FACULTAD DE INGENIERÍA

DISEÑO DE PROTOTIPO DE UN CONTROLADOR INTELIGENTE BASADO EN LÓGICA DIFUSA PARA CONTROLAR LAS PLANTAS DEL LABORATORIO DE AUTOMATIZACIÓN DEL CITT DE LA UNIVERSIDAD DON BOSCO

TRABAJO DE GRADUACIÓN
PARA OPTAR AL TÍTULO DE
INGENIERO EN ELECTRÓNICA

PRESENTADO POR:

PEDRO JOSÉ MENDOZA PACAS
OSCAR ARMANDO GARCÍA LÓPEZ
WALTER ANTONIO AGUILAR PERALTA

ASESOR:
MSC. RIGOBERTO CHINCHILLA

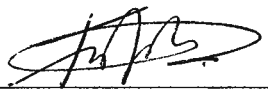
SOYAPANGO **30 DE ABRIL DE 1999**

ING. FEDERICO MIGUEL HUGUET RIVERA
RECTOR

PBRO. PEDRO JOSÉ GARCÍA CASTRO S.D.B.
SECRETARIO GENERAL

ING. CARLOS GUILLERMO BRAN
DECANO DE LA FACULTAD
DE INGENIERÍA

F. Ing. Oscar Giovanni Durán Vizcarra
JURADO



F. Ing. Federico José Laínez Olivares
JURADO

F. Msc. Rigoberto Chinchilla Salazar
ASESOR

Agradecimientos.

Gracias Dios por haberme dado la voluntad, perseverancia y paciencia para realizar la presente tesis.

Con mi papá, Dr. Pedro José Mendoza Aguilar, y mi mamá, Tecnólogo-médico María Evelyn Pacas de Mendoza, estaré siempre agradecido por brindarme un hogar lleno de amor. Agradezco a Dios por darme un papá y una mamá tan formidables.

A mis hermanos, Carlos y Guillermo, quienes "aguantan" mis arrebatos y saben darme la ayuda cuando más la he necesitado, gracias.

A mi novia, Natalia Amaya, por ser cariñosa y brindarme apoyo en los momentos difíciles de la elaboración de esta tesis, gracias.

Al Ing. Hector Carías por su colaboración brindada para esta tesis, gracias.

A mis compañeros universitarios con quienes pase varios años "batallando" para ser ingenieros, gracias por los momentos convividos.

A mis compañeros, Walter y Oscar, por el esfuerzo realizado para la elaboración de esta tesis, gracias.

A Don Oscar por el entusiasmo y vigor que nos animó durante el desarrollo de esta tesis, gracias.

Pedro José Mendoza Pacas.

Agradecimientos.

Doy mis infinitos agradecimientos a la Divinidad Suprema y a la Virgen María por las pruebas, obstáculos y oportunidades para poder alcanzar una de las principales metas en mi vida.

A mi familia, mi madre Candita y tía Elena que con su apoyo incondicional y su fe me ayudaron a perseverar hasta alcanzar mi objetivo, a las que hice partícipe de los desvelos que esta carrera nos obliga, y además de ser un motivo mas de sus preocupaciones, por todo esto y más, gracias por su amor y comprensión.

A la hoy mi esposa Normy, gracias a su comprensión y estímulo, por que supo esperarme, ya que le di el tiempo de ella a la carrera universitaria. Dedico este triunfo a mi futuro hijo(a), el cual es uno de mis motivos favoritos para seguir en esta lucha que nos proporciona la vida.

Gracias a todos por sus sinceros deseos a los demás familiares y amigos que me apoyaron y siempre estuvieron a mi lado.

A mis compañeros de tesis, que con sus conocimientos y esfuerzo logramos terminar una de las tantas metas personales, fuimos y seremos siempre un gran equipo.

Walter Antonio Aguilar Peralta.

Agradecimientos

Gracias a Dios por la vida, por este triunfo, y principalmente por haberme dado los mejores padres del mundo, Oscar Armando García y Luz Esperanza López de García, a quienes dedico lo presente.

Además quiero agradecer a todas las personas que siempre han estado alrededor de mí por su cariño y aprecio, mis abuelos, mis hermanos Jaime Ricardo y Norma Jeannette, mis tíos Fito, Toño, Carlos, Betty, Marina y Mema, mis primos Jessi y Pablo, mis mejores amigos Victor Guevara y Oscar Nóchez, mis compañeros de tésis y profesores.

Oscar Armando García López.

TABLA DE CONTENIDO.

LISTADO DE FIGURAS.

LISTADO DE TABLAS.

INTRODUCCION.

Al comenzar la elaboración de la presente tesis, lo único que se sabía de lógica difusa era que tenía aplicación en control automático y que era novedoso porque asimilaba el proceso humano para la toma de decisiones de control. Revisando libros de la biblioteca "Rafael Meza Ayau" de la Universidad Don Bosco, se encontró escasa literatura. Para apaciguar la curiosidad, se recurrió a consultar a ingenieros. Sorprendentemente solamente un ingeniero que trabaja en Siemens tiene experiencia con el tema. Entonces surgió la idea de que lógica difusa sería otra de las tantas tecnologías que pasan desapercibidas en El Salvador por varias décadas. Interesados en satisfacer la curiosidad e introducir una tecnología vanguardista de control automático al país, se realizó un trabajo exhaustivo cuyo resultado es el presente documento.

En este documento se ha sintetizado información bibliográfica de una variedad de artículos, libros y documentos. Internet ha sido una herramienta muy valiosa para la obtención de información. La tesis abarca mucho más que un estudio bibliográfico porque también se implemento un prototipo de control inteligente basado en lógica difusa con el fin de enriquecer el conocimiento teórico con el práctico.

Este documento esta dividido en seis capítulos y ocho apéndices. En el primer capítulo se desarrolla el tema de control difuso. En el capítulo dos se explica el prototipo de control difuso, enfocándose en la implementación del software de inferencia difusa. En el capítulo tres se explica el proceso de diseño del hardware para las interfaces del controlador difuso. En el capítulo cuatro se brinda la información acerca del simulador empleado y la interface de usuario diseñada e implementada. En el capítulo cinco se dan a conocer los modelos

difusos y los resultados obtenidas a lo largo del desarrollo de este proyecto. En el capítulo seis se presentan las conclusiones y recomendaciones.

El apéndice A provee toda la información conceptual necesaria de lógica difusa. En el apéndice B se desarrolla la metodología para el diseño de control difuso. En el apéndice C se brinda un resumen de conceptos básicos de control automático. En el apéndice D se describen las plantas de laboratorio del Centro de Investigación y Transferencia de Tecnología (CITT) de la Universidad Don Bosco, las cuales serán manejadas a través de control difuso. En los apéndices E y F se proveen los listados y flujogramas del programa del control difuso. En el apéndice G y H se proporcionan los listados y flujogramas para la interface de usuario. Por último, en los anexos se muestran datos técnicos del microcontrolador y los dispositivos empleados para la construcción de las interfaces.

La mayor parte de la bibliografía consultada esta escrita en el idioma inglés, por lo cual muchas de las terminologías no se tradujeron al español ya que no existen palabras equivalentes. En algunos casos se han utilizado anglicismos.

Este documento provee un medio a estudiantes e ingenieros para aprender lógica difusa e implementar controladores difusos. Además este documento pretende abrir el camino para futuras investigaciones.

DEFINICION DEL TEMA.

El desarrollo tecnológico y académico es una de las preocupaciones más importantes que se tienen en las Universidades de El Salvador. Debido a la importación de tecnologías orientadas a la industria y al consumo, es necesario actualizarse y tomar una actitud más vanguardista, solo de esta forma se podrá hacer frente al futuro. Para el caso, es de suma importancia e interés llevar a cabo investigaciones sobre lógica difusa ya que por el momento no existe el conocimiento suficiente en el ámbito académico, y en los casos prácticos, se ignoran sus ventajas y variadas aplicaciones.

Existen casos aislados en nuestro país donde se ha aplicado dicha tecnología, pero aun no se ha podido aprovechar su potencialidad para muchas aplicaciones. La Constancia S.A., empresa dedicada a la producción de cerveza, tiene controladores difusos para manejar temperaturas de los tanques de fermentación. En Molinos de El Salvador, empresa dedicada a la producción alimenticia, tiene tres controladores difusos para los intercambiadores de calor. Estas y algunas otras aplicaciones han sido hechas por Siemens de El Salvador, empresa alemana especializada en sistemas de control automático entre otras cosas.[1]

Considerando que esta tecnología ha tenido un gran impacto en los países industrializados, no pasará mucho tiempo en que productos "fuzzy" comiencen a invadir el mercado salvadoreño. Por eso el tema a desarrollarse en el proyecto de graduación busca introducir esta nueva tecnología a nivel técnico-científico a través del diseño de un prototipo de un controlador con inteligencia difusa.

---

[1] Información obtenida de la entrevista con el Ing. Alexander T. Guzmán Korth, División Energía y Automatización, Depto. de Proyectos de Automatización Industrial, Siemens El Salvador.

OBJETIVO GENERAL.

Introducir el conocimiento de la lógica difusa por medio de una investigación bibliográfica e implementar una aplicación, diseñando un prototipo de un controlador inteligente basado en lógica difusa para controlar las plantas del laboratorio de automatización del CITT de la Universidad Don Bosco.

OBJETIVOS ESPECIFICOS.

1. Elaborar un documento que sirva como base para el aprendizaje de conceptos y el funcionamiento de lógica difusa.
2. Implementar un software que permita desarrollar lógica difusa para lograr las tareas de control de las variables de temperatura, velocidad, posición y nivel de las plantas de laboratorio de automatización del CITT.
3. Diseñar una interface de usuario a través de software para programar los parámetros del controlador relacionados a las variables que se quieran manejar.
4. Diseñar interfaces entre los sensores de variables físicas y el microcontrolador.
5. Implementar el software y el hardware para efectos de demostración del diseño del prototipo de control con inteligencia difusa.

CAPITULO 1. CONTROL DIFUSO.

Al terminar de leer este capitulo, el lector tendrá una comprensión de los factores a considerar para el diseño de controladores difusos. Conocerá las diferencias entre lógica difusa y lógica Booleana, y como la implementación de estas lógicas incide en el diseño y funcionamiento de controladores automáticos. También se plantearán los criterios de hardware y software para el diseño de control difuso. Además adquirirá una mejor visión de las equivalencias entre PID y control difuso, y como pueden estos interaccionar para crear controladores híbridos. Se recomienda leer el apéndice A antes de proseguir con la lectura de este capítulo.

1.1. NUEVO PARADIGMA.

En teoría de control existen dos fundamentos comunes para los sistemas de lazo cerrado:

- La planta a controlar debe conocerse a través de un modelo que permita predecir el funcionamiento a partir de las variables de entrada.
- El control debe plasmarse a través de una ecuación matemática.

A medida que los sistemas se vuelven más complejos, modelarlos matemáticamente es casi imposible. Ya que las computadoras han evolucionado mucho, con ellas se han construido sistemas expertos de control con los cuales se trata de manipular plantas no lineales que tienen muchas variables intervinientes.

La desventaja de estos sistemas es que funcionan con algoritmos que buscan una operación funcional pero no se puede asegurar que tal sea la más óptima.

El nuevo paradigma en control es hacer uso de computación simbólica tal como la clásica aplicación de inteligencia artificial en el diseño de cualquier algoritmo de control. Esta nueva metodología da una nueva perspectiva a la heurística en ingeniería de control:

- Para obtener un buen controlador PID, es necesario tomar en cuenta factores como operación, interface operador y efecto de actuadores no lineales.
- Controladores multivariables y autosintonizables que no sólo se basen en leyes teóricas de control sino que también significativamente en la heurística.
- La heurística se aplica en todo procedimiento de toma de decisión para lograr un desempeño óptimo del controlador.
- Se elimina la etapa de pruebas iterativas para la construcción del modelo. Un índice de desempeño no es necesario expresarlo de forma explícita.[2]

1.2. GENERALIDADES DE LOGICA DIFUSA.

El concepto de lógica difusa fue concebido por Lofti Zadeh, profesor de la Universidad de California en Berkley. Este concepto no fue presentado como una metodología de control, sino como una forma de procesar datos permitiendo membresías parciales de conjuntos en vez de membresías "crisp". Debido a la baja capacidad de las computadoras de aquel entonces, esta teoría de conjuntos no fue aplicada en sistemas de control hasta los años 70.

Lógica difusa ofrece características únicas que la hacen una buena elección para muchos problemas de control:

- Es inherentemente robusto. No requiere de entradas precisas y libres de ruido y puede programarse para fallos si el sensor de retroalimentación se daña.
- La salida de control es una función de control suave no importando el amplio rango de variación de la entrada.
- Ya que los procesos de control son definidos por el usuario, estos pueden modificarse para mejorar o drásticamente alterar el desempeño del sistema.
- Nuevos sensores pueden fácilmente ser incorporados en el sistema simplemente generando más reglas de control.
- Lógica difusa no esta limitada a una entrada de retroalimentación y a uno o dos salidas de control, por lo que no es necesario la medición de variación entre parámetros para ser implementados. Cualquier sensor que muestre alguna indicación del comportamiento del sistema es suficiente. Esto permite que los sensores sean baratos e imprecisos, manteniendo bajos el costo y la complejidad del sistema.
- Lógica difusa puede controlar sistemas no lineales que serían difíciles o imposibles de modelar matemáticamente.[15]

En una perspectiva matemática, los conjuntos difusos y probabilidades existen como parte de una teoría de información generalizada que incluye muchos formalismos para representar incertidumbre. Las expresiones probabilísticas son acerca de la similitud del resultado; un evento ocurre o no. Pero con lo difuso, se trata de modelar el grado en que un evento ocurrió.[16]

La lógica difusa debe verse como una teoría matemática formal para la representación de incertidumbre. Incertidumbre es crucial en el manejo de sistemas reales. La presencia de incertidumbre es el precio que se paga al manejar sistemas complejos. Sin embargo, la lógica difusa es un formalismo matemático, y un grado de membresía es un número preciso. Debe comprenderse que lógica difusa es la lógica de lo difuso, no una lógica que en sí es difusa.[16]

1.3. LOGICA BOOLEANA Y DIFUSA.

La diferencia esencial entre la lógica difusa y la Booleana tiene que ver con la cantidad de información que ofrecen. Los resultados matemáticos de la lógica Booleana indican solamente si un elemento pertenece a un conjunto. Mientras tanto la lógica difusa ofrece mayor información al indicar el grado de pertenencia para ponderar el efecto sobre otros conjuntos difusos.

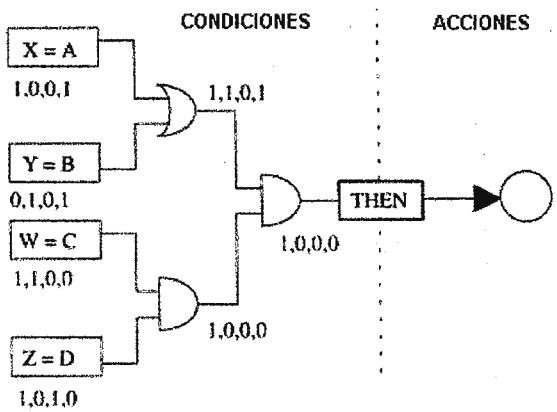


Figura 1. 1 Evaluación de condiciones con lógica Booleana [1].

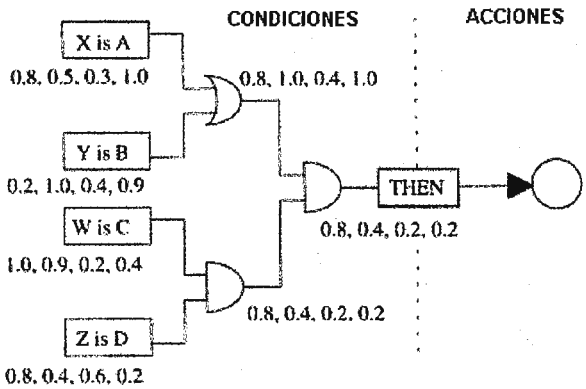


Figura 1. 2 Evaluación de condiciones con lógica difusa [1].

Es esto lo que permite a la lógica difusa ser una herramienta atractiva para el diseño e implementación de sistemas de control automático. Como puede apreciarse en las Figuras 1.1 y 1.2, un cero o uno nada más indican la pertenencia

a un conjunto; mientras que los grados de pertenencias obtenidos con lógica difusa permite ponderar la fuerza de la acción a tomar.

Además, la forma de las funciones de membresía[1] permite que los controles difusos tengan mayor inmunidad al ruido que los controles programables basados en lógica Booleana. Esta inmunidad contribuye a que el control difuso sea más robusto. En controles basados en lógica Booleana suceden cambios abruptos en la salida cuando hay pequeñas fluctuaciones en valores próximos a los límites de los conjuntos Booleanos.

En el caso de lógica difusa, el hecho que las funciones de membresía puedan tener transiciones suaves hace posible que las desviaciones en la respuesta se minimicen. Esto es muy importante en ambientes ruidosos. Si se diseñan adecuadamente las funciones de membresía, la inmunidad al ruido será superior a los sistemas de control basados en lógica Booleana.

A continuación se muestran los resultados obtenidos de un experimento en donde controles difusos y Booleanos se sometieron a ruido aditivo Gaussiano en las entradas[2]. La variable lingüística de entrada esta compuesta por tres funciones de membresía de tipo Gaussiana: Pequeño, Mediano y Largo (ver Figura 1.3).

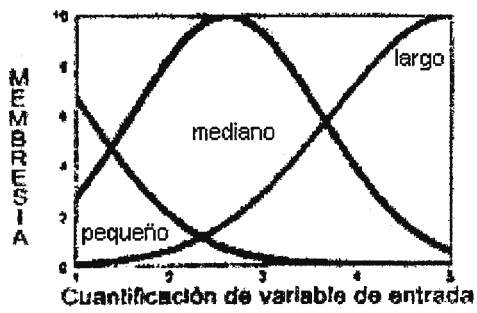


Figura 1. 3 Funciones de membresía de tipo Gaussiana para una variable lingüística.

[1] Ver apéndice A.
[2] Este experimento se obtuvo de[2].

Al hacer una prueba a sistemas de control difuso y Booleana sometidos a ruido con distribución Gaussiana se obtuvieron los resultados mostrados en la Figura 1.4.

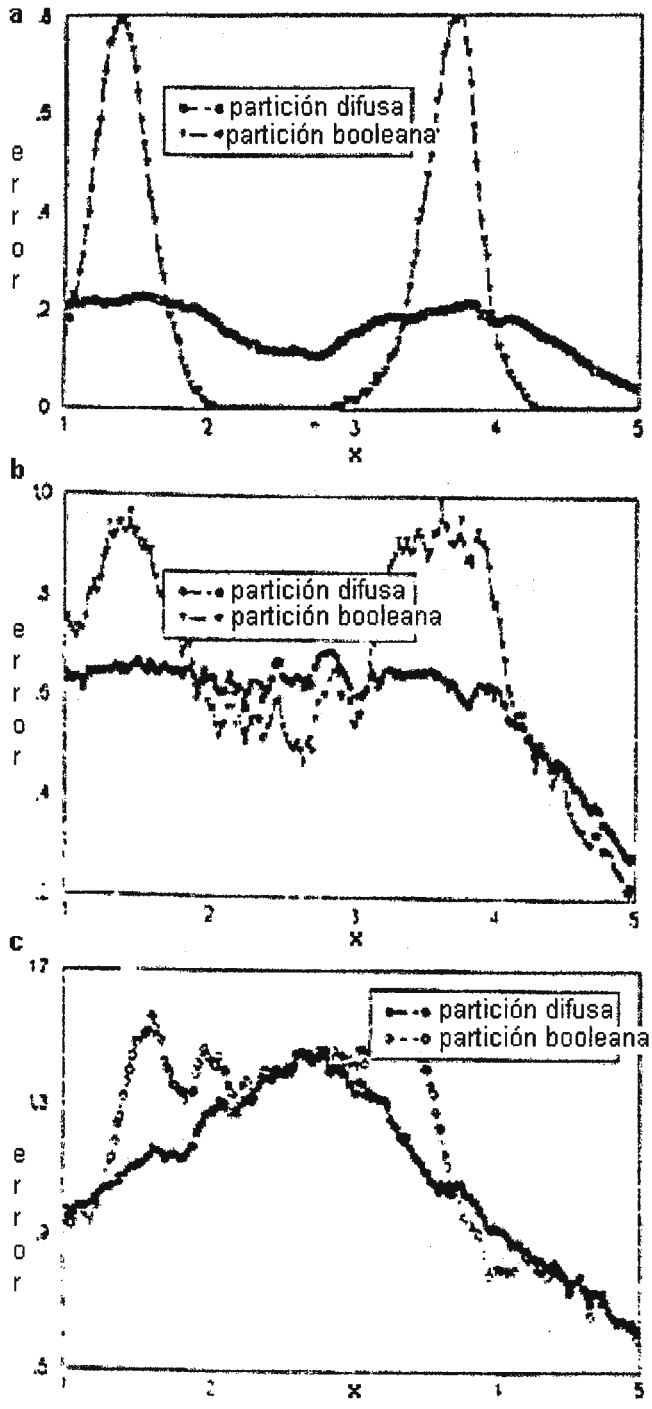


Figura 1. 4 Error vrs. valor de entrada con desviación estándar de 0.25 (1), 1.0 (2), 3.0 (3).

En las Figura 1.4 puede verse que en lógica Booleana una pequeña desviación genera cambios abruptos que inciden en un mayor error; mientras que en control difuso las variaciones son más suaves y pequeñas. De las gráficas de barras de la Figura 1.5 se puede contemplar que la distribución de error esta más concentrada para la partición difusa.

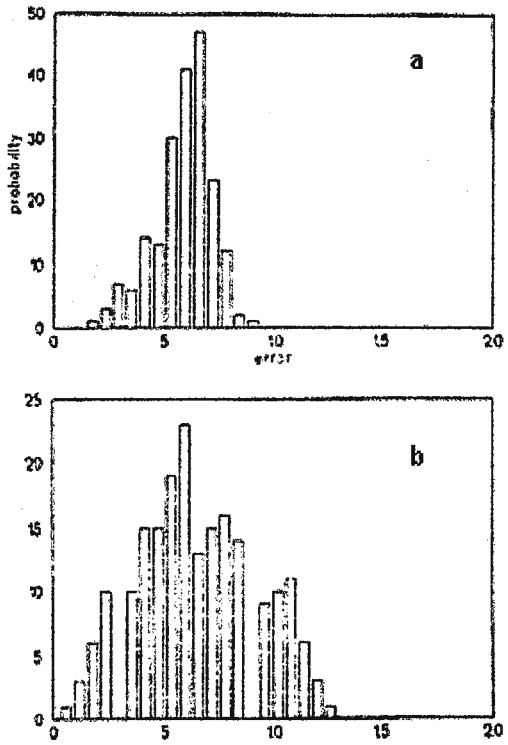


Figura 1. 5 Histograma de salida para desviación de 1.0, partición difusa (a) y booleana (b).

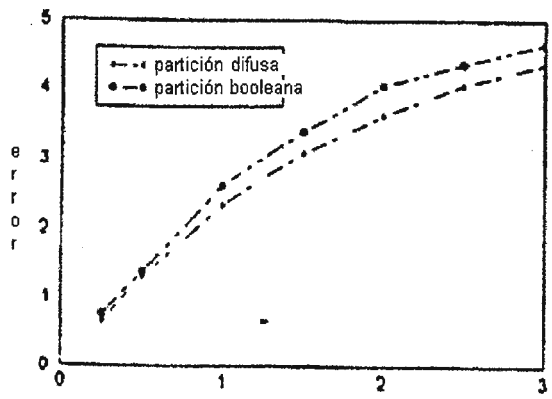


Figura 1. 6 Cuadro comparativo de particiones difusa y booleana versus desviación estándar.

1.4. OPERACIONES SOBRE CONJUNTOS DIFUSOS.

Las leyes de Morgan, las cuales son válidas en lógica Booleana, son aplicables en lógica difusa:

$$\overline{(A \cap B)} = \overline{A} \cup \overline{B}$$
$$\overline{(A \cup B)} = \overline{A} \cap \overline{B}$$

También las siguientes leyes son válidas:

Comutatividad	$A \cup B = B \cup A$
	$A \cap B = B \cap A$
Asociatividad	$(A \cup B) \cup C = A \cup (B \cup C)$
	$(A \cap B) \cap C = A \cap (B \cap C)$
Distributiva	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
Absorción	$(A \cap B) \cup A = A$
	$(A \cup B) \cap A = A$
Idempotencia	$A \cup A = A$
	$A \cap A = A$

Las siguientes leyes establecen la diferencia entre lógica Booleana y difusa:

Ley de exclusión	$A \cup \overline{A} \neq X$
Ley de contradicción	$A \cap \overline{A} \neq \phi$

En la Figura 1.7 se muestran las operaciones básicas que se aplican a conjuntos difusos. La gráfica del complemento demuestra una de las diferencias conceptuales entre lógica Booleana y difusa. En lógica Booleana el complemento es todo elemento que no forma parte del conjunto. Sin embargo para lógica difusa,

el complemento es la función resultante de la operación 1-μ(x), la cual expresa cuanto le falta al elemento x para tener membresía de 100% en el conjunto.

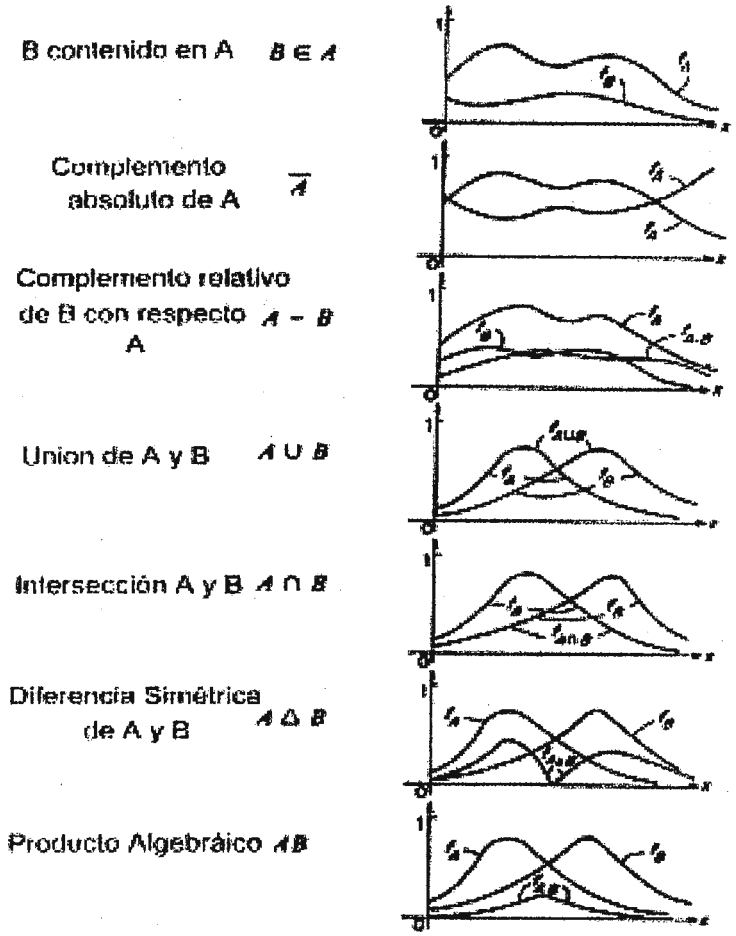


Figura 1. 7 Operadores difusos más utilizados.

También existen otros operadores que se aplican en control difuso. Cuando se utilizan tablas de búsqueda[3] al implementar el control difuso, estos operadores permiten establecer la relación entre los conjuntos difusos de las variables de entrada y salida. Lógica difusa es tan flexible que el diseñador de una aplicación de control difuso puede crear sus propios operadores. En la Tabla 1.1 se muestran algunos operadores comúnmente utilizados en control difuso[4].

[3] Ver sección 1.7: Criterios Basados en Hardware y Algoritmos.
[4] Esta es una recopilación de [2], [3] y [6].

Operador de Implicación	$\phi[\mu_A(x),\mu_B(y)] =$
Mamdani	$\mu_A(x) \wedge \mu_B(y)$
Larsen	$\mu_A(x) \cdot \mu_B(y)$
Lukasiewicz	$1 \wedge (1 - \mu_A(x) + \mu_B(y))$
Kleen-Dienes	$(1 - \mu_A(x)) \vee \mu_B(y)$
Producto Union	$0 \vee (\mu_A(x) + \mu_B(y) - 1)$
Zadeh	$(\mu_A(x) \wedge \mu_B(y)) \vee (1 - \mu_A(x))$
Estandar	$1 \quad if \quad \mu_A(x) \leq \mu_B(y)$ $0 \quad if \quad \mu_B(x) > \mu_A(y)$
Producto Drástico	$\mu_A(x) \quad if \; \mu_B(y) = 1$ $\mu_B(x) \quad if \; \mu_A(y) = 1$ $0 \quad if \; \mu_A(y) < 1 \; \mu_B(y) < 1$
Gougen	$1 \quad if \quad \mu_A(x) \leq \mu_B(y)$ $\frac{\mu_B(y)}{\mu_A(x)} \quad if \quad \mu_B(x) > \mu_A(y)$
Godelian	$1 \quad if \quad \mu_A(x) \leq \mu_B(y)$ $\mu_B(y) \quad if \quad \mu_B(x) > \mu_A(y)$

Tabla 1. 1 Operadores difusos para relaciones de conjuntos [1].

1.5. DISEÑO DE CONTROLADORES DIFUSOS.

Como se ha mencionado anteriormente, no es necesario modelar matemáticamente la planta. Realmente, lógica difusa aprovecha el concepto de caja negra, en el cual no importa que proceso intermedio existe para obtener salidas a partir de las entradas. Es decir, no hay que conocer la estructura interna de la planta, sino establecer relaciones entre entradas y salidas.

Para saber aspectos generales del funcionamiento de la planta, conocer las variables intervinientes y los resultados esperados debe entrevistarse al operario o a cualquier persona supervisora. A la persona puede solicitársele escribir un pequeño ensayo en que describa la planta y su funcionamiento junto con los detalles anteriormente mencionados. Con esta información se puede categorizar las entradas y salidas del sistema. También es útil hacer mediciones para obtener gráficas que indiquen numéricamente el funcionamiento de la planta tanto para las salidas y entradas, así como cualquier otro parámetro interviniente. Ya con la información anterior, se puede optar por un motor de inferencia basado en matrices o reglas IF-Then.

Para sistemas difusos matriciales, se analizan las entradas y salidas para cuantificar la relación de los valores críticos o más importantes. Estos valores se almacenan en una matriz donde el desplazamiento sobre los ejes esta relacionado con las funciones de membresía de las variables lingüísticas. Dependiendo de los operadores de implicación que se utilicen, así serán los procesos aritméticos aplicados a las matrices para el manejo de fusificación y defusificación. El otro tipo de sistema de control difuso utiliza la inferencia modus ponens. Este tipo de inferencia hace uso de variables lingüísticas con las cuales se establecen relaciones entre las variables por medio de reglas IF-THEN.

1.6. CRITERIOS PARA EL DISEÑO DE VARIABLES LINGÜISTICAS.

La teoría de sistemas difusos no explica la complejidad de cómo el cerebro humano trabaja, sino que provee la capacidad matemática para imitar procesos cognoscitivos en una forma sencilla. Gracias al concepto de variables lingüísticas es que puede imitarse el pensamiento humano primitivo. La idea principal detrás de las mediciones difusas es construir un puente analítico entre matemáticas y el fenómeno de incertidumbre encontrado en la práctica. El diseño de sistemas difusos es el intento de sistematizar las variaciones naturales en la percepción humana de la verdad e imitar habilidades rudimentarias de aproximación [1]. En control difuso es importante el diseño de las variables lingüísticas y de sus respectivas funciones de membresía. El número de variables y las formas de las funciones de membresía tendrán efecto en la eficiencia y eficacia del sistema para reaccionar ante las entradas. A continuación se presentan algunos criterios a considerar:

- Conservación/tolerancia: Este criterio se utiliza para las funciones de membresía y las reglas cuando las consecuencias son conocidas. En las funciones de membresía, la tolerancia/conservación se refiere a la formación de conjuntos tomando en cuenta un margen de error por mala categorización de los elementos. Para las reglas, a través de variables de peso se modifica la contribución de cada regla al proceso de agregación, siendo cada peso como un indicador de la tolerancia de incertidumbre.

- Optimismo/pesimismo: Este criterio se emplea cuando no se conocen las consecuencias. Esta íntimamente relacionado con la evaluación probabilística de condiciones para determinar las correspondientes acciones.

- Reactividad/retardo: Este criterio se aplica al proceso de agregación, del cual depende el manejo y procesamiento de datos para determinar y cuantificar el

resultado. Si el proceso responde adecuadamente para sistemas donde hay cambios abruptos en la salida, entonces es reactivo. Si el sistema tiene variaciones suaves, el sistema es retardado.

- Precisión, exactitud y costo: La precisión esta ligada con el número de funciones de membresía y reglas. Entre más funciones de membresía se declaran, la precisión aumenta. También ocurre lo mismo para las reglas. Sin embargo el costo se comporta de forma inversa. El costo se relaciona con las decisiones y reglas. Si estas se modifican, ¿cual sería el costo en términos de los anteriores criterios?.

- Robustez/sensitividad: Esto tiene relación con la capacidad del sistema de responder ante perturbaciones. Entre más inmune es el sistema ante perturbaciones, la robustez del sistema aumenta. Si se presentan grandes variaciones en el comportamiento de la planta por ruido, entonces el sistema se dice que es sensitivo.

Los sistemas de control basados en la lógica Booleana son menos robustos que los controles difusos. Una pequeña variación en la variable de entrada puede modificar abruptamente la pertenencia en los conjuntos establecidos, lo cual generará grandes cambios en la salida. La suma de las diferencias entre el valor de activación esperado y el valor obtenido por cada función de membresía de una variable lingüística sirve de indicador de la robustez del sistema.

$$r(x) = \left|A_1(x) - A_1(x')\right| + \left|A_2(x) - A_2(x')\right| + \ldots \left|A_n(x) - A_n(x')\right| \qquad \text{Ec. 1. 1}$$

en donde,

- r(x) es el índice de robustez,
- $A_n(x)$ es el grado de pertenencia de la entrada x en el conjunto A (si es un conjunto Booleano los valores a tomar son 0 ó 1;si es difuso, cualquier valor entre 0 y 1),
- x es la variable de entrada y x´ la variable de entrada más ruido.

1.7. CRITERIOS BASADOS EN HARDWARE Y ALGORITMOS.

Al diseñar los sistemas físicos que ejercerán el control difuso es imperativo que se considere un dispositivo digital capaz de procesar información. Es decir, el controlador debe incluir ya sea un microprocesador o un microcontrolador. Al utilizar los anteriores dispositivos debe considerarse los requerimientos de memoria y tiempo de procesamiento. Como se gobernará un sistema en tiempo real, los tiempos de acceso a memoria y los ciclos de reloj que consume la ejecución de instrucciones deben tomarse en cuenta ya que sus efectos son acumulativos, provocando un retraso en la reacción. Es importante dimensionar la memoria y verificar la velocidad del dispositivo procesador para que el controlador no caiga en las siguientes categorías:

- Un modelo que haga uso intensivo de memoria.
- Un modelo de ejecución muy lento.
- Una arquitectura de procesador inadecuado para las necesidades del modelo: número de bits para codificación digital, capacidad de direccionamiento de memoria.

Muchas aplicaciones requieren que se hagan consideraciones entre memoria y velocidad. Existen dos métodos para implementar modelos difusos en sistemas procesadores: tablas de búsqueda e inferencia activa.

En la tabla de búsqueda se analizan los datos de entrada, sus relaciones y el efecto en las salidas. Con todos estos datos se construye una matriz en donde cada elemento es la cuantificación de la relación entre variables de entrada y las variables de salida. El sistema controlador solamente debe transformar las entradas en índices de desplazamiento para buscar en memoria el valor correspondiente. Para que el controlador sea eficiente, debe de hacerse muchos

experimentos y análisis para obtener una matriz óptima, la cual finalmente quedará almacenada en el sistema controlador.

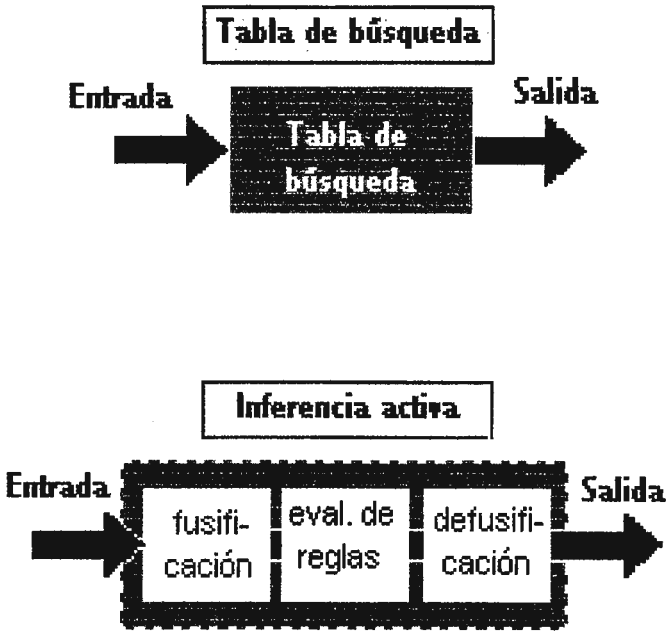


Figura 1. 8 Esquemas de inferencias difusas.

La inferencia activa es el método en que los procesos de defusificación, evaluación de reglas y defusificación se incorporan al sistema controlador. Comparándose con la tabla de búsqueda, la inferencia activa provee de mayor inteligencia al controlador difuso. La reactividad de controladores utilizando tablas de búsquedas depende mucho de las relaciones establecidas en la matriz. En la inferencia activa la representación de los conjuntos difusos se hace utilizando formas geométricas sencillas, ya que para representarlas en el sistema controlador sólo basta con proveer puntos y pendientes, siendo interpretados por un algoritmo que cuantifica y modifica dinámicamente la relación entre entradas y salidas. La inferencia activa posee la capacidad de generar un continuo, mientras que la tabla de búsqueda es un sistema discreto al cual puede proveerse de algoritmos para interpolar pero no alcanzaría a generar el mismo continuo.

En sistemas con inferencia activa pueden construirse modelos de control difuso más complejos. Para sistemas de alta reactividad, las tablas de búsqueda son efectivas porque no involucran mucho procesamiento, sin embargo la efectividad de control esta íntimamente ligada con la precisión con que se cuantifica la relación entre entradas y salidas que se almacenan en memoria.

Ventajas de Tablas de Búsqueda	**Ventajas de Inferencia Activa**
1. Rápido procesamiento de inferencia	1. Ajustes son más fáciles. 2. En un sistema difuso adaptativo es necesario. 3. Generalmente requiere menos memoria.

Tabla 1. 2 Cuadro Comparativo de tablas de búsqueda e inferencia activa.

1.8. CONTROL DIFUSO DISCRETO.

Al implementarse controladores difusos usando procesadores de gran capacidad, generalmente las funciones de membresía se diseñan para el dominio continuo. Existe otro tipo de control difuso en que el dominio se convierte en segmentos discretos para utilizar tablas de búsqueda en vez de reglas IF-THEN. Las características principales del control discreto son:

- El dominio continuo se cuantifica en un número finito de segmentos llamados niveles de cuantización. Cada segmento es un elemento genérico y el conjunto de todos estos elementos forman un universo discreto de discurso.
- A cada elemento genérico que forma la variable lingüística se le asigna un grado de membresía[5].

[5] Recopilación de [6].

En el siguiente ejemplo se aplica el manejo y procesamiento de conjuntos difusos discretos. Para el ejemplo hay dos conjuntos difusos que se definen de la siguiente manera:

A(x)=0/1∪0.25/2∪ 0.5/3∪0.75/4∪1/5∪0.75/6∪0.5/7∪0.25/8∪0/9∪0/10

B(X)=0/1∪0.25/2∪ 0.5/3∪0.75/4∪1/5∪1/6∪0.75/7∪0.5/8∪0.25/9∪0/10

Y la entrada al sistema es:

I(x)= 0/1∪0/2∪ 0/3∪1/4∪0/5∪0/6∪0/7∪0/8∪0/9∪0/10

El vector I(x) significa que en la entrada se obtuvo el escalar 4, que para el sistema representa la activación del cuarto singleton. El siguiente paso es aplicar el operador de implicación de Larsen, Ec.1.2:

$$\mu(x,y) = \Phi[\mu_A(x), \mu_B(y)] = \mu_A(x) \cdot \mu_B(y)$$

$$\mu(x,y) = \begin{bmatrix} 0 \\ 0.25 \\ 0.5 \\ 0.75 \\ 1 \\ 0.75 \\ 0.5 \\ 0.25 \\ 0 \\ 0 \end{bmatrix} * [0 \;\; 0.25 \;\; 0.5 \;\; 0.75 \;\; 1 \;\; 1 \;\; 0.75 \;\; 0.5 \;\; 0.25 \;\; 0]$$

$$\mu(x,y) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.06 & 0.13 & 0.19 & 0.25 & 0.25 & 0.19 & 0.13 & 0.06 & 0 \\ 0 & 0.13 & 0.25 & 0.38 & 0.50 & 0.50 & 0.38 & 0.25 & 0.13 & 0 \\ 0 & 0.19 & 0.38 & 0.56 & 0.75 & 0.75 & 0.56 & 0.38 & 0.19 & 0 \\ 0 & 0.25 & 0.50 & 0.75 & 1.00 & 1.00 & 0.75 & 0.50 & 0.25 & 0 \\ 0 & 0.19 & 0.38 & 0.56 & 0.75 & 0.75 & 0.56 & 0.38 & 0.19 & 0 \\ 0 & 0.13 & 0.25 & 0.38 & 0.50 & 0.50 & 0.38 & 0.25 & 0.13 & 0 \\ 0 & 0.06 & 0.13 & 0.19 & 0.25 & 0.25 & 0.19 & 0.13 & 0.06 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mu(x,y) = \mu_a(x) * \mu_b(y) \qquad \text{Ec. 1. 2}$$

$$\mu'_B(y) = I(x) \circ \mu(x,y)$$

$$\mu'_B(y) = [0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0] \circ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.06 & 0.13 & 0.19 & 0.25 & 0.25 & 0.19 & 0.13 & 0.06 & 0 \\ 0 & 0.13 & 0.25 & 0.38 & 0.50 & 0.50 & 0.38 & 0.25 & 0.13 & 0 \\ 0 & 0.19 & 0.38 & 0.56 & 0.75 & 0.75 & 0.56 & 0.38 & 0.19 & 0 \\ 0 & 0.25 & 0.50 & 0.75 & 1.00 & 1.00 & 0.75 & 0.50 & 0.25 & 0 \\ 0 & 0.19 & 0.38 & 0.56 & 0.75 & 0.75 & 0.56 & 0.38 & 0.19 & 0 \\ 0 & 0.13 & 0.25 & 0.38 & 0.50 & 0.50 & 0.38 & 0.25 & 0.13 & 0 \\ 0 & 0.06 & 0.13 & 0.19 & 0.25 & 0.25 & 0.19 & 0.13 & 0.06 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mu'_B(y) = 0/1 \cup 0.19/2 \cup 0.38/3 \cup 0.56/4 \cup 0.75/5 \cup 0.75/6 \cup 0.56/7 \cup 0.38/8 \cup 0.19/9 \cup 0/10$$

$$\mu'_B(y) = [0 \quad 0.19 \quad 0.38 \quad 0.56 \quad 0.75 \quad 0.75 \quad 0.56 \quad 0.38 \quad 0.19 \quad 0]$$

El resultado final es un vector que representaría el área sombreada de una función de membresía después de la defusificación para la inferencia activa [1].

1.9. CONTROL DIFUSO Y PID.

Existen muchos sistemas que poseen controles automáticos. Si el comportamiento de la planta es lineal y el rango a controlar es pequeño sin someterse a perturbaciones extremas, es muy posible que un control PID este haciendo la función de controlador. Sin embargo, al tener la planta un comportamiento no lineal y ser expuesta a perturbaciones extremas, con seguridad el controlador emplea control difuso. Pero como se ve en el siguiente ejemplo, con lógica difusa puede diseñarse un controlador difuso que haga las veces de un controlador PID.

La función característica de un controlador proporcional-integral (PI) es:

$$u = -K_p e - K_i \int e dt \qquad \text{Ec. 1. 3}$$

$$e(t) = y(t) - r(t) \qquad \text{Ec. 1. 4}$$

en donde, u es la señal de control; y e, la señal de error entre el estado de operación y el deseado. Siendo la señal de error la diferencia entre la señal de salida y(t) y la señal de referencia r(t).

Con estas ecuaciones puede apreciarse que si el error es positivo, entonces el control genera una señal negativa para compensar el sistema de tal forma que el error disminuya. Las constantes Kp y Ki sirven para acelerar o retardar la recuperación del sistema a un estado de equilibrio (e=0).

El conocimiento adquirido a través del análisis de la ecuación, puede expresarse en términos de lógica difusa de la manera mostrada en la Tabla 1.3.

Control	**Señal de Error**	**Señal de Error Integral**
Muy negativo=MN	Negativo grande= NG	Negativo grande= NG
Negativo bastante grande=NB	Negativo pequeño= SN	Negativo pequeño= SN
Negativo grande=NG	Cero= CR	Cero= CR
Negativo medio= NM	Positivo pequeño= PP	Positivo pequeño= PP
Negativo pequeño= NP	Positivo grande= PG	Positivo grande= PG
Cero= CR		
Positivo pequeño=PP		
Positivo medio= PM		
Positivo grande= PG		
Positivo bastante grande= PB		
Muy Positivo= MP		

Tabla 1.3 Variables lingüísticas y funciones de membresía.

Las reglas base IF-THEN se exponen en la representación matricial de la Tabla 1.4:

ERROR / INTEGRAL / ERROR	NG	NP	CR	PP	PG
PG	CR	PM	PG	PB	MP
PP	NM	CR	PP	PM	PB
CR	NG	NP	CR	PP	PG
NP	NB	NM	NP	CR	PM
NG	MN	NB	NG	NM	CR

Tabla 1.4 Matriz de reglas IF-THEN.

Luego se obtiene un diagrama normalizado fijando parámetros r, Kp y Ki.

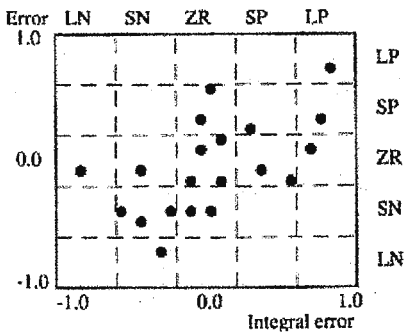


Figura 1. 9 Diagrama normalizado de error con el cual se delimitan funciones de membresía.

Con este diagrama se puede esbozar los conjuntos difusos o las funciones de membresía, quedando al criterio del diseñador la forma de estas. En la Figura 1.10 se muestra un esquema del controlador difuso equivalente al control PI.

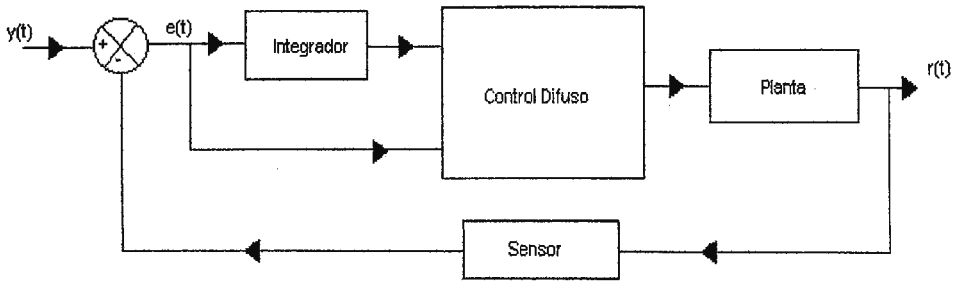


Figura 1. 10 Esquema de control difuso equivalente a control PI.

En el ejemplo anterior se demuestra que existen equivalentes de control difuso para controles PID. La verdadera utilidad de control difuso es el control de plantas no lineales y también ofrece la fácil incorporación de varias variables, expresando la relación entre ellas a través de reglas If-Then[6].

1.10. CONTROLES PID VS. CONTROLES DIFUSOS.

El control proporcional-integral-derivativo (PID) tiene cincuenta años de ser utilizado y todavía tiene una amplia aplicación en controles industriales. El PID es simple, fácil de entender, fácil de implementar en hardware y software y no requiere un modelo preciso para comenzar o mantener. Debido a sus méritos, el PID ha sido la mayor fuerza de trabajo para procesos de control. Sin embargo, el PID también tiene grandes deficiencias:

Primero, el PID trabaja para procesos que son básicamente lineales o invariantes en el tiempo. Una gran cantidad de procesos industriales son bilineales o no lineales y variables en el tiempo. Segundo, el PID debe ajustarse correctamente. Los parámetros Kp, Ti y Td deben ajustarse basado en la dinámica

[6] Ejemplo tomado de[1].

del proceso. Si la dinámica del proceso varia ya sea por cambios de carga, por ejemplo, el PID debe de ajustarse de nuevo. En aplicaciones reales, el ajuste de un PID es frecuentemente una experiencia frustrante. Tercero, el PID tiene problemas para controlar sistemas complejos que son usualmente no lineales, variantes en el tiempo y tienen muchos parámetros inciertos.

Debido a las serias desventajas mencionadas anteriormente, se han diseñado controladores PID capaces de ajustarse automáticamente (Adaptivo). Si el ajuste esta basado en un modelo, encontrar y mantener un buen modelo del proceso en línea para el reajuste del PID es un gran reto. Frecuentemente se requiere introducir señales de prueba al proceso en el lazo cerrado. Si el reajuste se basa en reglas, es usualmente difícil distinguir entre efectos de perturbaciones y cambios en el proceso dinámico. El controlador puede reaccionar demasiado ante perturbaciones y crear transiciones de adaptación innecesarias. Además, la confiabilidad del ajuste puede ser cuestionable ya que no existe un análisis de estabilidad certero para sistemas basados en reglas.

Los controladores basados en modelos lineales como el control PID pueden diseñarse de acuerdo a algún criterio óptimo, y la estabilidad como la optimización pueden probarse. Sin embargo es muy difícil diseñar un controlador óptimo y estable para un proceso no muy definido, dinámico y no lineal. Un método práctico es simplificar y linealizar un modelo no lineal. Con la simplificación, la optimización y la estabilidad sólo se aplicará para el modelo simplificado. Para un proceso no muy fácil de definir, su modelo no se conoce y es imposible usar métodos convencionales de diseño de controladores. Es aquí donde lógica difusa entra como alternativa al permitir diseñar controladores que incorporen la experiencia del operador humano. El desempeño de controladores PID depende bastante en los parámetros de operación del sistema. Cualquier cambio en el sistema, se requiere mucho tiempo para ajustar el controlador[7].

[7] Información obtenida de [17].

En la siguiente Tabla se compara el proceso de diseño de controladores PID y difuso:

PID	Lógica Difusa
1. Modelado: Construir un modelo matemático que describa el proceso. 2. Linealización del modelo. 3. Resolver ecuaciones: Hacer pruebas de diseño basadas en control óptimo u otros criterios. 4. Simulación: Simular el diseño. Si resultados no son satisfactorios, volver al paso 1.	1. Análisis: Analizar proceso. 2. Adquisición de reglas: Adquirir reglas de control a través de la experiencia de los operadores. 3. Simulación: Simular el controlador difuso. Si no satisface, volver al paso 1.

Tabla 1 .5 Cuadro comparativo de procesos para diseñar controladores PID y difusos.

Comparando con sistemas de control PID, lógica difusa tiene sus ventajas y desventajas. El control difuso es más versátil, flexible y robusto pero los controladores PID ofrecen mayor precisión.

El control difuso posee la flexibilidad de adaptación del control difuso. Por medio del modelo difuso puede establecerse modelos lineales equivalentes a los establecidos en controladores PID (ver Figura 1.11), así como también establecer relaciones no lineales de entrada/salida mucho más complejos que controladores PID adaptivos (ver Figura 1.12).

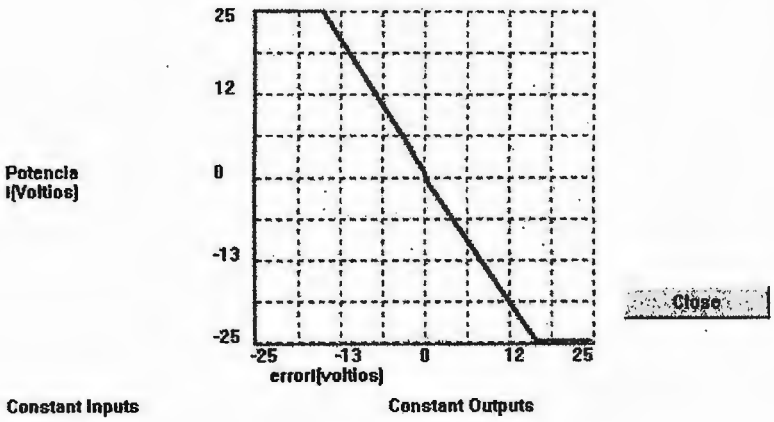


Figura 1. 11 Relación lineal entre salida/entrada.

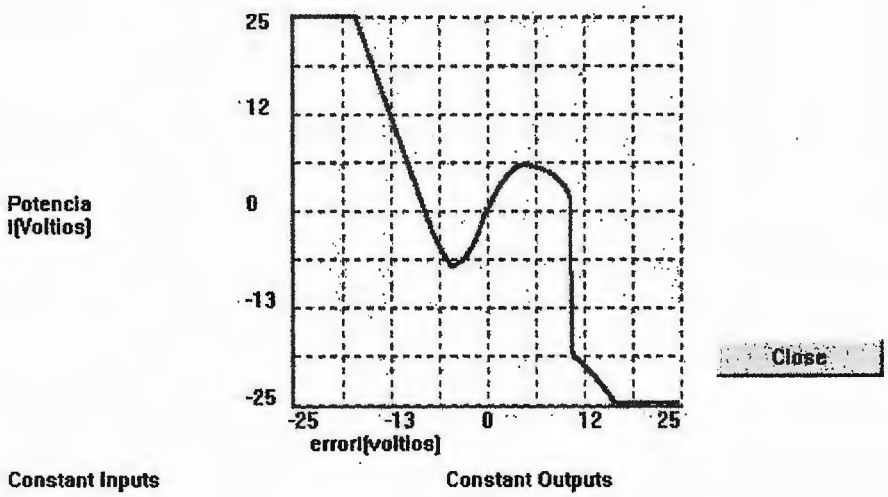


Figura 1. 12 Relación de entrada/salida compleja.

En lógica difusa establecer las relaciones entre salidas y entradas es mucho más fácil que en controladores PID. El control difuso sólo requiere declararse y modificarse las funciones de membresía así como crear y modificar reglas; mientras que en controladores PID, los cuales solamente pueden usarse en sistemas lineales, únicamente pueden modificarse los parámetros de ganancia proporcional, tiempo integral y tiempo derivativo.

Diseñar un controlador PID que sea capaz de tomar en cuenta muchas variables de entrada para manejar una salida es un proceso complicado, ya que debe encontrarse un modelo matemático, él cual si se llegase a establecer esta probablemente será válido para ciertos rangos de valores de entrada. La gran bondad de lógica difusa es la facilidad de obtener modelos difusos donde muchas variables de entrada sean consideradas y cuya aplicación sea para un rango mayor que el permitido por un controlador equivalente PID.

Un control difuso puede diseñarse y modificarse fácilmente para que sea más robusto o sensible. La inmunidad al ruido puede mejorarse modificando el modelo difuso, lo cual en controladores PID esta subordinado en gran parte por la construcción interna del controlador.

Un controlador difuso por su misma naturaleza tiene la desventaja de ser un sistema con retardo. El retardo se debe al procesamiento de datos que está muy ligado con la base de tiempo con que trabaja la unidad de procesamiento. Por tanto la velocidad de muestreo esta limitada a la velocidad de procesamiento, limitando aplicar el controlador difuso en aquellas situaciones en donde la frecuencia de cada parámetro de la planta sea la mitad de la frecuencia de muestreo (teorema de Nyquist).

Al Dr. Lofti Zadeh se le formuló la siguiente pregunta: ¿por qué preocuparse por el control difuso, si el PID y otras formas tradicionales de sistemas de control están bien desarrolladas? Su respuesta fue: "Sólo porque tiene algunas ventajas: en muchos casos, el modelo matemático del proceso de control puede que no exista o pueda ser muy costoso en términos de poder de procesamiento y memoria; por tanto, un sistema de control difuso basado en reglas empíricas puede ser más efectivo. Además, lógica difusa se adecua para implementaciones de bajo costo con sensores baratos, convertidores analógicos/digitales de baja resolución y microcontroladores de 4 u 8 bits, y tales sistemas pueden ser fácilmente actualizados agregando nuevos elementos o añadiendo más reglas

para mejorar el desempeño. En muchos casos, el control difuso puede usarse para mejorar sistemas de control existentes al añadir una capa extra de inteligencia al método de control actual...Los controladores difusos son muy simples conceptualmente, estos consisten de una etapa de entrada, una etapa de procesamiento y una etapa de salida. La etapa de entrada mapea el sensor u otras entradas a las funciones de membresía y valores de verdad apropiados; la etapa de procesamiento invoca cada regla apropiada y genera un resultado para cada una, luego combina los resultados de las reglas; finalmente la etapa de salida convierte el resultado combinado a un valor de salida de control específico."

1.11. CONTROLES HIBRIDOS.

El controlador híbrido toma ventaja de las características no lineales del control difuso y la precisión alrededor del punto de operación que garantiza un controlador PID estándar [2].

También existen controles híbridos en donde uno supervisa el otro para mejora el rendimiento o ambos funcionan en paralelo. Hay sistemas en donde el controlador difuso directamente ejerce el control. En aquellas aplicaciones donde pueden establecerse variables de comportamiento lineal y no lineales, el control híbrido paralelo es una alternativa de solución ya que el control difuso puede manejar las variables no lineales y el PID las lineales. Otra situación podría presentarse en donde hayan muchas variables intervinientes que no puedan relacionarse todas a través de una ecuación matemática. Aquí el control difuso manejaría todas las variables imposibles de modelar matemáticamente y el PID absorbería las variables matemáticamente modelables.

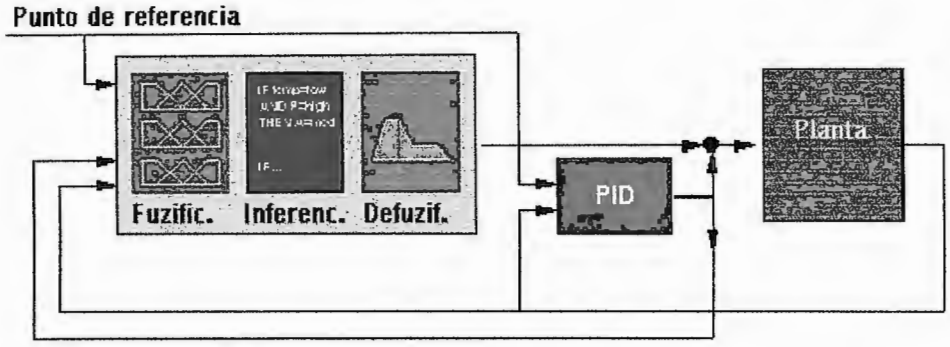


Figura 1. 13 Control difuso y PID funcionando en paralelo.

Otro sistema de control es utilizar un control difuso para controlar los parámetros de un PID. En este escenario se pretende que el controlador difuso procese todas las variables no lineales y genere un resultado que permita al PID trabajar en un rango lineal. Puede tomarse al control difuso como un linealizador del comportamiento de la planta[8].

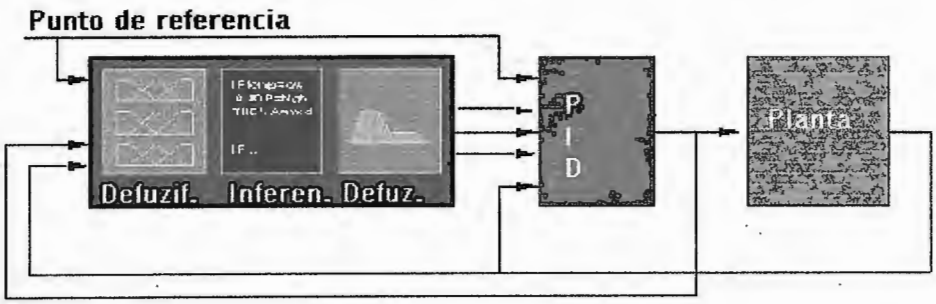


Figura 1. 14 Control difuso manejando parámetros de PID.

[8] Información recopilada de [2] y [18].

CAPITULO 2. PROTOTIPO DE CONTROL DIFUSO.

En este capítulo se explicará el diseño del prototipo de controlador difuso, haciéndose mayor énfasis en el funcionamiento del programa que lleva acabo técnicas de lógica difusa para ejecutar una función de control a través de reglas lingüísticas.

2.1. DISEÑO DE PROTOTIPO DEL CONTROLADOR DIFUSO.

Para la implementación del control difuso es necesario hardware y software. Con respecto al hardware, el prototipo del controlador inteligente lo conforman los siguientes componentes:

- Una computadora.
- El microcontrolador MC68HC11 de Motorola.
- Un acoplador de señales.

La computadora es el medio para la simulación y programación del microcontrolador. Es a través de la computadora que el usuario programa y se comunica con el microcontrolador. Una vez programado el microcontrolador, este ejecuta rutinas para interpretar las señales de los sensores y generar las señales de control. El acoplador de señales transforma las señales de los sensores a señales adecuadas para los convertidores de análogo a digital del microcontrolador, así como hacer la transformación inversa de las salidas digitales del microcontrolador para los actuadores de las plantas.

El controlador difuso consta de cuatro entradas y dos salidas. El software de lógica difusa se ha diseñado para ejecutarse en un microcontrolador MC68HC11, el cual incorpora puertos de comunicación, memoria RAM, convertidores analógicos y otros elementos para desarrollar la aplicación de control. También el sistema incluye una interface digital-analógica en cada una de sus salidas así como acondicionadores de señal para los sensores. Un esquema básico de la aplicación se muestra en la Figura 2.1. Cuando el usuario termine de introducir todos estos datos, el programa procede a generar los códigos necesarios para la base de conocimientos que inmediatamente se cargarán en la memoria del microcontrolador. Ya con los programas y datos cargados en el microcontrolador, la computadora puede desconectarse y el microcontrolador ya puede funcionar de forma independiente.

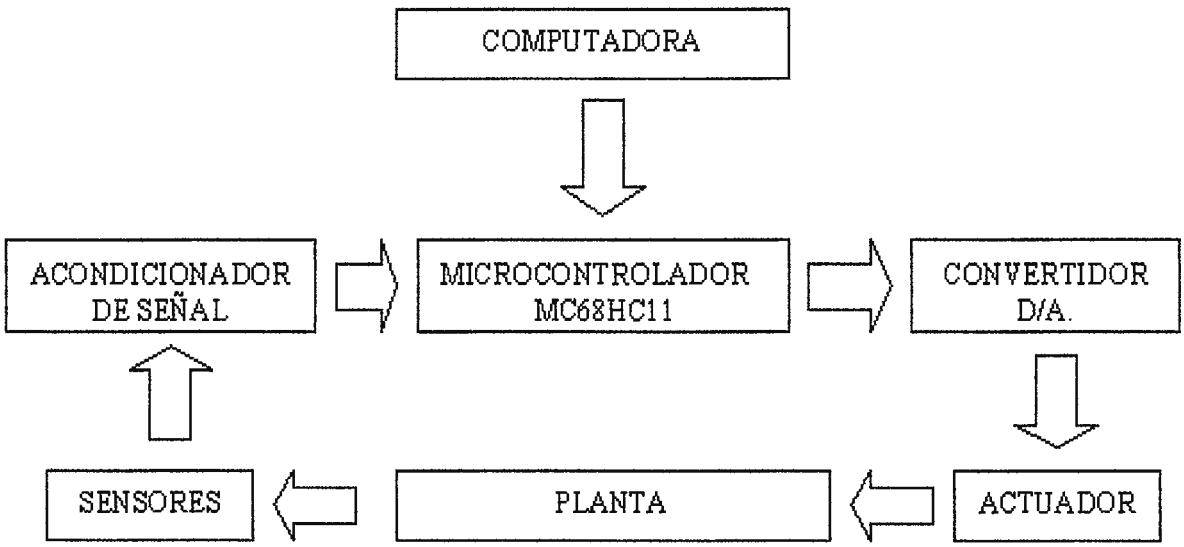


Figura 2. 1 Diagrama de bloques de prototipo.

El acondicionador de señal es utilizado para acoplar la información proveniente de los sensores a los puertos analógicos de entrada del microcontrolador. La función del acoplador es generar la suficiente impedancia para no cargar la planta y transformar los voltajes del sensor para que estén dentro del rango de conversión de los puertos. La señal transformada es procesada por el convertidor interno análogo digital del microcontrolador para

generar una cifra binaria equivalente de 8 bits. Como el microcontrolador sólo tiene puertos de salida digitales, es necesario añadir convertidores digitales analógicos para poder controlar los actuadores o etapas de potencia.

2.2. DISEÑO DEL NUCLEO DE INFERENCIA DIFUSO.

Lógica difusa es útil para un rango muy amplio de aplicaciones. Esta tesis se concentra en aplicaciones de control donde se programa el microcontrolador MC68HC11A8 para llevar a cabo algoritmos de la inferencia difusa. El control difuso se diseño basándose en la inferencia activa de reglas IF-THEN también conocida como inferencia modus ponens[1]. Un programa núcleo de inferencia difuso de propósito general, que incluyen los procesos de fusificación, evaluación de reglas y defusificación, se usa para obtener el control automático.

2.3. ESTRUCTURA GLOBAL DEL NUCLEO DE INFERENCIA DIFUSO.

La Figura 2.2 muestra el diagrama de bloques de un programa de un sistema de inferencia difuso. La mitad derecha de la Figura 2.2 muestra las tres partes del núcleo de inferencia difuso que procesa el programa en el tiempo de ejecución. La mitad izquierda de la Figura 2.2 muestra la estructura de datos que comprende la base de conocimiento para una aplicación de control específica. Por cada una de las tres partes del programa núcleo hay una estructura de datos correspondientes en la base de conocimiento. Las entradas del núcleo son valores que provienen de sensores que deben digitalizarse antes de pasar al núcleo de inferencia difuso. Las salidas del núcleo son valores digitales que deben

[1] Ver sección INFERENCIAS COMUNES en el apéndice A.

convertirse a valores que sean apropiados para manejar los componentes de la planta.

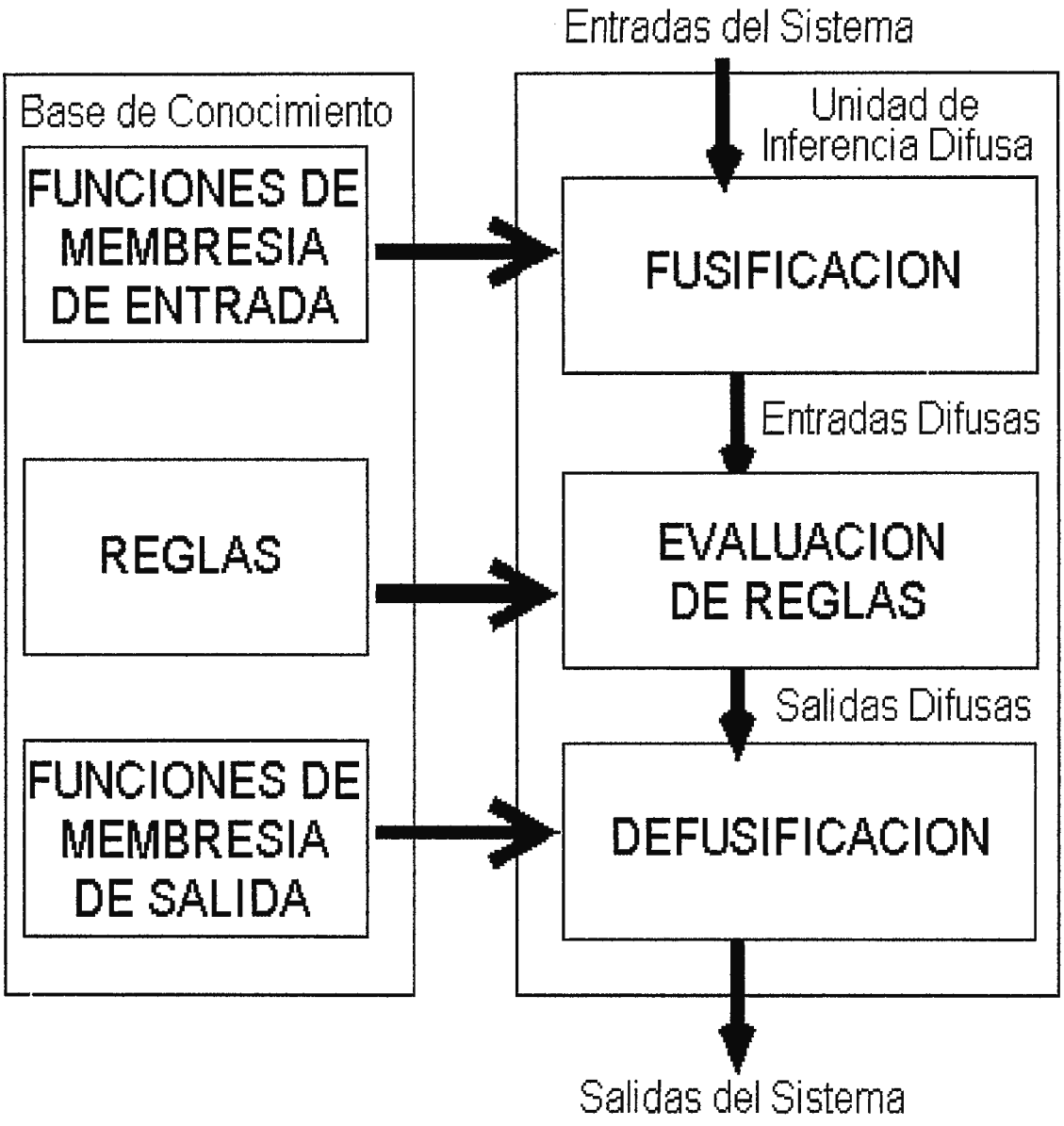


Figura 2. 2 Esquema del programa para inferencia difusa.

La base de conocimiento contiene la información de las reglas y las funciones de membresía. Para determinado sistema de entrada se aplica la información de la base de conocimiento y se calcula el valor de salida del sistema "crisp" usando la técnica de inferencia mínimo-máximo y el método de defusificación del centro de gravedad (usando funciones de salida singletons).

El paso de fusificación toma los valores actuales de los sensores de entrada del sistema, los compara contra las funciones de membresía de entrada de la base de conocimiento, y guarda valores de entrada difusos en una estructura de datos de la memoria RAM. Hay una entrada difusa por cada etiqueta lingüística de cada entrada. Para un sistema con dos entradas y siete etiquetas lingüísticas por entrada habrá catorce entradas difusas. El paso de evaluación de las reglas procesa una lista de reglas de la base de conocimiento usando la información de entradas difusas obtenidas en el paso previo de fusificación, generándose las salidas difusas que se almacenan en RAM. Para un sistema con una salida y siete etiquetas lingüísticas por salida, habrá siete salidas difusas. Defusificación usa las salidas difusas del paso anterior y las definiciones de funciones membresía de salida de la base de conocimiento para producir un valor de salida precisa para manipular la planta.

El núcleo de inferencia difuso que se usa en el programa del microcontrolador de esta tesis acepta cuatro entradas con ocho funciones de membresía por entrada, y también permite dos salidas con ocho singletons por salida. Una regla no tiene límites para el número de proposiciones atómicas del antecedente, pero el consecuente esta limitado hasta el momento a una proposición atómica. Considerando el núcleo de inferencia difuso como un conjunto, la meta es proporcionar un valor específico de salida (o conjunto de valores de salida) por cada combinación posible de valores de la entrada del sistema. Esta es realmente la definición justa de cualquier sistema de control. Si corrientemente se usa una metodología de control proporcional-íntegro-diferencial

(PID), se puede reemplazar simplemente el bloque PID del programa con un núcleo de inferencia difuso.

2.4. NUCLEO DE INFERENCIA DIFUSO.

A continuación se explica el funcionamiento del núcleo de inferencia difuso que se utiliza en programa que se carga al microcontrolador. El listado comentado del programa se muestra en el apéndice E. La explicación se divide en las siguientes partes:

- Declaración de Etiquetas y Espacios de Memoria.
- Obtención de las Entradas del Sistema.
- Fusificación.
- Evaluación de las Reglas.
- Defusificación.

2.4.1. DECLARACION DE ETIQUETAS Y ESPACIOS DE MEMORIA.

Esta es la parte inicial del programa. Aquí se indica, a través de etiquetas, las diferentes direcciones iniciales de los bloques de memoria reservada del microcontrolador en donde se almacenan los parámetros que definen: las variables lingüísticas de las entradas y salidas junto con las respectivas funciones de membresía, las reglas y los puertos de entrada y salida. También se le indica al microcontrolador, a través de la instrucción ORG, donde se almacenará el programa. Para la presente aplicación se le indica al microcontrolador que el programa sea ensamblado de forma absoluta a partir de la dirección $C000.

También se procede a cargar datos en direcciones específicas del microcontrolador para inicializar los puertos de entradas y salidas.

2.4.2. OBTENCION DE LAS ENTRADAS DEL SISTEMA.

El primer paso antes de la fusificación es que el núcleo de inferencia difuso debe obtener los valores de entrada "crisp" actuales. La etiqueta "ENTRADAS_DEL_SISTEMA" es la localización de la dirección de inicio para las entradas "crisp" del sistema. Las entradas "crisp" del sistema son los dispositivos que regulan los valores de operación deseados y los sensores de las plantas a controlar, que son conectadas a los convertidores de análogo a digital del microcontrolador en el puerto E, cuyos registros de datos están ubicados en las direcciones $1031, $1032, $1033 y $1034, respectivamente para cada uno de los cuatro canales de conversión. Dichos convertidores realizan su tarea de conversión en cada ciclo de actualización de las salidas.

Como ya se había mencionado el programa es capaz de aceptar cuatro entradas al mismo tiempo. Los valores de entrada que se obtiene de los convertidores son colocados en la pila para su futuro procesamiento dentro del programa, de la siguiente forma:

PUNTERO DE PILA(SP)	ENTRADA #3
-1	ENTRADA #2
-1	ENTRADA #1
-1	ENTRADA #0

A continuación de la obtención de las entradas "crisp" son calculadas las entradas difusas, en el paso de la fusificación, que son los valores

correspondientes a los grados de pertenencia de las entradas del sistema para cada una de sus correspondientes funciones de membresía.

2.4.3. FUSIFICACION.

Las entradas del sistema entran a este bloque como un valor hexadecimal de 8-bit (por medio de los convertidores de análogo a digital), que representa la lectura presente de un sensor del sistema o de un dispositivo que regula el nivel de operación para una salida determinada. Por cada entrada puede haber hasta ocho etiquetas lingüísticas y cada una tiene una definición de su función de membresía correspondiente en la base de conocimiento. La Figura 2.3 muestra una función de membresía de entrada trapezoidal para "CALUROSO" que es una etiqueta asociada con la "TEMPERATURA" de entrada del sistema. La entrada presente corresponde a una posición en el eje "x" de la función de membresía. Un parte del programa calcula la intersección en el eje "y", y la almacena en la memoria RAM como el valor de verdad o grado de pertenencia para la etiqueta lingüística respectiva, dicho valor es una de las entradas difusas. Esto se hace para todas las etiquetas de una entrada. Entonces se lee otra entrada y se repite el lazo para cada etiqueta de esa entrada.

En la Figura 2.3 los ejes han sido etiquetados con valores que son más apropiados para el sistema basado en un microcontrolador. El valor de entrada se ha adaptado para encajar en una escala de $00 a $FF y el eje "y" ahora corre de $00 (falso) a $FF (completamente verdadero). En la base de conocimiento cada función de membresía es definida por cuatro valores de 8-bit, la posición en "x" de dos puntos y dos valores de pendientes (ver Figura 2.3). El primer punto (PUNTO 1), define el punto final izquierdo de la base. La primer pendiente (PNTE. 1), define la pendiente del lado de inclinación izquierdo del trapezoide. El segundo punto (PUNTO 2), define el extremo derecho de la cima del trapezoide y la segunda

pendiente (PNTE. 2), define la pendiente del lado de inclinación derecho del trapezoide. Se divide el eje "x" entero en tres segmentos. El segmento 0 comprende todo lo que está a la izquierda del trapezoide. El segmento 1 comienza en el PUNTO 1 e incluye cualquier punto en el lado de inclinación izquierdo o la cima del trapezoide. El segmento 2 comienza en el PUNTO 2 e incluye cualquier punto en el lado de inclinación derecho o a la derecha del trapezoide.

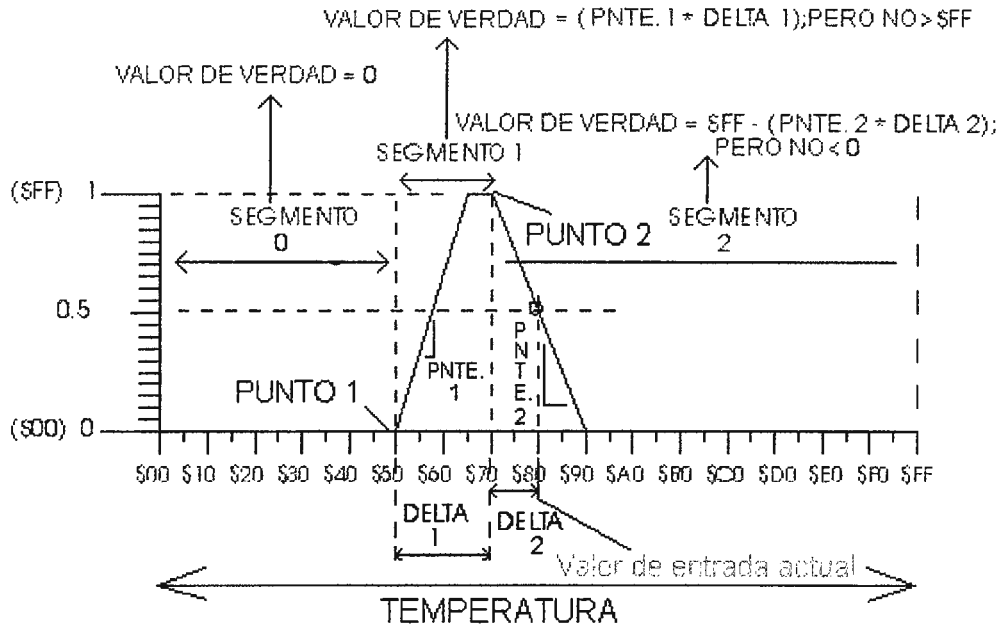


Figura 2. 3 Parámetros de función de membresía.

Se denota a continuación que los bytes, que representan las funciones de membresía de entrada que usa el núcleo de inferencia difuso, tienen las siguientes características:

BYTE 1(PUNTO 1): Puede estar comprendido entre $00 y $FF, y se asume que esta colocado sobre el eje x (grado de membresía cero).

BYTE 2(PNTE. 1): Puede estar comprendida entre $00 y $FF, y se asume que es positiva.

BYTE 3(PUNTO 2): Puede estar comprendido entre $00 y $FF, y se asume que tiene un grado de membresía uno.

BYTE 4(PNTE. 2): Puede estar comprendida entre $00 y $FF, y se asume que es negativa.

Nota: Un caso especial existe cuando una pendiente es igual a $00, esto podría ordinariamente representar una línea horizontal a lo largo del eje x, lo cual no es muy útil para este caso; En lugar de ello esto se reserva para representar pendientes infinitas(líneas verticales).

El núcleo de inferencia difuso tiene una localidad especifica en la memoria RAM para localizar toda la información en la base de conocimiento las funciones de membresía de entrada, la cual se declara en el programa por la etiqueta FUNCIONES_DE_MEMBRESIA en la dirección $C200, y dicha información es ordenada de la siguiente manera:

```
$C200     PUNTO 1          ;función de membresía 0, entrada 0
$C201     PENDIENTE 1
$C202     PUNTO 2
$C203     PENDIENTE 2
   -                        ;siguientes funciones de membresía
   -                        ; a la derecha, entrada 0
   -
$C21C     PUNTO 1          ;función de membresía 7, entrada 0
$C21D     PENDIENTE 1
$C21E     PUNTO 2
$C21F     PENDIENTE 2
$C220     PUNTO 1          ; función de membresía 0, entrada 1
$C221     PENDIENTE 1
$C222     PUNTO 2
$C223     PENDIENTE 2
   -                        ;así sucesivamente
   -
   -                        ;hasta
$C27C     PUNTO 1          ;función de membresía 7, entrada 3
$C27D     PENDIENTE 1
$C27E     PUNTO 2
$C27F     PENDIENTE 2
```

Para obtener las entradas difusas en el programa de inferencia se calculan dos valores de delta. Delta 1 es la diferencia entre el valor de entrada presente y el

PUNTO 1, y DELTA 2 es la diferencia entre al valor de entrada y el PUNTO 2. En segmento 0 ambos valores del delta son negativos y el valor de la verdad es cero, en segmento 1 el valor de la verdad es DELTA 1 por la PNTE. 1 (sujeto a un límite superior de $FF), y en el segmento 2 el valor de la verdad es $FF menos la cantidad DELTA 2 por la PNTE. 2 (sujeto a un límite inferior de cero); ver el siguiente ejemplo de la Figura 2.4, 85° ($B4) esta en el segmento 2, por lo tanto el valor de entrada difuso es:

$FF – ($B4 - $AE) * $6
= $FF – ($07) * $6
= $FF - $2A
=$D5

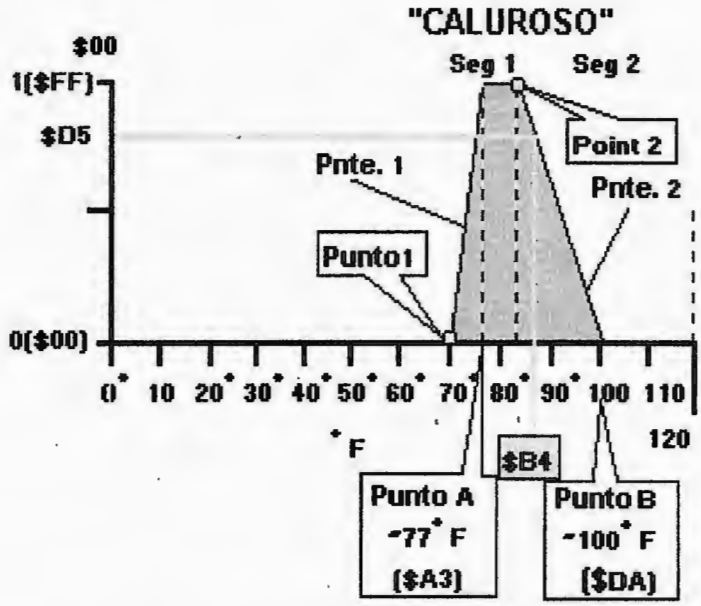


Figura 2. 4 Ilustración del proceso de fusificación.

Al finalizar el proceso de fusificación, todas las entradas difusas quedan ubicadas en localidades contiguas de memoria la memoria RAM a partir de la dirección de la etiqueta ENTRADAS _DIFUSAS, como se muestra en la Figura 2.5.

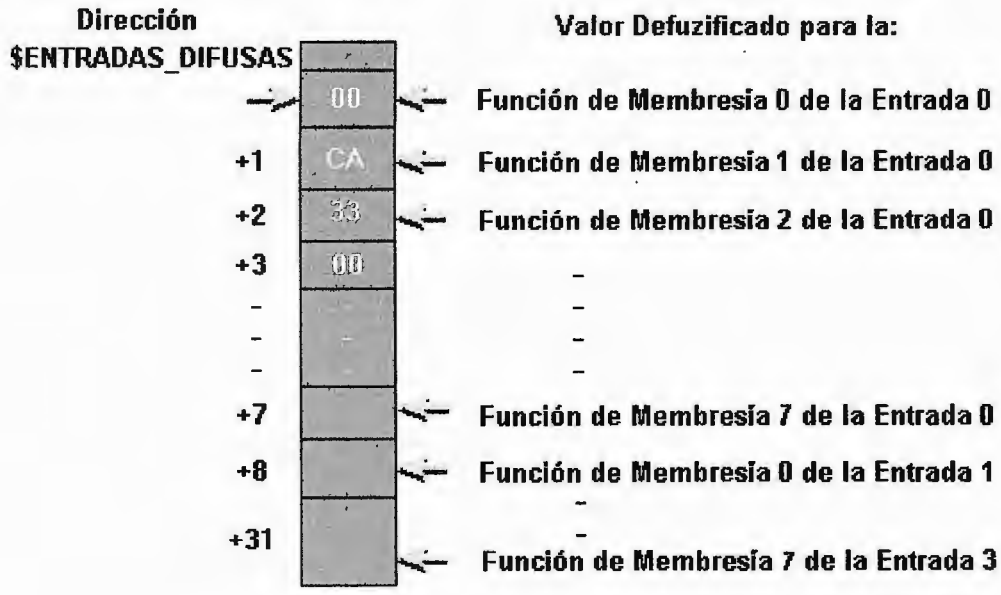


Figura 2. 5 Esquema de RAM donde se almacenan resultados de fusificación.

2.4.4. EVALUACION DE LAS REGLAS.

Se usa una estructura de reglas relativamente simple para el programa del núcleo de inferencia difuso en el microcontrolador. Una regla tal como "IF(Sí) la TEMPERATURA es CALUROSA AND(Y) la PRESION es BAJA THEN(Entonces) fijar el CALOR a ALTO," tiene dos antecedentes (partes IF) y un consecuente (parte THEN). Un sistema de control completo tendría una lista de muchas reglas similares, que juntas describen la respuesta del sistema de control. El orden de reglas en esta lista no afecta la respuesta del sistema. Se supone que se evalúan todas las reglas simultáneamente, pero en un sistema basado en un programa de software ellas se procesan de manera secuencial.

Cada expresión del antecedente tal como "TEMPERATURA es CALUROSA" se puede reemplazar por el valor presente de la entrada difusa correspondiente (un valor situado en RAM que se determinó durante la fusificación). "IF" introduce los antecedentes que se requieren para hacer la regla

verdadera e indica el inicio de una regla. "THEN" separa las condiciones antecedentes de las acciones consecuentes. Antecedentes son conectados por el operador "AND" y hay un operador "OR" implícito entre reglas sucesivas. Las reglas pueden ser guardadas en la base de conocimiento a partir de la dirección asignada por la etiqueta REGLAS, la cual es la dirección $C290.

En el programa para el núcleo de inferencia difuso cada antecedente de regla es representado por un byte. Dos bits se usan para designar una de las cuatro entradas (XX) y tres bits se utilizan para designar una de las ocho funciones de membresía (AAA), como se muestra en la Figura 2.6.

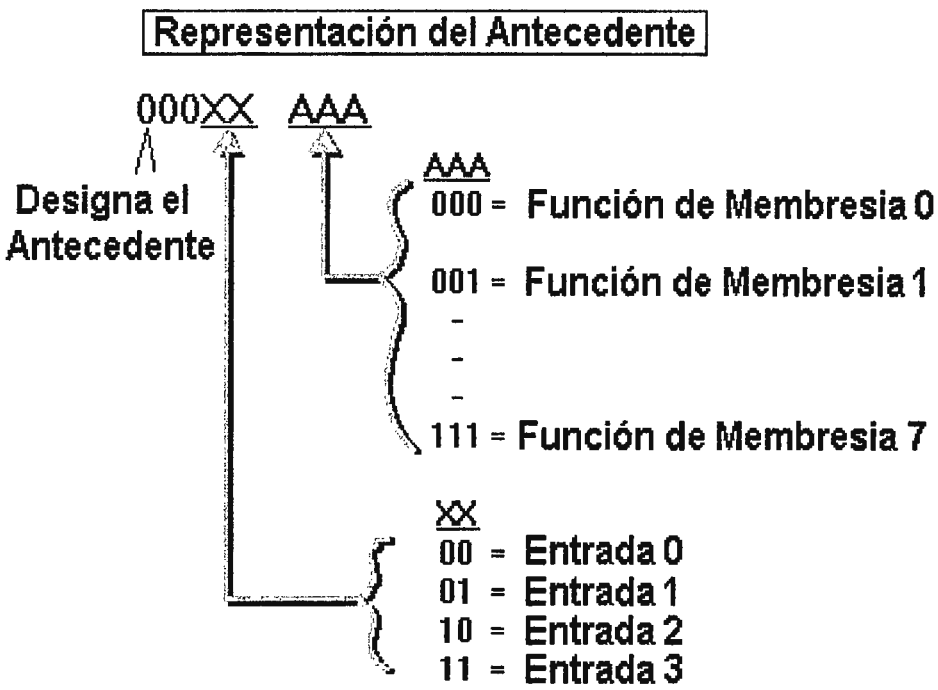


Figura 2. 6 Codificación de antecedentes.

Además, cada consecuente de regla también es representado por un byte. Un bit designa si pertenece a la salida 0 ó a la salida 1(X) y tres designan a cual función de membresía aplican (AAA), como se muestra en la Figura 2.7. El bit más significativo de cada consecuente es colocado a "uno" para diferenciarlo del antecedente.

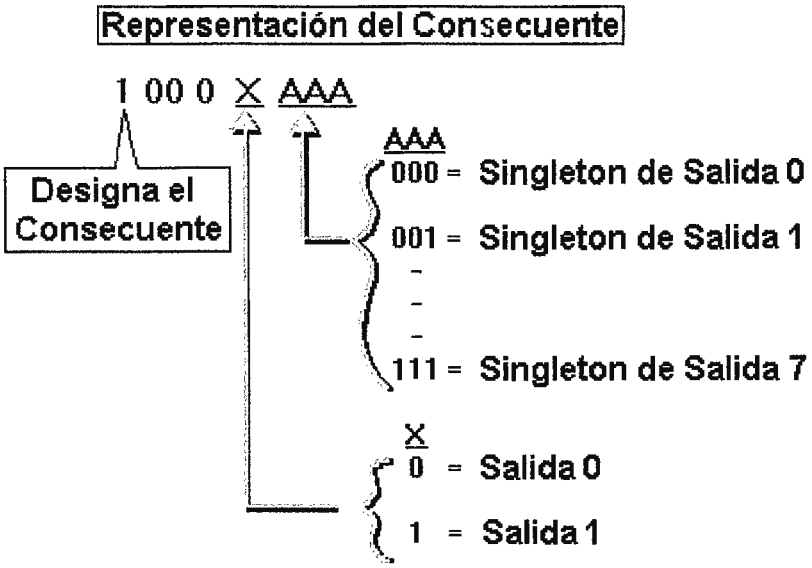


Figura 2. 7 Codificación de consecuente.

Las reglas son construidas colocando la representación apropiada en secuencia ascendente de bytes de memoria. El núcleo de inferencia difuso reconoce cuando finaliza una regla por la notación de "uno" en el bit más significativo de las acciones consecuentes. Cualquier número de reglas puede ser acomodado. El final de las reglas se denota por la colocación de un byte conteniendo el número hexadecimal $FF. La Figura 2.8 muestra un ejemplo de como una regla es creada. Al final del proceso la representación de las reglas debe ser codificada como sigue:

```
$C290      03        ;primera regla
$C291      08
$C292      82
$C293      0X        ;segunda regla
  -        -
  -        -
  -        -         ;siguientes reglas
  -        -
  -        -
  -        -         ;última regla
  -        -
  -        -
  -        FF        ;marcador de fin de regla
```

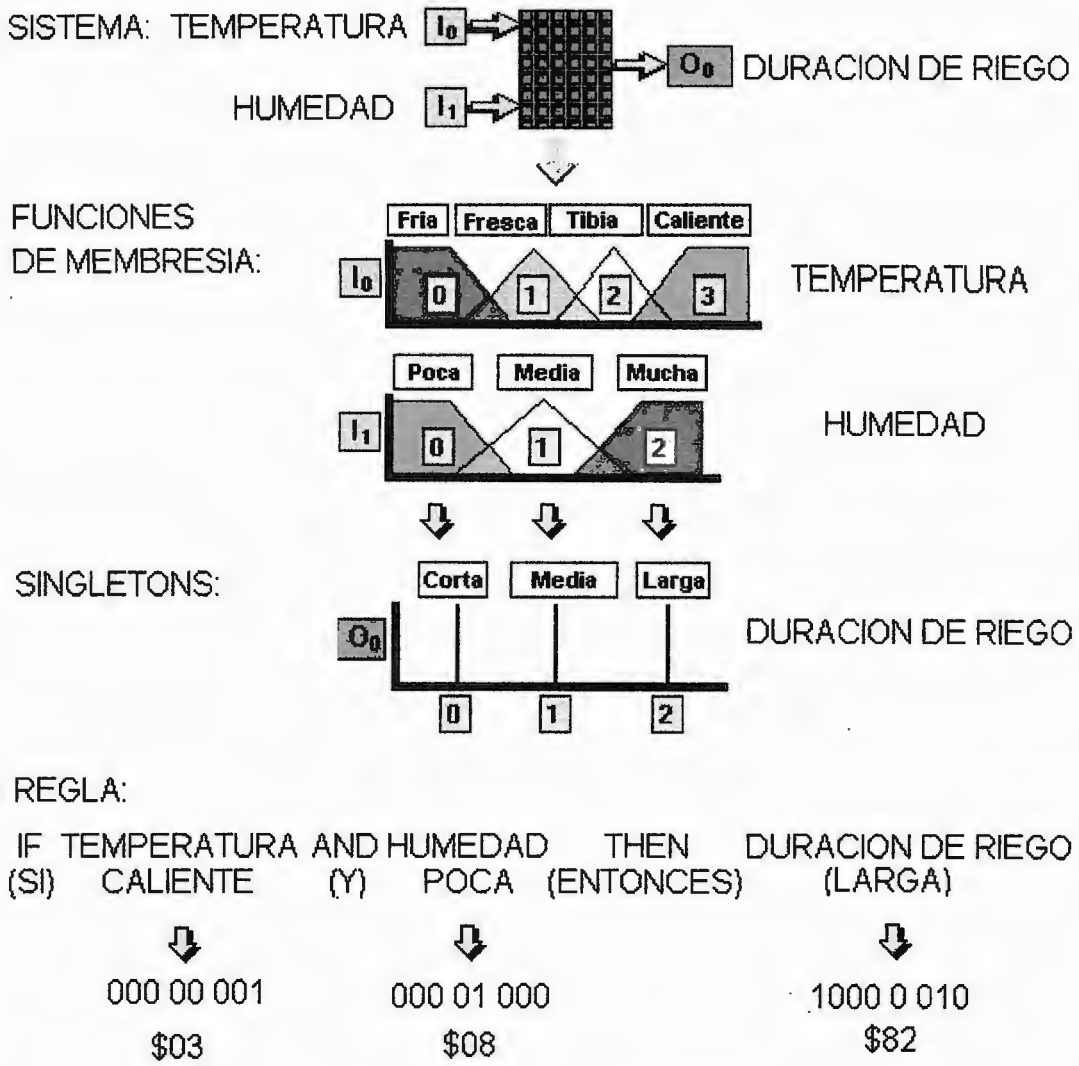
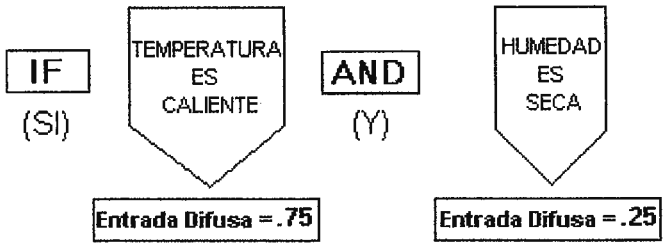


Figura 2. 8 Codificación de reglas.

El núcleo de inferencia difuso utiliza la evaluación de reglas mínimo-máximo (la cual trabaja bien para muchas aplicaciones del control desarrolladas). Debido a que hay dos salidas con hasta ocho funciones de membresía singletons por salida, un total de dieciséis posibles salidas difusas existen. Por lo tanto, la fuerza de las reglas será calculada y aplicada para dieciséis posibles acciones consecuentes. La etiqueta SALIDAS_DIFUSAS designa la dirección de inicio para que sean colocados los máximos valores de las fuerzas de regla para cada una de las dieciséis acciones consecuentes ó salidas difusas. Inicialmente el programa limpia todas las salidas difusas. El siguiente paso es encontrar la fuerza para la primera

regla, esta será el valor mínimo valor de las entradas difusas para el antecedente de la regla, ver ejemplo de la Figura 2.9.



Fuerza de Regla = Min (.75, .25) = .25

Figura 2. 9 Determinación de fuerza de regla.

Las entradas difusas son fracciones de peso binario entre $00 y $FF con el punto decimal asumido a estar a la izquierda del bit más significativo. Por ejemplo:

1 000 0000 = .5
1 100 0000 = .75

Es muy importante recordar la manera en que las reglas son estructuradas y la localidad de las entradas difusas. Para encontrar el grado de membresía de cada antecedente, simplemente se busca el valor de verdad en la localidad que resulta de la siguiente expresión:

$ENTRADAS_DIFUSAS + ENTRADA # + FUNCION DE MEMBRESIA #

O

$ENTRADAS_DIFUSAS + $000XXAAA

Por ejemplo, si el antecedente se refiere a la entrada número uno y su función de membresía número cero (designado $08), el grado de membresía para este antecedente esta localizado en la dirección $ENTRADAS_DIFUSAS + 1*8 + 0 = $ENTRADAS_DIFUSAS + 8(que es asignado como XX en la Figura 2.10).

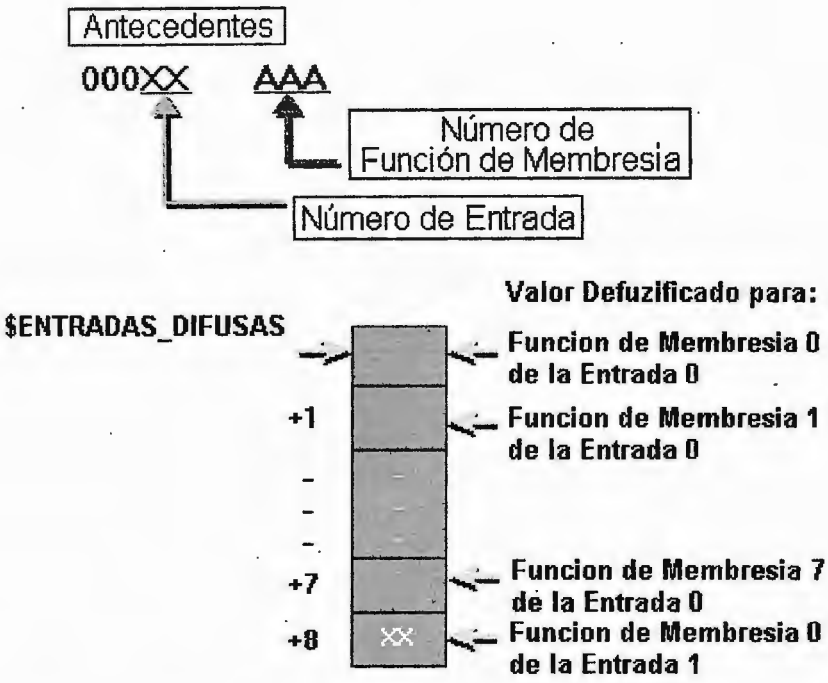


Figura 2. 10 Desplazamientos de memoria para procesamiento de reglas.

Una vez el grado de membresía es determinado para cada antecedente, solamente el valor mínimo es retenido. Por ejemplo, la regla "IF(Sí) A AND(Y) B THEN(Entonces) C" tiene tres segmentos. Cada antecedente y cada consecuente esta representado por un byte el cual puede estar en términos de un segmento de regla. Cuando un segmento de regla es encontrado el cual comienza con $8X, entonces el núcleo de inferencia difuso sabe que ya no hay más antecedentes en esta regla y el mínimo valor es calculado, que es la fuerza de regla para esta regla.

El paso final de la evaluación de reglas es colocar la información de salida en la regla. En el ejemplo previo, la acción de salida llamada para la salida uno, es el singleton dos. Usando la dirección:

$SALIDAS_DIFUSAS + SALIDA # * 8 + SINGLETON # =
$SALIDAS_DIFUSAS + 1 * 8 + 2 =
$SALIDAS_DIFUSAS + 10 (que es asignado XX en la Figura 2.11)

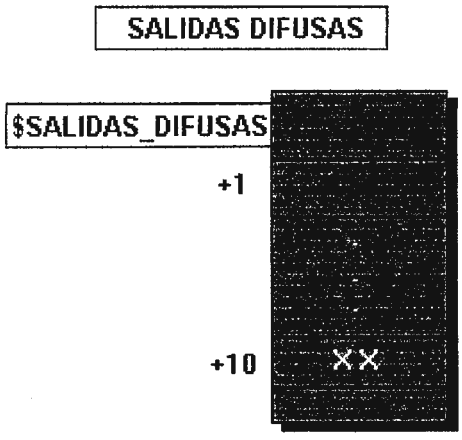


Figura 2. 11 Bloque de memoria para almacenar salidas difusas.

El núcleo de inferencia difuso determina si la fuerza de la regla determinada excede el valor ya almacenado en esta localidad. Si la nueva fuerza de regla excede el valor almacenado, la nueva fuerza de regla será escrita en esta localidad. Por lo tanto la máxima fuerza de regla para cada acción de salida será registrada en el bloque de almacenamiento para las salidas difusas, en la conclusión de toda la evaluación de reglas. Esto completa la inferencia mínimo-máximo.

En resumen la evaluación de las reglas se lleva cabo de la siguiente manera, se apunta hacia el inicio de la lista de reglas, y se procesan las reglas sucesivas según el algoritmo siguiente. Encuentra el antecedente más pequeño (mínimo) de una regla y guarda este resultado a cada consecuente de la regla a menos que la salida consecuente difusa ya sea la más grande (máximo). Repite hasta que alcanzar el marcador de fin de reglas ($FF). Los resultados de la evaluación de la regla serán una tabla de valores de salidas difusas en RAM a partir de la dirección de la etiqueta SALIDAS_DIFUSAS (como se ilustra en la Figura 2.11). Se puede pensar en las salidas difusas como el resultado intermedio de considerar todas las reglas que gobiernan el sistema. Las salidas difusas necesitarán ser procesadas más allá para obtener un resultado compuesto simple para cada salida del sistema (en el paso de defusificación).

2.4.5. DEFUSIFICACION.

Para el programa utilizado en esta tesis, se usan singletons para definir las funciones de membresía de salida. Se define una función de membresía singleton por un solo valor de 8-bit en la base de conocimiento (que representa una posición en el eje x). La Figura 2.12 muestra un ejemplo para la función de membresía de salida singleton "medio".

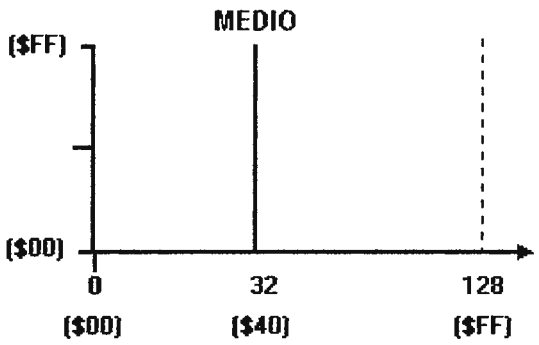


Figura 2. 12 Función de membresía singletons.

La defusificación requiere la posición en "x" de las funciones singletons, que se colocan a partir de la dirección de la etiqueta SINGLETONS que es $C280, y su correspondiente valor de salida difusa(ver ejemplo de la Figura 2.13). Las salidas difusas asociadas proporcionan un peso por cada etiqueta de salida del sistema. El paso final de la defusificación es determinar la acción de salida crisp asociada a cada salida del sistema. El núcleo de inferencia difuso utiliza el método de defusificación del cálculo de centro de gravedad, el cual usa la fórmula Ec. 2.5 para determinar una sola posición en el eje "x" como la salida final del sistema:

$$\frac{\sum_{i=1}^{n} Fi \times Si}{\sum_{i=1}^{n} Fi} \qquad \text{Ec. 2.1}$$

Donde n es el número de salidas difusas asociadas con la salida del sistema, Fi es un peso (valor de la salida difusa), y Si es la función de membresía de posición singleton.

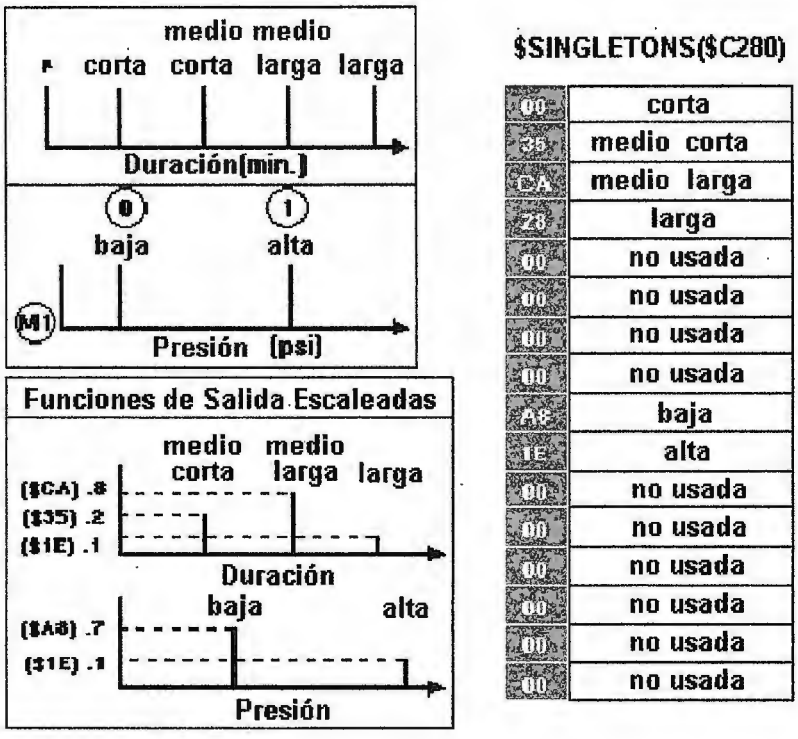


Figura 2. 13 Funciones de membresía almacenadas en memoria.

El resultado de este cálculo es la posición en el eje "x" de la acción de salida deseada. Fi y Si son valores de 8-bit y n es típicamente 8 (número máximo de funciones de membresía de salida) ó menos. Esto hace al numerador de un valor de 19-bit y el denominador de un valor de 11-bit. Porque el numerador y denominador no son valores independientes, se sabe que el resultado es un valor de 8-bit. Como un ejemplo, para la salida del sistema Presión (ver Figura 2.14), su salida "crisp" es calculada de la siguiente manera:

$$\frac{.7 \times 30\ psi \ + \ .1 \times 60\ psi}{.7 \ + \ .1} = \frac{21 \ + \ 6}{.8} = 33.75$$

El resultado de esta operación es colocado en una de las salidas del sistema, que se encuentran en los puertos B y C del microcontrolador (que son de 8 bits). Las dos salidas del sistema o salidas "crisp" se encuentran situadas a partir de la dirección que se asigna a la etiqueta SALIDAS_DEL_SISTEMA, que es $1003 (que corresponde al registro de datos del puerto C).

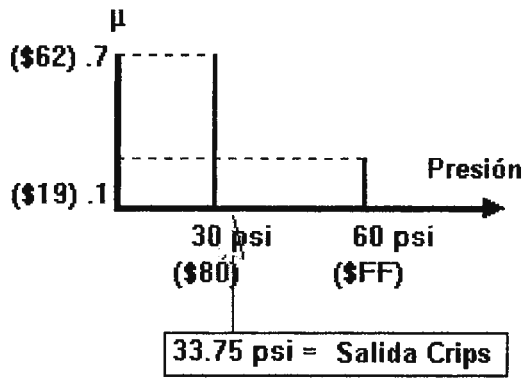


Figura 2. 14 Cálculo de salidas "crisp".

El proceso de defusificación es realizado dos veces durante un ciclo de inferencia difuso, una vez para cada una de las dos salidas del sistema. En el primer proceso de defusificación se calcula la salida crisp para la salida 1, que es el puerto C, y en el segundo proceso se ejecuta para la salida 2, que es el puerto B (que corresponde al registro de datos del puerto B, cuya dirección es $1004).

2.5. CONSIDERACIONES DE TIEMPO.

La cantidad de tiempo requerida para convertir una entrada "crisp" en una salida "crisp" es un parámetro critico a considerar en el diseño de controladores difusos. En el diseño del controlador difuso de esta tesis se ejecuta dicha conversión continuamente para actualizar el valor de salida según va cambiando

la señal de entrada. Varias tareas son realizadas en la actualización, las cuales cada una de ellas consumen cantidades diferentes de tiempo.

Los tiempos de ejecución son calculados a partir de la frecuencia de reloj de la tarjeta de evaluación del microcontrolador 68HC11, la cual es de 2 MHz. La conversión analógica a digital de las entradas a defusificar consume 176 ciclos de reloj, lo cual produce un tiempo de ejecución de 88 microsegundos. La fusificación la cual comprende de la comparación de cada una de las entradas contra sus ocho funciones de membresía se realiza en 2453 ciclos de reloj, lo cual produce un tiempo de ejecución de 12.26 milisegundos. El tiempo de ejecución de la evaluación de reglas puede variar de acuerdo con la cantidad de reglas que el usuario decida utilizar (considerando que solo se usen dos antecedentes y un consecuente por regla), por ejemplo, el uso de una regla consume 70 microsegundos, dos reglas consumen 121.5 microsegundos, tres reglas consumen 178 microsegundos, seis reglas consumen 340 microsegundos, ocho reglas consumen 448 microsegundos, etc. La última tarea la cual es la defusificación, comprende de 8 singletons por salida y tiene un consumo de 1452 ciclos de reloj los cuales produce un tiempo de ejecución de 726 microsegundos.

En general todo el ciclo completo cuando se utilizan 8 reglas con dos antecedente y un consecuente para la defusificación, se ejecuta en 2.5 milisegundos aproximadamente.

CAPITULO 3. DISEÑO DE INTERFACES DE HARDWARE.

Finalizando la lectura del presente capítulo, el lector comprenderá el diseño de las interfaces de hardware para el microcontrolador MC68HC11. A continuación se encontrarán diagramas, fórmulas y explicaciones de los diferentes circuitos que conforman las interfaces del acoplador de señales.

