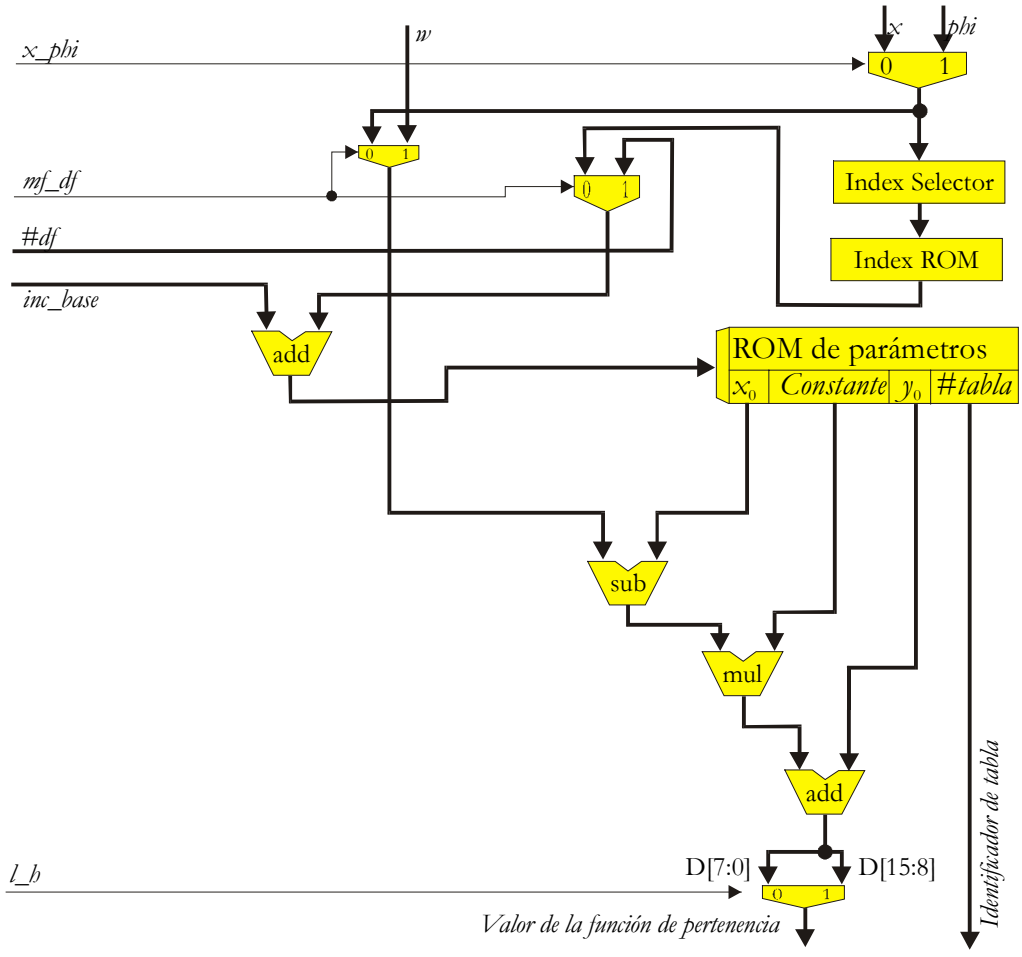


Fig. 6.15 - Circuito de cálculo de las funciones de pertenencia del ejemplo de Kosko

El circuito ***SELECTOR-INDICE*** puede ser sintetizado para trabajar de forma fija o programable a medida de la aplicación:

- Fija. El circuito incluye un módulo específico para comparar las magnitudes de la aplicación. Se utilizan puertas básicas para implementar las funciones *booleanas* que determinan cual segmento incluye a la variable. Este circuito puede ser adecuadamente optimizado para reducir el número de puertas necesarias.

- Programable. El circuito incluye una memoria estándar (RAM o ROM) y un módulo genérico para comparar magnitudes. La principal ventaja es la posibilidad de ajustar los límites de los segmentos en línea, pero si se implementa en un ASIC, se paga un coste superior en área.

6.9 - FUNDAMENTOS DE LA ARQUITECTURA

Características de la arquitectura propuesta:

- Las funciones de pertenencia determinan cual regla se activa, así, el conjunto de reglas que se aplica es determinado por el valor de las variables de entrada. De esta forma se evita calcular reglas que no aporten información.

- Las operaciones usan registros internos de la UAL, cuya dirección es determinada por registros índices automáticos. Esto permite reducir el número de líneas de control de entrada/salida.

- Se usan diferentes UAL para el cálculo de operaciones aritméticas y reticulares.

- Sólo se accede a una parte de las funciones de decodificación.

- Se realizan los cálculos aritméticos y reticulares sólo cuando tienen

un valor significativo.

- El microprograma de control debe ser lineal (sin saltos) para reducir el área de control.

6.10 - RESUMEN

En este capítulo se propone una arquitectura alternativa que asiste durante el diseño de controladores difusos de altas prestaciones. La propuesta basa su diseño en mejorar la etapa de fusificación y el motor de inferencias. Por un lado se usa la aproximación de un polinomio lineal para calcular las funciones de pertenencia durante la fusificación; que es la generalización del enfoque por aproximación por trapecios.

Por el otro lado, en lugar de tratar de obtener las expresiones reticulares optimizadas que implementen las reglas de inferencia, se sigue una estrategia alternativa; se basa en la siguiente observación: dado un conjunto de valores de variables de entrada el número de funciones de pertenencia cuyo valor sea *no nulo* es mucho menor que el total de funciones. El método propuesto consiste en (1) determinar cual función genera un valor no nulo, (2) calcularlo, y (3) implementar sólo las reglas de inferencia seleccionadas por las funciones de pertenencia cuyo valor es no nulo.

Finalmente ambas ideas (la aproximación de funciones de pertenencia por un polinomio lineal y la materialización de la inferencia por reglas activas) son unidas con una posible ruta de datos que termina de definir el controlador. Otra característica es la utilización en el circuito de múltiples unidades de cálculo y varios bloques de memoria distribuidos en todo el diseño. Durante la etapa de defusificación se utiliza un circuito combinacional especializado que realiza la división, y como consecuencia se simplifica la ruta de datos y el microprograma.

7 – HERRAMIENTAS DE LA ARQUITECTURA POR REGLAS ACTIVAS

7.1 - INTRODUCCIÓN

Para la generación de un controlador dirigido por reglas, como el presentado en el capítulo 6, se desarrolla una herramienta genérica (descrita detalladamente en el Apéndice A) que toma una descripción del FLC de un archivo en un lenguaje 'tipo' FIL (presentado en la sección 5.3.1). El conjunto de herramientas se presenta en un sistema con una interfaz única, que permite navegar todas las etapas del proyecto, y asiste al diseñador de FLC desde la definición de la aplicación en FIL, hasta la generación del código VHDL que una vez sintetizado pasará a ser el controlador; pasando por todas las etapas que pueden incluir varios ciclos de simulación / modificación / ajustes. Por otra parte, si bien las herramientas son genéricas, aplicables a cualquier FLC a definir, en este capítulo se mostrará su funcionamiento a través del FLC definido por Kosko [Kos92] y descrito en 3.7.1.

Las herramientas necesarias, si bien se incluyen en un único sistema, constan de cinco herramientas perfectamente diferenciadas por su función:

- Análisis y verificación de los requisitos del FLC definido para que se pueda generar automáticamente.
- Generación del código VHDL que una vez sintetizado, define el circuito del controlador.
- Generación del código VHDL que una vez sintetizado, define el circuito de cálculo de las funciones de pertenencia.

- Generación del microprograma que controla la ruta de datos del controlador.
- Generación del código (C y Delphi), para hacer la simulación de alto nivel del controlador.

En la Fig. 7.1 se muestra el ciclo que aplica el diseñador de controladores difusos utilizando la herramienta propuesta. Para realizar cualquier operación, primero se realiza un análisis del archivo con la descripción del controlador, para validar el diseño desde el punto de vista gramatical y semántico. Una vez que esa etapa ha sido superada, se puede generar el controlador hardware en tres etapas, y el simulador software del controlador.

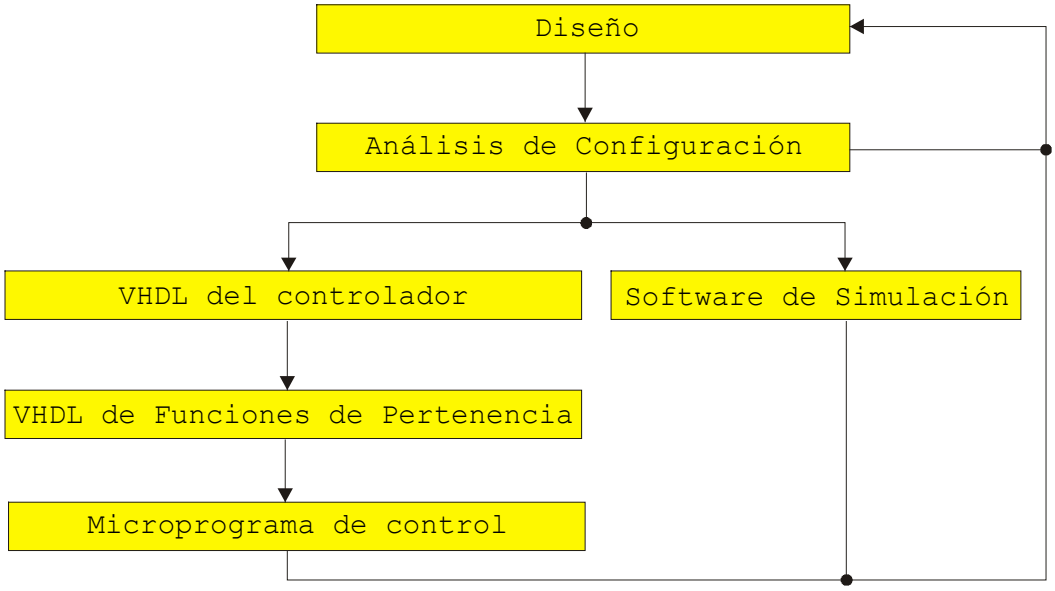


Fig. 7.1 – Ciclo de diseño de controladores difusos

7.2 – ANALIZADOR DE CONFIGURACIÓN

Para que el controlador pueda ser automáticamente generado, el archivo que lo define (tipo FIL, definido en 5.3.1) debe ser analizado para determinar los parámetros del sistema de control difuso. En primer lugar se debe analizar para determinar: a) La cantidad de segmentos que se superponen, b) El tamaño de los buses de las rutas de datos. c) El tamaño de los registros. d) El tamaño y el

contenido de los bloques de memoria. e) La posibilidad de codificación de la posición inicial del segmento.

7.2.1 - CANTIDAD DE SEGMENTOS SUPERPUESTOS

Un parámetro fundamental para la generación automática del motor de inferencia por reglas activas es la cantidad de segmentos que se superponen (el ***O*** definido en la sección 2.8, Fig. 2.6). Este valor se utiliza para determinar la cantidad máxima de reglas que se pueden activar al mismo tiempo (cantidad de variables de entrada elevado a la cantidad de segmentos superpuestos, ***O***).

El algoritmo para realizar este cálculo necesita recorrer todos los dominios de todas las variables (***x*** y ***phi*** en ejemplo de Kosko, descrito en 3.7.1), verificando en cada punto la cantidad de segmentos que cubren el punto. A medida que se recorre el dominio de la variable, se almacena el máximo valor de superposición. A continuación se presenta el algoritmo utilizado para determinar la cantidad de segmentos superpuestos, que lo calcula en la variable ***MÁXIMO***.

```
maximo := 0;
FOR cada variable var DO
  FOR cada valor v del dominio de la variable var DO
    cantidad := 0;
    FOR cada función de pertenencia f de var DO
       IF f(v) <> 0 THEN inc( cantidad );
    IF cantidad > maximo THEN maximo := cantidad;
```

7.2.2 – TAMAÑO DE LA RUTA DE DATOS

Este módulo realiza el análisis del diseño para definir el tamaño de los buses de las rutas de datos. Entre ellos: las variables de entrada (***x*** y ***phi***) y salida (***theta***), el tamaño de palabra interno del motor de inferencia, las señales de control de la ruta de datos, la palabra de dirección de las memorias y el ancho de palabra de las memorias.

Algunos tamaños de palabra no son directamente calculables, por ejemplo los registros acumuladores ***D*** y ***N***, de la Fig. 6.8 dependen tanto del ancho de palabra a acumular como de la cantidad máxima de valores que se deben sumar.

El tamaño de los buses para las variables de entrada y de salida se pueden deducir de su declaración en el lenguaje de especificación del FLC, donde la cantidad de bits necesarios para el bus de una variable es igual a la cantidad de bits necesarios para representar el valor MaxX en binario puro (sentencia INVAR del lenguaje NFIL).

7.2.3 – TAMAÑO DE LOS REGISTROS

Se analiza el diseño y el tamaño de las rutas de datos para poder definir el tamaño de los registros. Se deben destacar los que almacenan los valores de las variables fusificadas y sus identificadores de las funciones de pertenencia, ***SOP*** que almacena los valores durante la inferencia, y por último, ***D*** y ***N*** para calcular la sumatoria de los valores durante la defusificación. Los registros de acumulación (***D*** y ***N***) dependen de las rutas de datos, de la cantidad de segmentos superpuestos y de la cantidad de funciones de pertenencia de salida del diseño.

7.2.4 – BLOQUES DE MEMORIA

Este módulo se utiliza para calcular el tamaño y contenido de los bloques de memoria correspondientes a cada variable. El archivo de especificación del controlador es analizado para generar los parámetros necesarios para realizar el cálculo de las funciones de pertenencia. Por un lado, se genera el selector del primer segmento activo, usando el valor de la variable de entrada se determina cual es el primer segmento que la contiene. Por otro lado, el siguiente segmento es determinado por el identificador del primer segmento.

```
selX : PROCESS ( X )
   BEGIN
      IF (X >  0)and(x < 30) THEN v1 <= "0000"; ---   0;
   ELSIF (X > 30)and(x < 96) THEN v1 <= "0001"; ---   1;
   . . . . . . . . . .
   ELSIF (X >158)and(x <224) THEN v1 <= "1000"; ---   8;
   ELSE                           v1 <= "1001"; ---   9;
   END IF;
   END PROCESS;
```

```
x_ROM_ib1 : PROCESS ( v1 )
   BEGIN
   CASE conv_integer( v1 ) IS
   WHEN 0 => sROMib1 <= "0010";
   WHEN 1 => sROMib1 <= "0010";
   . . . . . .
   WHEN 9 => sROMib1 <= "0111";
   WHEN OTHERS => sROMib1 <= "0000";
   END CASE;
   END PROCESS;
```

Además, se genera la memoria que contiene los parámetros de cada segmento. El primer parámetro ***sROMfp*** se determina con el identificador de la función de pertenencia activada (es el valor que relaciona el segmento a la función que pertenece), el segundo parámetro ***sROMx0*** se determina con la posición inicial del segmento, el tercer parámetro es el valor de ordenada ***sROMy0*** correspondiente al punto ***sROMx0***, el cuarto parámetro ***sROMang*** es la pendiente del segmento, y el último valor es el signo ***sROMsig*** de la pendiente. En este caso, nótese que el valor ***sROMy0*** no está codificado, aunque podría estarlo (descrito en la sección 7.2.5). Las relaciones de estos campos de la ROM con los propuestos en la Fig. 6.16 son las siguientes: ***sROMx0*** $\rightarrow$ X_0, ***sROMy0*** $\rightarrow$ Y_0, ***sROMang*** $\rightarrow$ ***Constante***, ***sROMsig*** $\rightarrow$ signo(***Constante***), y ***sROMfp*** $\rightarrow$ ***#Tabla***.

```
x_ROM_fp : PROCESS ( v2 )
variable t:std_logic_vector(32 downto 0);
   BEGIN
   CASE conv_integer( v2 ) IS
   WHEN   0    => t<="00000000000111111110000000000000";
   WHEN   1    => t<="00000011110111111110001111011011";
   . . . . . .
   WHEN   9    => t<="10011100000111111110000000000000";
   WHEN OTHERS => t<="00000000000000000000000000000000";
   END CASE;
   sROMfp  <= t(32 downto 30);
   sROMx0  <= t(29 downto 22);
   sROMy0  <= t(21 downto 14);
   sROMang <= t(13 downto 1);
   sROMsig <= t(0);
   END PROCESS;
```

Por último, se genera la memoria que contiene los pesos de las funciones de defusificación. De acuerdo con el identificador, se determina cual es el peso de la función de decodificación.

```
rom_fd : PROCESS( df )
   BEGIN
   CASE conv_integer( df ) IS
   WHEN      0 => sROMfd<="00001100";  -- 12
   . . . . . . . . .
   WHEN      6 => sROMfd<="11110010";  -- 242
   WHEN OTHERS => sROMfd<="00000000";
   END CASE;
   END PROCESS;
```

7.2.5 – CODIFICACIÓN DE POSICIÓN

Este módulo es el encargado de analizar si es posible codificar el parámetro que determina la posición inicial ***Y*** del segmento. En caso que todos los segmentos comiencen o terminen en el máximo (2^n-1, donde ***n*** es el número de bits con los que se representa el número) o mínimo valor (0) de la función. La codificación de este valor con sólo un bit es posible debido a que en una arquitectura con ***n*** = 8 bits, 0 determina 00000000, mientras que 1 representa 11111111, así en lugar de almacenar los 8 ceros u 8 unos, sólo es necesario almacenar un bit (que debe ser replicado).

En caso de que haya segmentos cuyo máximo o mínimo valor ***Y*** no llegue a los límites, es imposible codificar, y se deben almacenar todas las posiciones.

7.3 – MOTOR DE INFERENCIA

En esta sección se analiza el circuito generado por la herramienta de asistencia al diseñador de FLC (Apéndice A). El sistema, a partir del archivo de especificación escrito en FIL, genera el código de definición del módulo principal conectando todos los módulos que forman el controlador por medio de una lista de conexiones VHDL (*net list*).

Para evitar que el resultado del diseño del motor de inferencia se vea afectado por la técnica de fusificación empleada, se utiliza un bloque fusificador para cada variable. En esta sección se analiza el circuito generado basándose en el ejemplo de Kosko [Kos92] presentado en la sección 3.7.1, donde los módulos ***MF_X*** y

MF_PHI son los fusificadores para las variables ***X*** y ***PHI***, respectivamente. El fusificador por segmentos se presenta en la sección 7.4, donde si bien se muestra el circuito correspondiente a la variable ***X***, los de ambas variables trabajan en paralelo y con el mismo principio de funcionamiento.

El motor de inferencia es definido en función de sus 9 módulos VHDL (ver Fig. 7.2[1]): ***REG***, ***MIN***, ***MAX***, ***FAM***, ***SOP***, ***SUMA_D***, ***SUMA_N, RegIO*** y ***DIVISOR***. Estos módulos se conectan a los de fusificación, que por diseño hay uno por cada variable de entrada. Exceptuando los bloques ***REG*** y ***RegIO***, de todos los demás se implementa uno por cada variable de salida del controlador. A continuación se describen en detalle dichos bloques.

7.3.1 – BLOQUE: 'REG'

El bloque **REG** define los registros de almacenamiento de los valores de la fusificación y de identificación de las funciones de pertenencia activadas, para cada variable de entrada. Este banco de registros, tiene como señales de control: ***WE_REG*** para activar los registros para la escritura, ***RESET*** que permite realizar la puesta a cero asincrónica de todos los elementos de memoria, ***CLK*** es la señal de reloj cuyo flanco ascendente es utilizado para almacenar los valores, ***DIR_X*** y ***DIR_PHI*** direccionan los registros de cada variable por separado.

Debido a que el tamaño del código VHDL completo es muy extenso, sólo se muestra el código correspondiente a la variable ***X***, mientras que el código de la variable ***PHI*** es similar. El bloque de memoria ***RegistroX*** almacena los valores difusos de las variables fusificadas y el código de los registros que identifican las funciones de pertenencia activadas.

[1] Note que la Fig. 7.2 es la misma que la Fig. 6.8, se muestra nuevamente para facilitar el análisis al lector.

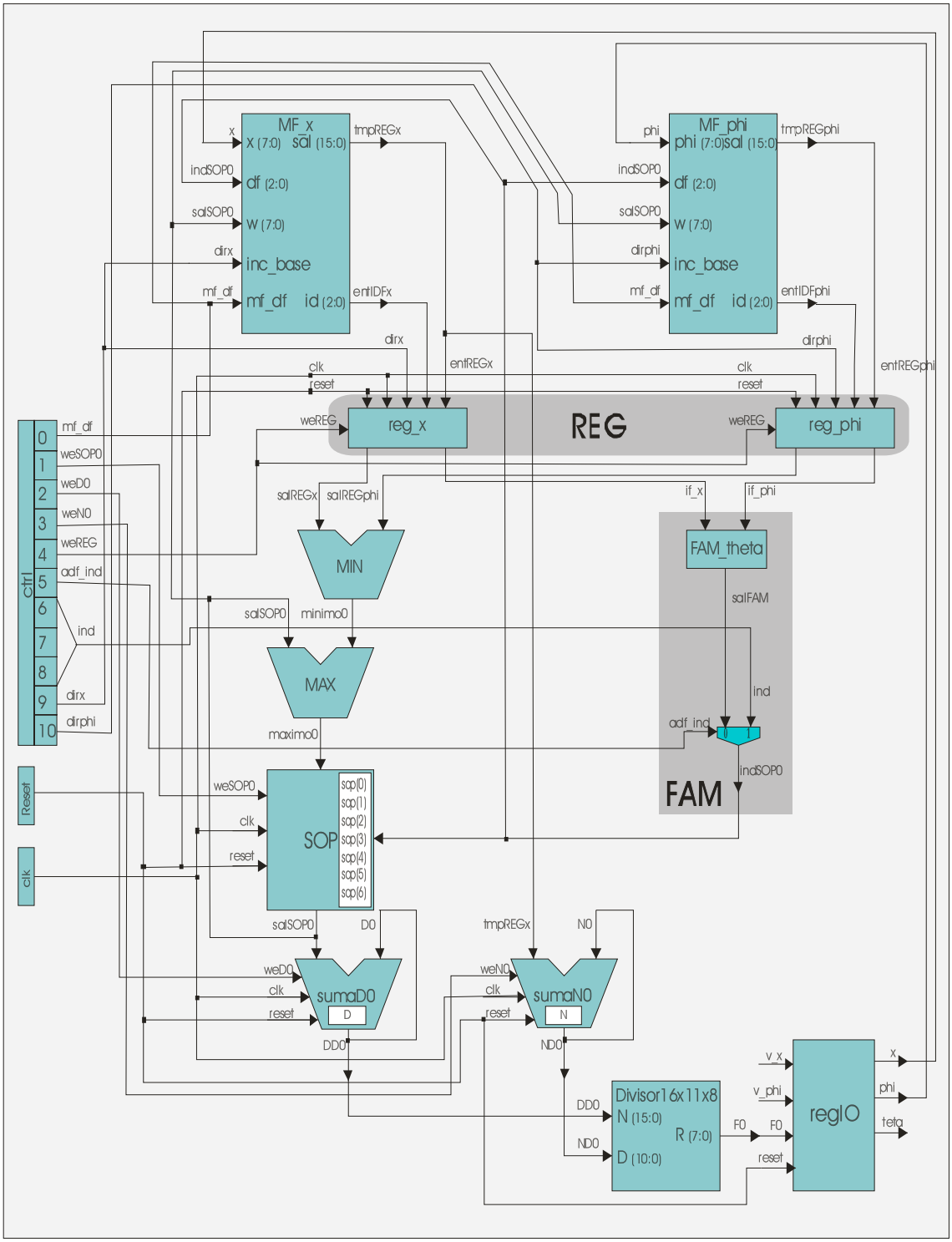


Fig. 7.2 - Diagrama de la arquitectura activa del FLC

```
registroX : PROCESS (CLK, RESET, WE_REG)
    BEGIN
    IF (RESET = '1') THEN
        RegX0 <="00000000"; RegX1 <="00000000";
        IDFx0 <="000";      IDFx1 <="000";
    ELSIF (WE_REG = '1') THEN
        IF (CLK'event and CLK='1') THEN
           IF (dir_x='0') THEN RegX0<=fx; IDFx0<=ifx;
                          ELSE RegX1<=fx; IDFx1<=ifx;
           END IF;
        END IF;
    END IF;
END PROCESS;
WITH dir_x SELECT ReX <= RegX0 WHEN '0', RegX1 WHEN others;
WITH dir_x SELECT IdX <= IDFx0 WHEN '0', IDFx1 WHEN others;
```

7.3.2 – BLOQUES: 'MIN' Y 'MAX'

Los módulos ***MIN*** y ***MAX*** son circuitos combinacionales utilizados para calcular el mínimo y máximo en el motor de inferencia. Para ***MIN*** las señales de entrada son ***salREGx*** y ***salREGphi***, y la salida es ***minimo0***. Para ***MAX*** las señales de entrada son ***salSOP0*** y ***minimo0***, mientras que la salida es ***maximo0***. Las señales ***salREGx*** y ***salREGphi*** son las salidas de los registros de las variables de entrada, y ***salSOP0*** es la salida del registro ***SOP***.

```
min0 : PROCESS (salREGx,salREGphi)
   BEGIN
   IF (salREGx > salREGphi) THEN minimo0 <= salREGphi;
                            ELSE minimo0 <= salREGx;  END IF;
   END PROCESS;

max0 : PROCESS (salSOP0,minimo0)
   BEGIN
   IF (salSOP0 < minimo0) THEN maximo0 <= minimo0;
                          ELSE maximo0 <= salSOP0; END IF;
   END PROCESS;
```

7.3.3 – BLOQUE: 'FAM'

El módulo **FAM** determina la función de pertenencia de salida que se activa por cada regla ejecutada. Se compone de dos partes: a) una memoria que se direcciona por dos valores ***if_x*** e ***if_phi***, que representa la matriz con todas reglas difusas del FLC. b) un multiplexor que permite direccionar el banco de registros por ***IND*** o ***SalFAM***.

```
PROCESS (ifx, ifp)
    BEGIN
    sal := "000";
    --- 1ra. fila
    IF    (ifx=le)and(ifp=rb) THEN sal := ps;
    ELSIF (ifx=lc)and(ifp=rb) THEN sal := pm;
    ELSIF (ifx=ce)and(ifp=rb) THEN sal := pm;
    ELSIF (ifx=rc)and(ifp=rb) THEN sal := pb;
    ELSIF (ifx=ri)and(ifp=rb) THEN sal := pb;
    --- 2da. fila
    ELSIF (ifx=le)and(ifp=ru) THEN sal := ns;
    ELSIF (ifx=lc)and(ifp=ru) THEN sal := ps;
    ELSIF (ifx=ce)and(ifp=ru) THEN sal := pm;
    ELSIF (ifx=rc)and(ifp=ru) THEN sal := pb;
    ELSIF (ifx=ri)and(ifp=ru) THEN sal := pb;
    --- 3er. fila
    ELSIF (ifx=le)and(ifp=rv) THEN sal := nm;
    ELSIF (ifx=lc)and(ifp=rv) THEN sal := ns;
    ELSIF (ifx=ce)and(ifp=rv) THEN sal := ps;
    ELSIF (ifx=rc)and(ifp=rv) THEN sal := pm;
    ELSIF (ifx=ri)and(ifp=rv) THEN sal := pb;
    --- 4ta. fila
    ELSIF (ifx=le)and(ifp=ve) THEN sal := nm;
    ELSIF (ifx=lc)and(ifp=ve) THEN sal := nm;
    ELSIF (ifx=ce)and(ifp=ve) THEN sal := ze;
    ELSIF (ifx=rc)and(ifp=ve) THEN sal := pm;
    ELSIF (ifx=ri)and(ifp=ve) THEN sal := pm;
    --- 5ta. fila
    ELSIF (ifx=le)and(ifp=lv) THEN sal := nb;
    ELSIF (ifx=lc)and(ifp=lv) THEN sal := nm;
    ELSIF (ifx=ce)and(ifp=lv) THEN sal := ns;
    ELSIF (ifx=rc)and(ifp=lv) THEN sal := ps;
    ELSIF (ifx=ri)and(ifp=lv) THEN sal := pm;
    --- 6ta. fila
    ELSIF (ifx=le)and(ifp=lu) THEN sal := nb;
    ELSIF (ifx=lc)and(ifp=lu) THEN sal := nb;
    ELSIF (ifx=ce)and(ifp=lu) THEN sal := nm;
    ELSIF (ifx=rc)and(ifp=lu) THEN sal := ns;
    ELSIF (ifx=ri)and(ifp=lu) THEN sal := ps;
    --- 7ma. fila
    ELSIF (ifx=le)and(ifp=lb) THEN sal := nb;
    ELSIF (ifx=lc)and(ifp=lb) THEN sal := nb;
    ELSIF (ifx=ce)and(ifp=lb) THEN sal := nm;
    ELSIF (ifx=rc)and(ifp=lb) THEN sal := nm;
    ELSIF (ifx=ri)and(ifp=lb) THEN sal := ns;
    END IF;
salFAM0 <= sal;
END PROCESS;
```

El valor del registro ***salFAM0*** es almacenado en el banco de registros del bloque ***SOP***, que tiene tantos registros como funciones de pertenencia de salida tenga la variable. El registro en el que se realiza el almacenamiento es seleccionado por el valor ***indSOP***, que toma el valor de ***IND*** o de ***salFAM***.

```
mx_ADFind : PROCESS (adf_ind, salFAM0, ind)
    BEGIN
    CASE adf_ind IS
    WHEN '0'    => indSOP <= salFAM0;
    WHEN OTHERS => indSOP <= ind;
    END CASE;
    END PROCESS;
```

7.3.4 – BLOQUE: 'SOP'

El bloque **SOP** representa a un banco de registros, con tantos registros como funciones de pertenencia de salida tenga el controlador. Este banco es direccionado por ***indSOP0***, la señal de control ***WE_SOP0*** habilita para escritura al banco de registros, ***RESET*** se utiliza como señal de inicialización asincrona, mientras que la escritura se realiza con el flanco ascendente del reloj ***CLK***. La señal ***maximo0*** contiene el valor de entrada para almacenar durante una operación de escritura, mientras que ***salSOP0*** es el valor generado de salida.

```
PROCESS (CLK, RESET, indSOP0, weSOP0)
   BEGIN
   IF (RESET='1') THEN
      SOP_theta0 <="00000000"; SOP_theta1 <="00000000";
      SOP_theta2 <="00000000"; SOP_theta3 <="00000000";
      SOP_theta4 <="00000000"; SOP_theta5 <="00000000";
      SOP_theta6 <="00000000"; SOP_theta7 <="00000000";
    ELSIF (weSOP0 = '1') THEN
      IF (CLK'event and CLK='1') THEN
         IF    (dir="000") THEN SOP_theta0<=maximo0;
         ELSIF (dir="001") THEN SOP_theta1<=maximo0;
         ELSIF (dir="010") THEN SOP_theta2<=maximo0;
         ELSIF (dir="011") THEN SOP_theta3<=maximo0;
         ELSIF (dir="100") THEN SOP_theta4<=maximo0;
         ELSIF (dir="101") THEN SOP_theta5<=maximo0;
         ELSIF (dir="110") THEN SOP_theta6<=maximo0;
         ELSIF (dir="111") THEN SOP_theta7<=maximo0;
         END IF;
      END IF;
    END IF;
END PROCESS;

WITH indSOP0 SELECT
salSOP <= SOP_theta0 WHEN "000",  SOP_theta1 WHEN "001",
          SOP_theta2 WHEN "010",  SOP_theta3 WHEN "011",
          SOP_theta4 WHEN "100",  SOP_theta5 WHEN "101",
          SOP_theta6 WHEN "110",  SOP_theta7 WHEN others;
```

7.3.5 – BLOQUES: 'SUMA_N' Y 'SUMA_D'

Los módulos **SumaD** y **SumaN** permiten acumular el valor del denominador ***D*** y del numerador ***N***, respectivamente. Las señales de habilitación de escritura de cada registro son ***WE_D*** y ***WE_N***, para ***D*** y ***N*** respectivamente. El código de los

módulos es similar, por tal razón sólo se muestra el correspondiente al acumulador del registro ***D***; donde la señal ***D0*** es un registro temporal, ***salSOP0*** es el valor de entrada, y el valor ***D*** es la señal de salida.

```
sumaD : PROCESS (CLK, RESET, weD, D0)
    BEGIN
    IF (RESET=´1´) THEN D0<=”000000”;
    ELSIF (weD=´1´) THEN
     IF(CLK´event and CLK=´1´) THEN D0<=D0+salSOP0; END IF;
    END IF;
    D<=D0;
    END PROCESS;
```

7.3.6 – BLOQUE: ‘DIV’

El módulo **DIV** es un circuito combinacional, para ser utilizado durante la defusificación, que materializa el algoritmo de división con restauración. La materialización de este algoritmo se ha detallado en el capítulo 6, sección 3 (Fig. 6.7).

El algoritmo de división se basa en dos procedimientos: A) ***suma*** cuyos operandos son ***vL***[10:0], ***vR***[10:0], ***cc*** es la señal de acarreo de entrada, ***co*** es la señal de acarreo de salida, y ***vO***[10:0] es el resultado de la operación ***vL***+***vR***+***cc***. B) ***dividir*** es el otro procedimiento, cuyos operandos son ***A***[15:0] y ***B***[10:0] como entradas, y ***C***[7:0] para el resultado.

```
PROCEDURE suma
   (vL:  in  STD_LOGIC_VECTOR(10 downto 0);
    vR:  in  STD_LOGIC_VECTOR(10 downto 0);
    cc:  in STD_LOGIC;
    co: out STD_LOGIC;
    vO: out STD_LOGIC_VECTOR(10 downto 0)  ) is
variable ca : STD_LOGIC;
BEGIN
  ca := cc;
  FOR i IN vL'RIGHT TO vL'LEFT LOOP
    vO(i) := vL(i) xor vR(i) xor carry;
    ca := (vL(i) and vR(i))or(vL(i) and ca)or(vR(i) and ca);
  END LOOP;
  co := ca;
  return;
END suma;
```

El procedimiento ***dividir*** se encarga de implementar cada una de las filas del circuito de la Fig. 6.7. El primer paso que se realiza es la alineación de los bits más significativos del dividendo y del divisor, para lo cual duplica el número de

bits del dividendo agregando ceros a los bits más significativos. Luego realiza la resta como suma en complemento a dos, y por último de acuerdo al acarreo obtenido se verifica si es necesario restaurar el valor o no. El bit de acarreo es desplazado dentro del resultado, y en este caso el resto calculado no es utilizado, ya que se aplica una división entera. La cantidad de veces que se repite este ciclo es igual al doble de la cantidad de bits del dividendo menos la cantidad de bits del divisor.

```
PROCEDURE Dividir
   (A:  in  STD_LOGIC_VECTOR(15 downto 0);
    B:  in  STD_LOGIC_VECTOR(10 downto 0);
    C: out STD_LOGIC_VECTOR( 7 downto 0)) is
variable carry    : STD_LOGIC;
variable x, y, nb : STD_LOGIC_VECTOR(10 downto 0);
variable z        : STD_LOGIC_VECTOR(15 downto 0);
variable r        : STD_LOGIC_VECTOR(31 downto 0);
BEGIN
   r  := "0000000000000000" & A;
   nb := "11111111111" xor B;
   z  := "0000000000000000";
   FOR i IN 1 TO 21 LOOP
      r := r (30 downto 0) & '0';
      x := r (31 downto 21);
      suma(x, nb, '1',carry, y);
      IF carry = '1' THEN r(31 downto 21) := y;  END IF;
      z := z(14 downto 0) & carry;
   END LOOP;
   C:= z(7 downto 0);
   return;
END Dividir;
```

La entidad es utilizada por medio de tres parámetros, las señales ***N*** y ***D*** que corresponden al numerador (de 16 bits) y al denominador (de 11 bits), respectivamente; mientras que el resultado de siete bits es generado en la señal ***R***. El circuito completo se compone de un proceso llamado ***division*** sensible a ***N*** y ***D***.

```
ENTITY Divisor16x11x8 IS
   port ( N :  in STD_LOGIC_VECTOR(15 downto 0);
          D :  in STD_LOGIC_VECTOR(10 downto 0);
          R : out STD_LOGIC_VECTOR(7 downto 0)  );
END Divisor16x11x8;

ARCHITECTURE Divisor16x11x8_Arch of Divisor16x11x8 is
BEGIN
Division : PROCESS ( N, D)
    variable NN : STD_LOGIC_VECTOR(15 downto 0);
    variable DD : STD_LOGIC_VECTOR(10 downto 0);
    variable RR : STD_LOGIC_VECTOR( 7 downto 0);
    BEGIN
    NN := N;
    DD := D;
    Dividir(NN,DD,RR);
    R <= RR;
    END PROCESS;
END Divisor16x11x8_Arch;
```

7.3.7 – SEÑALES DE CONTROL

Las señales de control, que determinan el comportamiento del circuito durante cada ciclo de cálculo, forman el microprograma que permite realizar las operaciones de fusificación, inferencia y defusificación.

El ancho de las señales de control es:

- Log2 (***O***): para ***dir_x***, ***dir_p***.
- Log2 (número de funciones de pertenencia de salida): para ***ind***.
- Un bit: para ***reset***, ***CLK***, ***weREG***, ***adf_ind***, ***mf_df***, ***weSOP***, ***weN***, y ***weD***.

7.4 – CÁLCULO DE FUNCIONES DE PERTENENCIA

En esta sección se analiza el circuito generado por la herramienta (Apéndice A) para el fusificador. Como el circuito de fusificación es hecho completamente a medida de la aplicación especificada en FIL, la arquitectura de cálculo descrita en el capítulo 6 se replica para todas las variables de entrada del controlador. En este caso sólo se analiza el circuito de la variable ***X*** del ejemplo de Kosko [Kos92]

presentado en la sección 3.7.1 y cuya arquitectura global se ha mostrado en la sección 7.3.

El sistema genera el código del módulo principal, del circuito de cálculo de las funciones de pertenencia, mediante una lista de conexiones (*net list*) que vincula los 11 módulos VHDL (Fig. 7.3): ***SelX***, ***x_ROM_ib1***, ***x_incBASE***, ***x_ROMfp***, ***ROMfd***, ***Resta***, ***MFDF_Selection***, ***Multiplicador***, ***Trunc***, ***SumRes***, y ***MFDF_OutSelection***.

7.4.1 – BLOQUE: 'SELX'

El módulo **SELX** determina la posición del primer segmento, en la memoria ***x_ROM_fp***, de acuerdo al valor de la variable de entrada ***X***. Para ello, ***X*** es comparado con la posición inicial de cada segmento (abscisa), y una vez encontrado la señal ***v1*** contiene su posición.

```
selx : PROCESS ( x )
variable Tv1:STD_LOGIC_VECTOR(3 downto 0);
BEGIN
   IF (x >=   0) and (x <  30) THEN Tv1:="0000";  --   0
ELSIF (x >=  30) and (x <  96) THEN Tv1:="0001";  --   1
ELSIF (x >=  76) and (x < 102) THEN Tv1:="0010";  --   2
ELSIF (x >= 102) and (x < 127) THEN Tv1:="0011";  --   3
ELSIF (x >= 114) and (x < 127) THEN Tv1:="0100";  --   4
ELSIF (x >= 127) and (x < 140) THEN Tv1:="0101";  --   5
ELSIF (x >= 127) and (x < 153) THEN Tv1:="0110";  --   6
ELSIF (x >= 153) and (x < 178) THEN Tv1:="0111";  --   7
ELSIF (x >= 158) and (x < 224) THEN Tv1:="1000";  --   8
ELSE                               Tv1:="1001";  --   9
END IF;
v1 <= Tv1;
END PROCESS;
```

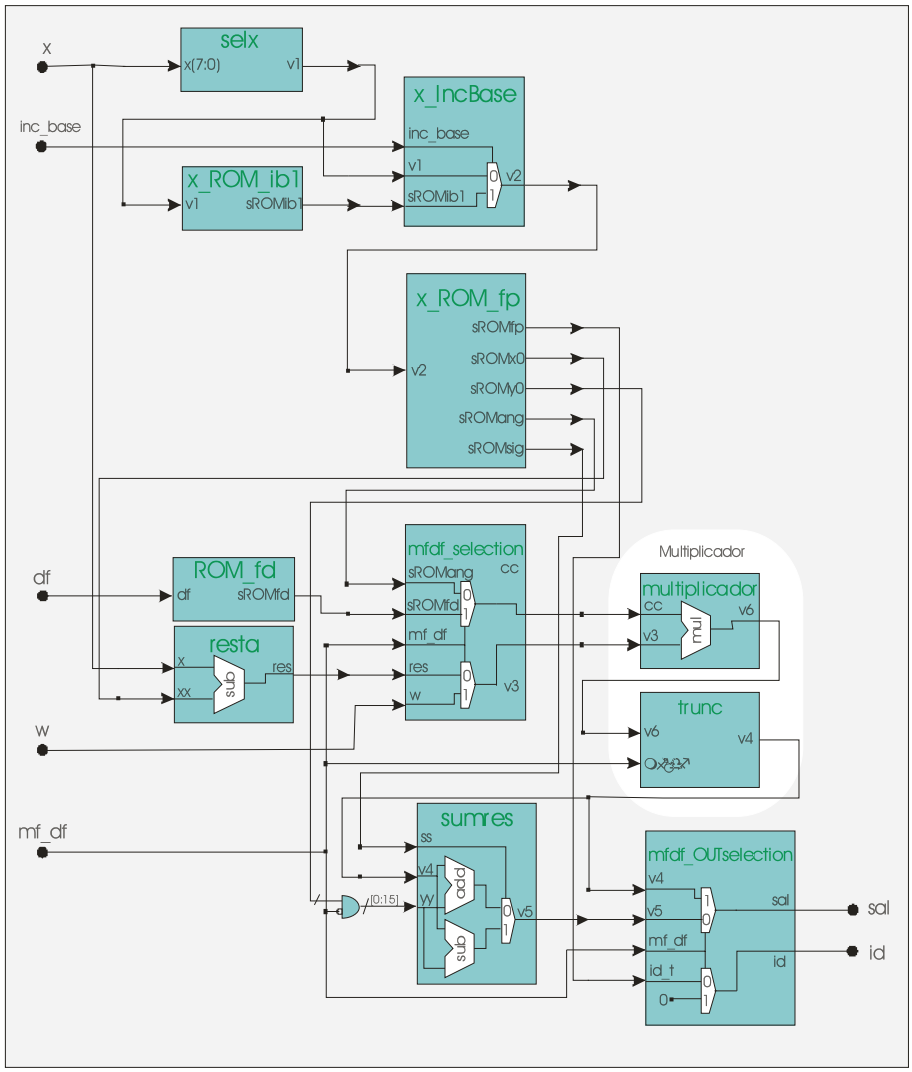


Fig. 7.3 – Diagrama del circuito de cálculo de las funciones de pertenencia

7.4.2 – BLOQUE: 'X_ROM_IB1'

Este módulo determina cual es la posición del siguiente segmento en la memoria ***x_ROM_fp***, de acuerdo al valor de la posición del primer segmento. La variable ***v1*** contiene la posición del primer segmento y se usa para determinar la posición del segundo segmento, en la señal ***sROMib1***.

```
x_ROM_ib1: PROCESS ( v1 )
variable t1:STD_LOGIC_VECTOR(3 downto 0);
BEGIN
CASE CONV_INTEGER( v1 ) IS
      WHEN      0 => t1:="0010";
      WHEN      1 => t1:="0010";
      WHEN      2 => t1:="0100";
      WHEN      3 => t1:="0100";
      WHEN      4 => t1:="0110";
      WHEN      5 => t1:="0110";
      WHEN      6 => t1:="1000";
      WHEN      7 => t1:="1000";
      WHEN      8 => t1:="0111";
      WHEN      9 => t1:="0111";
      WHEN others => t1:="0000";
      END CASE;
      sROMib1 <= t1;
END PROCESS;
```

7.4.3 – BLOQUE: 'X_INC_BASE'

Este bloque selecciona por medio de la señal ***inc_base*** entre la posición del primer segmento ***v1*** y los ***O*** segmentos que se superponen. La posición del siguiente segmento está en la señal ***sROMib1***, mientras que el resultado se genera en la señal ***v2***.

```
x_IncBase: PROCESS (inc_base, sROMib1,  v1)
BEGIN
   CASE CONV_INTEGER( '0'&inc_base ) IS
       WHEN 0      => v2 <= v1;
       WHEN OTHERS => v2 <= sROMib1;
   END CASE;
END PROCESS;
```

7.4.4 – BLOQUE: 'X_ROM_FP'

El módulo es una memoria **ROM** que contiene los parámetros para el cálculo de cada uno de los segmentos. Las señales son: ***v2*** representa la dirección a consultar (4 bits), ***sROMx0*** es la posición de la abscisa inicial (8 bits), ***sROMang*** es el valor de la pendiente del segmento (13 bits), ***sROMsig*** es el signo de la pendiente (1 bit), ***sROMy0*** es el valor de ordenada de la posición inicial del segmento (al no estar codificado se representa con 8 bits), y ***sROMfp*** es el identificador de función de pertenencia al que corresponde el segmento (4 bits).

```
x_ROM_fp: PROCESS ( v2 )
   variable t:STD_LOGIC_VECTOR(32 downto 0);
BEGIN
CASE CONV_INTEGER( v2 ) IS
                    --sROMfp_sROMx0_sROMy0_sROMang_sROMsig
   WHEN      0 => t:= "00000000000111111110000000000000";
   WHEN      1 => t:= "00000011110111111110001111011011";
   WHEN      2 => t:= "00101001100000000000100111001100";
   WHEN      3 => t:= "00101100110111111110101000110011";
   WHEN      4 => t:= "01001110010000000010011100111010";
   . . . . .
   WHEN      8 => t:= "10010011110000000000011110111010";
   WHEN      9 => t:= "10011100000111111110000000000000";
   WHEN OTHERS => t:= "00000000000000000000000000000000";
   END CASE;
   sROMfp  <= t(32 downto 30);
   sROMx0  <= t(29 downto 22);
   sROMy0  <= t(21 downto 14);
   sROMang <= t(13 downto 1);
   sROMsig <= t(0);
END PROCESS;
```

7.4.5 – BLOQUE: 'RESTA'

El bloque **RESTA** calcula la resta del valor de la variable de entrada ***X*** menos el valor ***sROMx0*** de la ***x_ROM_fp***. El resultado de esta operación se genera en la señal ***RES***. Esta resta es calculada si el valor de ***X*** no es menor que ***sROMx0***, caso contrario el resultado es cero.

```
resta: PROCESS ( xx, x )
BEGIN
   IF x < xx THEN  res <= "00000000";
   ELSE            res <= x - xx;
   END IF;
END PROCESS;
```

7.4.6 – BLOQUE: 'ROM_FD'

Este bloque contiene los pesos de las funciones de decodificación. La señal ***df*** representa la dirección de la función a consultar (3 bits). El resultado se genera en ***sROMfd***.

```
ROM_fd: PROCESS ( df )
   variable t:STD_LOGIC_VECTOR(7 downto 0);
BEGIN
   CASE CONV_INTEGER( df ) IS
   WHEN      0 => t := "00001100";  --- 12
   WHEN      1 => t := "00100010";  --- 34
   WHEN      2 => t := "01010000";  --- 80
   WHEN      3 => t := "01111111";  --- 127
   WHEN      4 => t := "10101110";  --- 174
   WHEN      5 => t := "11011101";  --- 221
   WHEN      6 => t := "11110010";  --- 242
   WHEN OTHERS => t := "00000000"; --- 0
   END CASE;
   sROMfd <= t;
END PROCESS;
```

7.4.7 – BLOQUE: 'MFDF_SELECTION'

Este bloque implementa un multiplexor. La señal ***mf_df*** determina si el circuito es utilizado para fusificar o para defusificar, seleccionando los valores que deben ser multiplicados por el circuito. Si ***mf_df*** contiene un cero, selecciona las señales ***sROMang*** y ***sROMfd***; si contiene un uno selecciona las señales ***res*** y ***w***. Los valores seleccionados son devueltos en las señales ***cc*** y ***v3***, respectivamente.

```
fmdf_selection: PROCESS (  mf_df, res, sROMang, w, sROMfd  )
BEGIN
   IF mf_df = '0' THEN  cc<=sROMang;           v3<=res;
                  ELSE  cc<=sROMfd & "00000";  v3<=w;
   END IF;
END PROCESS;
```

7.4.8 – BLOQUE: 'MULTIPLICADOR'

El circuito de multiplicación calcula el producto entre la pendiente ***CC*** y ***v3***, donde la multiplicación puede corresponder: a la pendiente ***sROMang*** por ***resta*** si está fusificando, o al valor externo ***W*** por ***sROMfd*** si está defusificando. El módulo multiplicador es un circuito combinacional, donde ***A*** y ***B*** son las señales a multiplicar y ***R*** el resultado.

```
ENTITY Multiplicador8x13 IS
  port ( A  :  in STD_LOGIC_VECTOR(7 downto 0);
         B  :  in STD_LOGIC_VECTOR(12 downto 0);
         R  : out STD_LOGIC_VECTOR(20 downto 0)  );
END Multiplicador8x13;

ARCHITECTURE Multiplicador8x13A OF Multiplicador8x13 IS
BEGIN
p_multipli : PROCESS( A , B )
  variable AA : STD_LOGIC_VECTOR(7 downto 0);
  variable BB : STD_LOGIC_VECTOR(12 downto 0);
  variable RR : STD_LOGIC_VECTOR(20 downto 0);
BEGIN
  AA := A;
  BB := B;
  multipli (aa,bb,rr) ;
  R <= RR;
END PROCESS;
END Multiplicador8x13A;
```

La entidad multiplicador internamente está formada por el proceso ***p_multipli*** y los procedimientos ***multipli*** y ***suma***. El procedimiento ***suma*** implementa el ***full-adder***, y ***multipli*** implementa las filas del circuito. El proceso ***p_multipli*** es el proceso de mayor nivel y es sensible a las señales ***A*** y ***B***.

```
PROCEDURE multipli
       (AA   :  in STD_LOGIC_VECTOR(7 downto 0);
        BB   :  in STD_LOGIC_VECTOR(12 downto 0);
        RR   : out STD_LOGIC_VECTOR(20 downto 0)) is
  variable v, carry    : STD_LOGIC;
  variable r1, r3, r4  : STD_LOGIC_VECTOR(7 downto 0);
  variable r2          : STD_LOGIC_VECTOR(12 downto 0);
BEGIN
  carry := '0';
  r1 := "00000000";
  r2 := "0000000000000";
  FOR i IN BB'RIGHT  TO BB'LEFT LOOP
     v  := BB(i) ;
     r3 := ( v, v, v, v, v, v, v, v );
     r3 := AA and r3;
     suma(r3,r1,carry,r4);
          r2(12 downto 0) := r4(0) & r2(12 downto 1);
          r1(7 downto 0) := Carry & r4(7 downto 1);
  END LOOP;
  RR(20 downto 13) := r1;
  RR(12 downto 0) := r2;
RETURN;
END procedure multipli;
```

7.4.9 – BLOQUE: 'TRUNC'

El bloque **TRUNC** se utiliza para ajustar el tamaño de salida del multiplicador al tamaño de la señal ***sal*** (que contiene el resultado de la fusificación o

defusificación). Este ajuste es necesario debido al re-uso del multiplicador en ambas etapas, por otra parte como se necesita el producto entero se eliminan los bits menos significativos (decimales).

```
trunc: PROCESS ( v6 , mf_df )
BEGIN
   IF mf_df = '0' THEN  v4 <= "000" & v6(20 downto 8);
                  ELSE  v4 <= v6(20 downto 5);
   END IF;
END PROCESS;
```

7.4.10 – BLOQUE: 'SUMRES'

El módulo **SUMRES** realiza el cálculo de la suma o resta del valor de la ordenada ***YY*** con el valor de la multiplicación ***v4***. Se determina la operación a realizar de acuerdo al valor de signo ***sROMsig*** de la ***x_ROM_fp***, representado por la señal ***ss***. El resultado de esta operación se genera en la señal ***v5***. Nótese que se pasa como parámetro el signo al procedimiento suma, de tal forma que sume el signo de acuerdo a la operación a realizar, si ***ss*** es uno se resta, si es cero se suma.

```
sumres: PROCESS ( ss, yy, v4 )
   variable tv5:STD_LOGIC_VECTOR(15 downto 0);
BEGIN
   IF ss = '1' THEN  tv5 := yy - v4;
               ELSE  tv5 := yy + v4; END IF;
   v5 <= tv5;
END PROCESS;
```

7.4.11 – BLOQUE: 'FMDF_OUTSELECTION'

Este bloque selecciona el valor de salida de acuerdo al valor de la señal de entrada ***mf_df***.

```
fmdf_OUTselection: PROCESS ( mf_df, v5, id_t, v4 )
BEGIN
   IF mf_df = '0' THEN  sal <= v5;  id <= id_t;
                  ELSE  sal <= v4;  id <= "000"; END IF;
END PROCESS;
```

7.4.12 – TAMAÑO DE LAS SEÑALES DEL PROYECTO

Para la completa automatización del proyecto se debe poder calcular el tamaño de todas las señales de control. Del análisis se determina que la cantidad de bits necesarias es de:

- Un bit para ***mf_df*** (empleada para representar si se está utilizando para fusificar o defusificar).
- Log2 (*máximo número de funciones de decodificación para todas las variables*) bits para ***df***.
- Log2 (*máximo número de segmentos superpuestos*) bits para ***INC_BASE*** (que determina cual segmento se está calculando).
- Log2 (*máximo valor que puede tomar la variable en el eje X*) bits para ***X*** (que representa el valor de la variable de entrada).
- Log2 (*máximo valor que puede tomar la variable en el eje Y*) bits para ***W*** (que es el valor difuso determinado por el motor de inferencia).
- Log2 (*cantidad de funciones de pertenencia para todas las variables*) + Log2 (*máximo valor que puede tomar la variable en el eje X*) bits para ***Sal*** (que representa el valor de salida).

7.5 – GENERADOR DEL MICROPROGRAMA

La generación del microprograma puede ser 'casi' directa, se limita al control de la ruta de datos, debido a que los módulos hardware son generados totalmente a medida de la aplicación.

Por ejemplo, como se adapta el tamaño de los registros y la ruta de datos al tamaño requerido por la aplicación, las instrucciones que necesitan múltiples operaciones simples para su ejecución son realizadas en un único ciclo.

Además, como la arquitectura dispone de unidades de memoria a medida distribuidas en los lugares en que se necesitan, sólo se debe controlar su habilitación de escritura cuando corresponda.

#µ orden	dir_phi	dir_x	ind	adf_ind	wREG	wN0	wD0	WSOP0	mf_df	reset
0	0	0	000	0	0	0	0	0	0	1
1	0	0	000	0	1	0	0	0	0	0
2	1	1	000	0	1	0	0	0	0	0
3	0	0	000	0	0	0	0	1	0	0
4	1	0	000	0	0	0	0	1	0	0
5	0	1	000	0	0	0	0	1	0	0
6	1	1	000	0	0	0	0	1	0	0
7	0	0	000	1	0	1	1	0	1	0
8	0	0	001	1	0	1	1	0	1	0
9	0	0	010	1	0	1	1	0	1	0
10	0	0	011	1	0	1	1	0	1	0
11	0	0	100	1	0	1	1	0	1	0
12	0	0	101	1	0	1	1	0	1	0
13	0	0	110	1	0	1	1	0	1	0
14	0	0	000	0	0	0	0	0	0	1

Las señales que forman el microprograma son las siguientes: A) ***dir_phi***, durante el proceso de inferencia se utiliza para direccionar el valor de la variable ***phi*** fusificada y su identificador de tabla. B) ***dir_x***, durante el proceso de inferencia se utiliza para direccionar el valor de la variable ***x*** fusificada y su identificador de tabla. C) ***IND***, dirección del banco de registros **SOP**, activo durante la defusificación. D) ***ADF_IND***, selector del modo de direccionamiento del banco de registros **SOP**. E) ***wREG***, señal de habilitación de escritura del módulo **REG**, durante la fusificación. F) ***wN***, señal de habilitación de escritura del registro ***N***, activo durante la defusificación. G) ***wD***, señal de habilitación de escritura del registro ***D***, activa durante la defusificación. H) ***wSOP***, señal de habilitación de escritura del módulo **SOP**, activa durante la inferencia. I) ***mf_df***, selecciona el modo del circuito de cálculo de las funciones de pertenencia, durante la fusificación o defusificación. J) ***RESET***, inicializa el circuito.

La funcionalidad de cada microinstrucción según su orden de ejecución es la siguiente: La primera (0) registra los valores de entrada y salida; Las dos siguientes (1 y 2) fusifican las variables de entrada; Las cuatro que siguen (3 a 6) realizan la inferencia difusa; Las siete siguientes (7 a 13) defusifican la variable de salida, acumulando los registros ***D*** y ***N*** y realizando la división; Mientras que la última microinstrucción (la 14) escribe el resultado en las variables de salida e inicialización del circuito.

7.6 – GENERADOR DEL CÓDIGO DE SIMULACIÓN

El código generado para el controlador puede ser definido utilizando los siguientes lenguajes de programación: C y Pascal (o Delphi). En todos los casos el software se limita a materializar el algoritmo propuesto de tal forma que se pueda utilizar este programa para poner a punto el controlador, mas sólo con un delicado trabajo de optimización éste código podría convertirse en una versión operativa en software sobre una computadora genérica.

7.7 - RESUMEN

En este capítulo se presenta una metodología y un conjunto de herramientas que asisten al diseñador en la construcción de controladores difusos de altas prestaciones. Las herramientas realizan las siguientes funciones:

- Análisis del archivo de configuración para determinar si el archivo que define el controlador no tiene errores, y es posible su generación automática.
- Generación del código VHDL del controlador, cuyo motor de inferencias trabaje por reglas activas.
- Generación del código VHDL del circuito de fusificación por segmentos.
- Generación del contenido de la ROM del microprograma de control.
- Generación de código del controlador en un lenguaje de programación (C y Pascal). Este controlador software es útil en la puesta a punto del controlador por medio de la simulación.

El código VHDL generado por el sistema puede ser automáticamente sintetizado para materializar el controlador utilizando una FPGA como plataforma.

Como principales características de la arquitectura del controlador difuso se pueden destacar:

- Número reducido de registros.
- Bancos de memoria a medida de las necesidades de la aplicación (en cantidad de palabras y en el ancho de palabra).
- Múltiples unidades de cálculo dedicadas y totalmente a medida.
- Rutas de datos a medida.

8 –MÉTRICAS

8.1 - INTRODUCCIÓN

El objetivo de este capítulo es describir y comparar los resultados obtenidos en la implementación de controladores generados automáticamente. La decisión de cual de las arquitecturas propuestas se debe adoptar es muy difícil, con sólo conocer el funcionamiento de las distintas máquinas de cálculo difuso propuestas.

Una de las formas de comparar las arquitecturas propuestas es por la velocidad máxima a la que pueden trabajar, para lo cual se utilizan dos unidades **FIPS** (*Fuzzy Inference Per Second*) y **FRPS** (*Fuzzy Rules Per Second*). Por supuesto, desde el punto de vista del diseñador, la velocidad no es el único parámetro importante para juzgar la calidad de un procesador difuso. También interesa tener en cuenta el impacto producido al variar: la cantidad de variables de entrada, la cantidad de variables de salida, el dominio de las funciones de pertenencia, la cantidad de funciones de pertenencia, la cantidad de reglas, la cantidad de memoria utilizada, la cantidad de puertos de entrada/salida, el tamaño de arquitectura interna, etc.

Las métricas propuestas consisten en dos enfoques principales, uno orientado al análisis del impacto de la arquitectura interna considerando diferentes tamaños de rutas de datos, y el otro con el propósito de comparar aplicaciones que presenten diferentes características. Por otra parte, cabe destacar que cada uno de tales enfoques es analizado tanto para materializaciones software como hardware (o físicas) de cada FLC.

SOFTWARE. El objetivo de las secciones software es determinar la velocidad de ejecución de los programas que implementan el comportamiento de los controladores. Esta implementación se realiza a partir del programa en código *C* obtenido automáticamente de las herramientas, sobre la plataforma *Mandrake Linux* 9.1, en un PC con un microprocesador Celeron de 400 Mhz y 256 MB de memoria

RAM, y utilizando el compilador *gcc*. Con el propósito de medir el tiempo de ejecución requerido por cada programa se agrega al código fuente llamadas al sistema operativo *Linux* para la lectura del reloj interno. Cada programa implementa un ciclo con llamadas a la función *Fuzzy*, usando como parámetros todas las combinaciones de todos los posibles valores que pueden adoptar las variables de entrada. En todos los casos, la cantidad de valores que pueden adoptar las variables de entradas es de 256 (de 8-bit), por lo tanto la cantidad de iteraciones que se realizan es de $256^{\text{cantidad variables entrada}}$ para cualquier programa. En cada ciclo se realiza el cálculo de una inferencia formada por el conjunto de reglas que corresponda. Se mide el tiempo de ejecución del programa para realizar la totalidad de los ciclos (en centésimas de segundo), para luego determinar el tiempo requerido por cada ciclo y calcular la velocidad.

HARDWARE. Mediante la implementación física se analizan las características y requerimientos necesarios para la materialización de las arquitecturas propuestas en una FPGA. La implementación física tiene como principales parámetros de comparación a la máxima frecuencia de operación, la cantidad de puertas equivalentes y el área de la FPGA requerida por el controlador. El circuito que materializa las funciones de pertenencia no es incluido en el controlador de la arquitectura pasiva, en su lugar se genera el acceso a una memoria externa. Por el otro lado, para la arquitectura activa se realizan dos implementaciones, una que incluya el bloque de cálculo de las funciones de pertenencia y la otra que acceda a las funciones de forma externa. La implementación física de todos los controladores es realizada para un dispositivo de la familia *SPARTAN2 de Xilinx* y cuyo modelo varia de acuerdo a las características del controlador diseñado. El primer paso en la implementación física es la síntesis del código VHDL del controlador, utilizando el *FPGA Express 3.4 de Synopsys*. Para lo cual se incluye el código VHDL del FLC en un proyecto del *FPGA Express*, se crea la implementación sobre un dispositivo *SPARTAN2*, y una vez generada la implementación se exporta el *NetList*. Por

último, con el *Design Manager 3.1i de Xilinx* se incluye la *NetList* en un nuevo proyecto, se selecciona el dispositivo y finalmente se realiza la implementación.

8.2 –DIFERENTES TAMAÑOS DE PALABRA

Esta sección analiza los resultados de la implementación de los controladores de dos aplicaciones, para cada uno de los cuales se diseñaron dos controladores con distintos tamaños de palabra (6, 8, 10 y 12 bits). Los ejemplos utilizados son el camión de 35 reglas propuesto por [Kos92] y el control de un péndulo invertido de 49 reglas descrito en [Yam87, Son00], ambos con dos variables de entrada y una de salida.

8.2.1 – MATERIALIZACIÓN SOFTWARE

Los resultados de las mediciones de la ejecución de los programas de los ejemplos del camión y el péndulo, con sus diferentes arquitecturas y anchos de palabra, se pueden ver en la Tabla 8.1.

	Tamaño de la Arquitectura	Ciclos de ejecución	Arquitectura PASIVA		Arquitectura ACTIVA	
			Tiempo		Tiempo	
			1 Ciclo (µs)	Total (seg./100)	1 Ciclo (µs)	Total (seg./100)
Camión	6 bits	384.400	1,66	64	8,16	314
	8 bits	65.536	1,67	11	8,54	56
	10 bits	1.048.576	1,73	182	8,63	905
	12 bits	16.777.216	1,76	2.958	8,69	14.593
Péndulo	6 bits	384.400	1,74	67	8,71	335
	8 bits	65.536	1,83	12	9,15	60
	10 bits	1.048.576	1,85	194	9,16	961
	12 bits	16.777.216	1,87	3.151	9,18	15.414

Tabla 8.1 - Tiempos de la implementación por software

Con los resultados de la Tabla 8.1 se puede calcular la velocidad en *FRPS* y *FIPS* como muestra la Tabla 8.2. El tiempo requerido para ejecutar un ciclo es

equivalente al tiempo para realizar una inferencia y se puede determinar la cantidad de inferencias por segundo haciendo 1/*Tiempo*. Con este valor es fácilmente calculable la cantidad de reglas evaluadas por segundo.

	Tamaño de la Arquitectura	Arquitectura PASIVA		Arquitectura ACTIVA	
		FIPS	FRPS	FIPS	FRPS
Camión (35 reglas)	6 bits	602.409,63	21.084.337,05	122.549,01	4.289.215,35
	8 bits	598.802,39	20.958.083,65	117.096,01	4.098.360,35
	10 bits	578.034,68	20.231.213,80	115.874,85	4.055.619,75
	12 bits	568.181,81	19.886.363,35	115.074,79	4.027.617.65
Péndulo (49 reglas)	6 bits	574.712,64	28.160.919,36	114.746.26	5.622.566.74
	8 bits	546.448,08	26.775.955,92	109.289,61	5.355.190,89
	10 bits	540.540,54	26.486.486,46	109.170,30	5.349.344,70
	12 bits	534.759,35	26.203.208,15	108.932,46	5.337.690,54

Tabla 8.2. - Velocidad en la implementación por software

A partir de los resultados mostrados en la Tabla 8.2, se puede concluir que para estas materializaciones software la arquitectura pasiva es más rápida que la activa. Ambos ejemplos tienen la misma cantidad de variables de entrada y salida, pero distinta cantidad de reglas. Las *FIPS* de la arquitectura pasiva son muy similares en ambos ejemplos, mientras que en *FRPS* el péndulo tiene mayor velocidad. En la arquitectura activa, las *FIPS* favorecen al camión y en *FRPS* al péndulo. Por otro lado, se puede observar que la velocidad disminuye a medida que aumenta el ancho de la arquitectura utilizada, entonces, al momento de diseñar un controlador se debe tratar de utilizar el menor ancho de palabra posible si se desea obtener una mayor velocidad de cálculo.

8.2.2 – MATERIALIZACIÓN FÍSICA O HARDWARE

Los resultados obtenidos de la implementación física de la arquitectura pasiva de los ejemplos del camión y el péndulo invertido, con distintos tamaños de arquitectura se muestran en la Tabla 8.3. Para la implementación de la arquitectura pasiva del camión y el péndulo en todas sus variantes de utiliza un dispositivo

2S50PQ208 de 50.000 puertas de la familia *Spartan2*.

La Tabla 8.4 muestra los resultados obtenidos para la arquitectura activa de los ejemplos del camión y el péndulo. Cuando la arquitectura activa incluye los bloques de cálculo de las funciones de pertenencia se usan diferentes dispositivos *Spartan2*, para los diseños de 6 y 8 bits un *2S50PQ208* de 50.000 puertas, mientras que para los 10 y 12 bits un *2S100PQ208* de 100.000 puertas. Para la implementación sin el bloque de cálculo de funciones de pertenencia, se utiliza un dispositivo *2S50PQ208* de 50.000 puertas para todos los diseños.

Arq. PASIVA		ÁREA								Puertas Equiv.	Frecuencia Máxima (Mhz)
		Slices (de 768)		IOBs (de 140)		GCLKIOBs (de 4)		GCLKs (de 4)			
	Bits	Nº	%	Nº	%	Nº	%	Nº	%		
Camión	6	241	31	62	44	3	75	4	100	3.930	36,430
	8	303	39	72	51	3	75	4	100	5.015	33,679
	10	367	47	82	58	3	75	4	100	6.118	30,445
	12	440	57	92	65	3	75	4	100	7.326	30,484
Péndulo	6	241	31	62	44	3	75	4	100	3.936	34,375
	8	303	39	72	51	3	75	4	100	5.021	33,044
	10	367	47	82	58	3	75	4	100	6.124	30,818
	12	440	57	92	65	3	75	4	100	7.332	28,925

Tabla 8.3 - Resultados de la implementación física de la arquitectura pasiva

La Tabla 8.4 muestra como se reduce el área y aumenta la cantidad *IOBs*, al no incluir el circuito de cálculo de las funciones de pertenencia. Se destaca del análisis de las Tablas 8.3 y 8.4 que, a mayor ancho de la arquitectura mayor área requiere y menor es la frecuencia máxima. Es destacable también que en la arquitectura activa con funciones de pertenencia, el agregado de 2 bits tiene un costo mucho mayor en área (entre 100 y 200 slices) que en la pasiva (60 slices), para el FLC del camión.

Un parámetro a tener en cuenta al momento de decidir cual arquitectura es más conveniente, es el área requerida por el controlador para ser implementado en un dispositivo. El área requerida por la arquitectura pasiva es mucho menor que la

requerida por la arquitectura activa, en el caso de los diseños de 10 y 12 bits de la arquitectura activa es necesario utilizar un modelo de dispositivo más grande. Sin embargo, si el cálculo de las funciones en la arquitectura activa es implementado usando una memoria externa, los valores entre la pasiva y la activa se aproximan bastante.

Arq. ACTIVA		ÁREA								Puertas Equiv.	Frec. Máxima (Mhz)
		Slices		IOBs (de 140)		GCLKIOBs (de 4)		GCLKs (de 4)			
	Bits	Nº	%	Nº	%	Nº	%	Nº	%		
Camión con FP	6	587 (768)	76	29	20	2	50	2	50	7.828	9,690
	8	697 (768)	90	33	23	2	50	2	50	9.400	7,372
	10	887(1200)	73	37	26	2	50	2	50	11.923	5,719
	12	1056(1200)	88	41	29	2	50	2	50	14.137	4,731
Camión sin FP	6	267 (768)	34	42	30	2	50	2	50	3.721	8,936
	8	356 (768)	46	46	32	2	50	2	50	4.946	6,912
	10	430 (768)	55	50	35	2	50	2	50	5.970	5,467
	12	542 (768)	70	54	38	2	50	2	50	7,420	4,621
Péndulo con FP	6	540 (768)	70	29	20	2	50	2	50	7.219	9,402
	8	755 (768)	98	33	23	2	50	2	50	10.078	7,250
	10	867(1200)	72	37	26	2	50	2	50	11.665	5,382
	12	1060(1200)	88	41	29	2	50	2	50	14.119	4,536
Péndulo sin FP	6	267 (768)	34	42	30	2	50	2	50	3.724	9,628
	8	359 (678)	46	46	32	2	50	2	50	4.976	7,191
	10	432 (678)	56	50	35	2	50	2	50	6.000	5,659
	12	544 (678)	70	54	38	2	50	2	50	7.450	4,682

Tabla 8.4 - Resultados de la implementación física de la arquitectura activa

La columna "*frecuencia máxima*" obtenida en la implementación no es la velocidad real del controlador, pero si permite calcularla. Para calcular la velocidad máxima de cálculo en FIPS es necesario dividir la frecuencia máxima de operación por la cantidad de instrucciones que tenga el microprograma del controlador. Mientras que, para calcular la velocidad en FRPS hay que multiplicar la cantidad de reglas por la cantidad de inferencias por segundo. En el caso particular de la arquitectura pasiva, hay que tener en cuenta que la ejecución de una instrucción requiere de dos ciclos,

por lo tanto es necesario dividir la frecuencia máxima por dos. En la Tabla 8.5 se pueden ver los resultados obtenidos de calcular la velocidad en términos de *FIPS* y *FRPS*.

		Bits	Reglas	Instrucciones	Frecuencia Máxima (Mhz)	F I P S	F R P S
Arquitectura PASIVA	Camión	6	35	173	36,430	105.289,02	3.685.115,61
		8			33,679	97.338,15	3.406.835,26
		10			30,445	87.991,33	3.079.696,53
		12			30,484	88.104,05	3.083.641,62
	Péndulo	6	49	203	34,375	84.667,49	4.148.706,90
		8			33,044	81.389,16	3.988.068,97
		10			30,818	75.906,40	3.719.413,79
		12			28,925	71.243,84	3.490.948,28
Arquitectura ACTIVA	Camión con FP	6	35	15	9,690	646.000,00	22.610.000,00
		8			7,372	491.466,66	17.201.333,10
		10			5,719	381.266,66	13.344.333,10
		12			4,731	315.400,00	11.039.000,00
	Camión sin FP	6			8,936	595.733,33	20.850.666,67
		8			6,912	460.800,00	16.128.000,00
		10			5,467	364.466,67	12.756.333,33
		12			4,621	308.066,67	10.782.333,33
	Péndulo con FP	6	49	15	9,402	626.800,00	30.713.200,00
		8			7,250	483.333,33	23.683.333,17
		10			5,382	358.800,00	17.581.200,00
		12			4,536	302.400,00	14.817.600,00
	Péndulo sin FP	6			9,628	641.866,67	31.451.466,67
		8			7,191	479.400,00	23.490.600,00
		10			5,659	377.266,67	18.486.066,67
		12			4,682	312.133,33	15.294.533,33

Tabla 8.5 - Velocidad de cálculo en la implementación física

Del análisis de la Tabla 8.5 se puede concluir que la arquitectura activa es la más conveniente en cuanto a velocidad de cálculo. Pasar de 6 a 12 bits utilizando la arquitectura pasiva tiene un impacto mínimo en velocidad (de 3.6 MFRPS a 3.0 MFRPS), comparado con la utilización de la arquitectura activa (de 22.6 MFRPS a 11.0 MFRPS). Por otro lado los diseños de 6 y 8 bits para ambas

arquitecturas se implementaron en el mismo tipo de dispositivo, por lo cual en ese caso es mejor la activa. El área de la arquitectura pasiva aumenta aproximadamente un 10% usando la arquitectura activa sin funciones de pertenencia, mientras que la velocidad aumenta más de una 300% (con picos mayores de 700%). En el caso de la arquitectura activa sin el cálculo de las funciones de pertenencia, la velocidad sigue siendo superior respecto a la arquitectura pasiva y el área requerida es similar.

8.3 – MATERIALIZACIÓN DE OTRAS APLICACIONES

Esta sección se dedica al análisis de otros ejemplos, para probar el funcionamiento de las técnicas propuestas y poner a punto las herramientas de desarrollo materializadas.

La mayoría de los ejemplos se han extraído de la sección de demostraciones del ambiente de desarrollo de FLC comercializado por Aptronix "FIDE" [Aptro]. La lista de aplicaciones es la siguiente: sistema de autoenfoque simple para cámaras fotográficas (**focus**), control de un servomotor (**servomot**), control de un péndulo invertido doble o de dos tramos (**pendulo2**), control de temperatura para un horno de fundición de vidrio (**tglass** y **tgerror**), control de temperatura para un reactor (**tplasma**), control de un sistema de enfoque para el ensamble de láser (**assembly**), control para una lavadora de ropa automática (**washing**), y control de navegación para un automóvil (**auto**). Una especificación detallada de cada aplicación puede verse en el Apéndice B.

8.3.1 – IMPLEMENTACIÓN SOFTWARE

Los resultados de las mediciones de la ejecución de los programas de los ejemplos de Aptronix, para la arquitectura pasiva y la activa, se pueden ver en la Tabla 8.6.

Con los resultados de la Tabla 8.6 se calcula la velocidad en *FRPS* y *FIPS* que se muestra en la Tabla 8.7.

	Variables Entrada	Variables Salida	Reglas	Ciclos de ejecución	Arquitectura Pasiva		Arquitectura Activa	
					Tiempo		Tiempo	
					1 Ciclo (µs)	Total (seg./100)	1 Ciclo (µs)	Total (seg./100)
Auto	4	2	36	4.294.967.296	2,0	876.539	7,7	3.320.241
Assembly	2	2	40	65.536	1,3	9	5,0	33
Focus	3	3	81	16.777.216	3,0	5.070	9,3	15.711
Péndulo2	2	1	30	65.536	1,3	9	8,2	54
Servomot	2	1	64	65.536	1,9	13	11,3	73
TGerror	2	1	49	65.536	1,6	11	7,7	51
TGlass	2	1	24	65.536	1,2	8	8,2	54
Washing	2	1	9	65.536	0,9	6	3,8	25
TPlasma	3	2	294	16.777.216	28,0	47.034	18,7	31.519

Tabla 8.6. - Tiempos de la implementación por software para los ejemplos de Aptronix

	Reglas	Arquitectura PASIVA		Arquitectura ACTIVA	
		FIPS	FRPS	FIPS	FRPS
Auto	36	490.196,08	17.647.058,82	129.366,11	4.657.179,82
Assembly	40	729.927,01	29.197.080,29	198.807,16	7.952.286,28
Focus	81	331.125,83	26.821.192,05	106.837,61	8.653.846,15
Péndulo2	30	729.927,01	21.897.810,22	121.506,68	3.645.200,49
Servomot	64	505.050,51	32.323.232,32	89.847,26	5.750.224,62
TGerror	49	598.802,40	29.341.317,37	128.534,70	6.298.200,51
TGlass	24	819.672,13	19.672.131,15	121.506,68	2.916.160,39
Washing	9	1.098.901,10	9.890.109,89	262.467,19	2.362.204,72
TPlasma	294	35.676,06	10.488.762,04	53.248,14	15.654.952,08

Tabla 8.7. - Velocidad de otros ejemplos en la implementación por software

De las materializaciones software que muestra la Tabla 8.7 se determina claramente que la arquitectura pasiva tiene un mejor rendimiento con un menor número de reglas, mientras que con un número muy grande de reglas la

arquitectura activa es claramente superior a la pasiva.

8.3.2 – IMPLEMENTACIÓN HARDWARE

Los resultados obtenidos de la implementación física de la arquitectura pasiva se pueden ver en la Tabla 8.8. Donde se utiliza un dispositivo *2S50PQ208* de 50.000 puertas de la familia *SPARTAN2*, excepto en el ejemplo *TPlasma* que se utiliza un dispositivo *2S200PQ208* de 200.000 puertas de la misma familia.

Arq. PASIVA	ÁREA								Puertas Equiv.	Frecuencia Máxima (Mhz)
	Slices		IOBs (de 140)		GCLKIOBs (de 4)		GCLKs (de 4)			
	Nº	%	Nº	%	Nº	%	Nº	%		
Auto	418 (768)	54	88	62	3	75	4	100	7.053	26,796
Assembly	473 (768)	61	75	63	3	75	4	100	7.919	27,397
Focus	712 (768)	92	83	59	3	75	4	100	11.927	28,385
Péndulo2	287 (768)	37	72	51	3	75	4	100	4.695	33,466
Servomot	357 (768)	46	72	51	3	75	4	100	6.001	30,390
TGerror	323 (768)	42	72	51	3	75	4	100	5.241	30,279
TGlass	303 (768)	39	72	51	3	75	4	100	5.015	32,191
Washing	235 (768)	30	69	49	3	75	4	100	4.239	35,486
TPlasma	2274 (2352)	96	90	64	3	75	4	100	40.407	18,966

Tabla 8.8 - Resultados de la implementación física de la arquitectura pasiva para otros ejemplos

Los resultados obtenidos para la arquitectura activa se muestran en la Tabla 8.9. Para la implementación de la arquitectura activa con funciones de pertenencia se utilizan distintos dispositivos de la familia *SPARTAN2* de acuerdo a los requerimientos de cada ejemplo: *2S50PQ208* de 50.000 puertas (*camión, péndulo, péndulo2, servomot, tgerror, tglass, washing*), *2S100PQ208* de 100.000 puertas (*assembly*) y *2S150PQ208* de 150.000 puertas (*auto, focus, tplasma*). La implementación física de *Tplasma* arquitectura activa requiere menos área que la arquitectura pasiva, por esa causa se implementa en un dispositivo de menor capacidad. Para la implementación sin funciones de pertenencia, se utiliza un dispositivo *2S50PQ208* de 50.000 puertas,

excepto para el ejemplo *focus* que cabe en un *2S100PQ208* de 100.000 puertas.

Arq. ACTIVA		ÁREA								Puertas Equiv.	Frec. Máx. (Mhz)
		Slices		IOBs (de 140)		GCLKIOBs (de 4)		GCLKs (de 4)			
		Nº	%	Nº	%	Nº	%	Nº	%		
Con funciones de pertenencia	Auto	1257 (1728)	72	59	42	2	50	2	50	16.997	7,029
	Assembly	959 (1200)	79	39	27	2	50	2	50	12.820	7,042
	Focus	1341 (1728)	77	66	47	2	50	2	50	18.241	7,264
	Péndulo2	731 (768)	95	35	25	2	50	2	50	9.733	7,169
	Servomot	750 (768)	97	32	25	2	50	2	50	10.052	6,978
	TGerror	739 (768)	96	35	25	2	50	2	50	9.807	6,741
	TGlass	628 (768)	81	35	25	2	50	2	50	8.530	7,283
	Washing	603 (768)	78	33	23	2	50	2	50	8.172	7,379
	TPlasma	1270 (1728)	73	55	39	2	50	2	50	17.136	7,193
Sin funciones de pertenencia	Auto	661 (768)	86	82	58	2	50	2	50	9.124	7,390
	Assembly	595 (768)	77	60	42	2	50	2	50	8.034	7,120
	Focus	903 (1200)	75	95	67	2	50	2	50	12.373	7,213
	Péndulo2	331 (768)	43	48	34	2	50	2	50	4.607	7,350
	Servomot	354 (768)	46	50	35	2	50	2	50	4.919	7,619
	TGerror	329 (768)	42	48	34	2	50	2	50	4.525	7,418
	TGlass	336 (768)	43	48	34	2	50	2	50	4.689	7,527
	Washing	328 (768)	42	44	31	2	50	2	50	4.552	7,431
	TPlasma	715 (768)	93	78	55	2	50	2	50	9.948	7,059

Tabla 8.9 - Resultados de la implementación física de la arquitectura activa para otros ejemplos

Con los resultados mostrados en las Tablas 8.8 y 8.9 se puede calcular la velocidad real del controlador difuso, en *FRPS* y *FIPS*, resumidos en la Tabla 8.10. Cabe recordar que para la arquitectura pasiva, hay que tener en cuenta que la ejecución de una instrucción requiere de dos ciclos de las señales de control, por lo tanto es necesario dividir la frecuencia máxima por dos.

		Cant. Reglas	Cantidad de Instrucciones	Frecuencia Máxima (Mhz)	FIPS	FRPS
Arq. PASIVA	Auto	36	238	26,796	56.294,12	2.026.588,24
	Assembly	40	196	27,397	69.890,31	2.795.612,24
	Focus	81	330	28,385	43.007,58	3.483.613,64
	Péndulo2	30	156	33,466	107.262,82	3.217.884,62
	Servomot	64	235	30,390	64.659,57	4.138.212,77
	TGerror	49	191	30,279	79.264,40	3.883.955,50
	TGlass	24	149	32,191	108.023,49	2.592.563,76
	Washing	9	103	35,486	172.262,14	1.550.359,22
	TPlasma	294	649	18,966	14.611,71	4.295.842,84
Arq. ACTIVA c /FP	Auto	36	17	7,029	413.470,59	14.884.941,24
	Assembly	40	11	7,042	640.181,82	25.607.272,80
	Focus	81	16	7,264	454.000,00	36.774.000,00
	Péndulo2	30	14	7,169	512.071,43	15.362.142,90
	Servomot	64	15	6,978	465.200,00	29.772.800,00
	TGerror	49	13	6,741	518.538,46	25.408.384,54
	TGlass	24	15	7,283	485.533,33	11.652.799,92
	Washing	9	13	7,379	567.615,38	5.108.538,42
	TPlasma	294	19	7,193	378.578,95	111.302.211,30
Arq. ACTIVA s /FP	Auto	36	17	7,390	434.705,88	15.649.411,76
	Assembly	40	11	7,120	647.272,73	25.890.909,09
	Focus	81	16	7,213	450.812,50	36.515.812,50
	Péndulo2	30	14	7,350	525.000,00	15.750.000,00
	Servomot	64	15	7,619	507.933,33	32.507.733,33
	TGerror	49	13	7,418	570.615,38	27.960.153,85
	TGlass	24	15	7,527	501.800,00	12.043.200,00
	Washing	9	13	7,431	571.615,38	5.144.538,46
	TPlasma	294	19	7,059	371.526,32	109.228.736,84

Tabla 8.10. - Velocidad de cálculo de otros ejemplos en la implementación física

En la Tabla 8.10 se puede ver que en general la arquitectura activa tiene mayor velocidad en FIPS y FRPS. Para problemas pequeños el área requerida por la arquitectura pasiva es menor que para la activa, sin embargo, si el problema tiene un gran número de reglas la situación se invierte (Note por ejemplo, que la arquitectura activa sin funciones de pertenencia de TPlasma ocupa aproximadamente un

tercio del área de la arquitectura activa, mientras que la velocidad pasa de 4.2 MFRPS a 109.2 MFRPS).

El tamaño en la cantidad de reglas tiene un impacto directo sobre el beneficio de usar una u otra arquitectura. Por ejemplo, el factor de mejora pasa de 2.0 MFRPS a 5.1 MFRPS para la lavadora (Washing con 9 reglas), mientras que para el control de temperatura (TPlasma con 294 reglas) pasa de 1.5 MFRPS a 111.3 MFRPS.

Con respecto al área se destaca que, al usar las funciones de pertenencia externas en la arquitectura activa, el área requerida por el controlador se reduce considerablemente, pudiéndose utilizar los mismos dispositivos que los utilizados para la arquitectura pasiva.

9 – CONCLUSIONES

9.1 - APORTES

Los objetivos de la tesis, expuestos en el capítulo 1, pueden resumirse en el siguiente párrafo:

> *Proponer una metodología de diseño y las herramientas para la síntesis automática de arquitecturas que materialicen controladores difusos de forma eficiente usando tecnologías FPGA.*

Las conclusiones acerca de las diferentes partes en que se ha dividido el trabajo para alcanzar los objetivos pretendidos, se han ido presentando a lo largo de la presente memoria al final de cada capítulo. En este apartado se presenta una recapitulación global de las mismas, junto con las aportaciones de la tesis.

9.1.1 – ARQUITECTURA PASIVA

Se presenta una arquitectura de controladores difusos, basada en una ruta de datos controlada por un microprograma, que usa una única unidad de proceso. Se analizan: el conjunto de señales y su sincronización, la estructura de las tablas que implementan las funciones de pertenencia y de decodificación, los bancos de registros, y por último, la unidad aritmética y lógica.

El circuito consta de una ruta de datos, adaptable mediante el uso de parámetros, controlada por un microprograma sin saltos. Se describe principalmente su estructura y el funcionamiento de la ruta de datos. Con el diseño propuesto se fabricó un ASIC usando tecnología de ES2 de una micra (Apéndice C), totalizando una equivalencia a 18858 transistores.

Se presenta una metodología, para el desarrollo de aplicaciones utilizando la arquitectura propuesta (Fig. 5.1), que se apoya en varias herramientas para la

generación automática del microprograma y del contenido de las memorias de las funciones de pertenencia y decodificación. Entre tales herramientas se describen las que asisten para los siguientes propósitos:

- Especificación de controladores difusos por medio de un lenguaje definido para tal fin;
- Generación de simuladores software (programas) que ejecutan el algoritmo de control difuso;
- Generación de un esquema de cálculo básico;
- Modificación del esquema de cálculo mediante la optimización reticular y aritmética;
- Generación de un programa con asignación óptima de registros de memoria; y finalmente,
- Generación del microprograma para controlar la ruta de datos que implementa dicho algoritmo;

9.1.2 - FUSIFICACIÓN POR SEGMENTOS

Se presenta una arquitectura digital para implementar un circuito de cálculo de las funciones de pertenencias de altas prestaciones. Se presenta una arquitectura que realiza el cálculo de las funciones de pertenencia siguiendo polinomios lineales en un esquema enfocado al cálculo. Este circuito, para el cálculo de funciones de pertenencia, soporta un método selectivo de inferencia para determinar el conjunto de reglas que se activan, de acuerdo a los valores a fusificar.

9.1.3 - ARQUITECTURA DE ALTAS PRESTACIONES

En esta tesis se propone una metodología alternativa para el desarrollo de controladores difusos, orientada hacia controladores de altas prestaciones. Las características principales de la propuesta son: el cálculo de las funciones de

pertenencia utilizando la técnica de segmentos lineales, el motor de inferencia difuso que trabaja por reglas activas, y un defusificador que utiliza un circuito especial para el cálculo de la división. La arquitectura utiliza múltiples unidades de cálculo a medida para realizar las operaciones explotando el paralelismo inherente del algoritmo de control difuso. Estas unidades son generadas automáticamente a medida de la aplicación, lo que permite reducir el número de instrucciones y ajustar el tamaño de palabra al mínimo en cada operación.

La metodología se apoya en varias herramientas para la generación automática del circuito que materializa la arquitectura, del microprograma de control, y del circuito que materializa las funciones de pertenencia o del contenido de las memorias de las funciones de pertenencia y decodificación. Entre las herramientas se destacan:

- Generación del código VHDL sintetizable para implementar las funciones de pertenencia por segmentos.
- Generación del código VHDL sintetizable para implementar el motor de inferencia por reglas activas.
- Generación del microprograma de control.
- Generación de simuladores software (programas) que ejecutan el algoritmo de control difuso.

Las ventajas del uso combinado de un fusificador por segmentos en la arquitectura de altas prestaciones son:

- Reducción del número de ciclos de reloj necesarios para calcular todas las funciones de pertenencia de la aplicación.
- Soporte de polinomios lineales definidos por un número arbitrario de puntos.
- Reducción del área de control al permitir el uso de algoritmos lineales (sin bifurcaciones).

- Reducción del ancho de la palabra de control.
- Reducción del área del circuito.
- Reducción de la memoria necesaria mediante el uso de variables cuya representación sea a medida de la aplicación.
- Reducción de la cantidad de instrucciones.

9.2 - ANÁLISIS

Las diferencias en las arquitecturas se pueden resumir en los siguientes aspectos principales: a) **Velocidad**, se utiliza el paralelismo inherente en el algoritmo de cálculo difuso, para lo cual se hace uso de múltiples unidades de cálculo. b) **Área**, las rutas de datos, el tamaño de los bloques de memoria y las unidades de cálculo son elementos realizados a medida de la aplicación. c) **Bloques de memoria**, se utilizan pequeños bloques distribuidos en el diseño con acceso restringido, la ruta de datos sólo permite almacenar el resultado de una única unidad de cálculo.

El tener unidades de cálculo totalmente a medida, evita tener que realizar múltiples operaciones de simple palabra para obtener un resultado de múltiples palabras, reduciendo el número de microinstrucciones. Los bloques de memoria distribuidos permiten simplificar la ruta de datos, reducir el tamaño del decodificador de direcciones, reducir el tamaño de la palabra de control y minimizar la cantidad de registros que no tienen utilidad real. Estos tres aspectos tienen consecuencias excelentes, al producir un diseño más compacto y rápido.

El parámetro velocidad es mejorado mucho más que la simple comparación de la frecuencia de reloj, debido a que hay que tener en cuenta que la arquitectura pasiva necesita 173 instrucciones, mientras que la arquitectura activa necesita sólo 15; para el ejemplo del camión. En la Tabla 8.10 se muestra la velocidad de las diferentes materializaciones. Comparando un diseño en la misma plataforma,

aunque la frecuencia pasa de 34-Mhz a 7-Mhz, la velocidad aumenta en un factor cercano a 5.

9.3 - CONCLUSIONES

Teniendo en cuenta los resultados mostrados a lo largo de esta memoria, las principales conclusiones son las siguientes:

- Con las tecnologías actuales solo es posible la materialización de un controlador en una FPGA de las familias más nuevas y de gran tamaño (familias Virtex o Spartan), que incorporen bancos de memoria internos.

- Las arquitecturas que han demostrado ser más eficientes son las que utilizan bancos de memoria distribuidos, y que utilizan múltiples unidades de cálculo.

- La utilidad de un fusificador por segmentos se demuestra en el caso que no se pueda poner un bloque de RAM del tamaño log2(*tamaño máximo de variable*)*(*cantidad de variables*).

- Las velocidades obtenidas en los diseños no pueden compararse a las conseguidas con tecnología ASIC, así como tampoco tiene comparación su coste considerando series muy pequeñas. Puede ser muy beneficioso la utilización de las técnicas aplicadas en esta tesis para el desarrollo de controladores ASIC de altas prestaciones.

- El conjunto de herramientas brindan al diseñador una asistencia en todas las etapas, lo que permite desarrollar una aplicación en un plazo adecuado.

9.4 - CONTRIBUCIONES A LA LITERATURA CIENTÍFICA

Los resultados obtenidos en esta tesis han dado origen a una serie de publicaciones y ponencias en congresos, tanto nacionales como internacionales, que se relacionan según su temática en cinco bloques.

9.4.1 - HERRAMIENTAS Y EXPERIENCIAS

1996-Septiembre, XXV Jornadas Argentinas de Informática e Investigación Operativa, Buenos Aires, Argentina; "*Algoritmo de Optimización de Funciones Reticulares Aplicado al Diseño de Controladores Difusos*", J-P Deschamps y N. Acosta [Des96]. Para la materialización de un sistema de control difuso es conveniente reducir las expresiones generadas a partir de los antecedentes de las reglas de inferencia a esquemas de cálculo incluyendo exclusivamente operaciones máximo y mínimo de dos operandos. En este artículo se propone un algoritmo heurístico de generación de esquemas a partir de expresiones reticulares conjuntivas o disyuntivas.

1999-Marzo, IBERCHIP V, Lima, Perú; "*Una herramienta de análisis de retardos de interconexión en FPGAs*", E. Boemo, N. Acosta y E. Todorovich [Boe99]. Se presenta una herramienta para el análisis de la interconexión de circuitos en FPGAs. A partir de los ficheros de información *post-layout*, el programa permite seleccionar subconjuntos de pistas de un determinado *fan-out*, graficar histogramas de retardos, calcular estadísticas, analizar *skew* de reloj, o exportar esta información a formatos de programas de análisis más avanzados tales como Origin o Mathlab.

1999-Marzo, IBERCHIP V, Lima, Perú; "*Prototipo de simulador HDL*", N. Acosta y H. Curti [Aco99]. En un proyecto de co-diseño es necesario contar con herramientas que permitan: a) simular el mundo real por medio de programas de aplicación, b) interactuar con hardware (real y/o emulado), y c) simular implementaciones hardware en modo comportamental. Estas tres herramientas

deben comunicarse para permitir el codiseño de un sistema completo. Este artículo propone un simulador de HDL en un entorno de codiseño.

2000-Noviembre, XV DCIS (Conference on Design of Circuits and Integrated Systems), Montpellier, Le Corum (France); "*End-user low-power alternatives at topological and physical levels. Some examples on FPGAs*", Todorovich E., Sutter G., Acosta N. & Boemo E. [Tod00]. Se analizan las condiciones esenciales para el bajo consumo en circuitos de alta velocidad sobre FPGAs: profundidad lógica, cantidad de puertas, y la red de interconexión. Se proponen opciones de diseño basadas en la modificación de la topología, de la estrategia de diseño físico o de la arquitectura (basadas en pipelining o paralelismo).

9.4.2 - GENERADORES DE CONTROLADORES DIFUSOS

1997, artículo en el libro: "*Manufacturing Systems: Manufacturing, Management and Control*", ISBN: 0-08-042616-6, Editorial: Elsevier Science, Publicado por la IFAC (*International Federation on Automatic Control*). El artículo seleccionado es: "*Customized Fuzzy Logic Controller Generator*", N. Acosta, J-P Deschamps y G. Sutter [Aco97]. La arquitectura básica de un controlador difuso usa unidades de cálculo que realizan operaciones de fusificación, defusificación, y operaciones aritméticas y reticulares. El generador incluye varias herramientas que traducen una especificación inicial del problema en un circuito específico que lo materializa y su microprograma de control. El controlador se especifica mediante descripciones en un lenguaje creado *ad-hoc*, con una sintaxis parecida a *C*.

1997-Febrero, IBERCHIP III, Méjico, D.F.; "*Herramientas para la materialización de controladores difusos microprogramados*", N. Acosta, J-P Deschamps y G. Sutter [Aco97c]. Se define un lenguaje de especificación de controladores difusos, inspirado en el FIL, que por medio de transformaciones genera el microprograma para plataformas hardware a ser materializadas en FPGAs o ASICs.

1997-Abril, AARTC'97 (Algorithms and Architectures for Real-Time Control), Organizado por la IFAC, Vilamoura, Portugal; "*Automatic Program Generator for Customized Fuzzy Logic Controllers*", N. Acosta, J-P Deschamps y G. Sutter [Aco97b]. El generador incluye varias herramientas que traducen una especificación inicial del controlador hecha en FIL, en: a) un circuito específico que materializa la arquitectura, b) el microprograma de control, c) un programa que una vez compilado permite la simulación del controlador.

1999-Marzo, IBERCHIP V, Lima, Perú; "*A high-level synthesis tool for generating fuzzy logic controllers*", N. Acosta y E. Todorovich [Aco99b]. Se propone una arquitectura dirigida por reglas para implementar controladores difusos. El generador incluye varias herramientas que permiten traducir el problema inicial a un circuito específico que lo materializa. El circuito es generado mediante una descripción HDL (tales como VHDL, Verilog, Handel-C, o ABEL). La descripción puede ser sintetizada para obtener la cadena de configuración de una FPGA.