3.1. GENERALIDADES.

El prototipo del controlador difuso es capaz de adaptarse para controlar cualquiera de las tres plantas del laboratorio: nivel, temperatura, velocidad ó posición. El componente medular del prototipo lo forma el microcontrolador MC68HC11 de Motorola. Este viene integrado en la tarjeta evaluadora EVB11.

Cada planta posee sus propios sensores y módulos de control que funcionan con voltajes variados. Como el microcontrolador trabaja con niveles de voltaje entre 0 y +5 VDC, fue necesario diseñar y construir interfaces que permitieran a los diferentes sensores y módulos de control interactuar con el controlador difuso (ver Figura 3.1).

Para las variables de entrada del controlador difuso se aprovecharon los convertidores analógicos-digitales internos del microcontrolador. Para las salidas digitales del microcontrolador, fue necesario diseñar y construir convertidores digitales-analógicos para generar las señales de voltajes equivalentes que gobernaran los actuadores de los módulos de control de cada planta. Tanto para las señales de entrada y salida del microcontrolador, los convertidores también

incluyen etapas de amplificación/atenuación y voltajes de desplazamiento (offset). La aplicación de voltajes de desplazamiento en los convertidores de entrada permite manipular el rango de voltajes del microcontrolador para que el control difuso pueda interpretar valores negativos y positivos. Para los convertidores de salida, el voltaje de desplazamiento junto con las etapas de amplificación permiten generar voltajes positivos y negativos necesarios para algunos actuadores.

A través del programa de control difuso, al microcontrolador se le ha dado la facultad de procesar cuatro señales de entrada y dos de salida. Las cuatro señales analógicas de entrada son introducidas al microcontrolador a través de los pines PE4, PE5, PE6, PE7 del puerto E programado previamente como entrada. Las señales digitales de salida se envían por los puertos C y B respectivamente. Cada puerto de salida genera códigos binarios de ocho bits.

Para los circuitos acopladores de señales de entrada se utilizaron amplificadores operacionales LF347 conjuntamente con resistencias variables y fijas. El amplificador operacional LF347 posee compensación interna y entradas JFET que proveen una muy alta impedancia de entrada con lo cual se evita cargar a los otros circuitos; además, tiene otras características tales como: bajo costo, alta velocidad, gran ancho de banda, alto slew rate, bajo ruido, etc. (ver hoja de datos eléctricos en Anexos). Para las salidas del microcontrolador se diseñaron y construyeron circuitos convertidores D/A del tipo escalera R-2R, usándose resistencias con tolerancia de 2%.

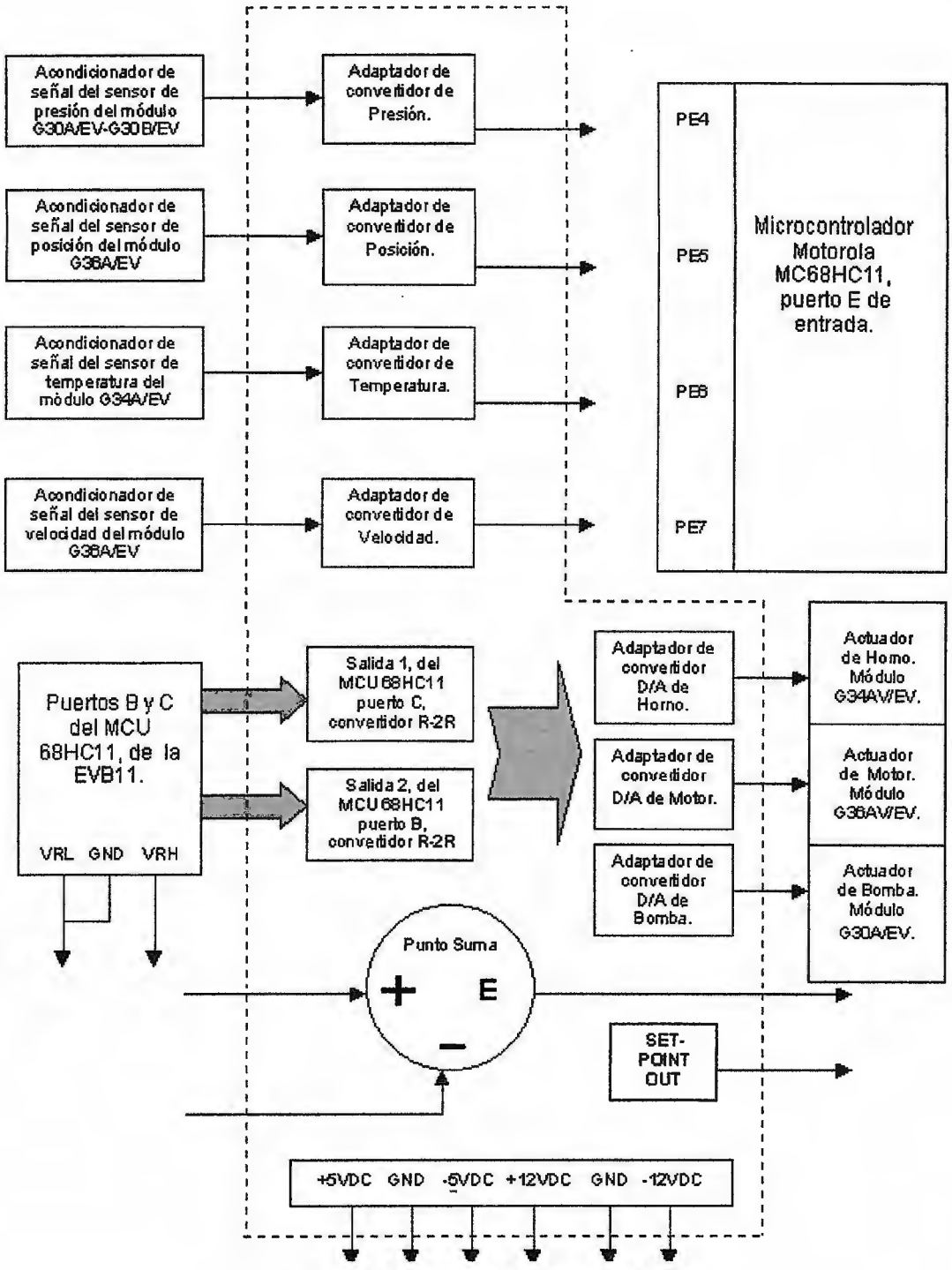


Figura 3.1 Diagrama de bloques de circuitos interfaces y convertidores D/A.

3.2. CONVERTIDOR DE SEÑAL PARA EL SENSOR DE PRESION.

Como se mencionó en la sección anterior, los voltajes producidos por los acondicionadores de señal de los módulos de control de cada planta deben convertirse a voltajes equivalentes que este dentro del rango de 0 a +5VDC. Los factores de amplificación/atenuación y voltajes de desplazamiento están subordinados a los voltajes producidos por los acondicionadores de señal de los sensores. En todos los convertidores, la última etapa brinda alta impedancia para no cargar las entradas del microcontrolador. Debe recordarse que las salidas de los convertidores de entrada se conectan a los pines del puerto E según el orden establecido por las variables lingüísticas de entrada declaradas en el modelo difuso.

El sensor de nivel es un transductor extenciométrico que mide la presión ejercida por la columna de líquido acumulada en el tanque. La señal del transductor pasa al acondicionador de señales del módulo de control donde se produce un voltaje de salida que varía entre 0 y +500 mV. El convertidor (ver Figura 3.2) toma este voltaje y le aplica un factor de amplificación de 100. La etapa IC-2A del convertidor provee una ganancia de –10. La etapa IC-2B invierte la polaridad de la señal resultante de la etapa IC-2A y alta impedancia. La señal resultante de salida del convertidor que varía entre 0 y +5 VDC equivale a la lectura de nivel o volumen entre 0 y 500 mililitros (0-5 litros).

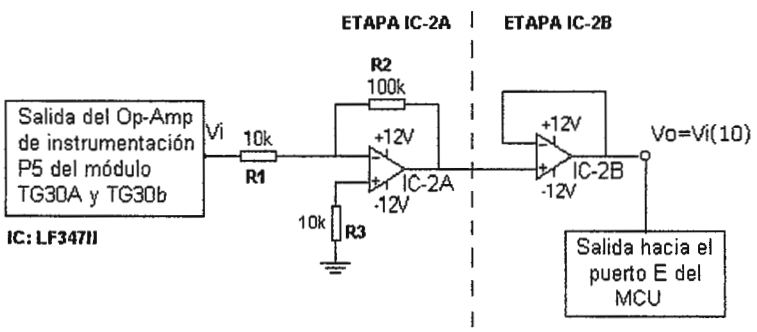


Figura 3.2 Convertidor de nivel del acondicionador de presión.

3.3. CONVERTIDOR DE SEÑAL PARA EL SENSOR DE POSICION.

El sensor de posición es una resistencia variable cuyo cursor esta conectado al eje del motor. La variación del voltaje del sensor es ±8VDC, donde -8VDC equivale a una posición de 0° y +8VDC a 360°. En la etapa IC-3A (ver Figura 3.3), se aplica una atenuación de -0.3125 a la señal del sensor del acondicionador del módulo de control. En la etapa IC-3B se aplica un voltaje de desplazamiento de 2.5. Por último, en la etapa IC-3C se invierte la polaridad de la señal y se produce alta impedancia para no cargar otros circuitos que se conecten.

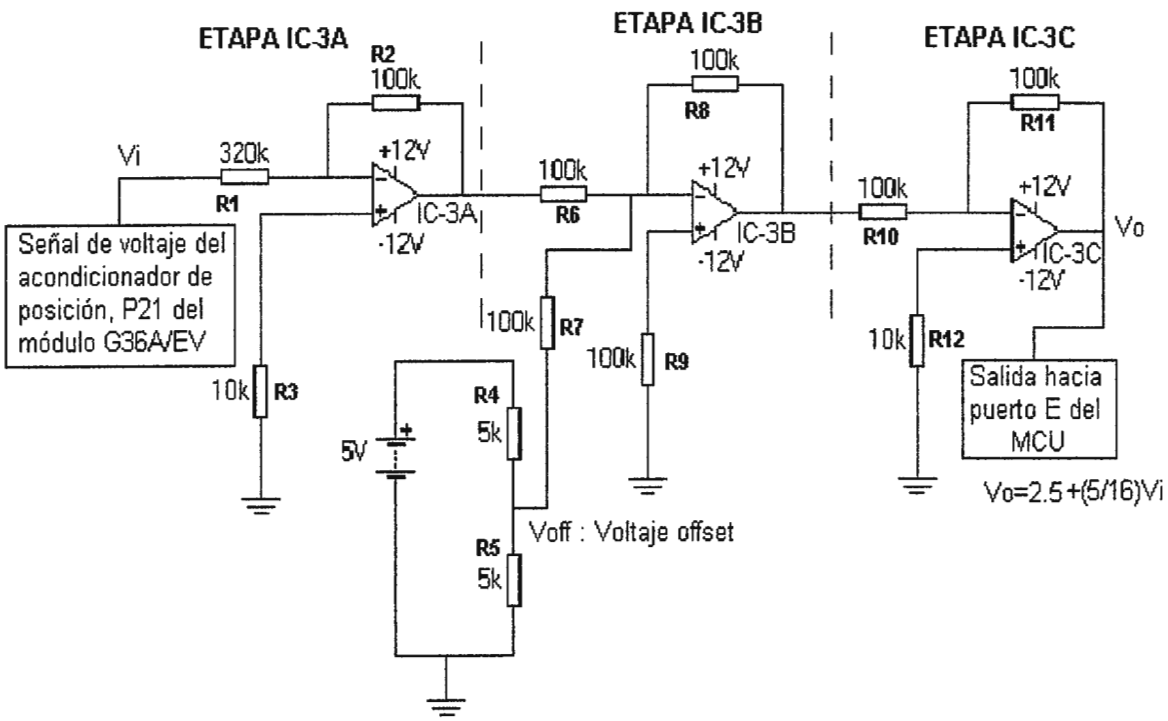


Figura 3.3 Convertidor de nivel del acondicionador de posición.

3.4. CONVERTIDOR DE SEÑAL PARA EL SENSOR DE VELOCIDAD.

El sensor de velocidad es un tacómetro. El rango de voltaje de variación del sensor es ±8VDC. La polaridad del voltaje indica el sentido de giro. El convertidor fue diseñado solamente para el rango de 0 a +8VDC. La etapa IC-4A aplica un factor de atenuación de –5/8. La etapa IC-4B invierta la polaridad de la salida de la etapa IC-4A y brinda alta impedancia. El rango de voltaje de salida de 0 a +5VDC corresponde al rango de velocidad del motor de 0 a 4000 rpm.

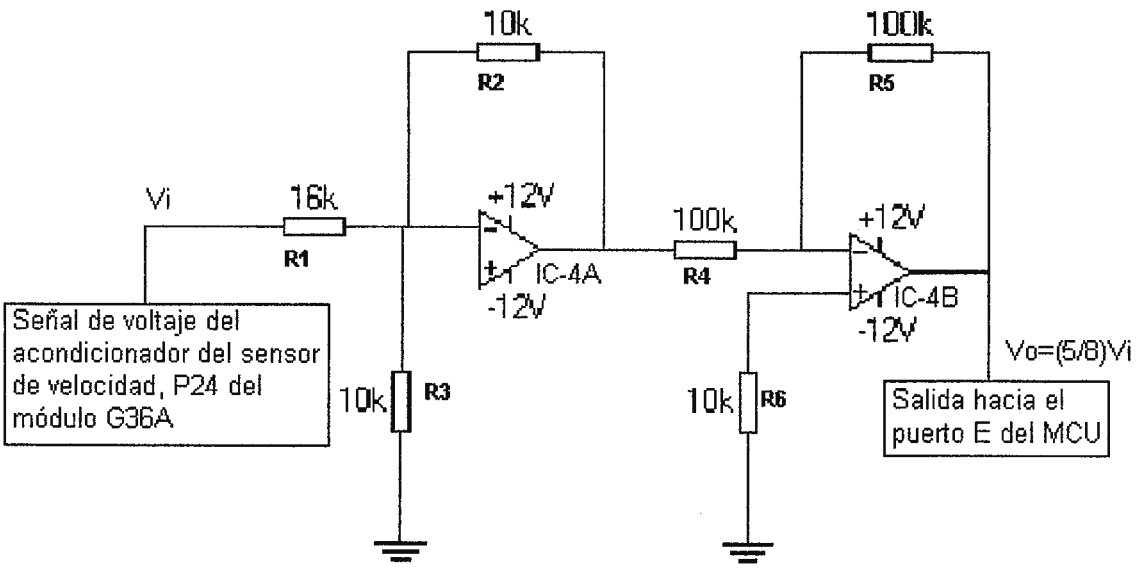


Figura 3.4 Convertidor de nivel del acondicionador de velocidad.

3.5. CONVERTIDOR DE SEÑAL PARA EL SENSOR DE TEMPERATURA.

El horno tiene cuatro sensores, pero para las pruebas de control solamente se usó la termocupla. Este sensor debe calibrarse siguiendo el procedimiento documentado en el Manual Teórico Experimental TG34/EV. El sensor de temperatura utilizado genera voltajes que varían de 0 a +8VDC para la variación de temperatura desde 0C° hasta 250C°. El diseño de este convertidor (ver Figura 3.5) es idéntico al convertidor de señal para el sensor de velocidad.

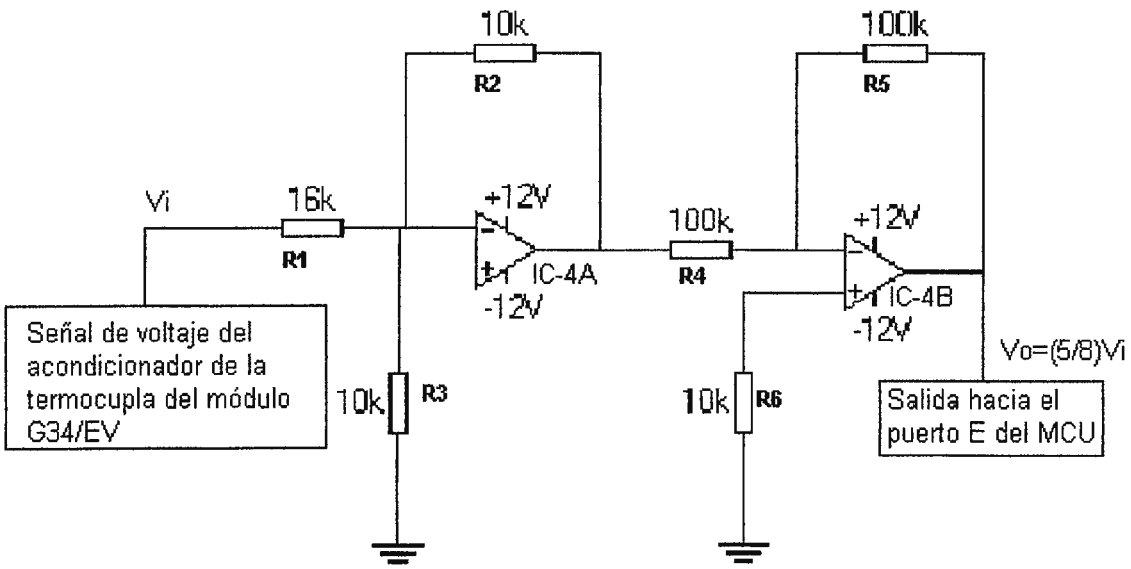


Figura 3.5 Convertidor de nivel del acondicionador de temperatura.

3.6. CONVERTIDOR DE SALIDA PARA EL ACTUADOR DE LA PLANTA DE NIVEL.

Luego que se han descrito los circuitos de interface de entrada se explicarán los de salida. Las salidas digitales de puerto E del microcontrolador generan las señales de control para cualquiera de los tres actuadores analógicos. Se diseñaron dos convertidores D/A de tipo R-2R en escalera, donde la etapa resistiva es la que se conecta a la entrada de cualquiera de los tres actuadores. El diseño de los convertidores para los puertos C y B se muestra en la Figura 3. 6.

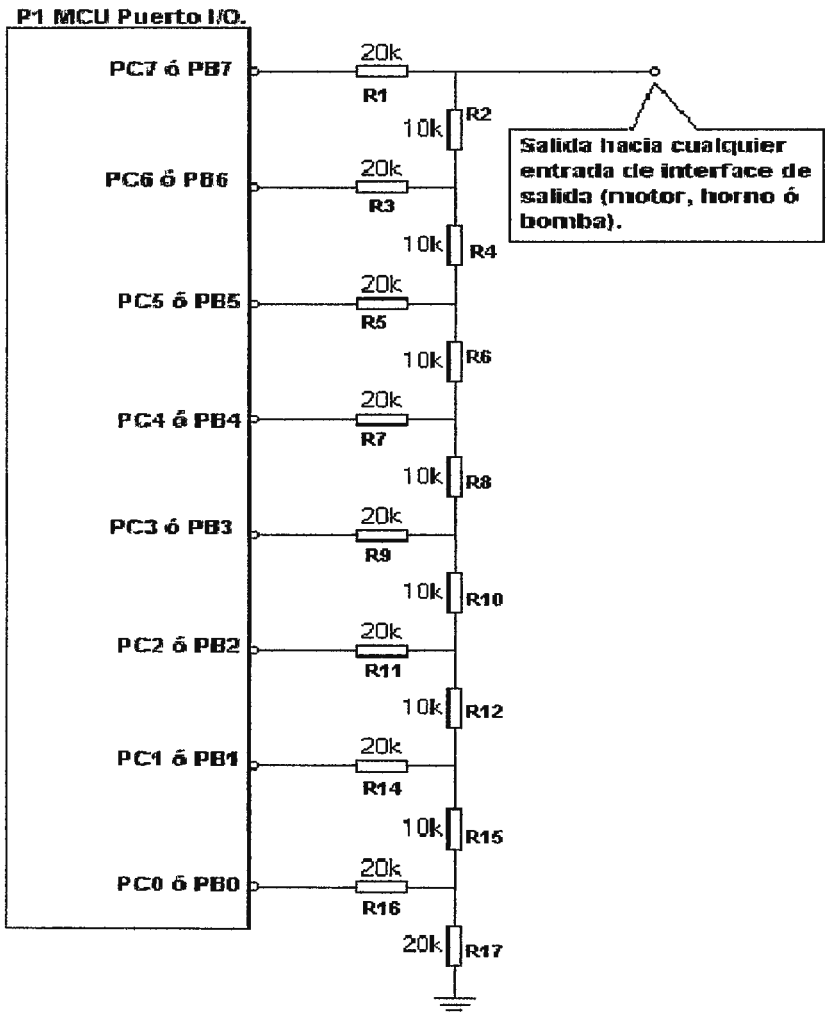


Figura 3.6 Puerto digital conectado al convertidor D/A R-2R.

El convertidor mostrado en la Figura 3.7 transforma el voltaje generado por la red resistiva R-2R a un voltaje capaz de gobernar el actuador de la bomba. La etapa IC-6A aplica un factor de amplificación de 2.25. y la etapa IC-6B brinda alta impedancia. El resultado del convertidor es una señal de voltaje que varía entre 0 y +11.2VDC. Esta señal de voltaje resultante controla el flujo de líquido vertido por la bomba.

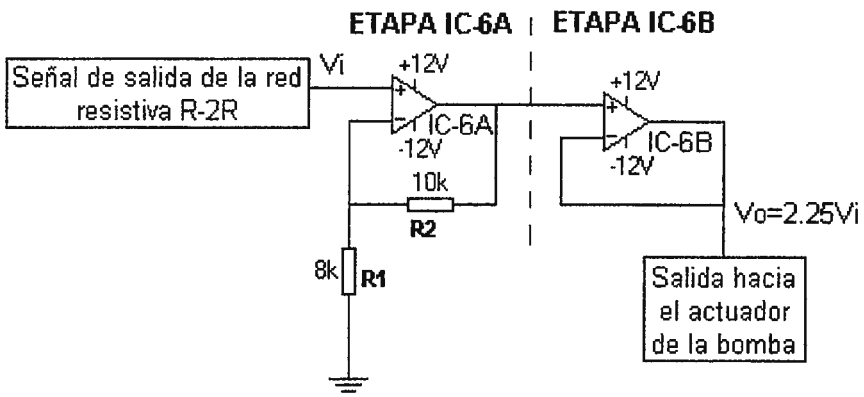


Figura 3.7 Convertidor para actuador de bomba.

3.7. CONVERTIDOR DE SALIDA PARA EL ACTUADOR DE LAS PLANTAS DE POSICION Y VELOCIDAD.

Las plantas de posición y velocidad utilizan un mismo convertidor de salida. Este convertidor (ver Figura 3.8) transforma la señal de salida de la red resistiva R-2R a un voltaje que varía entre –11.25 y +11.25VDC para el actuador de un motor. La etapa IC-7A ofrece aislamiento de carga a la salida del convertidor D/A R-2R. En esta etapa se genera una ganancia un poco mayor a la unidad (1.001). Esta ganancia "residual" no afecta el desempeño del convertidor. En la etapa IC-7B se aplica un voltaje de desplazamiento de –2.5VDC. Luego en la etapa IC-7C se amplifica el resultado de la etapa IC-7B por un factor de 4.5. Al igual que en los

convertidores para los sensores de entrada, el voltaje de desplazamiento junto con el factor de amplificación permite generar voltajes de ambas polaridades a partir de una señal de voltaje positiva.

La salida del convertidor se conecta al punto 18 del limitador de corriente del motor del módulo G36A/EV.

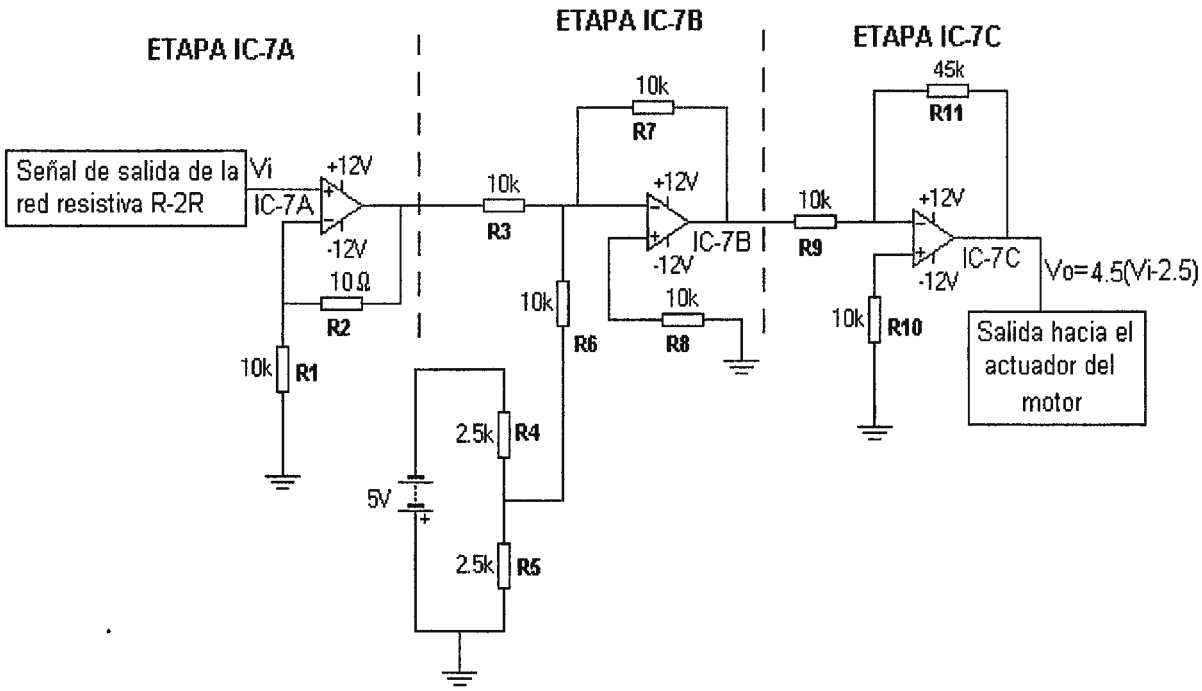


Figura 3.8 Convertidor para actuador de motor.

3.8. CONVERTIDOR DE SALIDA PARA EL ACTUADOR DE LA PLANTA DE TEMPERATURA.

Al igual que el convertidor anterior, a la señal resultante de la red R-2R se le aplica un voltaje de desplazamiento y un factor de amplificación. La etapa IC-8A al igual que la etapa IC-7A del convertidor anterior ofrece aislamiento de carga para la salida del convertidor D/A R-2R. La etapa IC-8B genera un voltaje de desplazamiento de -2.5VDC. La etapa IC-8C invierte la polaridad, aplica un factor de amplificación de 3.2 y provee aislamiento de carga. El voltaje resultante de salida del convertidor varía entre –8 y +8VDC. Los voltajes positivos calientan la resistencia del horno y los voltajes negativos gobiernan el ventilador de enfriamiento.

Nótese que la entrada del actuador es el punto 11 en el amplificador de potencia de la unidad de temperatura del módulo G34/EV. Debe fijarse el interruptor S14 en modo automático para que el ventilador de enfriamiento sea gobernado por el controlador.

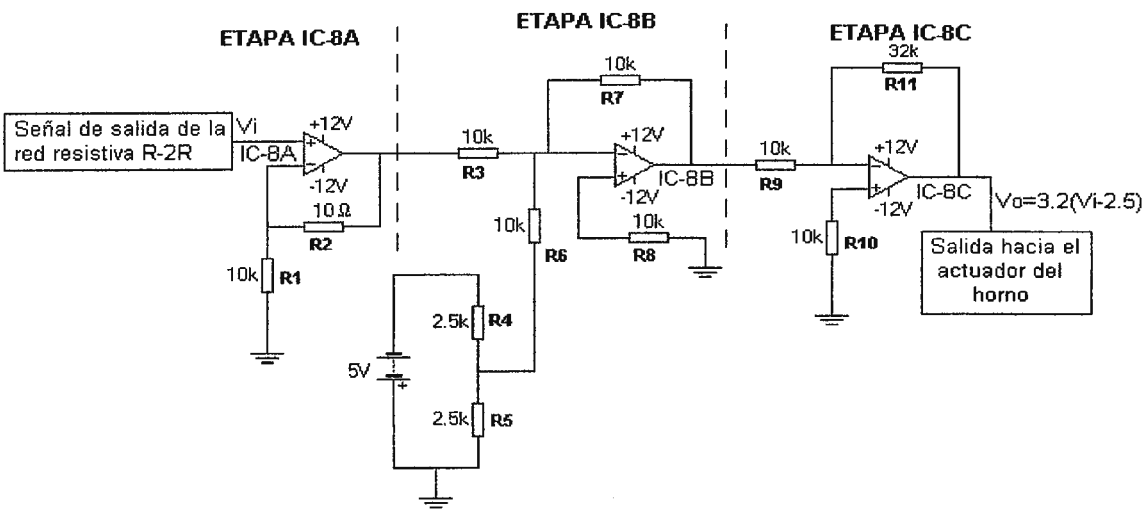


Figura 3.9 Convertidor para actuador de temperatura.

3.9. PUNTO DE AJUSTE (SET-POINT).

El voltaje del punto de ajuste varía de 0 a +5 VDC y se controla de forma manual a través de una resistencia variable (ver Figura 3.10). El controlador difuso obedece al valor del punto de ajuste para ejecutar la labor de control, independientemente del sistema que sé este gobernando ya sea la planta de nivel, posición, velocidad ó temperatura.

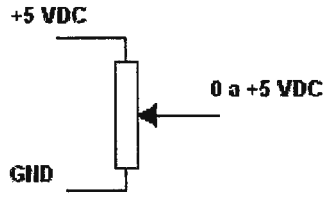


Figura 3.10 Punto de ajuste.

3.10. PUNTO SUMA O SEÑAL DE ERROR.

Para comparar el nivel del punto de ajuste y la señal de retroalimentación, se necesita un punto suma. Es posible incluir un algoritmo en el programa del control difuso para hace la comparación, pero esto conllevaría a incrementar el tiempo de procesamiento, lo cual incrementaría el tiempo de retardo del controlador. Considerando lo anterior, se decidió implementar el circuito mostrado en la Figura 3.11. Como la señal resultante se introduce a alguna de las cuatro entradas del microcontrolador, el circuito genera voltajes entre 0 y +5VDC.

En la etapa IC-10A se obtiene la diferencia de las dos señales de entrada. Luego en la etapa IC-10B se aplica un factor de atenuación de 0.5. Finalmente en la etapa IC-10C se aplica un voltaje de desplazamiento de –2.5VDC.

Si alguna variable lingüística representa al punto suma en el modelo difuso, es importante considerar lo siguiente: cuando ambas entradas sean iguales, la salida del punto suma es de +2.5VDC; cuando la diferencia entre señales de entrada sea negativa, entonces la salida del punto suma será mayor que +2.5VDC; cuando la diferencia entre entradas sea positiva, entonces la salida del punto suma será menor que +2.5VDC. Al conectar las señales de entrada al punto suma debe verificarse que la diferencia de voltajes concuerden con los rangos establecidos en la variable lingüística que representa al punto suma (señal de error) dentro del modelo difuso.

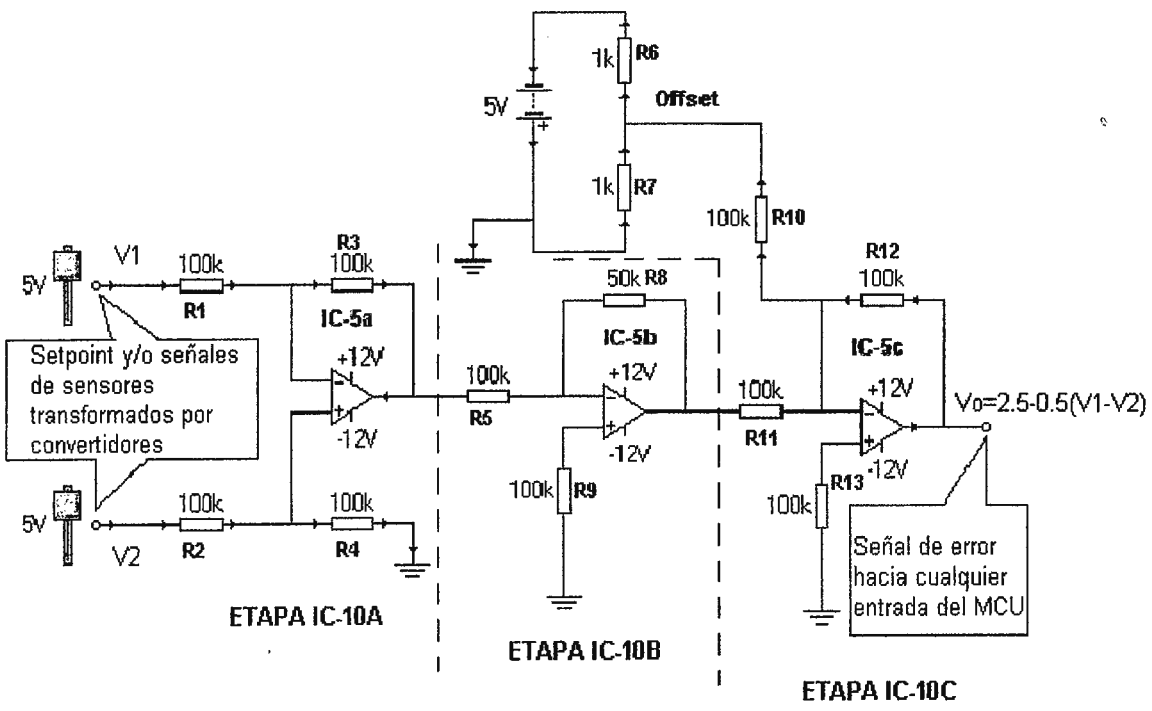


Figura 3.11 Circuito punto suma o de error.

3.11. MÓDULO DE INTERFACES Y CONVERTIDORES.

Los convertidores de entradas y salidas han sido colocados dentro de un gabinete cuyo panel principal se muestra en la Figura 3.12. Todo el conjunto de interfaces utilizan fuentes de alimentación contínua de +5, -5, +12 y –12 VDC. Es importante que el terminal de tierra sea común tanto para los circuitos de interface, la tarjeta EVB11 del microcontrolador y los módulos de los acondicionadores de señal de los sensores de las plantas a gobernar. Además los pines VRL y VRH de la tarjeta EVB11 del microcontrolador deben de conectarse a tierra y a +5VDC respectivamente, los cuales sirven para fijar los niveles de conversión A/D de las entradas analógicas del puerto E.

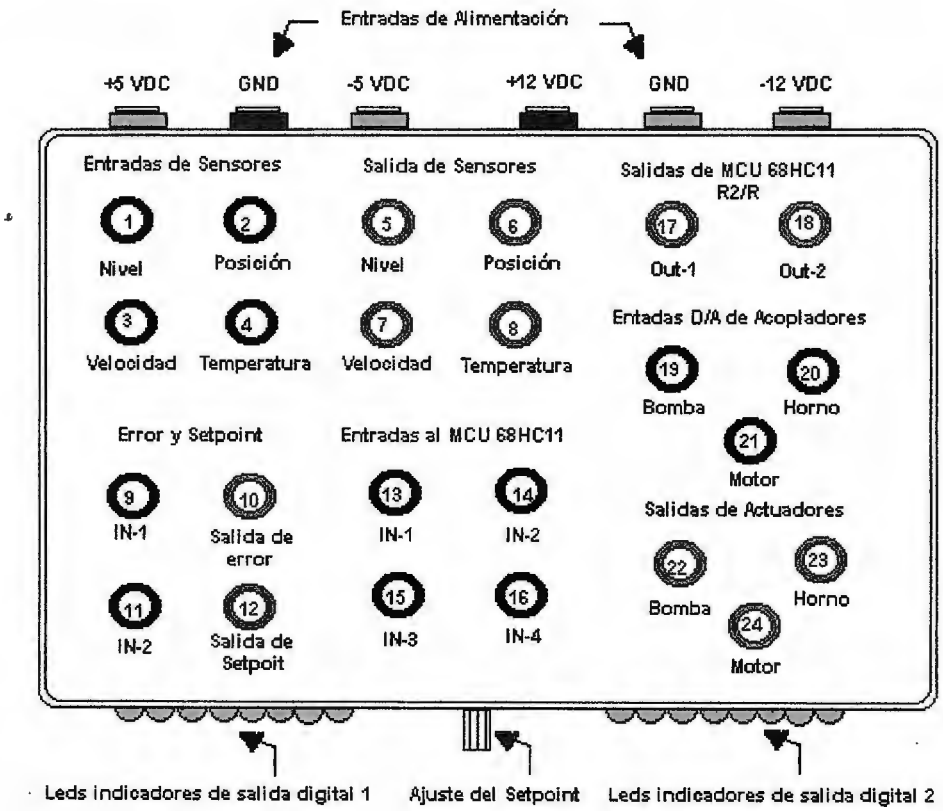


Figura 3.12. Módulo de interface y convertidores.

Todas las conexiones entre salida y entrada se hacen por medio de cables con bananas. Cada borne esta debidamente etiquetado. Para ver ejemplos de como interconectar los diferentes bornes, consultar el capítulo cinco en donde se encontrarán las conexiones realizadas para cada prueba. Los LEDs que aparecen en un lateral del gabinete del módulo corresponden a los bits de los dos puertos de salida. Estos LEDs sirven para conocer el estado de cada bit cuando el controlador difuso esta operando.

CAPITULO 4: SIMULADOR E INTERFAZ DE USUARIO.

El siguiente capítulo tiene como principal propósito explicar el diseño e interacción de los programas del controlador inteligente basado en lógica difusa. Al terminar la lectura de este capítulo, el lector tendrá el suficiente conocimiento para utilizar el simulador y la interfaz de usuario.

4.1. GENERALIDADES DE PROGRAMAS.

El programa de simulación se llama FUDGE[1], el cual se adquirió a través de Internet. FUDGE es un programa que permite simular el control difuso para microcontroladores de Motorola tales como MC68HC05, MC68HC11, MC68HC16 y M68000. Este simulador fue diseñado para auxiliar el proceso de diseño de sistemas de control difuso basados en la inferencia de Mamdani (Max-Min y centro de gravedad). Los modelos difusos que fueron probados en este simulador no excedían de dos variables lingüísticas de entrada y una de salida, y a lo sumo quince reglas.

Una limitación de FUDGE es que no puede interactuar con el microcontrolador directamente. FUDGE no posee la capacidad para cargar los datos al microcontrolador. Por esta última razón, se diseño e implemento una interfaz de usuario para liberar al usuario de la tarea del cargado manual de los datos del modelo difuso en el microcontrolador. Esta interfaz permite al usuario:

---

[1] Fuzzy Design Generator (FUDGE) Versión 1.02, desarrollado por Alex DeCastro, Jason Spielman, John Dumas. Motorola 1994. FUDGE es un programa de dominio público (freeware).

- Introducir variables lingüísticas y sus respectivas funciones.
- Generar reglas de forma manual.
- Generar combinaciones posibles de reglas.
- Generar listados de reglas.
- Almacenar/Recuperar modelos difusos.
- Cargar variables lingüísticas, funciones de membresía y reglas al microcontrolador 68HC11.
- Modificar/Borrar variables, funciones y reglas.

Con el simulador se realizan pruebas de diferentes modelos de control difuso. Al obtener un modelo aceptable, el usuario procede a introducir las definiciones de las variables lingüísticas, las funciones de membresía y las reglas del modelo a la interfaz de usuario llamada MCFUZLOG.

Para carga el programa del microcontrolador que lleva acabo el control difuso, MCFUZLOG necesita de un protocolo de comunicación. Por esta razón se utiliza el programa Hyperterminal de Windows95 para cargar el programa ejecutable FUZZY.HEX[2].

En síntesis, el proceso para diseñar, construir y cargar un modelo de control difuso para el microcontrolador involucra los siguientes pasos:

- Simular los modelos difusos usando el programa FUDGE.
- Ejecutar el programa MCFUZLOG para introducir las variables lingüísticas, las funciones de membresía y reglas simuladas en el programa FUDGE.
- Ejecutar el programa HYPERTERMINAL de Windows95 para cargar el programa FUZZY.HEX.

Una vez hecho lo anterior, se envía al microcontrolador la instrucción para ejecutar el programa. En este momento, el microcontrolador actúa

[2] El capítulo tres y los apéndices D y E explican detalladamente el programa FUZZY.HEX

independientemente de la computadora. Para mejorar o cambiar el modelo de control difuso, se debe seguir el procedimiento anteriormente mencionado.

4.2. PROGRAMA FUDGE.

Este simulador funciona bajo la plataforma Windows[3]. La pantalla principal muestra un menú principal en la parte superior. De cada opción del menú principal, aparecen otros submenús. En la opción edit se encuentra el submenú que permite introducir los datos referente a las variables lingüísticas de las entradas/salidas, así como la generación de reglas (ver Figura 4.1).

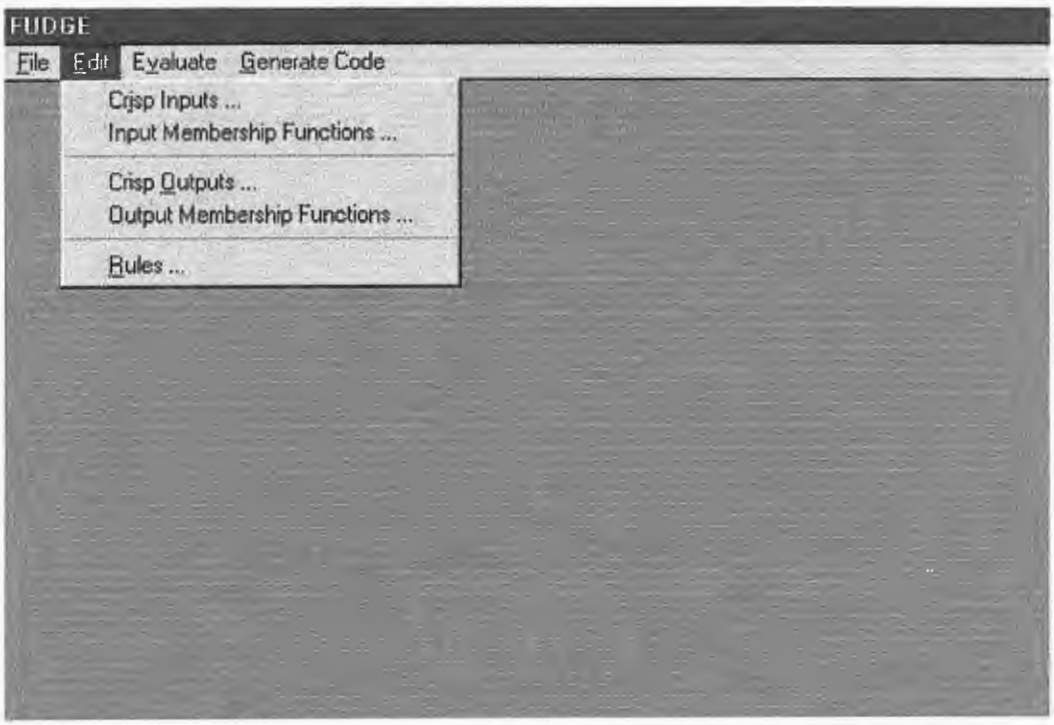


Figura 4.1 Menú principal del programa FUDGE.

[3] Windows es un sistema operativo desarrollado por Microsoft. Existen diferentes versiones de este sistema operativo: Windows 3.1, Windows95 y Windows98.

A través de las ventanas activadas por las opciones "Crisp Inputs" y "Crisp Outputs" se declaran y dimensionan las variables lingüísticas de las entradas/salidas "crisp". En la ventana mostrada en la Figura 4.2 se introduce el nombre de la variable, los intervalos permisibles, las unidades de medida y el número de funciones de membresía a declarar.

Después de introducir la información anterior para todas las variables, deben declararse las funciones de membresía a través de la ventana mostrada en la Figura 4.3 que es activada por las opciones de funciones de membresía de entradas/salidas.

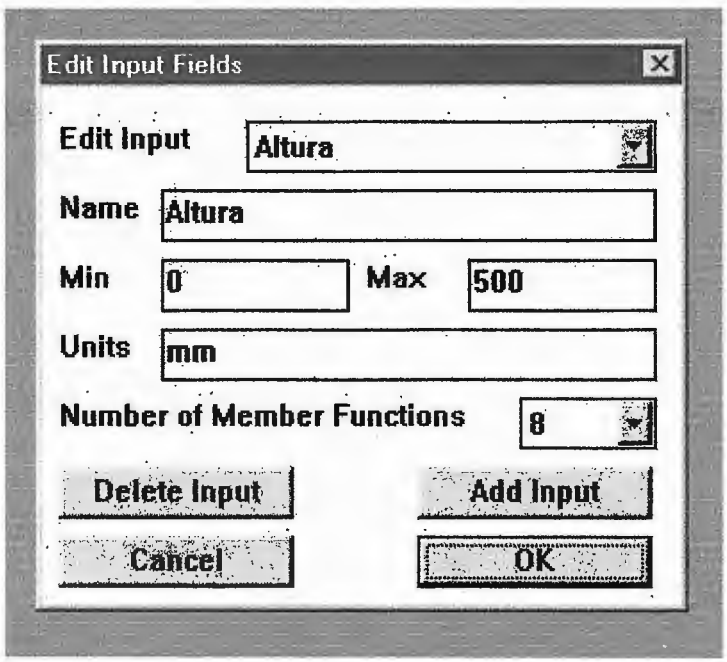


Figura 4.2 Ventana para introducir variables lingüísticas.

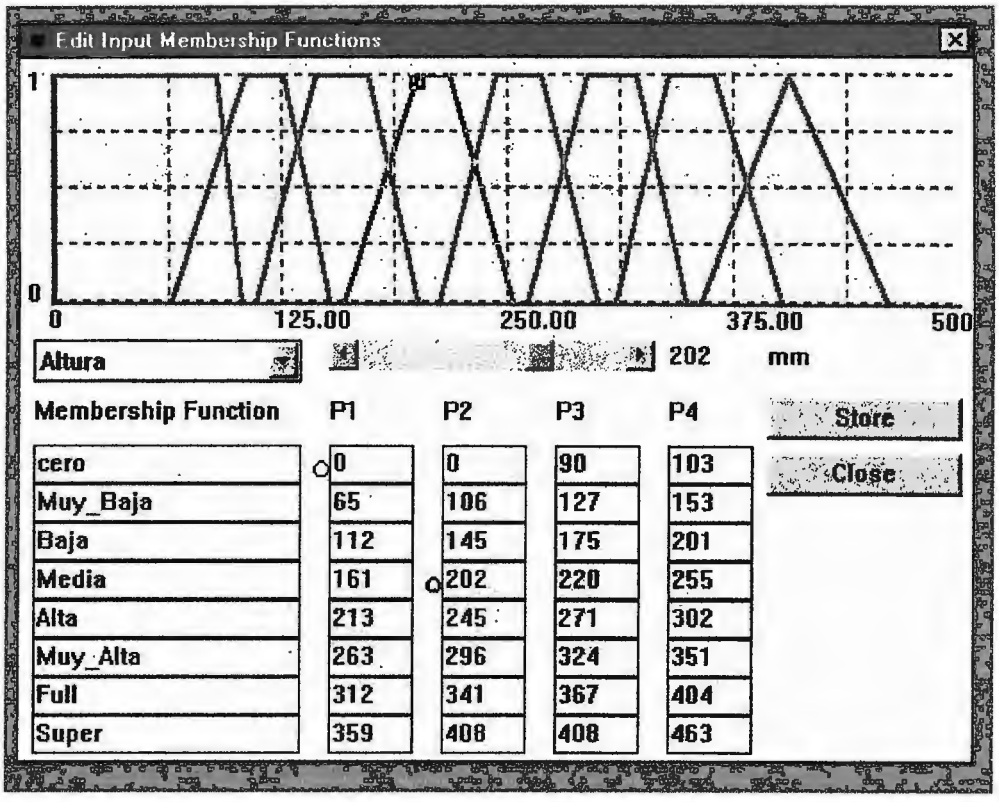


Figura 4.3 Declaración de funciones de membresía.

En la parte superior de la ventana se presenta la forma de la función de membresía. En la parte inferior se presenta una matriz en la cual cada fila corresponde a cada función de membresía, la primera columna corresponde al nombre y las otras cuatro columnas contienen los vértices con los cuales pueden declararse funciones de forma triangular, trapezoidal, rectangular o singleton. El nombre de la variable lingüística que se edita aparece en el recuadro inmediatamente abajo del gráfico al lado izquierdo. En este recuadro se elige la variable a editar. En vez de llenar la matriz de vértices, la barra deslizante que aparece a la par del recuadro sirve para modificar utilizando el ratón la forma de la función cuyos bordes estén activos (los bordes negros indican que la función ha sido seleccionada). También puede modificarse la función arrastrando los vértices con el ratón. Para almacenar las modificaciones debe presionarse al final el botón STORE.

El siguiente paso es configurar las diferentes reglas. Al seleccionar la opción de reglas aparece la ventana mostrada en la Figura 4.4. En la ventanilla principal aparecen en formato IF-THEN las reglas enumeradas. El botón superior del lado izquierdo borra la regla cuyo número este sobre saltado por la barra azul. El botón de añadir presenta la ventana mostrada en la Figura 4.5. En esta ventana se generan las reglas seleccionando las variables lingüísticas y funciones de membresía para el antecedente y consecuente de la regla. Si un antecedente es compuesto, entonces se presiona el botón marcado con AND, sino se presiona el botón THEN para introducir el consecuente.

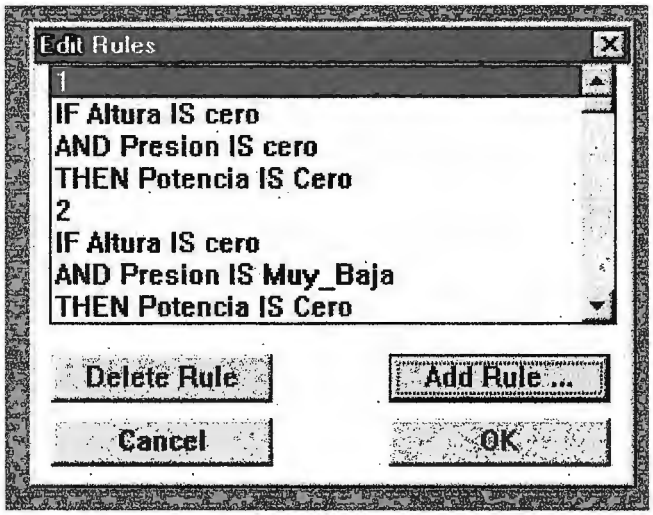


Figura 4.4 Vista de reglas.

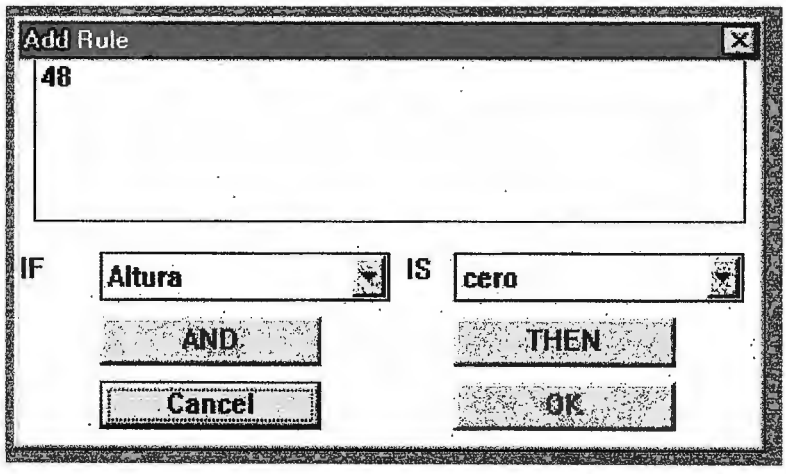


Figura 4.5 Formulación de reglas.

Con toda la información del modelo difuso introducida, en el submenú de evaluación (ver Figura 4.6) del menú principal puede activarse módulos que permiten estudiar el comportamiento del control difuso.

Figura 4.6 Opción de evaluación.

Al seleccionar el evaluador difuso (ver Figura 4.7), aparece una ventana en la cual puede observarse como diferentes valores de entrada generan grados de pertenencia y modifican la fuerza de las reglas. En la parte superior de la ventana aparecen la representación gráfica para las funciones de entrada y salida. Al lado izquierdo pueden observarse las funciones de membresía para una entrada. Si se desea observar alguna otra entrada debe de elegirse esta en el recuadro que aparece inmediatamente debajo. Al lado derecho aparece la representación gráfica de las variables de salida e igual a las entradas estas pueden elegirse a través del recuadro que aparece inmediatamente abajo. La barra de desplazamiento que aparece al lado izquierdo sirve para ubicar el valor de entrada que se desea aplicar a la variable de entrada que se ha elegido. Para observar la interrelación entre variables de entrada/salida puede fijarse un valor para una y luego seleccionar la otra variable y sobre la barra desplazarse para ver el

comportamiento del control difuso. Este comportamiento puede apreciarse en la representación gráfica de las salidas en donde aparecerá un singleton flotante que indican el comportamiento del proceso de defusificación.

Debajo de la representación gráfica aparecen enumeradas las funciones de membresía para las variables de entradas y salidas que se hayan seleccionado para observarse en la parte gráfica. A la par de cada función de membresía de entrada aparece el valor obtenido por el proceso de fusificación; y para las salidas el valor que aparece es producido por el motor de inferencia a través de la evaluación de reglas. Por último, en la parte inferior de la ventana se encuentra una recuadro donde puede verse una regla a la vez y su respectiva fuerza o contribución.

Para observar el comportamiento del modelo difuso para todo el rango de valores de una variable de entrada, se utiliza la superficie de control (submenú de evaluación, ver Figura 4.8). La limitación de la superficie de control es que solamente permite observar el comportamiento de una entrada variable contra una variable de salida. Si el modelo involucra más de una entrada, entonces el usuario elige una variable de entrada y el programa tratará las demás como constantes, permitiendo al usuario fijar los valores (ver Figura 4.9 y Figura 4.10).

FUDGE permite la generación de un archivo texto en el cual aparece el modelo difuso codificado en formato hexadecimal del microcontrolador (ver Figura 4.11). El usuario puede elegir el microcontrolador desde el submenú de Generación de Código (ver Figura 4.12) en donde aparece un listado de microcontroladores para los cuales FUDGE es capaz de generar códigos.

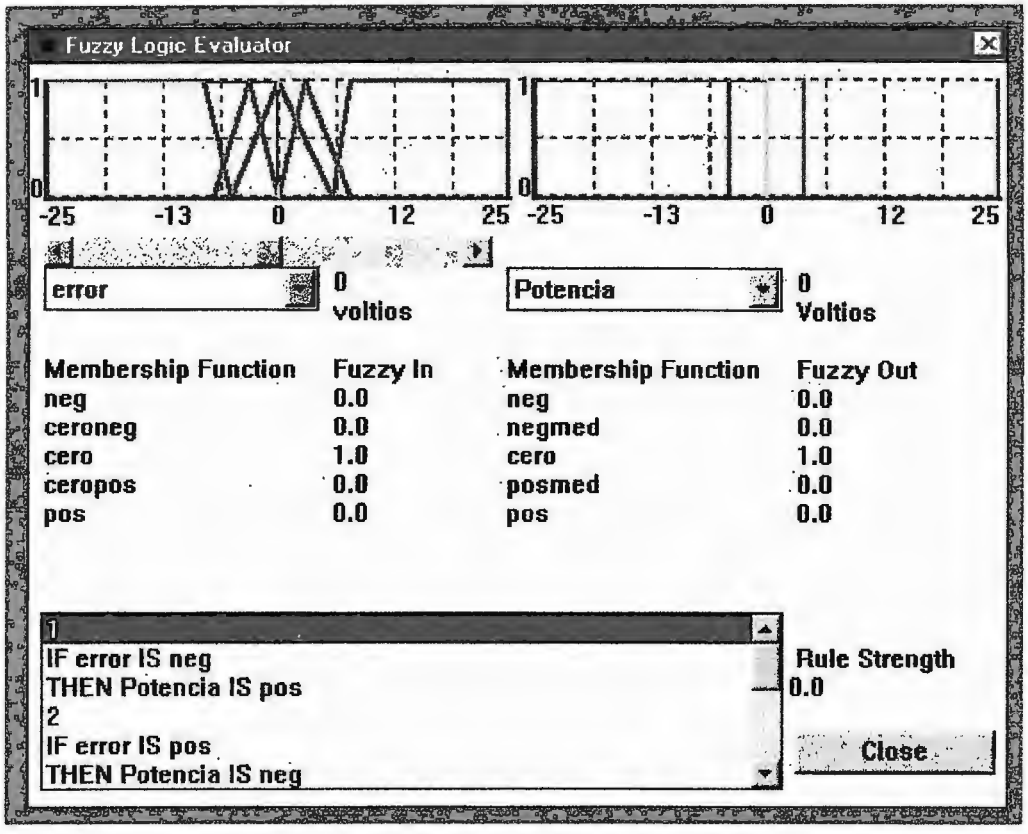


Figura 4.7 Evaluador Difuso.

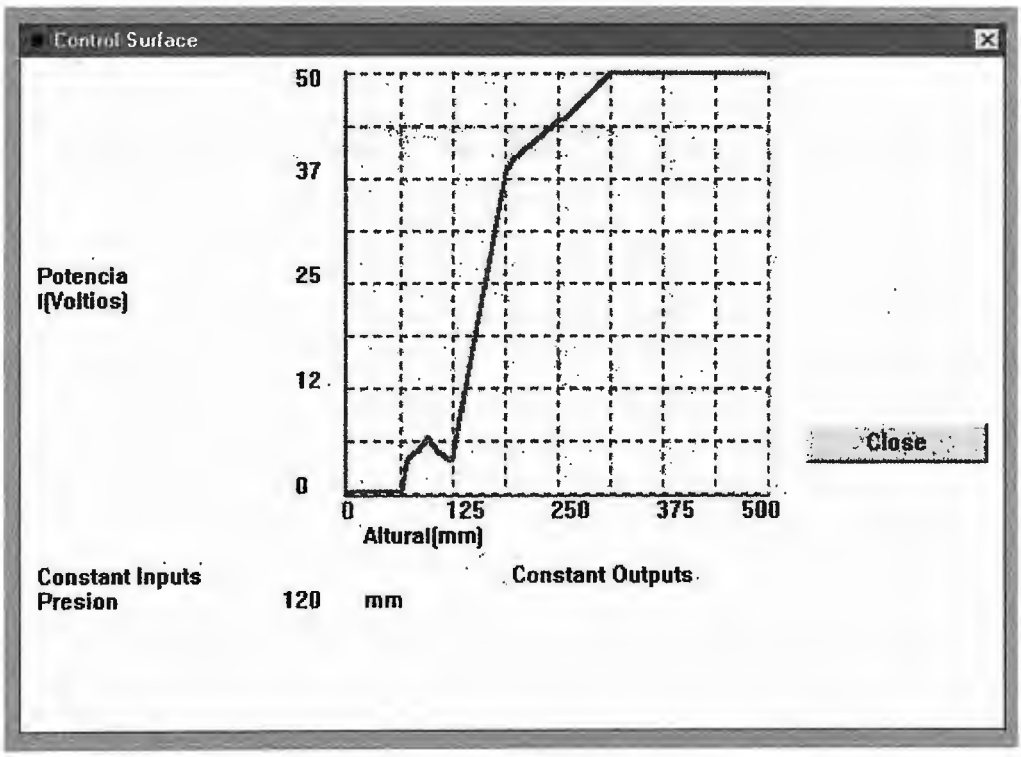


Figura 4.8 Superficie de Control.

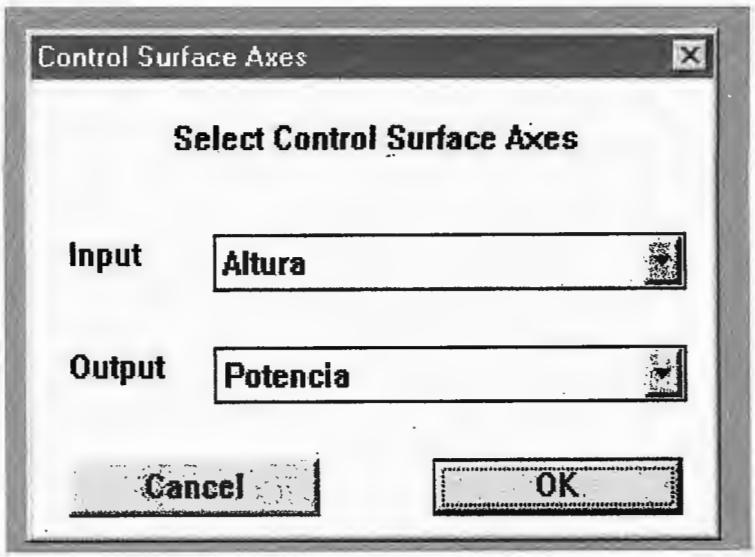


Figura 4.9 Selección de variables para superficie de control.

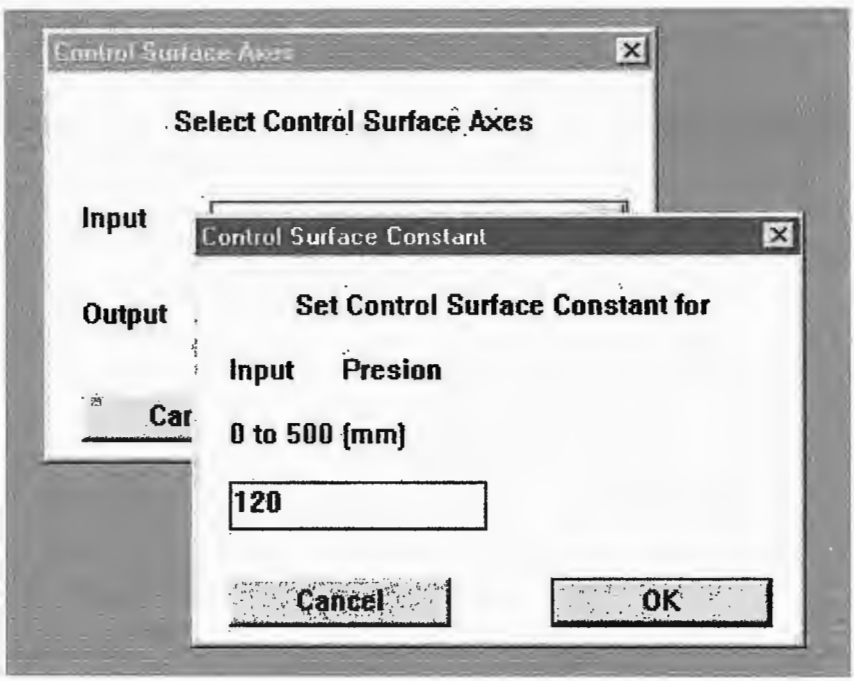


Figura 4.10 Introducción de constantes para la superficie de control.

```
*       Fuzzy Development and Generation Environment (FUDGE) Version V1.02
*       MC68HC11 assembly file
*       John Dumas & Jason Spielman & Alex DeCastro
*       Copyright Motorola 1994

INPUT_MFS   EQU   *         ; Input Membership Functions
IN0MF       EQU   *         ;       Altura
            FCB   $00,$00,$21,$08 ;       cero
            FCB   $21,$08,$41,$08 ;     Muy_Baja
            FCB   $40,$08,$61,$08 ;       Baja
            FCB   $61,$08,$80,$08 ;      Media
            FCB   $80,$08,$a1,$08 ;       Alta
            FCB   $a1,$08,$bf,$08 ;       Full
            FCB   $c2,$08,$ff,$00 ;      Super
            FCB   $00,$00,$00,$00 ;        ~
IN1MF       EQU   *         ;       Presion
            FCB   $00,$00,$2a,$0c ;       cero
            FCB   $1f,$08,$40,$08 ;     Muy_Baja
            FCB   $40,$08,$61,$08 ;       Baja
            FCB   $61,$08,$80,$08 ;      Media
            FCB   $80,$08,$a1,$08 ;       Alta
            FCB   $9e,$08,$bf,$08 ;     Muy_Alta
            FCB   $bf,$07,$ff,$00 ;      Extrema
            FCB   $00,$00,$00,$00 ;        ~
SGLTN_POS   EQU   *         ; Output Membership Functions
OUT0MF      EQU   *         ;       Potencia
            FCB   $00             ;       Cero
            FCB   $42             ;     Muy_Baja
            FCB   $80             ;       Baja
            FCB   $a3             ;      Media
            FCB   $c2             ;       Alta
            FCB   $e0             ;     Muy_Alta
            FCB   $ff             ;       Full
            FCB   $00             ;        ~
RULE_START  EQU   *         ; Rules follow:
            FCB   $00
            FCB   $08
            FCB   $80
            FCB   $00
            FCB   $09
            FCB   $80
            FCB   $FF
```

Figura 4.11 Listado de códigos hexadecimales para el MC68HC11.

Por último, en el submenú de archivo (ver Figura 4.13) se encuentra las opciones de cargar, guardar y crear un modelo difuso. También se encuentra la opción de crear un archivo texto (ver figura 4.14), el cual genera un listado del

modelo difuso para entendimiento del usuario. Además en este submenú se encuentran las opciones de salir e información de la versión del programa FUDGE y sus creadores.

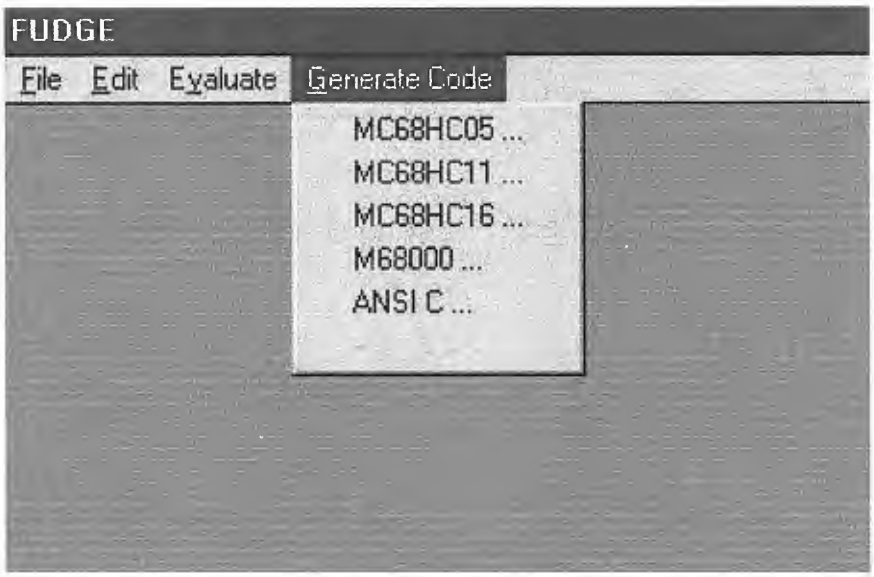


Figura 4.12 Listado de microcontroladores a los cuales puede codificarse modelo difuso.

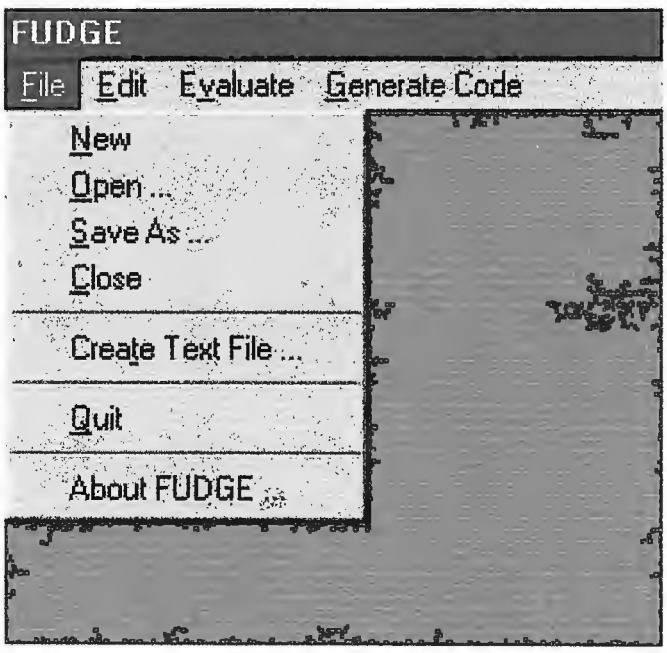


Figura 4.13 Opción Archivo.

```
*       FUzzy Development and Generation Environment (FUDGE) Versión V1.02
*       User readable knowledge base file
*       Alex DeCastro & Jason Spielman & John Dumas
*       Copyright Motorola 1994
        INPUTS
Input #1
    INPUT NAME          MIN     MAX             UNITS
      Altura    0       500         mm
MEMBERSHIP FUNCTION         P1      P2      P3      P4
        cero        0       0       65      125
      Muy_Baja      65      127     127     190
        Baja        125     190     190     251
       Media        190     250     250     315
        Alta        250     315     315     376
        Full        315     375     375     440
       Super        380     440     500     500
          ~         0       0       0       0
Input #2
    INPUT NAME          MIN     MAX             UNITS
      Presion   0       500         mm
MEMBERSHIP FUNCTION         P1      P2      P3      P4
        cero        0       0       82      125
      Muy_Baja      60      125     125     190
        Baja        125     190     190     250
       Media        190     250     250     310
        Alta        250     315     315     375
      Muy_Alta      310     375     375     440
      Extrema       375     445     500     500
          ~         0       0       0       0
        OUTPUTS
Output #0
    OUTPUT NAME         MIN     MAX             UNITS
      Potencia  0       50      Voltios
MEMBERSHIP FUNCTION         P1
        Cero        0
      Muy_Baja      13
        Baja        25
       Media        32
        Alta        38
      Muy_Alta      44
        Full        50
          ~         0
        RULES
Rule #1
IF Altura IS cero
AND Presion IS cero
THEN Potencia IS Cero
Rule #2
IF Altura IS cero
AND Presion IS Muy_Baja
THEN Potencia IS Cero...
```

Figura 4.14 Listado del modelo difuso para el entendimento del usuario.

4.3. PROGRAMA MCFUZLOG.

La necesidad de este programa se debe a que el simulador FUDGE no tiene interfaz alguna con el microcontrolador MC68HC11. Debido a lo anterior, el usuario debe de introducir cada código hexadecimal en la memoria RAM del microcontrolador. Para evitar al usuario esta tediosa labor, el programa MCFUZLOG se creó con el objetivo de brindar una interfaz amigable y flexible para introducir los datos del modelo difuso al microcontrolador. El programa MCFUZLOG no pretende sustituir a FUDGE, sino complementarlo.

Al diseñarse un modelo difuso primero se utiliza el programa FUDGE para simular todas las posibles alternativas de solución. Al obtener un modelo aceptable se procede a introducirse los datos del modelo en el programa MCFUZLOG para que a través de este se carguen los códigos adecuados en el microcontrolador. MCFUZLOG tiene cuatro funciones principales: recepción de modelo difuso, almacenamiento/recuperación/edición de modelos, generación de reglas, y transferencia de modelos a microcontrolador.

FUDGE tiene la desventaja de trabajar solamente con números enteros para delimitar las formas de las funciones. Si se necesita especificar fracciones, en FUDGE debe especificarse el rango de tal forma que el número entero incluya el valor fraccionario. Por ejemplo, si se tiene un rango de 0 a 20 en donde es necesario manejar un valor como 12.5 entonces en FUDGE debe declararse un rango de 0 a 200 donde el valor 12.5 será 125. Para evitar este problema, MCFUZLOG permite la introducción de datos fraccionarios si el usuario desea utilizar el rango real de trabajo.

Otra desventaja de FUDGE es que la generación de reglas es manual. En MCFUZLOG existe la generación de reglas de forma manual y semiautomática. En la forma manual, el usuario arma las reglas una por una. En el proceso

semiautomático, el usuario especifica cuantas y cuales variables conformarán el antecedente y también se especifica la variable de salida para el consecuente. Automáticamente MCFUZLOG genera todas las combinaciones posibles para el antecedente. Por cada combinación, MCFUZLOG permite al usuario introducir la acción a tomar dependiendo de la variable de salida seleccionada anteriormente.

A continuación se describe con más detalle el programa MCFUZLOG.

4.3.1. PLATAFORMA DE DESARROLLO Y REQUERIMIENTOS.

MCFUZLOG es un programa escrito en lenguaje C++ para DOS. Una de las razones para utilizar C++ como lenguaje de programación es la facilidad de inclusión de lenguaje ensamblador en el programa fuente. Otra ventaja fundamental es la portabilidad de programas escritos en C++, ya que C++ es un lenguaje de bajo y alto nivel. Es decir, el programa MCFUZLOG no tiene problemas en ejecutarse en diversas computadoras con diferente arquitectura, siempre y cuando todas sean compatibles con IBM.

Para ejecutar MCFUZLOG en una computadora, ésta al menos debe tener 2 MB de memoria RAM, procesador 386 o posterior, el sistema operativo debe ser al menos DOS 3.X o posterior, un puerto serie habilitado y un medio del almacenamiento como disco flexible o disco duro (de preferencia un disco duro de al menos 75MB).

Ya que es necesario un programa de comunicación para cargar el programa FUZZY.HEX al microcontrolador, MCFUZLOG hace uso de los programas de terminal o hyperterminal de Windows3.x o Windows95 ya que la computadora en que se trabajó tiene uno de estos sistemas operativos.

4.3.2 MANUAL DE USUARIO.

MCFUZLOG es un programa hecho para DOS y carece de interfaz gráfica. El menú principal es un listado con trece opciones (ver Figura 4.15):

```
                    MCFUZLOG

                  MENU PRINCIPAL

1.- DECLARACION DE FUNCIONES
2.- AÑADIR VARIABLE
3.- BORRAR VARIABLE O FUNCION
4.- MODIFICAR FORMA DE FUNCION
5.- VARIABLES
6.- GENERACION DE REGLAS
7.- GENERADOR DE REGLAS
8.- LISTADO DE REGLAS
9.- GUARDAR MODELO DIFUSO
10.- RECUPERAR MODELO DIFUSO
11.- CARGAR MCU CON FUNCIONES DIFUSAS
12.- CARGAR MCU CON PROGRAMA
13.- SALIR
Seleccion:
```

Figura 4.15 Menú principal de MCFUZLOG.

En la declaración de variables, MCFUZLOG pide al usuario introducir cuantas variables de entrada tendrá el modelo difuso. Para cada variable pide el nombre lingüístico, las unidades de medida, el rango de valores y el número de funciones de membresía. Para cada función de membresía de cada variable, MCFUZLOG pide el respectivo nombre y los cuatro vértices de la figura. MCFUZLOG revisa que los valores introducidos para los vértices sean coherentes. Sin embargo, MCFUZLOG no tiene la capacidad de validar la entrada de datos numéricos o alfanuméricos.

Después de introducir todos los datos de las variables de entrada, MCFUZLOG solicita los datos de las variables de salida. El programa solicita los mismos tipos de datos, solamente que para la definición de cada función de membresía pide un dato ya que las funciones de membresía para las salidas son singletons.

Terminado lo anterior, MCFUZLOG regresa al menú principal. El siguiente paso recomendado al usuario es introducir las reglas. La opción de Generación de reglas permite al usuario formar una regla a la vez. La pantalla para introducir los datos tiene el siguiente formato (ver Figura 4.16):

Variable de		Función de membresía:
Entrada:	Antecedente	Cero
Altura	Altura	Medio
	ES	Máximo
	Cumbre	
Variable de	Consecuente	
Salida:	Potencia	
Potencia	ES	
	Cero	

Regla 1: Altura es Cumbre Entonces Potencia es Cero

Figura 4.16: Pantalla para generación de reglas.

Al lado izquierdo superior de la pantalla aparecen las variables de entrada y salida. Inmediatamente debajo de la palabra antecedente aparece el cursor. MCFUZLOG espera a que el usuario seleccione la variable de entrada a través de las teclas cursoras[4] hacia arriba o hacia abajo. Según presione las teclas, el

[4] Teclas que tienen marcadas flechas, ubicadas entre el teclado alfanumérico y el teclado numérico.

programa muestra en pantalla la variable dependiendo de la secuencia en que se introdujo en la declaración de variables. Cuando aparece la variable deseada, el usuario debe presionar la tecla ENTER para proseguir. El cursor reaparece debajo de ES y en el lado derecho de la pantalla aparecen todas las posibles funciones de membresía para la variable anteriormente seleccionada. Igual como en el antecedente, el usuario selecciona la función de membresía deseada. Si se presiona ENTER, el cursor pasa debajo del consecuente; si presiona la letra 'Y', MCFUZLOG interpreta que se desea añadir otra proposición al antecedente unido con el conector lógico Y, entonces el cursor otra vez se ubica debajo de la palabra ANTECEDENTE. MCFUZLOG no posee un limite de proposiciones para el antecedente, sin embargo sólo permite un consecuente por regla. También MCFUZLOG no impone limite a la cantidad de reglas que pueda contener el modelo difuso. El limite de reglas y proposiciones están regidos por el espacio disponible en la memoria del microcontrolador.

A medida que se arman las reglas, el usuario verá en la parte inferior como estas se estructuran en forma lingüística. Para terminar de introducir reglas, el usuario debe presionar la tecla 'T' para terminar cuando el cursor este debajo de la palabra ANTECEDENTE, en cualquier otro lugar MCFUZLOG no lo interpretará correctamente.

Para la opción de generador de reglas, MCFUZLOG pregunta cuantas y cuales variables de membresía se utilizarán para hacer las reglas y después pregunta por la variable de salida a ocupar. Para la solicitud del dato numérico, el usuario debe introducirlo. Para elegir las variables se utilizan las teclas cursoras de la misma forma en que se han ocupado anteriormente.

Luego de introducir la anterior información, MCFUZLOG comienza a generar las diferentes combinaciones. Cada combinación se presenta en pantalla junto con el consecuente, MCFUZLOG espera que el usuario complete la regla introduciendo, a través de las flechas cursoras, la función de membresía para la

acción (ver Figura 4.17). Inmediatamente se complete la última combinación, MCFUZLOG regresa al menú principal.

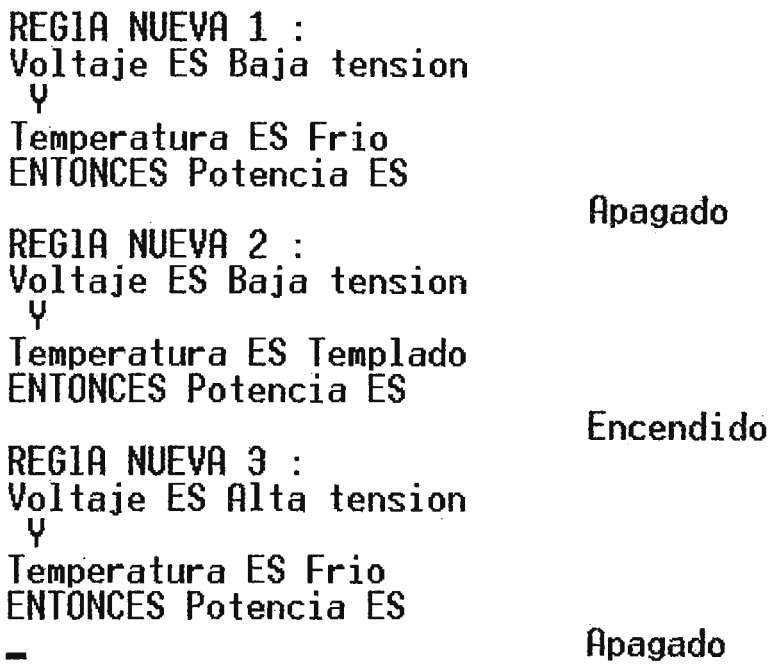


Figura 4.17 Generador de reglas.

La opción listado de reglas permite al usuario observar todas las reglas que se han introducido al modelo difuso (ver Figura 4.18). Aquí MCFUZLOG permite borrar las reglas que el usuario desee. Al presionar 'B' para borrar, el programa pregunta por el número de la regla. Todas las reglas seleccionadas para borrar se eliminan automáticamente al terminar de ver el listado de reglas.

El modelo difuso puede almacenarse en disco al seleccionar la opción de Guardar Modelo Difuso. El usuario sólo debe digitar un nombre sin extensión para el archivo. Si se desea colocar el archivo en otro directorio, entonces antes del nombre debe incluirse la dirección completa (ejemplo: C:\Archivo\Modelo). En caso de que no sea posible guardar los datos, MCFUZLOG envía un mensaje en pantalla informando que no fue posible guardar el modelo difuso.

```
REGLAS DIFUSAS
REGLA 1 :
Voltaje ES Baja tension
 Y
Temperatura ES Frio
 ENTONCES
Potencia ES Apagado
REGLA 2 :
Voltaje ES Baja tension
 Y
Temperatura ES Templado
 ENTONCES
Potencia ES Encendido
REGLA 3 :
Voltaje ES Alta tension
 Y
Temperatura ES Frio

Por favor presionar cualquier tecla para continuar (T=terminar y B=borrar)
```

Figura 4.18 Listado de reglas.

MCFUZLOG crea dos archivos: uno con extensión MFC y otro con extensión S. El archivo con extensión MFC contiene la suficiente información del modelo difuso para recuperarlo en ocasiones posteriores. El archivo con extensión S almacena los códigos hexadecimales del modelo difuso para que el usuario pueda verlo a través de un editor de texto y saber que datos se cargan al microcontrolador.

En la opción de Recuperar el Modelo Difuso, el usuario solamente debe proveer el nombre del archivo incluyendo la dirección completa si se encuentra en otro directorio. Si no se puede recuperar el archivo, MCFUZLOG envía un mensaje a pantalla dando la advertencia. Ya teniendo el modelo difuso cargado en el programa, con la opción de Cargar MCU con Modelo Difuso MCFUZLOG genera los códigos hexadecimales y los carga al microcontrolador de forma automática.

Durante el proceso de cargado, en pantalla se visualizan los datos hexadecimales transmitidos al microcontrolador.

Luego de cargar el modelo, debe cargarse el programa que le da la inteligencia difusa al microcontrolador. Con la opción de Cargar MCU con Programa, MCFUZLOG activa el programa de comunicación HYPERTERMINAL para que el usuario solamente seleccione el protocolo de comunicación y el archivo a enviar, el cual es FUZZY.HEX.

MCFUZLOG tiene otras opciones que permiten la edición de variables. La edición solamente se permite si el modelo aun no posee reglas al momento de editar. La opción de Añadir Variable es la única que no esta sujeta a lo anterior. Para esta opción el programa pregunta si se introducirá una variable de entrada o salida. Dependiendo de la elección, MCFUZLOG muestra un listado de todas las variables de entrada o salida (ver Figura 4.19). Luego pregunta cuantas variables más se introducirán. De aquí en adelante el procedimiento es igual al de la opción Declaración de Variables.

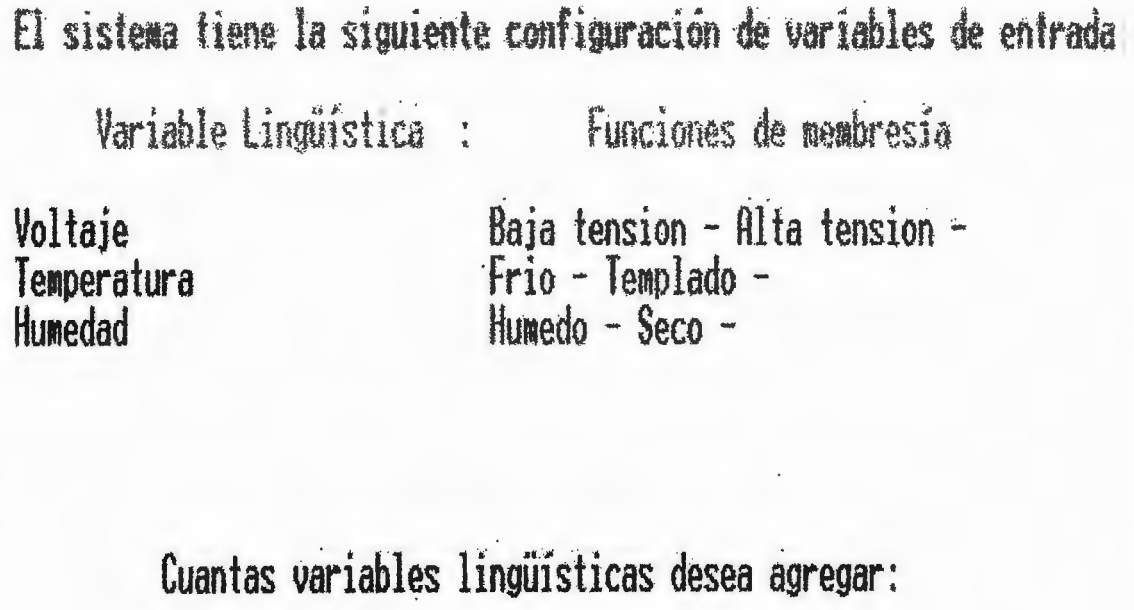


Figura 4.19 Listado de variables lingüísticas antes de añadir una variable.

La opción de Variables de Modelo Difuso permite al usuario ver las variables lingüísticas de entrada y salida junto con las definiciones de las respectivas funciones de membresía (ver Figura 4.20).

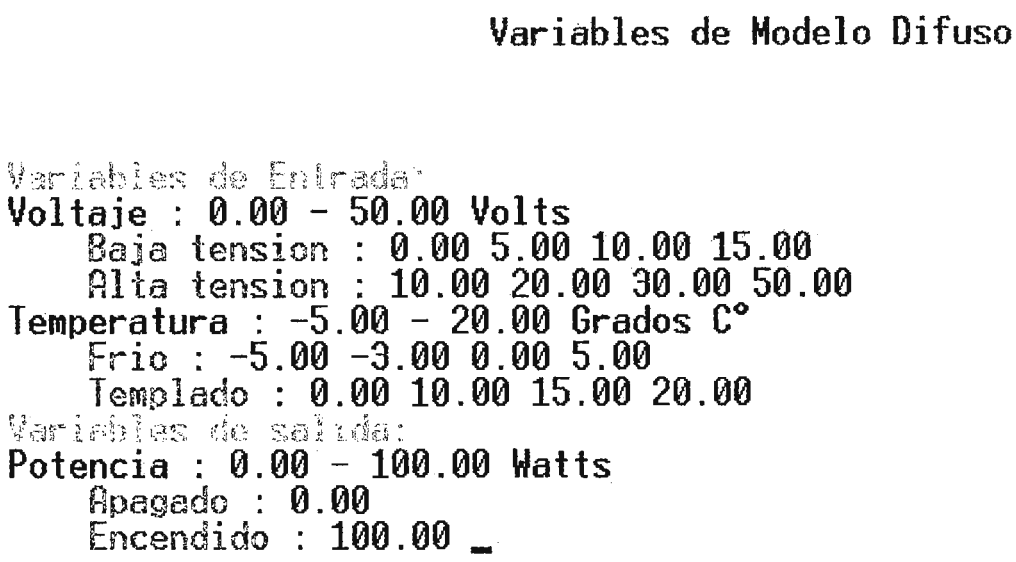


Figura 4.20 Variables lingüísticas del modelo difuso.

En la opción Modificar Forma de Función, MCFUZLOG pregunta por la variable lingüística y luego por la función de membresía a modificar. La elección de la variable y la función se hace a través de las flechas cursoras. Luego de indicar la función a modificar, MCFUZLOG pide al usuario introducir los nuevos vértices de la función (ver Figura 4.21).

También MCFUZLOG, a través de la opción Borrar Variable o Función, permite al usuario eliminar variables completas o solamente alguna función de membresía. MCFUZLOG pide al usuario que indique si la acción se hará sobre alguna variable de entrada o salida. De aquí en adelante el usuario sólo tiene que seleccionar lo que desea borrar a través de las teclas cursoras (ver Figura 4.22).

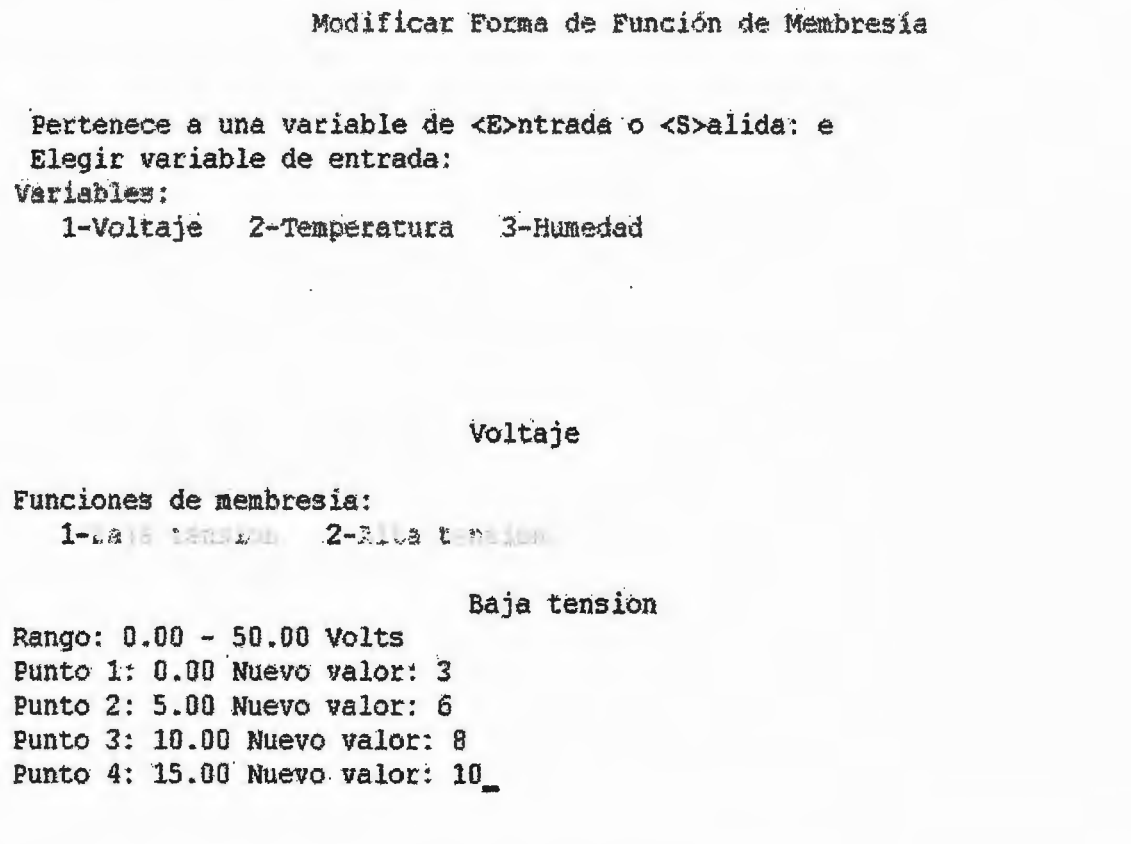


Figura 4.21 Modificar forma de función.

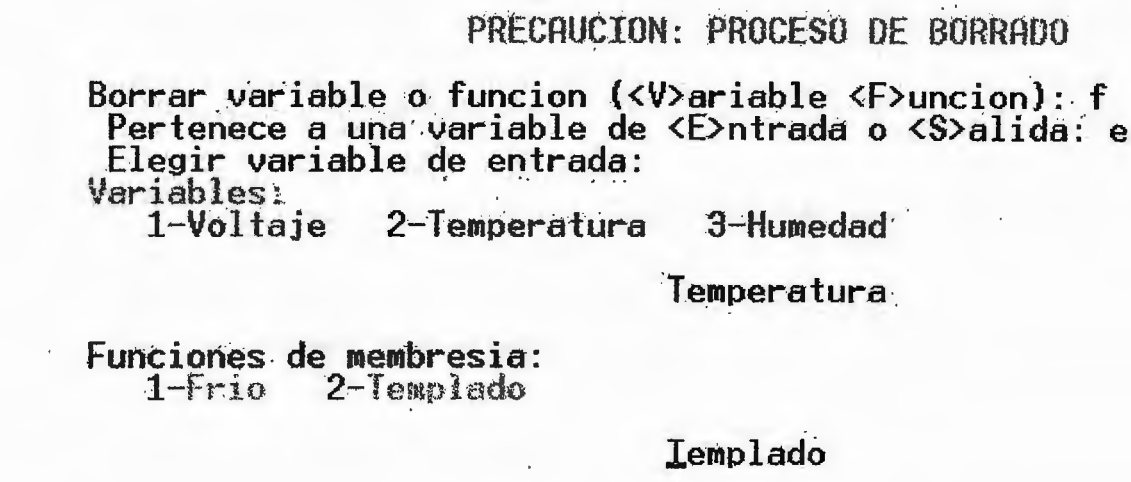


Figura 4.22 Borrar función.

Con toda la información anterior el usuario se encuentra apto para manejar MCFUZLOG.

CAPÍTULO 5: RESULTADOS.

A lo largo del desarrollo de esta tesis se ha recopilado información y resultados que permiten afirmar que lógica difusa tiene una verdadera aplicación en el área de control automático, ampliando, simplificando y mejorando el diseño e implementación de controladores. Se realizaron pruebas con el control difuso y PID. No se utilizaron métodos de sintonización como Ziegler-Nichols para ajustar los controladores PID. El ajuste de controladores PID se realizó basándose en prueba y error hasta obtener un control de buen desempeño. Los datos obtenidos de las pruebas de PID sirvieron como parámetro de evaluación y comparación con los datos del control difuso. A continuación se presentan los resultados obtenidos de los diferentes modelos difusos de control para las plantas de nivel, posición, velocidad y temperatura.

Para cada planta a controlar, el modelo difuso fue diseñado e implementado usando una o dos variables lingüísticas de entrada, una variable lingüística de salida y a lo sumo quince reglas. A través de prueba y error, las relaciones entre las entradas y salidas se modificaron hasta obtener un control aceptable.

5.1. RESULTADOS DE PRUEBAS CON PLANTA DE NIVEL DE LIQUIDO.

El comportamiento de la planta de nivel en lazo abierto es característico a los sistemas de primer orden (Ver Sección C.6 Sistemas de Primer Orden). El control que gobierna la bomba de la planta de nivel debe provee un flujo de líquido para mantener el nivel en el tanque, contrarrestando la perturbación generada por la válvula de escape.

El modelo difuso[1] esta diseñado para controlar el flujo de líquido de la bomba a través del monitoreo de una señal de error. La señal de error se obtiene de la diferencia entre la señal de referencia y la señal del sensor de presión, el cual a través de la presión ejercida por el líquido acumulado en el tanque produce una señal que indica el nivel del líquido en el tanque.

Como puede verse en la Figura 5.1, la variable lingüística Error tiene tres funciones de membresía: error_positivo, error_cero y error_negativo. La señal de error se categoriza de la siguiente forma:

- Si señal del sensor es mayor que la señal de referencia, entonces el valor crisp esta dentro de la función de membresía error_positivo.
- Si señal del sensor es menor que la señal de referencia, entonces el valor crisp esta dentro de la función de membresía error_negativo.
- Si ambas señales son iguales, entonces el valor crisp esta dentro de la función de membresía de error_cero.

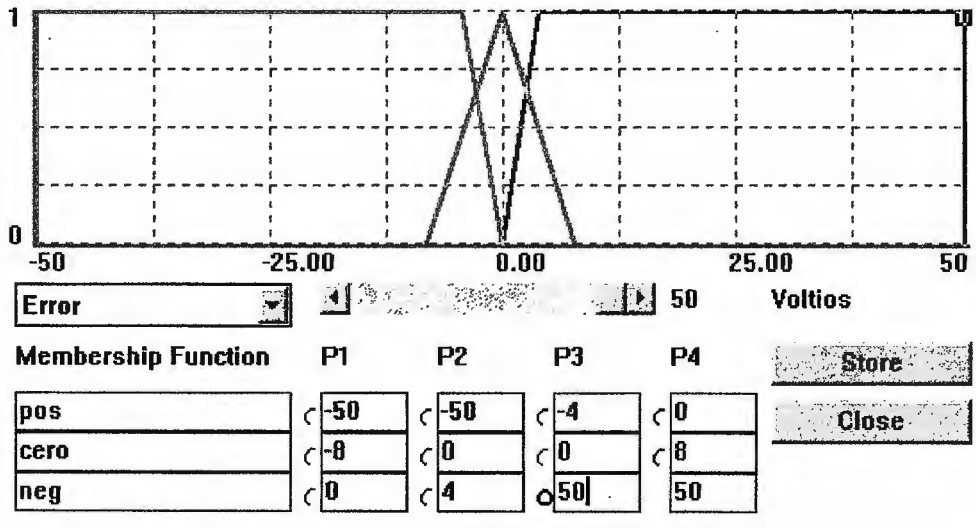


Figura 5.1 Definición de variable lingüística Error.

[1] El simulador FUDGE fue utilizado para el diseño de los modelos difusos. Los rangos definidos para las variables permite ocupar valores fraccionarios (leer sección 4.2 FUDGE).

La bomba no es capaz de succionar, solamente puede tomarse la acción de suministrar o cortar el flujo de líquido. Por esta razón se definió una variable lingüística, Bomba, con dos funciones de membresía: llenar y cortar[2] (ver Figurar 5.2).

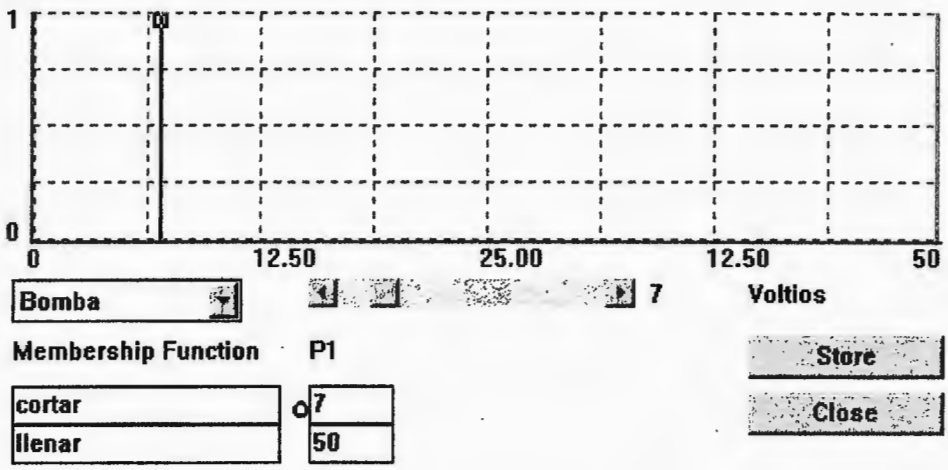


Figura 5.2 Definición de variable lingüística Bomba.

La función singleton cortar está desplazado hacia la derecha del cero para proveer a los actuadores un voltaje de 2V aproximadamente para vencer la inercia de la bomba. Abajo de este voltaje la bomba no funciona.

El comportamiento de la planta se resume a continuación:

- Cuando el error es positivo, entonces apagar la bomba para eliminar el fluyo de líquido entrante.
- Cuando el error es negativo, entonces activar la bomba para suministrar un flujo de líquido.
- Cuando el error es cero, entonces mantener la bomba apagada.

La anterior información del comportamiento de la planta se incorporó al modelo difuso a través del conjunto de reglas mostrado en la Tabla 5.1.

Antecedente	Consecuente
Si Error es Error_positivo	Bomba es Cortar
Si Error es Error_cero	Bomba es Cortar
Si Error es Error_negativo	Bomba es Llenar

Tabla 5. 1 Reglas difusas para gobernar la planta de nivel.

El control que ejerce el modelo difuso puede apreciarse en la Figura 5.3. En la superficie de control se observa la relación entre la señal de error y la señal de control sobre la bomba.

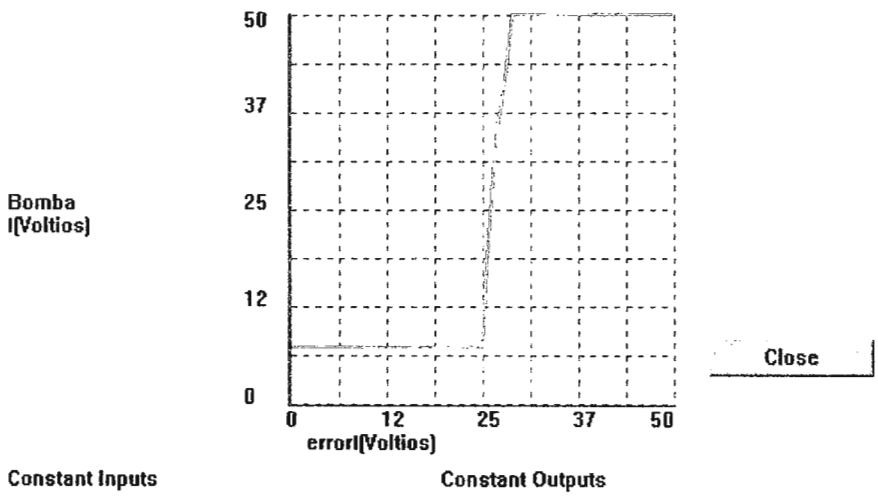


Figura 5.3 Superficie de control para el sistema de control difuso de nivel.

[2] El rango de valores indicado en FUDGE permite incluir valores fraccionarios de la entrada crisp. En las demás definiciones de variables se modifica el rango con la misma intención.

En la Figura 5.4 se muestra las conexiones del modulo acoplador de señales para la implementación del modelo difuso.

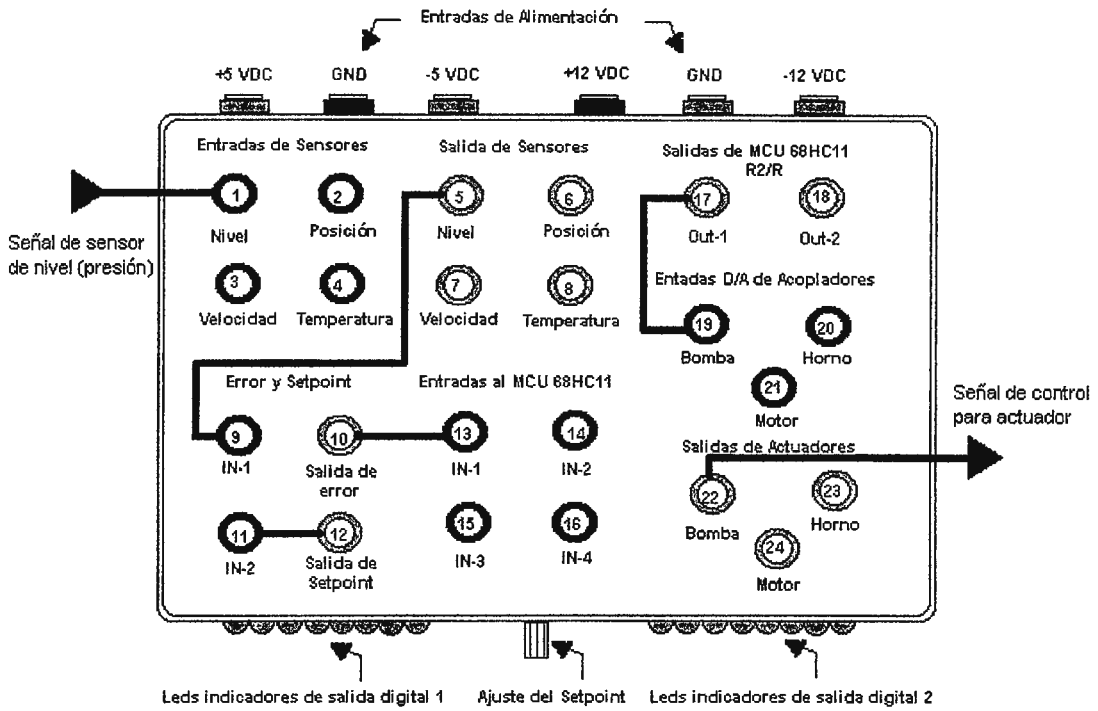


Figura 5.4 Conexión de Acoplador.

A continuación se muestran gráficas que ilustran el comportamiento de la planta gobernada por el microcontrolador programado con el modelo difuso anteriormente descrito. En la primera prueba la válvula de escape fue cerrada completamente para visualizar la rapidez de llenado al ajustar el valor de referencia a 100mm, 200mm, 300mm, 400mm y 500mm (ver Figura 5.5, el rango del eje vertical de 0 a 8 voltios corresponde a los niveles de 0mm y 500mm).

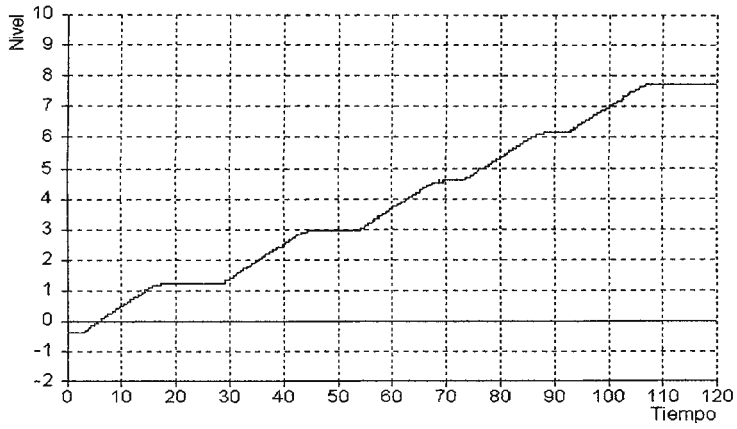


Figura 5.5 Salida de sistema de nivel bajo control difuso para diferentes valores de referencia y con perturbación nula (válvula de escape cerrada). Cada región plana arriba de cero corresponde a 100mm, 200mm, 300mm, 400mm y 500 de líquido.

Al someterse la planta de líquido a la perturbación provocada por la abertura de la válvula de escape, se obtuvo la gráfica mostrada en la Figura 5.6. Al fijar el valor de referencia para cierto nivel y manteniendo la válvula de escape cerrada, el control difuso exige el máximo de caudal a la bomba para rápidamente llegar al nivel deseado. Próximo al punto de referencia, el controlador paulatinamente reduce el caudal para que no sobrepasar demasiado el nivel deseado. Si el controlador no reaccionara adecuadamente, recordando que la bomba no tiene capacidad de succión, el error sería demasiado.

El controlador difuso mantuvo el sobreimpulso menor al 12%. Al abrir totalmente la válvula, el control difuso reacciona inmediatamente para generar un flujo de líquido igual al flujo de escape. La gráfica de la Figura 5.6 muestra la variación de la salida para diferentes valores de referencia sometidos a perturbaciones extremas. El resultado fue un error no mayor de 5% al someter el sistema a la perturbación máxima (válvula totalmente abierta). Como se aprecia en la gráfica, el nivel baja un poco y lo mantiene. El controlador al sensar una baja en el nivel genera un voltaje suficiente para forzar un mayor flujo de la bomba para reponer el flujo que escapa por la válvula.

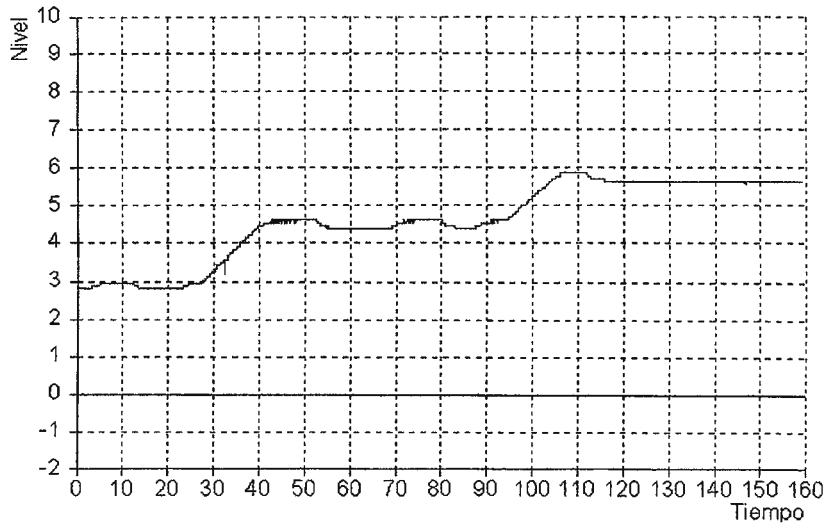


Figura 5.6 Salida de sistema de nivel bajo control difuso sometido a perturbación mínima (válvula cerrada) y perturbación máxima (válvula abierta). La gráfica muestra las variaciones para los niveles de 200mm, 300mm y 380mm.

Con el control PID se obtuvo el gráfico mostrado en la Figura 5.7. Como se aprecia en la gráfica, el margen de error aumenta según más alto se haya ajustado el punto de referencia.

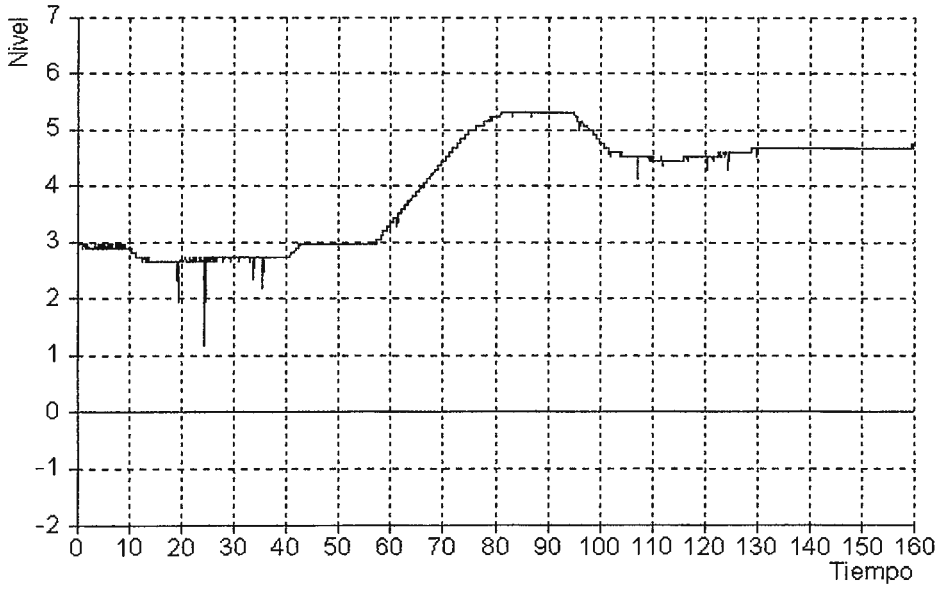


Figura 5.7 Salida de sistema de nivel bajo control PID (P=38 Ti=max Td=max) sometido a perturbación mínima y máxima para 200mm y 380mm.

5.2. RESULTADOS DE PRUEBAS CON PLANTA DE POSICIÓN.

La planta de posición cuenta con un disco giratorio que se desplaza en proporción al voltaje aplicado al motor. El comportamiento del sistema es característico a los sistemas de segundo orden (ver sección C.7 Sistemas de Segundo Orden). El control debe desplazar el disco hasta una posición indicada por la señal de referencia. El modelo difuso que se implementó para el control de esta planta puede apreciarse en las Figuras 5.8, 5.9 y 5.10.