2000-Agosto, Congreso Argentino de Control Automático, Buenos Aires, Argentina; "*Generador automático de controladores difusos*", Todorovich E., Acosta N. y Acosta G. [Tod00b]. Se utiliza un algoritmo evolutivo para la generación y puesta a punto de un controlador difuso hardware descrito en VHDL. Como aplicación se diseña un FLC de un motor de inducción.

2001-Mayo, WICC (Workshop de Investigadores en Ciencias de la Computación), San Luis, Argentina; "*Controladores Difusos de altas prestaciones: sistema para la generación automática*", N. Acosta [Aco01]. Se presenta un sistema de generación automática de controladores difusos software para PC.

2001-Octubre, CACIC (Congreso Argentino de Ciencias de la Computación), El Calafate, Argentina; "*Materialización de Controladores Difusos Activos*", N. Acosta [Aco01c]. Se presentan técnicas que permiten el desarrollo de FLC de altas prestaciones usando reglas activas. Se propone fusificación y defusificación activa.

9.4.3 - ARQUITECTURAS DE CÁLCULO

1998-Febrero, EIS'98 (International Symposium on Engineering of Intelligent Systems), Tenerife, España; "*Optimized active rule fuzzy logic custom controller architecture synthesis*", N. Acosta, J-P Deschamps y J. Garrido [Aco98]. Este artículo presenta un esquema de cálculo optimizado para el motor de inferencia difuso de un controlador. Se propone un esquema dirigido por reglas activas, que calcula sólo las reglas relevantes. Se propone una metodología para la síntesis automática de controladores difusos usando este motor de cálculo.

1998-Febrero, EIS'98 (International Symposium on Engineering of Intelligent Systems), Tenerife, España; "*Segment representation for membership-functions*", N. Acosta, J. Garrido y J-P Deschamps [Aco98b]. Este artículo muestra como implementar circuitos de cálculo de funciones de pertenencia, descriptas por polinomios lineales con un número variable de puntos. Este circuito soporta un método de inferencia selectivo, para determinar el conjunto de reglas difusas activas.

1998-Marzo, IBERCHIP IV, Mar del Plata, Argentina; "*Membership function and inference rule implementation*", N. Acosta, J-P Deschamps y J. Garrido [Aco98e]. Se propone una generalización del método de representación de funciones de pertenencia, agregando segmentos inútiles en vía de poder representar un mayor número de funciones de pertenencia.

2000-Marzo, IBERCHIP VI, Sao Paolo, Brasil; "*Multiplicadores paralelos: estado del arte y análisis de su materialización en FPGA*", N. Acosta, E. Todorovich, C. Collado y K. Larsen [Aco00]. En la primer sección se muestra y analizan siete multiplicadores paralelos (con y sin *pipelining*); Mientras que la segunda muestra los resultados de 700 materializaciones en FPGA. Se analizan parámetros tales como: área, velocidad, puertas equivalentes, frecuencia máxima de operación y cantidad de *flip-flops* necesarios.

2002, Journal of Computer Science & Technology nro 7; "*Custom Architectures for Fuzzy and Neural Networks Controllers*", N. Acosta, y M.

Tosini [Aco02]. Se presentan enfoques arquitecturales para el diseño de controladores usando circuitos específicos y herramientas para la generación automática de la descripción VHDL.

9.4.4 - ARQUITECTURAS EN FPGAS

1998-Marzo, IBERCHIP IV, Mar del Plata, Argentina; "*Diseño de sistemas digitales específicos basados en hardware reconfigurable*", N. Acosta y E. Todorovich [Aco98d]. Este artículo presenta un análisis de implementaciones software/hardware basadas en FPGAs acopladas a una computadora personal. Se implementan tres aplicaciones: a) generador de números de Fibonacci, b) búsqueda binaria y c) unidad aritmético-lógica para controladores difusos. Se usan distintos anchos de palabra, para determinar el grado de adaptabilidad a la arquitectura soporte.

1998-Agosto, XXVI Congreso Argentino de Control Automático, Buenos Aires, Argentina; "*Implementación de controladores difusos sobre FPGAs*", N. Acosta y E. Todorovich [Aco98c]. Analiza implementaciones software/hardware basadas en FPGAs acopladas a una computadora personal. Se implementa una UAL para un controlador difuso mediante dos técnicas: síntesis lógica a partir de VHDL y diseños específicos, para distintos anchos de palabra, de tal forma que se pueda determinar el grado de adaptabilidad a la arquitectura soporte, y el desempeño de la herramienta de síntesis.

1999-Marzo, IBERCHIP V, Lima, Perú; "*Co simulación de Hw/Sw usando FPGAs*", E. Todorovich, N. Acosta y C. Vazquez [Tod99]. Se propone sintetizar una descripción en VHDL para configurar una FPGA, mientras que el resto del sistema o entorno es descrito en ADA. La aplicación en ADA se comunica con una estación de trabajo mediante tarjetas de entradas y salidas digitales conectadas a la FPGA utilizando software basado en TCP/IP. Se describe el protocolo de comunicación utilizado, el *driver* NT del servidor de entradas y salidas digitales, y el paquete ADA para utilizar el canal de comunicación.

1999, XIV DCIS (Conference on Design of Circuits and Integrated Systems), Palma de Mallorca, España; "*Local versus global interconnections in pipelined arrays: an example of interaction between architecture and technology*", E. Boemo, S. López-Buedo, N. Acosta y E. Todorovich [Boe99b]. Este artículo muestra el efecto del uso de líneas de conexión local y global en parámetros tales como el área, la latencia y la frecuencia de funcionamiento efectiva. Se trabaja sobre la topología de un multiplicador con pipelining en diferentes direcciones.

2000-Abril, ICIE (Congreso Internacional en Ing. Informática), Buenos Aires, Argentina; "*Motores de Inferencia para Controladores Difusos: análisis de materializaciones hardware*", N. Acosta y H. Curti [Aco00b]. Este artículo presenta un análisis de diversas materializaciones de motores de inferencia difusos sobre plataformas hardware a medida. Se analizan parámetros tales como: tiempo de cálculo, frecuencia de operación máxima, cantidad de conexiones a rutar, cantidad de puertas lógicas equivalentes, cantidad de unidades de almacenamiento, cantidad de señales de control, cantidad de instrucciones del microprograma de control y velocidad máxima.

2003-Septiembre, Jornadas Sobre Computación Reconfigurable y Aplicaciones, Madrid, España; "*Adaptación de la Arquitectura de un Microcontrolador Estándar orientado a FPGA para el soporte de Algoritmos Difusos*", N. Acosta, M. Vázquez, E. Todorovich, E. Boemo y D. Simonelli [Aco03]. En este trabajo se implementa un FLC dedicado en una FPGA, partiendo de la especificación de un procesador estándar bien definido. Se busca un controlador más rápido (hardware y software) sin gran esfuerzo en el diseño.

9.4.5 - DISEÑO DE APLICACIONES

"*Control de un vehículo autónomo entre paredes*", patentado en agosto de 1998. Número en el Registro de Propiedad Intelectual: M-73385. Número de inscripción: 00 / 1998 / 20864. Se registra un controlador difuso que permite guiar un vehículo (de forma autónoma y eficiente), utilizando para ello sensores ópticos de detección de obstáculos.

2001-Octubre, CACIC (Congreso Argentino de Ciencias de la Computación), El Calafate, Argentina; "*Diferentes arquitecturas aplicadas a la implementación de un Controlador Difuso para el péndulo invertido*", N. Acosta y D. Simonelli [Aco01b]. Se evalúan cinco diferentes arquitecturas obteniéndose métricas del desempeño para el control del péndulo invertido.

2003-Octubre, CACIC (Congreso Argentino de Ciencias de la Computación), La Plata, Argentina; "*Estacionamiento Automático de un Vehículo Autoguiado usando Lógica Difusa*", N. Acosta, C. Aciti, y M. Berlusconi [Aco03b]. Este trabajo estudia y aplica técnicas de control basadas en lógica difusa, para la materialización de un modulo dedicado al estacionamiento paralelo automático.

2004-Mayo, WICC (Workshop de Investigadores en Ciencias de la Computación), Neuquen, Argentina; "*Identificación por hardware de posición de jugadores en futbol de robots*", M. Tosini, N. Acosta, C. Aciti, y S. Iarrar [Tos04]. Se presenta un prototipo de procesador a medida para la identificación de patrones en tiempo real sobre una plataforma FPGA. Se proponen dispositivos orientados a la velocidad y aplicabilidad en entornos industriales. Se usa lógica difusa y redes neuronales.

9.5 - LÍNEAS DE TRABAJOS FUTUROS

De continuar con las arquitecturas para control difuso las opciones seleccionadas parecen prácticamente agotadas, sin embargo, el estudio de alternativas de realización de arquitecturas que consideren otras opciones, tales como motores de inferencia, circuitos de fusificación, y algoritmos de defusificación, constituyen un tema de investigación atractivo.

Un punto no analizado y fundamental para el diseño de controladores difusos de altas prestaciones es el circuito de defusificación. Una vez resuelta por segmentos la fusificación, es la defusificación donde se utiliza la operación que más tiempo consume de todo el algoritmo: la *división*. Los controladores emplean algoritmos de división por etapas sucesivas, que pueden ser reemplazados por

divisores rápidos basados en tablas de búsqueda (LUT), cuyas salidas hayan sido previamente calculadas, para luego interpolar y realizar ajustes.

Queda sin explorar el realizar modificaciones a las arquitecturas propuestas aplicando técnicas de segmentación (*pipeline*). Esta opción es prometedora, conociendo la ventaja en no tener que tratar *saltos condicionales*, debido a que el algoritmo o esquema de cálculo es completamente definido sin necesitar saltos.

APÉNDICE A: HERRAMIENTA DE SÍNTESIS

A.1 - INTRODUCCIÓN

Esta sección está dedicada a la descripción de la herramienta materializada para demostrar las técnicas propuestas para la síntesis de alto nivel de los FLC propuestos.

A continuación se describen algunas características de la herramienta:

- **Integración**. Permite integrar todas las herramientas de ambas propuestas en una única aplicación.
- **Generación de los contenidos de las memorias externas**. El sistema permite la generación del contenido de todas las memorias externas. Esto incluye las funciones de pertenencia por tablas o tablas optimizadas en espacio y también el microprograma de control.
- **VHDL para síntesis de las dos arquitecturas propuestas**. En un único paso se generan todos los módulos VHDL de ambas arquitecturas (pasiva y activa).
- **Simulación del controlador**. Se realiza la ejecución del esquema normal de la arquitectura pasiva, de tal forma de contar con una herramienta que permita validar los resultados en la etapa de diseño o prueba.
- **Código compilable C y Pascal del controlador**. Se genera código del controlador en C y Pascal para permitir la depuración y ajuste del controlador.

- **Código C o Pascal de las funciones de pertenencia**. Se genera código C y Pascal que materializa las funciones de pertenencia, lo que permite la rápida depuración y adaptación del formato de los datos de las memorias al requiera el sistema de grabación.

A.2 – INTERFAZ DEL SISTEMA

El sistema desarrollado presenta una ventana que incluye un menú principal (con las opciones Archivo, Edición, Ver, Compilar, Simulación, Herramientas, Ventana y Ayuda). Donde:

- **Archivo** permite crear, abrir, cerrar, guardar, guardar con otro nombre, o imprimir un archivo de definición de FLC.
- **Edición** permite acceder a las herramientas de edición clásicas (copiar, pegar, cortar, seleccionar todo, buscar o buscar y reemplazar).
- **Ver** permite configurar las herramientas (herramientas de archivo, de edición o de ventanas) y la ventana de mensajes haciendolas o no visibles.
- **Compilar** permite activar el análisis del archivo de definición y la generación de todos los archivos de forma automática.
- **Simulación** permite simular el FLC sobre la computadora, facilitando la depuración dentro de la herramienta.
- **Herramientas** permite guardar uno o varios archivos generados (pascal, c, datos, microprogramas, vhdl, etc.).
- **Ventana** permite organizar las ventanas dentro de la aplicación.

La ventana principal de la aplicación se muestra dividida en dos partes (Fig. A.1). La ventana izquierda muestra un arbol con la jerarquía de archivos generados a partir del archivo de definición. La ventana derecha muestra el

archivo actual selecionado (por defecto se muestra el archivo de definición). Cada rama del árbol puede expandirse o contraerse (dinámicamente) para enfocar la atención a la parte del diseño en que se esté trabajando en ese momento.

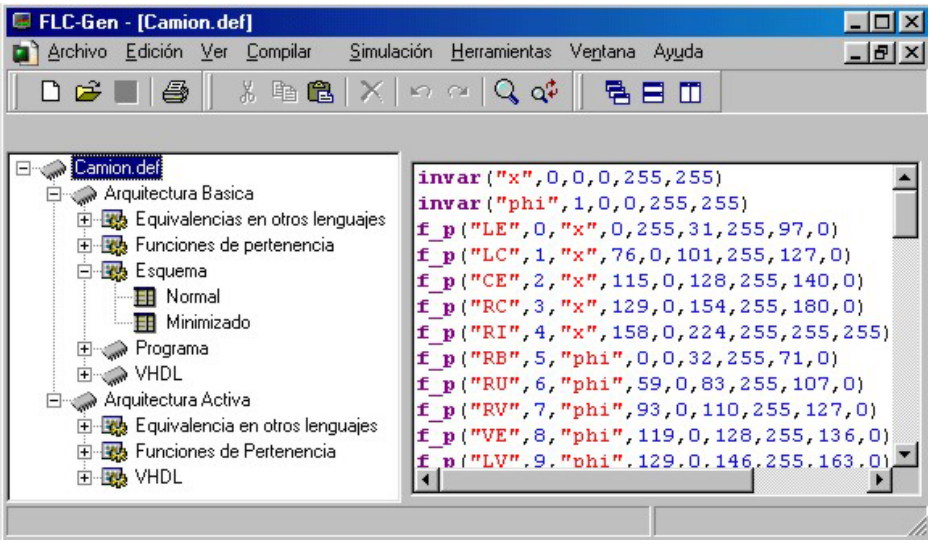


Fig. A.1 – Pantalla principal del sistema en un archivo de definición de FLC

La jerarquía se compone del siguiente árbol:

- Archivo de definición del FLC (.DEF).
  - **Arquitectura Básica o PASIVA**, con las opciones: Equivalencias en otros lenguajes del FLC (C y Pascal), Funciones de pertenencia en lenguajes C y Pascal o memorias (un banco por función de pertenencia o ***O*** bancos), Esquema de cálculo (normal y minimizado), Programa (microprograma en ensamblador o en binario, informe del microprograma), y VHDL (todos los módulos VHDL que componen el circuito de la arquitectura pasiva).
  - **Arquitectura ACTIVA**, con las opciones: Equivalencias en otros lenguajes del FLC (C y Pascal), Funciones de pertenencia en lenguajes C y Pascal o memorias (un banco por función de

pertenencia o ***O*** bancos), VHDL (), y VHDL (todos los módulos VHDL que componen el circuito de la arquitectura activa y el microprograma en binario).

Una vez ejecutado el sistema, permite abrir un archivos (.DEF) con la definición del FLC. Ese archivo debe ser compilado para permitir que el sistema analice la especificación (sintáctica y semánticamente), y si no hay errores se procede a la generación de todas las alternativas de materialización y simulación. Una vez que se ha compilado, se puede navegar por el árbol de vistas para ver las diferentes alternativas analizadas (en la ventana derecha de la aplicación).

A.3 – ARQUITECTURA PASIVA

A.3.1 – FUNCIONES DE PERTENENCIA

La Fig. A.2 muestra la gráfica para todas las funciones de pertenencia de todas las variables. Al seleccionar la carpeta adecuada se muestra la gráfica de cada una de las variables: ***X*** (superior) y ***PHI*** (inferior).

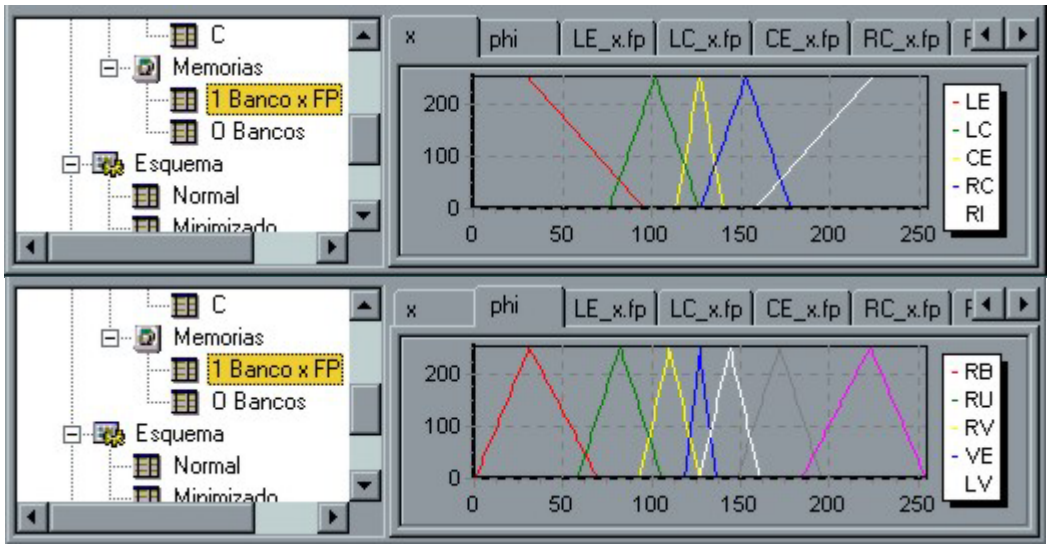


Fig. A.2 – Gráficos de las funciones de pertenencia de las variables de la aplicación base

A.3.2 – UN BANCO DE MEMORIA POR FUNCIÓN

Esta opción permite analizar y obtener los datos de cada una de las funciones de pertenencia por separado. La Fig. A.3 muestra la gráfica para la función de pertenencia ***LE*** de la variable ***X*** (en la parte superior la gráfica mientras que

los datos de la tabla se muestran abajo).

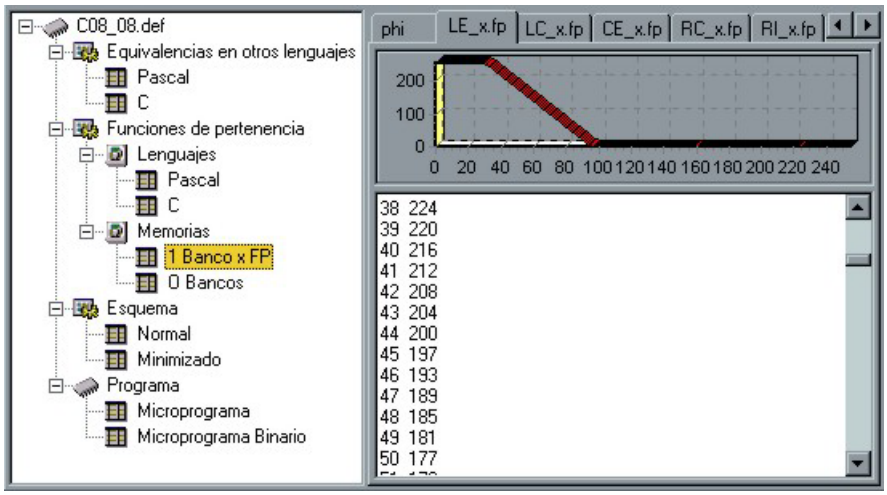


Fig. A.3 – Interfaz de generación de una funciones de pertenencia por banco

A.3.3 – O BANCOS DE MEMORIA

Esta opción permite generar los ***O*** bancos de memoria en que se pueden comprimir dichas funciones de pertenencia de acuerdo a cómo se superpongan. La Fig. A.4 muestra la gráfica para las funciones de pertenencia ***LE***, ***CE*** y ***RI*** de la variable ***X*** en un único banco de memoria (en la parte superior la gráfica, mientras que los datos de la tabla se muestran abajo con el formato: posición valor).

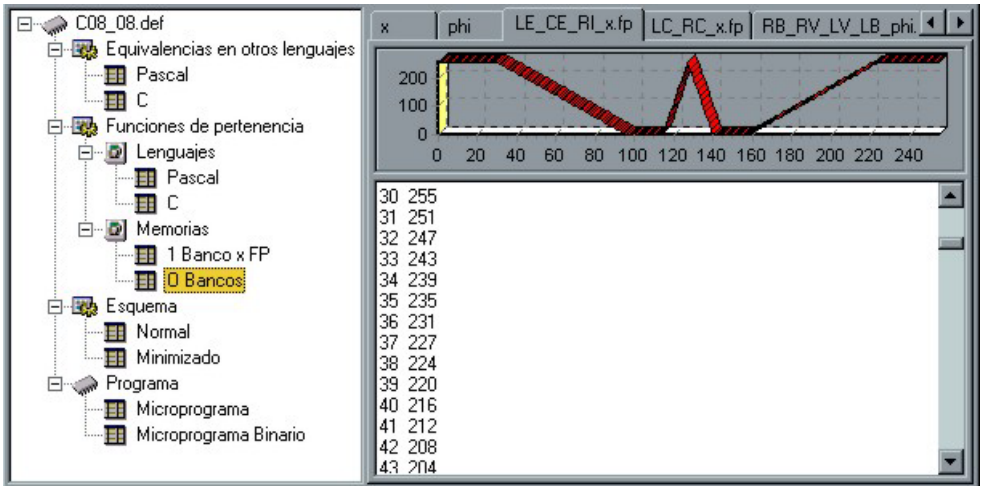


Fig. A.4 – Interfaz de generación de O funciones de pertenencia por banco

A.3.4 – LENGUAJES DE PROGRAMACIÓN

A partir del código con la especificación del FLC se generan versiones C y Pascal del controlador completo (Fig. A.5) y del cálculo de las funciones de pertenencia (Fig. A.6). De tal forma que permita generar el formato deseado de memorias para las funciones de pertenencias, o permitir realizar ajustes durante la etapa de definición del FLC mediante algún sistema de simulación.

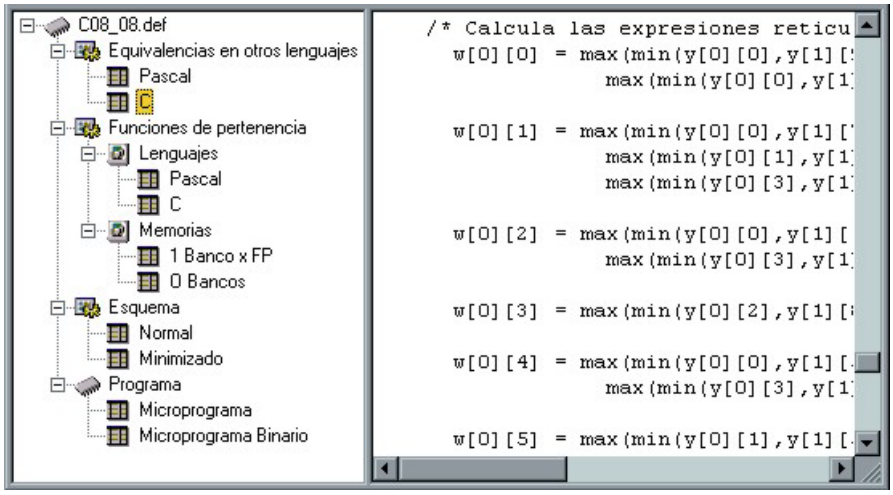


Fig. A.5 – Interfaz al código C o Pascal generado (C en este caso)

La Fig. A.5 muestra una porción de código C que materializa el programa principal del FLC durante el cálculo de la inferencia del ejemplo. La Fig. A.6 muestra el código Pascal que materializa la función de pertenencia ***CE***.

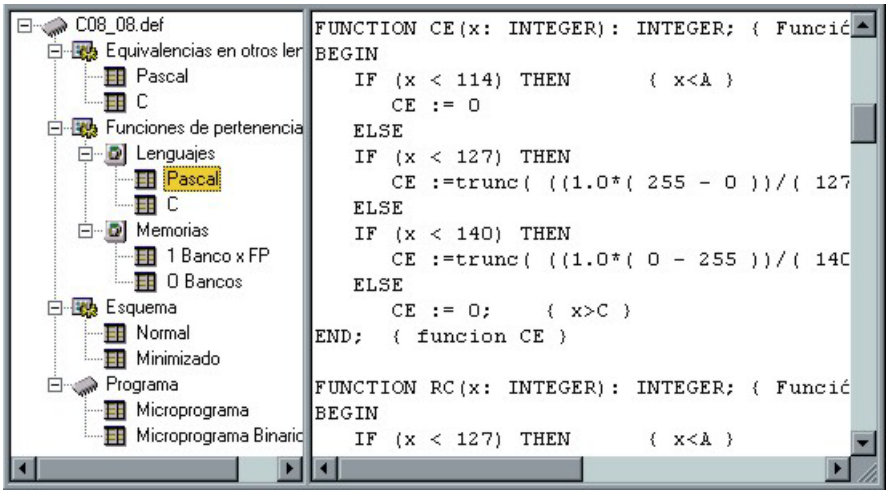


Fig. A.6 – Código Pascal que materializa las funciones de pertenencia

A.3.5 – MICROPROGRAMA

Como se ha planteado en capítulos anteriores, el sistema genera un esquema básico, que luego es minimizado, para luego generar el microprograma (Fig. A.7) y luego la representación binaria del microprograma (Fig. A.8). Estas cuatro representaciones del control del FLC pueden verse en las dos últimas opciones del árbol (equina inferior izquierda de la Fig. A.6).

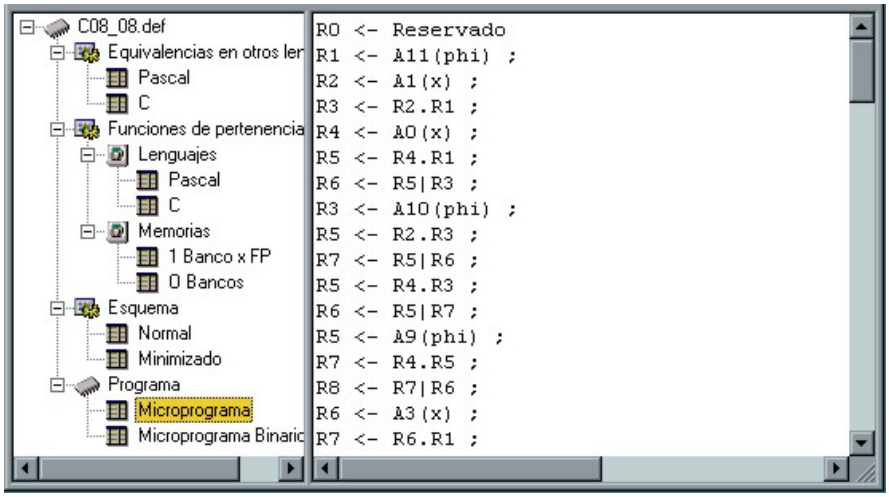


Fig. A.7 – Microprograma de la Arq. Pasiva en PSeudo Assembler

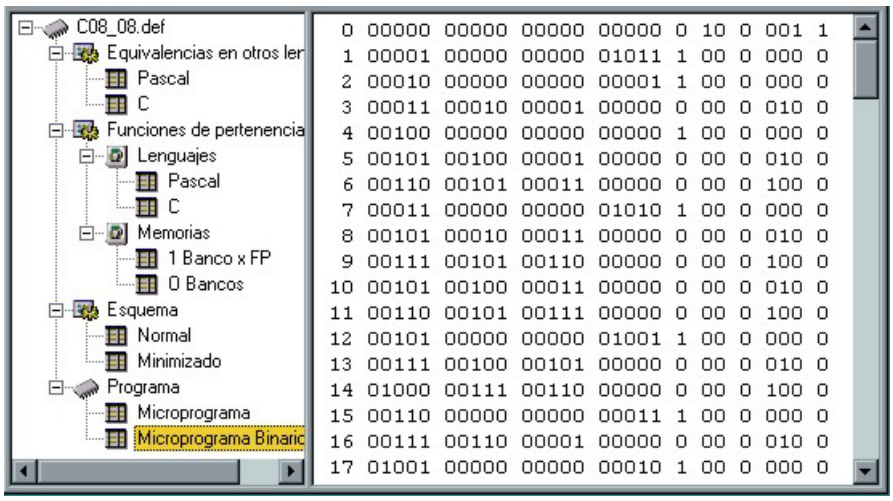


Fig. A.8 – Microprograma Arquitectura Pasiva en Binario

A.3.6 – GENERACIÓN DE VHDL

Si bien en el caso de la arquitectura Pasiva la descripción VHDL puede ser totalmente paramétrica, debido a que sólo pueden cambiar los anchos de las rutas de datos. El sistema realiza la generación de dicho código para permitir su integración en el ambiente de manera transparente. Se genera VHDL para el módulo principal (Fig. A.9), la unidad aritmético lógica, el banco de registros, las tablas y la memoria de doble puerto.

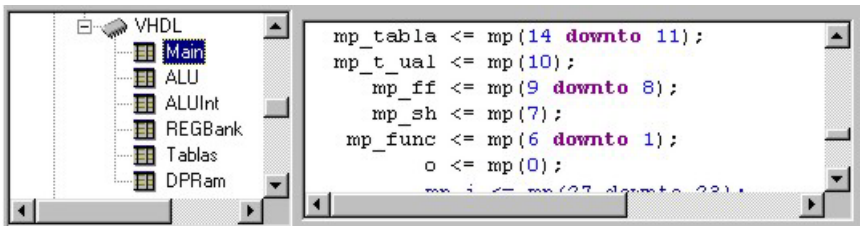


Fig. A.9 – VHDL de la Arq. Pasiva, módulo MAIN

A.4 – ARQUITECTURA ACTIVA

A.4.1 – FUNCIONES DE PERTENENCIA

La Fig. A.10 muestra la gráfica de las funciones de pertenencia de la variable ***X***. Además se puede apreciar como se agrupan las ***O*** funciones para ahorrar memoria del sistema.

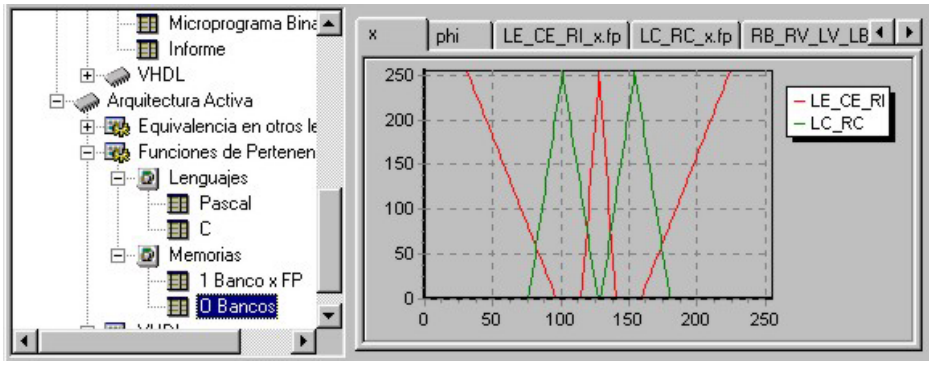


Fig. A.10 – Bancos de memoria, todas las funciones de pertenencia de la variable ***X***.

Los gráficos a mostrarse pueden ser seleccinoados por medio de las persianas; por otra parte, las persianas visibles pueden ser desplazadas por medio de los botones (en la derecha). La Fig. A.11 muestra uno de los ***O*** bancos de memoria,

en este caso se almacenan las funciones de pertenencia ***LE***, ***CE*** y ***RI***..

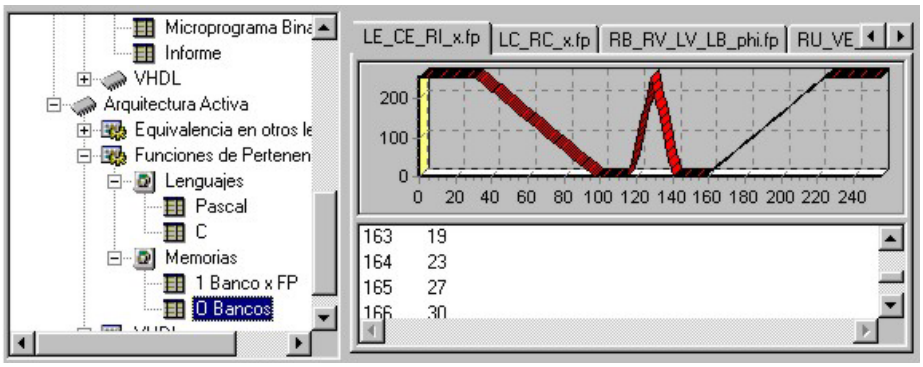


Fig. A.11 – ***O*** bancos de memoria para almacenar las 5 funciones de pertenencia de la variable ***X***.

A.4.2 – OTROS LENGUAJES

La Fig. A.12 muestra el código Pascal generado para la materialización de la arquitectura activa; en este caso se ve una sección del procedimiento principal, donde se realiza la fusificación y se comienza con la inferencia.

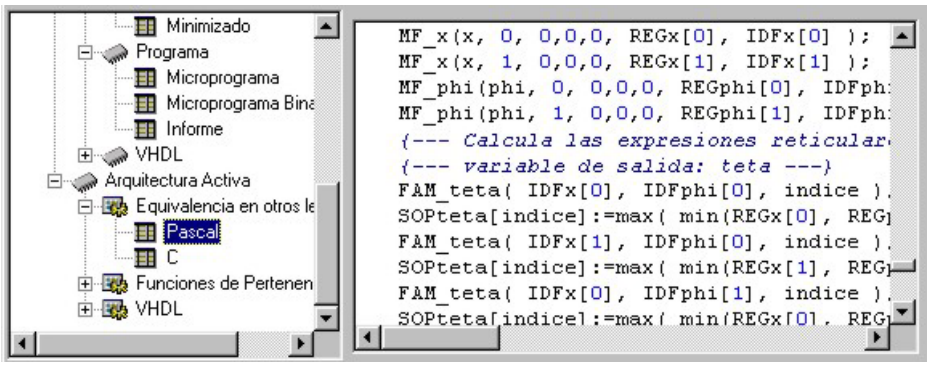


Fig. A.12 – Código Pascal de la Arq. Activa

A.4.3 – GENERACIÓN DE LA DESCRIPCIÓN VHDL

Una de las principales ventajas de esta herramienta es la generación de VHDL que sea sintetizado automáticamente por alguna herramienta del mercado. El árbol de la arquitectura activa, muestra como hojas los módulos que componen dicha implementación. Donde:

- Main es el módulo principal.
- Multiplicador materializa los multiplicadores necesarios, por tamaño y se seleccionan por la persiana.
- Celda es la unidad replicada para formar el multiplicador.
- Divisor contiene los circuitos de división necesarios (Fig. A.13), en caso de necesitar de varios tamaños, se selecciona utilizando las persianas.
- Microprograma (Fig. A.15).
- Microfunciones todos los circuitos para el cálculo de las funciones de pertenencia (Fig. A.14).

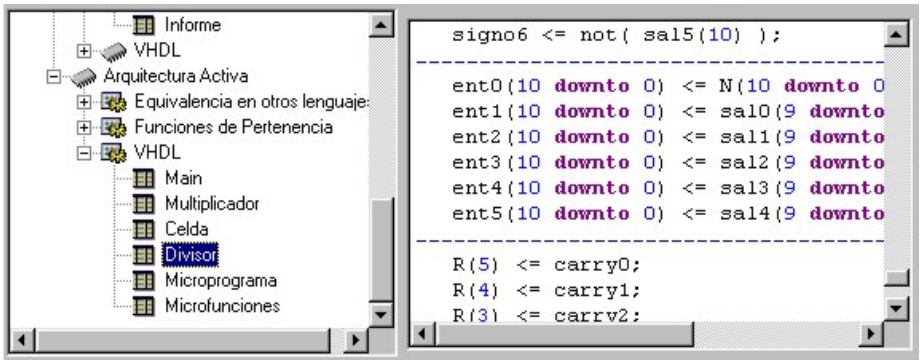


Fig. A.13 – Código VHDL generado. Divisor.

Hay módulos VHDL cuya cantidad y nombres dependen directamente de la aplicación, divisores, multiplicadores y cálculo de funciones de pertenencia. La Fig. A.14 muestra el código VHDL que materializa las funciones de pertenencia de las variables ***X*** y ***PHI***, donde se selecciona mediante las persianas.

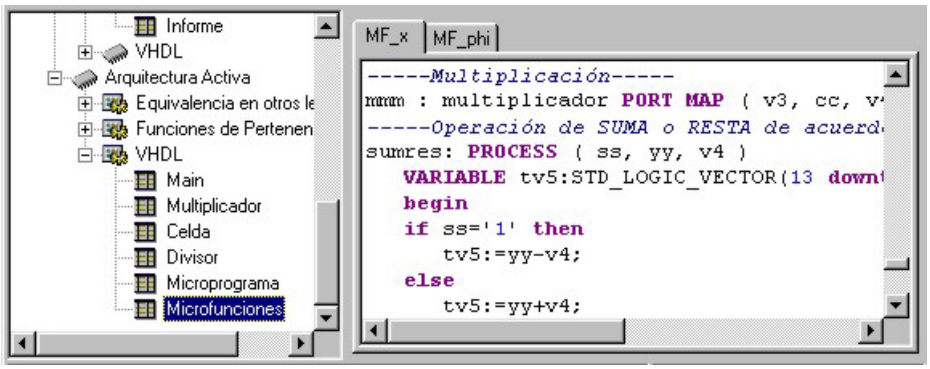


Fig. A.14 – Código VHDL generado.-Microfunciones.

A.4.4 – MICROPROGRAMA

El sistema genera el contenido de la memoria del microprograma (Fig. A.15).

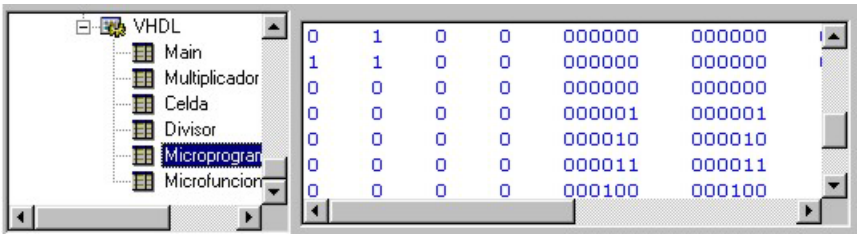


Fig. A.15 – Microprograma de la arquitectura activa

A.5 – HERRAMIENTA DE SIMULACIÓN

La herramienta de simulación del FLC permite ejecutar el controlador definido sobre la opción arquitectural propuesta. La Fig. A.16 muestra la ventana del simulador que se basa en la arquitectura pasiva. En el simulador se pueden definir los valores de entrada de cada variable para calcular el valor de las variables de salida. El panel derecho muestra cada una de las etapas de cálculo y su valor actual; que puede recorrerse para todo el esquema de cálculo. Este simulador permite también realizar la ejecución paso a paso y detener o reanudar el cálculo de forma interactiva.

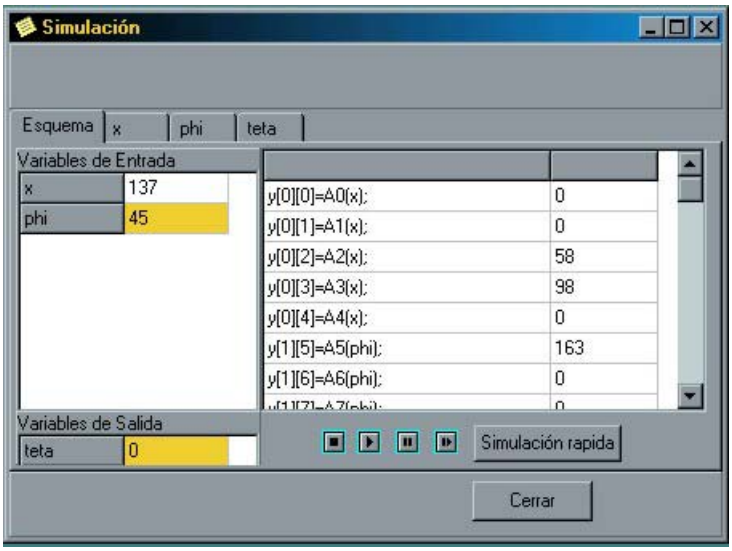


Fig. A.16 – Interfaz del simulador de la arquitectura pasiva

APÉNDICE B: OTROS CONTROLADORES

Esta sección está dedicada a la descripción de otros ejemplos utilizados para probar el funcionamiento de las técnicas propuestas y poner a punto las herramientas de desarrollo materializadas. Estos ejemplos se proveen sólo como demostración de sistemas de control simples, con el objetivo de probar sistemas de desarrollo. Para sistemas de control industrial es inapropiado el uso de estas descripciones de FLC. Por otro lado, no se ha realizado un estudio detallado con respecto a su funcionamiento.

Los ejemplos se han extraido de la sección de demostraciones del ambiente de desarrollo de FLC comercializado por Aptronix "FIDE". Por mayor información: Aptronix Incorporated; 2040 Kington Place; Santa Clara, CA 95051; Tel (408) 261-1898; Fax (408) 490-2729; FuzzyNet: *http://www.aptronix.com/fuzzynet*.

Las aplicaciones descritas en este apéndice permiten aplicar las técnicas propuestas en algunos ejemplos creados para demostración de herramientas comerciales. La lista es la siguiente: autoenfoque simple para cámaras fotográficas (b1: **focus**), control de un servomotor (b2: **servomot**), control de un péndulo invertido doble o de dos tramos (b3: **pendulo2**), control de temperatura para un horno de fundición de vidrio (b5: **tglass** y b6: **terror**), control de temperatura para un reactor (b7: **tplasma**), control de un sistema de enfoque para el ensamble de láser (b8: **assembly**), control para una lavadora de ropa automática (b9: **washing**), y control de navegación para un automóvil (b10: **auto**).

B.1 - AUTOENFOQUE

Las cámaras utilizan usualmente un método de ajuste automático del enfoque basándose en la medida de la distancia al centro de la vista del objetivo. Este método, por otra parte, es erróneo cuando el objeto de interés no está al

centro del objetivo (Fig. B.1). Trabajar con más de una distancia es una técnica que puede resolver éste problema. Este ejemplo muestra la aplicación de inferencia difusa como principal función la de determinar la distancia correcta al foco.

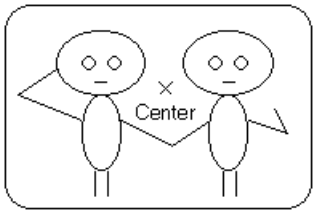


Fig. B.1 – Enfoque basado en la distancia al centro

El objetivo es determinar la distancia al objeto usando tres mediciones de distancia para un sistema de enfoque automático de una cámara.

Las variables de entrada al motor de inferencias difusas son las tres distancias medidas en puntos a izquierda, centro y derecha de la vista del ocular. Las salidas son los valores asociados con estos tres puntos (Fig. B.2). El punto con la más alta verosimilitud es considerado como el objeto de interés, y su distancia es enviada al sistema de enfoque.

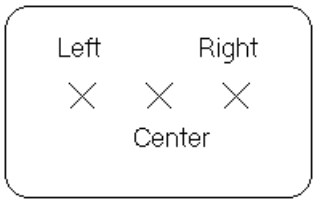


Fig. B.2 – Medición de tres distancias

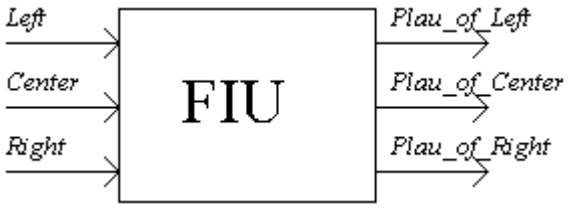


Fig. B.3 – Motor de inferencias difusas[1]

El sistema es representado con tres variables de entrada y tres de salida (Fig. B.3). Cada variable de entrada representa la distancia según sus tres funciones de

[1] FIU = Fuzzy Inference Unit, o Motor de Inferencias Difusas

pertenencia: ***Near***, ***Medium*** y ***Far***. Cada variable de salida, representa la verosimilitud, utilizando para ello cuatro funciones de pertenencia: ***Low***, ***Medium***, ***High*** y ***VeryHigh***. Las funciones de pertenencia correspondientes a estos rótulos se muestran en las gráficas Fig. B.4 y Fig. B.5.

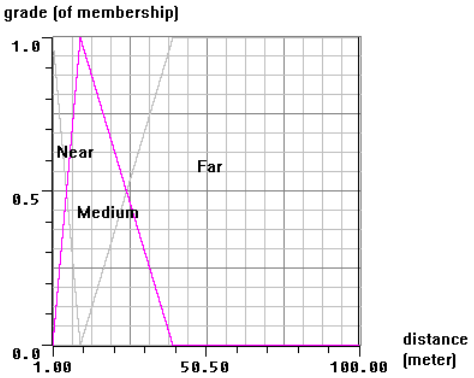


Fig. B.4 – Funciones de pertenencia de ***distancia***

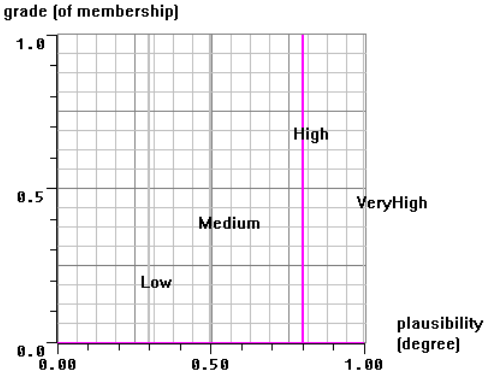


Fig. B.5 – Funciones de pertenencia de las variables de salida ***Plausibility***

El principio guía para establecer las reglas de este sistema de enfoque automático es la probabilidad de que el objeto esté a una distancia media sea alta (típicamente a 10 metros), y se reduce a medida la distancia se incrementa (más de 40 metros). El conjunto de reglas que define el comportamiento del FLC es mostrado en la Tabla B.1.

SI	(cen=cFAR)	**ENTONCES**	pc=cLO
SI	(cen=cMED)	**ENTONCES**	pc=cHI
SI	(cen=cNEA)	**ENTONCES**	pc=cME
SI	(lef=lMED) Y (cen=cFAR)	**ENTONCES**	pc=cLO
SI	(rig=rMED) Y (cen=cFAR)	**ENTONCES**	pc=cLO
SI	(lef=lFAR) Y (cen=cFAR) Y (rig=rFAR)	**ENTONCES**	pc=cHI
SI	(lef=lMED) Y (cen=cMED) Y (rig=rMED)	**ENTONCES**	pc=cVH
SI	(lef=lNEA) Y (cen=cNEA) Y (rig=rNEA)	**ENTONCES**	pc=cHI

SI	(lef=lFAR)	**ENTONCES**	pl=lLO
SI	(lef=lMED)	**ENTONCES**	pl=lHI
SI	(lef=lNEA)	**ENTONCES**	pl=lME
SI	(lef=lMED) Y (cen=cMED)	**ENTONCES**	pl=lLO
SI	(lef=lNEA) Y (cen=cNEA)	**ENTONCES**	pl=lLO
SI	(rig=rFAR)	**ENTONCES**	pr=rLO
SI	(rig=rMED)	**ENTONCES**	pr=rHI
SI	(rig=rNEA)	**ENTONCES**	pr=rME
SI	(rig=rMED) Y (cen=cMED)	**ENTONCES**	pr=rLO
SI	(rig=rNEA) Y (cen=cNEA)	**ENTONCES**	pr=rLO

Tabla B.1 – Conjunto de reglas del autoenfoque

B.2 - CONTROL DE UN SERVOMOTOR

Los servomotores son ampliamente utilizados para control en sistemas de fabricación automática. El objetivo de control puede ser la posición, la velocidad o la fuerza, entre otras. Para esta aplicación de ejemplo se tomará la fuerza como la variable controlada.

Para implementar el control de fuerza, se necesita conocer la dependencia (respuesta) del objeto controlado a la fuerza. La ganancia de retroalimentación en los bucles de control cambia como una función de dependencia.

Sujetar un amplio rango de objetos, desde una liviana pelota de tenis hasta una pesada pelota de acero, usando técnicas de control para servos convencionales es una tarea difícil. Los modelos de control tradicional no manejan muy bien una gran variedad de objetos con diferentes características materiales. Así, el sistema puede tornarse inestable. La lógica difusa, con su flexibilidad, puede ser empleada efectivamente como una alternativa efectiva en dicha situación.

El objetivo de control es agarrar objetos de varias características, por ejemplo en el rango de una pelota de tenis a una pelota de acero, con una constante de fuerza.