Al igual que la anterior planta de nivel, el control se logra a través de una señal de error. Para el control difuso de la planta de posición también se definió la variable lingüística error pero con un conjunto de funciones de membresía diferente. Esta variable representa la diferencia entre el valor de referencia del punto destino y la señal del sensor que indica la posición efectiva del indicador.

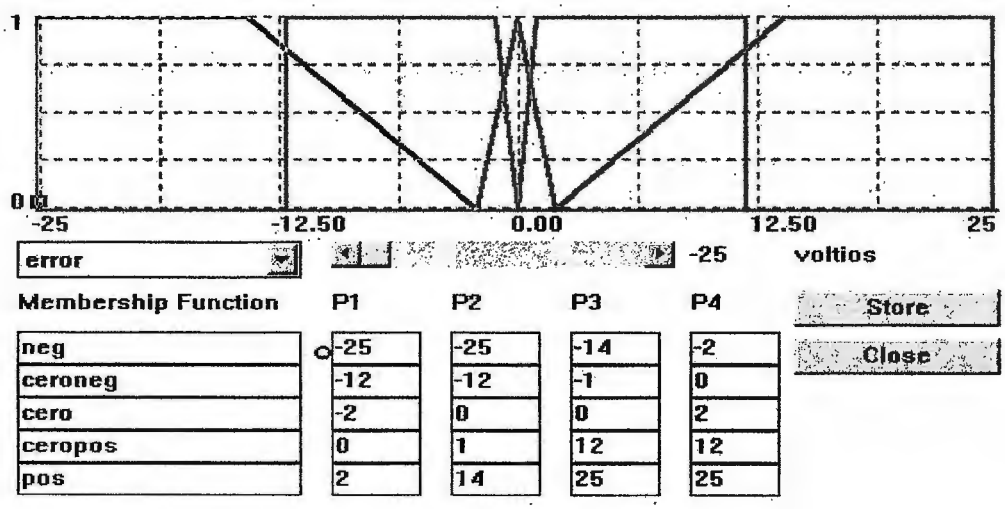


Figura 5.8 Funciones de membresía para la variable lingüística de entrada Error[3].

[3] El rango –25 a 25 representa el rango de voltajes entre –2.5 y 2.5 generado por la señal de error.

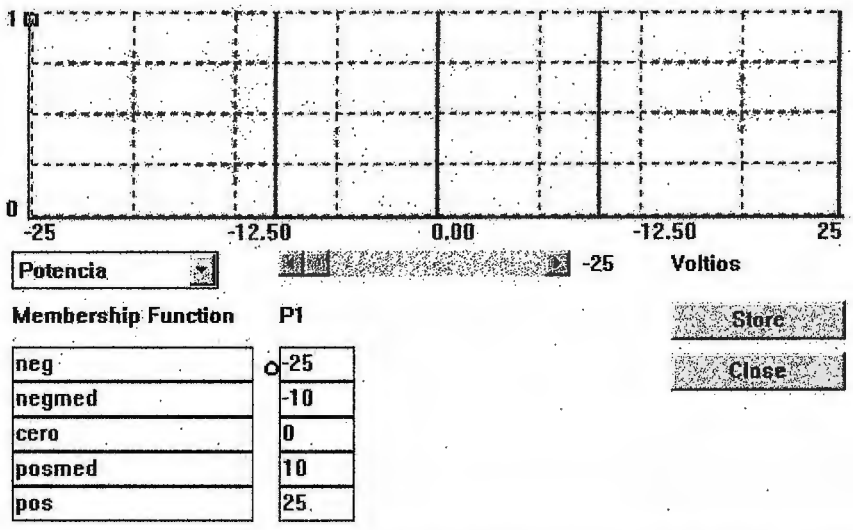


Figura 5.9 Funciones singletons para variable lingüística de salida Potencia.

El comportamiento de la planta se resume a continuación:

- Si el punto inicial del indicador se encuentra muy retirado del punto destino en sentido horario, entonces aplicar máxima aceleración al motor en sentido horario.
- Si el punto inicial del indicador se encuentra próximo al punto destino en sentido horario, entonces acelerar lentamente el motor.
- Si el punto inicial del indicador se encuentra en el punto destino en sentido horario, entonces desacelerar totalmente el motor.
- Si el punto inicial del indicador se encuentra muy retirado del punto destino en sentido antihorario, entonces aplicar máxima aceleración en sentido antihorario.
- Si el punto inicial del indicador se encuentra próximo al punto destino en sentido antihorario, entonces acelerar lentamente motor.

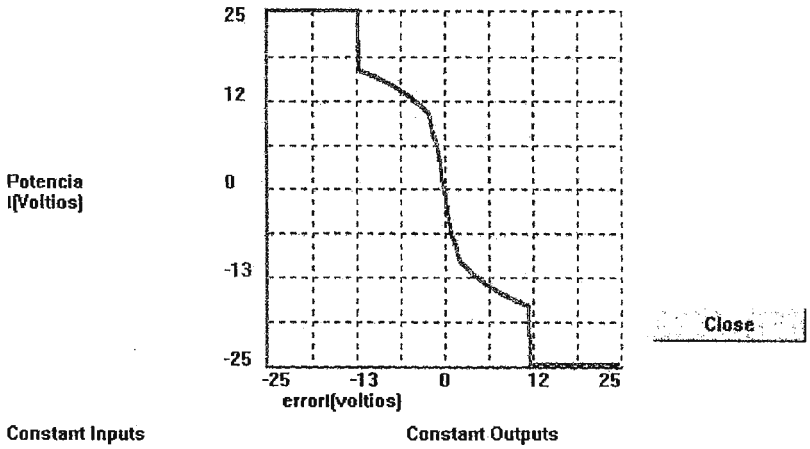


Figura 5.10 Relación Salida/Entrada.

Antecedente	Consecuente
Si Error es Neg	Potencia es Pos
Si Error es Ceroneg	Potencia es Posmed
Si Error es Cero	Potencia es Cero
Si Error es Ceropos	Potencia es Negmed
Si Error es Pos	Potencia es Neg

Tabla 5.2 Reglas difusas para sistema de posición.

En la Figura 5.9 puede apreciarse que las funciones de membresías tienen formas trapezoidales o triangulares. La forma de estas funciones junto con las funciones de salida y las reglas están íntimamente ligadas para formar la superficie de control que se muestra en la Figura 5.10.

La forma de la superficie de control se estableció de esta manera para contrarrestar los sobreimpulsos generados por las altas velocidades de desplazamiento cuando se corrige el error. Si el error es demasiado grande, el controlador reacciona acelerando el motor para desplazar el indicador de tal forma que se corrija el error lo más pronto posible. Cuando el error es cero, el motor

lleva una gran velocidad. El proceso de frenado del controlador es muy lento para detenerlo inmediatamente y por ello se ocasiona el sobreimpulso.

Por la anterior razón se adoptó un modelo difuso con la relación entrada/salida de la Figura 5.10. Esta relación permite acelerar el sistema al máximo pero al aproximarse al punto de reposo hay una desaceleración abrupta para minimizar el sobreimpulso.

El modulo acoplador para implementar el modelo difuso se conectó como se muestra en la Figura 5.11.

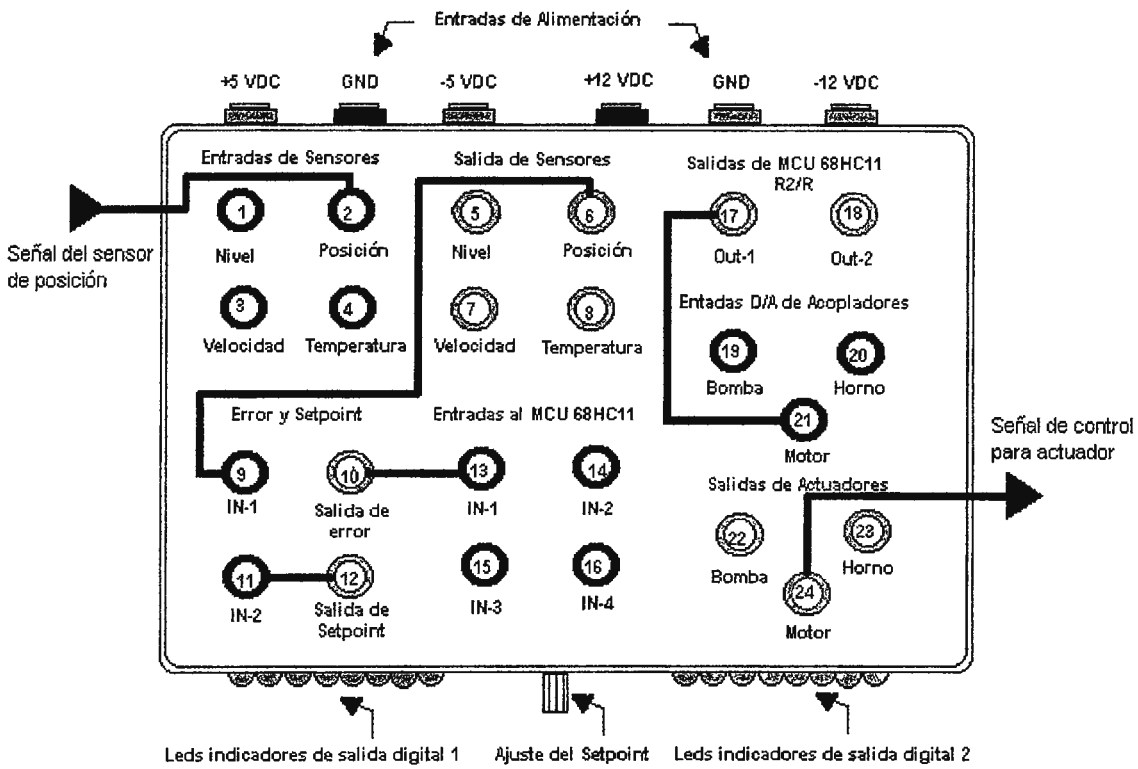


Figura 5.11Conexión de acoplador.

En la Figura 5.12 y 5.13 se aprecian las respuestas del control difuso y control PID, respectivamente.

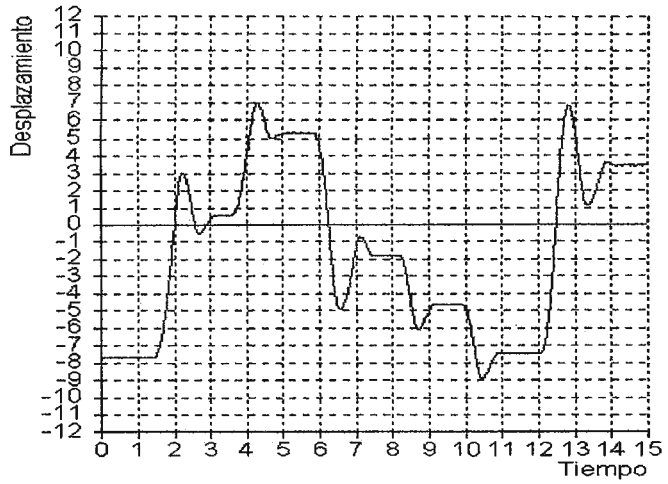


Figura 5.12 Salida de sistema de desplazamiento bajo control difuso para diferentes puntos de referencia

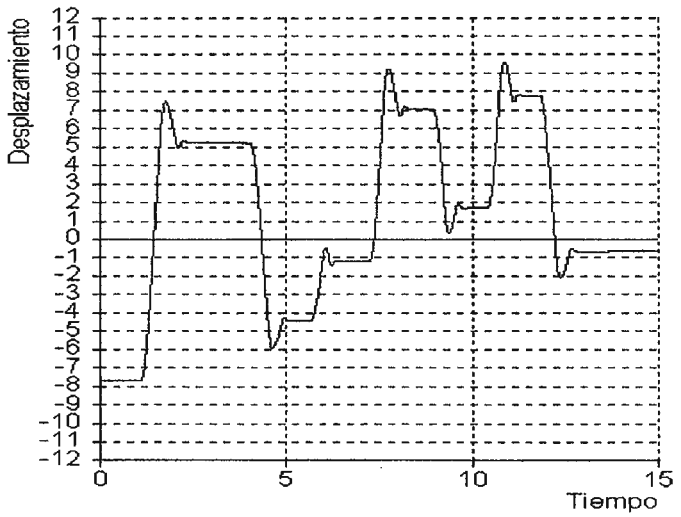


Figura 5.13 Gráfica de desplazamiento usando control PID.

El eje vertical de ambas figuras anteriores tiene un rango de –12 a 12, donde el rango de –8 a 8 corresponde al rango de 0 a 360 grados. De la Figura 5.12 se observa que el sobreimpulso del sistema bajo control difuso se incrementa de acuerdo a la distancia a recorrer para llegar al nuevo punto de referencia. Para el último sobreimpulso mostrado en la gráfica, el punto de referencia estando en 0 repentinamente se ubico en 252 grados. El controlador difuso responde con una aceleración brusca que al llegar al punto destino el frenado no es suficiente y

sobrepasa en más de 80 grados al punto de reposo. Como se aprecia para los otros puntos de referencia, el sobreimpulso para el modelo difuso implementado es proporcional al cambio súbito del punto de referencia. El sistema bajo control PID tiene menor sobreimpulso (ver Figura 5.13).

5.3. RESULTADOS DE PRUEBAS CON PLANTA DE VELOCIDAD.

El comportamiento es característico a los sistemas de segundo orden. El modelo difuso para controlar la planta de velocidad considera dos variables de entrada: la señal del sensor de velocidad y la señal de error. Con estas dos variables se controla la variable de salida que acciona el motor. En las Figuras 5.14, 5.15 y 5.16 puede apreciarse la configuración de las variables lingüísticas de entrada y salida.

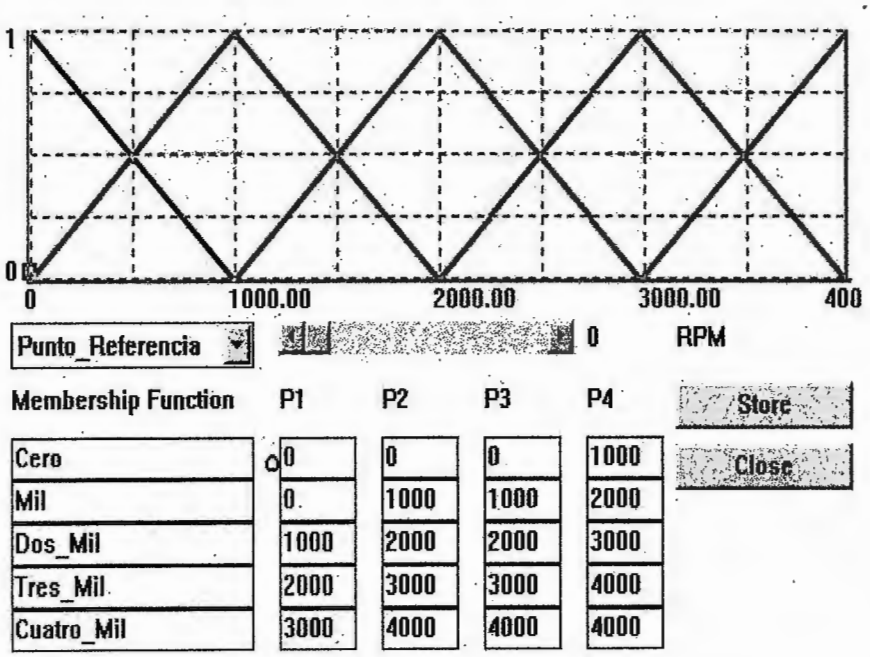


Figura 5.14 Variable lingüística de entrada Punto_referencia.

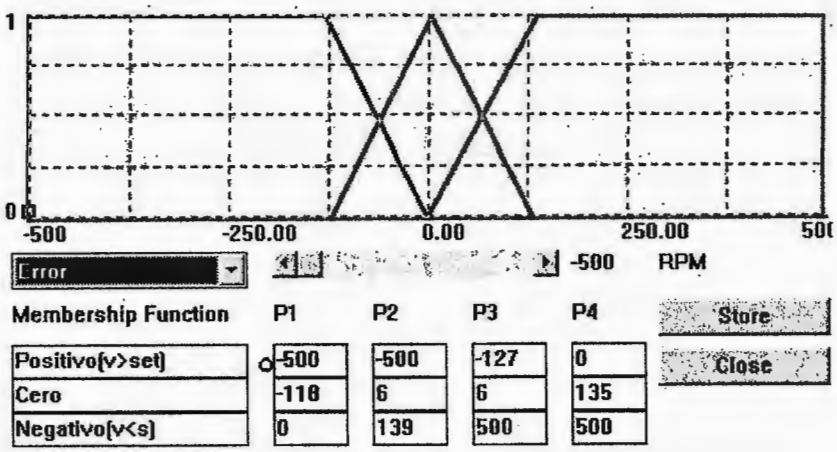


Figura 5.15 Variable lingüística de entrada Error.

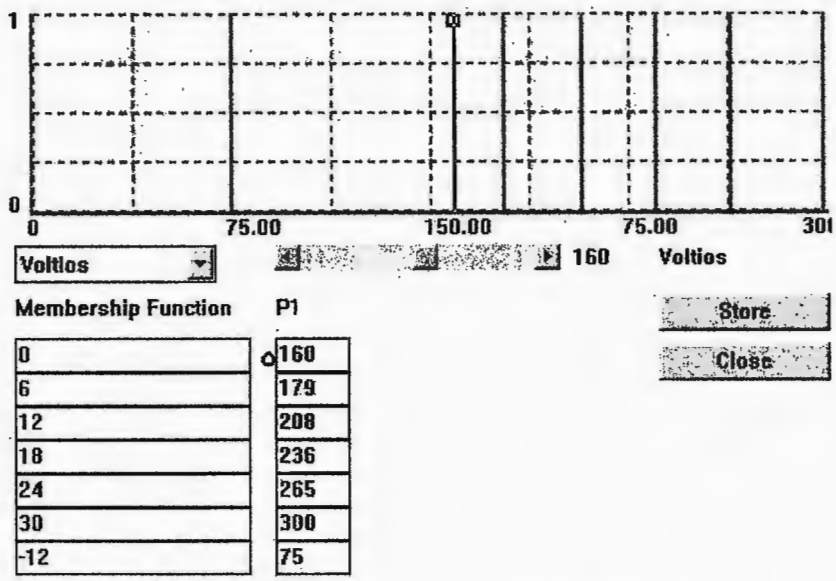


Figura 5.16 Variable lingüística de salida Voltios.

La señal de salida tiene los singletons concentrados en la parte derecha. Esto se debe a que el modelo se ha diseñado para controlar la velocidad hacia un sentido ya que el acoplador de la señal del sensor de velocidad para el microcontrolador fue diseñado solamente para responder a movimientos giratorios hacia un solo sentido. El único singleton que aparece en el lado izquierdo se utiliza para ejercer una acción de frenado.

Para comprender mejor el diseño de las variables lingüísticas, se procede a resumir el comportamiento de la planta:

- Para mantener una velocidad constante debe de aplicarse un voltaje constante al motor.
- Si el sistema esta libre de perturbación (freno), para cada velocidad corresponde un nivel de voltaje.
- Al aplicarse el freno, debe aplicarse un mayor voltaje al motor para mantener la velocidad.
- Si la velocidad excede el nivel fijado por el punto de referencia, entonces debe aplicarse un voltaje para desacelerar el motor.
- Si la velocidad es menor al nivel fijado por el punto de referencia, entonces debe aplicarse un voltaje para acelerar el motor.

De acuerdo a lo anterior, la variable lingüística Punto_referencia se diseño para proveer el voltaje necesario que mantenga la velocidad constante cuando el sistema no es sometido a la perturbación de frenado. Cada punto de referencia corresponde a una velocidad, por tanto hay un voltaje para cada velocidad.

Cuando el sistema se somete a la perturbación de frenado, la variable lingüística Error entra en acción. El objetivo del control difuso es proveer el suficiente voltaje para no alterar la velocidad, contrarrestando los efectos de la perturbación. La señal crisp de la variable Error proviene de la diferencia entre el punto de referencia y el sensor de velocidad. Con esta señal de error el control difuso, a través de la evaluación de reglas mostradas en la Tabla 5.3, compensa el voltaje suministrado al motor añadiendo mayor voltaje.

Las superficies de control para este modelo difuso no pudieron realizarse con FUDGE. Sin embargo a través del Evaluador Difuso de FUDGE se obtuvieron valores cuyas gráficas se muestran en las Figuras 5.17 y 5.18.

El modulo acoplador para implementar el modelo difuso se conectó como se muestra en la Figura 5.19.

Antecedente	Consecuente
Si Punto_Referencia es Cero y Error es Positivo	Voltios es –12
Si Punto_Referencia es Cero y Error es Cero	Voltios es 0
Si Punto_Referencia es Cero y Error es Negativo	Voltios es 30
Si Punto_Referencia es Mil y Error es Positivo	Voltios es –12
Si Punto_Referencia es Mil y Error es Cero	Voltios es 6
Si Punto_Referencia es Mil y Error es Negativo	Voltios es 30
Si Punto_Referencia es Dos_mil y Error es Positivo	Voltios es –12
Si Punto_Referencia es Dos_mil y Error es Cero	Voltios es 12
Si Punto_Referencia es Dos_mil y Error es Negativo	Voltios es 30
Si Punto_Referencia es Tres_mil y Error es Positivo	Voltios es –12
Si Punto Referencia es Tres_mil y Error es Cero	Voltios es 18
Si Punto_Referencia es Tres_mil y Error es Negativo	Voltios es 30
Si Punto_Referencia es Cuatro_mil y Error es Positivo	Voltios es –12
Si Punto_Referencia es Cuatro_mil y Error es Cero	Voltios es 24
Si Punto_Referencia es Cuatro_mil y Error es Negativo	Voltios es 30

Tabla 5.3. Reglas del modelo difuso para la planta de velocidad.

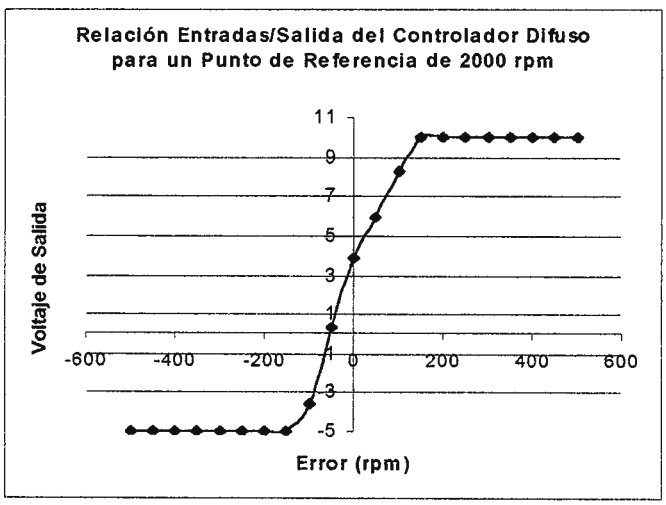


Figura 5.17 Relación Entradas/Salida cuando el punto de referencia es 3000rmp.

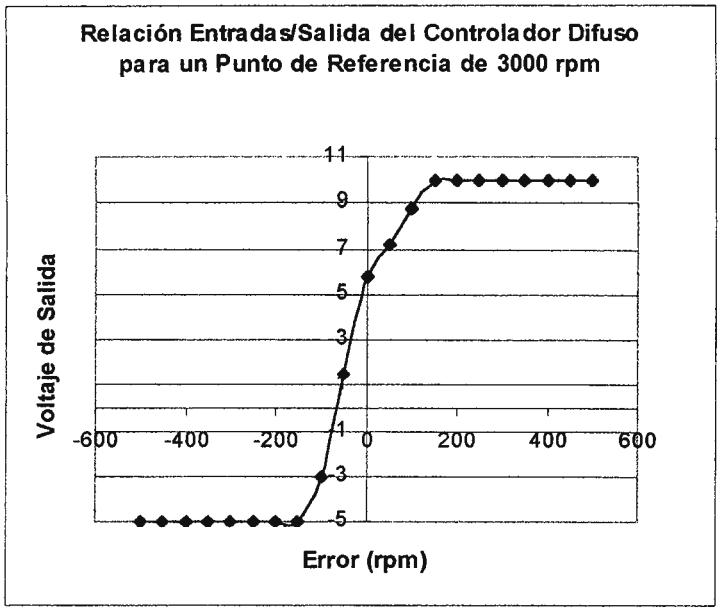


Figura 5.18 Relación Entradas/Salida cuando el punto de referencia es 3000rmp.

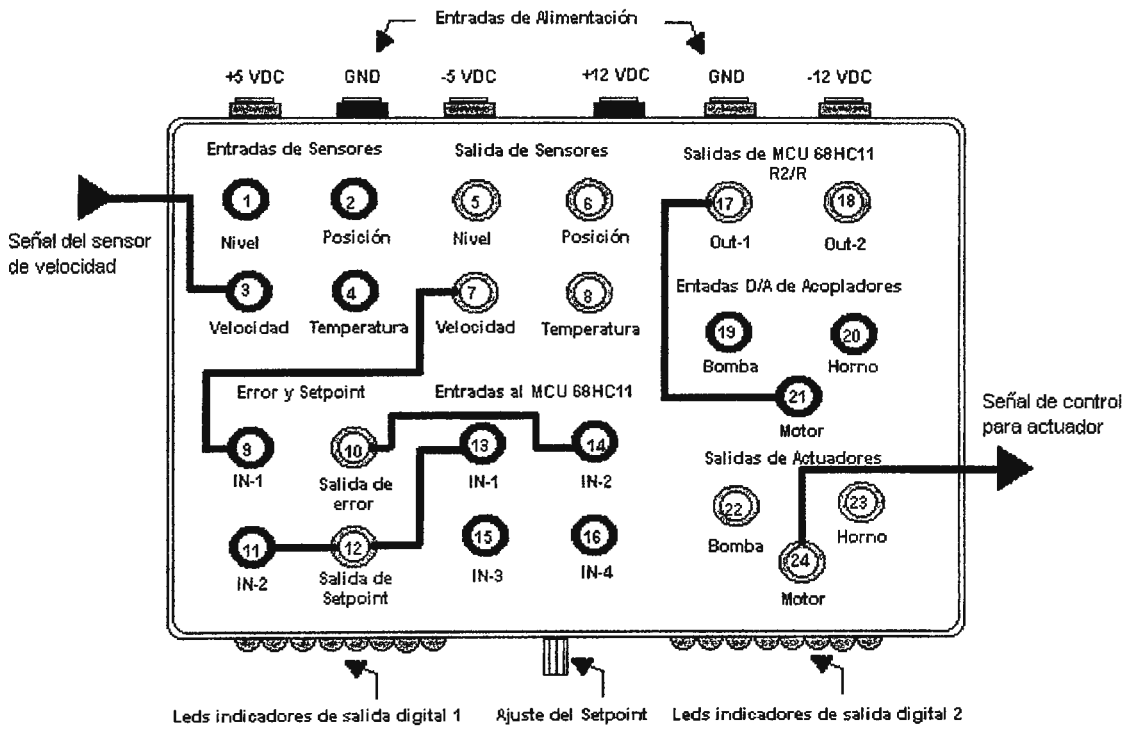


Figura 5.19 Conexión de acoplador.

Como puede observarse en las Figuras 5.17 y 5.18, el modelo difuso ante perturbaciones responde con un mayor voltaje a la salida para contrarrestar sus efectos y mantener la salida invariable. En las Figuras 5.20[4], se observa como el controlador difuso responde para diferentes puntos de referencia. Si el voltaje del sensor es mayor al del punto de referencia (área a la izquierda del cero), la salida es un voltaje negativo que desacelerar el motor. Si el voltaje del sensor es menor al del punto de referencia (área a la derecha del cero), la salida es un voltaje positivo que acelera el motor.

Cuando el sistema se fija para una velocidad de 3000 rpm y se somete a perturbaciones de frenado, el sistema bajo control difuso responde como se muestra en la Figura 5.21. Al presentarse la acción de frenado, la señal de error aumenta. El controlador difuso procesa el error y evalúa las reglas para generar un voltaje que mantenga al motor a la misma velocidad o al menos no permitir que baje demasiado. Cuando la acción de frenado es muy grande el motor cede y baja la velocidad. El control difuso para esta situación responde con el máximo voltaje pero esto no brinda la suficiente potencia al motor para contrarrestar el frenado. Lo mismo sucede para el control PID (ver Figura 5.22 y 5.23). Al comparar las dos gráficas puede verse que el control difuso ofrece un control muy similar al PID.

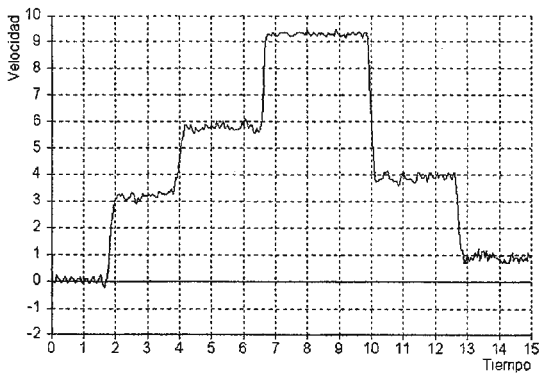


Figura 5.20 Salida de sistema de velocidad bajo control difuso sin perturbación para varios puntos de referencia.

[4] El rango del eje vertical de 0 a 10 corresponde al rango de 0rpm a 5000 rpm. Esta conversión también se aplica a las otras gráficas de esta sección.

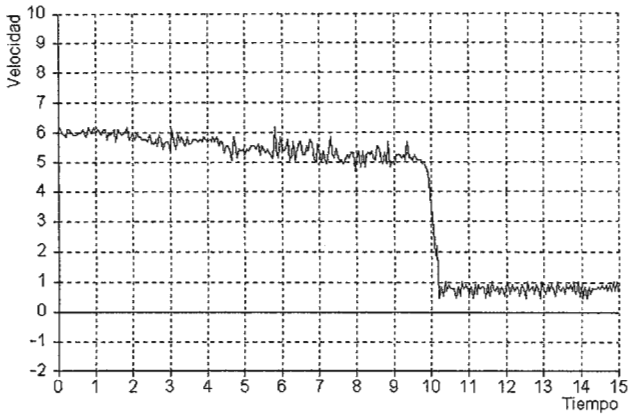


Figura 5.21 Sistema de velocidad bajo control difuso con un punto de referencia de 3000 rpm sometido a una perturbación de frenado.

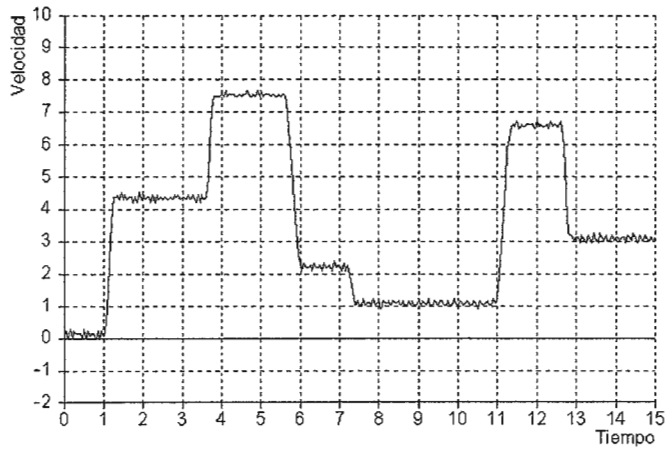


Figura 5.22 Sistema de velocidad bajo control PID para diferentes puntos de referencia.

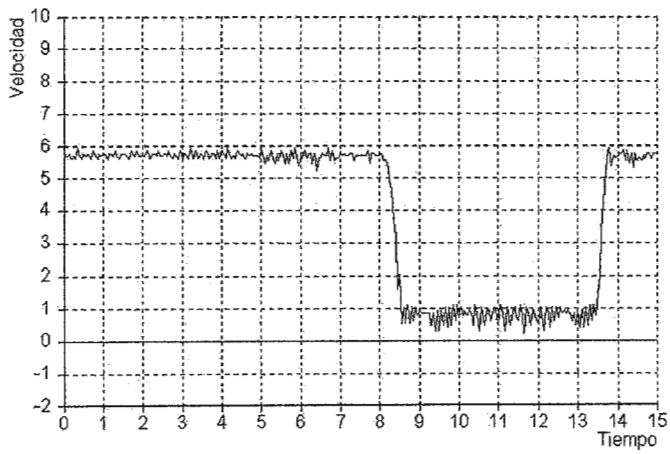


Figura 5.23 Sistema de velocidad bajo control PID fijado a un punto de referencia de 3000 rpm y sometido a una perturbación de frenado.

5.4. RESULTADOS DE PRUEBAS CON PLANTA DE TEMPERATURA.

El comportamiento de la planta de temperatura es característico de sistemas de primer orden. Para esta planta solamente se utilizó el sensor termocupla. El diseño del modelo difuso es similar al control difuso del sistema de velocidad. La diferencia esta en la declaración de las funciones de membresía para la variable de Error y el corrimiento del singleton –12 de la variable Voltios.

Igual que en la planta de velocidad, la variable Punto_Referencia proporciona información al controlador difuso para fijar un voltaje que caliente la resistencia en proporción a un punto de referencia establecido. La variable Error indica al controlador si la salida relacionada con la variable Punto_Referencia necesita mayor/menor voltaje para contrarrestar alguna perturbación presente en el sistema. La señal de Error crisp se obtiene de la diferencia entre la señal de la termocupla y la señal del punto de referencia. Las Figuras 5.24, 5.25 y 5.26 muestran las definiciones de las variables para el modelo difuso.

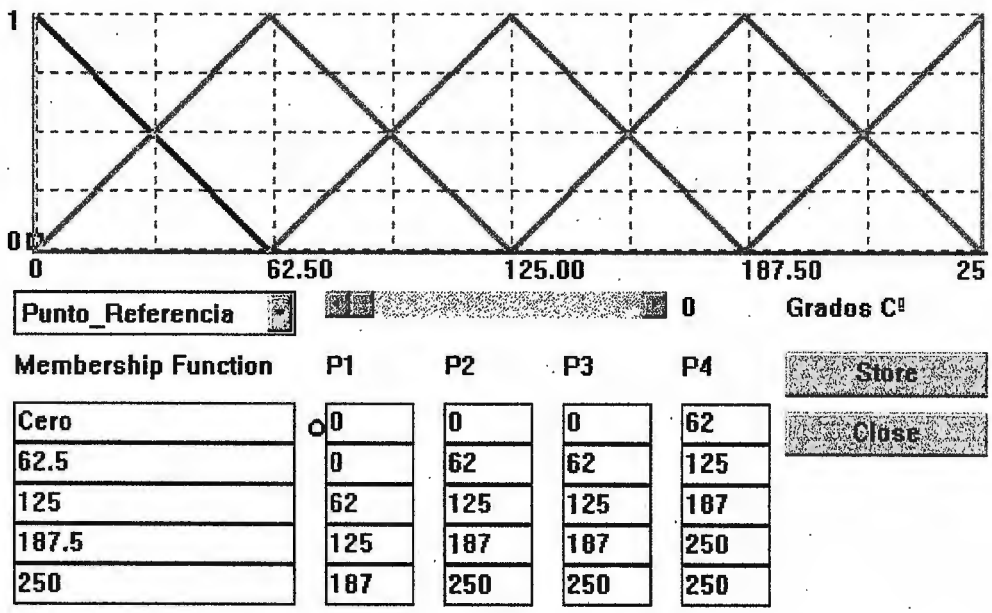


Figura 5.24 Definición de la variable lingüística Punto_Referencia.

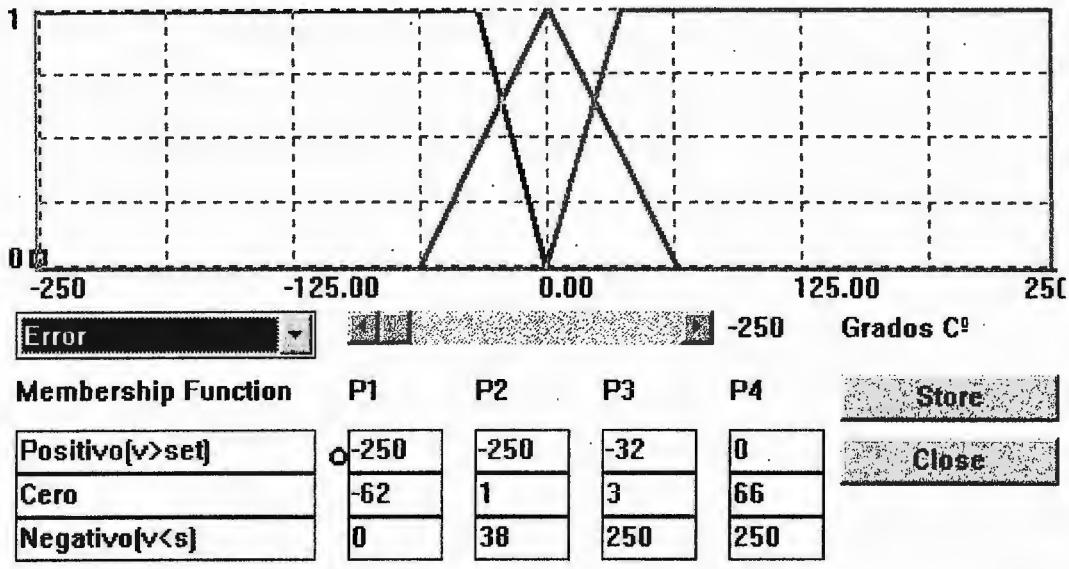


Figura 5.25 Definición de la variable lingüística Error.

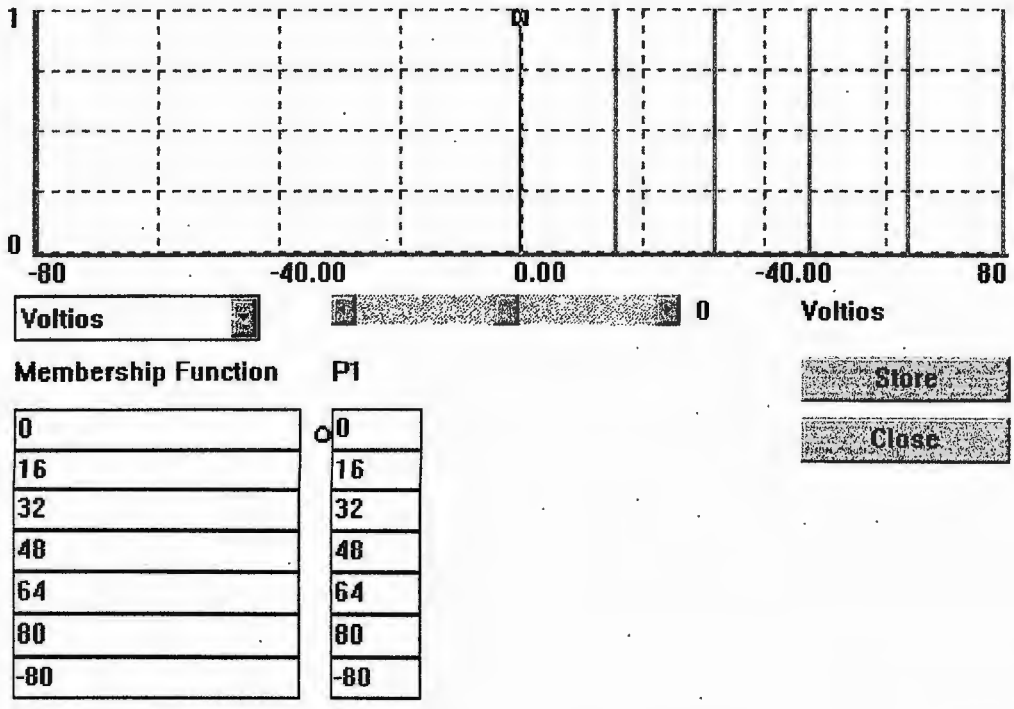


Figura 5.26 Definición de la variable lingüística Voltios.

El comportamiento de la planta se resume a continuación:

- Para mantener la temperatura estable, es necesario mantener un voltaje constante proporcional al punto de referencia.
- Si la temperatura del horno es menor al indicado por el punto de referencia, entonces aumentar el voltaje de la resistencia.
- Si la temperatura del horno es mayor al indicado por el punto de referencia, entonces activar el ventilador para enfriar la resistencia.

Con esta información se generaron las reglas para el modelo difuso (ver Tabla 5.4).

Antecedente	Consecuente
Si Punto_Referencia es 0 y Error es Positivo	Voltios es –80
Si Punto_Referencia es 0 y Error es Cero	Voltios es 0
Si Punto_Referencia es 0 y Error es Negativo	Voltios es 80
Si Punto_Referencia es 62.5 y Error es Positivo	Voltios es –80
Si Punto_Referencia es 62.5 y Error es Cero	Voltios es 16
Si Punto_Referencia es 62.5 y Error es Negativo	Voltios es 80
Si Punto_Referencia es 125 y Error es Positivo	Voltios es –80
Si Punto_Referencia es 125 y Error es Cero	Voltios es 32
Si Punto_Referencia es 125 y Error es Negativo	Voltios es 80
Si Punto_Referencia es 187.5 y Error es Positivo	Voltios es –80
Si Punto_Referencia es 187.5 y Error es Cero	Voltios es 48
Si Punto_Referencia es 187.5 y Error es Negativo	Voltios es 80
Si Punto_Referencia es 250 y Error es Positivo	Voltios es –80
Si Punto_Referencia es 250 y Error es Cero	Voltios es 64
Si Punto_Referencia es 250 y Error es Negativo	Voltios es 80

Tabla 5.4 Reglas difusas para sistema de temperatura.

Las Figuras 5.27 y 5.28 muestran la relación entradas/salida del sistema de temperatura bajo control difuso. El comportamiento es similar a la planta de velocidad. Dependiendo del punto de referencia, los voltajes para corrección de error se desplazan sobre el eje que corresponde a la variable de salida para compensar la perturbación provocada por el ventilador de enfriamiento.

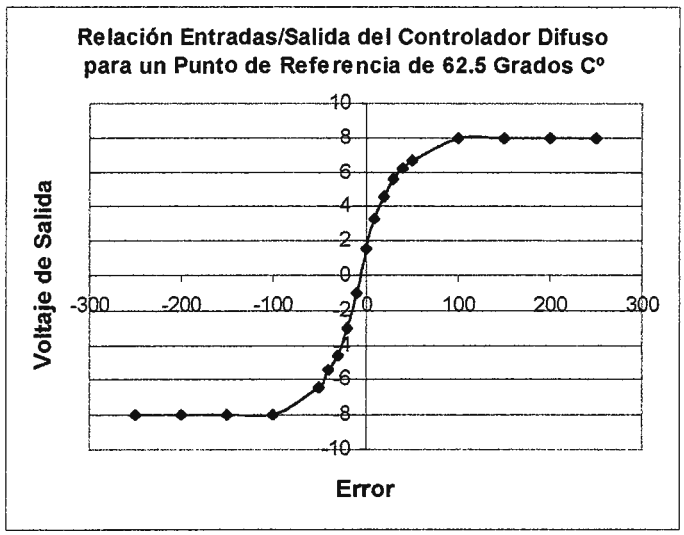


Figura 5.27 Relación entradas/salida de sistema de temperatura bajo control difuso.

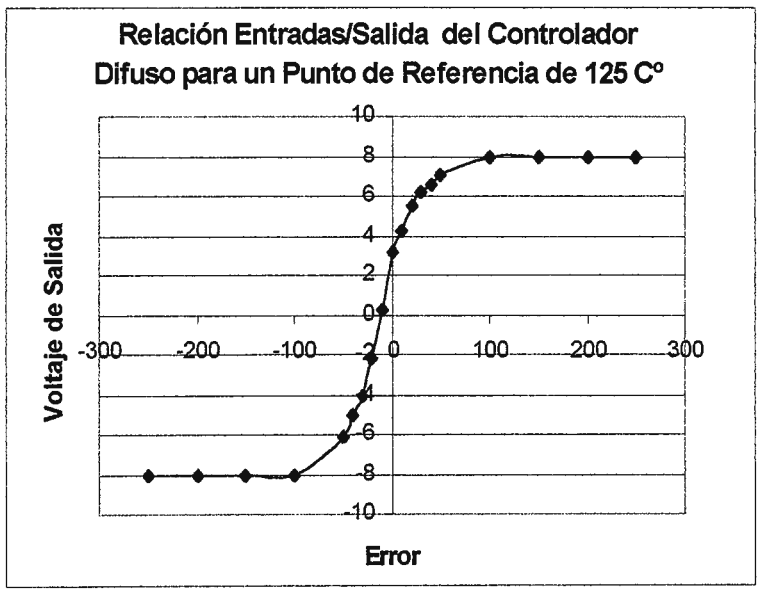


Figura 5.28 Relación entradas/salida de sistema de temperatura bajo control difuso.

El módulo acoplador para implementar el modelo difuso se conectó como se muestra en la Figura 5.29.

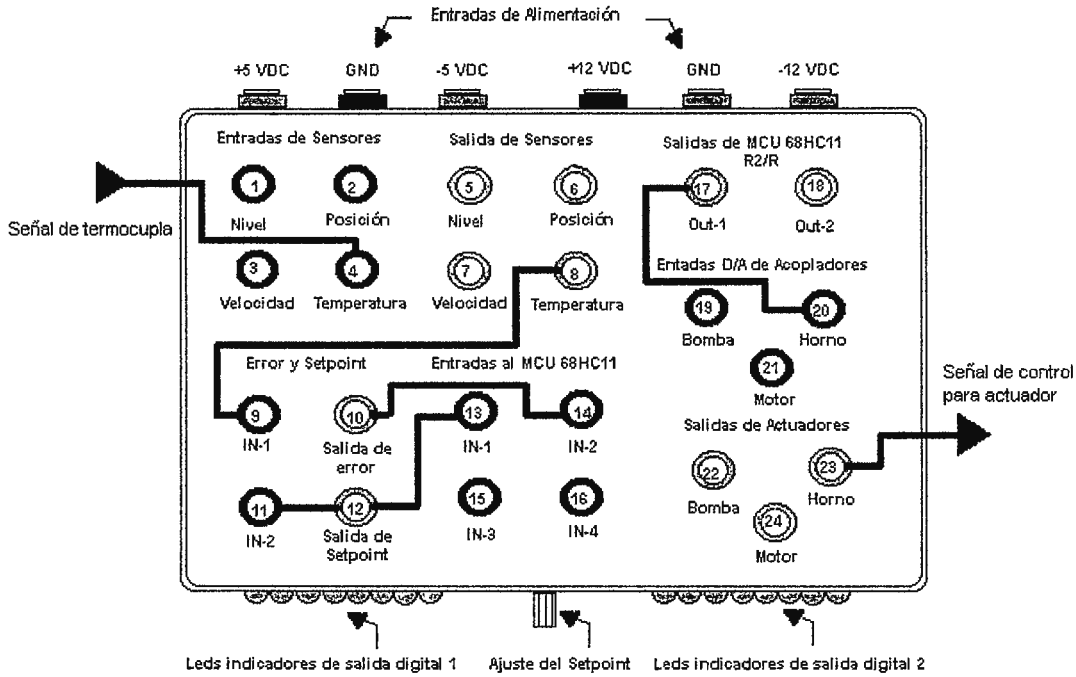


Figura 5.29 Conexión de acoplador.

El control difuso hace que el comportamiento del sistema sea como se ilustra en las Figuras 5.30 y 5.31, sin perturbación y con perturbación. (El rango del eje vertical entre 0 y 8 corresponde al rango de temperatura de 0 hasta 250).

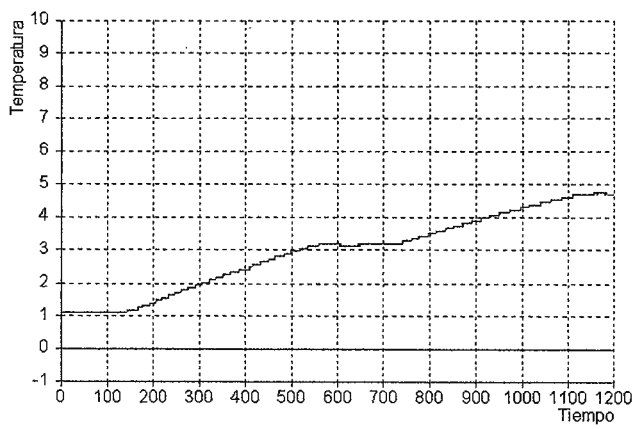


Figura 5.30 Salida del sistema de temperatura bajo control difuso. No existe perturbación. Debido a la gran inercia del sistema, solamente se ha graficado para puntos de referencia que corresponden a 100C° (3.2 Voltios) y 150C° (4.8 Voltios).

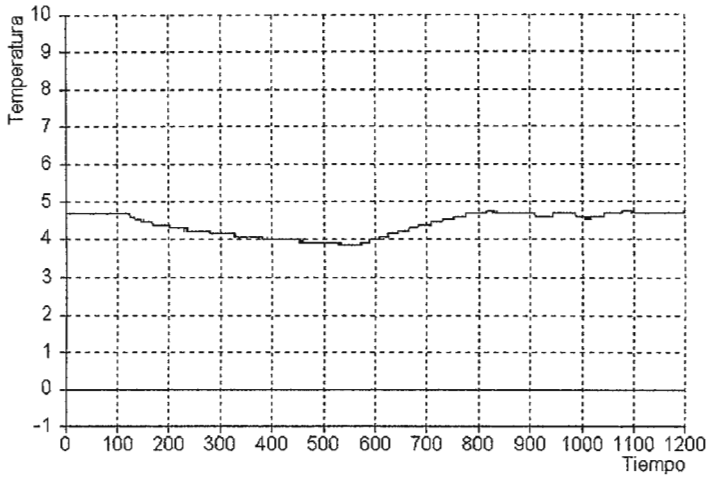


Figura 5.31 Salida de sistema de temperatura sometido a perturbación cuando el punto de referencia es de 150C°.

Tanto para el control difuso y control PD[5] no son capaces de reponerse ante una perturbación permanente. El enfriamiento constante excede la capacidad calorífica del horno. Pero cuando la perturbación es de corta duración, el control difuso y el control PD funcionan adecuadamente. El control difuso puede mantener constante la temperatura en el horno, mientras que el PD mantiene una oscilación pequeña alrededor del punto de referencia.

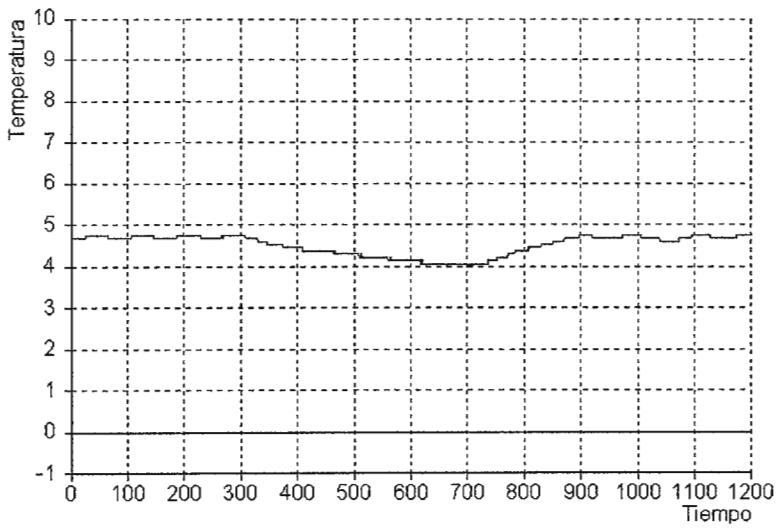


Figura 5.32 Salida del sistema de temperatura bajo control PD (P=max y D=max) sometido a perturbación cuando el punto de referencia es de 150C°.

[5] El componente integral del controlador es un valor fijo. Al realizar pruebas con el PID, el componente integral retarda la reacción del sistema. Por ello, se utilizó solamente las etapas P y D.

Al lado izquierdo de ambas gráficas, 5.31 y 5.32, se muestra el valor de establecimiento para 150C° cuando no hay perturbación; en la parte media se muestra el efecto que tiene una perturbación de 450 segundos; finalmente se presenta la salida para perturbaciones de corta duración. El control difuso permite al sistema recuperarse sin que se generen oscilaciones. El control PD tiene siempre una oscilación permanente.

CAPÍTULO 6: CONCLUSIONES Y RECOMENDACIONES.

Durante el desarrollo de la tesis se ha tenido la oportunidad de conocer y aplicar una nueva tecnología en control automático. La investigación bibliográfica y las pruebas realizadas con el prototipo de controlador inteligente basado en lógica difusa permitió adquirir más conocimientos al contraponer el estudio teórico con el práctico. A continuación se presentan las conclusiones y recomendaciones.

6.1. CONCLUSIONES.

Al cursar asignaturas de control automático se requiere un gran esfuerzo mental. Para entender totalmente los conceptos es imperioso hacer análisis matemáticos complejos. Una de las grandes inquietudes era si el entendimiento de lógica difusa estaría subordinada a un riguroso análisis matemático. La experiencia con la tesis permite concluir que el estudio de lógica difusa requiere menos esfuerzo que otras técnicas de control automático clásico o moderno. Esta aseveración la sustenta el hecho que lógica difusa adapta la matemática al proceso de control en vez de someter el control a esquemas matemáticos rígidos. Con las pruebas de laboratorio se constató lo anterior. El control ejercido con lógica difusa se logró a través de variables con alto contenido de incertidumbre en vez de matemáticas deterministas. Realmente es la representación de la incertidumbre usando conjuntos difusos lo que da un valor único a lógica difusa en aplicaciones de control automático.

Para esta tesis la exigencia matemática se redujo a teoría de conjuntos, matemática matricial y métodos de cálculos de centros de gravedad. El estudio de

lógica difusa para estudiantes o ingenieros que tengan conocimientos sobre control automático resultará más comprensible. El modelo difuso para cada planta se diseño basándose en términos lingüísticos que describían el proceso de control. Como lógica difusa se parece mucho al razonamiento humano, la descripción del modelo a través de reglas IF-THEN permite al diseñador una mejor visión del proceso de control. Las ecuaciones matemáticas muchas veces poseen información que no es fácil de apreciar a primera vista, en cambio las reglas difusas ofrecen mayor información fácil de asimilar. Al hacer las pruebas con los controladores PID resulto difícil establecer los factores proporcionales, integrales y derivativos óptimos. Sin embargo la facilidad lingüística que ofreció lógica difusa permitió entender mejor la relación entre las variables del modelo, con lo cual el proceso de optimización se volvió más placentero.

Las pruebas de laboratorio permitieron probar conceptos del control difuso. Como al principio de la tesis no se conocía los alcances de lógica difusa, las pruebas no explotan el potencial del control difuso. Sin embargo la facilidad de incorporar nuevas variables y modificar comportamientos de control permite concluir que el control difuso es versátil y capaz de adaptarse al criterio del diseñador.

Comprobado en las pruebas de laboratorio, lógica difusa permite controlar plantas cuyos comportamientos sean desconocidos. Para las pruebas, solamente basto estudiar y definir el comportamiento de las plantas, las variables intervinientes de entradas/salidas y las relaciones entre ellas expresadas a través de reglas. Se evitó realizar estudios matemáticos excesivos para obtener un modelo del comportamiento de la planta, el cual es requerido para el diseño de controladores basados en la teoría clásica de control automático.

A pesar que no fue necesario un análisis matemático riguroso para el diseño y análisis de modelos difuso, se dependió mucho de un simulador. Muchos ingenieros reprochan que no existen métodos matemáticos para analizar controles

difusos y que se tenga que depender mucho de simuladores. En las pruebas de laboratorio se comprobó la imperiosa necesidad de utilizar el simulador FUDGE por la carencia de herramientas matemáticas para el análisis de controles difusos.

Para controlar las diferentes plantas del laboratorio fue importante entender el dinamismo de estas. Comprender el comportamiento de las plantas es crucial. Los primeros modelos difusos para cada planta no funcionaron correctamente porque no se comprendía el dinamismo de las diferentes plantas. Una vez entendido el comportamiento de cada planta, el control difuso simulado en FUDGE aplicado en el microcontrolador tenía un buen desempeño. La diferencia del comportamiento obtenido en el simulador contra la aplicación real se debió a parámetros como la inercia del sistema que FUDGE no es capaz de considerar (a través de las pruebas se observó que la inercia de las plantas sirve como criterio para la distribución de los singletons de la variable lingüística de salida). Las correcciones hechas a los modelos difusos simulados fueron ajustes de rangos, funciones de membresía y reglas hasta obtener un control efectivo. Con el prototipo pudo comprobarse que controladores difusos deben someterse a muchas pruebas hasta obtener un control efectivo.

Las pruebas realizadas al controlador prototipo permite concluir que la eficiencia del control esta íntimamente ligado al diseño del modelo difuso, el dispositivo procesador y los acopladores de señal. Para el modelo difuso es importante establecer rangos adecuados, la densidad de funciones de membresías y la estructura de las reglas. El rango para cada variable a establecerse depende de la precisión de los acopladores de señal y el número de bits del bus de datos del dispositivo controlador.

El acoplador de señales utilizado en las pruebas de laboratorio transforma los voltajes de variación de los sensores a voltajes adecuados para los convertidores analógicos-digitales del microcontrolador. La variación del parámetro físico sensible al microcontrolador depende del paso de cuantización, el cual esta

relacionado con el número de bits del bus de datos y el factor de transformación. Con la experiencia obtenida de las pruebas, se concluye que la sensibilidad de los convertidores del microcontrolador se degrada al aumentar el rango de variación de la variable física y al disminuir el número de bits del bus de datos.

Otro factor que influye en el desempeño del control difuso es la zona de traslape. Esta zona la determina el diseñador basándose en el comportamiento que se desea. El traslape puede considerarse como zonas de transiciones donde la función que predomina es la que tiene mayor pendiente, pero el efecto que tenga sobre la salida esta sujeto a la estructura de las reglas.

Un aspecto importante a tomarse en cuenta al combinar variables en reglas IF-THEN es la activación de funciones. Bajo la inferencia de Mandami, al combinar variables lingüísticas de entrada en una regla, la variable de entrada con mayor grado de pertenencia afecta la variable de salida relacionada a través del consecuente. Al generar reglas separadas para cada variable de entrada, la salida generada por el método de centro de gravedad será la combinación de las contribuciones individuales de cada regla.

En algunas pruebas de laboratorio se observó que el control difuso no era muy efectivo. Si la dinámica de la planta a controlar era demasiado rápida, por ejemplo la planta de posición, la reacción del control difuso tardaba en corregir por la tardanza en el procesamiento de datos en el microcontrolador. Concluyendo al respecto, la velocidad del dispositivo es un factor influyente en la capacidad de reacción del controlador difuso, limitando la aplicación en donde se respete el teorema de muestreo de Nyquist.

6.2. RECOMENDACIONES.

La tesis tuvo como principal objetivo brindar una introducción a la teoría de lógica difusa. Aunque el documento ofrece mucho más de lo que originalmente se planeó, futuros trabajos de investigación de mayor profundidad podrían llevarse a cabo. También podría mejorarse el controlador prototipo o implementar otro con otras técnicas de control difuso.

Para futuros trabajos sería conveniente realizar pruebas con plantas más complejas donde intervengan muchas variables. El controlador prototipo fue diseñado para manejar cuatro entradas y dos salidas. Como las plantas de laboratorio no involucraban más de dos variables, no se le pudo sacar el máximo provecho.

El diseño del controlador prototipo puede mejorarse en hardware y software. Respecto al hardware, podría calibrarse mejor los acopladores de señal y sustituir algunos elementos con otros de mayor precisión. En tanto al software, el núcleo de inferencia difusa puede mejorarse el proceso de defusificación al incluir otras formas de funciones de membresía además de singletons. También para la interfaz de usuario se sugiere cambiar el ambiente texto a un ambiente gráfico.

Para futuros prototipos de controladores difusos, se sugiere considerar un microcontrolador con mayores capacidades, tal como el MC68HC12 de Motorola que incluye instrucciones de lógica difusa.

Si se desea desarrollar una investigación similar a la de esta tesis, se recomienda que el investigador tenga conocimientos en electrónica digital y analógica, microprocesadores y lógica de programación en algún lenguaje de alto nivel.

Respecto al diseño de modelos difusos, se aconseja tomar en cuenta los siguientes puntos:

- Si se desea mayor precisión para la variable a controlar, es aconsejable utilizar conjuntos de forma triangular para las funciones de membresía.
- Si se desea que el control difuso tenga mayor inmunidad al ruido, entonces utilizar conjuntos de forma triangular para las funciones de membresía.
- Para controles difusos de reacción rápida, las pendientes de las funciones de membresía deben ser pronunciadas. Para menor pendiente, el sistema será menos sensible a los cambios de entrada.
- La proposición con menor valor en un antecedente compuesto predomina. Si se desea que cada proposición contribuya en la variable de salida, entonces debe formarse reglas con un antecedente de una sola proposición.

Si se desea implementar el modelo difuso para la planta de posición documentado en el capitulo 5, incluir una segunda variable que mida la velocidad y generar reglas relacionando en el antecedente esta variable con la variable de error. Este nuevo modelo podría reducir el sobreimpulso que se obtuvo en las pruebas realizadas con únicamente la variable de señal de error.

APENDICES

APENDICE A. LOGICA DIFUSA.

En este apéndice se explican los fundamentos y conceptos básicos que se requieren para comprender la lógica difusa, partiendo desde lo general a lo particular. A través de un ejemplo práctico, como lo es un sistema de riego casero, donde el sistema controlador implementado utiliza dos variables de entrada (temperatura del aire y humedad del suelo) y una salida que gobierna un contador de tiempo para la válvula de riego.

A.1. FUNDAMENTOS DE LOGICA DIFUSA.

Uno de los tópicos interesantes dentro de la investigación de control automático es el uso de técnicas difusas para la implementación rápida, fácil y transparente de estrategias de control no-lineal para procesos existentes, los cuales difícilmente pueden ser modelados por un acercamiento físico-matemático.

A finales del siglo XIX el matemático alemán George Cantor, inició el estudio y la aplicación de la Teoría de Conjuntos, punto de partida de la matemática moderna. A una colección de objetos se le llama conjunto, y los objetos individuales de dicha colección se dice que son elementos del conjunto o que pertenecen al conjunto. En los años 30 Lukasiewicz trato de explicar matemáticamente el modo de pensar de las personas, creando la Lógica Multi-Nivel. En 1965, Lofti Zadeh, profesor de Ingeniería Eléctrica y Ciencias de la Computación, de la Universidad de Berkeley, basándose en el trabajo de Lukasiewicz, introduce la teoría de lógica difusa. El principal concepto de esta teoría es el conjunto difuso, el cual es un aglomerado de elementos que poseen grados de pertenencia parcial o completa. Frecuentemente el término de lógica difusa es usado como sinónimo de teoría de conjunto difuso, un conjunto de

principios matemáticos para modelar información basado en grados de membresía.

El Dr. Lofty Zadeh reconoció que la certeza ó falsedad natural de la lógica Booleana no toma en cuenta muchas de las variaciones encontradas en el mundo real. Para tomar en cuenta las graduaciones infinitas entre verdadero y falso, Zadeh expandió la idea de un conjunto clásico a lo que él determinó como un conjunto difuso. A diferencia de la lógica Booleana, la lógica difusa es multivaluada, en vez de que un elemento sea 100% una cosa u otra, o que sea una proposición enteramente cierta o completamente falsa, la lógica difusa trata en grados de membresía y grados de verdad algo que puede ser parcialmente cierto y parcialmente falso a la misma vez. Bart Kosko, quien fue alumno de Lofty Zadeh, comprobó que la lógica Booleana es un caso particular de la lógica difusa.

A través del siguiente ejemplo se explica el concepto fundamental de los conjuntos difusos. Por ejemplo, ¿Es 80°F tibio ó caliente? Como puede verse en la Figura A.1, en términos de lógica difusa y del mundo real la respuesta podría ser "algo de ambos". Según la representación hecha con los conjuntos difusos, la temperatura de 80° es parcialmente tibio y parcialmente caliente a la misma vez. Por lo tanto, en lógica difusa un elemento puede pertenecer a más de un conjunto y tener grados de membresías entre cero y uno para cada conjunto. Como puede compararse en las Figuras A.1 y A.2, lógica difusa permite establecer traslapes graduales entre conjuntos, mientras que en lógica Booleana la transición entre conjuntos adyacentes es.

En lógica Booleana al usar conjuntos convencionales (ver Figura A.3) el 79.9°F se clasifica como "tibio" y 80.1°F como "caliente". En lógica Booleana un pequeño cambio en el valor cerca del limite del conjunto podría causar reacciones significativamente distintas. Un sistema difuso puede diseñarse los conjuntos difusos (ver Figura A.4) para que pequeñas oscilaciones en la entrada no causen grandes fluctuaciones en la salida del sistema.

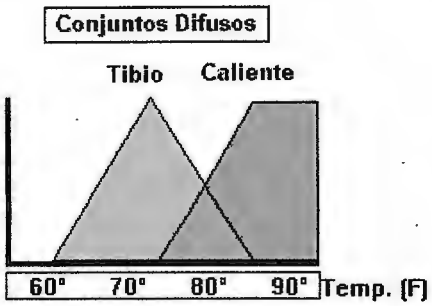


Figura A. 1 Lógica difusa.

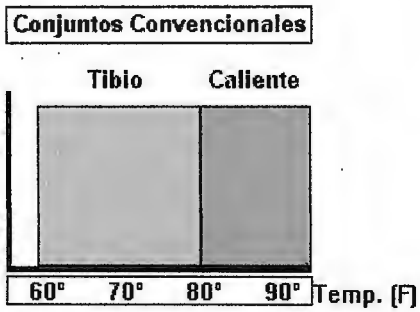


Figura A. 2 Lógica Booleana.

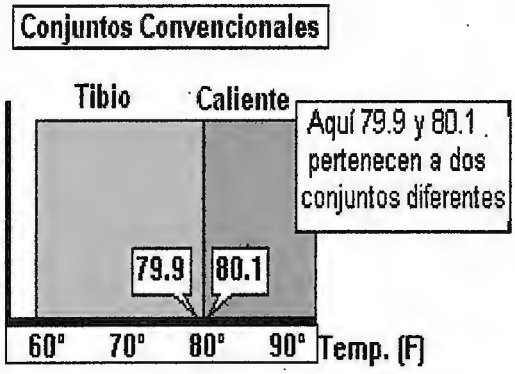


Figura A. 3 Lógica Booleana.

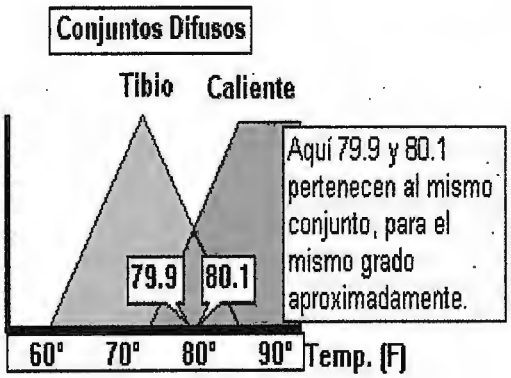


Figura A. 4 Lógica difusa.

A.2. CONJUNTOS DIFUSOS.

La teoría de conjunto difuso provee una base matemática y buenos fundamentos para diseñar controladores automáticos que tengan inteligencia similar al del complicado razonamiento humano. En la vida diaria el ser humano enfrenta muchas situaciones en donde debe tomarse una decisión a partir de datos inciertos. Un ejemplo es el estacionamiento de un vehículo, donde el conductor obtiene datos imprecisos de su entorno con los cuales toma decisiones para maniobrar hasta lograr estacionarse correctamente. De manera similar, lógica difusa brinda la anterior inteligencia humana a los controladores automáticos para tomar acciones precisas a partir de datos inciertos. Algunas veces esta teoría es llamada modelado de indecisiones, distinguiéndose los siguientes tipos:

- Indecisión estocástica: Un evento ocurre con una probabilidad dada, por ejemplo con los juegos de dados.
- Indecisión léxica o lingüística: La descripción imprecisa de un objeto, como por ejemplo gran apartamento, bajo precio, etc.
- Indecisión informacional: La indecisión causada por la pérdida de información o información incompleta.

En teoría de conjuntos clásico o convencional, un conjunto S se define por una función f_s, llamada "función característica de S". La función f_s genera dos únicos valores: cero indica que un elemento x esta fuera del conjunto y uno indica que el elemento pertenece dentro del conjunto. Por lo tanto, la definición matemática siguiente se lee: la función f_s se aplica al conjunto S formado por los números reales entre cero y uno, generando el valor de uno cuando el elemento x sea un número real dentro del conjunto S y cero cuando el elemento x no pertenezca a S.

$fs : S \longrightarrow [0,1]$, para un elemento x de S, $fs(x) = \begin{cases} 1, si..x \in S \\ 0, si..x \notin S \end{cases}$

En la teoría de conjuntos difusos, el conjunto S esta definido por una función μ_s, llamada "función de membresía de S". Esta función define el grado de pertenencia de un elemento del universo en el conjunto S. Los valores generados por la función μ_s están regidos por la forma del conjunto difuso. El mínimo valor que la función puede generar es cero e indica que el elemento no pertenece al conjunto; mientras que el máximo valor es uno e indica pertenencia completa al conjunto. Por lo tanto, la siguiente definición se lee: la función de membresía μ_s para el conjunto difuso S delimitado entre los números reales de cero y dos genera grados de membresía para los valores entre cero y uno utilizando la ecuación $\mu_s(x)=x$, y para los valores entre uno y dos se emplea la ecuación $\mu_s(x)=2-x$.

$\mu_s : S \longrightarrow [0,2]$, para un elemento x de S, $\mu_s(x) = \begin{cases} x, si..0 \le x \le 1 \\ 2-x, si..1 < x \le 2 \end{cases}$

Enseguida se provee un ejemplo de conjuntos Booleanos. Primero se delimita un conjunto universal X delimitado por los números reales entre cero y diez. Luego se define un subconjunto A formado por los números reales entre 5 y 8.

$$A = [5,8]$$

Ahora se verá la función característica del subconjunto A. Esta función asigna el valor de uno o cero para cada elemento del conjunto X, dependiendo si el elemento pertenece al subconjunto A. La Figura A.5. muestra el resultado de la función para cada elemento de X.

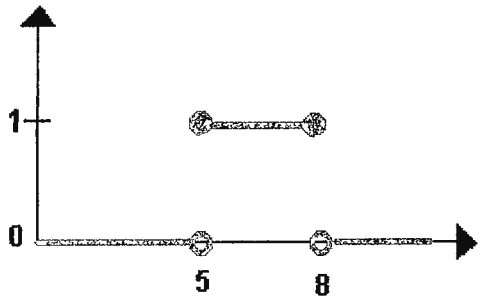


Figura A. 5 Gráfica de conjunto Booleano.

Muchas aplicaciones de control se basan en teoría de conjuntos Booleanos. Sin embargo existen otras aplicaciones reales donde existe mucha flexibilidad e incertidumbre y resulta difícil de representar a través conjuntos Booleanos. Con el siguiente ejemplo del conjunto de Gente Joven se ilustrará como los conjuntos difusos permiten manejar flexibilidad e incertidumbre.

Formalmente se puede denotar así: B = {Conjunto de personas Jóvenes}

Se tomará las edades entre cero a veinte años. Entonces se tiene un intervalo "crisp" así:

$$B = [0,20]$$

Al preguntarse, ¿una persona con 20 años es joven? La respuesta sería afirmativa. El conflicto surge al pasar un día, ¿esa misma persona sigue siendo joven? Aunque la respuesta racional es sí, según el conjunto B, esa persona ya no pertenece al grupo de personas jóvenes. Este súbito cambio es un problema estructural que persiste aunque se modifiquen los limites del conjunto.

Para resolver el anterior conflicto se ha modificado la función del conjunto B tal como se muestra en la Figura A.6. La pendiente entre veinte y treinta años permite representar el cambio paulatino entre juventud y madurez. Según este

nuevo conjunto B, una persona de 25 años se considera 50% joven. A través de la pendiente se trata de representar la incertidumbre entre juventud y madurez.

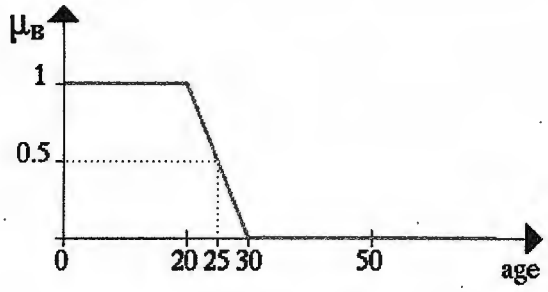


Figura A. 6 Conjunto de "personas jóvenes".

Con el ejemplo anterior, se muestra la flexibilidad que brinda lógica difusa para representar las degradaciones infinitas que existen entre conjuntos. Lógica difusa, a través de las pendientes de las funciones de membresía, permite cuantificar la incertidumbre (ver Figura A.7).

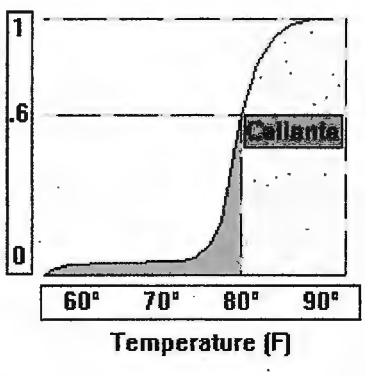


Figura A. 7 Grado de membresía.

A.3. VARIABLE LINGÜISTICA.

Parece contradictorio, pero lógica difusa provee un método riguroso y preciso para alcanzar decisiones precisas. Para entender un sistema de lógica difusa es necesario familiarizarse con los siguientes conceptos (ver Figura A.8).

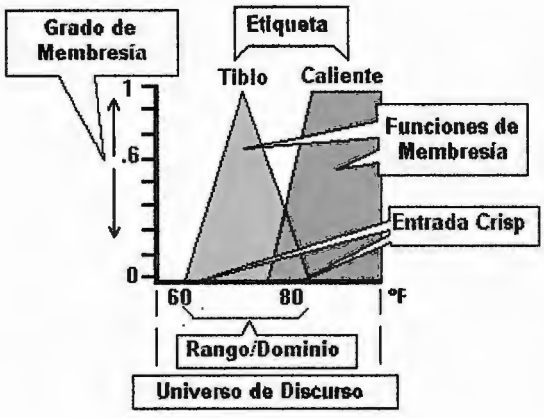


Figura A. 8 Conceptos de lógica difusa.

- Grado de Membresía: Es el grado en el cual un valor "crisp" es compatible con una función de membresía (de 0 a 1). También se refiere como grado de membresía, el valor de certeza ó de entrada difusa.
- Etiqueta: Es la que describe el nombre de la función de membresía.
- Función de Membresía: Es la que define un conjunto difuso, que contiene a las entradas "crisp" en el dominio para un grado de membresía.
- Entrada "Crisp": Diferentes entradas bien definidas, como por ejemplo 90° grados Fahrenheit.
- Alcance/Dominio: Es el ancho de la función de membresía, el concepto de rango ó alcance son usualmente los valores sobre la cual una función de membresía es descrita. Para el caso de la Figura A.8, el dominio del conjunto difuso (función de membresía) es de 60° a 80° y el rango ó alcance es de 20°.
- Universo de Discurso: Es el rango de todos los valores posibles y aplicables a un sistema variable.

Para representar las entradas y salidas de un sistema difuso se utilizan las variables lingüísticas. El universo de discurso delimita la variable lingüística. Dentro del universo de discurso puede formarse funciones de membresía (conjuntos difusos) que permiten identificar "comportamientos" de la variable lingüística. Por ejemplo, una señal de voltaje que varía entre –5VDC y 5VDC puede representarse en un sistema difuso de la siguiente forma: el rango de –5VDC a 5VDC forma el universo de discurso y dentro de este se declaran cinco funciones de membresía (nb=negativo grande, n=negativo, 0=aproximadamente cero, p=positivo y pb=positivo grande). En la Figura A.9 se observa como se han distribuido las funciones de membresía dentro del universo de discurso.

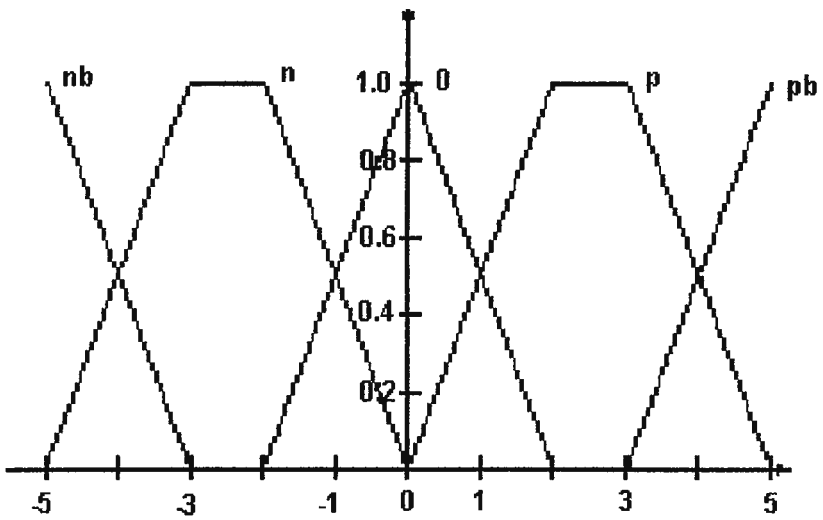


Figura A. 9 Funciones de membresía.

En los sistemas difusos es importante considerar la forma de representar las variables lingüísticas en la memoria de un dispositivo procesador, sea un microprocesador o microcontrolador. Para alcanzar una ejecución de alta velocidad, en general son usadas las funciones de pertenencia lineales ya que solamente es necesario almacenar los vértices de las líneas y a través de interpolación se calculan los valores intermedios. Otra alternativa es almacenar en memoria una matriz de valores que contiene los grados de pertenencia claves, donde las filas representan las funciones de membresías y las columnas

corresponden a mediciones esenciales de la entrada/salida "crisps". La Figura A.10 es la representación matricial de las funciones de membresías mostradas en la Figura A.9.

$\rightarrow u$	-5	-4	-3	-2	-1	0	1	2	3	4	5
$\downarrow term$	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
$nb \rightarrow$	1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$n \rightarrow$	0.0	0.5	1.0	1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0
$0 \rightarrow$	0.0	0.0	0.0	0.0	0.5	1.0	0.5	0.0	0.0	0.0	0.0
$p \rightarrow$	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	1.0	0.5	0.0
$pb \rightarrow$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0

Figura A. 10 Matriz de relación.

A.4. LOGICA DIFUSA Y OPERADORES DIFUSOS.

Los tres operadores primarios en la teoría de conjuntos difusos son la intersección (operador de lógica básica AND), la unión (operador de lógica básica OR), y la negación (operador de lógica básica NOT). Estos operadores primarios, excepto el operador NOT, tienen las mismas funciones que los operadores correspondientes de teoría de conjuntos clásico.

Tomando como ejemplo la búsqueda de un apartamento nuevo, podría definirse los siguientes criterios: el apartamento es barato y grande, luego iré y lo arrendaré. Se debe tener una idea que es lo que expresa precio y tamaño como variables lingüísticas. Primero se tendrá un conocimiento sobre lo que significa barato o caro. Segundo, se aplicará la operación de unión (AND), lo cual significa que un apartamento grande definitivamente no se escogerá si el precio es sólo un

poco alto. El apartamento puede ser escogido si es económico en general. Para realizar estas reflexiones se llega a un punto más de interés concerniente al conocimiento impreciso. Los operadores lingüísticos como AND y OR son necesarios para realizar estos acercamientos.

A.4.1. INTERSECCION.

La operación lógica AND corresponde a la intersección de dos conjuntos (ver Figura A.11). La intersección de dos conjuntos se determina al tomar punto a punto el mínimo valor de las dos funciones de membresía así: $\mu_{A\cap B}(x) = \min [\mu_A(x), \mu_B(x)]$, sobre todos los valores de X. Se le llama intersección de dos conjuntos A y B, si el nuevo conjunto contiene exactamente aquellos elementos que están contenidos tanto en A como en B (ver Figura A.12).

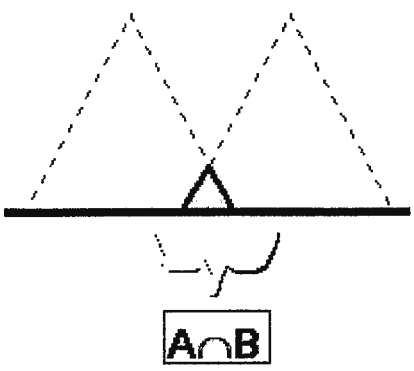


Figura A. 11 Unión A y B.

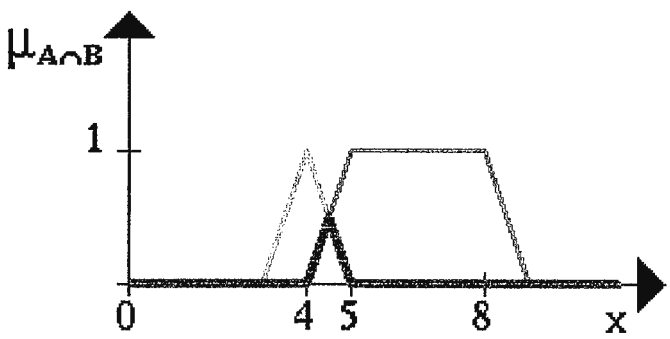


Figura A. 12 Intersección de A y B, función AND.

A.4.2. UNION.

La operación **OR** en lógica difusa, corresponde a la unión de dos conjuntos difusos (ver Figura A.13). La unión de dos conjuntos difusos A y B con grados de funciones de membresía $\mu_A(x)$ y $\mu_B(x)$ respectivamente es: $\mu_{A\cup B}(x) = \max [\mu_A(x), \mu_B(x)]$, sobre todos los valores de X. La unión de dos conjuntos difusos se determina tomando los máximos valores punto a punto de las dos funciones de membresía. Se le llama unión de dos conjuntos A y B, si el nuevo conjunto contiene todos los elementos que están contenidos en A y en B (ver Figura A.14).

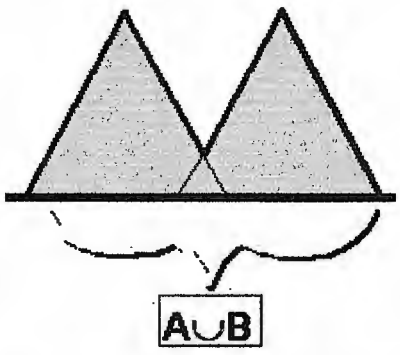


Figura A. 13 Unión de conjunto A y B.

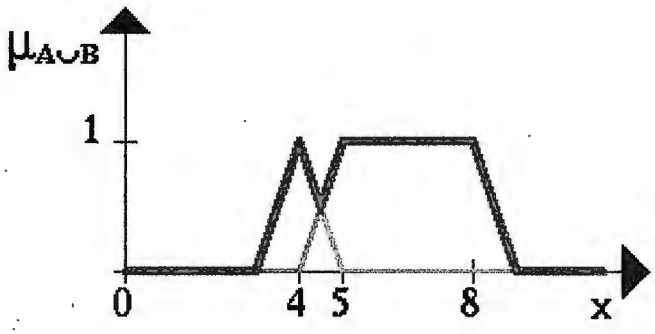


Figura A. 14 Operación OR.

A.4.3. NEGACION.

La operación lógica NOT corresponde al complemento de un conjunto (ver Figura A.15). El complemento de un conjunto difuso A, denotado como ¬A ó A, tiene una aplicación distinta a su similar en lógica Booleana. En lógica difusa, el operador NOT indica cuanto hace falta a un elemento de un conjunto difuso para tener una

pertenencia del 100%. Por lo tanto, el operador NOT puede definirse como: $\mu_A(x)=1-\mu_A(x)$ (ver Figura A.16).

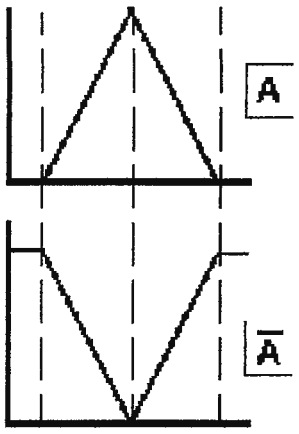


Figura A. 15 Negación del conjunto A.

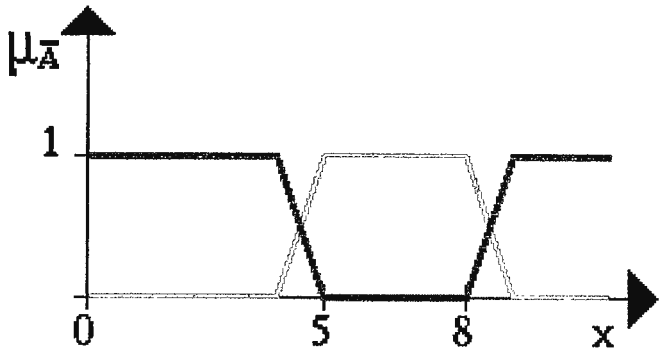


Figura A. 16 Operación NOT A.

A.5. FUSIFICACION Y FUNCIONES DE MEMBRESIA.

El uso de técnicas difusas para el control automático (como ya se había mencionado anteriormente), trata de imitar el comportamiento consciente de un operador humano controlando procesos no lineales complejos o plantas de producción las cuales difícilmente pueden ser modeladas por reflexiones físico-matemáticas. Estas circunstancias deshabilitan el diseño sistemático de un controlador basado en modelos convencionales. Ejemplos de esta clase de procesos son:

- Procesos de producción biotecnológicos.
- Procesos químicos, procesamiento de imágenes.
- Plantas de tratamiento de aguas residuales, entre otros.

De acuerdo con Von Altrock (1991) un control difuso de lazo cerrado (estándar) es como el que se muestra en la Figura A.17. En lugar de un controlador lineal o no lineal, se usa el controlador difuso. Para obtener una solución "crisp" al usar lógica difusa, el dispositivo procesador debe llevar acabo los siguientes tres pasos (ver Figura A.18):

- Fusificación.
- Reglas de Evaluación ó Inferencia.
- Defusificación.

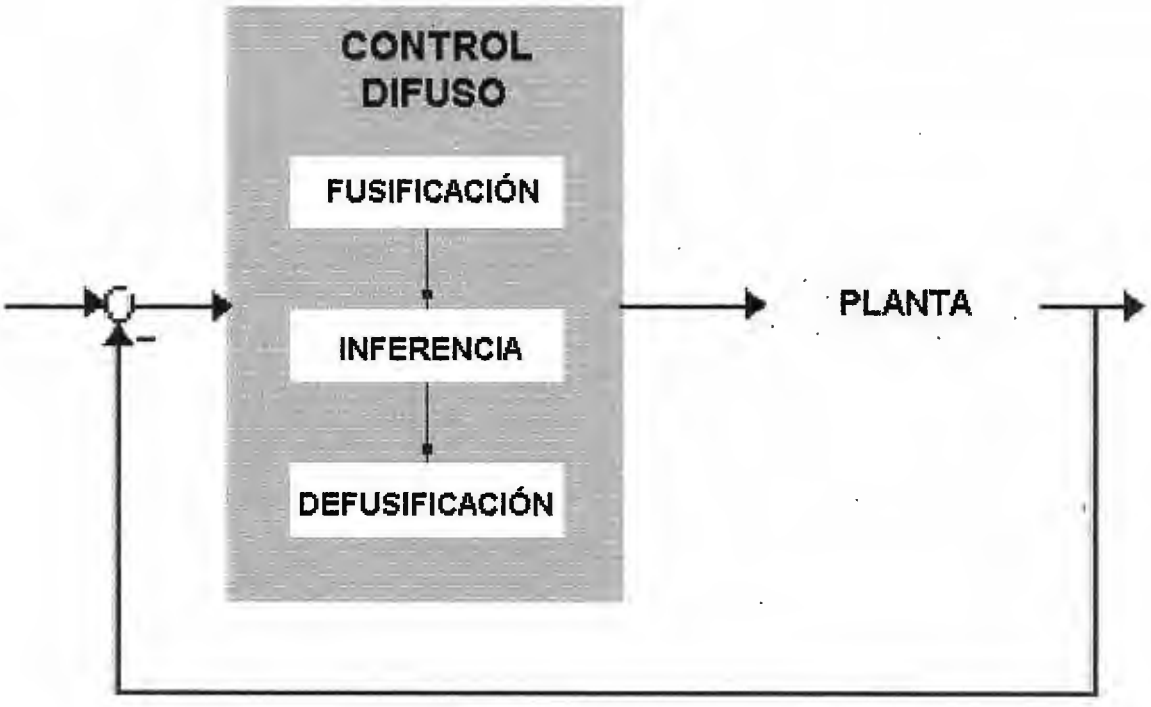


Figura A. 17 Control difuso.

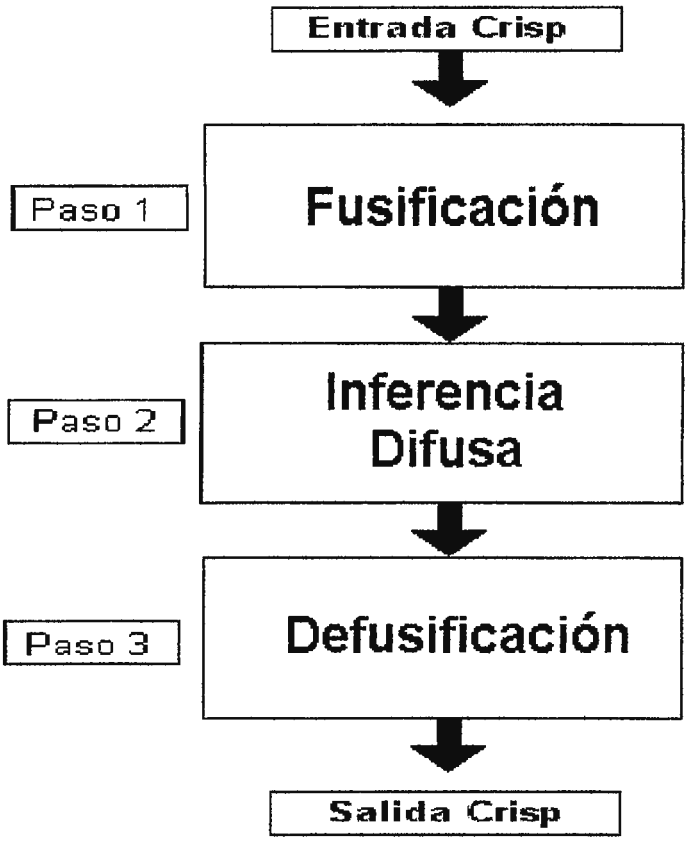


Figura A. 18 Pasos de una solución "crisp".

El primer paso en un sistema de control difuso es la transformación de dominio llamada Fusificación (ver Figura A.19). Las entradas "crisp" son transformadas a entradas difusas. Por ejemplo, una entrada "crisp" de 78° debe ser transformada al término difuso "tibio"; 90 Mph debe ser "rápido" y así sucesivamente. Para transformar las entradas "crisp" a entradas difusas, primero deben determinarse funciones de membresía para cada entrada. Una vez las funciones de membresía son asignadas, la Fusificación toma un valor de entrada en tiempo real, como el de una temperatura, y lo compara con la información de las funciones de membresías almacenadas para producir grados de pertenencia.

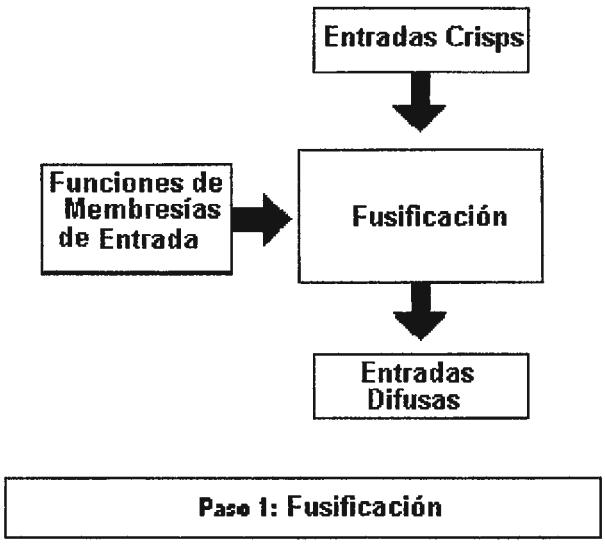


Figura A. 19 Fusificación.

Para ilustrar el proceso de Fusificación, se estudia un sistema de riego casero (ver Figura A.20). El controlador difuso en este sistema usa dos entradas, la temperatura del aire y la humedad del suelo, para calcular la duración del riego. El primer paso en la Fusificación es asignar las etiquetas lingüísticas en el universo de discurso para cada una de las entradas "crisp". Para la temperatura, se han asignado las etiquetas mostradas en la Figura A.21. Para la humedad del aire se han establecido las etiquetas mostradas en la Figura A.22.

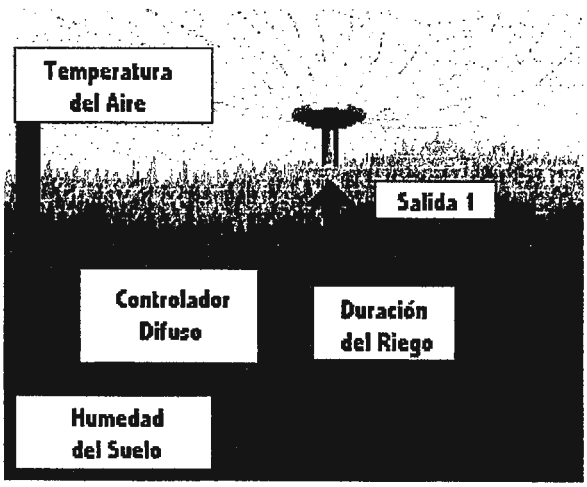


Figura A. 20 Sistema de riego casero.

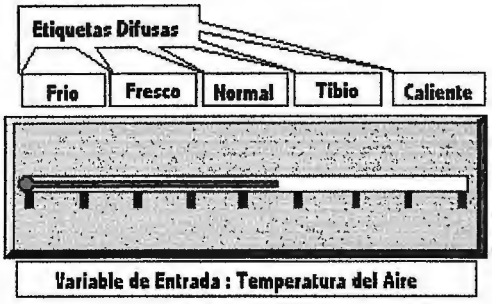


Figura A. 21 Etiquetas de temperatura.

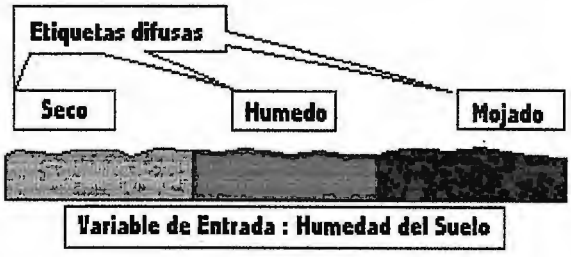


Figura A. 22 Etiquetas de Humedad.

Cada entrada "crisp" en el sistema difuso puede tener múltiples etiquetas asignadas a ella. En general un número grande de etiquetas para describir una variable de entrada permite diseñar un sistema difuso de alta resolución con una respuesta suave de control difuso. Sin embargo un número grande de etiquetas requiere mayor tiempo de procesamiento y además puede llevar al sistema difuso a la inestabilidad. Por lo anterior, el número común de etiquetas por variable en sistema difuso es entre 3 y 9. Los números de etiquetas por lo general son impares (3,5,7,9), pero no siempre. Generalmente, en un sistema difuso se busca siempre una distribución simétrica de funciones de membresías alrededor del punto central del universo de discurso. Para el caso de la variable temperatura esta podría tener una etiqueta "FRIO" a la que le correspondería otra etiqueta contraria "CALIENTE", ambas manteniendo una simetría de acuerdo a la etiqueta central "NORMAL" (ver Figura A.23).

Las funciones de membresía son establecidas para brindar un significado numérico a cada etiqueta. Cada función de membresía identifica el rango de valores de entrada que corresponden a una etiqueta (ver Figura A.24).

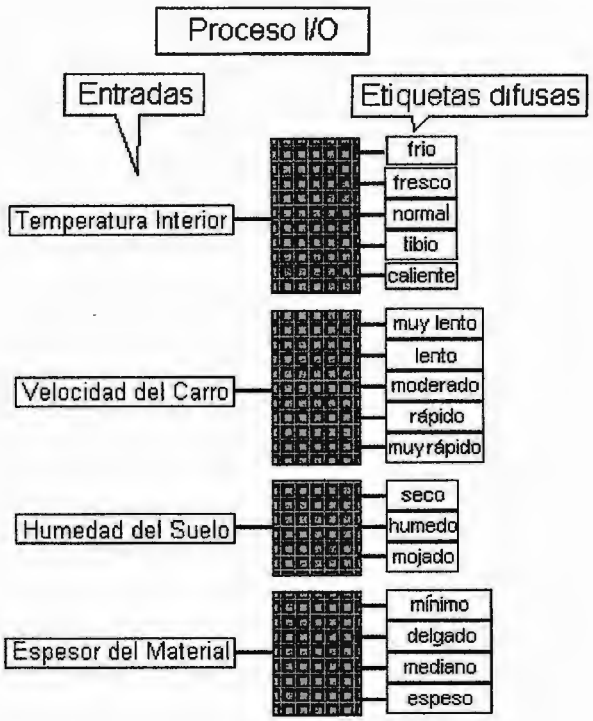


Figura A. 23 Etiquetas difusas.

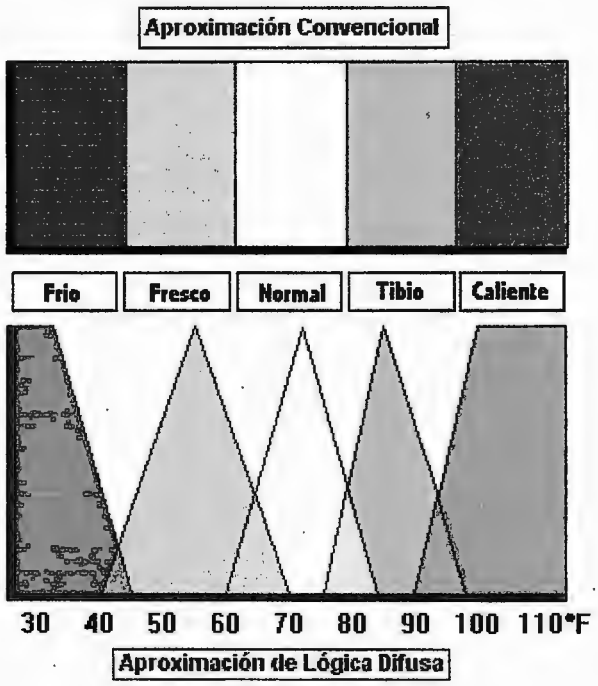


Figura A. 24 Aproximaciones convencionales y difusas.

Distinto a la lógica Booleana, las funciones de membresía de una etiqueta no definen una frontera donde la etiqueta aplica completamente en un lado del límite y nada en el otro lado del límite. En lugar de ello hay un cambio gradual del grado de pertenencia. En la Figura A.25 se muestra las funciones de membresías para una variable de salida de sistema de riego.

La forma de una función de membresía afecta el proceso difuso en distintas maneras. Por ejemplo, la forma de una función directamente afecta el tiempo y requerimientos de espacio en memoria de un microcontrolador que realiza las tareas de Fusificación y Defusificación.

Las funciones de membresías pueden tomar varias formas diferentes. Las funciones de membresías con formas trapezoidales y triangulares son las más comúnmente usadas (ver Figura A.26). Sin embargo otras formas pueden ser más representativas para las variables de entradas y salidas, pero estas requieran ecuaciones más complicadas o largas tablas para representarlas con precisión. Una forma sencilla fácilmente posible de representar en un dispositivo procesador es el Singleton. Esta forma permite crear algoritmos de Defusificación simples. Por lo tanto, frecuentemente los Singletons se utilizan para describir salidas difusas.

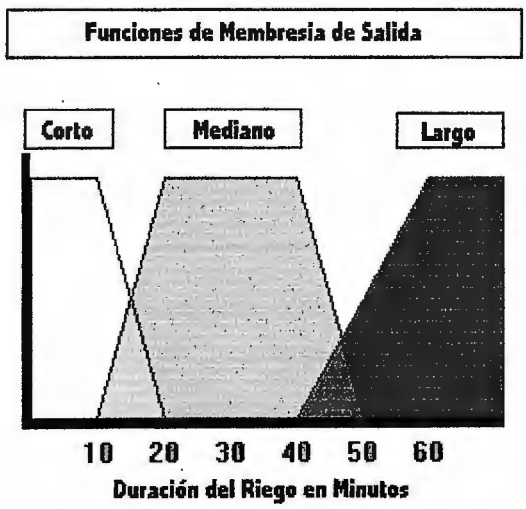


Figura A. 25 Funciones de salida.

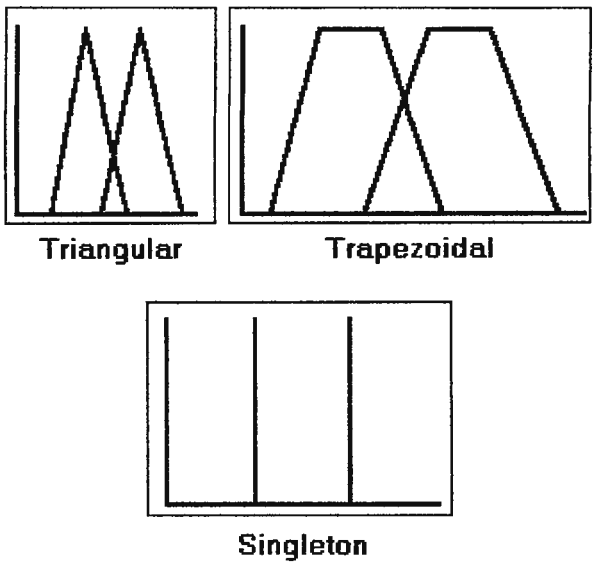


Figura A. 26 Formas de las funciones de membresía.

Dependiendo de las formas de las funciones de membresía, varios métodos son usados para representar las funciones en un microcontrolador. Una representación punto-pendiente permite que funciones trapezoidales, triangulares y Singletons sean representadas con una mínima cantidad de espacio de memoria. Una función de membresía trapezoidal puede ser representada con cuatro bytes. Para el programa difuso utilizado en este trabajo de graduación se empleó la representación que se muestra en la Figura A.27.

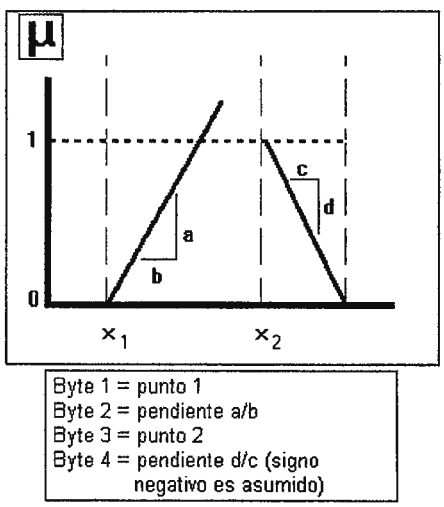


Figura A. 27 Representación de funciones de membresía.

El núcleo difuso puede restringir ecuaciones de línea para valores de grados de pertenencia entre 0 y 1. Una tabla es una representación común para una función de forma arbitraria (ver Figura A.28). Aun cuando esta tabla es probablemente la representación más rápida de una función, esta consume mayor cantidad de memoria. Además, funciones de forma arbitraria pueden tomar más tiempo de procesamiento en la Defusificación si un microcontrolador usa el método del centro de gravedad. Cuando Singletons son usados para describir funciones de membresía de salida, la Defusificación por el método del centro de gravedad se reduce a un simple cálculo de peso promedio. Estudios han demostrado que utilizar salidas Singletons permite diseñar controladores difusos con un funcionamiento razonable y buena velocidad de respuesta.

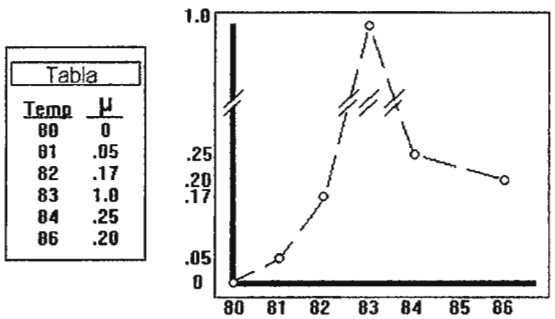


Figura A. 28 Tabla de representación de una función de membresía arbitraria.

Una función de membresía establece el grado de pertenencia para cada posible valor de entrada/salida "crisp" (ver Figura A.29). La Figura A.30 es una representación gráfica de funciones de membresías para una variable lingüística, donde el eje-y indica los grados de pertenencias y el eje-x representa los valores de entrada "crisp". Para encontrar el grado de pertenencia de una entrada "crisp" primero debe trazarse una recta perpendicular al eje-x sobre el valor de la entrada "crisp"; luego, sobre el punto donde la recta perpendicular corte la función de membresía trazar una recta paralela; por último, el punto donde la recta paralela corte al eje-y corresponde al grado de membresía del valor de entrada crisp para la función de membresía con que se ha asociado. Como se observa en la Figura A.30, los valores de entrada "crisp" pueden pertenecer a más de un conjunto difuso

(función de membresía). Por ejemplo, la temperatura de 92° se encuentra dentro las funciones de membresía "TIBIO" y "CALIENTE". Al final, el proceso de Fusificación genera un grado de pertenencia de 0.45 para la función de membresía "TIBIO" y 0.45 para "CALIENTE". Describir las entradas "crisp" en términos difusos permite al sistema difuso responder fácilmente a cambios graduales en la temperatura de entrada.

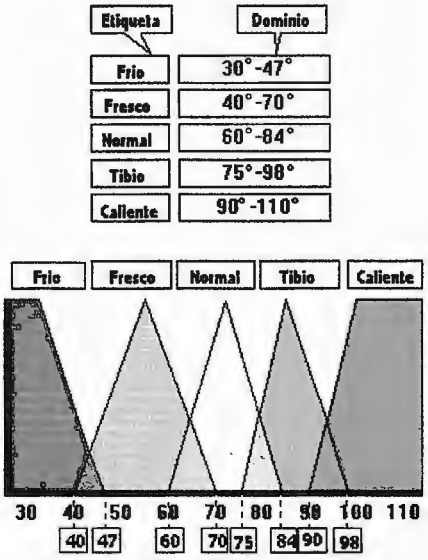


Figura A. 29 Funciones de membresía temperatura de aire.

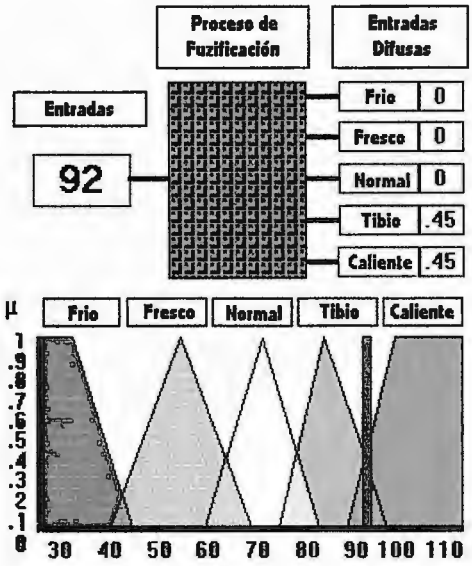


Figura A. 30 Grado de pertenencia para 92°F.

A.6. INFERENCIA DIFUSA.

En la evaluación de reglas ó inferencia, el procesador difuso utiliza reglas lingüísticas para determinar que acción de control ocurrirá en respuesta a un conjunto de valores de entrada dados. La evaluación de las reglas, también llamada inferencia difusa, aplica las reglas a las entradas difusas que se obtuvieron en el proceso de Fusificación (ver Figuras A.31 y A.32).

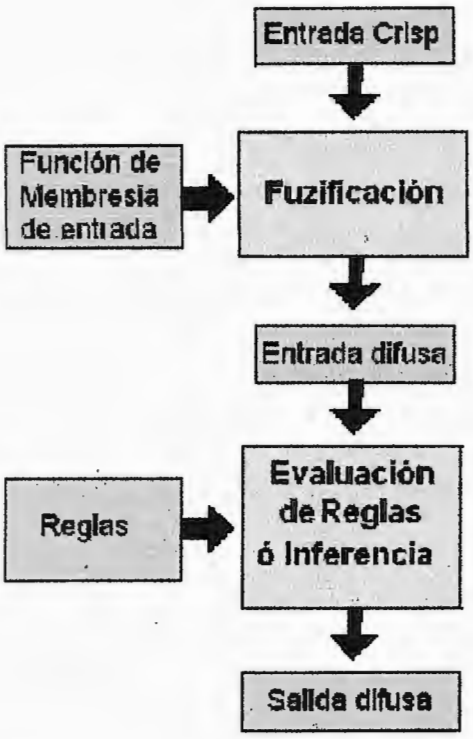


Figura A. 31 Proceso de lógica difusa.

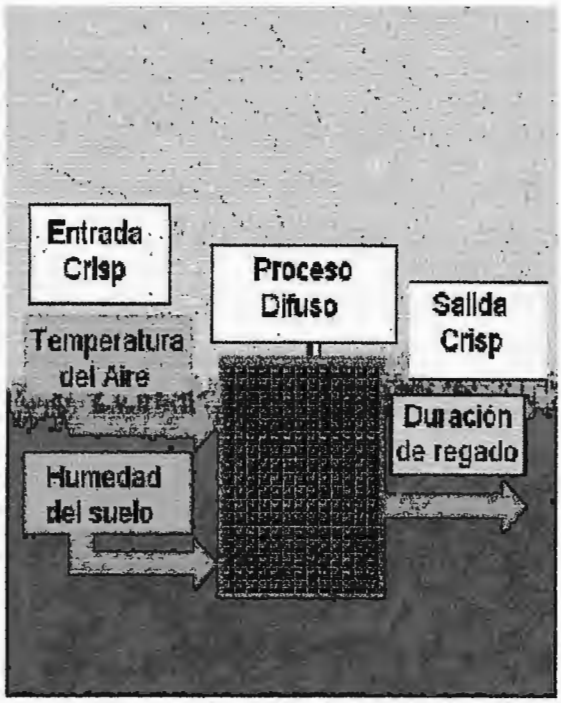


Figura A. 32 Sistema de riego casero.

El elemento fundamental de una regla difusa es la proposición, la cual en su forma más simple se le llama proposición difusa atómica. Esta proposición fundamental la compone la variable lingüística interconectada a una función de membresía a través de conectores lingüísticos como: IS (ES) y IS NOT (NO ES). En la Tabla A.1 se muestran los conectores lingüísticos comúnmente utilizados:

Teoría Clásica	Símbolo	Teoría Difusa	Símbolo
Igual A	=	Es	•
No igual a	≠	No Es	¬
Mayor que	>		
Menor que	<		

Tabla A. 1 Conectores lógicos.

Un ejemplo sencillo de una proposición atómica es la siguiente: "Error es pequeño". Aquí la variable lingüística es Error y la función de membresía a que se relaciona es pequeño. Varias proposiciones atómicas pueden combinarse a través de los operadores AND y OR para construir expresiones más complejas. Cuando dos proposiciones forman un conjunto en donde la función de membresía de una se convierte en la variable lingüística de la otra proposición, entonces se produce el principio del vínculo. El resultado de este principio, como se aprecia en el siguiente ejemplo, es una función de membresía que es subconjunto de la otra.

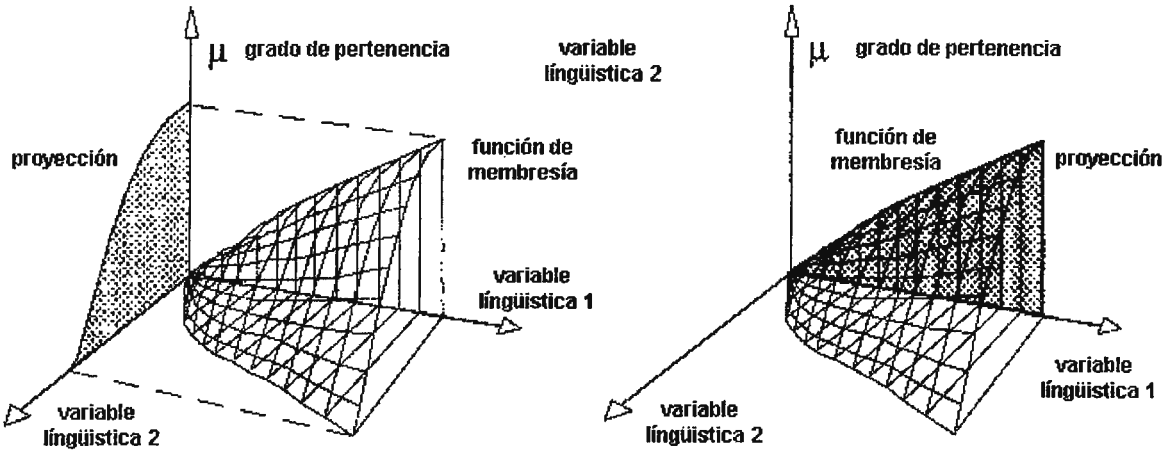
- El clima es bonito $X \bullet P$.
- Bonito es templado $P \subset A$.
- Entonces el clima es templado $X \bullet A$

Este principio es útil para minimizar expresiones y evitar redundancias, eliminando tiempo en procesamiento de un control difuso que puede conllevar a retardos en la reacción de sistemas de control. También se presentan casos en que se aplica el principio conjuntivo. Bajo este principio, dos variables lingüísticas distintas de dos proposiciones se unen para generar una tercera variable que se relaciona a una nueva función de membresía formada a través del producto cartesiano de las dos funciones de membresía aplicando el operador de máximos. La nueva función de membresía produce una matriz construida a partir de los vectores de las funciones originales.

- Temperatura del líquido A es templado $X \bullet W$.
- Temperatura del líquido B es fría $Y \bullet C$.
- La temperatura del líquido A y B esta entre frío y templado $(X,Y) \bullet W \times C$.
- El producto cartesiano de $W \times C$ es $\mu_{wxc} = \mu_w(w) \wedge \mu_c(c)$.

Otro término útil cuando se tiene dos variables intervinientes es la proyección (ver Figura A.33). Con dos variables lingüísticas se genera una superficie que

representa una nueva función de membresía que ocupa el plano tridimensional. La proyección consiste en hacer un corte a la superficie paralelo a cualquiera de los ejes y la sombra que se refleja en el plano bidimensional es la proyección.



$$\mu P(V2) = \bigvee_{V1xV2} [\mu_{f}(V1,V2)]$$

Figura A. 33 Proyecciones de funciones.

La anterior ecuación indica que la proyección P, que no es más que una función de membresía para la variable lingüística 2, es el conjunto de valores máximos del producto cartesiano de las dos variables lingüísticas que forman la función de membresía tridimensional. En control difuso esto es útil cuando se desea obtener las funciones de membresías a partir de la representación gráfica de las variables "crisps".

El principio de Extensión no es más que la existencia de una relación entre resultados de las reglas y limitantes aplicadas a variables lingüísticas. En otras palabras, las variables lingüísticas se comportarán en función de los resultados de las reglas. Los factores de peso pueden diseñarse en función de los resultados de la evaluación de reglas e utilizarse como modificadores de variables lingüísticas, lográndose con esto un control adaptivo.

Tomando en consideración los anteriores principios, el diseñador puede "educar" al sistema difuso como debe de "comportarse". Proveer la forma de "pensar" al sistema difuso se conoce como inferencia difusa. A continuación se describen algunas de las inferencias difusas comunes.

La inferencia composicional implica establecer la relación entre dos conjuntos difusos a partir de la relación existente entre otros conjuntos difusos que ambos tengan en común.

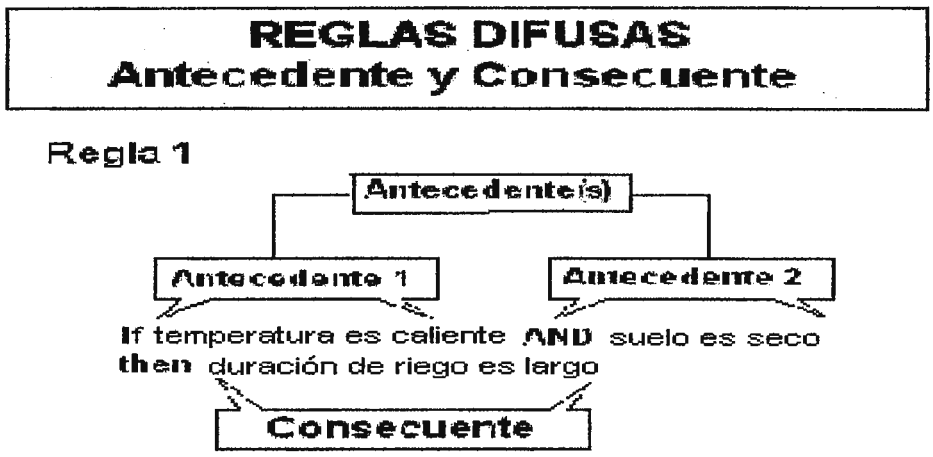
- Temperatura es peligrosa T•D.
- La relación presión-temperatura es una operación insegura (T,D)•U.
- Presión esta compuesta de características peligrosas e inseguras P•(U°D).

El método de inferencia Modus Ponens se basa en la configuración de reglas If-Then. Estas reglas establecen la relación entre las entradas y salidas del sistema. El Modus Tollens es el inverso del Modus Ponens, ya que a través de la salida se determina lo que sucede en la entrada. Cabe mencionar que para este último tipo de inferencia, pueden existir muchas combinaciones de entradas que generen el mismo resultado. Para aplicaciones de control difuso, la inferencia Modus Ponens es la que más se aplica. A través de reglas formuladas con sentencias If-Then, el procesamiento de evaluación de las reglas se facilita.

En la inferencia Modus Ponens, las reglas difusas son usualmente declaraciones **If-Then (*Sí-Entonces*)** que describen la acción a tomar en respuesta a las entradas difusas. Las siguientes sentencias son ejemplos de reglas difusas:

- **If** el suelo está húmedo y la temperatura caliente **Then** la duración del riego es corta.
- **If** el carro se mueve rápido y el pavimento esta seco **Then** aplique fuertemente los frenos.
- **If** la ducha está muy caliente **Then** incremente suavemente el flujo de agua fría.

Si bien las reglas se ven que están en lenguaje natural, estas son regidas por una sintaxis estricta (ver Figura A.34).



If Antercedente1 **AND** Antecedente2... **Then** Consecuente1

Figura A. 34 Sintaxis de reglas.

Una regla difusa esta compuesta por un antecedente y un consecuente. La parte **If** es el antecedente y la parte **Then** es el consecuente. Cada parte de la regla difusa puede esta compuesta por varias proposiciones atómicas unidas a través del operador lógico **AND** u **OR.** Las reglas siguen el sentido común del comportamiento del sistema y son escritas en términos de funciones de membresía y etiquetas lingüísticas. Para un sistema de dos entradas y una salida las reglas pueden ser leídas y representadas en una matriz como se muestra en la Figura A.35.

Las reglas difusas permiten hacer una aproximación intuitiva del comportamiento de un sistema. Con estas reglas se evita un análisis matemático exhaustivo para encontrar una ecuación que describa el comportamiento fundamental del sistema. A continuación se muestran algunas reglas construidas a partir de la matriz de la Figura A.35:

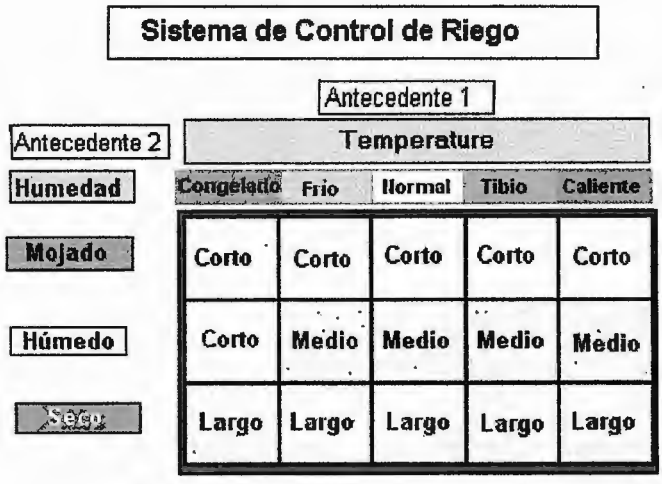


Figura A. 35 Matriz de etiquetas.

- **If** la temperatura es caliente **AND** el suelo está seco **Then** la duración del riego es largo.
- **If** la temperatura es fría **AND** el suelo está mojado **Then** la duración del riego es corto.

El primer paso en la evaluación de reglas es ponderar el grado de relevancia del antecedente. A través de la Fusificación se obtiene los grados de pertenencias para las entradas crisps. Dependiendo de la asociación entre la variable lingüística y las funciones de membresías establecidas en cada proposición del antecedente, se obtiene el grado de pertenencia para cada proposición. En el sistema de riego el proceso de Fusificación para la entrada de 92°F de la temperatura del aire produce los grados de pertenencia de 0.2 del conjunto difuso "TIBIO" y 0.46 del conjunto difuso "CALIENTE"; para la entrada de 11% de humedad del suelo los grados de pertenencia son de 0.25 del conjunto difuso "SECO" y 0.75 del conjunto difuso "HUMEDO"(ver Figura A.36).

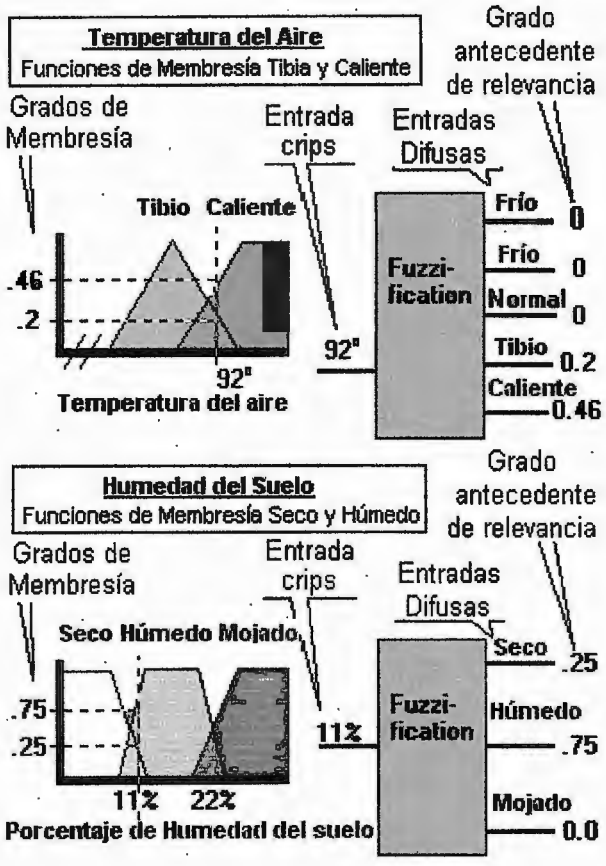


Figura A. 36 Relevancia de las reglas.

Una vez completado el proceso de Fusificación, se procede a sustituir cada proposición de los antecedentes con los grados de pertenencia. Por ejemplo, la proposición "Temperatura es Tibio" se sustituye con el valor de 0.2. Después de hacer las sustituciones en el antecedente, se procede a encontrar el grado de verdad o fuerza de la regla. Los operadores difusos que se ocupan para formar proposiciones compuestas influyen en la fuerza de la regla. Con el operador difuso AND, el mínimo valor de grado de pertenencia del antecedente es seleccionado como la fuerza de la regla. Con el operador difuso OR el máximo valor de grado de pertenencia del antecedente es seleccionado como la fuerza de la regla. Sin embargo recomiendan el uso del operador AND siempre y cuando sea posible. En la Figura A.37 se ilustra como lógicamente se convierten reglas con operadores OR en múltiples reglas con operador

AND. En la Tabla A.2 se brindan ejemplos para el sistema de riego cuando la temperatura del aire es de 92ºF y la humedad del suelo es de 11%.

Temperatura del Aire = 92°F	**Humedad del suelo = 11%**
REGLA	**FUERZA DE LA REGLA**
If temperatura es caliente (0.46) **AND** suelo es seco (0.25), **Then** el tiempo de riego es medio.	0.25
If temperatura es tibia (0.2) **AND** suelo es húmedo (0.25), **Then** el tiempo de riego es medio.	0.2
If temperatura es tibia (0.2) **AND** suelo es seco(0.25), **Then** el tiempo de riego es larga.	0.2
If temperatura es caliente (0.46) **AND** suelo es húmedo (0.75), **Then** el tiempo de riego es medio.	0.46

Tabla A.2 Fuerza de reglas.

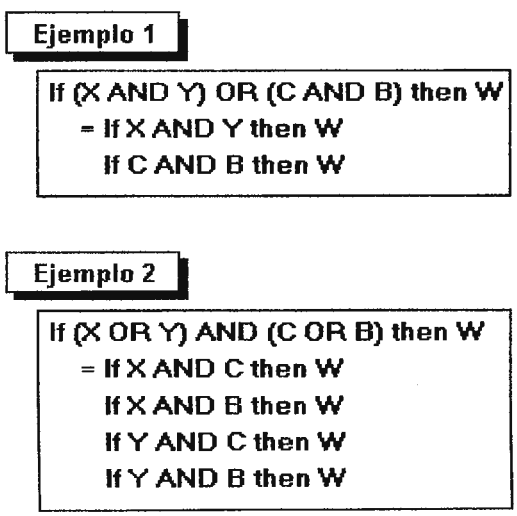


Figura A. 37 Reglas múltiples.

El siguiente paso es determinar el grado de verdad (fuerza de la regla) que se aplicará a la función de membresía que ha sido relacionada a través del consecuente de la regla a una variable lingüística de salida difusa. En este paso se transforma el grado de verdad en grado de pertenencia para la función de membresía asociada en el consecuente. Por ejemplo, si el antecedente produce un grado de verdad de 0.3 y el consecuente es "Tiempo de riego es CORTO", entonces para la variable lingüística "Tiempo de riego" se aplica la función de membresía "CORTO" con un grado de pertenencia de 0.3. Para el caso donde varias reglas tienen el mismo consecuente, el valor que se aplica a la función de membresía de la variable lingüística difusa es la fuerza de verdad máxima de todas las reglas.

Para el sistema de riego con las entradas de temperatura de 92ºF y 11% de humedad del suelo, puede observarse como las reglas 2 y 4 dictan la acción "tiempo de riego medio" con diferentes grados de verdad (fuerza de regla); y las reglas 1 y 3 indican "riego largo" también con diferentes grados de verdad. Como se ve en la Figura A.38, el resultado final para el consecuente con "Tiempo de riego es LARGO" es de 0.25 y para "Tiempo de riego es MEDIO" el valor es 0.46. En términos simples, si dos ó más reglas tratan de afectar la misma salida, entonces dominará la que tiene mayor grado de verdad.

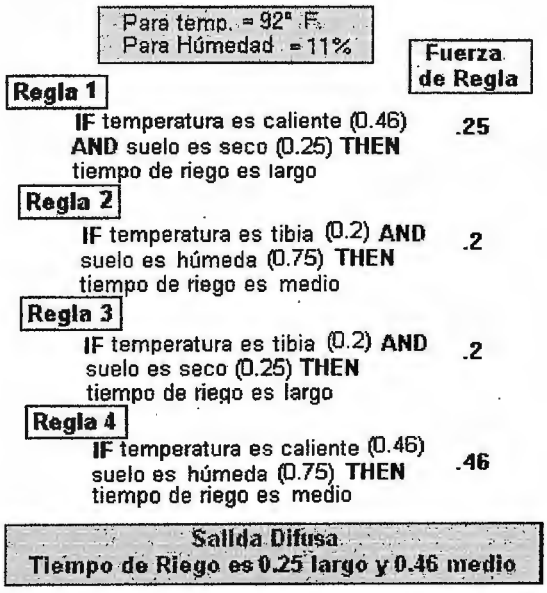


Figura A. 38 Valor final de las reglas.

A continuación se resumen los pasos que se ejecutan para la evaluación de reglas (ver Figura A.39):

- Crear reglas que describan el comportamiento deseado del sistema.
- Para un valor de entrada "crisp" particular determinar el grado de verdad de cada antecedente según la transformación de Fusificación utilizada.
- Encontrar el peso ó la fuerza completa de la regla, la cual es igual al mínimo de los grados de verdad de los antecedentes.
- Derivar la salida difusa, la cual es igual a la máxima fuerza ó peso de la regla de cada etiqueta consecuente.
- El método de evaluación de reglas utilizado aquí es llamado "Inferencia Mín-Max", dado que toma el valor mínimo de los antecedentes para determinar la fuerza de la regla; y el valor máximo de la fuerza de la regla de cada consecuente para determinar la salida difusa.

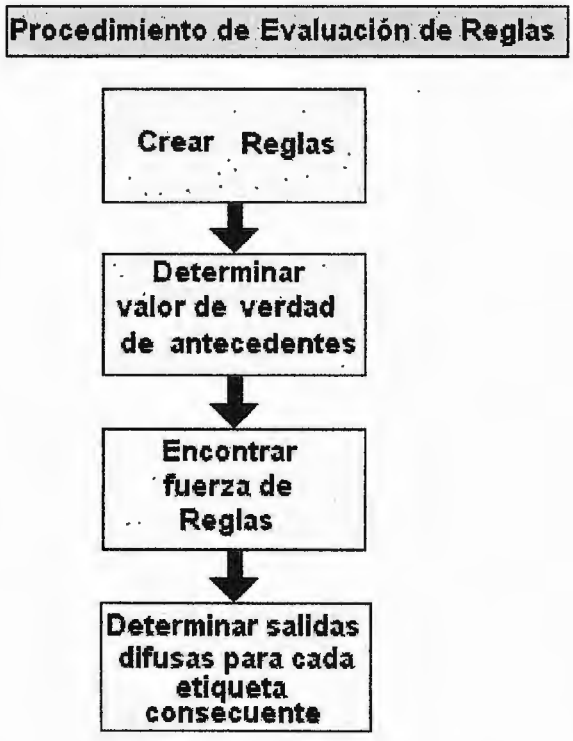


Figura A. 39 Procesos de evaluación de reglas.

Los anteriores pasos del proceso de evaluación de reglas se dividen en tres etapas. La primera etapa, llamada Agregación, concierne al desglose de cada antecedente de cada regla en proposiciones atómicas para obtener sus respectivos grados de membresías. El proceso de Agregación considera un conector ya sea AND u OR entre las proposiciones para obtener el valor final de grado de verdad de la regla. La siguiente etapa es la Activación. Esto involucra llevar el resultado final de antecedente al plano de la variable de salida, subordinado a la función de membresía del consecuente y aplicando la operación difusa determinada por el diseñador. La última etapa es la Acumulación que combina los resultados de todas las reglas para cada una de las variables lingüísticas de salida. En la siguiente Figura A.40 se aprecia la Acumulación utilizando el operador máximo.

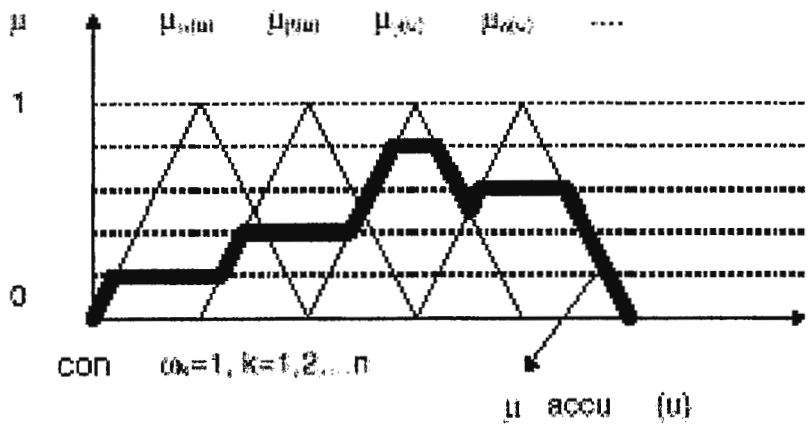


Figura A. 40 Acumulación.

A.7. DEFUSIFICACION.

En la Defusificación (ver Figura A.41), el resultado de la acumulación se convierte a un valor específico para la variable de salida. Una de las técnicas de Defusificación más comúnmente usadas es llamada método del Centro de Gravedad. En este método, para cada función de membresía de la variable lingüística de salida se recorta hasta el nivel indicado por el grado de pertenencia resultante de la evaluación

de reglas. A este recorte se le llama Corte Lamba (λ). Al aplicar el Corte Lambda, todo valor superior de la función de membresía es truncado($\mu_A(x) = \min[\mu_A(x), \lambda]$, ver Figura A.42). Las funciones de membresía "recortadas" resultantes son entonces combinadas para calcular el centro de gravedad total.

Figura A. 41 Defusificación.

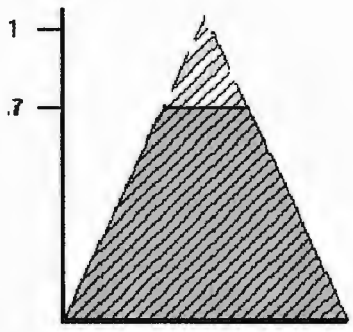


Figura A. 42 Función de membresía de salida recortada en el valor de salida difusa.

A diferencia de las entradas de cortes Alfa, los cuales son establecidos por el diseñador (o el usuario en algunos casos), las entradas Lambda son determinadas por el núcleo de inferencia difuso durante la ejecución de su proceso. Un ejemplo de los

Cortes Lambda es el corte de las funciones de membresía en la etapa final del proceso de evaluación de reglas. En el ejemplo del sistema de riego, cuando la temperatura es de 92°F y la humedad del suelo es del 11%, la función de membresía de salida podría lucir como el diagrama A de la Figura A.43. La salida difusa para cada función de membresía es mostrada en el diagrama B de la Figura A.43. Usando el método del centro de gravedad, las funciones de membresía de salida son recortadas como se muestra en el diagrama C de la Figura A.43.

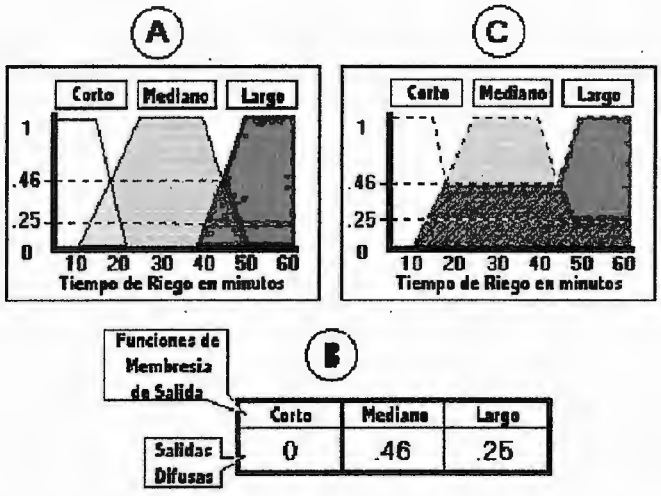


Figura A. 43 Funciones de membresía de salida para la duración de riego.

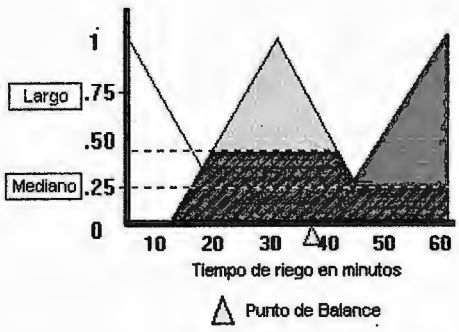


Figura A. 44 Salida defusificada.

El siguiente paso es encontrar el "punto de balance", del centro de gravedad del área sombreada (ver Figura A.44), esto representa la salida defusificada. Para el

modelo difuso construido, 38 minutos es la duración de riego correcta para una humedad del suelo del 11% y 92°F de temperatura del aire para es centro de gravedad resultante. La fórmula para calcular el centro de gravedad es:

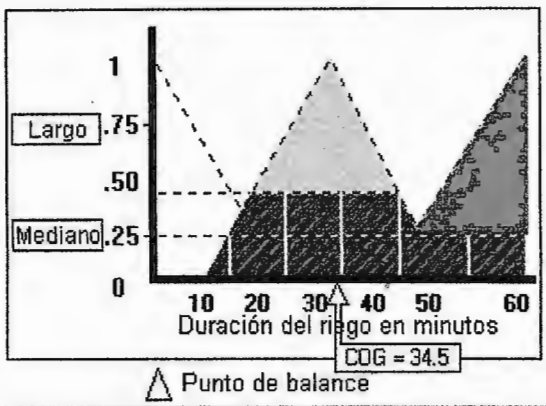
$$COG = \frac{\int_a^b \mu(x).x dx}{\int_a^b \mu(x) dx}$$

En la práctica las funciones de membresía de salida Singletons (que se describirán más adelante) son usadas muy a menudo. Esas funciones simplifican considerablemente el cálculo del centro de gravedad. En teoría, los Singletons permiten calcular el centro de gravedad sobre una continuidad de puntos en el dominio de salida "crisp".

$$COG = \frac{\sum_a^b \mu(x).x}{\sum_a^b \mu(x)}$$

Sin embargo, se podrá hacer un estimado bastante fiable sobre una muestra de puntos en el dominio de salida (ver Figura A.45). Experimentando con la técnica de Defusificación del centro de gravedad se puede observar como cambian las funciones de membresía de salida y sus respectivos puntos de balance (ver Figura A.46).

El método de Defusificación del centro de gravedad puede también aplicarse a funciones de membresía Singletons. El truncamiento de una función de membresía de Singleton resulta en una reducción de su altura como se ilustra en la Figura A.47.



$$COG = \frac{(15)\times(.3)+(25)\times(.46)+(35)\times(.46)+(45)\times(.46)+(55)\times(.25)}{.3+.46+.46+.46+.25} = \frac{66.55}{1.93} = 34.5$$

Figura A. 45 Cálculo del centro de gravedad.

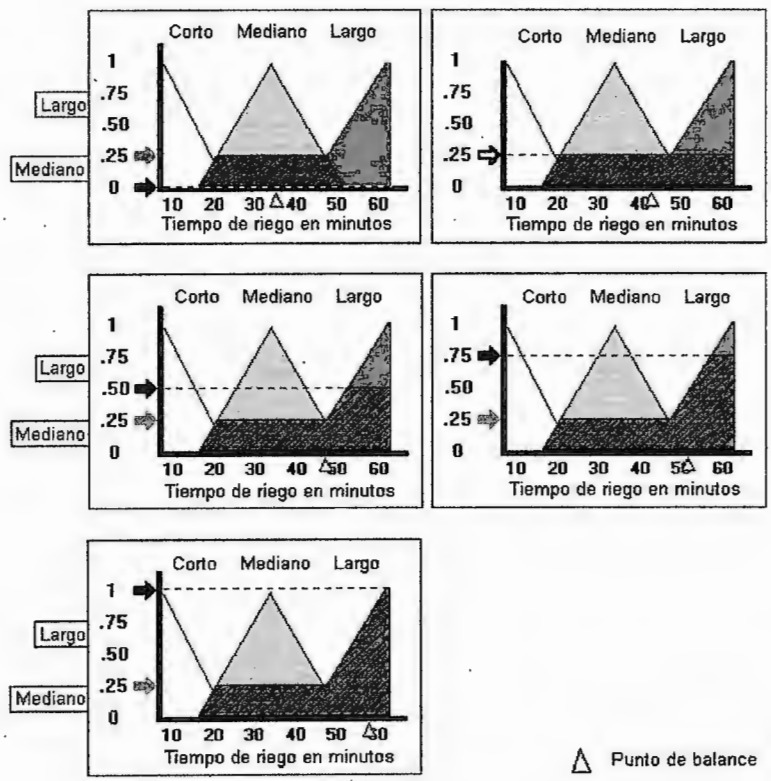


Figura A. 46 Cambios de puntos de balance.

Usando el método de Defusificación del centro de gravedad, las funciones Singletons son combinados usando un promedio de peso. La formula del centro de gravedad para el cálculo de Singletons se reduce a:

$$Salida\ (Y) = \frac{\sum_i (salida\ \ difusa\ _i) \times (posición\ \ del\ \ singleton\ \ en\ el\ eje\ x_i)}{\sum_i (salida\ \ difusa\ _i)}$$

Para el ejemplo mostrado en la Figura A.47, la salida es (ver Figura A.48):

$$\frac{(0)\times(0)+(.25)\times(30)+(.75)\times(60)}{0+.25+.75} = 52.5$$

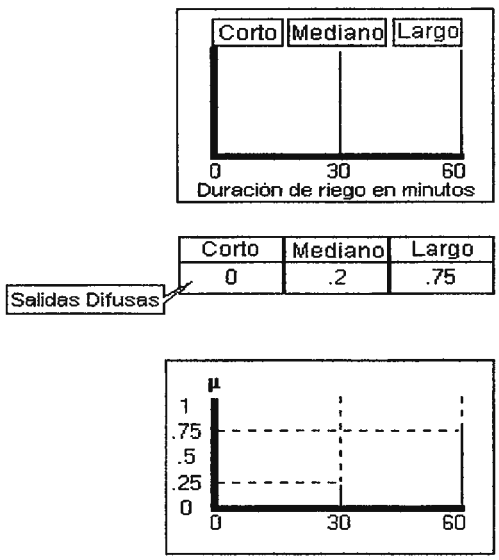


Figura A. 47 Valor de membresía de salida Singleton.

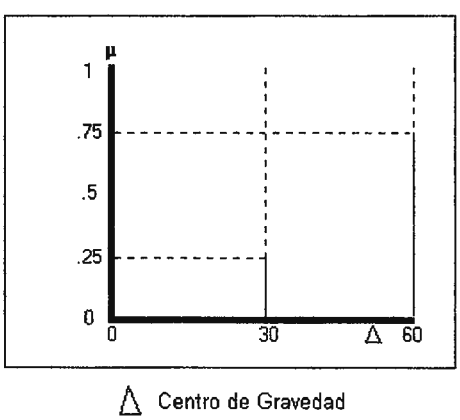


Figura A. 48 Singleton de salida.

La Defusificación de funciones Singletons tiene significativamente menos cálculo intensivo que otras funciones de membresía. Los Singletons son utilizados frecuentemente para representar las variables difusas de salida porque se obtienen controladores difusos de buen desempeño. Sin embargo usar funciones de membresía triangular o trapezoidal permite al controlador difuso tener salidas más finas. [1]

[1] NOTA: Mayor parte de esta información fue obtenida y traducida de [9] y [19].

APENDICE B. METODOLOGIA DE DISEÑO DE UN SISTEMA DIFUSO.

En este apéndice se estudia una secuencia de cinco pasos sobre los principios y procedimientos para diseñar un sistema basado en reglas de lógica difusa. Dichos pasos son los siguientes:

- Análisis y partición del sistema de control.
- Definición de las superficies de entrada y salida.
- Escritura de las reglas.
- Observación del comportamiento del modelo, inicio de verificación y sintonización.
- Optimización del sistema para la plataforma objetivo.

B.1 ANALISIS Y PARTICION DEL SISTEMA DE CONTROL.

El paso uno se subdivide en tres partes:

- Identificación de entradas y salidas.
- Análisis y simplificación del problema.
- Identificación de la unidad difusa.

B.1.1. IDENTIFICACION DE ENTRADAS Y SALIDAS.

Como en cualquier proceso de diseño, se debe comenzar con describir el sistema global a través de variables de entradas y salidas difusas y no difusa.

Específicamente, se necesita identificar que entradas van hacia el sistema y que salidas se obtendrán del sistema (ver Figura B.1).

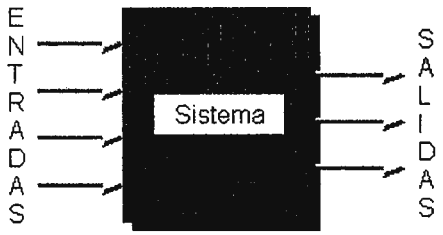


Figura B. 1 Identificación de entradas y salidas de un sistema.

Para ilustración de este paso se enfoca en el ejemplo del sistema de riego. El sistema de riego recibe entradas por medio de un arreglo de sensores (ver Figura B.2). Dichos sensores detectan condiciones tales como:

- Temperatura del Aire.
- Humedad del Suelo.
- Humedad del Ambiente.
- Tiempos de riego fijados por el usuario. (Este sistema esta equipado con un cronómetro de tiempo programable que permite irrigar varias porciones del terreno en intervalos de tiempo específicos).

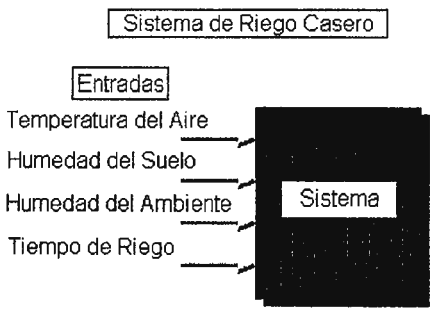


Figura B. 2 Entradas para el sistema de riego.

Las salidas del sistema serán las duraciones de tiempo de riego (ver Figura B.3) controladas por señales eléctricas que apagan y encienden varias válvulas para regular el riego de la siguiente manera:

- Duración de tiempo 1.
- Duración de tiempo 2.
- Duración de tiempo 3.

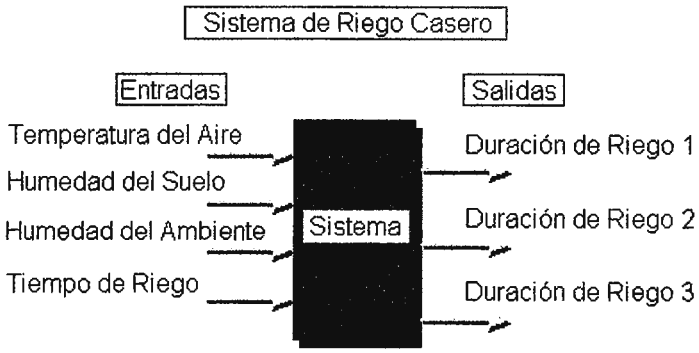


Figura B. 3 Salidas para el sistema de riego.

Luego que se han definido las variables de entrada y salida del sistema, se puede continuar con el siguiente paso. Sin embargo, como el diseño es un proceso iterativo, es necesario volver e ir hacia adelante entre este paso y el próximo (ver Figura B.4) para analizar y simplificar el sistema.

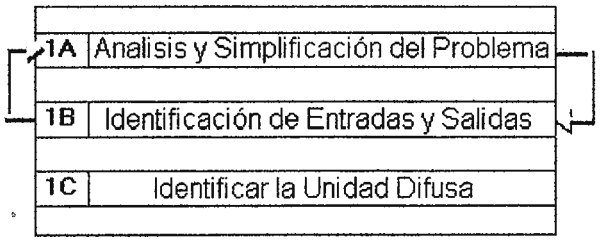


Figura B. 4. Proceso iterativo de diseño.

B.1.2. ANALISIS Y SIMPLIFICACION DEL PROBLEMA.

Ahora que se han identificado las entradas y salidas globales del sistema, se puede continuar con la descomposición funcional del mismo. En este paso, es necesario identificar subsistemas funcionales individuales (sus entradas y salidas) que puedan ser modeladas independientemente. Para el ejemplo, si las salidas de duración de riego no están relacionadas, se debe desear tener tres subsistemas funcionales, uno para cada salida de control (ver Figura B.5).

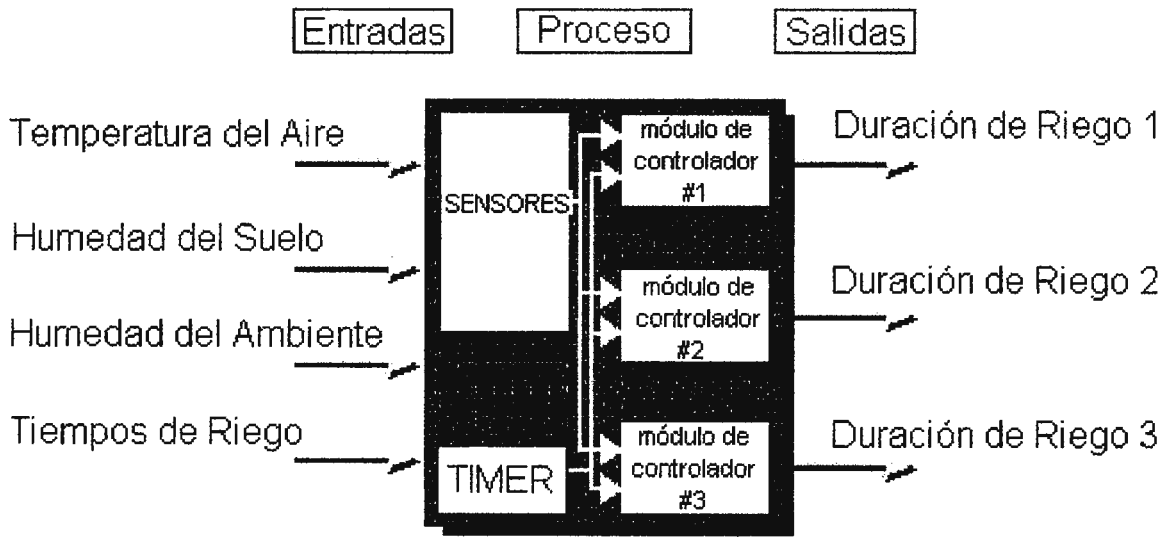


Figura B. 5. Subsistemas funcionales para el sistema de riego.

La meta es tener un subsistema funcional que pueda ser modelado fácilmente. Si el subsistema que se ha creado no cumple esto, entonces este se debe descomponer más.

B.1.3. IDENTIFICACION DE LA UNIDAD DIFUSA.

Después que se ha descompuesto funcionalmente un sistema, se deben identificar subsistemas candidatos que puedan ser modelados usando lógica difusa (ver Figura B.6).

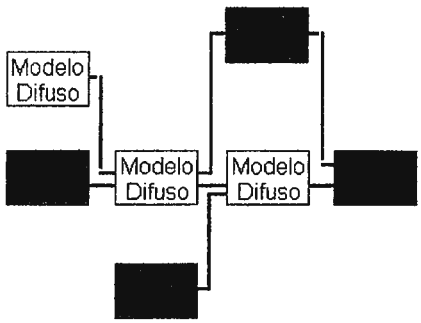


Figura B. 6. Identificación de subsistemas candidatos a ser modelados por lógica difusa.

Un buen subsistema candidato para la lógica difusa es una unidad funcional en la cual el flujo de información sea impreciso (por ejemplo ruidoso u orientado lingüísticamente). Aplicando esos principios al ejemplo del sistema de riego se facilitará la identificación de las unidades difusas. Debido a que el temporizador del sistema de riego es muy preciso, este no requiere lógica difusa. La relación entre la temperatura del aire, la humedad del suelo y la duración de riego es un poco vaga. El módulo controlador para estas variables debe ser un buen candidato para el modelado difuso.

B.2. DEFINICION DE LAS SUPERFICIES DE ENTRADA Y SALIDA.

Una vez se han identificado las unidades difusas de un sistema, se puede empezar a construir el modelo difuso. Este paso se subdivide en los siguientes procesos:

- Especificación del universo de discurso.
- Ajuste de escalas del universo en discurso.
- Determinación del numero y distribución de funciones de membresía.
- Comparación de los métodos de defusificación.

B.2.1. ESPECIFICACION DEL UNIVERSO DE DISCURSO.

Los expertos definen el universo de discurso de diferentes maneras: algunos lo especifican dentro de un contexto muy amplio como el rango de los números reales desde el infinito negativo hasta el infinito positivo en las unidades de las variables de entrada y salida bajo consideración; otros limitan el universo de discurso al rango de interés para la aplicación dada. A lo largo de todo este documento se utilizará la definición últimamente descrita, ya que es esta la que teóricamente mejor se adecua para la mayoría de aplicaciones. La razón de esto se explicará cuando se discuta la defusificación. La Figura B.7 ilustra unas funciones de membresía dentro de un posible universo de discurso.

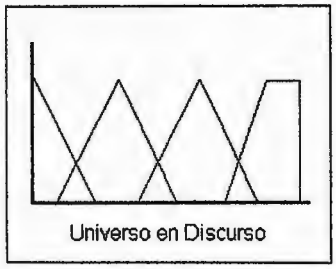


Figura B. 7. Funciones de membresía de un universo de discurso.

Para especificar un universo de discurso, primero se debe determinar el rango aplicable para una variable característica en el contexto en que el sistema ha sido diseñado. El rango seleccionado debe ser cuidadosamente considerado. Por ejemplo, si se especifica un rango que es muy corto, regularmente ocurre que algunos datos quedan fuera de la escala (ver Figura B.8), lo cual puede impactar en el funcionamiento del sistema global.

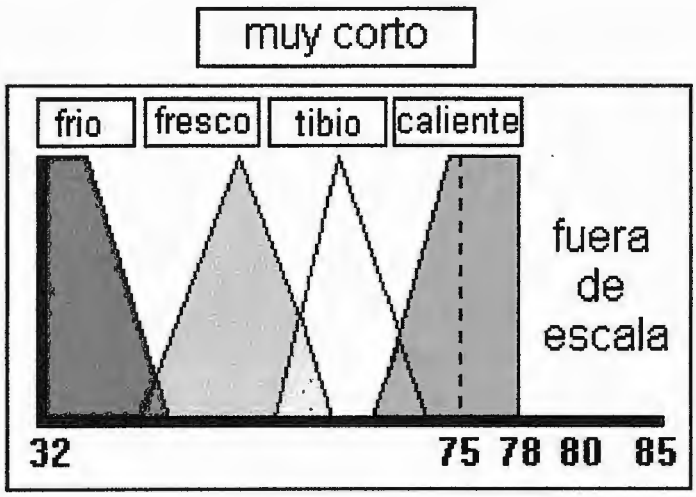


Figura B. 8. Universo de discurso muy corto.

Refiriéndose a la Figura B.8, si el límite final del rango de temperatura es colocado a 78°F y la temperatura en la región regularmente excede ese valor, la respuesta del sistema dependerá de la implementación del hardware en lugar de un buen diseño y simulación del modelo difuso. Recíprocamente, si el universo de discurso para una entrada es muy largo, la tendencia será funciones de

membresía de "respuesta plana" muy grande (significa que los cambios en los valores de entrada no cambiarán la respuesta de la salida)y afectará la captura de los valores de entrada de los extremos (ver Figura B.9). Esto estaría bien si se busca alguna salida saturada del sistema para los valores de entrada extremos, pero fuera de eso puede ser inapropiado. Para el ejemplo del sistema de riego, se ha especificado el rango de temperatura del aire entre 32°F y 110°F para el universo de discurso (ver Figura B.10). El límite más pequeño, 32°F, tiene sentido debido a que el sistema de riego no funcionará debajo del congelamiento. El límite más alto es también lógico debido a que la mayoría de personas podrían considerar que por encima de 98°F es demasiado caliente y echaría a perder el césped. De esta manera, se ha definido subjetivamente el universo de discurso para la variable difusa temperatura del aire.

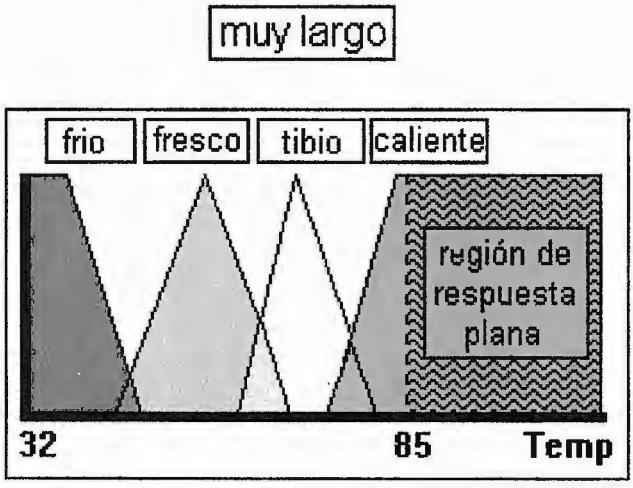


Figura B. 9. Universo de discurso muy largo.

Lineamientos similares se usan para determinar subjetivamente el universo de discurso para la humedad del suelo que es de 0% a 30%. La Figura B.10 muestra las funciones de membresía para las dos variables de entrada.

El rango del universo de discurso para cada salida debe de ser también cuidadosamente considerado. Generalmente ocurren problemas si el universo de discurso de la salida es muy largo, ya que ambos extremos estarían muy lejos de

los límites de "máximo significado". Al igual que las variables de entrada, la tendencia a hacer las funciones de membresía de la variable de salida muy largas dará como resultado una "respuesta plana en cada extremo" (ver Figura B.11).

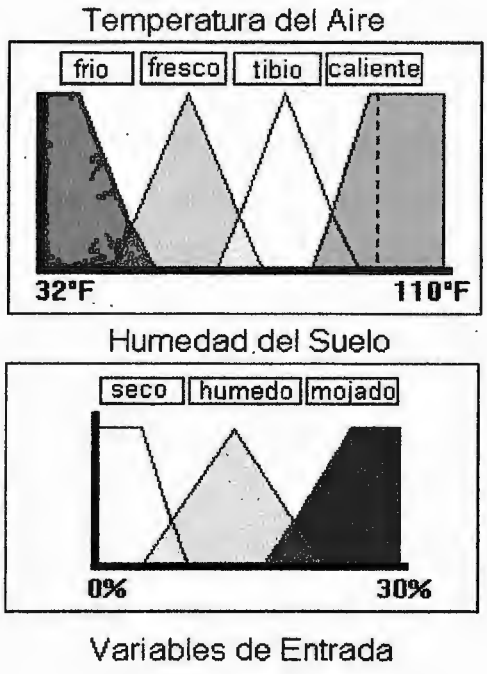


Figura B. 10. Funciones de membresía para las variables entradas.

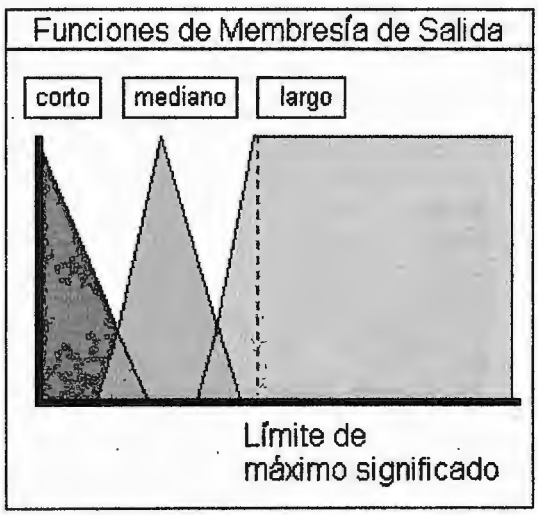


Figura B. 11. Rango del universo de discurso muy largo.

Este problema se manifiesta en la defusificación, tales funciones de membresía desproporcionadamente largas predominarán sobre las otras por prestar su masa total para el proceso de defusificación (ver Figura B.12). Cuando las funciones de membresía son desproporcionadamente largas, el resultado de la defusificación será fuertemente influenciado por la masa de la función de membresía larga. Un efecto similar puede ocurrir cuando los singletons de salida son posicionados desproporcionadamente al extremo izquierdo y/o derecho, esto podría provocar una respuesta del sistema similar a cuando un niño se sienta en el extremo de una silla de balance con el otro extremo vacío. La Figura B.13 muestra las funciones de membresía para la variable de salida duración de riego del sistema de riego con el universo de discurso fijado entre 0 min. y 70 min.

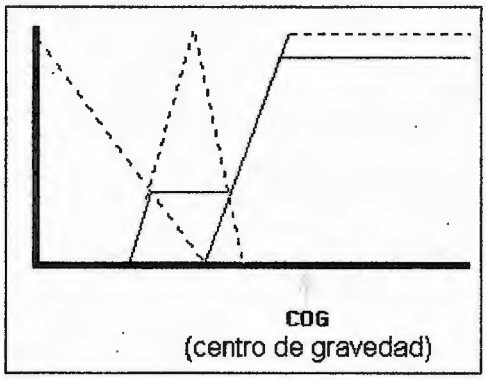


Figura B. 12. Resultado de las funciones de membresía muy largas en la defusificación.

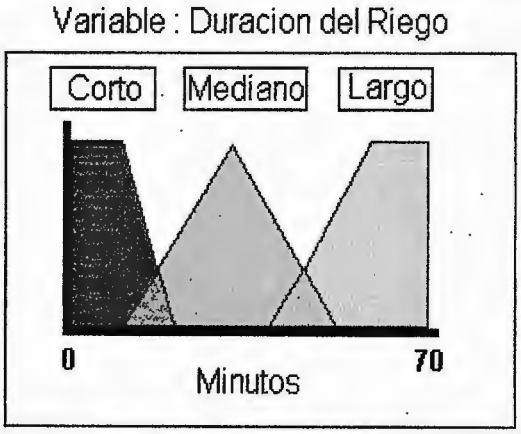


Figura B. 13. Funciones de membresía de salida.

B.2.2. AJUSTE DE ESCALA DEL UNIVERSO DE DISCURSO.

Es usualmente deseable, y a menudo necesario, el ajustar o normalizar el universo de discurso de una variable de entrada o salida. Por ejemplo, el sensor de temperatura en el sistema de riego debe proporcionar una salida de voltaje entre –12VDC y 12VDC, la cual representa el rango de lecturas de temperatura que van desde –20°F hasta 120°F. Sin embargo el rango de temperatura de interés debe estar solamente entre –32°F y 110°F (como se definió en la sección previa). Pero aún más importante es que los componentes de hardware puedan ser capaces de aceptar voltajes de entrada entre 0 y 5 voltios con una resolución de 8 bits. El ajuste de escala es necesario para trazar el voltaje medido dentro de diferentes rangos.

Cuando el rango de las entradas o salidas cubre un rango extremadamente ancho, la representación logarítmica puede ser apropiada. Otras escalas no lineales pueden también ser usadas. Dichas escalas son más apropiadas cuando la clasificación más activa esta en el área de la menor distorsión lineal, ver Figura B.14.

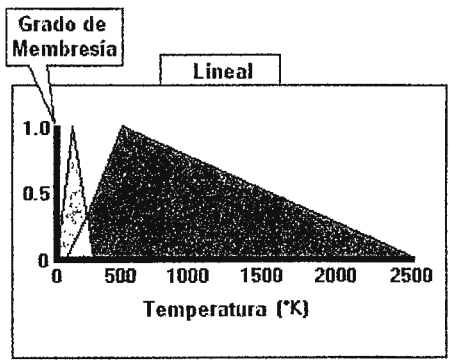


Figura B. 14. Representación logarítmica para funciones de membresía.

Hay que prestar particular atención a las unidades de medida cuando se especifica un universo de discurso, ya que variables correspondientes al mismo fenómeno físico deben de ser medidas en las mismas unidades. Por ejemplo, si se mide un potencial en libras pies, no se debe especificar la distancia en metros. La falta de cohesión en las dimensiones de calibración entre el control y la medición de variables se presta a causar fallas en el modelo.

A menudo, los sensores que trasladan valores observados a valores apropiados para las máquinas son no lineales. Similarmente, actuadores de salida pueden tener relaciones no lineales con la salida eventual del fenómeno (ejemplo, la lectura de la temperatura podría tener una relación no lineal con la salida de voltaje de un termostato). Si una relación no lineal existe entre la entrada del sistema deseado (la temperatura en este caso) y la salida de un circuito sensor de detección, los diseñadores prefieren manejar directamente con las unidades de valores de escala de la circuitería de detección del sensor. De esta manera, en lugar de tener un universo de discurso que va desde –40°F a 140°F, este puede ser de 0 a 5 voltios. La Figura B.15 nos muestra una ilustración de lo antes discutido.

La salida ideal de un convertidor analógico a digital de 8 bits de un microcontrolador varía desde $00 (0 = -40°F) hasta $FF (5 V = 140°F), donde la resolución mínima es:

$$\frac{5}{2^8} = \frac{5}{246} = 0.0195\,(voltios) = 3.52\,(°F)$$

Si se desarrolla una aplicación sin tomar en cuenta la implementación electrónica, un proceso de conversión debe ser requerido. Por ejemplo, si se desean funciones de salida que han sido identificadas (ver el diagrama A de la Figura B.16) y la función de transferencia del sensor de detección y el circuito de escala es no lineal (ver diagrama B de la Figura B.16), entonces se tienen varias opciones:

- Transformar las funciones de membresía usando tablas de búsqueda para compensar la falta de linealidad del sensor.
- Aplicar las unidades de escala implementadas a las funciones de membresía y ajustar esas funciones tratando de mantener la forma (ver Figura B.17).

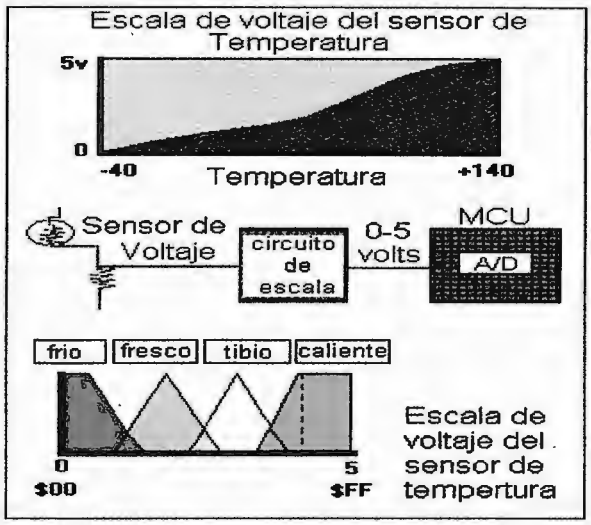


Figura B. 15. Consideraciones de implementación para los sensores.

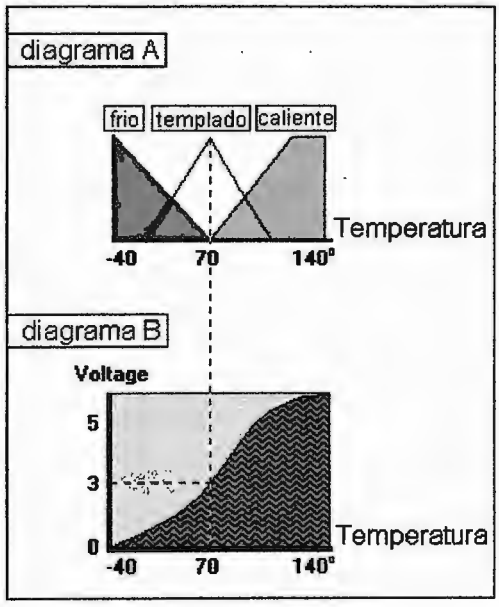


Figura B. 16. Funciones de membresía y función de transferencia del sensor de temperatura.

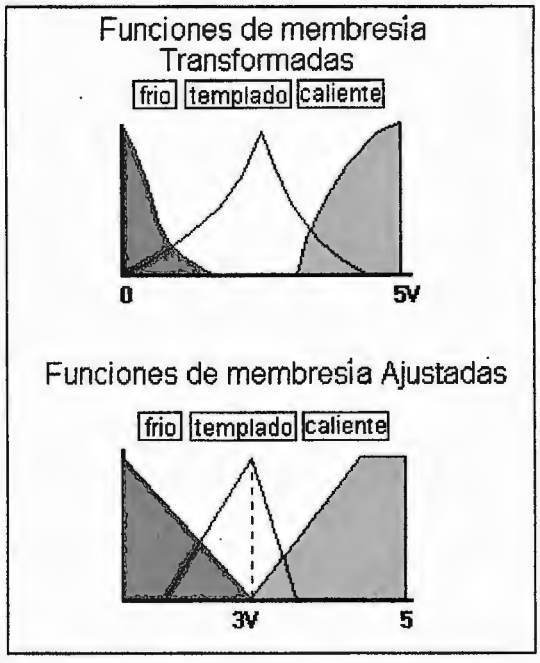


Figura B.17. Ajuste de las funciones de membresía.

B.2.3. DETERMINACION DEL NUMERO Y DISTRIBUCION DE LAS FUNCIONES DE MEMBRESIA.

Primero se estudiará la determinación del número de funciones de membresía; y al final de esta sección, lo referente a la distribución de las funciones de membresía.

Hay varios aspectos a considerar cuando se determina el número de funciones de membresía y sus características de traslapamiento. El número de funciones de membresía es a menudo impar, por lo general, cualquier número entre 3 y 9. Como una recomendación para un mayor control (por ejemplo, incrementar la sensibilidad de salida para los cambios de entrada), debe de

aumentarse la densidad de funciones de membresía para la región de la entrada de interés (ver Figura B.18).

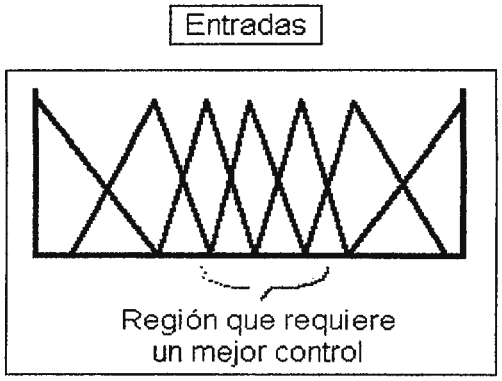


Figura B. 18. Mayor densidad de las funciones de membresía.

Si se tienen muy pocas funciones de membresía para una aplicación dada (ver el diagrama A de la Figura B.19), la respuesta puede ser muy letárgica y puede fallar al no proporcionar a tiempo la salida de control para recuperarse de un cambio de entrada pequeño. Esto puede causar que el sistema oscile alrededor del valor deseado. Muchas funciones de membresía (ver el diagrama B de la Figura B.19) pueden rápidamente activar muchas reglas para pequeños cambios en los valores de entrada, resultando en grandes cambios de la salida que puede causar inestabilidad en el sistema.

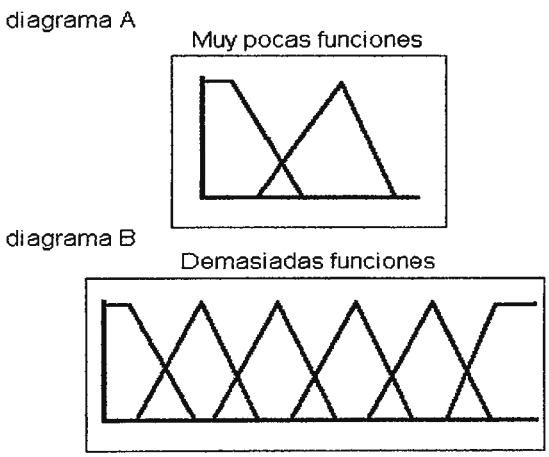


Figura B. 19. Determinación del número de funciones de membresía.

En un sistema de control difuso, el grado de traslapamiento entre las funciones de membresía impacta sobre el funcionamiento del sistema. Por otra parte, un sistema difuso sin funciones de membresía traslapadas se convierte en un sistema Booleano. Dado un conjunto de entradas "crisp", se activarán múltiples reglas debido al traslapamiento de las funciones de membresía definidas para cada variable de entrada. El inapropiado traslapamiento de las funciones de membresía puede llevar al sistema a comportamientos aleatorios del de control difuso resultante.

Los lineamientos generales concernientes al traslapamiento de las funciones de membresía son:

- Cada punto en el universo de discurso debe corresponder al dominio de al menos una función de membresía; a la misma vez, cada punto debe de pertenecer al dominio de no más de dos funciones de membresía.
- Dos funciones de membresía no deben de tener el mismo punto de máximo significado.
- Cuando dos funciones de membresía se traslapan, la suma de los grados de membresía para cualquier punto en el traslapamiento debe de ser menor o igual que uno.
- Cuando dos funciones de membresía se traslapan, el traslapamiento no debe de cruzar el punto de máximo significado de cualquier función de membresía.

Steve Marsh (del Centro de la Tecnología Computacional Emergente de Motorola Inc.) ha propuesto dos índices para describir el traslape de las funciones de membresía cuantitativamente. La Figura B.20 ilustra de donde se han estos índices.

$$\text{Relación de Traslapamiento} = \frac{\text{pendiente de traslapamiento}}{\text{pendiente de función de membresia adyacente}}$$

$$\text{Robustez de Traslapamiento} = \frac{\text{área de traslapamiento sumado}}{\text{área máxima de traslapamiento sumado}}$$

$$= \frac{\int_L^U (\mu_{A1} + \mu_{A2})dx}{2(U-L)}$$

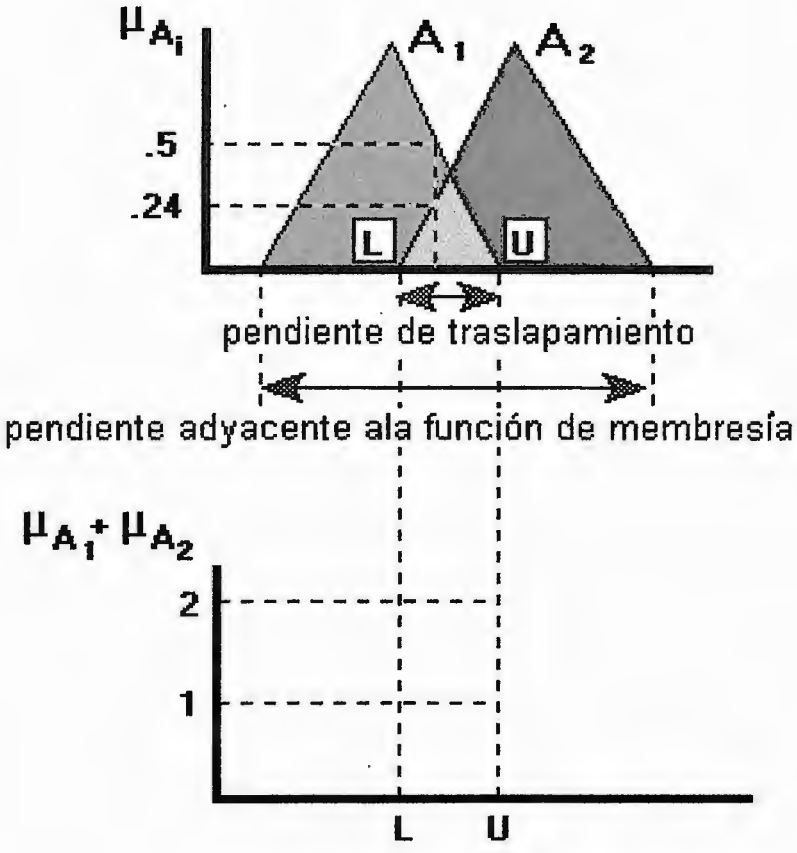
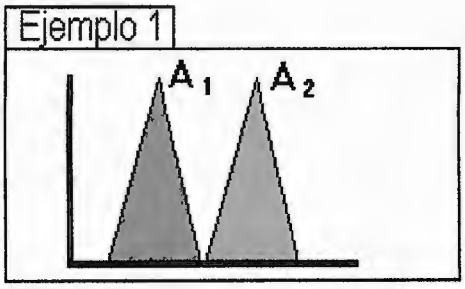


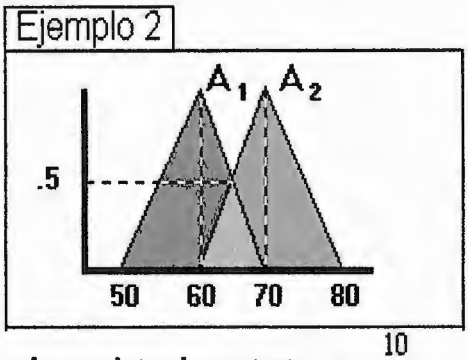
Figura B. 20. Indices de traslapamiento.

Además se muestran en la Figura B.21 ejemplos de traslapamiento de funciones de membresía.



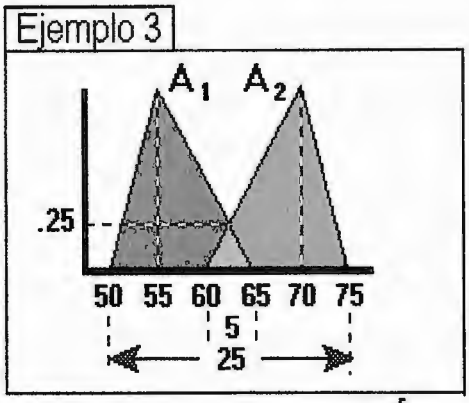
relacion de traslapamiento = 0

robustez del traslapamiento = 0



$$\textit{relacion de traslapamiento} = \frac{10}{30} = 0.333$$

$$\textit{robustez del traslapamiento} = \frac{10}{20} = 0.5$$



$$\textit{relacion de traslapamiento} = \frac{5}{25} = 0.2$$

$$\textit{robustez del traslapamiento} = \frac{2.5}{10} = 0.25$$

Figura B. 21. Ejemplos de traslapamiento.

Como una recomendación, la relación de traslapamiento debe mantenerse entre 0.2 y 0.6. El valor de la robustez de traslapamiento es usualmente más grande que la relación de traslapamiento; ella debe mantenerse entre 0.3 y 0.7. Un modelo de control difuso con una relación de traslapamiento o robustez de traslapamiento muy alta puede usualmente enfrentar más ambigüedad. Para una operación más uniforme de un modelo de control difuso, se debe seleccionar una adecuada relación de traslapamiento (alrededor de 0.33) y una robustez del traslapamiento (alrededor de 0.5). Si se desea un grado más grande de control, una relación de traslapamiento y una robustez del traslapamiento más alta usualmente ayudará. Valores índices más altos conllevan a una operación más uniforme y un mejor control de relaciones ambiguas de entrada/salida. En cambio

un valor índice más bajo conviene para relaciones de entrada/salida con alta correlación.

La forma de las funciones de membresía influye en los índices de traslapamiento. Las formas más comunes para las funciones de membresía son la trapezoidal y el singleton. Estas son usadas para acomodar implementaciones de hardware de bajo costo. Ocasionalmente, funciones de membresía de forma de campana son usadas pero la suavidad adicional proporcionada por esta forma es a menudo no perceptible en aplicaciones de control. La Figura B.22. muestra las diferentes formas posibles para las funciones de membresía en las aplicaciones de control. Periódicamente, las formas de las funciones de membresía representan una compleja respuesta del sistema del modelo que esta siendo usado. En implementación de hardware se usan tablas de búsqueda para definir la forma de las funciones de membresía complejas.

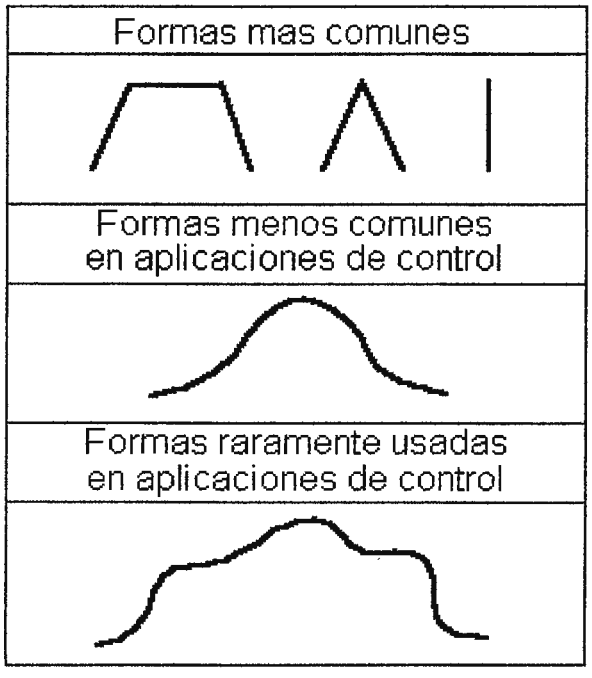


Figura B. 22. Formas de las funciones de membresía.

B.2.4. COMPARACION DE LOS METODOS DE DEFUSIFICACION.

Como se menciona anteriormente en este documento, el proceso de defusificación manipula los resultados de la evaluación de reglas difusas para obtener un valor de salida "crisp" que pueda ser entendido por la mayoría de hombres y máquinas. El método de defusificación debe definirse en el comienzo del desarrollo de un sistema junto con la definición de las funciones de membresía de entrada y salida. La tabla B.1 brinda una vasta comparación de diversos métodos de defusificación.

Características de la salidas "crips"	**Método de Defusificación**
Requerimiento de Memoria (1) Velocidad (5) Algunas Fluctuaciones de Salida	Centro de Gravedad (con funciones de membresía de salida Singletons)
Requerimiento de Memoria (1) Velocidad (5) Salida Suave	Centro de Gravedad (con funciones de membresía de salida no Singletons)
Requerimiento de Memoria (2) Velocidad (4) Salida un Poco Optimista	Máximo Izquierdo
Requerimiento de Memoria (2) Velocidad (4) Salida muy Optimista	Máximo Derecho
Requerimiento de Memoria (4) Velocidad (3) Salida Optimista	Promedio del Máximo
Requerimiento de Memoria (3) Velocidad (4) Salida Optimista	Punto Medio del Máximo
Requerimiento de Memoria (5) Velocidad (1) Salida Suave	Medio

Tabla B.1. Cuadro comparativo de métodos de defusificación.

Nota: Requerimiento de Memoria: (1) = bajo y (5) = alto

Velocidad de Procesamiento: (1) =baja y (5) = alta

B.3. ESCRITURA DE REGLAS.

Para la escritura de las reglas, se involucran dos fases o procesos:

- Codificación de reglas.
- Sintonización de reglas.

En la primera fase se debe de codificar el conocimiento del comportamiento del sistema descrito. Esta codificación resulta de las observaciones obtenidas de:

- Operadores del sistema.
- Entrevistas con expertos.
- Uso de superficies de control existente que describen el comportamiento modelo.

Las generalidades anteriores proveen una buena descripción del modelo. Dicho comportamiento del modelo es definido por las reglas, las cuales trazan la relación existente entre las etiquetas de entrada y las etiquetas de salida. Las reglas representan el conocimiento base de un sistema o aplicación. En esta fase se involucra la descripción de todas las reglas que son prácticas para describir un sistema, en esencia lo que se hace es:

- Primero escribir todas las reglas que son obvias.
- Luego se escribirán las reglas menos obvias pero intuitivamente son correctas.

Después de que el modelo ha sido simulado y su comportamiento observado, se estará listo para la segunda fase de la escritura de reglas: sintonizar. En esta fase las reglas individuales se sintonizan de tal manera que las contribuciones de cada una reflejen en la superficie de control el comportamiento deseado del sistema que esta siendo modelado. Las áreas a considerar en la fase de sintonía son:

- Operadores Compensatorios.
- Cortes Alpha.
- Contribución de pesos.

Estas técnicas son opcionales y dependen de la precisión y funcionamiento del modelo deseado, así como la habilidad del ambiente de desarrollo para soportar estas opciones.

B.3.1. ESCRITURA DE REGLAS OBVIAS.

Para un sistema de dos entradas y una salida, se pueden escribir las reglas preparando una matriz, la cual es referida como una Memoria Asociativa Difusa (MAD). Una MAD usa las etiquetas de una variable de entrada para nombrar la fila y la otra etiqueta de la otra variable de entrada para nombrar la columna. Las celdas en la matriz se llenan con la etiqueta de salida deseada para cualquier combinación de entradas. Entonces cada celda puede ser convertida a una regla lingüística. Una MAD puede también facilitar la escritura de reglas para un sistema de control con más de dos entradas y una salida. Para hacer esto se prepara una MAD de dos entradas y una salida manteniendo otras entradas y salidas constantes (ver Figura B.23). Llenando cada celda en la MAD podría describirse por completo el comportamiento del modelo. Aunque es posible, usar MAD´s para modelos complejos con muchas entradas y salidas es impráctico.

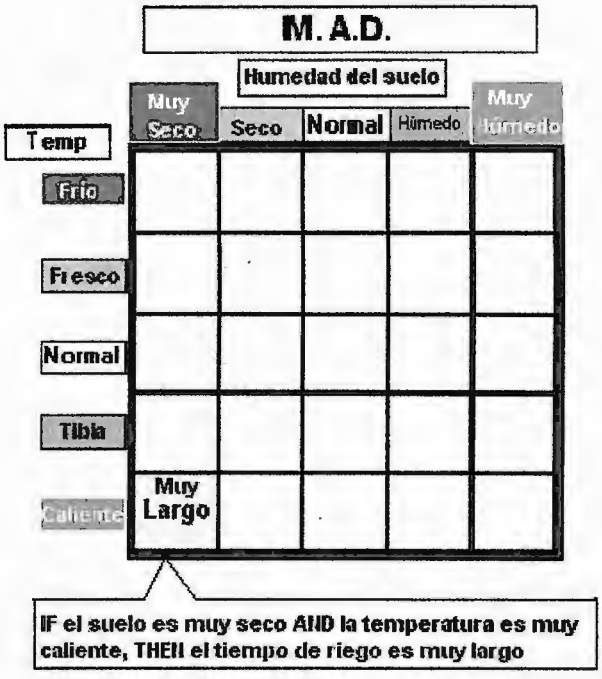


Figura B. 23. Memoria asociativa difusa.

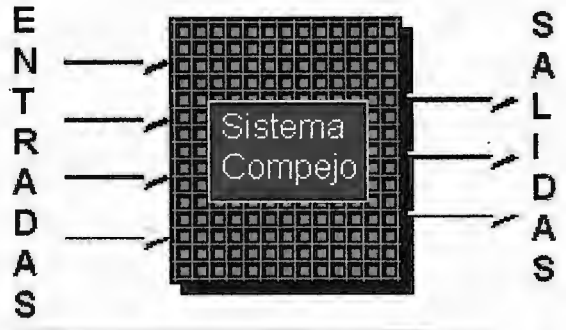
Sistemas complejos con múltiples entradas y salidas podría requerir de muchas matrices para describir por completo todas las combinaciones de entradas y salidas, para estos casos se enfrentaría el problema de la siguiente manera:

- Describir las reglas que se sobreentienden.
- Describir las reglas que son vagas pero intuitivamente correctas.

El número de reglas eventualmente instaladas en el controlador difuso puede ser menos que el número total de reglas posibles. Esto es especialmente cierto en un sistema con muchas entradas, salidas y etiquetas, (ver Figura B.24).

Para construir reglas con una MAD, primero se llenan las áreas donde las acciones son intuitivamente obvias. Para el ejemplo del sistema de riego, las acciones obvias de la MAD corresponden a las esquinas superior derecha e

inferior izquierda. La duración del riego para las condiciones representadas aquí es bastantes obvia (ver Figura B.25).



Si cada entrada tiene 5 etiquetas, el número de reglas posibles puede ser:

$$5^4 \times 3 = 1{,}875$$

En realidad, el número de reglas podría ser mas como 20 - 40-

Figura B.24. Número de reglas para un sistema complejo.

Temp	Humedad del suelo: Muy Seco	Seco	Normal	Húmedo	Muy Húmedo
Frío				Corto	Corto
Fresco					Corto
Normal	Largo				Corto
Tibio	Largo				
Caliente	Largo	Largo			

Figura B. 25. Memoria asociativa difusa para la duración de riego.

B.3.2. ESCRITURA DE REGLAS MENOS OBVIAS.

El siguiente paso es llenar las celdas vacías de la MAD con reglas menos obvias, tratando de evitar transiciones bruscas entre celdas (ver Figura B.26). Retomando el ejemplo anterior del sistema de riego, las siguientes reglas ofrecen una apropiada transición de tiempo de riego largo a tiempo de riego corto sobre diferentes condiciones ambientales.

- **IF** temperatura es caliente **AND** humedad del suelo es normal **THEN** el tiempo de riego es largo.
- **IF** temperatura es tibia **AND** humedad del suelo es húmeda **THEN** el tiempo de riego es medio.

Las celdas restantes de la MAD deben ser llenadas en forma similar. Para cualquier aplicación la primera fase de escritura de reglas termina aquí.

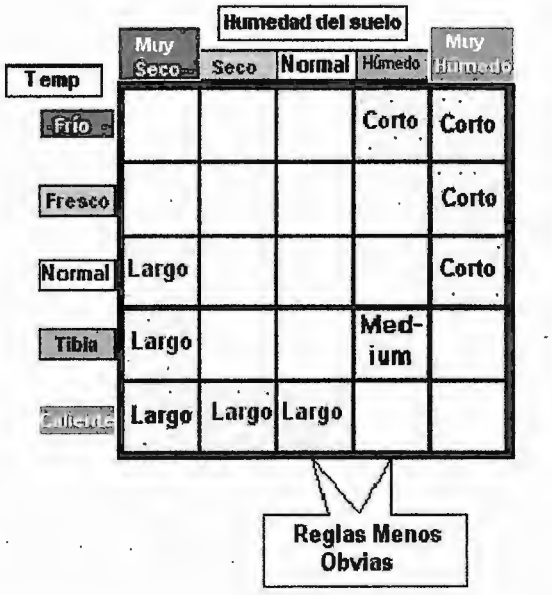


Figura B.26. Escritura de las reglas menos obvias.

B.3.3. CASOS ESPECIALES.

Algunas veces existen comportamientos particulares del sistema a controlar que deben considerarse. Estos casos especiales pueden representarse en el modelo difuso como afirmaciones y excepciones.

En algunas aplicaciones, se puede tener un detalle adicional que no se toma en cuenta y que obliga al sistema a manejar la salida en una forma en particular (ver Figura B.27). Esto es usualmente manejado por una afirmación. Por ejemplo, si la conservación del agua es obligatoriamente importante, se puede forzar el tiempo de riego a que sea "corto", incluyendo la siguiente afirmación:

"Tiempo de riego es corto"

Esto forzará a la superficie de salida hacia un tiempo de riego corto. Como resultado se evitarán los tiempos de riego extremadamente cortos y largos, haciendo todo el sistema más eficiente.

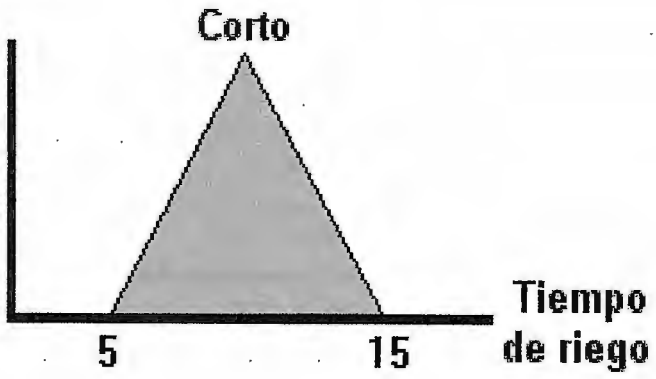


Figura B.27. Afirmaciones.

En un modelo difuso una afirmación puede ser condicional o incondicional. Una afirmación incondicional establece una región difusa que actúa como límite fronterizo del espacio de la solución consecuente. Por ejemplo, el enunciado "nuestros precios deben de ser altos" es una afirmación incondicional. Cuando una afirmación incondicional usando al conjunto difuso ***W*** se aplica a la región difusa ***S***, la región consecuente ***S**** se le impone el limite:

$$S^* = min\,[\mu_W(X), \mu_S(X)]$$

En cambio, una afirmación condicional difusa constituye una regla difusa, por ejemplo: "Si la competencia de precios no es muy alta, entonces nuestros precios deben de estar cerca al de la competencia". El valor real de esta afirmación será determinado por el método de inferencia seleccionado por diseñador del sistema difuso.

Las excepciones son condiciones que requieren de una acción a ser tomada fuera del modelo original. Tradicionalmente las excepciones se manejan como una función límite antes de utilizar el modelo de control actual. Sin embargo en un sistema de control difuso, el manejo de excepciones puede ser parte del modelo de control. Por ejemplo, el tiempo de riego no puede ser utilizado para compensar temperaturas muy frías. Si se detectan temperaturas frías, entonces se puede alertar al operador para tomar una acción adicional que proteja el jardín. Tal acción podría ser codificada como una regla adicional dentro d el modelo (siempre y cuando el ambiente de desarrollo lo permita), ver Figura B.28.

Se debe de recordar que las reglas sólo contribuyen a la salida si estas obtienen grados de verdad en sus antecedentes. Como la defusificación intentará inferir un valor crisp de una región difusa (o grupo de singletons), es importante que más de una regla se active para un conjunto de entradas porque hace al sistema difuso más robusto y menos vulnerables a perdidas de señal o a combinación de entradas inesperadas. Por lo anteriormente mencionado es

necesario familiarizarse como opera todo estos procesos sobre los conjuntos difusos. Usualmente durante la escritura de reglas se puede asumir lo siguiente:

- Operaciones Mínimo y Máximo para los operadores AND y OR.
- Método MIN-MAX para la inferencia difusa.
- Método de Centro de Gravedad (CDG), para la defusificación.

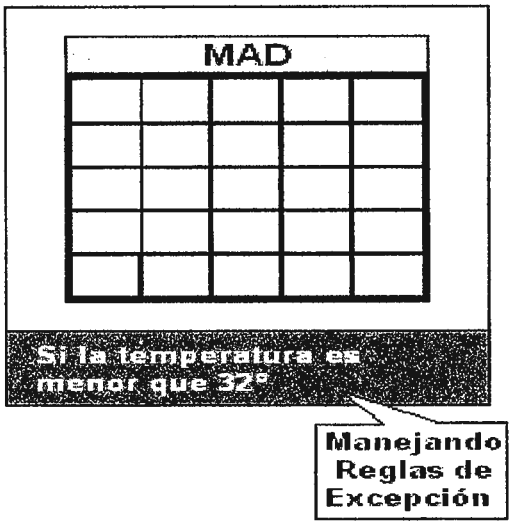


Figura B.28. Excepciones.

B.3.4. SINTONIA DE REGLAS.

Una vez el modelo es ejecutado y la salida simulada determinada, el comportamiento de salida puede ser modificado al ajustar las reglas individualmente para que reflejen con más precisión la salida deseada del sistema. Las tres características que deben de ser ajustadas son:

- Cortes Alpha.
- Contribución de pesos.
- Operadores compensatorios

La necesidad de utilizar estas técnicas es determinada por el comportamiento de salida observado. También hay que tomar en cuenta que no todos los ambientes (plataformas) de desarrollo soportan estas técnicas.

Si la salida de un sistema muestra una pequeña cantidad de ruido en áreas para la cual la respuesta debe de ser plana, es probable que algunas reglas con valores pequeños de grados de verdad se están activando. Esto puede arreglarse rastreando la regla que produce el error y agregando cortes Alpha arriba del nivel de ruido. Por ejemplo, se puede fijar cortes Alpha cuando no se quiere un tiempo de riego muy largo a menos que las reglas que gobiernen la acción tenga una fuerza mayor de 0.4. Dependiendo de la máquina de inferencia, la implementación de cortes Alpha podría reducir el tiempo de ejecución del control difuso(ver Figura B.29).

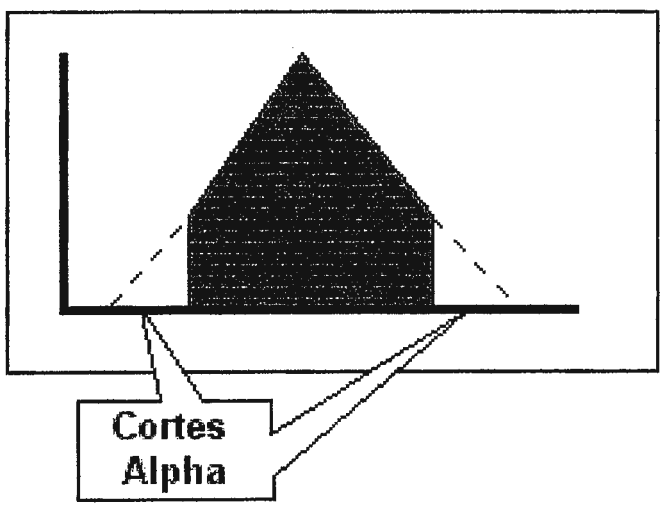


Figura B.29. Cortes Alpha.

El peso de las reglas se utiliza cuando no se desea que todas las reglas tengan igual impacto sobre la superficie de control (salida de control). Es posible que en un control difuso exista una regla particular para manejar condiciones especiales que se activa a menudo y contribuye a la superficie de salida. Para reducir el efecto de esta regla se le puede dar un menor peso que a las otras reglas. Esto limitará el impacto sobre la superficie del control. La contribución de pesos permite también dar extraordinariamente alta prioridad a reglas selectas, como por ejemplo:

- ***IF*** suelo está seco ***THEN*** el tiempo de riego es largo.
- ***IF*** la temperatura es fría ***THEN*** el tiempo de riego es corto.

Si la primera regla es más importante que la segunda, se le puede reducir el peso a esta dándole un valor menor que uno. El peso de la regla es una constante que se asocia a la regla, la cual es igual o menor a la unidad. El peso se multiplica con la fuerza de la regla para aumentar o disminuir la contribución de la regla sobre la acción de salida del control difuso.

El tipo de operadores compensatorios en el proceso de evaluación de reglas impacta en la manera en que la superficie del control es creada. Un posible problema con el operador MIN-MAX es que reglas con muchas proposiciones en los antecedentes lógicamente conectados con el operador AND tiende a llevar el valor de certeza a cero; mientras el operador OR lo lleva a uno. Si este tipo de comportamiento no es aceptable para el diseño del sistema, quizás un operador promedio o uno de suma podría ser el más adecuado. Los operadores compensatorios pueden mezclarse y relacionarse en una base de reglas.

B.4. OBSERVACION DEL COMPORTAMIENTO DEL MODELO, INICIO DE VERIFICACION Y SINTONIZACION.

En este paso se echará andar el modelo sometiéndolo a varias combinaciones de entradas y observando la actividad de las salidas asociadas, tratando de cubrir toda el universo de discurso para cada variable de entrada (ver Figura B.30).

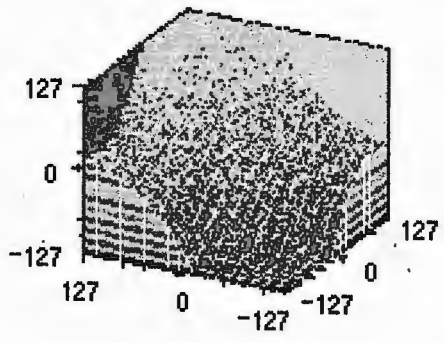


Figura B.30. Observación del comportamiento del modelo.

Las características que se necesitan verificar en un modelo son:

- Valores de salida: son los valores defusificados que el modelo debe de entregar y así poder verificar su precisión.
- Superficie de control: es la representación gráfica de las entradas y la salida del sistema. Si este tiene dos entradas y una salida, el gráfico resultante ocupará el plano tridimensional.
- Tiempo de simulación: es el comportamiento del modelo en el tiempo, donde se debe de poner bastante atención a la estabilidad y sensibilidad.

Basándose en la observación, es posible regresar a cualquier paso previo, la sintonización de las funciones de membresía, reglas ó métodos de fusificación, (ver Figura B.31).

Figura B.31. Sintonización de las funciones de membresía.

B.4.1. REVISION DE RESULTADOS DE SALIDA.

Aquí es necesario poner valores en la entrada para producir valores conocidos a la salida. Con estas pruebas se verifica que el modelo produzca las salidas sensatas. Para cada combinación de entradas hay que verificar lo siguiente:

- Que las salidas tengan sentido. Si esto no sucede, entonces identificar y modificar las reglas y funciones de membresía que produzcan la salida errónea.

- Que las reglas identifiquen correctamente el comportamiento deseado.
- Que las etiquetas asociadas a los valores de entradas tengan la forma y traslape correcto (ver Figura B.32).

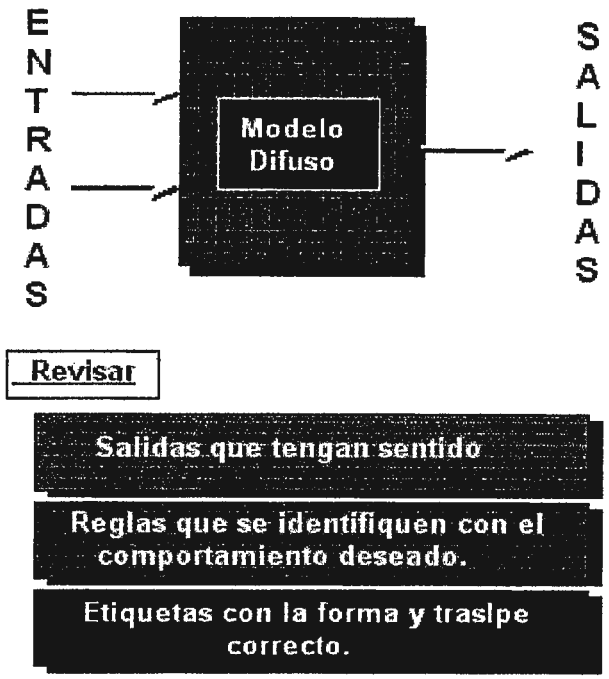


Figura B.32. Revisión de resultados de salida.

Después de examinar las salidas, se prueba exhaustivamente el modelo aplicándole varias combinaciones de entradas, poniendo particular atención a las siguientes regiones:

- Valores de entrada en los extremos del universo de discurso.
- Valores entrada en los extremos del dominio individual de las funciones de membresía.
- Valores de entrada correspondientes al traslape de las funciones de membresía.
- Valores de entrada con algún grado de verdad en tantos antecedentes como sea posible.

Para cada combinación verificar que el modelo pase del sentido común (ver Figura B.33).

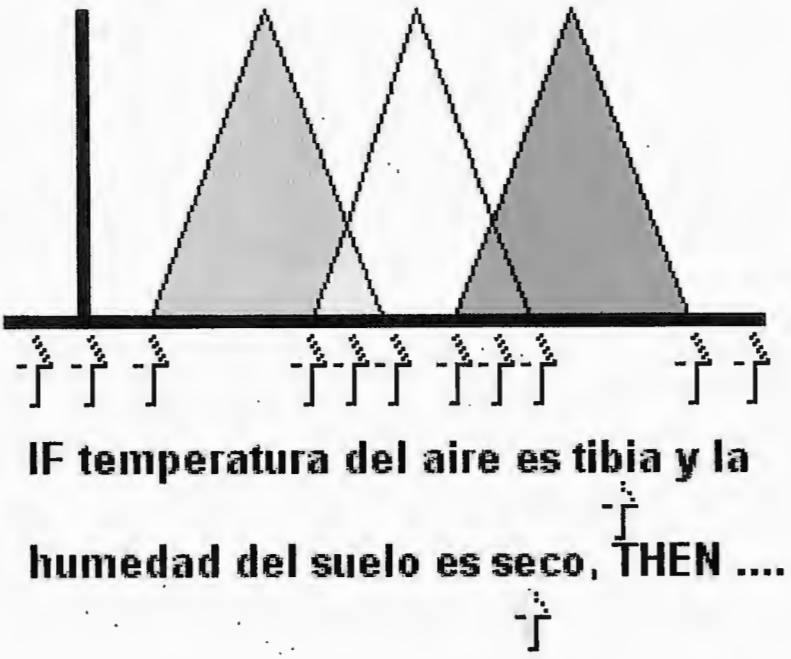


Figura B.33. Verificación del modelo.

B.4.2. REVISION DE LA SUPERFICIE DE CONTROL.

Sobre un gran número de simulaciones se debe de verificar lo siguiente:

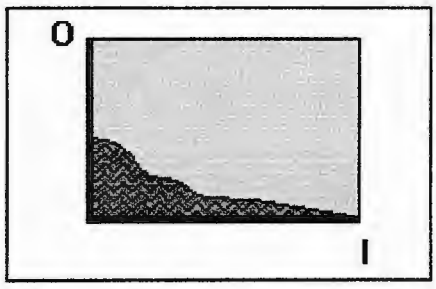
- Que la actividad de las variables de entrada y salida sean robustas y bien distribuidas a través de sus respectivos universos de discurso.
- Que las reglas generen una superficie de control de salida centrando alrededor del 80% del espacio de salida (ver Figura B.30).

Se debe de verificar que la superficie de control cumpla con las expectativas trazadas, si estas no se cumplen es posible que se deban a una de estas tres razones:

- Mala especificación de reglas.
- Funciones de membresía incorrectamente definidas.
- Operadores difusos inapropiados.

Dado que las reglas están correctamente declaradas, se podría efectuar lo siguiente para encontrar la dificultad en el control difuso modelo.

- Asegurarse que el rango de valores de salida este debidamente relacionado con el rango de valores de entrada. Por incompatibilidad entre los conjuntos difusos de entrada y salida se pueden originar problemas (ver Indice de Compatibilidad en el capítulo de tópicos avanzados para mayor información), ver Figura B.34.
- Asegurarse que la información de superficie, (máxima proporción de cambios de entrada versus salidas), no este aislada en los bordes de todas las vistas de las superficies (ver Figura B.35).



Se podría esperar ver u amplio rango de valores de salida por medio de las entadas seleccionadas.

Figura B. 34. Incompatibilidad entre conjuntos de entrada y salida.

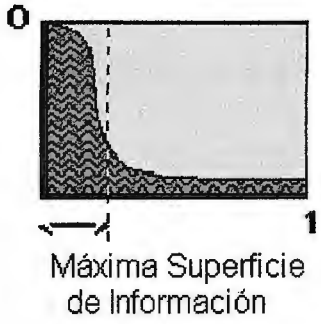


Figura B. 35. Máxima proporción de cambios de entrada versus salidas.

Si estos fueran los casos, se requerirá de:

- Reexaminar y modificar en lo posible el universo de discurso.
- Hacer más alta a densidad de funciones de membresía en el área de máxima superficie de información. Esto tenderá a hacer las pendientes más suaves.
- Verificar si las superficies tienen anomalías inesperadas, una causa común de esto se debe a que las reglas referidas están mal etiquetadas (ver Figura B.36).
- Examinar la compatibilidad de los datos de superficie con las reglas base, (ver Indice de Compatibilidad en el capítulo de tópicos avanzados para mayor información). Examinar la superficie para asegurarse que tiene sentido intuitivo (ver Figura B.37).

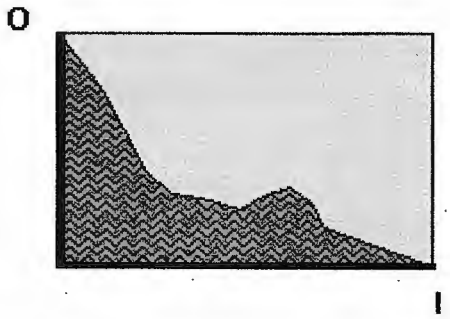


Figura B. 36. Reglas mal etiquetadas.

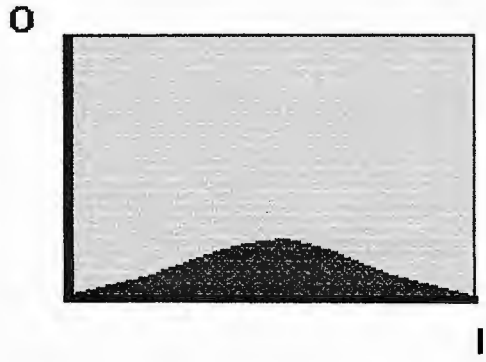


Figura B. 37. Examinamiento de la superficie.

B.5. TÓPICOS AVANZADOS

Estos son una serie de criterios extras que se aplican en un diseño o modelado difuso, los cuales ayudan a optimizar el comportamiento del sistema a controlar.

B.5.1. CORTES ALPHA.

Un corte Alpha es un diseño específico de umbrales el cual controla:

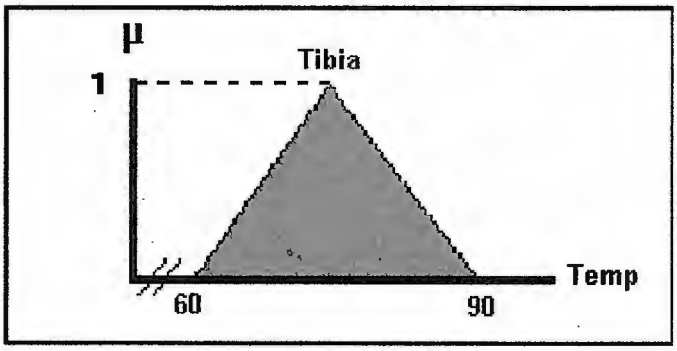
. Como las proposiciones difusas son evaluadas.
. Cuales reglas son permitidas para contribuir en el proceso de inferencia.

Esto quiere decir que se puede ajustar el valor de la función de membresía a un valor mayor que el umbral α perteneciente entre 0 y 1. Los cortes Alpha se dividen en dos niveles:

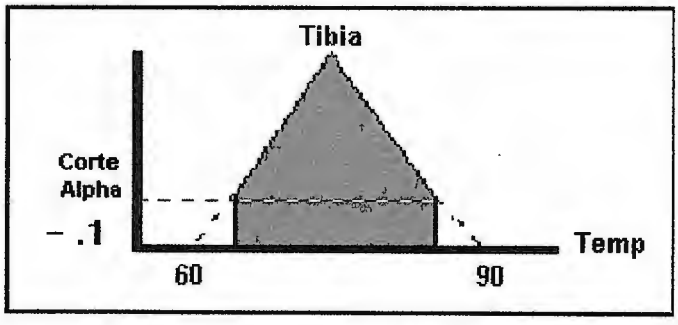
1. Nivel de conjunto difuso.
2. Nivel de regla.

- Cortes Alpha de nivel de conjunto difuso:

Este tipo de corte es aplicado al nivel individual de los conjuntos difusos. Si se toma el siguiente argumento de regla "IF temperatura es tibia AND ...." y se asigna un corte Alpha de 0.1 al conjunto difuso "tibia", esto causará que el valor de verdad de cada proposición que involucre la etiqueta "tibia" tome el valor de cero si el valor de verdad es menor de 0.1 (ver Figura B.38).



Para la función de membresía etiquetada como "Tibia", la entrada crisp puede ser fusificada con cualquier valor entre 0 y 1.



Todos los valores fusificados que esten debajo del corte alpha se les aplicará un grado de pertenencia cero.

Figura B.38. Concepto de Corte Alpha.

El propósito más común del corte Alpha de nivel es el aumento de velocidad en el proceso de inferencia. Si los operadores difusos son limitados al AND de Zadeh para reglas con múltiples antecedentes (reglas compuestas), cualquier proposición será evaluada en la pertenencia del corte Alpha, causando que la fuerza de la regla sea cero (ver Figura B.39).

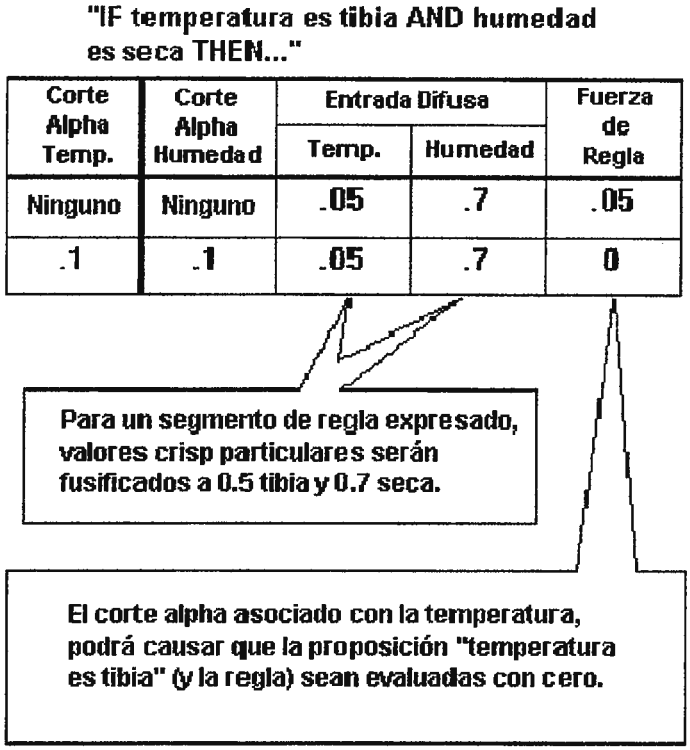


Figura B.39. Efecto del corte Alpha.

- Corte Alpha de nivel de regla:

También conocidos como "amortiguadores de barrera" o "amortiguadores de umbral". Este corte se aplica a todas las reglas. Si el valor de verdad de la supuesta regla cae por debajo del corte Alpha esta no se activará y su fuerza será cero. Con esto se acelera la ejecución del proceso de inferencia si este pudiera ser programado a que obvie reglas con fuerza cero.

El corte Alpha de nivel de regla también sirve como una alarma, el cual activa reglas importantes cuando ciertas condiciones especiales existan. En este caso, a una regla podría asignársele un alto amortiguamiento de umbral. Típicamente evaluaciones de las fuerzas de las reglas moderas no activan este tipo de reglas.

La acción tomada por la regla si se activada puede ser brusca (ejemplo: "Si la regla se activa, entonces apague el sistema"), o puede impactar significativamente la respuesta del sistema si se le ha asignado un alto índices de contribución, o modificar las funciones de membresía con diferentes dominios.

Los cortes Alpha de nivel de conjuntos difusos son los que más se implementan de forma global, mientras que los cortes Alpha de nivel de regla son frecuentemente implementados en sistemas basados en reglas. Una regla podría ser afectada por los dos tipos de cortes Alpha.

Los cortes Alpha son inapropiados para muchas aplicaciones de control usando microcontroladores. Al incluir cortes Alpha podrían crearse "escalones cuánticos" en la respuesta de salida que degradarían el funcionamiento del sistema. Los cortes Alpha son usados cuando se tratan con funciones de membresía con retardos largos, tal retardo tiende a causar que muchas reglas tengan pequeños grados de verdad y esto podría prolongar el tiempo de procesamiento en la inferencia.

B.5.2. AFIRMACIONES.

Si bien las afirmaciones son frecuentemente utilizadas en sistemas predictivos difusos, estas también pueden ser usadas en aplicaciones de control para incorporar al modelo reglas que sólo tenga la parte del consecuente, por ejemplo: "Duración de riego es mínimo" con una función de membresía mostrada en la Figura B.40.

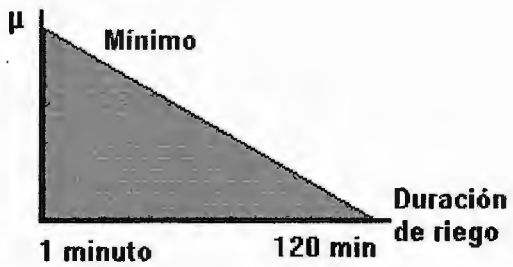


Figura B.40.. Esquema de una afirmación.

Una afirmación impacta en el funcionamiento total del sistema al colocar los límites máximos a lo largo del espacio truncado de la salida. Con el siguiente ejemplo se observará el efecto de las afirmaciones sobre sistemas difusos. La Figura B.41 muestra las funciones de membresía para la salida y en la Figura B.42. puede apreciarse la acumulación después de la evaluación de reglas. Al aplicar la afirmación "Duración de riego es mínimo", el área sombreada es recortada y el centro de gravedad se desplaza (ver Figura B.43).

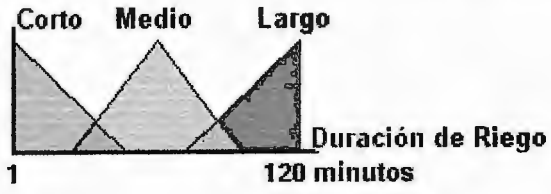


Figura B.41. Funciones de salida.

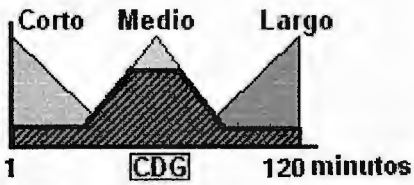


Figura B.42. Salida Truncada.

Figura B.43. Impacto de una afirmación.

Como puede verse en la Figura B.43, la afirmación provoca el desplazamiento del centro de gravedad (CDG) generando una duración más corto del tiempo de riego.

B.5.3. INDICE DE COMPATIBILIDAD.

Este índice mide el grado de compatibilidad entre los valores "crisps" de entrada de un sistema y los valores obtenidos del sistema difuso basado en reglas. El índice de compatibilidad es la máxima certeza o valor de membresía de la región difusa resultante (con las funciones de membresía truncadas) para un conjunto dado de entradas crisp.

Para aclarar este concepto, se brinda el siguiente ejemplo de un modelo de reglas difusas para estimar el peso de una persona:

- IF la altura es alta, THEN el peso es pesado
- IF la cintura es grande, THEN el peso es muy pesado.
- IF los hombros son anchos, THEN el peso es algo pesado.
- IF el tamaño del zapato es por debajo de la mitad, THEN el peso es algo ligero.

Los valores de altura, cintura, hombros y tamaño del zapato definen la

forma final de región difusa resultante (ver Figura B.44).

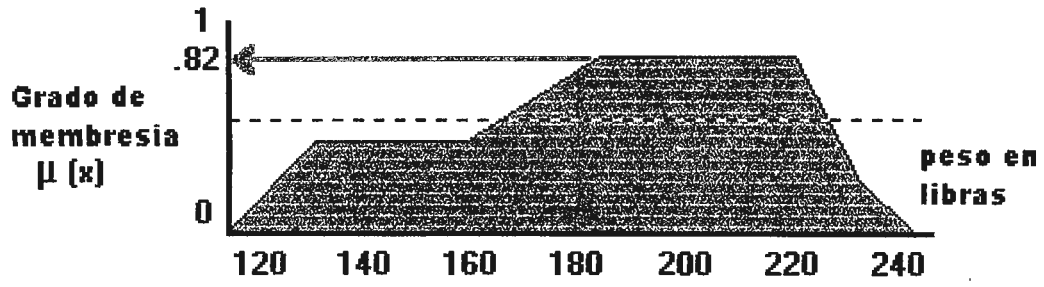


Figura B.44. Función difusa de salida.

En este modelo la región difusa resultante (salida difusa en la etapa de defusificación) tiene un máximo valor de verdad de 0.82, el cual es el índice de compatibilidad. Este índice establece que tan bien encajan los datos con las reglas. Nótese que a través del procedimiento de centro de gravedad se produce un peso de 180 libras.

Ahora si se considera el mismo modelo pero con diferentes datos de entrada, la defusificación podría ser la misma pero el máximo valor de la región difusa resultante es más bajo (ver Figura B.45). Este nuevo índice muestra que los datos son menos compatibles con las reglas y los conjuntos difusos.

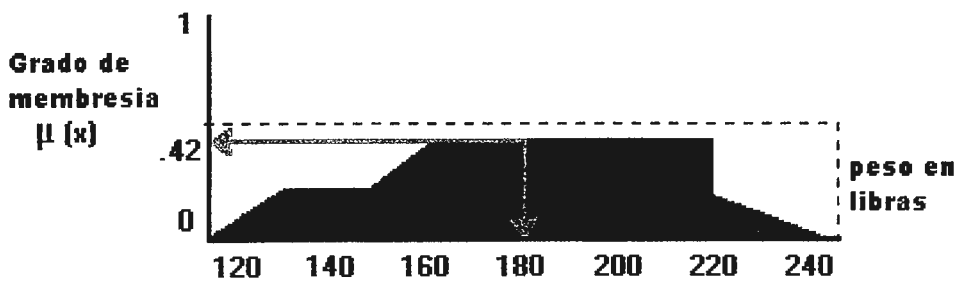


Figura B.45. Función difusa modificada.

Un índice de compatibilidad se utiliza para medir como un modelo responde

durante la simulación y validación para un flujo de datos esperado. Un modelo difuso con buen comportamiento genera un índice promedio de compatibilidad que oscila entre 0.2 y 0.8. Es importante hacer muchas mediciones y obtener un índice de compatibilidad promedio. Una medición aislada de este índice no tiene valor alguno para catalogar el modelo.

Los índices de compatibilidad que son consistentemente muy bajos o muy altos sugiere que los datos y el conjunto de reglas son incompatibles. Algunas de las causas de incompatibilidad son:

- Los datos consistentes quedan en los extremos de los conjuntos difusos.
- Las reglas relacionan inapropiadamente los datos de entrada con los conjuntos difusos.
- Las funciones de membresía son incorrectamente formadas y traslapadas.
- Operadores difusos inapropiados.
- Afirmaciones y reglas difusas han sido mezcladas incorrectamente.

B.5.4. PESOS DE CONTRIBUCION.

No todas las reglas difusas son creadas igual. En lógica difusa tradicional, las reglas son seleccionadas y activadas en base a la compatibilidad entre las reglas y el vector de datos entrante. La contribución de la fuerza de una regla en la región difusa de salida depende únicamente del nivel de compatibilidad. Para priorizar reglas, se le pueden asignar factores de peso que multipliquen a cada regla. Este factor es llamado peso de contribución, dado que ajusta la contribución de la regla en la región difusa de la variable de salida. Este peso es normalmente un intervalo entre 0 y 1. Cuando el peso de contribución se utiliza, el antecedente es multiplicado por el peso antes de que sea procesado por la inferencia difusa.

Cuando el peso toma un valor entre 0 y 1, este se utiliza para reducir la

contribución de la regla. También este factor puede ser mayor que la unidad para obtener un efecto amplificador. Por ejemplo, el rango del peso podría estar entre 0 y 2 para incrementar el impacto de la regla. Cuando el peso es mayor a la unidad, se debe aplicar la siguiente ecuación para no sobrepasar el máximo grado de membresía (100%):

$$\min[\text{peso}*\mu(x),1]$$

A veces se utiliza un peso con valor cero para inhabilitar reglas.