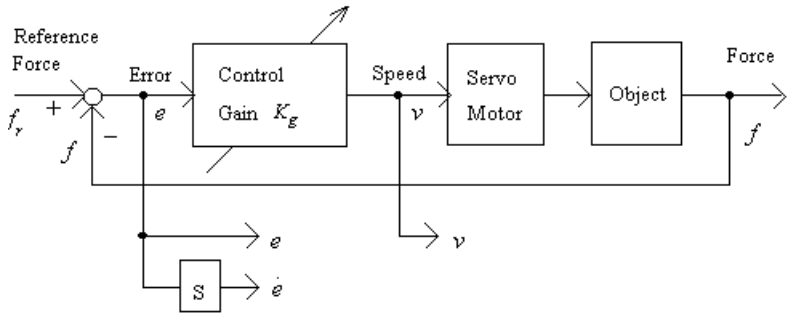


Fig. B.6 – Control de fuerza de un servomotor

La Fig. B.6 muestra el diagrama del bloque de control. La fuerza de salida, aplicada al objeto es medida por un sensor y comparada contra una fuerza de referencia para obtener la diferencia. Una ganancia de control K***g*** es aplicada para disminuir la diferencia de las fuerzas. Esta ganancia también varía como una función dependiente del objeto que se ha agarrado. Así, el control de ganancia K***g*** es afectado por dos factores: (1) la dependencia de las características del objeto, y (2) la diferencia entre una fuerza de referencia y la fuerza medida. El error ***e*** y la velocidad ***v***, del diagrama anterior representan variables a ser usadas para determinar el control de ganancia. El control de ganancia y los componentes del diagrama son mostrados en la Fig. B.7.

$$\mathrm{K}\boldsymbol{g} = \mathrm{K}\boldsymbol{s} \cdot \mathrm{K}\boldsymbol{f}$$

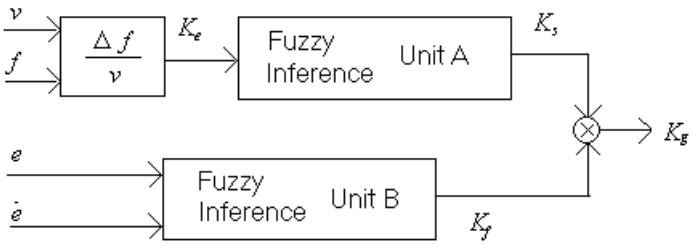


Fig. B.7 – Ganancia de control

K***s*** (componente dependiente) es una función de K***e*** • K***f*** (componente de fuerza) es una función de error ***e*** y su derivada de tiempo ***é***. Ambos pueden ser inferidos por medio de lógica difusa.

La dependencia de K***e*** es determinada por la inyección de velocidad controlando ***n*** en el servomotor y midiendo la fuerza de salida ***f***. Esta dependencia

se expresa así:

$$K\boldsymbol{e} = d\boldsymbol{f} / d\boldsymbol{x} = (d\boldsymbol{f} / dt) / (d\boldsymbol{x} / dt) = D\,\boldsymbol{f} / \boldsymbol{n}$$

Así obtenemos:

$$K\boldsymbol{e} = (\boldsymbol{f}\,\boldsymbol{k} - \boldsymbol{f}\,\boldsymbol{k}\text{-}1) / \boldsymbol{n}\,\boldsymbol{k}\text{-}1$$

La dependencia puede describirse como un cambio en la fuerza (d***f***) requerido para una deformación (d***x***) de un objeto. Por ejemplo, una pelota de tenis tiene una gran dependencia porque la fuerza necesaria para iniciar la deformación es pequeña, pero incrementa significativamente cuando el proceso de deformación procede. El cambio en la fuerza, desde la iniciación hasta la terminación, es grande. Por el otro lado está la pelota de acero, que tiene una dependencia pequeña. Aunque la fuerza requerida para iniciar la deformación es grande, la fuerza para continuar la deformación no cambia significativamente. Consecuentemente, el cambio en las fuerzas al iniciar y al terminar es pequeño.

Es conocido que la ganancia de control K***e*** es la reciproca de la dependiente K***e***, así K***s*** puede ser inferido de K***e*** por las siguientes reglas:

IF K***e*** is small THEN K***s*** is large
IF K***e*** is large THEN K***s*** is small

Estas dos reglas ajustan el motor de inferencia A conectando K***e*** con K***s***.

Si se considera el motor de inferencia B, inferiendo K***f*** de ***e*** y ***é*** (Fig. B.7). Las dos entradas a la unidad B son error ***e*** y su derivada de tiempo ***é***. ***e*** es la diferencia entre una fuerza de referencia y la fuerza de salida aplicada. Las funciones de pertenencia para ***e*** y ***é*** son definidas en la Fig. B.8 y Fig. B.9, respectivamente. La Fig. B. 10 muestra las funciones de pertenencia para la variable de salida K***f***.

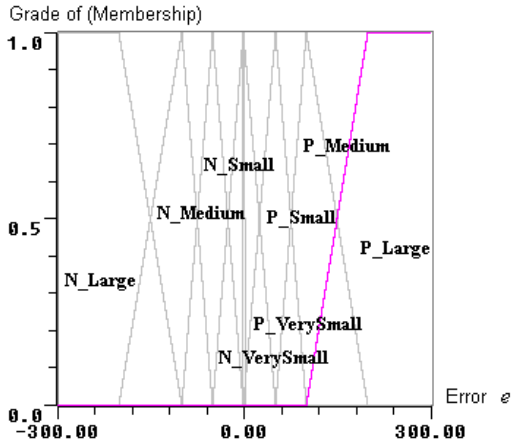


Fig. B.8 – Funciones de pertenencia de la variable ***e***

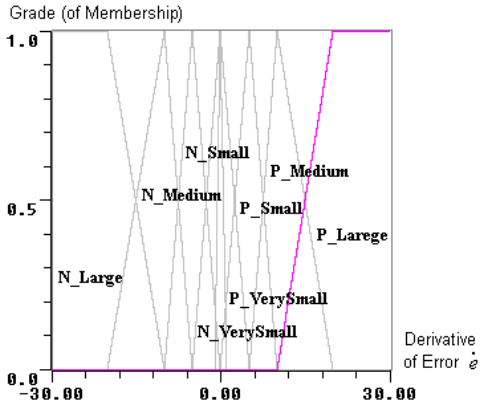


Fig. B.9 – Funciones de pertenencia de la variable ***é***

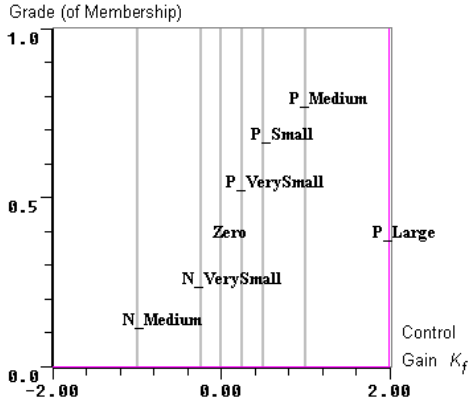


Fig. B.10 – Funciones de pertenencia de la variable k_f

Por medio de experimentación podemos obtener el conjunto de reglas para inferir K***e*** de la velocidad ***n*** y la fuerza medida ***f***.. El conjunto de reglas propuesto por [Aptro] para esta aplicación se muestra en la Tabla B.2.

		Error							
		eNLA	**eNME**	**eNSM**	**eNVS**	**ePVS**	**ePSM**	**ePME**	**ePLA**
deriv	**dNLA**	gPSM	gZER	gNVS	gNME	gPME	gPME	gPME	gPME
	dNME	gPSM	gPSM	gNVS	gNVS	gPME	gPME	gPME	gPME
	dNSM	gPME	gPME	gPSM	gPVS	gPME	gPME	gPME	gPME
	dNVS	gPME	gPME	gPME	gPLA	gPME	gPME	gPME	gPME
	dPVS	gPME	gPME	gPME	gPME	gPLA	gPME	gPME	gPME
	dPSM	gPME	gPME	gPME	gPME	gPVS	gPSM	gPME	gPME
	dPME	gPME	gPME	gPME	gPME	gNVS	gPVS	gPSM	gPVS
	dPLA	gPME	gPME	gPME	gPME	gNME	gPVS	gPSM	gPVS

Tabla B.2 – Reglas del control del servomotor

Algunos nombres son los mismos para diferentes variables, los conjuntos asociados con esos rótulos pueden ser diferentes. Por ejemplo la velocidad ***n***, el rótulo ***large*** podría ser un conjunto difuso como se muestra en la Fig. B.11a, y por la dependencia K***e***, el rótulo ***large*** puede ser otro conjunto difuso como muestra la Fig. B.11b.

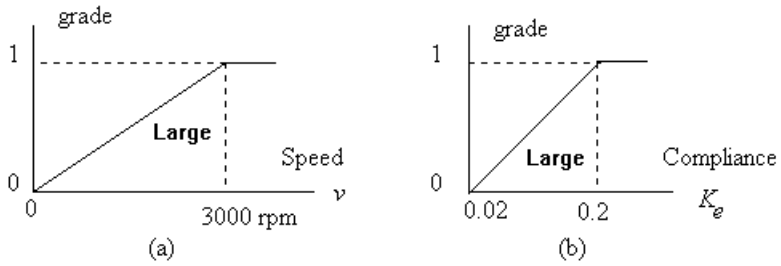


Fig. B.11 – Diferentes significados de ***large*** para diferentes variables

Los rangos de estas variables pueden ser determinados por experimentos con los dispositivos y objetos de interés. Por ejemplo, la dependencia de datos para una pelota de tenis y una dura pelota de acero puede ser usada para definir los rótulos ***large*** y ***small*** respectivamente para la variable K***e***. Si se usa el motor de inferencia para determinar K***e***, la función de ganancia del control ahora depende de 3 motores de inferencia y una operación de bloques se muestra en la Fig. B.12. Este motor implementaría K***g*** = K***s*** • K***f*** .

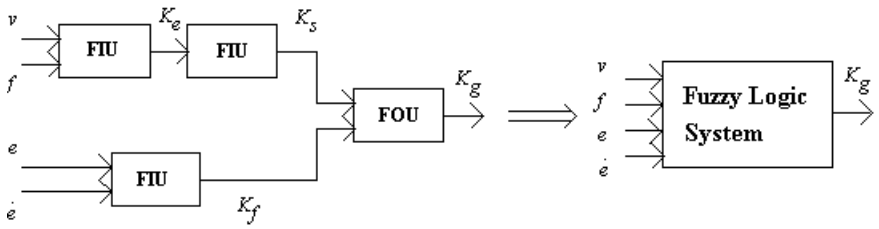


Fig. B.12 – FLS: relación entre FIU y FOU

B.3 – PÉNDULO INVERTIDO DOBLE

En esta nota de aplicación, péndulo invertido doble, se muestran detalles de todos los aspectos del diseño de sistemas de control basados en lógica difusa. Primero se analizará el sistema a diseñar; analizando el comportamiento del péndulo invertido doble. Luego se muestra como diseñar un controlador para el sistema. Se describe la curva de control y como difiere el uso de FLC del control convencional. Finalmente, se discute como el uso de tal curva determina rótulos y funciones de pertenencia para variables, así como crea las reglas para el controlador.

El sistema del péndulo invertido doble se muestra en la Fig. B.13. Un servomotor produce una fuerza *f* para mover el carro para balancear las dos etapas del péndulo invertido sobre el carro; por ejemplo, para mantener θ_1 y θ_2 en cero. θ_1 es el ángulo de la primer etapa del péndulo con la vertical, mientras que θ_2 es el ángulo de la segunda etapa del péndulo desde la vertical. Los dos ángulos se miden utilizando potenciómetros.

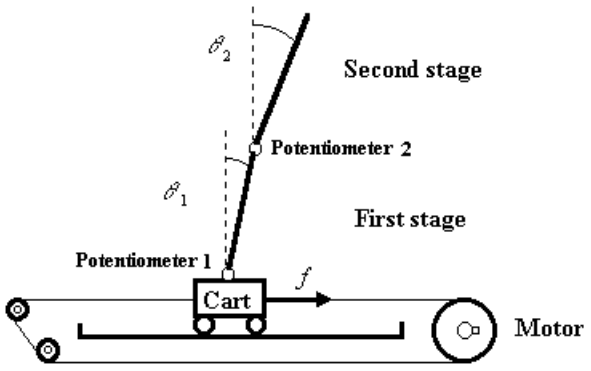


Fig. B.13 – Péndulo invertido de dos etapas

La Fig. B.14 muestra el diagrama del sistema de control. Hay dos bucles de control en este sistema. El bucle interno es para controlar la primera etapa del péndulo, mientras que el bucle externo es para controlar la segunda etapa del péndulo. Sin el bucle externo sería la solución al sistema de péndulo invertido simple. En ese caso, nuestro objetivo de control es $\theta_1 = 0$. Aunque cuando se tienen dos etapas, el objetivo de control de la primer etapa no el mismo, ahora se tiene que ajustar la primera etapa a un ángulo acorde con el ángulo de la segunda etapa (Fig. B.15). En otras palabras, en un sistema de péndulo invertido doble, el propósito del bucle externo es producir un ángulo para la primer etapa.

Esta estrategia de control puede ser resumida así:

- De acuerdo al ángulo de la segunda etapa θ_2, determinar el ángulo θ para la primer etapa. Así, se asigna $\theta = \theta_2$.
- Se ajusta la primer etapa para alcanzar $\theta_1 = \theta$.
- Como el objetivo del bucle externo es $\theta_2 = 0$, el objetivo final del sistema es $\theta_1 = \theta_2 = 0$.

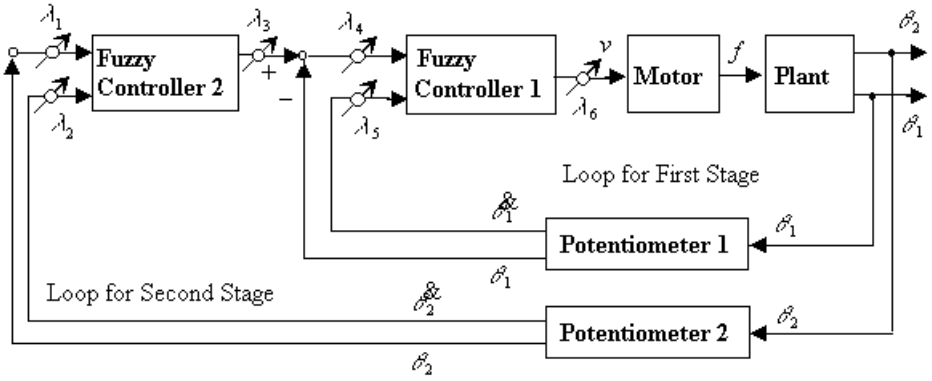


Fig. B.14 – Sistema de control del péndulo invertido de dos etapas

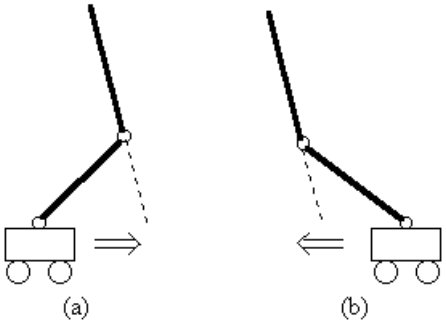


Fig. B.15 – Estrategia de control

Antes de crear el controlador difuso se analiza el comportamiento del sistema del péndulo invertido. Por simplicidad, se utiliza un péndulo invertido de una única etapa (Fig. B.16).

Básicamente, un controlador reflejará la relación entre las dos variables θ y *f*. En otras palabras, si conocemos la relación entre θ y *f* se puede escribir las reglas de control. Sabemos que si θ es grande, *f* debe ser grande; si θ es pequeña, *f* debe ser pequeña también.

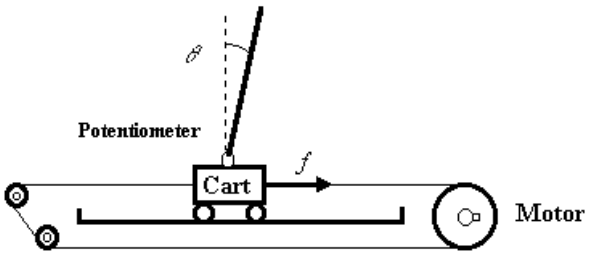


Fig. B.16 – Péndulo invertido de una etapa

La relación entre θ y *f* puede ser descrita como la curva de control (Fig. B.17). Usando un controlador convencional PID, esa curva de control es una línea recta (Fig. B.17a). La relación entre θ y *f* es no lineal, porque la fuerza podría incrementarse muy rápido cuando el ángulo θ es grande buscando obtener θ = 0. Esto mejora las respuestas porque la fuerza es más grande que aquella definida por la curva lineal cuando el error es grande. Por otro lado, cuando el ángulo es muy pequeño, la fuerza *f* debe ser muy pequeña para reducir la sobre valoración. Una típica curva de control no lineal se muestra en Fig. B.17b.

La curva de control desarrollada es un poco diferente de la mostrada en la Fig. B.17b debido a las características físicas de la aplicación (por la fricción, la fuerza debe ser un poco más grande para mover el carro). En suma, dada la limitación de salida del servomotor, la fuerza alcanza su máximo valor a cierto ángulo. La curva de control para el sistema de control del péndulo invertido es mostrada en la Fig. B.17c (se utilizará esta curva para desarrollar el FLC).

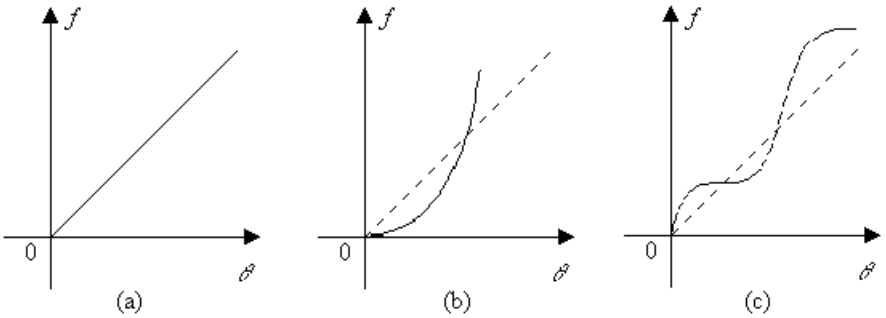


Fig. B.17 – Curva de control: Lineal and No lineal

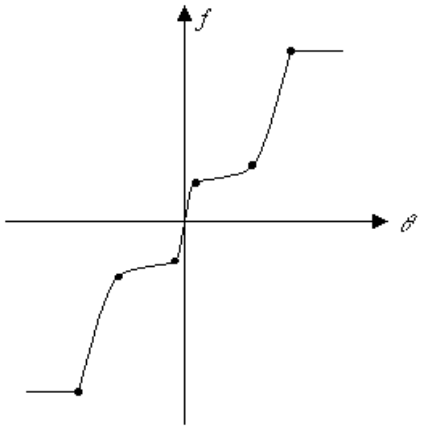


Fig. B.18 – Puntos críticos de la curva de control

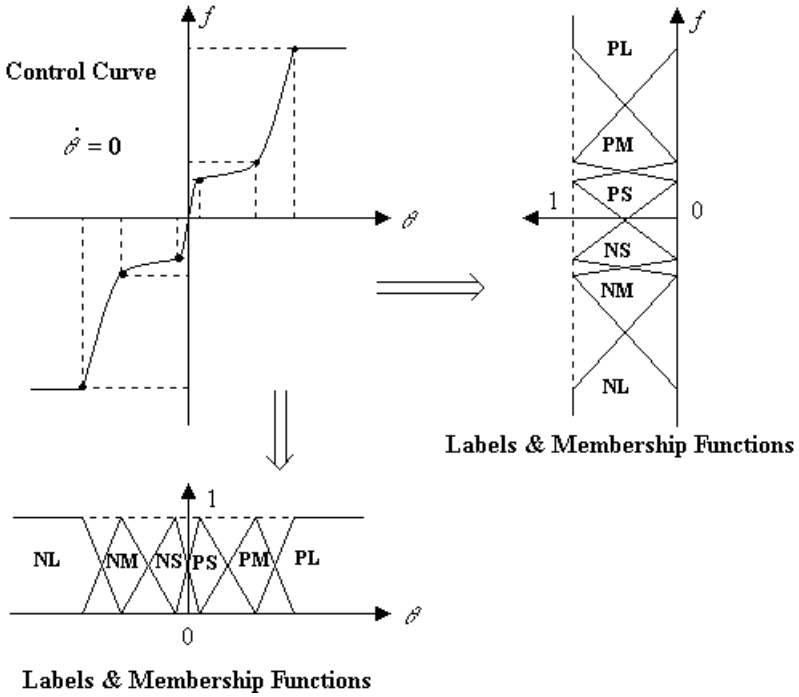


Fig. B.19 – Funciones de pertenencia de la curva de control

Sobre la curva de control, hay algunos puntos críticos que deben ser tenidos en cuenta para el diseñar el FLC. La curva de control de la Fig. B.18 muestra esos seis puntos críticos.

Las reglas pueden ser obtenidas a partir de la curva de control. Hay dos variables de entrada para el controlador. Si la velocidad de cambio de ángulo es cero, se tiene la curva de control para θ como muestra la Fig. B.19. De esta curva se obtienen las siguientes reglas:

IF θ is NL **THEN** *f* is NL
IF θ is NM **THEN** *f* is NM
IF θ is NS **THEN** *f* is NS
IF θ is PS **THEN** *f* is PS
IF θ is PM **THEN** *f* is PM
IF θ is PL **THEN** *f* is PL

Note que estas reglas son obtenidas cuando la velocidad de cambio del ángulo θ& es igual a cero. Cuando θ& no es cero, la curva de control debe ser desplazada. La velocidad es definida por 5 rótulos y se obtienen las reglas listadas en la Tabla B.3 y mostradas en la Fig. B.20. De esa figura, se puede ver que la

curva no se obtiene simplemente desplazándola. Si la curva se obtuviera mediante un desplazamiento, más rótulos para el ángulo θ serían necesarios. Se utilizarán tan pocos rótulos como sea posible para mantener la simpleza del controlador.

		VELOCIDAD				
		NL	**NM**	**ZR**	**PM**	**PL**
ANGULO	**NL**	NL	NL	NL	NM	NS
	NM	NL	NL	NM	NS	PS
	NS	NL	NM	NS	PS	PM
	PS	NM	NS	PS	PM	PL
	PM	NS	PS	PM	PL	PL
	PL	PS	PM	PL	PL	PL

Table B.3 – Reglas de Control del péndulo invertido doble

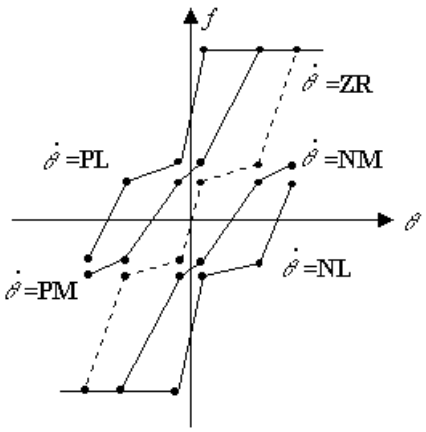


Fig. B.20 – Curvas de control cuando la velocidad no es cero

La salida de la unidad de cálculo difuso es un voltaje, que es enviado al motor para controlar la fuerza (el principio de diseño es el mismo para las otras variables a controlar).

Luego del ajuste y puesta a punto de las funciones de pertenencia se puede obtener una nueva curva de control (Fig. B.19). Los rótulos y las funciones de pertenencia para las variables del controlador se muestran en la Fig. B.21, Fig. B.22 y Fig. B.23.

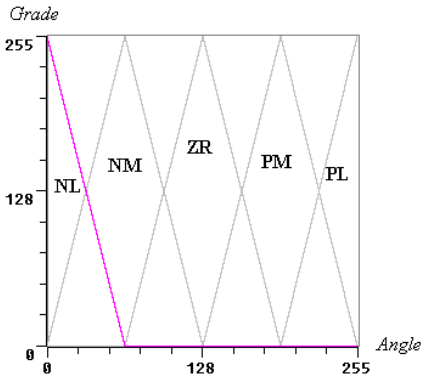


Fig. B.21 – Funciones de pertenencia de la variable ***Angle***

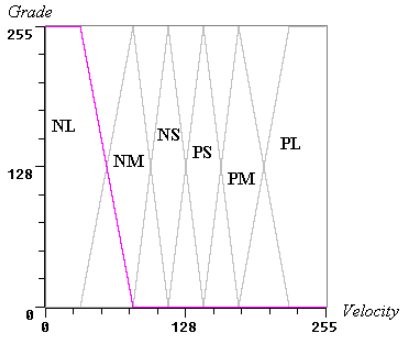


Fig. B.22 – Funciones de pertenencia de la variable ***Velocity***

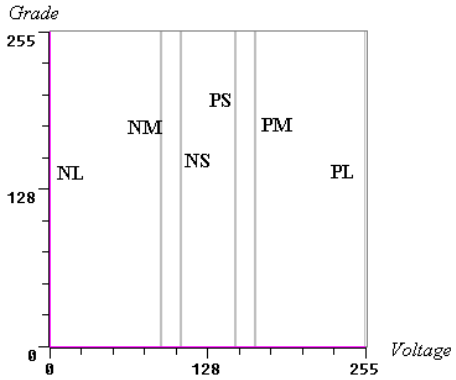


Fig. B.23 – Funciones de pertenencia de la variable ***Voltage***

B.4 – CONTROL DE TEMPERATURAS GENÉRICO

El control de temperatura es utilizado ampliamente en varios procesos. Estos procesos no difieren si es una gran planta industrial o una aplicación hogareña, comparten las mismas características desfavorables. Estas características son la no linealidad, interferencia, tiempo muerto y perturbaciones externas, entre otras. Los enfoques convencionales usualmente pueden no resultar

satisfactorios. En esta aplicación se muestra un ejemplo de un control de temperatura en varias situaciones diferentes.

Los sistemas de control de temperatura, usando controladores basados en lógica difusa, han sido puestos en operación y brindan un desempeño mejor que los sistemas de control convencional. Los controladores difusos también muestran respuesta robusta en el manejo del comportamiento en el proceso de tiempo muerto [Aptro].

Un horno de fundición de vidrio normalmente tiene dos habitaciones, el horno y la refinería. Los materiales crudos son fundidos a alta temperatura en el horno. La temperatura de fundición del vidrio es ajustada de acuerdo a su proceso de formación. Toma un largo tiempo el cambiar la temperatura en la fundición, lo que es un ejemplo de un tiempo muerto en este proceso. El material crudo durante el proceso de mezcla, el color del vidrio y la cantidad de vidrio son algunos de los factores que contribuyen a complicar dicho control. Como hay tantas variables y el proceso es complejo, es muy difícil el diseño de un controlador efectivo para esta aplicación.

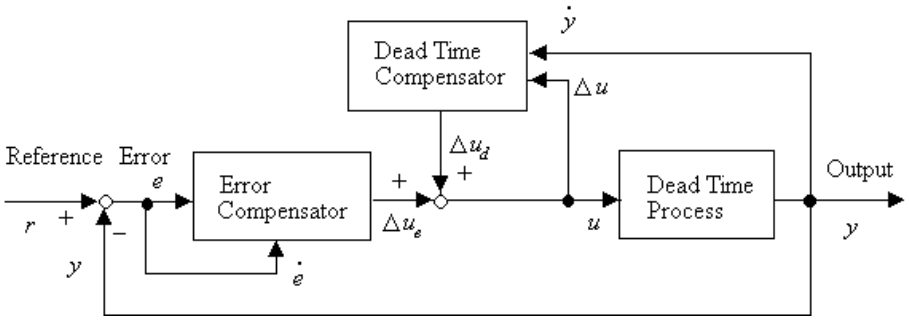


Fig. B.24 – Control difuso para un horno de fundición de vidrio

El diagrama del bloque de control para un horno de fundición de vidrio es mostrado en la Fig. B.24. El valor de control ***U*** es aplicado al proceso para ajustar la temperatura. Este valor es cambiado por dos compensadores. La variación de ***U*** puede ser descrita como: $\Delta \boldsymbol{U} = \Delta \boldsymbol{U}\text{d} + \Delta \boldsymbol{U}\text{e}$; donde $\Delta \boldsymbol{U}\text{d}$ es la salida del compensador de *tiempo muerto*, y $\Delta \boldsymbol{U}\text{e}$ es la salida del compensador de *error*. El compensador de *tiempo muerto* es usado para reducir el efecto que el tiempo

muerto tiene sobre el proceso. La salida (Δ***U***d) representa el cambio incremental en el valor de control; es obtenida por el cambio producido en el valor de control (actual y anterior denominado Δ***U***) y el tiempo diferencial de temperatura de salida (***é***). El compensador de error es usado para reducir la diferencia entre la temperatura deseada y la actual temperatura del horno. Su salida (Δ***U***e), un cambio incremental en el valor de control, es determinada de la diferencia (error) ***e*** y su diferencial de tiempo ***é*** (actualmente se usa la variación Δ***e***). Así, Δ***U*** d y Δ***U e*** son combinados para cambiar la variable de control ***U***.

Estos dos bloques se presentan como un controlador separado, debido a que el objetivo de este documento no es el análisis de controladores de temperatura, es presentar un ejemplo; por lo cual se presentan dos FLC: compensador de tiempo muerto (*aplicación denominada TGlass*) y compensador de error (*aplicación denominada TError*).

B.5 – COMPENSADOR DE TIEMPO MUERTO

Las funciones de pertenencia de la variable de entrada "***Pre_var_ctrl***" (varC) del compensador de tiempo muerto son presentadas por la Fig. B.25. Se utilizan seis funciones de pertenencia: ***N_Small*** (vNS), ***N_Medium*** (vNM), ***N_Large*** (vNL), ***P_Small*** (vPS), ***P_Medium*** (vPM), y ***P_Large*** (vPL).

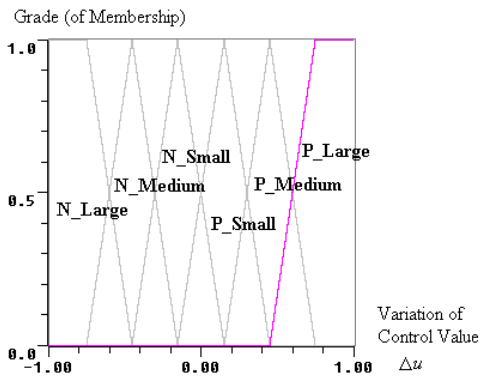


Fig. B.25 – Funciones de pertenencia de ***Prev_Var_Ctrl***

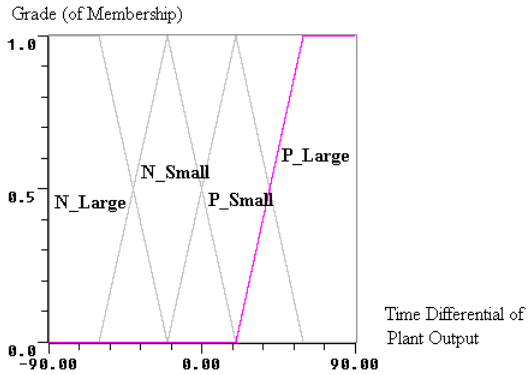


Fig. B.26 – Funciones de pertenencia de ***TimeDiff_Output***

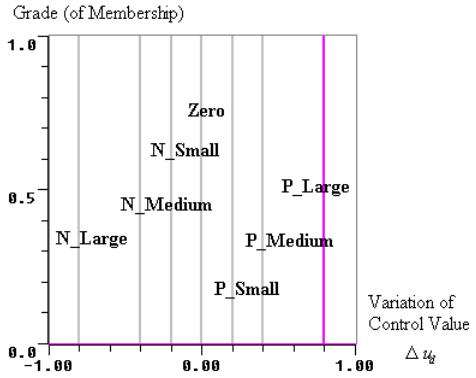


Fig. B.27 – Funciones de pertenencia de ***Var_Ctrl***

Las funciones de pertenencia de la variable de entrada "***TimeDiff_Output***" (***Time***) son presentadas por la Fig. B.26. Se utilizan cuatro funciones de pertenencia: ***N_Small*** (tNS), ***N_Large*** (tNL), ***P_Small*** (tPS), y ***P_Large*** (tPL).

Las funciones de pertenencia de la variable de salida "***Var_Ctrl***" son presentadas por la Fig. B.27. Se utilizan siete funciones de pertenencia: ***N_Small*** (cNS), ***N_Medium*** (cNM), ***N_Large*** (cNL), ***Zero*** (cZE), ***P_Small*** (cPS), ***P_Medium*** (cPM), y ***P_Large*** (cPL). La Tabla B.4 presenta el conjunto de reglas utilizadas para definir el comportamiento del controlador.

		varC					
		vNL	vNM	vNS	vPS	vPM	vPL
Time	tNL	cPL	cPM	cPS	cZE	cZE	cZE
	tNS	cPM	cPS	cZE	cZE	cZE	cNS
	tPS	cPS	cZE	cZE	cZE	cNS	cNM
	tPL	cZE	cZE	cZE	cNS	cNM	cNL

Tabla B.4 – Conjunto de reglas del motor de inferencias del compensador de tiempo muerto (***TGlass***)

B.6 – COMPENSADOR DE ERROR

Las funciones de pertenencia de la variable de entrada "***Error***" del compensador de error son presentadas por la Fig. B.28. Se utilizan siete funciones de pertenencia: ***N_Small*** (vNS), ***N_Medium*** (vNM), ***N_Large*** (vNL), ***Zero*** (vZE), ***P_Small*** (vPS), ***P_Medium*** (vPM), y ***P_Large*** (vPL).

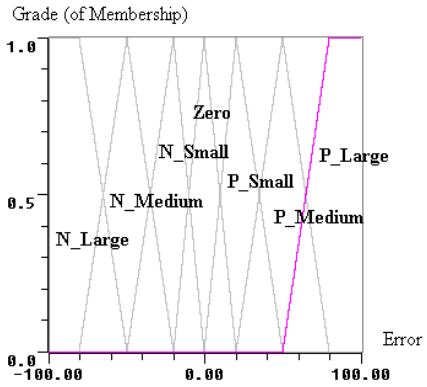


Fig. B.28 – Funciones de pertenencia de ***Error***

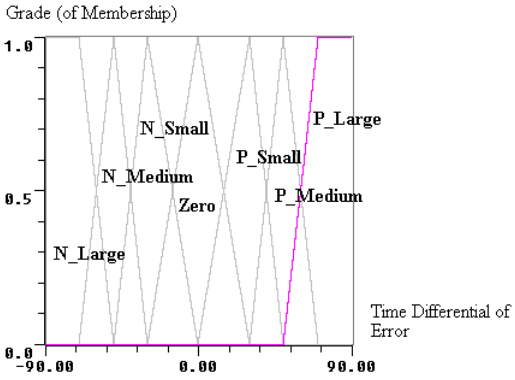


Fig. B.29 – Funciones de pertenencia de ***TimeDiff_Error***

Las funciones de pertenencia de la variable de entrada "***TimeDiff_Error***" (***timed***) del compensador de error son presentadas por la Fig. B.29. Se utilizan siete funciones de pertenencia: ***N_Small*** (tNS), ***N_Medium*** (tNM), ***N_Large*** (tNL), ***Zero*** (tZE), ***P_Small*** (tPS), ***P_Medium*** (tPM), y ***P_Large*** (tPL).

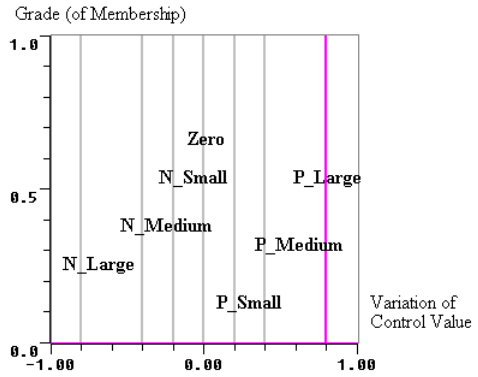


Fig. B.30 – Funciones de pertenencia de ***Var_Ctrl***

Las funciones de pertenencia de la variable de salida "***Var_Ctrl***" son presentadas por la Fig. B.30. Se utilizan siete funciones de pertenencia: ***N_Small*** (cNS), ***N_Medium*** (cNM), ***N_Large*** (cNL), ***Zero*** (cZE), ***P_Small*** (cPS), ***P_Medium*** (cPM), y ***P_Large*** (cPL). La Tabla B.5 presenta el conjunto de reglas utilizadas para definir el comportamiento del controlador.

		Error						
		vNL	vNM	vNS	vZE	vPS	vPM	vPL
Timed	tNL				cNL	cNM		
	tNM		cNL		cNM		cZE	
	tNS			cZE	cZE			
	tZE	cNL	cNM	cZE	cZE	cZE	cPM	cPL
	tPS				cZE	cZE		
	tPM		cZE		cPM		cPL	
	tPL			cPM	cPL			

Tabla B.5 – Conjunto de reglas del motor de inferencias del compensador de error (***TError***)

B.7 – CONTROL DE TEMPERATURAS DE UN REACTOR

El control de temperatura se usa ampliamente en los varios procesos. Estos procesos, tanto si es el proceso de una gran planta industrial como un proceso empotrado en un artefacto del hogar, comparten varios rasgos desfavorables como la no-linealidad, interferencia, tiempo muerto y perturbación externa. Para este tipo de procesos las soluciones basadas en el control convencional normalmente no pueden lograr resultados satisfactorios a bajo coste.

Este ejemplo se propone usando la presión como variable de entrada, lográndose un control de temperatura más sofisticado. La presión es usada porque tiene una gran influencia en la temperatura a ser controlada; alta presión produce alta temperatura. Un diagrama del equipo se muestra en Fig. B.31. El gas del reactor es drenado hacia afuera, mientras se inyecta un poco de gas argón. Se genera un voltaje de alta frecuencia que se descarga entre el electrodo y la base. Los iones de argón disociados chocan hacia el blanco (*target*) y extienden los átomos divididos formando una película sobre la base. La temperatura de la base es crítica en el proceso de generación de la película, por tal razón es tan importante el control de temperatura.

La Fig. B.31 muestra el diagrama del sistema de control de temperatura difuso que se usa en este tipo de reactor. Las entradas al controlador son la temperatura deseada de la base, la temperatura medida de la base, y la presión medida en el reactor. Las señales de salida del controlador ajustan el calentador y la válvula electromagnética para el agua refrigerante del reactor.

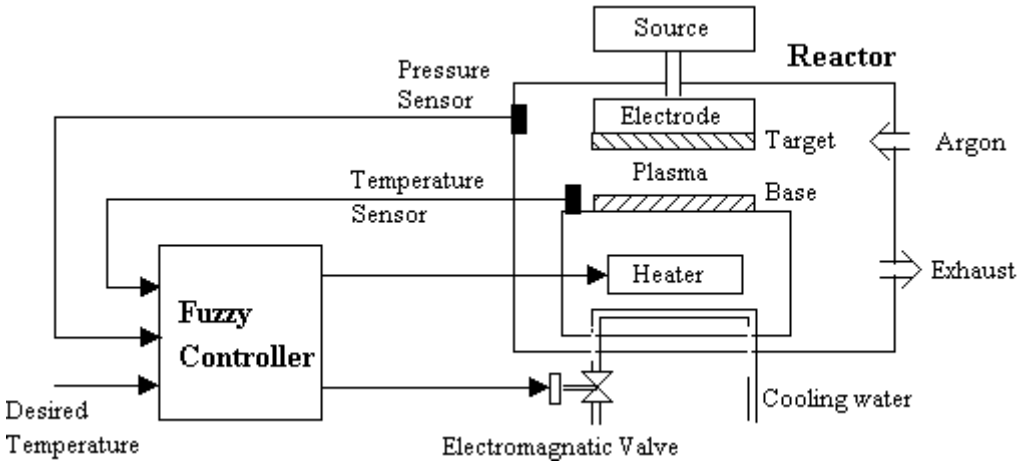


Fig. B.31 – Diagrama del sistema del control de temperatura del reactor

La Fig. B.32 muestra el FLC con tres variables de entrada y dos variables de salida. Las entradas pueden ser preparadas de las señales de retroalimentación de los sensores. La variable de entrada ***error*** es la diferencia entre la temperatura deseada ***td*** y la temperatura medida ***t***, es decir $\boldsymbol{e} = \boldsymbol{td} - \boldsymbol{t}$; la variable de entrada ***variación de error*** es la variación entre el $\boldsymbol{e}_i$ (actual) y el $\boldsymbol{e}_{i-1}$ (anterior), es decir $\boldsymbol{\Delta e} = \boldsymbol{e}_i - \boldsymbol{e}_{i-1}$; la variable de entrada ***presión*** representa la medición actual ***p***.

Dos variables de salida son los datos de control para el calentador ***heater*** y la válvula de enfriado ***valve***.

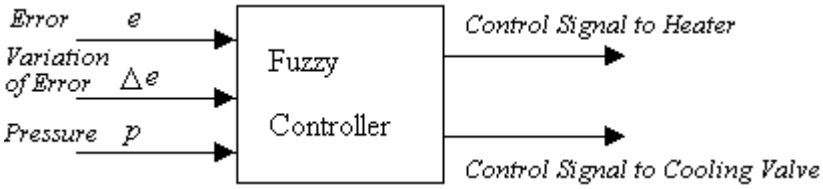


Fig. B.32 – FLC destinado al control de temperatura del reactor

Un sistema de control convencional de simple retroalimentación ajusta la temperatura sólo dependiendo del valor de la temperatura registrada por el sensor; mientras que la influencia de los cambios en la presión es tratada como ruidos del sistema.

Los rótulos y las funciones de pertenencia de las variables de entrada son definidas en la Fig. B.33 y Fig. B.34, mientras que las variables de salida se muestran en la Fig. B.35. Note que todas las variables están normalizadas en el rango de [-1, 1] o [0, 1].

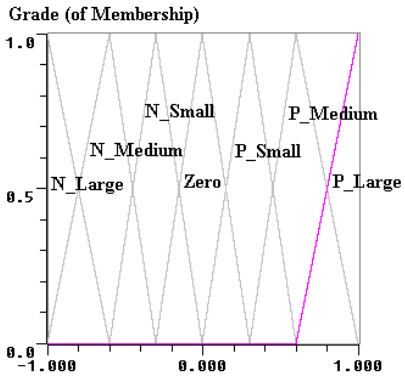


Fig. B.33 – Funciones de pertenencia de las variables ***Error*** y ***Var_Error***

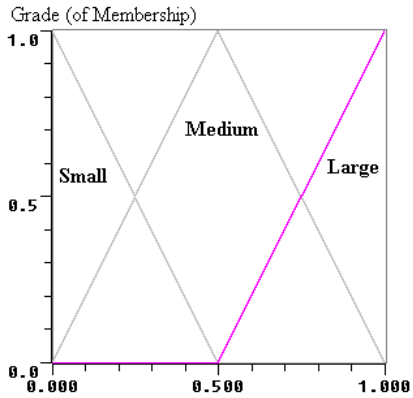


Fig. B.34 – Funciones de pertenencia de las variables de entrada ***presión***

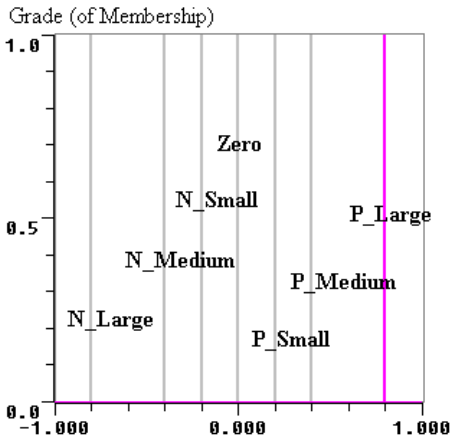


Fig. B.35 - Funciones de pertenencia de las variables de salida ***Var_Heater*** y ***Var_Cooling***

Debido a la cantidad de reglas que tiene esta aplicación, se divide el conjunto de reglas en tres tablas para cada variable de salida (***var_Cooler*** y ***var_Heater***); considerando por separado cada una de las funciones de pertenencia de la variable de entrada presión (***Large***, ***Medium***, ***Small***). La Tabla B.6, la Tabla B.7 y la Tabla B.8 presentan las reglas que consideran la presión como ***Large***, ***Medium***, y ***Small***, para el ***Cooler***. La Tabla B.9, la Tabla B.10 y la Tabla B.11 presentan las reglas que consideran la presión como ***Large***, ***Medium***, y ***Small***, para el ***Heater***.

		Error						
		eNL	**eNM**	**eNS**	**eZE**	**ePS**	**ePM**	**ePL**
var error	**vNL**	cPL	cPL	cPL	cPM	cZE	cZE	cNS
	vNM	cPL	cPL	cPM	cPS	cZE	cZE	cNM
	vNS	cPL	cPM	cPM	cZE	cZE	cNM	cNL
	vPS	cPS	cPS	cZE	cNS	cNS	cNM	cNL
	vPM	cPM	cZE	cZE	cNS	cNM	cNM	cNL
	vPL	cPL	cZE	cZE	cNS	cNM	cNL	cNL

Tabla B.6 – Reglas para el control de temperatura de un reactor, variable COOLER con Presión = LARGE

.

		Error						
		eNL	**eNM**	**eNS**	**eZE**	**ePS**	**ePM**	**ePL**
var error	**vNL**	cPL	cPL	cPM	cPM	cZE	cZE	cNS
	vNM	cPL	cPM	cPS	cPS	cZE	cZE	cNM
	vNS	cPL	cPM	cPS	cZE	cZE	cNM	cNL
	vPS	cPL	cPS	cZE	cNS	cNS	cNM	cNL
	vPM	cPM	cZE	cZE	cNS	cNM	cNM	cNL
	vPL	cPS	cZE	cZE	cNS	cNM	cNL	cNL

Tabla B.7 – Reglas para el control de temperatura de un reactor, variable COOLER con Presión = MEDIUM

.

		Error						
		eNL	**eNM**	**eNS**	**eZE**	**ePS**	**ePM**	**ePL**
var error	**vNL**	cPL	cPL	cPM	cPM	cZE	cZE	cNS
	vNM	cPL	cPM	cPS	cPS	cZE	cZE	cNM
	vNS	cPL	cPM	cPS	cZE	cZE	cNM	cNL
	vPS	cPL	cPS	cZE	cNS	cNS	cNM	cNL
	vPM	cPM	cZE	cZE	cNS	cNM	cNL	cNL
	vPL	cPS	cZE	cZE	cNM	cNL	cNL	cNL

Tabla B.8 – Reglas para el control de temperatura de un reactor, variable COOLER con Presión = SMALL

.

		Error						
		eNL	**eNM**	**eNS**	**eZE**	**ePS**	**ePM**	**ePL**
var error	**vNL**	hNL	hNL	hNL	hNL	hNM	hNS	hZE
	vNM	hNL	hNL	hNM	hNM	hNM	hZE	hPS
	vNS	hNL	hNL	hNS	hNM	hNS	hPS	hPM
	vPS	hNL	hNL	hNM	hZE	hPS	hPS	hPM
	vPM	hNL	hNL	hNS	hPS	hPS	hPM	hPM
	vPL	hNM	hNM	hZE	hPM	hPM	hPM	hPM

Tabla B.9 – Reglas para el control de temperatura de un reactor, variable HEATER con Presión = LARGE

.

		Error						
		eNL	**eNM**	**eNS**	**eZE**	**ePS**	**ePM**	**ePL**
var error	**vNL**	hNL	hNL	hNL	hNM	hNS	hZE	hPS
	vNM	hNL	hNL	hNM	hNS	hNS	hPS	hPM
	vNS	hNL	hNM	hNS	hNS	hZE	hPM	hPL
	vPS	hNL	hNM	hNS	hNS	hPM	hPM	hPL
	vPM	hNM	hNM	hZE	hPM	hPM	hPL	hPL
	vPL	hNS	hNS	hPS	hPL	hPL	hPL	hPL

Tabla B.10 – Reglas para el control de temperatura de un reactor, variable HEATER con Presión = MEDIUM

.