B.5.5. ADAPTABILIDAD.

Un sistema de control adaptivo se ajusta automáticamente:

- Mide la característica dinámica (función de transferencia) de la planta (el objeto físico a ser controlado).
- Compara esta con la característica deseada (modelo).
- Y usa la diferencia para variar los parámetros ajustables del sistema (por ejemplo: factores de escala, reglas difusas, conjuntos difusos y operadores lógicos difusos en un sistema de control difuso).
- Se puede mantener un optimo desempeño sin importar los cambios en el ambiente.

Un sistema de control adaptivo mide constantemente su propio funcionamiento. De acuerdo a un índice de desempeño especificado por el diseñador realiza modificaciones de sus propios parámetros para mantener un funcionamiento óptimo no importando los cambios en el ambiente.

Un control adaptivo usualmente debe de tener las siguientes opciones (ver Figura B.46):

- Identificación de las características dinámicas de la planta, expresadas en términos de algún índice de funcionamiento.
- Tomar decisiones basadas en los índices de funcionamiento.
- Modificar o actuar en base a la decisión tomada.

En un sistema de control difuso adaptivo usualmente se ajustan automáticamente los factores de escala. Sistemas difusos adaptivos más complejos pueden reajustar automáticamente otros parámetros tales como: pesos de contribución, los conjuntos difusos y operadores difusos

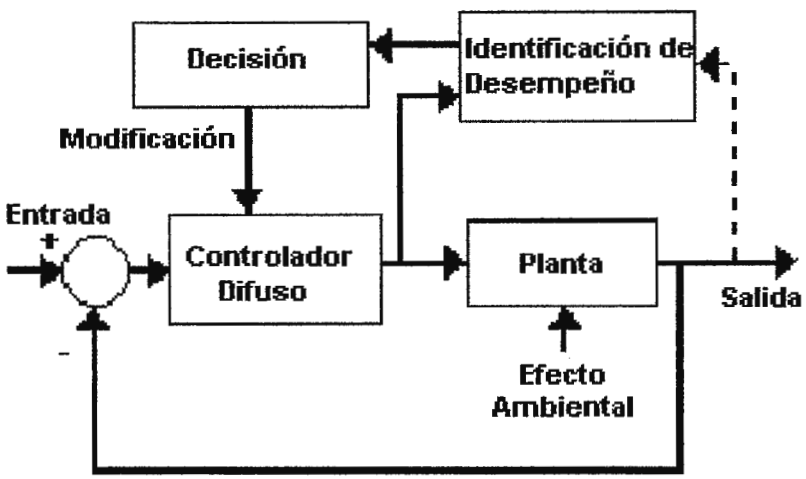


Figura B.46. Control difuso adaptivo.

Entre las ventajas de un sistema de control adaptivo difuso se pueden mencionar:

- Un sistema de este tipo siempre es más robusto que su contraparte no adaptiva.

- El proceso de adaptación es frecuentemente fácil de implementar en un sistema de control difuso por su cualidad natural.
- Los sistemas difusos comparten mucha de la filosofía que utilizan otros sistemas inteligentes como las redes neuronales.

Entre las desventajas de un sistema de control adaptivo difuso se pueden mencionar:

- La estabilidad de los sistemas adaptivos difusos no puede ser comprobada.
- El proceso de adaptación involucra procesamiento intensivo.

En realidad no es posible aplicar algún criterio de estabilidad al sistema de control difuso porque son sistemas no lineales e invariantes en el tiempo. Además, por el momento no hay herramientas matemáticas disponibles[1].

[1] NOTA: Mayor parte de esta información fue obtenida y traducida de [9] y [19].

APENDICE C. CONTROL AUTOMATICO.

En esta sección se hará un breve repaso de la teoría de sistemas de control automático. El objetivo es que el lector obtenga una perspectiva lo suficientemente amplia para que posteriormente se puedan apreciar las ventajas y desventajas comparándose con lógica difusa. No se pretende profundizar en técnicas y métodos de diseño de sistemas de control automático. A lo largo de esta sección encontrará referencias de libros en donde el lector puede consultar y ahondar en los temas.

C.1. SISTEMAS DE CONTROL AUTOMATICO.

Un sistema implica varios componentes trabajando conjuntamente en una forma prescrita para alcanzar una meta específica. Esta meta es el control de cierta cantidad física a través de procesos automáticos que no requieren la supervisión humana.

El ser humano fue por muchos años el encargado directo de controlar las máquinas. Sin embargo al ser las máquinas más dinámicas que estáticas, las mismas limitaciones del ser humano no permitían reacciones inmediatas y oportunas ante perturbaciones. Entonces los ingenieros tuvieron que diseñar y construir dispositivos de control que mejoran el desempeño de los equipos y salvaguardaran al hombre de procesos peligrosos. Aunque el control es un proceso común en la naturaleza, el hacer dispositivos que asimilen el proceso de control ha sido un estudio muy complejo y exhaustivo.

Para poder diseñar los dispositivos controladores se requiere de un análisis matemático del sistema o planta que se desea controlar. Una planta se entiende como el equipo físico que usualmente se considera inalterable y posee algún parámetro a ser controlado. Una planta puede ser tan sencilla como un reostato hasta tan compleja como un transbordador espacial. Una planta tiene entradas y salidas que están interrelacionadas. El control automático pretende mantener la relación entre entradas y salidas contrarrestando cualquier tipo de perturbación interna o externa. Dependiendo del control que se desea, los sistemas de control automático se categorizan en sistemas de control de lazo abierto o de lazo cerrado. La diferencia entre ambas es la existencia de retroalimentación.

C.2. MODELOS MATEMATICOS.

En control clásico y moderno para poder analizar y diseñar dispositivos de control, las plantas deben modelarse a través de ecuaciones matemáticas. Una de las herramientas son las ecuaciones diferenciales que proporcionan una descripción completa de la planta en sentido de que se puede determinar la salida cualesquiera que sean las condiciones iniciales y la entrada.

El conjunto de ecuaciones diferenciales que describen las leyes físicas que determinan el comportamiento de los componentes individuales de la planta es el punto de partida básico. El uso de ecuaciones diferenciales en el dominio del tiempo resulta muchas veces muy complejo. Los problemas se facilitan enormemente, aunque siempre involucra mucho trabajo, si las ecuaciones se trasladan del dominio del tiempo al de la frecuencia a través de las transformadas de Laplace. El análisis en el dominio de la frecuencia en régimen permanente es más versátil que en el dominio del tiempo.

Control clásico sólo puede aplicarse para aquellas plantas que solamente tengan una entrada y una salida. Para análisis de múltiples entradas y salidas se ocupa la teoría de control moderno, la cual se basa en la matemática vectorial y matricial para el análisis de variables de estado.

Para facilitar la representación de plantas físicas en modelos matemáticos se utilizan diagramas de bloques. En esta representación gráfica cada bloque contiene una función matemática que describe la relación entre la señal de entrada y la salida. Cada bloque esta interconectado con otro a través de flechas que indican el flujo de las variables. A partir de lo anterior se han diseñado y perfeccionado varios métodos de análisis matemáticos:

- Función de Transferencia.
- Diagrama de bloques.
- Gráficos de flujo de señal.
- Método del espacio de estado.
- Criterio de estabilidad de Routh.
- Diagrama del lugar de las raíces.
- Diagramas de Bode.
- Criterio de estabilidad de Nyquist.

Sin embargo, no importa que tan bien se haya modelado nominalmente un sistema, siempre habrá dinámicas no modeladas que producen perturbaciones en la planta. El diseñador de sistemas de control tiene la tarea de elegir todos aquellos parámetros intervinientes en la planta que considere importantes en la incidencia de los resultados obtenidos en la salida.

La función de transferencia es un concepto importante en control automático sobre la cual se basa muchos de los análisis matemáticos mencionados anteriormente. La función de transferencia de un sistema es un modelo matemático que permite un método operacional para expresar la ecuación

diferencial que relaciona la variable de salida con la variable de entrada, dando una descripción completa de las características dinámicas del sistema. Sin embargo la aplicación de la función transferencia queda limitada a sistemas de ecuaciones diferenciales lineales invariables en el tiempo.

La función de transferencia no brinda información respecto a la estructura física del sistema. Con la función de transferencia se puede estudiar la reacción del sistema a varios tipos de entradas. A través de la transformada de Laplace, la función de transferencia ofrece un método para conocer los polos y ceros, los cuales sirven de indicadores de los puntos críticos del sistema. Los polos son los valores que toma la variable S en los factores del denominador que hacen al sistema generar una respuesta tendiendo a infinito o a saturación (cualquier ecuación dividida entre cero tiende al infinito). Los ceros son los valores que toma la variable S en los factores de numerador que generan una respuesta cero.

Otro aspecto obligatorio muy importante para modelar sistemas mediante la función de transferencia es la propiedad de linealidad en los sistemas. Es decir, el principio de superposición debe aplicarse. Este principio es válido para regiones de comportamiento lineal de la planta a controlar. La superposición permite analizar la planta parametrizando la función de transferencia de diferente forma, pero siempre el resultado final es la suma de las partes. Este principio no puede aplicarse a plantas que tengan comportamientos no lineales. La mayoría de los sistemas físicos no son lineales, pero se puede restringir la variación de las variables dentro de las regiones de comportamiento lineal. Los controladores que se diseñen serán solamente efectivos en esas regiones.

En la Figura C.1 se muestra una relación no lineal entre variables de entra y salida. Como puede observarse en la gráfica, se ha delimitado una región de la curva que describe un comportamiento lineal, para la cual se podrá diseñar un controlador utilizando herramientas matemáticas como la función de transferencia.

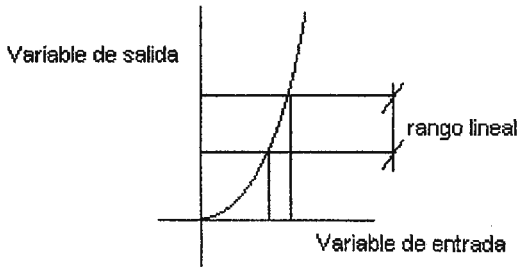


Figura C. 1 Curva de comportamiento para sistemas no lineales.

Los demás métodos de análisis matemáticos pretenden describir y analizar la planta a controlar utilizando ecuaciones. El objetivo del análisis matemático es servir de herramienta para la construcción de controladores eficientes.

C.3. SISTEMAS DE CONTROL DE LAZO ABIERTO.

Los sistemas de control en los que la salida no tiene efecto sobre la acción de control, se denominan sistemas de control de lazo abierto.

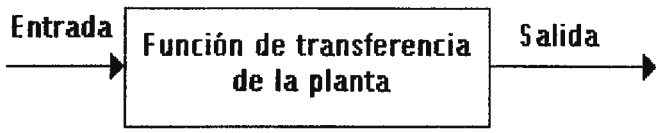


Figura C. 2 Diagrama de bloques de un sistema lazo abierto.

En cualquier sistema de control de lazo abierto no se compara la salida con la entrada de referencia. Por tanto, para cada entrada de referencia corresponde una condición de operación fija. Así, la precisión del sistema depende de la calibración. En presencia de perturbaciones, un sistema de control de lazo abierto no cumple su función asignada. En la práctica el control de lazo abierto se utiliza sólo si la relación entre la entrada y la salida es conocida, y si no existen

perturbaciones. Cualquier sistema de control que funciona sobre una base de tiempos, es un sistema de lazo abierto, un ejemplo son los semáforos.

C.4. SISTEMAS DE CONTROL DE LAZO CERRADO.

Se utiliza indistintamente la denominación control retroalimentado o control de lazo cerrado. La señal de error actuante, que es la diferencia entre la señal de entrada y la de retroalimentación, entra al controlador para reducir el error y llevar la salida del sistema a un valor deseado. El término lazo cerrado implica siempre el uso de la acción de control retroalimentado para reducir el error del sistema.

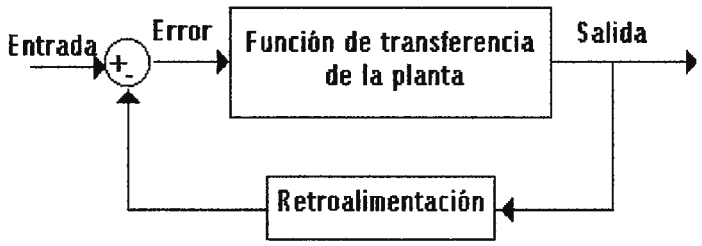


Figura C. 3 Diagrama de bloques de sistema de lazo cerrado.

El lazo de retroalimentación permite obtener un control pero a expensas de la degradación de otros parámetros del sistema, entre los cuales cabe mencionar:

- La reducción de sensibilidad para las variaciones de los parámetros de la planta.
- La reducción de sensibilidad a perturbaciones de salida.
- La capacidad para controlar el ancho de banda del sistema.
- La estabilidad de un sistema estable.
- La capacidad para controlar la respuesta transitoria del sistema.

A partir de la incorporación de un lazo de retroalimentación en un sistema, se han elaborado muchas de las técnicas enumeradas anteriormente para determinar la estabilidad de un sistema. Si la ganancia del lazo de retroalimentación no se ajusta adecuadamente, un sistema estable en sistema de lazo abierto se puede convertir en totalmente inestable en lazo cerrado.

C.5. ESTABILIDAD.

En el estudio de la estabilidad de los sistemas lineales se tratan tres problemas fundamentales. El primer problema, de la estabilidad absoluta, es de naturaleza cualitativa, en el cual se busca una respuesta sencilla "si" o "no" en relación con la estabilidad del sistema. El segundo problema, de la estabilidad relativa, es de naturaleza cuantitativa y está asociado con el problema de determinar qué tan estable es un sistema. En el tercero se examinan las cualidades de robustez con el fin de determinar qué tanto se puede perturbar una planta y aun así conservar la estabilidad. Al referirse a estabilidad, este puede entenderse como:

- Un sistema lineal e invariable con el tiempo es estable si su salida está acotada para cada entrada acotada.
- Un sistema lineal e invaginare con el tiempo es estable si su función ponderada es absolutamente integrable en un intervalo de tiempo infinito, es decir, si $\int_0^\infty |\omega(t)|dt$ es finita.
- Un sistema lineal e invariable con el tiempo es estable si todos los polos de la función de transferencia en lazo cerrado $\frac{Y(s)}{R(s)}$ se encuentran en el semiplano izquierdo del plano s.

La estabilidad de un sistema lineal de lazo cerrado se puede determinar por la ubicación de los polos de lazo cerrado en el plano S. Si cualquiera de esos polos queda en el semiplano derecho del plano S, al transcurrir el tiempo dan lugar al modo dominante y la respuesta transitoria aumenta en forma monótona u oscila con amplitud creciente, esto representa un sistema inestable. Para tal sistema, al momento de conectar la energía, la salida comienza a aumentar al transcurrir el tiempo. Si no se produce saturación en el sistema, y no se presenta alguna detención mecánica, el sistema puede sufrir daños y fallas, pues la respuesta de un sistema real no puede aumentar indefinidamente. Por lo tanto, no se admiten los polos en el semiplano derecho de S, en los sistemas de control lineales. Si todos los polos de lazo cerrado quedan a la izquierda del eje $j\omega$, cualquier respuesta transitoria alcanza el equilibrio, esto representa un sistema estable.

Que un sistema lineal sea estable o inestable es una propiedad del sistema en sí, y no depende de la entrada o función excitadora del sistema. Los polos de la entrada, o función excitadora, no afectan la propiedad de estabilidad del sistema, y sólo contribuyen a los términos de respuesta en estado estacionario de la solución. Así, el problema de estabilidad absoluta puede resolverse fácilmente si no se colocan polos de lazo cerrado en el semiplano derecho, ni sobre el eje $j\omega$. Matemáticamente, los polos de lazo cerrado sobre el eje $j\omega$ producen oscilaciones, cuya amplitud no aumenta ni disminuye con el tiempo. En los casos prácticos, la amplitud de las oscilaciones puede aumentar a una velocidad determinada por el nivel de potencia del ruido. Por lo tanto, un sistema de control no debería tener polos de lazo cerrado sobre el eje $j\omega$.

Como la estabilidad relativa y el comportamiento transitorio de un sistema de control de lazo cerrado están directamente relacionados con la configuración de polos y ceros de lazo cerrado en el plano S, se suele ajustar uno o más parámetros del sistema para obtener configuraciones adecuadas.

C.6. ERRORES EN ESTADO PERMANENTE.

Los sistemas de control automático deben responder adecuadamente a los estímulos que recibe en la entrada. Como pudo verse en los sistemas de segundo orden, existe una región en donde la salida varia hasta llegar a un estado estable en la cual se mantendrá mientras no haya otro estímulo. En muchos sistemas la salida tiene un margen de error con respecto al punto de ajuste establecido. Este error afectará en la precisión del sistema, siendo esto una consideración a tomar en la tolerancia.[7]

Para conocer el porcentaje de error de los sistemas de control automático, estos se someten a estímulos característicos, tales como: escalón unitario, rampa y parabólico. Si se posee la ecuación del sistema, el teorema de valor final de la transformada de Laplace sirve de instrumento para investigar la operación en el estado permanente. Este teorema plantea que si la señal e(t) tiende hacia un valor constante, el valor en estado permanente se puede calcular mediante la ecuación:

$$e_{ss} = \lim_{t\to\infty} e(t) = \lim_{s\to 0} sE(s) \qquad \text{Ec. C. 1}$$

La Tabla C.1 resume las magnitudes de los errores en estado permanente para tres estímulos diferentes. El tipo del sistema y la forma de la señal de entrada influyen en la magnitud de la señal de error en estado permanente. Como lo demuestra la Tabla C.1, la magnitud del error es inversamente proporcional a la ganancia K. Al aumentar la ganancia se disminuye la magnitud del error en estado permanente.

Tabla	Error en estado estacionario en términos de la ganancia K		
	Entrada escalón $r(t) = 1$	Entrada rampa $r(t) = t$	Entrada aceleración $r(t) = \frac{1}{2}t^2$
Sistema tipo 0	$\frac{1}{1+K}$	∞	∞
Sistema tipo 1	0	$\frac{1}{K}$	∞
Sistema tipo 2	0	0	$\frac{1}{K}$

Tabla C. 1 Cuadro resumen de errores en estado permanente para sistemas de cero, primer y segundo orden.[8]

C.7. SISTEMAS DE PRIMER ORDEN.

Al modelar matemáticamente las plantas, estas pueden categorizarse a través de sus respectivas ecuaciones. Al estudiar las plantas utilizando la transformada de Laplace, el exponente de la variable S es un indicador útil para clasificarlas. Para conocer el grado de la función de transferencia se simplifica la ecuación aplicando el límite con la variable S tendiendo a infinito. La función de transferencia final permite definir las plantas como sistemas de primer, segundo u orden superior.

Las plantas que se clasifican como sistemas de primer orden tienen el siguiente tipo de ecuación diferencial característica simplificada:

$$\frac{dy(t)}{dt} + \frac{1}{\Im} y(t) = \frac{A}{\Im} r(t) \qquad \text{Ec. C. 2}$$

Al aplicar la transformada de Laplace, la función de transferencia es:

$$\frac{Y(s)}{R(s)} = \frac{\frac{A}{\Im}}{s + \frac{1}{\Im}} = \frac{A}{\Im s + 1} \qquad \text{Ec. C. 3}$$

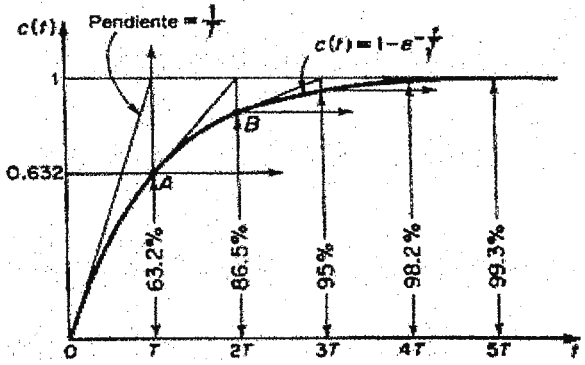


Figura C. 4 Gráfica de la variable de salida de un sistema de primer orden sometido a un escalón unitario.[8]

El parámetro $\Im$ es la constante de tiempo del sistema. Este parámetro es propio de los sistemas de primer orden y proporciona la información referente a la velocidad de respuesta del sistema. El otro parámetro, *A*, se le conoce como parámetro de ganancia en cd del sistema y establece el valor final al que se aproxima la salida en la respuesta al escalón unitario.

C.8. SISTEMAS DE SEGUNDO ORDEN.

Para que las plantas se categorizen como sistemas de segundo orden, la ecuación diferencial característica simplificada debe ser:

$$\frac{d^2 y(t)}{dt} + 2\xi\omega_n \frac{dy(t)}{dt} + \omega_n{}^2 y(t) = \omega_n{}^2 r(t) \qquad \text{Ec. C. 4}$$

Al aplicar Laplace, la ecuación de transferencia se transforma en:

$$\frac{Y(s)}{R(s)} = \frac{\omega_n^{\,2}}{s^2 + 2\xi\omega_n s + \omega_n^{\,2}} \qquad \text{Ec. C. 5}$$

El comportamiento dinámico del sistema de segundo orden se puede describir en términos de dos parámetros: la frecuencia natural no amortiguada (ω_n) y el factor de amortiguamiento (ξ). El primer parámetro establece el tipo de oscilación que se presentará en el sistema y la duración de este, y ξ la relación entre el amortiguamiento efectivo y el amortiguamiento crítico. En base ξ el comportamiento de la respuesta transitoria de un sistema de segundo orden puede clasificarse en:

- Subamortiguado ($0<\xi<1$): La respuesta transitoria es oscilatoria.
- Amortiguado ($\xi=1$) : La respuesta es una curva ascendente que alcanza el valor estable de forma suave y paulatina.
- Sobreamortiguado ($\xi,>1$) : La respuesta tarda mucho en llegar a su estado final. La curva es similar al del caso dos pero con una reacción del sistema muy lenta.

El manejo de ξ permite modificar el comportamiento de los sistemas de segundo orden para lograr acciones de control determinadas. Si se desean sistemas que alcancen de forma rápida el estado final, entonces se aplica el primer caso pero se encuentra la desventaja de oscilaciones y sobre impulsos. El segundo caso se aplica en aquellas situaciones en que se quiere obtener el estado final sin alguna oscilación. El tercer caso es aplicable en aquellos sistemas en que se desea dilatar el tiempo para alcanzar el estado final.

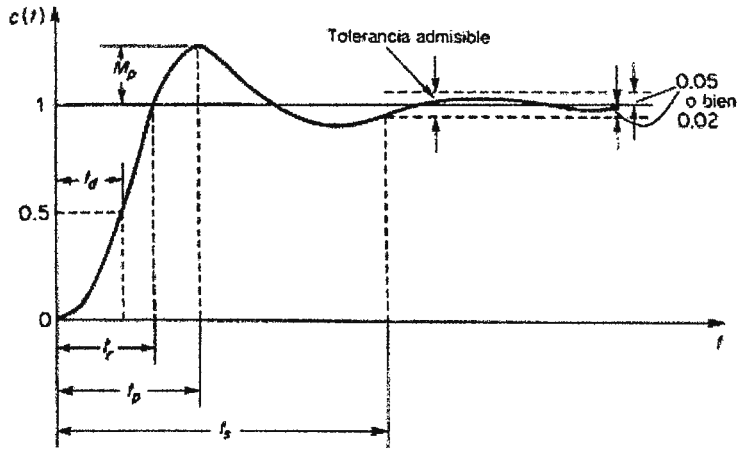


Figura C. 5 Gráfica de variable de salida para un sistema de segundo orden sometido a una entrada escalón unitario.[8]

C.9. CONTROLADORES AUTOMATICOS.

Un controlador automático compara el valor real de la salida de una planta con la entrada de referencia (valor deseado), determina el error, y produce una señal de control que reducirá el error a cero, o a un valor muy pequeño. La forma como el controlador automático produce la señal de control, se denomina acción de control. Los controladores industriales analógicos se pueden clasificar de acuerdo con sus acciones de control, de la siguiente forma:

- Controladores de dos posiciones, o intermitentes.

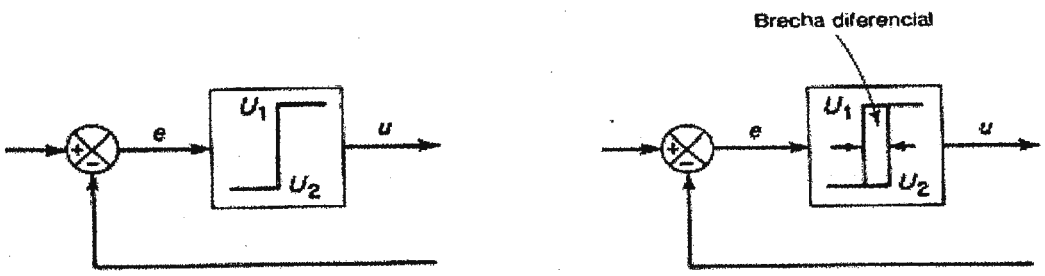


Figura C. 6 Diagrama de bloques de controladores de dos posiciones: On-Off y On-Off con brecha diferencial.[8]

- Controladores proporcionales.

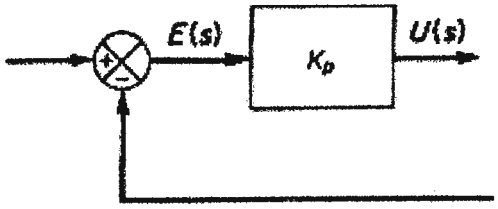


Figura C. 7 Diagrama de bloques de controlador proporcional.[8]

$$u(t) = Kpe(t) \qquad \text{Ec. C. 6}$$

- Controladores integrales.

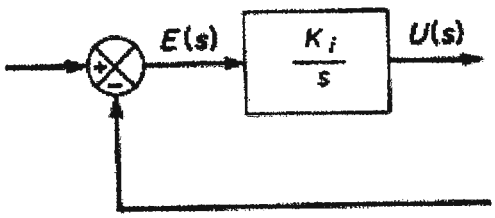


Figura C. 8 Diagrama de bloques de controlador integral.[8]

$$u(t) = Ki\int e(t)dt \qquad \text{Ec. C. 7}$$

- Controladores proporcional-integral.

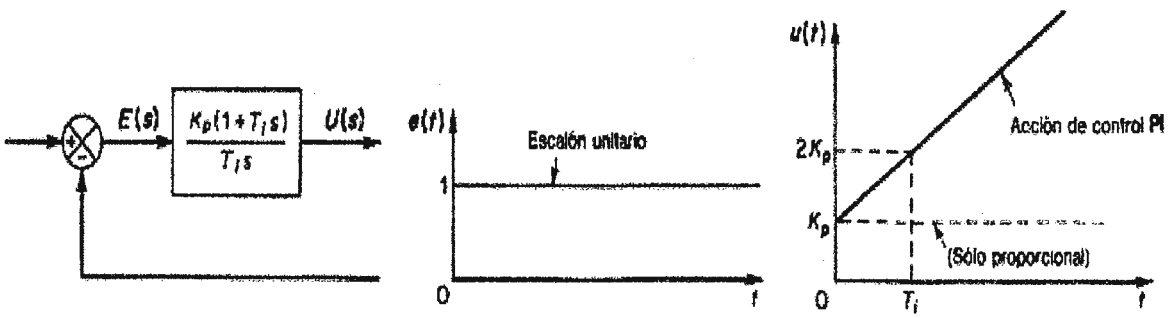


Figura C. 9 Diagrama de bloques de un controlador proporcional-integral con una entrada escalón unitario y su salida.[8]

$$u(t) = Kpe(t) + \frac{Kp}{Ti}\int e(t)dt \qquad \text{Ec. C. 8}$$

- Controladores tipo proporcional-derivativo.

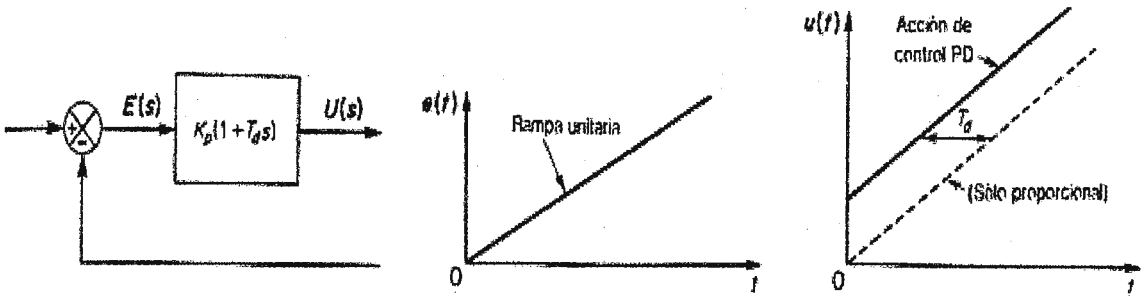


Figura C. 10 Diagrama de bloques de un controlador proporcional-derivativo con una entrada rampa unitaria y su salida.[8]

$$u(t) = Kpe(t) + KdTd\frac{de(t)}{dt} \qquad \text{Ec. C. 9}$$

- Controladores tipo proporcional-integral-derivativo.

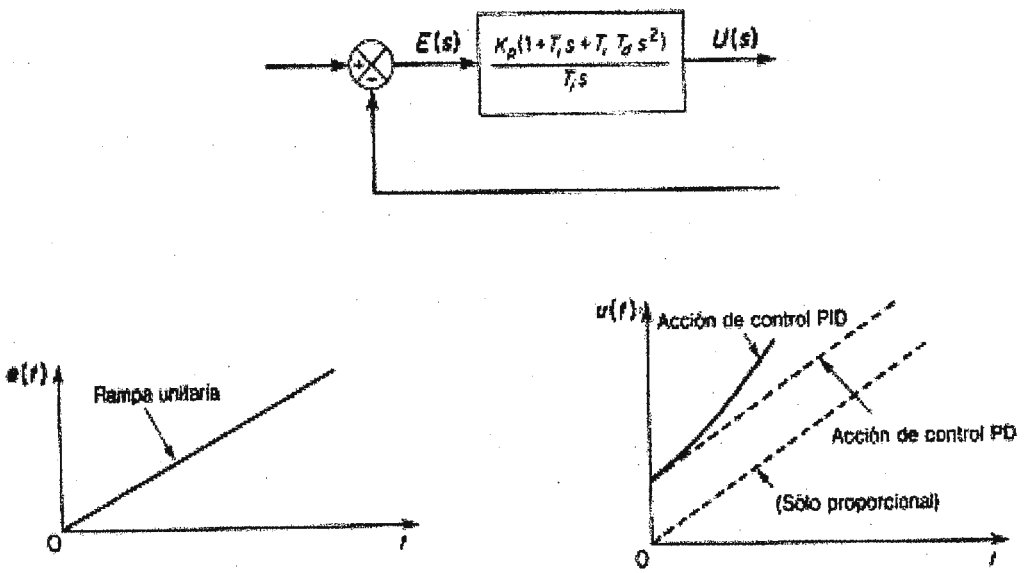


Figura C. 11 Controlador Proporcional-Integral-Derivativo con una entrada rampa unitaria y su salida.[8]

$$u(t) = Kpe(t) + \frac{Kp}{Ti}\int e(t)dt + KpTd\frac{de(t)}{dt} \qquad \text{Ec. C. 10}$$

La mayoría de los controladores analógicos industriales utilizan electricidad o algún fluido, como aceite o aire a presión, a modo de fuentes de potencia. Los controladores analógicos también se pueden clasificar según el tipo de potencia que utilizan en su operación, como neumáticos, hidráulicos o electrónicos. La clase de controlador a usar se decidirá en base a la naturaleza de la planta y las condiciones de operación, incluyendo consideraciones tales como seguridad, costo, disponibilidad, confiabilidad, exactitud, peso y tamaño.

En la Figura C.12 se puede apreciar una planta industrial con los sensores, actuadores y controlador automático. El controlador detecta la señal de error, que suele estar a un nivel de potencia muy bajo, y la amplifica a un nivel suficientemente alto(así, el controlador automático está constituido por un detector de error y un amplificador). También suele haber un circuito de retroalimentación adecuado, junto con un amplificador, que se utilizan para alterar la señal de error, amplificándola, y a veces diferenciándola y/o integrándola, para producir una mejor señal de control. El actuador es un dispositivo de potencia que produce la entrada a la planta, de acuerdo con la señal de control, de modo que la señal de retroalimentación corresponda a la señal de entrada de referencia. La salida de un controlador automático alimenta a un actuador o accionador, que bien pueden ser un motor o una válvula neumática, un motor hidráulico o uno eléctrico.

El sensor o elemento de medición es un dispositivo que convierte la variable de salida, como un desplazamiento, presión, o voltaje, en otra variable adecuada que se utilizan para comparar la salida con la señal de entrada de referencia. Este elemento es el camino de retroalimentación en el sistema de lazo cerrado. El punto de ajuste del controlador debe convertirse en una entrada de referencia con

las mismas unidades que la señal de retroalimentación del sensor o el elemento de medición[1].

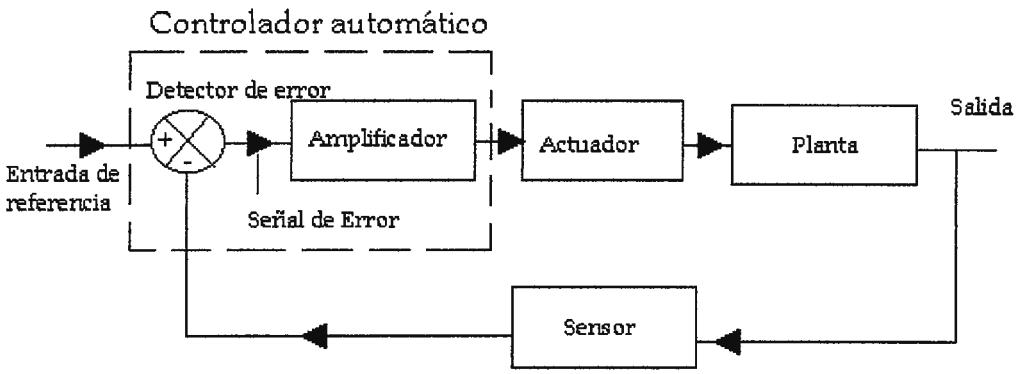


Figura C. 12 Diagrama de bloques de un controlador automático.

[1] El Apéndice 1 es una recopilación hecha de[7] y [8].

APENDICE D. PLANTAS DEL LABORATORIO DE AUTOMATIZACION DEL CITT.

En este apéndice se brinda una breve explicación de las plantas del laboratorio de CITT que se utilizaron para la comprobación del funcionamiento del controlador difuso.

D.1. PLANTA DE TEMPERATURA.

Esta planta permite el estudio de controles automáticos para gobernar la magnitud física de temperatura. La planta de temperatura básicamente es un pequeño horno. El elemento térmico de la planta es una resistencia alojada dentro de una placa de aluminio. Para minimizar los efectos ambientales, la placa se encuentra alojada dentro de una cámara térmica la cual posee un ventilador que dinamiza el proceso de enfriamiento (ver Figura D.1). La placa de aluminio esta provista de enchufes para la conexión de tres tipos de transductores de temperatura comúnmente utilizados en la industria, tales como: PTC, termoresistencia y termopar. También en la placa se encuentra el alojamiento para el termómetro de mercurio con el cual se mide la temperatura de la resistencia. La planta consume hasta 100 watts de potencia para calentar la resistencia hasta 250ºC.

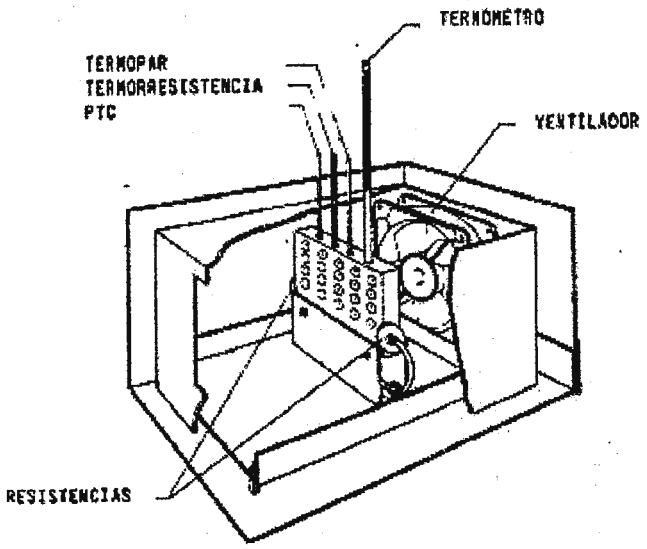


Figura D. 1 Planta de temperatura.

D.1.1. PROCESO DE CONTROL PARA PLANTA DE TEMPERATURA.

El proceso de control de temperatura se muestra en el esquema de bloques de la Figura D.2, sus elementos fundamentales son:

- El amplificador de potencia para proveer un factor de ganancia a la señal de salida del lazo de control que gobierna el actuador de la resistencia.
- El "set-point" genera una señal equivalente a la temperatura de referencia fijado por el usuario.
- El transductor de temperatura (uno de los tres disponibles) genera una señal que se acopla a un acondicionador de señal para compararlo con la señal del "set-point".

- El controlador PID manipula la señal de error (ε) para mantener la temperatura fijada por el usuario a través de "set-point".

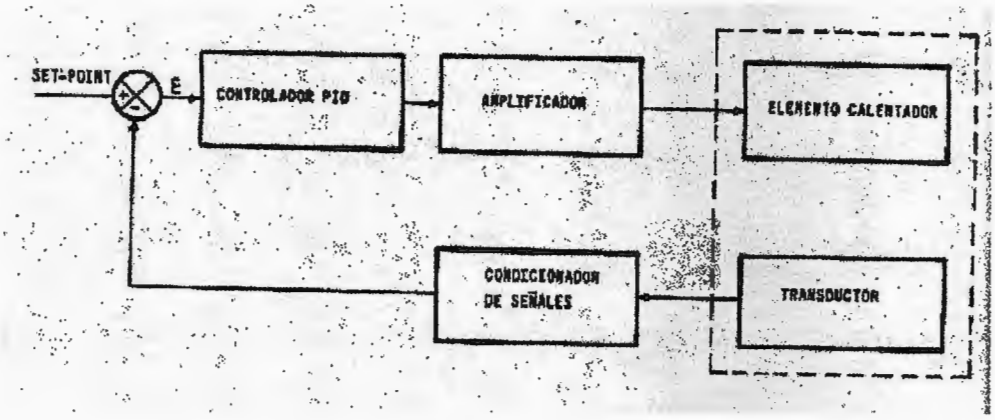


Figura D. 2 Diagrama de bloques de la planta.

D.2. PLANTA DE VELOCIDAD Y POSICION.

Esta planta permite estudiar de modo experimental las técnicas de control automático para gobernar las magnitudes de velocidad y posición. La planta cuenta con un motor de corriente continua de imanes permanentes. En un extremo del eje del motor esta sujeto un dínamo taquimétrico (disco con sectores opacos ubicado entre un conjunto de fototransmisor-fotoreceptor). En el otro extremo del eje se encuentra un motorreductor, un potenciometro que sirve de sensor y un disco giratorio que muestra el desplazamiento del eje. En la Figura D.3 se ilustra la unidad del motor de corriente continua.

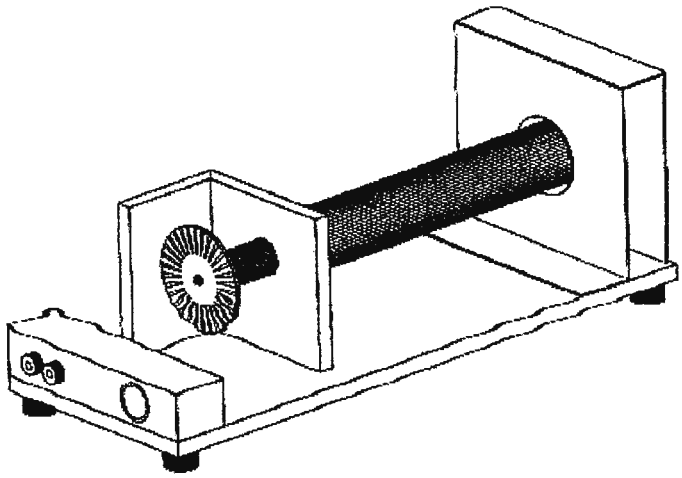


Figura D. 3 Planta de posicionamiento.

El motorreductor aplica un factor de reducción de 50; esto significa que cada rotación del eje por el lado del motorreductor equivale a cincuenta giros completos por el lado de la dínamo. El eje del motorreductor acciona el disco giratorio que posee una aguja roja la cual indica el desplazamiento sobre una escala graduada. La posición del eje del motorreductor se transmite al potenciómetro (que es al transductor de posición) por medio de dos ruedas dentadas cuya relación es 1:2. El potenciómetro tiene la característica de poder girar continuamente sin estropearse, permitiendo realizar las pruebas de velocidad.

La unidad posee un freno mecánico (accionable por medio del manubrio lateral) que sirva para generar una perturbación de carga al motor. El margen de variación del proceso de velocidad angular es de –4000 a 4000 r.p.m. y el del proceso de posición angular es de 0º a 360º.

D.2.1. PROCESO DE CONTROL PARA PLANTA DE VELOCIDAD.

El sistema de control de la planta de velocidad se halla esquematizado en la Figura D.4. Los elementos fundamentales que constituyen el sistema de control son:

- El "set-point" genera una señal equivalente a la velocidad de referencia fijado por el usuario.
- El controlador PID para manipular la ganancia proporcional, integral y derivadora.
- El amplificador de potencia para suministrar la potencia eléctrica al motor eléctrico.
- El transductor de velocidad es un dínamo taquimétrico que se conecta a un acondicionador de señales (respectivamente el "TACHO-GEN CONDITIONER" y el "SPEED DETECTOR") para poder comparla a la señal del "set-point".
- El amplificador de error obtiene la diferencia entre el "set-point" y la señal retroalimentada a través del transductor de velocidad y aplica un facto de ganancia fija (el módulo no permite variarla).

Además de estos bloques indispensables en el control, se halla montado otro bloque denominado "CURRENT LIMIT" que protege al motor limitando la corriente máxima a través de este.

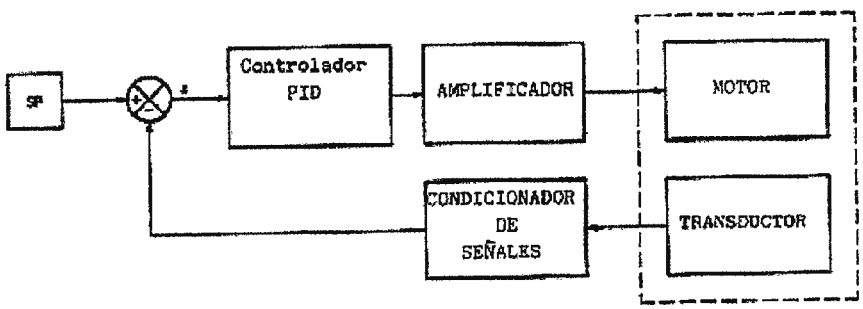


Figura D. 4 Diagrama de bloques de planta de velocidad.

D.2.2. PROCESO DE CONTROL PARA PLANTA DE POSICION.

En la siguiente Figura D.5 se halla esquematizado el control para la planta de posición. Además de los bloques vistos anteriormente (los utilizados por el control de velocidad) este sistema consta de los siguientes bloques adicionales:

- Un segundo amplificador de error con la misma función del anterior amplificador descrito para el proceso de control de la planta de velocidad.
- El transductor de posición es un potenciometro que proporciona una señal de voltaje proporcional al desplazamiento angular.

D.3. PLANTA DE NIVEL Y CAUDAL.

Esta planta constituye un equipo didáctico con el que se experimenta técnicas de control automático para las magnitudes físicas de caudal y nivel. La unidad incluye dos transductores para medir cada una de las magnitudes. Además consta de dos tanques (ver Figura D.6). El tanque inferior sirve para depositar y almacenar el líquido. El control se aplica al tanque superior en donde se pretende mantener el nivel del líquido fijado por un "set-point". Una bomba se encarga trasladar el líquido del tanque inferior al superior. El tanque superior incluye una válvula de fuga manual con la cual se evacua el líquido al tanque inferior.

Para medir niveles o presiones, se utiliza el sensor de presión situado en la parte inferior de la columna vertical del módulo (tanque superior). Este sensor es un transductor extensiométrico que mide la presión ejercida por la columna de líquido acumulado para determinar el nivel (ver Figura D.7). Para medir el caudal se ocupa el medidor de paletas o de turbina, el cual brinda la información sobre el

caudal que la bomba produce (ver Figura D.8). Dependiendo del nivel que se quiera regular en el tanque superior, se incrementa y decrementa el caudal producido por la bomba. La bomba actúa según la información que le brindan los dos sensores (para las pruebas realizadas con el controlador difuso solamente se utilizó el sensor de nivel).

El proceso de control para la planta de nivel y caudal lo conforman bloques con funciones ya explicados anteriormente. Por lo tanto solamente se brinda el esquema mostrado en la Figura D.5.

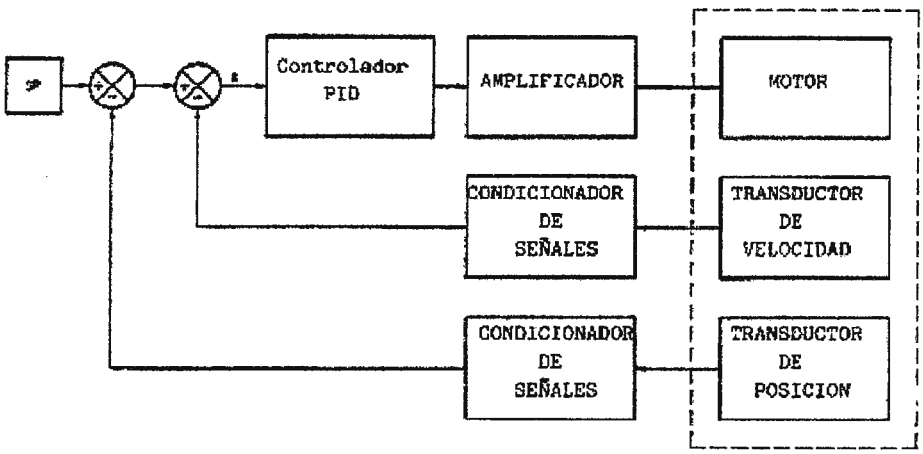


Figura D. 5 Diagrama de bloques para la planta de control de posición.

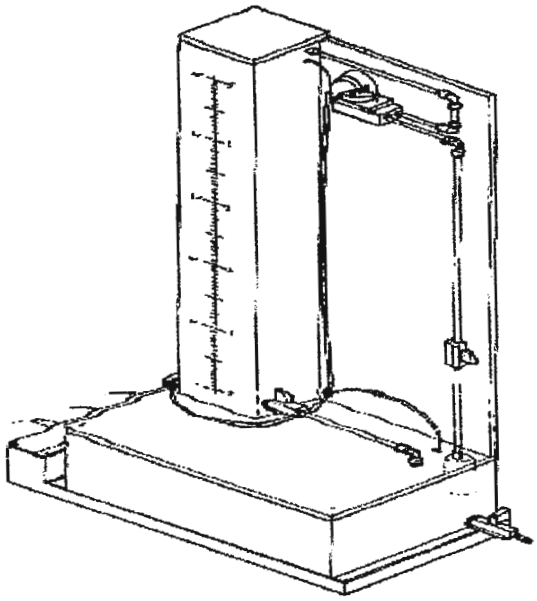


Figura D. 6 Planta de control de nivel.

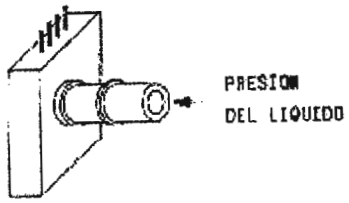


Figura D. 7 Transductor extensiométrico.

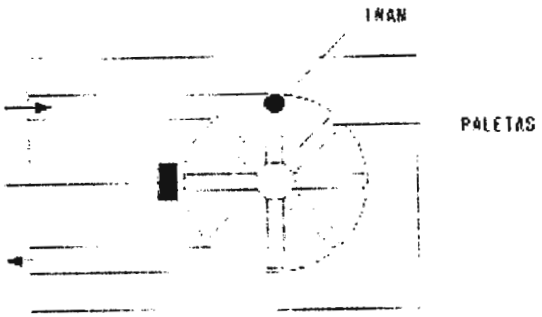
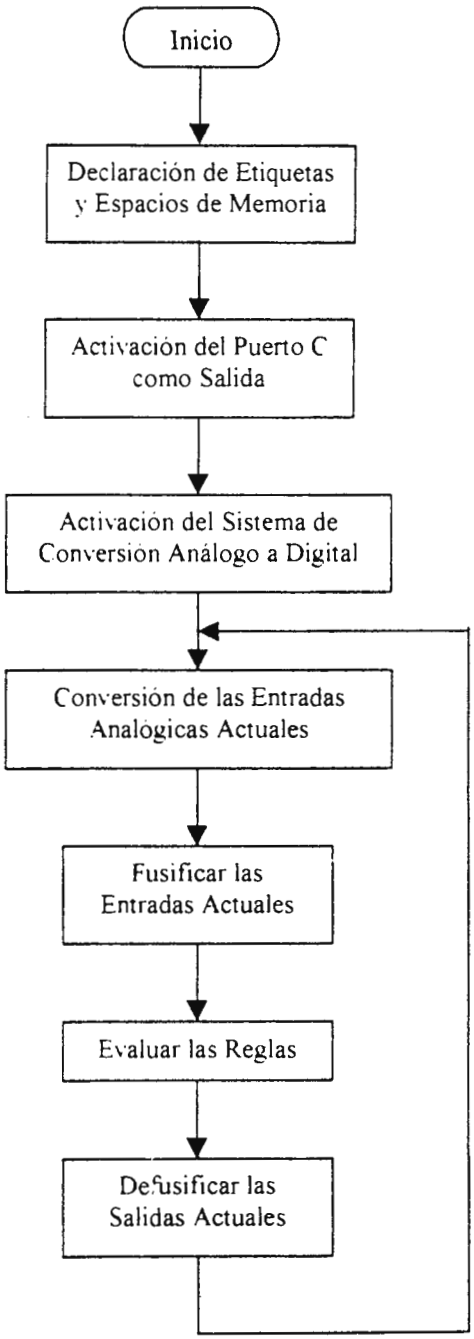
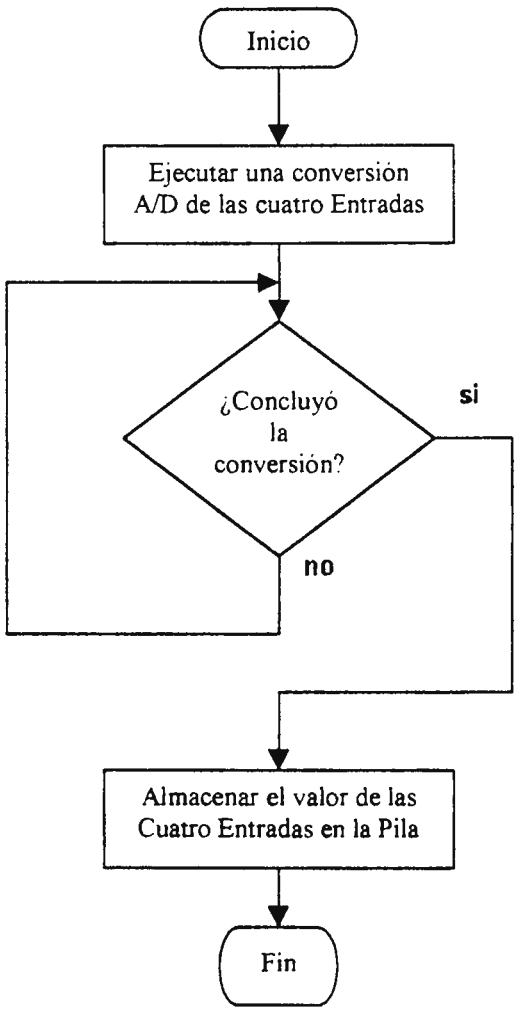


Figura D. 8 Medidor de caudal de paletas.

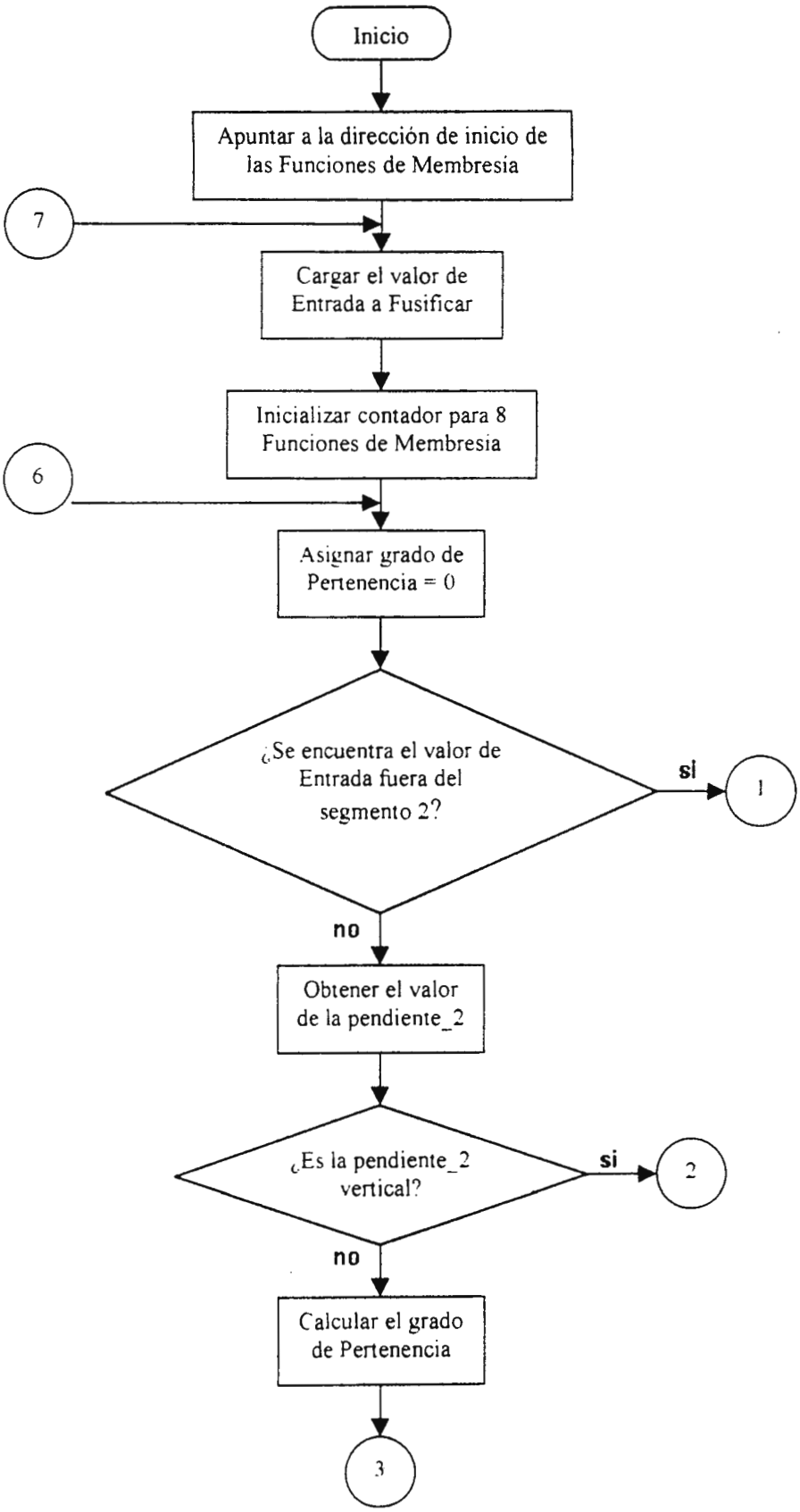
APENDICE E. FLUJOGRAMA PRINCIPAL DEL PROGRAMA DE INFERENCIA DIFUSA.

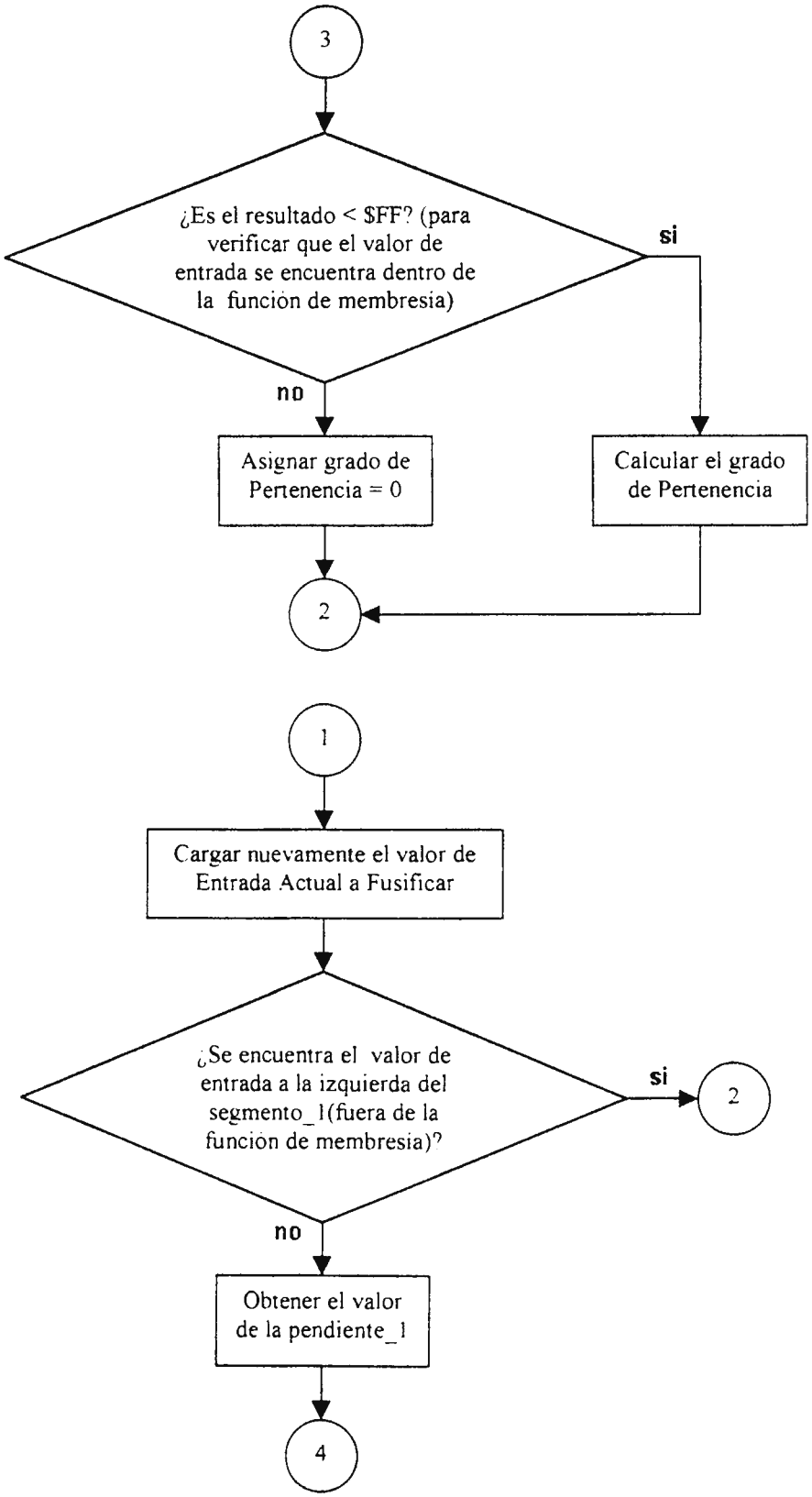


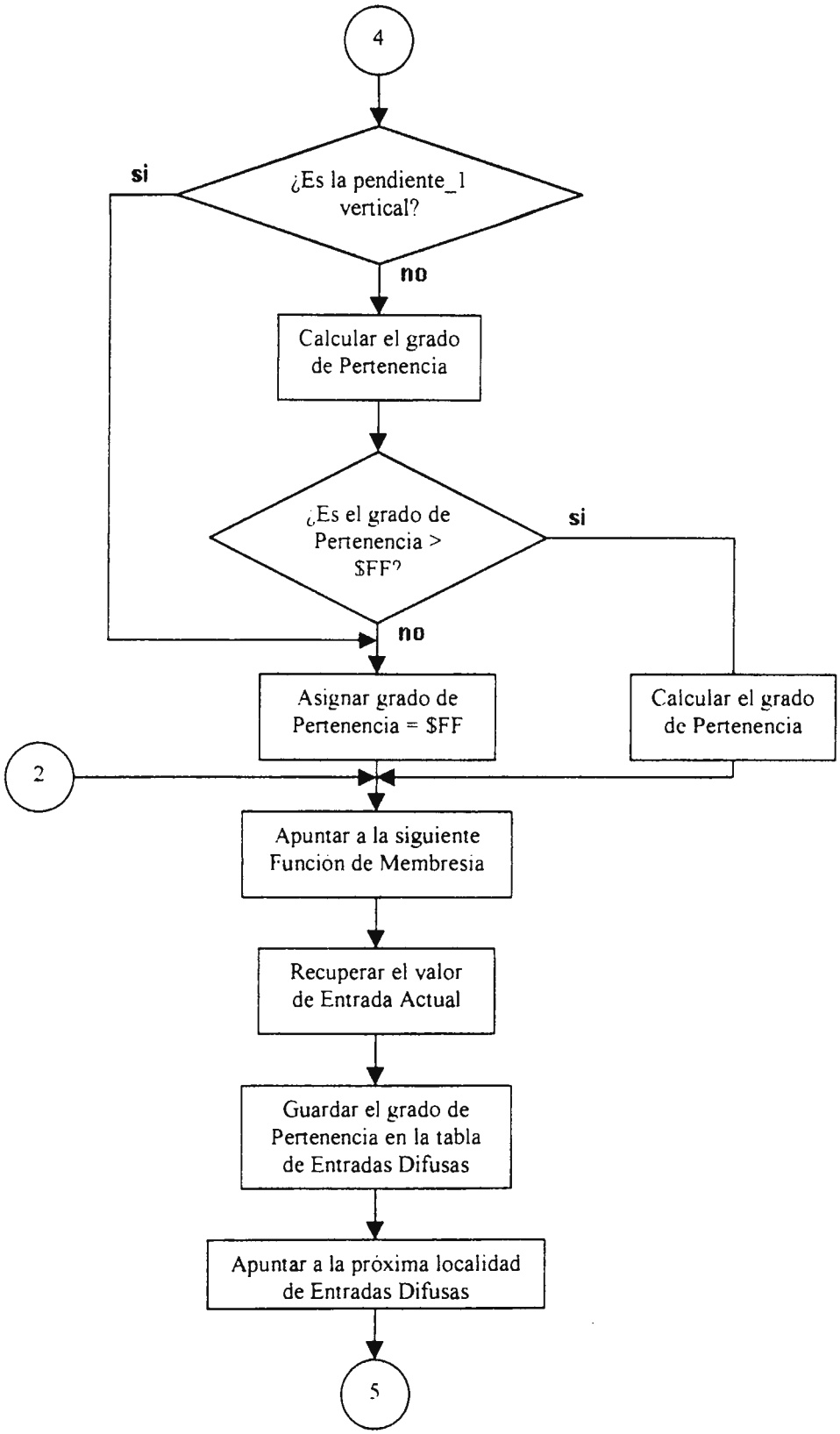
E.1. FLUJOGRAMA PARA LA CONVERSIÓN DE ENTRADAS ANALÓGICAS.

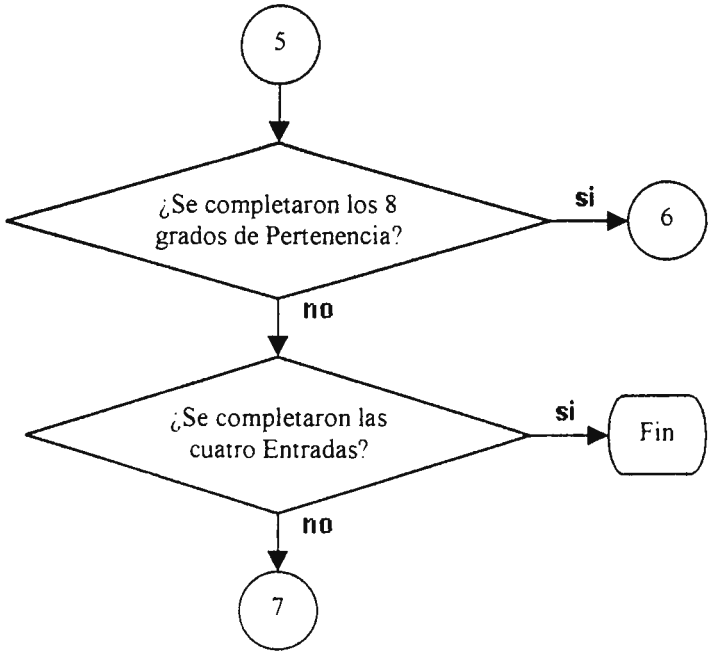


E.2. FLUJOGRAMA PARA LA FUSIFICACIÓN.

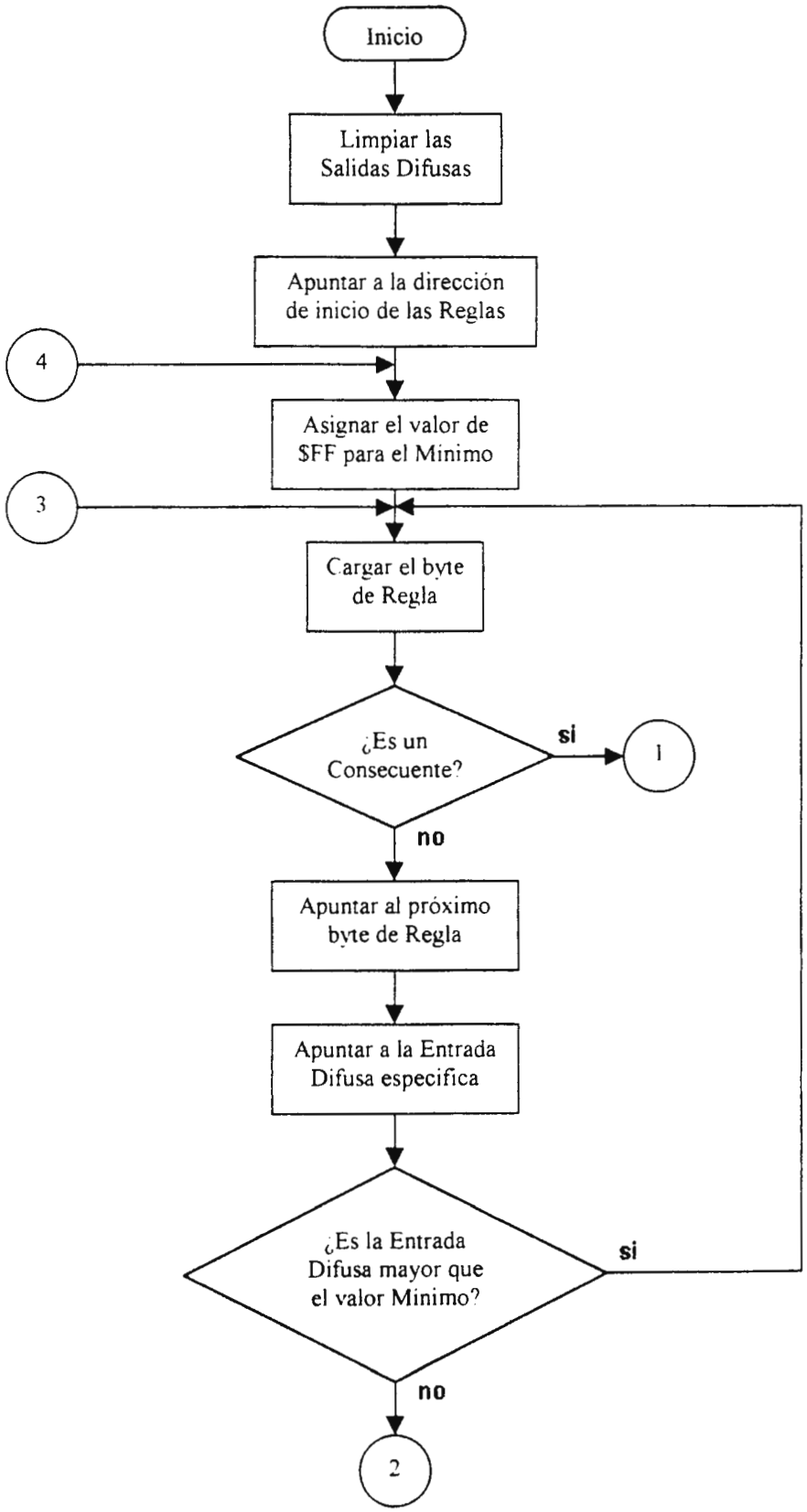


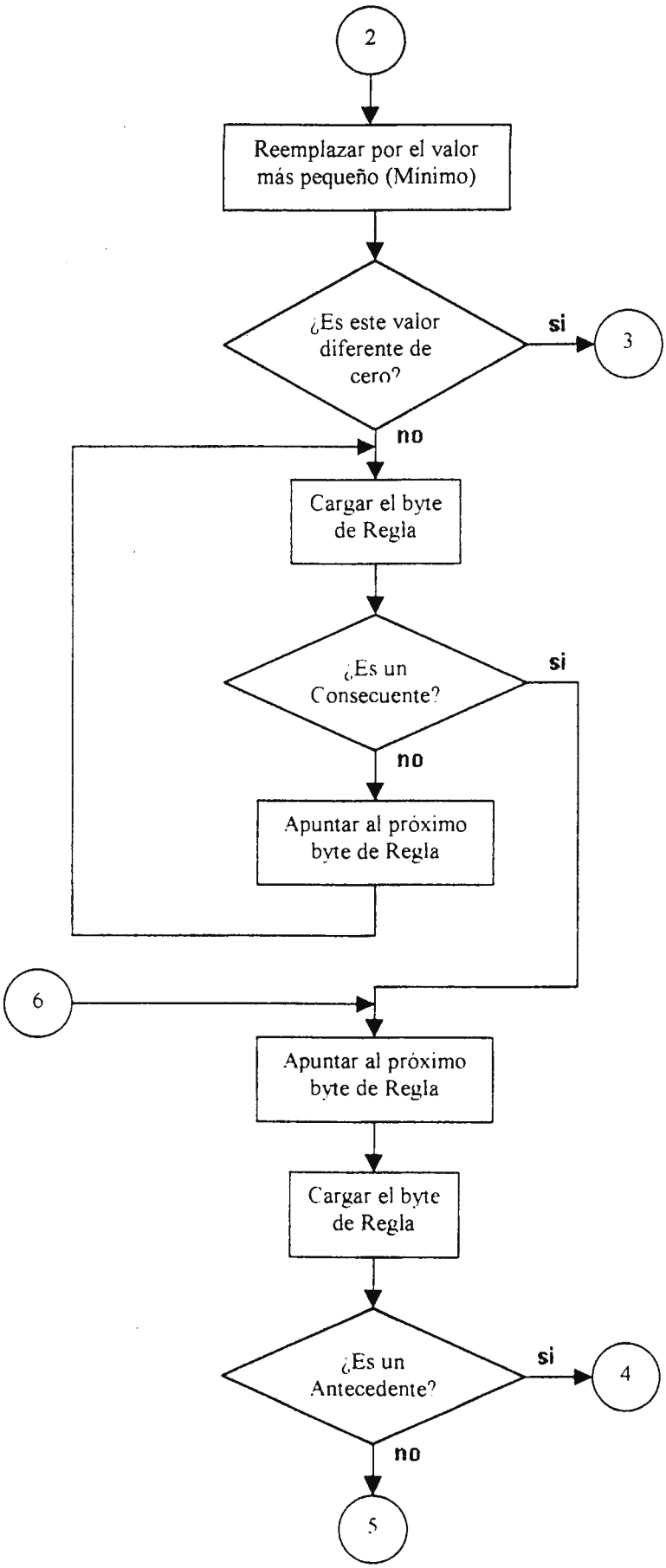

3
¿Es el resultado < $FF? (para verificar que el valor de entrada se encuentra dentro de la función de membresia)
si
no
Asignar grado de Pertenencia = 0
Calcular el grado de Pertenencia
2
1
Cargar nuevamente el valor de Entrada Actual a Fusificar
¿Se encuentra el valor de entrada a la izquierda del segmento_1(fuera de la función de membresia)?
si
2
no
Obtener el valor de la pendiente_1
4

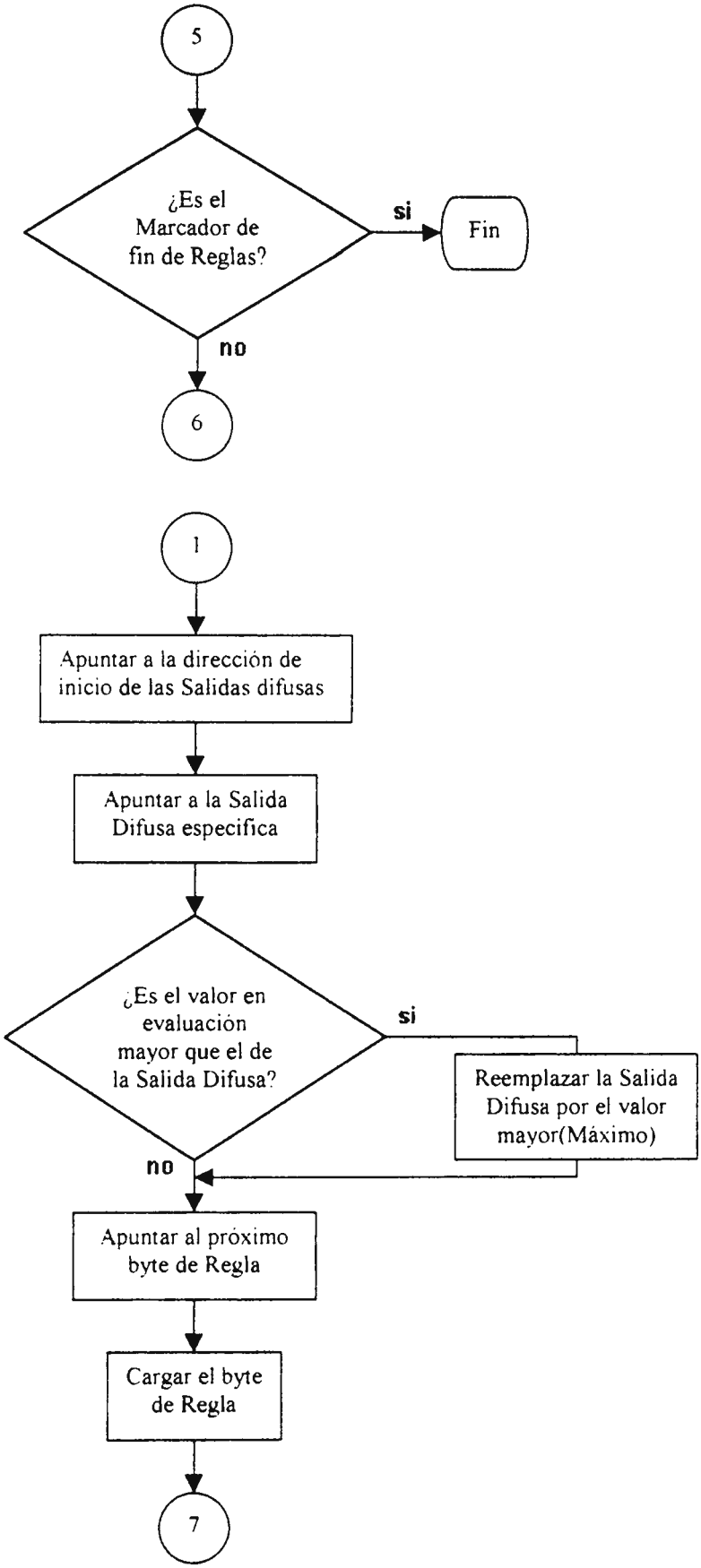

4
¿Es la pendiente_1 vertical?
si
no
Calcular el grado de Pertenencia
¿Es el grado de Pertenencia > $FF?
si
no
Asignar grado de Pertenencia = $FF
Calcular el grado de Pertenencia
2
Apuntar a la siguiente Función de Membresia
Recuperar el valor de Entrada Actual
Guardar el grado de Pertenencia en la tabla de Entradas Difusas
Apuntar a la próxima localidad de Entradas Difusas
5

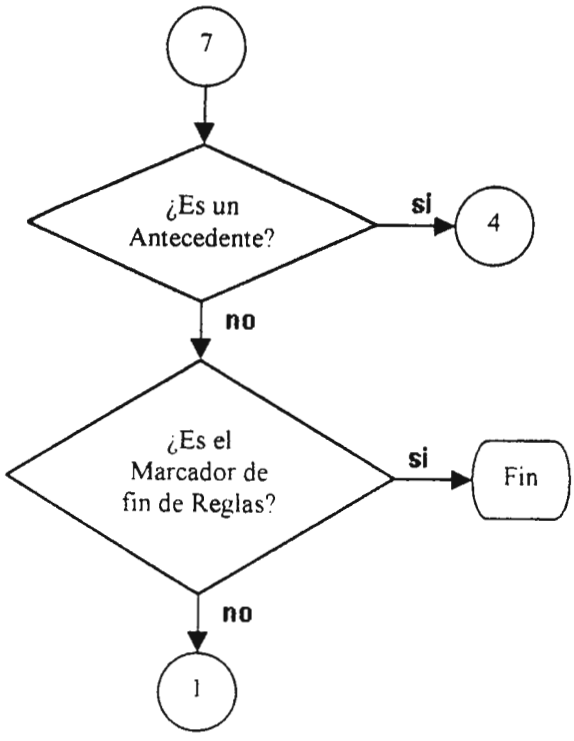

5
¿Se completaron los 8 grados de Pertenencia?
si
6
no
¿Se completaron las cuatro Entradas?
si
Fin
no
7

E.3. FLUJOGRAMA PARA LA EVALUACIÓN DE REGLAS.

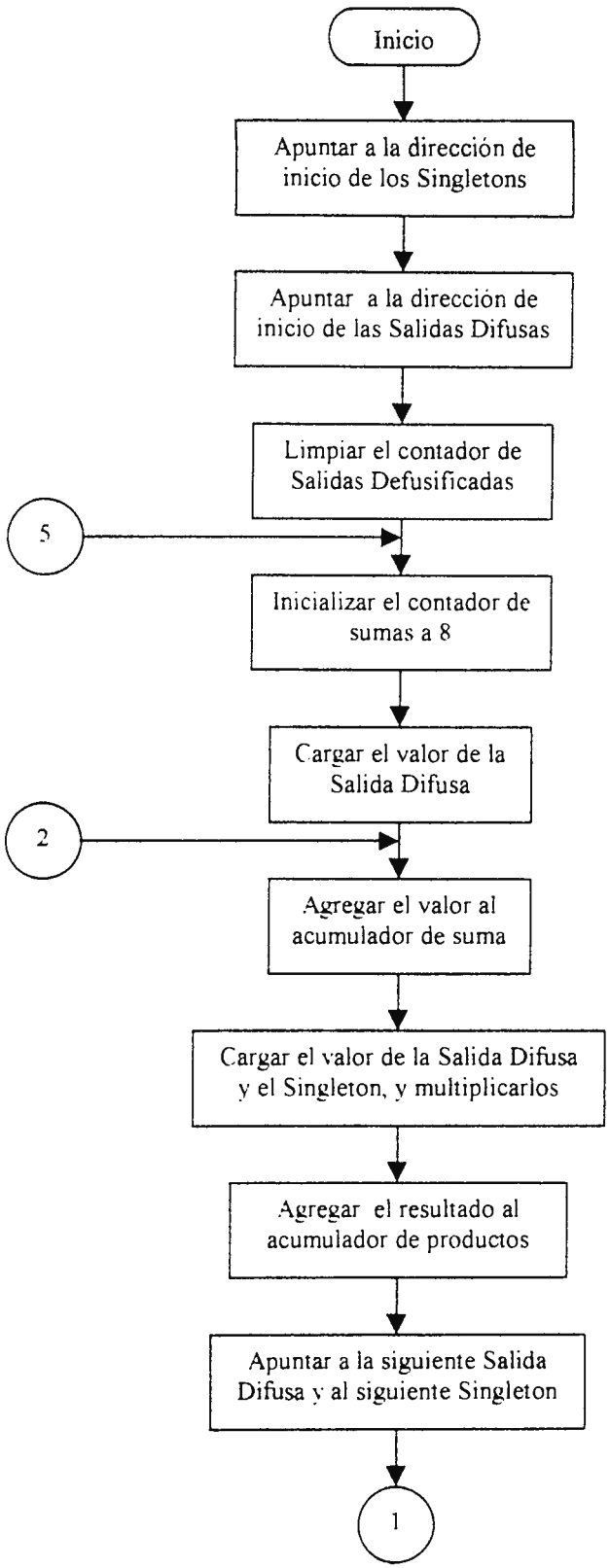


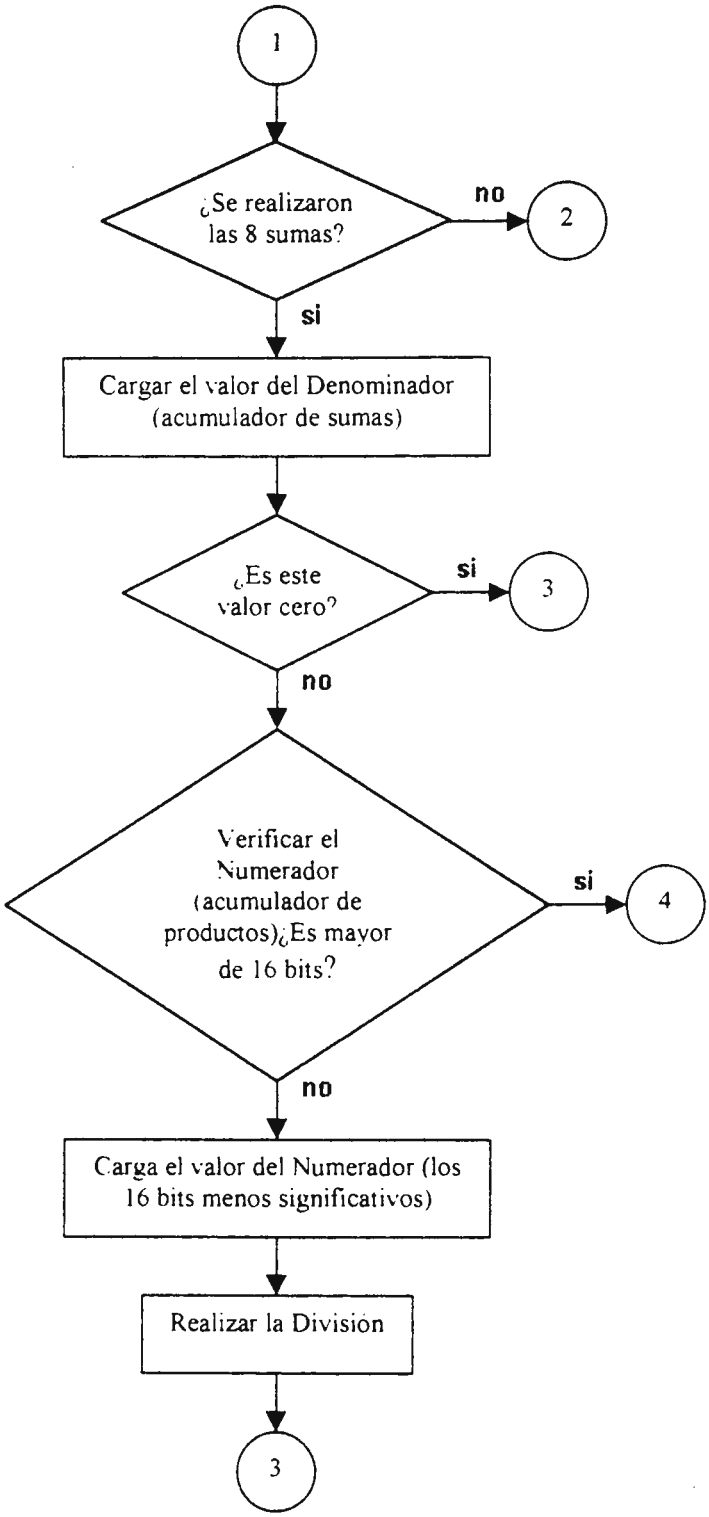

2
Reemplazar por el valor más pequeño (Mínimo)
¿Es este valor diferente de cero?
si
3
no
Cargar el byte de Regla
¿Es un Consecuente?
si
no
Apuntar al próximo byte de Regla
6
Apuntar al próximo byte de Regla
Cargar el byte de Regla
¿Es un Antecedente?
si
4
no
5

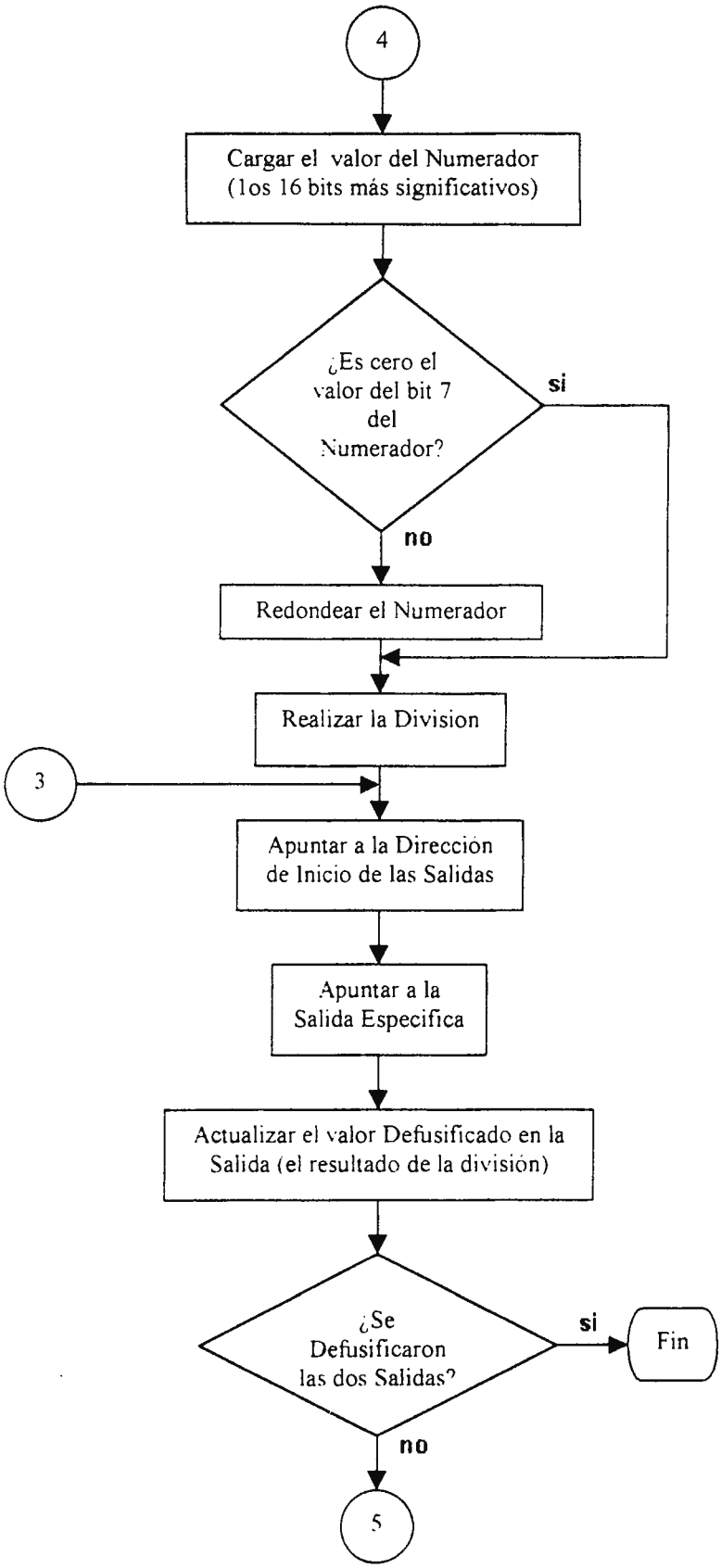

5
¿Es el Marcador de fin de Reglas?
si
Fin
no
6
1
Apuntar a la dirección de inicio de las Salidas difusas
Apuntar a la Salida Difusa especifica
¿Es el valor en evaluación mayor que el de la Salida Difusa?
si
Reemplazar la Salida Difusa por el valor mayor(Máximo)
no
Apuntar al próximo byte de Regla
Cargar el byte de Regla
7


7
¿Es un Antecedente?
si
4
no
¿Es el Marcador de fin de Reglas?
si
Fin
no
1

E.4. FLUJOGRAMA PARA LA DEFUSIFICACIÓN.




1
¿Se realizaron las 8 sumas?
no
2
si
Cargar el valor del Denominador (acumulador de sumas)
¿Es este valor cero?
si
3
no
Verificar el Numerador (acumulador de productos)¿Es mayor de 16 bits?
si
4
no
Carga el valor del Numerador (los 16 bits menos significativos)
Realizar la División
3


4
Cargar el valor del Numerador
(1os 16 bits más significativos)
¿Es cero el
valor del bit 7
del
Numerador?
si
no
Redondear el Numerador
Realizar la Division
3
Apuntar a la Direcciòn
de Inicio de las Salidas
Apuntar a la
Salida Especifica
Actualizar el valor Defusificado en la
Salida (el resultado de la divisiòn)
¿Se
Defusificaron
las dos Salidas?
si
Fin
no
5

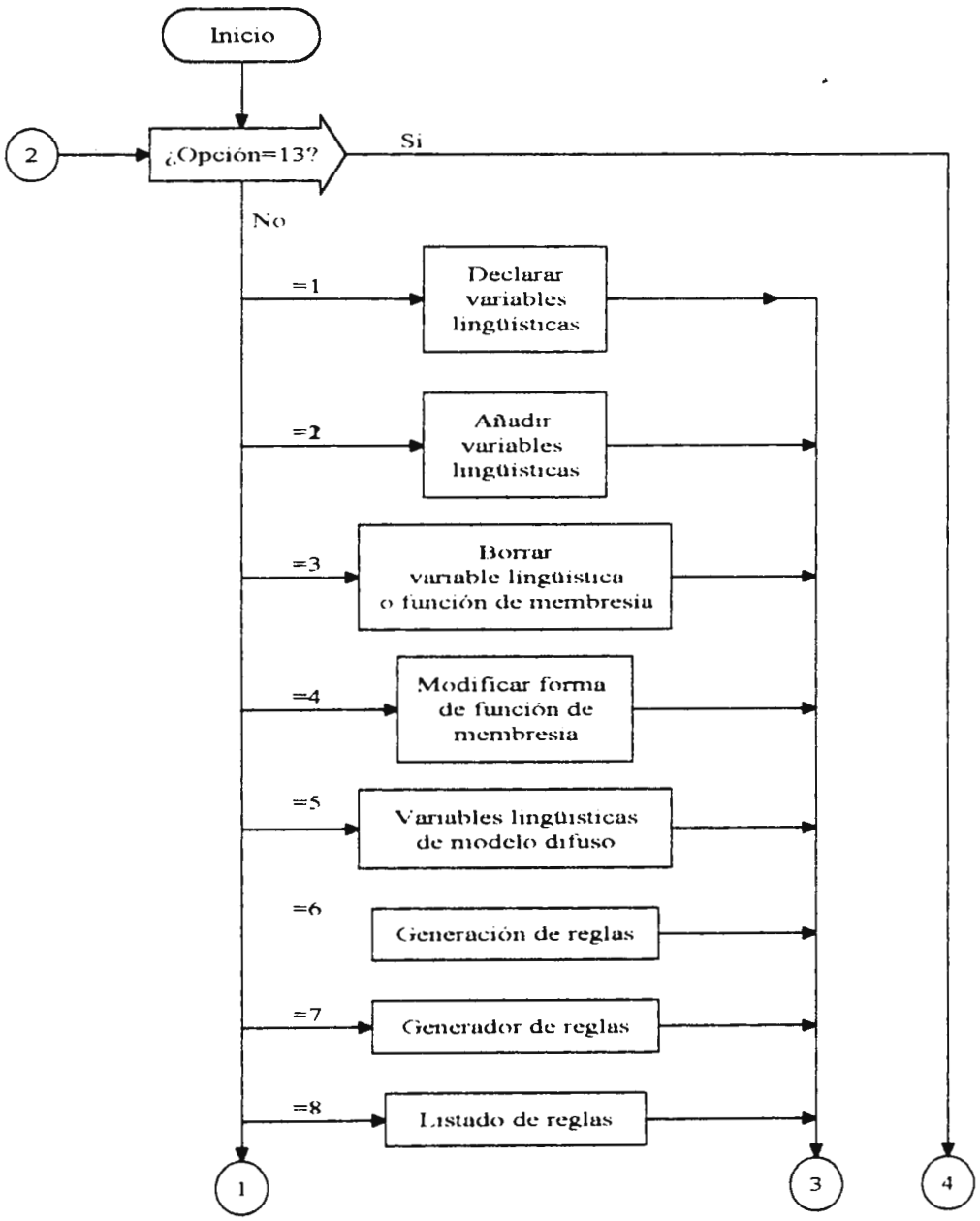
APENDICE F. PROGRAMA DE INFERENCIA DIFUSA PARA EL MC68HC11.

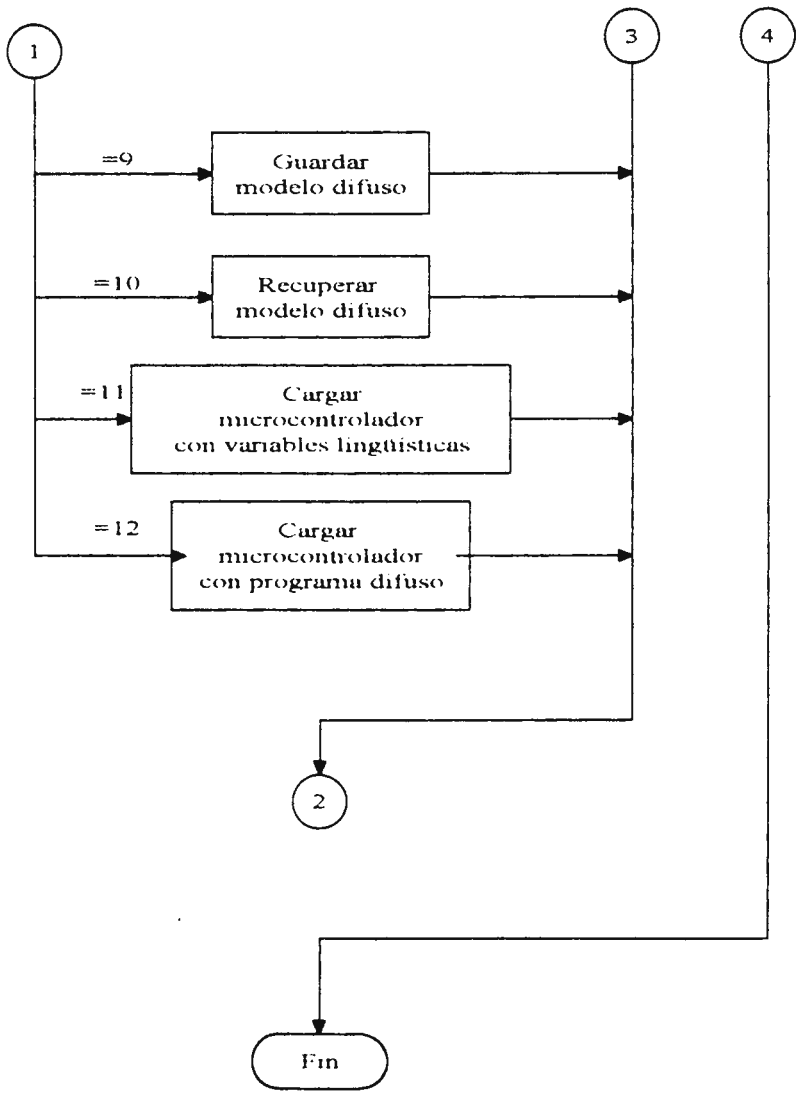
```
                          ORG     $C000                   ;inicio del programa

ENTRADAS_DIFUSAS          RMB     32                      ;almacenamiento para las entradas difusas
SALIDAS_DIFUSAS           RMB     16                      ;almacenamiento para las salidas difusas
ENTRADAS_DEL_SISTEMA      EQU     $1031                   ;dirección de inicio de los registros de datos de entrada del sistema
SALIDAS_DEL_SISTEMA       EQU     $1003                   ;dirección de inicio de los registros de salida
FUNCIONES_DE_MEMBRESIA    EQU     $C200                   ;dirección de inicio de las funciones de membresía entrada
CONTADOR_DE_MEMBRESIAS    RMB     1                       ;almacenamiento para el contador de funciones de membresía
REGLAS                    EQU     $C290                   ;dirección de inicio de las reglas del sistema
SINGLETONS                EQU     $C280                   ;dirección de inicio de las funciones de membresía de salida
LAZO_DE_SALIDA            RMB     1                       ;almacenamiento para el contador del lazo de salida
CONTADOR_DE_SUMA          RMB     1                       ;almacenamiento para el contador de la suma difusa
SUMA_FUZZY                RMB     2                       ;almacenamiento para la suma difusa
SUMA_DE_PRODUCTOS         RMB     3                       ;almacenamiento para la suma de productos difusos

                          LDAA    #$FF
                          STAA    $1007                   ;activa el puerto C como de salida

                          LDAA    #$80
                          ORAA    $1039
                          STAA    $1039                   ;activa el sistema de conversión análogo a digital

;inicio del ciclo de inferencia difuso

INICIO                    LDAA    #$14
                          STAA    $1030                   ;ejecuta una conversión A/D para los pines 4,5,6 y 7 del puerto E

CONVERSION_INCOMPLETA     LDAA    $1030                   ;se verifica si concluye la conversión
                          ANDA    #$80
                          BEQ     CONVERSION_INCOMPLETA

;inicio de la defusificación

                          LDY     #ENTRADAS_DIFUSAS       ;puntero de la tabla de las entradas difusas
                          LDX     #ENTRADAS_DEL_SISTEMA   ;puntero de las entradas del sistema a procesar
                          LDD     2,X                     ;se almacenan los valores de las entradas del sistema en la pila
                          PSHB
                          PSHA
                          LDD     0,X
                          PSHB
                          PSHA

                          LDX     #FUNCIONES_DE_MEMBRESIA ;puntero de las funciones de membresía
ENTRADA_SIGUIENTE         PULA                            ;carga el valor de la entrada actual
                          LDAB    #7                      ;coloca el valor para 8 funciones de membresía
                          STAB    CONTADOR_DE_MEMBRESIAS  ;inicializa el contador del lazo

;inicio del proceso para obtener el grado de pertenencia de la entrada actual para cada función de membresía

OBTENER_GRADO             PSHA                            ;salva el valor de entrada actual
                          CLRB                            ;en caso de que el grado de pertenencia sea cero se limpia
                          SUBA    2,X                     ;valor de entrada menos punto 2
                          BLS     FUERA_DEL_SEGMENTO_2    ;salta si el valor esta fuera del segmento 2(entrada < punto 2)
                          LDAB    3,X                     ;obtiene el valor de la pendiente 2
                          BEQ     GUARDAR_GRADO           ;salta si la pendiente es vertical
                          MUL                             ;se procede a obtener el grado de pertenencia
                          TSTA                            ;verifica si la entrada se encuentra dentro de la función de membresía
                          BEQ     DENTRO_DE_MEMBRESIA     ;si lo esta salta a dentro de membresía
                          CLRB                            ;si no lo esta asignar el valor de cero al grado de pertenencia
                          BRA     GUARDAR_GRADO           ;salta a guardar el grado de pertenencia

DENTRO_DE_MEMBRESIA       SUBB    #$FF                    ;se resta para encontrar el grado de pertenencia
                          NEGB                            ;se niega para obtener el grado de pertenencia
                          BRA     GUARDAR_GRADO           ;salta a guardar el grado de pertenencia

FUERA_DEL_SEGMENTO_2      ADDA    2,X                     ;se recupera el valor de entrada
                          SUBA    0,X                     ;valor de entrada menos punto 1
                          BLO     GUARDAR_GRADO           ;si valor de entrada < punto 1, salta a guardar grado de pertenencia = 0
                          LDAB    1,X                     ;obtiene el valor de la pendiente 1
```

```
                        BEQ     PENDIENTE_VERTICAL      ;salta si la pendiente es vertical
                        MUL                             ;se procede a obtener el grado de pertenencia
                        TSTA                            ;verifica si el valor es mayor que $FF
                        BEQ     GUARDAR_GRADO           ;salte a guardar el grado de pertenencia

PENDIENTE_VERTICAL      LDAB    #$FF                    ;obtiene el grado de membresía $FF

GUARDAR_GRADO           INX                             ;se apunta a la siguiente función de membresía
                        INX
                        INX
                        INX
                        PULA                            ;recupera el valor de entrada actual
                        STAB    0,Y                     ;guarda  el grado de pertenencia en la tabla de entradas difusas
                        INY                             ;apunta a la próxima entrada difusa
                        DEC     CONTADOR_DE_MEMBRESIAS  ;decrementa el contador de membresias procesadas
                        BPL     OBTENER_GRADO           ;si no se completaron las 8 salta a la función siguiente
                        CMPY    #ENTRADAS_DIFUSAS+32    ;verifica si se completaron las cuatro entradas
                        BNE     ENTRADA_SIGUIENTE       ;si no se completaron salta a entrada siguiente

;limpia las 16 salidas difusas

                        LDX     #SALIDAS_DIFUSAS        ;puntero de la tabla de las salidas difusas
                        LDAA    #16                     ;se inicializa para 16 salidas difusas
LIMPIAR_SALIDAS         CLR     0,X                     ;limpia una salida difusa
                        INX                             ;apunta a la siguiente salida difusa
                        DECA                            ;decrementa del contador del lazo
                        BNE     LIMPIAR_SALIDAS         ;continúa hasta limpiarlas todas

;inicio de la evaluación de las reglas

                        LDY     #REGLAS                 ;puntero  de las reglas
EVALUACION              LDAA    #$FF                    ;comienza una cadena de reglas

LAZO_IF                 LDAB    0,Y                     ;obtiene el byte de regla
                        BMI     LAZO_THEN               ;salte si es un consecuente(MSB=1) hacia el lazo THEN
                        INY                             ;apunta al próximo byte de regla
                        LDX     #ENTRADAS_DIFUSAS       ;puntero de las entradas difusas
                        ABX                             ;apunta a la entrada difusa especifica
                        CMPA    0,X                     ;es esta entrada difusa más alta?
                        BLS     LAZO_IF                 ;si lo es salta a obtener el próximo byte de regla
                        LDAA    0,X                     ;reemplaza la salida difusa por la más baja
                        BNE     LAZO_IF                 ;si es cero salta a la parte THEN

;un valor de cero de entrada difusa hace que la regla sea completamente no verdadera
;por lo tanto, salta el resto de la parte IF y todas las partes THEN para empezar la próxima regla

ENCONTRAR_THEN          LDAB    0,Y                     ;obtiene el próximo byte de regla
                        BMI     ENCONTRAR_IF            ;salte si es un consecuente(MSB=1)
                        INY                             ;apunta a la próxima regla
                        BRA     ENCONTRAR_THEN          ;sale del lazo hasta encontrar un consecuente

ENCONTRAR_IF            INY                             ;incrementa el puntero de regla para la parte IF
                        LDAB    0,Y                     ;obtiene el próximo byte de regla
                        BPL     EVALUACION              ;salte si es un antecedente(MSB=0)
                        CMPB    #$FF                    ;busca el marcador de fin de reglas($FF)
                        BNE     ENCONTRAR_IF            ;continúa buscando un antecedente o el marcador de fin de reglas $FF
                        BRA     DEFUZZYFICACION         ;si completo todas las reglas, salta a la defusificación

LAZO_THEN               LDX     #SALIDAS_DIFUSAS        ;puntero de las salidas difusa
                        ANDB    #$0F                    ;retiene el número de salida y de etiqueta
                        ABX                             ;apunta a la salida difusa especifica

;el grado de membresía para la regla esta en el acumulador A

                        CMPA    0,X                     ;compara la salida difusa
                        BLO     MENOR                   ;salta si no es más alto
                        STAA    0,X                     ;actualiza por grado más alto(Máximo)

MENOR                   INY                             ;apunta a la próxima regla
                        LDAB    0,Y                     ;obtiene el  byte de regla
                        BPL     EVALUACION              ;si es un antecedente(MSB=0)salta a obtener la nueva regla
                        CMPB    #$FF                    ;busca el marcador de fin de reglas($FF)
                        BNE     LAZO_THEN               ;sino lo es, es un consecuente

;inicio del proceso de defusificación

DEFUZZYFICACION         LDY     #SINGLETONS             ;puntero de las funciones de membresía de salida
                        LDX     #SALIDAS_DIFUSAS        ;puntero de la tabla de las salidas difusas
                        CLR     LAZO_DE_SALIDA          ;limpia el contador del lazo de defusificación de salidas
LAZO_DE_C.G.            LDAB    #8
```

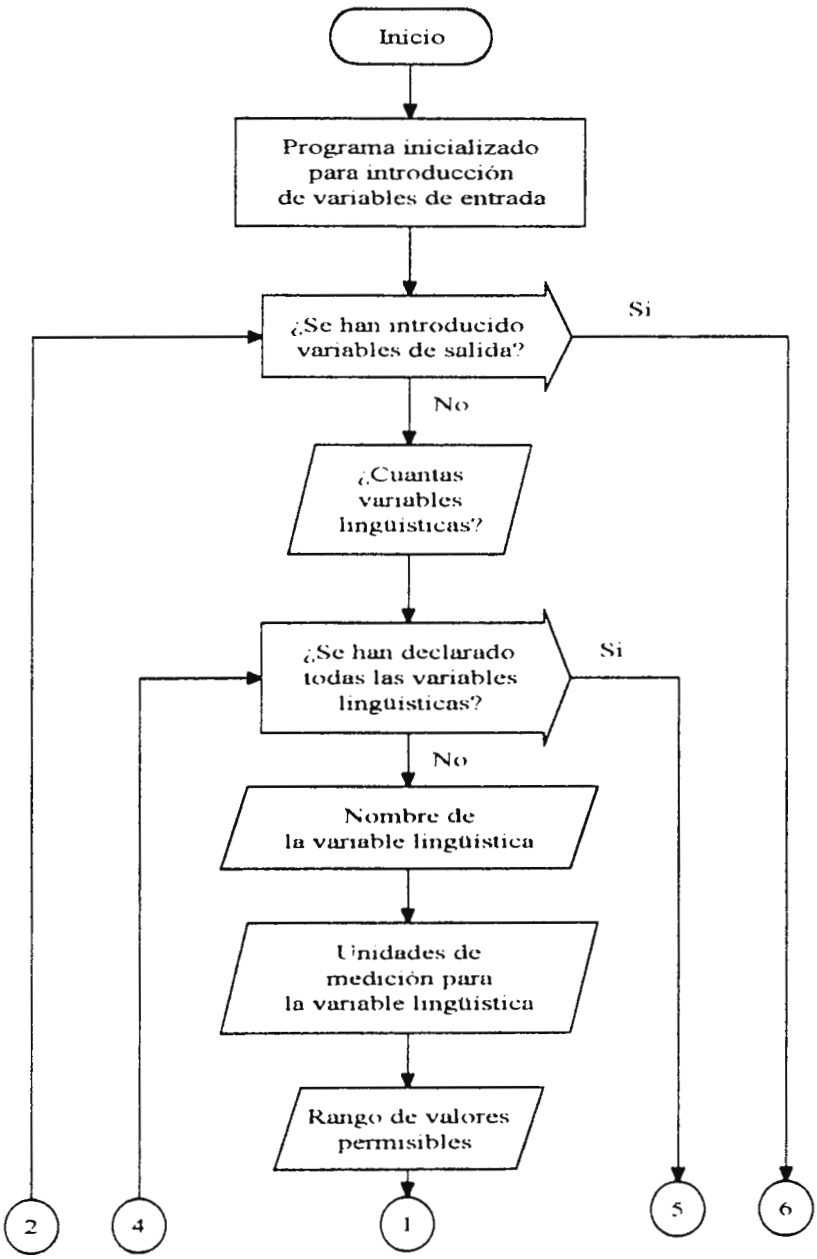
```
                    STAB    CONTADOR_DE_SUMA    ;inicializa la cuenta del lazo de sumas
                    LDD     #$0000
                    STD     SUMA_FUZZY          ;limpia la suma
                    STD     SUMA_DE_PRODUCTOS+1 ;limpia los 16 bits menos significativos de la suma de productos
                    STAA    SUMA_DE_PRODUCTOS   ;limpia los 8 bits más significativos de la suma de productos

LAZO_DE_SUMAS       LDAB    0,X                 ;obtiene una salida difusa
                    CLRA                        ;se limpia el acumulador A
                    ADDD    SUMA_FUZZY          agrega la salida difusa presente a la suma de salidas difusas
                    STD     SUMA_FUZZY          ;actualiza la suma de salidas difusas
                    LDAA    0,X                 ;obtiene la salida difusa nuevamente
                    LDAB    0,Y                 ;obtiene la función de membresía de salida singleton
                    MUL                         ;realiza la multiplicación de la salida difusa por el singleton
                    ADDD    SUMA_DE_PRODUCTOS+1 ;agrega el resultado de la multiplicación a los 16 lsb suma de productos
                    STD     SUMA_DE_PRODUCTOS+1 ;actualiza los 16 lsb de  la suma de productos
                    LDAA    SUMA_DE_PRODUCTOS   ;obtiene el valor de los 8 msb de la suma de productos
                    ADCA    #0                  ,agrega el acarreo de la suma de los 16 lsb a los 8 msb
                    STAA    SUMA_DE_PRODUCTOS   ;actualiza el valor de los 8 msb de la suma de productos
                    INY                         ;apunta hacia la siguiente función de membresía singleton
                    INX                         ;apunta hacia la siguiente salida difusa
                    DEC     CONTADOR_DE_SUMA    ;decrementa el contador del lazo de suma
                    BNE     LAZO_DE_SUMAS       ;si realiza las ocho sumas continúa, sino las termina
                    PSHX                        ;guarda el valor de la salida difusa
                    CLRA                        ;limpia el acumulador en caso que el numerador sea cero
                    LDX     SUMA_FUZZY          ;obtiene el valor del denominador
                    BEQ     GUARDAR_SALIDA      ;salta si es cero
                    TST     SUMA_DE_PRODUCTOS   ;verifica si el numerador es mayor de 16 bits
                    BNE     MAYOR_DE_DOS_BYTES  ;si lo es salta
                    LDD     SUMA_DE_PRODUCTOS+1 ;obtiene el valor del numerador
                    IDIV                        ;realiza la división
                    XGDX                        ;pasa el resultado al acumulador b
                    TBA                         ;pasa el resultado al acumulador a
                    BRA     GUARDAR_SALIDA      ;salta a actualizar la salida
MAYOR_DE_DOS_BYTES  LDD     SUMA_DE_PRODUCTOS   ;obtiene los 16 bits más significativos de numerador
                    TST     SUMA_DE_PRODUCTOS+2 ;verifica si el bit 8 del numerador es cero
                    BPL     NO_REDONDEAR        ;si lo es salta
                    ADDD    #1                  ;redondea el numerador

NO_REDONDEAR        FDIV                        ;realiza la división
                    XGDX                        ;pasa el resultado al acumulador a

GUARDAR_SALIDA      LDX     #SALIDAS_DEL_SISTEMA ;puntero de las salidas del sistema
                    LDAB    LAZO_DE_SALIDA      ;obtiene el número de la salida presente
                    ABX                         ;apunta a la salida presente
                    STAA    0,X                 ;actualiza la salida presente
                    PULX                        ;recobra e valor de la salida difusa
                    INCB                        ;incrementa el contador del lazo de defusificación de salidas
                    STAB    LAZO_DE_SALIDA      ;actualiza el contador del lazo de defusificación de salidas
                    CMPB    #2                  ;verifica si defusificó las dos salidas
                    BNE     LAZO_DE_C.G.        ,si lo hizo continúa, sino defusifica la segunda salida

                    JMP     INICIO              ;inicia un nuevo ciclo de inferencia
```

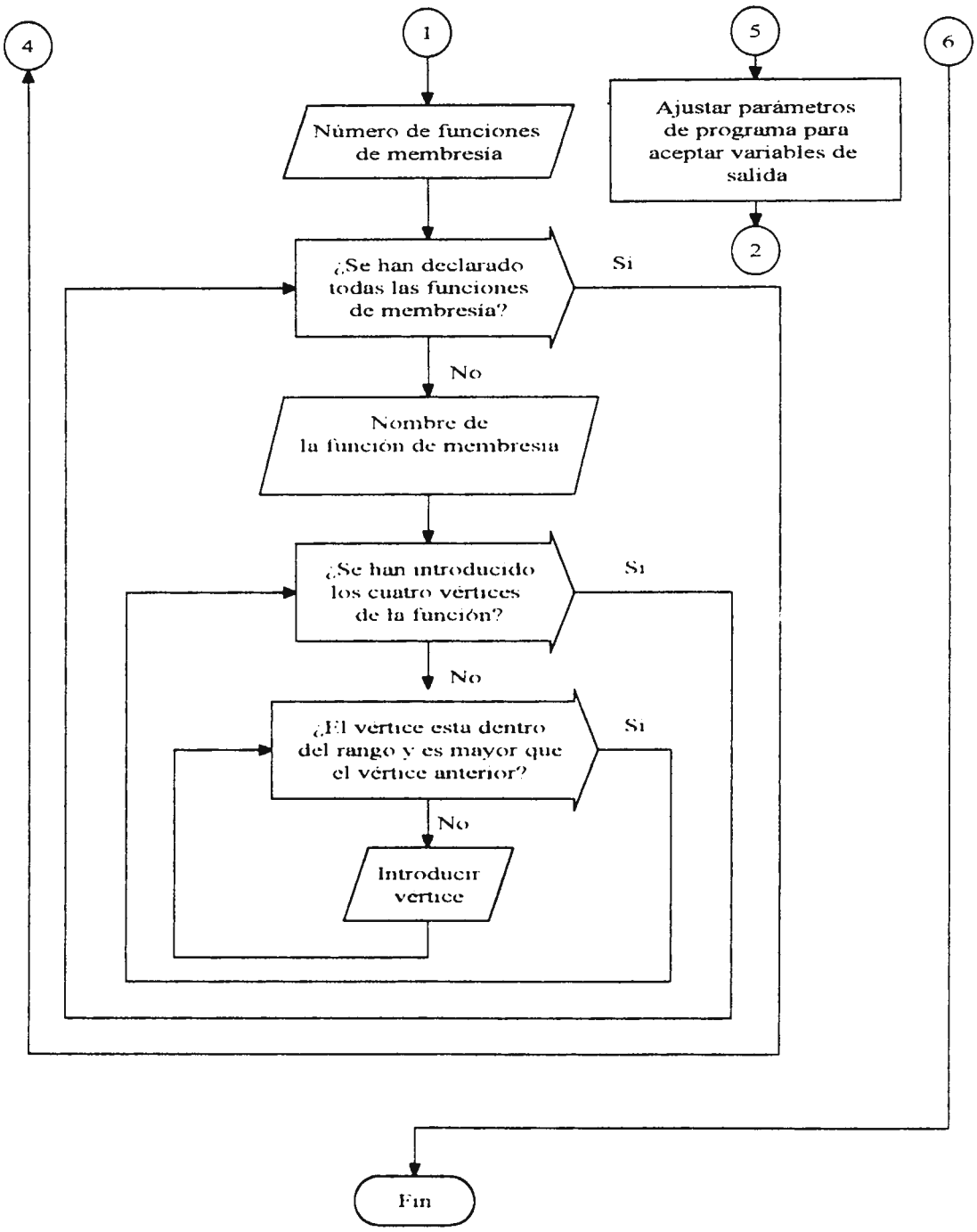
APENDICE G. FLUJOGRAMA DE MENU PRINCIPAL DE PROGRAMA MCFUZLOG



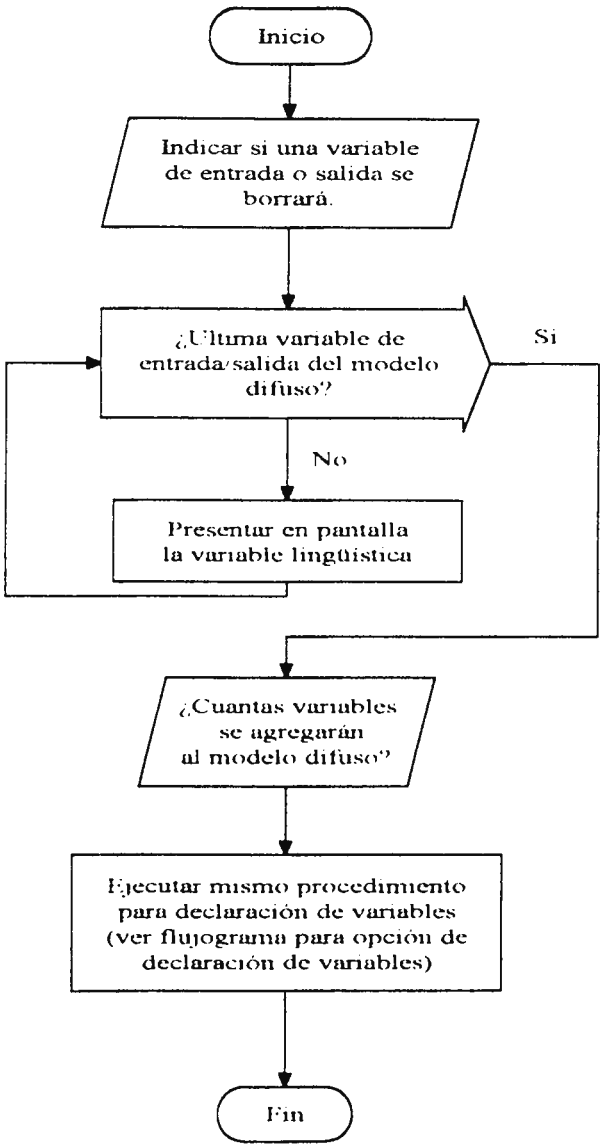

1
3
4
=9
Guardar modelo difuso
=10
Recuperar modelo difuso
=11
Cargar microcontrolador con variables lingüísticas
=12
Cargar microcontrolador con programa difuso
2
Fin

G.1. FLUJOGRAMA DE OPCION DECLARAR VARIABLE LINGÜÍSTICA.

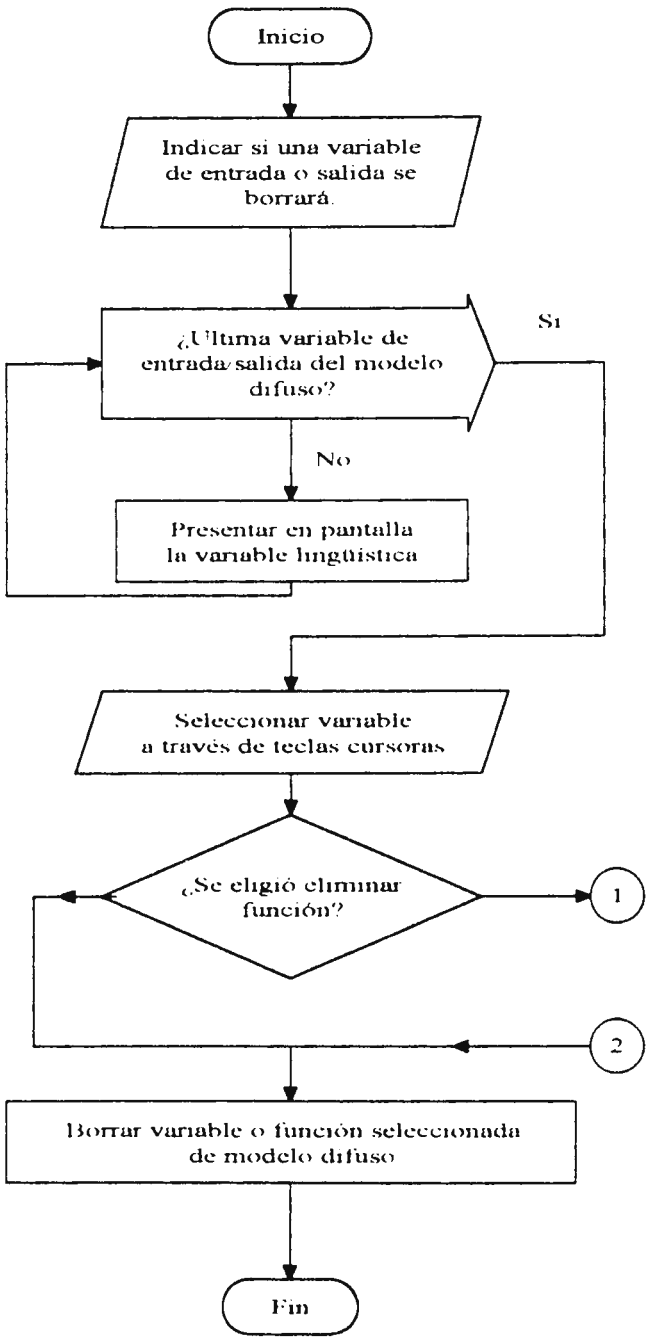


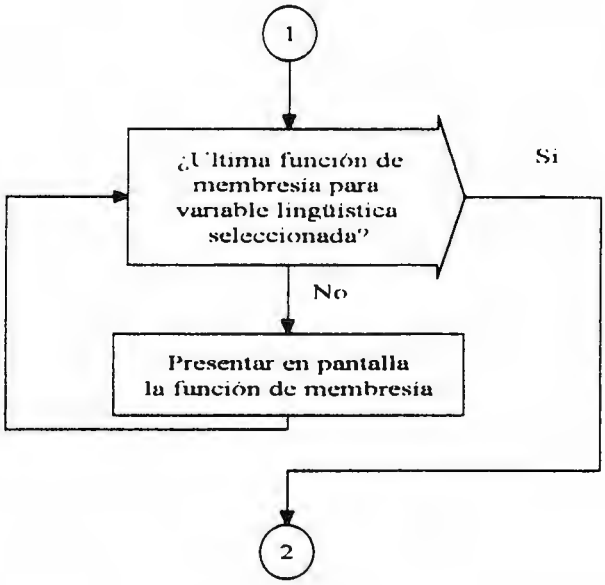

4
1
5
6
Número de funciones de membresía
Ajustar parámetros de programa para aceptar variables de salida
2
¿Se han declarado todas las funciones de membresía?
Si
No
Nombre de la función de membresía
¿Se han introducido los cuatro vértices de la función?
Si
No
¿El vértice esta dentro del rango y es mayor que el vértice anterior?
Si
No
Introducir vertice
Fin

G.2. FLUJOGRAMA DE OPCION AÑADIR VARIABLE.

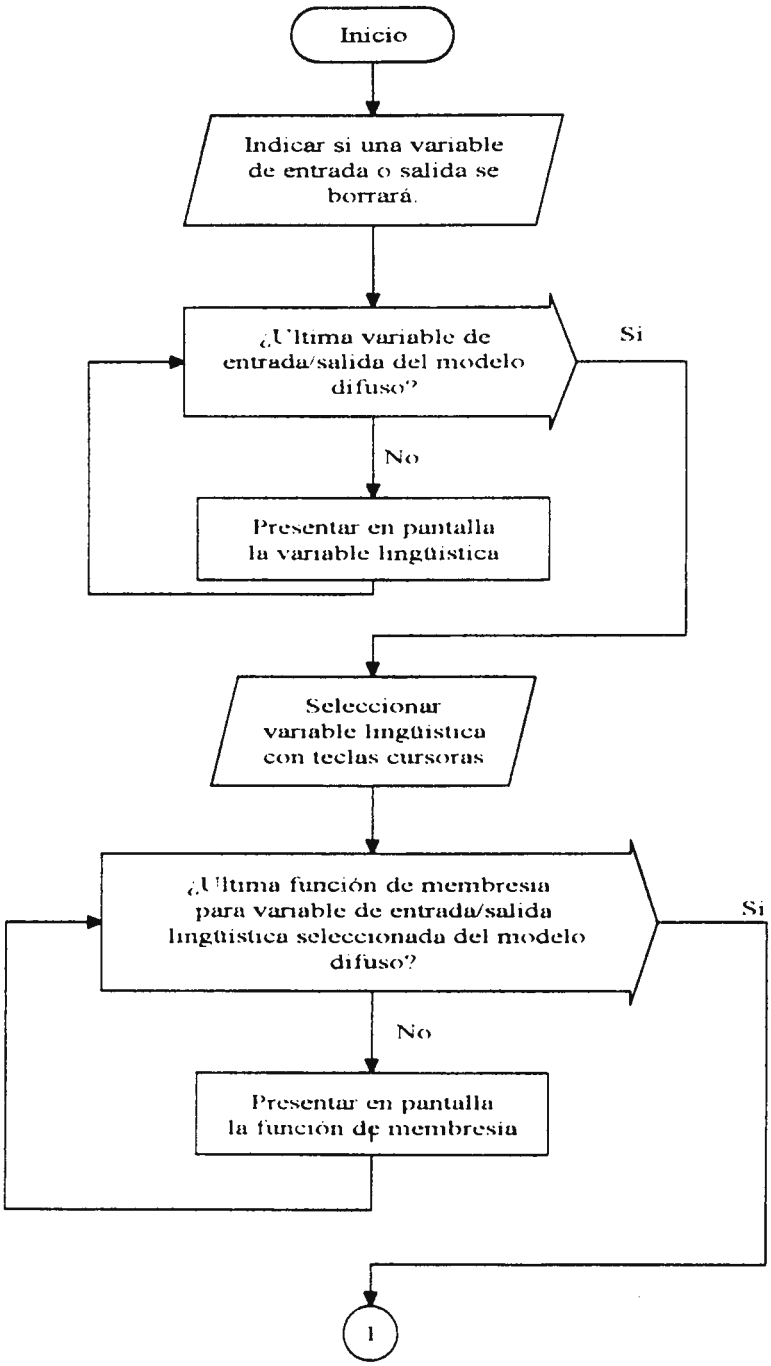


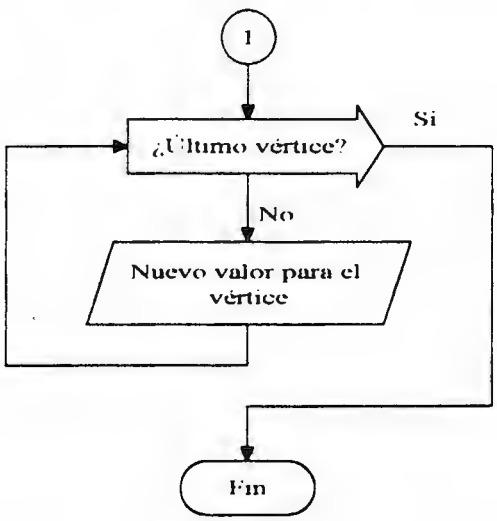
G.3. FLUJOGRAMA DE OPCION BORRAR VARIABLE.



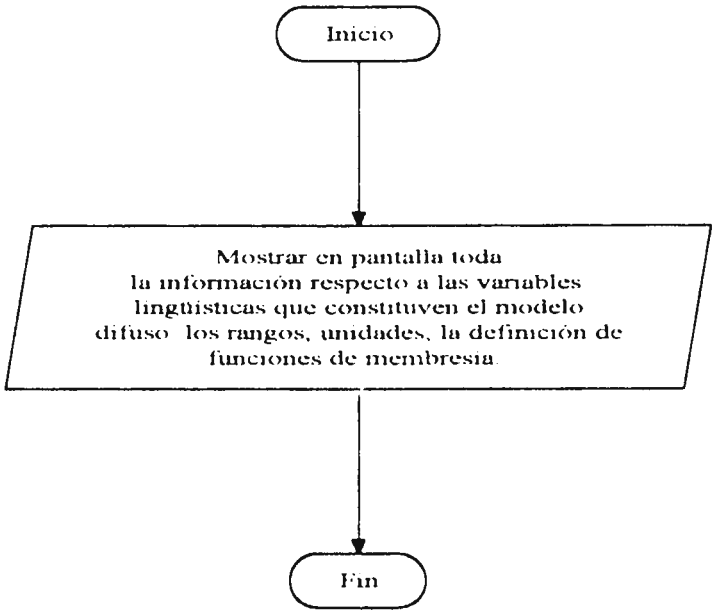

1
¿Ultima función de membresía para variable lingüística seleccionada?
Si
No
Presentar en pantalla la función de membresía
2

G.4 FLUJOGRAMA DE OPCION MODIFICAR FUNCION DE MEMBRESIA.

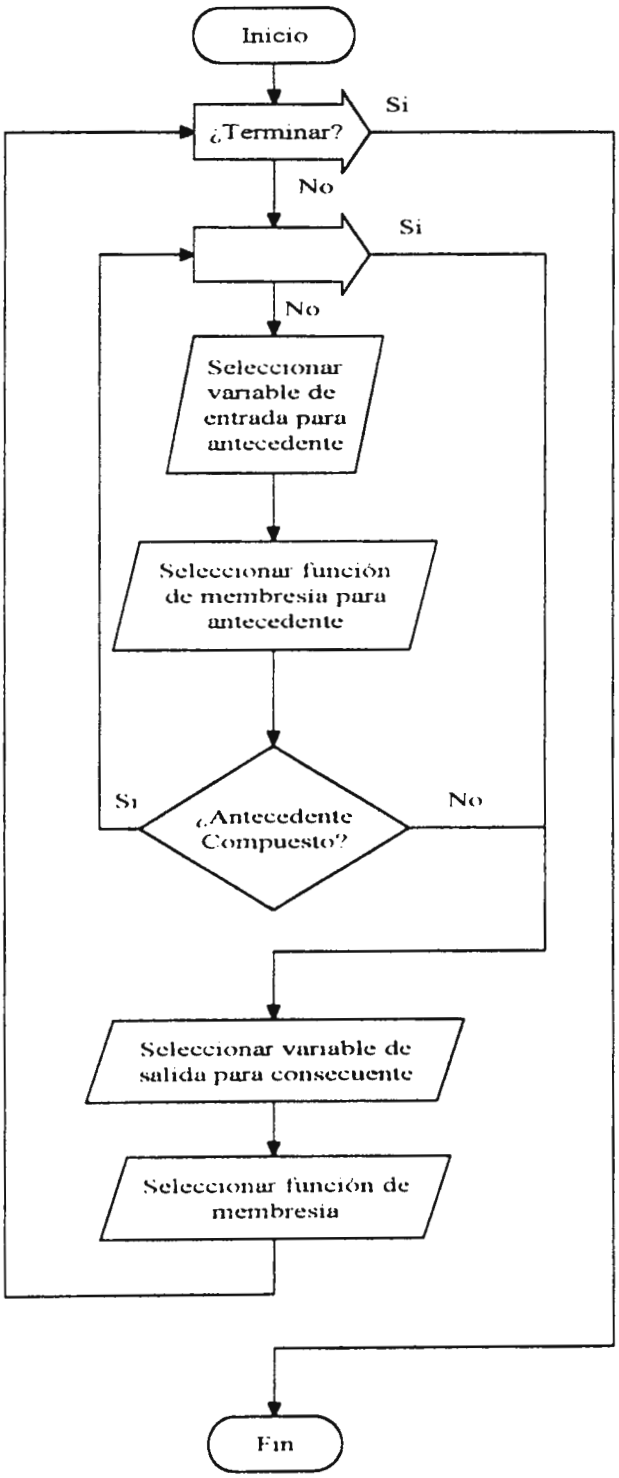



1
¿Último vértice?
Si
No
Nuevo valor para el vértice
Fin

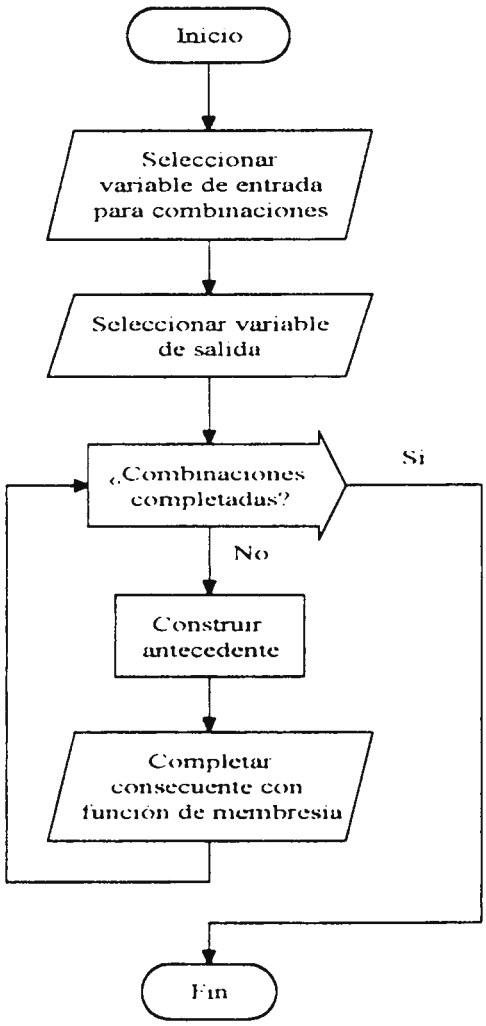
G.5. FLUJOGRAMA DE OPCION VARIABLES DE MODELO DIFUSO.



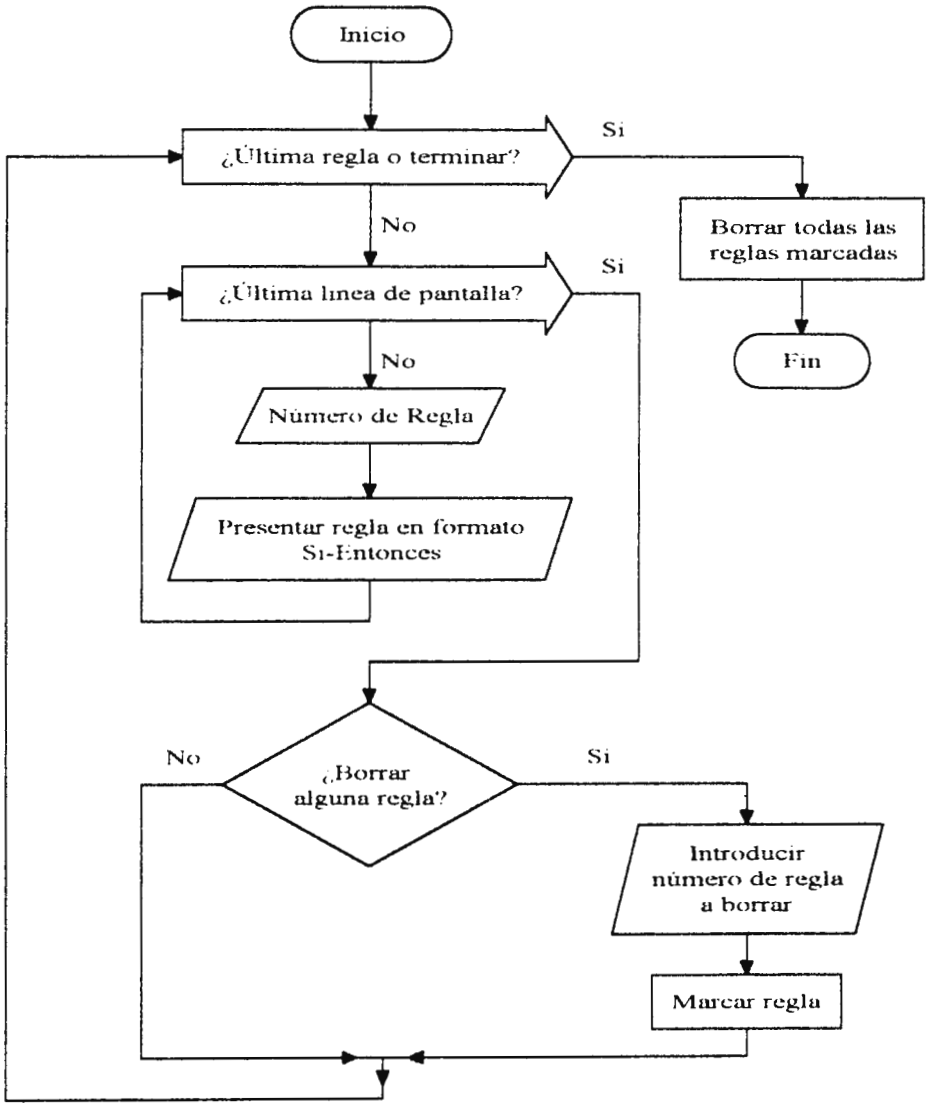
G.6. FLUJOGRAMA DE OPCION GENERAR REGLAS.



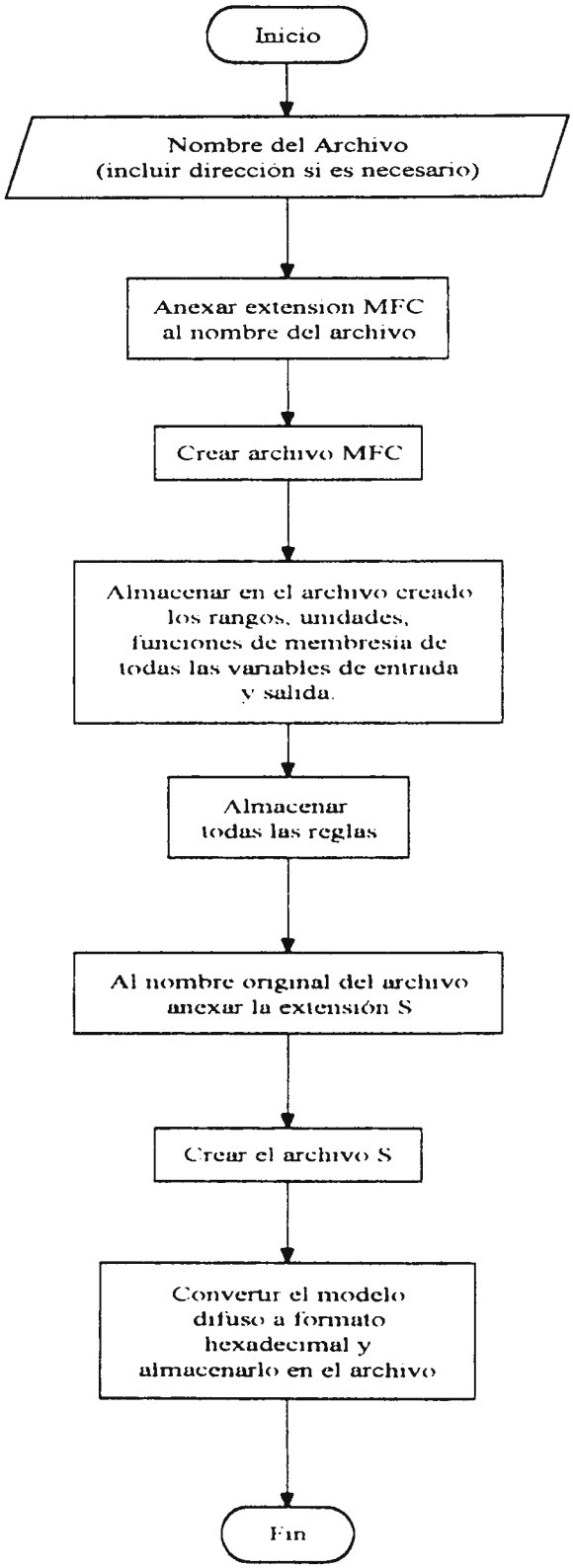
G.7. FLUJOGRAMA DE OPCION GENERADOR DE REGLAS.



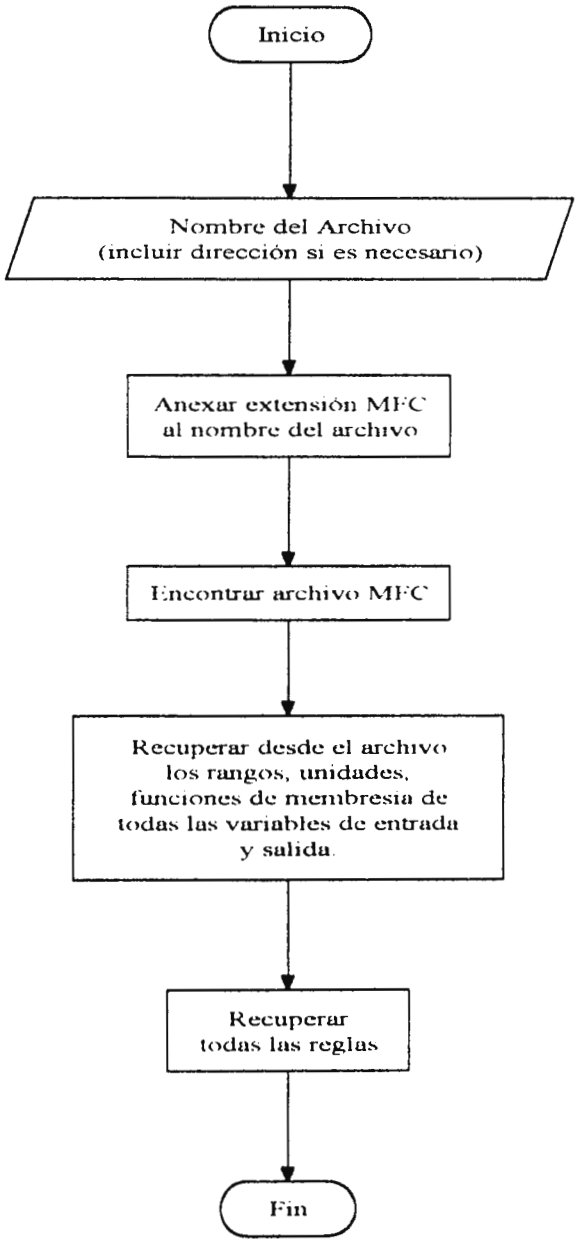
G.8. FLUJOGRAMA DE OPCION LISTADO DE REGLAS.



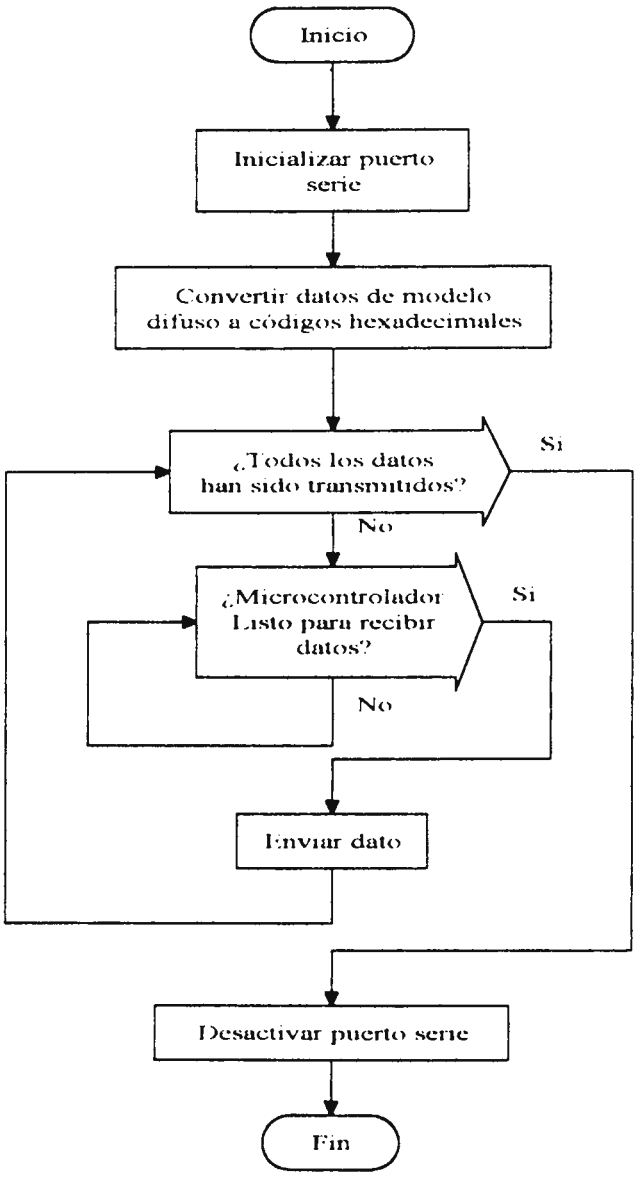
G.9. FLUJOGRAMA DE OPCION GUARDAR MODELO DIFUSO.



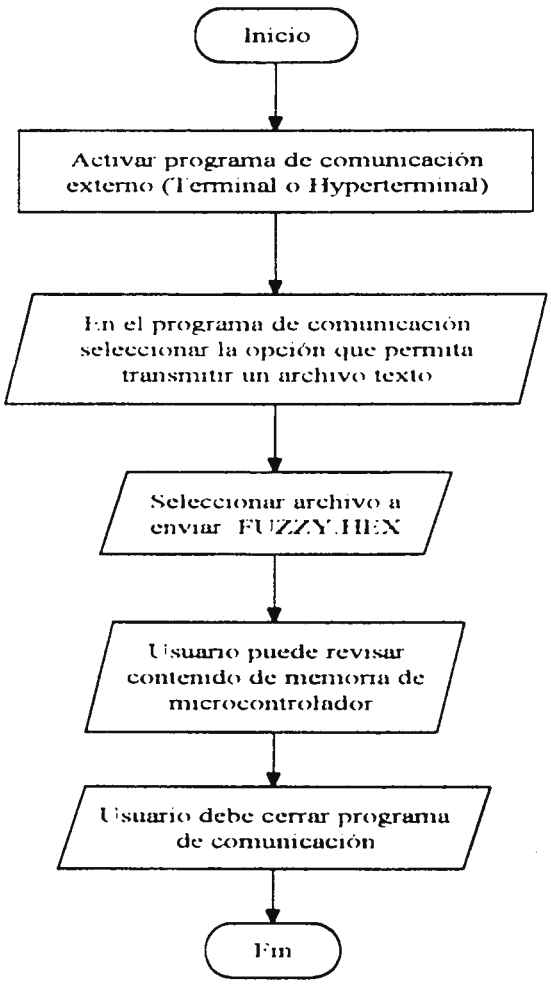
G.10. FLUJOGRAMA DE OPCION RECUPERAR MODELO DIFUSO.



G.11. FLUJOGRAMA DE OPCION CARGAR MICROCONTROLADOR CON MODELO DIFUSO.



G.12. FLUJOGRAMA DE OPCION CARGAR MICROCONTROLADOR CON PROGRAMA DIFUSO.



APENDICE H. LISTADO DEL PROGRAMA MCFUZLOG.

```
/*
 Programa: MCFUZLOG.C
 Objetivo: Interfaz entre usuario y microcontrolador MC68HC11
*/

#include "stdio.h"
#include "string.h"
#include "stdlib.h"
#include "process.h"
#include "conio.h"
#include "ctype.h"
#include "alloc.h"
#include "math.h"
#include "dos.h"

/* DECLARACIÓN DE CONSTANTES */

#define cabecera 6
#define vector1 4
#define vector2 8
#define digit 100
#define longitud 50
#define points  4
#define caracter 10
#define input   0
#define output  1
#define inout   2
#define enter 1000
#define terminar 2000
#define operadory 3000
#define limite 1000
#define proposicion
#define opcion 13


/*PUNTERO DE ARCHIVO*/

FILE *out,*in;

/* DECLARACIÓN DE VARIABLES */

int io,v[inout],fm[inout][vector1],n_or,i,j,k,w,x,z;
int p, control, point=3,controle=0,controlf=0;
int vec1=vector1, vec2=vector2;
int b[8],l=0,sel;
unsigned int bin;
char v_nombre[inout][vector1][longitud],unidades[inout][vector1][longitud],fm_nombre[inout][vector1][vector2][longitud];
char nombre[longitud];
char ans;
char hexa_uc[inout][vector1][vector2][points][caracter],law_hex[limite][caracter];
float aux,punto_anterior,lim_inf[inout][vector1],lim_sup[inout][vector1];
float hexa[inout][vector1][vector2][points], paso_cuan[inout][vector1];
float punto[inout][vector1][vector2][points];
float escala=256.0,fac_mul=100.0,portadora;
char *direccion="entrada";
char *regla;
char answ[3];
char arch_name[20];
char *arch_nombre, *name;
int law[limite];
char *opciones[opcion]={"DECLARACIÓN DE VARIABLES","AÑADIR VARIABLE",
                        "BORRAR VARIABLE O FUNCIÓN",
                        "MODIFICAR FORMA DE FUNCIÓN" "VARIABLES DE MODELO DIFUSO",
                        "GENERACIÓN DE REGLAS", "GENERADOR DE REGLAS",
                        "LISTADO DE REGLAS","GUARDAR MODELO DIFUSO",
                        "RECUPERAR MODELO DIFUSO",
                        "CARGAR MCU CON FUNCIONES DIFUSAS",
                        "CARGAR MCU CON PROGRAMA"
                        "SALIR"};


/* DECLARACIÓN DE FUNCIONES */
```

```
float cuantizar(void);
int crear_arch(char *arch_nombre);
int abrir_arch(char *nombre);
char (*formato_hexa(float));
int cursor(int es, int dato);
void guardar(int nivel,int dato1, int dato2,int dato3);
int binario(int &dato);
int binario2(int &dato);
void lista_reglas(void);
void principal(void);
int menu(void);
void archivo(void);
void recuperar(void);
void save(int);
void saves(char *cad);
int restore(void);
char *restores(void);
void libre(void);
void mandar(int dato);
void imagen(void);
void txin_mcu(char *ins);
void mcuini(void);
int restore(void);
void con_hex(void);
void reglas(void);
void gen_reglas(void);
void anadir_v(void)
void cerrar_puerto(void);



main(int _argc  char *_argv[])
{
name=arch_nombre+80;
i=0;
do
{
 do
 {
  textcolor(WHITE);
  sel=menu();
 }
 while ((sel<1)||(sel>opcion));

 controlf=0
        switch (sel)
        {
                case 1  /* DECLARACIÓN DE VARIABLES LINGUISTICAS */

                        controle=0;
                        clrscr();

                        /* REVISAR SI EXISTE ALGUN MODELO CARGADO EN MEMORIA
                          Y ALERTAR */

                        if (v[input]!=0)
                        {
                                gotoxy(10,10);
                                textcolor(LIGHTMAGENTA
                                cprintf("SE HA DETECTADO UN MODELO CARGADO EN MEMORIA");
                                printf("\n");
                                gotoxy(10,11);
                                textcolor(LIGHTRED);
                                cprintf("---CUIDADO, SI PROCEDE EL MODELO SE PERDERµ!!!");
                                textcolor(WHITE);
                                do
                                {
                                        gotoxy(20,12)
                                        printf("Desea continuar (<S>i/<N>o)");
                                        ans=getch();
                                }
                                while((toupper(ans)!='S')&&(toupper(ans)!='N'));
                                if (toupper(ans)=='N')
                                        break;

                        }

                        /* EJECUTAR RUTINA PARA DECLARACIÓN DE VARIABLES */
```

```
                principal();
                break;
        case 6:

                /* GENERACIÓN DE REGLAS MANUALMENTE */

                reglas();
                break;
        case 7:

                 /* GENERADOR DE REGLAS-GENERACIÓN AUTOMÁTICA DE COMBINACIONES
*/

                controle=1;
                gen_reglas();
                break;
        case 8:

                /* GENERAR LISTADO DE REGLAS PARA VER EN PANTALLA */

                lista_reglas();
                break;
        case 9:

                /* REVISAR SI EXISTE MODELO DIFUSO CARGAO EN MEMORIA
                  ANTES DE GUARDAR EN DISCO */

                if (v[io]==0)
                {
                        clrscr();
                        gotoxy(10,10);
                        textcolor(LIGHTRED);
                        cprintf("---No existen variables!!!");
                        getch();
                        textcolor(WHITE);
                        break;
                }

                /* RUTINA PARA CREAR ARCHIVO MFC*/

                archivo();
                 for (io=0;io<2;io++)
                {
                        /* RUTINA PARA CONVERTIR VALORES DECIMALES A
                          HEXADECIMALES ENTRE 0 A FF */

                        cuantizar();
                        con_hex();
                }
                strcpy(arch_nombre,name);
                strcat(arch_nombre,".s");

                /* CREAR ARCHIVO S */

                if (crear_arch(arch_nombre)==0)
                        continue;
                strcpy(nombre,";Entrada\n");
                fwrite(nombre,strlen(nombre),1,out)
                for (x=0;x<v[input];x++)
                {
                        /* RUTINA PARA GUARDAR DATOS DE CADA VARIABLE */

                        guardar(0,x,fm[input][x],4);
                }
                strcpy(nombre,";Salida\n");
                fwrite(nombre,strlen(nombre),1,out)
                for (x=0;x<v[output];x++)
                {
                        guardar(1,x,fm[output][x],1);
                }

                strcpy(nombre,";Reglas\n");
                fwrite(nombre,strlen(nombre),1,out)
                for (i=0;i<l;i++)
                {
strcpy(law_hex[i],"");
                        strcat(law_hex[i],formato_hexa(law[i]));
                }
                for (j=0;j<l;j++)
                {
```

```
                fwrite("DC.B ",strlen("DC.B "),1,out);
                strcpy(nombre,law_hex[j])
                fwrite(nombre,strlen(nombre),1,out);
                fwrite("\n",strlen("\n"),1,out);
        }
        fwrite("DC.B ",strlen("DC.B "),1,out);
        strcpy(nombre,"FF");
        fwrite(nombre,strlen(nombre),1,out);
        fwrite("\n",strlen("\n"),1,out);
        strcpy(law_hex[l],"FF");
        fclose(out);
        break;
case 11:

        clrscr();

        /* REVISAR EXISTENCIA DE MODELO DIFUSO EN MEMORIA */

        if (v[io]==0)
        {
                clrscr();
                gotoxy(10,10);
                textcolor(LIGHTRED);
                cprintf("—No existen variables!!!");
                getch();
                textcolor(WHITE);
                break;
        }

        for (io=0,io<2,io++)
        {
                cuantizar();
                con_hex();
        }

        /* CARGAR MODELO DIFUSO A MICROCONTROLADOR */
        /* INICIALIZAR EL PUERTO SERIE*/

        mcuini();

        /* RUTINA PARA MANDAR DATOS */

        mandar(' ');
        mandar(13);

        /* RUTINA PARA MANDAR INSTRUCCIONES AL MICROCONTROLADOR */

        txin_mcu("BF C200 C300 FF")
        txin_mcu("MM C200");
        w=0;
        for(i=0,i<2,i++)
        {
                if(i==1)
                {
                        mandar(13)
                        txin_mcu("MM C280");
                }
                for (w=0,w<v[i],w++)
                {
                        for(x=0,x<fm[i][w],x++)
                                for(z=0,z<4,z++)
                                {
                                        aux=strlen(hexa_uc[i][w][x][z]);
                                        for(j=0,j<aux,j++)
                                        {
                                                mandar(hexa_uc[i][w][x][z][j]);
                                        }

                                        if (i==1)
                                                z=5;
                                        mandar(' ');
                                }
                }
        }

        mandar(13);
        txin_mcu("MM C290");


        for (x=0,x<l,x++)
```

```
        {
                for(j=0;j<strlen(law_hex[x]);j++)
                {
                        mandar(law_hex[x][j]);
                }
                mandar(' ');
        }
        mandar('F');
        mandar('F');
        mandar(13);
        break;
case 12:

        /* RUTINA PARA VERIFICAR QUE MICROCONTROLADOR ESTA LISTO PARA
          RECIBIR */

        libre();
        txin_mcu("load t");
        libre();
        cerrar_puerto();
        system("COMMU.BAT");
        break;
case 10

        /* RUTINA PARA RECUPERAR MODELO DIFUSO DESDE ARCHIVO */

        recuperar();
        break;
case 2
        if (v[io]==0)
        {
                clrscr();
                gotoxy(10,10);
                textcolor(LIGHTRED)

                cprintf("---No existen variables!!!");
                getch();
                textcolor(WHITE);
                break;
        }
        controlf=1;

        /* GENERAR ARCHIVO TEMPORAL PARA PODER PASAR DATOS AL
          PROGRAMA COMPLEMENTARIO MCFUZ2 */

        strcpy(arch_nombre,"mcfuzzy tmp")
        archivo();
        strcpy(arch_name,arch_nombre);
        itoa(sel,answ,10);
        spawnlp(P_WAIT,"mcfuz2.exe"," ",answ,arch_name,NULL);
        strcpy(arch_nombre,"mcfuzzy tmp");
        controlf=1;
        recuperar();
        break;

case 3
        if (v[io]==0)
        {
                clrscr();
                gotoxy(10,10);
                textcolor(LIGHTRED)
                cprintf("---No existen variables!!!");
                getch();
                textcolor(WHITE);
                break;
        }
        controlf=1;
        strcpy(arch_nombre,"mcfuzzy tmp");
        archivo();
        strcpy(arch_name,arch_nombre);
        itoa(sel,answ,10);
        spawnlp(P_WAIT,"mcfuz2.exe"," ",answ,arch_name,NULL);
        strcpy(arch_nombre,"mcfuzzy tmp");
        controlf=1;
        recuperar();
        break;

case 5
        if (v[io]==0)
        {
```

```
                        clrscr();
                        gotoxy(10,10);
                        textcolor(LIGHTRED);
                        cprintf("---No existen variables!!!");
                        getch();
                        textcolor(WHITE);
                        break;
                }
                controlf=1;
                strcpy(arch_nombre,"mcfuzzy.tmp");
                archivo();
                strcpy(arch_name,arch_nombre);
                itoa(sel,answ,10);
                spawnlp(P_WAIT,"mcfuz2.exe"," ",answ,arch_name,NULL);
                strcpy(arch_nombre,"mcfuzzy.tmp");
                controlf=1;
                recuperar();
                break;

        case 4:
                if (v[io]==0)
                {
                        clrscr();
                        gotoxy(10,10);
                        textcolor(LIGHTRED);
                        cprintf("---No existen variables!!!");
                        getch();
                        textcolor(WHITE);
                        break;
                }
                controlf=1;
                strcpy(arch_nombre,"mcfuzzy.tmp");
                archivo();
                strcpy(arch_name,arch_nombre);
                itoa(sel,answ,10);
                spawnlp(P_WAIT,"mcfuz2.exe"," ",answ,arch_name,NULL);
                strcpy(arch_nombre,"mcfuzzy.tmp");
                controlf=1;
                recuperar();
                break;


        )
  }
  while(sel!=opcion);
}

int menu(void)
{

/* RUTINA PARA GENERAR MENU */

int dato;
clrscr();
gotoxy(35,2);
textcolor(LIGHTMAGENTA);
cprintf("MCFUZLOG");
printf("\n \n");
textcolor(LIGHTRED);
gotoxy(32,wherey());
cprintf("MENU PRINCIPAL");
textcolor(LIGHTBLUE);
gotoxy(15,6);
for (x=0;x<opcion;x++)
{
    cprintf("%d- ",x+1);
    puts(opciones[x]);
    gotoxy(15,wherey());
}
textcolor(LIGHTGREEN);
printf("Seleccion: ");
scanf("%d",&dato);
textcolor(WHITE);
return(dato);
}



void principal(void)
```

```
{
/* RUTINA PARA INTRODUCCIÓN DE VARIABLES LINGÜÍSTICAS */

io=0;
do
{
        do
        {
                clrscr();
                textcolor(LIGHTRED);
                gotoxy(15,10);
                cprintf("El programa esta diseñado para manejar un máximo de");
                gotoxy(20,11);
                if (io==0)
                        cprintf("cuatro variables l¡ng□¡sticas de entrada. ");
                if (io==1)
                        cprintf("dos variables l¡ng□¡sticas de salida. ");
                textcolor(YELLOW);
                gotoxy(10,12);
                cprintf("Cuantas variables lingüísticas de %s desea controlar: ",direccion);
                scanf("%f",&aux);
                v[io]=aux;
                if ((io==1)&&(aux>2))
                        v[io]=-1;
        }
        while ((v[io]<0)||(v[io]>vec1));
        textcolor(WHITE);

        for (j=0;j<v[io];j++)
        {
                clrscr();
                fflush(stdin);
                printf("Nombre de la variable %i: ",j+1);
                gets(v_nombre[io][j]);
                printf("Unidades: ");
                gets(unidades[io][j]);
                printf("Limite inferior: ");
                scanf("%f",&aux);
                lim_inf[io][j]=aux;
                do
                {
                        printf("Limite superior: ");
                        scanf("%f",&aux);
                }
                while(aux<lim_inf[io][j]);
                lim_sup[io][j]=aux;
                do
                {
                        textcolor(LIGHTRED);
                        cprintf("El sistema solo permite un máximo de ocho funciones de membresia");
                        printf("\n");
                        textcolor(YELLOW);
                        cprintf("Cuantas funciones de membresia: ");
                        textcolor(WHITE);
                        scanf("%f",&aux);
                        fm[io][j]=aux;
                }
                while ((fm[io][j]<0)||(fm[io][j]>vec2));
                clrscr();
                for (i=0;i<fm[io][j];i++)
                {
                        clrscr();
                        gotoxy(20,wherey());
                        puts(v_nombre[io][j]);
                        gotoxy(20,wherey());
                        printf("Rango: %3.2f - %3.2f %s \n",lim_inf[io][j],lim_sup[io][j],unidades[io][j]);
                        gotoxy(10,wherey());
                        fflush(stdin);
                        printf("Nombre de función de membresia %i: ",i+1);
                        gets(nombre);
                        strcpy(fm_nombre[io][j][i],nombre);
                        punto_anterior=lim_inf[io][j];
                        point=3;
                        if (io==1)
                                point=0;
                        for (z=0;z<=point;z++)
                        {
                                do
                                {
```

```
                                        printf("Punto %i: ",z+1);
                                        scanf("%f",&aux);
                                }
                                while ((aux<lim_inf[io][j])||(aux>lim_sup[io][j])||(aux<punto_anterior));
                                punto_anterior=aux;
                                punto[io][j][i][z]=aux;
                        }
                }
        }


        io++;
        strcpy(direccion,"salida");
        }
        while(io<=1);


}

void reglas(void)
{

/* RUTINA PARA GENERAR REGLAS-USUARIO ARMA REGLA POR REGLA */

        clrscr();
        if (v[input]==0)
        {
                textcolor(LIGHTRED);
                gotoxy(10,10);
                cprintf("—No han sido declaradas variables linguisticas'!!");
                getch();
                textcolor(WHITE);
                return;
        }

        printf("Variables de \n Entrada ");
        for (j=0;j<=v[input];j++)
                printf ("\n %s",v_nombre[input][j]);
        printf("\n");
        printf("Variables de \n Salida ")
        for (j=0;j<=v[output];j++)
                printf("\n %s",v_nombre[output][j]);
        k=0;
        n_or=0;
        i=0;
        gotoxy (20,2);
        printf(" Antecedente");
        gotoxy (20,4);
        printf(" ES");
        gotoxy (20,6);
        printf(" Concecuente");
        gotoxy (20,8);
        printf(" ES");
        do

                {
                law[l]=0;
                p=0;
                strcpy(regla,"");
                control=0;
                do
                {
                        j=-1;
                        gotoxy (20,3);
                        printf("                ");
                        do
                        {
                        do
                        {
                                gotoxy (20,3);
                                aux=j;
                                j=cursor(v[input],j);
                                if ((j!=enter)&&(j!=terminar)&&(j!=operadory))
                                {
                                        printf("                ");
                                        gotoxy(20,3);
                                        printf(" %s",v_nombre[input][j]);
                                        aux=j;
                                }
```

```
        }
        while ((j!=enter)&&(j!=terminar));
        }
        while ((aux==-1)&&(j!=terminar));

        if(j!=terminar)
        {
                j=aux;
                strcat(regla,strupr(v_nombre[input][j]));
                strcat(regla," ES ");
                for (i=0;i<8;i++)
                    b[i]=0;
                do
                {
                        p=binario(j);
                        if (p==9)
                        {
                                p=0
                                bin=0
                        }
                        else
                                bin=1
                        b[p]=bin;
                }
                while (j>0);
                law[l]=b[1]*16+b[0]*8;


        }
        else
        {
                control=1;
                continue;
        }
        j=aux;
        gotoxy(1,22);
        clreol();
        gotoxy(1,22);
        printf("Regla %d :",n_or);
        puts(regla);
        gotoxy (40,2);
        printf("Funciones de Membresia \n");
        gotoxy(45,wherey());
        for (i=0;i<=fm[input][j];i++)
        {
                printf("                \n");
                gotoxy(45, (wherey()));

        }
        gotoxy(45,3);
        for (i=0;i<=fm[input][j];i++)
        {

                printf("%s \n",fm_nombre[input][j][i]);
                gotoxy(45, wherey());
        }
        x=-1;
        gotoxy (20,5);
        printf("              ");
        do
        {
        do
        {
                gotoxy(20,5);
                aux=x;
                x=cursor(fm[input][j],x);
                if ((x!=enter)&&(x!=operadory)&&(x!=terminar))
                {
                printf("                ");
                gotoxy(20,5);
                printf(" %s ",fm_nombre[input][j][x]);
                }
        }
        while ((x!=enter)&&(x!=operadory));
        }
        while ((aux==-1));
z=x;
x=aux;
for (i=0;i<8;i++)
```

```
        b[i]=0;
    if (z!=terminar)
            {
                    strcat(regla,strupr(fm_nombre[input][j][x]));                    if (z==operadory)
                            strcat(regla," Y ");

                    do
                    {

                            p=binario(x);
                            if (p==9)
                            {
                                    p=0;
                                    bin=0;
                            }
                            else
                                    bin=1;
                            b[p]=bin;
                    }
                    while (x>0);
                    law[l]=law[l]+b[2]*4+b[1]*2+b[0];
                    l++;

            }
    else
    {
            control=1;
            continue;
    }

    gotoxy(1,22);
    clreol();
    gotoxy(1,22);
    printf("Regla %d  ",n_or);
    puts(regla);
    }

    while ((z!=enter)&&(control==0));
    if (control==1)
            continue;
    z=-1;
    strcat(regla," ENTONCES ");
    gotoxy(1,22);
    clreol();
    gotoxy(1,22);
    printf("Regla %d  ",n_or);
    puts(regla);
    law[l]=0;
    gotoxy (20,7);
    printf("                ");
    do
    {
    do
    {

            gotoxy (20,7);
            aux=z;
            z=cursor(v[output],z);
            if ((z!=enter)&&(z!=terminar)&&(z!=operadory))
            {
                    printf("                ");
                    gotoxy(20,7);
                    printf(" %s",v_nombre[output][z]);
            }
    }
    while ((z!=enter));
    }
    while ((aux==-1));
    if (z!=terminar)
    {
            z=aux;
            strcat(regla,strupr(v_nombre[output][z]));
for (i=0,i<8,i++)
                    b[i]=0;
            do
                    {
                            p=binario(z);
                            if (p==9)
                            {
                                    p=0
```

```
                                        bin=0;
                                }
                                else
                                        bin=1;
                                b[p]=bin;
                        }
                        while (z>0);
                        law[l]=128+b[1]*16+b[0]*8;

}
else
{
        control=1;
        continue;
}
z=aux;
gotoxy(1,22);
clreol();
gotoxy(1,22);
printf("Regla %d :",n_or);
puts(regla);
strcat(regla, " ES ");
gotoxy (40,2);
printf("Funciones de Membresia:\n");
gotoxy(45,3);
for (i=0;i<vector2;i++)
{
        printf("              \n");
        gotoxy(45, (wherey()));
}
gotoxy (45,3);
for (i=0,i<fm[output][z],i++)
{
        gotoxy(45, (wherey()));
        printf("%s \n",fm_nombre[output][z][i]);
        gotoxy(45, (wherey()));
}
w=-1;
gotoxy (20,9);
printf ("                ");
do
{
do
{
    gotoxy (20,9);
        aux=w;
        w=cursor(fm[output][z],w);
        if ((w!=enter)&&(w!=terminar)&&(w!=operadory))
        {
                printf("                ");
                gotoxy(20,9);
                printf(" %s ",fm_nombre[output][z][w]);
        }
}
while ((w!=enter));
}
while((aux==-1)&&(w!=terminar));
x=w;
w=aux;
if (w!=terminar)
{
        strcat(regla, strupr(fm_nombre[output][z][w]));
        for (i=0;i<8;i++)
                b[i]=0;
        do
                {
                        p=binario(w);
                        if (p==9)
                        {
                                p=0;
                                bin=0;
                        }
                        else
                                bin=1;
                        b[p]=bin;
                }
                while (w>0);
                law[l]=law[l]+b[2]*4+b[1]*2+b[0];
                l++;
```

```
                    }
                    else
                    {
                            control=1;
                            continue;
                    }
                    gotoxy(1,22);
                    clreol();

                    gotoxy(1,22);
                    printf("Regla %d :",n_or);
                    puts(regla);
                    w=aux;
                    }
                    while ((j!=terminar)&&(control==0));
                    clrscr();

}



void guardar(int nivel,int dato1, int dato2,int dato3)
{

/* RUTINA PARA GUARDAR DATO EN ARCHIVO */

 char *numero
        for(i=0;i<dato2;i++)
                {
                for(z=0;z<dato3;z++)
                        {

                                fwrite("DC B",strlen("DC B "),1,out);
                                strcpy(numero,hexa_uc[nivel][dato1][i][z]);
                                fwrite(numero,strlen(numero),1,out);
                                fwrite("\n",strlen("\n"),1,out);
                        }
                }



}

int crear_arch(char *nombre)
{

/* RUTINA PARA CREAR ARCHIVOS */

  clrscr();
  if ((out = fopen(nombre, "wt"))
     == NULL)
  {
    clrscr();
    gotoxy(10,10);
    textcolor(LIGHTRED);
    cprintf("No se pudo crear archivo, presione cualquier tecla para");
    gotoxy(25,11);
    cprintf("para continuar");

    getch();
    textcolor(WHITE);
    return(0);
   }
return(1);
}

int abrir_arch(char *nombre)
{

/* RUTINA PARA ABRIR ARCHIVOS */

  clrscr();
  if ((in= fopen(nombre, "r+"))
     == NULL)
  {
    clrscr();
    gotoxy(10,10);
    textcolor(LIGHTRED);
    cprintf("No se pudo crear archivo, presione cualquier tecla para")
```

```
    gotoxy(25,11);
    cprintf("para continuar");

    getch();
    textcolor(WHITE);
    return(0);
   }
return(1);
}


int binario(int &dato)
{

        double n=0, resp;
        if (dato>1)
        {
                do
                {
                        resp=pow(2,n);
                        n++;
                }
                while(resp<dato);
                n--;
                dato=dato-pow(2,n);
                return(n);
        }
        else
        {
        if (dato==1)
        {
                dato=0;
                return(0);

        }
        else

            return(9);
        }
}

int binario2(int &dato)
{
        double n=-1, resp;
        if (dato>1)
        {
                do
                {
                        n++;
                        resp=pow(2,n);

                }
                while(resp<=dato);
                n--;
                dato=dato-pow(2,n);
                return(n);
        }
        else
        {
        if (dato==1)
        {
                dato=0;
                return(0);

        }
        else

            return(9);
        }
}

void lista_reglas(void)
{

/* RUTINA PARA GENERAR LISTADO DE REGLAS QUE CONTIENE EL MODELO DIFUSO */

        unsigned int r=0,x,b_ant=0,n=0,W=0,y=1;
        char ans;
        clrscr()
        if(i==0)
```

```
  {
        gotoxy(5,10);
        textcolor(LIGHTRED);
        cprintf("No hay reglas!!!");
        textcolor(WHITE);
getch();
        return;
  }
  strcpy(regla,"");
  clrscr();
  printf("REGLAS DIFUSAS\n");
  textcolor(YELLOW);
  cprintf("REGLA %d : ",y);
  printf("\n");
  textcolor(WHITE);
   do
   {


                do
                {

                for (x=0;x<8;x++)
                        b[x]=0;
                j=law[r];
                        do
                        {
                                p=binario2(j);
                                if (p==9)
                                {
                                        p=0;
                                        bin=0
                                }
                                else
                                {

                                        bin=1
                                }
                                b[p]=bin;
                        }
                        while (j>0);
                j=b[0]+b[1]*2+b[2]*4;
                i=b[3]+b[4]*2;
                if ((b[7]==0)&&(b_ant==1))
                {
                        textcolor(YELLOW);
                        cprintf("REGLA %d  ",++y);
                        printf("\n");

                        textcolor(WHITE);
                }

                if (r!=0)
                        if (((b[7]==0)&&(b_ant==0))||((b[7]==1)&&(b_ant==1)))
                        {
                                puts(" Y ");

                        }
                 if ((b[7]==1)&&(b_ant==0))
                        puts(" ENTONCES ");
                 if (((b[7]==0)&&(b_ant==0))||((b[7]==0)&&(b_ant==1)))
                 {
                        strcpy(regla,"");
                        strcpy(regla,v_nombre[input][i]);
                        strcat(regla," ES ");
                        strcat(regla,fm_nombre[input][i][j]);
                        puts(regla);


                 }
                 if ((b[7]==1)&&(b_ant==0))
                 {
                        strcpy(regla,"");
                        strcpy(regla,v_nombre[output][i]);
                        strcat(regla," ES ");
                        strcat(regla,fm_nombre[output][i][j])
                        puts(regla);

                 }
```

```
                b_ant=b[7];
                r++;
                }
                while ((wherey()<18)&&(r<l));
                if(r<l)
                {
                        textcolor(GREEN);
                        cprintf("(SIGUE....)");
                        printf("\n");
                        textcolor(WHITE);
                }

        pnntf("Por favor presionar cualquier tecla para continuar (T=terminar y B=borrar)\n");
        ans=getch();
        if (toupper(ans)=='T')
                return;
        if (toupper(ans)=='B')
        {
                textcolor(LIGHTRED);
                k=0;
                cpnntf("Numero de la regla que desea borrar (Solo pueden borrarse las reglas anteriores hasta la £ltima desplegada):");
                scanf("%d",&k);
                textcolor(WHITE);
                i=1;
                x=0;
                do
                {
                 if ((law[i]>127)&&(law[i+1]<127))
                    x++;
                 if (x==k)
    break;
                 i++;
                }
                while((i<r)&&(x<=k));

                if (x==k)
                {

                        do
                        {

                                law[i]=255;
                                i--;
                                if (i<=0)
                  break;
                                else
                                        w=law[i-1];
                        }
                        while((law[i]>127)||(w<127));
                        law[i]=255;
                }

        }
        clrscr();
        }
        while (r<l);
        for(x=0;x<l;x++)
                if (law[x]==255)
                {
                        n++;
                        w=x;
                        do
                        {
                                law[w]=law[w+1];
                                w++;
                        }
                        while(w<=l);
                        x--;
                }
l=l-n;
}

void archivo(void)
{

/* RUTINA PARA ALMACENAR LOS DATOS EN ARCHIVO MFC */

        int aux;
        char *cad;
```

```
        int dec,sign;
        cad=arch_nombre+20;
        clrscr();
        if (controlf!=1)
        {
                textcolor(GREEN);
                gotoxy(20,10);
                cprintf("Se almacenar  el modelo difuso");
                textcolor(LIGHTCYAN);
                gotoxy(10,11);

                cprintf("Por favor proveer un nombre para el archivo: ");
                textcolor(WHITE);
                gotoxy(20,12);
                fflush(stdin);
                gets(arch_nombre);
                strcpy(name,arch_nombre);
                strcat(arch_nombre,".mfc");
        }
        if (crear_arch(arch_nombre)==0)
                return;

        for (j=0;j<2;j++)
        {
                save(v[j]);
                for (x=0;x<v[j];x++)
                {
                        saves(v_nombre[j][x]);
                        saves(unidades[j][x]);
                        save(lim_inf[j][x]);
                        save(digit*(lim_inf[j][x]-(int)lim_inf[j][x]));
                        save(lim_sup[j][x]);
                        save(digit*(lim_sup[j][x]-(int)lim_sup[j][x]));
                        save(fm[j][x]);
                        for (w=0;w<fm[j][x];w++)
                        {
                                saves(fm_nombre[j][x][w])
                                for(z=0;z<4;z++)
                                {
                                        save(punto[j][x][w][z]);
                                        save(digit*(punto[j][x][w][z]-(int)punto[j][x][w][z]));
                                        if (j==1)
                                                z=5
                                }
                        }
                }
        }
        save(l);
        for (x=0;x<l;x++)
                save(law[x]);
    fclose(out);
}

void recuperar(void)
{

/* RUTINA PARA RECUPERAR DATOS DESDE ARCHIVO */

        float aux;
        char *cad;
        int dec,sign;
        if (controlf==0)
        {
                clrscr();
                textcolor(GREEN);
                gotoxy(20,10);
                cprintf("Se recuperar  el modelo difuso")
                textcolor(LIGHTCYAN);
                gotoxy(10,11);
                cprintf("Por favor proveer un nombre para el archivo ");
                textcolor(WHITE);
                gotoxy(20,12);
                fflush(stdin);
                gets(arch_nombre);
                strcat(arch_nombre,".mfc");
        }
        if (abrir_arch(arch_nombre)==0)
                return;
     fseek(in,SEEK_SET,0);
        aux=digit;
```

```
        for (j=0;j<2;j++)
        {
                v[j]=restore();
                for (x=0;x<v[j];x++)
                {
                        strcpy(v_nombre[j][x],restores());
                        strcpy(unidades[j][x],restores());
                        lim_inf[j][x]=restore();
                        lim_inf[j][x]+=(restore()/aux);
                        lim_sup[j][x]=restore();
                        lim_sup[j][x]+=(restore()/aux);
                        fm[j][x]=restore();
                        for (w=0;w<fm[j][x];w++)
                        {
                                strcpy(fm_nombre[j][x][w],restores());
                                for(z=0;z<4;z++)
                                {
                                        punto[j][x][w][z]=restore();
                                        punto[j][x][w][z]+=(restore()/digit);
                                        if (j==1)
                                                z=5;
                                }
                        }
                }
        }
        l=restore();
        for (x=0;x<l;x++)
        {
                law[x]=restore();
                strcpy(law_hex[x],"");
                strcat(law_hex[x],formato_hexa(law[x]));
        }

fclose(in);
}

int restore(void)
{

/* RUTINA PARA RECUPERAR DATO NUMERICO DESDE ARCHIVO */

        char *cad;
        char *dato;
        cad=arch_nombre+20;
        dato=arch_nombre+50;
        strcpy(dato,"");
        strcpy(cad,"");
        do
        {
                fread(dato,sizeof(char),1,in);
                if (dato[0]!=',')
                        strcat(cad,dato);
        }
        while(dato[0]!=',');
        return(atoi(cad));
}
char *restores(void)
{

/* RUTINA PARA RECUPERAR CADENA DE CARACTERES DESDE ARCHIVO */

        char *cad;
        char *dato;
        cad=arch_nombre+20;
        dato=arch_nombre+50;
        strcpy(dato,"");
        strcpy(cad,"");
        do
        {
                fread(dato,sizeof(char),1,in);
                if (dato[0]!=',')
                        strcat(cad,dato);
        }
        while(dato[0]!=',');


return(cad);
}
```

```
void save(int dato)
{
/* GUARDAR DATO NUMÉRICO EN ARCHIVO */
        char *cad;
    cad=arch_nombre+20;
        itoa(dato,cad,10);
        fwrite(cad,strlen(cad),1,out);
        fwrite(",",1,1,out);
}
void saves(char *cad)
{
/* GUARDAR CADENA DE CARACTERES EN ARCHIVO */
        fwrite(cad,strlen(cad),1,out);
        fwrite(",",1,1,out);
}

void gen_reglas(void)
{
/* GENERADOR DE COMBINACIONES PARA ANTECEDENTES DE REGLAS */
int dato,dato1,incluir[4]={0,0,0,0},
int w=0,aux=0,p y,l_ant,b_ant=0,reglas[limite][4],c=0,
clrscr();
gotoxy(30,1),
textcolor(LIGHTCYAN),
cprintf("Generador de Reglas"),
textcolor(WHITE)
printf("\n"),
dato=0;
if (v[input]==0)
{
        textcolor(LIGHTRED),
        gotoxy(10,10),
        cprintf("---No han sido declaradas variables linguisticas!!!"),
        getch(),
        textcolor(WHITE),
        return,
}
do
{
        gotoxy(5,6),
        printf("Cuantas variables a combinar (El Sistema tiene %d variables de entrada) ",v[0]),
        scanf("%d",&dato),
        if (dato>(v[input]))
                dato=-1,

}
while((dato<0)),
gotoxy(5,8),
printf("Cual Salida(El Sistema tiene %d salidas posibles) ",v[1]),
w=0,
do
{
        if (dato1==1)
                break,
        aux=w,
        w=cursor(v[output],w),
        if (w<20)
        {
                gotoxy(60,8),
                puts(v_nombre[output][w]),
    }
}
while(w!=enter),
dato1=aux;;
clrscr();
for(x=0;x<dato,x++)
{
        if (dato==v[input])
                break,
        w=0,
        gotoxy(20,10),
        printf("Variable de entrada %d ",x+1),
```

```
                do
                {
                        aux=w;
                        w=cursor(v[input],w);
                        if (w<20)
                        {
                        gotoxy(50,10);
                        printf("                ");
                        gotoxy(50,10);
                        puts(v_nombre[0][w]);-
                        }
                }
                while(w!=enter);
                incluir[aux]=1;
        }
        if (dato==v[input])
                for(x=0;x<dato;x++)
                        incluir[x]=1;
        l_ant=l;
        z=0;
        y=0;
        w=0;
        p=0;
        do
        {
                do
                {
                        do
                        {
                                do
                                {
                                        if (incluir[0]==1)
                                        {

                                                law[l]=p
                                                l++;
                                        }

                                        if (incluir[1]==1)
                                        {

                                                law[l]=8+(w)
                                                l++;
                                        }
                                        if (incluir[2]==1)
                                        {
                                                law[l]=16+(y)
                                                l++;
                                        }
                                        if (incluir[3]==1)
                                        {

                                                law[l]=24+y;
                                                l++;
                                        }
                                        law[l]=250;
                                        l++;
                                        z++;
                                }
                                while((z<fm[input][3])&&(incluir[3]==1));
                                z=0;
                                y++;
                        }
                        while((y<fm[input][2])&&(incluir[2]==1));
                        y=0;
                        w++;
                }
                while((w<fm[input][1])&&(incluir[1]==1));
                w=0;
                p++;
        }
        while((p<fm[input][0])&&(incluir[0]==1));

        c=l_ant;

        b_ant=1;
        y=0;
        clrscr();
        do
        {
```

```
for (x=0;x<8;x++)
        b[x]=0;
j=law[c];
do
{
        p=binario2(j);
        if (p==9)
        {
                p=0;
                bin=0;
        }
        else
        {
                bin=1;
        }
        b[p]=bin;
}
while (j>0);
j=b[0]+b[1]*2+b[2]*4;
i=b[3]+b[4]*2;
if ((b[7]==0)&&(b_ant==1))
{
        textcolor(YELLOW);
        cprintf("REGlA NUEVA %d  ",++y);
        printf("\n");
        textcolor(WHITE);
}
if(c!=0)
        if (((b[7]==0)&&(b_ant==0))||((b[7]==1)&&(b_ant==1)))
        {
                puts(" Y ");
        }
if (law[c]<127)
{
        strcpy(regla,"");
        strcpy(regla,v_nombre[input][i]);
        strcat(regla," ES ");
        strcat(regla,fm_nombre[input][i][j]);
        puts(regla);
}
if (law[c]==250)
{
        w=0;
        strcpy(regla,"");
        printf( "ENTONCES ");
        strcpy(regla,v_nombre[output][dato1])
        strcat(regla," ES ");
        puts(regla);
        w=-1;
        do
        {
                aux=w;
                w=cursor(fm[output][dato1],w);
                textcolor(LIGHTCYAN);
                if (w<20)
                {

                        printf("                              \n ");
                        gotoxy(30,wherey()-1)
                        cputs(fm_nombre[output][dato1][w]);
                        gotoxy(1,wherey());
                }
                textcolor(WHITE);
        }
        while(w!=enter);
        printf("\n");
        w=aux;
        if (dato1==0)
                law[c]=128+w;
        if (dato1==1)
                law[c]=128+8+w;


}

c++;
b_ant=b[7];
}
while(c<l);
```

```
}

void anadir_v(void)
{
/* AÑADIR VARIABLE LINGÜÍSTICA CON TODOS LOS DATOS RESPECTIVOS:
   RANGOS, FUNCIONES DE MEMBRESÍAS */

char ans;
io=0;
clrscr();
if ((v[input]==0)&&(v[output]==0))
{
        textcolor(LIGHTRED);
        gotoxy(2,10);
        cprintf("---No existen variables declaradas. Por favor utilice la opción uno!!!");
        getch();
        return;
}
clrscr();
do
{
        gotoxy(10,10);
        printf("Desea añadir entradas o salidas (<E>ntrada / <S>alida)");
    fflush(stdin);
        ans=getche();
}
while((toupper(ans)!='E')&&(toupper(ans)!='E'));
if (toupper(ans)=='E')
        io=0
else
        io=1;
do
{
                clrscr();
                textcolor(LIGHTRED);
                gotoxy(15,10);
                cprintf("El programa esta diseñado para manejar un máximo de");
                gotoxy(20,11);
                if (io==0)
                {
                        cprintf("cuatro variables linguisticas de entrada \n ");
                        getch();
                        clrscr();
                        textcolor(MAGENTA);
                        cprintf("El sistema tiene la siguiente configuración de variables de entrada:\n");
                        printf("\n");
                        textcolor(LIGHTBLUE);
                        cprintf("    Variable Linguistica        Funciones de membresia\n");

                        for(x=0;x<v[io];x++)
                        {
                                printf("\n");
                                textcolor(YELLOW);
                                cprintf("%s",v_nombre[io][x]);
                                gotoxy(30,wherey());
                                for(z=0;z<fm[io][x];z++)
                                {

                                        textcolor(LIGHTGREEN);
                                        cprintf("%s - ",fm_nombre[io][x][z]);
                                }
                        }
                        textcolor(WHITE);
                }
                if (io==1)
                {
                        cprintf("dos variables linguisticas de salida  ");
                        textcolor(MAGENTA);
                        cprintf("El sistema tiene la siguiente configuración de variables de salida:\n");
                        textcolor(LIGHTBLUE);
                        cprintf("    Variable Linguistica        Funciones de membresia\n");
                        for(x=0;x<v[io];x++)
                        {
                                textcolor(YELLOW);
                                cprintf("%s",v_nombre[x])
                                for(z=0;z<fm[io][x];z++)
                                {
                                        textcolor(LIGHTGREEN);
```

```
                                        cprintf("%s - ",fm_nombre[io][z]);
                                }
                        }
                        textcolor(WHITE);
                }
                textcolor(YELLOW);
                gotoxy(10,12);
                cprintf("Cuantas variables lingüisticas de %s desea agregar: ");
                fflush(stdin);
                scanf("%f",&aux);
                if ((aux+v[io]>5)||(aux<0))
                        aux=-1;

        }
        while (aux==-1);
        v[io]+=aux;
        textcolor(WHITE);
        for (j=(v[io]-aux);j<v[io];j++)
        {
                clrscr();
                fflush(stdin);
                printf("Nombre de la variable %i: ",j+1);
                gets(v_nombre[io][j]);
                printf("Unidades: ");
                gets(unidades[io][j]);
                printf("Limite inferior ");
                scanf("%f",&aux);
                lim_inf[io][j]=aux;
                do
                {       fflush(stdin);
                        printf("Limite superior ");
                        scanf("%f",&aux);
                }
                while(aux<lim_inf[io][j]);
                lim_sup[io][j]=aux;

                do
                {
                        textcolor(LIGHTRED);
                        cprintf("El sistema solo permite un máximo de ocho funciones de membresia");
                        printf("\n");
                        textcolor(YELLOW);
                        cprintf("Cuantas funciones de membresia ");
                        textcolor(WHITE);
                        scanf("%f",&aux);
                        fm[io][j]=aux;
                }
                while ((fm[io][j]<0)||(fm[io][j]>vec2));
                clrscr();
                for (i=0;i<fm[io][j];i++)
                {
                        clrscr();
                        gotoxy(20,wherey());
                        puts(v_nombre[io][j]);
                        gotoxy(20,wherey());
                        printf("Rango  %3.2f - %3.2f %s \n",lim_inf[io][j],lim_sup[io][j],unidades[io][j]);
                        gotoxy(10,wherey());
                        fflush(stdin);
                        printf("Nombre de función de membresia %i ",i+1);
                        gets(nombre);
                        strcpy(fm_nombre[io][j][i],nombre);
                        punto_anterior=lim_inf[io][j];
                        point=3;
                        if (io==1)
                                point=0;
                        for (z=0;z<=point;z++)
                        {
                                do
                                {
                                        fflush(stdin);
                                        printf("Punto %i ",z+1);
                                        scanf("%f",&aux);
                                }
                                while ((aux<lim_inf[io][j])||(aux>lim_sup[io][j])||(aux<punto_anterior));
                                punto_anterior=aux;
                                punto[io][j][i][z]=aux;
                        }
                }
        }
```

```
        io++;

}

void mcuini(void)
{
/* INICIALIZAR PUERTO SERIE */
        asm{
                mov ax,00F3h
                mov dx,0000
                int 14h
          }
mandar(13);

}

void txin_mcu(char *ins)
{
int i=0,j=0,longins;
longins=strlen(ins);
for (j=0;j<longins;j++)
{
        mandar(ins[j]);

}
mandar(13);
libre();
}



void mandar(int dato)
{
/* RUTINA PARA ENVIAR DATOS A MICROCONTROLADOR */

libre();
        asm{
                mov ax,dato
                mov ah,01
                int 14h

          }

}

void libre(void)
{
/* REVISAR SI MICROCONTROLADOR ESTA LISTO PARA RECIBIR DATO */

int dato;

                do
                {
                        asm{
                                mov dx,0;
                                mov ah,02
                                int 14h
                            /*  mov ah,00
                                int 14h
                                */
                                mov dx,0
                                mov dl,al
                                mov ah,06
                                int 21h
                                mov ah,00



                        }

                }
                while (_AX!=0);
```

```
}

float cuantizar(void)
{

/* PASAR DATOS DEL RANGO DE VARIABLE A VALORES ENTRE 0 A 256 Y CONVERTIRLOS A
   FORMATO HEXADECIMAL */

        for (j=0;j<v[io];j++)
        {
                portadora=0;
                paso_cuan[io][j]=(fac_mul*((lim_sup[io][j]-lim_inf[io][j]))/escala);
                if (lim_inf[io][j]==0)
                        portadora=0;
                else
                        portadora=(-1)*lim_inf[io][j];
                for (i=0;i<fm[io][j];i++)
                {
                        for (z=0;z<4;z++)
                        {
                                punto[io][j][i][z]+=portadora;
                                hexa[io][j][i][z]=((fac_mul*punto[io][j][i][z]))/paso_cuan[io][j];

                        }
                        if (io==0)
                        {
                                if ((hexa[io][j][i][1]-hexa[io][j][i][0])==0)
                                        hexa[io][j][i][1]=0
                                else
                                        hexa[io][j][i][1]=256/(hexa[io][j][i][1]-hexa[io][j][i][0]);
                                if ((hexa[io][j][i][3]-hexa[io][j][i][2])==0)
                                        hexa[io][j][i][3]=0
                                else
                                        hexa[io][j][i][3]=256/(hexa[io][j][i][3]-hexa[io][j][i][2]);
                        }
                }
        }

return(0)
}

void con_hex(void)
{
for (j=0;j<v[io];j++)
                for (i=0;i<fm[io][j];i++)
                        for (z=0;z<4;z++)
                        {
                                strcpy(hexa_uc[io][j][i][z],"");
                                strcat(hexa_uc[io][j][i][z],formato_hexa(hexa[io][j][i][z]));
                                if (strcmp(hexa_uc[io][j][i][z],"100")==0)
                          strcpy(hexa_uc[io][j][i][z],"FF");
                                printf("%s ",hexa_uc[io][j][i][z])
                        }
}

char (*formato_hexa(float dato))
{

        div_t resultado;
        char hexa[10]="";
        char *conv_hexa

        conv_hexa=arch_nombre+20;

        do
        {
                if (dato<10)
                {
                        resultado.rem=dato;
                        resultado.quot=0;
                }
                else
                        resultado=div(dato,16)
                if (dato==5.0)
        resultado.rem=5;
                        strcpy(conv_hexa,hexa);
                        switch (resultado.rem) {
                                        case 0  strcpy(hexa,"0")
                                                break;
                                        case 1  strcpy(hexa,"1")
```

```
                                        break;
                                case 2: strcpy(hexa,"2");
                                        break;
                                case 3: strcpy(hexa,"3");
                                        break;
                                case 4: strcpy(hexa,"4");
                                        break;
                                case 5: strcpy(hexa,"5");
                                        break;
                                case 6: strcpy(hexa,"6");
                                        break;
                                case 7: strcpy(hexa,"7");
                                        break;
                                case 8: strcpy(hexa,"8");
                                        break;
                                case 9: strcpy(hexa,"9");
                                        break;
                                case 10: strcpy(hexa "A")
                                        break;
                                case 11. strcpy(hexa "B");
                                        break;
                                case 12: strcpy(hexa "C");
                                        break;
                                case 13. strcpy(hexa,"D");
                                        break;
                                case 14' strcpy(hexa,"E");
                                        break;
                                case 15 strcpy(hexa,"F");
                                        break;
                        }

        strcat(hexa,conv_hexa);
        if (resultado.quot<1)
                return(hexa);
    if (resultado.rem<16)
                dato=resultado.quot;
        }
        while (1);
}

int cursor(int es, int dato)

{

/* RUTINA PARA RECONOCER TECLAS PRESIONADAS */

        char a;
        fflush(stdin);
        a=getch();
        switch (toupper(a))
        {
                case '\x0': a=getch();
                        switch (a)
                        {
                                case 'H' if (dato'=0)
                                                dato--
                                        if (dato<0)
                                                dato=0;

                                        break;
                                case 'P' if (dato<(es-1))
                                                dato++;
                                        if (dato>vector2)
                                dato=0;
                                        break;

                        }
                        break;
                case 'T'

                                return(terminar);

                case '\r'
                                if (control'=1)
                                {
                                        gotoxy(1,22)
                                        printf("                                        ");
                                        gotoxy(1,22)
                                        printf("Regla %d  ",n_or);
                                        puts(regla)
                }
                                return(enter);
```

```
                    case 'Y':

                                        gotoxy(1,22);
                                        printf("                                        ");
                                        gotoxy(1,22);
                                        printf("Regla %d :",n_or);
                                        puts(regla);
                                        return(operadory);



        }
        if (dato<0)

                dato=0;
        return(dato);
}

void cerrar_puerto(void)
{

/* RUTINA PARA DESACTIVAR PUERTO SERIE */

asm{
        mov dx,0
        mov ax,0
        int 14h
  }
}

*****************************************************************************************************************************************************
*****************************************************************************************************************************************************

/*
Programa  MCFUZ2.C
Objetivo  Complementar funciones de MCFUZLOG.C
*/

#include "stdio.h"
#include "string.h"
#include "stdlib.h"
#include "conio.h"
#include "ctype.h"
#include "alloc.h"
#include "math.h"


/* DECLARACIÓN DE CONSTANTES */

#define cabecera 6
#define vector1 4
#define vector2 8
#define digit 100
#define longitud 50
#define points  4
#define caracter 10
#define input   0
#define output  1
#define inout   2
#define enter 1000
#define terminar 2000
#define operadory 3000
#define limite 1000
#define proposicion
#define opcion 14

/*PUNTERO DE ARCHIVO*/

FILE *out,*in;

/* DECLARACIÓN DE VARIABLES */

int io,v[inout],fm[inout][vector1],n_or,i,j,k,w,x,z;
int p, control, point=3,controle=0;
int vec1=vector1, vec2=vector2;
int b[8],l=0,sel;
unsigned int bin;
char v_nombre[inout][vector1][longitud],unidades[inout][vector1][longitud],fm_nombre[inout][vector1][vector2][longitud];
```

```
char nombre[longitud];
char ans;
char hexa_uc[inout][vector1][vector2][points][caracter],law_hex[limite][caracter];
float aux,punto_anterior,lim_inf[inout][vector1],lim_sup[inout][vector1];
float hexa[inout][vector1][vector2][points], paso_cuan[inout][vector1];
float punto[inout][vector1][vector2][points];
float escala=256.0,fac_mul=100.0,portador;
char *direccion="entrada";
char *regla;
char *arch_nombre;
int law[limite];
char *opciones[opcion]={"DECLARACIÓN DE FUNCIONES","GENERACIÓN DE REGLAS", "GENERADOR DE REGLAS","LISTADO DE
REGLAS","GUARDAR MODELO DIFUSO", "CREAR ARCHIVO MFC",
                        "CARGAR MCU CON FUNCIONES DIFUSAS","CARGAR MCU CON PROGRAMA","RECUPERAR MODELO
DIFUSO","A¥ADIR VARIABLE","BORRAR VARIABLE O FUNCIÓN",
                        "VARIABLES","MODIFICAR FORMA DE FUNCIÓN","SALIR"};



/* DECLARACIÓN DE FUNCIONES */

float cuantizar(void);
int crear_arch(char *arch_nombre);
int abrir_arch(char *nombre);
char (*formato_hexa(int));
int cursor(int es, int dato);
void guardar(int nivel,int dato1, int dato2,int dato3);
int binario(int &dato);
int binario2(int &dato);
void lista_reglas(void);
void principal(void);
int menu(void);
void archivo(void);
void recuperar(void);
void save(int);
void saves(char *cad);
int restore(void);
char *restores(void);
void libre(void);
void mandar(int dato);
void imagen(void);
void txin_mcu(char *ins);
void mcuini(void);
int restore(void);
void con_hex(void);
void reglas(void);
void gen_reglas(void);
void anadir_v(void);
void borrar_v(void);
void lista_variables(void);
void eleg_v(void);
void eleg_fm(void);
void modi_fm(void);
void limpiar(int, int);


main(int _argc,char *_argv[])

{

sel=atoi(_argv[1]);

clrscr()
        switch (sel)
        {
                case 2
                        recuperar();
                        anadir_v();
                        archivo();
                        break;
                case 3
                        recuperar();
                        borrar_v();
                        archivo();
                        break;
                case 5
                        recuperar();
                        lista_variables();
                        archivo();
```

```
                        break;
                case 4:
                        recuperar();
                        modi_fm();
                        archivo();
                        break;


        }

}

int cursor(int es, int dato)

{
        char a;
        fflush(stdin);
        a=getch();
        switch (toupper(a))
        {
                case '\x0': a=getch();
                        switch (a)
                        {
                                case 'H': if (dato!=0)
                                                dato--;
                                        if (dato<0)
                                                dato=0;

                                        break;
                                 case 'P': if (dato<(es-1))
                                                dato++;
                                        if (dato>vector2)
                        dato=0
                                        break;
                           }
                           break;
                case 'T':

                                return(terminar)

                case '\r':
                                if ((controle!=1)&&(sel==6))
                                {
                                        gotoxy(1,22);
                                        printf("                                                ");
                                        gotoxy(1,22);
                                        printf("Regla %d  ",n_or);
                                        puts(regla);
                }
                                return(enter);
                case 'Y':
                                if (sel==6)
                                {
                                        gotoxy(1,22);
                                        printf("                                                ");
                                        gotoxy(1,22);
                                        printf("Regla %d  ",n_or);
                                        puts(regla);
                                }
                                return(operadory);


        }
        if (dato<0)
                dato=0;
        return(dato);
}

void guardar(int nivel,int dato1, int dato2,int dato3)
{
 char *numero;
        for(i=0;i<dato2;i++)
                {
                for(z=0;z<dato3;z++)
                        {

                                fwrite("DC B",strlen("DC B "),1,out);
                                strcpy(numero,hexa_uc[nivel][dato1][i][z]);
                                fwrite(numero,strlen(numero),1,out);
```

```
                                        fwrite("\n",strlen("\n"),1,out);
                                }
                        }


}

int crear_arch(char *nombre)
{
  clrscr();
  if ((out = fopen(nombre, "wt"))
    == NULL)
  {
    clrscr();
    gotoxy(10,10);
    textcolor(LIGHTRED);
    cprintf("No se pudo crear archivo, presione cualquier tecla para");
    gotoxy(25,11);
    cprintf("para continuar");

    getch();
    textcolor(WHITE);
    return(0);
  }
return(1);
}

int abnr_arch(char *nombre)
{
  clrscr();
  puts(nombre);
  if ((in= fopen("mcfuzzy.tmp", "r+"))== NULL)
  {
    clrscr();
    gotoxy(10,10);
    textcolor(LIGHTRED);
    cprintf("No se pudo crear archivo, presione cualquier tecla para");
    gotoxy(25,11);
    cprintf("para continuar");

    getch();
    textcolor(WHITE);
    return(0);
  }
return(1);
}


int binario(int &dato)
{
        double n=0, resp;
        if (dato>1)
        {
                do
                {
                        resp=pow(2,n);
                        n++;
                }
                while(resp<dato);
                n--;
                dato=dato-pow(2,n);
                return(n);
        }
        else
        {
        if (dato==1)
        {
                dato=0;
                return(0);

        }
        else

            return(9);
        }
}

int binario2(int &dato)
{
```

```
        double n=-1, resp;
        if (dato>1)
        {
                do
                {
                        n++;
                        resp=pow(2,n);

                }
                while(resp<=dato);
                n--;
                dato=dato-pow(2,n);
                return(n);
        }
        else
        {
        if (dato==1)
        {
                dato=0;
                return(0);

        }
        else

            return(9);
        }
}

void lista_reglas(void)
{
        unsigned int r=0,x,b_ant=0,n=0,W=0,y=1;
        char ans;
        clrscr();
        if(l==0)
        {
            gotoxy(5,10);
            textcolor(LIGHTRED);
            cprintf("No hay reglas!!!");
            textcolor(WHITE);
       getch();
            return;
        }
        strcpy(regla,"");
        clrscr();
        printf("REGLAS DIFUSAS\n");
        textcolor(YELLOW);
        cprintf("REGLA %d  ",y);
        printf("\n");
        textcolor(WHITE);
         do
         {



                do
                {

                for (x=0;x<8;x++)
                        b[x]=0;
                j=law[r];
                        do
                        {
                                p=binario2(j);
                                if (p==9)
                                {
                                        p=0;
                                        bin=0;
                                }
                                else
                                {

                                        bin=1;
                                }
                                b[p]=bin;
                        }
                        while (j>0);
                j=b[0]+b[1]*2+b[2]*4;
                i=b[3]+b[4]*2;
                if ((b[7]==0)&&(b_ant==1))
                {
```

```
                textcolor(YELLOW);
                cprintf("REGLA %d : ",++y);
                printf("\n");

                textcolor(WHITE);
        }

        if (r!=0)
                if (((b[7]==0)&&(b_ant==0))||((b[7]==1)&&(b_ant==1)))
                {
                        puts(" Y ");

                }
         if ((b[7]==1)&&(b_ant==0))
                puts(" ENTONCES ");
         if (((b[7]==0)&&(b_ant==0))||((b[7]==0)&&(b_ant==1)))
         {
                strcpy(regla,"");
                strcpy(regla,v_nombre[input][i]);
                strcat(regla," ES ");
                strcat(regla,fm_nombre[input][i][j]);
                puts(regla);


         }
         if ((b[7]==1)&&(b_ant==0))
         {
                strcpy(regla,"");
                strcpy(regla,v_nombre[output][i]);
                strcat(regla," ES ");
                strcat(regla,fm_nombre[output][i][j])
                puts(regla);

         }

         b_ant=b[7];
         r++;
        }
        while ((wherey()<18)&&(r<l));
        if(r<l)
        {
                textcolor(GREEN);
                cprintf("(SIGUE ...)");
                printf("\n");
                textcolor(WHITE);
        }

printf("Por favor presionar cualquier tecla para continuar (T=terminar y B=borrar)\n");
ans=getch();
if (toupper(ans)=='T')
        return;
if (toupper(ans)=='B')
{
        textcolor(LIGHTRED);
        k=0;
        cprintf("Numero de la regla que desea borrar (Solo pueden borrarse las reglas anteriores hasta la ÚLtima desplegada) ");
        scanf("%d",&k);
        textcolor(WHITE);
        i=1;
        x=0;
        do
        {
         if ((law[i]>127)&&(law[i+1]<127))
            x++;
         if (x==k)
break;
         i++;
        }
        while((i<r)&&(x<=k));

        if (x==k)
        {

                do
                {

                        law[i]=255;
                        i--;
                        if (i<=0)
         break;
```

```
                                        else
                                                w=law[i-1];
                                }
                                while((law[i]>127)||(w<127));
                                law[i]=255;
                        }

                }
                clrscr();
                }
                while (r<l);
                for(x=0;x<l;x++)
                        if (law[x]==255)
                        {
                                n++;
                                w=x;
                                do
                                {
                                        law[w]=law[w+1];
                                        w++;
                                }
                                while(w<=l);
                                x--;
                        }
l=l-n;
}

void archivo(void)
{
        int aux;
        char *cad;
        int dec,sign;
        clrscr();
        if (crear_arch("mcfuzzy.tmp")==0)
                return;

        for (j=0;j<2;j++)
        {
                save(v[j]);
                for (x=0;x<v[j];x++)
                {
                        saves(v_nombre[j][x]);
                        saves(unidades[j][x]);
                        save(lim_inf[j][x]);
                        save(digit*(lim_inf[j][x]-(int)lim_inf[j][x]));
                        save(lim_sup[j][x]);
                        save(digit*(lim_sup[j][x]-(int)lim_sup[j][x]));
                        save(fm[j][x]);
                        for (w=0;w<fm[j][x];w++)
                        {
                                saves(fm_nombre[j][x][w]);
                                for(z=0;z<4;z++)
                                {
                                        save(punto[j][x][w][z]);
                                        save(digit*(punto[j][x][w][z]-(int)punto[j][x][w][z]));
                                        if (j==1)
                                                z=5;
                                }
                        }
                }
        }
        save(l);
        for (x=0;x<l;x++)
                save(law[x]);
    fclose(out);
}

void recuperar(void)
{
        float aux;
        char *cad;
        int dec,sign;
        if (abrir_arch(arch_nombre)==0)
                return;
     fseek(in,SEEK_SET,0);
        aux=digit;
        for (j=0;j<2;j++)
        {
                v[j]=restore();
                for (x=0;x<v[j];x++)
```

```
                {
                        strcpy(v_nombre[j][x],restores());
                        strcpy(unidades[j][x],restores());
                        lim_inf[j][x]=restore();
                        lim_inf[j][x]+=(restore()/aux);
                        lim_sup[j][x]=restore();
                        lim_sup[j][x]+=(restore()/aux);
                        fm[j][x]=restore();
                        for (w=0;w<fm[j][x];w++)
                        {
                                strcpy(fm_nombre[j][x][w],restores());
                                for(z=0;z<4;z++)
                                {
                                        punto[j][x][w][z]=restore();
                                        punto[j][x][w][z]+=(restore()/aux);
                                        if (j==1)
                                                z=5;
                                }
                        }
                }
        }
        l=restore();
        for (x=0;x<l;x++)
        {
                law[x]=restore();
        }

fclose(in);
}

int restore(void)
{
        char *cad;
        char *dato;
        cad=arch_nombre+20;
        dato=arch_nombre+50;
        strcpy(dato," ");
        strcpy(cad," ");
        do
        {
                fread(dato,sizeof(char),1,in);
                if (dato[0]!=',')
                        strcat(cad,dato);
        }
        while(dato[0]!=',');
        return(atoi(cad));
}
char *restores(void)
{
        char *cad;
        char *dato;
        cad=arch_nombre+20;
        dato=arch_nombre+50;
        strcpy(dato,"");
        strcpy(cad,"");
        do
        {
                fread(dato,sizeof(char),1,in);
                if (dato[0]!=',')
                        strcat(cad,dato);
        }
        while(dato[0]!=',');


return(cad);
}


void save(int dato)
{
        char *cad;
        itoa(dato,cad,10);
        fwrite(cad,strlen(cad),1,out);
        fwrite(",",1,1,out);
}
void saves(char *cad)
{
        fwrite(cad,strlen(cad),1,out);
        fwrite(",",1,1,out);
}
```

```
void gen_reglas(void)
{
int dato,dato1,incluir[4]={0,0,0,0};
int w=0,aux=0,p,y,l_ant,b_ant=0,reglas[limite][4],c=0;
clrscr();
gotoxy(30,1);
textcolor(LIGHTCYAN);
cprintf("Generador de Reglas");
textcolor(WHITE);
printf("\n");
dato=0;
if (v[input]==0)
{
        textcolor(LIGHTRED);
        gotoxy(10,10);
        cprintf("—No han sido declaradas variables lingüísticas!!!");
        getch();
        textcolor(WHITE);
        return;
}
do
{
        gotoxy(5,6);
        printf("Cuantas variables a combinar (El Sistema tiene %d variables de entrada): ",v[0]);
        scanf("%d",&dato);
        if (dato>(v[input]))
                dato=-1;

}
while((dato<0));
gotoxy(5,8)
printf("Cual Salida(El Sistema tiene %d salidas posibles): ",v[1])
w=0;
do
{
        if (dato1==1)
                break;
        aux=w;
        w=cursor(v[output],w);
        if (w<20)
        {
                gotoxy(60,8);
                puts(v_nombre[output][w]);
    }
}
while(w!=enter);
dato1=aux;
clrscr();
for(x=0;x<dato;x++)
{
        if (dato==v[input])
                break;
        w=0;
        gotoxy(20,10);
        printf("Variable de entrada %d ",x+1);
        do
        {
                aux=w;
                w=cursor(v[input],w);
                if (w<20)
                {
                gotoxy(50,10);
                printf("                ");
                gotoxy(50,10);
                puts(v_nombre[0][w]);
                }
        }
        while(w!=enter);
        incluir[aux]=1;
}
if (dato==v[input])
        for(x=0;x<dato;x++)
                incluir[x]=1;
l_ant=l;
z=0;
y=0;
w=0;
p=0;
do
```

```
{
        do
        {
                do
                {
                        do
                        {
                                if (incluir[0]==1)
                                {

                                        law[l]=p;
                                        l++;
                                }

                                if (incluir[1]==1)
                                {

                                        law[l]=8+(w);
                                        l++;
                                }
                                if (incluir[2]==1)
                                {
                                        law[l]=16+(y);
                                        l++;
                                }
                                if (incluir[3]==1)
                                {

                                        law[l]=24+y;
                                        l++;
                                }
                                law[l]=250;
                                l++;
                                z++;
                        }
                        while((z<fm[input][3])&&(incluir[3]==1));
                        z=0;
                        y++;
                }
                while((y<fm[input][2])&&(incluir[2]==1));
                y=0;
                w++;
        }
        while((w<fm[input][1])&&(incluir[1]==1));
        w=0
        p++
}
while((p<fm[input][0])&&(incluir[0]==1));

c=l_ant;

b_ant=1;
y=0;
clrscr();
do
{

        for (x=0;x<8;x++)
                b[x]=0;
        j=law[c];
        do
        {
                p=binario2(j);
                if (p==9)
                {
                        p=0;
                        bin=0;
                }
                else
                {
                        bin=1;
                }
                b[p]=bin;
        }
        while (j>0);
        j=b[0]+b[1]*2+b[2]*4;
        i=b[3]+b[4]*2;
        if ((b[7]==0)&&(b_ant==1))
        {
                textcolor(YELLOW);
```

```
                cprintf("REGLA NUEVA %d : ",++y);
                printf("\n");
                textcolor(WHITE);
        }
        if(c'=0)
                if (((b[7]==0)&&(b_ant==0))||((b[7]==1)&&(b_ant==1)))
                {
                        puts(" Y ");
                }
        if (law[c]<127)
        {
                strcpy(regla,"");
                strcpy(regla,v_nombre[input][i]);
                strcat(regla," ES ");
                strcat(regla,fm_nombre[input][i][j]);
                puts(regla);
        }
        if (law[c]==250)
        {
                w=0;
                strcpy(regla,"");
                printf( "ENTONCES ");
                strcpy(regla,v_nombre[output][dato1]);
                strcat(regla," ES ");
                puts(regla);
                w=-1;
                do
                {
                        aux=w;
                        w=cursor(fm[output][dato1],w);
                        textcolor(LIGHTCYAN);
                        if (w<20)
                        {

                                printf("                                \n ");
                                gotoxy(30,wherey()-1);
                                cputs(fm_nombre[output][dato1][w]);
                                gotoxy(1,wherey());
                        }
                        textcolor(WHITE);
                }
                while(w'=enter);
                printf("\n");
                w=aux;
                if (dato1==0)
                        law[c]=128+w;
                if (dato1==1)
                        law[c]=128+8+w;


        }

c++;
b_ant=b[7];
}
while(c<l);
}


void anadir_v(void)
{
char ans;
io=0;
clrscr();
if ((v[input]==0)&&(v[output]==0))
{
        textcolor(LIGHTRED);
        gotoxy(2,10);
        cprintf("---No existen variables declaradas  Por favor utilice la opción uno!!!");
        getch();
        return;
}
clrscr();
do
{
        gotoxy(10,10);
        printf("Desea anadir entradas o salidas (<E>ntrada / <S>alida)");
    fflush(stdin);
        ans=getche();
}
```

```
while((toupper(ans)!='E')&&(toupper(ans)!='S'));
if (toupper(ans)=='E')
        io=0;
else
        io=1;
do
{
                clrscr();
                textcolor(LIGHTRED);
                gotoxy(15,10);
                cprintf("El programa esta diseñado para manejar un máximo de");
                gotoxy(20,11);
                if (io==0)
                {
                        cprintf("cuatro variables l¡ngü¡sticas de entrada.\n ");
                        getch();
                        clrscr();
                        textcolor(MAGENTA);
                        cprintf("El sistema tiene la siguiente configuración de variables de entrada:\n");
                        printf("\n");
                        textcolor(LIGHTBLUE);
                        cprintf("    Variable Lingüistica      Funciones de membresia\n");

                        for(x=0;x<v[io];x++)
                        {
                                printf("\n");
                                textcolor(YELLOW);
                                cprintf("%s",v_nombre[io][x]);
                                gotoxy(30,wherey());
                                for(z=0;z<fm[io][x];z++)
                                {

                                        textcolor(LIGHTGREEN);
                                        cprintf("%s - ",fm_nombre[io][x][z]);
                                }
                        }
                        textcolor(WHITE);
                }
                if (io==1)
                {
                        cprintf("dos variables l¡ng  ¡sticas de salida. ");
                        textcolor(MAGENTA);
                        getch();
                        clrscr();
                        cprintf("El sistema tiene la siguiente configuración de variables de salida:\n\n");
                        textcolor(LIGHTBLUE);
                        cprintf("    Variable Lingüistica      Funciones de membresia\n");
                        for(x=0;x<v[io];x++)
                        {
                                printf("\n");
                                textcolor(YELLOW);
                                cprintf("%s",v_nombre[io][x]);
                                gotoxy(30,wherey());
                                for(z=0;z<fm[io][x];z++)
                                {
                                        textcolor(LIGHTGREEN);
                                        cprintf("%s - ",fm_nombre[io][x][z]);
                                }
                        }
                        textcolor(WHITE);
                }
                textcolor(YELLOW);
                gotoxy(10,12);
                cprintf("Cuantas variables linguisticas desea agregar ");
                fflush(stdin);
                scanf("%f",&aux);
        if (io==0)
                        if ((aux+v[io]>4)||(aux<0))
                                aux=-1;
                if (io==1)
                        if ((aux+v[io]>2)||(aux<0))
                                aux=-1;
        }
        while (aux==-1);
        v[io]+=aux;
        textcolor(WHITE);
        for (j=(v[io]-aux);j<v[io];j++)
        {
                clrscr();
                fflush(stdin);
```

```
                printf("Nombre de la variable %i: ",j+1);
                gets(v_nombre[io][j]);
                printf("Unidades: ");
                gets(unidades[io][j]);
                printf("Limite inferior: ");
                scanf("%f",&aux);
                lim_inf[io][j]=aux;
                do
                {       fflush(stdin);
                        printf("Limite superior  ");
                        scanf("%f",&aux);
                }
                while(aux<lim_inf[io][j]);
                lim_sup[io][j]=aux;

                do
                {
                        textcolor(LIGHTRED);
                        cprintf("El sistema solo permite un maximo de ocho funciones de membresia");
                        printf("\n");
                        textcolor(YELLOW);
                        cprintf("Cuantas funciones de membresia  ");
                        textcolor(WHITE);
                        scanf("%f",&aux);
                        fm[io][j]=aux;
                }
                while ((fm[io][j]<0)||(fm[io][j]>vec2));
                clrscr();
                for (i=0;i<fm[io][j];i++)
                {
                        clrscr();
                        gotoxy(20,wherey());
                        puts(v_nombre[io][j]);
                        gotoxy(20,wherey());
                        printf("Rango  %3.2f - %3.2f %s \n",lim_inf[io][j],lim_sup[io][j],unidades[io][j]);
                        gotoxy(10,wherey());
                        fflush(stdin);
                        printf("Nombre de función de membresia %i  ",i+1);
                        gets(nombre);
                        stpcpy(fm_nombre[io][j][i],nombre)
                        punto_anterior=lim_inf[io][j];
                        point=3;
                        if (io==1)
                                point=0;
                        for (z=0;z<=point;z++)
                        {
                                do
                                {
                                        fflush(stdin)
                                        printf("Punto %  ",z+1);
                                        scanf("%f",&aux);
                                }
                                while ((aux<lim_inf[io][j])||(aux>lim_sup[io][j])||(aux<punto_anterior));
                                punto_anterior=aux;
                                punto[io][j][i][z]=aux;
                        }
                }
        }
        getch()
        io++


}

void borrar_v(void)
{
FILE *temp;
char ans;
char *cad;
int comienzo,final,coma=0;
clrscr();
if (v[io]==0)
{
        clrscr();
        gotoxy(10,10);
        textcolor(LIGHTRED);
        cprintf("NO HAN SIDO DECLARADAS VARIABLES");
        getch();
        textcolor(WHITE);
        return;
```

```
}
if (l!=0)
{
        textcolor(LIGHTRED);
        gotoxy(20,10);
        cprintf("YA EXISTEN REGLAS--NO SE PUEDE BORRAR VARIABLES NI FUNCIONES");
        getch();
        textcolor(WHITE);
        return;
}
gotoxy(20,5);
textcolor(LIGHTRED);
cprintf("PRECAUCON: PROCESO DE BORRADO\n");
textcolor(WHITE);

printf("\nBorrar variable o función (<V>ariable <F>unción): ");
do
{
        fflush(stdin);
        ans=getch();

}
while ((toupper(ans)!='V')&&(toupper(ans)!='F'));
printf("%c",ans);
j=0;
if (toupper(ans)=='V')
{
        printf("\n <E>ntrada o <S>alida ");
        do
        {
                fflush(stdin);
                ans=getch();
                if (toupper(ans)=='E')
                        io=0;
                else
                        io=1;
        }
        while((toupper(ans)!='E')&&(toupper(ans)!='S'));
        printf("%c",ans);
        if (io==0)
                printf("\n Elegir variable de entrada a borrar \n");
        else
                printf("\n Elegir variable de salida a borrar \n");
        eleg_v();
        for(x=j;x<vector1;x++)
        {
                if ((x+1)==vector1)
                {
                        strcpy(v_nombre[io][x],"");
                        strcpy(unidades[io][x],"");
                        lim_inf[io][x]=0;
                        lim_sup[io][x]=0;
                }
                else
                {
                        strcpy(v_nombre[io][x],v_nombre[io][x+1])
                        strcpy(unidades[io][x],unidades[io][x+1])
                        lim_inf[io][x]=lim_inf[io][x+1];
                        lim_sup[io][x]=lim_sup[io][x+1];
                }
                if ((x+1)==vector1)
                        fm[io][x]=0;
                else
                        fm[io][x]=fm[io][x+1];
                for(w=0;w!=fm[io][x];w++)
                {
                        if ((w+1)==vector2)
                                strcpy(fm_nombre[io][x][w],"");
                        else
                                strcpy(fm_nombre[io][x][w],fm_nombre[io][x+1][w]);
                        if (io==0)
                                point=4;
                        else
                                point=1;
                        for(z=0;z<point;z++)
                        {
                                if (io==0)
                                {
                                 if((x+1)==vector1)
                                     punto[io][x][w][z]=0;
```

```
                                        else
                                            punto[io][x][w][z]=punto[io][x+1][w][z];
                                        }
                                        if (io==1)
                                        {
                                         if((x+1)>=2)
                                                punto[io][x][w][z]=0;
                                         else
                                            punto[io][x][w][z]=punto[io][x+1][w][z];
                                        }
                    }
                        }

            }
            v[io]--;

}
if (toupper(ans)=='F')
{
            printf("\n Pertenece a una variable de <E>ntrada o <S>alida  ");
            do
            {
                        fflush(stdin);
                        ans=getche();
                        if (toupper(ans)=='E')
                                    io=0;
                        else
                                    io=1;
            }
            while((toupper(ans)!='E')&&(toupper(ans)!='S'));
            if (io==0)
            {
                        printf("\n Elegir variable de entrada  \n")

            }
            else
                        printf("\n Elegir variable de salida  \n");
            textcolor(LIGHTMAGENTA);

             eleg_v();
             eleg_fm()
             for(x=w;x<fm[io][j];x++)
             {
                        if ((x+1)==vector2)
                                    strcpy(fm_nombre[io][j][x] "");
                        else
                                    strcpy(fm_nombre[io][j][x],fm_nombre[io][j][x+1]);
                        if (io==0)
                                    point=4;
                        else
                                    point=1
                        for(z=0;z<point;z++)
                                    if((z+1)==4)
                                                punto[io][x][w][z]=0;
                                    else
                                                punto[io][x][w][z]=punto[io][x+1][w][z];          punto[io][x][w][z]=punto[io][x+1][w][z];
            punto[io][x][w][z]=punto[io][x+1][w][z];



             }
             fm[io][j]--;

}
}

void lista_variables(void)
{
clrscr();
io=0;
gotoxy(25,1);
textcolor(LIGHTCYAN)
cprintf("Variables de Modelo Difuso\n\r");
textcolor(BROWN);
printf("\n\n\n");
cprintf("Variables de Entrada \n\r");
do
{
for (x=0;x<v[io];x++)
{
```

```
            textcolor(YELLOW);
            cputs(v_nombre[io][x]);
            printf(" : %4.2f - %4.2f ",lim_inf[io][x],lim_sup[io][x]);
      cputs(unidades[io][x]);
            printf("\n");
            for (w=0;w<fm[io][x];w++)
            {
                        textcolor(LIGHTGREEN);
                        gotoxy(5,wherey());
                        cputs(fm_nombre[io][x][w]);
                        cprintf(" : ");
                        if (io==0)
                                    point=4;
                        else
                                    point=1;
                        for(z=0;z<point;z++)
                                    printf("%4.2f ",punto[io][x][w][z]);
                        printf("\n");
                        if (wherey()==22)
                        {
                                    printf("Presione cualquier tecla para continuar");
                                    getch();
                                    clrscr();
                        }
            }
}
io++;
textcolor(BROWN);
if (io==1)
            cprintf("Variables de salida  \n\r")

}

while(io!=2)
getch();
textcolor(WHITE);
}

void eleg_v(void)
{
            textcolor(LIGHTMAGENTA);
            cprintf("Variables  \n\r");
            textcolor(LIGHTGREEN);
            for (x=0;x<v[io];x++)
            {
                        printf("   %d-",x+1);
                        cputs(v_nombre[io][x])

            }


            textcolor(WHITE);


            j=-1;
            do
            {
                        gotoxy(30,13);
                        aux=j
                        j=cursor(v[io],j);
                        if ((j!=enter)&&(j!=terminar)&&(j!=operadory))
                        {
                                    printf("                ");
                                    gotoxy(30,13);
                                    puts(v_nombre[io][j]);
                                    gotoxy(30,wherey()-1);
                                    aux=j;
                        }
             if (aux==-1)
                        j=-1
             }

             while((j!=enter));
            j=aux;
}

void eleg_fm(void)
{
            gotoxy(1,15);
            textcolor(YELLOW);
```

```
        cprintf("Funciones de membresia: ");
        printf("\n");
        textcolor(LIGHTBLUE);
        for (x=0;x<fm[io][j];x++)
        {
                printf("   %d-",x+1);
                cputs(fm_nombre[io][j][x]);

        }
        textcolor(WHITE);

        w=-1;
        gotoxy(20,17);
        textcolor(LIGHTGREEN);
        cprintf("<T>erminar modificación");
        textcolor(WHITE);
        do

        {
                gotoxy(30,18);
                aux=w;
                w=cursor(fm[io][j],w);
                if ((w!=enter)&&(w!=terminar)&&(w!=operadory))
                {
                        printf("                ");
                        gotoxy(30,18);
                        puts(fm_nombre[io][j][w]);
                        gotoxy(30,wherey()-1);
                        aux=w;
                }
                if (aux==-1)
                        w=-1;
        }

        while((w!=enter)&&(w!=terminar));
        if (w!=terminar)
            w=aux;

}

void modi_fm(void)
{
        clrscr();
        gotoxy(20,1);
        textcolor(LIGHTCYAN);
        cprintf("Modificar Forma de Funcion de Membresia");
        textcolor(WHITE);
        gotoxy(1,3);
        printf("\n Pertenece a una variable de <E>ntrada o <S>alida  ");
        do
        {
                fflush(stdin);
                ans=getche();
                if (toupper(ans)=='E')
                        io=0;
                else
                        io=1;
        }
        while((toupper(ans)!='E')&&(toupper(ans)!='S'));
        if (io==0)
        {
                printf("\n Elegir variable de entrada  \n");

        }
        else
                printf("\n Elegir vanable de salida  \n")
        textcolor(LIGHTMAGENTA);
        eleg_v();
        do
        {
                eleg_fm();
                if (w==terminar)
                        break;
                if (io==0)
                        point=4;
                else
                        point=1;
                textcolor(WHITE);
                printf("\n");
                gotoxy(1,19);
```

```
                printf("Rango: %4.2f - %4.2f ",lim_inf[io][j],lim_sup[io][j]);
                puts(unidades[io][j]);
                punto_anterior=lim_inf[io][j];
                for (z=0;z<point;z++)
                {
                        do
                        {
                                printf("Punto %i: %4.2f Nuevo valor: ",z+1,punto[io][j][w][z]);
                                scanf("%f",&aux);
                                gotoxy(1,(wherey()-1));

                        }
                        while ((aux<lim_inf[io][j])||(aux>lim_sup[io][j])||(aux<punto_anterior));
                        printf("\n");
                        punto_anterior=aux;
                        punto[io][j][w][z]=aux;
                }
                limpiar(18,24);
        }
        while (1);
}

void limpiar (int comienzo, int final)
{
int m;
        gotoxy(1,comienzo);
for (m=comienzo;m<=final;m++)
{

        clreol();
        printf("\n");
}
}
```

```
/*
ProgramaCOMMU.BAT
Objetivo: Activar programa HYPERTERMINAL de Windows95
*/

c:
cd\
C:\ARCHIV~1\ACCESO~1\HYPERT~1\HYPERTRM.EXE C:\ARCHIV~1\ACCESO~1\HYPERT~1\MCU HT
d:
```

GLOSARIO.