		Error						
		eNL	**eNM**	**eNS**	**eZE**	**ePS**	**ePM**	**ePL**
var error	**vNL**	hNM	hNM	hNM	hNS	hZE	hPS	hPM
	vNM	hNM	hNM	hNS	hZE	hZE	hPM	hPL
	vNS	hNM	hNS	hZE	hZE	hPS	hPL	hPL
	vPS	hNM	hNS	hZE	hPM	hPL	hPL	hPL
	vPM	hNS	hNS	hPS	hPL	hPL	hPL	hPL
	vPL	hZE	hZE	hPM	hPL	hPL	hPL	hPL

Tabla B.11 – Reglas para el control de temperatura de un reactor, variable HEATER con Presión = SMALL

B.8 – ENSAMBLADOR (ASSEMBLY)

Esta aplicación es presentada por Aptronix con el comentario: "*diseño de una máquina de estados difusa para el control de fabricación automática*", y su autor ha sido Ted Heske de AT&T, división NCR. NCR fabrica en su planta de Atlanta diodos láser que trabajan en el espectro visible. La salida del diodo laser diverge a través de un gran sólido ángulo. Para utilizar estos diodos se debe ensamblar un dispositivo óptico (Fig. 36), que incluye lentes. Estos lentes enfocan el laser de tal forma que su salida sea útil. El equipo automático construye el dispositivo que contiene el laser y los lentes, enfocando el rayo a una específica distancia para cada diferente producto fabricado. Debido a variaciones en el ajuste mecánico, la longitud focal y las dimensiones del diodo laser; cada dispositivo requiere un ajuste para alcanzar enfoque a la distancia correcta.

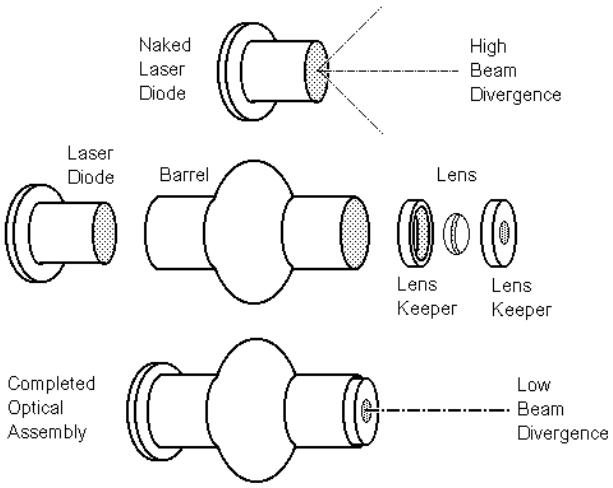


Fig. B.36 – Componentes de un dispositivo de un diodo láser

La Fig. B.37 muestra las piezas del equipo de enfoque automático de láser. El lente es sujeto seguramente en el lugar, mientras un motor paso a paso desplaza el diodo láser (acerca o aleja). Una varilla roscada convierte cada paso del motor en un movimiento linear de aproximadamente 0.001" (pulgada). El blanco a que apunta el dispositivo es un instrumento de medición de ancho del rayo a una distancia ***dT*** de los lentes. Las leyes de la óptica describen el problema como: 1 / ***o*** + 1 / ***i*** = 1 / ***f***, donde ***o*** es la distancia objeto, i es la distancia de la imágen, y ***f*** es la distancia focal. Como el motor permite reducir ***o***, el punto focal dado por la distancia de la imágen varía. El blanco será el mínimo rayo cuando ***i*** = ***dT***. Como ***o*** es decrementado por el motor, el rayo se estrechará a su mínimo. Si el motor continúa funcionando el rayo crecerá nuevamente. El método propuesto detiene el motor cuando el diametro alcanza un ancho óptimo predeterminado. El método requiere que ***dT*** sea menor que la distancia focal deseada.

Como el motor paso a paso empuja el barril sobre el lente, el tamaño del rayo en el destino sigue la función mostrada en la Fig. B.38. La Fig. B.38 mouestra como se divide el espacio en tres diferentes estados: ***s0***, ***s1*** y ***s2***. Todos los ensamblajes comienzan en ***s0***, mientras la pendiente es negativa, y donde el ancho del rayo está sobre el mínimo. El estado ***s1*** define la región del piso del gráfico, donde el ancho del rayo es pequeño; mientras que ***s2*** representa la región donde la pendiente es positiva y el ancho del rayo se encuentra entre el mínimo y el ideal.

Finalmente, cuando el rayo alcanza el ideal, el ajuste del enfoque está completo y el motor es detenido.

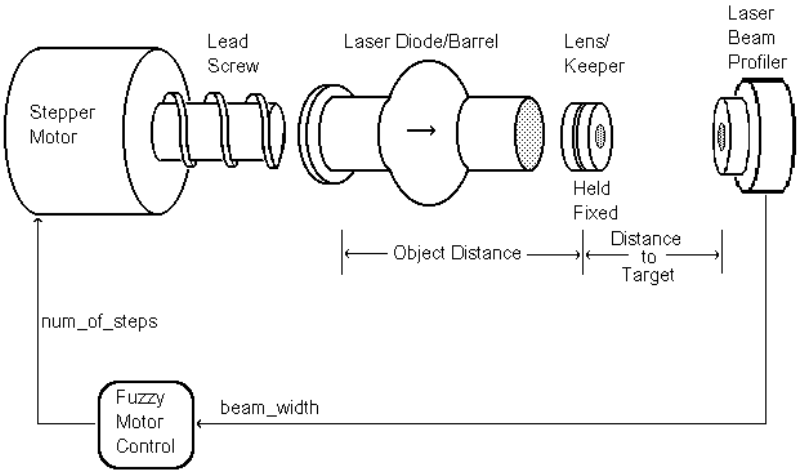


Fig. B.37 – Componentes del sistema de enfoque automático.

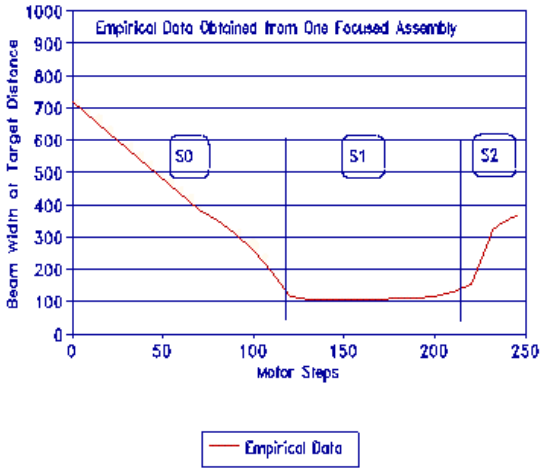


Fig. B.38 – División de estados del ancho del rayo VS posición del motor.

El principal obstáculo para alcanzar el objetivo son las condiciones de ruido y la naturaleza intermitente de la información del ancho del rayo. El ambiente de producción es ruidoso, y desafortunadamente el ancho del rayo es sensible tanto a la vibración mecánica como la interferencia electromagnética. La solución previa promediaba varias lecturas para obtener el ancho del rayo. Este enfoque, aunque más preciso asignaba iguales pesos a mediciones de anchos reales tanto como los erroneos. Para mejorar la precisión, un operador debia filtrar las mediciones erroneas de acuerdo a su experticia. Así, se condicionan las mediciones del ancho

del rayo al procesamiento humano, la Fig. B.39 muestra el paradigma difuso propuesto que resuelve el problema.

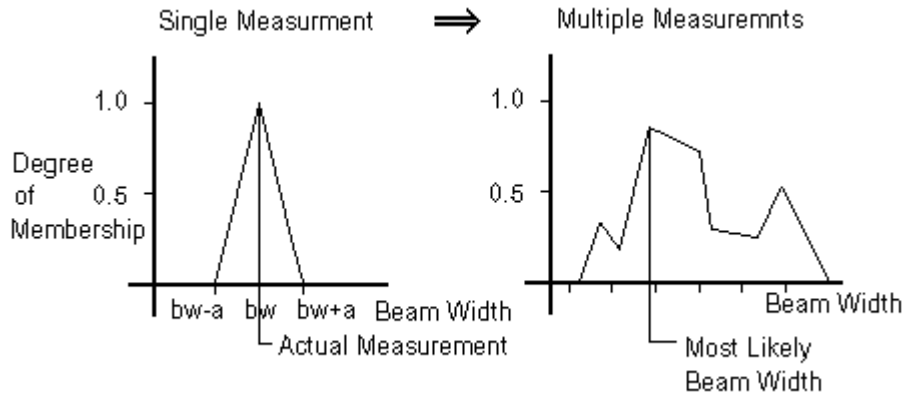


Fig. B.39 – Paradigma difuso aplicado a la señal de entrada.

Cada medida del ancho del rayo es tratada como un conjunto difuso centrado sobre la lectura actual del instrumento. Las regiones difusas generadas por las múltiples lecturas son sumadas y normalizadas. El ancho real del rayo es seleccionado como el ancho de rayo con el máximo grado de pertenencia. La pendiente en la gráfica del ancho del rayo es también usada como entrada al FLC. Las funciones de pertenencia se muestran en la Fig. B.40 y la Fig. B.41.

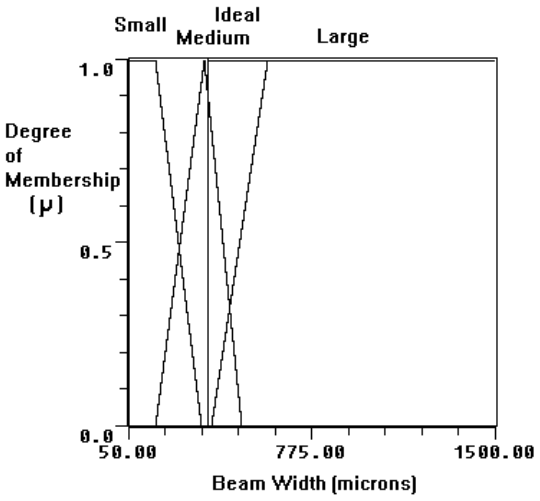


Fig. B.40 – Funciones de pertenencia de la entrada (ancho del rayo) ***bw***.

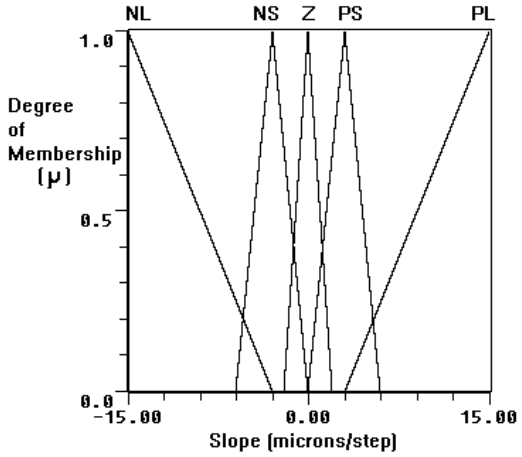


Fig. B.41 – Funciones de pertenencia de la variable de entrada (pendiente) ***slope***.

Las reglas difusas son generadas por la experiencia con el sistema y refinadas mediante pruebas. Como la lógica booleana es un subconjunto de la lógica difusa, es simple incluir condiciones para el manejo de excepciones en el FLC. Una excepción es generada cuando el ancho del rayo alcanza su valor ideal. Como el módulo de control difuso es una parte de una aplicación más grande, éste notifica al resto del sistema generando una señal (***done***) cuando se alcanza el ancho ideal del rayo. La Fig. B.42 muestra las funciones de pertenencia de salida que generan la señal ***done*** mientras que las reglas se muestran en la Tabla B.12.

		SLOPE				
		NL	NS	Z	PS	PL
Ancho del rayo	Small	L	L	M	M	M
	Medium	L	M	S	S	S
	Large	L	L	L	L	L

a) Reglas para la salida: STEPS

		SLOPE				
		NL	NS	Z	PS	PL
Ancho del rayo	Ideal	NO	NO	NO	YES	YES

b) Reglas para la salida: DONE

Tabla B.12 – Reglas para ASSEMBLY

Una de las salidas del FLC controla el motor paso a paso indicando el número de pasos a dar. La medida del ancho del rayo y el bucle del motor continúan

hasta que el ancho ideal del rayo es alcnazado. Como las funciones de pertenencia de salida (Fig. B.43) son singletons, la salida es calculada como el promedio ponderado de las reglas que contribuyen en la inferencia.

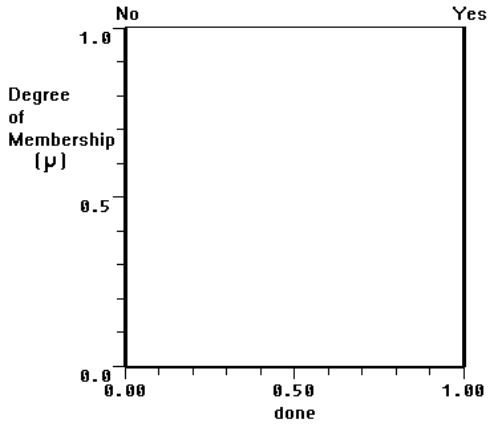


Fig. B.42 – Funciones de pertenencia de la variable de salida (Hecho) ***done***.

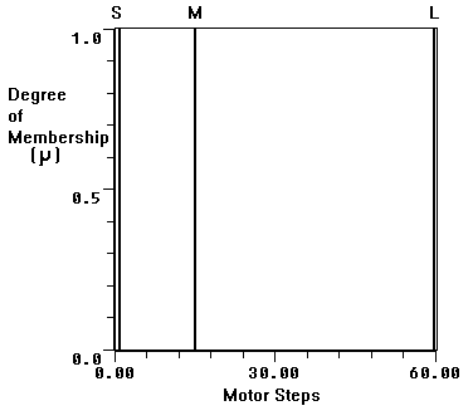


Fig. B.43 – Funciones de pertenencia de la variable de salida (pasos) ***steps***.

B.9 – CONTROL DE UNA LAVADORA AUTOMÁTICA

Cuando se utiliza una lavadora de ropa, generalmente se debe seleccionar el tiempo de labado, basado en la cantidad de ropa y el tipo y grado de suciedad que esta tenga. Para automatizar este proceso, se usan sensores para detectar estos parámetros (volúmen de ropa, grado de suciedad y tipo de suciedad). Así, el tiempo de labado es determinado a partir de estos datos. Desafortunadamente, no hay una relación matemática entre el volúmen de ropa y suciedad que

determine el tiempo de lavado necesario. Por esta causa, normalmente los usuarios seleccionan manualmente los parámetros de acuerdo a su experiencia.

Para construir una lavadora totalmente automática (que incluya la selección automática del tiempo de lavado) hay dos subsistemas de la máquina: el mecanimo sensor y la unidad de control. El sistema de sensores provee las señales de entrada a la máquina, quien será la encargada de tomar las decisiones. Es la responsabilidad del controlador tomar las decisiones y generar la señal de salida. Como la relación de entrada/salida no es perfectamente clara se recomienda el uso de FL para su control.

Así, el objetivo es el diseño de un FLC para una lavadora que determine el correcto tiempo de lavado, aún cuando no se cuente con un preciso modelo que relacione las entradas con las salidas.

La Fig. B.44 muestra un diagrama general del FLC. El FLC tiene dos entradas, el grado de suciedad de la ropa (***mugre***) y el tipo de suciedad que tienen las ropas (***tipoM***). Estas dos entradas pueden ser obtenidas de un simple sensor óptico. El grado de suciedad es determinado por la transparencia del agua del lavado. A ropas más sucias corresponderá una menor transparencia sobre una cantidad fija de agua. Por el otro lado, el tipo de suciedad es determinado por el tiempo de saturación, el tiempo que toma alcanzar la saturación. Saturación es el punto en el cual el cambio en la transparencia del agua es cercano a cero (o menor que un número determinado). Las ropas grasosas, por ejemplo, toman un tiempo más grande para saturar la transparencia del agua porque la grasa es menos soluble en agua que otras formas de suciedad. De esta forma, un sistema de sensores brindan las entradas necesarias para el FLC.

Fig. B.44 – Diagrama global del controlador.

Antes de diseñar el FLC se debe determinar el rango de posibles valores para la entrada y salida de variables, definiendo las funciones de pertenencia encargadas de la fusificación y defusificación. La Fig. B.45 muestra los rótulos y funciones de pertenencia de la variable de entrada suciedad (***mugre***). La Fig. B.46 muestra los rótulos y funciones de pertenencia de la variable de entrada tipo de suciedad (***tipoM***). Los valores de ambas variables son normalizadas en el rango de 0 a 100, sobre el dominio de los valores de los sensores ópticos.

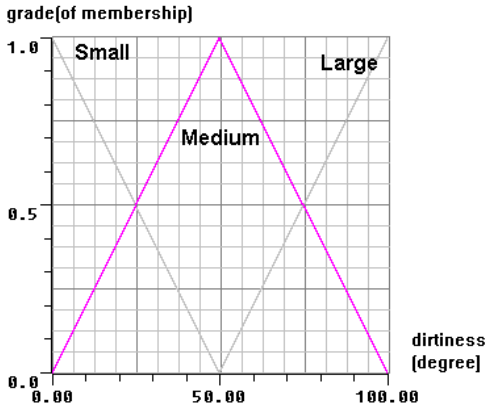


Fig. B.45 - Funciones de pertenencia de la variable de entrada ***mugre***.

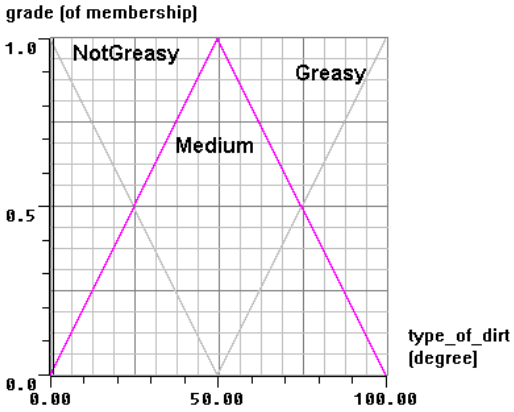


Fig. B.46 - Funciones de pertenencia de la variable de entrada ***tipoM***.

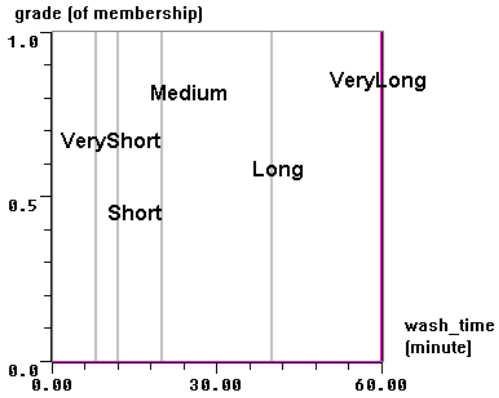


Fig. B.47 - Funciones de pertenencia de la variable de salida ***tiempo***.

La Fig. B.47 muestra las funciones de pertenencia de la variable de salida que determina el tiempo de lavado (***tiempo***) usando singletons. Se utilizan cinco funciones de pertenencia: ***Very_Short*** (VS), ***Short*** (SH), ***Medium*** (ME), ***Long*** (LO) y ***Very_Long*** (VL). La Tabla B.13 presenta el conjunto de reglas utilizadas para definir el comportamiento del controlador. La capacidad de tomar decisiones de un FLC está codificada en el conjunto de reglas que utiliza su motor de inferencias. Las reglas para la lavadora automática son derivadas del sentido común, de datos tomados del típico uso hogareño y experimentación en ambientes controlados.

		tipoM		
		NGR	MDM	GRS
Mugre	LAR	ME	LO	VL
	MED	SH	ME	LO
	SMA	VS	ME	LO

Tabla B.13 – Conjunto de reglas de la lavadora

Una lavadora totalmente automática puede ser diseñada utilizando esta tecnología. El proceso de diseño con FL imita la intuición humana, lo que facilita el desarrollo y posterior mantenimiento del sistema. En este ejemplo sólo se controla el tiempo de lavado de la máquina, el diseño puede extenderse sin demasiadas complicaciones a controlar otras variables, tales como el nivel de agua o la velocidad de rotación en el secado.

B.10 – NAVEGADOR PARA UN AUTOMÓVIL

Se presenta un sistema de control de un vehículo orientado a evitar obstáculos en un entorno no conocido, trabajando por conocimiento local del entorno (sin tener información de todo el escenario donde debe interactuar). El sistema de control se basa en lógica difusa, implementándose sólo el nivel donde se evitan los obstáculos y suponiendo que hay un nivel superior que determina el camino único en que el controlador debe guiar al vehículo. El vehículo tiene las restricciones ergonométricas de movimiento de un automóvil.

La información de entrada es el valor medido por 3 detectores de proximidad de obstáculos (Fig. B.48); ***LAbs*** en la izquierda, ***VAbs*** al centro y ***RAbs*** en la derecha. El sistema determina el valor de la velocidad (***motor***) y el del ángulo de las ruedas del vehículo (***lenkung***).

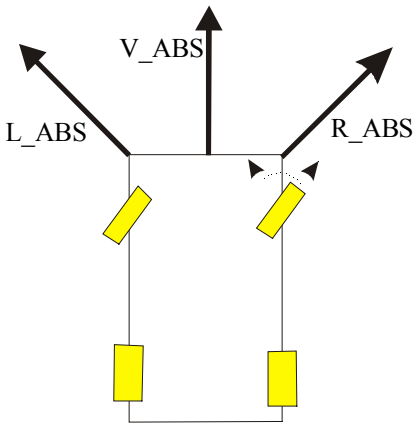


Fig. B.48 – Diagrama de ubicación de los sensores de distancia del vehículo en la aplicación AUTO

El dominio de las variables de entrada y el grado de pertenencia es representado con 8 bits. El dominio de la variable de entrada ***gesch*** es representado por 3 funciones de pertenencia (Fig. B.49): ***langsam***, ***mittel*** y ***schnell***.

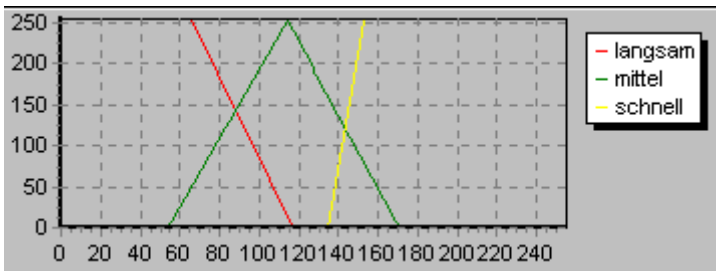


Fig. B.49 – Funciones de pertenencia de la variable de entrada ***gesch***.

El dominio de cada detector es representado por 3 funciones de pertenencia: ***nah***, ***mittel*** y ***weit***. La Fig. B.50 muestra las funciones de pertenencia de los sensores ubicados a la izquierda (***LAbs***) y a la derecha del vehículo (***RAbs***). La Fig. B.51 muestra las funciones de pertenencia del sensor central (***VAbs***).

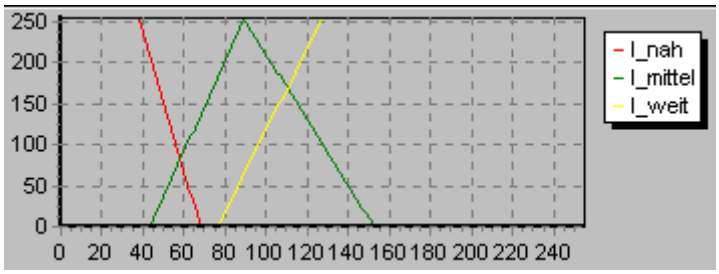


Fig. B.50 – Funciones de pertenencia de la variable de entrada ***LAbs*** y ***RAbs***.

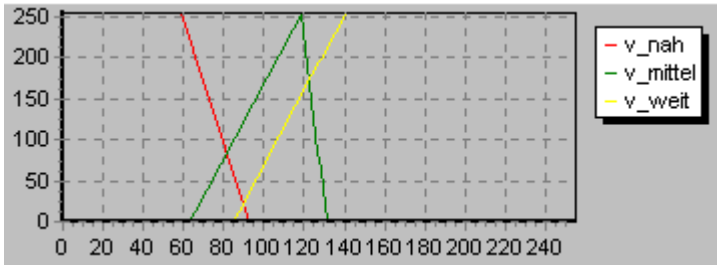


Fig. B.51 – Funciones de pertenencia de la variable de entrada ***VAbs***.

El dominio de las variables de salida es [0..30] para la ***motor*** y [0..40] para el ángulo de las ruedas de dirección (***lenkung***). Para ***motor*** se utilizan 3 funciones de pertenencia: ***drosseln***, ***ok*** y ***erhohen***; mientras que para ***lenkung*** se utilizan cinco funciones de pertenencia: ***stark_r***, ***rechts***, ***gerade***, ***links*** y ***stark_l***. Las funciones de pertenencia de las variables de salida se muestran en la Fig. B.52.

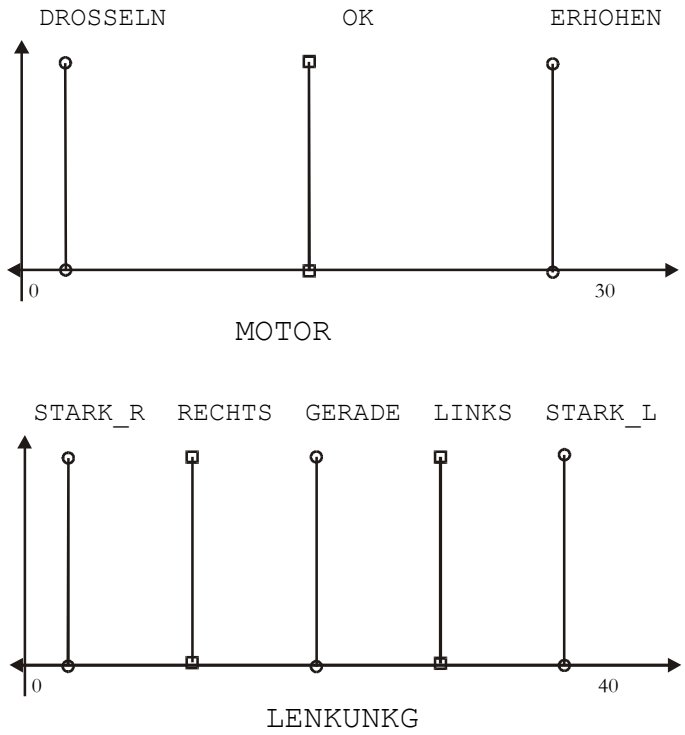


Fig. B.52 – Funciones de pertenencia de salida de las variables de salida ***motor*** y ***lenkung*** de AUTO.

El controlador es definido por 16 reglas, mostradas en la Tabla B.14.

IF (l_abs=.)	**(v_abs=.)**	**(r_abs=.)**	**(gesch=.)**	**THEN**
	v_nah		mittel	motor=drosseln
	v_nah		schnell	motor=drosseln
	v_nah		langsam	motor=ok
	v_mittel		mittel	motor=ok
	v_mittel		schnell	motor=ok
	v_weit		mittel	motor=ok
	v_weit		schnell	motor=ok
	v_mittel		langsam	motor=erhohen
	v_weit		langsam	motor=erhohen
l_nah	v_nah	r_weit		lenkung=stark_r
l_nah	v_nah	r_mittel		lenkung=rechts
l_nah	v_nah	r_nah		lenkung=gerade
l_mittel	v_nah	r_mittel		lenkung=gerade
l_weit	v_nah	r_weit		lenkung=gerade
l_mittel	v_nah	r_nah		lenkung=links
l_weit	v_nah	r_nah		lenkung=stark_l

Tabla B.14 – Conjunto de reglas de la aplicación AUTO

APÉNDICE C: CIRCUITO DE LA ARQUITECTURA BÁSICA

La Fig. C.1 muestra la captura de esquemas usando la herramienta *ViewDraw* (de *Viewlogic*) para el circuito completo con la interfaz de entrada salida. Se distinguen los *pads* de entrada y salida, el circuito propiamente dicho, los *pads* de alimentación y algo de la lógica adicional necesaria para el funcionamiento.

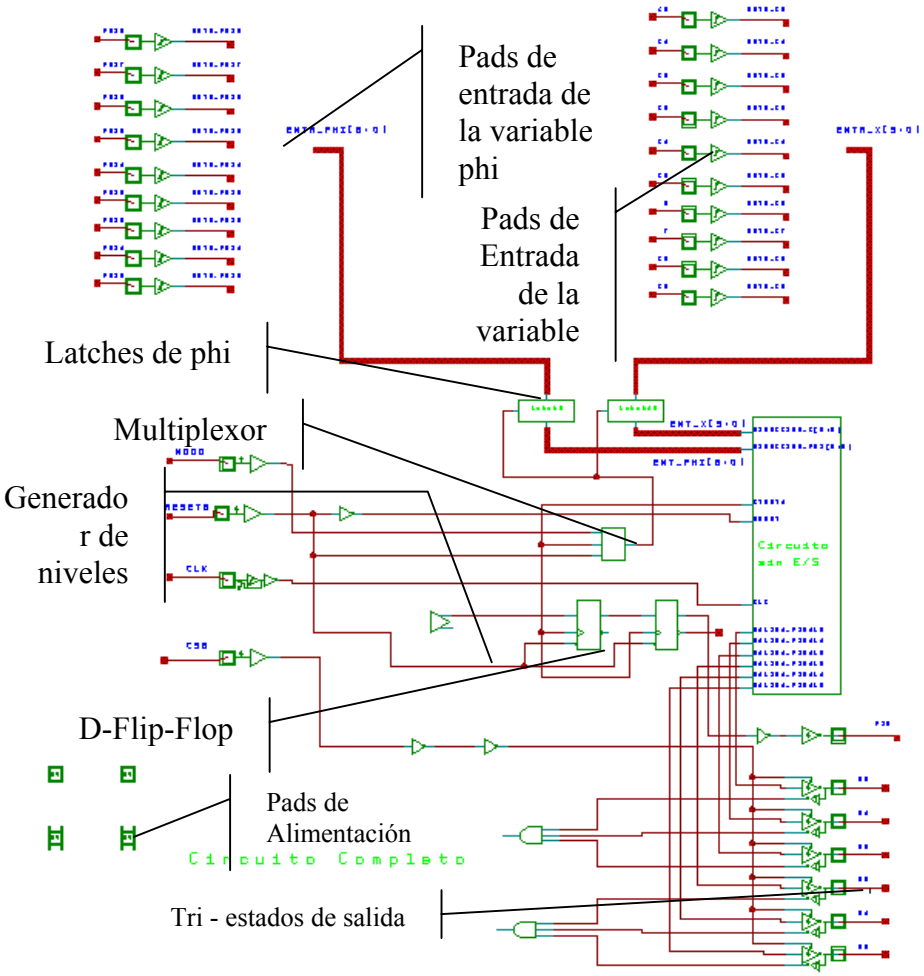


Fig. C.1 - Circuito completo con los puertos de entrada / salida

Además se puede ver en el circuito 2 *Flip-Flop* tipo D con reset

(DFFR: *D-type Flip-flop with reset*), los que forman un contador de 4 estados. Se puede ver también un *multiplexor* MUX21 (*2-input multiplexer*). El componente LOG2 (*Logical level*) es exigido por ES2 para conectar puertas a 0 o 1 (para las etapa de *place & route* no se pueden dejar referencias a VDD y GND).

C.1 - CIRCUITO COMPLETO

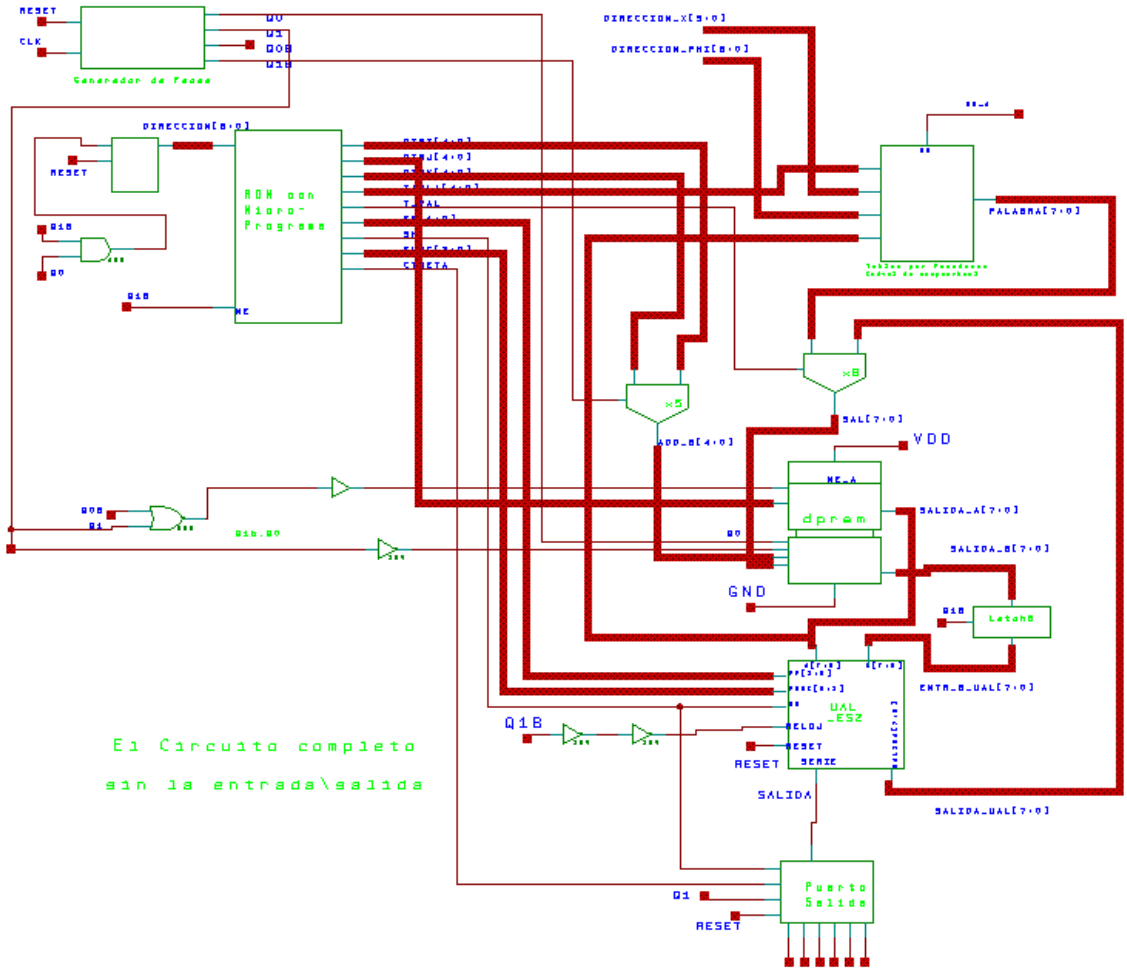


Fig. C.2 - Circuito completo sin las entradas salidas

El diagrama de bloques, de la Fig. C.2, es similar a los descriptos anteriormente. Naturalmente el contenido de los bloques será ahora compuertas de ES2. Existen algunas compuertas que efectúan cierta lógica adicional como son compuertas AND, OR e inversores, en tanto el contenido se especifica de aquí en adelante.

C.2 - GENERADOR DE FASES

Es un contador de 4 estados (2 bits), que genera las cuatro fases en que se descompone cada ciclo de microinstrucción (Fig. C.3). La señal de *reset* hace que vaya al estado 3, es decir en uno cada Flip-Flop. Consta de 2 Flip-Flop D con puesta a 1 (DFFS: *D-type Flyp-flop with Set*) y varios inversores (INV: *Inverter*). El doble inversor se utiliza para generar retardos.

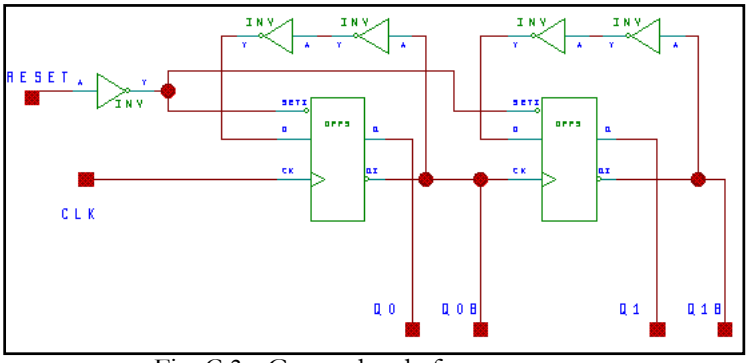


Fig. C.3 - Generador de fases

C.3 - CONTADOR DEL MICROPROGRAMA

Genera los 125 estados del microprograma, es decir cuenta de 0 a 124 y vuelve a cero (Fig. C.4). Cuenta además con una señal de *reset* que permite reiniciar el contador. Con una compuerta NOR y AND se generan los semi - sumadores (*Half Adders*) que efectúan la suma en uno. Como se puede observar, para el caso del primer bit basta con un inversor.

Los siete *Flip-Flops* D con *reset* se utilizan para guardar el estado actual. Para la vuelta a cero luego del número 124 se detecta este valor (en binario 1111100) a través de un NAND de 5 entradas, el que al ponerse en valor cero hace que todos los Flip-Flop se vuelvan a cero.

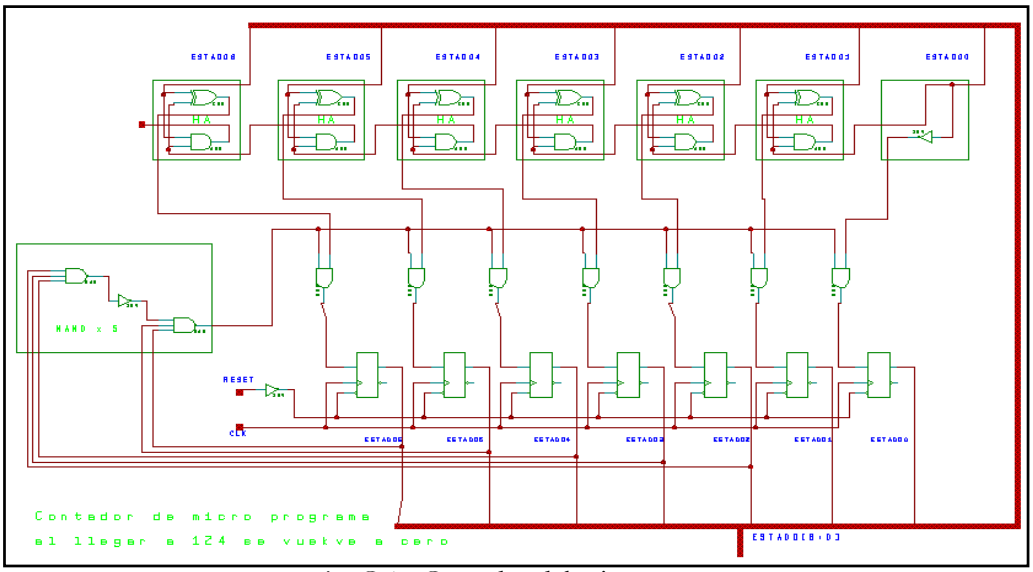


Fig. C.4 - Contador del microprograma

C.4 - ROM DE MICROPROGRAMA

Es una macrocélula generada en el CNM. Es una ROM de 125 palabras de 28 bits cada una, cuyo contenido se puede observar en la sección 5.3.6, del capítulo 5. El contenido del símbolo es código VHDL producido por el compilador de macrocélulas.

C.5 - RAM DE DOBLE PUERTO

Es otra macrocélula generada en el CNM. Es una RAM con dos puertos de lectura simultáneos y uno de escritura. Cuenta con 32 registros (direcciones de 5 bits) de 8 bits cada uno. Si bien no son necesarios 32 registros para la ejecución de este microprograma las memorias se generan en potencias de a dos.

C.6 - UNIDAD ARITMÉTICO LÓGICA

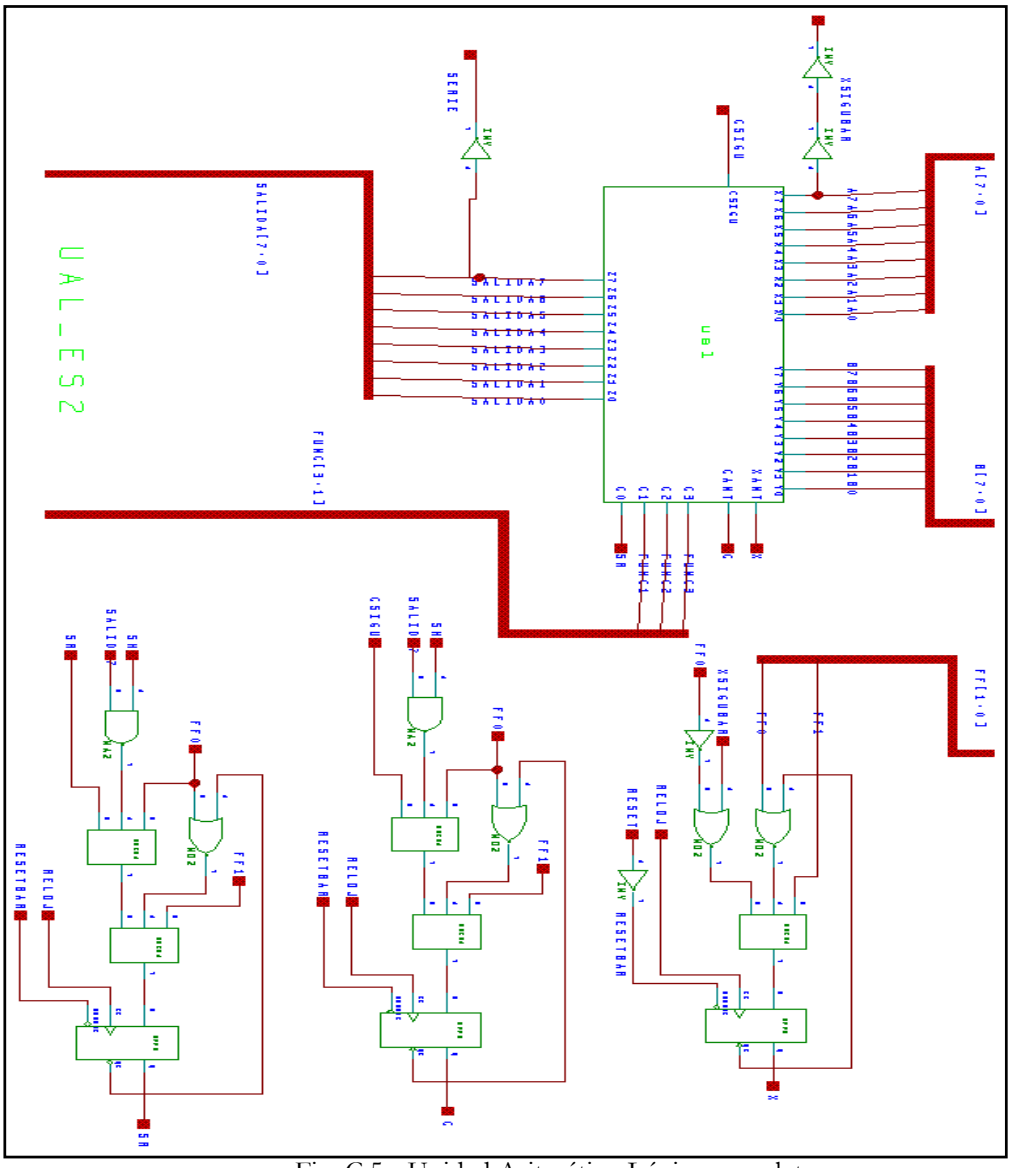


Fig. C.5 - Unidad Aritmético Lógica completa

Para una visualización mas clara se ha organizado el circuito que implementa la UAL en varias partes. En la Fig. C.5 se ve el circuito completo que consta de la generación de señales y la UAL interna que se muestra en la Fig. C.6. La UAL interna esta conformada por 8 UALs de un bit iguales, lo que deja entrever la facilidad de generación de unidades de diferentes precisiones.

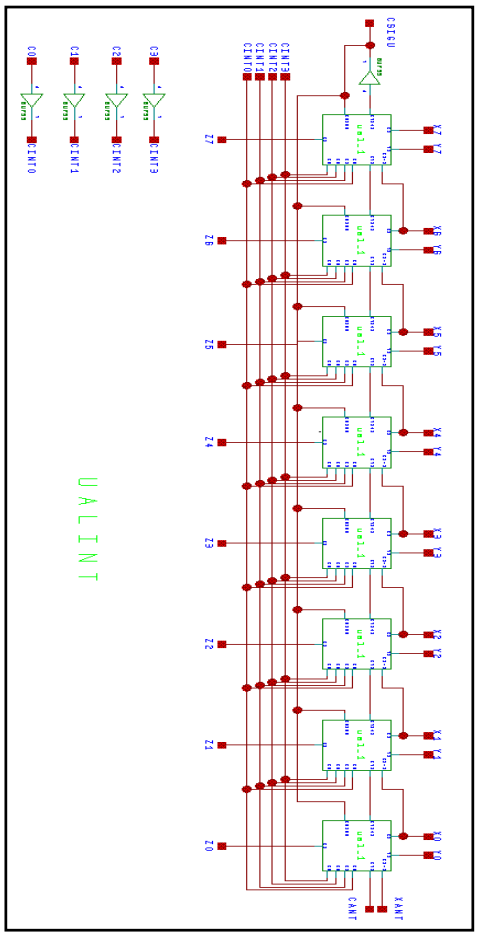


Fig. C.6 - UAL parte interna

La Fig. C.7 muestra como esta construida la UAL de un bit, donde se pueden ver los 3 multiplexores de 2 bits de entrada, el sumador completo de un bit y las compuertas NAND, XOR y NOT accesorias. La explicación de la lógica y funcionamiento de la UAL se puede encontrar en el capítulo 4 (Arquitectura Básica) sección 4.7.

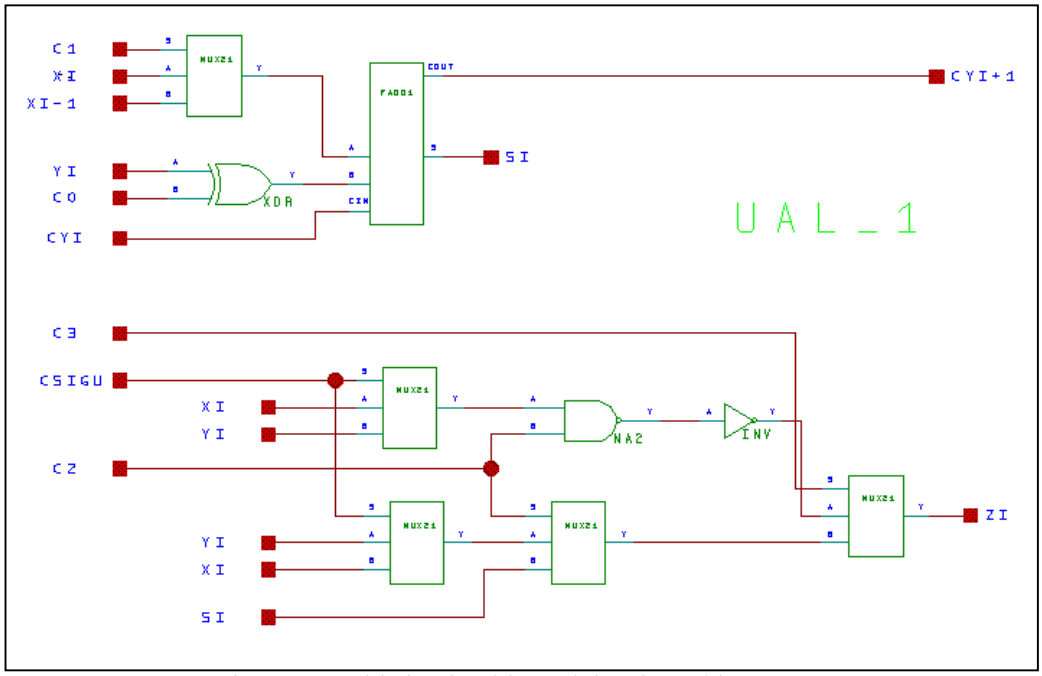


Fig. C.7 - Unidad Aritmético Lógica de un bit

C.7 - LATCH DE 8 BITS

Esta báscula que guarda el valor del puerto B de la DP-RAM tiene como objeto mantener el valor de este operando constante para los cálculos de la UAL (recuérdese que el puerto B de la DPRAM es también el destinatario del resultado del cálculo). Constructivamente no son mas que 8 latches conectados a una señal de habilitación en común (Fig. C.8).

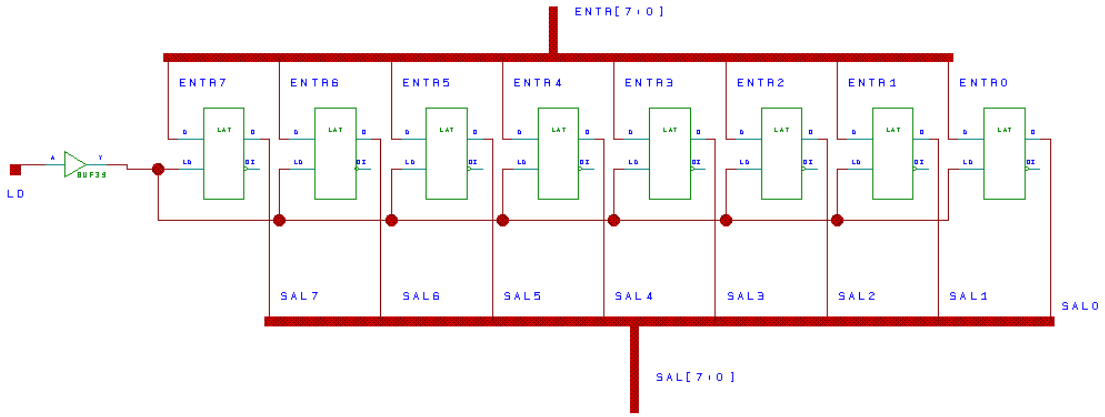


Fig. C.8 - Unidad Aritmético Lógica de un bit

C.8 - MULTIPLEXORES DE 5 Y 8 BITS

Simplemente son varios multiplexores juntos que comparten la señal de selección. El multiplexor de 8 bits utiliza 2 multiplexores de 4 bits en tanto que el de 5 utiliza uno de 4 y otro de 1. En la Fig. C.9 se puede ver el multiplexor de 8 bits

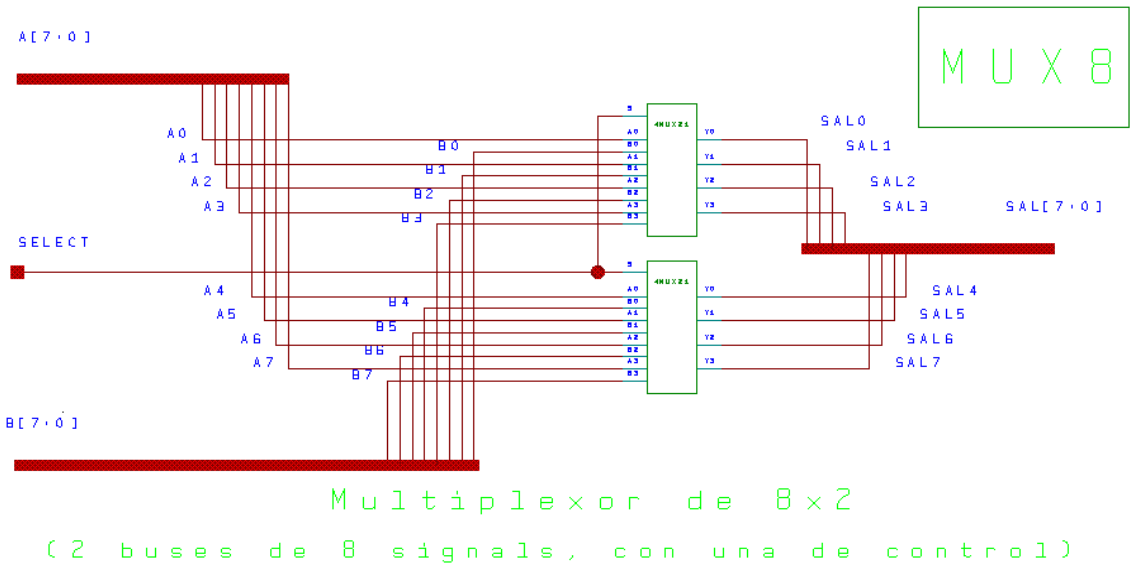


Fig. C.9 - Multiplexor de 8 bits y uno de control

C.9 - PUERTO DE SALIDA

El puerto de salida (Fig. C.10) es el encargado de recoger el resultado de la división que realiza la UAL, por este motivo la entrada al circuito es una señal de tamaño un bit (*SERIN*). Una serie de *Flip-Flop* D son los encargados de hacer las veces de registro de desplazamiento (*shift register*), la señal CTHETA (que proviene del microprograma) es la encargada de ordenar a los *latches* que guarden el resultado interno de la división y lo dejen disponible como salida del circuito.

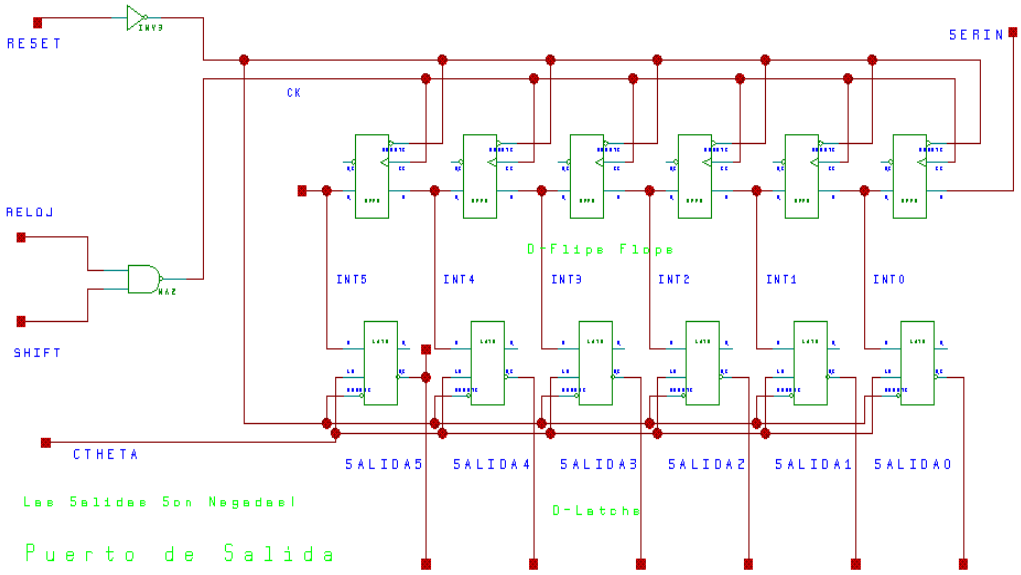


Fig. C.10 - Puerto de salida

C.10 - TABLAS DE FUSIFICACIÓN Y DEFUSIFICACIÓN

Como se mencionó anteriormente las tablas almacenadas en ROM tendrían un tamaño demasiado grande, por ello es que se implemento un circuito que efectuara los cálculos de las funciones de pertenencia y descodificación (Fig. C.11). Los detalles teóricos de esta implementación pueden encontrarse en [Aco96].

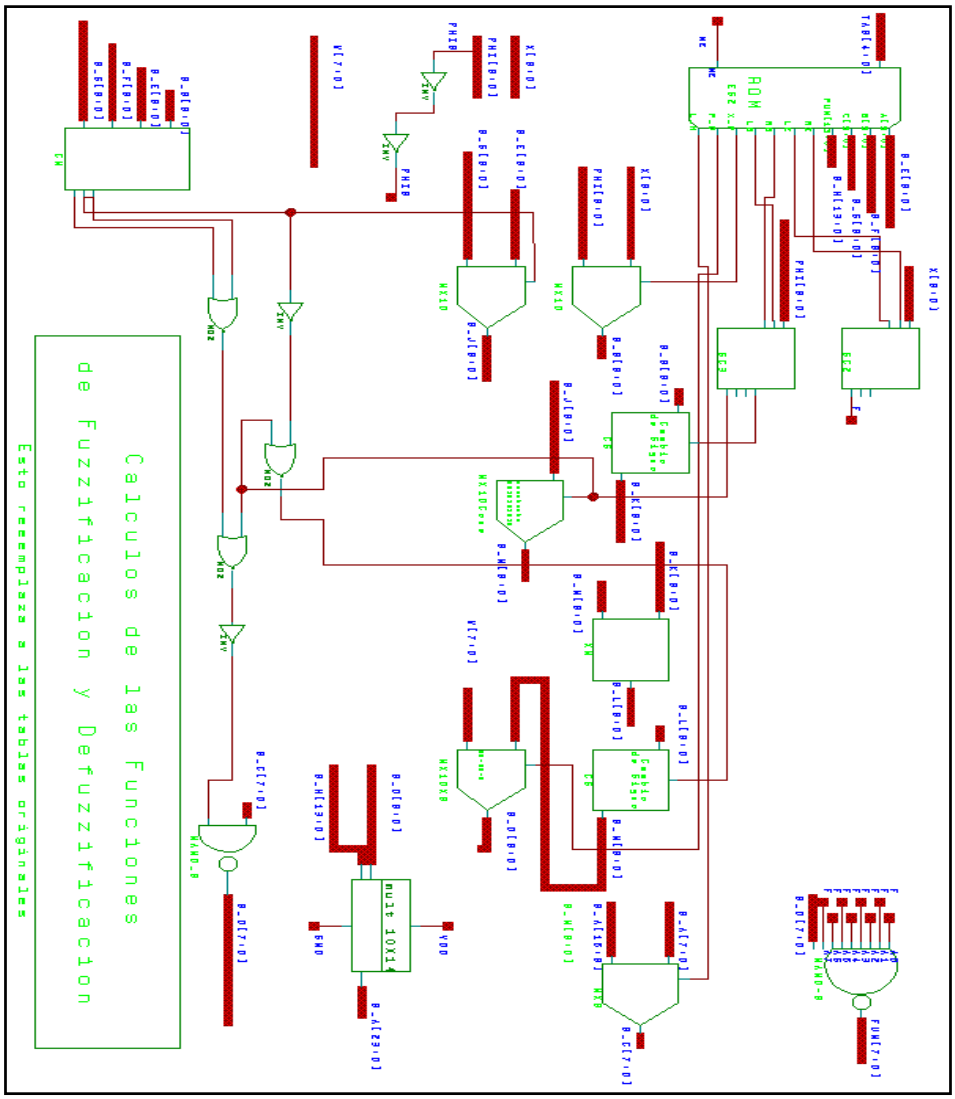


Fig. C.11 - Circuito de cálculo de las funciones de pertenencia y decodificación

La ROM de este circuito contiene los valores de las pendientes de las funciones de codificación y decodificación, luego una serie de comparardores, multiplexores y un multiplicador realizan el trabajo de calculo de las funciones. La ROM de 31 palabras de 51 bits así como el multiplicador de 14 × 10 bits son macrocélulas generadas en el CNM.

En la Fig. C.12. Se puede ver la estructura del módulo CM que compara el valor de entrada *VAR* con los tres valores *A*, *B* y *C*, e informa si es mayor, menor o igual que cada valor de referencia.

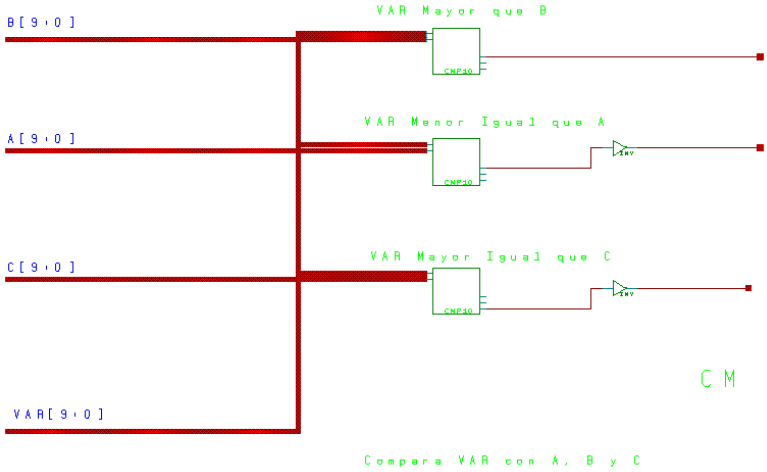


Fig. C.12 - Modulo CM que compara un valor con otros 3 de referencia

La estructura de los comparadores de 10 bits se pueden ver en la sección C.13. Estos a su vez se componen de 9 comparadores de un bit mas uno especial para el bit de menor peso, ya que este es más simple. El módulo toma entonces 2 valores de 10 bits e informa a través de 3 bits de salida si es menor, igual o mayor uno respecto del otro (Lower, Equal, or Grater).

El módulo CM1 correspondiente al comparador de un bit se puede ver en la Fig. C.14 donde se distinguen las entradas (3 bits de la etapa anteriores mas los 2 bits de los valore a comparar A y B) y las salidas (siendo los 3 bits para la etapa posterior).

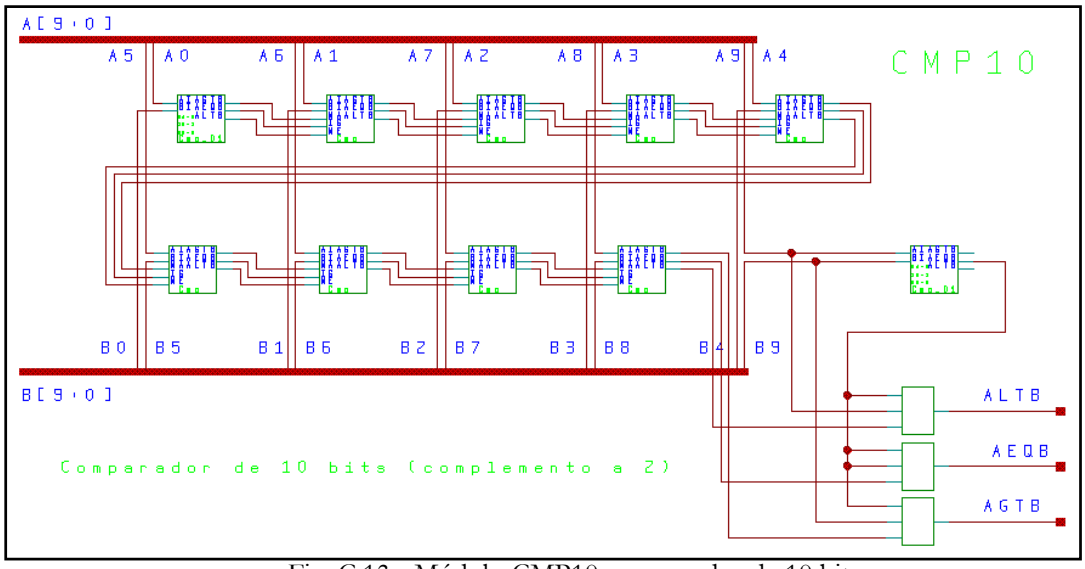


Fig. C.13 - Módulo CMP10 comparador de 10 bits

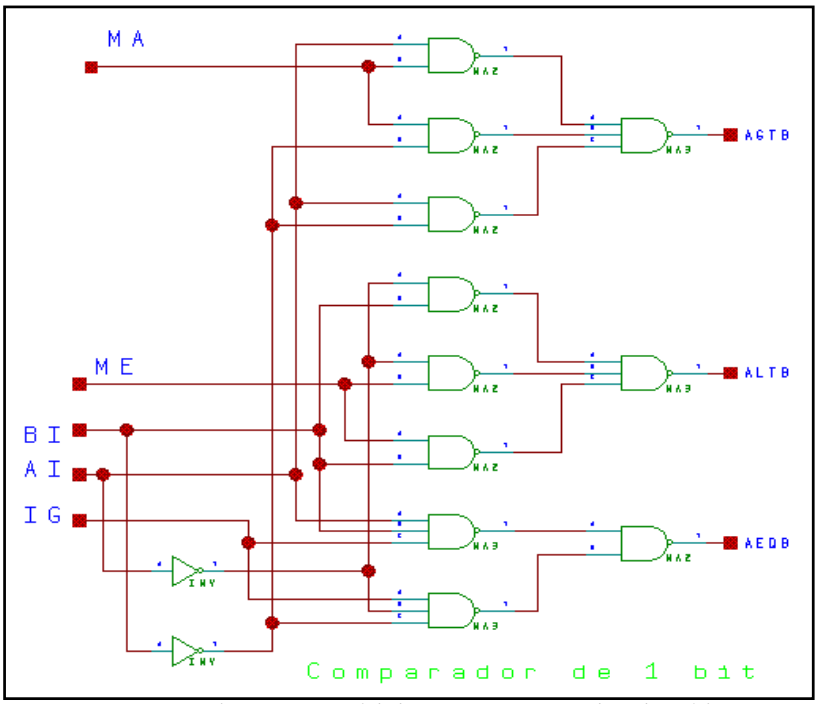


Fig. C.14 - Módulo CM1 comparador de 1 bit

La entrada IG (iguales) corresponde a la salida AEQB (A igual B) del comparador anterior, se compara este bit con los valores de AI y BI (valores *i-esimos* de los vectores a comparar) y en caso de seguir siendo iguales se pone en uno la salida AEQB (A igual a B). La entrada MA corresponde a la salida AGTB (A mayor que B) de la etapa anterior si A esta en uno o B en cero, directamente la salida AGTB (A mayor que B) de esta etapa se hace uno. El razonamiento para el resto de las combinaciones es análogo.

Obsérvese la utilización de un doble plano NAND en vez de un AND-OR (o recíprocamente OR-AND) para la resolución de estos problemas *booleanos*. Esto se justifica en la menor cantidad de transistores necesarios para construir las compuertas NAND respecto de las otras y que es preferible integrar compuertas homogéneas que no combinaciones de varias otras.

Los *multiplexores* de 10 bits se pueden ver en la Fig. C.15 no son mas que dos multiplexores de 4 bits de tamaño (4MUX21: *4×2-input multiplexer*), mas dos de un bit (MUX21: *2-input multiplexer*). A la salida de los multiplexores se han agregado amplificadores.

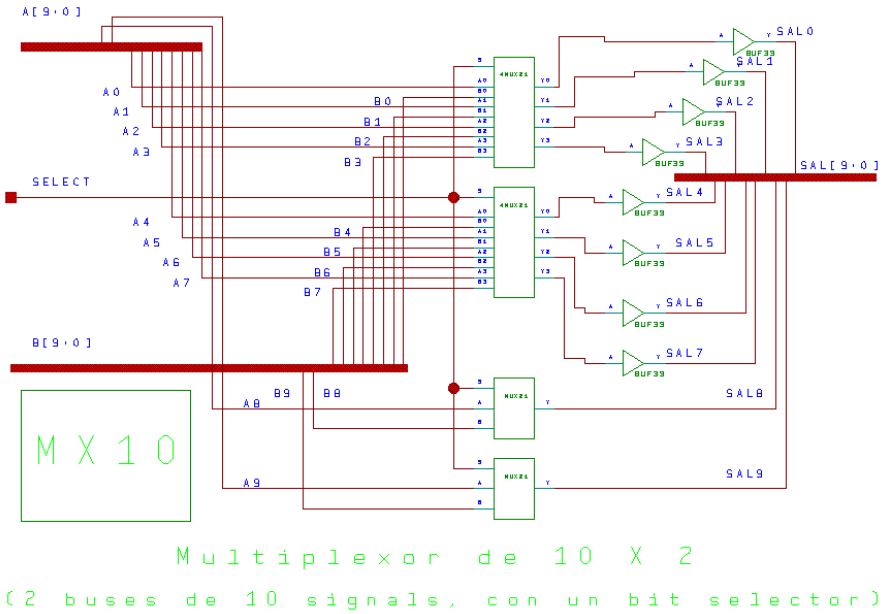


Fig. C.15 - Multiplexor de 10 bits

El módulo CS es de cambio de signo y se puede ver en la Fig. C.16. Los números están representados en complemento a la base con 10 bits y se utiliza el clásico algoritmo en que se dejan los bits inferiores tal como son, hasta llegar al primer uno del vector de entrada, donde a partir del siguiente a este se comienza a complementar bit por bit. La señal EN (*Enabled* - *habilitado*) en uno hace que el circuito complemente, con cero deja el número tal como era.

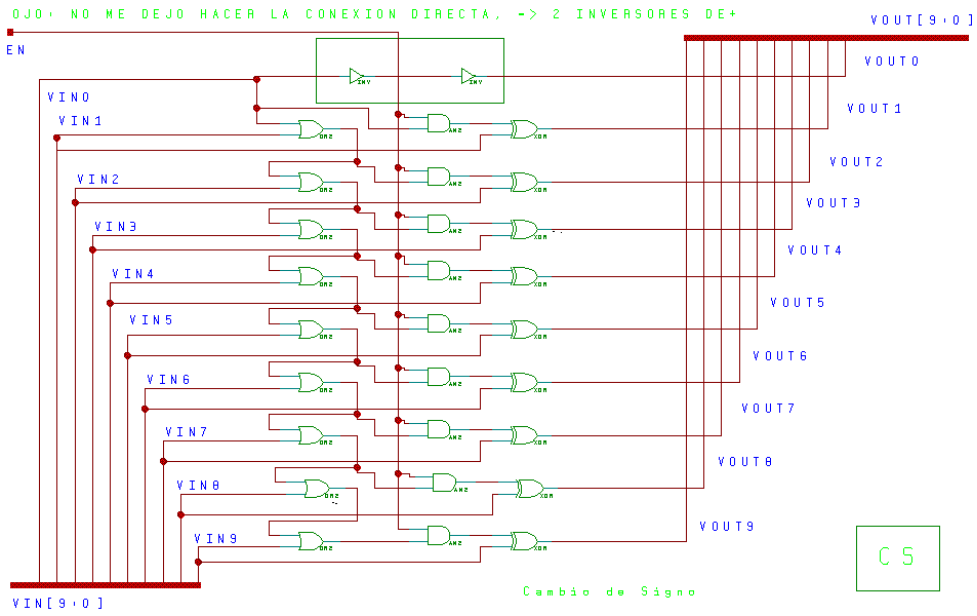


Fig. C.16 - Circuito de cambio de signo en complemento a la base

El módulo XM se trata de un restador de 10 bits en complemento a dos y se puede observar en la Fig. C.17. El restador complementa el valor del operando *VINB* y se lo suma al operando *VINA* mas uno (del cambio de signo en base 2). Se utiliza sumadores completos (full Adders) provisto en la biblioteca de ES2 (FA) no siendo para el bits de menor peso en que se construyo a "mano" un sumador completo cuyo señal de acarreo previo ya estaba en uno (naturalmente así se utilizan menos transistores que usando un full adder tradicional).

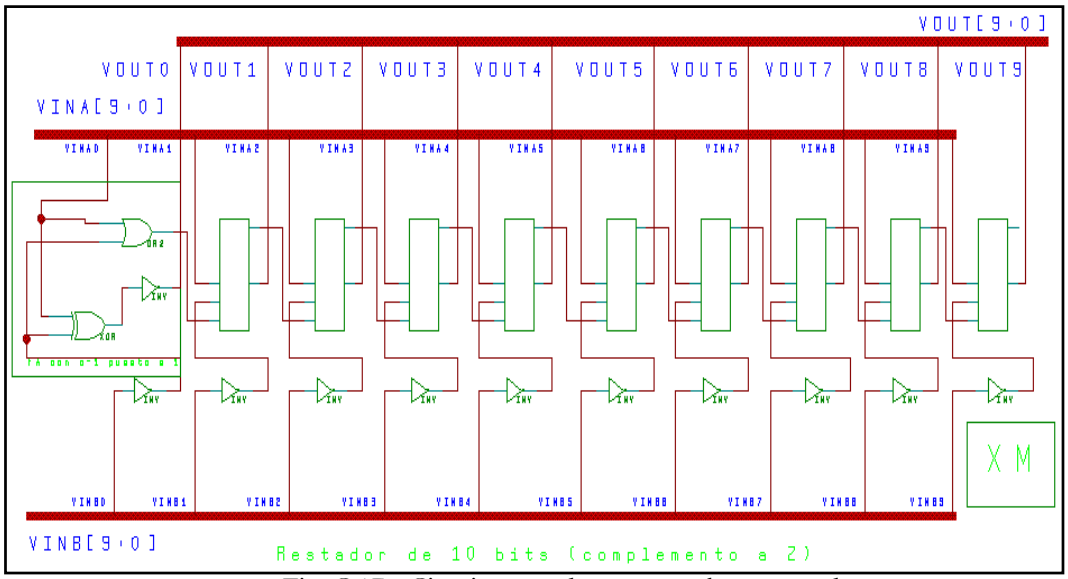


Fig. C.17 - Circuito restador en complemento a dos

El módulo MX10CONS es un multiplexor de 10 bits y una señal de control, pero uno de sus valores es siempre constante. Por ello se reemplazo el clásico circuito del multiplexor por el que se muestra en la Fig. C.18. La constante utilizada como entrada B es el valor 0010101010. El resultado de esto es un circuito con muchas menos compuertas que la implementación clásica.

El resto de los módulos no se muestran aquí, los módulos SC2 y SC3 efectúan comparaciones de los valores de entrada X y PHI respectivamente con respecto a valores constantes. El módulo NAND_8 efectúa un NAND bit por bit al vector de entrada; está implementado por 8-NANDs de dos entradas con una señal en común para todos.

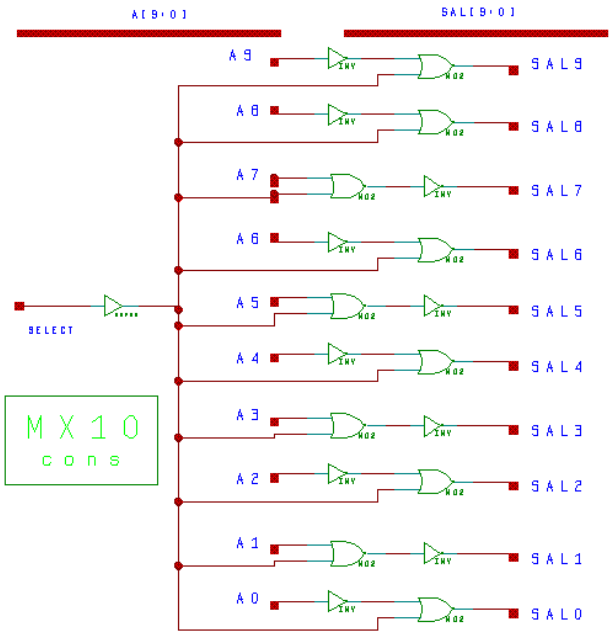


Fig. C.18 - Multiplexor de con una entrada de 10 bits y la constante 0010101010

C.11 - DEPURACIÓN Y SIMULACIÓN DEL CIRCUITO

Al igual que la simulación de modelos VHDL se puede utilizar todas las ventajas y facilidades provistas por ViewLogic. Para la etapa de depuración de los circuitos se utilizó la herramienta ViewTrace que proporciona la posibilidad de seleccionar las señales internas a visualizar, la cantidad y precisión de ciclos de simulación y otras informaciones del circuito. En la Fig. C.19 se puede ver el aspecto presentado por esta herramienta, donde se han colocado varias señales y buses para su inspección y se ha simulado por alrededor de 45 ciclos de la señal de clock, en [Vie93] se puede encontrar una descripción más amplia de la herramienta.

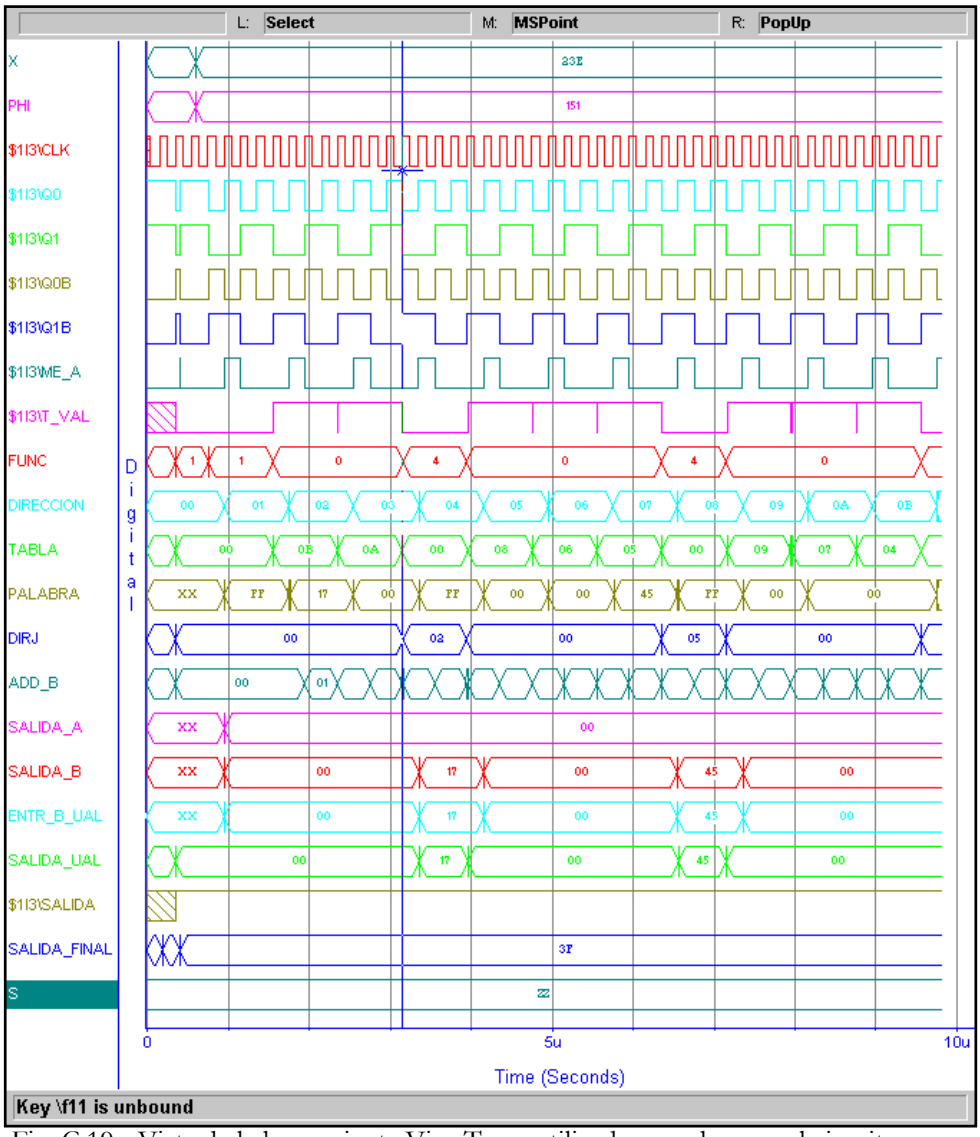


Fig. C.19 - Vista de la herramienta ViewTrace utilizada para depurar el circuito

La simulación del circuito tiene el mismo aspecto y lenguaje de comandos que en el caso de los modelos VHDL, mas aun se utilizan los mismos archivos de comandos, a fin de comparar los resultados con los antes obtenidos con la simulación en software y más tarde en VHDL. En esta etapa se disponía de un gran número de valores de entrada - salida generados con anterioridad, tanto en la simulación software como con los modelos VHDL los que sirven para contrastar con los valores que ahora se producen.

APÉNDICE D: SIMULACIÓN DE LOS FLC

D.1 - INTRODUCCIÓN

El objetivo de este Apéndice es estudiar y describir los resultados obtenidos de la simulación de los FLC generados con las herramientas automáticas propuestas y descritas en esta memoria. La simulación del código VHDL es realizada utilizando *Active HDL 3.6* de Xilinx. Para simular el funcionamiento de todos los componentes externos se implementa un módulo VHDL adicional llamado TestBench, para estimular las señales del controlador de forma sincronizada. Una vez realizado el diseño del FLC mediante el archivo de especificación de controladores se genera la descripción VHDL de ambas arquitecturas. Luego, se crea un nuevo proyecto en el *Active HDL* para cada arquitectura, y se incluyen los archivos con las descripciones en cada uno.

D.2 – CARACTERISTICAS DEL FLC

Para estudiar y describir las simulaciones utilizando *Active HDL 3.6* se elige el FLC planteado por el Dr. Bart Kosko, el que plantea el problema del vehículo que debe moverse para que su parte trasera coincida con el centro de un muelle de carga, conocido en la literatura como "*Truck backer-upper control systems*" [Kos92].

Las variables de entrada son: ***X*** (posición en el eje horizontal sobre el cual está el muelle de carga, en el rango de -500 a 500 centímetros) y ***PHI*** (ángulo del camión con respecto a la horizontal en el rango -180 a 180 grados), mientras que la de salida es: ***THETA*** (ángulo de las ruedas del camión en el rango de -30 a 30 grados). Se consideran cinco funciones de pertenencia para ***X*** (***LE***, ***LC***, ***CE***, ***RC***, ***RI***) y siete para ***PHI*** (***RB***, ***RU***, ***RV***, ***VE***, ***LV***, ***LU***, ***LB***). Todas las variables y

sus funciones de pertenencia se representan con 8-bits. Se recuerda que la solución propuesta utiliza 35 reglas. El archivo con la descripción NFIL es el siguiente:

```
invar("x"  , 0, 0, 0, 255, 255)
invar("phi", 1, 0, 0, 255, 255)
f_p("LE",0,"x",0,255,31,255,97,0)
f_p("LC",1,"x",76,0,101,255,127,0)
f_p("CE",2,"x",115,0,128,255,140,0)
f_p("RC",3,"x",129,0,154,255,180,0)
f_p("RI",4,"x",158,0,224,255,255,255)
f_p("RB",5,"phi",0,0,32,255,71,0)
f_p("RU",6,"phi",59,0,83,255,107,0)
f_p("RV",7,"phi",93,0,110,255,127,0)
f_p("VE",8,"phi",119,0,128,255,136,0)
f_p("LV",9,"phi",129,0,146,255,163,0)
f_p("LU",10,"phi",148,0,172,255,196,0)
f_p("LB",11,"phi",184,0,223,255,255,0)
outvar("theta", 0, 0, 0, 60, 255)
f_d("NB",0,"theta",0,255,0,255,14,0)
f_d("NM",1,"theta",5,0,15,255,25,0)
f_d("NS",2,"theta",17,0,24,255,30,0)
f_d("ZE",3,"theta",25,0,30,255,35,0)
f_d("PS",4,"theta",30,0,36,255,43,0)
f_d("PM",5,"theta",35,0,45,255,55,0)
f_d("PB",6,"theta",46,0,60,255,60,255)
si([is("x","LE"),is("phi","RB")],is("theta","PS"))
si([is("x","LC"),is("phi","RB")],is("theta","PM"))
si([is("x","CE"),is("phi","RB")],is("theta","PM"))
si([is("x","RC"),is("phi","RB")],is("theta","PB"))
si([is("x","RI"),is("phi","RB")],is("theta","PB"))
si([is("x","LE"),is("phi","RU")],is("theta","NS"))
si([is("x","LC"),is("phi","RU")],is("theta","PS"))
si([is("x","CE"),is("phi","RU")],is("theta","PM"))
si([is("x","RC"),is("phi","RU")],is("theta","PB"))
si([is("x","RI"),is("phi","RU")],is("theta","PB"))
si([is("x","LE"),is("phi","RV")],is("theta","NM"))
si([is("x","LC"),is("phi","RV")],is("theta","NS"))
si([is("x","CE"),is("phi","RV")],is("theta","PS"))
si([is("x","RC"),is("phi","RV")],is("theta","PM"))
si([is("x","RI"),is("phi","RV")],is("theta","PB"))
si([is("x","LE"),is("phi","VE")],is("theta","NM"))
si([is("x","LC"),is("phi","VE")],is("theta","NM"))
si([is("x","CE"),is("phi","VE")],is("theta","ZE"))
si([is("x","RC"),is("phi","VE")],is("theta","PM"))
si([is("x","RI"),is("phi","VE")],is("theta","PM"))
si([is("x","LE"),is("phi","LV")],is("theta","NB"))
si([is("x","LC"),is("phi","LV")],is("theta","NM"))
si([is("x","CE"),is("phi","LV")],is("theta","NS"))
si([is("x","RC"),is("phi","LV")],is("theta","PS"))
si([is("x","RI"),is("phi","LV")],is("theta","PM"))
si([is("x","LE"),is("phi","LU")],is("theta","NB"))
si([is("x","LC"),is("phi","LU")],is("theta","NB"))
si([is("x","CE"),is("phi","LU")],is("theta","NM"))
si([is("x","RC"),is("phi","LU")],is("theta","NS"))
si([is("x","RI"),is("phi","LU")],is("theta","PS"))
si([is("x","LE"),is("phi","LB")],is("theta","NB"))
si([is("x","LC"),is("phi","LB")],is("theta","NB"))
si([is("x","CE"),is("phi","LB")],is("theta","NM"))
si([is("x","RC"),is("phi","LB")],is("theta","NM"))
si([is("x","RI"),is("phi","LB")],is("theta","NS"))
```

D.3 – SIMULACIÓN DE LA ARQUITECTURA PASIVA

Para realizar la simulación del controlador para arquitectura pasiva es necesario incluir en el proyecto del *Active HDL 3.6* los seis archivos VHDL (Tabla D.1).

Archivo VHDL	Descripción
main.vhd	Módulo principal del controlador
alu.vhd	Módulo principal de la unidad aritmético-lógica
Aluint.vhd	Módulo de la ALU encargado de realizar los cálculos
regbank.vhd	Módulo que implementa el banco de registros
dpram.vhd	Módulo que implementa la memoria de doble puerto
tablas.vhd	Módulo que implementa las tablas de memoria

Tabla D.1 – Archivos VHDL para la arquitectura pasiva

Cabe destacar que el número de archivos generados para la arquitectura pasiva es independiente del controlador diseñado, las diferencias se encuentran en el ancho de arquitectura, en el número de registros, la cantidad de tablas, etc. El código del módulo TestBench se muestra a continuación.

```
001  entity main_tb is
002  end main_tb;
003
004  architecture TB_ARCHITECTURE of main_tb is
005  component main
006  port(      i_mp : in std_logic_vector(27 downto 0);
007             reloj : in std_logic;
008             reset : in std_logic;
009             v_x   : in STD_LOGIC_VECTOR(7 downto 0);
010             v_phi : in STD_LOGIC_VECTOR(7 downto 0);
011             ph0 : in std_logic;
012             ph1 : in std_logic;
013             ph2 : in std_logic;
014             o_serie : out std_logic;
015             o_o : out std_logic;
016             o_salida : out std_logic_vector(7 downto 0) );
017  end component;
018   signal i_mp : std_logic_vector(27 downto 0);
019   signal reloj : std_logic;
020   signal reset : std_logic;
021  signal v_x   : STD_LOGIC_VECTOR(7 downto 0);
022   signal v_phi : STD_LOGIC_VECTOR(7 downto 0);
023   signal ph0 : std_logic;
024   signal ph1 : std_logic;
025   signal ph2 : std_logic;
```

```
026  signal o_serie : std_logic;
027  signal o_o : std_logic;
028  signal o_salida : std_logic_vector(7 downto 0);
029
030  begin
031  STIMUL: process
032  begin
033    i_mp       <= "00000000000000000000001000011";
034    v_x        <= "01100100";
035    v_phi      <= "01100100";
036    reset      <= '1';
037    wait for 100 ns;
038    i_mp       <= "00000000000000000000001000011";
039    reset      <= '0';
040    wait for 100 ns;
041    i_mp       <= "00001000000000000101110000000";   wait for 100 ns;
042    i_mp       <= "00010000000000000000110000000";   wait for 100 ns;
043    ...
044    i_mp       <= "10011100110010100000001011110";   wait for 100 ns;
045    i_mp       <= "00100100110000000000001111100";   wait for 100 ns;
046    report "Testeo de FLC completado";
047    wait;
048  end process;
049
050  UUT : main port map (i_mp => i_mp,
051                       reloj => reloj,
052                       reset => reset,
053                       v_x => v_x,
054                       v_phi => v_phi,
055                       ph0 => ph0,
056                       ph1 => ph1,
057                       ph2 => ph2,
058                       o_serie => o_serie,
059                       o_o => o_o,
060                       o_salida => o_salida );
061  end TB_ARCHITECTURE;
062
063  configuration TESTBENCH_FOR_main of main_tb is
064  for TB_ARCHITECTURE
065          for UUT : main use entity work.main(main_arch); end for;
066  end for;
067  end TESTBENCH_FOR_main;
```

El módulo TestBench se compone de dos bloques, el bloque ***UUT*** que mapea las señales del controlador, y el bloque ***STIMUL*** encargado de estimular las señales. Las señales que lo conectan con el controlador son: ***i_MP*** con la microinstrucción a ejecutar, ***v_X*** con el valor de la variable de entrada ***X***, ***v_PHI*** con el valor de la variable de entrada ***PHI***, ***O_serie*** es la señal de salida serie de la ALU, ***O_salida*** es la señal de salida con el valor de la defusificación, ***Reset*** para la puesta a cero asincrónica de todo el circuito, mientras que ***Reloj***, ***Ph0***, ***Ph1*** y ***Ph2*** son señales de sincronización.

El proceso ***STIMUL*** es el encargado de estimular las señales del controlador, haciendo ejecutar el microprograma; Para lo cual realiza la siguiente secuencia de pasos:

1º Paso: Consiste en inicializar el controlador, mediante la ejecución de las instrucciones de las líneas 33 a la 37. La señal ***i_MP*** es inicializada con la primera microinstrucción, las señales ***v_X*** y ***v_PHI*** se inicializan con el valor correspondiente a las variables de entrada (100 en este caso), y la señal ***Reset*** recibe el valor 1 que permite realizar la puesta a cero asincrónica de todos los elementos. Para realizar la simulación con distintos valores de entrada, sólo es necesario cambiar los valores que se asignan a ***v_X*** y ***v_PHI***. Luego de realizadas las asignaciones se espera por 100ns.

2º Paso: Restaura el valor de la señal ***Reset***, mediante la ejecución de las instrucciones de las líneas 38 a la 40. Se asigna el valor cero a ***Reset***, y por lo tanto se causa que el controlador ejecute la microinstrucción contenida en la señal ***i_MP.*** Luego se espera por otros 100ns a que la ejecución de la microinstrucción finalice.

3º Paso: Carga la próxima microinstrucción que debe ejecutar el FLC. Una vez asignada la microinstrucción se espera 100ns (tiempo de ejecución), y luego se asigna la próxima microinstrucción. El 3º Paso se repite tantas veces como instrucciones tenga el microprograma, en este caso desde la línea 41 a la 45.

En resumen se puede decir que el proceso ***STIMUL*** tiene como función estimular al controlador asignándo una microinstrucción cada 100ns.

La Fig. D.1 muestra el diagrama de señales del Waveform, con la simulación de la arquitectura pasiva, que han sido almacenadas en el archivo "**Camion.awf**". Donde la primera columna del gráfico (**Name**) contiene los nombres de las señales que se están analizando. En la segunda columna del gráfico (**Value**) se muestra el valor de las señales en un determinado de punto del tiempo, en la tercer columna (**Stimulator**) se puede ver el tipo de estimulo que reciben las señales y por último, en la parte derecha del gráfico, se puede observar la

representación gráfica de las señales y sobre ellas la escala de tiempo utilizada.

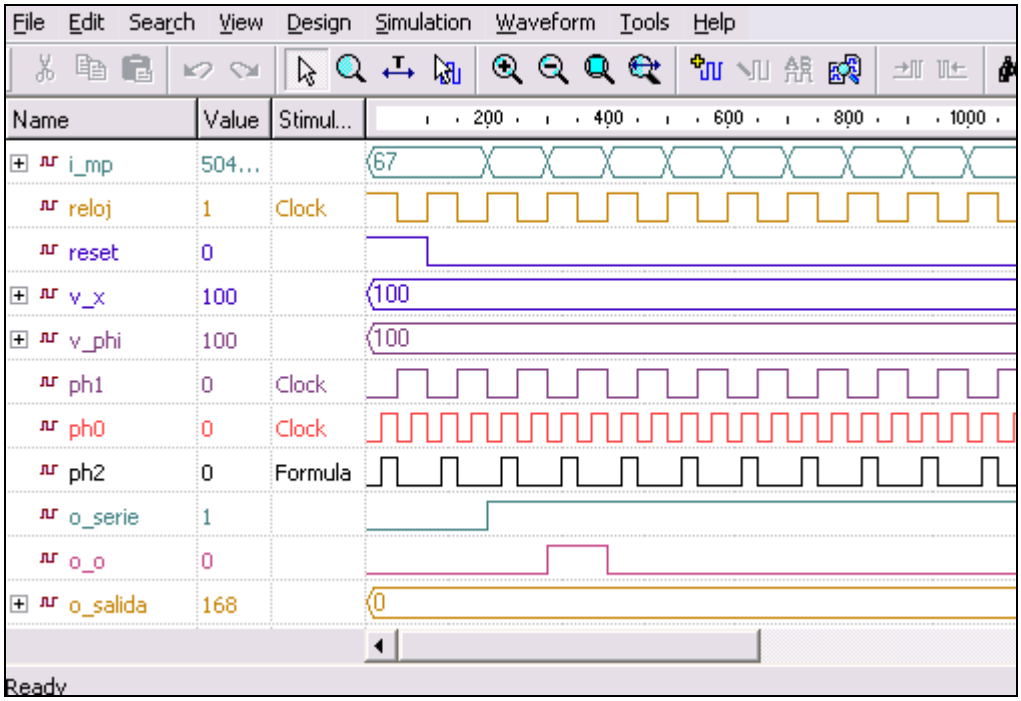


Fig. D.1 – Diagrama de señales del Waveform de la arquitectura pasiva

El proceso ***STIMUL*** es el encargado de estimular las señales del controlador, pero las señales de sincronización ***Reloj***, ***Ph0***, ***Ph1*** y ***Ph2*** se estimulan por medio de primitivas del WaveForm. Estas señales sincronizan el FLC y se conectan en el Waveform de la siguiente manera:

- ***Reloj***, se conecta a un reloj con un ciclo de 100ns (primer pulso alto).
- ***Ph0***, se conecta a un reloj con un ciclo de 50ns (primer pulso bajo).
- ***Ph1***, se conecta a un reloj con un ciclo de 100ns (primer pulso bajo).
- ***Ph2***, se conecta a un reloj descrito por formula con el ciclo (pulso alto de 25ns y pulso bajo de 75ns), donde la señal comienza con un pulso bajo de 25 ns.

D.3.1.-Fusificación

La fusificación es implementada como accesos a las tablas, que contiene los valores de las funciones de pertenencia. En la Fig. D.2 se muestra el diagrama de señales de acceso a las tablas de las funciones de pertenencia, correspondientes al

acceso a la tabla de la función ***RV*** de la variable ***PHI***.

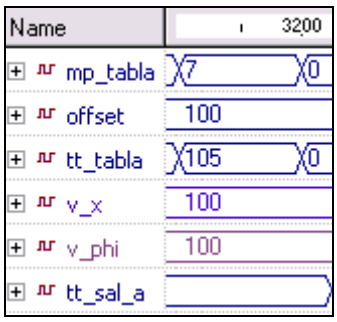


Fig. D.2 – Señales de acceso a las tablas de las Funciones de Pertenencia

Donde las señales son: ***mp_tabla*** es el número de tabla (definido por la microinstrucción, 7 para la función ***RV***), ***offset*** es la dirección de desplazamiento dentro de la tabla (100 para ***phi***) y ***tt_tabla*** devuelve el valor leído de la tabla. Las señales ***v_x*** y ***v_phi*** corresponden a los valores de las variables de entrada y ***tt_sal_a*** corresponde al valor de salida del puerto ***A*** del banco de registros. La señal ***offset*** toma el valor de ***tt_sal_a*** cuando se accede a una función de decodificación.

D.3.2.-Elementos de Memoria

La Fig. D.3 muestra las señales que describen el funcionamiento del banco de registros (izquierda) y la memoria de doble puerto DPRAM (derecha).

La diferenciación en el tratamiento se debe a que para acceder al banco de registro se utilizan las direcciones de tres registros, sin embargo el banco de registros puede acceder a la memoria DPRAM de a dos registros simultáneamente. Por esta razón es necesario lograr una sincronización exacta para poder leer primero los registros correspondientes a los operandos, realizar las operaciones y escribir el resultado, todo en un mismo ciclo de reloj.