Adaptabilidad: La adaptabilidad de un sistema de control se manifiesta en la contínua medición automática de las características dinámicas del sistema, comparando estas con las características deseadas y usando las diferencias para efectuar los ajustes de los parámetros del sistema, optimizando su funcionamiento sin importar los cambios en el ambiente.

Afirmación condicional: Es una regla difusa completa (antecedente+consecuente).

Afirmación incondicional: Es una regla difusa sin antecedente.

Alcance/Dominio: Ancho de la función de membresía. El rango numérico sobre el cual se delimita la función de membresía o variable lingüística.

Antecedente : Es una unidad lógica en la porción IF (SI) de una regla difusa. Por ejemplo: IF "X es A1" AND "Y es B1", THEN "Z es C1", donde "X es C1" y "Y es B1" conforman el antecedente de la regla.

Centro de Gravedad (COG): Es un método de defusificación. La fórmula para calcular el centro de gravedad (COG) de una región consecuente difusa es:

$$COG = \frac{\sum_{i=1}^{n} y_i \, \mu_S(y_i)}{\sum_{i=1}^{n} \mu_S(y_i)}$$

Complemento difuso: Es la negación de un conjunto difuso que indica cuanto falto al elemento del conjunto para lograr un grado de membresía de 100%.

Conjunto booleano: Agrupación de elementos regidos por la teoría de conjuntos clásica, donde un elemento sólo puede tener uno de dos status: pertenece o no pertenece a la agrupación.

Conjunto difuso: También conocido como función de membresía. El cocnjunto difuso estable el grado de membresía entre 0 y 1 para todos los elementos que esten en su dominio.

Conjunto: Agrupación de elementos.

Consecuente: Es una unidad lógica en la porción THEN (ENTONCES) de una regla difusa. En el ejemplo: IF "X es A1" AND "Y es B1", THEN "Z es C1", el consecuente es "Z es C1".

Corte alpha: Es un valor que fija el umbral de activación para un conjunto difuso. Los grados de pertenencia por debajo del corte umbral son ignorados en los procesos de evaluación de reglas y defusificación.

Corte lambda: Restringe la máxima certeza de una función de membresía o región difusa.

Defusificación: Es el paso final en el procesamiento de lógica difusa donde se transforma las salidas difusas en valores específicos aplicables al sistema controlado.

Dominio: Es el rango de números reales sobre el cual se ha definido un conjunto difuso.
Entrada crisp : Entradas divididas de forma distintiva y rigurosamente.

Entrada difusa: Transformación de las entradas "crisps" a etiquetas de funciones de membresía.

Estabilidad asintótica: Es el sentido de estabilidad desde el punto de vista de Liapunov, el cual debe de exhibir las siguientes características:

1. Siempre que la magnitud de las condiciones iniciales de un sistema sean suficientemente pequeñas, entonces pequeñas perturbaciones en las condiciones iniciales producirán pequeñas perturbaciones en la solución correspondiente.
2. Existe un dominio de atracción tal que, siempre que la condición inicial pertenezca a este dominio, la correspondiente solución se aproximará a cero cuando *t* se aproxime a infinito.

Etiqueta: Es la descripción lingüística o nombre asignado a un conjunto difuso.

Evaluación de reglas: Ver inferencia difusa.

Fuerza de regla: Es el valor de verdad de una regla. A nivel de simple regla, la fuerza es determinada por la verdad o certeza del antecedente de la regla. A nivel de regla multiple, la fuerza de la regla de todas las reglas que posean el mismo consecuente es determinado por el proceso de evaluación de reglas.

Función de membresía: Es una función matemática que provee un significado numérico al conjunto difuso. Delimita los grados de pertencencia para cada entrada crisp que se encuentre dentro del dominio.

Fusificación: Es el primer paso en el procesamiento de lógica difusa, el cual toma un valor crisp, tal como una medida de temperatura, y la combina con alguna función de membresía almacenada y así producir una entrada difusa. Usualmente pueden existir múltiples entradas difusas correspondiente a una entrada crisp ya que esta puede tener grados de membresía parciales en diferentes conjuntos difusos.

Grado de membresía: Conocida también como valor de verdad. Es el grado en el cual el valor de la variable es compatible con un conjunto difuso asociado, y se representa por un número entre 0 (sin membresía) y 1 (membresía total).

Heurística: De acuerdo con ANSI/IEEE, la heurística trata de métodos o algoritmos exploratorios durante la resolución de problemas en los cuales las soluciones se descubren por la evaluación del progreso logrado en la búsqueda de un resultado final.

Indice de compatibilidad: Es un número real entre cero y uno que indica el grado en el cual las variables son compatibles con el un conjunto difuso de un sistema difuso.

Inferencia difusa: Es el segundo paso en el procesamiento difuso, también conocido como evaluación de reglas. En este paso se obtiene valore numéricos para las salidas difusas de cada acción del consecuente.

Ley de exclusión media: Es la ley que afirma que un elemento tiene dos estados respectivos en un conjunto de membresías: verdad (membresía total), falso (membresía cero). Esto significa que la intersección de un conjunto con su complemento, es el conjunto nulo (vacío). Dado que los conjuntos difusos permiten membresía parcial, un elemento puede estar parcialmente contenido en un conjunto y en el complemento del conjunto a la misma vez, esto viola la ley de exclusión media.

Lógica booleana: También referida como lógica convencional. Es una teoría creada por George Boole (1815-1864). Esta teoría gira en torno al concepto de conjunto y se desarrolló un sistema combinacional algebraico que trata las variables a través de operadores AND (Y), OR (O), NOT (NO), IF (SI) y THEM (ENTONCES).

Lógica difusa: Es una teoría matemática propuesta por Lofti Zadeh en 1965 para modelar información basándose en grados de membresía.

Memoria asociativa difusa (MAD): Es una matriz utilizada para representar la relación entre las variables de entrada y de salida de un sistema de control difuso. Las etiquetas de una de las variables de entrada de un sistema de control se utilizan para nombrar las filas y las etiquetas de la otra variable de entrada para nombrar las columnas. Cada celda de la matriz contiene una etiqueta de la variable de salida que denota el resultado específico de la combinación de entradas representadas en las filas y en las columnas.

Método directo de Liapunov: es un método matemático para el análisis de estabilidad el cual determina si el control es asintóticamente estable.
Normalización: Es el proceso de traducir las entradas del mundo real al universo de discurso de un grupo de conjuntos difusos. Los factores de la normalización también se conocen como "factores de escala".

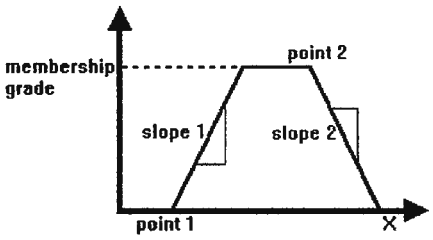
Operadores compensatorios: Son operadores difusos que compensan la rigidez de las reglas combinatorias de Zadeh para la intersección (AND) y la unión (OR).
Operadores difusos: Son operadores de conexión (AND y OR) que permiten combinar proposiciones del antecedente de la regla para obtener un valor de verdad final.

Peso de contribución: Es un factor asignado a una regla difusa tal que el diseñador tiene un medio de priorizarización de reglas. Este factor permite "amplificar" o "atenuar" la contribución de la regla en región difusa de salida.

Planta: Es un equipo, quizá simplemente un juego de piezas de una máquina, funcionando conjuntamente, cuyo objetivo es realizar una operación determinada.

Punto de balance: Es un punto que pertenece al universo de discurso ó eje de las *X* de las funciones de membresía; para una variable de salida a la cual las salidas aplicadas a las funciones de membresía son apropiadamente pesadas ó balanceadas. El punto de balance identifica un valor de salida crisp.

Representación punto pendiente: Es un esquema para representar las funciones de membresía difusas por puntos y pendientes. Por ejemplo una función de membresía trapezoidal puede ser representada por dos puntos-pendientes como se ve en la figura:



Singleton: Es un tipo especial de conjunto difuso que contiene solo un elemento.

Sistemas de control de lazo abierto: Son los sistemas de control en los que la salida no tiene efecto sobre la acción de control. En otras palabras, en un sistema de control de lazo abierto la salida ni se mide ni se retroalimenta para compararla con la entrada.

Sistemas de control de lazo cerrado: Con frecuencia se llama así a los sistemas de control retroalimentado. La señal de error actuante, que es la diferencia entre la señal de entrada y la de retroalimentación, entra al controlador para reducir un error y llevar la salida del sistema a un valor deseado.

Sistemas de control retroalimentado: Es aquel que tiende a mantener una relación preestablecida entre la salida y alguna entrada de referencia, comparándolas y utilizando la diferencia como medio de control.

Sistemas: Es una combinación de componentes que actúan conjuntamente y cumplen determinado objetivo. Un sistema no está limitado a objetivos físicos. El concepto de sistema puede aplicarse a fenómenos dinámicos abstractos, como los que se encuentran en economía.

Superficie de control: Usualmente se refiere a una gráfica de tridimensiones, donde los ejes *X* e *Y* son las entradas de variables y el eje *Z* es la salida de la variable controlada. La gráfica revela una superficie que denota la relación entre las variables de entrada y salida en un sistema de control.

Universo de discurso: Rango de valores asociados con una variable difusa. Varios conjuntos difusos pueden definirse dentro de un universo de discurso con traslapes de los dominios de los conjuntos difusos vecinos.

BIBLIOGRAFÍA.

[1]
Berkan, Riza C.
Trubatch, Sheldon L., *Fuzzy Systems Design Principles: Building Fuzzy IF-THEN Rule Bases*, 1997, Institute of Electrical and Electronics Engineers, Inc., Estados Unidos.

[2]
Pedrycz, Witold, *Fuzzy Control and Fuzzy Systems*, segunda edición, 1993, Research Studies Press Ltd, Inglaterra.

[3]
Kandel, Abraham,
Lee, Samuel C., *Fuzzy Switching and Automata: Theory and Applications*, 1979, Crane, Russak & Company, Inc., Estados Unidos.

[4]
International Electrotechnical Commission IEC, *EIC1131-Programmable Controllers, Part 7: Fuzzy Control Programming*, junio 1997.

[5]
Chen, C. H., *Fuzzy Logic and Neuronal Network Handbook*, 1996, McGraw-Hill, Estados Unidos.

[6]
Driankov, Dimiter
Hellendoorn, Hans
Reinfrank, Michael, *An Introduction to Difusa Control*, foreword by Lennart Ljung, Springer-Verlag Berlin Heidelberg, 1993, printed in U.S.A.

[7]
Rohrs Charles E.
Melsa, James L.
Schultz, Donald G., *Sistemas de Control Lineal*, 1994, McGraw-Hill / Interamericana de México, S. A. De C. V. Impreso en México.

[8]
Ogata, Katsuhiko, *Ingeniería de Control Moderna*, segunda edición (1993), Prentice-Hall Hispanoamericana, S.A.

[9]
Cortex Communications, Inc., Austin, Texas, U.S.A., *Fuzzy Logic Education Program 2.0*, 1992-1994, Center for emerging computer technologies, Motorola Inc.

[10]
Mendel, Jerry M., *Crisp Thoughts on Fuzzy Logic*, artículo de la revista Control Systems IEEE, volumen 14 número 4, agosto 94.

[11]
Cox, Earl, *Fuzzy Control Furor*, artículo de la revista Control Systems IEEE, volumen 13 número 4, agosto 93.

[12]
Heckenthaler Thomas,
Engel Sebastian, *Approximately Time-Optimal Fuzzy Control of a Two-Tank System*, artículo de la revista Control Systems IEEE, Volumen 14 número 3, junio 94.

[13]
Layne, Jeffery R.
Passino, Kevin M., *Fuzzy Model Reference Learning Control for Cargo Ship Steering*, artículo de la revista Control Systems IEEE, Volumen 13 número 6, diciembre 93.

[14]
Aptronix Inc, *Aptronix News AFN-92-09*, septiembre 92.

[15]
Kaehler, Steven, *Fuzzy Logic: An Introduction-Part 2*, Encoder Newsletter, Seattle Robotics Society, http://www.seattlerobotics.org/encoder/mar98/fuz/fl_part2.html

[16]
Kantrowitz, Mark, *FAQ: Fuzzy Logic and Fuzzy Expert Systems*, http://www.cs.emu.edu/Web/Groups/AI/html/faqs/ai/fuzzy/part1/faq.html

[17]
Aptronix Inc, *Aptronix Fuzzynet: Fuzzy Control Systems*, www.aptronix.com.

[18]
Von Altrock, Constantin, *INFORM 1990-1998*, www.fuzzytech.com.

[19]
Universidad Lasalle, *Artículos: Lógica Difusa*, Bogotá Colombia, http: //atenea. Lasalle.edu.co/~fochoam/

[20]
INFORM Software Corp., *FyzzyTech 5.10 80c166 Hardware Demo 1991-1998*, U.S.A.

[21]
Scaccia, Kobayashi, *A Fuzzy Logic-Based Smart Automatic Windshield Wiper*, artículo del la revista Control Systems IEEE, Volumen 16 número 6, diciembre 96.

[22]
Electrónica Veneta,
Inel Spa, *Modulo G36A/Ev: Transductores y Control de Velocidad y Posición-- Texto teórico experimental código TG36A*, Roma, Italia.

[23]
Electrónica Veneta,
Inel Spa, *Modulos G30A/EV-G30B/EV: Transductores y Control de Caudal y Nivel de Líquidos-- Texto teórico experimental códigos TG30A y TG30B*, Roma, Italia.

[24]
Electrónica Veneta,
Inel Spa, *Modulo G34A/Ev: Transductores y Control de Temperatura-- Manual teórico teórico experimental código TG34*, Roma, Italia.

[25]
Chen, Chi-Tsong,
Liu, Ching-Shau, *On Control System Design: A Comparative Study*, artículo de revista Control Systems IEEE, Volumen 14 número 5, octubre 1994.

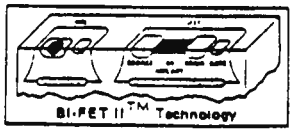
[26]
Layne, Jeffery R.,
Passino, Kevin M., *Fuzzy Model Reference Learning Control for Cargo Ship Steering*, artículo de revista Control Systems IEEE, diciembre 1993.

[27]
Hang, C. C.,
Sin, K. K., *A Comparative Performance Study of PID Auto-Tuners*, artículo de revista Control Systems IEEE, Volúmen 11 número 5, agosto 1997.

[28]
Siemens, *Fuzzy Lógic*, http://www.siemens.de/research/siefuzzy/fuzzy/

[29]
Bart Kosko, *Feedback*, artículo de revista Control Systems IEEE, Volúmen 14 número 6, diciembre 1994.

National Semiconductor



LF147/LF347/LF347B Wide Bandwidth Quad JFET Input Operational Amplifiers

General Description

The LF147 is a low cost, high speed quad JFET input operational amplifier with an internally trimmed input offset voltage (BI-FET II™ technology). The device requires a low supply current and yet maintains a large gain bandwidth product and a fast slew rate. In addition, well matched high voltage JFET input devices provide very low input bias and offset currents. The LF147 is pin compatible with the standard LM148. This feature allows designers to immediately upgrade the overall performance of existing LF148 and LM124 designs.

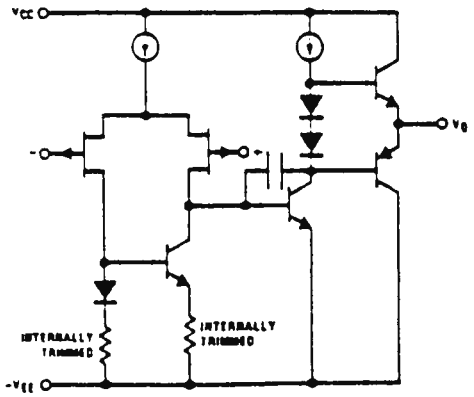
The LF147 may be used in applications such as high speed integrators, fast D/A converters, sample-and-hold circuits and many other circuits requiring low input offset voltage, low input bias current, high input impedance, high slew rate and wide bandwidth. The device has low noise and offset voltage drift.

Features

■ Internally trimmed offset voltage	5 mV max
■ Low input bias current	50 pA
■ Low input noise current	0.01 pA/$\sqrt{Hz}$
■ Wide gain bandwidth	4 MHz
■ High slew rate	13 V/μs
■ Low supply current	7.2 mA
■ High input impedance	$10^{12}\Omega$
■ Low total harmonic distortion $A_V = 10$, $R_L = 10k$, $V_O = 20$ Vp-p, BW = 20 Hz – 20 kHz	<0.02%
■ Low 1/f noise corner	50 Hz
■ Fast settling time to 0.01%	2 μs

Simplified Schematic

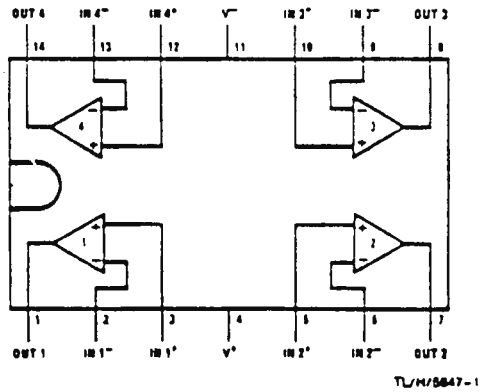
¼ Quad



TL/H/5647–13

Connection Diagram

Dual-In-Line Package



TL/H/5647–1

Top View

Order Number LF147D, LF347D, LF147J, LF347BJ, LF347J, LF347M, LF347WM, LF347BN or LF347N
See NS Package Number D14E, J14A, M14A, M14B or N14A

Absolute Maximum Ratings

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

	LF147	LF347B/LF347
Supply Voltage	±22V	±18V
Differential Input Voltage	±38V	±30V
Input Voltage Range (Note 1)	±19V	±15V
Output Short Circuit Duration (Note 2)	Continuous	Continuous
Power Dissipation (Notes 3 and 9)	900 mW	1000 mW
T_j max	150°C	150°C
θ_{jA}		
Cavity DIP (D) Package		80°C/W
Ceramic DIP (J) Package		70°C/W
Plastic DIP (N) Package		75°C/W
Surface Mount Narrow (M)		100°C/W
Surface Mount Wide (WM)		85°C/W

	LF147	LF347B/LF347
Operating Temperature Range	(Note 4)	(Note 4)
Storage Temperature Range	$-65°C \le T_A \le 150°C$	
Lead Temperature (Soldering, 10 sec.)	260°C	260°C
Soldering Information		
Dual-In-Line Package		
Soldering (10 seconds)		260°C
Small Outline Package		
Vapor Phase (60 seconds)		215°C
Infrared (15 seconds)		220°C

See AN-450 "Surface Mounting Methods and Their Effect on Product Reliability" for other methods of soldering surface mount devices.

ESD rating to be determined.

DC Electrical Characteristics (Note 5)

Symbol	Parameter	Conditions	LF147			LF347B			LF347			Units
			Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
V_{OS}	Input Offset Voltage	$R_S = 10\ k\Omega$, $T_A = 25°C$		1	5		3	5		5	10	mV
		Over Temperature			8			7			13	mV
$\Delta V_{OS}/\Delta T$	Average TC of Input Offset Voltage	$R_S = 10\ k\Omega$		10			10			10		µV/°C
I_{OS}	Input Offset Current	$T_j = 25°C$, (Notes 5, 6)		25	100		25	100		25	100	pA
		Over Temperature			25			4			4	nA
I_B	Input Bias Current	$T_j = 25°C$, (Notes 5, 6)		50	200		50	200		50	200	pA
		Over Temperature			50			8			8	nA
R_{IN}	Input Resistance	$T_j = 25°C$		10^{12}			10^{12}			10^{12}		Ω
A_{VOL}	Large Signal Voltage Gain	$V_S = \pm 15V$, $T_A = 25°C$	50	100		50	100		25	100		V/mV
		$V_O = \pm 10V$, $R_L = 2\ k\Omega$										
		Over Temperature	25			25			15			V/mV
V_O	Output Voltage Swing	$V_S = \pm 15V$, $R_L = 10\ k\Omega$	±12	±13.5		±12	±13.5		±12	±13.5		V
V_{CM}	Input Common-Mode Voltage Range	$V_S = \pm 15V$	±11	+15 −12		±11	+15 −12		±11	+15 −12		V V
CMRR	Common-Mode Rejection Ratio	$R_S \le 10\ k\Omega$	80	100		80	100		70	100		dB
PSRR	Supply Voltage Rejection Ratio	(Note 7)	80	100		80	100		70	100		dB
I_S	Supply Current			7.2	11		7.2	11		7.2	11	mA

AC Electrical Characteristics (Note 5)

Symbol	Parameter	Conditions	LF147			LF347B			LF347			Units
			Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
	Amplifier to Amplifier Coupling	$T_A = 25°C$, f = 1 Hz – 20 kHz (Input Referred)		−120			−120			−120		dB
SR	Slew Rate	$V_S = \pm 15V$, $T_A = 25°C$	8	13		8	13		8	13		V/μs
GBW	Gain-Bandwidth Product	$V_S = \pm 15V$, $T_A = 25°C$	2.2	4		2.2	4		2.2	4		MHz
e_n	Equivalent Input Noise Voltage	$T_A = 25°C$, $R_S = 100\Omega$, f = 1000 Hz		20			20			20		nV/$\sqrt{Hz}$
i_n	Equivalent Input Noise Current	$T_j = 25°C$, f = 1000 Hz		0.01			0.01			0.01		pA/$\sqrt{Hz}$

Note 1: Unless otherwise specified the absolute maximum negative input voltage is equal to the negative power supply voltage.

Note 2: Any of the amplifier outputs can be shorted to ground indefinitely, however, more than one should not be simultaneously shorted as the maximum junction temperature will be exceeded.

Note 3: For operating at elevated temperature, these devices must be derated based on a thermal resistance of θ_{jA}.

Note 4: The LF147 is available in the military temperature range $-55°C \leq T_A \leq 125°C$, while the LF347B and the LF347 are available in the commercial temperature range $0°C \leq T_A \leq 70°C$. Junction temperature can rise to T_j max = 150°C.

Note 5: Unless otherwise specified the specifications apply over the full temperature range and for $V_S = \pm 20V$ for the LF147 and for $V_S = \pm 15V$ for the LF347B/LF347. V_{OS}, I_B, and I_{OS} are measured at $V_{CM} = 0$.

Note 6: The input bias currents are junction leakage currents which approximately double for every 10°C increase in the junction temperature, T_j. Due to limited production test time, the input bias currents measured are correlated to junction temperature. In normal operation the junction temperature rises above the ambient temperature as a result of internal power dissipation, P_D. $T_j = T_A + \theta_{jA} P_D$ where θ_{jA} is the thermal resistance from junction to ambient. Use of a heat sink is recommended if input bias current is to be kept to a minimum.

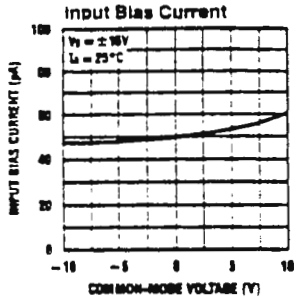
Note 7: Supply voltage rejection ratio is measured for both supply magnitudes increasing or decreasing simultaneously in accordance with common practice from $V_S = \pm 5V$ to $\pm 15V$ for the LF347 and LF347B and from $V_S = \pm 20V$ to $\pm 5V$ for the LF147.

Note 8: Refer to RETS147X for LF147D and LF147J military specifications.

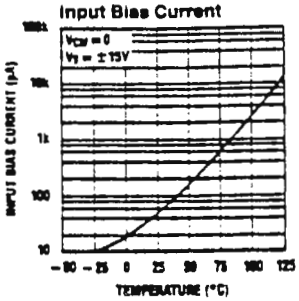
Note 9: Max. Power Dissipation is defined by the package characteristics. Operating the part near the Max. Power Dissipation may cause the part to operate outside guaranteed limits.

Typical Performance Characteristics

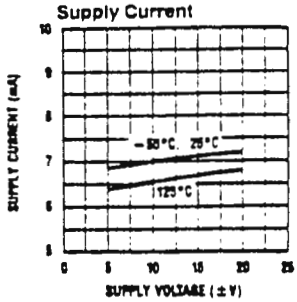
Input Bias Current



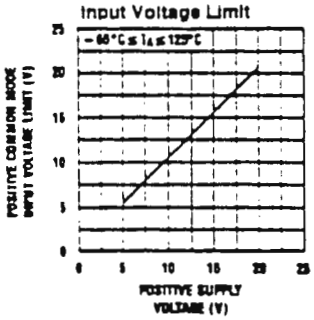
Input Bias Current



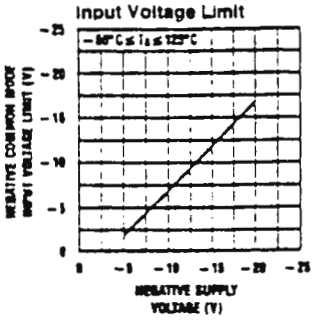
Supply Current



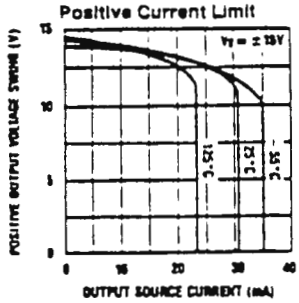
Positive Common-Mode Input Voltage Limit



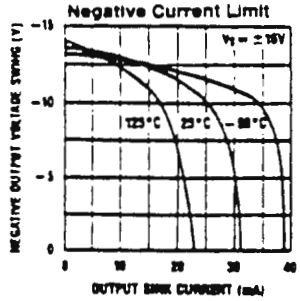
Negative Common-Mode Input Voltage Limit



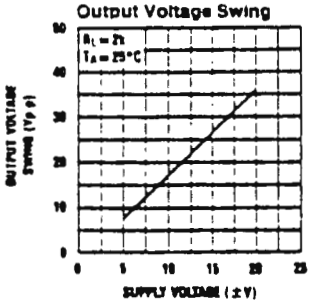
Positive Current Limit



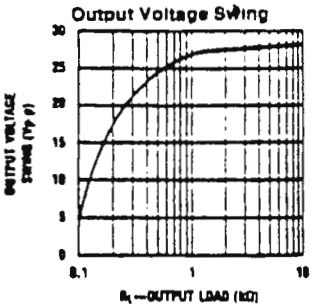
Negative Current Limit



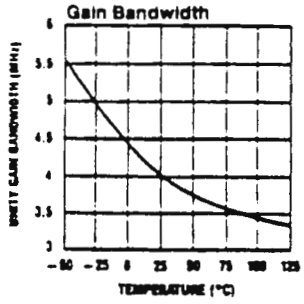
Output Voltage Swing



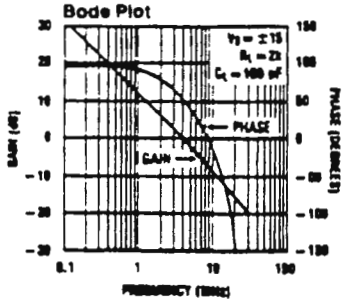
Output Voltage Swing



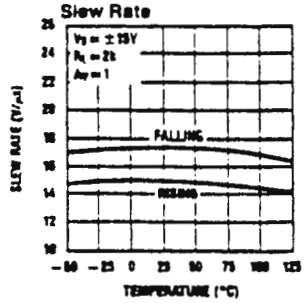
Gain Bandwidth



Bode Plot



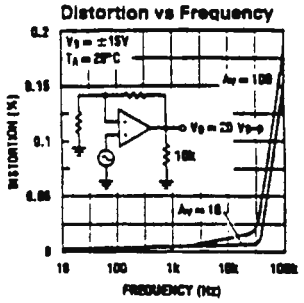
Slew Rate



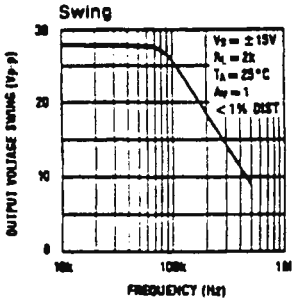
TL/H/5647-2

Typical Performance Characteristics (Continued)

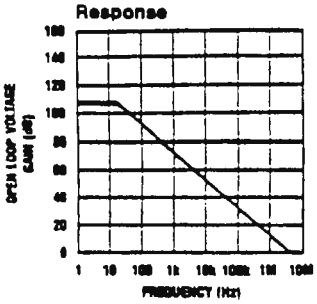
Distortion vs Frequency



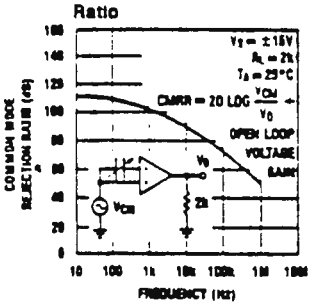
Undistorted Output Voltage Swing



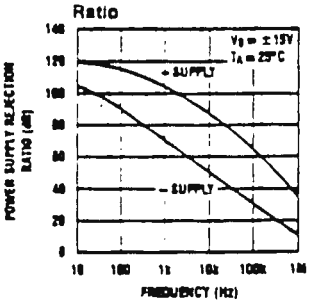
Open Loop Frequency Response



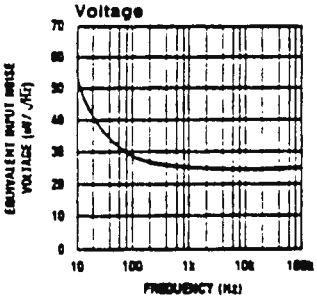
Common-Mode Rejection Ratio



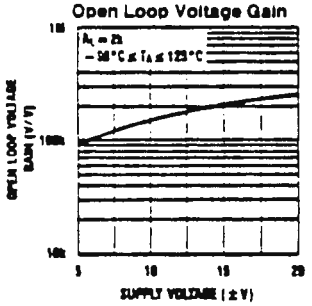
Power Supply Rejection Ratio



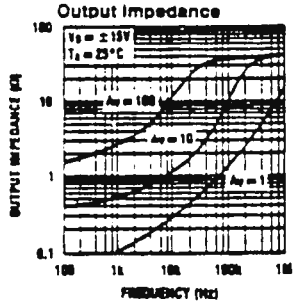
Equivalent Input Noise Voltage



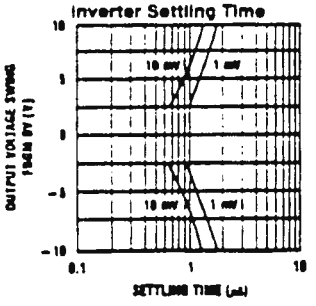
Open Loop Voltage Gain



Output Impedance



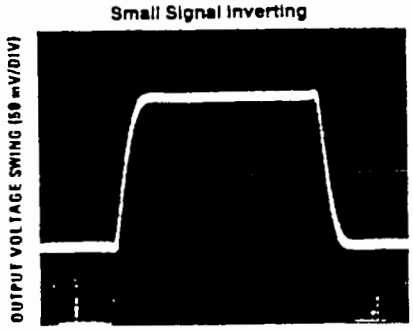
Inverter Settling Time



TL/H/5647–3

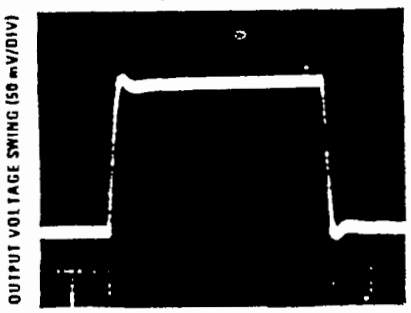
Pulse Response $R_L = 2\ k\Omega$, $C_L = 10\ pF$

Small Signal Inverting



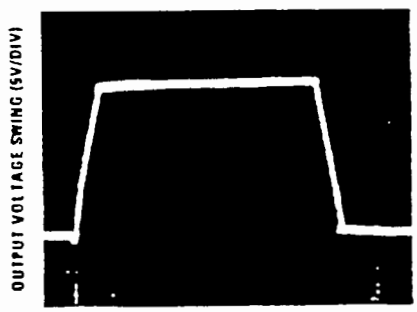
TL/H/5647–4

Small Signal Non-Inverting



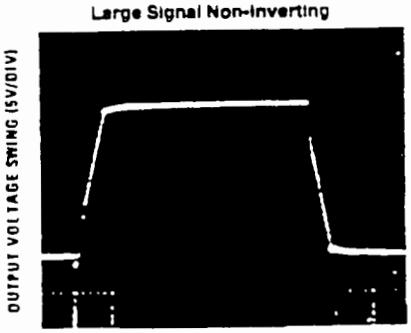
TL/H/5647–5

Large Signal Inverting



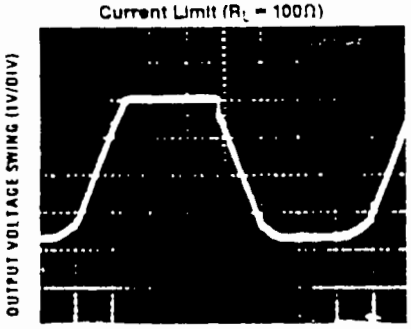
TL/H/5647–6

Large Signal Non-Inverting



TL/H/5647–7

Current Limit ($R_L = 100\Omega$)



TL/H/5647–8

Application Hints

The LF147 is an op amp with an internally trimmed input offset voltage and JFET input devices (BI-FET II™). These JFETs have large reverse breakdown voltages from gate to source and drain eliminating the need for clamps across the inputs. Therefore, large differential input voltages can easily be accommodated without a large increase in input current. The maximum differential input voltage is independent of the supply voltages. However, neither of the input voltages should be allowed to exceed the negative supply as this will cause large currents to flow which can result in a destroyed unit.

Exceeding the negative common-mode limit on either input will force the output to a high state, potentially causing a reversal of phase to the output. Exceeding the negative common-mode limit on both inputs will force the amplifier

Application Hints (Continued)

output to a high state. In neither case does a latch occur since raising the input back within the common-mode range again puts the input stage and thus the amplifier in a normal operating mode.

Exceeding the positive common-mode limit on a single input will not change the phase of the output; however, if both inputs exceed the limit, the output of the amplifier will be forced to a high state.

The amplifiers will operate with a common-mode input voltage equal to the positive supply; however, the gain bandwidth and slew rate may be decreased in this condition. When the negative common-mode voltage swings to within 3V of the negative supply, an increase in input offset voltage may occur.

Each amplifier is individually biased by a zener reference which allows normal circuit operation on ±4.5V power supplies. Supply voltages less than these may result in lower gain bandwidth and slew rate.

The LF147 will drive a 2 kΩ load resistance to ±10V over the full temperature range. If the amplifier is forced to drive heavier load currents, however, an increase in input offset voltage may occur on the negative voltage swing and finally reach an active current limit on both positive and negative swings.

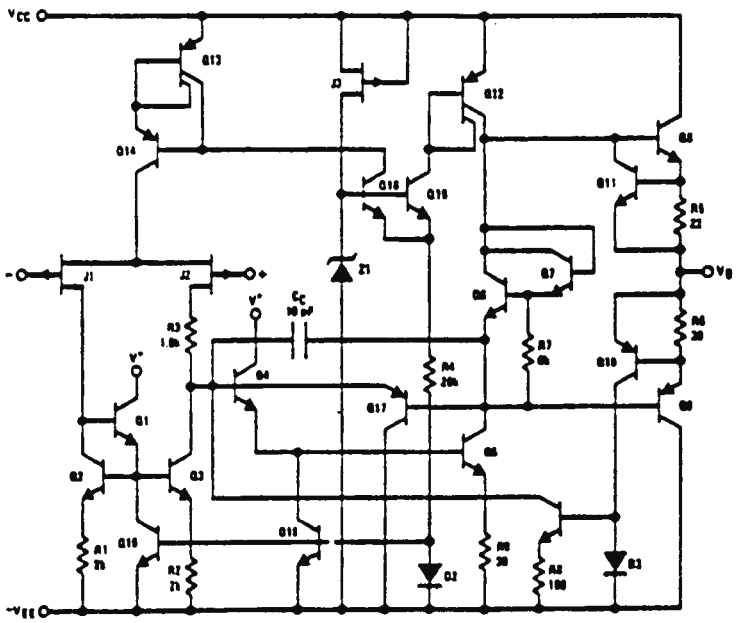
Precautions should be taken to ensure that the power supply for the integrated circuit never becomes reversed in polarity or that the unit is not inadvertently installed backwards in a socket as an unlimited current surge through the resulting forward diode within the IC could cause fusing of the internal conductors and result in a destroyed unit.

Because these amplifiers are JFET rather than MOSFET input op amps they do not require special handling.

As with most amplifiers, care should be taken with lead dress, component placement and supply decoupling in order to ensure stability. For example, resistors from the output to an input should be placed with the body close to the input to minimize "pick-up" and maximize the frequency of the feedback pole by minimizing the capacitance from the input to ground.

A feedback pole is created when the feedback around any amplifier is resistive. The parallel resistance and capacitance from the input of the device (usually the inverting input) to AC ground set the frequency of the pole. In many instances the frequency of this pole is much greater than the expected 3 dB frequency of the closed loop gain and consequently there is negligible effect on stability margin. However, if the feedback pole is less than approximately 6 times the expected 3 dB frequency a lead capacitor should be placed from the output to the input of the op amp. The value of the added capacitor should be such that the RC time constant of this capacitor and the resistance it parallels is greater than or equal to the original feedback pole time constant.

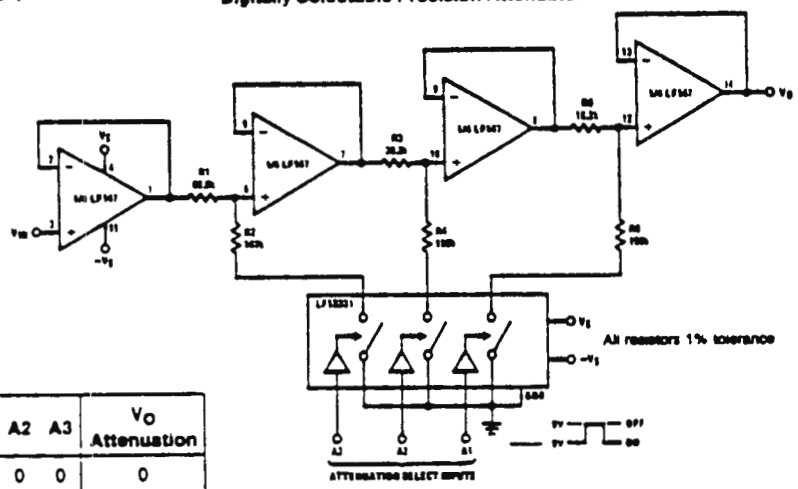
Detailed Schematic



TL/H/5647–9

Typical Applications

Digitally Selectable Precision Attenuator



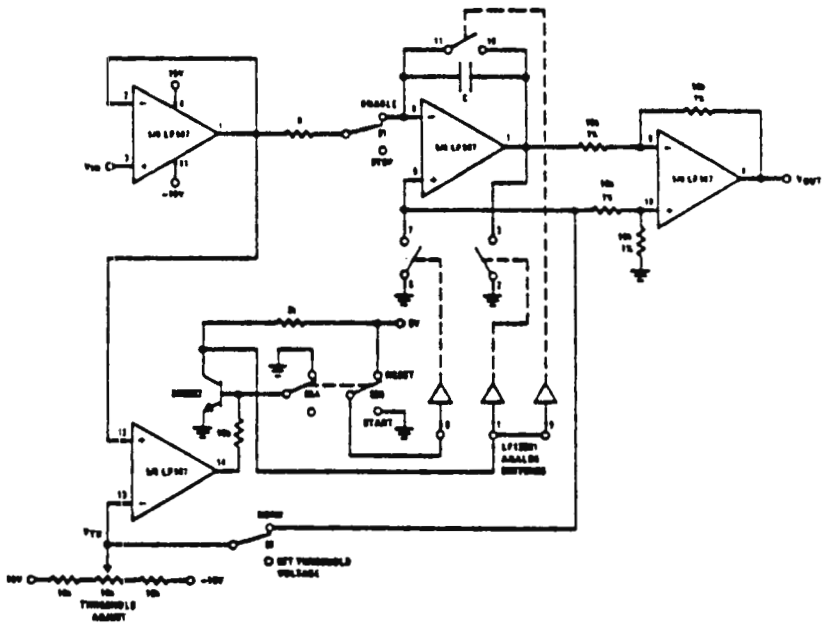
All resistors 1% tolerance

A1	A2	A3	V_O Attenuation
0	0	0	0
0	0	1	−1 dB
0	1	0	−2 dB
0	1	1	−3 dB
1	0	0	−4 dB
1	0	1	−5 dB
1	1	0	−6 dB
1	1	1	−7 dB

TL/H/5647–10

- Accuracy of better than 0.4% with standard 1% value resistors
- No offset adjustment necessary
- Expandable to any number of stages
- Very high input impedance

Long Time Integrator with Reset, Hold and Starting Threshold Adjustment



TL/H/5647–11

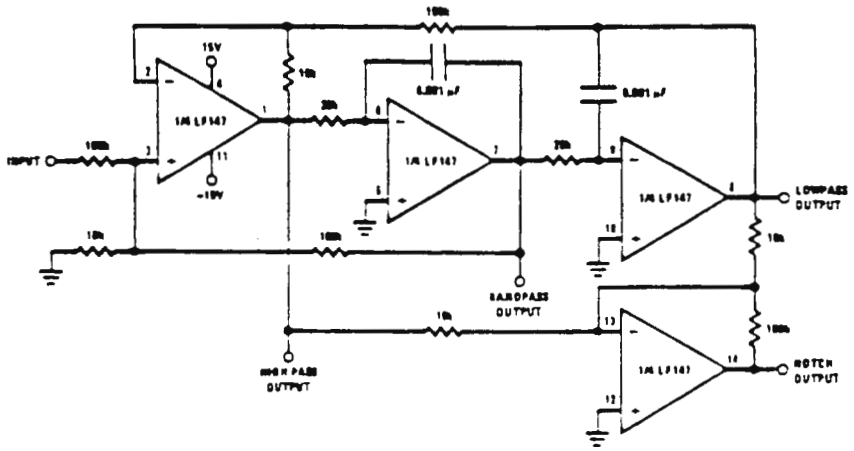
- V_{OUT} starts from zero and is equal to the integral of the input voltage with respect to the threshold voltage:

$$V_{OUT} = \frac{1}{RC}\int_0^t (V_{IN} - V_{TH})dt$$

- Output starts when $V_{IN} \geq V_{TH}$
- Switch S1 permits stopping and holding any output value
- Switch S2 resets system to zero

Typical Applications (Continued)

Universal State Variable Filter



TL/H/5847-12

For circuit shown:

f_o = 3 kHz, f_{NOTCH} = 8.5 kHz

Q = 3.4

Passband gain:

Highpass—0.1

Bandpass—1

Lowpass—1

Notch—10

- $f_o \times Q \leq 200$ kHz
- 10V peak sinusoidal output swing without slew limiting to 200 kHz
- See LM148 data sheet for design equations

1 INTRODUCTION

The HCMOS MC68HC11A8 is an advanced 8-bit microcontroller (MCU) with highly sophisticated on-chip peripheral capabilities. A fully static design and high-density complementary metal-oxide semiconductor (HCMOS) fabrication process allow E-series devices to operate at frequencies from 3 MHz to dc, with very low power consumption.

1.1 Features

The following are some of the hardware and software highlights.

1.1.1 Hardware Features

- 8 Kbytes of ROM
- 512 Bytes of EEPROM
- 256 Bytes of RAM (All Saved During Standby) Relocatable to Any 4K Boundary
- Enhanced 16-Bit Timer System:
  - — Four Stage Programmable Prescaler
  - — Three Input Capture Functions
  - — Five Output Compare Functions
- 8-Bit Pulse Accumulator Circuit
- Enhanced NRZ Serial Communications Interface (SCI)
- Serial Peripheral Interface (SPI)
- Eight Channel, 8-Bit Analog-to-Digital Converter
- Real Time Interrupt Circuit
- Computer Operating Properly (COP) Watchdog System
- Available in Dual-In-Line or Leaded Chip Carrier Packages

1.1.2 Software Features

- Enhanced M6800/M6801 Instruction Set
- 16 x 16 Integer and Fractional Divide Features
- Bit Manipulation
- WAIT Mode
- STOP Mode

1.2 General Description

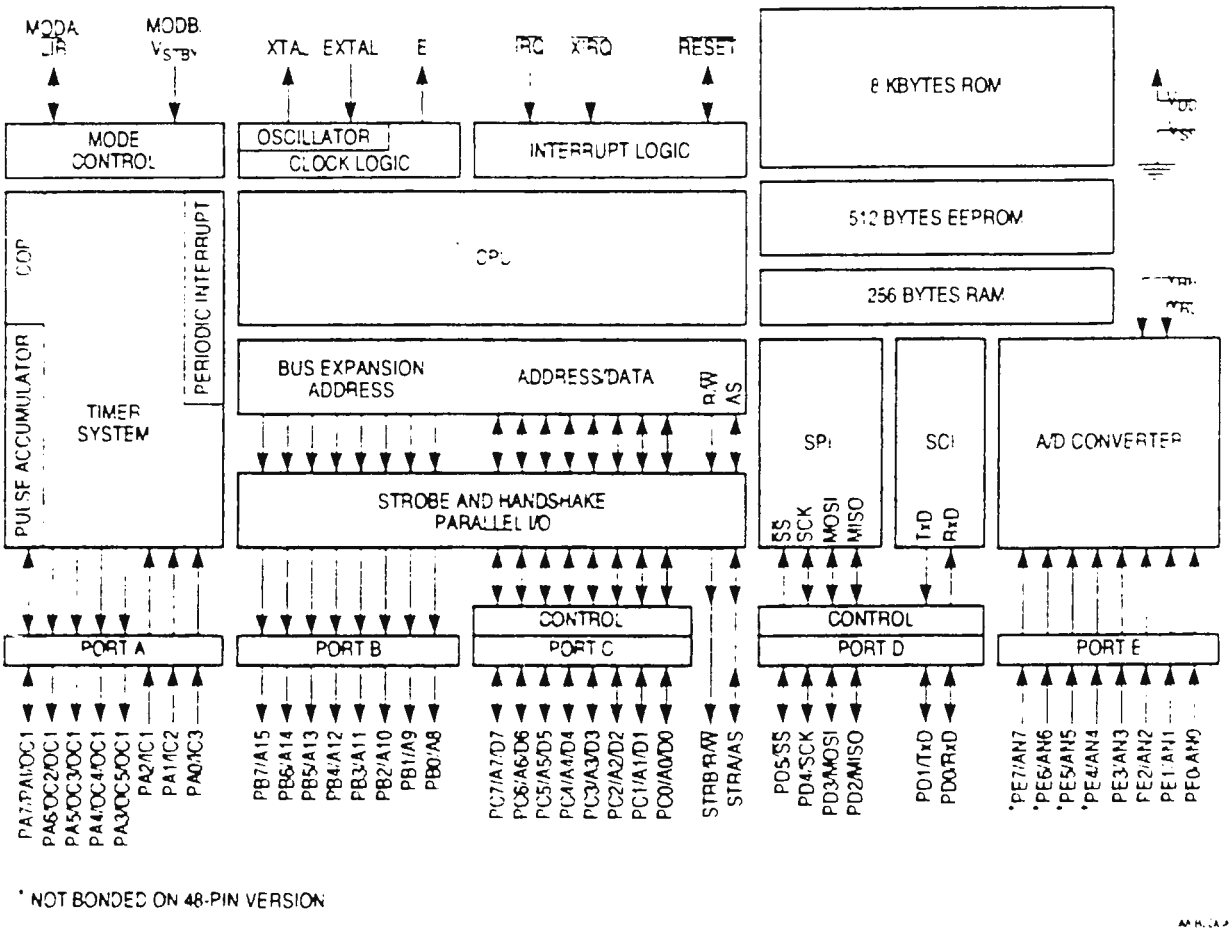
The high-density CMOS technology (HCMOS) used on the MC68HC11A8 combines smaller size and higher speeds with the low power and high noise immunity of CMOS. On-chip memory systems include 8 Kbytes of ROM, 512 bytes of electrically erasable programmable ROM (EEPROM), and 256 bytes of static RAM.

A block diagram of the MC68HC11A8 is shown in **Figure 1-1**. Major peripheral functions are provided on-chip. An eight channel analog-to-digital (A/D) converter is included with eight bits of resolution. An asynchronous serial communications interface

(SCI) and a separate synchronous serial peripheral interface (SPI) are included. The main 16-bit free-running timer system has three input capture lines, five output compare lines, and a real-time interrupt function. An 8-bit pulse accumulator subsystem can count external events or measure external periods.

Self monitoring circuitry is included on-chip to protect against system errors. A computer operating properly (COP) watchdog system protects against software failures. A clock monitor system generates a system reset in case the clock is lost or runs too slow. An illegal opcode detection circuit provides a non-maskable interrupt if an illegal opcode is detected.

Two software controlled operating modes. WAIT and STOP, are available to conserve additional power.



* NOT BONDED ON 48-PIN VERSION

Figure 1-1 Block Diagram

3 Programmer's Model

In addition to being able to execute all M6800 and M6801 instructions, the MC68HC11A8 allows execution of 91 new opcodes **Figure 1-2** shows the seven CPU registers which are available to the programmer.

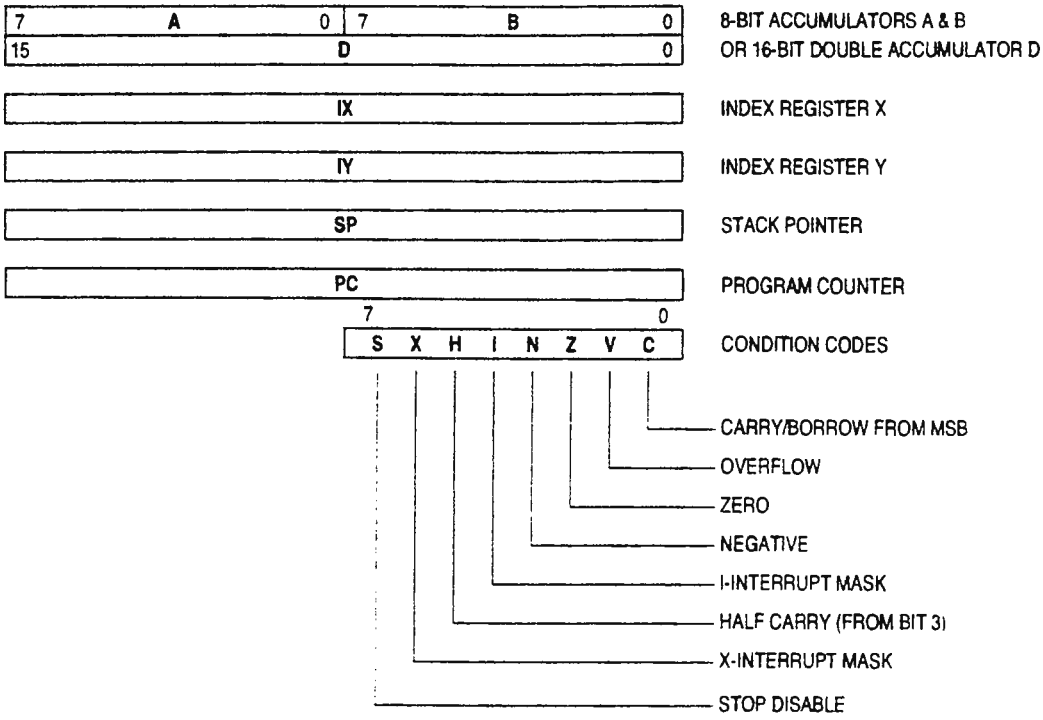


Figure 1-2 Programming Model

1.4 Summary of M68HC11 Family

Table 1-1 and the following paragraphs summarize the current members of the M68HC11 family of MCUs. This technical data book describes the MC68HC11A8 version and can be used as a primary reference for several other versions of the M68HC11 family. However, with the exception of the CPU, some newer members differ greatly from the MC68HC11A8 MCU and their respective technical literature should be referenced.

Several of the device series within the M68HC11 family have 'x1 and 'x0 versions. These are identical to the main member of the series but have some of their on-chip resources disabled. For instance, an MC68HC11A1 is identical to the MC68HC11A8 except that its ROM is disabled. An MC68HC11A0 has disabled EPROM **and** EEPROM arrays. Refer to **Table 1-1**.

Nearly all series within the M68HC11 family have both a ROM version and an EPROM version. Any device in the M68HC11 family that has a 7 preceding the 11 is a device containing EPROM instead of ROM (e.g., MC68HC711E9). These devices operate exactly as the custom ROM-based version (e.g., MC68HC11E9) but can be programmed by the user. EPROM-based devices in a windowed package can be erased and reprogrammed indefinitely. EPROM-based devices in standard packages are one-time-programmable (OTP). Refer to **Table 1-1**.

2 SIGNAL DESCRIPTIONS AND OPERATING MODES

The signal descriptions and operating modes are presented in this section. When the microcontroller is in an expanded multiplexed operating mode, 18 pins change function to support a multiplexed address/data bus.

2.1 Signal Pin Descriptions

The following paragraphs provide a description of the input/output signals. Reference is made, where applicable, to other sections that contain more detail about the function being performed.

2.1.1 Input Power (V_{DD}) and Ground (V_{SS})

Power is supplied to the microcontroller using these pins. V_{DD} is the positive power input and V_{SS} is ground. Although the MC68HC11A8 is a CMOS device, very fast signal transitions are present on many of its pins. Short rise and fall times are present even when the microcontroller is operating at slow clock rates. Special care must be taken to provide good power supply bypassing at the MCU. Recommended bypassing would include a 0.1 µF ceramic capacitor between the V_{DD} and V_{SS} pins and physically adjacent to one of the two pins. A bulk capacitance, whose size depends on the other circuitry in the system, should also be present on the circuit board.

2.1.2 Reset ($\overline{\text{RESET}}$)

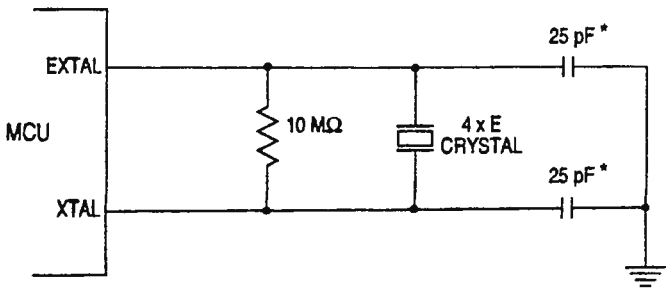
This active low bidirectional control signal is used as an input to initialize the MC68HC11A8 to a known start-up state, and as an open-drain output to indicate that an internal failure has been detected in either the clock monitor or computer operating properly (COP) watchdog circuit. This reset signal is significantly different from the reset signal used on other Motorola MCUs. Please refer to **9 RESETS, INTERRUPTS, AND LOW POWER MODES** before designing circuitry to generate or monitor this signal.

2.1.3 Crystal Driver and External Clock Input (XTAL, EXTAL)

These two pins provide the interface for either a crystal or a CMOS compatible clock to control the internal clock generator circuitry. The frequency applied to these pins shall be four times higher than the desired E clock rate. The XTAL pin is normally left unterminated when using an external CMOS compatible clock input to the EXTAL pin. However, a 10K to 100K load resistor to ground may be used to reduce RFI noise emission. The XTAL output is normally intended to drive only a crystal.

The XTAL output may be buffered with a high-input-impedance buffer such as the 74HC04, or it may be used to drive the EXTAL input of another M68HC11.

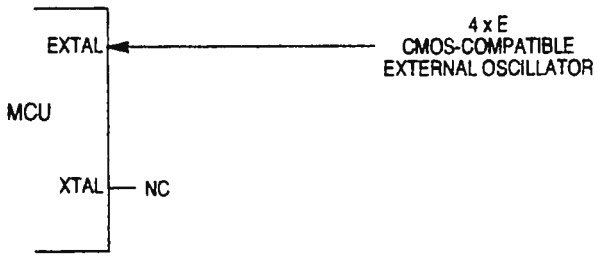
In all cases take extra care in the circuit board layout around the oscillator pins. Load capacitances shown in the oscillator circuits include all stray layout capacitances. Refer to **Figure 2-1**, **Figure 2-2**, and **Figure 2-3** for diagrams of oscillator circuits.



* THIS VALUE INCLUDES ALL STRAY CAPACITANCES.

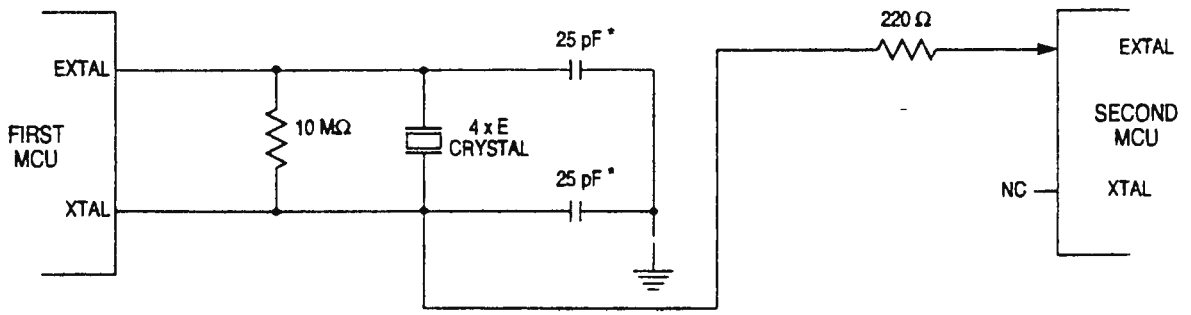
COMMON XTAL CONN

Figure 2-1 Common Crystal Connections



EXT EXTAL CONN

Figure 2-2 External Oscillator Connections



* THIS VALUE INCLUDES ALL STRAY CAPACITANCES.

DUAL-MCU XTAL CONN

Figure 2-3 One Crystal Driving Two MCUs

2.1.4 E Clock Output (E)

This is the output connection for the internally generated E clock which can be used as a timing reference. The frequency of the E clock output is actually one fourth that of the input frequency at the XTAL and EXTAL pins. When the E clock output is low an internal process is taking place and, when high, data is being accessed. The E clock signal is halted when the MCU is in STOP mode.

2.1.5 Interrupt Request ($\overline{IRQ}$)

The $\overline{IRQ}$ input provides a means for requesting asynchronous interrupts to the MC68HC11A8. It is program selectable (OPTION register) with a choice of either negative edge-sensitive or level-sensitive triggering, and is always configured to level-sensitive triggering by reset. The $\overline{IRQ}$ pin requires an external pull-up resistor to V_{DD} (typically 4.7K ohm).

2.1.6 Non-Maskable Interrupt ($\overline{XIRQ}$)

This input provides a means for requesting a non-maskable interrupt, after reset initialization. During reset, the X bit in the condition code register is set and any interrupt is masked until MCU software enables it. The $\overline{XIRQ}$ input is level sensitive and requires an external pull-up resistor to V_{DD}.

2.1.7 Mode A/Load Instruction Register and Mode B/Standby Voltage (MODA/$\overline{LIR}$, MODB/V_{STBY})

During reset, MODA and MODB are used to select one of the four operating modes. Refer to **Table 2-1**. Paragraph **2.2 Operating Modes** provides additional information.

Table 2-1 Operating Modes vs. MODA and MODB

MODB	MODA	Mode Selected
1	0	Single Chip
1	1	Expanded Multiplexed
0	0	Special Bootstrap
0	1	Special Test

After the operating mode has been selected, the $\overline{LIR}$ pin provides an open-drain output to indicate that an instruction is starting. All instructions are made up of a series of E clock cycles. The $\overline{LIR}$ signal goes low during the first E clock cycle of each instruction (opcode fetch). This output is provided as an aid in program debugging.

The V_{STBY} signal is used as the input for RAM standby power. When the voltage on this pin is more than one MOS threshold (about 0.7 volts) above the V_{DD} voltage, the internal 256-byte RAM and part of the reset logic are powered from this signal rather than the V_{DD} input. This allows RAM contents to be retained without V_{DD} power applied to the MCU. Reset must be driven low before V_{DD} is removed and must remain low until V_{DD} has been restored to a valid level.

2.1.8 A/D Converter Reference Voltages (V_{RL}, V_{RH})

These two inputs provide the reference voltages for the analog-to-digital converter circuitry.

2.1.9 Strobe B and Read/Write (STRB/R/$\overline{W}$)

This signal acts as a strobe B output or as a data bus direction indicator depending on the operating mode.

In single-chip operating mode, the STRB output acts as a programmable strobe for handshake with other parallel I/O devices. Refer to **4 PARALLEL I/O** for additional information.

In expanded multiplexed operating mode, R/$\overline{W}$ is used to control the direction of transfers on the external data bus. A low on the R/$\overline{W}$ signal indicates data is being written to the external data bus. A high on this signal indicates that a read cycle is in progress. R/$\overline{W}$ will stay low during consecutive data bus write cycles, such as in a double-byte store. The NAND of inverted R/$\overline{W}$ with the E clock should be used as the write enable signal for an external static RAM.

2.1.10 Strobe A and Address Strobe (STRA/AS)

This signal acts as an edge detecting strobe A input or as an address strobe bus control output depending on the operating mode.

In single-chip operating mode, the STRA input acts as a programmable strobe for handshake with other parallel I/O devices. Refer to **4 PARALLEL I/O** for additional information.

In expanded multiplexed operating mode, the AS output is used to demultiplex the address and data signals at port C. Refer to **2.2.2 Expanded Multiplexed Operating Mode** for additional information.

2.1.11 Port Signals

Ports A, D, and E signals are independent of the operating mode. Port B provides eight general purpose output signals in single-chip operating modes and provides eight high-order address signals when the microcontroller is in expanded multiplexed operating modes. Port C provides eight general purpose input/output signals when the microcontroller is in singlechip operating modes. When the microcontroller is in expanded multiplexed operating modes, port C is used for a multiplexed address/data bus. **Table 2-2** shows a summary of the 40 port signals as they relate to the operating modes. Unused inputs and I/O pins configured as inputs should be terminated high or low.

2.1.11.1 Port A

Port A may be configured for: three input capture functions (IC1, IC2, IC3), four output compare functions (OC2, OC3, OC4, OC5), and either a pulse accumulator input (PAI) or a fifth output compare function (OC1). Refer to **8.1 Programmable Timer** for additional information.

Any port A pin that is not used for its alternate timer function may be used as a general-purpose input or output line.

2.1.11.2 Port B

While in single-chip operating modes, all of the port B pins are general-purpose output pins. During MCU reads of this port, the level sensed at the input side of the port B output drivers is read. Port B may also be used in a simple strobed output mode where an output pulse appears at the STRB signal each time data is written to port B.

When in expanded multiplexed operating modes, all of the port B pins act as high order address output signals. During each MCU cycle, bits 8 through 15 of the address are output on the PB0-PB7 lines respectively.

2.1.11.3 Port C

While in single-chip operating modes, all port C pins are general-purpose input/output pins. Port C inputs can be latched by providing an input transition to the STRA signal. Port C may also be used in full handshake modes of parallel I/O where the STRA input and STRB output act as handshake control lines.

When in expanded multiplexed operating modes, all port C pins are configured as multiplexed address/data signals. During the address portion of each MCU cycle, bits 0 through 7 of the address are output on the PC0-PC7 lines. During the data portion of each MCU cycle (E high), pins 0 through 7 are bidirectional data signals (D0-D7). The direction of data at the port C pins is indicated by the R/$\overline{W}$ signal.

2.1.11.4 Port D

Port D pins 0-5 may be used for general-purpose I/O signals. Port D pins alternately serve as the serial communications interface (SCI) and serial peripheral interface (SPI) signals when those subsystems are enabled.

Pin PD0 is the receive data input (RxD) signal for the serial communication interface (SCI).

Pin PD1 is the transmit data output (TxD) signal for the SCI.

Pins PD2 through PD5 are dedicated to the SPI. PD2 is the master-in-slave-out (MISO) signal. PD3 is the master-out-slave-in (MOSI) signal. PD4 is the serial clock (SCK) signal and PD5 is the slave select ($\overline{SS}$) input.

2.1.11.5 Port E

Port E is used for general-purpose inputs and/or analog-to-digital (A/D) input channels. Reading port E during the sampling portion of an A/D conversion could cause very small disturbances and affect the accuracy of that result. If very high accuracy is required, avoid reading port E during conversions.

2.2 Operating Modes

There are four operating modes for the MC68HC11A8: single-chip operating mode, expanded multiplexed operating mode, special bootstrap operating mode, and special test operating mode.

Table 2-1 shows how the operating mode is selected. The following paragraphs describe these operating modes.

2.2.1 Single-Chip Operating Mode

In single-chip operating mode, the MC68HC11A8 functions as a monolithic microcontroller without external address or data buses. Port B, port C, strobe A, and strobe B function as general purpose I/O and handshake signals. Refer to **4 PARALLEL I/O** for additional information.

2.2.2 Expanded Multiplexed Operating Mode

In expanded multiplexed operating mode, the MC68HC11A8 has the capability of accessing a 64 Kbyte address space. This total address space includes the same on-chip memory addresses used for single-chip operating mode plus external peripheral and memory devices. The expansion bus is made up of port B and port C, and control signals AS and R/$\overline{W}$. **Figure 2-4** shows a recommended way of demultiplexing low order addresses from data at port C. The address, R/$\overline{W}$, and AS signals are active and valid for all bus cycles including accesses to internal memory locations.

Table 2-2 Port Signal Summary

Port-Bit	Single Chip and Bootstrap Mode	Expanded Multiplexed and Special Test Mode
A-0	PA0/IC3	PA0/IC3
A-1	PA1/IC2	PA1/IC2
A-2	PA2/IC1	PA2/IC1
A-3	PA3/OC5/OC1	PA3/OC5/OC1
A-4	PA4/OC4/OC1	PA4/OC4/OC1
A-5	PA5/OC3/OC1	PA5/OC3/OC1
A-6	PA6/OC2/OC1	PA6/OC2/OC1
A-7	PA7/PAI/OC1	PA7/PAI/OC1
B-0	PB0	A8
B-1	PB1	A9
B-2	PB2	A10
B-3	PB3	A11
B-4	PB4	A12
B-5	PB5	A13
B-6	PB6	A14
B-7	PB7	A15
C-0	PC0	A0/D0
C-1	PC1	A1/D1
C-2	PC2	A2/D2
C-3	PC3	A3/D3
C-4	PC4	A4/D4
C-5	PC5	A5/D5
C-6	PC6	A6/D6
C-7	PC7	A7/D7
D-0	PD0/RXD	PD0/RXD
D-1	PD1/TXD	PD1/TXD
D-2	PD2/MISO	PD2/MISO
D-3	PD3/MOSI	PD3/MOSI
D-4	PD4/SCK	PD4/SCK
D-5	PD5/SS	PD5/SS
	STRA	AS
	STRB	R/W
E-0	PE0/AN0	PE0/AN0
E-1	PE1/AN1	PE1/AN1
E-2	PE2/AN2	PE2/AN2
E-3	PE3/AN3	PE3/AN3
E-4	PE4/AN4##	PE4/AN4##
E-5	PE5/AN5##	PE5/AN5##
E-6	PE6/AN6##	PE6/AN6##
E-7	PE7/AN7##	PE7/AN7##

Not bonded in 48-pin versions

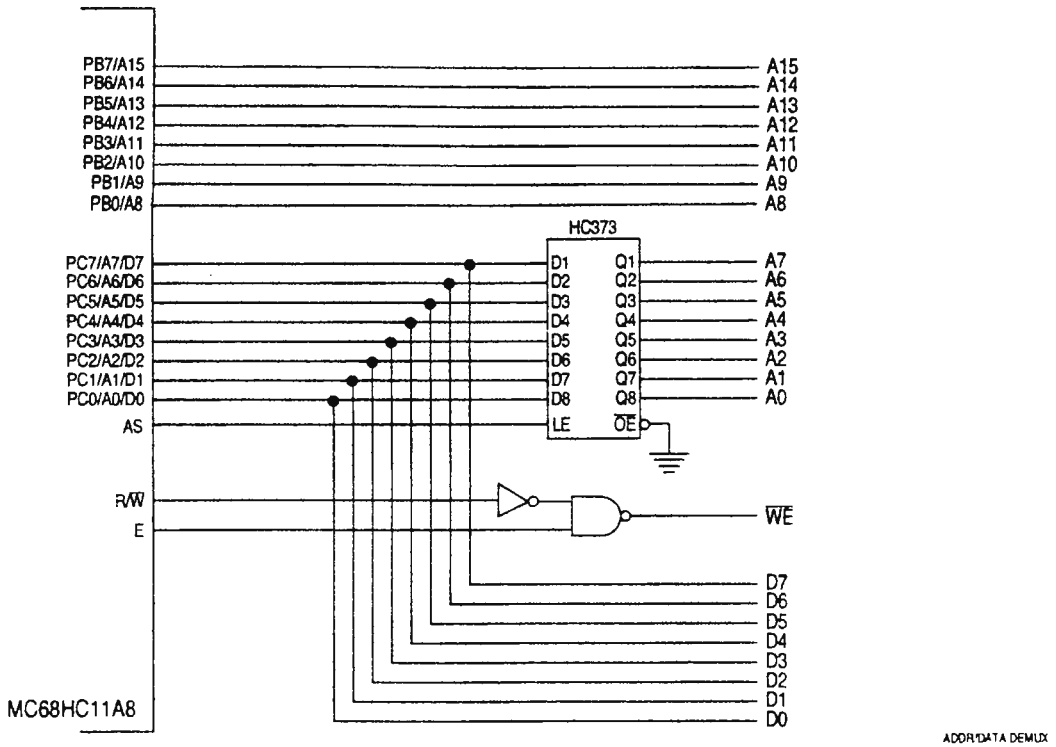


Figure 2-4 Address/Data Demultiplexing

2.2.3 Special Bootstrap Operating Mode

The bootstrap mode is considered a special operating mode as distinguished from the normal single-chip operating mode. This is a very versatile operating mode since there are essentially no limitations on the special purpose program that can be loaded into the internal RAM. The boot loader program is contained in the 192 byte bootstrap ROM. This ROM is enabled only if the MCU is reset in special bootstrap operating mode, and appears as internal memory space at locations $BF40-$BFFF. The boot loader program will use the SCI to read a 256 byte program into on-chip RAM at locations $0000-$00FF. After the character for address $00FF is received, control is automatically passed to that program at location $0000.

The MC68HC11A8 communicates through the SCI port. After reset in special bootstrap operating mode, the SCI is running at E clock/16 (7812 baud for E clock equal 2 MHz). If the security feature was specified and the security bit is set, $FF is output by the SCI transmitter. The EEPROM is then erased. If erasure is unsuccessful, $FF is output again and erasure is attempted again. Upon successful erasure of the EEPROM, all internal RAM is written over with $FF. The CONFIG register is then erased. The boot loader program now proceeds as though the part had not been in security mode.

If the part is not in security mode (or has completed the above erase sequence), a break character is output by the SCI transmitter. For normal use of the boot loader program, the user sends $FF to the SCI receiver at either E clock/16 (7812 baud for E clock = 2 MHz) or E clock/104 (1200 baud for E clock = 2 MHz).

NOTE

This $FF is not echoed through the SCI transmitter.

Now the user must download 256 bytes of program data to be put into RAM starting at location $0000. These characters are echoed through the transmitter. When loading is complete, the program jumps to location $0000 and begins executing that code.

If the SCI transmitter pin is to be used, an external pull-up resistor is required because port D pins are configured for wire-OR operation.

In special bootstrap operating mode the interrupt vectors are directed to RAM as shown in **Table 2-3**. This allows the user to use interrupts by way of a jump table. For example: to use the SWI interrupt, a jump instruction would be placed in RAM at locations $00F4, $00F5, and $00F6. When an SWI is encountered, the vector (which is in the boot loader ROM program) will direct program control to location $00F4 in RAM which in turn contains a JUMP instruction to the interrupt service routine.

Table 2-3 Bootstrap Mode Interrupt Vectors

Address	Vector
00C4	SCI
00C7	SPI
00CA	Pulse Accumulator Input Edge
00CD	Pulse Accumulator Overflow
00D0	Timer Overflow
00D3	Timer Output Compare 5
00D6	Timer Output Compare 4
00D9	Timer Output Compare 3
00DC	Timer Output Compare 2
00DF	Timer Output Compare 1
00E2	Timer Input Capture 3
00E5	Timer Input Capture 2
00E8	Timer Input Capture 1
00EB	Real Time Interrupt
00EE	IRQ
00F1	XIRQ
00F4	SWI
00F7	Illegal Opcode
00FA	COP Fail
00FD	Clock Monitor
BF40 (Boot)	Reset

2.2.4 Additional Boot Loader Program Options

The user may transmit a $55 (only at E clock/16) as the first character rather than the normal $FF. This will cause the program to jump directly to location $0000, skipping the download.

The user may tie the receiver to the transmitter (with an external pull-up resistor). This will cause the program to jump directly to the beginning of EEPROM ($B600). Another way to cause the program to jump directly to EEPROM is to transmit either a break or $00 as the first character rather than the normal $FF.

Note that none of these options bypass the security check and so do not compromise those customers using security.

Keep in mind that upon entry to the downloaded program at location $0000, some registers have been changed from their reset states. The SCI transmitter and receiver are enabled which cause port D pins 0 and 1 to be dedicated to SCI use. Also port D is configured for wired-OR operation. It may be necessary for the user to write to the SCCR2 and SPCR registers to disable the SCI and/or port D wire-OR operation.

2.2.5 Special Test Operating Mode

The test mode is a special operating mode intended primarily for factory testing. This mode is very similar to the expanded multiplexed operating mode. In special test operating mode, the reset and interrupt vectors are fetched from external memory locations $BFC0–$BFFF rather than $FFC0–$FFFF. There are no time limits for protection of the TMSK2, OPTION, and INIT registers, so these registers may be written repeatedly. Also a special TEST1 register is enabled which allows several factory test functions to be invoked.

The special test operating mode is not recommended for use by an end user because of the reduced system security; however, an end user may wish to come out of reset in special test operating mode. Then, after some initialization, the SMOD and MDA bits could be rewritten to select a normal operating mode to re-enable the protection features.

3 ON-CHIP MEMORY

This section describes the on-chip ROM, RAM, and EEPROM memories. The memory maps for each mode of operation are shown and the RAM and I/O mapping register (INIT) is described. The INIT register allows the on-chip RAM and the 64 control registers to be moved to suit the needs of a particular application.

3.1 Memory Maps

Composite memory maps for each mode of operation are shown in **Figure 3-1**. Memory locations are shown in the shaded areas and the contents of these shaded areas are shown to the right. These modes include single-chip, expanded multiplexed, special bootstrap, and special test.

Single-chip operating modes do not generate external addresses. Refer to **Table 3-1** for a full list of the registers.

3

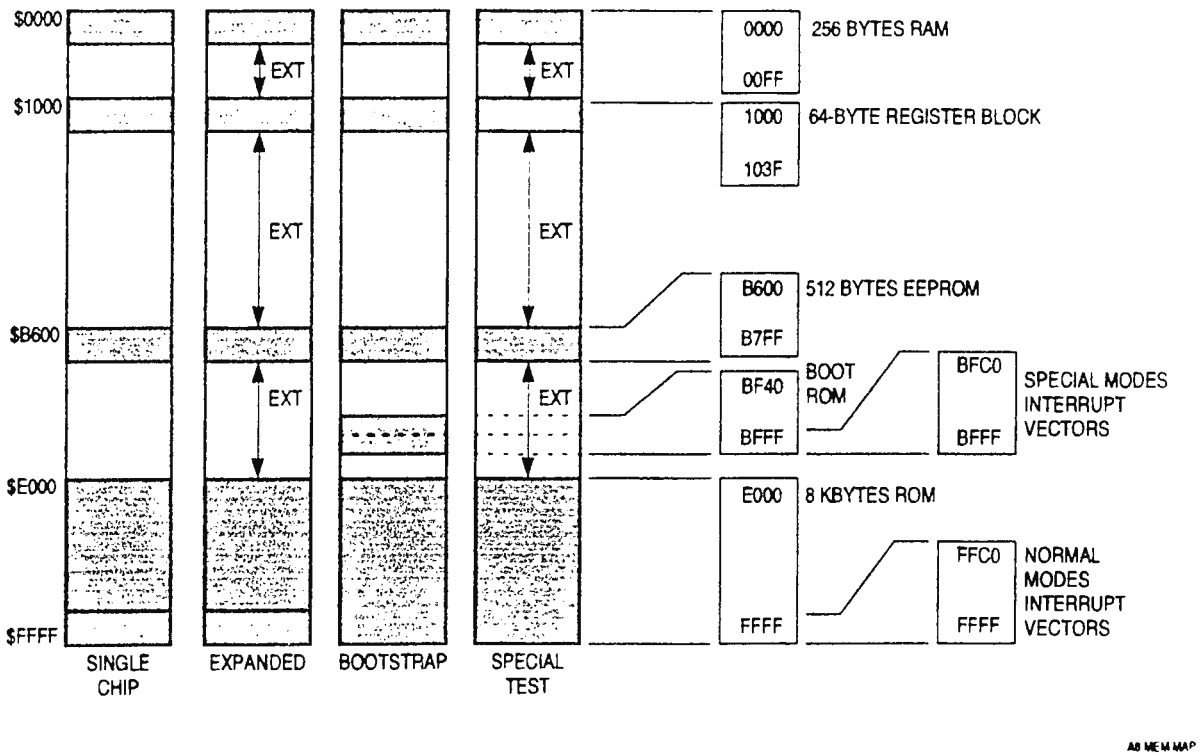


Figure 3-1 Memory Maps

Table 3-1 Register and Control Bit Assignments (Sheet 1 of 2)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		
$1000	Bit 7	–	–	–	–	–	–	Bit 0	PORTA	I/O Port A
$1001									Reserved	
$1002	STAF	STAI	CWOM	HNDS	OIN	PLS	EGA	INVB	PIOC	Parallel I/O Control Register
$1003	Bit 7	–	–	–	–	–	–	Bit 0	PORTC	I/O Port C
$1004	Bit 7	–	–	–	–	–	–	Bit 0	PORTB	Output Port B
$1005	Bit 7	–	–	–	–	–	–	Bit 0	PORTCL	Alternate Latched Port C
$1006									Reserved	
$1007	Bit 7	–	–	–	–	–	–	Bit 0	DDRC	Data Direction for Port C
$1008			Bit 5	–	–	–	–	Bit 0	PORTD	I/O Port D
$1009			Bit 5	–	–	–	–	Bit 0	DDRD	Data Direction for Port D
$100A	Bit 7	–	–	–	–	–	–	Bit 0	PORTE	Input Port E
$100B	FOC1	FOC2	FOC3	FOC4	FOC5				CFORC	Compare Force Register
$100C	OC1M7	OC1M6	OC1M5	OC1M4	OC1M3				OC1M	OC1 Action Mask Register
$100D	OC1D7	OC1D6	OC1D5	OC1D4	OC1D3				OC1D	OC1 Action Data Register
$100E	Bit 15	–	–	–	–	–	–	Bit 8	TCNT	Timer Counter Register
$100F	Bit 7	–	–	–	–	–	–	Bit 0		
$1010	Bit 15	–	–	–	–	–	–	Bit 8	TIC1	Input Capture 1 Register
$1011	Bit 7	–	–	–	–	–	–	Bit 0		
$1012	Bit 15	–	–	–	–	–	–	Bit 8	TIC2	Input Capture 2 Register
$1013	Bit 7	–	–	–	–	–	–	Bit 0		
$1014	Bit 15	–	–	–	–	–	–	Bit 8	TIC3	Input Capture 3 Register
$1015	Bit 7	–	–	–	–	–	–	Bit 0		
$1016	Bit 15	–	–	–	–	–	–	Bit 8	TOC1	Output Compare 1 Register
$1017	Bit 7	–	–	–	–	–	–	Bit 0		
$1018	Bit 15	–	–	–	–	–	–	Bit 8	TOC2	Output Compare 2 Register
$1019	Bit 7	–	–	–	–	–	–	Bit 0		
$101A	Bit 15	–	–	–	–	–	–	Bit 8	TOC3	Output Compare 3 Register
$101B	Bit 7	–	–	–	–	–	–	Bit 0		
$101C	Bit 15	–	–	–	–	–	–	Bit 8	TOC4	Output Compare 4 Register
$101D	Bit 7	–	–	–	–	–	–	Bit 0		
$101E	Bit 15	–	–	–	–	–	–	Bit 8	TCO5	Output Compare 5 Register
$101F	Bit 7	–	–	–	–	–	–	Bit 0		

Table 3-1 Register and Control Bit Assignments (Sheet 2 of 2)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		
$1020	OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5	TCTL1	Timer Control Register 1
$1021			EDG1B	EDG1A	EDG2B	EDG2A	EDG3B	EDG3A	TCTL2	Timer Control Register 2
$1022	OC1I	OC2I	OC3I	OC4I	OC5I	IC1I	IC2I	IC3I	TMSK1	Timer Interrupt Mask Register 1
$1023	OC1F	OC2F	OC3F	OC4F	OC5F	IC1F	IC2F	IC3F	TFLG1	Timer Interrupt Flag Register 1
$1024	TOI	RTII	PAOVI	PAII			PR1	PR0	TMSK2	Timer Interrupt Mask Register 2
$1025	TOF	RTIF	PAOVF	PAIF					TFLG2	Timer Interrupt Flag Register 2
$1026	DDRA7	PAEN	PAMOD	PEDGE			RTR1	RTR0	PACTL	Pulse Accumulator Control Register
$1027	Bit 7	–	–	–	–	–	–	Bit 0	PACNT	Pulse Accumulator Count Register
$1028	SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR	SPI Control Register
$1029	SPIF	WCOL		MODF					SPSR	SPI Status Register
$102A	Bit 7	–	–	–	–	–	–	Bit 0	SPDR	SPI Data Register
$102B	TCLR		SCP1	SCP0	RCKB	SCR2	SCR1	SCR0	BAUD	SCI Baud Rate Control
$102C	R8	T8		M	WAKE				SCCR1	SCI Control Register 1
$102D	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCCR2	SCI Control Register 2
$102E	TRDE	TC	RDRF	IDLE	OR	NF	FE		SCSR	SCI Status Register
$102F	Bit 7	–	–	–	–	–	–	Bit 0	SCDR	SCI Data (Read RDR, Write TDR)
$1030	CCF		SCAN	MULT	CD	CC	CB	CA	ADCTL	A/D Control Register
$1031	Bit 7	–	–	–	–	–	–	Bit 0	ADR1	A/D Result Register 1
$1032	Bit 7	–	–	–	–	–	–	Bit 0	ADR2	A/D Result Register 2
$1033	Bit 7	–	–	–	–	–	–	Bit 0	ADR3	A/D Result Register 3
$1034	Bit 7	–	–	–	–	–	–	Bit 0	ADR4	A/D Result Register 4
$1035 thru $1038									Reserved	
$1039	ADPU	CSEL	IRQE	DLY	CME		CR1	CR0	OPTION	System Configuration Options
$103A	Bit 7	–	–	–	–	–	–	Bit 0	COPRST	Arm/Reset COP Timer Circuitry
$103B	ODD	EVEN		BYTE	ROW	ERASE	EELAT	EEPGM	PPROG	EEPROM Program Control Register
$103C	RBOOT	SMOD	MDA	IRV	PSEL3	PSEL2	PSEL1	PSEL0	HPRIO	Highest Priority I-Bit Int and Misc
$103D	RAM3	RAM2	RAM1	RAM0	REG3	REG2	REG1	REG0	INIT	RAM and I/O Mapping Register
$103E	TILOP		OCCR	CBYP	DISR	FCM	FCOP	TCON	TEST1	Factory TEST Control Register
$103F	–	–	–	–	NOSEC	NOCOP	ROMON	EEON	CONFIG	COP, ROM, and EEPROM Enables

In expanded multiplexed operating modes, memory locations are basically the same as the single- chip operating modes; however, the locations between the shaded areas (designated EXT) are for externally addressed memory and I/O. If an external memory or I/O device is located to overlap an enabled internal resource, the internal resource will take priority. For reads of such an address the data (if any) driving the port C data inputs is ignored and will not result in any harmful conflict with the internal read. For writes to such an address data is driven out of the port C data pins as well as to the internal location. No external devices should drive port C during write accesses to internal locations; however, there is normally no conflict since the external address decode and/ or data direction control should incorporate the R/$\overline{W}$ signal in their development. The R/$\overline{W}$, AS, address, and write data signals are valid for all accesses including accesses to internal memory and registers.

The special bootstrap operating mode memory locations are similar to the single-chip operating mode memory locations except that a bootstrap program at memory locations $BF40 through $BFFF is enabled. The reset and interrupt vectors are addressed at $BFC0–$BFFF while in the special bootstrap operating mode. These vector addresses are within the 192 byte memory used for the bootstrap program.

The special test operating mode memory map is the same as the expanded multiplexed operating mode memory map except that the reset and interrupt vectors are located at external memory locations $BFC0–$BFFF.

3.2 RAM and I/O Mapping Register (INIT)

There are 64 internal registers which are used to control the operation of the MCU. These registers can be relocated on 4K boundaries within the memory space, using the INIT register. Refer to **Table 3-1** for a complete list of the registers. The registers and control bits are explained throughout this document.

The INIT register is a special-purpose 8-bit register which may be used during initialization to change the default locations of RAM and control registers within the MCU memory map. It may be written to only once within the initial 64 E clock cycles after a reset and thereafter becomes a read-only register.

	7	6	5	4	3	2	1	0	
$103D	RAM3	RAM2	RAM1	RAM0	REG3	REG2	REG1	REG0	INIT
RESET	0	0	0	0	0	0	0	1	

The default starting address for internal RAM is $0000 and the default starting address for the 64 control registers is $1000 (the INIT register is set to $01 at reset). The upper four bits of the INIT register specify the starting address for the 256 byte RAM and the lower four bits of INIT specify the starting address for the 64 control registers. These four bits are matched to the upper four bits of the 16-bit address.

Throughout this document, the control register addresses will be displayed with the high-order digit shown as a bold "**1**" to indicate that the register block may be relocated to some 4K memory page other than its default position of $1000-$103F.

Note that if the RAM is relocated to either $E000 or $F000, which is in conflict with the internal ROM, (no conflict if the ROMON bit in the configuration register is zero), RAM will take priority and the conflicting ROM will become inaccessible. Also, if the 64 control registers are relocated so that they conflict with the RAM and/or ROM, then the 64 control registers take priority and the RAM and/or ROM at those locations become inaccessible. No harmful conflicts result, the lower priority resources simply become inaccessible. Similarly, if an internal resource conflicts with an external device no harmful conflict results. Data from the external device will not be applied to the internal data bus and cannot interfere with the internal read.

Note that there are unused register locations in the 64 byte control register block. Reads of these unused registers will return data from the undriven internal data bus and not from another resource that happens to be located at the same address.

3.3 ROM

The internal 8K ROM occupies the highest 8K of the memory map ($E000–$FFFF). This ROM is disabled when the ROMON bit in the CONFIG register is clear. The ROMON bit is implemented with an EEPROM cell and is programmed using the same procedures for programming the on-chip EEPROM. For further information refer to **3.5.3 System Configuration Register (CONFIG)**.

In the single-chip operating mode, internal ROM is enabled regardless of the state of the ROMON bit.

There is also a 192 byte mask programmed boot ROM in the MC68HC11A8. This bootstrap program ROM controls the operation of the special bootstrap operating mode and is only enabled following reset in the special bootstrap operating mode. For more information refer to **2.2.3 Special Bootstrap Operating Mode.**

3.4 RAM

The 256 byte internal RAM may be relocated during initialization by writing to the INIT register. The reset default position is $0000 through $00FF. This RAM is implemented with static cells and retains its contents during the WAIT and STOP modes.

The contents of the 256-byte RAM can also be retained by supplying a low current backup power source to the MODB/V_{STBY} pin. When using a standby power source, V_{DD} may be removed; however, $\overline{RESET}$ must go low before V_{DD} is removed and remain low until V_{DD} has been restored.

3.5 EEPROM

The 512 bytes of EEPROM are located at $B600 through $B7FF and have the same read cycle time as the internal ROM. The write (or programming) mechanism for the EEPROM is controlled by the PPROG register. The EEPROM is disabled when the EEON bit in the CONFIG register is zero. The EEON bit is implemented with an EEPROM cell.

7 ANALOG-TO-DIGITAL CONVERTER

The MC68HC11A8 includes an 8-channel, multiplexed-input, successive approximation, analog-to-digital (A/D) converter with sample and hold to minimize conversion errors caused by rapidly changing input signals. Two dedicated lines (V_{RL}, V_{RH}) are provided for the reference voltage inputs. These pins may be connected to a separate or isolated power supply to ensure full accuracy of the A/D conversion. The 8-bit A/D converter has a total error of ±1 LSB which includes ± 1/2 LSB of quantization error and accepts analog inputs which range from V_{RL} to V_{RH}. Smaller analog input ranges can also be obtained by adjusting V_{RH} and V_{RL} to the desired upper and lower limits. Conversion is specified and tested for V_{RL} = 0 V and V_{RH} = 5 V ± 10%; however, laboratory characterization over the full temperature range indicates little or no degradation with V_{RH}-V_{RL} as low as 2.5 to 3 V. The A/D system can be operated with V_{RH} below V_{DD} and/or V_{RL} above V_{SS} as long as V_{RH} is above V_{RL} by enough to support the conversions (2.5 to 5.0 V). Each conversion is accomplished in 32 MCU E clock cycles, provided the E clock rate is greater than 750 kHz. For systems which operate at clock rates less than 750 kHz, an internal R-C oscillator must be used to clock the A/D system. The internal R-C oscillator is selected by setting the CSEL bit in the OPTION register.

NOTE

Only four A/D input channels are available in the 48-pin version.

7.1 Conversion Process

The A/D converter is ratiometric. An input voltage equal to V_{RL} converts to $00 and an input voltage equal to V_{RH} converts to $FF (full scale), with no overflow indication. For ratiometric conversions, the source of each analog input should use V_{RH} as the supply voltage and be referenced to V_{RL}.

Figure 7-1 shows the detailed sequence for a set of four conversions. This sequence begins one E clock cycle after a write to the A/D control/status register (ADCTL). **Figure 7-2** shows a model of the port E A/D channel inputs. This model is useful for understanding the effects of external circuitry on the accuracy of A/D conversions.

7.2 Channel Assignments

A multiplexer allows the single A/D converter to select one of sixteen analog signals. Eight of these channels correspond to port E input lines to the MCU, four of the channels are for internal reference points or test functions, and four channels are reserved for future use. **Table 7-1** shows the signals selected by the four channel select control bits.

7.3 Single-Channel Operation

There are two variations of single-channel operation. In the first variation (SCAN = 0), the single selected channel is converted four consecutive times with the first result being stored in A/D result register 1 (ADR1) and the fourth result being stored in register ADR4. After the fourth conversion is complete, all conversion activity is halted until a new conversion command is written to the ADCTL register. In the second variation (SCAN = 1), conversions continue to be performed on the selected channel with the fifth conversion being stored in register ADR1 (overwriting the first conversion result), the sixth conversion overwrites ADR2, and so on.

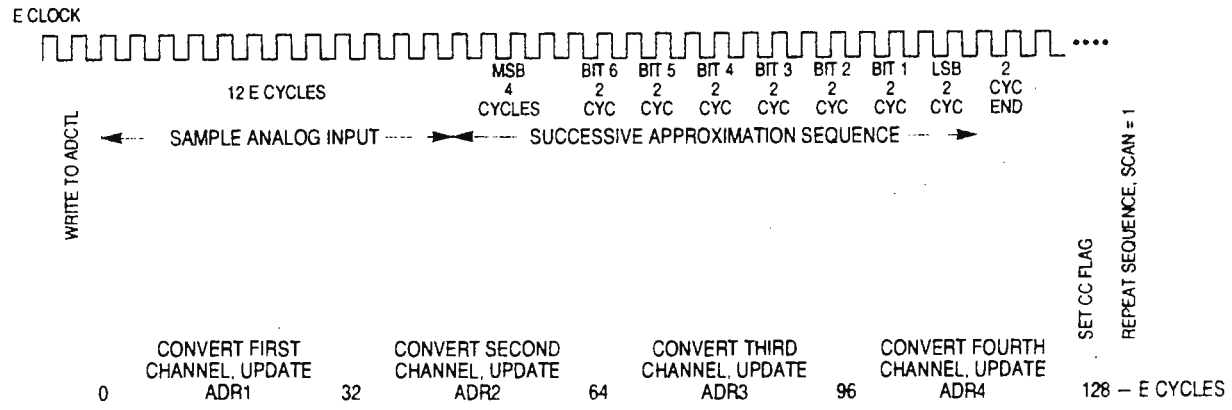
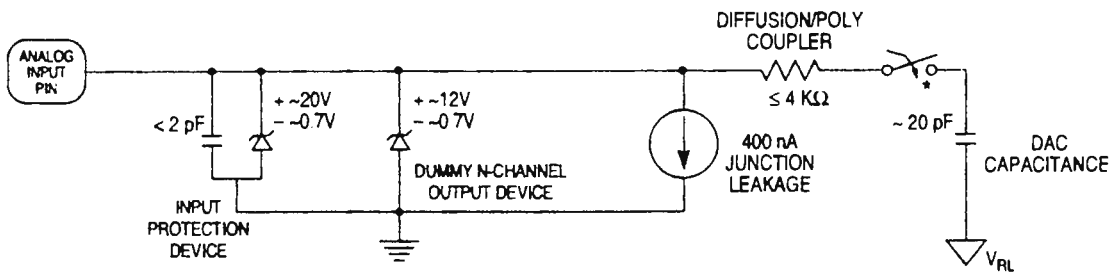


Figure 7-1 A/D Conversion Sequence



* THIS ANALOG SWITCH IS CLOSED ONLY DURING THE 12-CYCLE SAMPLE TIME.

Figure 7-2 A/D Pin Model

7.4 Multiple-Channel Operation

There are two variations in multiple-channel operation. In the first variation (SCAN = 0), the selected group of four channels are converted, one time each, with the first result being stored in register ADR1 and the fourth result being stored in register ADR4. After the fourth conversion is complete, all conversion activity is halted until a new conversion command is written to the ADCTL register. In the second variation (SCAN = 1), conversions continue to be performed on the selected group of channels with the fifth conversion being stored in register ADR1 (replacing the earlier conversion result for the first channel in the group), the sixth conversion overwrites ADR2, and so on.

7.5 Operation in STOP and WAIT Modes

If a conversion sequence is still in process when either the STOP or WAIT mode is entered, the conversion of the current channel is suspended. When the MCU resumes normal operation, that channel will be re-sampled and the conversion sequence resumed. As the MCU exits the WAIT mode, the A/D circuits are stable and valid results can be obtained on the first conversion. However, in STOP mode, all analog bias currents are disabled and it becomes necessary to allow a stabilization period when leaving the STOP mode. If the STOP mode is exited with a delay, there will be enough time for these circuits to stabilize before the first conversion. If the STOP mode is exited with no delay (DLY bit in OPTION register equal to zero), sufficient time must be allowed for the A/D circuitry to stabilize to avoid invalid results (see **7.8 A/D Power-Up and Clock Select**).

7.6 A/D Control/Status Register (ADCTL)

All bits in this register may be read or written, except bit 7 which is a read-only status indicator and bit 6 which always reads as a zero.

	7	6	5	4	3	2	1	0	
$1030	CCF	0	SCAN	MULT	CD	CC	CB	CA	ADCTL
RESET	0	0	U	U	U	U	U	U	

CCF — Conversions Complete Flag

This read-only status indicator is set when all four A/D result registers contain valid conversion results. Each time the ADCTL register is written, this bit is automatically cleared to zero and a conversion sequence is started. In the continuous modes, conversions continue in a round-robin fashion and the result registers continue to be updated with current data even though the CCF bit remains set.

NOTE

The user must write to register ADCTL to initiate conversion. To abort a conversion in progress, write to the ADCTL register and a new conversion sequence is initiated immediately.

Bit 6 — Not Implemented

This bit always reads zero.

SCAN — Continuous Scan Control

When this control bit is clear, the four requested conversions are performed once to fill the four result registers. When this control bit is set, conversions continue in a round-robin fashion with the result registers being updated as data becomes available.

MULT — Multiple-Channel/Single Channel Control

When this bit is clear, the A/D system is configured to perform four consecutive conversions on the single channel specified by the four channel select bits CD through CA (bits [3:0] of the ADCTL register). When this bit is set, the A/D system is configured to perform a conversion on each of four channels where each result register corresponds to one channel.

CAUTION

When the multiple channel continuous scan mode is used, extra care is needed in the design of circuitry driving the A/D inputs. Refer to the A/D Pin Model and A/D Conversion Sequence figures in addition to the following discussion. The charge on the capacitive DAC array prior to the sample time is related to the voltage on the previously converted channel. A charge share situation exists between the internal DAC capacitance and the external circuit capacitance. Although the amount of charge involved is small the rate at which it is repeated is every 64 microseconds for an E clock of 2 MHz. The RC charging rate of the external circuit must be balanced against this charge sharing effect to avoid accuracy errors.

CD — Channel Select D
CC — Channel Select C
CB — Channel Select B
CA — Channel Select A

These four bits are used to select one of 16 A/D channels (see **Table 7-1**). When a multiple channel mode is selected (MULT = 1), the two least-significant channel select bits (CB and CA) have no meaning and the CD and CC bits specify which group of four channels are to be converted. The channels selected by the four channel select control bits are shown in **Table 7-1**.

Table 7-1 Analog-to-Digital Channel Assignments

CD	CC	CB	CA	Channel Signal	Result in ADRx if MULT=1
0	0	0	0	AN0	ADR1
0	0	0	1	AN1	ADR2
0	0	1	0	AN2	ADR3
0	0	1	1	AN3	ADR4
0	1	0	0	AN4*	ADR1
0	1	0	1	AN5*	ADR2
0	1	1	0	AN6*	ADR3
0	1	1	1	AN7*	ADR4
1	0	0	0	Reserved	ADR1
1	0	0	1	Reserved	ADR2
1	0	1	0	Reserved	ADR3
1	0	1	1	Reserved	ADR4
1	1	0	0	V_{RH} Pin**	ADR1
1	1	0	1	V_{RL} Pin**	ADR2
1	1	1	0	$(V_{RH})/2$**	ADR3
1	1	1	1	Reserved**	ADR4

*Not available in 48-pin package versions.
**This group of channels used during factory test.

7.7 A/D Result Registers 1, 2, 3, and 4 (ADR1, ADR2, ADR3, and ADR4)

The A/D result registers are read-only registers used to hold an 8-bit conversion result. Writes to these registers have no effect. Data in the A/D result registers is valid when the CCF flag bit in the ADCTL register is set, indicating a conversion sequence is complete. If conversion results are needed sooner refer to **Figure 7-1**. For example the ADR1 result is valid 33 cycles after an ADCTL write. Refer to the A/D channel assignments in **Table 7-1** for the relationship between the channels and the result registers.

7.8 A/D Power-Up and Clock Select

A/D power-up is controlled by bit 7 (ADPU) of the OPTION register. When ADPU is cleared, power to the A/D system is disabled. When ADPU is set, the A/D system is enabled. A delay of as much as 100 microseconds is required after turning on the A/D converter to allow the analog bias voltages to stabilize.

Clock select is controlled by bit 6 (CSEL) of the OPTION register. When CSEL is cleared, the A/D system uses the system E clock. When CSEL is set, the A/D system uses an internal R-C clock source, which runs at about 1.5 MHz. The MCU E clock is not suitable to drive the A/D system if it is operating below 750 kHz, in which case the R-C internal clock should be selected. A delay of 10 ms is required after changing CSEL from zero to one to allow the R-C oscillator to start and internal bias voltages to settle. Refer to **9.1.5 Configuration Options Register (OPTION)** for additional information. Note that the CSEL control bit also enables a separate R-C oscillator to drive the EEPROM charge pump.

When the A/D system is operating with the MCU E clock, all switching and comparator operations are synchronized to the MCU clocks. This allows the comparator results to be sampled at quiet clock times to minimize noise errors. The internal R-C oscillator is asynchronous to the MCU clock so noise will affect A/D results more while CSEL = 1.

A ELECTRICAL CHARACTERISTICS

Table A-1 Maximum Rating

Rating	Symbol	Value	Unit
Supply Voltage	V_{DD}	– 0.3 to + 7.0	V
Input Voltage	V_{in}	– 0.3 to + 7.0	V
Operating Temperature Range MC68HC11A8 MC68HC11A8C MC68HC11A8V MC68HC11A8M MC68L11A8	T_A	T_L to T_H 0 to 70 – 40 to 85 – 40 to 105 – 40 to 125 – 20 to 70	°C
Storage Temperature Range	T_{stg}	– 55 to 150	°C
Current Drain per Pin* Excluding V_{DD}, V_{SS}, V_{RH}, and V_{RL}	I_D	25	mA

*One pin at a time, observing maximum power dissipation limits.

Internal circuitry protects the inputs against damage caused by high static voltages or electric fields; however, normal precautions are necessary to avoid application of any voltage higher than maximum-rated voltages to this high-impedance circuit. Extended operation at the maximum ratings can adversely affect device reliability. Tying unused inputs to an appropriate logic voltage level (either GND or V_{DD}) enhances reliability of operation.

A

Table A-2 Thermal Characteristics

Characteristic	Symbol	Value	Unit
Average Junction Temperature	T_J	$T_A + (P_D \times \Theta_{JA})$	°C
Ambient Temperature	T_A	User-determined	°C
Package Thermal Resistance (Junction-to-Ambient) 52-Pin Plastic Quad Pack (PLCC) 48-Pin Plastic Dual In-Line Package (DIP)	Θ_{JA}	 50 40	°C/W
Total Power Dissipation	P_D	$P_{INT} + P_{I/O}$ $K \div (T_J + 273°C)$ (Note 1)	W
Device Internal Power Dissipation	P_{INT}	$I_{DD} \times V_{DD}$	W
I/O Pin Power Dissipation	$P_{I/O}$ (Note 2)	User-determined	W
A Constant	K	$P_D \times (T_A + 273°C) +$ $\Theta_{JA} \times P_D 2$ (Note 3)	W · °C

NOTES:

1. This is an approximate value, neglecting $P_{I/O}$.
2. For most applications $P_{I/O} \ll P_{INT}$ and can be neglected.
3. K is a constant pertaining to the device. Solve for K with a known T_A and a measured P_D (at equilibrium). Use this value of K to solve for P_D and T_J iteratively for any value of T_A.

Table A-3 DC Electrical Characteristics

V_{DD} = 5.0 Vdc ± 10%, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H, unless otherwise noted

Characteristics		Symbol	Min	Max	Unit
Output Voltage (Note 1)	All Outputs except XTAL	V_{OL}	—	0.1	V
	All Outputs Except XTAL, $\overline{\text{RESET}}$,	V_{OH}	V_{DD} – 01	—	
I_{Load} =± 10.0 µA	and MODA				
Output High Voltage (Note 1)	All Outputs Except XTAL,	V_{OH}	V_{DD} – 0.8	—	V
I_{Load} = – 0.8 mA, V_{DD} = 4.5 V	$\overline{\text{RESET}}$, and MODA				
Output Low Voltage	All Outputs Except XTAL	V_{OL}	—	0.4	V
I_{Load} = 1.6 mA					
Input High Voltage	All Inputs Except $\overline{\text{RESET}}$	V_{IH}	0.7 x V_{DD}	V_{DD} + 0.3	V
	$\overline{\text{RESET}}$		0.8 x V_{DD}	V_{DD} + 0.3	
Input Low Voltage	All Inputs	V_{IL}	V_{SS} – 0.3	0.2 x V_{DD}	V
I/O Ports, Three-State Leakage	PA7, PC0-PC7, PD0-PD5,	I_{OZ}	—	±10	µA
V_{in} = V_{IH} or V_{IL}	AS/STRA, MODA/$\overline{\text{LIR}}$, $\overline{\text{RESET}}$				
Input Leakage Current (Note 2)		I_{in}			µA
V_{in} = V_{DD} or V_{SS}	PA0-PA2, $\overline{\text{IRQ}}$, $\overline{\text{XIRQ}}$		—	±1	
V_{in} = V_{DD} or V_{SS}	MODB/V_{STBY}		—	±10	
RAM Standby Voltage	Power down	V_{SB}	4.0	V_{DD}	V
RAM Standby Current	Power down	I_{SB}	—	10	µA
Total Supply Current (Note 3)					
RUN:		I_{DD}			mA
Single-Chip Mode		dc – 2 MHz	—	15	
		3 MHz	—	27	
Expanded Multiplexed Mode		dc – 2 MHz	—	27	
		3 MHz	—	35	
WAIT:		W_{IDD}			mA
All Peripheral Functions Shut Down					
Single-Chip Mode		dc – 2 MHz	—	6	
		3 MHz	—	15	
Expanded Multiplexed Mode		dc – 2 MHz	—	10	
		3 MHz	—	20	
STOP:		S_{IDD}			µA
No Clocks, Single-Chip Mode		dc – 2 MHz	—	50	
		3 MHz	—	150	
Input Capacitance	PA0-PA2, PE0-PE7, $\overline{\text{IRQ}}$, $\overline{\text{XIRQ}}$, EXTAL	C_{in}	—	8	pF
	PA7, PC0-PC7, PD0-PD5, AS/STRA, MODA/$\overline{\text{LIR}}$, $\overline{\text{RESET}}$		—	12	
Power Dissipation		P_D			mW
	Single-Chip Mode	2 MHz	—	85	
	Expanded Multiplexed Mode		—	150	
	Single-Chip Mode	3 MHz	—	150	
	Expanded Multiplexed Mode		—	195	

NOTES:

1. V_{OH} specification for $\overline{\text{RESET}}$ and MODA is not applicable because they are open-drain pins. V_{OH} specification not applicable to ports C and D in wired-OR mode.
2. Refer to A/D specification for leakage current for port E.
3. EXTAL is driven with a square wave, and
   t_{cyc} = 500 ns for 2 MHz rating;
   t_{cyc} = 333 ns for 3 MHz rating.
   $V_{IL} \leq 0.2$ V
   $V_{IH} \geq V_{DD} - 0.2$ V
   No dc loads.