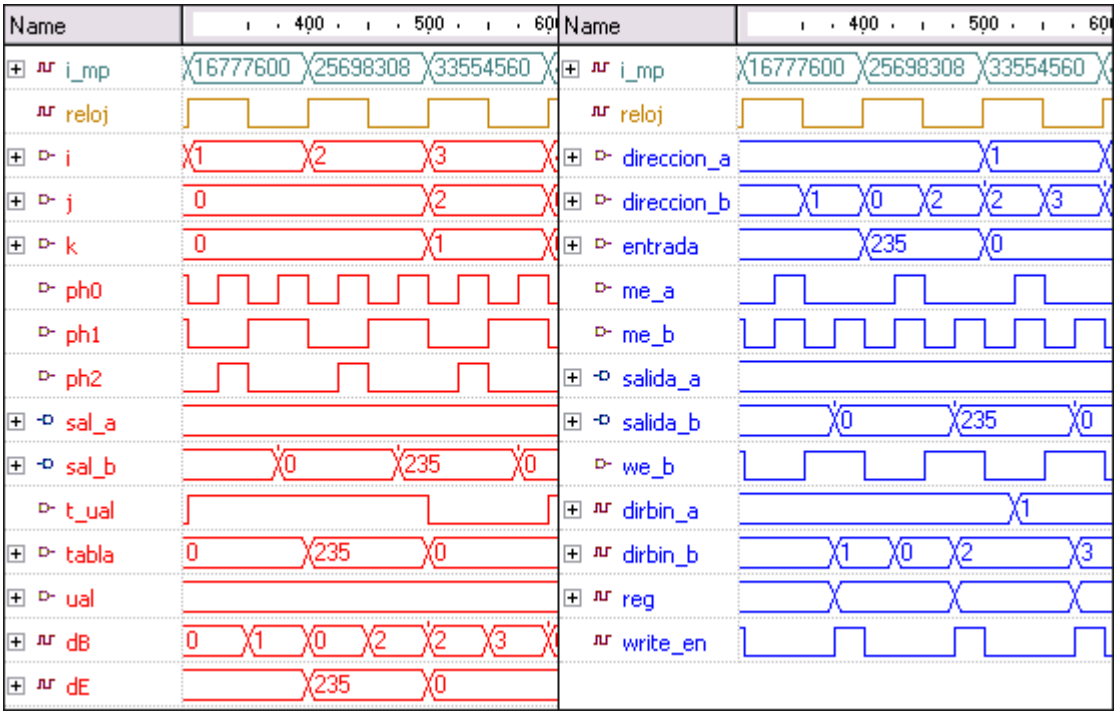


Fig. D.3 – Señales del banco de registro (izq) y de la DPRAM (der)

La Fig. D.3 muestra el diagrama de señales del banco de registro (izquierda). Las señales ***i***, ***j*** y ***k*** son los identificadores de los registros involucrados en la operación, ***i*** es el registro de escritura y los otros dos corresponden a los operandos. Para realizar la sincronización son utilizadas las señales ***Ph0***, ***Ph1*** y ***Ph2*****,** por cada ciclo de reloj hay dos ciclos de ***Ph0***. Las señales ***sal_a*** y ***sal_b*** corresponden a las salidas de los puertos ***A*** y ***B*** respectivamente, correspondientes a la lectura de los registros. La señal ***t_ual*** indica si el valor que se debe almacenar proviene de una tabla o de la UAL. Las señales ***tabla*** y ***ual*** corresponden a los valores obtenidos de la lectura de las tablas y de la unidad aritmético-lógica respectivamente. La señal ***db*** es la que contiene la dirección del registro que se quiere acceder a través del puerto ***B*** de lectura/escritura de la DPRAM. Cuando la señal ***ph1*** tiene un valor bajo la señal ***db*** toma el de ***j*** y cuando tiene un valor alto toma el valor de ***i***. La señal ***de*** es pasada a la DPRAM con el valor que se quiere almacenar, según el estado de ***t_ual*** toma el valor de ***tabla*** o ***ual.***

Estas son todas las señales utilizadas en banco de registros y mediante las cuales puede acceder a la memoria DPRAM, de a dos registros simultáneamente y en forma sincronizada.

Por otro lado, la Fig. D.3 muestra el diagrama de señales del banco de registros de la DPRAM (parte derecha). Las señales ***direccion_a*** y ***direccion_b*** contienen las direcciones de los registros que son accedidos a través de los puertos ***A*** y ***B*** respectivamente, ***entrada*** es el valor que se escribe en el puerto ***B***, ***salida_a*** y ***salida_b*** son los valores leídos de los puertos ***A*** y ***B***. Las señales utilizadas para el control de lectura y escritura de los registros son ***me_b***, ***we_b*** y ***me_a*** que se corresponden con las señales ***ph0***, ***ph1*** y ***ph2*** respectivamente. Cuando ***me_a*** alcanza un valor alto habilita la lectura del puerto ***A*** y ***me_b*** alcanza un valor alto habilita la lectura del puerto ***B***. La habilitación de escritura del puerto ***B*** se logra cuando la señal ***write_en*** toma un valor alto, dicha señal alcanza el valor alto cuando las señales ***me_b*** y ***we_b*** tienen un valor alto. La señal interna ***dirbin_a*** contiene la dirección del registro a acceder a través del puerto ***A*** y cuyo valor lo toma de ***direccion_a*** cuando ***me_a*** alcanza un valor alto. La señal interna ***dirbin_b*** contiene la dirección del registro a acceder a través del puerto ***A*** y cuyo valor lo toma de ***direccion_b*** cuando ***me_b*** alcanza un valor alto.

D.3.3.-Inferencia difusa

La inferencia difusa se realiza en la unidad aritmético-lógica, la Fig. D.4 muestra el diagrama de señales durante el cálculo de un Mínimo y un Máximo.

Las señales ***a*** y ***b*** son los valores sobre los cuales se realiza la operación; ***ff*** y ***sh*** se utilizan durante la realización de operaciones de precisión múltiple; ***func*** es la operación que debe realizar la unidad aritmético-lógica; y el resultado de la operación es devuelto en ***salida***. En la Fig. D.4 se puede observar la ejecución de las microinstrucciones ***min*** (mínimo) y ***max*** (máximo). Durante la ejecución de la primera microinstrucción ***func*** toma el valor 010 con lo cual realiza la operación ***min*** entre 235 y 63 (***a*** y ***b***), devolviendo en ***salida*** el valor 63. Luego, durante la segunda microinstrucción, ***func*** toma el valor 100 con lo cual realiza la operación

max entre 63 y 0 (***a*** y ***b***), devolviendo el valor 63 en la señal salida.

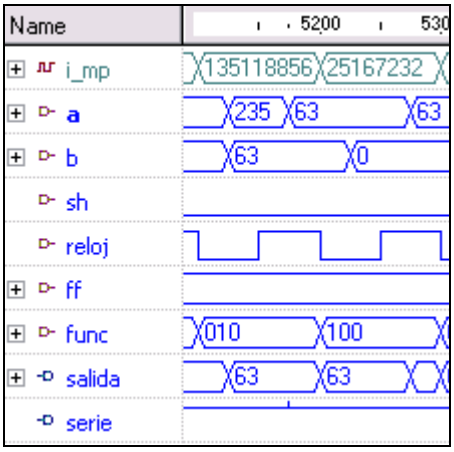


Fig. D.4 – Señales de la ALU durante el cálculo de un Mínimo y Máximo

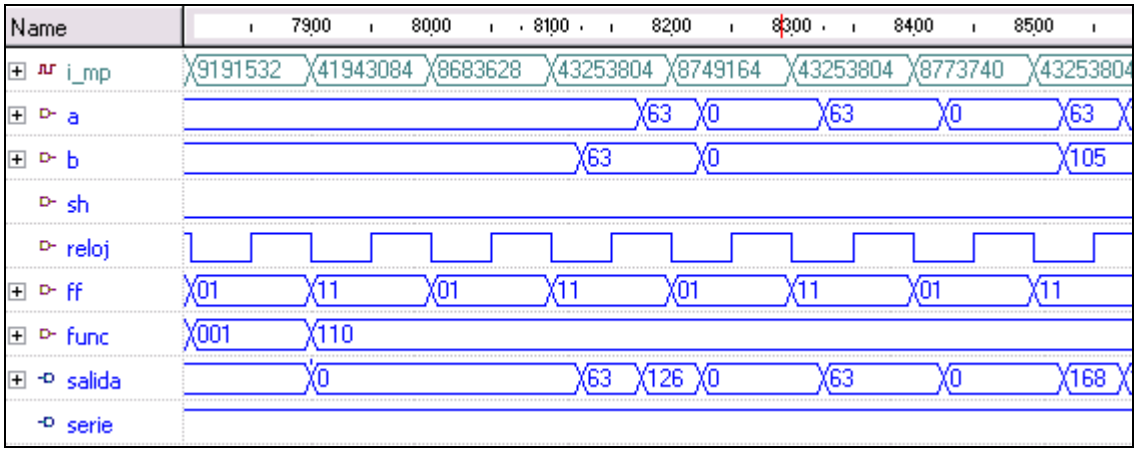


Fig. D.5 – Señales de la ALU en la sumatoria Wi

En la Fig. D.5 se puede ver el funcionamiento de la ALU durante el comienzo de la sumatoria de los W_i. Obsérvese que el primer valor que toma ***func*** es 001 con lo cual inicializa la UAL, luego toma el valor 110 con lo cual realiza la operación *x+y+cent*.

D.3.4.-Defusificación

Durante la defusificación es necesario acceder a las tablas de memoria, en este caso a las que representan a las funciones de decodificación. La diferencia con las de inferencia es que la dirección de desplazamiento dentro de la tabla es obtenida de los valores almacenados en los registros y obtenidos durante la inferencia.

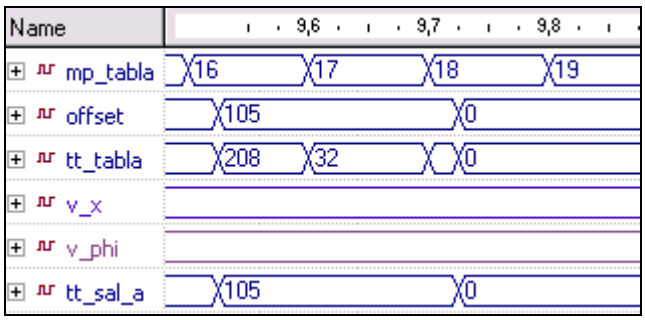


Fig. D.6 – Señales de acceso a las tablas de las Funciones de Defusificación

En la Fig. D.6 se puede observar las señales correspondientes al acceso a las tablas de defusificación, obtenidas durante la simulación. La señal ***offset*** toma su valor de la señal proveniente del banco de registros ***tt_sal_a***. También cabe destacar que la defusificación devuelve un resultado de múltiple precisión, por lo tanto requiere de la lectura de dos tablas para obtener la parte más significativa y la menos significativa del resultado, respectivamente. En la gráfica se puede observar, como se accede a las tablas 16 y 17 a través de ***mp_tabla*** con el mismo valor de desplazamiento ***offset*** (105), obteniendo en ***tt_tabla*** los resultados de ambas lecturas, 208 y 32. Los valores 208 y 32 representan la parte baja y la parte alta del resultado, respectivamente.

En la Fig. D.7 se puede ver el diagrama obtenido durante la simulación de las principales señales que intervienen en el cálculo de la división.

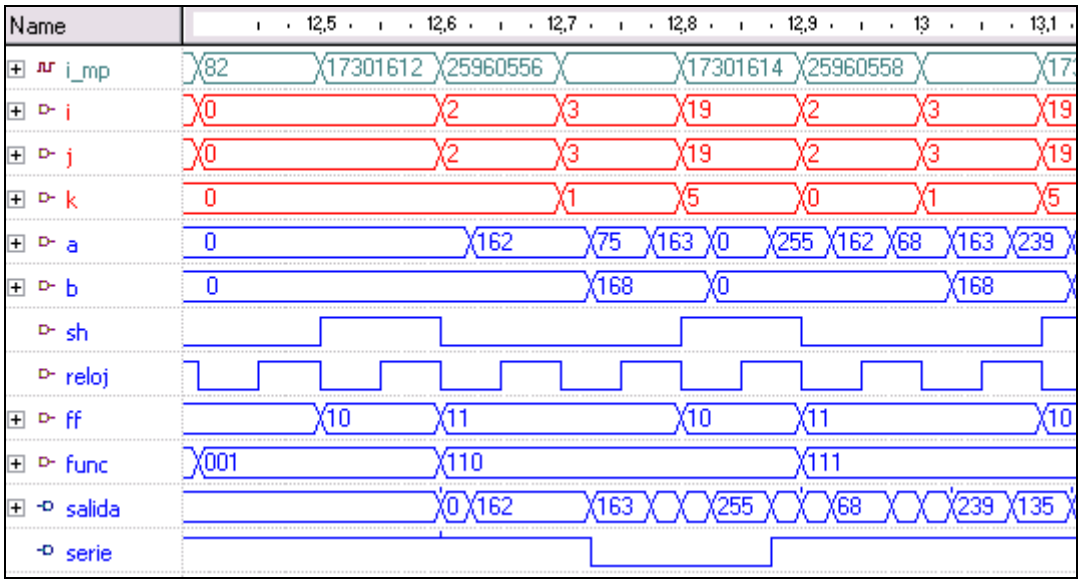


Fig. D.7 – Diagrama de señales durante el cálculo de la división

Las señales ***i***, ***j*** y ***k*** son las direcciones de los registros que intervienen en las operaciones, donde ***i*** corresponde a la dirección del registro de escritura y las otras dos a la de los operandos. Las señales ***a*** y ***b*** son los valores que intervienen en la operación y que se obtienen del banco de registros. Las señales ***sh*** y ***ff*** se utilizan para realizar operaciones de múltiple precisión. La señal ***func*** determina que operación se debe realizar. La división es una operación de múltiple precisión, ya que los valores del numerador y del denominador están contenidos en tres registros. Para llevar cabo la división primero es necesario inicializar la ALU, esto de logra mediante la asignación del valor 001 a ***func*** y de cero a ***i***, ***j*** y ***k***. Luego se realiza la operación x±y+cent sobre los tres registros que contienen los valores mediante la asignación del valor 110 a ***func*** y a continuación toma el valor 111 para realizar la operación 2•x+xent±y+cent, esta secuencia se repite continuamente hasta completar la cantidad de bits requeridos.

D.4 – SIMULACIÓN DE LA ARQUITECTURA ACTIVA

Para realizar la simulación del controlador para arquitectura activa es necesario incluir en el proyecto del Active HDL los archivos VHDL obtenidos. En éste caso la aplicación generó cinco archivos VHDL que se describen en la Tabla D.2.

Archivo VHDL	Descripción
camion.vhd	Módulo principal del controlador
mf_x.vhd	Módulo para el cálculo de las microfunciones de la variable x
mf_phi.vhd	Módulo para el cálculo de las microfunciones de la variable phi
Multiplicador8x13.vhd	Módulo que implementa el circuito de multiplicación
Divisor16x11x8.vhd	Módulo que implementa el circuito de división

Tabla D.2 – Archivos VHDL para la arquitectura activa

El número de archivos que genera la aplicación para la arquitectura activa depende del controlador diseñado, por ejemplo, la aplicación generará tantos archivos para el cálculo de microfunciones como variables de entrada tenga el controlador, y generara tantos archivos dedicados al cálculo de la división o multiplicación como sean requeridos por el controlador. La diferencia que existe

entre los distintos circuitos de división y multiplicación está en el ancho de la arquitectura sobre la que se realiza el cálculo. El módulo TestBench dedicado a la estimulación de las señales del controlador durante la simulación, es automáticamente generado por la aplicación. A continuación se muestra dicho código VHDL:

```
001  entity flc_tb is
002  end flc_tb;
003
004  architecture TB_ARCHITECTURE of flc_tb is
005            component flc
006            port(
007                        v_x : in std_logic_vector(7 downto 0);
008                        v_phi : in std_logic_vector(7 downto 0);
009                        ctrl : in std_logic_vector(10 downto 0);
010                        reset : in std_logic;
011                        clk : in std_logic;
012                     theta : out std_logic_vector(7 downto 0) );
013  end component;
014  signal v_x : std_logic_vector(7 downto 0);
015  signal v_phi : std_logic_vector(7 downto 0);
016  signal ctrl : std_logic_vector(10 downto 0);
017  signal reset : std_logic;
018  signal clk : std_logic;
019  signal theta : std_logic_vector(7 downto 0);
020  begin
021
022  STIMUL: process
023  begin
024    v_x          <= "01100100";
025    v_phi        <= "01100100";
026    ctrl    <= "00000000000";
027    reset   <= '0';
028    wait for 25 ns;
029    reset   <= '1'; wait for 50 ns;
030    reset   <= '0'; wait for 50 ns;
031
032    -- fusificación
033    ctrl    <= "00000010000";   wait for 100 ns;
034    ctrl    <= "11000010000";   wait for 100 ns;
035
036    -- inferencia
037    ctrl    <= "00000000010";   wait for 100 ns;
038    ctrl    <= "10000000010";   wait for 100 ns;
039    ctrl    <= "01000000010";   wait for 100 ns;
040    ctrl    <= "11000000010";   wait for 100 ns;
041
042    -- acumulación de valores para la defusificación y división
043    ctrl    <= "00000101101";   wait for 100 ns;
044    ctrl    <= "00001101101";   wait for 100 ns;
045    ctrl    <= "00010101101";   wait for 100 ns;
046    ctrl    <= "00011101101";   wait for 100 ns;
047    ctrl    <= "00100101101";   wait for 100 ns;
048    ctrl    <= "00101101101";   wait for 100 ns;
049    ctrl    <= "00110101101";   wait for 100 ns;
050
051    -- reset y registro de las variables de entrada y salida
```

```
052    ctrl    <= "00000000000";   wait for 25 ns;
053    reset   <= '1';
054    wait for 50 ns;
055    reset   <= '0';
056    wait for 50 ns;
057
058    wait for 50 ns;
059    report "Testeo de FLC completado";
060    wait;
061    end process;
062
063  UUT : flc port map (v_x => v_x,
064                      v_phi => v_phi,
065                      ctrl => ctrl,
066                      reset => reset,
067                      clk => clk,
068                      theta => theta );
069  end TB_ARCHITECTURE;
070
071  configuration TESTBENCH_FOR_flc of flc_tb is
072          for TB_ARCHITECTURE
073              for UUT : flc  use entity work.flc(flc_par); end for;
074          end for;
075  end TESTBENCH_FOR_flc;
```

Al igual que en la arquitectura pasiva el módulo TestBench se compone de dos bloques, el bloque ***UUT*** mediante el cual se mapean las señales del controlador y el bloque ***STIMUL*** que es el proceso mediante el cual se estimulan las señales. Las señales utilizadas en el módulo para mapear las señales del controlador son las siguientes:

- ***v_x*** para el valor de la variable de entrada ***x***
- ***v_phi*** para el valor de la variable de entrada ***phi***
- ***ctrl*** es la señal de control que determina que partes del circuito se activan (recibe las instrucciones del microprograma)
- ***reset*** inicializa el circuito, genera las salidas y almacena las entradas.
- ***clk*** es la señal de reloj
- ***theta*** es la señal de salida correspondiente a la variable ***theta***

El proceso ***STIMUL*** es el encargado de estimular las señales del controlador, mediante los siguientes pasos:

1º Paso: Consiste en inicializar el circuito mediante la ejecución de las instrucciones de la línea 24 a la 28. Con las instrucciones de la línea 24 a la 27 realiza la asignación de valores a las señales de entrada del controlador, asignándole el valor 100 a las variables de entrada ***x*** y ***phi***, y poniendo a cero las señales ***ctrl*** y ***reset.*** Luego de 25 ns, le asigna el valor 1 a la señal de ***reset***, produciendo así la inicialización del circuito y obligando a que los valores de las variables de entrada sean almacenados para su posterior uso en los cálculos. Luego de 50 ns, la señal de ***reset*** es restablecida al valor 0 y espera otros 50 ns. Una vez realizada estas operaciones el circuito queda en condiciones de recibir las señales de control apropiadas para realizar los cálculos. La ejecución del microprograma consiste en la asignar una microinstrucción a la señal de control (***ctrl***) cada 100 ns, de esta forma se logra realizar las operaciones de cálculo de los pasos siguientes.

2º Paso: Realiza la fusificación ejecutando las líneas 33 y 34.

3º Paso: Realiza la inferencia ejecutando las instrucciones de la línea 37 a la 40.

4º Paso: Acumula los valores para la defusificación y división, mediante la ejecución de las instrucciones de la línea 43 a la 49.

5º Paso: Almacena el resultado obtenido en la variable de salida y prepara el controlador para realizar los cálculos nuevamente sobre los valores de entrada, que pueden haber cambiado o no (instrucciones de la línea 52 a la 56). Primero se pone a cero la señal de control ***ctrl,*** luego de 25 ns se le asigna el valor 1 a la señal ***reset***, logrando de esta manera, la inicialización del circuito, que los valores de entrada sean almacenados y que los resultados del cálculo anterior sean puestos en las salidas. Luego de 50 ns, la señal de reset es restablecida al valor cero y espera otros 50 ns. De esta forma el circuito está en condiciones de comenzar a realizar nuevamente los cálculos, repitiendo los pasos 2, 3, 4 y 5.

Para realizar la simulación de la arquitectura activa también es necesario definir en el Active HDL un archivo Waveform llamado **Camion.awf**. La Fig. D.8 muestra el diagrama de señales de la arquitectura activa. Obsérvese que la primera columna del gráfico (Name) contiene los nombres de las señales que se están analizando, las primeras seis señales corresponden a las señales de entrada y salida del

controlador difuso, el resto son señales internas.

Observando la representación gráfica de las señales internas, se puede analizar el comportamiento de las señales de los distintos bloques del circuito o ver cuando son activados estos bloques. El Waveform también es utilizado para estimular la señal ***clk*** del controlador conectándola a una señal de reloj con un ciclo de 100 ns (en simulación).

En la Fig D.8 se muestra el diagrama de señales de la arquitectura activa, donde se puede ver como la señal de control ***ctrl*** (con la microinstrucción del microprograma) es descompuesta en las siguientes señales internas:

- ***mf_df*** determina si ***MF_x*** y ***MF_phi*** deben realizar una fusificación o una defusificación.
- ***weSOP0*** determina si se debe escribir en el registro ***SOP***.
- ***weD0*** determina si se debe escribir en el registro ***D***.
- ***weN0*** determina si se debe escribir en el registro ***N***.
- ***weREG*** determina si se debe escribir en los registros de los bloques de memoria ***REG_*x y ***REG_*phi.
- ***ADF_IND*** selecciona la dirección para el registro ***SOP***, por medio de ***IND*** durante la defusificación y utilizando a ***FAM_theta*** durante la inferencia.
- ***IND*** contiene la dirección del banco de registros ***SOP*** durante la etapa de defusificación.
- ***dirx*** y ***dirphi*** son las direcciones para acceder a los bloques de memoria ***REG_*x y ***REG_*phi.

Las señales ***SOPtheta0*** a ***SOPtheta6*** son las señales correspondientes a los registros del bloque ***SOP***. Las señales ***mREGx0***, ***mREGx1***, ***mIDFx0*** y ***mIDFx1*** corresponden al bloque de memoria ***REG_x***. Las señales ***mREGphi0***, ***mREGphi1***, ***mIDFphi0*** y ***mIDFphi1*** corresponden al bloque de memoria ***REG_phi***. La señal ***nd0*** corresponde al registro ***N***, la señal ***dd0*** corresponde al registro ***D*** y la señal ***F0*** corresponde al resultado de la división.

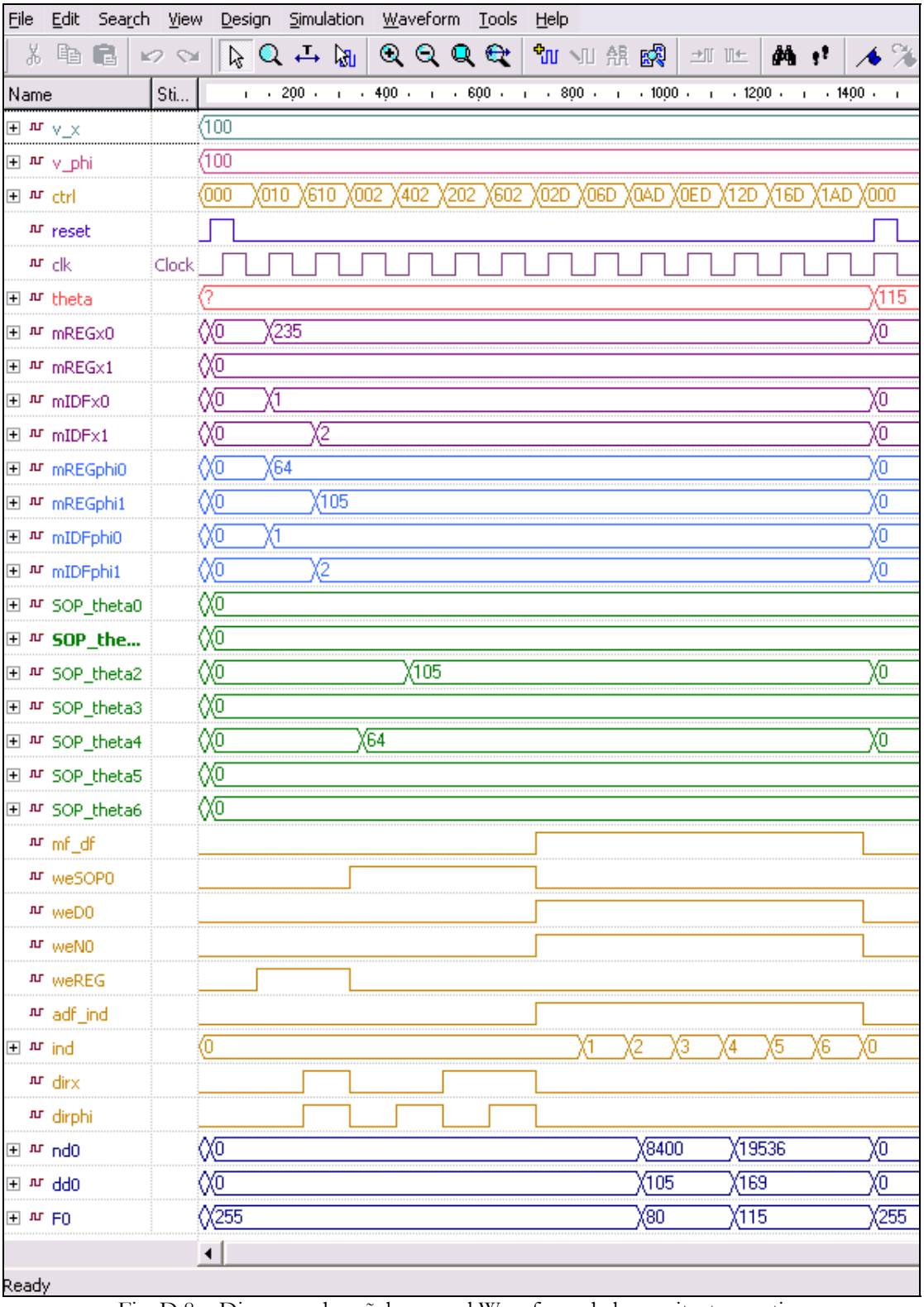


Fig. D.8 – Diagrama de señales con el Waveform de la arquitectura activa

D.4.1.-Fusificación

El bloque ***MF_x*** del circuito es el encargado de realizar la fusificación de la

variable ***x*** (Fig. D.9). La primer microinstrucción para realizar la fusificación es recibida a los 125 ns, la señal ***x*** con el valor de entrada de la variable esta en 100, las señales ***indSOP0*** y ***salSOP0*** no son utilizadas durante la fusificación. La señal ***mf_df*** se mantiene en un valor bajo para activar el circuito para la fusificación.

En este ejemplo para realizar la fusificación son necesarias dos microinstrucciones, durante la primera microinstrucción, la señal ***dirx*** que corresponde a la dirección del segmento que se está seleccionando, en este caso con un valor bajo permite seleccionar el primer segmento y durante la segunda microinstrucción esta en un valor alto para seleccionar el próximo segmento.

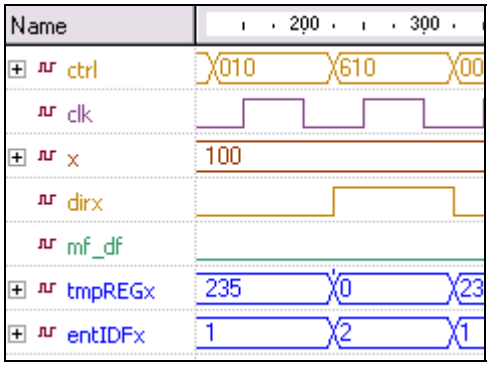


Fig. D.9 – Señales de entrada y salida del bloque MF_X durante al fusificación

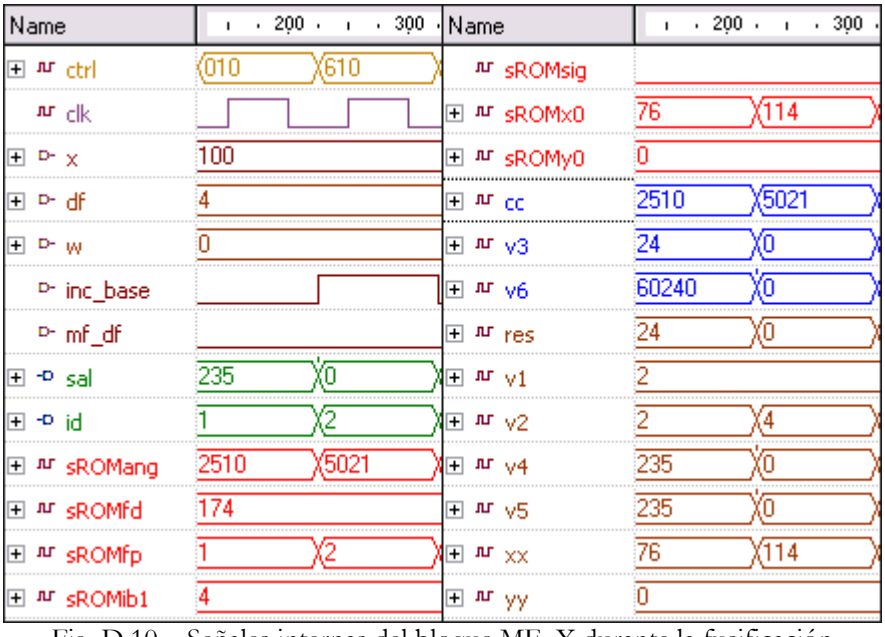


Fig. D.10 – Señales internas del bloque MF_X durante la fusificación

La señal ***tmpREGx*** corresponde al resultado de la fusificación y la señal ***entIDFx*** corresponde al identificador de la función de pertenencia

activada. La Fig. D.9 muestra el diagrama de señales de entrada y salida del bloque ***MF_X*** con los resultados obtenidos para el cálculo de los dos segmentos.

En la Fig. D.10 se puede ver el estado de las señales dentro del bloque ***MF_x*** durante la fusificación. La señal ***x*** corresponde al valor de la variable de entrada, ***df*** y ***w*** no son utilizadas durante la fusificación. La señal ***inc_base*** selecciona el segmento sobre el que se realiza la fusificación. En las señales ***sROMang***, ***sROMfd***, ***sROMfp***, ***sROMib1***, ***sROMsig***, ***sROMx0*** y ***sROMy0*** se pueden ver los valores leídos de las distintas memorias ROM que componen el circuito, para ambos segmentos. La señal ***cc*** corresponde al valor de la pendiente de la función leído de la memoria ROM a través de la señal ***sROMang***, ***xx*** es valor leído de la ROM a través de la señal ***sROMx0*** que corresponde al valor inicial de la función, ***res*** es el resultado de realizar la operación ***x – xx*** y que en la fusificación es pasado a la señal ***v3***. La señal ***v6*** es el resultado del bloque ***multiplicador*** que realiza la multiplicación entre las señales ***cc*** y ***v3***. La señal ***v4*** es el resultado de eliminar la parte decimal de la señal ***v6***, la señal ***v5*** es el resultado de realiza la operación ***v4*** $\pm$ ***yy*** y para la fusificación su valor es pasado a la señal de salida ***sal***.

D.4.2.-Inferencia difusa

La Fig. D.11 muestra el diagrama de las señales obtenidas durante la simulación de las 4 microinstrucciones encargadas de realizar la inferencia difusa en la arquitectura activa. La etapa de inferencia comienza a los 325 ns, inmediatamente la señal ***weSOP0*** pasa a tener un valor alto habilitando de esta manera la escritura de los registros del bloque ***SOP.*** Las señales ***dirx*** y ***dirphi*** indican la dirección de los segmentos cuyos valores se quieren leer de los registros del bloque ***REG,*** correspondientes a las variables ***x*** y ***phi*** respectivamente. Un valor bajo indica que se quiere leer el primer segmento y un valor alto que se quiere leer el siguiente segmento. La lectura de los segmentos se hace realizando todas las combinaciones de ambas señales (00, 01, 10, 11). La señal ***if_x*** corresponde al registro en el bloque ***REG*** con la dirección de la función de pertenencia activada para variable ***x*** y el segmento indicado por ***dirx***. La señal ***if_phi*** corresponde al registro en el bloque

REG con la dirección de la función de pertenencia activada para variable ***phi*** y el segmento indicado por ***dirphi.***

Durante la ejecución de cada microinstrucción con las señales ***if_x*** e ***if_phi*** se accede al bloque de memoria FAM, obteniendo de esta manera el identificador de la función de decodificación correspondiente al valor de inferencia calculado y utilizado para indicar en que en que registro del bloque ***SOP*** se debe leer y escribir, este identificador es representado por la señal ***indSOP0***. La señal ***salREGx*** corresponde al registro en el bloque ***REG*** con el valor obtenido durante la fusificación para variable ***x*** y el segmento indicado por ***dirx.*** La señal ***salREGphi*** corresponde al registro en el bloque ***REG*** con el valor obtenido durante la fusificación para variable ***phi*** y el segmento indicado por ***dirphi.***

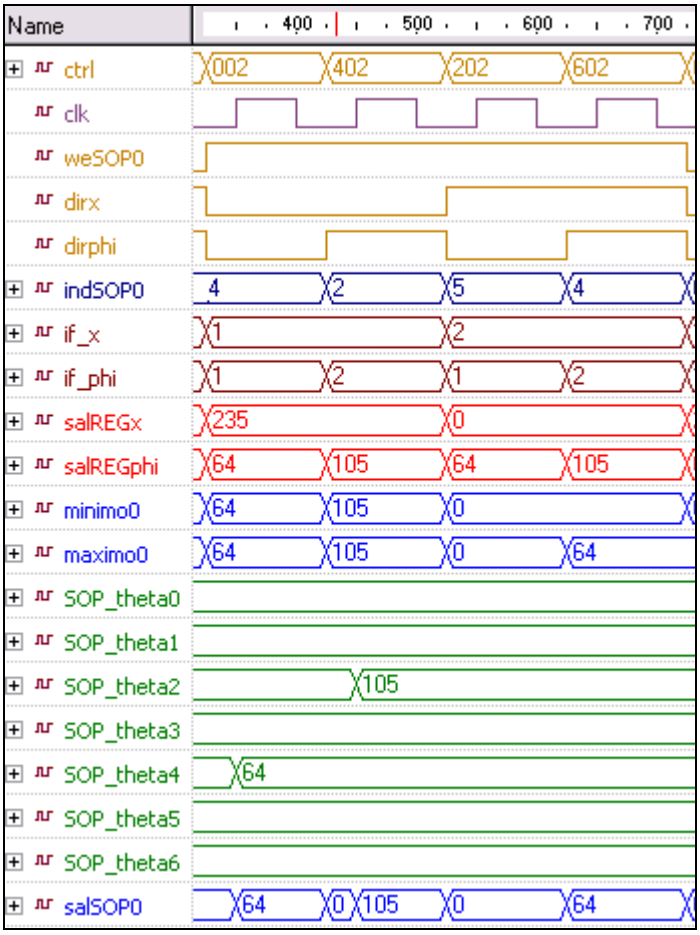


Fig. D.11 – Diagrama de señales durante la inferencia de la arquitectura activa

La señal ***minimo0*** es la salida del bloque ***MIN*** encargado de realizar la operación mínimo entre ***salREGx*** y ***salREGphi***. La señal ***maximo0*** es la salida del bloque ***MAX*** encargado de realizar la operación máximo entre ***minimo0*** y ***salSOP0***. Las señales ***SOP_theta0*** a ***SOP_theta6*** corresponden a los registros del bloque ***SOP***, ***salSOP0*** es la señal de salida del bloque SOP correspondiente al registro que esta siendo accedido.

Durante la primer instrucción se selecciona el primer segmento de ambas variables mediante ***if_x*** y ***if_phi***, obteniendo el valor 4 en la señal ***indSOP0*** correspondiente al índice del registro que se debe escribir en el bloque ***SOP***, una vez calculado los valores ***maximo0*** y ***minimo0*** el valor de ***maximo0*** es almacenado en el registro ***SOP_theta4***. La operación se repite para la segunda instrucción con la diferencia que se seleccionan el primer segmento de ***x*** y el segundo segmento de ***phi,*** y el valor de ***maximo0*** es almacenado en el registro ***SOP_theta2***. Para las otras dos instrucciones la operación es igual variando los segmentos seleccionados.

D.4.3.-Defusificación

Para llevar a cabo la defusificación (Fig. D.12), el cálculo de los numeradores, el cálculo de los denominadores y la división son necesarias siete microinstrucciones. La Fig. D.12 muestra el diagrama de señales de la simulación del bloque ***MF_x***, durante la ejecución de las instrucciones de defusificación. Durante la defusificación éste bloque es usado para realizar el cálculo para ambas variables de entrada (***x*** y ***phi***).

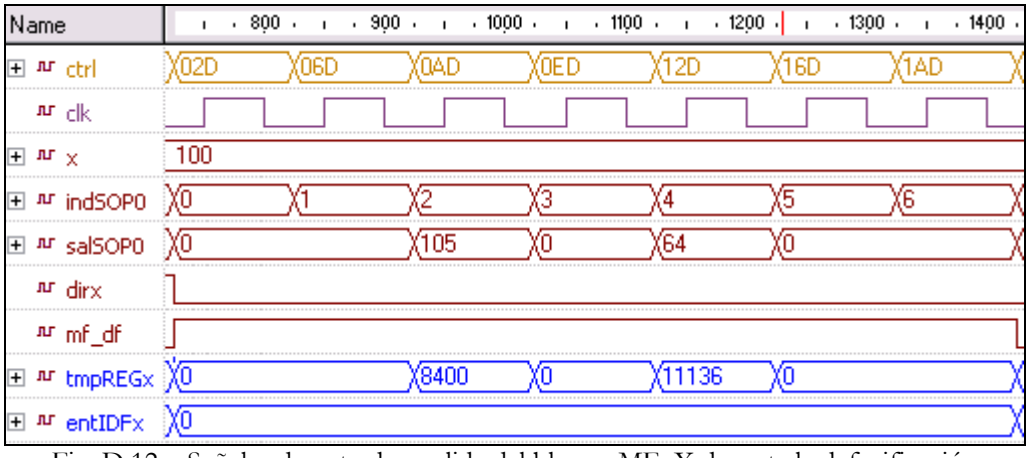


Fig. D.12 – Señales de entrada y salida del bloque MF_X durante la defusificación

Como se puede ver el valor de la variable ***x*** se mantiene en 100, aunque en este punto ya no es requerido. La señal ***mf_df*** pasa a tener un valor alto con lo cual el circuito esta listo para realizar la defusificación. La señal ***indSOP0*** contiene la dirección de la función de decodificación que se va a evaluar y la señal ***salSOP0*** contiene el valor obtenido durante la inferencia para dicha función. El valor de ***indSOP0*** es asignado por la microinstrucción. La señal ***dirx*** no es necesaria durante la defusificación y la salida ***entIDFx*** devuelve siempre cero. La multiplicación del valor de inferencia ***salSOP0*** y la señal ***tmpREGx*** devuelve el peso de la función.

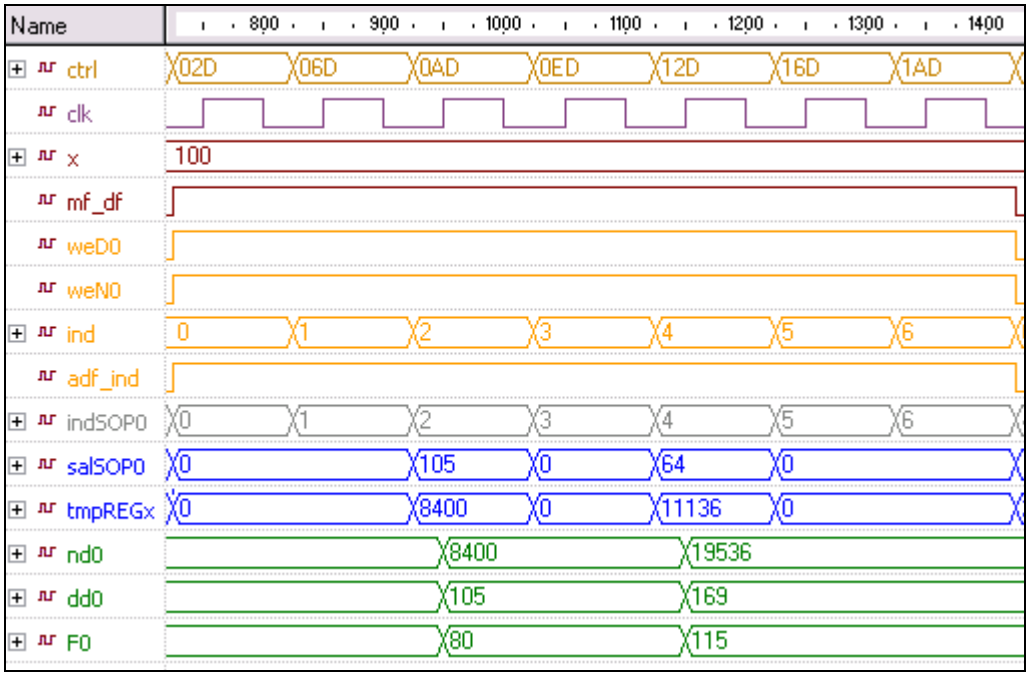


Fig. D.13 – Señales durante la defusificación de la arquitectura activa

La Fig. D.13 muestra el diagrama de señales durante la defusificación de la arquitectura activa. Las señales de ***weD0*** y ***weN0*** pasan a tener un valor alto habilitando la escritura de los registros del denominador ***dd0*** y del numerador ***nd0*** respectivamente. La señal ***ind*** contiene la identificación de la función de decodificación que se esta evaluando y es asignada por la microinstrucción. La señal ***adf_ind*** se pone en valor alto habilitando el uso de la identificación contenida en ***ind*** para acceder al registro correspondiente en el bloque ***SOP,*** mediante la señal ***indSOP0***. La señal ***salSOP0*** contiene el valor del registro leído del bloque ***SOP*** y seleccionado a través de ***indSOP0.*** La señal ***tmpREGx*** contiene el valor de la

defusificación calculado en el bloque ***MF_x*** para la función indicada por ***indSOP0.*** Los valores ***salSOP0*** obtenidos durante la ejecución de cada microinstrucción son acumulados en el registro del denominador ***dd0*** y los valores ***tmpREGx*** obtenidos son acumulados en el registro del numerador ***nd0. F0*** es el resultado de la división entre ***nd0*** y ***dd0.***

REFERENCIAS

[Aco00] "*Multiplicadores paralelos: estado del arte y análisis de su materialización en FPGA*". Iberchip'2000. Sao Paulo (Brasil). Marzo 2000. Acosta N., Todorovich E., Collado C. y Larsen K. Pp: 158-168.

[Aco00b] "*Motores de Inferencia para Controladores Difusos: análisis de materializaciones hardware*". Congreso Internacional en Ingeniería Informática (ICIE). UBA, Bs. As. 26-28 abril 2000. Acosta N. & Curti H.

[Aco01] "*Controladores Difusos de altas prestaciones: sistema para la generación automática*", Acosta N.. WICC'2001 (Workshop de Investigadores en Computación). Mayo'2001. San Luis (Arg).

[Aco01b] "*Diferentes arquitecturas aplicadas a la implementación de un Controlador Difuso para el péndulo invertido*", Acosta N. & Simonelli D. CACIC'2001 (Congreso Argentino de Ciencias de la Computación). Octubre'2001. El Calafate (Arg).

[Aco01c] "*Materialización de Controladores Difusos Activos*", Acosta N. CACIC'2001 (Congreso Argentino de Ciencias de la Computación). Octubre'2001. El Calafate (Arg).

[Aco02] "*Custom Architectures for Fuzzy and Neural Networks Controllers*", Acosta Nelson & Tosini Marcelo. Journal of Computer Science & Technology nro 7, año 2002.

[Aco03] "*Adaptación de la Arquitectura de un Microcontrolador Estándar orientado a FPGA para el soporte de Algoritmos Difusos*", N. Acosta, M. Vázquez, E. Todorovich, E. I. Boemo y D. Simonelli. III Jornadas Sobre Computación Reconfigurable y Aplicaciones; Madrid (España). Sept/2003.

[Aco03b] "*Estacionamiento Automático de un Vehículo Autoguiado usando Lógica Difusa*", N. Acosta, C. Aciti y M. Berlusconi. CACIC 2003 (Congreso Argentino de Ciencias de la Computación), La Plata, Octubre 2003.

[Aco96] "*Metodología de Síntesis de Controladores Difusos*", N. Acosta, G. Bioul, J-P Deschamps, G. Rigotti, G. Sutter, M. Tosini. Reporte Técnico del ISISTAN, Diciembre de 1996.

[Aco97] "*Customized fuzzy logic controller generator*", N. Acosta, J-P Deschamps y G. Sutter. International Congress on Manufacturing Systems: Manufacturing, Management and control (MIM'97), organizado por la IFAC (International Federation of Automatic Control), Vienna, Austria. Feb'97.

[Aco97b] "*Automatic program generator for customized fuzzy logic controllers*", N. Acosta, J-P Deschamps y G. Sutter. International Symposium on Algorithms and Architectures for real-time control (AART'97), organizado por la IFAC (International Federation of Automatic Control), Vilamoura, Portugal. Abril'97.

[Aco97c] "*Herramientas para la materialización de controladores difusos micro-programados*", N. Acosta, J-P Deschamps y G. Sutter. IBERCHIP III, México DF, México. Feb'97.

[Aco98] "*Optimized active rule fuzzy logic custom controller*", N. Acosta, J-P Deschamps y J. Garrido. International Symposium on Engineering of Intelligent Systems (EIS'98), Tennerife, España. Feb'98.

[Aco98b] "*Segment representation for membership functions*", N. Acosta, J. Garrido y J-P Deschamps. International Symposium on Engineering of Intelligent Systems (EIS'98), Tennerife, España. Feb'98.

[Aco98c] "*Implementación de controladores difusos sobre FPGAs*", N. Acosta, J-P Deschamps y E. Todorovich. XXVI Congreso Argentino de Control Automático, organizado por la Asociación Argentina de Control Automático, Buenos Aires, Argentina. Ago'98.

[Aco98d] "*Diseño de sistemas digitales específicos basados en hardware reconfigurable*", N. Acosta y E. Todorovich. IBERCHIP IV, Mar del Plata, Argentina, marzo de 1998.

[Aco99] "*Prototipo de simulador HDL en un entorno de codiseño*", N. Acosta y H. Curti. Iberchip V, Lima, Perú. Marzo de 1999.

[Aco99b] "*A high – level synthesis tool for generating fuzzy logic controllers*", N. Acosta y E. Todorovich. Iberchip V, Lima, Perú. Marzo de 1999.

[AlK03] "*An effcient data-driven fuzzy approach to the motion planning problem of a mobile robot*", Mohannad Al-Khatib, y Jean J. Saade. Fuzzy Sets and Systems 134 (2003) PP: 65–82.

[ALL01] "*Astable motion control system for manipulators via fuzzy self-tuning*", Miguel A. Llama, Rafael Kelly, y Victor Santibañez. Fuzzy Sets and Systems 124 (2001) 133-154.

[Aptro] http://www.aptronix.com. Aptronix FuzzyNet, 408-428-1883 Data USR V.32bis, Voice 408-428-1888, FAX 408-428-1884.

[Archi] Archimedes Software Inc., 303 Parkplace Center, Suite G125, Kirkland, WA 98033, Tel: (206) 822-6300, Fax: (206) 822-8632.

[Asc95] "*Designing for parallel fuzzy computing*", G. Ascia, V. Catania, B. Giacalone, M. Russo y L. Vita. IEEE Micro, Vol. 15, pp: 1-11, December 1995.

[Asc96] "*Rule-driven VLSI Fuzzy Processor*", Ascia G., Catania V., Russo M. and Vita L.. IEEE Micro, June, Vol. 16, pp: 62-74. (1996)

[Bar01] "*Una arquitectura distribuida para el control de robots autónomos móviles: un enfoque aplicado a la construcción de la plataforma Quaky-Ant*", Humberto Martínez Barberá. Tésis Doctoral, Depto. de Ingeniería de la Información y las Comunicaciones, Universidad de Murcia, España. Enero, 2001.

[Bar92] "*SPACE: a scalable cellular array architecture*", P. Barrie, P. Cockshott, G. Milne, P. Show, Workshop on Field-Programmable Logic & Applications / Vienna / Aug-Sep'92.

[Bar96] "*Automatic Synthesis of Fuzzy Logic Controllers*", A. Barriga, S. Sánchez-Solano, C.J. Jiménez, D. Gálan and D.R. López. Mathware & Soft Computing, Vol. III, nº 3, pp. 425-434, Septiembre 1996.

[Bar98] "*A Design Methodology for Application Specific Fuzzy Integrated Circuits*", A. Barriga, R. Senhadji, C. J. Jiménez, I. Baturone and S. Sánchez-Solano. IEEE International Conference on Electronics, Circuits and Systems (ICECS'98), Vol. 1, pp. 431-434, Lisboa, September 7 - 10, 1998.

[Bar98b] "*Mixed-Signal design of a Fully Parallel Fuzzy Processor*", I. Baturone, A. Barriga, S. Sánchez-Solano and J. L. Huertas. Electronics Letters, Vol. 34, no. 5, pp. 437-438, March 1998.

[Bat00a] "*Microelectronic Design of Universal Fuzzy Controllers*", I. Baturone and S. Sánchez Solano. X Congreso Español sobre Tecnologías y Lógica Fuzzy (ESTYLF 2000), pp. 247-252, Sevilla, Septiembre 20-22, 2000.

[Bat00b] "*VLSI Design of Universal Approximator Systems*", I. Baturone, G. Hernández and S. Sánchez-Solano. Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU2000), V. 1, pp. 36-43, Madrid, Jul. 2000.

[Bat01] "*VLSI Design of Universal Approximator Neuro-Fuzzy Systems*", I. Baturone, S. Sánchez Solano, A. Barriga, C. J. Jiménez, R. Senhadji-Navarro and D. R. López. XVI Conference on Design of Circuits and Integrated Systems (DCIS2001), Oporto, Nov. 2001.

[Bat97] "*A Highly parallel mixed-signal design of a Fuzzy Processor*", I. Baturone, S. Sánchez Solano and J. L. Huertas. 1997 International Symposium on Nonlinear Theory and its Applications (NOLTA'97), Vol. 1, pp. 149-152, Honolulu, November 29 - December 2, 1997.

[Bat97b] "*Implementation of CMOS Fuzzy Controllers as Mixed-signal Integrated Circuits*", Baturone Y., Sanchez-Solano S., Barriga A. and Huertas J.. IEEE Trans. on Fuzzy Systems, vol. 5, Nro. 1, February, pp: 1-19. (1997)

[Bat98a] "*Optimization of Adaptive Fuzzy Processor Design*", I. Baturone, S. Sánchez Solano, A. Barriga and J. L. Huertas. XIII Conference on Design of Circuits and Integrated Systems (DCIS'98), pp. 316-321, Madrid, Noviembre 17-20. 1998.

[Bat98c] "*Mixed-Signal VLSI Design of Adaptive Fuzzy Systems*", I. Baturone, S. Sánchez-Solano and J. L. Huertas. Seventh IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'98), pp. 25-30, Anchorage - Alaska, May 4-9, 1998.

[Bat98d] "*Towards the IC Implementation of Adaptive Fuzzy Systems*", Iluminada Baturone, Santiago Sánchez-Solano and José L. Huertas. IEICE Trans. on Fundamentals of Electronics. Communications and Computer Sciences, Vol. E81-A, nº 9, pp. 1877-1885, Septiembre. 1998.

[Bat98e] "*A fuzzy system development environment*", Iluminada Baturone, Santiago Sánchez-Solano, A. arriga, C. J. Jiménez y D. R. López. CNM – Sevilla. Proc. 2nd. World Multiconference on Systemics, Cybernetics and Informatics (SCI'98), Orlando (USA). July 1998.

[Bat99a] "*Design issues for the VLSI implementation of Universal Approximator Fuzzy Systems*", I. Baturone, S. Sánchez Solano, A. Barriga and J. L. Huertas. Computational Intelligence and Applications, pp. 150-155, World Scientiÿc and Engineering Society Press, 1999.

[Bat99b] "*CMOS design of a Current-Mode multiplier/divider circuit with applications to Fuzzy Controllers*", I. Baturone, S. Sánchez-Solano and J. L. Huertas. Analog Integrated Circuits and Signal Processing, Kluwer Academic publishers. V. 23, N. 3, pp. 199-210, Jun. 2000.

[Ber01] "*FLEB: a Fuzzy Logic e-Book*", A. Bermúdez, A. Barriga, I. Baturone and S. Sánchez-Solano. European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems (EUNITE-2001) Tenerife, Dec. 2001.

[Bie03] "*Blend of soft computing techniques for effective human–machine interaction in service robotic systems*", Zeungnam Bien, y Won-Kyung Song. Fuzzy Sets and Systems 134 (2003) PP: 5–25.

[Boe99] "*Una herramienta de análisis de retardos de interconexiones en FPGAs*", E. Boemo, N. Acosta y E. Todorovich. Iberchip V, Lima, Perú. Marzo de 1999.

[Boe99b] "*Local versus global interconnections in pipelined arrays: an example of interaction between architecture and technology*", XIV DCIS'99 (Conference on Design of Circuits and Integrated Systems), in Palma de Mallorca (Baleares), Spain, November 16-19, 1999. Boemo E., López-Buedo S., Acosta N. & Todorovich E.. Pp: 181-186.

[Bog01] "*On the Use of Fuzzy Logic to Control Paralleled DC-DC Converters*", Bogdan Tomescu. PhD. Thesis, Faculty of the Virginia Polytechnic Intitute and State University, Blacksburg, Virginia (USA). October 24, 2001.

[Bon03] "*An architecture to coordinate fuzzy behaviors to control an autonomous robot*", Andrea Bonarini, Giovanni Invernizzi, Thomas Halva Labella, y Matteo Matteucci. Fuzzy Sets and Systems 134 (2003) PP: 101–115.

[Bow98] "*Handel-C: language reference manual. V.2.0*", Mattew Bowen. Embedded Solutions Limited, 1998.

[Box95] "*Common processor element packaging for CHAMP*", Box Brian, Nieznanski John, pp:39-44, IEEE FPGA for CCM - Napa – California – 1995.

[Bro95] "*FPGA*", Brown Stephen, Francis Robert, Rose Jonathan & Vranesic Zvonko. Kluwer Academic Publishers, Boston.

[Bru92] "*Using FPGAs to prototype a Self-Timed Computer*", Erik Brunvand. Workshop on Field-Programmable Logic & Applications, Vienna, Aug-Sep'92. Email: brunvand@cs.utah.edu.

[BSh98] "*Supervised fuzzy pattern recognition*", Boris Shukhat. Fuzzy Sets and Systems 100 (1998) 257-265.

[Bue96] "*SPLASH-II: FPGAs in a CCM*", Duncan Buell, Jeffrey Arnold y Walter Kleinfelder, IEEE Computer Society Press, Los Alamitos, California, 1996.

[Car00] "*Fuzzy PID controller: design, performance evolution and stability analysis*", James Carvajal, Guanrong Chen & Hawk Ogmen. Information Sciences 123 (2000). Pp. 249-270.

[Car97] "*Automatic synthesis of analog fuzzy controllers*", R. G. Carvajal, A. Torralba, F. Colodro y L. G. Franquelo, pp:369-373. XII DCIS'97, Sevilla, Noviembre de 1997.

[Car98] "*AFAN – A tool for automatic design of digital and analog neuro-fuzzy controllers*", R. G. Carvajal, M. A. Aguirre Echanova, A. J. Torralba Silgado y L. G. Franquelo, pp:77-90. Artículo publicado en: "*Fuzzy hardware: Architecture & applications*", A. Kandel y G. Langholz. Kluwer Academic Publishers, 1998.

[Cas93] "*Virtual Computing & the Virtual Computer*", Casselman Steven, pp:43-48, IEEE FPGA for CCM, Napa, California - 1993.

[Cas95] "*Hardware object programming on the EVC1: a reconfigurable computer*", Steve Casselman, Michael Thornburg y John Schewel. SPIE FPGA for fast board development and reconfigurable computing, 25-26 Oct, 1995. Philadelphia, Pennsylvania.

[Cas99] "*Integrating Fuzzy Geometric Maps and Topological Maps for Robot Navigation*", Jorge Casos & Alessandro Saffiotti. Proc. of the 3rd. Intenational ICSC Symposium on Soft Computinf (SOCO'99). Genova, Italy. June 1999. Pp: 754-760.

[CCo98] "*Fuzzy adaptive controller design for the joint space control of an agricultural robot*", Christophe Collewet, Guylaine Rault, St~phane Quellec, y Philippe Marchal. Fuzzy Sets and Systems 99 (1998) 1-25.

[Cha01] "*Design of robust fuzzy-model-based controller with sliding mode control for SISO non-linear systems*", Wook Chang, Jin Bae Park, Young Hoon Yoo & Guanrong Chen. Fuzzy Sets and Systems, 2001. Pp. 1-22.

[Chi92] "*Optimization of fuzzy logic inference architecture*", T. C. Chiueh. Computer, pp: 67-71, May 1992.

[Chu96] "On Fuzzy Associative Memory with Multiple-rule Storage Capacity", Chung F. and Lee T.. IEEE Trans. on Fuzzy Systems, vol. 4, Nro. 3, August, pp: 375-384. (1996)

[Clo95] "*VIP: An FPGA-based Processor for Image Processign and Neural Networks*", Jocelyn Cloutier, Eric Cosatto, Steven Pigeon, Francois Boyer & Patrice Simard. Universidad de Montreal, 1995. Email: cloutier@iro.umontreal.ca

[Cos96] "*Hardware design of asynchronous fuzzy controllers*", A. Costa, A. De Gloria y M. Olivieri. IEEE Trans. on Fuzzy Systems, Vol. 4, Nro. 3, pp: 328-338, Feb. 1996.

[Cot00] "*Diseño microelectrónico de controladores para convertidores conmutados continua-continua*", Eduardo Alarcón Cot. Tesis Doctoral, Departamento de Ingeniería Electrónica (DEE). Escuela Técnica Superior de Ingenieros de Telecomunicaciones, Universidad Politécnica de Barcelona. Barcelona (España). Enero 2000.

[CTL00] "*Water bath temperature control with a neural fuzzy inference network*", Chin-Teng Lin, Chia-Feng Juang, y Chung-Ping Li. Fuzzy Sets and Systems 111 (2000) 285-306.

[Cul95] "*TERAMAC: the configurable custom computer*", Culberston W. B., Amerson R., Carter R., Kuekes P. y Snider G.. SPIE – FPGA for Fast Board Development & Reconfigurable Computing. Oct'95, pp:201-209.

[CWL03] "*Construction of fuzzy systems using least-squares method and genetic algorithm*", Cheol W. Lee, Yung C. Shin. Fuzzy Sets and Systems 137 (2003) PP: 297–323.

[DaK99] "*An accurate and cost-effective COG defuzzifier without the multiplier and the divider*", Daijin Kim, y In-Hyun Cho. Fuzzy Sets and Systems 104 (1999) 229-244.

[Dav83] *"Digital Systems with Algorithm Implementation"*, M.Davio, J-P.Deschamps & A.Thayse. Wiley, Chichester, 1983.

[Dee03] "*A study of parameter values for a Mahalanobis Distance fuzzy classifier*", Peter J. Deer, y Peter Eklund. Fuzzy Sets and Systems 137 (2003) PP: 191–213.

[Des94] "*Diseño de circuitos integrados de aplicación específica: ASIC*", Jean-Pierre Deschamps. Editorial Paraninfo, Madrid, España. 1994.

[Des95] "*Metodología para la materialización de algoritmos borrosos: de una especificación de alto nivel a un ASIC*", J-P Deschamps y J. I. Martínez-Torre. ESTYLF'95, pp:250-256, Sept. 1995.

[Des96] "*Algoritmo de Optimización de Funciones Reticulares aplicado al Diseño de Controladores Difusos*", J-P Deschamps y N. Acosta. JAIIO'96, Buenos Aires, Sept. 1996.

[Des97] "*Metodología para la síntesis de controladores difusos*", Deschamps J-P & Bioul Géry. III Workshop Iberchip, CINVSTAV, Mexico D.F., 1997.

[Des98] "*Fuzzy controller synthesis method*", Jean Pierre Deschamps, pp:231-255. Artículo publicado en: "*Fuzzy hardware: Architecture & applications*", A. Kandel y G. Langholz. Kluwer Academic Publishers, 1998.

[Dra00] "*Consider Fuzzy Logic in Test Applications*", William Drago. Test & Measurement World, October 15, 2000. http://www.tmworld.com/articles/2000/1015_fuzzy.htm.

[Dra95] "*MORRPH: a modular & reprogrammable real-time processing hardware*", Drayer T. H., King IV W., Tront J. G. y Conners R. W., pp:11-19, IEEE FPGA for CCM, Napa, California, 1995.

[Ess92a] "*ES2 ECPD10 Library data book*". European Silicon Structures, Zone Industrille 13106 Rousset Cedex, France. 1992.

[Ess92b] "*ES2 ASIC Design Guidelines*". European Silicon Structures, Zone Industrille 13106 Rousset Cedex, France. 1992.

[Ess92c] "*ES2 ECPD10 Engineering Application Notes*". European Silicon Structures, Zone Industrille 13106 Rousset Cedex, France. 1992.

[Faw95] "*Field Programmable Gate Arrays & Reconfigurable Computing*", Bradly K. Fawcet. SPIE - FPGA for Fast Board Development & Reconfigurable Computing. Oct'95, PP:155-166.

[FLa02] "*Optimized fuzzy control of a greenhouse*", F. Lafont ., J.-F. Balmat. Fuzzy Sets and Systems 128 (2002) 47–59.

[FSu03] "*Neuro-fuzzy adaptive control based on dynamic inversion for robotic manipulators*", Fuchun Sun, Zengqi Sun, Lei Li, Han-Xiong Li. Fuzzy Sets and Systems 134 (2003) PP: 117–133.

[Gaj94] "*Specification and design of embedded systems*", D. D. Gajski, F. Vahid, S. Narayan and J. Gong. Prentice-Hall, 1994.

[GaJ99] "*Uncertainty representation for mobile robots: Perception, modeling and navigation in unknown environments*", Jorge Gasos, y Ailson Rosetti. Fuzzy Sets and Systems 107 (1999) 1-24.

[Gal95] "*VHDL package for Description of Fuzzy Logic Controllers*", D. Galán, C. J. Jiménez, A. Barriga and S. Sánchez Solano. European Design Automation Conference (EURO-VHDL'95), pp. 528-533, Brighton – Great Britain, September 18-22, 1995.

[GAr02] "*Hardware implementation ofa pipeline fuzzy controller and software tools*", Gerardo Aranguren, Luis A.L. Nozal, Xabier Basogain, José L. Martín, y Joseba L. Arroyabe. Fuzzy Sets and Systems 128 (2002) 61–79.

[GCr02] "*Pattern characteristics of an evolution between two classes*", Genevieve Cron y Bernard Dubuisson. Fuzzy Sets and Systems 126 (2002) 293–310.

[GNM01] "*Obstacle avoidance for a mobile robot: A neuro-fuzzy approach*", G.N. Marichal , L. Acosta, L. Moreno, J.A. M-endez, J.J. Rodrigo, y M. Sigut. Fuzzy Sets and Systems 124 (2001) 171-179.

[GuG00] "*Longitudinal fuzzy control of a submerged vehicle*", Guy Gavish, Ran Zaslavsky, y Abraham Kandel. Fuzzy Sets and Systems 115 (2000) 305-319.

[Gup93] "*Automating the design of computer systems*", A. P. Gupta, W. P. Birmingham y D. P. Siewiorek. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 12, Nro. 4, pp:473-487. April, 1993.

[Gup97] "*Using a programming language for digital system design*", R.K. Gupta and S. Liao. IEEE Design and Test of Computers, April-June, 1997.

[Hat96] "*A parametric model for using heterogeneous fuzzy data*", R. Hathaway, J. Bezdok & W. Pedrycz. IEEE Trans. on Fuzzy Systems, Vol. 4, Nro. 3. Aug'96.

[Hec94] "*Approximately time-optimal fuzzy control of a two-tank system*", T. Heckenthaler & S. Engell. IEEE Control Systems, Jun'94.

[Hee92] "*CHAMALEON: a workstation of different colour*", Beat Heeb & Cuno Pfister - Email: pfister@inf.ethz.ch, Workshop on Field-Programmable Logic & Applications / Vienna / Aug-Sep'92.

[Hen00] "*Fuzzy route choice model for traffic assignment*", Vincent Henn. Fuzzy Sets and Systems 116 (2000) 77-101.

[Her93] "*A reconfigurable computer for embedded control applications*", Herpel H. J., Wehn N., Gasteier M. y Glesner M., pp:111-120, IEEE FPGA for CCM - Napa – California - 1993.

[Hil96] "*A rule.-based FL controller for a PWM inverter in a stand alone wind energy convension scheme*", R. Hilloowala & A. Sharaf. IEEE Trans. on Industry Applications, Vol. 32, Nro. 1. Jan/Feb'96.

[HitFL] Hitachi Contact Information, web: http://www.hitachi.com

[Hog95] "*ENABLE+ +: a second generation processor*", Högl H, Kugel A, Ludvig J, Männer R, Noffz K y Zoz R., IEEE FPGA for CCM - Napa - California - April 19-21 / 1995.

[Hol97] "*A prototyping environment for fuzzy controllers*", T. Hollstein, A. Kirschbaum y M. Glesner., pp:482-490, Lecture Notes in Computer Science 1304, Ed. Springer, FPL, London, Septiembre de 1997.

[How92] "*The design and implementation of a massively-parallel fuzzy architecture*", N. Howard, R. Taylor y N. Allison, Proc. IEEE Int. Conf. on Fuzzy Systems, San Diego CA, pp: 545-552, Marzo de 1992.

[HPH02] "*Strategy-based decision making of a soccer robot system using a real-time self-organizing fuzzy decision tree*", Han-Pang Huang y Chao-Chiun Liang. Fuzzy Sets and Systems 127 (2002) 49–64.

[Hua03] "*Fuzzy unidirectional force control of constrained robotic manipulators*", L. Huang, S.S. Ge, y T.H. Lee. Fuzzy Sets and Systems 134 (2003) PP: 135–146.

[Hue96] "*Integrated Circuit Implementation of Fuzzy Controllers*", J. L. Huertas, S. Sánchez Solano, I. Baturone and A. Barriga. IEEE Journal of Solid-State Circuits, Vol. 31, nº 7, pp. 1051-1058, Jul. 1996.

[Ike91] "*A fuzzy inference processor with an active-rule-driven architecture*", H. Ikeda, Y. Hiromoto y N. Kisu. Symposium on VLSI Circuits (Oiso, Japón), May 1991.

[Ise93] "*SPYDER: a reconfigurable VLIW processor using FPGAs*", Iseli C. y Sánchez Eduardo, pp:17-24, IEEE FPGA for CCM – Napa – California – April 19-21 / 1993.

[Jag95] "*Fuzzy logic in control*", René Jager. PhD Thesis. Technische Universiteit Delft, 1995.

[Jan95] "*Neuro-fuzzy modeling & control*", J. Jang & C. Sun. Proc. of the IEEE, Vol. 83, Nro. 3, pp: 378-406. March'95.

[JDY01] "*An image retrieval model based on fuzzy triples*", Jae Dong Yang. Fuzzy Sets and Systems 121 (2001) 459-470.

[JHH02] "*Ambiguity distance: an edge evaluation measure using fuzziness of edges*", Joon H. Hana, Tae Y. Kimb. Fuzzy Sets and Systems 126 (2002) Pp: 311–324.

[JIH02] "*Application of fuzzy control to industrial bioprocesses in Japan*", Jun-Ichi Horiuchi, Michimasa Kishimoto. Fuzzy Sets and Systems 128 (2002) 117–124.

[Jim94] "*Digital Implementation of SISC Fuzzy Controllers*", C. J. Jiménez, A. Barriga and S. Sánchez Solano. 3rd. International Conference on Fuzzy Logic, Neural Nets and Soft Computing (IIZUKA'94), pp. 651-652, Iizuka – Japan, August 1-7, 1994.

[Jim95] "*Hardware Implementation of a General Purpose Fuzzy Controller*", C. J. Jiménez, S. Sánchez Solano and A. Barriga. Sixth International Fuzzy Systems Association World Congress (IFSA'95), Vol. 2, pp. 185-188, Sao Paulo - Brazil, July 21-28, 1995.

[JLe03] "*Towards a robust fuzzy clustering*", Jacek Leski. Fuzzy Sets and Systems 137 (2003) PP: 215–233.

[JMa03] "*The car following collision prevention controller based on the fuzzy basis function network*", Jeich Mar, Feng-Jie Lin, Hung-Ta Lin, y Li-Chin Hsu. Fuzzy Sets and Systems 139 (2003) PP: 167–183.

[JYi02] "*A new fuzzy controller for stabilization of parallel-type double inverted pendulum system*", Jianqiang Yi, Naoyoshi Yubazaki, Kaoru Hirota. Fuzzy Sets and Systems 126 (2002) Pp: 105–119.

[Kan93] "*Stability & control of fuzzy dynamic systems via cell-state transition in fuzzy hypercubes*", H. Kang. IEEE Trans. on Fuzzy Systems, Vol. 1, Nro. 4. Nov'93.

[Kan98] "*Fuzzy hardware: Architecture & applications*", A. Kandel y G. Langholz. Kluwer Academic Publishers, 1998.

[Kar00] "*Advanced Control of an Industrial Circulating Fluidized Bed Boiler using Fuzzy Logic*", Erkki Karppanen. Department of Process Engineering, University of Oulu, Oulu, Finland. 2000.

[Kas99] "*Intelligent Control in Multimedia Traffic Policing, Shaping and Congestion Avoidance over Broadband Networks*", Anastasios Kasiolas. MSc. Thesis, Department of Electrical and Computer Engineering, University of Ontario, London, Ontario, April, 1999.

[KCT02] "*Design and real-time implementation of a multivariable gyro-mirror line-of-sight stabilization platform*", K.C. Tana, T.H. Leea, E.F. Khora, D.C. Angb. Fuzzy Sets and Systems 128 (2002) 81–93.

[KHa99] "*Realization of nonlinear and linear PID controls using simplified indirect fuzzy inference method*", Kenichiro Hayashi, Akifumi Otsubo, Shuta Murakami, y Mikio Maed. Fuzzy Sets and Systems 105 (1999) 409-414.

[Kli88] "*Fuzzy sets, uncertainty and information*", G. J. Klir y T. A. Folger. Englewood Cliffs, NJ, Prentice-hall, 1988.

[Klo99] "*Discussing Cluster Shapes of Fuzzy Classifiers*", Andreas Nürnberger, Aljoscha Klose & Rudolf Kruse. Proc. 18th International Conference of the North American Fuzzy Information Processing Society (NAFIPS'99), pp. 546-550, IEEE,
New York, 1999.

[Kos91] "*Neural Networks for signal processing*", B. Kosko. Prentice Hall, Englewood Cliffs, JN, 1991.

[Kos92] "*Neural Networks and Fuzzy Systems: a dynamical systems approach to machine intelligence*", Bart Kosko. Prentice Hall, Englewood Cliffs, NJ, 1992.

[Kos94] "*Function approximation with additive fuzzy systems*", B. Kosko y J. A. Dickerson, capítulo 12 en "Theoretical aspects of fuzzy control" de H. Nguyen, M. Sugeno y R. Yager Eds., NY, Willey, 1994.

[Kov00] "*NonLinear Position Control By Using Multiple Position-Dependent Self-Organizing Fuzzy Logic Controllers*", Z. Kovacic, S. Bogdan & T. Reichenbach. 6th IFAC-Symposium on Robot Control SYROCO ´00, pp.229-233 , Vienna, Austria, 2000..

[Kov99] "*Robust Self-Learning Fuzzy Logic Servo Control with Neural Network-Based Load Compensator*", Z. Kovacic, V. Petik, T. Reichenbach & S. Bogdan. Theory and Practice of Control and Systems, Editor: A. Tornambe, World Scientific, 1999, pp. 658-665

[Lag97] "*FPGA implementation of Fuzzy Controllers*", E. Lago, M. A. Hinojosa, C. J. Jiménez, A. Barriga and S. Sánchez-Solano. XII Conference on Design of Circuits and Integrated Systems (DCIS'97), pp. 715-720, Sevilla, Noviembre 18-21. 1997

[Lag98] "*XFVHDL: a tool for the Synthesis of Fuzzy Logic Controllers*", E. Lago, C. J. Jiménez, D. R. López, S. Sánchez-Solano and A. Barriga. Design Automation and Test in Europe (DATE'98), pp. 102-107, Paris - France, February 23-26, 1998.

[LaK02] "*Adaptive fuzzy order statistics-rational hybrid filters for color image processing*", Lazhar Khriji, y Moncef Gabbouj. Fuzzy Sets and Systems 128 (2002) 35–46.

[Lem97] "*Run time reconfiguration of FPGA for scanning genomic databases*", Eric Lemoine y David Merceron. IEEE FPGA for custom computing machine, 1997.

[Lin93] "*WASMII: a data driven computer on a virtual hardware*", Ling X.P. y Amano H. IEEE FPGA for CCM - Napa - California – 1993, pp:33-42.

[Lin96] "*Integrating fuzzy control of the Dexterous National Taiwan University (NTU) Hand*", L. Lin & H. Huang. IEEE Trans. on Mechatronics, Vol. 1, Nro. 3. Sep'96.

[Lop97] "*Embedded system specification in ADA*", A. López, M. Veiga, P. Sánchez y E. Villar. XII Design of Circuits and Integrated Systems Conference (DCIS'97), Sevilla, España, pp:389-394, Nov. 1997.

[Lop97] "*XFL: a language for the definition of Fuzzy Systems*", D. López, F. J. Moreno, A. Barriga and S. Sánchez Solano. Sixth IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'97), Vol. 3, pp. 1585-1591, Barcelona - Spain, July 1-5, 1997.

[Lop98] "*XFUZZY: a design environment for Fuzzy Systems*", D. R. López, C. J. Jiménez, I. Baturone, A. Barriga and S. Sánchez-Solano. Seventh IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'98), pp. 1060-1065, Anchorage – Alaska, May 4-9, 1998.

[Los03] "*From classic statistical characterization to fuzzy windowing based characterization for the exploratory analysis of miscellaneous time variables: example in the field of car driving studies*", P. Loslever, J.C. Popieul, y P. Simon. Fuzzy Sets and Systems 137 (2003) PP: 271–296.

[Lun94] "*TRT on DecPerle-1. Algoritmo, implementación, test y futuro*", L. Lundheim, L. Moll, P. Boucard & J. Vuillemin. 1994.

[Ma00] "*A new approach for defuzzification*", Ming Ma, Abraham Kandel, y Menahem Friedman. Fuzzy Sets and Systems 111 (2000) 351-356.

[Mam75] "*An experiment in linguisitc synthesis with a fuzzy logic controller*", E. H. Mamdani y S. Assilian. Int. Journal on Man Machines Studies, Vol. 7, Nro. 1, pp: 1-13, 1975.

[Mam93] "*Twenty years of fuzzy control: experiences gained and lessons learnt*", E. H. Mamdani. Proc. Second IEEE Conf. on Fuzzy Systems, Fuzz-IEEE'93, San Francisco, California, pp: 339-344, 1993.

[Mas94] "*Xfuzzy 1.1: A Simulation Environment for Fuzzy Logic Based Control Systems*", Masa Marín Javier. CNM-Sevilla, 1994.

[Mch95] "*The WILDFIRE configurable custom computer*", McHenry John T. y Donaldson Robert L.- SPIE – FPGA for Fast Board Development & Reconfigurable Computing, Oct'95, pp: 190-200, jtmchen@afterlife.ncsc.mil

[Men95] "*Fuzzy Logic Systems for engineering: a tutorial*", Jerry M. Mendel. Proceedings of the IEEE, vol. 83, Nro. 3, pp: 345-376, March 1995.

[MLi01] "*Real-time task scheduling with fuzzy deadlines and processing times*", Marin Litoiu, y Roberto Tadei. Fuzzy Sets and Systems 117 (2001) 35-45.

[Mokom] Coprocessor Fuzzy Logic for PC, el FB-30AT. Omron Corp., International Public Relations Section, Omron Tokyo Bld., 3-4-10 Toranomon, Minato Ward, Tokyo 105. Farzin Mokhtarian. Email: farzin@apollo3.ntt.jp

[Moo96] "*An interactive fuzzy CAD tool*", Jason Moore y Mahmoud Manzoul, IEEE Micro, pp:68-74, April, 1996.

[Mor00] "*Control of a Pneumatic Servosystem using Fuzzy Logic*", Hipolito Moreno Llagostera. Proc. of the 1st. FPNI - PhD Symposium. Hamburg, 2000. Pp. 189-201.

[Mor01] "*XFL3: a new Fuzzy System Specification Language*", F. J. Moreno-Velo, S. Sánchez-Solano, A. Barriga, I. Baturone and D. R. López. 5th WSES /IEEE Multiconference on Circuits, Systems, Communications and Computers (CSCC 2001), pp. 361-366, Rethymnon, Julio 2001.

[Mor01b] "*XFSL: a tool for supervised learning of Fuzzy Systems*", F. J. Moreno Velo, I. Baturone, S. Sánchez Solano and A. Barriga. European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems (EUNITE-2001) Tenerife, Dec. 2001.

[Mor95] "*Architecture of a FPGA-based coprocessor: PAR-I*", Moran Carrera, Juarez Martinez, Alexander Fernandez y Meneses Chaus. IEEE FPGA for CCM, Napa, California, pp:20-29, April - 1995. Depto. Ingeniería Electrónica / Universidad Politécnica de Madrid – moran@die.upm.es

[Mor97] "*XFBPA: a tool for automatic Fuzzy System Learning*", F. J. Moreno, D. R. López, A. Barriga and S. Sánchez Solano. 5th European Congress on Intelligent Techniques and Soft Computing (EUFIT'97), Vol. 2, pp. 1084-1088, Aachen - Germany, September 8-11, 1997.

[MotFL] Fuzzy Logic Development and Generation Environment, Motorola Corp.. Email: fuzzy.info@ae.sps.mot.com

[Muc03] "*A fuzzy temporal rule-based velocity controller for mobile robotics*", M. Mucientes, R. Iglesias, C.V. Regueiro, A. Bugarín y S. Barro. Fuzzy Sets and Systems 134 (2003) PP: 83–99.

[Mun94] "Fuzzy Systems: An Overview", Munakata T. and Jani Y.. Communications of the ACM, vol. 37, pp. 69-76, Mar. 1994.

[Mur98] "*ASIC Design of a Fuzzy Logic Controller*", Valentin Murestan, Xiaojun Wang & John Yan. Proc. of the International Conference on Control and Applications (IASTED), Aug 12-14, 1998. Hawai, USA.

[Nak93] "*Fuzzy inference and fuzzy inference processor*", K. Nakamura, N. Sakashita, Y. Nitta, K. Shimomura y T. Tokuda. IEEE Micro, pp: 37-47, October 1993.

[Ngu89] "*The truck backer-upper: an example of self-learning in neural networks*", Nguyen D. Y Widrow B. Proceedings of International Joint Conference on Neural Networks (IJCNN'89), vol. 2, pp: 357-363, Junio de 1989.

[Nii00] "*Signal control using fuzzy logic*", Jarkko Niittymaki, y Matti Pursula. Fuzzy Sets and Systems 116 (2000) 11-22.

[Num00] "*Analysing Borders Between Parcially Contradicting Fuzzy Classification Rules*", Andreas Nümberger, Aljoscha Klose & Rudolf Kruse. Proc. 19th International Conference of the North American Fuzzy Information Processing Society (NAFIPS 2000), pp. 59-63, IEEE, Atlanta, 2000.

[OdM01] "*Information-theoretic fuzzy approach to data reliability and data mining*", Oded Maimon, Abraham Kandel, Mark Last. Fuzzy Sets and Systems 117 (2001) 183-194.

[Ori99] "*Real-Time Map Building and Navigation for Autonomous Robots in Unknown Environments*", Giuseppe Oriolo, Giovanni Ulivi & Marilena Vendittelli. IEEE Transactions on Systems, Man and Cybernetics, Nro. 17, 5/1999.

[Pap00] "*Pseudo-analysis and its application in railway routing*", Endre Pap, y Katarina Jegdié. Fuzzy Sets and Systems 116 (2000) 103-118.

[Pat99] "*Designing and Development of Intelligent Systems: PID and Fuzzy Logic Control of Teleoperation Systems*", Pattaraphol Batsomboon & Sabri Tosunoglu. 2nd. International conference on Recent Advances in Mechatronics (ICRAM'99). May 24-26, 1999. Istambul, Turkey.

[Paw00] "*Development of a Fuzzy Logic Speed and Steering Control System for an Autonomous Vehicle*", Scott Pawlikowski. University of Cincinnati, Department of Mechanical Engineering. January 10, 1999.

[Per95] "*Lofti A. Zadeh*", IEEE Spectrum, pp: 32-35, June 1995.

[Pir01] "*Development of a Fuzzy Logic Controller for a Rotary Dryer with Seft-Tuning of Scaling Factor*", Leonardo Pirrello, Leena Yliniemi & Kauko Leiviskä. OULU University, Control Engineering Laboratory, Report A17, OULU, Finlandia. ISBN 951-42-6424-X, ISSN 1238-9390. June 2001.

[Pit99] "*Fuzzy Logic Based Congestion Control*", Andreas Pitsillides & Ahmet Sekercioglu. Preliminary Program. The Third IEEE Symposium on Computers and Communications (ISCC'99), Athens, Greece, June 29 - July 2, 1999.

[Rai93] "*Fine grain parallelism on a MIMD machine using FPGAs*", Raimbault F., Lavenier D., Rubini S. y Pottier B.. IEEE FPGA for CCM - Napa - California - April 19-21, 1993, pp:2-8.

[RkM00] "*A self-tuning fuzzy PI controller*", Rajani K. Mudi y Nikhil R. Pal. Fuzzy Sets and Systems 115 (2000) 327-338.

[Rui95] "*A fuzzy controller with an optimized defuzzification algorithm*", Antonio Ruiz, Julio Gutierrez y Felipe Fernández, IEEE Micro, pp: 1-10, December 1995.

[Rui98] "*Fuzzy hardware based on encoded trapezoids*", Antonio Ruiz, Julio Gutierrez y Felipe Fernandez, pp:257-281. Artículo publicado en: "*Fuzzy hardware: Architecture & applications*", A. Kandel y G. Langholz. Kluwer Academic Publishers, 1998.

[Rus98] "*Fuzzy hardware research from historical point of view*", Marco Russo. Capítulo 1 en "Fuzzy hardware: architectures & applications", A. Kandel y G. Langholz. Kluwer Academic Publishers, 1998.

[SAh01] "*Design of a fuzzy servo-controller*", M.S. Ahmed, U.L. Bhatti, F.M. Al-Sunni, y M. El-Shafei. Fuzzy Sets and Systems 124 (2001) 231-247.

[Sal95] "*A multilevel systolic approach for fuzzy inference hardware*", Luis de Salvador Carrasco y Julio Gutierrez Ríos. IEEE Micro, pp:61-71, Oct. 1995.

[Sal98] "*Serial architectures for efficient digital fuzzy hardware processing*", Luis de Salvador Carrasco y Julio Gutierrez Ríos, pp:117-158. Artículo publicado en: "*Fuzzy hardware: Architecture & applications*", A. Kandel y G. Langholz. Kluwer Academic Publishers, 1998.

[San97] "*Design and Application of Digital Fuzzy Controllers*", S. Sánchez Solano, A. Barriga, C. J. Jiménez and J. L. Huertas. Sixth IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'97), Vol. 2, pp. 869-874, Barcelona - Spain, July 1-5, 1997.

[San98] "*Electronic implementation of complex controllers*", Alfredo Sanz y Jorge Falco, pp:391-406. Artículo publicado en: "*Fuzzy hardware: Architecture & applications*", A. Kandel y G. Langholz. Kluwer Academic Publishers, 1998.

[Sar03] "*Fuzzy model predictive control of non-linear processes using genetic algorithms*", Haralambos Sarimveis, George Bafas. Fuzzy Sets and Systems 139 (2003) PP: 59–80.

[Sat95] "*Architectural descriptions for FPGA circuits*", Satnam Singh. IEEE Symp. on FPGA CCM / Napa Valley California / Apr'95 / PP:145-154. Dept. of Computing Science / The University of Glascow / Glascow / G128QQ / UK. EMAIL: satnam@dcs.glascow.ac.uk.

[Sch95] "*Hidden Markov Modeling & Fuzzy Controllers in FPGAs*", Herman Schmit & Don Thomas. IEEE Symp. on FPGA CCM, Napa Valley California, Apr'95, PP:214-220. Dept. of ECE, Carnegie Mellon University, Pittsburg, PA 15213.

[Sch99] "*Qualitative Modeling and Controller Design using Dynamic Fuzzy Systems*", Klaus Schmidt & Volker Krebs. proceedings of the 3rd International Symposium on Mathematical Modelling, Vienna, Austria, February 2-4, 2000..

[Sen00] "*XFLAB: an on-line verification tool for fuzzy controllers*", R. Senhadji, S. Sánchez-Solano, D.R. López and A. Barriga. Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU2000), V. 1, pp. 44-49, Madrid, Jul. 2000.

[Set99] "*A case study on analitical analysis of the inverted pendulum real-time control system*", Danbing Seto & Lui Sha. Technical Report, CMU/SEI-99-TR023. Nov 1999. Pp. 1-42.

[ShT02] "*Direct adaptive fuzzy output tracking control of nonlinear systems*", Shaocheng Tonga y Han-Xiong Lib. Fuzzy Sets and Systems 128 (2002) 107–115.

[SKR00] "*Stable fuzzy logic design of point-to-point control for mechanical systems*", Sylvia Kohn-Rich, y Henryk Flashner. Fuzzy Sets and Systems 110 (2000) 389-411.

[Slu99] "*Control law reconfiguration by implementation of a multiple model by using robust linear parameter varying control*", Remco van der Sluis. Master Thesis, Control & Simulation Division, Faculty of Aerospace Engineering, Delf University of Technology, Neederlands. Nov 1999.

[Son00] "*Takagi-Sugeno Type Fuzzy logic Controller with only 3 RUles for a 4 Dimensional Inverted Pendulum System*", Feijun Song & Samuel M. Smith. IEEE International Conference on System, Man & Cybernetics, pp.3800-3805, 2000.

[Son99a] "*Cell state space based Fuzzy Logic Controller automatic design and optimization for high order systems*", Feujun Song. Phd. Thesis, College of Engineering, Florida Atlantic University, Boca Raton, Florida. Dec. 1999.

[Son99b] "*A fuzzy logic controller design methodology for 4D systems with optimal global performance using enhanced cell state space based best estimated direct search method*", Feijun Song, Samuel Smith & Charbel Rizk. IEEE International Conference on Systems, Man and Cybernetics, Vol VI, pp. 138-143. 1999.

[Sta97] "*Hardware/Software co-design: principles and practice*", J. Staunstrup y W. Wolf. KLUWER ACADEMIC Press, 1997.

[Ste94] "*FPGA Technology*", Stephen M. Trimberger, Dennis McCarty & Telle Whitney. KLUWER ACADEMIC PUB., 1994

[Sug93] "*A fuzzy-logic-based approach to qualitative modeling*", M. Sugeno y T. Yasukawa. IEEE Trans. on Fuzzy Systems, Vol. 1, Nro. 1, pp: 7-31, 1993.

[Sur92] "*Architecture of a fuzzy controller based on Field Programmable Gate Arrays*", H. Surmann, T. Tauber, A. P. Ungering y K. Goser. Workshop on Field Programmable Logic and Applications, Vienna, pp: 1-5, Aug. 1992.

[Sur95] "*Fuzzy rule-based system on general purpose processors*", H. Surmann y A. P. Ungering. IEEE Micro, Vol. 15, pp: 40-48, Aug. 1995.

[Sut97] "*Diseño de un circuito integrado que ejecuta un algoritmo de control difuso*", Gustavo Sutter. Tesina por el grado de Ingeniero de Sistemas, Fac. Cs. Exactas, UNCPBA, Tandil (Argentina). Noviembre 1997.

[Tak85] "*Fuzzy identification of systems and its implication to modeling and control*", T. Takagi y M. Sugeno. IEEE Trans. on Systems, Man and Cybernetics, Vol. SMC-15, Nro. 1, pp: 116-132, 1985.

[Teo00] "*Handwritten Digit Recognition with a Novel Vision Model that Extracts Linearly Separable Features*", Loo-Nin Teow & Kia-Fock Loe. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR-2000), Vol.2, pp.76-81.

[Tod00] "*End-user low-power alternatives at topological and physical levels. Some examples on FPGAs*". XV DCIS'2000 (Conference on Design of Circuits and Integrated Systems), Montpellier - Le Corum (France). Nov. 21 y 22, 2000. Todorovich E., Sutter G., Acosta N. & Boemo E. Pp: 640-644.

[Tod00b] "*Generador automático de controladores difusos*". Congreso Argentino de Control Automático Agosto'2000. Todorovich E., Acosta N. & Acosta G.. Pp. 315-320.

[Tod99] "*Co – simulación de HW/SW usando FPGAs*", E. Todorovich, N. Acosta y C. Vazquez. Iberchip V, Lima, Perú. Marzo de 1999.

[Tog85] "*A VLSI Implementation of a fuzzy inference engine: toward an expert system on chip*", M. Togai y H. Watanabe. Information Science, Vol. 38, Nro. 2, pp: 147-163, 1985.

[Tog96] "*Expert system on a chip: an engine for real-time approximate reasoning*", Masaki Togai y Hiroyuki Watanabe. IEEE Expert, Fall 96, pp: 55-62, 1996.

[Togai] FCA, ASIC kernel fuzzy coprocessor, de TOGAI Infralogic Inc., fuzzy-server@til.com, o info@til.com

[Tom96] "*A PWM Fuzzy Logic Controller*", Tombs J., Torralba A. and Franquelo L. IEEE Micro, October, pp: 68-71. (1996).

[Ton77] "*A control engineering review of fuzzy systems*", R. M. Tong. Automatica, Vol. 13, pp: 559-569, 1977.

[Tos04] "*Identificación por hardware de posición de jugadores en futbol de robots*", Marcelo Tosini, Nelson Acosta, Claudio Aciti y Silvia Iarrar. WICC 2004. Neuquen.

[Uan02] "*Robust adaptive optimal tracking design for uncertain missile systems: a fuzzy approach*", Huey-Jian Uang, Bor-Sen Chen. Fuzzy Sets and Systems 126 (2002) Pp: 63–87.

[Ung93] "*Architecture of a PDM VLSI fuzzy logic controller with an explicit rule base*", A. P. Ungering y K. Goser. Proc. of V World Congress of IFSA, pp:1386-1389, 1993.

[Vdb98] "*The practical Xilinx designer lab book*", Dave Van den Bout. Prentice-Hall, NJ, 1998.

[Ver97] "*Hardware/Software comunication and system integration*", S. Verlauteren y B. Lin. Design Automation for Embedded Systems, Vol. 2, pp: 359-382, may 1997.

[VGo02] "*Fuzzy logic and meteorological variables: a case study of solar irradiance*", V. Gomez, A. Casanovas. Fuzzy Sets and Systems 126 (2002) Pp: 121–128.

[Vie93] "*ViewTrace Users Guide*". ViewLogic, Viewlogic Systems Inc., 293 Boston Post Road West, Marlboro, Massachusetts, 01752-4615, 1993.

[Wai03] "*Robust fuzzy neural network control for nonlinear motor-toggle servomechanism*", Rong-Jong Wai. Fuzzy Sets and Systems 139 (2003) PP: 185–208.

[Wan94] "*Fuzzy dynamic system and fuzzy linguistic control clasification*", P. P. Wang y C. Y. Tyan. Automatica, Vol. 30, Nro. 1, pp: 1769-1774, 1994.

[Wan96] "*Stable adaptive fuzzy controllers with application to inverted pendulum tracking*", Li-Xin Wang. IEEE Trans. on Systems, Man & Cybernetics, part B, Vol. 26, Nro. 5. Oct'96.

[Wat90] "*A VLSI fuzzy logic controller with reconfigurable cascadable architecture*", IEEE Journal of Solid-State Circuits, Vol. 25, pp: 376-382, April 1990.

[Wat93] "*Evaluation of Fuzzy instructions in a RISC processor*", H. Watanabe y D. Chen. Proc. IEEE Int. Conference onFuzzy Systems (San Francisco, CA), pp: 521-526, 1993.

[Wat96] "*Evaluation of Min/Max instructions for fuzzy information processing*", H. Watanabe, D. Chan y S. Konuri. IEEE Transactions on Fuzzy Systems, Vol. 4, pp: 369-374, Aug. 1996.

[Waz93] "*PRISM-II: compiler & architecture*", Wazlowski M., Agarwal L., Lee T., Smith A., Lam E., Athanas P., Silverman H. y Ghosh S.. IEEE FPGA for CCM - Napa - California - April 19-21 / 1993, pp:9-16.

[Wir95] "*DISC: the dynamic instruction set computer*", Wirthlin Michael J. & Hutchings Brad L. Dep. of Electrical & Computer Engineering. PP:92-103. SPIE - FPGA for Fast Board Development & Reconfigurable Computing. Oct'95

[WTa01] "*A modiffied fuzzy PI controller for a exible-joint robot arm with uncertainties*", Weiming Tanga, Guanrong Chena, y Rongde Lub. Fuzzy Sets and Systems 118 (2001) 109-119.

[Xil96] "*The programmable logic data book*", Xilinx Inc., 1996.

[XiT00] "*Fuzzy control of thyristor-controlled series compensator in power system transients*", Xiaobo Tan, Naiyao Zhang, Luyuan Tong, y Zhonghong Wang. Fuzzy Sets and Systems 110 (2000) 429-436.

[Xu03] "*A new fuzzy logic learning control scheme for repetitive trajectory tracking problems*", Jian-Xin Xu y Jing Xu. Fuzzy Sets and Systems 133 (2003) PP: 57–75.

[Yam87] "*Stabilization of an inverted pendulum by a high-speed fuzzy logic controller hardware system*", T. Yamakawa. Fuzzy Sets and Systems, pp: 161-180, 1987.

[YHL01] "*Application of fuzzy control for a hydraulic forging machine*", Young-Hyun Lee, R. Kopp. Fuzzy Sets and Systems 118 (2001) 99-108.

[Yia01] "*Upswing and stabilization control of inverted pendulum system based on the SIRMs dynamically connected fuzzy inference model*", Jianqiang Yia, Naoyoshi Yubazakia, Kaoru Hirotab. Fuzzy Sets and Systems 122 (2001) 139-152.

[Yon99] "*Coordination of Excitation and Governing Control Based on Fuzzy Logic*", Taiyou Yong, Robert H. Lasseter & Wenjin Cui. PSerc Publications on the web. http://www.pserc.wisc.edu/ecow/get/publicatio/.

[YuJ01] "*Fuzzy thermal placement for multichip module applications*", Yu-Jung Huanga, Mei-hui Guob. Fuzzy Sets and Systems 122 (2001) 185-194.

[Zad00] "*With fuzzy logic in the new millennium*", Lofti Zadeh. Second International Discourse on Fuzzy Logic, Honorary Speaker. Mackay, Australia. 18-22 September, 2000.

[Zen96] "*Decomposition property of fuzzy systems and its applications*", X. Zeng & M. Singh. IEEE Trans. on Fuzzy Systems, Vol. 4, Nro. 2. May'96.

[Zha95] "*System Modeling, Identification and Control using Fuzzy Logic*", Jun Zhao. Tesis doctoral, Universidad Católica de Louvain, Bélgica. Oct. 1995.

[Zho03] "*Dynamic balance of a biped robot using fuzzy reinforcement learning agents*", Changjiu Zhou y Qingchun Meng. Fuzzy Sets and Systems 134 (2003). PP:169–187.

[Zim03] "*Practical applications of fuzzy technologies*", Hans-Jurgen Zimmermann (Ed.), in The Handbook of Fuzzy Sets Series, Didier Dubois and Henri Prade (Series Eds.), Kluwer Academic Publishers, 2003, 696pp, ISBN 0-7923-8628-0.

[Zim99] "*Practical Application of Fuzzy Technologies*", H. J. Zimmermann, Editor. Kluwer Academic Pub. 1999. Vol. 6, pp. 185-206.

[Zul99] "*Fuzzy logic control of PWM inverter-fed SPMSM drives*", Ibrahim Zulkifilie, PhD. Thesis. Electric machines & drives group, JM University, Liverpool (UK). Sept. 1999.

[ZWW00] "*A PID type fuzzy controller with self-tuning scaling factors*", Zhi-Wei Woo, Hung-Yuan Chung, y Jin-Jye Lin. Fuzzy Sets and Systems 115 (2000) 321-326.