Table A-3a DC Electrical Characteristics (MC68L11A8)

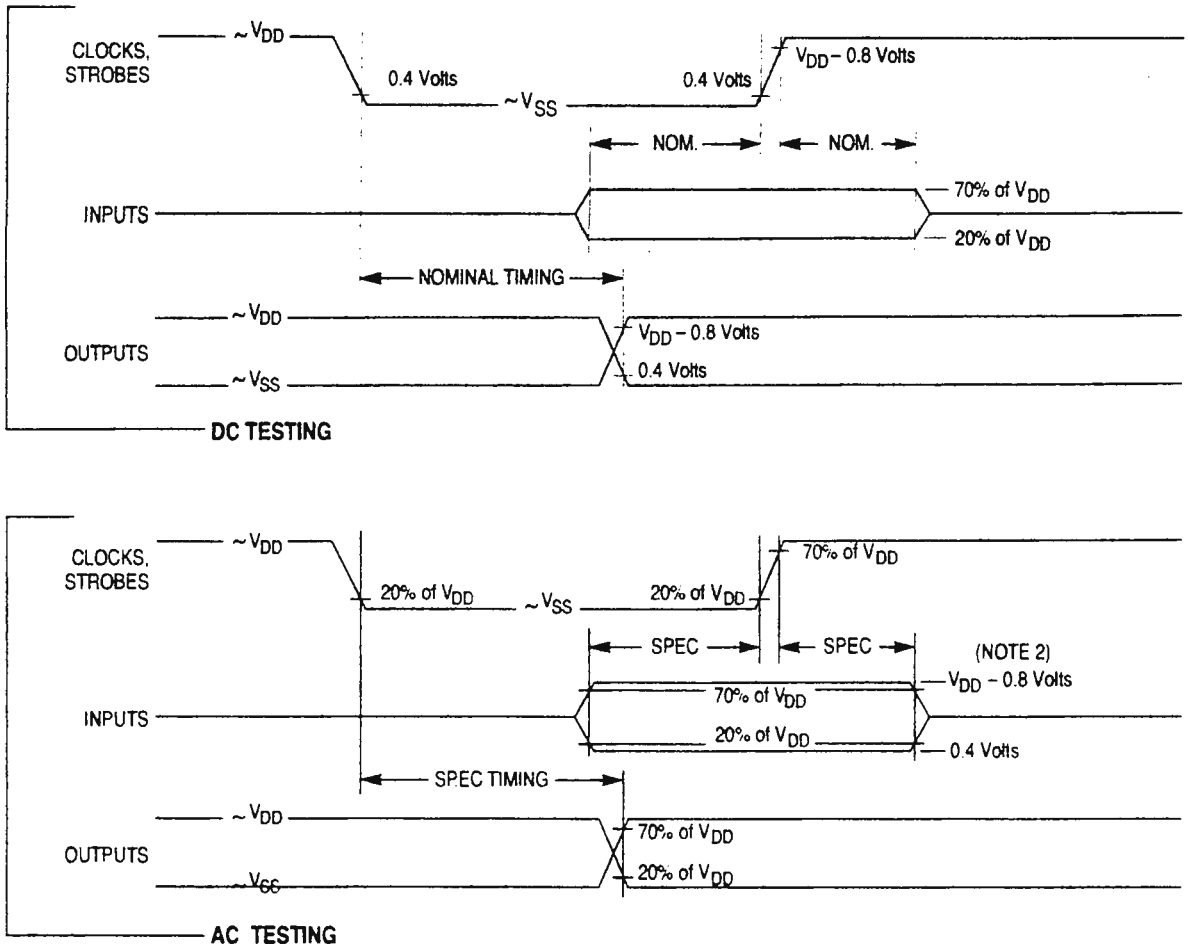
V_{DD} = 3.0 Vdc to 5.5 Vdc, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H, unless otherwise noted

Characteristics		Symbol	Min	Max	Unit
Output Voltage (Note 1)	All Outputs except XTAL	V_{OL}	—	0.1	V
	All Outputs Except XTAL, $\overline{\text{RESET}}$,	V_{OH}	V_{DD} – 01	—	
I_{Load} =± 10.0 µA	and MODA				
Output High Voltage (Note 1)	All Outputs Except XTAL,	V_{OH}	V_{DD} – 0.8	3	V
I_{Load} = – 0.8 mA, V_{DD} = 4.5 V	$\overline{\text{RESET}}$, and MODA				
Output Low Voltage	All Outputs Except XTAL	V_{OL}	—	0.4	V
I_{Load} = 1.6 mA					
Input High Voltage	All Inputs Except $\overline{\text{RESET}}$	V_{IH}	0.7 x V_{DD}	V_{DD} + 0.3	V
	$\overline{\text{RESET}}$		0.8 x V_{DD}	V_{DD} + 0.3	
Input Low VoltageAll Inputs		V_{IL}	V_{SS} – 0.3	0.2 x V_{DD}	V
I/O Ports, Three-State Leakage	PA7, PC0-PC7, PD0-PD5,	I_{OZ}	—	±10	µA
V_{in} = V_{IH} or V_{IL}	AS/STRA, MODA/$\overline{\text{LIR}}$, $\overline{\text{RESET}}$				
Input Leakage Current (Note 2)		I_{in}			µA
V_{in} = V_{DD} or V_{SS}	PA0-PA2, $\overline{\text{IRQ}}$, $\overline{\text{XIRQ}}$		—	±1	
V_{in} = V_{DD} or V_{SS}	MODB/V_{STBY}		—	±10	
RAM Standby Voltage	Power down	V_{SB}	2.0	V_{DD}	V
RAM Standby Current	Power down	I_{SB}	—	10	µA
Total Supply Current (Note 3)					
RUN:		I_{DD}			mA
Single-Chip Mode		dc – 1 MHz	—	4	
		2MHz	—	8	
Expanded Multiplexed Mode		dc – 1 MHz	—	7	
		2 MHz	—	14	
WAIT:		W_{IDD}			mA
All Peripheral Functions Shut Down					
Single-Chip Mode		dc – 1 MHz	—	3	
		2 MHz	—	6	
Expanded Multiplexed Mode		dc – 1 MHz	—	2.5	
		2 MHz	—	5	
STOP:		S_{IDD}			µA
No Clocks, Single-Chip Mode		dc – 1 MHz	—	25	
		2 MHz	—	25	
Input Capacitance	PA0-PA2, PE0-PE7, $\overline{\text{IRQ}}$, $\overline{\text{XIRQ}}$, EXTAL	C_{in}	—	8	pF
	PA7, PC0-PC7, PD0-PD5, AS/STRA, MODA/$\overline{\text{LIR}}$, $\overline{\text{RESET}}$		—	12	
Power Dissipation		P_D			mW
	Single-Chip Mode	1 MHz	—	12	
	Expanded Multiplexed Mode		—	21	
	Single-Chip Mode	2 MHz	—	24	
	Expanded Multiplexed Mode		—	42	

NOTES:

1. V_{OH} specification for $\overline{\text{RESET}}$ and MODA is not applicable because they are open-drain pins. V_{OH} specification not applicable to ports C and D in wired-OR mode.
2. Refer to A/D specification for leakage current for port E.
3. EXTAL is driven with a square wave, and
   t_{cyc} = 1000 ns for 1MHz rating;
   t_{cyc} = 500 ns for 2 MHz rating.
   $V_{IL} \leq 0.2$ V
   $V_{IH} \geq V_{DD} - 0.2$ V
   No dc loads.

A



NOTES:

1 Full test loads are applied during all DC electrical tests and AC timing measurements.

2 During AC timing measurements, inputs are driven to 0.4 volts and VDD – 0.8 volts while timing measurements are taken at the 20% and 70% of V_{DD} points

TEST METHODS

Figure A-1 Test Methods

Table A-4 Control Timing

V_{DD} = 5.0 Vdc ± 10%, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H

Characteristic	Symbol	1.0 MHz		2.0 MHz		3.0 MHz		Unit
		Min	Max	Min	Max	Min	Max	
Frequency of Operation	f_o	dc	1.0	dc	2.0	dc	3.0	MHz
E-Clock Period	t_{cyc}	1000	—	500	—	333	—	ns
Crystal Frequency	f_{XTAL}	—	4.0	—	8.0	—	12.0	MHz
External Oscillator Frequency	4 f_o	dc	4.0	dc	8.0	dc	12.0	MHz
Processor Control Setup Time t_{PCSU} = 1/4 t_{cyc} + 50 ns	t_{PCSU}	300	—	175	—	133	—	ns
Reset Input Pulse Width (To Guarantee External Reset Vector) (Note 1) (Minimum Input Time; Can Be Preempted by Internal Reset)	PW_{RSTL}	8 1	— —	8 1	— —	8 1	— —	t_{cyc}
Mode Programming Setup Time	t_{MPS}	2	—	2	—	2	—	t_{cyc}
Mode Programming Hold Time	t_{MPH}	10	—	10	—	10	—	ns
Interrupt Pulse Width, $\overline{IRQ}$ Edge-Sensitive Mode PW_{IRQ} = t_{cyc} + 20 ns	PW_{IRQ}	1020	—	520	—	353	—	ns
Wait Recovery Start-up Time	t_{WRS}	—	4	—	4	—	4	t_{cyc}
Timer Pulse Width Input Capture, Pulse Accumulator Input PW_{TIM} = t_{cyc} + 20 ns	PW_{TIM}	1020	—	520	—	353	—	ns

NOTES:

1. $\overline{RESET}$ is recognized during the first clock cycle it is held low. Internal circuitry then drives the pin low for four clock cycles, releases the pin, and samples the pin level two cycles later to determine the source of the interrupt. Refer to **9 RESETS, INTERRUPTS, AND LOW POWER MODES** for further detail.
2. All timing is shown with respect to 20% V_{DD} and 70% V_{DD}, unless otherwise noted.

A

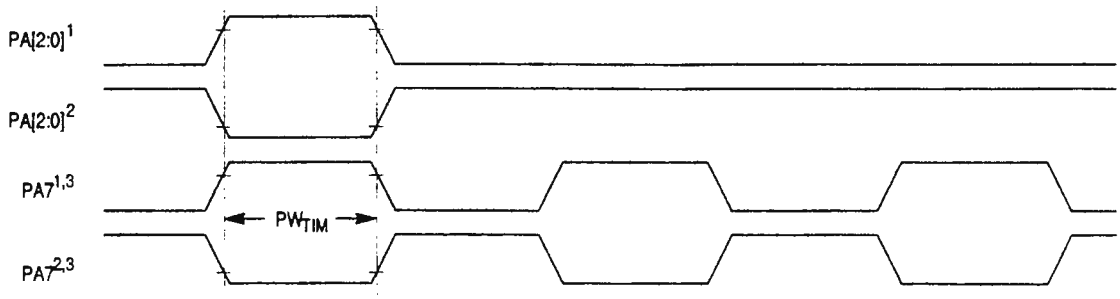
Table A-4a Control Timing (MC68L11A8)

V_{DD} = 3.0 Vdc to 5.5 Vdc, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H

Characteristic		Symbol	1.0 MHz		2.0 MHz		Unit
			Min	Max	Min	Max	
Frequency of Operation		f_o	dc	1.0	dc	2.0	MHz
E-Clock Period		t_{cyc}	1000	—	500	—	ns
Crystal Frequency		f_{XTAL}	—	4.0	—	8.0	MHz
External Oscillator Frequency		4 f_o	dc	4.0	dc	8.0	MHz
Processor Control Setup Time	t_{PCSU} = 1/4 t_{cyc} + 50 ns	t_{PCSU}	325	—	200	—	ns
Reset Input Pulse Width (Note 1)	(To Guarantee External Reset Vector)	PW_{RSTL}	8	—	8	—	t_{cyc}
	(Minimum Input Time; Can Be Preempted by Internal Reset)		1	—	1	—	
Mode Programming Setup Time		t_{MPS}	2	—	2	—	t_{cyc}
Mode Programming Hold Time		t_{MPH}	10	—	10	—	ns
Interrupt Pulse Width, $\overline{IRQ}$ Edge-Sensitive Mode	PW_{IRQ} = t_{cyc} + 20 ns	PW_{IRQ}	1020	—	520	—	ns
Wait Recovery Start-up Time		t_{WRS}	—	4	—	4	t_{cyc}
Timer Pulse Width Input Capture, Pulse Accumulator Input	PW_{TIM} = t_{cyc} + 20 ns	PW_{TIM}	1020	—	520	—	ns

NOTES:

1. $\overline{RESET}$ is recognized during the first clock cycle it is held low. Internal circuitry then drives the pin low for four clock cycles, releases the pin, and samples the pin level two cycles later to determine the source of the interrupt. Refer to **9 RESETS, INTERRUPTS, AND LOW POWER MODES** for further detail.
2. All timing is shown with respect to 20% V_{DD} and 70% V_{DD}, unless otherwise noted.



NOTES:
1. Rising edge sensitive input
2. Falling edge sensitive input
3. Maximum pulse accumulator clocking rate is E-clock frequency divided by 2.

TIMER INPUTS TIM

Figure A-2 Timer Inputs

A

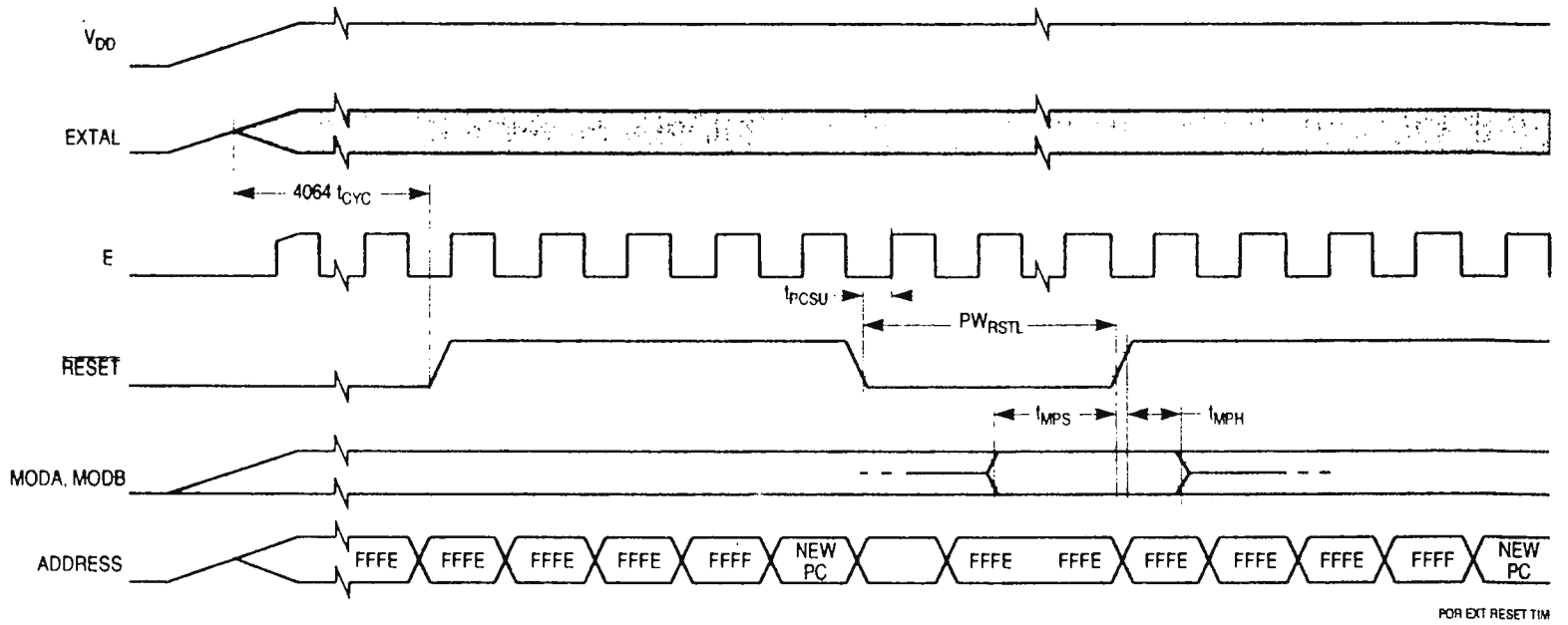
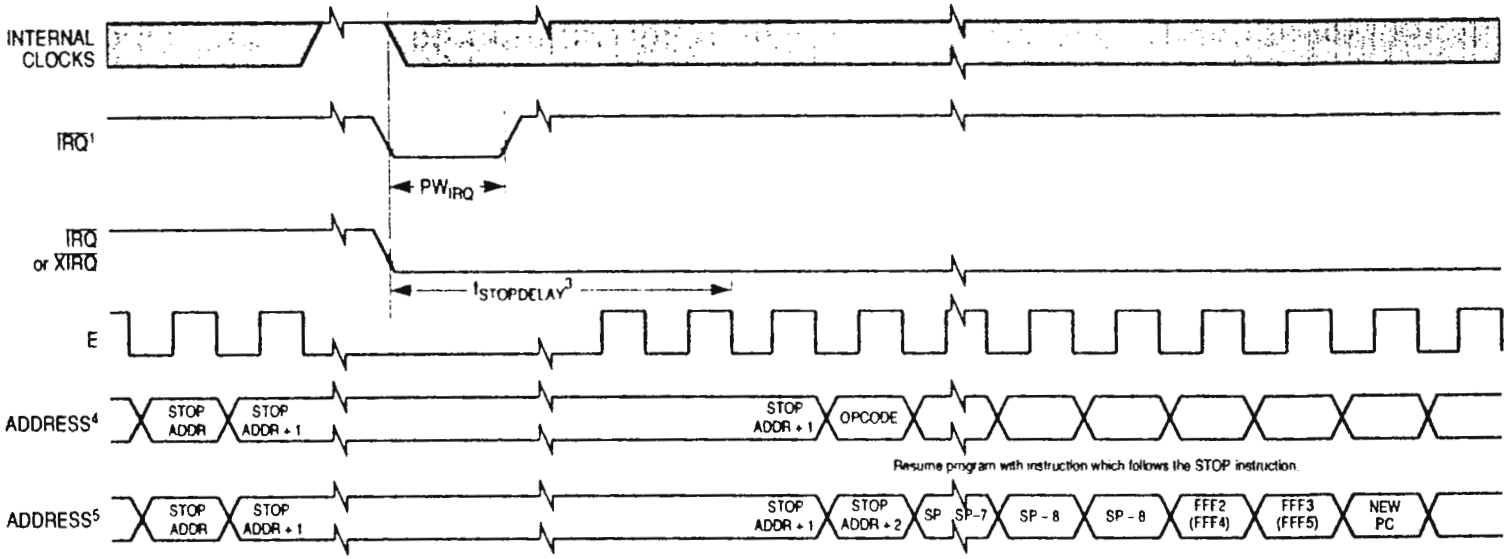


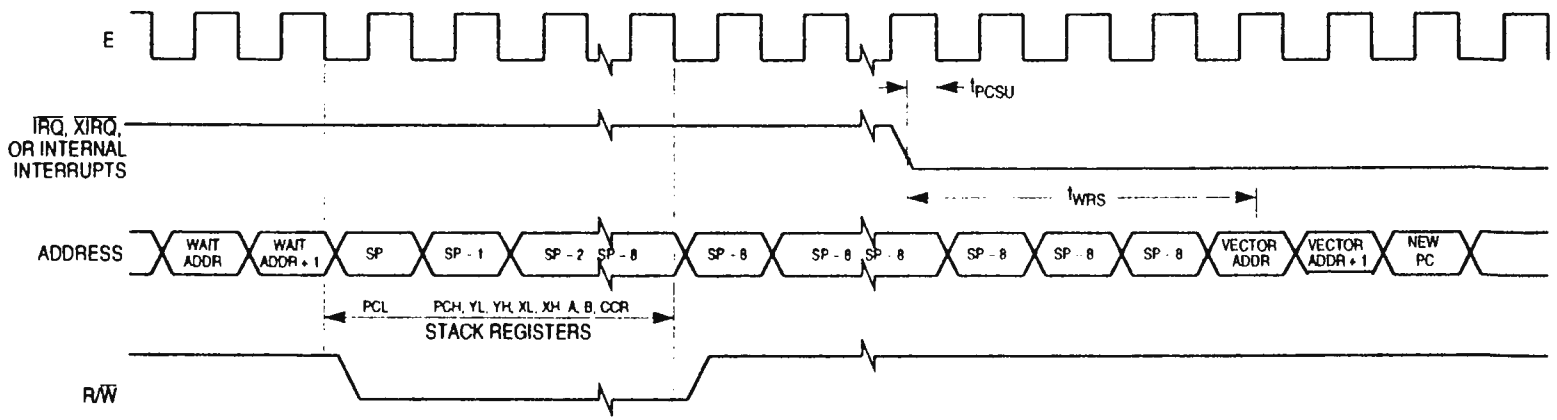
Figure A-3 POR and External Reset Timing Diagram



NOTES:
1. Edge Sensitive $\overline{IRQ}$ pin (IRQE bit = 1)
2. Level sensitive $\overline{IRQ}$ pin (IRQE bit = 0)
3. $t_{STOPDELAY}$ = 4064 t_{CYC} if DLY bit = 1 or 4 t_{CYC} if DLY = 0.
4. $\overline{XIRQ}$ with X bit in CCR = 1.
5. $\overline{IRQ}$ or ($\overline{XIRQ}$ with X bit in CCR = 0)

STOP RECOVERY TIM

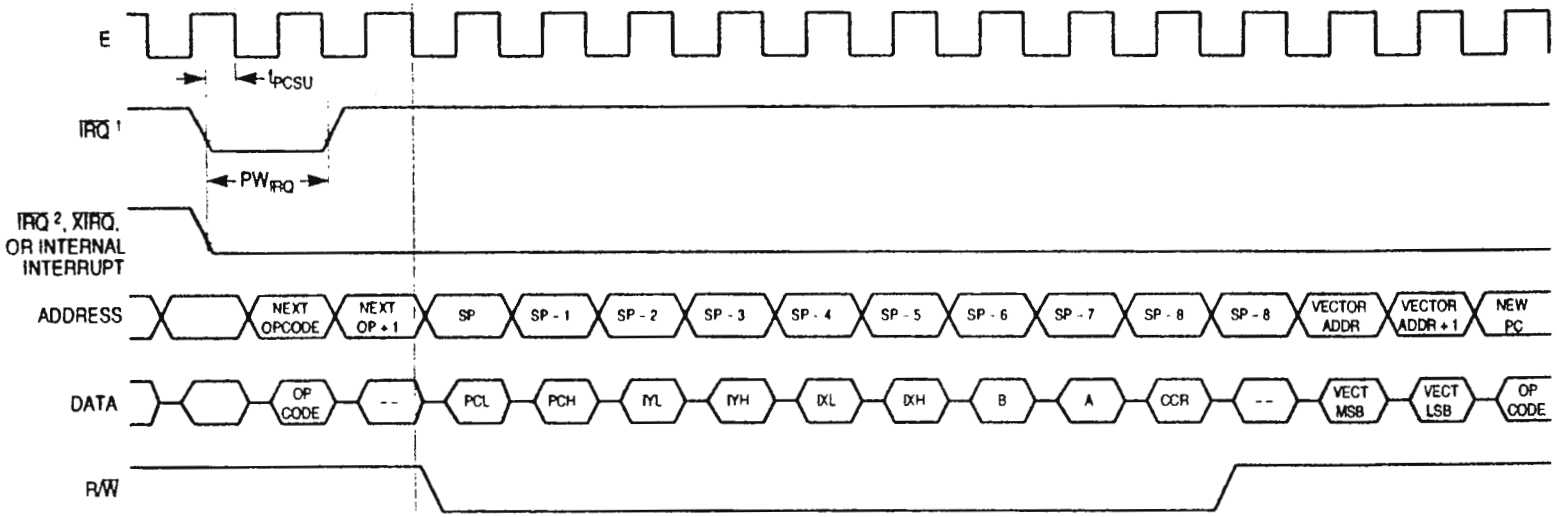
Figure A-4 STOP Recovery Timing Diagram



NOTE: RESET also causes recovery from WAIT.

WAIT RECOVERY TIM

Figure A-5 WAIT Recovery Timing Diagram



NOTES:
1. Edge sensitive $\overline{IRQ}$ pin (IRQE bit = 1)
2. Level sensitive $\overline{IRQ}$ pin (IRQE bit = 0)

INTERRUPT TM

Figure A-6 Interrupt Timing Diagram

Table A-5 Peripheral Port Timing

V_{DD} = 5.0 Vdc ± 10%, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H								
Characteristic	**Symbol**	**1.0 MHz**		**2.0 MHz**		**3.0 MHz**		**Unit**
		Min	**Max**	**Min**	**Max**	**Min**	**Max**	
Frequency of Operation (E-Clock Frequency)	f_o	dc	1.0	dc	2.0	dc	3.0	MHz
E-Clock Period	t_{cyc}	1000	—	500	—	333	—	ns
Peripheral Data Setup Time (MCU Read of Ports A, C, D, and E)	t_{PDSU}	100	—	100	—	100	—	ns
Peripheral Data Hold Time (MCU Read of Ports A, C, D, and E)	t_{PDH}	50	—	50	—	50	—	ns
Delay Time, Peripheral Data Write MCU Write to Port A MCU Writes to Ports B, C, and D t_{PWD} = 1/4 t_{cyc} + 100 ns	t_{PWD}	 — —	 200 350	 — —	 200 225	 — —	 200 183	ns
Input Data Setup Time (Port C)	t_{IS}	60	—	60	—	60	—	ns
Input Data Hold Time (Port C)	t_{IH}	100	—	100	—	100	—	ns
Delay Time, E Fall to STRB t_{DEB} = 1/4 t_{cyc} + 100 ns	t_{DEB}	—	350	—	225	—	183	ns
Setup Time, STRA Asserted to E Fall (Note 1)	t_{AES}	0	—	0	—	0	—	ns
Delay Time, STRA Asserted to Port C Data Output Valid	t_{PCD}	—	100	—	100	—	100	ns
Hold Time, STRA Negated to Port C Data	t_{PCH}	10	—	10	—	10	—	ns
Three-State Hold Time	t_{PCZ}	—	150	—	150	—	150	ns

NOTES:

1. If this setup time is met, STRB acknowledges in the next cycle. If it is not met, the response may be delayed one more cycle.
2. Port C and D timing is valid for active drive (CWOM and DWOM bits not set in PIOC and SPCR registers respectively).
3. All timing is shown with respect to 20% V_{DD} and 70% V_{DD}, unless otherwise noted.

Table A-5a Peripheral Port Timing (MC68L11A8)

V_{DD} = 3.0 Vdc to 5.5 Vdc, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H						
Characteristic	**Symbol**	**1.0 MHz**		**2.0 MHz**		**Unit**
		Min	**Max**	**Min**	**Max**	
Frequency of Operation (E-Clock Frequency)	f_o	dc	1.0	dc	2.0	MHz
E-Clock Period	t_{cyc}	1000	—	500	—	ns
Peripheral Data Setup Time (MCU Read of Ports A, C, D, and E)	t_{PDSU}	100	—	100	—	ns
Peripheral Data Hold Time (MCU Read of Ports A, C, D, and E)	t_{PDH}	50	—	50	—	ns
Delay Time, Peripheral Data Write MCU Write to Port A MCU Writes to Ports B, C, and D t_{PWD} = 1/4 t_{cyc} + 150 ns	t_{PWD}	 — —	 250 400	 — —	 250 275	ns
Input Data Setup Time (Port C)	t_{IS}	60	—	60	—	ns
Input Data Hold Time (Port C)	t_{IH}	100	—	100	—	ns
Delay Time, E Fall to STRB t_{DEB} = 1/4 t_{cyc} + 150 ns	t_{DEB}	—	400	—	275	ns
Setup Time, STRA Asserted to E Fall (Note 1)	t_{AES}	0	—	0	—	ns
Delay Time, STRA Asserted to Port C Data Output Valid	t_{PCD}	—	100	—	100	ns
Hold Time, STRA Negated to Port C Data	t_{PCH}	10	—	10	—	ns
Three-State Hold Time	t_{PCZ}	—	150	—	150	ns

NOTES:

1. If this setup time is met, STRB acknowledges in the next cycle. If it is not met, the response may be delayed one more cycle.
2. Port C and D timing is valid for active drive (CWOM and DWOM bits not set in PIOC and SPCR registers respectively).
3. All timing is shown with respect to 20% V_{DD} and 70% V_{DD}, unless otherwise noted.

A

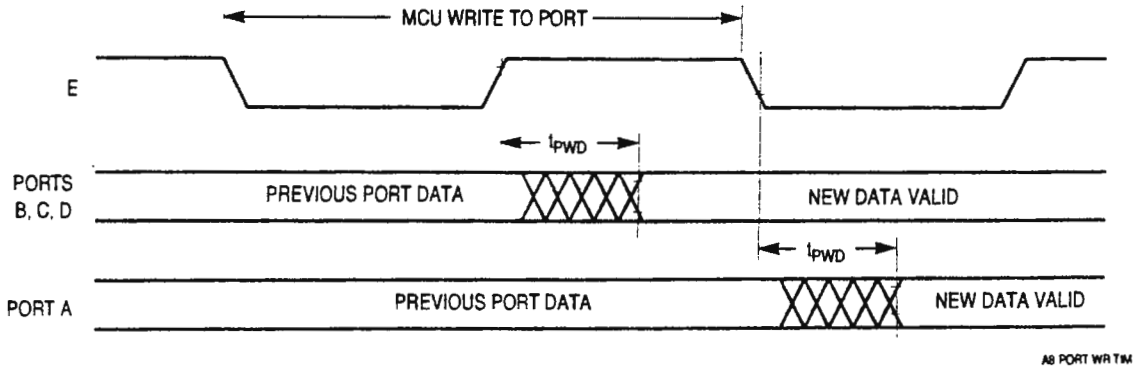


Figure A-7 Port Write Timing Diagram

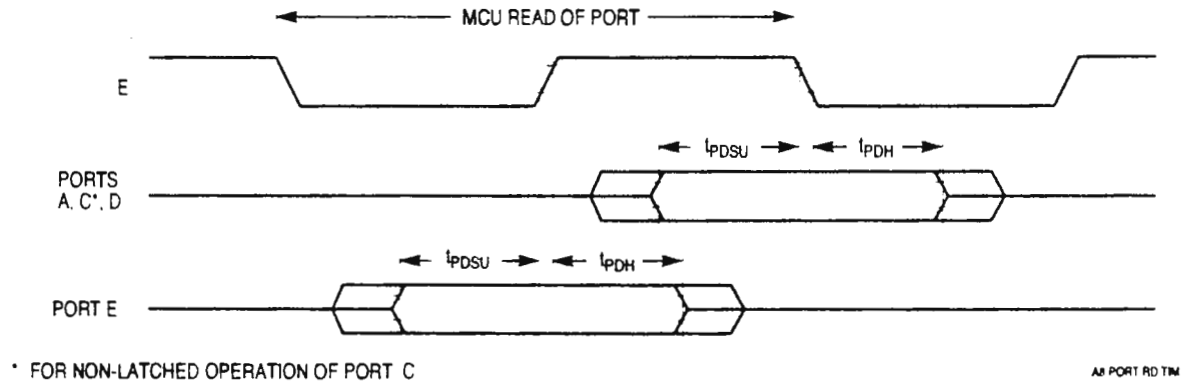


Figure A-8 Port Read Timing Diagram

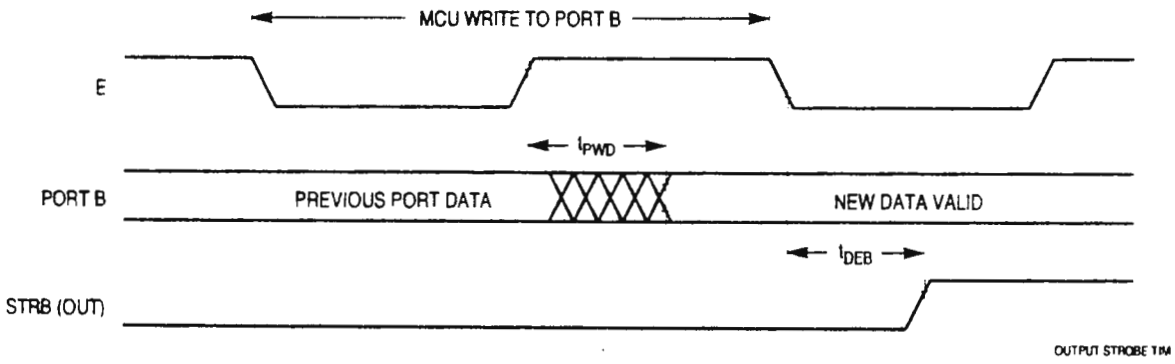


Figure A-9 Simple Output Strobe Timing Diagram

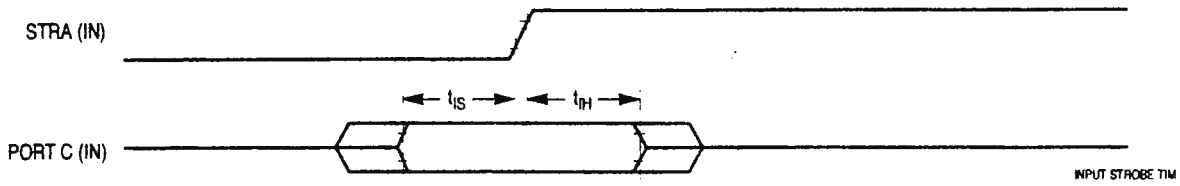
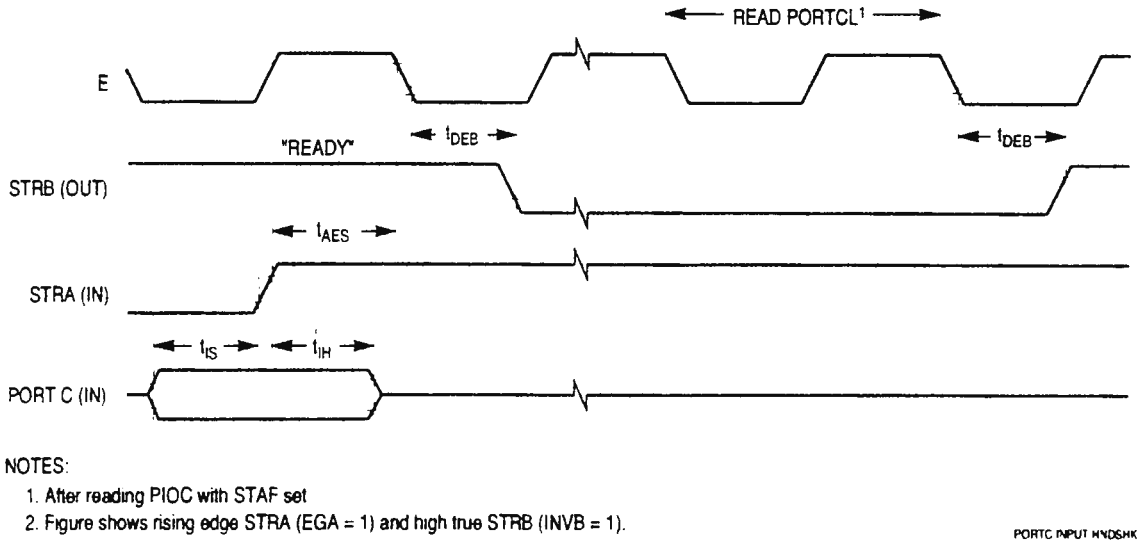
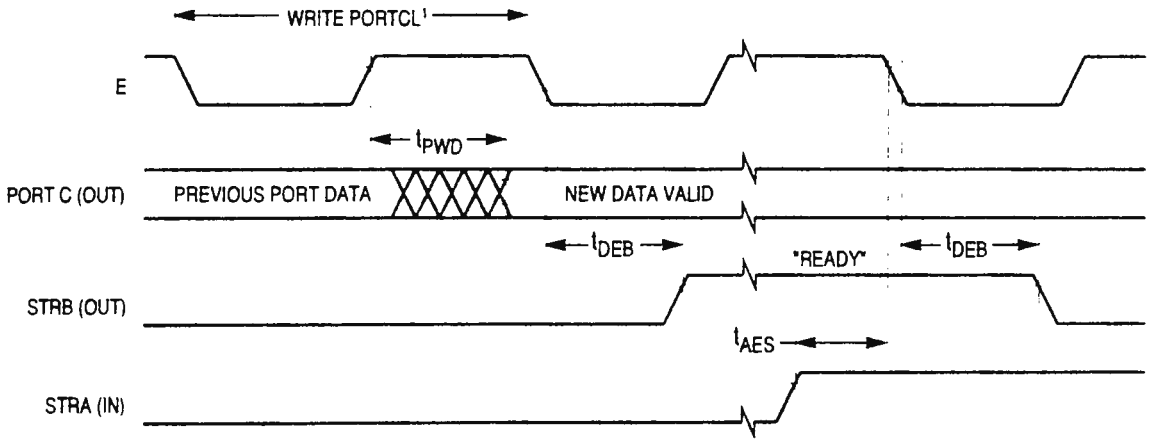


Figure A-10 Simple Input Strobe Timing Diagram



NOTES:
1. After reading PIOC with STAF set
2. Figure shows rising edge STRA (EGA = 1) and high true STRB (INVB = 1).

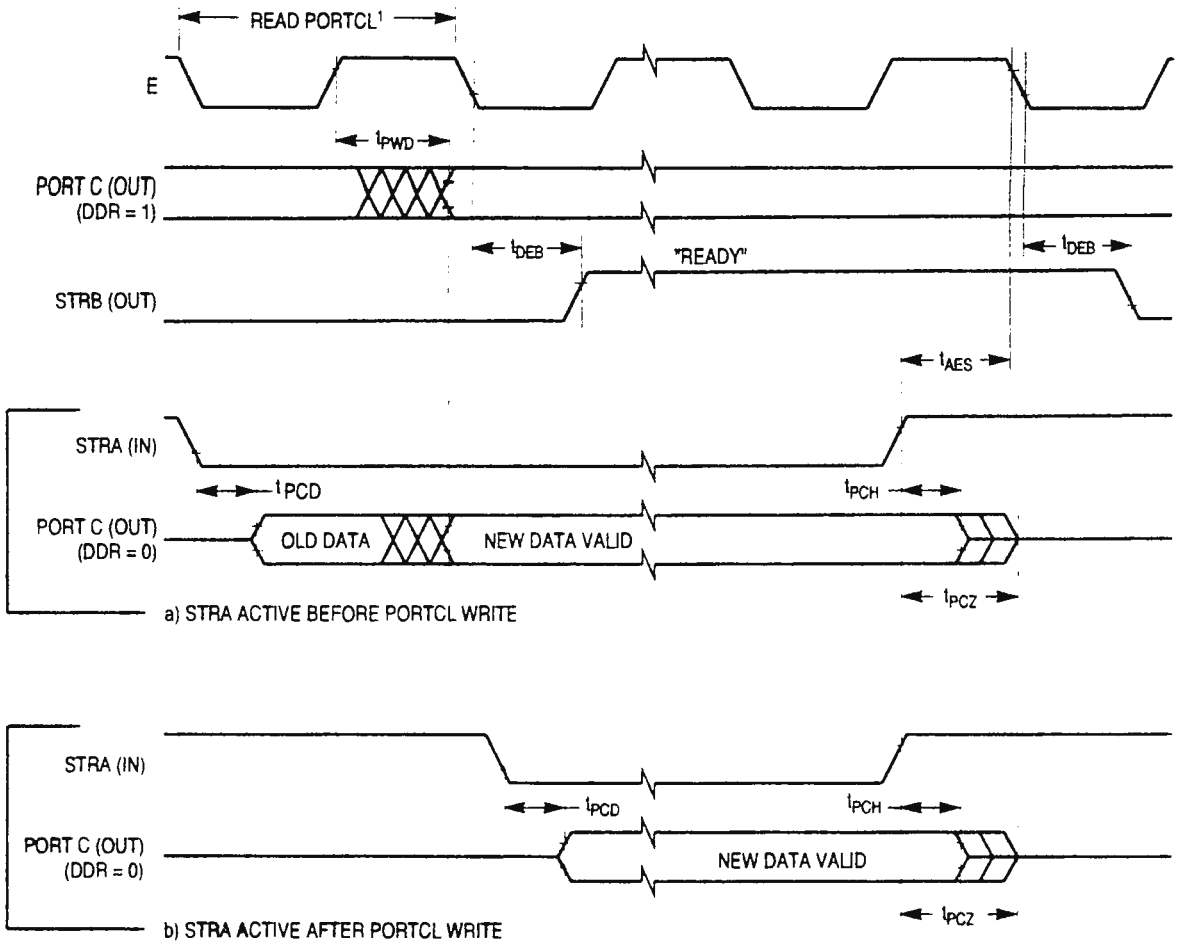
Figure A-11 Port C Input Handshake Timing Diagram



NOTES:
1. After reading PIOC with STAF set
2. Figure shows rising edge STRA (EGA = 1) and high true STRB (INVB = 1).

Figure A-12 Port C Output Handshake Timing Diagram

A



NOTES:

1. After reading PIOC with STAF set
2. Figure shows rising edge STRA (EGA = 1) and high true STRB (INVB = 1).

3STATE VAR HNDSHK

Figure A-13 Three-State Variation of Output Handshake Timing Diagram (STRA Enables Output Buffer)

Table A-6 Analog-To-Digital Converter Characteristics

V_{DD} = 5.0 Vdc ± 10%, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H, 750 kHz ≤ E ≤ 3.0 MHz, unless otherwise noted						
Characteristic	**Parameter**	**Min**	**Absolute**	**2.0 MHz**	**3.0 MHz**	**Unit**
				Max	**Max**	
Resolution	Number of Bits Resolved by A/D Converter	—	8	—	—	Bits
Non-Linearity	Maximum Deviation from the Ideal A/D Transfer Characteristics	—	—	± 1/2	± 1	LSB
Zero Error	Difference Between the Output of an Ideal and an Actual for Zero Input Voltage	—	—	± 1/2	± 1	LSB
Full Scale Error	Difference Between the Output of an Ideal and an Actual A/D for Full-Scale Input Voltage	—	—	± 1/2	± 1	LSB
Total Unadjusted Error	Maximum Sum of Non-Linearity, Zero Error, and Full-Scale Error	—	—	± 1/2	± 1 1/2	LSB
Quantization Error	Uncertainty Because of Converter Resolution	—	—	± 1/2	± 1/2	LSB
Absolute Accuracy	Difference Between the Actual Input Voltage and the Full-Scale Weighted Equivalent of the Binary Output Code, All Error Sources Included	—	—	± 1	± 2	LSB
Conversion Range	Analog Input Voltage Range	V_{RL}	—	V_{RH}	V_{RH}	V
V_{RH}	Maximum Analog Reference Voltage (Note 2)	V_{RL}	—	V_{DD} + 0.1	V_{DD} + 0.1	V
V_{RL}	Minimum Analog Reference Voltage (Note 2)	V_{SS} − 0.1	—	V_{RH}	V_{RH}	V
ΔV_R	Minimum Difference between V_{RH} and V_{RL} (Note 2)	3	—	—	—	V
Conversion Time	Total Time to Perform a Single Analog-to-Digital Conversion: a. E Clock b. Internal RC Oscillator	 — —	 32 —	 — t_{cyc} + 32	 — t_{cyc} + 32	 t_{cyc} µs
Monotonicity	Conversion Result Never Decreases with an Increase in Input Voltage and has no Missing Codes	—	Guaranteed	—	—	—
Zero Input Reading	Conversion Result when $V_{in} = V_{RL}$	00	—	—	—	Hex
Full Scale Reading	Conversion Result when $V_{in} = V_{RH}$	—	—	FF	FF	Hex
Sample Acquisition Time	Analog Input Acquisition Sampling Time: a. E Clock b. Internal RC Oscillator	 — —	 12 —	 — 12	 — 12	 t_{cyc} µs
Sample/Hold Capacitance	Input Capacitance during Sample PE0-PE7	—	20 (Typ)	—	—	pF
Input Leakage	Input Leakage on A/D PinsPE0-PE7 V_{RL}, V_{RH}	— —	— —	400 1.0	400 1.0	nA µA

NOTES:

1. Source impedances greater than 10 kΩ affect accuracy adversely because of input leakage.
2. Performance verified down to 2.5 V ΔV_R, but accuracy is tested and guaranteed at ΔV_R = 5 V ± 10%.

A

Table A-6a Analog-To-Digital Converter Characteristics (MC68L11A8)

V_{DD} = 3.0 Vdc to 5.5 Vdc, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H, 750 kHz ≤ E ≤ 2.0 MHz, unless otherwise noted					
Characteristic	**Parameter**	**Min**	**Absolute**	**Max**	**Unit**
Resolution	Number of Bits Resolved by A/D Converter	—	8	—	Bits
Non-Linearity	Maximum Deviation from the Ideal A/D Transfer Characteristics	—	—	± 1	LSB
Zero Error	Difference Between the Output of an Ideal and an Actual for Zero Input Voltage	—	—	± 1	LSB
Full Scale Error	Difference Between the Output of an Ideal and an Actual A/D for Full-Scale Input Voltage	—	—	± 1	LSB
Total Unadjusted Error	Maximum Sum of Non-Linearity, Zero Error, and Full-Scale Error	—	—	±1 1/2	LSB
Quantization Error	Uncertainty Because of Converter Resolution	—	—	± 1/2	LSB
Absolute Accuracy	Difference Between the Actual Input Voltage and the Full-Scale Weighted Equivalent of the Binary Output Code, All Error Sources Included	—	—	± 2	LSB
Conversion Range	Analog Input Voltage Range	V_{RL}	—	V_{RH}	V
V_{RH}	Maximum Analog Reference Voltage(Note 2)	V_{RL}	—	V_{DD} + 0.1	V
V_{RL}	Minimum Analog Reference Voltage (Note 2)	V_{SS} – 0.1	—	V_{RH}	V
ΔV_R	Minimum Difference between V_{RH} and V_{RL} (Note 2)	3	—	—	V
Conversion Time	Total Time to Perform a Single Analog-to-Digital Conversion: a. E Clock b. Internal RC Oscillator	 — —	 32 —	 — t_{cyc} + 32	 t_{cyc} µs
Monotonicity	Conversion Result Never Decreases with an increase in Input Voltage and has no Missing Codes	—	Guaranteed	—	—
Zero Input Reading	Conversion Result when $V_{in} = V_{RL}$	00	—	—	Hex
Full Scale Reading	Conversion Result when $V_{in} = V_{RH}$	—	—	FF	Hex
Sample Acquisition Time	Analog Input Acquisition Sampling Time: a. E Clock b. Internal RC Oscillator	 — —	 12 —	 — 12	 t_{cyc} µs
Sample/Hold Capacitance	Input Capacitance during Sample PE0-PE7	—	20 (Typ)	—	pF
Input Leakage	Input Leakage on A/D PinsPE0-PE7 V_{RL}, V_{RH}	— —	— —	400 1.0	nA µA

NOTES:

1. Source impedances greater than 10 kΩ affect accuracy adversely because of input leakage.

Table A-7 Expansion Bus Timing

V_{DD} = 5.0 Vdc ± 10%, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H									
Num	**Characteristic**	**Symbol**	**1.0 MHz**		**2.0 MHz**		**3.0 MHz**		**Unit**
			Min	**Max**	**Min**	**Max**	**Min**	**Max**	
	Frequency of Operation (E-Clock Frequency)	f_o	dc	1.0	dc	2.0	dc	3.0	MHz
1	Cycle Time	t_{cyc}	1000	—	500	—	333	—	ns
2	Pulse Width, E Low PW_{EL} = 1/2 t_{cyc} – 23 ns(Note 1)	PW_{EL}	477	—	227	—	146	—	ns
3	Pulse Width, E High PW_{EH} = 1/2 t_{cyc} – 28 ns(Note 1)	PW_{EH}	472	—	222	—	141	—	ns
4a, b	E and AS Rise and Fall Time	t_r t_f	—	20 20	—	20 20	—	20 15	ns
9	Address Hold Time t_{AH} = 1/8 t_{cyc} – 29.5 ns (Note 1, 2a)	t_{AH}	95.5	—	33	—	26	—	ns
12	Non-Muxed Address Valid Time to E Rise $t_{AV} = PW_{EL} - (t_{ASD} + 80$ ns)(Note 1, 2a)	t_{AV}	281.5	—	94	—	54	—	ns
17	Read Data Setup Time	t_{DSR}	30	—	30	—	30	—	ns
18	Read Data Hold Time (Max = t_{MAD})	t_{DHR}	0	145.5	0	83	0	51	ns
19	Write Data Delay Time t_{DDW} = 1/8 t_{cyc} + 65.5 ns (Note 1, 2a)	t_{DDW}	—	190.5	—	128	—	71	ns
21	Write Data Hold Time t_{DHW} = 1/8 t_{cyc} – 29.5 ns(Note 1, 2a)	t_{DHW}	95.5	—	33	—	26	—	ns
22	Muxed Address Valid Time to E Rise $t_{AVM} = PW_{EL} - (t_{ASD} + 90$ ns)(Note 1, 2a)	t_{AVM}	271.5	—	84	—	54	—	ns
24	Muxed Address Valid Time to AS Fall $t_{ASL} = PW_{ASH}$ – 70 ns(Note 1)	t_{ASL}	151	—	26	—	13	—	ns
25	Muxed Address Hold Time t_{AHL} = 1/8 t_{cyc} – 29.5 ns(Note 1, 2b)	t_{AHL}	95.5	—	33	—	31	—	ns
26	Delay Time, E to AS Rise t_{ASD} = 1/8 t_{cyc} – 9.5 ns (Note 1, 2a)	t_{ASD}	115.5	—	53	—	31	—	ns
27	Pulse Width, AS High PW_{ASH} = 1/4 t_{cyc} – 29 ns(Note 1)	PW_{ASH}	221	—	96	—	63	—	ns
28	Delay Time, AS to E Rise t_{ASED} = 1/8 t_{cyc} – 9.5 ns(Note 1, 2b)	t_{ASED}	115.5	—	53	—	31	—	ns
29	MPU Address Access Time(Note 2a) $t_{ACCA} = t_{cyc} - (PW_{EL} - t_{AVM}) - t_{DSR} - t_f$	t_{ACCA}	744.5	—	307	—	196	—	ns
35	MPU Access Time $t_{ACCE} = PW_{EH} - t_{DSR}$	t_{ACCE}	—	442	—	192	—	111	ns
36	Muxed Address Delay (Previous Cycle MPU Read) $t_{MAD} = t_{ASD}$ + 30 ns(Note 1, 2a)	t_{MAD}	145.5	—	83	—	51	—	ns

NOTES:

1. Formula only for dc to 2 MHz.
2. Input clocks with duty cycles other than 50% affect bus performance. Timing parameters affected by input clock duty cycle are identified by (a) and (b). To recalculate the approximate bus timing values, substitute the following expressions in place of 1/8 t_{cyc} in the above formulas, where applicable:
   (a) (1-DC) x 1/4 t_{cyc}
   (b) DC x 1/4 t_{cyc}

   Where:
   DC is the decimal value of duty cycle percentage (high time).
3. All timing is shown with respect to 20% V_{DD} and 70% V_{DD}, unless otherwise noted.

A

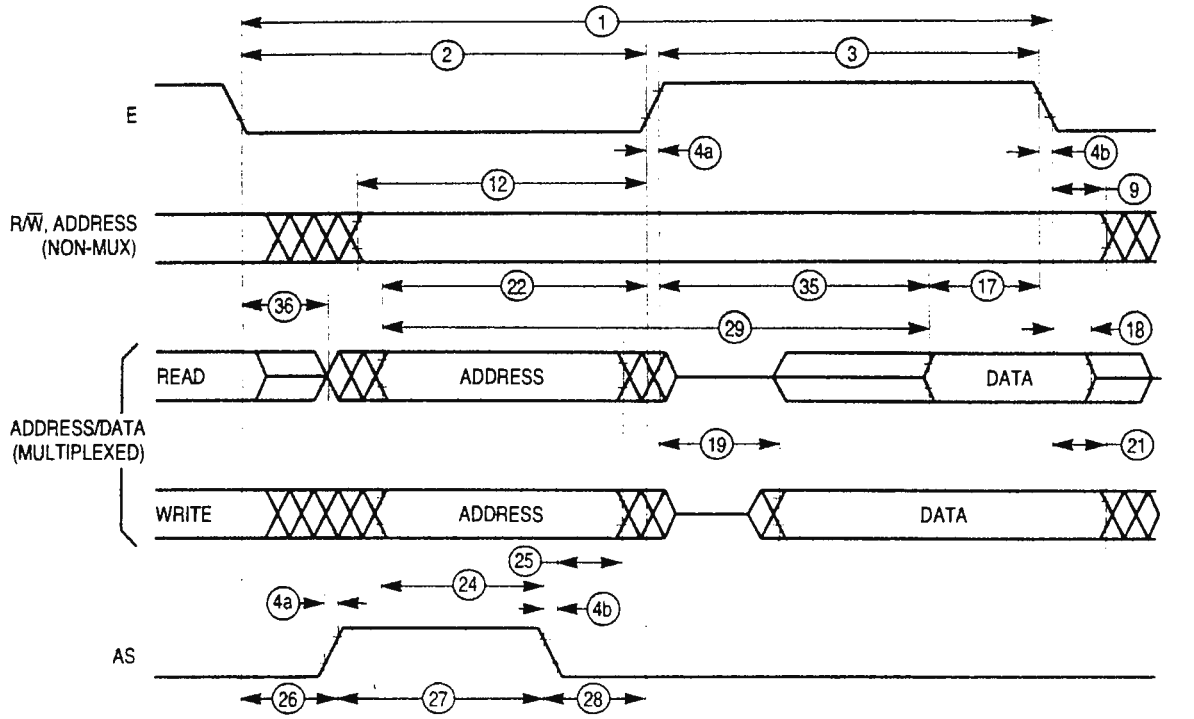
Table A-7a Expansion Bus Timing (MC68L11A8)

V_{DD} = 3.0 Vdc to 5.5 Vdc, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H							
Num	**Characteristic**	**Symbol**	**1.0 MHz**		**2.0 MHz**		**Unit**
			Min	**Max**	**Min**	**Max**	
	Frequency of Operation (E-Clock Frequency)	f_o	dc	1.0	dc	2.0	MHz
1	Cycle Time	t_{cyc}	1000	—	500	—	ns
2	Pulse Width, E Low PW_{EL} = 1/2 t_{cyc} – 23 ns (Note 1)	PW_{EL}	475	—	225	—	ns
3	Pulse Width, E High PW_{EH} = 1/2 t_{cyc} – 28 ns (Note 1)	PW_{EH}	470	—	220	—	ns
4a, b	E and AS Rise and Fall Time	t_r t_f	—	25 25	—	25 25	ns
9	Address Hold Time t_{AH} = 1/8 t_{cyc} – 29.5 ns (Note 1, 2a)	t_{AH}	95	—	33	—	ns
12	Non-Muxed Address Valid Time to E Rise $t_{AV} = PW_{EL} - (t_{ASD} + 80$ ns) (Note 1, 2a)	t_{AV}	275	—	88	—	ns
17	Read Data Setup Time	t_{DSR}	30	—	30	—	ns
18	Read Data Hold Time (Max = t_{MAD})	t_{DHR}	0	150	0	88	ns
19	Write Data Delay Time t_{DDW} = 1/8 t_{cyc} + 65.5 ns (Note 1, 2a)	t_{DDW}	—	195	—	133	ns
21	Write Data Hold Time t_{DHW} = 1/8 t_{cyc} – 29.5 ns (Note 1, 2a)	t_{DHW}	95	—	33	—	ns
22	Muxed Address Valid Time to E Rise $t_{AVM} = PW_{EL} - (t_{ASD} + 90$ ns) (Note 1, 2a)	t_{AVM}	265	—	78	—	ns
24	Muxed Address Valid Time to AS Fall $t_{ASL} = PW_{ASH} - 70$ ns (Note 1)	t_{ASL}	150	—	25	—	ns
25	Muxed Address Hold Time t_{AHL} = 1/8 t_{cyc} – 29.5 ns (Note 1, 2b)	t_{AHL}	95	—	33	—	ns
26	Delay Time, E to AS Rise t_{ASD} = 1/8 t_{cyc} – 9.5 ns (Note 1, 2a)	t_{ASD}	120	—	58	—	ns
27	Pulse Width, AS High PW_{ASH} = 1/4 t_{cyc} – 29 ns (Note 1)	PW_{ASH}	220	—	95	—	ns
28	Delay Time, AS to E Rise t_{ASED} = 1/8 t_{cyc} – 9.5 ns (Note 1, 2b)	t_{ASED}	120	—	58	—	ns
29	MPU Address Access Time (Note 2a) $t_{ACCA} = t_{cyc} - (PW_{EL} - t_{AVM}) - t_{DSR} - t_f$	t_{ACCA}	735	—	298	—	ns
35	MPU Access Time $t_{ACCE} = PW_{EH} - t_{DSR}$	t_{ACCE}	—	440	—	190	ns
36	Muxed Address Delay (Previous Cycle MPU Read) $t_{MAD} = t_{ASD} + 30$ ns (Note 1, 2a)	t_{MAD}	150	—	88	—	ns

NOTES:

1. Formula only for dc to 2 MHz.
2. Input clocks with duty cycles other than 50% affect bus performance. Timing parameters affected by input clock duty cycle are identified by (a) and (b). To recalculate the approximate bus timing values, substitute the following expressions in place of 1/8 t_{cyc} in the above formulas, where applicable:
   (a) (1-DC) x 1/4 t_{cyc}
   (b) DC x 1/4 t_{cyc}

   Where:
   DC is the decimal value of duty cycle percentage (high time).
3. All timing is shown with respect to 20% V_{DD} and 70% V_{DD}, unless otherwise noted.



NOTE: Measurement points shown are 20% and 70% of V_{DD}.

MUX BUS TIM

Figure A-14 Multiplexed Expansion Bus Timing Diagram

Table A-8 Serial Peripheral Interface (SPI) Timing

V_{DD} = 5.0 Vdc ± 10%, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H

Num	Characteristic	Symbol	2.0 MHz		3.0 MHz		Unit
			Min	Max	Min	Max	
	Operating Frequency Master Slave	 $f_{op(m)}$ $f_{op(s)}$	 dc dc	 0.5 2.0	 dc dc	 0.5 3.0	 f_{op} MHz
1	Cycle Time Master Slave	 $t_{cyc(m)}$ $t_{cyc(s)}$	 2.0 500	 — —	 2.0 333	 — —	 t_{cyc} ns
2	Enable Lead Time Master (Note 2) Slave	 $t_{lead(m)}$ $t_{lead(s)}$	 — 250	 — —	 — 240	 — —	 ns ns
3	Enable Lag Time Master (Note 2) Slave	 $t_{lag(m)}$ $t_{lag(s)}$	 — 250	 — —	 — 240	 — —	 ns ns
4	Clock (SCK) High Time Master Slave	 $t_{w(SCKH)m}$ $t_{w(SCKH)s}$	 340 190	 — —	 227 127	 — —	 ns ns
5	Clock (SCK) Low Time Master Slave	 $t_{w(SCKL)m}$ $t_{w(SCKL)s}$	 340 190	 — —	 227 127	 — —	 ns ns
6	Data Setup Time (Inputs) Master Slave	 $t_{su(m)}$ $t_{su(s)}$	 100 100	 — —	 100 100	 — —	 ns ns
7	Data Hold Time (Inputs) Master Slave	 $t_{h(m)}$ $t_{h(s)}$	 100 100	 — —	 100 100	 — —	 ns ns
8	Access Time (Time to Data Active from High-Impedance State) Slave	 t_a	 0	 120	 0	 120	 ns
9	Disable Time (Hold Time to High-Impedance State) Slave	 t_{dis}	 —	 240	 —	 167	 ns
10	Data Valid (After Enable Edge)(Note 3)	$t_{v(s)}$	—	240	—	167	ns
11	Data Hold Time (Outputs) (After Enable Edge)	t_{ho}	0	—	0	—	ns
12	Rise Time (20% V_{DD} to 70% V_{DD}, CL = 200 pF) SPI Outputs (SCK, MOSI, and MISO) SPI Inputs (SCK, MOSI, MISO, and $\overline{SS}$)	 t_{rm} t_{rs}	 — —	 100 2.0	 — —	 100 2.0	 ns µs
13	Fall Time (70% V_{DD} to 20% V_{DD}, C_L = 200 pF) SPI Outputs (SCK, MOSI, and MISO) SPI Inputs (SCK, MOSI, MISO, and $\overline{SS}$)	 t_{fm} t_{fs}	 — —	 100 2.0	 — —	 100 2.0	 ns µs

NOTES:

1. All timing is shown with respect to 20% V_{DD} and 70% V_{DD}, unless otherwise noted.
2. Signal production depends on software.
3. Assumes 200 pF load on all SPI pins.

Table A-8a Serial Peripheral Interface (SPI) Timing (MC68L11A8)

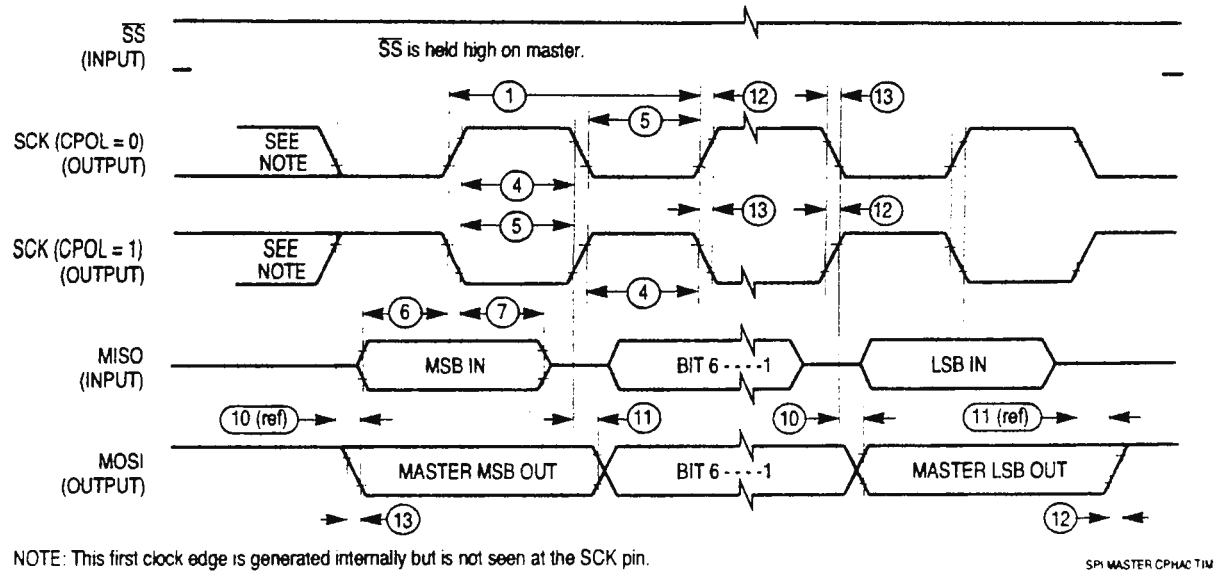
V_{DD} = 3.0 Vdc to 5.5 Vdc, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H

Num	Characteristic	Symbol	1.0 MHz		2.0 MHz		Unit
			Min	Max	Min	Max	
	Operating Frequency Master Slave	 $f_{op(m)}$ $f_{op(s)}$	 dc dc	 0.5 1.0	 dc dc	 0.5 2.0	 f_{op} MHz
1	Cycle Time Master Slave	 $t_{cyc(m)}$ $t_{cyc(s)}$	 2.0 1000	 — —	 2.0 500	 — —	 t_{cyc} ns
2	Enable Lead Time Master (Note 2) Slave	 $t_{lead(m)}$ $t_{lead(s)}$	 — 500	 — —	 — 250	 — —	 ns ns
3	Enable Lag Time Master (Note 2) Slave	 $t_{lag(m)}$ $t_{lag(s)}$	 — 500	 — —	 — 250	 — —	 ns ns
4	Clock (SCK) High Time Master Slave	 $t_{w(SCKH)m}$ $t_{w(SCKH)s}$	 680 380	 — —	 340 190	 — —	 ns ns
5	Clock (SCK) Low Time Master Slave	 $t_{w(SCKL)m}$ $t_{w(SCKL)s}$	 680 380	 — —	 340 190	 — —	 ns ns
6	Data Setup Time (Inputs) Master Slave	 $t_{su(m)}$ $t_{su(s)}$	 100 100	 — —	 100 100	 — —	 ns ns
7	Data Hold Time (Inputs) Master Slave	 $t_{h(m)}$ $t_{h(s)}$	 100 100	 — —	 100 100	 — —	 ns ns
8	Access Time (Time to Data Active from High-Impedance State) Slave	 t_a	 0	 120	 0	 120	 ns
9	Disable Time (Hold Time to High-Impedance State) Slave	 t_{dis}	 —	 240	 —	 240	 ns
10	Data Valid (After Enable Edge)(Note 3)	$t_{v(s)}$	—	240	—	240	ns
11	Data Hold Time (Outputs) (After Enable Edge)	t_{ho}	0	—	0	—	ns
12	Rise Time (20% V_{DD} to 70% V_{DD}, CL = 200 pF) SPI Outputs (SCK, MOSI, and MISO) SPI Inputs (SCK, MOSI, MISO, and $\overline{SS}$)	 t_{rm} t_{rs}	 — —	 100 2.0	 — —	 100 2.0	 ns µs
13	Fall Time (70% V_{DD} to 20% V_{DD}, C_L = 200 pF) SPI Outputs (SCK, MOSI, and MISO) SPI Inputs (SCK, MOSI, MISO, and $\overline{SS}$)	 t_{fm} t_{fs}	 — —	 100 2.0	 — —	 100 2.0	 ns µs

NOTES:

1. All timing is shown with respect to 20% V_{DD} and 70% V_{DD}, unless otherwise noted.
2. Signal production depends on software.
3. Assumes 200 pF load on all SPI pins.

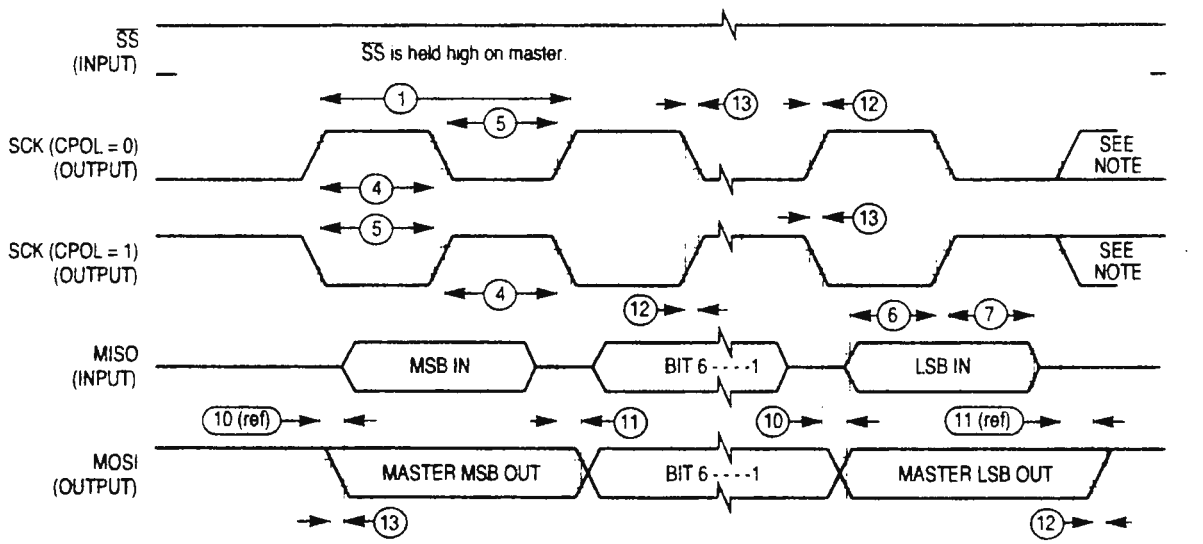
A



NOTE: This first clock edge is generated internally but is not seen at the SCK pin.

SPI MASTER CPHA0 TIM

a) SPI Master Timing (CPHA = 0)

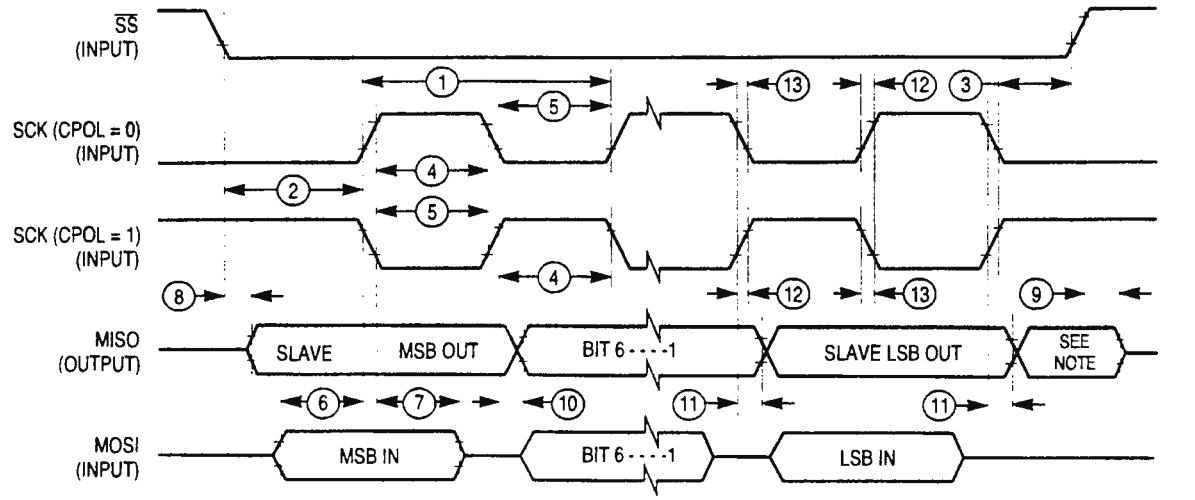


NOTE: This last clock edge is generated internally but is not seen at the SCK pin.

SPI MASTER CPHA1 TIM

b) SPI Master Timing (CPHA = 1)

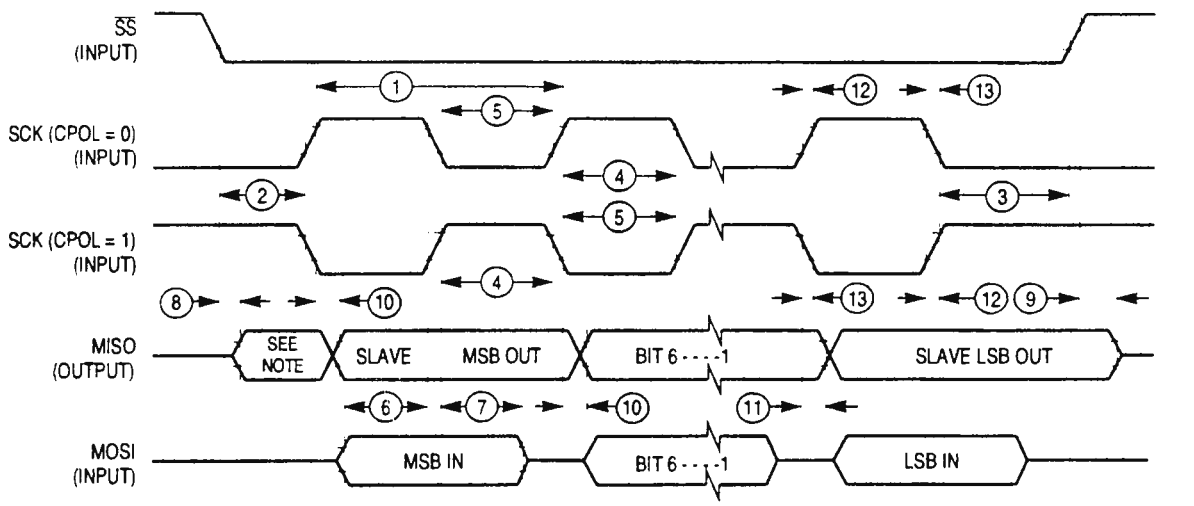
Figure A-15 SPI Timing Diagram (1 of 2)



NOTE: Not defined but normally MSB of character just received.

SPI SLAVE CPHA0 TIM

c) SPI Slave Timing (CPHA = 0)



NOTE: Not defined but normally LSB of character previously transmitted.

SPI SLAVE CPHA1 TIM

d) SPI Slave Timing (CPHA = 1)

Figure A-15 SPI Timing Diagrams (2 of 2)

Table A-9 EEPROM Characteristics

V_{DD} = 5.0 Vdc ± 10%, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H

Characteristic		Temperature Range			Unit
		– 40 to 85° C	– 40 to 105° C	– 40 to 125° C	
Programming Time (Note 1)	<1.0 MHz, RCO Enabled	10	15	20	ms
	1.0 to 2.0 MHz, RCO Enabled	20	Must use RCO	Must use RCO	
	≥ 2.0 MHz (or Anytime RCO Enabled)	10	15	20	
Erase Time (Note 1)	Byte, Row and Bulk	10	10	10	ms
Write/Erase Endurance (Note 2)		10,000	10,000	10,000	Cycles
Data Retention (Note 2)		10	10	10	Years

NOTES:

1. The RC oscillator (RCO) must be enabled (by setting the CSEL bit in the OPTION register) for EEPROM programming and erasure when the E-clock frequency is below 1.0 MHz.
2. Refer to Reliability Monitor Report (current quarterly issue) for current failure rate information.

Table A-9a EEPROM Characteristics (MC68L11A8)

V_{DD} = 3.0 Vdc to 5.5 Vdc, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H

Characteristic		Temperature Range – 20 to 70° C	Unit
Programming Time (Note 1)	3 V, E ≤ 2.0 MHz, RCO Enabled	25	ms
	5 V, E ≤ 2.0 MHz, RCO Enabled	10	
Erase Time (Byte, Row and Bulk) (Note 1)	3 V, E ≤ 2.0 MHz, RCO Enabled	25	ms
	5 V, E ≤ 2.0 MHz, RCO Enabled	10	
Write/Erase Endurance (Note 2)		10,000	Cycles
Data Retention (Note 2)		10	Years

NOTES:

1. The RC oscillator (RCO) must be enabled (by setting the CSEL bit in the OPTION register) for EEPROM programming and erasure.
2. Refer to Reliability Monitor Report (current quarterly issue) for current failure rate information.

CHAPTER 1
GENERAL INFORMATION

1.1 INTRODUCTION

This manual provides general information, hardware preparation, installation instructions, monitor program description, operating instructions. hardware description, and support information for the M68HC11 Evaluation Board (EVB).

Downloading S-record information is contained in Appendix A. While a listing of the EVB monitor program is stored on the diskettes supplied with the EVB (see file buf25.asm). (This file may be viewed using any text reader capable of handling a 123K file.)

NOTE

Unless otherwise specified. all address references are in hexadecimal throughout this manual.

An asterisk (*) following the signal name denotes that the signal is true or valid when the signal is low.

1.2 FEATURES

EVB features include:

- An economical means of debugging user assembled code and evaluating target systems incorporating MC68HC11 microcomputer unit (MCU) device
- One-line assembler/disassembler
- Host computer downloading capability
- MC68HC11 MCU based debugging/evaluating circuitry
- MC68HC24 Port Replacement Unit (PRU) based MCU I/O expansion circuitry
- MC6850 Asynchronous Communications Interface Adapter (ACIA) based terminal I/O port circuitry
- RS-232C compatible terminal/host computer I/O ports

1.3 SPECIFICATIONS

Table 1-1 lists the EVB specifications.

Table 1-1. EVB Specifications

Characteristics	Specifications
MCU	MC68HC11A1FN
PRU	MC68HC24FN
ACIA	MC68B50
I/O ports: Terminal Host computer MCU extension	 RS-232C compatible RS-232C compatible HCMOS-TTL compatible
Temperature: Operating Storage	 +25 degrees C -40 to +85 degrees C
Relative humidity	0 to 90% (non-condensing)
Power requirements	+5 Vdc @ 0.5 A (max) +12 Vdc @ 0.1 A (max) -12 Vdc @ 0.1 A (max)
Dimensions: Width Length	 7.062 in. (17.8 cm) 4.625 in. (11.75 cm)

1.4 GENERAL DESCRIPTION

The MC68HC11 MCU device is an advanced single-chip MCU with on-chip memory and peripheral functions. Refer to the MC68HC11 MCU data sheet for additional device information. To demonstrate the capabilities of this MCU, the EVB functions with a debug monitor program called BUFFALO (Bit User Fast Friendly Aid to Logical Operations). This monitor program is contained within an on-board EPROM (external to the MCU).

The EVB provides a low cost tool for debugging and evaluation of MC68HC11 MCU-based target system equipment (Figure 1-1 is a block diagram of the EVB). The EVB is not intended to be a replacement for a much more powerful and flexible tool, such as the Motorola M68HC11EVM Evaluation Module. The EVB operates in either the debugging or evaluation (emulation) mode of operation.

The first mode of operation lets you debug your code using the BUFFALO monitor program. User code is assembled on the EVB or on a host computer and then downloaded to the EVB RAM via Motorola S-records. The second mode of operation lets you evaluate (emulate) user code in a target system environment utilizing the memory of the MC68HC11 MCU. The EVB emulates the single-chip mode of operation, even though the EVB operates in the expanded multiplexed mode of operation at all times.

Overall evaluation/debugging control of the EVB is provided by the BUFFALO monitor program via terminal interaction. The target system interface is provided by the MCU and PRU devices. RS-232C terminal/host I/O port interface circuitry provides communication and data transfer operations between the EVB and external terminal/host computer devices.

Independent baud rate selection capabilities are provided for the terminal and host I/O ports. Hardware selectable (300-9600) baud rates are provided for the ACIA-based terminal port. A non-selectable (fixed) 9600 baud rate is provided for the host port via the MCU Serial Communications Interface (SCI).

The EVB requires a user-supplied +5, +12, and -12 Vdc power supply and an RS-232C compatible terminal for operation. An RS-232C compatible host computer is used with the EVB to download Motorola S-records via the BUFFALO monitor commands.

The Motorola S-record format was devised for the purpose of encoding programs or data files in a printable format for transportation between computer systems. The transportation process can therefore be monitored and the S-records can be easily edited. Refer to Appendix A for additional S-record information.

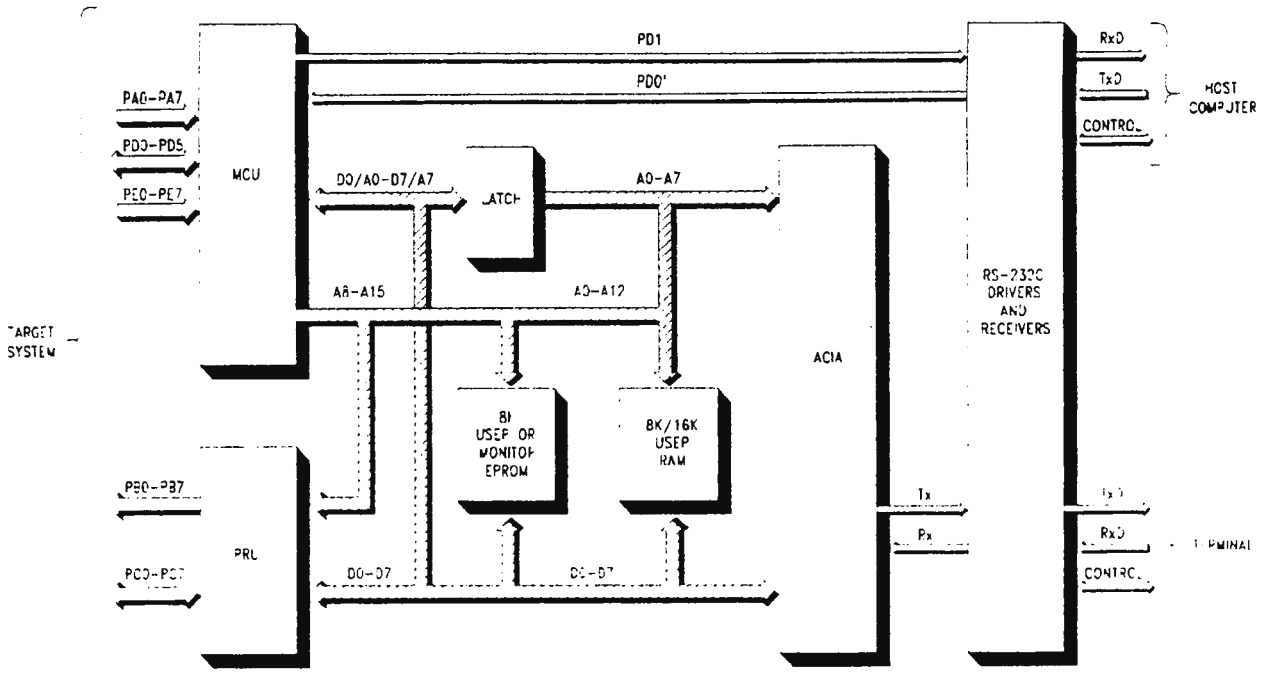


Figure 1-1. EVB Block Diagram

1.5 EQUIPMENT REQUIRED

Table 1-2 lists the external equipment requirements for EVB operation.

Table 1-2. External Equipment Requirements

External Equipment
+5, +12, -12 Vdc power supply(1)
Terminal (RS-232C compatible)
Host computer (RS-232C compatible)(2)
Terminal/host computer - EVB RS-232C cable assembly(1)
Target system - EVB MCU I/O port extension cable assembly(1)

1. Refer to Chapter 2 for details.
2. Optional - not required for basic operation

CHAPTER 5
HARDWARE DESCRIPTION

5.1 INTRODUCTION

This chapter provides an overall general description of the EVB hardware. This description is supported by a simplified block diagram (Figure 5-1) and a memory map diagram (Figure 5-2). The EVB schematic diagram, located in Chapter 6, can also be referred to for the following descriptions.

5.2 GENERAL DESCRIPTION

Overall evaluation/debugging control of the EVB is provided by the monitor BUFFALO program residing in EPROM (external to the MCU) via terminal intervention. The target system interface is provided by the MCU and PRU devices. RS-232C terminal/host I/O port interface circuitry provides communication and data transfer operations between the EVB and external terminal/host computer devices.

5.2.1 Microcomputer

The M68HC11A1 MCU (U10) operates in the expanded mode of operation. This is accomplished by +5 Vdc applied to the MCU MODA and MODB pins. The MCU configuration (CONFIG) register (implemented in EEPROM) is programmed such that the ROMON bit is cleared for EVB operations. When this bit is cleared, MCU internal ROM is disabled , and that memory space becomes externally accessed space. This allows the memory at $E000-$FFFF to contain the monitor BUFFALO program (or user program for emulation) in external EPROM.

The monitor program uses the MCU internal RAM located at $0048-$00FF. The control registers are located at $1000-$103F.

The EVB allows the user to use all the features of the monitor BUFFALO program, however it should be noted that the monitor program uses the MCU on-chip RAM locations $0048-$00FF leaving only 72 bytes for the user (i.e., $0000-$0047). This should be remembered when writing code.

5.2.2 Port Replacement Unit

The EVB operates in the expanded multiplexed mode of operation. An MC68HC24 PRU device (U1) is used to replace the MCU I/O ports B and C (including STRA and STRB control lines) used for single chip mode of operation. The PRU provides the required single chip mode I/O lines for target system evaluation (emulation) via the EVB MCU extension I/O port connector P1.

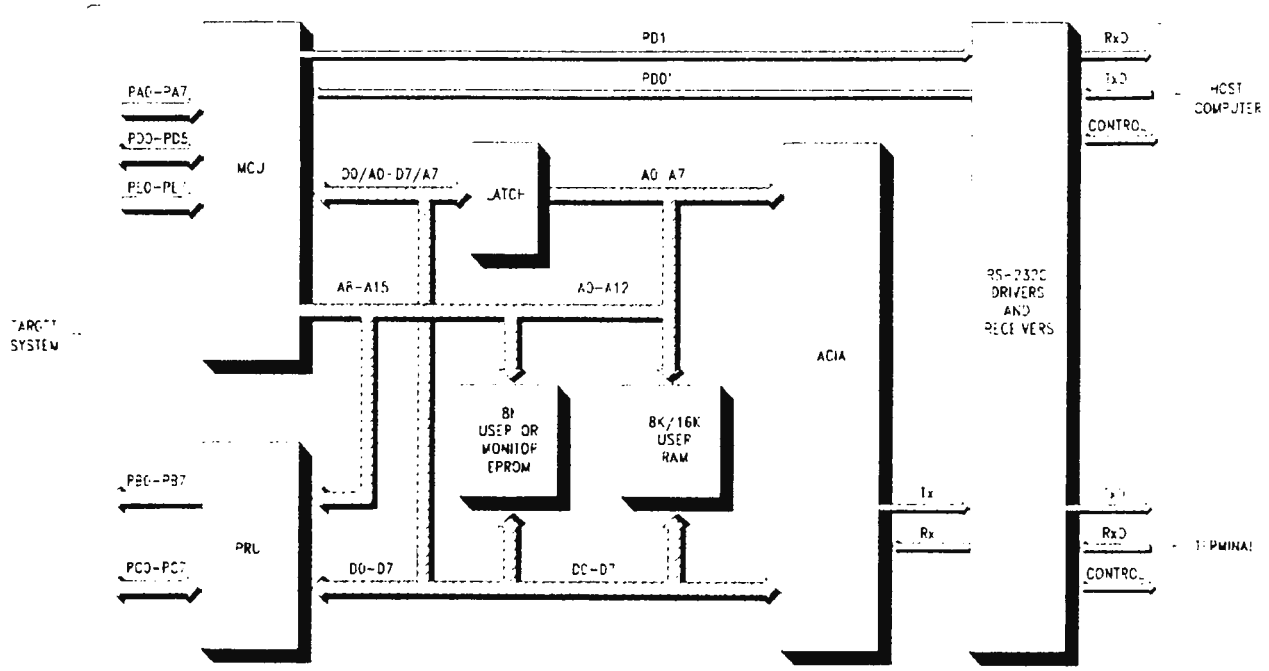
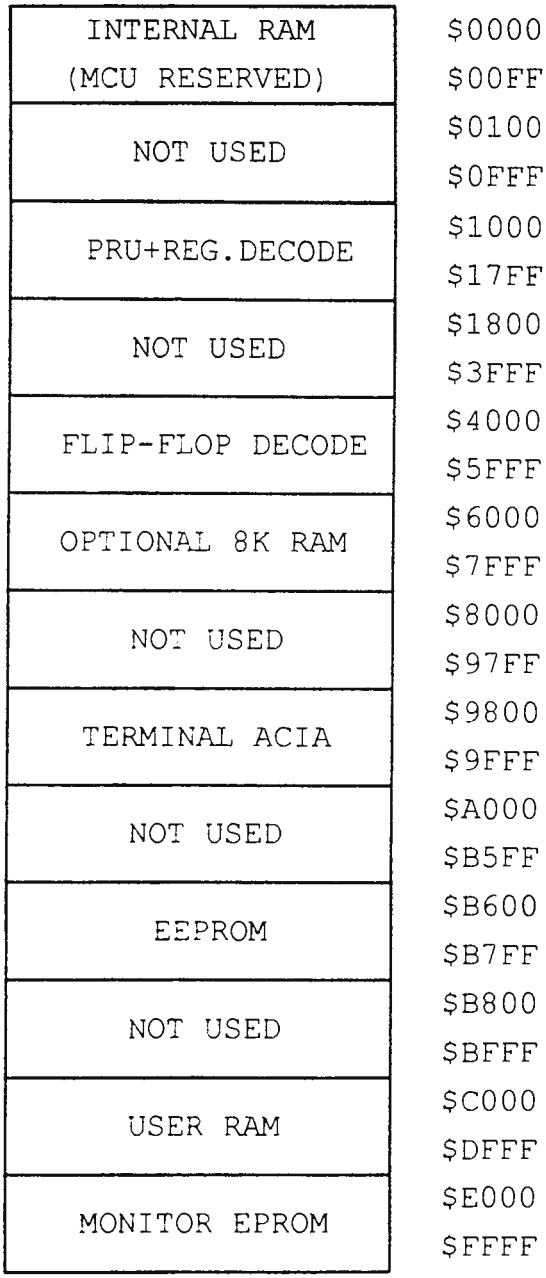


Figure 5-1. EVB Block Diagram



$0000-$0032 USER RAM
$0033-$0047 USER STACK POINTER
$0048-$00C3 MONITOR VARIABLES
$00C4-$00FF VECTOR JUMP TABLE

Figure 5-2. EVB Memory Map Diagram

5.2.3 Memory

The EVB memory map is a single map design. User RAM resides at different address locations to that of the MCU ROM. Any code debugged in the RAM which is not re-locatable will require modification before being transferred to EPROM and executed instead of the monitor (evaluation mode).

To evaluate programs normally held in ROM, an 8k byte RAM is provided (socket U5). An access time of 250 nanoseconds is necessary for a bus frequency of 2.1 MHz. Jumper headers J3 and J7 configure socket U4 for an additional 8k byte RAM supplied by the user if required. Refer to paragraph 2.3.3 for additional memory select information.

5.2.4 Address Decoding/De-multiplexing

Address decoding is accomplished via a MC74HC138 device (U6) and is segmented into 8k byte blocks. The low order address and data lines are de-multiplexed using a MC74HC373 device (U2), to communicate with ROM, RAM, and the ACIA. The PRU uses a multiplexed input direct from the MCU.

5.2.5 RS-232C I/O Port Interface Circuits

The EVB uses an MC68B50 ACIA device (U9) to communicate to a terminal via an RS-232C driver/receiver interface (terminal I/O port). The terminal I/O port baud rate is hardware selectable (300-9600 baud) via jumper header J5.

A second RS-232C driver/receiver interface (host I/O port) is fixed at 9600 baud via the MCU SCI using a 2 MHz E clock external bus. This baud rate can be changed by software by reprogramming the BAUD register in the ONSCI subroutine of the BUFFALO monitor program. Refer to the buf25.asm file on the EVB diskette for additional information pertaining to the ONSCI subroutine.

The host I/O port is provided for downloading Motorola S-records via the BUFFALO monitor commands. When using the host I/O port, either by executing the HOST or LOAD commands, The target system Serial Communications Interface (SCI) is switched to the host I/O port via the MC74HC4066 digital switch device (U7) and MC74HC74 latch device (U11). The receiver connection from the host I/O port is now connected to the RXD port of the MCU. The switching of the receiver line from the target system to the host I/O port is accomplished by writing a logic one in the bit 0 to any address in the range $4000-$5FFF. Likewise, writing a zero in bit 0 to any address in the same range results in the target system being connected to the RXD pin of the MCU.

As the RS-232C handshake lines are not used, a delay of approximately 300 milliseconds is present between successive characters sent to the host computer during the execution of the LOAD command in the monitor program.

CHAPTER 6
SUPPORT INFORMATION

6.1 INTRODUCTION

This chapter provides the connector signal descriptions, parts list with associated parts location diagram, and schematic diagrams for the EVB.

6.2 CONNECTOR SIGNAL DESCRIPTIONS

The EVB provides one input/output (I/O) connector that is used to interconnect the EVB to a target system. Connector P1 facilitates this interconnection.

Connectors P2 and P3 are also provided to facilitate interconnection of a terminal and a host computer, respectively. Connector P4 interconnects an external power supply to the EVB.

Pin assignments for the above connectors (P1 through P4) are identified in Tables 6-1 through 6-4, respectively. Connector signals are identified by pin number, signal mnemonic, and signal name and description.

Table 6-1. MCU I/O Port Connector (P1) Pin Assignments

Pin Number	Signal Mnemonic	Signal Name and Description
1	GND	Ground
2	MODB	MODE B - An input control line used for MCU mode selection. A high level enables the expanded multiplexed mode, and a low level enables the special test mode of the EVB MCU.
3	MODA	MODE A - An input control line used for MCU mode selection during reset. An open-drain output line used for LIR* status indication.
4	STRA	STROBE A - An input control line used for parallel port I/O operations.
5	E	ENABLE CLOCK - An output control line used for timing reference. E clock frequency is one fourth the frequency of the XTAL and EXTAL pins.
6	STRB	STROBE B - An output control line used for parallel port I/O operations.
7	EXTAL	EXTAL - External MCU clock input line.
8	XTAL	XTAL - Internal MCU clock line used to control the EVB clock generator circuitry.
9-16	PC0-PC7	PORT C (bits 0-7) - General purpose I/O lines.
17	RESET*	RESET - An active low bi-directional control line used to initialize the MCU.
18	XIRQ*	X INTERRUPT REQUEST - An active low input line used to request asynchronous non-maskable interrupts to the MCU.

Table 6-1. MCU I/O Port Connector (P1) Pin Assignments (continued)

Pin Number	Signal Mnemonic	Signal Name and Description
19	IRQ*	INTERRUPT REQUEST – An active low input line used to request asynchronous interrupts to the MCU.
20 21 22 23 24 25	PD0 PD1 PD2 PD3 PD4 PD5	PORT D (bits 0-5) – General purpose I/O lines. These lines can be used with the MCU Serial Communications Interface(SCI) and Serial Peripheral Interface(SPI).
26	VDD	VDD – +5.0 Vdc power.
27-34	PA7-PA0	PORT A (bits 7-0) – General purpose I/O lines.
35-42	PB7-PB0	PORT B (bits 7-0) – General purpose output lines.
43 44 45 46 47 48 49 50	PE0 PE4 PE1 PE5 PE2 PE6 PE3 PE7	PORT E (bits 0-7) – General purpose input or A/D channel input lines.
51	VRL	VOLTAGE REFERENCE LOW - Input reference supply voltage (low) line for the MCU analog-to-digital (A/D) converter. Used to increase accuracy of the A/D conversion.
52	VRH	VOLTAGE REFERENCE HIGH - Input reference supply voltage (high) line. Same purpose as pin 51.
53-60	NC	Not connected.

Table 6-2. Terminal I/O Port Connector (P2) Pin Assignments

Pin Number	Signal Mnemonic	Signal Name and Description
1	GND	PROTECTIVE GROUND
2	RXD	RECEIVED DATA - Serial data input line.
3	TXD	TRANSMITTED DATA - Serial data output line.
4	NC	Not connected.
5	CTS	CLEAR TO SEND - An output signal used to indicate ready-to-transfer data status. This pin is connected to both DSR pin 6 and DCD pin 8.
6	DSR	DATA SET READY - An output signal used to indicate an on-line/in-service/active status. This pin is connected to both CTS pin 5 and DCD pin 8.
7	SIG-GND	SIGNAL GROUND - This line provides signal ground or common return connection (common ground reference) between the EVB and RS-232C compatible terminal.
8	DCD	DATA CARRIER DETECT - An output signal used to indicate an acceptable received line (carrier) signal has been detected. This pin is connected to both CTS pin 5 and DSR pin 6.
9-19	NC	Not connected.
20	DTR	DATA TERMINAL READY - An input line used to indicate an on-line/in-service/active status.
21-25	NC	Not connected.

Table 6-3. Host I/O Port Connector (P3) Pin Assignments

Pin Number	Signal Mnemonic	Signal Name and Description
1	GND	PROTECTIVE GROUND
2	RXD	RECEIVED DATA - Serial data output line.
3	TXD	TRANSMITTED DATA - Serial data input line.
4, 5	NC	Not connected.
6	DSR	DATA SET READY - An output signal used to indicate an on-line/in-service/active status. This pin is connected to DCD pin 8.
7	SIG-GND	SIGNAL GROUND - This line provides signal ground or common return connection (common ground reference) between the EVB and RS-232C compatible host computer.
8	DCD	DATA CARRIER DETECT - An output signal used to indicate an acceptable received line (carrier) signal has been detected. This pin is connected to DSR pin 6.
9-19	NC	Not connected.
20	DTR	DATA TERMINAL READY - An input line used to indicate an on-line/in-service/active status.
21-25	NC	Not connected.

Table 6-4. Input Power Connector (P4) Pin Assignments

Pin Number	Signal Mnemonic	Signal Name and Description
1	-12 V	-12 Vdc Power - Input voltage (-12 Vdc @ 0.1 A) used by the EVB logic circuits.
2	GND	GROUND
3	+5 V	+5 Vdc Power - Input voltage (+5 Vdc @ 0.5 A) used by the EVB logic circuits.
4	+12 V	+12 Vdc Power - Input voltage (+12 Vdc @ 0.1 A) used by the EVB logic circuits.

6.3 PARTS LIST

Table 6-5 lists the components of the EVB by reference designation order. The reference designation is used to identify the particular part on the parts location diagram (Figure 6-1) that is associated with the parts list table. This parts list reflects the latest issue of hardware at the time of printing.

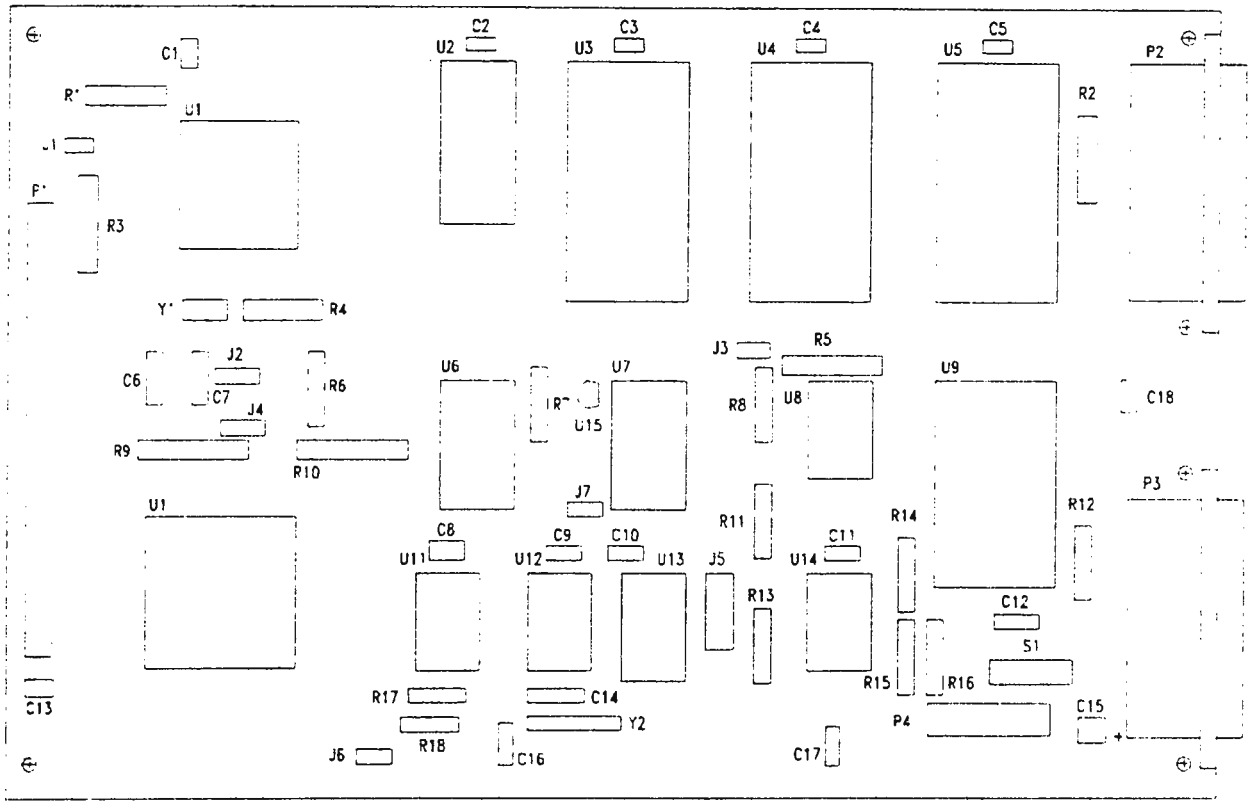


Figure 6-1. EVB Parts Location Diagram

Table 6-5. EVB Parts List

Reference Designation	Component Description
Printed Wiring Board (PWB)	M68HC11EVB
C1-C5, C8-C11, C13, C14, C17, C18	Capacitor, 0.1 uF @ 50 Vdc, +/-20%
C6, C7, C16	Capacitor, 24 pF @ 50 Vdc, +/-20%
C12	Capacitor, 1.0 uF @ 50 Vdc, +/-20%
C15	Capacitor, electrolytic, 100 uF @ 50 Vdc, +/-20%
J1, J3, J6	Header, jumper, single row post, 2 pin, Aptronics # 929705-01-02
J2, J4	Header, jumper, single row post, 3 pin, Aptronics # 929705-01-03
J5	Header, jumper, double row post, 12 pin, Aptronics # 929715-01-06
P1	Header, double row post, 60 pin, Aptronics # 929715-01-30 (MCU I/O port connector)
P2, P3	Connector, cable, 25-pin, ITT # DBP-25SAA (terminal/host I/O port connector)
P4	Terminal block, 4 position, Electrovert # 25.112.0453 (power supply connector)
R1-R3, R5-R10	Resistor, 10k ohm, 5%, 1/4W
R4	Resistor, 10M ohm, 5%, 1/4W
R11, R13-R15	Resistor, 27k ohm, 5%, 1/4W
R12	Resistor, 620 ohm, 5%, 1/4W

Table 6-5. EVB Parts List (continued)

Reference Designation	Component Description
R16	Resistor, 100k ohm, 5%, 1/4W
R17, R18	Resistor, 2.2k ohm, 5%, 1/4W
S1	Switch, push-button, SPDT, C&K #8125-R2/7527
U1	I.C., MC68HC24FN, PRU
U2	I.C., MC74HC373N, transparent latch
U3	I.C., 2764, 8k EPROM, 250 nS
U4	I.C., MCM6164, 8k RAM (user supplied)
U5	I.C., MCM6164, 8k RAM, 250 nS
U6	I.C., MC74HC138, decoder/de-multiplexer
U7	I.C., MC74HC4066/MC14066B, digital switch
U8	I.C., MC1488P, RS-232C driver
U9	I.C., MC68B50P, ACIA
U10	I.C., MC68HC11A1FN, MCU (Note 1)
U11	I.C., MC74HC74, D-type flip-flop
U12	I.C., MC74HC14, inverter
U13	I.C., MC74HC4040, binary ripple counter
U14	I.C., MC1489P, RS-232C receiver
U15	Voltage detector, 3.80-4.20 Vdc, Motorola #MC34064P or Seiko # S-8054HN
XU1	Socket, PC mount, 44 pin, PLCC, AMP # 821-551-1 (use with U1)

Table 6-5. EVB Parts List (continued)

Reference Designation	Component Description
XU3-XU5	Socket, 28 pin, DIP, Robinson Nugent # ICL-286-S7-TG (use with U3-U5)
XU9	Socket, 24 pin, DIP, Robinson Nugent # ICL-246-S7-TG (use with U9)
XU10	Socket, PC mount, 52 pin, PLCC, AMP # 821-575-1 (use with U10)
Y1	Crystal, MCU, 8.0 MHz (Notes 2 & 3)
Y2	Crystal, ACIA, 2.4576 MHz
Fabricated jumper	Aptronics # 929955-00 (use with jumper headers J1-J6)

NOTES

1. MCU supplied with the EVB have the configuration (CONFIG) register ROMON bit cleared to disable MCU internal ROM, thereby allowing external EPROM containing the BUFFALO program to control EVB operations.
2. Crystal frequencies from 4 to 8 MHz (up to 8.4 MHz) can be used without any changes to the 24 pF capacitors (C6 and C7) and 10M ohm resistor (R5) values.
3. 8 MHz crystal obtains 2 MHz E-clock/9600 baud MCU SCI operation. 4 MHz crystal obtains 1 MHz E-clock/4800 baud MCU SCI operation.

6.4 DIAGRAMS

Figure 6-2 is the EVB schematic diagram.

NOTES:

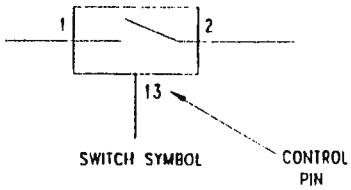
1. UNLESS OTHERWISE SPECIFIED:
   ALL RESISTORS ARE IN OHMS, ±5%, 1/4W.
   ALL CAPACITORS ARE IN uF.
   ALL VOLTAGES ARE DC.

2. DEVICE TYPE NUMBERS LISTED BELOW ARE FOR REFERENCE ONLY. DEVICE TYPE NUMBER VARIES WITH MANUFACTURER.

REF DES	DEVICE TYPE	NOTES	GND	+5V	+12V	-12V
U1	MC68HC24	PRU	29	17		
U2	MC74HC373	TRANSPARENT LATCH	10	20		
U3	2764	8K EPROM (250ns)	14	28		
U4	(USER SUPPLIED)	8K RAM (250ns)	14	28		
U5	MCM6164	8K RAM (250ns)	14	28		
U6	MC74HC138	DECODER/DEMUX	8	16		
U7	MC74HC4066	DIGITAL SWITCH	7	14		
U8	MC1488P	RS-232C DRIVER	7		14	1
U9	MC68B50P	ACIA	1	12		
U10	MC68HC11A1	MCU	1	26		
U11	MC74HC74	D-TYPE FLIP-FLOP	7	14		
U12	MC74HC14	INVERTER	7	14		
U13	MC74HC4040	RIPPLE COUNTER	8	16		
U14	MC1489P	RS-232C RECEIVER	7			

3. DIGITAL SWITH U7 OPERATES AS FOLLOWS:

CONTROL PINS (5, 6, 12, OR 13)	SWITCH OPERATION
LOGIC ZERO (0)	OFF/OPEN
LOGIC ONE (1)	ON/CLOSED



4. NOT USED DEVICE.

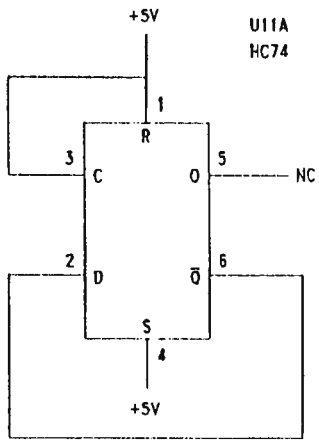


Figure 6-2. EVB Schematic Diagram (Sheet 1 of 2)

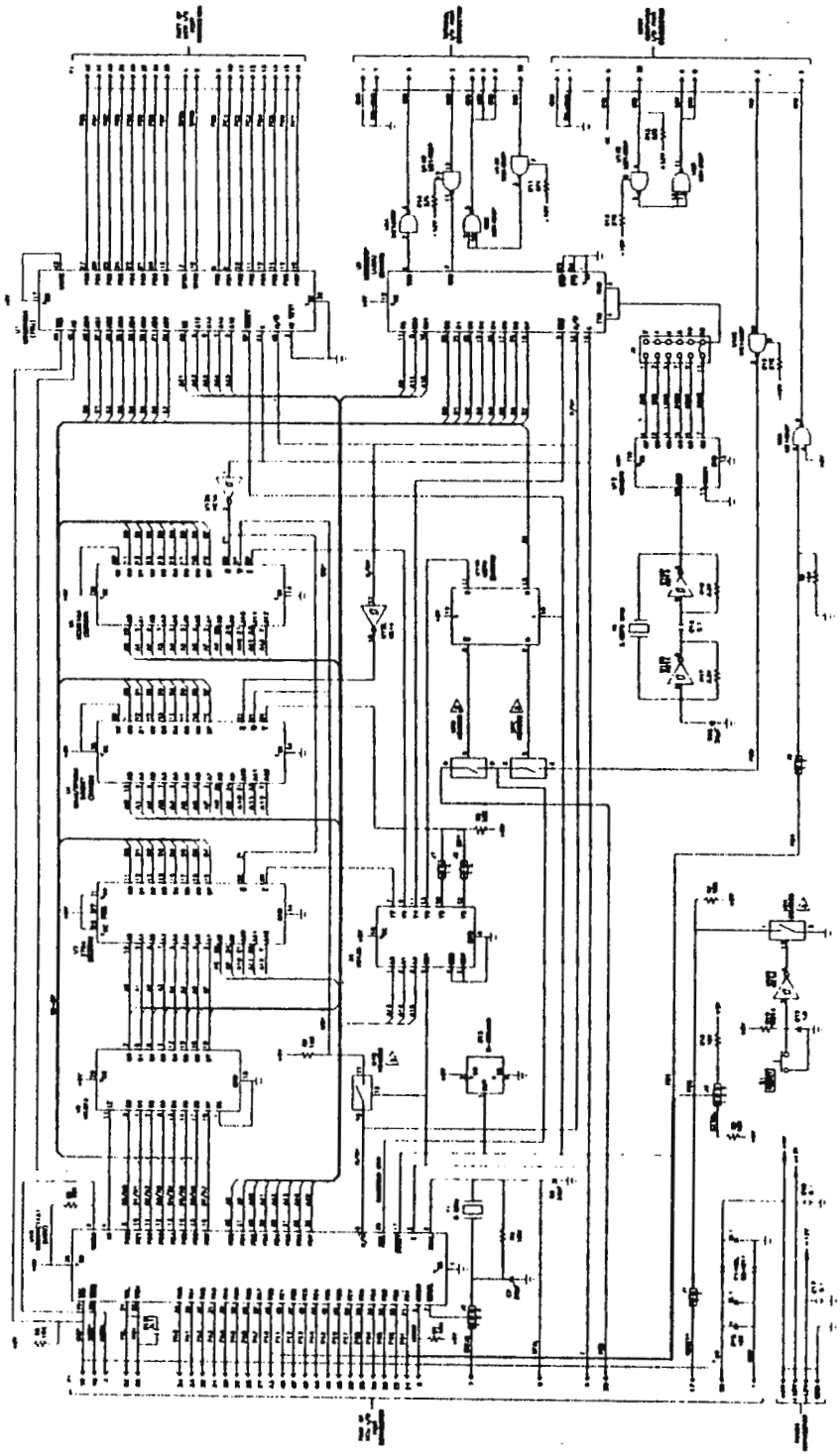


Figure 6-2. EVB Schematic Diagram (Sheet 2 of 2)